Neural Machine Translation for Bilingually Low-Resource Scenarios

by

Poorya Zaremoodi

MONASH University

Thesis

Submitted for the fulfilment of the requirements for the degree of

Doctor of Philosophy

Faculty of Information Technology

Monash University

January, 2020

To my parents.

Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Poorya Zaremoodi
January 30, 2020

Abstract

Neural Machine Translation (NMT) is revolutionising machine translation and has achieved great success across several language pairs. Like many deep learning methods, NMT is a data-hungry technology. However, we do not have the luxury of having large parallel datasets for many languages, a setting referred to as bilingually low-resource scenario. This thesis aims to improve the quality of NMT in this setting by leveraging curated monolingual linguistic resources for compensating the shortage of bilingual training data.

Linguistic resources are available in the form of annotated datasets, e.g., treebanks for syntactic parsing or part-of-speech tagged sentences. Two main approaches for incorporating the linguistic knowledge in these resources are: (1) use off-the-shelf/pre-trained tools to generate linguistic annotations; (2) directly injecting the linguistic knowledge using Multi-Task Learning (MTL). In this thesis, we investigate the use of these approaches to improve the quality of NMT in bilingually low-resource scenarios.

Firstly, we look at the case of using automatically generated linguistic annotations for improving the translation quality. We argue these annotations are generated by imperfect tools, and inevitably incorporate errors and uncertainties. We, therefore, propose a novel approach for handling these uncertainties and errors in the translation process by leveraging a *collection* of these generated annotations, e.g., using a parse forest instead of a single parse tree. This amounts to a novel forest-to-sequence neural transducer, which conditions on multiple linguistic analyses of the source sentence in the translation process.

Secondly, we take an MTL approach to inject a combination of semantic and syntactic knowledge into the translation model. This is achieved by sharing parts of parameters of the models trained for the main translation and various auxiliary syntactic/semantic tasks. We propose effective

parameter sharing strategies for making a better use of knowledge in linguistic resources. As it may be hard to decide manually which parameters to share, we propose an approach to *automatically learn* the sharing strategy.

Finally, we introduce a novel training schedule for MTL by proposing a rigorous data-driven approach to adaptively change the importance of tasks throughout the training of an MTL model to make the best use of auxiliary syntactic and semantic tasks. We introduce a novel framework for *automatically learning* effective training schedules by formulating MTL training as a sequential decision making, i.e., which task to select to get data for the parameter update in the next training step. We then formulate this as a Markov Decision Process (MDP), and make use of Imitation Learning to learn a reasonable policy for effective scheduling of the training process in MTL.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Assoc. Prof. Gholamreza (Reza) Haffari, Prof. Wray Buntine and Dr Sarvnaz Karimi (CSIRO Data61). I was lucky to foster the development of research skills under their patient guidance, motivation and immense knowledge. I owe my deepest gratitude to Reza for his unconditional support, and being a role model for "Never Give Up".

I am grateful to my examiners Dr Marine Carpuat (University of Maryland) and Dr Shahram Khadivi (eBay) for their constructive feedback on this thesis.

I would like to thank Monash University for generously supporting me with Monash International Postgraduate Research Scholarship and Monash Graduate Scholarship. Also, I thank CSIRO Data61 for supporting me with a top-up scholarship.

I have greatly benefited from Multi-modal Australian ScienceS Imaging and Visualisation Environment (MASSIVE) and Monash Advanced Research Computing Hybrid (MonARCH) by providing me with the opportunity to run many resource-intensive experiments.

Special gratitude goes to my parents for their love, motivation, encouragement, endless moral and emotional support over many years; words cannot describe how thankful I am.

I am grateful to my colleagues and officemates in Monash, Ehsan, Fahimeh, Ming, Najam, Narjes, Philip, Sameen, Snow, Trang, Xuanli, for their support, kindness, and the fun we have had over the last years. Finally, I would like to offer my special thanks to my dearest friends that are more like my family: Shokoufeh, Farshid, Mohammad, Hooman, Keyvan, Ali, and Zohreh.

Publications

The publications arising from my thesis are:

- (Published) P. Zaremoodi, G. Haffari, "Incorporating Syntactic Uncertainty in Neural Machine Translation with a Forest-to-Sequence Model", Proceedings of the 27th International Conference on Computational Linguistics (COLING), 2018.
- (Published) P. Zaremoodi, G. Haffari, "Neural Machine Translation for Low Resource Scenarios: A Deep Multi-Task Learning Approach", Proceedings of Annual Meeting for North American Chapter of Association of Computational Linguistics (NAACL), 2018.
- (Published) P. Zaremoodi, W. Buntine, G. Haffari, "Adaptive Knowledge Sharing in MTL: Improving Low-Resource NMT", Proceedings of Annual Meeting of Computational Linguistics (ACL), 2018.
- (Published) P. Zaremoodi, G. Haffari, "Adaptively Scheduled Multitask Learning: The Case of Low-Resource Neural Machine Translation", Proceedings of the 3rd Workshop on Neural Machine Translation and Generation, Co-located with EMNLP 2019.
- (Submitted) P. Zaremoodi, G. Haffari, "Learning to Multi-Task Learn for Better Neural Machine Translation", Submitted to the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI), 2020.

Contents

List of Figures

List of Notations

Markov Decision Process

s_t The state at time step t

$\boldsymbol{a}_t$ The actions at time step t

$R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ The instantaneous reward

$\pi_{\boldsymbol{\phi}}$ The policy function

$\boldsymbol{\phi}$ The parameters of the policy function

Other Symbols

$H[.]$ The entropy of a distribution

Translation and Multi-Task Learning

$\boldsymbol{x}$ The input/source sentence

$\boldsymbol{y}$ The target/output sentence

$\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)}$ The i^{th} source or target sentence of task k

$\boldsymbol{b}_t^k$ The t^{th} mini-batch of task k

$\mathcal{D}^k := \{(\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)})\}_{i=0}^{N_k}$ Training set of task k

$w_i^{(k)}$ The weight of i^{th} sentence pair of task k

$\boldsymbol{\Theta}_{mtl}$ The parameters of the MTL model

$\boldsymbol{h}_t$ The hidden representation of t^{th} step in the encoder or decoder RNN

$\boldsymbol{E}_S, \boldsymbol{E}_T$ The embedding table of the input or target space

$\boldsymbol{W}, \boldsymbol{U}$ Parameter matrices

List of Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
AL	Active Learning
GRU	Gated Recurrent Unit
IL	Imitation Learning
LM	Language Model
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
MT	Machine Translation
MTL	Multi-Task Learning
NLP	Natural Language Processing
NMT	Neural Machine Translation
RL	Reinforcement Learning
RNN	Recurrent Neural Network

Chapter 1

Introduction

1.1 Motivation

Deep learning is a comparatively new player in artificial intelligence (AI), but with an old history. Its origin goes back to decades ago when the subject of artificial neural networks started (McCulloch and Pitts, 1943; Rosenblatt, 1958; Minsky and Papert, 1969). Neural networks are, in fact, *representation learning* methods based on a multi-layer hierarchy of data abstraction. The power of neural networks is rooted in the fact that the representations are learned automatically and data-driven via a general-purpose algorithm, removing the need for hand-engineered features. Thanks to the recent advances in parallel computing and the abundance of big data, deep neural networks, aka deep learning, has started to transform AI. The deep learning tsunami[1] lapped the shores of many areas in AI, including computer vision, self-driving vehicles, robotics, natural language processing, and more specifically Machine Translation (MT).

Machine Translation is the task of automatically translating a text from one natural language to another. The translation should preserve the meaning of the source sentence while being fluent in the target language. Statistical Machine Translation (SMT) models dominated this area for decades. The most successful SMT model is phrase-based (Koehn et al., 2003; Brown et al., 1990), which segments the source sentence into phrases, translates these phrases, and re-orders them. The SMT systems consist of several sub-components which should be tuned separately, some of them requiring hand-engineered features along with large amounts of memory to store information in the form of phrase tables/dictionaries.

[1] A term coined by Manning (2015)

The deep learning revolution has now reached MT, leading to the state-of-the-art for many language pairs (Luong et al., 2015b,a; Wu et al., 2016; Edunov et al., 2018; Wu et al., 2019). This approach, called Neural Machine Translation (NMT), translates via a large neural network which *reads* the input sentence and outputs its corresponding translation.

The advantage of NMT models is their ability to learn end-to-end, without the need for many brittle design choices and hand-engineered features of those in SMT. However, as stated by Andrew Ng, one of the deep learning pioneers, deep learning is a powerful yet data-hungry technology.[2] However, we do not have the luxury of having large parallel datasets for many languages, a setting referred to as bilingually low-resource scenario. Koehn and Knowles (2017); Lample et al. (2018b) have reported NMT needs large amounts of bilingual training data to achieve reasonable translation quality; furthermore, it underperforms phrase-based SMT (PBSMT) models in bilingually low-resource scenarios where we. These findings have motivated research for improving the low-resource NMT, mostly by incorporating linguistic and monolingual resources.

More recently, Sennrich and Zhang (2019) has re-visited the case of low-resource NMT and shown that low-resource NMT with an optimised practice[3] is, in fact, a realistic option for low-resource regimes as well; see Figure 1.1. Further, they argue that "best practices differ between high-resource and low-resource settings". To summarise, there are two key messages in these findings: (1) low-resource NMT is very sensitive, and its best practice is different from those of high-resource; (2) the performance of an NMT model trained with small amount of training data still is much less than the one trained with millions of sentence pairs, and there is a great room for improvement to take full advantage of NMT models in small bilingual training data conditions. These key findings have shown the importance and need for investigating NMT exclusively for the case of bilingually low-resource scenarios.

For compensating the lack of bilingual data, a practical approach is to use auxiliary data in the form of curated monolingual linguistic resources, monolingual sentences or multilingual sentence pairs (a mixture of high- and low-resource). In this thesis, the focus is on using curated monolingual resources; however, this work is orthogonal to other approaches and could be combined with the others, e.g., a combination of linguistic resources and monolingual sentences. More specifically, we assume that rich

[2]https://www.wired.com/brandlab/2015/05/andrew-ng-deep-learning-mandate-humans-not-just-machines/

[3]In terms of architectural design, techniques and hyperparameters such as dropout, BPE (and its vocabulary size), tying parameters, and others.

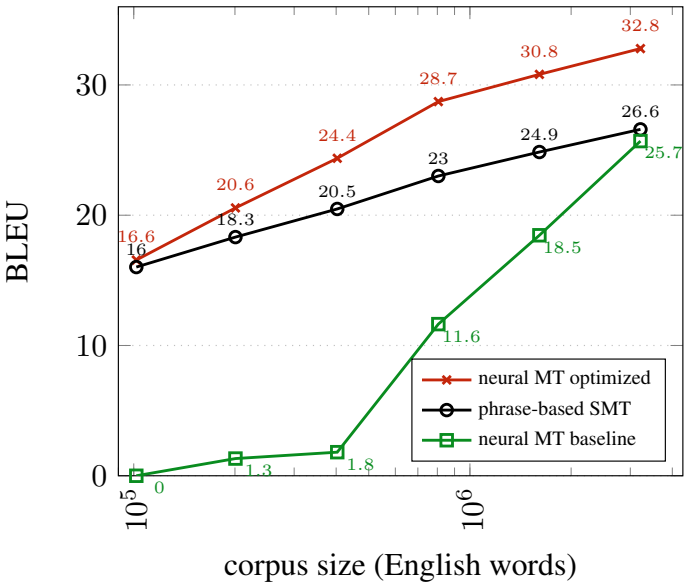


Figure 1.1: BLEU scores (higher is better) for PBSMT and NMT models on German → English translation task using the TED data from the IWSLT 2014 shared translation task. The number of words in parallel training data varies from 0.1 to 3.2 million. Image source: (Sennrich and Zhang, 2019).

linguistic annotation is available on the source language. The linguistic resources are available in the form of annotated datasets, e.g., treebanks for syntactic parsing or part-of-speech tagged sentences. There are two main approaches for incorporating linguistic knowledge in MT:

1. Use these monolingual resources to build tools which can then be used to annotate bilingual sentences. These annotations convey linguistic information in the form of combinatorial structures, and modelling them changes the translation task from a sequence-to-sequence task to the transduction of more complex combinatorial structures, e.g., tree-to-sequence (Eriguchi et al., 2016; Chen et al., 2017a).

2. Directly injecting linguistic knowledge into the translation model to improve its performance. Multi-Task Learning (MTL) is an effective approach to inject knowledge into a task, which is learned from other related tasks, aka auxiliary tasks.

1.1.1 Transduction of Complex Structures for Incorporating Linguistic Annotations

One of the central premises about natural language is that words of a sentence are interrelated according to a (latent) hierarchical structure (Chomsky, 1957), i.e., the syntactic tree. Therefore, it is expected that modelling the syntactic structure should improve the performance of NMT, especially in low-resource or linguistically divergent scenarios, such as English-Farsi, by learning better reorderings. Recently, Eriguchi et al. (2016); Chen et al. (2017a) have proposed methods to incorporate the hierarchical syntactic constituency information of the source sentence. They propose tree-to-sequence NMT models, which use the top-1 parse tree of the source sentence *generated* by a parser. They showed that the syntax-aware NMT model outperforms the vanilla sequence-to-sequence NMT model.

This approach relies on the accuracy of top-1 parse tree while automatically generated annotations are error-prone. As mentioned, curated monolingual linguistic resources can be used to build tools which then can provide the NMT model with the linguistic annotations of the source sentences. One should note that the automatically generated annotations incorporate errors and uncertainties of the tools, e.g., generated top-1 syntactic constituency trees are prone to parser errors; moreover, they cannot capture semantic ambiguities of the source sentence.

Although this is a practical approach for one type of linguistic information, adding more types of linguistic annotation requires the model to be re-designed and gets more and more complicated.

1.1.2 Injecting Linguistic Inductive Biases via Multi-Task Learning

As the translation task is, in fact, "function approximation", we can start by analysing it from the learning theory perspective. Neural networks are universal function approximators (Csáji, 2001), and theoretically, they can learn any function under mild conditions. While NMT models have infinite hypothesis space, they perform relatively weak in low-resource regimes. Therefore, the small sample size could be a reason for their weak generalisation. The generalisation is the cornerstone of machine translation (machine learning in general), and we need inductive biases in learning generalisation (Mitchell, 1980). Recent research has shown that NMT models can learn linguistic-related inductive biases (Shi et al., 2016; Belinkov et al., 2017; Poliak

et al., 2018). In addition, Dalvi et al. (2017) shows explicitly injecting morphology can improve translation quality. Hence, incorporating linguistically based inductive biases is a promising direction to improve the generalisation of NMT models in low-resource data regimes.

Multi-Task Learning is a practical approach to acquire inductive biases from *related* tasks, and its primary goal is to improve the generalisation performance (Caruana, 1997). MTL provides a *general* framework to incorporate different kinds of linguistic information into the translation model, and multiple different types of linguistic knowledge can be used (or added) without changing the model architecture or the need to replace the model with a more complicated one.

MTL is effective yet challenging. Various recent works have attempted to improve NMT with an MTL approach (Domhan and Hieber, 2017; Zhang and Zong, 2016; Niehues and Cho, 2017; Kiperwasser and Ballesteros, 2018); although their results are promising, these methods usually suffer from shortcomings which prevent them from maximally exploiting the knowledge in auxiliary tasks. These shortcomings are mainly rooted in the fact that injecting knowledge from auxiliary tasks is a double-edged sword. While "positive transfer" may help to improve the performance of the main translation task, "negative transfer" (i.e. task interference) may have unfavourable effects and degrade the translation quality. Addressing the negative transfer phenomena is challenging and needs to be tackled at different levels: architectural design and training schedule.

Architectural design: MTL in neural networks is best achieved by parameter sharing. The trivial approach of *fully* sharing the entire model parameters provokes the task interference issue, because tasks may need to learn task-specific knowledge which then should be learned in the fully shared parameters. Another point is that sharing parameters among *all* tasks may also lead to task interference or inability to leverage commonalities among *subsets* of tasks. Therefore, a capable MTL architecture is crucial to share the parameters of various neural components among *subsets* of tasks.

Training schedule: With shared parameters, the negative transfer phenomenon is inevitable (Ruder, 2017). Therefore, it is crucial to have an effective *training schedule* to balance out the importance, i.e., the participation rate of different tasks throughout

the training process, in order to make the best use of knowledge provided by auxiliary tasks.

1.2 Research Objectives

The main purpose of this research is to improve the performance of Neural Machine Translation in bilingually low-resource scenarios using linguistic knowledge, inspired by the following hypothesis:

Incorporating linguistic knowledge can improve the quality of Neural Machine Translation in bilingually low-resource scenarios.

Motivated by the literature gaps mentioned in the previous section, this thesis will address the following objectives:

- **Incorporating uncertainty of automatically generated annotations in translation (Part I).** As the off-the-shelf/pre-trained tools are error-prone, the errors and uncertainties of these tools should be taken into account in the translation process. Otherwise, it may affect the quality of the generated translation. For overcoming this challenge for syntactic annotations, this thesis proposes a novel approach for efficiently considering parser uncertainties and errors in annotations for generating better translations (Chapter 3).

- **Effective MTL architectural design for injecting both semantic and syntactic knowledge into the underlying translation task (Part II).** This thesis injects linguistic inductive biases into the translation using a combination of semantic and syntactic auxiliary tasks. It does so by proposing *partial* parameter tying and adversarial training to make better use of linguistic knowledge (Chapter 4). Then, it proposes an effective method to *automatically* learn an MTL architecture, mitigating the need to search for parameter sharing strategies (Chapter 5).

- **Effectively train an MTL model to improve the translation the most (Part III).** The training schedule is the beating heart of MTL. It is responsible for balancing the importance of tasks during the training. A proper balance is crucial for making the best use of auxiliary linguistic tasks in favour of the translation task. This thesis proposes a data-driven approach for *adaptively* and *dynamically* training an MTL model (Chapter 6). In addition, a novel framework is proposed for *learning* to multi-task learn (Chapter 7).

1.3 Thesis Outline and Contributions

Chapter 2: Background

In Chapter 2, we review the foundations and related work for the research in this thesis. We start by discussing the foundations of deep learning and its use in NLP. Then, we review NMT and focus on the case of bilingually low-resource scenarios.

Part I: This part is dedicated to the transduction of complex structures where the model uses automatically generated annotations.

Chapter 3: An Attentional Forest-To-Sequence Model

> This chapter is based on:
> P. Zaremoodi, G. Haffari, "Incorporating Syntactic Uncertainty in Neural Machine Translation with a Forest-to-Sequence Model", Proceedings of the 27th International Conference on Computational Linguistics (COLING), 2018.

To handle errors and uncertainties in the generated syntactic annotations, we propose a model to translate from a source sentence along with its corresponding parse forest provided by a parser. By using a forest of parse trees, our forest-to-sequence model efficiently considers combinatorially many constituency trees, hence taking into account the parser uncertainties and errors.

Part II: This part focuses on MTL in order to directly inject the linguistic knowledge into the translation model. We contribute to the architectural design aspect of MTL:

Chapter 4: Deep Seq2Seq MTL for NMT

> This chapter is based on:
> P. Zaremoodi, G. Haffari, "Neural Machine Translation for Low Resource Scenarios: A Deep Multi-Task Learning Approach", Proceedings of Annual Meeting for North American Chapter of Association of Computational Linguistics (NAACL), 2018.

We make use of curated monolingual linguistic resources in the source side to improve NMT in bilingually scarce scenarios. We scaffold the machine translation task on auxiliary tasks including syntactic parsing, named-entity recognition and semantic parsing. To the best of our knowledge, our work is the first to inject semantic

knowledge into a neural translation model using MTL. This is achieved by casting the auxiliary tasks as sequence-to-sequence (SEQ2SEQ) transduction tasks and tying the parameters of these neural models with those of the main translation task. Our MTL architecture makes use of deep stacked models, where the parameters of the top layers are shared across the tasks. We further prevent the contamination of common knowledge with task-specific information using a technique so-called adversarial training. Our extensive empirical results demonstrate the effectiveness of our MTL approach in improving the translation quality.

Chapter 5: Adaptive Knowledge sharing in Deep Seq2Seq MTL

> This chapter is based on:
> P. Zaremoodi, W. Buntine, G. Haffari, "Adaptive knowledge Sharing in MTL: Improving Low-Resource NMT", Proceedings of Annual Meeting of Computational Linguistics (ACL), 2018.

We propose an MTL model, which learns how to control the amount of sharing among all tasks dynamically. We propose a new neural recurrent unit by extending existing ones to process multiple information flows through time. The proposed unit is equipped with a trainable *routing* mechanism that enables adaptive collaboration by dynamic sharing of information flows conditioned on the task at hand, input, and model state. Our experimental results and analyses show the effectiveness of the proposed approach on leveraging commonalities among subsets of tasks without the need to search on parameter sharing strategies.

Part III: This part is focused on the *effective training* of an MTL model for making the best use of auxiliary linguistic knowledge in translation.

Chapter 6: Adaptive Scheduling for Deep Seq2Seq MTL

> This chapter is based on:
> P. Zaremoodi, G. Haffari, "Adaptively Scheduled Multitask Learning: The Case of Low-Resource Neural Machine Translation", Proceedings of the 3rd Workshop on Neural Machine Translation and Generation, Co-located with EMNLP 2019.

We propose a rigorous general approach for adaptively changing the training schedule in MTL to make the best use of auxiliary syntactic and semantic tasks. To balance the importance of the auxiliary tasks versus the main translation task, we re-weight

training data of the tasks based on their contributions to the generalisation capabilities of the resulted translation model. Our main idea is to learn these importance weights automatically by maximising the performance of the main task on a validation set, separated from the training set, in each parameter update step. In addition, our analysis shed light on the relative importance of auxiliary linguistic tasks throughout the training of an MTL model, showing a tendency from syntax to semantics.

Chapter 7: Learning to Multi-Task Learn

> This chapter is based on:
> P. Zaremoodi, G. Haffari, "Learning to Multi-Task Learn for Better Neural Machine Translation", Submitted to the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI), 2020.

We propose a novel framework for *learning* how to multi-task learn to maximally improve the main NMT task. This is achieved by formulating the problem as a sequential decision making in a Markov Decision Process (MDP), enabling to treat the training scheduler as the policy. We then use the re-weighting mechanism proposed in Chapter 6 as the *oracle policy*. In order to scale up, we use the oracle policy as a teacher to train a scheduler network within the Imitation Learning framework, using DAGGER (Ross et al., 2011), to train and use the network simultaneously. As a result, the scheduler and MTL model is learned in the course of a single training run.

Chapter 8: Conclusions and Future Directions

Chapter 8 concludes this thesis by outlining the contributions as well as potential directions for future research.

Chapter 2

Background

In this chapter, the foundations and prior related works for the research in this thesis are overviewed. As the aim of this thesis is to improve bilingually low-resource Neural Machine Translation (NMT) by injecting linguistic knowledge, we review the recent progress in: deep learning fundamentals, Neural Machine Translation and low-resource NMT.

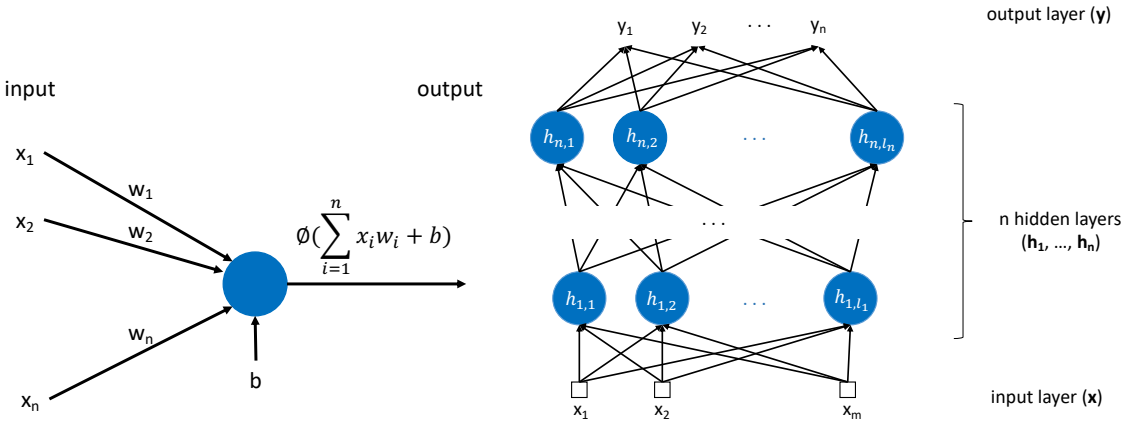
In Section 2.1, we start by overviewing fundamental concepts and methods in deep learning by discussing multi-layer perceptron (MLP), recurrent neural network (RNN) and its variants along with two popular regularisation techniques. Then, we cover the use of deep learning in the fundamentals of NLP: word embedding and statical language model.

In Section 2.2, we review the recent progress in NMT and discuss the leading families of architectures based on RNN, convolution and self-attention. Moreover, we describe how to evaluate the performance of an NMT model using different metrics.

In Section 2.3, we outline recent progress in bilingually low-resource NMT with the focus on describing how monolingual resources are used to compensate for the lack of bilingual data. Also, we narrow down on the usage of linguistic knowledge and explain two commonly used techniques for doing so: (1) transduction of complex structures for incorporating linguistic annotations; (2) Multi-Task Learning for directly injecting the linguistic knowledge into the NMT system.

2.1 Deep Learning

Deep learning, in fact, is a rebranding of Artificial Neural Networks with multiple layers (aka Deep Neural Networks). Although this popularity has been gained recently, the history of Neural Networks goes back to decades ago (McCulloch and Pitts, 1943; Rosenblatt, 1958; Minsky and Papert, 1969).



(a) Perceptron (b) Multi-layer Perceptron

Figure 2.1: Perceptron and Multi-layer Perceptron models.

Deep learning methods are representation learning methods composed of multiple stacked-up layers of data abstraction. Starting with the raw data, they transform the representation of each level to a slightly higher abstract representation in the next level by using simple non-linear modules. Very complex functions can be learned with this architecture if enough composition of this transformations is stacked-up (LeCun et al., 2015). For example, in a classification task, higher levels show representations which reflects important information for discriminating classes without contamination by irrelevant information. The key point in learning deep models is that intricate structures can be learned using a general-purpose learning procedure.

In the rest of this section, we cover related neural network methods which are building blocks of more complicated models. Also, we discuss two popular regularisation methods broadly used in deep learning.

2.1.1 Multi-Layer Perceptron (MLP)

Perceptron is the building block of the MLP model. As depicted in Figure 2.1a, a neuron maps its input to output by passing the weighted input ($\sum_i w_i x_i + b$) through an activation function ϕ. Activation functions should be nonlinear; otherwise, a multi-layer network can be reduced to an equivalent single-layer network realising a linear transform. One popular choice is the sigmoid function whose range is [0, 1]:

$$\phi(v) = \frac{1}{1 + e^{-v}}. \tag{2.1}$$

The other choice is the hyperbolic tangent function whose range is [-1, 1]:

$$\phi(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}}. \tag{2.2}$$

A single layer of neurons (perceptrons) can learn simple mapping but not able to learn nonlinearly separable data. MLP (Rumelhart et al., 1985) addresses this issue by adding multiple intermediate layers where the inputs to neurons in a layer are the outputs of neurons in previous layers, as shown in Figure 2.1b. The output calculated by this network is equivalent to the following equation:

$$\boldsymbol{y} = \phi_o \left(\boldsymbol{U} \phi_n \left(\boldsymbol{W}_n \phi_{n-1} \left(\boldsymbol{W}_{n-1} ... \phi_1 \left(\boldsymbol{W}_1 \boldsymbol{x} \right) \right) \right) \right), \tag{2.3}$$

where $\boldsymbol{W}_{(i)}$ shows the matrix weight of i-th hidden layer, $\boldsymbol{U}$ is the weight matrix connecting last hidden layer to the output layer, and ϕ_i is the activation function. Generally, element-wise functions like sigmoid and hyperbolic tangent are used in the hidden layer, and normalisation functions like softmax are used for the output layer. If we define each layer i as a non-linear module m_i,

$$m_i(\boldsymbol{h}) = \phi_i(\boldsymbol{W}_i \boldsymbol{h}), \tag{2.4}$$

then

$$\boldsymbol{y} = \phi_o \left(\boldsymbol{U} \left(m_n \circ m_{n-1} \circ ... \circ m_1(\boldsymbol{x}) \right) \right), \tag{2.5}$$

where $\circ$ denotes the function composition. As seen, MLP is, in fact, composed of multiple-layers of parametrised non-linear modules (Bengio et al., 2007). Each layer transforms representation of its previous layer to a higher, slightly more abstract representation using a non-linear module:

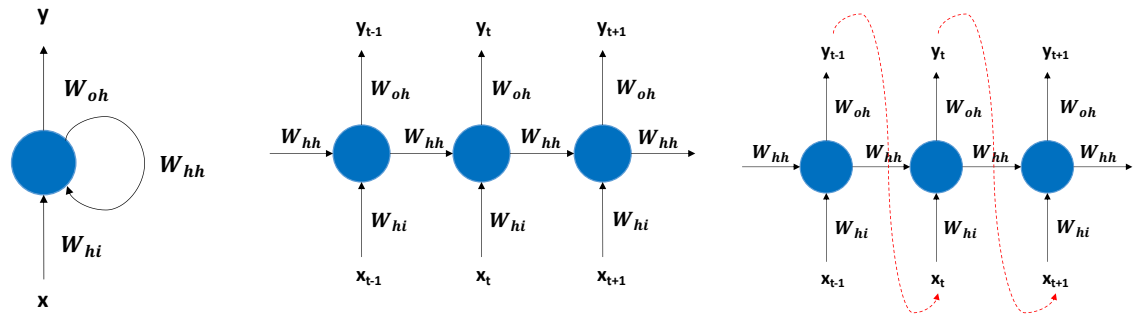
$$\boldsymbol{h}_i = m_i(\boldsymbol{h}_{i-1}) = \phi_i(\boldsymbol{W}_i \boldsymbol{h}_{i-1}). \tag{2.6}$$

MLP is a simple yet powerful computational model, and an MLP with as little as one hidden-layer is proved to be a universal function approximator under certain conditions (Hornik, 1991).

2.1.2 Recurrent Neural Network (RNN)

A recurrent neural network is a network of neurons for which the connection between nodes forms a cyclic directed graph. In this cyclic architecture, the model can use its internal representations as a memory to store the summary of its previous inputs to the moment. Because of this feature, RNN is a proper choice for processing variable-length sequences, e.g., sentences.

Elman network (Elman, 1990) is one of the most popular RNN architectures. It is similar to a single-layer MLP with a self-feedback connection, as depicted in Figure 2.2a. Unfolding it through time results in a network similar to a multi-layer



(a) Elman RNN (b) unfolded Elman RNN (c) unfolded generative RNN

Figure 2.2: Elman RNN, unfolded Elman RNN and unfolded generative RNN.

MLP; see Figure 2.2b. Here, the representation in the hidden layer aka "state" of the model at time t is calculated as:

$$\boldsymbol{h}_t = \phi_h(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{hi}\boldsymbol{x}_t) \tag{2.7}$$

where $\boldsymbol{W}_{hh}$ and $\boldsymbol{W}_{hi}$ are weight matrices corresponding to feedback loop and input to hidden layer, respectively. ϕ_h is activation function (e.g. sigmoid), and $\boldsymbol{x}_t$ is the input vector at time t. The output at time t is:

$$\boldsymbol{y}_t = \phi_o(\boldsymbol{W}_{oh}\boldsymbol{h}_t), \tag{2.8}$$

where $\boldsymbol{W}_{oh}$ is the weight matrix of connection between hidden and output layers, and ϕ_o is a normalisation function, e.g. softmax.

Generative RNN is another kind of RNN architecture proposed to generate a sequence. In this model, output is feed-backed as the next input (Figure 2.2c). In an NLP task, we normally use a normalisation function as the activation function of the output layer (ϕ_o), then the output vector $\boldsymbol{y}_i$ would be a probability distribution on symbols (e.g., words) at time i. At each time step, we draw a sample symbol (word) from this distribution and the symbol will be the input of the network at the next time step.

2.1.3 Backpropagation

Backpropagation (BP) is an algorithm for the supervised learning of neural models. It calculates the gradients in an MLP, which in turn is essential for training the model, and is consist of two phases: forward and backward. In the forward phase, the input is fed to the model and output is calculated. In the backward pass BP calculates

the error between the output $\boldsymbol{y}$ and the target output $\boldsymbol{t}$ with respect to an objective function, for example:

$$\boldsymbol{J}(\boldsymbol{\Theta}) = \frac{1}{2}(\boldsymbol{t} - \boldsymbol{z})^2, \tag{2.9}$$

where $\boldsymbol{\Theta}$ denotes all weights and biases in the network. For each node i in layer l, BP calculates a local gradient $\delta_i^{(l)}$ that is a measure showing how much the node was responsible for the error in the output. Then gradients are fed into an optimisation algorithm like Stochastic Gradient Descent (SGD) to update the weights:

$$\boldsymbol{W}_{ij}^{(l)} = \boldsymbol{W}_{ij}^{(l)} - \eta \frac{\partial}{\partial \boldsymbol{W}_{ij}^{(l)}} \boldsymbol{J}(\boldsymbol{\Theta}), \tag{2.10}$$

where $\boldsymbol{W}_{ij}^{(l)}$ refers to the weight connecting j-th node of layer $(l-1)$ to the i-th node at the layer l, and η is the step size (aka learning rate).

We can calculate the derivative of the error with respect to each weight in the network using the chain rule. There is a systematic way to perform it layer-by-layer. In this way, we calculate the local gradient of nodes layer-by-layer (descending order) then we use them to calculate gradients of weights. The local gradients for the neurons at the output layer is calculated as:

$$\delta_j^{(o)} = (t_i - y_i)\phi_o'(v_j^o), \tag{2.11}$$

where v_j^o is the input of the neuron (inner product of previous layer signals with corresponding weights). The error is backpropagated to the other layers with chain rule, which results in the following equation for the local gradients of j-th node in l-th layer:

$$\delta_j^{(l)} = \phi'(v_j^l) \sum_k \delta_k^{(l)} \boldsymbol{W}_{kj}^{(l+1)}. \tag{2.12}$$

Now we calculate the desired partial derivatives as follows:

$$\frac{\partial}{\partial \boldsymbol{W}_{ij}^{(l)}} \boldsymbol{J}(\boldsymbol{\Theta}) = \phi_j(\boldsymbol{v}_j^{(l)}) \delta_i^{(l+1)}, \tag{2.13}$$

where $v_j^{(l)}$ is the input of the j-th neuron in l-th layer.

Backpropagation is designed for networks without a loop, e.g. MLP. There is a modified version of it called Backpropagation Through Time (BPTT) which is used for RNNs. It simply unrolls RNN and applies the standard BP algorithm. In BPTT, for each weight, we sum its gradients at different time steps.

Hochreiter (1991) first discovered the vanishing and exploding gradient problem in training RNNs using backpropagation. Here we analyse this problem in the one-dimensional scenario as it is easier to understand; the extension to higher dimensions

can be found in (Pascanu et al., 2013). The hidden state of RNN at time t can be calculated by eqn. 2.7. If we simplify it by assuming only *one* neuron in the hidden layer and also removing the input, then we have:

$$h_t = \phi_h(w_{hh}h_{t-1}). \tag{2.14}$$

In order to show the problem more clearly, we directly use the chain rule instead of the systematic layer-by-layer approach. Thus, by taking the derivative of the hidden layer at time $t+l$ with respect to it at time t we have:

$$\frac{\partial h_{t+l}}{\partial h_t} = \prod_{k=1}^{l} w_{hh}\phi'(w_{hh}h_{t+(l-k)}) = w_{hh}^{l}\prod_{k=1}^{l}\phi'(wh_{t+(l-k)}). \tag{2.15}$$

The w_{hh}^{l} is the factor that causes the problem. As seen, if the weight is not equal to 1, it will decay to zero, or grow exponentially fast. This vanishing or exploding of the gradient causes RNN to be unable to learn long-term dependencies efficiently. One way to avoid this problem is to use ReLU activation function:

$$\mathrm{ReLU}(x) = \max(0, x). \tag{2.16}$$

The derivation of this function is whether 0 or 1. Thus, it is not as likely to suffer from the vanishing or exploding problem. A more popular approach is Long Short-Term memory which is explained in the following.

2.1.4 Long Short-Term Memory (LSTM)

LSTM is a special kind of RNN proposed to be able to learn long-term dependencies (Hochreiter and Schmidhuber, 1997b). The LSTM addresses the vanishing and exploding gradient problem by using a gating mechanism.

The LSTM keeps a cell state ($\boldsymbol{c}$), and add or remove information from the cell. This process is regulated through structures called gates. Gates are sigmoid layers that generate numbers in the range [0, 1]. These numbers show how much the corresponding component can participate in the information flow. The architecture of the LSTM unit is depicted in Figure 2.3.

The first gate we analyse is the forget gate. It controls how much information we should keep from the previous cell state $\boldsymbol{c}_{t-1}$. For example, in language modelling, it may keep information about the subject to remember that present verb should end with "s". However, if the current input is a new subject, the model may want to

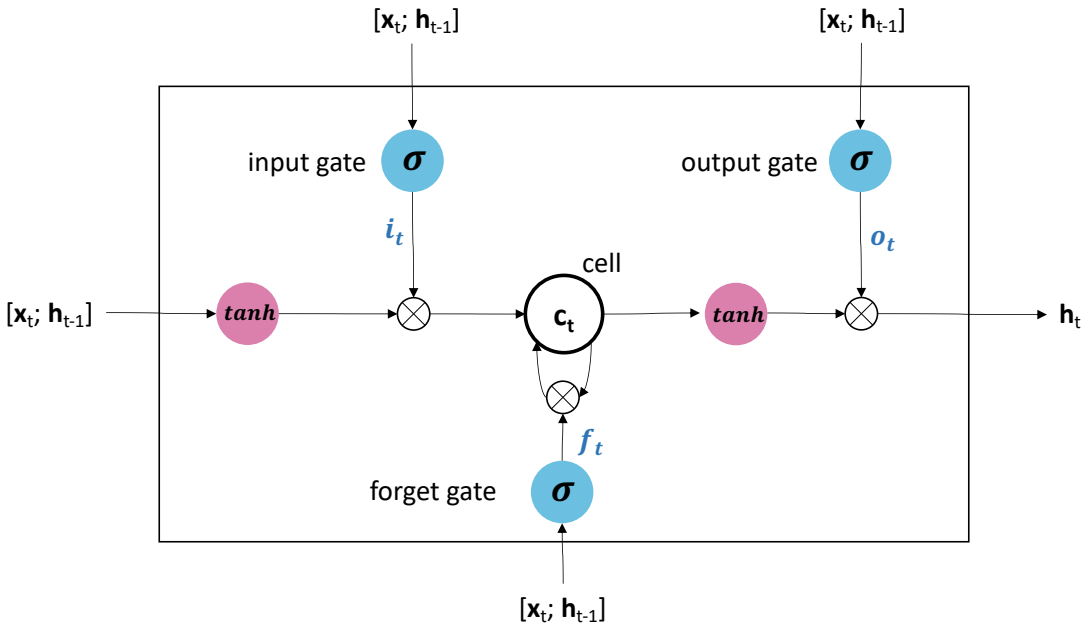


Figure 2.3: Architecture of the LSTM unit.

forget the old one. In the LSTM model, forget gate is fed with the previous hidden state and current input:

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{U}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}), \tag{2.17}$$

where $\boldsymbol{W}$ and $\boldsymbol{U}$ are weight matrices, and $\boldsymbol{b}$ is bias vector.

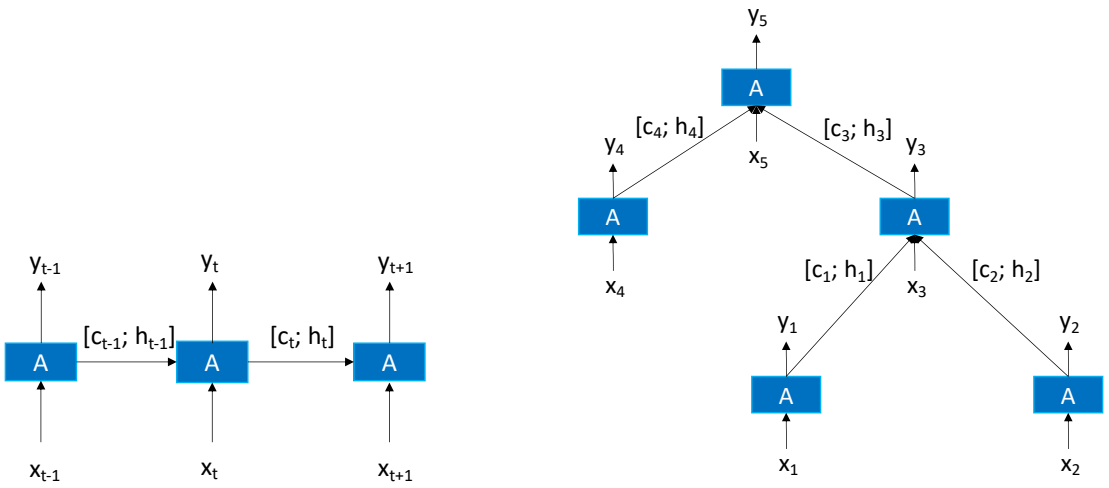
The forget gate determines how much information should remain from the previous cell state. In order to update the cell state, we also need to determine which information we want to add to it. As mentioned for the language modelling example, it is clear that we want to add information provided by the current input with respect to the current hidden state. Thus, at the first step, we create a new candidate vector, $\tilde{\boldsymbol{c}}_t$, that could be added to the cell:

$$\tilde{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_t + \boldsymbol{U}^{(\tilde{c})}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(\tilde{c})}). \tag{2.18}$$

Now, for a higher level regulation on the added information, we calculate an input gate which is very similar to the forget gate but with a different set of weights:

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{U}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}). \tag{2.19}$$

Now, everything is ready to update the cell state. In terms of language modelling example, we want to drop information related to the old subject, and replace them



(a) Chain-structured LSTM

(b) Tree-structured LSTM

Figure 2.4: Tree and sequence structured LSTMs. Here we used "A" to show an LSTM unit and emphasise that the unit and weights remain the same while traversing the structure.

with information about the new subject:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t. \tag{2.20}$$

The output of the LSTM unit, $\boldsymbol{h_t}$, shows the hidden representation of the LSTM, which can be used by exterior units. The output is a filtered version of the cell, and the output gate is responsible for gating the information:

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{U}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}). \tag{2.21}$$

Then, a hyperbolic tangent is applied to the cell to push values in the [-1, 1] range. Finally, the output of the unit would be:

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t). \tag{2.22}$$

As seen, the cell is a linear unit with a fixed-weight self-connection (Hochreiter and Schmidhuber, 1997a) , and avoids the vanishing or exploding gradient problem. Inspired by LSTM, Cho et al. (2014b) proposed Gated Recurrent Unit (GRU) that modulates the flow of information using two gates, without having a separate memory cell.

2.1.4.1 Tree-LSTM

The LSTM has the ability to preserve long-term information over time. This ability causes the LSTM to be a powerful tool for chain-structured topologies like temporal

sequences. However, in some tasks, we deal with the tree-structure data flow. Tai et al. (2015) proposed two extensions to the original LSTM architecture for tree-structured network typologies: Child-sum Tree-LSTM and N-ary Tree-LSTM.

In the standard LSTM, the unit updates the cell concerning the current input and the state of the cell in the previous time step $t-1$. We can think about this temporal sequence as a linear tree; see Figure 2.4a. Thus, an LSTM at position t is dependent on the current input and its child (the unit state at time $t-1$). Now, tree-structure topology (Figure 2.4b) can be considered as a generalisation to the linear tree structure. Here each node may have more than one child. Hence, Tree-LSTM has a forget gate for each child.

Child-sum Tree-LSTM: If we are given a tree and $C(j)$ denotes the children of node j, the transition functions for this unit are as follows:

$$\tilde{\boldsymbol{h}}_j = \sum_{k \in C(j)} \boldsymbol{h}_k, \tag{2.23}$$

$$\boldsymbol{i}_j = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_j + \boldsymbol{U}^{(i)}\tilde{\boldsymbol{h}}_j + \boldsymbol{b}^{(i)}), \tag{2.24}$$

$$\boldsymbol{f}_{jk} = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_j + \boldsymbol{U}^{(f)}\tilde{\boldsymbol{h}}_j + \boldsymbol{b}^{(f)}), \tag{2.25}$$

$$\boldsymbol{o}_j = \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_j + \boldsymbol{U}^{(o)}\tilde{\boldsymbol{h}}_j + \boldsymbol{b}^{(o)}), \tag{2.26}$$

$$\tilde{\boldsymbol{c}} = \tanh(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_j + \boldsymbol{U}^{(\tilde{c})}\tilde{\boldsymbol{h}}_j + \boldsymbol{b}^{(\tilde{c})}). \tag{2.27}$$

$$\boldsymbol{c}_j = \sum_{k \in C(j)} \boldsymbol{f}_{jk} \odot \boldsymbol{c}_k + \boldsymbol{i}_j \odot \tilde{\boldsymbol{c}}_j, \tag{2.28}$$

$$\boldsymbol{h}_j = \boldsymbol{o}_j \odot \tanh(\boldsymbol{c}_j). \tag{2.29}$$

Here, for each child k we calculate a forget gate $\boldsymbol{f}_{jk}$. In this architecture, each component is conditioned on the sum of children hidden states $\boldsymbol{h}_k$. Thus, it is a good option when the branching factor is high, and the order of children is not important.

N-ary Tree-LSTM: This kind of Tree-LSTM is designed for the cases that the branching factor is at most N and the order is important. The mathematical equations of this model are as follows:

$$\boldsymbol{i}_j = \sigma(\boldsymbol{W}^{(i)}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{U}_l^{(i)}\tilde{\boldsymbol{h}}_{jl} + \boldsymbol{b}^{(i)}), \tag{2.30}$$

$$\boldsymbol{f}_{jk} = \sigma(\boldsymbol{W}^{(f)}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{U}_{kl}^{(f)}\tilde{\boldsymbol{h}}_{jl} + \boldsymbol{b}^{(f)}), \tag{2.31}$$

$$\boldsymbol{o}_j = \sigma(\boldsymbol{W}^{(o)}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{U}_l^{(o)}\tilde{\boldsymbol{h}}_{jl} + \boldsymbol{b}^{(o)}), \tag{2.32}$$

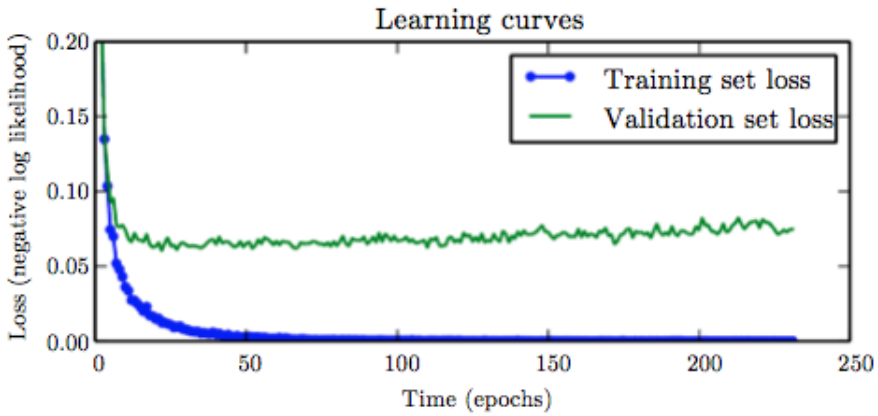


Figure 2.5: An example of learning curves that show the behaviour of the negative log-likelihood loss for the training set and validation set; Image source: (Goodfellow et al., 2016).

$$\tilde{\boldsymbol{c}} = \tanh(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_j + \sum_{l=1}^{N} \boldsymbol{U}_l^{(\tilde{c})}\tilde{\boldsymbol{h}}_{jl} + \boldsymbol{b}^{(\tilde{c})}), \tag{2.33}$$

$$\boldsymbol{c}_j = \sum_{l=1}^{N} \boldsymbol{f}_{jl} \odot \boldsymbol{c}_{jl} + \boldsymbol{i}_j \odot \tilde{\boldsymbol{c}}_j, \tag{2.34}$$

$$\boldsymbol{h}_j = \boldsymbol{o}_j \odot \tanh(\boldsymbol{c}_j). \tag{2.35}$$

In contrast with Child-sum Tree-LSTM that shares a set of weights among children, this model keeps a different set of weights for each child.

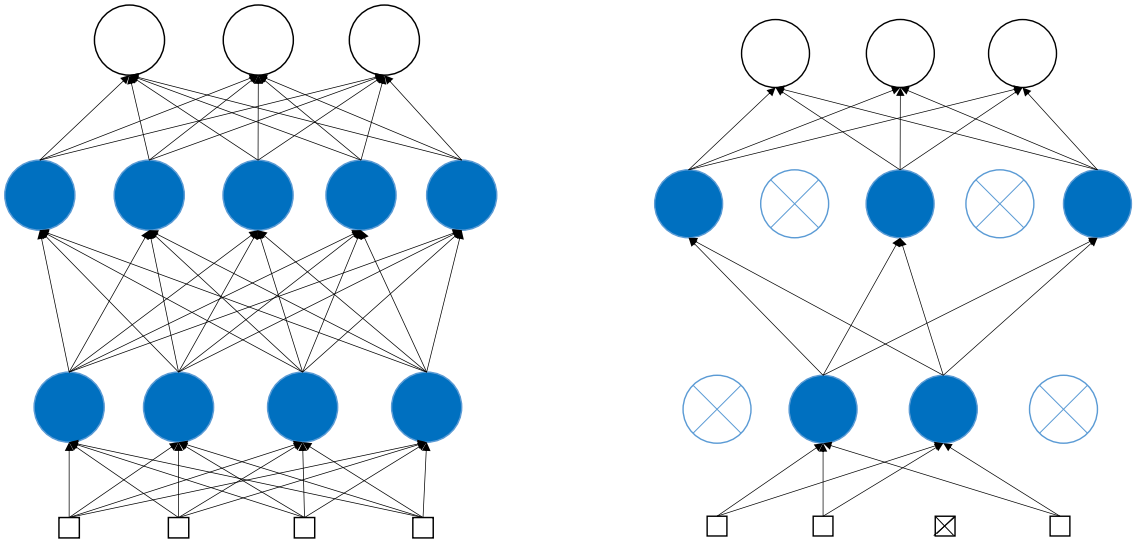
2.1.5 Regularisation

A major challenge in deep learning and general machine learning is to develop a method that performs well not just on the training data but also on new inputs. Regularisation is an umbrella name for strategies that help to prevent a machine learning model from overfitting and reduce test errors, possibly at the expense of an increase in the training error (Goodfellow et al., 2016). In the following, we discuss two popular methods for regularisation in deep models.

2.1.5.1 Early Stopping

When we train a model with a large set of parameters, the model may overfit the task. We often observe the error in the training set steadily decreases while at some point, error in the validation set starts to increase (Figure 2.5).

In this approach, we save a copy of the model parameters when the error on the validation set decreases. We run an optimisation algorithm until the error of the



(a) Standard neural network (b) After applying dropout

Figure 2.6: Dropout out mechanism. Left: a standard two-layer network. Right: a thinned version of the network after applying dropout and dropping crossed units (Srivastava et al., 2014).

validation set has not decreased for some amount of time. After termination, the last saved model is returned. By using this approach, we obtain a model with a better validation set error which hopefully results in a better test set error.

2.1.5.2 Dropout

Large neural networks are usually slow to run and train; thus, it is challenging to overcome overfitting by techniques like bagging (Breiman, 1996) that combines many different models. Dropout is proposed to deal with this problem (Srivastava et al., 2014). The idea is to drop some neurons and their corresponding connections randomly. This approach prevents co-adapting of neurons.

The standard dropout randomly drops neurons from non-output layers at the training time. It uses an independent Bernoulli random variable with parameter p for each neuron to decide whether it should be dropped or not. This process can be thought as sampling from an exponential number of different "thinned" networks as depicted in Figure 2.6. If the network has n neurons in the input and hidden layers, then this process samples from a set of 2^n different networks. As averaging the predictions of exponentially many "thinned" models is not feasible in the test time, authors proposed a simple approximate averaging method. During the test, they use the original network without dropping neurons. In this network, outgoing weights

of each neuron are multiplied by p to obtain a scaled-down version of the trained weights.

Dropout is widely used in the modern deep learning models, and have shown empirically to significantly lower generalisation error. Some works investigate a Bayesian perspective of the dropout and propose different variants of it. Wang and Manning (2013) shows that applying dropout in training can be seen as a Monte Carlo approximation. They propose Gaussian dropout to do fast dropout training by sampling from or integrating a Gaussian approximation instead. Kingma et al. (2015) proposes Variational dropout, which is a generalisation of Gaussian dropout, and optimal dropout rates are inferred from the data. Gal and Ghahramani (2016) proposes Monte Carlo dropout, which applies dropout at test time as well. At test time, it performs T stochastic forward passes through the network and averages the results.

2.1.6 Deep Learning for NLP

2.1.6.1 Word Embedding

Word is a building block of language; similarly, word embedding is a building block of NLP. Each language consists of a collection of words, aka dictionary. In order to feed in/out words from a neural network, we need to convert them into numerical representations. There are three ways to represent words:

- **Dictionary Lookup**. This is the simplest form which a word is represented by its index in the dictionary. The problem is that it imposes an explicit ordering on words, e.g., the model may put more importance on words with higher index while there is no ordering in words.

- **One-Hot Encoding**. This approach addresses the ordinal representation of the previous approach by representing each word in a one-hot encoding format. The idea is to create a binary vector with the size of vocabulary where all elements are zero except one. For representing a word, its corresponding columns will be filled by 1. This immense and sparse representation of input/output provokes a need for larger weight matrices in input/output layers and imposes memory and computation burden.

- **Distributed Representation**. In this approach, words are mapped into a real-valued vector in low-dimensional continuous space where each dimension is responsible for a latent feature of the word. Ideally, in the mapped space, semantically similar words would be located close to each other. Although the

embedding table of words can be learned end-to-end, it also can be pre-trained and used for transfer learning. There are different approaches for training the word embeddings; however, the idea behind all of them is that semantically similar words tend to appear within the same context. There are two main types of pre-training methods: (1) prediction-based methods which learn the word embeddings in such a way to improve the predictive ability of the target word given context words, e.g. Word2Vec (Mikolov et al., 2013) for local context and Global Vectors (GloVe) (Pennington et al., 2014) for global context; (2) count-based methods that construct co-occurrence counts matrix with the aim to capture global statistics and perform dimensionality reduction techniques over the matrix (Church and Hanks, 1989; Deerwester et al., 1990; Turney and Pantel, 2010; Levy et al., 2015).

2.1.6.2 Statistical Language Modelling

Statistical Language Models (LM) are one of the key building blocks in most of the natural language processing tasks, including machine translation. A statistical language model predicts the probability of a sequence of tokens, e.g., words $(w_1 w_2 ... w_T)$. It is used to predict the next token (e.g., character, word, sentence) considering the preceding context. At time t, the probability of token w_t given its proceedings is:

$$p(w_i|w_{i-1}, ..., w_1). \tag{2.36}$$

And, applying it for T times gives the probability of the sequence:

$$p(\boldsymbol{w}) = p(w_1, w_2, .., w_T) = \prod_{i=1}^{T} p(w_i|w_{i-1}, ..., w_1). \tag{2.37}$$

n-gram Language Models Considering millions of words in the vocabularies of languages, a fundamental problem which makes the language modelling difficult is the *curse of dimensionality*. A practical approach for reducing the curse of dimensionality is to take advantage of word orders and approximate the history by just a few preceding words.

An n-gram language model applies Markov assumption to condition each word only on n preceding words:

$$p(\boldsymbol{w}) = p(w_1, w_2, .., w_T) \approx \prod_{i=1}^{T} p(w_i|w_{i-1}, .., w_{i-n+1}) \tag{2.38}$$

The main issues in the n-gram LMs are as follows:

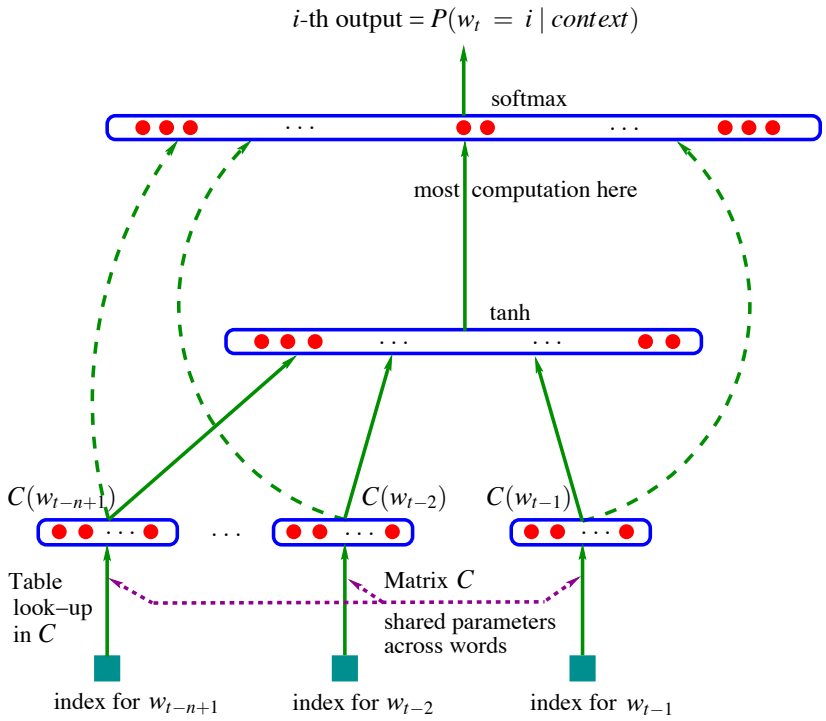


Figure 2.7: Architecture of an n-gram neural probabilistic language model; Image source: (Bengio et al., 2003).

- **Sparsity** Estimating the conditional probability in eqn. 2.38 using the maximum likelihood estimate results in:

$$p(w_i|w_{i-1},..,w_{i-n+1}) \approx \frac{\text{Count}(w_i, w_{i-1},..,w_{i-n+1})}{\text{Count}(w_{i-1},..,w_{i-n+1})},$$

  In fact, conditional probability is estimated by the frequency ratio of co-occurrence. This estimation suffers from the *sparsity problem*. It may arise when a given word sequence has not been observed during the training, or the test data contains unseen words aka out-of-vocabulary. This problem is exacerbated when the training data is small, e.g., in low-resource machine translation.

- **The curse of dimensionality** Usually languages contain millions of words and even with a small n, the matrix of co-occurrence would be enormously large.

- **Fixed-length context** Applying the Markov assumption makes the n-gram language model restrictive to a fixed-length context. Therefore, it would be unable to capture long term dependencies.

Neural Probabilistic Language Model Bengio et al. (2003) proposed the first Neural Probabilistic Language Model (NPLM) to address the sparsity and curse of dimensionality issues in n-gram LMs by learning a distributed representation for words. As depicted in Figure 2.7, the core idea is to convert words to distributed embeddings and use a feed-forward neural network to predict the next word. As the feed-forward neural network requires a fixed-size input, each word can be conditioned on fixed-size window of its context $p(w_i|w_{i-1}, ..., w_{i-n+1})$.

Recurrent Neural Network Language Model Although NPLM addresses the sparsity and curse of dimensionality issues of the n-gram LM, it still suffers from the fixed-length context. Mikolov et al. (2010, 2011) address this issue by proposing a Recurrent Neural Network Language Model (RNNLM) by replacing the feed-forward component in the NPLM by a recurrent neural network component. By using a generative RNN, the model would be able to process arbitrary-length context. At each time step i, the hidden state of the unit is calculated as follows:

$$\boldsymbol{h}_i = \text{RNN}(\boldsymbol{h}_{i-1}, \boldsymbol{E}[w_i])$$

where $\boldsymbol{E}$ is the distributed embedding table. The hidden state of RNN will capture the context for all the previous words $(w_{i-1}, ..., w_1)$ and the probability of the i-th words can be determined as follows:

$$w_i \sim \text{softmax}(\boldsymbol{W}_{oh}\boldsymbol{h}_t).$$

2.2 Neural Machine Translation

Neural Machine Translation refers to machine translation based purely on neural networks. An NMT model often is composed of an encoder to *read* the input sentence, and a decoder to *generate* the output. The encoder reads the source sentence yielding fixed-length vector representation(s) which then are used by a decoder to generate translation. As the translation task is intrinsically transduction of a sequence to another one, a sequence-to-sequence (Seq2Seq) model is a natural option.

2.2.1 Seq2Seq model

Sutskever et al. (2014) proposed the first Seq2Seq model. As shown in Figure 2.9, the encoder reads the words in the source sentence sequentially and generates a vector representation as the summary of the source sentence. The summary vector is then

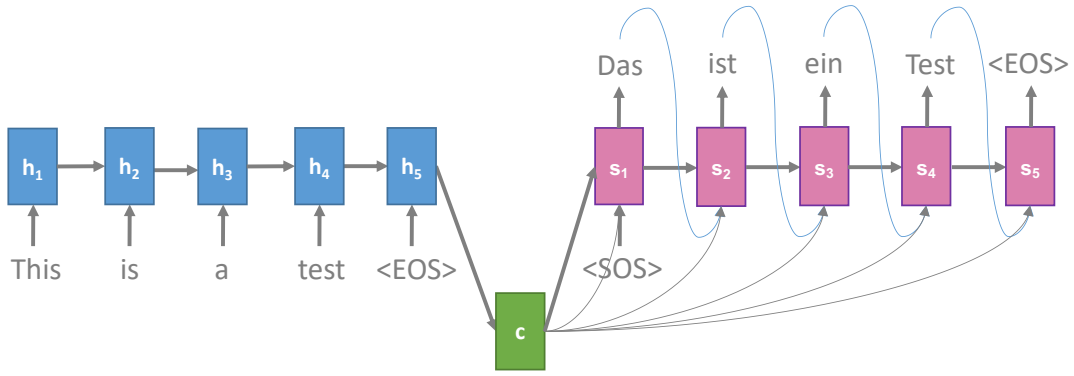


Figure 2.8: Example of SEQ2SEQ model for English to German translation.

used by the decoder as the context to generate the target sentence word by word. As the length of sentences can have arbitrary lengths, a special end-of-sentence (EOS) symbol is used to signal encoder and decoder about the end of the sentence.

Encoder. The encoder is a uni-directional RNN whose hidden states represent tokens of the input sequence. These representations capture information not only of the corresponding token but also other tokens in the sequence to leverage the context. The RNN runs in the left-to-right direction over the input sequence:

$$\boldsymbol{h}_i = \text{RNN}(\boldsymbol{h}_{i-1}, \boldsymbol{E}_S[x_i]) \tag{2.39}$$

where $\boldsymbol{E}_S[x_i]$ is the embedding of the token x_i from the embedding table $\boldsymbol{E}_S$ of the input (source) space, and $\boldsymbol{h}_i$ is the hidden states of the RNNs which can be based on the LSTM or GRU units. The hidden state, after reading the last symbol is the fixed-length vector representation of the source sentence, which is called context ($\boldsymbol{c} = \boldsymbol{h}_T$).

Decoder. The backbone of the decoder is a uni-directional RNN which generates the token of the output one-by-one from left to right. The generation of each token y_j is conditioned on all of the previously generated tokens $\boldsymbol{y}_{<j}$ via the state of the RNN decoder $\boldsymbol{s}_j$, and the input sequence via a *fixed* context vector $\boldsymbol{c}$:

$$y_j \sim \text{softmax}(\boldsymbol{W}_y \cdot \boldsymbol{r}_j + \boldsymbol{b}_r) \tag{2.40}$$

$$\boldsymbol{r}_j = \tanh(\boldsymbol{s}_j + \boldsymbol{W}_{rc} \cdot \boldsymbol{c} + \boldsymbol{W}_{rj} \cdot \boldsymbol{E}_T[y_{j-1}]) \tag{2.41}$$

$$\boldsymbol{s}_j = \tanh(\boldsymbol{W}_s \cdot \boldsymbol{s}_{j-1} + \boldsymbol{W}_{sj} \cdot \boldsymbol{E}_T[y_{j-1}] + \boldsymbol{W}_{sc} \cdot \boldsymbol{c}) \tag{2.42}$$

where $\boldsymbol{E}_T[y_j]$ is the embedding of the token y_j from the embedding table $\boldsymbol{E}_T$ of the output (target) space, and the $\boldsymbol{W}$ matrices and $\boldsymbol{b}_r$ vector are the parameters.

Training and Decoding. Suppose we are given a training set $\mathcal{D} := \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=0}^{N}$, the model parameters are trained end-to-end by maximising the (regularised) log-likelihood of the training data:

$$\arg\max_{\boldsymbol{\Theta}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log P_{\boldsymbol{\Theta}}(\boldsymbol{y}|\boldsymbol{x}),$$

where according to eqn. 2.40 it can be defined as:

$$\arg\max_{\boldsymbol{\Theta}} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \sum_{j=1}^{|\boldsymbol{y}|} \log P_{\boldsymbol{\Theta}}(y_j|\boldsymbol{y}_{<j}, \boldsymbol{x}).$$

In the decoding time, the best output sequence for a given input sequence is produced by

$$\arg\max_{\boldsymbol{y}} P_{\boldsymbol{\Theta}}(\boldsymbol{y}|\boldsymbol{x}) = \prod_{j} P_{\boldsymbol{\Theta}}(y_j|\boldsymbol{y}_{<j}, \boldsymbol{x}).$$

Greedy decoding or beam search algorithms can be employed to find an approximate solution, since solving the above optimisation problem exactly is computationally hard.

The bottleneck of this model is that the model should compress all the necessary information of source sentence into a fixed-length vector. Cho et al. (2014a) showed that this method performs relatively well on short sentences, but its performance deteriorates rapidly with the increase in the length of the source sentence.

2.2.2 Attentional Seq2Seq model

To address the bottleneck of fixed-vector representation in the SEQ2SEQ model, Bahdanau et al. (2015) proposed an attention mechanism which dynamically attends to relevant parts of the input sequence necessary for generating the next token in the output sequence; see Figure 2.9. Moreover, attentional SEQ2SEQ model is usually equipped with a bi-directional encoder to capture context of both past and future.

Bi-directional encoder. The encoder is a bi-directional RNN consists of two RNNs running in the left-to-right and right-to-left directions over the input sequence:

$$\overrightarrow{\boldsymbol{h}_i} = \mathrm{RNN}(\overrightarrow{\boldsymbol{h}}_{i-1}, \boldsymbol{E}_S[x_i]) \tag{2.43}$$

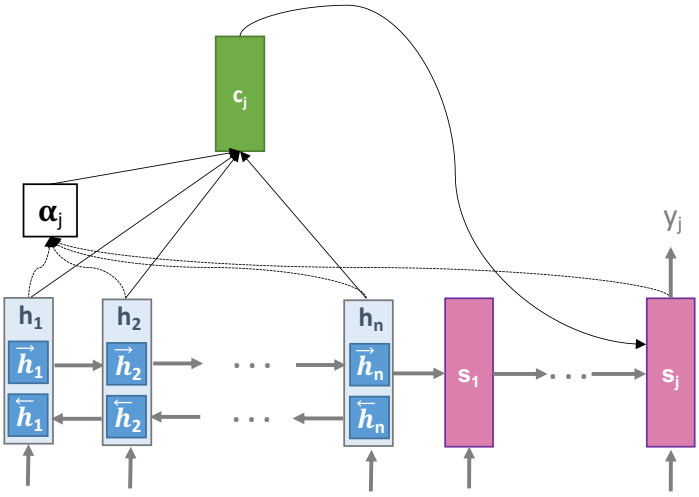


Figure 2.9: Attentional Encoder-Decoder model.

$$\overleftarrow{\boldsymbol{h}}_i = \text{RNN}(\overleftarrow{\boldsymbol{h}}_{i+1}, \boldsymbol{E}_S[x_i]) \tag{2.44}$$

where $\overrightarrow{\boldsymbol{h}}_i$ and $\overleftarrow{\boldsymbol{h}}_i$ are the hidden states of the forward and backward RNNs which can be based on the LSTM or GRU units. Each source token is then represented by the concatenation of the corresponding bidirectional hidden states, $\boldsymbol{h}_i = [\overrightarrow{\boldsymbol{h}}_i; \overleftarrow{\boldsymbol{h}}_i]$.

Attentional decoder. The backbone of the decoder is similar to SEQ2SEQ model with the difference that the generation of each token y_j is conditioned on the input sequence via a *dynamic* context vector $\boldsymbol{c}_j$ (explained shortly):

$$y_j \sim \text{softmax}(\boldsymbol{W}_y \cdot \boldsymbol{r}_j + \boldsymbol{b}_r) \tag{2.45}$$

$$\boldsymbol{r}_j = \tanh(\boldsymbol{s}_j + \boldsymbol{W}_{rc} \cdot \boldsymbol{c}_j + \boldsymbol{W}_{rj} \cdot \boldsymbol{E}_T[y_{j-1}]) \tag{2.46}$$

$$\boldsymbol{s}_j = \tanh(\boldsymbol{W}_s \cdot \boldsymbol{s}_{j-1} + \boldsymbol{W}_{sj} \cdot \boldsymbol{E}_T[y_{j-1}] + \boldsymbol{W}_{sc} \cdot \boldsymbol{c}_j) \tag{2.47}$$

where $\boldsymbol{E}_T[y_j]$ is the embedding of the token y_j from the embedding table $\boldsymbol{E}_T$ of the output (target) space, and the $\boldsymbol{W}$ matrices and $\boldsymbol{b}_r$ vector are the parameters.

A crucial element of the decoder is the *attention* mechanism which dynamically attends to relevant parts of the input sequence necessary for generating the next token in the output sequence. Before generating the next token y_j, the decoder computes the attention vector $\boldsymbol{\alpha}_j$ over the input token:

$$\boldsymbol{\alpha}_j = \text{softmax}(\boldsymbol{a}_j)$$

$$a_{ji} = \boldsymbol{v} \cdot \tanh(\boldsymbol{W}_{ae} \cdot \boldsymbol{h}_i + \mathbf{W}_{at} \cdot \boldsymbol{s}_{j-1})$$

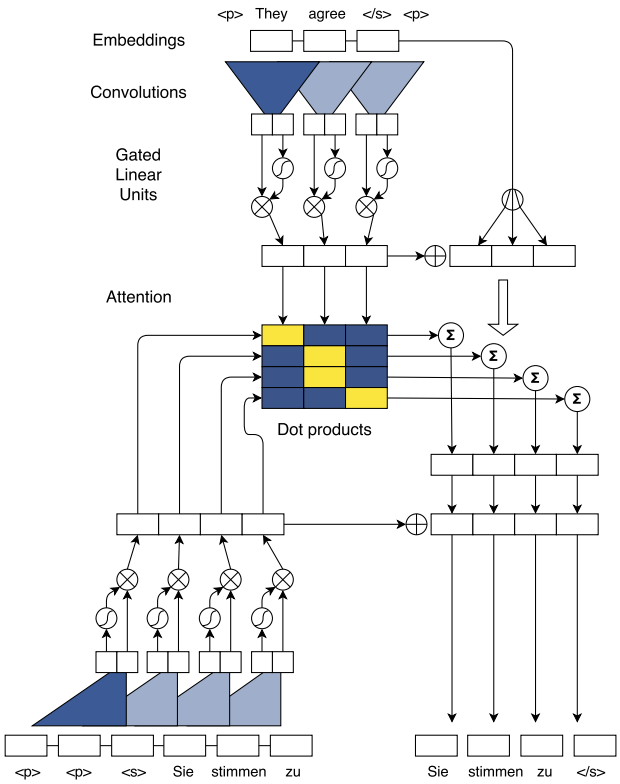


Figure 2.10: The general architecture of a convolutional SEQ2SEQ model. The English source sentence (top) is encoded, and the attention for four target words are calculated simultaneously (middle). Image source: (Gehring et al., 2017).

which intuitively is similar to the notion of *alignment* in word/phrase-based statistical MT (Brown et al., 1993). The attention vector is then used to compute a fixed-length dynamic representation of the source sentence

$$\boldsymbol{c}_j = \sum_i \alpha_{ji} \boldsymbol{h}_i. \tag{2.48}$$

which is conditioned upon in the RNN decoder when computing the next state or generating the output word (as mentioned above).

The RNN-based attentional SEQ2SEQ models have two limitations. First, as RNN needs to maintain a hidden state that imposes sequential dependency, it cannot be parallelised. Second, RNNs, even including LSTM, are infamous for their weakness in capturing long-term dependencies.

2.2.3 Convolutional Seq2Seq model

To address the parallelisation and to some extent the long-term dependency issues of RNN-based SEQ2SEQ model, convolutional based neural networks (LeCun and

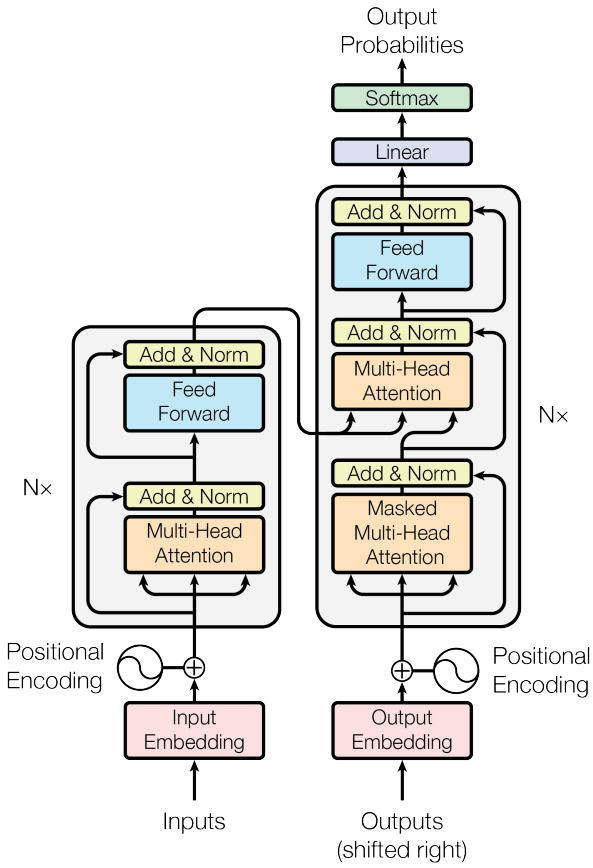


Figure 2.11: The general architecture of the Transformer model. Image source: (Vaswani et al., 2017).

Bengio, 1998) are proposed e.g., ByteNet (Kalchbrenner et al., 2017) and ConvS2S (Gehring et al., 2017). The high-level architecture of ConvS2S model is depicted in Figure 2.10. As seen, convolutional layers operate over a fixed-size window of input tokens and therefore can be operated in parallel. Although the generated representations are for a fixed-size context, stacking convolutional layers leads to a larger effective window size. A multi-layer convolutional neural network creates a hierarchical representation over the input sentence where lower layer capture nearby dependencies and higher layers capture long-term dependencies. Assume we are given a sentence with n tokens, a convolutional network with the kernel size of k can capture dependencies with applying $\mathcal{O}(\frac{n}{k})$ operations. The number of operations for a recurrent unit is linear to the size of input $\mathcal{O}(n)$.

2.2.4 Self-attention Seq2Seq model

Although convolutional neural network-based models are able to parallelise the training of SEQ2SEQ model and capture longer dependencies than RNN-based models, they still suffer from the inability to capture long-term dependencies. Vaswani et al. (2017) proposed Transformer, an architecture solely based on self-attention. The Transformer can be seen as an adaptive weighted convolutional kernel with the window-length of the sequence size. In the convolution kernel, the weights are dependent on the position and not content. Therefore, when we slide the windows through the sequence, weights are fixed. However, in the Transformer, weights are determined with respect to the representation of the current position and the representation of other positions. In other words, the idea is that the position should not be a limitation, and if two tokens are related, they should have higher weights for each other even if they are distant.

As depicted in Figure 2.11, each layer is consists of a Multi-head attention sublayer followed by a feed-forward neural network. The difference between the encoder and the decoder is two-fold: (1) the decoder also attends to the encoder representations; (2) the decoder does not have attention to the future.

Attention Mechanism in Transformer Transformer is solely based on self-attention, and benefits from two extensions to the current attention mechanism: (1) scaled dot-product attention; (2) multi-head attention.

Vaswani et al. (2017) proposed a scaled version of the dot-product attention where the input to the attention mechanism is consist of queries and keys with d_k dimensions and values with d_v dimensions. The scaled dot-product attention is similar to vanilla dot-product attention that is scaled by the factor of $\sqrt{d_k}$:

$$\text{ScaledAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{2.49}$$

The scaling extension is proposed to address the poor performance of the vanilla dot-product attention in comparison to additive attention (Bahdanau et al., 2015) for large values of d_k (Britz et al., 2017). It is suspected that for large values of d_k, the vanilla dot-product grows large in magnitude and pushes the softmax function into regions with extremely small gradients, and scaling can fix this issue.

The Transformer also uses a novel multi-head attention mechanism with the idea of projecting the key, query and value into several representation subspaces, and jointly attend to information in these subspaces. Intuitively, each subspace may capture a

different kind of information, and the result could be considered as an ensemble of attentions.

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Concat}(\text{head}_1, ..., \text{head}_h) \tag{2.50}$$

$$\text{where head}_i = \text{ScaledAttention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \tag{2.51}$$

where $\boldsymbol{W}_i$s are learned parameters. The Transformer applies the mentioned attention mechanism in three different ways:

1. **Self-attention in Encoder** In order to obtain the representation of the source sentence, the Transformer employs self-attention where the $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ come from the encoder, i.e., embedding of words for the first layer or embedding of the previous layer for the rest.

2. **Cross-attention from Decoder to Encoder** As the generation of the target sequence should be conditioned on the input sentence, the decoder needs to attend to the relevant part of the source sentence. Therefore, the attention mechanism is employed with queries $\boldsymbol{Q}$ that comes from the previous layer of decoder along with key $\boldsymbol{K}$ and values $\boldsymbol{V}$ from the encoder output.

3. **Self-attention in Decoder** In order to obtain the representation of the generated sequence so far, self-attention is applied on $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ come from the decoder, i.e., embedding of generated words for the first layer or embedding of the previous layer for the rest.

Positional embedding Unlike RNN, the Transformer does not process input tokens with respect to their temporal position. Therefore, it needs to be informed about the relative or absolute position of the tokens in the sequence in order to make use of sequence order. To encode the position, Vaswani et al. (2017) used sinusoidal functions with different frequencies to extrapolate to sequence lengths even larger than those observed during the training. Finally, the position embedding is added to the word embeddings to enrich them with positional information.

2.2.5 Evaluation metrics

There are several methods for evaluating machine translation models. In this section, we explain four popular evaluation measures.

Perplexity (PPL): is the inverse probability of the translation sentence, normalised by the number of words.

$$PPL(\boldsymbol{y}) = \sqrt[T]{\frac{1}{\prod_{i=1}^{T} p(y_i|y_{<i}, \boldsymbol{x})}}. \tag{2.52}$$

Intuitively, PPL is a measure to determine "how confused is the model about its decision?" (Neubig, 2017).

TER (Snover et al., 2006): Translation Edit Rate (TER) is a machine translation measure that determines the amount of post-editing required for a generated translation to exactly matches one of the references. It counts the minimum number of possible edits including insertion, deletion, shifts of sequence and substitution of words, and normalise it with the average weight of the references:

$$\text{TER} = \frac{\text{Minimum \# of required edits}}{\text{Average \# of reference words}} \tag{2.53}$$

BLEU[1] **(Papineni et al., 2002)**: The idea behind this measure is "the closer a machine translation is to a professional human translation, the better it is" (Papineni et al., 2002). This measure has a high correlation with the human judgment of quality and is one of the most popular evaluation measures for machine translation. It is based on modified precision for n-grams. The n-gram is a contiguous sequence of words in a given sequence. Modified n-gram precision for a candidate translation generated by the model is calculated:

$$p_n = \frac{\sum_{n\text{-gram}\in\{\text{candidate translation }\}} \text{Count}_{clip}(n\text{-gram})}{\sum_{n\text{-gram}'\in\{\text{candidate translation }\}} \text{Count}(n\text{-gram}')}, \tag{2.54}$$

where Count(n-gram) is the number of mutual n-grams in a candidate translation and true translation. $\text{Count}_{clip}(n\text{-gram})$ is the number of mutual n-grams clipped by the maximum repetition of n-grams in the true translation.

Next, for a candidate with the length c and the true translation of length r, the BLEU score is calculated as follows:

$$\text{BLEU} = \text{BP} \cdot \left(\sum_{n=1}^{N} w_n \log p_n \right), \tag{2.55}$$

where w_n is weight for the modified n-gram precision and usually is uniform. BP is the Brevity Penalty:

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases} \tag{2.56}$$

METEOR[2] **(Banerjee and Lavie, 2005)**: This score measures sentence-level

[1]BiLingual Evaluation Understudy

[2]Metric for Evaluation for Translation with Explicit Ordering

similarity scores by explicitly aligning words between the generated translation and a given reference translation. The matching of two words is done by applying the following matchers: Exact (identical surface form), Stem (identical stem), Synonym (common synonym). It then calculates an F-score:

$$F_{mean} = \frac{P \cdot R}{\alpha \cdot P + (1 - \alpha) \cdot R}, \tag{2.57}$$

where P and R are precision and recall based on single-word matches. In the next step, METEOR penalises the translation concerning the order of matched words in the generated translation and reference. The penalty coefficient is $Pen \propto c/m$ where c is the smallest number of "chunks" of matched words such that the words in each chunk are adjacent (in the generated translation and reference) and in same word order, and m is the number of matched words:

$$\text{METEOR} = (1 - Pen) \cdot F_{mean}. \tag{2.58}$$

Statistical Significance: One approach for measuring the statistical significance is to use approximate randomisation (AR) test (Clark et al., 2011). It randomly exchanges sentences between the generated translations and references, and estimates the p-value that a measured score arose by chance. Clark et al. (2011) provides a tool[3] that calculates statistical significance for TER, BLEU and METEOR scores.

2.3 Low-Resource Neural Machine Translation

Like many deep learning methods, NMT requires a large amount of annotated data, i.e., bilingual sentence pairs, to train a model with a reasonable translation quality (Koehn and Knowles, 2017). However, for many languages, we do not have the luxury of having large parallel datasets, a setting referred to as bilingually low-resource scenario. Therefore, it is critical to compensate for the lack of sizeable bilingual train data using effective approaches.

The difference between low- and high-resource NMT is more than bilingual data availability! It has been shown that in bilingually low-resource scenarios, Phrase-Based Statical Machine Translation (PBSMT) models outperform NMT models while for the high-resource the situation is reversed (Koehn and Knowles, 2017; Lample et al., 2018b). Other works reported different improvements of a model or

[3]https://github.com/jhclark/multeval

technique in different data condition regimes (Qi et al., 2018; Kiperwasser and Ballesteros, 2018). Recently, Sennrich and Zhang (2019) re-visited low-resource NMT and showed that low-resource NMT is very sensitive to hyperparameters, architectural design and other design choices. They have shown that typical settings for high-resource are not the best for low-resource, and NMT can be unleashed and outperform PBSMT in low-resource scenarios with a proper setting. However, note that the performance of an NMT model trained with hundreds of bilingual pairs still is much less than the one trained with millions.

A practical approach to compensate for the lack of bilingual data in low-resource NMT is to use auxiliary data. The auxiliary data could be in the form of curated monolingual linguistic resources, monolingual sentences or multilingual sentence pairs, i.e., a mixture of high- and low-resource. In this thesis, we will focus on the first type and leverage curated monolingual for compensating the shortage of bilingual training data. These resources are mostly available in the form of annotated datasets, e.g., treebanks for syntactic parsing or part-of-speech tagged sentences. There are two main approaches to incorporate these resources into the training of an NMT model: (1) Using these monolingual resources to build tools which can then be used to annotate source sentences (covered in Section 2.3.1); (2) Directly injecting linguistic knowledge into the translation model (covered in Section 2.3.2). There are other approaches for improving low-resource NMT that are mainly focusing on using auxiliary monolingual and multilingual data, and we will cover them in Section 2.3.3.

2.3.1 Incorporate Linguistic Annotation by Transduction of Complex Structures

Attentional tree-to-sequence Model

Eriguchi et al. (2016) proposed a method to incorporate the hierarchical syntactic information of the source sentence. Their method is based on the hypothesis that incorporating syntactic information in NMT can lead to better reorderings, particularly useful when the language pairs are syntactically highly divergent or when the training bitext is not large. Their model is an extension to the attentional Seq2Seq model (Section 2.2.2) by calculating embeddings of phrases in addition to the words. Then, in the decoding process, it will attend on both words and phrases to take into account the hierarchical syntactic information of the source sentence; see Figure 2.12. Phrase embeddings are calculated in a bottom-up fashion which is directed by the

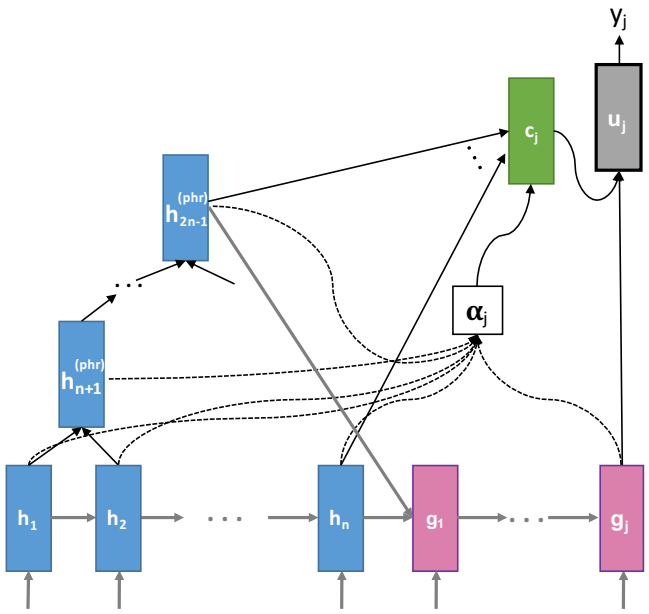


Figure 2.12: Attentional Tree-to-Sequence model.

parse tree of the source sentence. As generating gold-standard parse trees is not possible in real-world scenarios, the author proposed to use binary constituency parse trees[4] generated by an off-the-shelf parser.

Tree Encoder It consists of sequential and recursive parts. The sequential part is the vanilla sequence encoder discussed in Section 2.2.1, which calculates the embeddings of words. Then, the embeddings of phrases are calculated using the embeddings of their constituent words in a recursive bottom-up fashion:

$$\boldsymbol{h}_k^{(phr)} = \text{TreeLSTM}(\boldsymbol{h}_k^l, \boldsymbol{h}_k^r).$$

where $\boldsymbol{h}_k^l$ and $\boldsymbol{h}_k^r$ are hidden states of left and right children respectively. This method uses TreeLSTM units (Tai et al., 2015) to calculate the embedding of a parent node using its two children units as follow:

$$\begin{aligned}
\boldsymbol{i} &= \sigma(\boldsymbol{U}_l^{(i)}\boldsymbol{h}^l + \boldsymbol{U}_r^{(i)}\boldsymbol{h}^r + \boldsymbol{b}^{(i)}) \\
\boldsymbol{f}^l &= \sigma(\boldsymbol{U}_l^{(f_l)}\boldsymbol{h}^l + \boldsymbol{U}_r^{(f_l)}\boldsymbol{h}^r + \boldsymbol{b}^{(f_l)}) \\
\boldsymbol{f}^r &= \sigma(\boldsymbol{U}_l^{(f_r)}\boldsymbol{h}^r + \boldsymbol{U}_r^{(f_r)}\boldsymbol{h}^r + \boldsymbol{b}^{(f_r)}) \\
\boldsymbol{o} &= \sigma(\boldsymbol{U}_l^{(o)}\boldsymbol{h}^l + \boldsymbol{U}_r^{(o)}\boldsymbol{h}^r + \boldsymbol{b}^{(o)}) \\
\tilde{\boldsymbol{c}} &= \tanh(\boldsymbol{U}_l^{(\tilde{c})}\boldsymbol{h}^l + \boldsymbol{U}_r^{(\tilde{c})}\boldsymbol{h}^r + \boldsymbol{b}^{(\tilde{c})})
\end{aligned}$$

[4]A constituency parse tree breaks a sentence into sub-phrases.

$$\begin{aligned} \boldsymbol{c}^{(phr)} &= \boldsymbol{i} \odot \tilde{\boldsymbol{c}} + \boldsymbol{f}^l \odot \boldsymbol{c}^l + \boldsymbol{f}^r \odot \boldsymbol{c}^r \\ \boldsymbol{h}^{(phr)} &= \boldsymbol{o} \odot \tanh(\boldsymbol{c}^{(phr)}) \end{aligned}$$

where $\boldsymbol{i}$, $\boldsymbol{f}^l$, $\boldsymbol{f}^r$, $\boldsymbol{o}_j$,$\tilde{\boldsymbol{c}}_j$ are the input gate, left and right forget gates, output gate, and a state for updating memory cell; $\boldsymbol{c}^r$ and $\boldsymbol{c}^l$ are memory cells of the right and left units.

Sequential Decoder Eriguchi et al. set the initial state of the decoder by combining the final state of the sequential and tree encoders as follow:

$$\boldsymbol{g}_0 = \text{TreeLSTM}(\boldsymbol{h}_n, \boldsymbol{h}_{root}^{(phr)}),$$

The rest of the decoder is similar to the vanilla attentional decoder discussed in Section 2.2.2. The difference is that, in this model, the attention mechanism makes use of phrases as well as words. Thus, the dynamic context is calculated as follows:

$$\boldsymbol{c}_j = \sum_{i=1}^{n} \alpha_{ji} \boldsymbol{h}_i + \sum_{i'=n+1}^{2n-1} \alpha_{ji'} \boldsymbol{h}_{i'}^{phr}$$

Chen et al. (2017a) extended this work by proposing a bidirectional tree encoder. Though the results for Tree2Seq models are promising, the top-1 trees are prone to the parser error, and cannot capture semantic ambiguities of the source sentence. In Chapter 3, we address the issues mentioned above by using combinatorially many trees encoded in a forest instead of a single top-1 parse tree. We capture the parser uncertainty by considering many parse trees along with their probabilities using our ForestLSTM architecture. Following our work, Ma et al. (2018) has also proposed a forest-based NMT model with a different approach of linearising the forest and using a Seq2Seq model.

Attentional graph-to-sequence Model

A graph is a mathematical structure to capture relations between entities and is able to capture more generalised structures than a tree. Graph-to-sequence models are based on encoding the relationship among the words in the source sentences where the relationship among the words are given in the form of a graph.

Bastings et al. (2017) proposed a graph-to-sequence method to incorporate the dependency parse tree of the source sentence. In the dependency parse tree, each node corresponds to a word in the sentence, and a unique root node (ROOT) is added. For a sentence of length N, dependency parse tree can be represented by a

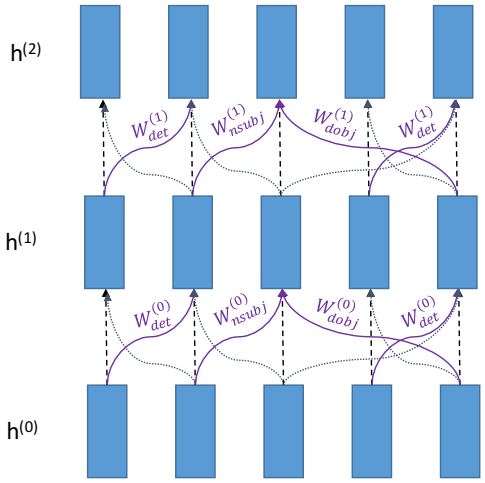


Figure 2.13: A two-layer syntactic GCN on top of the embeddings, updating them concerning a dependency parse tree. To simplify image, gates and some labels are removed. Image source: (Bastings et al., 2017)

.

set of tuples: $(w_i,\ H_{w_i},\ l_{w_i})$ where w_i is i-th word in the sentence, its parent node (head) is $H_{w_i} \in \{w_1, ..., w_n, Root\}$, and l_{w_i} is a dependency tag (label). The encoding process has two phases: First, they calculate an initial embeddings of words. Second, they refine and enrich these embeddings concerning the dependency tree.

Initial embeddings can be obtained from an embedding weight matrix or calculated using an encoder to incorporate contextual information. In the second phase, a syntactic Graph Convolutional Network (Marcheggiani and Titov, 2017) is used to update the initial embeddings. As shown in Figure 2.13, each layer updates the embeddings of nodes with respect to their neighbours in the graph (in this case dependency tree). It means that with k layers, nodes will receive information from the neighbours at most k hopes away. H-dimensional embedding of word v in the $(j+1)$-th layer is updated:

$$\boldsymbol{h}_v^{(j+1)} = \rho\left(\sum_{u \in N(v)} g_{u,v}^{(j)}\left(\boldsymbol{W}_{dir(u,v)}^{(j)}\boldsymbol{h}_u^{(j)} + \boldsymbol{b}_{lab(u,v)}^{(j)}\right)\right), \tag{2.59}$$

where $N(v)$ is neighbors of v in dependency tree, and ρ is activation function. $\boldsymbol{W}_{dir(u,v)}^{(j)} \in \mathbb{R}^{H\times H}$ is a weight matrix associated with the direction of edge between u and v, and $\boldsymbol{b}_{lab(u,v)}^{(j)} \in \mathbb{R}^H$ is a bias vector associated with the type of edge (dependency type). $g_{u,v}^{(j)}$ is an edge-wise gating layer to control the contribution of each edge, and

is calculated as follows:

$$g_{u,v}^{(j)} = \sigma\left(\boldsymbol{h}_u^{(j)}.\hat{\boldsymbol{w}}_{dir(u,v)}^{(j)} + \hat{b}_{lab(u,v)}^{(j)}\right), \tag{2.60}$$

where $\hat{\boldsymbol{w}}_{dir(u,v)} \in \mathbb{R}^H$ and $\hat{b}_{lab(u,v)} \in \mathbb{R}$ are learned parameters.

Marcheggiani et al. (2018) has applied the architecture mentioned above to incorporate semantic dependency graphs. Beck et al. (2018) has taken another approach for graph-to-sequence transduction by using Gated Graph Neural Networks (Li et al., 2015). Recently, Song et al. (2019) has proposed another variant by adopting Graph Recurrent Network (GCN) (Song et al., 2018) to incorporate semantic knowledge within an Abstract Meaning Representation (AMR) graph.

In a different direction, Hashimoto and Tsuruoka (2017) proposed an NMT model with source-side *latent* graph parsing. Their method is an extension of attentional SEQ2SEQ model which learn a latent graph parser as part of the encoder. They calculate two embeddings for each word. First, they run a sequential encoder on source sentence to calculate embeddings of words. Second, they calculate another embedding for each word with respect to the graph.

The graph structure is like a dependency tree with "soft" connections, and without constraint on the tree structure. This latent graph is presented as a set of tuples: (w_i, $p(H_{w_i}|w_i)$, $p(l_{w_i}|w_i)$) where $p(H_{w_i}|w_i)$ is the probability distribution of the parent of w_i, and $p(l_{w_i}|w_i)$ is the probability over dependency tags. In order to obtain this latent graph for a sentence, they used dependency parsing model proposed by Hashimoto et al. (2016). First, a bidirectional LSTM traverses over the sentence and calculates $\boldsymbol{h}_i^{(1)}$ for words. Then, for each word, the hidden state is fed to a softmax layer to predict a probability distribution $\boldsymbol{p}_i^{(1)} \in \mathbb{R}^{C^{(1)}}$ over Part Of Speech (POS) tags. Then, another bidirectional LSTM layer traverses over the sentence to calculate second layer embeddings. In order to calculate $\boldsymbol{h}_i^{(2)}$, the LSTM is fed with the concatenation of $\boldsymbol{h}_i^{(1)}$ and $\boldsymbol{W}_l^{(1)}\boldsymbol{p}_i^{(1)}$, where $\boldsymbol{W}_l^{(1)}$ is a weight matrix. After calculating these two layers, "soft" edges of the latent graph are calculated as follows:

$$p(H_{w_i} = w_j|w_i) = \frac{\exp(m(i,j))}{\sum_{k\neq i}\exp(m(i,k))}, \tag{2.61}$$

where $m(i,j) = \boldsymbol{h}_k^{(2)T}\boldsymbol{W}_{dp}\boldsymbol{h}_i^{(2)}$ is a scoring function with weight matrix $\boldsymbol{W}_{dp}$. To predict the probability distribution $p(l_{w_i}|w_i)$, they feed $[\boldsymbol{h}_i^{(2)}; z(H_{w_i})]$ into a softmax function. $z(H_{w_i})$ is the weighted average of parent's hidden states: $\sum_{j\neq i} p(H_{w_i} = w_j|w_i)\boldsymbol{h}_j^{(2)}$. The parameters of this parsing model are learned by propagating errors from the translation objective function.

Figure 2.14: An example of a source sentence, and its translation in the form of linearised lexicalised constituency tree (Aharoni and Goldberg, 2017).

After calculating the latent graph, this model uses a modified version of the attentional SEQ2SEQ model to translate. In the encoder, a sequential encoder generates embedding of words $\boldsymbol{e_i}$. Then, for each word, another embedding is calculated with respect to the latent graph:

$$\boldsymbol{dep}_i = \tanh(\boldsymbol{W}_{dep}[\boldsymbol{e}_i; \bar{\boldsymbol{h}}(H_{w_i}); p(l_{w_i}|w_i)]), \tag{2.62}$$

where $\boldsymbol{W}_{dep}$ is weight matrix, and $\bar{\boldsymbol{h}}(H_{w_i}) = \sum_{j \neq i} p(H_{w_i} = w_j|w_i)\boldsymbol{e}_j$ is a weighted sum of parent's hidden states. The decoder of this model is a standard LSTM-based decoder that attends on both types of embeddings ($\boldsymbol{e}_i$ and $\boldsymbol{dep}_i$).

Attentional sequence-to-tree Model

Aharoni and Goldberg (2017) proposed a method to incorporate syntactic constituency information of the target language in NMT. As shown in Figure 2.14, they replaced the target sentence with its linearised lexicalised constituency tree. Then, they use an attentional SEQ2SEQ model that learns to parse and translate.

Eriguchi et al. (2017) proposed another method for learning to parse and translate. Their method inspired by the idea of Multi-Task Learning (Caruana, 1997; Collobert et al., 2011), and implicitly incorporate linguistics priors. The idea is to modify the decoder part of standard attentional sequence-to-sequence model in order to generate parse tree and translation simultaneously. It is done by hybridising decoder using Recurrent Neural Network Grammar (RNNG) (Dyer et al., 2016). RNNG is a generative model that models both words and their tree-based compositions. By hybridising, the model can generate a sentence with its parse tree, which is conditioned on the source sentence.

2.3.2 Multi-Task Learning for Directly Injecting Auxiliary Knowledge

Multi-Task Learning (Caruana, 1997) is an elegant approach to inject inductive biases contained in the training signals of *related* tasks in order to improve generalisation. It performs the transfer of inductive biases by parallel training of tasks and sharing representations. The inductive biases cause the MTL model to prefer hypotheses that explain more than a single task (Ruder, 2017).

2.3.2.1 Multi-Task Learning

Sharing strategies aka architectural design. Multi-task learning is possible by sharing representations among tasks; and is best achieved by sharing parameters among them. Therefore, a subset of all of the parameters is tied across the tasks. This is the most commonly used type of sharing and goes back to (Caruana, 1997). There is another form of sharing inspired by the idea of generating similar (not necessarily identical) representations for different tasks. This type of sharing is based on keeping a separate set of parameters for each task, and *regularise* the distance between the parameters across them (Duong et al., 2015; Yang and Hospedales, 2016). This setting is referred to as soft parameter sharing (Ruder, 2017), and is not widely used as it is sensitive to the tuning of regularisation sensitivity parameters. From the amount of sharing aspect, MTL models can be divided into the following categories:

- **Full sharing**: In this case, a single model performs all of the tasks. For informing the model regarding the desired task, a special tag could be added to the corresponding input.
- **Partial sharing**: The idea behind this model is that not all of the representations need to be shared across tasks. Each task has its own set of parameters to learn task-specific representations, while some of the parameters are tied among tasks to make knowledge sharing possible.

Training Schedule. Training schedule is the beating heart of MTL, and has a critical role in the *performance* of the resulted model. Training schedule is responsible for balancing out the importance (participation rate) of different tasks throughout the training process, in order to make the best use of the knowledge provided them. There is more than one task involve in the training of an MTL model. We categorise the MTL approaches based on the flavour of MTL they consider:

- **General-MTL**: where the goal is to improve all of the tasks (Chen et al., 2018b; Guo et al., 2018);
- **Biased-MTL**: where the aim is to improve one of the tasks, referred to as the main task, the most (Kiperwasser and Ballesteros, 2018; Guo et al., 2019). As the main aim of this thesis is to improve the translation task, we use these flavour MTL.

Note that prior works solely use "MTL" to refer to either of these categories, however we distinguish between them to make comparison easier.[5]

2.3.2.2 Multi-Task Learning and Transfer Learning

In this Section, we want to show the relation of the MTL to other learning frameworks. Note that there is no universal definition for some of the learning frameworks. For example, researchers use different definitions of transfer learning; some try to put MTL in the umbrella of transfer learning (Pan and Yang, 2009) while others try to distinguish between them (Torrey and Shavlik, 2010). We stick with the definition of Pan and Yang (2009). Assume we are given a source set $\mathcal{D}^S := \{(\boldsymbol{x}_i^{(S)}, \boldsymbol{y}_i^{(S)})\}_{i=0}^{N_S}$ from the source joint $P_S(X, Y)$, and a target set $\mathcal{D}^T := \{(\boldsymbol{x}_i^{(T)}, \boldsymbol{y}_i^{(T)})\}_{i=0}^{N_T}$ comes from the target joint $P_T(X, Y)$. Transfer learning aims to improve the target learned hypothesis (model) h_T in $P_T(X)$ by using $\mathcal{D}^S$. Please note that with this definition, the General-MTL is not necessarily a transfer learning method because of following reasons: (1) in General-MTL we cannot define source and target tasks as all of the tasks are regarded as same; (2) the goal is to improve (in expectation) *all* of the tasks even with the cost of degradation in the performance of some of them. On the other side, Biased-MTL can be seen as *inductive transfer learning* where the source and target tasks are different and trained together. Now, we can consider auxiliary tasks as source tasks and the main task as the target.

Multi-task learning and positive/negative transfer. Sharing information among tasks is a double-edged sword. If transfer causes performance decrease, then *negative transfer* has occurred. If transfer helps to improve the performance, we then call it *positive transfer*. As all of the tasks are not entirely related, the main goal of MTL is to maximise the positive transfer while minimising the negative transfer.

Recently, Wang et al. (2019) proposed a formal definition of negative transfer in transfer learning. Following them, we formalise the negative transfer for Biased-MTL. Suppose we are given a set of a main task along with one auxiliary task (it can be easily extended to more auxiliary tasks). Each task has its own training set $\mathcal{D}^k := \{(\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)})\}_{i=0}^{N_k}$, where $k = 0$ denotes the main task. An MTL approach takes training data of tasks as input, and outputs a hypothesis $h = \text{MTL}\left(\{\mathcal{D}^k\}_{k=0}^{1}\right)$.

[5]Recently, Liu et al. (2019b) coined the term "auxiliary learning" to refer to Biased-MTL, however we stick with the term Biased-MTL as it is self-descriptive and more consistent with the part of literature that solely used "MTL" term.

Assuming ℓ is the loss for the main task, and $P_0(X, Y)$ is the joint distribution of the main task, then the standard expected risk is defined as follows:

$$R_{P_0}(h) := \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim P_0}[\ell(h(\boldsymbol{x}), \boldsymbol{y})].$$

Note that the expected risk should be measured on a separate test set. Then we define negative transfer condition for any Biased-MTL approach as:

$$R_{P_0}\left(\mathrm{MTL}\left(\{\mathcal{D}^0, \mathcal{D}^1\}\right)\right) > R_{P_0}\left(\mathrm{MTL}\left(\{\mathcal{D}^0\}\right)\right).$$

where $\mathrm{MTL}\left(\{\mathcal{D}^0\}\right)$ refers to the hypothesis generated for the single main task. This definition can also be extended to examine whether a training schedule and/or architecture A is better than B for capturing positive transfer and avoiding negative transfer as follows:

$$R_{P_0}\left(\mathrm{MTL}_B\left(\{\mathcal{D}^0, \mathcal{D}^1\}\right)\right) > R_{P_0}\left(\mathrm{MTL}_A\left(\{\mathcal{D}^0, \mathcal{D}^1\}\right)\right),$$

The current research on MTL is focused on encouraging positive transfer and preventing the negative transfer phenomena in two lines of research: architectural design and training schedule.

2.3.2.3 Multi-Task Learning in Practice

MTL has been applied in different areas of machine learning. For non-deep learning methods, MTL has been applied in different settings including support vector machines (Evgeniou and Pontil, 2004), multiple linear regression (Jalali et al., 2010), image classification (Yuan et al., 2012), relation extraction (Jiang, 2009), linguistic annotation (Reichart et al., 2008). Recently, with the raising wave of deep learning, deep MTL becomes popular for different applications including visual representation learning (Doersch and Zisserman, 2017), face detection (Ranjan et al., 2017), scene understanding (Kendall et al., 2018), medical image segmentation (Moeskops et al., 2016), speaker-role adaptation in conversational AI (Luan et al., 2017).

MTL and NLP. Deep MTL has also been used for various NLP problems, including graph-based parsing (Chen and Ye, 2011) and key-phrase boundary classification (Augenstein and Søgaard, 2017) . (Chen et al., 2017b) has applied to Multi-Task Learning for Chinese word segmentation, and (Liu et al., 2017) applied it for text classification problem. Both of these works have used adversarial training to make sure the shared layer extract only common knowledge.

MTL has been used effectively to learn from multimodal data. Luong et al. (2016) has proposed MTL architectures for neural SEQ2SEQ transduction for tasks including MT, image caption generation, and parsing. They fully share the encoders (many-to-one), the decoders (one-to-many), or some of the encoders and decoders (many-to-many). Pasunuru and Bansal (2017) has made use of an MTL approach to improve video captioning with auxiliary tasks, including video prediction and logical language entailment based on a many-to-many architecture.

MTL for NMT Multi-task learning has attracted attention to improve NMT in recent works. (Zhang and Zong, 2016) has made use of monolingual sentences in the source language in a Multi-Task Learning framework by sharing encoder in the attentional encoder-decoder model. Their auxiliary task is to reorder the source text to make it close to the target language word order. (Domhan and Hieber, 2017) proposed a two-layer stacked decoder, which the bottom layer is trained on language modelling on the target language text. The next word is jointly predicted by the bottom layer language model and the top layer attentional RNN decoder. They reported only moderate improvements over the baseline and fall short against using synthetic parallel data.

2.3.2.4 Multi-task learning for injecting linguistic knowledge in NMT

The current research on MTL is focused on encouraging positive transfer and preventing the negative transfer phenomena in two lines of research: (1) Architecture design: works in this area, including part II of this thesis, try to learn effective parameter sharing among tasks; (2) Training schedule: works in this area, including part III of this thesis, focus on setting the importance of tasks throughout the training.

Linguistic auxiliary tasks and architectural design Søgaard and Goldberg (2016) and Hashimoto et al. (2016) have proposed architectures by stacking up tasks on top of each other according to their linguistic level, e.g. from lower-level tasks (POS tagging) to higher-level tasks (syntactic parsing). In this approach, each task uses predicted annotations and hidden states of the lower-level tasks for making a better prediction. Dalvi et al. (2017) investigated the amount of learned morphology and how it can be injected using MTL. Kiperwasser and Ballesteros (2018) has investigated part-of-speech tagging and dependency parsing tasks where models with fully shared parameters are trained jointly on multiple tasks.

Niehues and Cho (2017) has made use of part-of-speech tagging and named-entity recognition tasks to improve NMT. They have used the attentional encoder-decoder with a shallow architecture, and share different parts, e.g. the encoder, decoder, and attention. They report the best performance with fully sharing the encoder.

In Chapter 4, for the first time to the best of our knowledge, we have used semantic parsing as an auxiliary task along with others, i.e. syntactic parsing, and named-entity recognition. We propose an architecture that uses partial sharing on deep stacked encoder and decoder components, and the results show that it is critical for NMT improvement in MTL. Furthermore, we propose adversarial training to prevent contamination of shared knowledge with task-specific details. Then, in Chapter 4, we propose a model that learns how to control the amount of sharing among tasks dynamically.

Training schedule. Since there is more than one task involved in MTL, training schedules are designed specifically for each flavour of MTL: General-MTL and Biased-MTL. Training schedules designed for the global-MTL are focused on co-evolving easy and difficult tasks uniformly. These methods are designed to achieve competitive performance with existing single-task models of each task (Chen et al., 2018b; Guo et al., 2018). On the other hand, training schedules for Biased-MTL focus on achieving greater improvements on the main task, and our method belongs to this category. Since this thesis aims to improve the translation task the most, we will focus on this flavour.

Training schedules can be fixed/dynamic throughout the training and be hand-engineered/adaptive. In Chapter 4, we propose a fixed hand-engineered schedule for improving low-resource NMT with auxiliary linguistic tasks. Recently, Guo et al. (2019) has proposed an adaptive way to calculate the importance weights of tasks. Instead of manual tuning of importance weights via an extensive grid search, they model the performance of each set of weights as a sample from a Gaussian Process (GP) and search for optimal values. Their method is not entirely adaptive as strong prior needs to be set for the main task. This method can be seen as a guided yet computationally exhaustive trial-and-error, where in each trial, MTL models need to be re-trained (from scratch) with the sampled weights. Moreover, the weight of tasks is fixed throughout the training. At least, for the case of low-resource NMT, Kiperwasser and Ballesteros (2018) shows that *dynamically* changing the weights throughout the training is essential to make better use of auxiliary tasks. They have

proposed *hand-engineered* training schedules for MTL in NMT, where they dynamically change the importance of the main task vs. the auxiliary tasks throughout the training process. Their method relies on *hand-engineered* schedules which should be tuned by trial-and-error, In Part III, we will introduce a *learning to multi-task learn* method to *adaptively* and *dynamically* set the importance of the tasks and learn the MTL model in the course of a single training run.

Zhang et al. (2018a) has proposed a learning-to-MTL framework in order to learn effective MTL *architectures* for generalising to new tasks. This is achieved by collecting historical multi-task experience, represented by tuples consisting of the MTL problem, MTL architecture, and its relative error. In contrast, our learning-to-MTL framework tackles the problem of learning effective training schedules to use auxiliary tasks in such a way to improve the main translation task the most.

2.3.3 Other Approaches

2.3.3.1 Active Learning

The key idea behind active learning (AL) is that we can achieve better accuracy with fewer labels if the model is able to choose the training data it learns from (Settles, 2009). AL approaches are based on the assumption that there is a *pool* or *stream* of unlabelled data and a limited budget for the annotation. Traditional AL methods are mostly based on heuristics to guide the selection of unlabelled data for the annotation. Tong and Chang (2001) proposed policies to select data considering the confidence of the classifier while Gilad-Bachrach et al. (2006) introduced an approach based on a query on a committee of classifiers. Yang et al. (2015) proposed an approach based on the heuristic of making the selected data as diverse as possible. There are much more heuristics in the literature which also could be mixed and matched. Although these heuristics have shown to be beneficial, the effectiveness of these heuristics is limited, and their performance varies with between datasets. Recently, deep Reinforcement Learning has been used for *learning* the active learning algorithm. Woodward and Finn (2017) combined the RL with one-shot learning for learning how and when to request labels in stream-based AL. Bachman et al. (2017) used policy gradients to select the unlabelled data, and also the order of selection for the pool-based AL. The learning to active learning methods also have been successfully applied on some NLP classification tasks (Fang et al., 2017; Liu et al., 2018; Vu et al., 2019).

For low-resource machine translation, we assume small bilingual corpora and a pool of monolingual sentences along with an annotation budget. The active learner

aims to select the unlabelled data in such a way to make the best use of the annotation budget in terms of the performance of the resulted translation model. There are several heuristics for statistical MT for selecting the untranslated sentences by considering the entropy of the potential translation, similarity-based selection, feature decay to increase the diversity (Haffari et al., 2009; Haffari and Sarkar, 2009; Biçici and Yuret, 2011). Recently, Liu et al. (2018) proposed an approach for *learning* active learning policies for low-resource NMT by formulating the training as a hierarchical MDP.

2.3.3.2 Back-translation and Dual Learning

A practical approach to leverage the monolingual sentences in source and target languages for data augmentation using back-translation. Sennrich et al. (2016a) proposed to use a pre-trained translation model to translate target monolingual sentences, and then use them as additional parallel data to re-train the source-to-target NMT model. Interestingly, it has been shown that even copying the monolingual target sentence as its source (i.e., identical source and target) is also beneficial for low-resource NMT (Currey et al., 2017). Hoang et al. (2018) suggest an iterative back-translation approach for generating increasingly better syntactic bilingual pairs. Zhang et al. (2018b) extended this approach to use both source and target monolingual data by jointly optimising a source-to-target and target-to-source NMT models. Further, a bi-directional NMT model can be served as both source-to-target and target-to-source translation models (Niu et al., 2018). More importantly, it has been shown that different methods for generating back-translation have different effects on high versus low-resource settings (Edunov et al., 2018).

Inspired by the success of back-translation, He et al. (2016) suggested a dual learning mechanism for improving NMT. It can be seen as an extension to the iterative back-translation by using deep RL approach., The idea is that any source-to-target translation task has a dual task in the form of target-to-source where the primal and dual tasks can form a closed loop. The primal and dual tasks can provide informative feedback regarding the generated translations of each other, which ensures that a translated sentence can be back-translated to the original one. Similarly, Cheng et al. (2016); Tu et al. (2017) have also used the reconstruction score to leverage monolingual corpora. In the dual learning, the models for primal and dual tasks are represented by agents and trained using deep RL as back-propagating through the sequence of discrete predictions is not differentiable. Lample et al. (2018a) instead propose to use symmetric architectures for both models and freeze primal/dual model

when training the other dual/primal model, and vice versa. As a result, they were able to apply their fully-differentiable model on unsupervised NMT effectively. Artetxe et al. (2018) also proposed an approach based on dual learning to train an NMT model in a completely unsupervised manner. They use a shared encoder for both primal and dual models and use pre-trained unsupervised cross-lingual embeddings (Artetxe et al., 2017). Therefore, the distributed representation of words would be language-independent, and the model only needs to learn how to compose representations for larger phrases. The last two approaches can be combined for a further improvement (Lample et al., 2018b).

2.3.3.3 Adversarial training

Inspired by advances in Generative Adversarial Networks (Goodfellow et al., 2014), adversarial training approaches have been proposed for co-training a discriminator network along with the NMT model (Yang et al., 2018; Wu et al., 2017). The NMT model (generator) tries to generate translations indistinguishable from human-translated ones while a discriminator tries to distinguish between machine-generated and human-translated ones. The two networks are co-trained by playing a min-max game and achieve a win-win situation when they reach the Nash Equilibrium. Policy gradient methods can be leveraged to co-train the NMT model and adversary. Although these approaches have achieved some performance gains, one of the primary bottlenecks in this direction is that the text generation remains a challenging task even for modern GAN architectures (Nie et al., 2019).

2.3.3.4 Zero/Few Shot Learning

Johnson et al. (2017); Ha et al. (2016) are the first who have shown that multilingual NMT models are somewhat capable of translating between untrained language pairs; the setting referred to as zero-shot learning. Gu et al. (2018a) proposed a new multilingual NMT model specifically to improve the translation of low-resource languages. It shares the universal lexical and sentence level representations across multiple source languages into one target language. This model enables the sharing of resources between high-resource languages and low-resource ones. Further, Gu et al. (2018b) proposed a meta-learning approach for low-resource NMT by using the universal lexical representations. Inspired by the idea of model-agnostic meta-learning algorithm (Finn et al., 2017), they view the training on NMT model on different low-resource language pairs as separate tasks. Then, they train the NMT model in such a way to rapidly adapt to new language pairs with a minimal amount

of bilingual language pairs. In a different direction, Neubig and Hu (2018) proposed to use a pre-trained multilingual NMT model and fine-tune it on new low-resource language pairs for rapid adaptation of NMT on new languages.

Recently, Arivazhagan et al. (2019) have investigated why multilingual NMT models do not perform well on unseen language pairs. They have found that the issue raises when the model is not able to learn language invariant features and propose auxiliary losses to impose language-invariant representations across languages. Gu et al. (2019) investigated the issue rises as the results of capturing spurious correlation in the training data. Then, they proposed to use language model pre-training and back-translation to help the model to disregard such correlation.

2.4 Summary

In this chapter, we covered the foundations and prior related works to this thesis. We overviewed deep learning fundamentals and its usage in NLP and machine translation. Then, we reviewed the case of bilingually low-resource NMT and described how linguistic resources could be used to compensate for the lack of bilingual data. This thesis aims to explore further and extend this line of research. In part I, we incorporate uncertainty in machine-generated linguistic annotations in Neural Machine Translation by proposing a forest-to-sequence model. In parts II and III, we focus on directly injecting the linguistic knowledge using Multi-Task Learning where Part II is mainly focused on the architectural design aspect and part III is dedicated to the training schedule.

Part I

Transduction of Complex Structures

Chapter 3

An Attentional Forest-To-Sequence Model

This chapter is based on:
P. Zaremoodi, G. Haffari, "Incorporating Syntactic Uncertainty in Neural Machine Translation with a Forest-to-Sequence Model", Proceedings of the 27th International Conference on Computational Linguistics (COLING), 2018.

The main aim of this thesis is to improve the performance of Neural Machine Translation in bilingually low-resource scenarios by incorporating linguistic knowledge. As mentioned in Section 2.3, one approach is to train or use pre-trained parsers to provide annotations of the source sentences. Incorporating syntactic information in Neural Machine Translation (NMT) can lead to better reorderings (Eriguchi et al., 2016), particularly useful when the language pairs are syntactically highly divergent or when the training bitext is not large. Previous work on using syntactic information, provided by top-1 parse trees generated by (inevitably error-prone) parsers, has been promising (Eriguchi et al., 2016; Chen et al., 2017a). In this chapter, we propose a *forest*-to-sequence NMT model to make use of combinatorially many parse trees of the source sentence to compensate for the parser errors. Our method represents the collection of parse trees as a packed forest and learns a neural transducer to translate from the input forest to the target sentence. Experiments on English to German, Chinese and Farsi translation tasks show the superiority of our approach over the sequence-to-sequence and tree-to-sequence neural translation models.

3.1 Introduction

One of the main premises about natural language is that words of a sentence are inter-related according to a (latent) hierarchical structure (Chomsky, 1957), i.e. a syntactic tree. Therefore, it is expected that modelling the hierarchical syntactic structure should improve the performance of NMT, especially in low-resource or linguistically divergent scenarios, such as English-Farsi. In this direction, Li et al. (2017) uses a sequence-to-sequence model, making use of *linearised* parse trees. Chen et al. (2018a) has proposed a model which uses syntax to constrain the dynamic encoding of the source sentence via structurally constrained attention. Bastings et al. (2017); Shuangzhi Wu (2017); Beck et al. (2018); Ding and Tao (2019) have incorporated syntactic information provided by the dependency tree of the source sentence. Marcheggiani et al. (2018) has proposed a model to inject semantic bias into the encoder of NMT model.

More related to our work, Eriguchi et al. (2016); Chen et al. (2017a) have proposed methods to incorporate the hierarchical syntactic constituency information of the source sentence. In addition to the embedding of words, calculated using the vanilla sequential encoder, they calculate the embeddings of phrases recursively, directed by the top-1 parse tree of the source sentence generated by a parser. Though the results are promising, the top-1 trees are prone to parser error, and furthermore cannot capture semantic ambiguities of the source sentence.

In this chapter, we address the issues mentioned above by using exponentially many trees encoded in a forest instead of a single top-1 parse tree. We capture the parser uncertainty by considering many parse trees and their probabilities. The encoding of each source sentence is guided by the forest and includes the forest nodes whose representations are calculated in a bottom-up fashion using our ForestLSTM architecture (Section 3.2). Thus, in the encoding stage of this approach, different ways of constructing a phrase are taken into consideration, along with the probability of rules in the corresponding trees. We evaluate our approach on English to Chinese, Farsi and German translation tasks, showing that forests lead to better performance compared to top-1 trees and sequential encoders (Section 3.4).

Following our work, Ma et al. (2018) has also proposed a forest-based NMT model. Instead of modelling the hierarchical structure of the forest, their model is based on linearising the forest and using a SEQ2SEQ model.

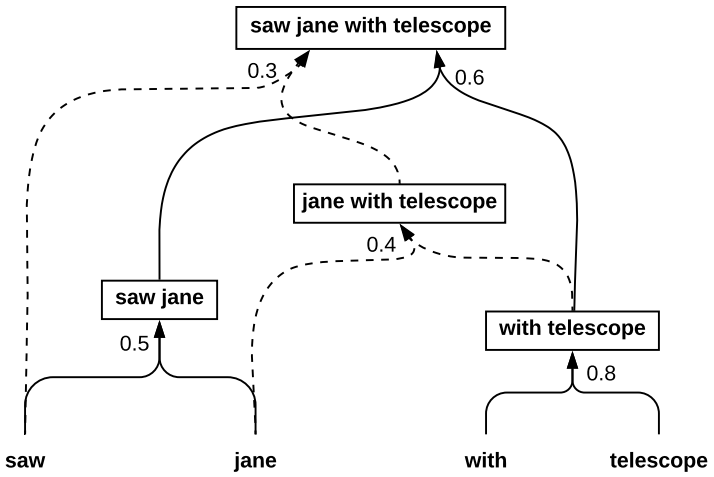


Figure 3.1: An example of generating a phrase from two different parse trees

3.2 Neural Forest-to-Sequence Translation

The TREE2SEQ model proposed by Eriguchi et al. (2016) (discussed in section 2.3.1) uses the top-1 parse tree generated by a parser. Mistakes and uncertainty in parsing could affect the performance of the translation. To address these issues, we propose a method to consider combinatorially many parse trees along with their corresponding probabilities. It consists of a *forest* encoder to encode a collection of packed parse trees, in order to reduce error propagation due to using only the top-1 parse tree. Our forest encoder calculates representations for words and phrases of the source sentence with respect to its parse forest. A sequential decoder, then, generates output words one-by-one from left-to-right by attending to both words and phrases (i.e., forest nodes).

3.2.1 Forest Encoder

The forest encoder consists of sequential and recursive parts for obtaining representation for both words and phrases. The sequential part is the vanilla sequence encoder (discussed in Section 2.2.1) that calculates the context-dependent representation of words. Then, the embeddings of phrases are calculated with respect to the source sentence parse forest in a recursive bottom-up fashion.

Embedding of Words. The source sentence is encoded by a sequential LSTM, as in the vanilla SEQ2SEQ model (Luong et al., 2015a):

$$\boldsymbol{h}_i = \text{SeqLSTM}(\boldsymbol{h}_{i-1}, \boldsymbol{E}_x[x_i])$$

where $\boldsymbol{E}_x[x_i]$ is the embedding of the word x_i in the embedding table $\boldsymbol{E}_x$ of the source language, and $\boldsymbol{h}_i$ is the context-dependent embedding of x_i.

Embedding of Phrases. We calculate the embedding of the forest nodes (phrases) in a bottom-up fashion. For each hyper-edge, we calculate the embedding of the head with respect to its tails using a TreeLSTM unit (Tai et al., 2015). In a forest, however, a phrase can be constructed in multiple ways using the incoming hyper-edges to a forest node, with different probabilities (see Figure 3.1). Our ForestLSTM combines the phrase embeddings resulted from these hyper-edges, and takes into account their probabilities in order to obtain a unified embedding for the forest node and its corresponding phrase:

$$
\begin{aligned}
\boldsymbol{\gamma}^l &= \tanh\left(\boldsymbol{U}^{\gamma}\sum_{l'=1}^{N}\mathbf{1}_{l\neq l'}\boldsymbol{h}^{l'} + \boldsymbol{W}^{\gamma}\boldsymbol{h}^l + \boldsymbol{v}^{\gamma}p^l + \boldsymbol{b}^{\gamma}\right)\\
\boldsymbol{f}^l &= \sigma\left(\boldsymbol{U}^{f}\sum_{l'=1}^{N}\mathbf{1}_{l\neq l'}[\boldsymbol{h}^{l'};\boldsymbol{\gamma}^{l'}] + \boldsymbol{W}^{f}[\boldsymbol{h}^l;\boldsymbol{\gamma}^l] + \boldsymbol{b}^l\right)\\
\boldsymbol{i} &= \sigma\left(\boldsymbol{U}^{i}\sum_{l=1}^{N}[\boldsymbol{h}^l;\boldsymbol{\gamma}^l] + \boldsymbol{b}^i\right)\\
\boldsymbol{o} &= \sigma\left(\boldsymbol{U}^{o}\sum_{l=1}^{N}[\boldsymbol{h}^l;\boldsymbol{\gamma}^l] + \boldsymbol{b}^o\right)
\end{aligned}
$$

where N is the number of incoming hyper-edges, $\boldsymbol{h}^l$ is the embedding for the head of the l-th incoming hyper-edge and p^l is its probability and $\boldsymbol{v}^{\gamma}$ is the learned weight for the probablity. $\boldsymbol{\gamma}^l$ is a probability-sensitive intermediate representation for the l-th incoming hyper-edge, which is then used in the computations of the forget gate $\boldsymbol{f}^l$, the input gate $\boldsymbol{i}$, and the output gate $\boldsymbol{o}$. The representation of the phrase $\boldsymbol{h}^{phr}$ is then calculated as

$$
\begin{aligned}
\tilde{\boldsymbol{c}} &= \tanh\left(\boldsymbol{U}^{\tilde{c}}\sum_{l=1}^{N}[\boldsymbol{h}^l;\boldsymbol{\gamma}^l] + \boldsymbol{b}^{\tilde{c}}\right)\\
\boldsymbol{c}^{phr} &= \boldsymbol{i}\odot\tilde{\boldsymbol{c}} + \sum_{l=1}^{N}\boldsymbol{f}^l\odot\boldsymbol{c}^l\\
\boldsymbol{h}^{phr} &= \boldsymbol{o}\odot\tanh(\boldsymbol{c}^{phr})
\end{aligned}
$$

where $\boldsymbol{c}^l$ is the memory cell of the TreeLSTM unit used to calculate the representation of the head for the l-th hyper-edge from its tail nodes.

3.2.2 Sequential Decoder

We use a sequential attentional decoder similar to that of the TREE2SEQ model, where the attention mechanism attends to both words and phrases in the forest:

$$\boldsymbol{c}_j = \sum_{i=1}^{n} \alpha_{ji} \boldsymbol{h}_i + \sum_{i'=1+n}^{n_p+n} \alpha_{ji'} \boldsymbol{h}_{i'}^{phr}$$

where n is the length of the input sentence, and n_p is the number of forest nodes.

We initialise the decoder's first state by using a TreeLSTM unit (Section 2.1.4.1) to combine the embeddings of the last word in the source sentence and the root of the forest:

$$\boldsymbol{g}_0 = \text{TreeLSTM}(\boldsymbol{h}_n, \boldsymbol{h}_{root}^{phr}).$$

This provides a summary of phrases and words in the source sentence to the decoder.

3.2.3 Training

Suppose we are given a training set $\mathcal{D} := \{(\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{F}_{\boldsymbol{x}_i})\}_{i=0}^{N}$, the model parameters are trained end-to-end by maximising the (regularised) log-likelihood of the training data:

$$\arg\max_{\boldsymbol{\Theta}} \sum_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{F}_{\boldsymbol{x}})\in\mathcal{D}} \log P_{\boldsymbol{\Theta}}(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{F}_{\boldsymbol{x}_i}),$$

where $\mathcal{D}$ is the set of triples consists of the bilingual training sentences $(\boldsymbol{x}, \boldsymbol{y})$ paired with the parse forests of the source sentences $\boldsymbol{F}_{\boldsymbol{x}}$.

3.3 Computational Complexity Analysis

We now analyse the computational complexity of inference for SEQ2SEQ, TREE2SEQ and FOREST2SEQ models. We show that, interestingly, our method process combinatorially many trees with only a small linear overhead.

Let $|x|$ denotes the length of the source sentence and $O(W_s)$ and $O(W_r)$ are computational complexity of forward-pass for the sequential and Tree/Forest LSTM units, respectively. Having N nodes in the tree/forest, the computational complexity of the encoding phase would be:

$$O(2W_s|x| + W_r N)$$

Sentence Length	Avg. tree nodes	Avg. forest nodes	Avg. # of trees in forests
<10	7.94	9.77	6.13E+4
10-19	12.3	18.99	2.62E+16
20-29	21.18	41.79	2.76E+22
>30	31	78.72	2.21E+15
all	10.33	14.84	1.41E+20

Table 3.1: The average number of nodes in trees and forests along with average number of trees in forests for En→Fa bucketed dataset.

where the first term shows the computational complexity of a bidirectional sequential encoder to calculate the embeddings of words, and the latter one is the time for computing the embeddings of phrases with respect to the corresponding tree/forest.

For generating each word in the target sentence, the attention mechanism performs soft attention on words and phrases of the source sentence. If $O(W_t)$ be the time for updating the decoder state and generating the next target word, for a target sentence with length $|y|$ the decoding phase computational complexity would be:

$$O(W_t|y| + |y|(N + |x|))$$

Hence, the total inference time for a sentence pair is:

$$O(2W_s|x| + W_rN + W_t|y| + |y|(N + |x|))$$

The difference among these three methods is N. For the SEQ2SEQ model N is 0. For the TREE2SEQ model, the number of nodes in the tree is a constant function of the input size: $N = |x| - 1$. Since we used *pruned* forests obtained from the parser in (Huang, 2008), the number of nodes in the forest is variable. Table 3.1 shows the average value of N for trees/forests for different source lengths for one of the datasets we used in experiments. As seen, while forests contain combinatorially many trees, on average, the number of nodes in parse forests is less than twice the number of nodes in the corresponding top-1 parse trees. It shows that our method considers combinatorially many trees instead of the top-1 tree using only a small linear overhead.

3.4 Experiments

3.4.1 The Setup

Datasets. We make use of three different language pairs, translating from English (En) to Farsi (Fa), Chinese (Ch), and German (De). Our research focus is to tackle

	Train	Dev	Test
En → Fa	337K	RND. 2k	RND. 2k
En → Ch	44k	devset1_2	devset_3
En → De	100K	newstest2013	newstest2014

Table 3.2: The statistics of bilingual corpora.

NMT issues for bilingually low-resource scenarios, and En→Fa is intrinsically a low-resource language. Moreover, we used small datasets for En→Ch and En→De language pairs to simulate low-resource scenarios, where the source and target languages are linguistically divergent and close, respectively. For En→Fa, we use the TEP corpus (Tiedemann, 2009) which is extracted from movie subtitles. For En→Ch, we use BTEC and for En→De, we use the first 100K sentences of Europarl[1]. The statistics of the datasets has been summarised in Table 3.2.

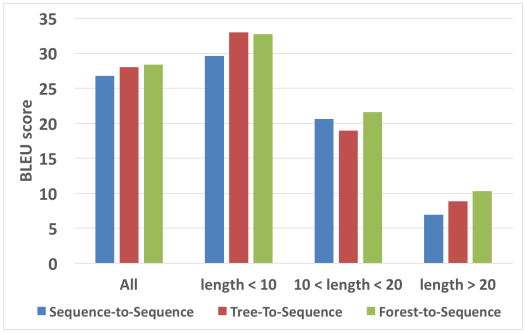
We lowercase and tokenise the corpora using Moses scripts (Koehn et al., 2007). Sentences longer than 50 words are removed, and words with the frequency less than 5 are replaced with <Unk>. Compact forests and trees for English source sentences are obtained from the parser in (Huang, 2008), where the forests are binarised, i.e. hyper-edges with more than two tail nodes are converted to multiple hyper-edges with two tail nodes. This is to ensure a fair comparison between our model and the Tree2Seq model (Eriguchi et al., 2016) where they use binary HPSG parse trees. Furthermore, we prune the forests by removing low probability hyper-edges, which significantly reduces the size of the forests. In all experiments, we use the development sets for setting the hyper-parameters, and the test sets for evaluation.

Implementation Details. We use Mantis implementation of attentional NMT (Cohn et al., 2016) to develop our code for Forest2Seq and Tree2Seq with DyNet (Neubig et al., 2017). All neural models are trained end-to-end using Stochastic Gradient Descent, where the mini-batch size is set to 128. The maximum training epochs is set to 20, and we use early stopping on the development set as a stopping condition. We generate translations using greedy decoding. The BLEU score is computed using the "multi-bleu.perl" script in Moses.

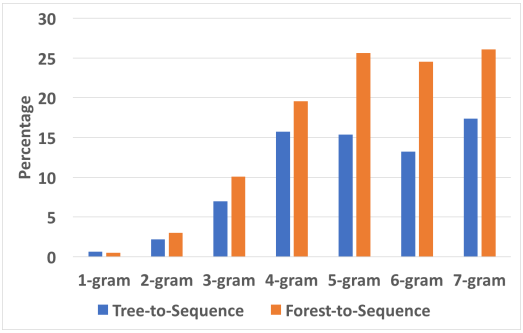
3.4.2 Results

The perplexity and BLEU scores of different models for all translation tasks are presented in Table 3.4. In all translation tasks, Forest2Seq outperforms Tree2Seq

[1]http://www.statmt.org/wmt14/translation-task.html



(a)



(b)

Figure 3.2: (a) BLEU scores for bucketed En→Ch dataset. (b) Percentage of more correct n-grams generated by the Tree2Seq and Forest2Seq models compared to Seq2Seq model for En→Ch dataset.

	Perplexity	BLEU
Forest encoder W/ sequential part	16.66	12.38
Forest encoder W/O sequential part	17.48	11.97

Table 3.3: The effect of the sequential part in the forest encoder (En → Fa).

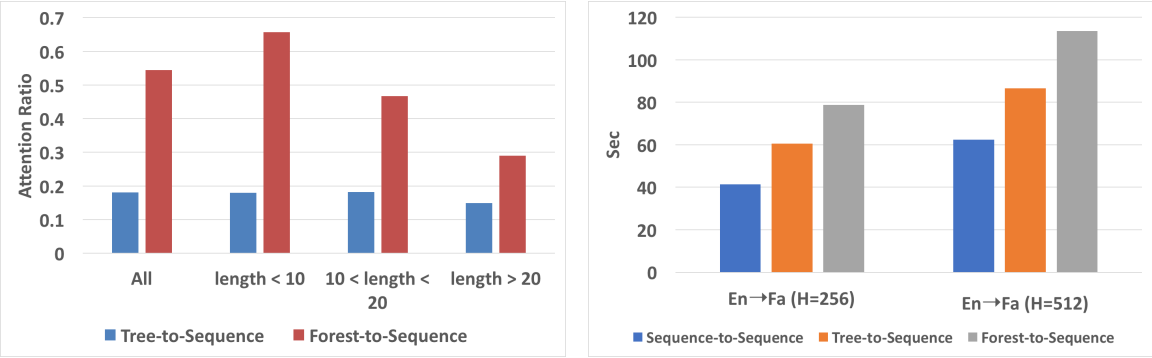
as it reduces syntactic errors by using forests instead of top-1 parse trees. Our results confirm those in (Eriguchi et al., 2016) and show that using syntactic trees in Tree2Seq improve the translation quality compared to the vanilla Seq2Seq. Comparing BLEU scores of the forest-based and tree-based models, the largest increase is observed for English to Farsi pair. This can be attributed to the syntactic divergence between English and Farsi (SVO vs. SOV) as well as the reduction of significant errors in the top-1 parser trees for this translation task, resulted from the domain mismatch between the parser's training data (i.e. Penn Tree Bank) and the English source (i.e. informal movie subtitles).

3.4.3 Analysis

The effect of the sequential part in the forest encoder The forest encoder consists of sequential and recursive parts, where the former is the vanilla sequence encoder. The attention mechanism attends to the embeddings of both sequential and recursive parts. We investigate the effect of the sequential part in the proposed forest encoder. Table 3.3 shows the results on the test set of En → Fa dataset. The results show that the sequential part in the forest encoder leads to improvement in results. Speculatively, the sequential part helps the forest encoder by providing the context-aware embeddings for words which then be used to construct phrase embeddings.

Method	H	English → German		English → Chinese		English → Farsi	
		Perplexity	BLEU	Perplexity	BLEU	Perplexity	BLEU
SEQ2SEQ	256	33.07	11.98	6.48	25.43	19.21	10.17
(Luong et al., 2015a)	512	32.61	12.21	6.12	26.77	18.4	10.93
TREE2SEQ	256	30.13	13	6.17	26.85	17.94	11.32
(Eriguchi et al., 2016)	512	31.86	13.05	5.71	28	16.28	11.71
our	256	30.83	**13.54**	6.16	27.08	17.62	11.91
FOREST2SEQ	512	**29.25**	13.43	**5.49**	**28.39**	**16.66**	**12.38**

Table 3.4: Comparison of the methods together with different hidden dimension size (H) for all datasets.



(a) (b)

Figure 3.3: (a) Attention ratios for En→Fa bucketed dataset.(b) Inference time (seconds) required for the test set of En→Fa dataset using trained models.

For which sentence lengths the forest-based model is more helpful? To investigate the effect of source sentence length, we divide En→Ch dataset into three buckets with respect to the length of source sentences. Figure 3.2a depicts the BLEU scores resulted from the models for different buckets. The FOREST2SEQ model performs better than vanilla SEQ2SEQ model on all buckets. Interestingly, while TREE2SEQ model has a lower BLEU compared to SEQ2SEQ model for the sentences whose lengths are between 10 and 20, the FOREST2SEQ model has a better BLEU possibly due to reducing parsing errors. We also have seen similar results for the other two language pairs.

The forest-based model results in correct larger n-grams in translations We further analyse the effect of the incorporated syntactic knowledge on improving the number of generated gold n-grams in translations. For each sentence, we compute the number of n-grams in the generated translations which are common with those in the gold translation. Then, after aggregating the results over the entire test set, we compute the percentage of additional gold n-grams generated by syntax-aware models, i.e. TREE2SEQ and FOREST2SEQ, compared to the SEQ2SEQ model. The results are depicted in Figure 3.2b. Generating correct high order n-grams is hard, and results show that incorporating syntax is beneficial. As n increases, the FOREST2SEQ model performs significantly better than the TREE2SEQ model in generating gold n-grams, possibly due to better reorderings between the source and target.

How much attention the forest-based and tree-based models pay to the syntactic information? We next analyse the extent by which the syntactic in-

formation is used by the Tree2Seq and Forest2Seq models. We compute the ratio of attention on phrases to words for both of the syntax-aware models in En→Fa translation task, where the source and target languages are highly syntactically divergent. For each triple in the test set, we calculate the sum of attention on words and phrases during decoding. Then, the ratio of attention on phrases to words is computed and is averaged for all triples. Figure 3.3a shows these attention ratios for bucketed En→Fa dataset. It shows that for all sentence lengths, the Forest2Seq model provides richer phrase embeddings compared to the Tree2Seq model, leading to more usage of the syntactic information.

Investigating the effect of using trees/forests on inference time We measured the inference time required for the test set of En→Fa dataset using the trained models. The results are depicted in Figure 3.3b. As seen, while using one parse tree increases the inference time linearly, interestingly, our Forest2Seq model considers combinatorially many trees also with a small linear overhead.

3.5 Summary

We have proposed a forest-to-sequence attentional NMT model, which uses a packed forest instead of the top-1 parse tree in the encoder. Using a forest of parse trees, our method efficiently considers combinatorially many constituency trees in order to take into account parser uncertainties and errors. Experimental results show our method is superior to the attentional tree-to-sequence model, which is more prone to the parsing errors.

Part II

Multi-Task Learning: Architectural Design

Chapter 4

Deep Seq2Seq MTL for NMT

This chapter is based on:
P. Zaremoodi, G. Haffari, "Neural Machine Translation for Low Resource Scenarios: A Deep Multi-Task Learning Approach", Proceedings of Annual Meeting for North American Chapter of Association of Computational Linguistics (NAACL), 2018.

The main aim of this thesis is to improve bilingually low-resource NMT by incorporating linguistic knowledge. In Part I, we incorporated linguistic knowledge provided in the form of annotations of source sentences. In this part and the next one, we will focus on directly injecting linguistic knowledge. More specifically, we scaffold the machine translation task on auxiliary tasks, including semantic parsing, syntactic parsing, and named-entity recognition. As discussed in Section 2.3.2, a practical approach for injecting knowledge from a task to others is Multi-Task Learning. It is based on tying parameters among different tasks to share statistical strength. In this chapter, we start by casting the auxiliary linguistic tasks as Seq2Seq transduction tasks to make all tasks structurally homogeneous. Then, we propose a multitask architecture that enables an effective sharing strategy between tasks by tying a *fraction* of their parameters with those of the main translation task. Further, we make use of adversarial training to protect shared representations from being contaminated by task-specific features. Our extensive experiments and analyses show the effectiveness of the proposed approach for improving bilingually low-resource NMT by incorporating linguistic knowledge in several language pairs.

4.1 Introduction

NMT with attentional encoder-decoder architectures (Luong et al., 2015c; Bahdanau et al., 2015) has revolutionised machine translation, and achieved state-of-the-art for several language pairs. However, NMT is notorious for its need for large amounts of bilingual data (Koehn and Knowles, 2017) to achieve reasonable translation quality. Leveraging existing monolingual resources is a potential approach for compensating this requirement in bilingually scarce scenarios. Ideally, semantic and syntactic knowledge learned from existing linguistic resources provides NMT with proper inductive biases, leading to increased generalisation and better translation quality.

Multi-task learning is an effective approach to inject knowledge into a task from other related tasks. Various recent works have attempted to improve NMT with an MTL approach (Peng et al., 2017; Liu et al., 2017; Zhang and Zong, 2016); however, they either do not make use of curated linguistic resources (Domhan and Hieber, 2017; Zhang and Zong, 2016), or their MTL architectures are restrictive, yielding mediocre improvements (Niehues and Cho, 2017). The current research leaves open how to best leverage curated linguistic resources in a suitable MTL framework to improve NMT.

In this chapter, we make use of curated monolingual linguistic resources in the source side to improve NMT in bilingually scarce scenarios. More specifically, we scaffold the machine translation task on auxiliary tasks, including semantic parsing, syntactic parsing, and named-entity recognition. This is achieved by casting the auxiliary tasks as SEQ2SEQ transduction tasks and tie the parameters of their encoders and/or decoders with those of the main translation task. Our MTL architectures make use of deep stacked encoders and decoders, where the parameters of the top layers are shared across the tasks. We further make use of adversarial training to prevent contamination of shared knowledge with task-specific information.

We present empirical results on translating from English into Vietnamese, Turkish, Farsi and Spanish; four target languages with varying degree of divergence compared to English. Empirical results demonstrate the effectiveness of the proposed approach in improving the translation quality for these four translation tasks in bilingually scarce scenarios.

In summary, the contributions of this chapter can be categorised as follows:

- We propose a partial sharing strategy for SEQ2SEQ MTL models and show it is crucial for improving low-resource NMT using curated linguistic resources.

- We further improve the partial sharing strategy by adding adversarial training to prevent contamination of the shared representation space.
- We conduct extensive experiments, comparison and analysis on four language pairs.
- We present an analysis of the effectiveness of different types of auxiliary tasks on translation quality. To the best of our knowledge, our work is the first to incorporate semantic parsing knowledge for improving NMT via MTL.

4.2 Seq2Seq Multi-Task Learning

We consider an extension of the basic attentional SEQ2SEQ model discussed in Section 2.2.2, where the encoder and decoder are equipped with deep stacked layers. Presumably, deeper layers capture more abstract information about a task; hence, they can be used as a mechanism to share useful generalisable information among multiple tasks.

Deep Stacked Encoder. The deep encoder consists of multiple layers, where the hidden states in layer $\ell - 1$ are the inputs to the hidden states at the next layer ℓ. That is,

$$\overrightarrow{\boldsymbol{h}}_i^{\ell} = \overrightarrow{\text{RNN}}_{\boldsymbol{\theta}_{\ell,enc}}^{\ell}(\overrightarrow{\boldsymbol{h}}_{i-1}^{\ell}, \boldsymbol{h}_i^{\ell-1})$$
$$\overleftarrow{\boldsymbol{h}}_i^{\ell} = \overleftarrow{\text{RNN}}_{\boldsymbol{\theta}_{\ell,enc}}^{\ell}(\overleftarrow{\boldsymbol{h}}_{i+1}^{\ell}, \boldsymbol{h}_i^{\ell-1})$$

where $\boldsymbol{h}_i^{\ell} = [\overrightarrow{\boldsymbol{h}}_i^{\ell}; \overleftarrow{\boldsymbol{h}}_i^{\ell}]$ is the hidden state of the ℓ'th layer RNN encoder for the i'th source sentence word. The inputs to the first layer forward/backward RNNs are the source word embeddings $\boldsymbol{E}_S[x_i]$. The representation of the source sentence is then the concatenation of the hidden states for all layers $\boldsymbol{h}_i = [\boldsymbol{h}_i^1; \ldots; \boldsymbol{h}_i^L]$ which is then used by the decoder.

Deep Stacked Decoder. Similar to the multi-layer RNN encoder, the decoder RNN has multiple layers:

$$\boldsymbol{s}_j^{\ell} = \text{RNN}_{\boldsymbol{\theta}_{\ell,dec}}^{\ell}(\boldsymbol{s}_{j-1}^{\ell}, \boldsymbol{s}_j^{\ell-1})$$

where the inputs to the first layer RNNs are

$$\boldsymbol{W}_{sj} \cdot \boldsymbol{E}_T[y_{j-1}] + \boldsymbol{W}_{sc} \cdot \boldsymbol{c}_j$$

in which $\boldsymbol{c}_j$ is the dynamic source context, as defined in eqn. 2.48. The state of the decoder is then the concatenation of the hidden states for all layers: $\boldsymbol{s}_j = [\boldsymbol{s}_j^1; \ldots; \boldsymbol{s}_j^L]$ which is then used in eqn. 2.41 as part of the "output generation module".

Shared Layer MTL. We share the deep layer RNNs in the encoders and/or decoders across the tasks, as a mechanism to share abstract knowledge and increase model generalisation.

Suppose we have a total of $M+1$ tasks, consisting of the main task plus M auxiliary tasks. Let $\boldsymbol{\Theta}^m_{enc} = \{\boldsymbol{\theta}^m_{\ell,enc}\}_{\ell=1}^{L}$ and $\boldsymbol{\Theta}^m_{dec} = \{\boldsymbol{\theta}^m_{\ell',dec}\}_{\ell'=1}^{L'}$ be the parameters of multi-layer encoder and decoder for the task m. Let $\{\boldsymbol{\Theta}^m_{enc}, \boldsymbol{\Theta}^m_{dec}\}_{m=1}^{M}$ and $\{\boldsymbol{\Theta}^0_{enc}, \boldsymbol{\Theta}^0_{dec}\}$ be the RNN parameters for the auxiliary tasks and the main task, respectively. We share the parameters of the deep-level encoders and decoders of the auxiliary tasks with those of the main task. That is,

$$\begin{aligned} \forall m \in [1,..,M]\ \forall \ell \in [1,..,L^m_{enc}] \quad &: \quad \boldsymbol{\theta}^m_{\ell,enc} = \boldsymbol{\theta}^0_{\ell,enc} \\ \forall m \in [1,..,M]\ \forall \ell' \in [1,..,L'^m_{dec}] \quad &: \quad \boldsymbol{\theta}^m_{\ell',dec} = \boldsymbol{\theta}^0_{\ell',dec} \end{aligned}$$

where L^m_{enc} and L'^m_{dec} specify the deep-layer RNNs need to be shared parameters. Other parameters to share across the tasks include those of the attention module, the source/target embedding tables, and the output generation module. As an extreme case, we can share *all* the parameters of SEQ2SEQ architectures across the tasks.

Training Objective. Suppose we are given a collection of K SEQ2SEQ transductions tasks, each of which is associated with a training set $\mathcal{D}^k := \{(\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)})\}_{i=0}^{N_k}$, where $k=0$ denotes the main translation task. The parameters are learned by maximising the MTL training objective:

$$\mathcal{L}_{mtl}(\boldsymbol{\Theta}_{mtl}) := \sum_{k=0}^{K} w^{(k)} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}^k} \log P_{\boldsymbol{\Theta}_{mtl}}(\boldsymbol{y}^{(k)}|\boldsymbol{x}^{(k)}). \tag{4.1}$$

where $\boldsymbol{\Theta}_{mtl}$ denotes all the parameters of the MTL architecture and $w^{(k)}$ denotes the importance weight of task k and we set it to

$$w^{(k)} = \frac{\gamma^{(k)}}{N_k}$$

where γ_m balances out its influence in the training objective. In Chapter 6, we will back to this training objective by proposing *adaptive* importance weights.

Training Schedule. Variants of stochastic gradient descent (SGD) can be used to optimise the objective in order to learn the parameters. Making the best use of tasks with different objective geometries is challenging, e.g. due to the scale of their gradients. One strategy for making an SGD update is to select the tasks from which the next data items should be chosen. In our training schedule, we randomly select a training data item from the main task and pair it with a data item selected from a randomly selected auxiliary task for making the next SGD update. This ensures the presence of a training signal from the main task in all SGD updates and avoids the training signal being washed out by the auxiliary tasks. In Chapter 7, we will go back to the scheduling of Biased-MTL and replace this predefined schedule by *learning* an adaptive schedule.

4.3 Adversarial Training

The learned shared knowledge could be contaminated by task-specific information. We address this issue by adding an adversarial objective. The basic idea is to augment the MTL training objective with additional terms so that the identity of a task cannot be predicted from its data items by the representations resulted from the shared encoder/decoder RNN layers.

Task Discriminator. The goal of the task discriminator is to predict the identity of a task for a data item based on the representations of the share layers. More specifically, our task discriminator consists of two RNNs with LSTM units, each of which encodes the sequence of hidden states in the shared layers of the encoder and the decoder.[1] The last hidden states of these two RNNs are then concatenated, giving rise to a fixed dimensional vector summarising the representations in the shared layers. The summary vector is passed through a fully connected layer, followed by a softmax to predict the probability distribution over the tasks:

$$P_{\boldsymbol{\Theta}_d}(\text{task id}|\boldsymbol{h}_d) \sim \text{softmax}(\boldsymbol{W}_d\boldsymbol{h}_d + \boldsymbol{b}_d)$$
$$\boldsymbol{h}_d := \text{disLSTMs}(\text{shrRep}_{\boldsymbol{\Theta}_{mtl}}(\boldsymbol{x}, \boldsymbol{y}))$$

where disLSTMs denotes the discriminator LSTMs, $\text{shrRep}_{\boldsymbol{\Theta}_{mtl}}(\boldsymbol{x}, \boldsymbol{y})$ denotes the representations in the shared layer of deep encoders and decoders in the MTL architecture, and $\boldsymbol{\Theta}_d$ includes the disLSTMs parameters as well as $\{\boldsymbol{W}_d, \boldsymbol{b}_d\}$.

[1] When multiple layers are shared, we concatenate their hidden states at each time step, which is then inputted to the task discriminator's LSTMs.

Adversarial Objective. Inspired by (Chen et al., 2017b), we add two additional terms to the MTL training objective in eqn. 4.1. The first term is $\mathcal{L}_{adv1}(\mathbf{\Theta}_d)$ defined as:

$$\sum_{k=0}^{K} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}^k} \log P_{\mathbf{\Theta}_d}(m|\,\text{disLSTMs}(\text{shrRep}_{\mathbf{\Theta}_{mtl}}(\boldsymbol{x},\boldsymbol{y}))).$$

Maximising the above objective over $\mathbf{\Theta}_d$ ensures proper training of the discriminator to predict the identity of the task. The second term ensures that the parameters of the shared layers are trained so that they confuse the discriminator by maximising the entropy of its predicted distribution over the task identities. That is, we add the term $\mathcal{L}_{adv2}(\mathbf{\Theta}_{mtl})$ to the training objective defined as:

$$\sum_{k=0}^{K} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}^k} H\big[P_{\mathbf{\Theta}_d}(.|\,\text{disLSTMs}(\text{shrRep}_{\mathbf{\Theta}_{mtl}}(\boldsymbol{x},\boldsymbol{y})))\big]$$

where $H[.]$ is the entropy of a distribution. In summary, the adversarial training leads to the following optimisation

$$\arg\max_{\mathbf{\Theta}_d,\mathbf{\Theta}_{mtl}} \mathcal{L}_{mtl}(\mathbf{\Theta}_{mtl}) + \mathcal{L}_{adv1}(\mathbf{\Theta}_d) + \lambda\mathcal{L}_{adv2}(\mathbf{\Theta}_{mtl}).$$

We maximise the above objective by SGD, and update the parameters by alternating between optimising $\mathcal{L}_{mtl}(\mathbf{\Theta}_{mtl}) + \lambda\mathcal{L}_{adv2}(\mathbf{\Theta}_{mtl})$ and $\mathcal{L}_{adv1}(\mathbf{\Theta}_d)$.

4.4 Experiments

4.4.1 Bilingual Corpora

We use four language-pairs, translating from English to Vietnamese, Turkish, Farsi and Spanish. We have chosen these languages to analyse the effect of Multi-Task Learning on languages with different underlying linguistic structures.[2] The sentences are segmented using BPE (Sennrich et al., 2016b) on the union of source and target vocabularies with a 40k vocabulary size for English to Vietnamese Spanish and Turkish. For English-Farsi, BPE is performed using separate vocabularies due to the disjoint alphabets with 30k vocabulary sizes.

Table 4.1 show some statistics about the bilingual corpora. Further details about the corpora and their pre-processing is as follows:

[2]English, Vietnamese and Spanish are SVO while Farsi and Turkish are SOV.

	Train	Dev	Test
En → Vi	133K	1,553	1,268
En → Tr	200K	2,910	2,898
En → Es	150K	2,928	2,898
En → Fa	98K	3,000	4,000

Table 4.1: The statistics of bilingual corpora.

- The English-Vietnamese is from the translation task in IWSLT 2015, and we use the preprocessed version provided by (Luong and Manning, 2015). The sentence pairs in which at least one of their sentences had more than 300 units (after applying BPE) are removed. "tst2012" and "tst2013" parts are used for validation and test sets, respectively.

- The English-Farsi corpus is assembled from all the parallel news text in LDC2016E93 *Farsi Representative Language Pack* from the Linguistic Data Consortium, combined with English-Farsi parallel subtitles from the TED corpus (Tiedemann, 2012). Since the TED subtitles are user-contributed, this text contained considerable variation in the encoding of its Perso-Arabic characters. To address this issue, we have normalised the corpus using the Hazm toolkit.[3] Sentence pairs in which one of the sentences has more than 80 (before applying BPE) are removed, and BPE is performed with a 30k vocabulary size. Random subsets of this corpus (3k and 4k sentences each) are held out as validation and test sets, respectively.

- The English-Turkish is from WMT parallel corpus (Bojar et al., 2016) with about 200K training pairs gathered from news articles. We use the Moses toolkit (Koehn et al., 2007) to filter out pairs where the number of tokens is more than 250 tokens (after applying BPE) and pairs with a source/target length ratio higher than 1.5. "newstest2016" and "newstest2018" parts are used as validation and test set.

- For English-Spanish pair, we have used the first 150K training pairs of Europarl corpus (Koehn, 2005). We also have applied the similar filtering process that we have done on English-Turkish. "newstest2011" and "newstest2013" parts are used as validation and test set, respectively.

[3] `www.sobhe.ir/hazm`

4.4.2 Auxiliary Tasks

We have chosen the following auxiliary tasks to provide the NMT model with syntactic and/or semantic knowledge, in order to enhance the quality of translation:

Semantic Parsing. Preserving the meaning is an important characteristic of the desired translation. Learning semantic parsing helps the model to abstract away the meaning from the surface in order to convey it in the target translation. For this task, we have used the Abstract Meaning Representation (AMR) corpus Release 2.0 (LDC2017T10).[4] This corpus contains natural language sentences from newswire, weblogs, web discussion forums and broadcast conversations. We cast this task to a SEQ2SEQ transduction task by linearising the AMR graphs (Konstas et al., 2017).

Syntactic Parsing. By learning the phrase structure of the input sentence, the model would be able to learn better re-ordering. Especially, in the case of language pairs with the high level of syntactic divergence (e.g. English-Farsi). We have used Penn Tree Bank parsing data with the standard split for training, development, and test (Marcus et al., 1993). We cast syntactic parsing to a SEQ2SEQ transduction task by linearising constituency trees (Vinyals et al., 2015).

Named-Entity Recognition (NER). It is expected that learning to recognise named-entities help the model to learn the translation pattern by masking out named-entities. We have used the NER data comes from the CONLL shared task.[5] Sentences in this dataset come from a collection of newswire articles from the Reuters Corpus. These sentences are annotated with four types of named entities: persons, locations, organisations and names of miscellaneous entities.

4.4.3 Models and Baselines

At the beginning of this project, methods were implemented using the Mantidae (Cohn et al., 2016) toolkit, on top of the DyNet (Neubig et al., 2017) library. However, as the idea of Chapter 6 required the support for the backpropagation through backpropagation operation, we moved to the PyTorch (Paszke et al., 2017) library. We have implemented the proposed Multi-Task Learning architectures using PyTorch, on top of the OpenNMT (Klein et al., 2017) toolkit. In our multi-task architecture, we do partial sharing of parameters, where the parameters of the top stacked layers

[4]https://catalog.ldc.upenn.edu/LDC2017T10
[5]https://www.clips.uantwerpen.be/conll2003/ner

	English → Vietnamese				English → Turkish				English → Spanish				English → Farsi			
	Dev		Test		Dev		Test		Dev		Test		Dev		Test	
	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU
NMT	58.4	22.83	55.7	24.15	104.2	8.55	101.2	8.5	73.1	14.49	73.6	13.44	96.1	12.16	96.7	11.95
MTL (Full)																
+ All Tasks	57.2†	22.71	55.1	24.71	90†	9.12†	88.8†	8.84	71.2†	14.63	71†	13.75	76.5†	12.67†	76.6†	12.45†
MTL (Partial)																
+ Semantic	56.8†	23.26	54.52†	25.09†	88.24†	9.24†	87.06†	8.25	71.1†	14.68	70.9†	13.95†	74.8†	12.73†	75.2	12.27
+ NER	57.7†	22.54	54.72†	25†	83.14†	9.3†	81.47†	9†	73.7	14.23	73.8	13.28	74.71†	13.34†	73.43†	13.17†
+ Syntactic	57.3†	23.02	54.72†	24.74	80.39†	9.72†	79.9†	9.01†	71.8†	14.39	71.6†	13.59	73.04†	13.43†	73.82†	13.43†
+ All Tasks	57.4†	23.42†	54.32†	25.22†	79.12†	10.06†	79.61†	9.53†	70.4†	15.14†	70.2†	14.11†	**72.84**†	**13.53**†	**73.43**†	**13.47**†
+ All+Adv.	**57**†	**23.49**†	**54.22**†	**25.56**†	**77.84**†	**10.44**†	**78.43**†	**9.98**†	**70.2**†	**15.2**†	**70**†	**14.28**†	73.33†	13.23†	73.43†	13.17†

Table 4.2: BLEU and TER scores of the baselines vs. our partial parameter sharing MTL architecture with various auxiliary tasks on the bilingual datasets. †: Statistically significantly better than the NMT baseline ($p < 0.05$).

are shared among the encoders of the tasks. Moreover, we share the parameters of the bottom layers of stacked decoders among the tasks. As we will show in Section 4.4.5, different sharing scenarios (i.e. the number of shared layers) lead to the best performance for different language pairs. In the experiments, we have used the best sharing scenario for each of the language pairs reported in Section 4.4.5. Additionally, source and target embedding tables are shared among the tasks, while the attention component is task-specific.[6] We compare against the following baselines:

- Baseline 1: The vanilla attentional SEQ2SEQ model trained only on translation training data (single-task).
- Baseline 2: The multi-tasking architecture proposed in (Niehues and Cho, 2017), which is a special case of our approach where all the parameters of encoders and/or decoders are shared among the tasks.[7] They have not used deep stacked layers in encoder and decoder as we do, so we extend their work to make it comparable with ours.

The configuration of models is as follows. The encoders and decoders make use of LSTM units with 512 hidden dimensions. For training, we used Adam algorithm (Kingma and Ba, 2014) with the initial learning rate of 0.001 for all of the tasks. Learning rates are halved when the performance on the corresponding dev set decreased. In order to speed-up the training, we use mini-batching with the size of 32. Dropout rates for both encoder and decoder are set to 0.3, and models are trained for 25 epochs where the best models are selected based on the perplexity on the dev set. λ for the adversarial training is set to 0.1. Once trained, the NMT model translates using the greedy search. We use BLEU (Papineni et al., 2002) and TER (Snover et al., 2006) to measure translation quality, and measure the statistical significance $p < 0.05$ using the approximate randomisation (Clark et al., 2011).

4.4.4 Results

Table 5.1 reports the BLEU and TER scores for the baseline and our proposed method on the four translation tasks mentioned above. It can be seen that the performance

[6] In our experiments, models with task-specific attention components achieved better results than those sharing them.

[7] We have used their reported best performing architecture (shared encoder) and changed the training schedule to ours.

	W/O Adaptation			W/ Adaptation		
	Partial	Part.+Adv.	Full	Partial	Part.+Adv.	Full
En → Vi	25.22	25.56	24.71	25.45	25.86	24.83
En → Tr	9.53	9.98	8.84	9.63	10.09	9.18
En → Sp	14.11	14.28	13.75	14.05	14.2	13.46
En → Fa	13.47	13.17	12.45	13.23	12.99	12.21

Table 4.3: Our method (partial parameter sharing) against Baseline 2 (full parameter sharing). Part.+Adv. means partial parameter sharing and adversarial training have been employed.

of Multi-Task Learning models is better than Baseline 1 (only MT task). This confirms that adding auxiliary tasks helps to increase the performance of the machine translation task.

As expected, the effect of different tasks are not similar across the language pairs, possibly due to the following reasons: (i) these translation tasks datasets come from different domains, so they have various degree of domain relatedness to the auxiliary tasks, and (ii) the BLEU and TER scores of the Baseline 1 show that the four translation models are on different quality levels which may entail that they benefit from auxiliary knowledge on different levels. In order to improve a model with low-quality translations due to language divergence, syntactic knowledge can be more helpful as they help to perform better reorderings. In a higher-quality model, however, semantic knowledge can be more useful as a higher-level linguistic knowledge. This pattern can be seen in the reported results: syntactic parsing leads to more improvement on Farsi translation which has a low BLEU score and high language divergence to English, and semantic parsing yields more improvement on the Vietnamese translation task which already has a high BLEU score. The NER task has led to a steady improvement in all the translation tasks, as it leads to better handling of named entities.

We have further added adversarial training to ensure the shared representation learned by the encoder is not contaminated by the task-specific information. As the MTL focuses on extracting extracting the common and task-invariant features. The results are in the last row of Table 5.1. The experiments show that adversarial training leads to further gains in MTL translation quality, except when translating into Farsi. We speculate this is due to the low quality of NMT for Farsi, where updating shared parameters with respect to the entropy of discriminator's predicted distribution may negatively affect the model.

Table 4.3 compares our Multi-Task Learning approach to Baseline 2. As Table 4.3, our partial parameter sharing mechanism is more effective than fully sharing

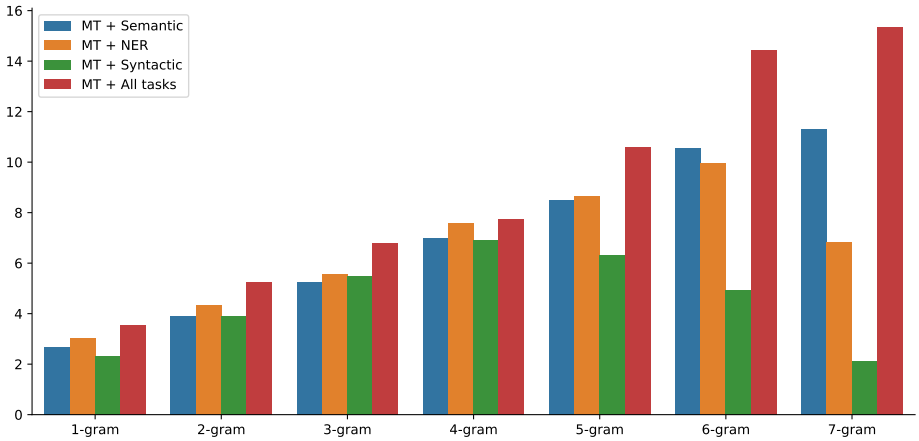


Figure 4.1: Percentage of more correct n-grams generated by the deep MTL models compared to the single-task model (only MT) for En→Vi translation.

the parameters (Baseline 2), due to its flexibility in allowing access to private task-specific knowledge. We also applied the adaptation technique (Niehues and Cho, 2017) as follows: upon finishing MTL training, we continue to train the best saved model only on the MT task for another 20 epochs and choose the best model based on perplexity on dev set. Adaptation has led to gains in the performance of our MTL architecture and Baseline 2 on two language pairs.

4.4.5 Analysis

How many layers of encoder/decoder to share? In this analysis, we want to see the effect of the number of shared layers in encoders and decoders on the performance of the MTL model. Figure 4.2 shows the results on the En→Vi translation task. The results confirm that the partial sharing of stacked layers is better than full sharing. Intuitively, partial sharing provides the model with an opportunity to learn task-specific skills via the private layers, while leveraging the knowledge learned from other tasks via shared layers. The figure shows the best sharing strategy for En→Vi translation task is 1-2 where 1 and 2 layers are shared among encoders and decoders, respectively. We have found that this strategy works very well for En→Tr and En→Fa as well. However, this sharing strategy does not work well for En→Sp task, and the performance of the resulted MTL model is even worse than the single-task NMT model. Interestingly, we have found it is better for En→Sp task to use 2-1 sharing strategy although the strategy does not work well for other language pairs. Therefore, there is a need for tuning over sharing strategies for each language pair. To

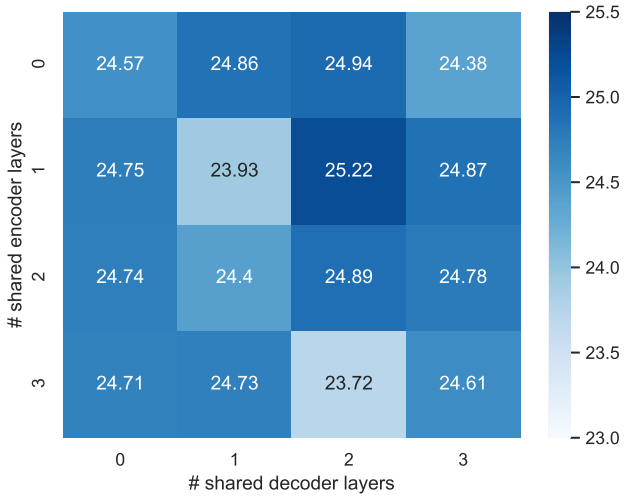


Figure 4.2: BLEU scores for different numbers of shared layers in encoder (from top) and decoder (from bottom). The vocabulary is shared among tasks while each task has its own attention mechanism.

speed up, the tuning process can be performed only on 1-2 and 2-1 sharing strategies.

Statistics of gold n-grams in MTL translations. Generating high order gold n-grams is hard. We analyse the effect of syntactic and semantic knowledge on generating gold n-grams in translations for En→Vi language pair.

For each sentence, we first extract n-grams in the gold translation, and then compute the number of n-grams which are common with the generated translations. Finally, after aggregating the results over the entire test set, we compute the percentage of additional gold n-grams generated by each MTL model compared to the ones in single-task MT model. The results are depicted in Figure 4.1. Interestingly, the MTL models generate more correct n-grams relative to the vanilla NMT model, as n increases.

Effect of the NER task. The NMT model has difficulty translating rarely occurred named-entities, particularly when the bilingual parallel data is scarce. We expect learning from the NER task leads the MTL model to recognise named-entities and learn underlying patterns for translating them. The top part in Table 4.4 shows an example of such a situation. As seen, the MTL is able to recognise all of the named-entities in the sentence and translate the while the single-task model missed "1981".

English	AIDS was discovered 1981 ; the virus , 1983 .
Reference	AIDS disease was discovered 1981 ; its virus on 1983 .
MT only model	AIDS was discovered 1998 ; virus , 1983 .
MT+NER model	AIDS was discovered 1981 ; virus , 1983 .
English	of course , you couldn't have done that all alone .
Reference	of course, you couldn't all that alone have done that .
MT only model	of course, you cannot have done that .
MT+semantic model	of course, you couldn't all that alone have done that .
English	in hospitals , for new medical instruments ; in streets for traffic control .
Reference	in hospitals , for instruments medical new ; in streets for control traffic .
MT only model	in hospitals , for equipments medical new , in streets for control instruments new .
MT+syntactic model	in hospitals , for instruments medical new ; in streets for control traffic .

Table 4.4: Example of translations on Farsi test set. In this examples each Farsi word is replaced with its English translation, and the order of words is reversed (Farsi is written right-to-left). The structure of Farsi is Subject-Object-Verb (SOV), leading to different word orders in English and Reference sentences.

For more analysis, we have applied a Farsi POS tagger (Feely et al., 2014) to gold translations. Then, we extracted n-grams with at least one noun in them, and report the statistics of correct such n-grams, similar to what reported in Figure 4.1. The resulting statistics is depicted in Figure 4.3. As seen, the MTL model trained on MT and NER tasks leads to the generation of more correct unigram noun phrases relative to the vanilla NMT, as n increases.

Effect of the semantic parsing task. Semantic parsing encourages a precise understanding of the source text, which would then be useful for conveying the correct meaning to the translation. The middle part in Table 4.4 is an example translation, showing that semantic parsing has helped NMT by understanding that "couldn't have done that all alone".

Effect of the syntactic parsing task. Recognising the syntactic structure of the source sentence helps NMT to translate phrases better. The bottom part of Table 4.4 shows an example of translation demonstrating such case. The source sentence is talking about "new medical instruments" and "traffic control", which the second term is correctly translated by the MTL model while vanilla NMT has mistakenly translated it to "control instruments new".

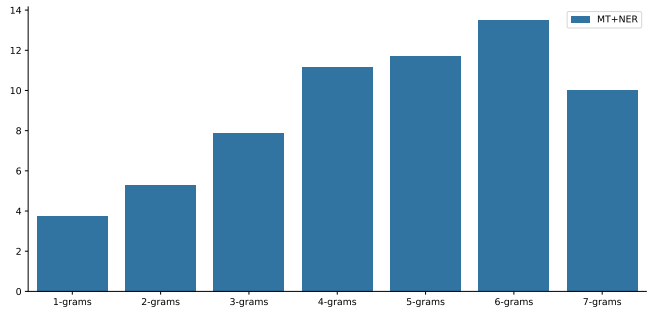


Figure 4.3: Percentage of more corrected n-grams with at least one noun generated by MT+NER model compared with the only MT model (only MT) for En→Vi language pair.

4.5 Summary

We have presented an approach to improve NMT in bilingually scarce scenarios, by leveraging curated linguistic resources in the source, including semantic parsing, syntactic parsing, and named entity recognition. This is achieved via a powerful MTL architecture, based on deep stacked encoders and decoders, to share common knowledge among the MT and auxiliary tasks. Our experimental results show the effectiveness of the proposed approach on improving the translation quality when translating from English to Vietnamese, Turkish, Spanish and Farsi in bilingually scarce scenarios.

Chapter 5

Adaptive Knowledge Sharing in Deep Seq2Seq MTL

This chapter is based on:
P. Zaremoodi, W. Buntine, G. Haffari, "Adaptive Knowledge Sharing in MTL: Improving Low-Resource NMT", Proceedings of Annual Meeting of Computational Linguistics (ACL), 2018.

In the previous chapter, we proposed an MTL architecture based on deep stacked layers in encoder/decoder and sharing the layers partially. The hypothesis behind partial sharing is that although similar tasks share some commonalities, they have some underlying differences which should be captured. Otherwise, the MTL models would not be able to make the best use of auxiliary tasks. Empirical results and analyses support the hypothesis by showing that dedicating task-specific parameters is crucial and provides the model with the capacity to learn task-specific representations. However, the approach has two limitations: (1) the shared components (layers) are shared among *all* of the tasks, causing this MTL approach to suffering from task interference; (2) finding the optimal sharing policy is a computationally expensive search. In this chapter, we address both of these limitations by learning how to *adaptively* control the amount of sharing. We extend the recurrent units with multiple *blocks* along with a *routing* network to dynamically control sharing of blocks conditioned on the task at hand, the input, and model state. Empirical results and analyses show the effectiveness of the proposed approach on four bilingually low-resource scenarios.

5.1 Introduction

MTL has been used for various NLP problems, e.g. dependency parsing (Peng et al., 2017), video captioning (Pasunuru and Bansal, 2017) key-phrase boundary classification (Augenstein and Søgaard, 2017), Chinese word segmentation, and text classification problem (Liu et al., 2017). More specifically, as seen in the previous chapter, MTL is an effective approach for injecting knowledge obtained from linguistic tasks to improve the performance of the underlying translation model. However, injecting knowledge from auxiliary tasks is a double-edged sword. While positive transfer may help to improve the performance of the main translation task, the negative transfer may have unfavourable effects and degrade the translation quality. This can be seen in the observations of the previous chapter that show dedicating task-specific parameters is crucial, as it provides the model with the capacity to learn task-specific representations. Although the proposed approach encouraged a more positive transfer compared to the full sharing approach, it still has two limitations: (1) Since the shared components (layers) are shared among *all* of the tasks, the approach still suffers from task interference, and it is not able to fully capture commonalities among *subsets* of tasks. (2) Experiments have shown that different language pairs have different optimal sharing strategy, and we need to perform a computationally expensive search to find the optimal strategy. Recently, Ruder et al. (2017) tried to address the task interference issue; however, their method is restrictive for SEQ2SEQ scenarios and does not consider the input at each time step to modulate parameter sharing.

In this chapter, we address the aforementioned issues by learning how to control the amount of sharing among the tasks "dynamically". We propose a novel recurrent unit which is an extension of conventional recurrent units. It has multiple flows of information, controlled by multiple *blocks*, and is equipped with a "routing network" to dynamically control sharing of blocks conditioning on the task at hand, the input, and model state. Empirical results on four low-resource translation scenarios, English to Vietnamese, Turkish, Spanish and Farsi, show the effectiveness of the proposed model.

In summary, the contributions of this chapter are as follows:

- We propose a novel recurrent unit with multiple flows of information along with a router network to dynamically control the amount of sharing among all tasks.
- We present empirical results and analyses that show the effectiveness of the proposed approach on leveraging the commonalities among subsets of tasks.

5.2 Routing Networks for Deep Neural Networks

Our work is related to the Mixture-of-Expert (MoE) architectures, where multiple experts (sub-networks) are learned to cover different subspaces of the problem. The beating heart of an MoE model is its *routing network* (aka gating network) responsible for modulating input and output of these experts. MoE has introduced two decades ago (Jacobs et al., 1991; Jordan and Jacobs, 1994), and has gained attention in deep learning recently. Shazeer et al. (2017) uses MoEs (feed-forward sub-networks) between stacked layers of recurrent units, to adaptively gate state information *vertically*. This is in contrast to our approach where the *horizontal* information flow is adaptively modulated, as we would like to minimise the task interference in MTL. Rosenbaum et al. (2018) has proposed a routing network for adaptive selection of non-linear functions for MTL. However, it is for fixed-size inputs based on a feed-forward architecture and is not applicable to SEQ2SEQ scenarios such as MT.

There are two variants of routing networks wrt type of their decisions: hard- versus soft-decision. In this chapter, we employed soft decision making. The hard-decision making is not differentiable, and could be solved by techniques like Gumbel-Softmax (Jang et al., 2016) or Reinforcement Learning (RL). It becomes more challenging as the environment is non-stationary as both the router and experts are trained simultaneously. After the publication of our work, it has been followed by Cases et al. (2019) that proposes a *hard* version of the routing mechanism using RL.

5.3 Seq2Seq MTL Using Recurrent Unit with Adaptive Routed Blocks

Our MTL model is based on the sequential encoder-decoder architecture with the attention mechanism (Section 2.2.2). The encoder/decoder consists of recurrent units to read/generate a sentence sequentially. In the previous chapter, we have done MTL by sharing the parameters of these recurrent units among tasks. Sharing the parameters of the recurrent units among different tasks is indeed sharing the *knowledge* for controlling the information flow in the hidden states. Sharing these parameters among *all* tasks may, however, lead to task interference or inability to leverages commonalities among *subsets* of tasks. We address this issue by extending the recurrent units with multiple *blocks*, each of which processing its own information flow through the time. The state of the recurrent unit at each time step is composed of the states of these blocks. The recurrent unit is equipped with a *routing* mechanism to softly

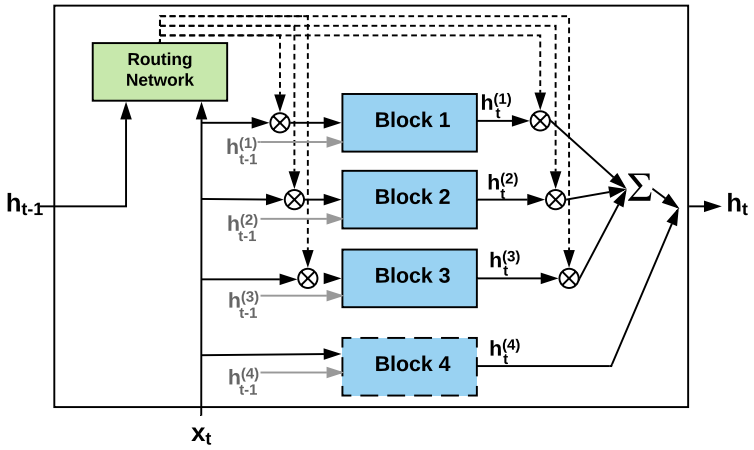


Figure 5.1: High-level architecture of the proposed recurrent unit with 3 shared blocks and 1 task-specific.

direct the input at each time step to these blocks (see Figure 5.1). Each block mimics an expert in handling different kinds of information, coordinated by the router. In MTL, the tasks can use different subsets of these shared experts.

Assuming there are n blocks in a recurrent unit, we share $n-1$ blocks among the tasks, and let the last one be task-specific.[1] The task-specific block receives the input of the unit directly while shared blocks are fed with modulated input by the routing network. The state of the unit at each time-step would be the aggregation of blocks' states.

5.3.1 Routing Mechanism

At each time step, the routing network is responsible to *softly* forward the input to the shared blocks conditioning on the input $\boldsymbol{x}_t$, and the previous hidden state of the unit $\boldsymbol{h}_{t-1}$ as follows:

$$\boldsymbol{s}_t = \text{ReLU}(\boldsymbol{W}_x \cdot \boldsymbol{x}_t + \boldsymbol{W}_h \cdot \boldsymbol{h}_{t-1} + \boldsymbol{b}_s), \tag{5.1}$$

$$\boldsymbol{\tau}_t = \text{softmax}(\boldsymbol{W}_\tau \cdot \boldsymbol{s}_t + \boldsymbol{b}_\tau), \tag{5.2}$$

where $\boldsymbol{W}$'s and $\boldsymbol{b}$'s are the parameters. Then, the i-th shared block is fed with the input of the unit modulated by the corresponding output of the routing network $\tilde{\boldsymbol{x}}_t^{(i)} = \boldsymbol{\tau}_t[i]\boldsymbol{x}_t$ where $\boldsymbol{\tau}_t[i]$ is the scalar output of the routing network for the i-th block.

The hidden state of the unit is the concatenation of the hidden state of the shared and task-specific parts $\boldsymbol{h}_t = [\boldsymbol{h}_t^{(shared)}; \boldsymbol{h}_t^{(task)}]$. The state of task-specific part is the

[1] Multiple recurrent units can be stacked on top of each other to consist a multi-layer component.

state of the corresponding block $\boldsymbol{h}_t^{(task)} = \boldsymbol{h}_t^{(n)}$, and the state of the shared part is the sum of states of shared blocks weighted by the outputs of the routing network $\boldsymbol{h}_t^{(shared)} = \sum_{i=1}^{n-1} \boldsymbol{\tau}_t[i]\boldsymbol{h}_t^{(i)}$.

5.3.2 Block Architecture

Each block is responsible to control its own flow of information via a standard gating mechanism. Our recurrent units are agnostic to the internal architecture of the blocks; we use the long short-term memory unit (Hochreiter and Schmidhuber, 1997b) in this chapter. For the i-th block the corresponding equations are as follows:

$$
\begin{aligned}
\boldsymbol{f}_t^{(i)} &= \sigma(\boldsymbol{W}_f^{(i)}\tilde{\boldsymbol{x}}_t^{(i)} + \boldsymbol{U}_f^{(i)}\boldsymbol{h}_{t-1}^{(i)} + \boldsymbol{b}_f^{(i)}),\\
\boldsymbol{r}_t^{(i)} &= \sigma(\boldsymbol{W}_r^{(i)}\tilde{\boldsymbol{x}}_t^{(i)} + \boldsymbol{U}_r^{(i)}\boldsymbol{h}_{t-1}^{(i)} + \boldsymbol{b}_r^{(i)}),\\
\boldsymbol{o}_t^{(i)} &= \sigma(\boldsymbol{W}_o^{(i)}\tilde{\boldsymbol{x}}_t^{(i)} + \boldsymbol{U}_o^{(i)}\boldsymbol{h}_{t-1}^{(i)} + \boldsymbol{b}_o^{(i)}),\\
\tilde{\boldsymbol{c}}_t^{(i)} &= \tanh(\boldsymbol{W}_{\tilde{c}}^{(i)}\tilde{\boldsymbol{x}}_t^{(i)} + \boldsymbol{U}_{\tilde{c}}^{(i)}\boldsymbol{h}_{t-1}^{(i)} + \boldsymbol{b}_{\tilde{c}}^{(i)}),\\
\boldsymbol{c}_t^{(i)} &= \boldsymbol{f}_t^{(i)} \odot \boldsymbol{c}_{t-1}^{(i)} + \boldsymbol{i}_t^{(i)} \odot \tilde{\boldsymbol{c}}_t^{(i)},\\
\boldsymbol{h}_t^{(i)} &= \boldsymbol{o}_t^{(i)} \odot \tanh(\boldsymbol{c}_t^{(i)}).
\end{aligned}
$$

5.3.3 Training Objective and Schedule.

The rest of the model is similar to attentional SEQ2SEQ model (Section 2.2.2) which computes the conditional probability of the target sequence given the source $P_{\boldsymbol{\Theta}}(\boldsymbol{y}|\boldsymbol{x}) = \prod_j P_{\boldsymbol{\Theta}}(y_j|\boldsymbol{y}_{<j}, \boldsymbol{x})$. For the case of training K SEQ2SEQ transduction tasks, we use the same objective function presented in Section 4.2. Since we aim to improve the underlying translation task the most, we use the Biased-MTL. We use the training schedule introduced in Section 4.2 where at each update iteration, selects a mini-batch from the main task and another one from a randomly selected auxiliary task. It ensures the presence of a training signal from the main task in all update iterations.

5.4 Experiments

For the experiments of this chapter, we have used the bilingual corpora and auxiliary tasks discussed in Sections 4.4.1 and 4.4.2, respectively.

	English → Vietnamese				English → Turkish				English → Spanish				English → Farsi			
	Dev		Test		Dev		Test		Dev		Test		Dev		Test	
	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU	TER	BLEU
NMT	58.4	22.83	55.7	24.15	104.2	8.55	101.2	8.5	73.1	14.49	73.6	13.44	96.1	12.16	96.7	11.95
MTL (Full)	57.2	22.71	55.1	24.71	90	9.12	88.8	8.84	71.2	14.63	71	13.75	76.5	12.67	76.6	12.45
MTL (Partial)	57.4	23.42	54.32	25.22	79.12	10.06	79.61	9.53	70.4	15.14	70.2	14.11	72.84	13.53	73.43	13.47
MTL (Routing)	**55.69**†	**24.17**†	**52.94**†	**26.39**†	**74.43**†	**11.09**†	**74.62**†	**10.62**†	**68.43**†	**16.23**†	**68.04**†	**15.1**†	**68.61**†	**14.9**†	**69.07**†	**14.64**†

Table 5.1: The performance of the baselines vs. our MTL architecture on the bilingual datasets. †: Statistically significantly better ($p < 0.05$) than the MTL (partial).

5.4.1 Models and Baselines

We have implemented the proposed MTL architecture along with the baselines in PyTorch on top of OpenNMT. For our MTL architecture, we used the proposed recurrent unit with 3 blocks in encoder and decoder. For the fair comparison in terms the of number of parameters, we used 3 stacked layers in both encoder and decoder components for the baselines. We compare against the following baselines:

- Baseline 1: The vanilla SEQ2SEQ model (Luong et al., 2015a) without any auxiliary task.
- Baseline 2: The MTL architecture proposed in (Niehues and Cho, 2017) which fully shares parameters in components. We have used their best performing architecture with our training schedule. We have extended their work with *deep* stacked layers for the sake of comparison.
- Baseline 3: The MTL architecture proposed in Chapter 4 which uses deep stacked layers in the components and shares the parameters of the top/bottom stacked layers among encoders/decoders of all tasks. For each language pair, we have used the best performing sharing strategy.

For the proposed MTL, we use recurrent units with 512 hidden dimensions for each block. The encoders and decoders of the baselines use GRU units with 400 hidden dimensions. The attention component has 512 dimensions. We use Adam optimiser (Kingma and Ba, 2014) with the initial learning rate of 0.001 for all the tasks. Learning rates are halved on the decrease in the performance on the dev set of the corresponding task. Mini-batch size is set to 32, and the dropout rate is 0.3. All models are trained for 25 epochs, and the best models are saved based on the perplexity on the dev set of the translation task.

For each task, we add special tokens to the beginning of source sequence (similar to (Johnson et al., 2017)) to indicate which task the sequence pair comes from. We have used the same data for all models and baselines.

We used greedy decoding to generate translation. In order to measure the translation quality, we use BLEU (Papineni et al., 2002) and TER (Snover et al., 2006) scores, and measure the statistical significance ($p < 0.05$) using the approximate randomisation (Clark et al., 2011).

Model	Number of Parameters
NMT	41M
MTL (Full)	51M
MTL (Partial)	100M
MTL (Routing)	105M

Table 5.2: The number of parameters for different models (for En→Vi language pair). For MTL models, we report the total number of parameters for all of the tasks (shared parameters are counted once).

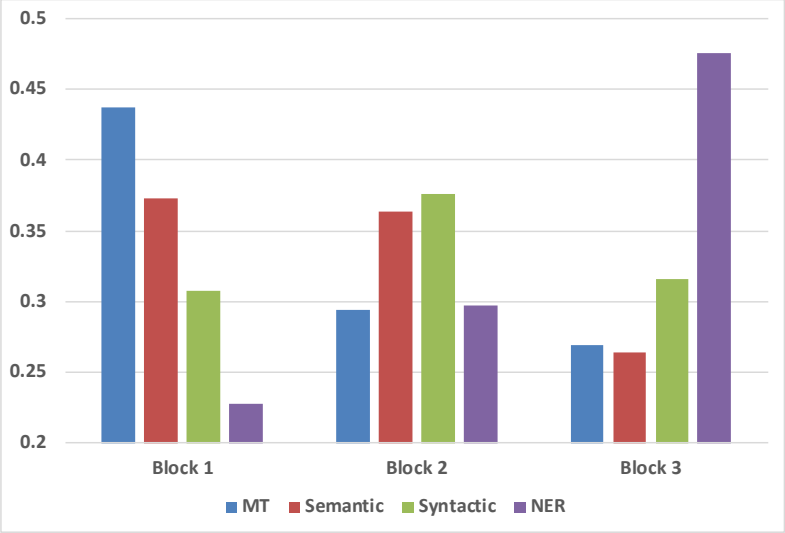


Figure 5.2: Average percentage of block usage for each task.

5.4.2 Results and analysis

Table 5.1 reports the results for the baselines and our proposed method on the two aforementioned translation tasks. As expected, the performance of MTL models is better than the baseline 1 (only MT task). As seen, partial parameter sharing is more effective than fully parameter sharing. Furthermore, our proposed architecture with adaptive sharing performs better than the other MTL methods on all tasks, and achieve up to +1 BLEU score improvements on the test sets. The improvements in the translation quality of NMT models trained by our MTL method may be attributed to less interference with multiple auxiliary tasks. The number of parameters for each model is reported in Table 5.2. As seen, our method performs better than the partial parameter sharing model by using a slightly higher number of parameters. In addition, the sharing scenario of Baseline 3 (partial sharing) should be manually tuned via a search that is computationally expensive. However, the proposed architecture does not require such a tuning as it performs on-the-fly adaptive sharing.

Figure 5.2 shows the average percentage of block usage for each task in an MTL model with 3 shared blocks, on the English-Farsi test set. We have aggregated the output of the routing network for the blocks in the encoder recurrent units over all the input tokens. Then, it is normalised by dividing the total number of input tokens. Based on Figure 5.2, the first and third blocks are more specialised (based on their usage) for the translation and NER tasks, respectively. The second block is mostly used by the semantic and syntactic parsing tasks, so specialised for them. This confirms our model leverages commonalities among subsets of tasks by dedicating common blocks to them to reduce task interference.

5.5 Summary

In this chapter, we address two of the main issues in previous MTL models: (1) task interference that causes the inability to fully capture the commonalities among subsets of tasks; (2) need for a computationally expensive search to find optimal sharing strategy. We address the issues by extending conventional recurrent units with multiple *blocks* along with a trainable *routing network*. Each block mimics an expert in handling a different kind of information, and the routing network guides the input to these blocks conditioning on the task at hand, the input, and the model state. Our experimental results on low-resource English to Vietnamese, Turkish, Spanish and Farsi datasets, show the effectiveness of the proposed approach compared to the full and partial sharing MTL models.

Part III

Multi-Task Learning: Training Schedule

Chapter 6

Adaptive scheduling for Deep Seq2Seq MTL

This chapter is based on:
P. Zaremoodi, G. Haffari, "Adaptively Scheduled Multitask Learning: The Case of Low-Resource Neural Machine Translation", Proceedings of the 3rd Workshop on Neural Machine Translation and Generation, Co-located with EMNLP 2019.

Scarcity of parallel sentence pairs is a major challenge for training high-quality Neural Machine Translation (NMT) models in bilingually low-resource scenarios, as NMT is data-hungry. As seen in part II, Multi-Task Learning can alleviate this issue by injecting inductive biases into NMT, using auxiliary syntactic and semantic tasks. However, an effective *training schedule* is required to balance the importance of tasks to get the best use of the training signal. The role of training schedule becomes even more crucial in *Biased-MTL* where the goal is to improve one of tasks the most, e.g. the translation quality in our setting. Current approaches for Biased-MTL are based on brittle *hand-engineered* heuristics that require trial and error, and should be (re-)designed for each learning scenario. To the best of our knowledge, ours is the first work on *adaptively* and *dynamically* changing the training schedule in Biased-MTL. We propose a rigorous approach for automatically reweighing the training data of the main and auxiliary tasks throughout the training process based on their contributions to the generalisability of the main NMT task. Empirical results and analyses show the effectiveness of the proposed approach on four bilingually low-resource scenarios. Additionally, our analyses shed light on the dynamic of needs throughout the training of NMT: from syntax to semantics.

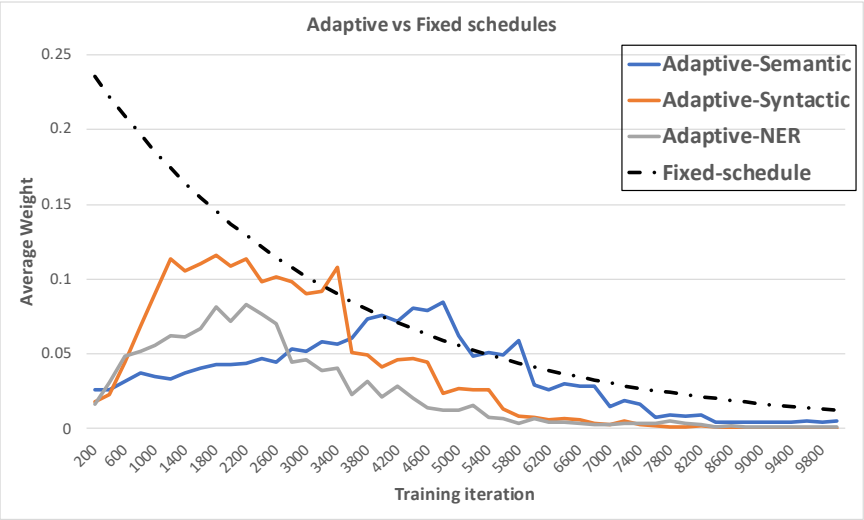


Figure 6.1: The dynamic in the relative importance of named entity recognition, syntactic parsing, and semantic parsing as the auxiliary tasks for the main machine translation task (based on our experiments in Section 6.3). The plot shows our proposed adaptive scheduling vs. fixed scheduling (Kiperwasser and Ballesteros, 2018) (scaled down for better illustration).

6.1 Introduction

The majority of the MTL literature, including part II of this thesis, has focused on investigating how to share common knowledge among the tasks through tying their parameters and joint training using standard algorithms. However, a big challenge of MTL is how to get the best signal from the tasks by changing their importance in the training process aka *training schedule*; see Figure 1.

Crucially, as discussed in Section 2.3.2, a proper training schedule would encourage positive transfer and prevent the negative transfer, as the inductive biases of the auxiliary tasks may interfere with those of the main task leading to degradation of generalisation capabilities. Most of the works on the training schedule focus on *general* MTL where the goal is to improve the performance of *all* tasks. They are based on addressing the imbalance in task difficulties and co-evolve easy and difficult tasks uniformly (performance-wise). These methods achieve competitive performance with existing single-task models of each task, and not necessarily much better performance (Chen et al., 2018b; Guo et al., 2018). On the other hand, *Biased-MTL* focuses on the *main* task to achieve higher improvements in it. In Chapter 4, we have proposed a fixed training schedule to balance out the importance of the main NMT task vs. auxiliary task to improve NMT the most. Kiperwasser and Ballesteros (2018) has shown the effectiveness of a changing training schedule through the MTL process.

However, their approach is based on *hand-engineered* heuristics, and should be (re-)designed and fine-tuned for every change in tasks or even training data.

In this chapter, for the first time to the best of our knowledge, we propose a method to *adaptively* and *dynamically* set the importance weights of tasks for *Biased-MTL*. By using *influence functions* from robust statistics (Cook and Weisberg, 1980; Koh and Liang, 2017), we adaptively examine the influence of training instances inside mini-batches of the tasks on the generalisation capabilities on the main task. The generalisation is measured as the performance of the main task on a validation set, separated from the training set, in each parameter update step *dynamically*. As our method is general and does not rely on hand-engineered heuristics, it can be used for effective learning of multi-task architectures beyond NMT.

We evaluate our method on translating from English to Vietnamese, Turkish, Spanish and Farsi, with auxiliary tasks including syntactic parsing, semantic parsing, and named entity recognition. Compared to the strong training schedule baselines, our method achieves considerable improvements in terms of BLEU score. Additionally, our analyses on the weights assigned by the proposed training schedule show that although the dynamic of weights are different for different language pairs, the underlying pattern is to gradually alter tasks importance from syntactic to semantic-related tasks.

In summary, our main contributions to MTL and low-resource NMT are as follows:

- We propose an effective training schedule for Biased-MTL that adaptively and dynamically set the importance of tasks throughout the training to improve the main task the most.
- We extensively evaluate four language pairs, and experimental results show that our model outperforms the hand-engineered heuristics.
- We present an analysis to better understand and shed light on the relative importance of auxiliary linguistic tasks throughout the training of an MTL model: from syntax to semantics.

6.2 Learning to Reweigh Mini-Batches

Suppose we are given a set of a main task along with K auxiliary tasks, each of which with its own training set $\mathcal{D}^k := \{(\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)})\}_{i=0}^{N_k}$. In multi-task formulation,

parameters are learned by maximising the log-likelihood objective:

$$\arg\max_{\boldsymbol{\Theta}_{mtl}} \sum_{k=0}^{K} \sum_{i=0}^{N_k} w_i^{(k)} \log P_{\boldsymbol{\Theta}_{mtl}}(\boldsymbol{y}_i^{(k)} | \boldsymbol{x}_i^{(k)}).$$

Without loss of generality, let us assume we use mini-batch-based stochastic gradient descent (SGD) to train the parameters of the multi-task architecture. In standard Multi-Task Learning $w_i^{(k)}$ is set to 1, assuming all of the tasks and their training instances have the same importance. Conceptually, these weights provide a mechanism to control the influence of the data instances from auxiliary tasks in order to maximise the benefit in the generalisation capabilities of the main task. Recently, (Kiperwasser and Ballesteros, 2018) and our work in Chapter 4 have proposed *hand-engineered* heuristics to set the importance weights. Following a fixed pre-defined schedule, the former one changes the importance of weights dynamically throughout the training process, e.g., iterations of the stochastic gradient descent (SGD). However, there is no guarantee that these fixed schedules give rise to learning the best inductive biases from the auxiliary tasks for the main task.

Our main idea is to adaptively determine the importance weights $w_i^{(k)}$ for *each* training instance based on its contribution to the generalisation capabilities of the MTL architecture for machine translation, measured on a validation set D^{val} separated from the training set which we call meta-validation set. As shown in Figure 6.2, at each parameter update iteration for the MTL architecture, the MTL training mini-batch is the concatenation of single mini-batches from all MTL tasks. We then assign an Adaptive Importance Weight (AIW) to *each* training instance in the MTL mini-batch, regardless of the task which they come from. In the experiments of Section 6.3, we will see that our proposed method *automatically* finds interesting patterns in how to best make use of the data from the auxiliary and main tasks, e.g. it starts by assigning higher weights (on average) to syntactic parsing which is then shifted to semantic parsing.

More specifically, we learn the AIWs based on the following optimisation problem:

$$\arg\min_{\hat{\boldsymbol{w}}} \; -\sum_{(\boldsymbol{x},\boldsymbol{y}) \in D^{val}} \log P_{\hat{\boldsymbol{\Theta}}_{mtl}(\hat{\boldsymbol{w}})}(\boldsymbol{y}|\boldsymbol{x}) \tag{6.1}$$

$$\hat{\boldsymbol{\Theta}}_{mtl}(\hat{\boldsymbol{w}}) := \boldsymbol{\Theta}_{mtl}^{(t)} + \eta \sum_{k=0}^{K} \sum_{i=0}^{|\boldsymbol{b}^{(k)}|-1} \hat{w}_i^{(k)} \nabla \log P_{\boldsymbol{\Theta}_{mtl}^{(t)}}(\boldsymbol{y}_i^{(k)} | \boldsymbol{x}_i^{(k)}) \tag{6.2}$$

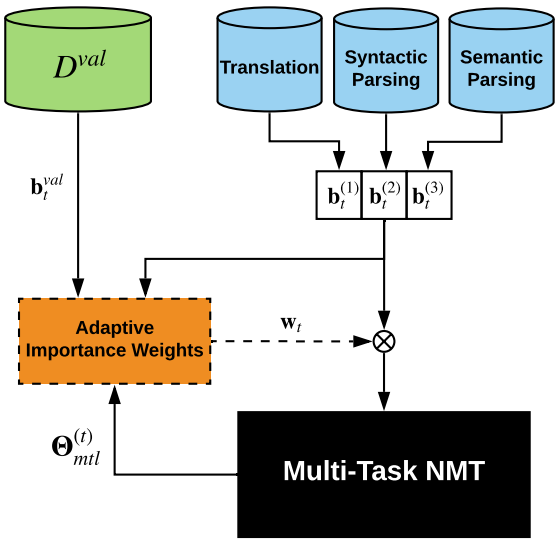


Figure 6.2: High-level idea for training an MTL architecture using adaptive importance weights (AIWs). Here, translation is the main task along with syntactic and semantic parsing as auxiliary linguistic tasks.

where $\hat{w}_i^{(k)}$ is the *raw* weight of the ith training instance in the mini-batch $\boldsymbol{b}^{(k)}$ of the kth task, $\hat{\mathbf{\Theta}}_{mtl}$ is the resulting parameters in case SGD update rule is applied on the current parameters $\mathbf{\Theta}_{mtl}^{(t)}$ using instances weighted by $\hat{\boldsymbol{w}}$. Following (Ren et al., 2018), we zero out negative raw weights, and then normalise them with respect to the other instances in the MTL training mini-batch to obtain the AIWs: $w_i^{(k)} = \frac{\tilde{w}_i^{(k)}}{\sum_{k'} \sum_{i'} \tilde{w}_{i'}^{(k')}}$ where $\tilde{w}_i^{(k)} = \text{ReLU}(\hat{w}_i^{(k)})$.

In the preliminary experiments, we observed that using $w_i^{(k)}$ as AIW does not perform well. We speculate that a small validation set does not provide a good estimation of the generalisation, hence influence of the training instances. This is exacerbated as we approximate the validation set by only one of its mini-batches for computational efficiency. Therefore, we hypothesise that the computed weights should not be regarded as the final verdict for the usefulness of the training instances. Instead, we regarded them as *rewards* for enhancing the training signals of instances that lead to a lower loss on the validation set. Hence, we use $1 + w_i^{(k)}$ as our AIWs in the experiments. The full algorithm is in Algorithm 1.

Implementation Details. As exactly solving the optimisation problem in eqn. 6.2 is challenging; we resort to an approximation and consider the raw weights as the gradient of the validation loss wrt the training instances' weights around zero. This is a notion called *influence* in robust statistics (Cook and Weisberg, 1980; Koh and

Algorithm 1 Adaptively Scheduled Multitask Learning

while t=0 ... T-1 **do**
  $\boldsymbol{b}^{(1)}, .., \boldsymbol{b}^{(K)} \leftarrow \text{SampleMB}(\boldsymbol{D}^{(1)}, .., \boldsymbol{D}^{(K)})$
  $\boldsymbol{b}^{(val)} \leftarrow \text{SampleMB}(\boldsymbol{D}^{(val)})$
  ▷ Step 1: Update model with initialised weights
  $\ell_i^{(k)} \leftarrow -\log P_{\boldsymbol{\Theta}^t_{mtl}}(\boldsymbol{y}_i^{(k)}|\boldsymbol{x}_i^{(k)})$ ▷ Forward
  $\hat{w}_{i,0}^{(k)} \leftarrow 0$ ▷ Initialise weights
  $\mathcal{L}_{trn} \leftarrow \sum_{k=1}^{K}\sum_{i=1}^{|\boldsymbol{b}^{(k)}|} \hat{w}_{i,0}^{(k)} \ell_i^{(k)}$
  $g_{trn} \leftarrow \text{Backward}(\mathcal{L}_{trn}, \boldsymbol{\Theta}^t_{mtl})$
  $\hat{\boldsymbol{\Theta}}^t_{mtl} = \boldsymbol{\Theta}^t_{mtl} + \eta g_{trn}$
  ▷ Step 2: Calculate loss of the updated model on validation MB
  $\mathcal{L}_{val} = -\sum_{i=1}^{|\boldsymbol{b}^{val}|} \log P_{\hat{\boldsymbol{\Theta}}^t_{mtl}}(\boldsymbol{y}_i|\boldsymbol{x}_i)$
  ▷ Step 3: Calculate raw weights.
  $g_{val} \leftarrow \text{Backward}(\mathcal{L}_{val}, \hat{w}_0^{(k)})$
  $\hat{w}^{(k)} = g_{val}$
  ▷ Step 4: Normalise weights to get AIWs
  $\tilde{w}_i^{(k)} = \text{ReLU}(\hat{w}_i^{(k)})$
  $w_i^{(k)} = \frac{\tilde{w}_i^{(k)}}{\sum_{k'}\sum_{i'} \tilde{w}_{i'}^{(k')}} + 1$
  ▷ Step 5: Update MTL with AIWs
  $\hat{\mathcal{L}}_{trn} \leftarrow \sum_{k=1}^{K}\sum_{i=1}^{|\boldsymbol{b}^{(k)}|} w_i^{(k)} \ell_i^{(k)}$
  $\hat{g}_{trn} \leftarrow \text{Backward}(\hat{\mathcal{L}}_{trn}, \boldsymbol{\Theta}^t_{mtl})$
  $\boldsymbol{\Theta}^{t+1}_{mtl} = \boldsymbol{\Theta}^t_{mtl} + \eta \hat{g}_{trn}$
end while

Liang, 2017).

More concretely, let us define the loss function $\mathcal{L}(\hat{\boldsymbol{\Theta}}_{mtl}) := -\sum_{i=0}^{|\boldsymbol{b}^{val}|-1} \log P_{\hat{\boldsymbol{\Theta}}_{mtl}}(\boldsymbol{y}_i|\boldsymbol{x}_i)$, where $\boldsymbol{b}^{val}$ is a mini-batch from the validation set. The training instances' raw weights, i.e. influences, are then calculated using the chain rule:

$$\begin{aligned}\hat{\boldsymbol{w}} &= \nabla_{\hat{\boldsymbol{w}}_0}\mathcal{L}(\hat{\boldsymbol{\Theta}}_{mtl}(\hat{\boldsymbol{w}}_0))\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}} \\ &= \nabla_{\hat{\boldsymbol{\Theta}}_{mtl}}\mathcal{L}(\hat{\boldsymbol{\Theta}}_{mtl})\Big|_{\hat{\boldsymbol{\Theta}}_{mtl}=\boldsymbol{\Theta}^{(t)}_{mtl}} \cdot \nabla_{\hat{\boldsymbol{w}}_0}\hat{\boldsymbol{\Theta}}_{mtl}(\hat{\boldsymbol{w}}_0)\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}}\end{aligned}$$

The last term $\nabla_{\hat{\boldsymbol{w}}_0}\hat{\boldsymbol{\Theta}}_{mtl}$ involves backpropagation through $\hat{\boldsymbol{\Theta}}_{mtl}$ wrt $\hat{\boldsymbol{w}}_0$, which according to eqn. 6.2, involves an inner backpropagation wrt $\boldsymbol{\Theta}_{mtl}$. The computation graph is depicted in Figure 6.3.

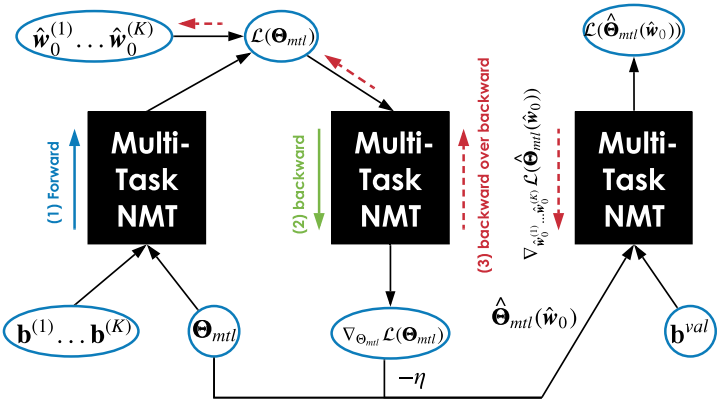


Figure 6.3: Computation graph of the proposed method for adaptively determining weights.

6.3 Experiments

6.3.1 Bilingual Corpora and Auxiliary Tasks

For the experiments of this chapter, we have used the bilingual corpora and auxiliary tasks discussed in Sections 4.4.1 and 4.4.2, respectively. In addition, we use separate Val (meta-validation) sets for the proposed AIW-based approach. For English-Vietnamese, "tst2012" is divided and used as Dev and Val sets (with the ratio 2 to 1). For English-Turkish and English-Spanish, we use "newstest2017" and "newstest2012" parts as Val set, respectively. For English-Farsi, a random 3K subset of the corpus is held out as Val set. For a fair comparison, we add the meta-validation set used in the AIW-based approach to the training set of the competing baselines.

6.3.2 MTL architecture and training schedule

Since partial sharing is shown to be more effective than full sharing, we use the MTL architecture proposed in Chapter 4 with the best sharing strategy for each language pair. We have implemented the methods using PyTorch[1] on top of OpenNMT.

Fixed *hand-engineered* schedule baselines. We use different MTL scheduling strategies where at each update iteration:

- **Uniform**: Selects a random mini-batch from all of the tasks.

[1]We have modified the PyTorch source code as the official version does not support some of the backpropagation-through-backpropagation operations on GPU at the time of writing this thesis.

- **Biased**: It is introduced in Section 4.2 and selects a random mini-batch from the translation task (bias towards the main task) and another one for a randomly selected task.

We also use schedules proposed in (Kiperwasser and Ballesteros, 2018). They consider a slope parameter[2] α and the fraction of training epochs done so far, $t = sents/||corpus||$. The schedules determine the probability of selecting each of the tasks as the source of the next training pair. In each of these schedules the probability of selecting the *main task* is:

- **Constant:** $P_m(t) = \alpha$; When α is set to 0.5, it is similar to the Biased schedule we have seen before.
- **Exponential:** $P_m(t) = 1 - e^{-\alpha t}$; In this schedule the probability of selecting the main task increases exponentially throughout the training.
- **Sigmoid:** $P_m(t) = \frac{1}{1 + e^{-\alpha t}}$; Similar to the previous schedule, the probability of selecting the main task increases, following a sigmoid function.

In each of these schedules, the rest of the probability is uniformly divided among the *remaining tasks*. By using them, a mini-batch can have training pairs from different tasks which makes it inefficient for partially shared MTL models. Hence, we modified these schedules to select the source of the next training mini-batch.

Combination of Adaptive and Fixed schedules As mentioned in Section 6.2, we assign an AIW to each training instance inside mini-batches of *all* tasks, i.e. applying AIWs on top of Uniform schedule. Additionally, we also apply it on top of Biased schedule to analyse the effect of the combination of AIWs (for instances) and a *hand-engineered* heuristic (for mini-batch selection).

1

6.3.3 Results and Analysis

The results for baselines and the proposed method are reported in Table 7.2.[3] As seen, our method has made better use of the auxiliary tasks and achieved the highest performance. It shows that while some of the *heuristic*-based schedules are beneficial, our proposed Adaptive Importance Weighting approach outperforms them. The

[2]Following their experiments, we set α to 0.5.

[3]METEOR score (Banerjee and Lavie, 2005) is reported only for Spanish as it is the only target languages in our experiments which is officially supported by it.

	English→Vietnamese		English→Turkish		English→Spanish				English→Farsi	
	BLEU		BLEU		BLEU		METEOR		BLEU	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
MT only	22.83	24.15	8.55	8.5	14.49	13.44	31.3	31.1	12.16	11.95
MTL with Fixed Schedule										
+ Uniform	23.10	24.81	9.14	8.94	12.81	12.12	29.6	29.5	13.51	13.22
+ Biased (Constant)[†‡]	23.42	25.22	10.06	9.53	15.14	14.11	31.8	31.3	13.53	13.47
+ Exponential[‡]	23.45	25.65	9.62	9.12	12.25	11.62	28.0	28.1	14.23	13.99
+ Sigmoid[‡]	23.35	25.36	9.55	9.01	11.55	11.34	26.6	26.9	14.05	13.88
MTL with Adaptive Schedule										
+ Biased + **AIW**	23.95[◇]	25.75	10.67[◇]	10.25[◇]	11.23	10.66	27.5	27.4	14.41	14.2
+ Uniform + **AIW**	**24.38**[◇]	**26.68**[◇]	**11.03**[◇]	**10.81**[◇]	**16.05**[◇]	**14.95**[◇]	**33.0**[◇]	**32.5**[◇]	**15.24**[◇]	**15.08**[◇]

Table 6.1: Results for three language pairs. "+ AIW" indicates Adaptive Importance Weighting is used in training. [†]: Proposed in Chapter 4, [‡]: Proposed in (Kiperwasser and Ballesteros, 2018). [◇]: Statistically significantly better ($p < 0.05$) than the best MTL with fixed schedule.

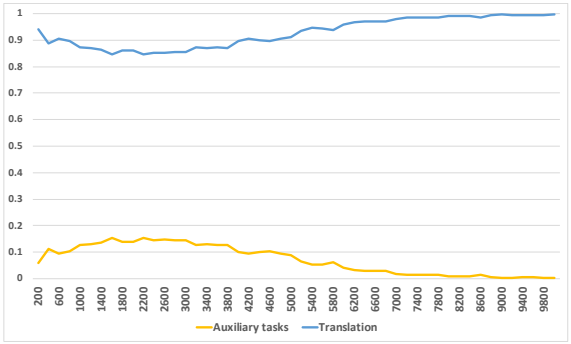
reason is likely that the *hand-engineered* strategies do not consider the state of the model, and they do not distinguish among the auxiliary tasks.

It is interesting to see that the Biased schedule is beneficial for standard MTL, while it is harmful when combined with the AIWs. The standard MTL is not able to select training signals on-demand, and using a biased heuristic strategy improves it. However, our weighting method can selectively filter out training signals; hence, it is better to provide all of the training signals and leave the selection to the AIWs.

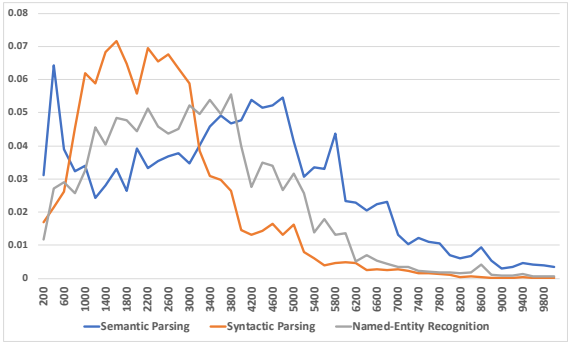
Analysis on how/when auxiliary tasks have been used? This analysis aims to shed light on how AIWs control the contribution of each task through the training. As seen, our method has the best result when it is combined with the Uniform MTL schedule. In this schedule, at each update iteration, we have one mini-batch from each of the tasks, and AIWs are determined for all of the training pairs in these mini-batches. For this analysis, we divide the training into 200 update iteration chunks. In each chunk, we calculate the average weights assigned to the training pairs of each task.

Figure 6.1 shows the results of this analysis for the MTL model trained with En→Vi as the main task. and Figure 6.4 shows the results of this analysis for En→Es and En→Tr. Also, it can be seen that at the beginning of the training, the Adaptive Importance Weighting mechanism gradually increases the training signals which come from the auxiliary tasks. However, after reaching a certain point in the training, it will gradually reduce the auxiliary training signals to concentrate more on the adaptation to the main task. It can be seen that the weighting mechanism distinguishes the importance of auxiliary tasks. More interestingly, it can be seen that for the En→Tr, the contribution of NER task is more than the syntactic parsing while for the other languages we have seen the reverse. It shows that our method can *adaptively* determine the contribution of the tasks by considering the demand of the main translation task.

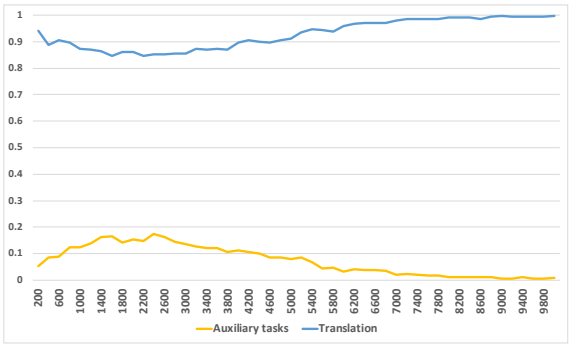
As seen, it gives more weight to the syntactic tasks at the beginning of the training while it gradually reduces their contribution and increases the involvement of the semantics-related task. We speculate the reason is that at the beginning of the training, the model requires more lower-level linguistic knowledge (e.g. syntactic parsing and NER) while over time, the needs of model gradually change to higher-level linguistic knowledge (e.g. semantic parsing).



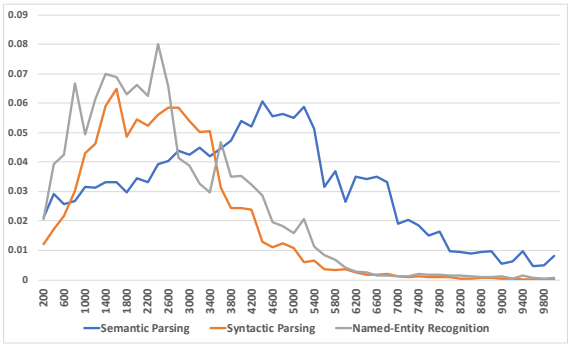
(a) Translation task (En→Es) vs. auxiliary tasks.



(b) Auxiliary tasks vs. each other.

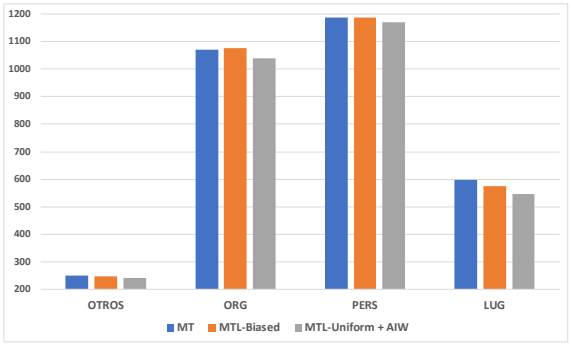


(c) Translation task (En→Tr) vs. auxiliary tasks.

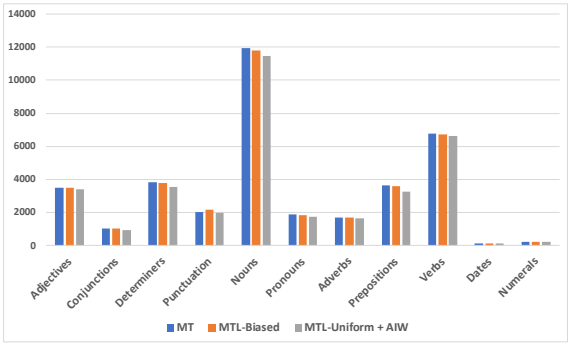


(d) Auxiliary tasks vs. each other.

Figure 6.4: Weights assigned to the training pairs of different tasks (averaged over 200 update iteration chunks). Y-axis shows the average weight and X-axis shows the number of update iteration. In the top figures, the main translation task is English→Spanish while in the bottom ones it is English→Turkish.



(a) Named-Entities



(b) Part-of-Speech tags

Figure 6.5: The number of words in the gold English→Spanish translation which are missed in the generated translations (lower is better). Missed words are categorised by their tags (Part-of-Speech and named-entity types).

Analysis of The Effect of Auxiliary Tasks on The Generated Translations In this analysis, we want to take a deeper look at the generated translations and see how the proposed method improves the quality of the translations. More specifically, we want to compare the number of words in the gold translations which are missed in the generated translations produced by the following systems: (i) MT only; (ii) MTL-Biased; (iii) MTL-Uniform + AIW. To find out what kind of knowledge is missed in the process of generating the translations, we categorise words by their Part-of-Speech tags and named-entities types. We have done this analysis on En→Es language pair as there are accurate annotators for the Spanish language. We use Stanford POS tagger (Toutanova et al., 2003) and named-entity recogniser (Finkel et al., 2005) to annotate Spanish gold translations. Then, we categorise the missed words in the generated translations concerning these tags, and count the number of missed words in each category. Figure 6.5 depicts the result. As seen in Figure 6.5a, the knowledge learned from auxiliary tasks helps the MTL model to miss less number of named-entities during translation. Moreover, AIWs help the MTL model further by making better use of the knowledge conveyed in the auxiliary tasks. We can see the same pattern for the POS of missed words. As seen, for most POS categories, the standard MTL has missed less number of words in comparison to the MT only baseline. Furthermore, our method helps the MTL model to miss even less amount of words in every of the POS categories (specifically in Noun and Preposition categories). We speculate the reason is that the AIWs makes it possible to control the contribution of each of the auxiliary tasks separately and taking into account the demand of the model at each stage of the training procedure.

6.4 Summary

This chapter presents a rigorous approach for adaptively and dynamically changing the training schedule in Biased-MTL to make the best use of auxiliary tasks. To balance the importance of the auxiliary tasks versus the main task, we re-weight training data of tasks to adjust their contributions to the generalisation capabilities of the resulted model on the main task. Our experimental results on English to Vietnamese/Turkish/Spanish/Farsi show up to +1.2 BLEU score improvement compared to strong baselines. Additionally, the analyses show that the proposed method *automatically* finds a schedule which places more importance on the auxiliary syntactic tasks at the beginning while gradually altering the importance toward the auxiliary semantic task.

Chapter 7

Learning to Multi-Task Learn

This chapter is based on:
P. Zaremoodi, G. Haffari, "Learning to Multi-Task Learn for Better Neural Machine Translation", Submitted to the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI), 2020.

As discussed in the previous chapter, one of the main challenges in MTL is to devise effective *training schedules*, prescribing when to make use of the auxiliary tasks during the training process to fill the knowledge gaps of the main task, a setting referred to as *Biased-MTL*. In that chapter, we present a rigorous approach for adaptively and dynamically changing the training schedule in Biased-MTL by re-weight training instances of tasks to adjust their contributions to the generalisation capabilities of the resulted model on the main task. We have shown that the proposed method is effective and has devised interesting weighting patterns. One major limitation, however, is the computational complexity of the method. This chapter is built upon the learned lessons of the previous chapter and introduces a generalised framework to extend our previous work and make it more efficient.

We propose a novel framework for *learning* the training schedule, i.e., *learning to multi-task learn*, for a given MTL setting. We formulate the training schedule as a Markov Decision Process which paves the way to employ policy learning methods to *learn* the scheduling policy. We effectively and efficiently learn the training schedule policy within the Imitation Learning framework using an *oracle policy* algorithm that dynamically sets the importance weights of auxiliary tasks based on their contributions to the generalisability of the main task. Experiments on both low-resource

and standard NMT settings show the resulting automatically learned training schedulers are competitive with the best heuristics and lead to up to +1.1 BLEU score improvements.

7.1 Introduction

The training schedule, the beating heart of MTL, is responsible for balancing out the importance (participation rate) of different tasks throughout the training process, in order to make the best use of the knowledge provided by the tasks. In the previous chapter, we have proposed an approach to automatically re-weight training instances of tasks to adjust their contributions to the generalisation capabilities of the resulted model on the main task. Although the proposed method is effective and has found interesting patterns, it is computationally expensive. This chapter is built upon the hypotheses derived from the lessons learnt from the previous chapter: (1) As seen, although we have calculated weights for individual training instances, interesting task-related patterns have also emerged, e.g., shifting the importance from syntax to semantic-related tasks. Therefore, we may approximate the instance-weights by task-weights. (2) We have seen the weights of auxiliary tasks are small relative to the main translation task, which means they have smaller contributions. However, the computational complexity for processing an instance/mini-batch is almost agnostic to the weight coefficient in its loss objective. As the task-weights determine the contribution of tasks, we can use them as "sampling ratios" instead of coefficients in the loss. Previously, at each iteration, we use a mini-batch from all of the tasks and use weights as coefficients in the loss objective. Now, we only select one of the tasks wrt the task-weights (i.e. higher weight means the higher chance for selection). The expected training objective would be the same while we save computation time. (3) On top of all the above mentioned hypotheses, the training schedule is a sequential decision making process, so we can *learn* it.

To the best of our knowledge, there is no approach for automatically *learning* a dynamic training schedule for Biased-MTL. In this chapter, we propose a novel framework for automatically *learning* how to multi-task learn to maximally improve the main translation task. This is achieved by formulating the problem as a Markov Decision Process (MDP), enabling us to treat the training scheduler as the policy. We solve the MDP by proposing an effective yet computationally expensive *oracle policy* (inspired from the previous chapter) that sets the participation rates of auxiliary tasks with respect to their contributions to the generalisation capability of the main

translation task. In order to scale up the decision making, we use the oracle policy as a teacher to train a scheduler network within the Imitation Learning framework using DAGGER (Ross et al., 2011). Thanks to the ∼8X speedup, we jointly train and use the scheduler along with the MTL model in the course of a *single* run. Our experimental results on low-resource (English to Vietnamese/Turkish/Spanish/Farsi) setting show up to +1.1 BLEU score improvements over heuristic training schedules and comparable to the approach of the previous chapter. Additionally, we investigate the effectiveness of the proposed approach on a high-resource setting for WMT17 English to German to show the scalability of the approach further. Our analyses on the automatically learned scheduling policies show interesting patterns among auxiliary tasks for high-resource setting, e.g. gradually altering importance from syntactic to auxiliary semantic tasks (similar to what we have seen for the low-resource setting in the previous chapter).

To summarise, this chapter contributions are as follows:

- We propose a novel framework for *learning* the training schedule in MTL by formulating it as an MDP.
- We use the approach proposed in the previous chapter as an algorithmic *oracle policy* that *adaptively* and *dynamically* selects tasks throughout the training. As the algorithmic oracle is computationally demanding, we scale up our approach by introducing a scheduler policy trained and used simultaneously using Imitation Learning to mimic the oracle policy.
- We evaluate our approach in low/high resource bilingual data scenarios using RNN/Transformer architectures. Our analyses unveil an interesting linguistic phenomenon in MTL training of NMT for high-resource setting: from syntax to semantics.

7.2 MTL training schedule as a Markov Decision Process

MTL Setup Suppose we are given a set of a main task along with K auxiliary tasks, each of which with its own training set $\mathcal{D}^k := \{(\boldsymbol{x}_i^{(k)}, \boldsymbol{y}_i^{(k)})\}_{i=0}^{N_k}$, where $k = 0$ denotes the main task. We are interested to train a probabilistic MTL architecture $P_{\boldsymbol{\Theta}_{mtl}}(\boldsymbol{y}|\boldsymbol{x})$, which accurately maps $\boldsymbol{x} \in \mathcal{X}^k$ to its corresponding $\boldsymbol{y} \in \mathcal{Y}^k$ for each task k. Without loss of generality, we assume that the model parameters $\boldsymbol{\Theta}$ are fully

shared across all the tasks. The MTL parameters are then learned by maximising the log-likelihood objective:

$$\arg\max_{\boldsymbol{\Theta}_{mtl}} \sum_{k=0}^{K} w^{(k)} \sum_{i=0}^{N_k} \log P_{\boldsymbol{\Theta}}(\boldsymbol{y}_i^{(k)}|\boldsymbol{x}_i^{(k)}). \tag{7.1}$$

Typically, the tasks are assumed to have a predefined importance, e.g. their weights are set to the uniform distribution $w^{(k)} = \frac{1}{K+1}$, or set proportional to the size of the tasks' training datasets. As mentioned, we consider task-level weights instead of instance-level ones.

Assuming an iterative learning algorithm, the standard steps in training the MTL architecture at time step t are as follows: (i) A collection of training mini-batches $\boldsymbol{b}_t := \{\boldsymbol{b}_t^k\}_{k=0}^K$ is provided, where $\boldsymbol{b}_t^k$ comes from the k-th task, and (ii) The MTL parameters are re-trained by combining information from the mini-batches according to tasks' importance weights. For example, the mini-batch gradients can be combined according to the tasks' weights, which can then be used to update the parameters using a gradient-based optimisation algorithm.

Markov Decision Process Formulation As shown in the previous chapter, tasks' weights in the MTL training objective (eqn. 7.1) should be *dynamically* changed during the training process to maximise the benefit of the main task from the auxiliary tasks. Intuitively, dynamically changing the weights provides a mechanism to inject proper linguistic knowledge to the MTL architecture, in order to maximally increase the generalisation capabilities on the main task.

In this chapter, we automatically learn MTL training policies/schedules from the data in order to optimally change tasks' importance weights (see Figure 7.1). To achieve this goal, we formulate the training process of an MTL architecture as a Markov Decision Process (MDP), where its elements $(S, A, Pr(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t), R)$ are as follows.

• At the time step t in the training process, the state $\boldsymbol{s}_t$ includes any feature which summarises the history of the training trajectory. For example, it may include the loss values encountered during the training of the intermediate models or any footprint of the model parameters. The state space S is accordingly specified by these features.

• Provided with a collection of training mini-batches $\boldsymbol{b}_t$, the *action* $\boldsymbol{a}_t$ then corresponds to the tasks' importance weights. That is, the action space A is the K-dimensional simplex $\triangle^K$, consisting of $K+1$ dimensional vectors with non-negative elements

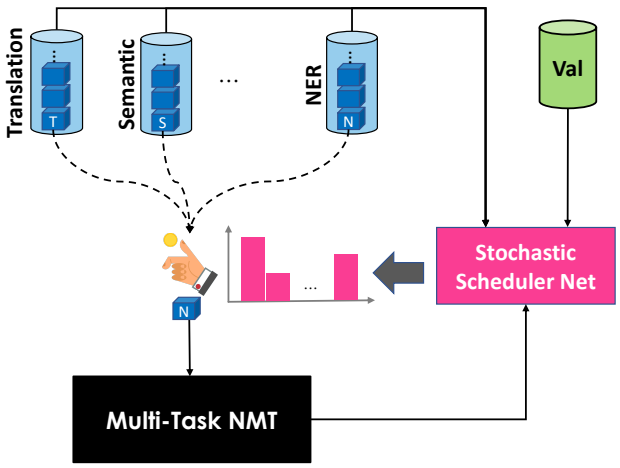


Figure 7.1: Overview of training an MTL architecture using adaptive scheduling. Translation is the main task with syntactic and semantic parsing as auxiliary linguistic tasks.

where the sum of the elements is one.

• $Pr(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ is the transition function, which determines updated model parameters in the next time step, having an action $\boldsymbol{a}_t \in A$ taken at the state $\boldsymbol{s}_t \in S$. In other words, it corresponds to the update rule of the underlying optimisation algorithm, e.g. Stochastic Gradient Descent (SGD), having decided about the tasks' weights in the MTL training objective (eqn. 7.1) in the current state.

• $R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$ specifies the instantaneous reward, having an action $\boldsymbol{a}_t$ taken from the state $\boldsymbol{s}_t$ and transitioned to the new state $\boldsymbol{s}_{t+1}$. In our case, it should show the increase in the generalisation capabilities on the main task, having decided upon the tasks' importance weights, and accordingly updated the MTL model parameters. It is not trivial how to formalise and quantify the increase of the generalisation of the MTL architecture on the main task. Our idea is to use a held-out validation set from the main task $\mathcal{D}^{val}$, and then use a loss function on this set to formalise the generalisation capability. More specifically, we take $-loss(\boldsymbol{\Theta}_{t+1}, \mathcal{D}^{val}) + loss(\boldsymbol{\Theta}_t, \mathcal{D}^{val})$ as a proxy for the increase in the generalisation capability of the main task.

• Having set up the MDP formulation, our aim is to find the *optimal* policy producing the best MTL training schedule $\boldsymbol{a}_t = \pi_{\boldsymbol{\phi}}(\boldsymbol{\Theta}_t, \boldsymbol{b}_t)$, where $\boldsymbol{\phi}$ is the parameter of the *policy network*. The policy prescribes what should be the importance of the MTL tasks at the current state, in order to get the best performing model on the main task in the long run. We consider $\pi_{\boldsymbol{\phi}}(\boldsymbol{\Theta}_t, \boldsymbol{b}_t) \in \triangle^K$ as a probability distribution for selecting the next task and its training mini-batch. It gives rise to a stochastic policy for task selection.

The optimal policy is found by maximising the following objective function:

$$\arg\max_{\boldsymbol{\phi}} \mathbb{E}_{\pi_{\boldsymbol{\phi}}}\Big[\sum_{t=0}^{T} R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\Big] \tag{7.2}$$

where T corresponds to the maximum training steps for the MTL architecture. Crucially, maximising the above long-term reward objective corresponds to finding a policy, under which the validation loss of the resulting MTL model (at the end of the training trajectory) is minimised. In this chapter, we will present methods to provide such optimal/reasonable stochastic policies for training MTL architectures.

7.3 An Oracle Policy for MTL-MDP

In this section, we provide an *oracle* policy π^{oracle}, which gives an approximation of the optimal policy inspired by the method proposed in the previous chapter. The basic idea is to find an importance weight vector for the tasks which minimises the loss of the main task on a validation set. In other words, our oracle policy *rolls-out* for one step from the current state in order to reduce computational complexity. However, we note that it is still extremely computationally demanding to compute oracle actions. Therefore, we scale up our approach in the next section using a scheduler policy which mimics the oracle policy.

More specifically, our oracle learns the optimal tasks' weights $\boldsymbol{w}_{opt}$ based on the following optimisation problem:

$$\boldsymbol{w}_{opt} := \arg\min_{\hat{\boldsymbol{w}} \in \triangle^K} \underbrace{-\sum_{(\boldsymbol{x},\boldsymbol{y}) \in \boldsymbol{b}^{val}} \log P_{\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}})}(\boldsymbol{y}|\boldsymbol{x})}_{\mathcal{L}(\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}))}, \tag{7.3}$$

such that

$$\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}) := \boldsymbol{\Theta}_t + \eta \sum_{k=0}^{K} \hat{w}^{(k)} \sum_{i=0}^{|\boldsymbol{b}^k|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_i^{(k)}|\boldsymbol{x}_i^{(k)}), \tag{7.4}$$

where we use a mini-batch $\boldsymbol{b}^{val}$ from the validation set for computational efficiency. To find the optimal importance weights, we do one-step projected gradient descent on the objective function $\mathcal{L}(\boldsymbol{\Theta}_t)$ starting from zero. That is, we firstly set $\hat{\boldsymbol{w}}_{opt} := \nabla_{\hat{\boldsymbol{w}}_0} \mathcal{L}(\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0))\Big|_{\hat{\boldsymbol{w}}_0 = \boldsymbol{0}}$, then zero out the negative elements of $\hat{\boldsymbol{w}}$, and finally normalise the resulting vector to project back to the simplex and produce $\boldsymbol{w}_{opt} \in \triangle^K$. Our approach to define $\hat{\boldsymbol{w}}$ is based on the classic notion of *influence* functions from robust statistics (Koh and Liang, 2017; Ren et al., 2018; Cook and Weisberg, 1980).

Computing $\nabla_{\hat{\boldsymbol{w}}_0}\mathcal{L}(\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0))$ is computationally expensive and complicated, as it involves backpropagation wrt $\hat{\boldsymbol{w}}_0$ to a function which itself includes backpropagation wrt $\boldsymbol{\Theta}$. Interestingly, as proved at the end of this section, the component $\hat{w}_{opt}^{(k)}$ is proportional to the inner-product of the gradients of the loss over the mini-batches from the validation set and the k-th,

$$\sum_{j=0}^{|\boldsymbol{b}^{val}|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_j^{(val)}|\boldsymbol{x}_j^{(val)}) \cdot \sum_{i=0}^{|\boldsymbol{b}^{k}|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_i^{(k)}|\boldsymbol{x}_i^{(k)}).$$

This approach is also computationally expensive and limits the scalability of the oracle, as it requires backpropagation over the mini-batches from *all* of the tasks and validation set.

Proof for calculating weights efficiently. We prove that the raw weight for k-th task ($\hat{w}^{(k)}$) is proportional to the inner-product of the gradients of the loss over the mini-batches from the validation set and the task.

$$\begin{aligned} \hat{w}_{opt}^{(k)} &= \nabla_{\hat{w}_0^{(k)}}\mathcal{L}(\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0))\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}} \\ &= \nabla_{\hat{\boldsymbol{\Theta}}}\mathcal{L}(\hat{\boldsymbol{\Theta}})\Big|_{\hat{\boldsymbol{\Theta}}=\boldsymbol{\Theta}_t} \cdot \nabla_{\hat{w}_0^{(k)}}\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0)\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}} \end{aligned} \tag{7.5}$$

where the first term is the gradient of the loss of the updated model ($\mathcal{L}(\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0))$ on Val set with respect to the parameters before update ($\hat{\boldsymbol{\Theta}}_t$). Since $\hat{\boldsymbol{w}}_0 = 0$, the value of parameters before and after update remain the same i.e. $\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0) = \boldsymbol{\Theta}_t$. Therefore:

$$\hat{w}_{opt}^{(k)} = \underbrace{\nabla_{\boldsymbol{\Theta}_t}\mathcal{L}(\boldsymbol{\Theta}_t)}_{\boldsymbol{g}^{val}} \cdot \nabla_{\hat{w}_0^{(k)}}\hat{\boldsymbol{\Theta}}(\hat{\boldsymbol{w}}_0)\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}}$$

where the second term is equal to:

$$\begin{aligned} &\nabla_{\hat{w}_0^{(k)}}[\boldsymbol{\Theta}_t + \eta \sum_{m=0}^{K} \hat{w}_0^{(m)} \sum_{i=0}^{|\boldsymbol{b}^{m}|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_i^{(m)}|\boldsymbol{x}_i^{(m)})]\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}} \\ &= \eta \nabla_{\hat{w}_0^{(k)}} \sum_{m=0}^{K} \hat{w}_0^{(m)} \sum_{i=0}^{|\boldsymbol{b}^{m}|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_i^{(m)}|\boldsymbol{x}_i^{(m)})\Big|_{\hat{\boldsymbol{w}}_0=\mathbf{0}} \\ &= \eta \sum_{i=0}^{|\boldsymbol{b}^{k}|-1} \underbrace{\nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_i^{(k)}|\boldsymbol{x}_i^{(k)})}_{\boldsymbol{g}^{(k)}} \end{aligned}$$

Therefore, the weight for the task is as follows:

$$\begin{aligned}\hat{w}_{opt}^{(k)} &= \eta(\boldsymbol{g}^{val} \cdot \boldsymbol{g}^{(k)}) \\ &= \eta \sum_{j=0}^{|\boldsymbol{b}^{val}|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_j^{val}|\boldsymbol{x}_j^{val}) \cdot \sum_{i=0}^{|\boldsymbol{b}^{k}|-1} \nabla \log P_{\boldsymbol{\Theta}_t}(\boldsymbol{y}_i^{(k)}|\boldsymbol{x}_i^{(k)}).\end{aligned} \tag{7.6}$$

Unlike eqn. 7.5, eqn. 7.6 does not require backpropagation of backpropagation and deep learning frameworks e.g. PyTorch can compute it more efficiently.

7.4 Learning to Multi-Task Learn

In the previous section, we present an effective while computationally expensive oracle policy to solve the MTL-MDP. In this section, we aim to scale up the decision making by *learning* an efficient *policy* to properly schedule the training process of the MTL architecture, i.e. learning to multi-task learn. Our MTL-MDP formulation paves the way to make use of a plethora of algorithms in Imitation Learning (IL) and Reinforcement Learning (RL) to learn the policy. It has been shown that we can expect potentially *exponentially* lower sample complexity in learning a task with IL than with RL algorithms (Sun et al., 2017). Therefore, in this chapter, we are going to explore the use of DAGGER (Ross et al., 2011), a simple and effective algorithm for learning the policy within the IL framework. In what follows, we first describe the architecture of our policy network and then mention its training using DAGGER .

Policy/Scheduler Network We adopt a two-layer dense feed-forward neural network, followed by a Softmax layer, as the policy network (see Figure 7.2). This is motivated by preliminary experiments on different architectures, which show that this architecture is effective for the MTL scenarios in this chapter.

The inputs to the network include the footprint of history of the training process until the current time step $\boldsymbol{s}_t$ as well as the those for the provided mini-batches as the possible actions $\boldsymbol{b}_t$. As the computational efficiency is critical when using the schedule network for large-scale MTL scenario, we opt to use light features for summarising the history of the training trajectory since the beginning. For each task k, we compute the moving average of its loss values l_{ma}^k over those time steps where a mini-batch from this task was selected for updating the MTL parameters. These features only depend on the mini-batches used in the history of the training process, and do not depend on the provided mini-batches at the current time step $\boldsymbol{b}_t$.

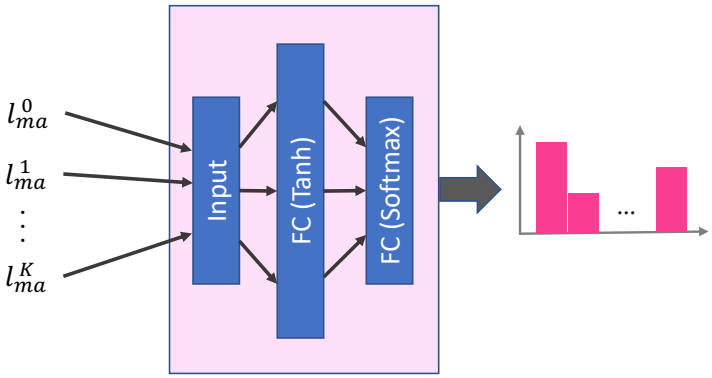


Figure 7.2: The policy/scheduler network.

These features are updated in an online manner during the MTL training process. More specifically, let us assume that a mini-batch from a task k_t is selected at the time step t to re-train the MTL architecture. After updating the parameters of the underlying MTL architecture, we update the moving average loss *only* for this task as follows:

$$l_{ma}^{k_t} \leftarrow (1-\gamma) l_{ma}^{k_t} + \gamma \text{loss}(\boldsymbol{\Theta}_t, \boldsymbol{b}_t^{k_t})$$

where $\gamma \in [0, 1]$. Importantly, the loss value is already computed when updating the MTL parameters, so this feature does not impose additional burden on the computational graph. These features are inspired by those used in MentorNet (Jiang et al., 2018), and our investigations with the oracle policy of Section 7.3 on finding informative features predictive of the selected tasks.

Learning the Policy with IL. Inspired by the Dataset Aggregation (DAGGER) algorithm (Ross et al., 2011), we learn the scheduler network jointly with the MTL architecture, in the course of a single training run. Algorithm 2 depicts the high-level procedure of the training and making use of the scheduler network.

At each update iteration, we decide between using or training the scheduler network with the probability of β[1] (line 9). In case of training the scheduler network (lines 12-14), we use the oracle policy π^{oracle} in Section 7.3 to generate the optimal weights. This creates a new training instance for the policy network, where the input is the current state and the output is the optimal weights. We add this new training instance to the memory replay buffer M, which is then used to re-train the policy/scheduler network. In case of making use of the scheduler network (line 10), we simply feed the state to the network and receive the predicted weights.

[1]For efficiency, we *use* the scheduler network at least 90% of times.

Algorithm 2 Learning the scheduler and MTL model

1: **Input**: Train sets for the tasks $\mathcal{D}^0..\mathcal{D}^K$, validation of the main task $\mathcal{D}^{val}$, scheduler usage ratio β
2: Init $\boldsymbol{\Theta}_0$ randomly ▷ MTL architecture parameters
3: Init $\boldsymbol{\phi}$ randomly ▷ scheduler network parameters
4: $M \leftarrow \{\}$ ▷ memory-replay buffer
5: $l^k_{ma} \leftarrow 0 \quad \forall k \in \{0, .., K\}$
6: $t \leftarrow 0$
7: **while** the stopping condition is not met **do**
8: $\boldsymbol{b}^0_t, .., \boldsymbol{b}^k_t, \boldsymbol{b}^{val} \leftarrow \text{sampleMB}(\mathcal{D}^0, .., \mathcal{D}^K, \mathcal{D}^{val})$
9: **if** $\text{Rand}(0, 1) < \beta$ **then**
▷ Use the scheduler policy
10: $\boldsymbol{w}_t \leftarrow \pi_{\boldsymbol{\phi}}(l^0_{ma}, .., l^K_{ma})$
11: **else**
▷ Train the scheduler policy
12: $\boldsymbol{w}_t \leftarrow \pi^{\text{oracle}}(\boldsymbol{b}^0_t, .., \boldsymbol{b}^K, \boldsymbol{b}^{val}, \boldsymbol{\Theta}_t)$
13: $M \leftarrow M + \{((l^0_{ma}, .., l^K_{ma}), \boldsymbol{w}_t)\}$
14: $\boldsymbol{\phi} \leftarrow \text{retrainScheduler}(\boldsymbol{\phi}, M)$
15: **end if**
16: $k_t \leftarrow \text{sampleTask}(\boldsymbol{w}_t)$
17: $\boldsymbol{\Theta}_{t+1}, \text{loss} \leftarrow \text{retrainModel}(\boldsymbol{\Theta}_t, \boldsymbol{b}^{k_t}_t)$
18: $l^{k_t}_{ma} \leftarrow (1 - \gamma) l^{k_t}_{ma} + \gamma \text{loss}$
19: $t \leftarrow t + 1$
20: **end while**

After getting the tasks' importance weights, the algorithm samples a task (according to the distribution $\boldsymbol{w}_t$) to re-train the MTL architecture and update the moving average of the selected task (lines 16-18).

Re-training the Scheduler Network To train our policy/scheduler network, we optimise the parameters such that the action prescribed by the network matches that which was prescribed by the oracle. More specifically, let $M := \{(\boldsymbol{s}_i, \boldsymbol{a}_i)\}_{i=1}^I$ be the collected states paired with their optimal actions. The state $\boldsymbol{s}_i$ comprises of moving averages for the tasks $(l^{i,0}_{ma}, .., l^{i,K}_{ma})$, and its paired action is the optimal tasks' importance weights $\boldsymbol{w}_i$. The training objective is $\min_{\boldsymbol{\phi}} \sum_{i=1}^I \text{loss}(\boldsymbol{a}_i, \pi_{\boldsymbol{\phi}}(\boldsymbol{s}_i))$, where we explore ℓ_1-norm of the difference of the two probability distributions as the loss function in the experiments of Section 7.5. To update the network parameters $\boldsymbol{\phi}$, we select a random mini-batch from the memory replay M, and make one SGD step based on the gradient of the training objective.

7.5 Experiments

We analyse the effectiveness of our scheduling method on MTL models learned on languages with different underlying linguistic structures, and under different data availability regimes. We have used the low-resource setting bilingual corpora and auxiliary tasks discussed in Sections 6.3. Further, we have experiments with WMT17 English to German translation task with ∼4.6M training pairs to show the scalability of the proposed approach on a high-resource scenario. We use "newstest2013" as the Dev set for early stopping, "newstest2017" as the Val (meta-validation) set for training the policy/scheduler network, and "newstest2014" for testing.

7.5.1 MTL architectures

As baselines, we have used the *hand-engineered* training schedules discussed in Section 6.3.2. We have implemented our method and baselines using PyTorch (Paszke et al., 2017) on top of OpenNMT (Klein et al., 2017). For the scheduler network, the number of hidden dimensions and the decay factor (γ) are set to 200 and 0.7, respectively. For its training, we use L1 loss function, and Adam optimiser with learning of 0.0001. For low-resource language pairs, we use LSTM setting as discussed in previous chapter while for the high resource scenario we have used Transformer. The β (ratio of using over training the scheduler) is set to 0.99/0.9 for Transformer/LSTM settings.

Transformer setting. For this setting, we use a 6-layer Transformer architecture. The embedding size, batch size and number of heads are set to 512, 2048 tokens and 8, respectively. For the optimisation, we use Adam optimiser (Kingma and Ba, 2014) with the initial learning rate of 2 with noam as its decay method, and gradients are computed based on single mini-batches. For sharing, we have shared 3 top/bottom layers of encoders/decoders while the vocabulary is not shared among tasks.[2] We train all of the models for 3 days on a single NVIDIA V100 GPU[3], and save the best model based on the perplexity on the Dev set.

Mixture of hand-engineered and the proposed *automatic* training schedules. As hand-engineered MTL training schedules for NMT do not distinguishes among auxiliary tasks, we create a new schedule by combining a hand-engineered schedule along with the proposed policy/scheduler network to see the effectiveness of

[2]In the preliminary experiments, we have found that sharing vocabulary is not beneficial for Transformer.

[3]Equivalent to ∼40 epochs of training for the MT-only model.

their combination. In this schedule, the probability of selection of the main task is determined by a hand-engineered schedule (Biased for low-resource setting and Exponential for high-resource setting). However, instead of uniformly distributing the remaining probability among the auxiliary tasks, we use the scheduler network to assign a probability to *each* of the auxiliary tasks based on their contribution to the generalisation of the MTL model.

7.5.2 Results

Table 7.1 reports the results for our proposed method and the baselines for the bilingually low-resource conditions, i.e. translation from English into Vietnamese, Turkish, Spanish and Farsi. As seen, the NMT models trained with our scheduler network perform the best across different language pairs. More specifically, the three MTL training heuristics are effective in producing models which outperform the MT-only baseline. Among the three heuristics, the Biased training strategy is more effective than Uniform and Exponential, and leads to trained models with substantially better translation quality than others. Although our policy learning is agnostic to this MTL setup, it has *automatically* learned effective training strategies, leading to further improvements compared to Uniform as the best heuristic training strategy. We further considered learning a training strategy which is a combination of the best heuristic (i.e., Biased) and the scheduler network, as described before. As seen, this combined policy is not as effective as the pure scheduler network, although it is still better than the best heuristic training strategy.

Table 7.2 shows the results for English to German translation task, as the high resource data condition. Among the three heuristics, Exponential is the most effective training strategy compared to Biased and Uniform. Our *automatically* learned scheduler network leads to an NMT model which is competitive with the model trained with the best heuristic. Furthermore, learning a combined policy resulted from the scheduler network and the Exponential heuristic leads to the most effective training strategy, where the trained NMT model outperforms all the other models wrt the three translation quality metrics.

In the following section, we will elaborate more on the reasons behind the success of the scheduler network by shedding light on how the scheduler controls the contribution of *each* task throughout training.

	BLEU ↑	METEOR ↑	TER ↓
MT only	24.32	43.5	58.1
Hand-engineered			
+ Uniform	24.04	43.0	58.2
+ Biased (Constant)†‡	23.37	42.8	59.0
+ Exponential‡	25.06	44.0	57.1
Scheduler network			
+ **SN**	24.6	43.5	57.4
+ Exponential + **SN**	**25.3**	**44.2**	**56.6**

Table 7.1: BLEU, METEOR and TER scores for English-German language pair. "+ SN" indicates Scheduler Network is used in training. †: Proposed in Chapter 4, ‡: Proposed in (Kiperwasser and Ballesteros, 2018).

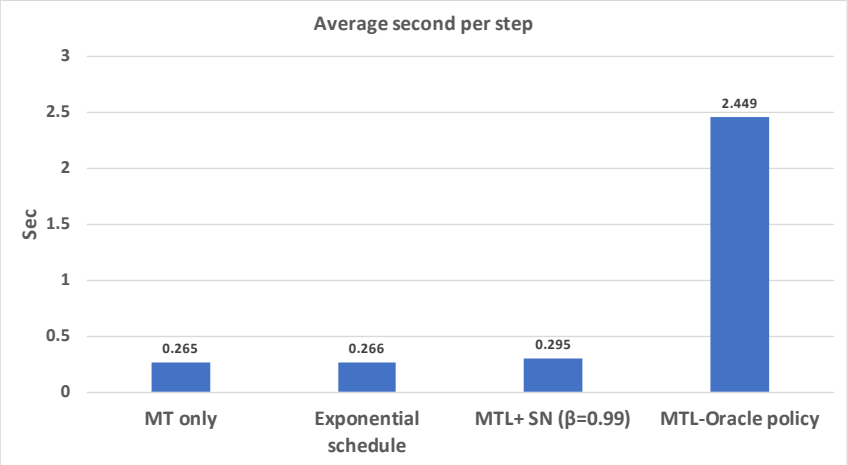


Figure 7.3: Average second per step for different MTL model on the Transformer setting (En→De). We achieve ∼8.3X speed up in the training of MTL by simultaneously training and using the scheduler network.

7.6 Analysis

Scalability of training with the scheduler network As discussed in Section 7.4, we introduce the scheduler to make the oracle policy scalable. To analyse the speed up, we calculated the average time of each step in the high-resource regime with Transformer setting. As mentioned, for the Transformer setting, we train all the models for a fixed time of 3 days (this time includes measuring perplexity and save the model after each epoch). We divide the total number of training steps by this time and depict the result in Figure 7.3. As seen, the oracle policy is computationally expensive, and interestingly using the scheduler network lead to ∼8.3X speed up in the MTL training. Although we both train and use the scheduler network in a single

	English→Vietnamese		English→Turkish		English→Spanish				English→Farsi	
	BLEU		BLEU		BLEU		METEOR		BLEU	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test	Dev	Test
MT only	22.83	24.15	8.55	8.5	14.49	13.44	31.3	31.1	12.16	11.95
MTL with Fixed Schedule										
+ Uniform	23.10	24.81	9.14	8.94	12.81	12.12	29.6	29.5	13.51	13.22
+ Biased (Constant)[†‡]	23.42	25.22	10.06	9.53	15.14	14.11	31.8	31.3	13.53	13.47
+ Exponential[‡]	23.45	25.65	9.62	9.12	12.25	11.62	28.0	28.1	14.23	13.99
+ Sigmoid[‡]	23.35	25.36	9.55	9.01	11.55	11.34	26.6	26.9	14.05	13.88
MTL with Adaptive Schedule (Chapter 6)										
+ Biased + **AIW**	23.95[◇]	25.75	10.67[◇]	10.25[◇]	11.23	10.66	27.5	27.4	14.41	14.2
+ Uniform + **AIW**	**24.38**[◇]	**26.68**[◇]	**11.03**[◇]	**10.81**[◇]	**16.05**[◇]	**14.95**[◇]	**33.0**[◇]	**32.5**[◇]	**15.24**[◇]	**15.08**[◇]
MTL with Scheduler Network										
+ **SN** + Biased	23.86	25.70	10.53[◇]	10.18[◇]	13.20	12.38	29.9	29.7	14.37	14.51[◇]
+ **SN** + Uniform	**24.21**[◇]	**26.45**[◇]	**10.92**[◇]	**10.62**[◇]	**16.14**[◇]	**15.12**[◇]	**33.1**[◇]	**32.7**[◇]	**15.14**[◇]	**14.95**[◇]

Table 7.2: Results for three language pairs. "+ SN" indicates Scheduler Network is used in training. [†]: Proposed in Chapter 4, [‡]: Proposed in (Kiperwasser and Ballesteros, 2018). [◇]: Statistically significantly better ($p < 0.05$) than the best MTL with fixed schedule.

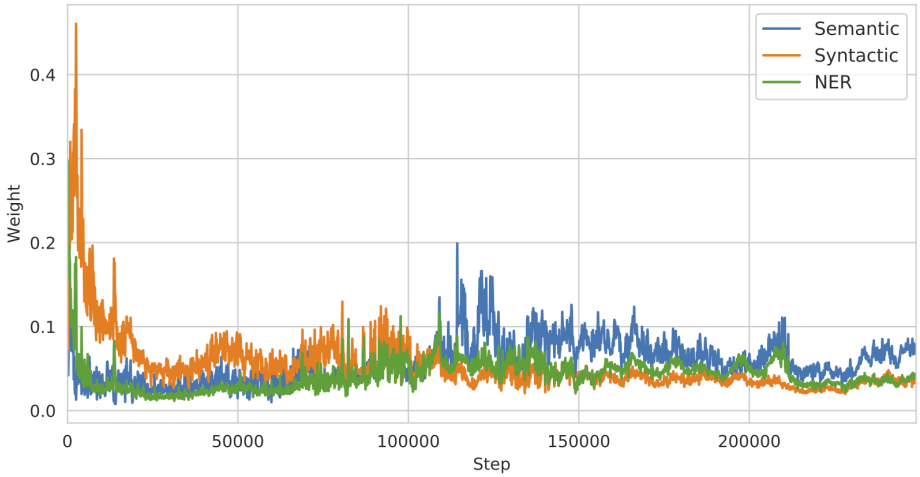


Figure 7.4: Average weights of auxiliary tasks during the training on the English to German language pair. Weights are averaged over 100-steps chunks.

run, thanks to using Imitation Learning, the procedure is so efficient that makes it comparable to hand-engineered heuristics. From the other side, it is not feasible (at least for us) to train an MTL model for the high-resource scenario by only using the oracle policy as we need to train the model for 25 days (instead of 3) to process the same number of steps.

How does the policy/scheduler network regulate the participation of each auxiliary task? In this analysis, we want to shed light on the behaviour of the scheduler network in a large-scale setting. We divide the training on the English to German language pair into 100-step chunks; then inside each chunk, we calculate the average of importance weights for each of the tasks. Figure 7.4 shows the weights of auxiliary tasks. We removed the weights of the main task in order to scale the plot for better visualisation, however it is easy to infer them as the sum of weights is 1. As expected, the scheduler network determines much higher weights for the main task. Similar to the low-resource setting analysis (Section 6.3.3), there is an interesting pattern in the weights of auxiliary tasks. In the beginning, the scheduler network dedicated more emphasis on syntactic-related tasks. Gradually, it decreases their participation and increases the involvement of the semantic-related tasks. We speculate the reason lies behind the requirement for lower-level linguistic knowledge at the beginning (e.g. syntax related knowledge) to find better re-orderings. Then, higher-level linguistic knowledge (i.e. semantic) is required for fully conveying the underlying meaning in the generated translations.

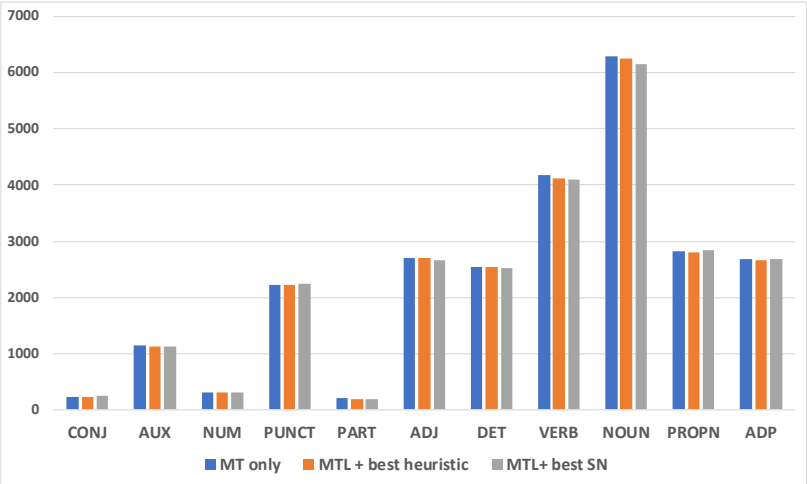


Figure 7.5: The number of missed words in the generated translations of English-to-German language pair. Words are categorised based on their POS tags.

How using the policy/scheduler network affects the generated translations in the high-resource setting? In this analysis, we want to compare the generated translations of different models on the English to German translation task based on the following measure: the number of missed words in the generated translation categorised by their Part-of-Speech. First, we apply Stanford POS tagger (Toutanova et al., 2003) on gold German translations. Then, for each of the generated translations, we look at the missing words in the generated translation and categorise them based on their POS tags. Finally, we count the number of missed words for each of the categories. Result is shown in Figure 7.5. As seen, compared to the MT only baseline, MTL models missed fewer correct words in their translation which is resulted by benefiting from the learned linguistic knowledge. Moreover, the MTL model enhanced with the scheduler network is even able to make better use of the auxiliary tasks and performs better than the other models specifically in noun, verb and adjective categories. We speculate the reason is that the proposed scheduler network is able to dynamically distinguish among auxiliary tasks throughout the training and make better use of them.

7.7 Summary

We introduce a novel approach for automatically learning effective training schedules for MTL. We formulate MTL training as a Markov Decision Process, paving the way to treat the training scheduler as the policy. We then introduce an effective *oracle policy* and use it to train a policy/scheduler network within the Imitation

Learning framework using DAGGER in an on-policy manner. Our results on low-resource (English to Vietnamese/Turkish/Spanish/Farsi) and high-resource (English to German) settings using LSTM and Transformer architectures show up to +1.1 BLEU score improvement compared to the strong hand-engineered heuristics.

Chapter 8

Conclusion and Future Directions

This thesis presents a comprehensive study on using auxiliary linguistic data for improving Neural Machine Translation (NMT) in bilingually low-resource scenarios. We have explored two main approaches for this purpose: (1) using automatically generated annotations; (2) directly injecting linguistic knowledge. For the first case, we have shown that considering the errors and uncertainties of annotations could help to improve the translation quality. For the second case, we have used Multi-Task Learning (MTL) to inject linguistic inductive biases into the translation model. We have shown that both the architecture and training schedule of MTL should be carefully crafted, as they play crucial roles in making the best use of linguistic tasks for improving the underlying translation task.

These thesis contributions can be categorised in three main directions: (1) handling the uncertainty of automatically generated annotations in the translation process (Part I); (2) Effective MTL architectural design for injecting both semantic and syntactic knowledge into the underlying translation model (Part II); (3) Effective strategies to train an MTL model to maximally improve the translation task (Part III).

In Chapter 3, we looked at the case of using pre-trained parsers for generating the syntactic constituency trees of source sentences. We argued that considering only the top-1 tree may lead to an inability to capture parser uncertainties as well as semantic ambiguities in the sentences. Then, we proposed a novel architecture for the transduction of a collection of trees, aka a forest, to a sequence. Our proposed forest-to-sequence considers combinatorially many parse trees of the source sentence along with their probabilities in an efficient bottom-up fashion. Interestingly, the analysis of computational complexity showed that our method processes a forest with combinatorially many trees with merely a small linear time overhead. The experimental

results and analyses backed our hypothesis and demonstrated the effectiveness and efficiency of the proposed method in handling parsing errors and uncertainties.

Chapter 4 was our starting point for injecting linguistic inductive biases into the translation task using MTL. We scaffolded the machine translation task on auxiliary linguistic tasks, including named-entity recognition, syntactic parsing and for the first time[1] semantic parsing. It was done by casting auxiliary tasks to sequence-to-sequence transduction tasks and proposing a *partial* sharing strategy for SEQ2SEQ MTL models. We have further improved the sharing strategy by incorporating *adversarial* training to ensure the shared parameters are not dominated by the representations of a minority of tasks (a potential cause for task interference). The experiments, comparisons and analyses on four language pairs led to the interesting findings. First, the effect of different linguistic tasks varies from a language pair to another one. Second, partial sharing is a better strategy than full sharing for making better use of auxiliary tasks. It also can be enhanced by the proposed adversarial training approach. Third, the best partial sharing practice can be different from one language pair to another one. Therefore, there is a need for tuning the amount of sharing.

Inspired by the findings mentioned above, Chapter 5 proposed a novel approach for adaptive knowledge sharing in deep SEQ2SEQ MTL. We extended the conventional recurrent units to keep multiple flows of information, controlled by multiple *blocks*. The amount of sharing among tasks is *adaptively* tuned via a *routing network* responsible for modulating the input and output of each block conditioning on the task at hand, the input, and model state. Empirical results showed the effectiveness of the proposed approach for different language pairs. In addition, the analysis unveiled that each block is mostly used by a subset of tasks, leveraging the commonalities among subgroups of tasks.

In Chapter 6, we shifted our focus towards the beating heart of the MTL, the training schedule. The training schedule is highly dependent on the aim of the MTL, which could relatively improve *all* the tasks or only one of them. While in the literature, the term "MTL" is used to refer to both of these scenarios, we make the comparison more clear and categorised them as General-MTL and Biased-MTL, to the best of our knowledge it was for the first time. This thesis was focused on the latter as our goal was to improve the translation task. We proposed an approach for *adaptively* and *dynamically* setting the importance of tasks throughout the training in Biased-MTL. We showed that the proposed method is able to *automatically* find

[1]To the best of our knowledge.

interesting schedules that shed light on dynamics of needs throughout the training of bilingually low-resource NMT: from syntax to semantics.

Chapter 7 was built upon the findings of its preceding chapter and proposed a novel framework for *learning* to multi-task learn. We formulated the training schedule of MTL as a Markov Decision Process (MDP), enabling to treat the training scheduler as the policy. We solved the MDP by proposing an *oracle policy* inspired by the approach proposed in Chapter 6, with several techniques to scale it up. Further, we introduced a scheduler policy trained and used simultaneously using Imitation Learning to mimic the oracle policy. The interesting point regarding the proposed approach is its ability to jointly train and use the scheduler along with the MTL model in the course of a *single* run. Moreover, thanks to the ∼8X speedup, we showed that the proposed approach is highly efficient and can even be used in high-resource scenarios without the need for extra training time in comparison to hand-crafted training schedules. Finally, we showed that the interesting scheduling pattern of "from syntax to semantics" is also valid in the high-resource cases.

8.1 Future Directions

There are various potential extensions to the findings of this thesis. Here we briefly discuss some of them:

- Many of the findings and proposed approaches in the MTL-related parts of this thesis are general and can be applied to other areas within NLP or other domains like computer vision. More specifically, we are keen to see the application of the adaptive training schedules on multi-task image classification. We expect it will unveil interesting scheduling patterns similar to the "from syntax to semantic" that we have seen in this thesis.

- In this thesis, we explored the incorporation of the linguistic knowledge for source languages. A natural direction would be investing the effect of injecting the linguistic knowledge related to target languages, or both source and target languages.

- An optimisation algorithm is the companion of the training schedule. In this thesis, we have introduced an adaptive training schedule for Biased-MTL. An important extension of our work is to devise an exclusive optimiser for Biased-MTL. Popular optimisation algorithms like Adam are based on *adaptive learning*

rates. Recently, it has been shown that the large variance of the adaptive learning rate could be problematic (Liu et al., 2019a). Therefore, applying variance reduction techniques is a natural direction as switching the tasks during the training of MTL may lead to a higher variance of learning rates.

Returning to the bigger picture, as mentioned, deep learning tsunami is mainly driven by the abundance of big data as well as computational power. The shortage of labelled data in some areas make it difficult for deep learning to fully unleash its capabilities. Therefore, this thesis is a step towards deep learning with less labelled data for the target task of interest. This is achieved by employing labelled data from related tasks with the aim to lap the waves of deep learning to the less potent areas.

Bibliography

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 132–140.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Roee Aharoni, Melvin Johnson, and Wolfgang Macherey. 2019. The missing ingredient in zero-shot neural machine translation. *arXiv preprint arXiv:1903.07091*.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2017. Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462.

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2018. Unsupervised neural machine translation. In *International Conference on Learning Representations*.

Isabelle Augenstein and Anders Søgaard. 2017. Multi-task learning of keyphrase boundary classification. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 341–346.

Philip Bachman, Alessandro Sordoni, and Adam Trischler. 2017. Learning algorithms for active learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 301–310.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations*.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of*

the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization, pages 65–72, Ann Arbor, Michigan.

Joost Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. 2017. Graph convolutional encoders for syntax-aware neural machine translation. *arXiv preprint arXiv:1704.04675*.

Daniel Beck, Gholamreza Haffari, and Trevor Cohn. 2018. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 273–283, Melbourne, Australia.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017. What do neural machine translation models learn about morphology? In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 861–872.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Yoshua Bengio, Yann LeCun, et al. 2007. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5):1–41.

Ergun Biçici and Deniz Yuret. 2011. Instance selection for machine translation using feature decay algorithms. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 272–283.

Ondrej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, et al. 2016. Findings of the 2016 conference on machine translation. In *ACL 2016 First Conference On Machine Translation (WMT16)*, pages 131–198.

Leo Breiman. 1996. Bagging predictors. *Machine learning*, 24(2):123–140.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451.

Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. 1990. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311.

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Ignacio Cases, Clemens Rosenbaum, Matthew Riemer, Atticus Geiger, Tim Klinger, Alex Tamkin, Olivia Li, Sandhini Agarwal, Joshua D Greene, Dan Jurafsky, et al. 2019. Recursive routing networks: Learning to compose modules for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3631–3648.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017a. Improved neural machine translation with a syntax-aware encoder and decoder. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1936–1945.

Kehai Chen, Rui Wang, Masao Utiyama, Eiichiro Sumita, and Tiejun Zhao. 2018a. Syntax-directed attention for neural machine translation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Xinchi Chen, Zhan Shi, Xipeng Qiu, and Xuanjing Huang. 2017b. Adversarial multi-criteria learning for chinese word segmentation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1193–1203.

Y. Chen and X. Ye. 2011. Projection Onto A Simplex. *ArXiv preprint arXiv:1101.6081*.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018b. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 793–802.

Yong Cheng, Wei Xu, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2016. Semi-supervised learning for neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1965–1974.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014a. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014b. Learning phrase representations using RNN encoder-decoder for statistical machine translation. pages 1724–1734.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton and Co., The Hague.

Kenneth Ward Church and Patrick Hanks. 1989. Word association norms, mutual information, and lexicography. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 76–83, Vancouver, British Columbia, Canada.

Jonathan H Clark, Chris Dyer, Alon Lavie, and Noah A Smith. 2011. Better hypothesis testing for statistical machine translation: Controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 176–181.

Trevor Cohn, Cong Duy Vu Hoang, Ekaterina Vymolova, Kaisheng Yao, Chris Dyer, and Gholamreza Haffari. 2016. Incorporating structural alignment biases into an attentional neural translation model. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537.

R Dennis Cook and Sanford Weisberg. 1980. Characterizations of an empirical influence function for detecting influential cases in regression. *Technometrics*, 22(4):495–508.

Balázs Csanád Csáji. 2001. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48.

Anna Currey, Antonio Valerio Miceli Barone, and Kenneth Heafield. 2017. Copied monolingual data improves low-resource neural machine translation. In *Proceedings of the Second Conference on Machine Translation*, pages 148–156.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, and Stephan Vogel. 2017. Understanding and improving morphological learning in the neural machine translation decoder. In *Proceedings of the International Joint Conference on Natural Language Processing*, pages 142–151.

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407.

Liang Ding and Dacheng Tao. 2019. Recurrent graph syntax encoder for neural machine translation. *arXiv preprint arXiv:1908.06559*.

Carl Doersch and Andrew Zisserman. 2017. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060.

Tobias Domhan and Felix Hieber. 2017. Using target-side monolingual data for neural machine translation through multi-task learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1501–1506.

Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. 2015. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 845–850, Beijing, China.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 199–209.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 823–833.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. *arXiv preprint arXiv:1702.03525*.

Theodoros Evgeniou and Massimiliano Pontil. 2004. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 109–117. ACM.

Meng Fang, Yuan Li, and Trevor Cohn. 2017. Learning how to active learn: A deep reinforcement learning approach. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 595–605.

Weston Feely, Mehdi Manshadi, Robert E Frederking, and Lori S Levin. 2014. The CMU METAL Farsi NLP Approach. In *LREC*, pages 4052–4055.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org.

Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1243–1252. JMLR. org.

Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. 2006. Query by committee made real. In *Advances in neural information processing systems*, pages 443–450.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Jiatao Gu, Hany Hassan, Jacob Devlin, and Victor OK Li. 2018a. Universal neural machine translation for extremely low resource languages. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 344–354.

Jiatao Gu, Yong Wang, Yun Chen, Victor OK Li, and Kyunghyun Cho. 2018b. Meta-learning for low-resource neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3622–3631.

Jiatao Gu, Yong Wang, Kyunghyun Cho, and Victor OK Li. 2019. Improved zero-shot neural machine translation via ignoring spurious correlations. *arXiv preprint arXiv:1906.01181*.

Han Guo, Ramakanth Pasunuru, and Mohit Bansal. 2019. Autosem: Automatic task selection and mixing in multi-task learning. *CoRR*, abs/1904.04153.

Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. 2018. Dynamic task prioritization for multitask learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 270–287.

Thanh-Le Ha, Jan Niehues, and Alexander Waibel. 2016. Toward multilingual neural machine translation with universal encoder and decoder. *arXiv preprint arXiv:1611.04798*.

Gholamreza Haffari, Maxim Roy, and Anoop Sarkar. 2009. Active learning for statistical phrase-based machine translation. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 415–423.

Gholamreza Haffari and Anoop Sarkar. 2009. Active learning for multilingual statistical machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 181–189.

Kazuma Hashimoto and Yoshimasa Tsuruoka. 2017. Neural machine translation with source-side latent graph parsing. *arXiv preprint arXiv:1702.02265*.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2016. A joint many-task model: Growing a neural network for multiple NLP tasks.

Di He, Yingce Xia, Tao Qin, Liwei Wang, Nenghai Yu, Tie-Yan Liu, and Wei-Ying Ma. 2016. Dual learning for machine translation. In *Advances in Neural Information Processing Systems*, pages 820–828.

Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24.

Sepp Hochreiter. 1991. *Untersuchungen zu dynamischen neuronalen Netzen*. Ph.D. thesis, Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Bechnische Universität München.

Sepp Hochreiter and Jiirgen Schmidhuber. 1997a. LTSM can solve hard time lag problems. In *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference*, pages 473–479.

Sepp Hochreiter and Jürgen Schmidhuber. 1997b. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *ACL*, pages 586–594.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, Geoffrey E Hinton, et al. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Ali Jalali, Sujay Sanghavi, Chao Ruan, and Pradeep K Ravikumar. 2010. A dirty model for multi-task learning. In *Advances in neural information processing systems*, pages 964–972.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Jing Jiang. 2009. Multi-task transfer learning for weakly-supervised relation extraction. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1012–1020, Suntec, Singapore.

Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2018. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2309–2318.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Googles multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 5:339–351.

Michael I Jordan and Robert A Jacobs. 1994. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2017. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Durk P Kingma, Tim Salimans, and Max Welling. 2015. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583.

Eliyahu Kiperwasser and Miguel Ballesteros. 2018. Scheduled multi-task learning: From syntax to translation. *Transactions of the Association for Computational Linguistics*, 6:225–240.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics on Interactive Poster and Demonstration Sessions*, pages 177–180.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1885–1894. JMLR. org.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. 2017. Neural amr: Sequence-to-sequence models for parsing and generation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 146–157.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018a. Unsupervised machine translation using monolingual corpora only. In *International Conference on Learning Representations*.

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018b. Phrase-based & neural unsupervised machine translation. *arXiv preprint arXiv:1804.07755*.

Yann LeCun and Yoshua Bengio. 1998. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature*, 521(7553):436–444.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Junhui Li, Deyi Xiong, Zhaopeng Tu, Muhua Zhu, Min Zhang, and Guodong Zhou. 2017. Modeling source syntax for neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 688–697.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2019a. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*.

Ming Liu, Wray Buntine, and Gholamreza Haffari. 2018. Learning to actively learn neural machine translation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 334–344.

Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2017. Adversarial multi-task learning for text classification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, pages 1–10.

Shikun Liu, Andrew J Davison, and Edward Johns. 2019b. Self-supervised generalisation with meta auxiliary learning. *arXiv preprint arXiv:1901.08933*.

Yi Luan, Chris Brockett, Bill Dolan, Jianfeng Gao, and Michel Galley. 2017. Multi-task learning for speaker-role adaptation in neural conversation models. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 605–614.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford neural machine translation systems for spoken language domain. In *International Workshop on Spoken Language Translation*, Da Nang, Vietnam.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015a. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. 2015b. Addressing the rare word problem in neural machine translation.

Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task sequence to sequence learning. In *International Conference on Learning Representations*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015c. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal.

Chunpeng Ma, Akihiro Tamura, Masao Utiyama, Tiejun Zhao, and Eiichiro Sumita. 2018. Forest-based neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1253–1263, Melbourne, Australia.

Christopher D Manning. 2015. Computational linguistics and deep learning. *Computational Linguistics*, 41(4):701–707.

Diego Marcheggiani, Joost Bastings, and Ivan Titov. 2018. Exploiting semantics in neural machine translation with graph convolutional networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 486–492, New Orleans, Louisiana.

Diego Marcheggiani and Ivan Titov. 2017. Encoding sentences with graph convolutional networks for semantic role labeling. *arXiv preprint arXiv:1703.04826*.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE.

Marvin Minsky and Seymour Papert. 1969. Perceptrons - an introduction to computational geometry.

Tom M Mitchell. 1980. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research .

Pim Moeskops, Jelmer M Wolterink, Bas HM van der Velden, Kenneth GA Gilhuijs, Tim Leiner, Max A Viergever, and Ivana Išgum. 2016. Deep learning for multi-task medical image segmentation in multiple modalities. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 478–486. Springer.

Graham Neubig. 2017. Neural machine translation and sequence-to-sequence models: A tutorial. *arXiv preprint arXiv:1703.01619*.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. DyNet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Graham Neubig and Junjie Hu. 2018. Rapid adaptation of neural machine translation to new languages. *arXiv preprint arXiv:1808.04189*.

Weili Nie, Nina Narodytska, and Ankit Patel. 2019. RelGAN: Relational generative adversarial networks for text generation. In *International Conference on Learning Representations*.

Jan Niehues and Eunah Cho. 2017. Exploiting linguistic resources for neural machine translation using multi-task learning. In *Proceedings of the Second Conference on Machine Translation*, pages 80–89.

Xing Niu, Michael Denkowski, and Marine Carpuat. 2018. Bi-directional neural machine translation with synthetic parallel data. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 84–91.

Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318.

Ramakanth Pasunuru and Mohit Bansal. 2017. Multi-task video captioning with video and entailment generation. In *Proceedings of ACL*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Hao Peng, Sam Thomson, and Noah A. Smith. 2017. Deep multitask learning for semantic dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2037–2048.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Adam Poliak, Yonatan Belinkov, James Glass, and Benjamin Van Durme. 2018. On the evaluation of semantic phenomena in neural machine translation using natural language inference. *arXiv preprint arXiv:1804.09779*.

Ye Qi, Devendra Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. 2018. When and why are pre-trained word embeddings useful for neural machine translation? In *Proceedings of the 2018 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pages 529–535, New Orleans, Louisiana.

Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. 2017. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1):121–135.

Roi Reichart, Katrin Tomanek, Udo Hahn, and Ari Rappoport. 2008. Multi-task active learning for linguistic annotations. In *Proceedings of ACL-08: HLT*, pages 861–869, Columbus, Ohio.

Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, pages 4331–4340.

Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. 2018. Routing networks: Adaptive selection of non-linear functions for multi-task learning. In *International Conference on Learning Representations*.

Frank Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*.

Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. 2017. Sluice networks: Learning what to share between loosely related tasks. *arXiv preprint arXiv:1705.08142*.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. Learning internal representations by error propagation. Technical report, DTIC Document.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Edinburgh neural machine translation systems for wmt 16. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 371–376.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725.

Rico Sennrich and Biao Zhang. 2019. Revisiting low-resource neural machine translation: A case study. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 211–221, Florence, Italy.

Burr Settles. 2009. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does string-based neural mt learn source syntax? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534.

Dongdong Zhang Shuangzhi Wu, Ming Zhou. 2017. Improved neural machine translation with source syntax. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4179–4185.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*.

Anders Søgaard and Yoav Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 231–235.

Linfeng Song, Daniel Gildea, Yue Zhang, Zhiguo Wang, and Jinsong Su. 2019. Semantic neural machine translation using AMR. *Transactions of the Association for Computational Linguistics*, 7:19–31.

Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. 2018. A graph-to-sequence model for amr-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1616–1626.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. 2017. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3309–3318. JMLR. org.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural information Processing Systems*, pages 3104–3112.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Jörg Tiedemann. 2009. News from OPUS-A collection of multilingual parallel corpora with tools and interfaces. In *Recent advances in natural language processing*, volume 5, pages 237–248.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Proceedings of the International Conference on Language Resources and Evaluation*, pages 2214–2218.

Simon Tong and Edward Chang. 2001. Support vector machine active learning for image retrieval. In *Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118. ACM.

Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264.

Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 conference of the North American chapter of the association for computational linguistics on human language technology-volume 1*, pages 173–180.

Zhaopeng Tu, Yang Liu, Lifeng Shang, Xiaohua Liu, and Hang Li. 2017. Neural machine translation with reconstruction. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Peter D Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Oriol Vinyals, Ł ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in Neural Information Processing Systems*, pages 2773–2781.

Thuy-Trang Vu, Ming Liu, Dinh Phung, and Gholamreza Haffari. 2019. Learning how to active learn by dreaming. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4091–4101, Florence, Italy.

Sida Wang and Christopher Manning. 2013. Fast dropout training. In *international conference on machine learning*, pages 118–126.

Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. 2019. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11293–11302.

Mark Woodward and Chelsea Finn. 2017. Active one-shot learning. *arXiv preprint arXiv:1702.06559*.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.

Lijun Wu, Yingce Xia, Li Zhao, Fei Tian, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2017. Adversarial neural machine translation. *arXiv preprint arXiv:1704.06933*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yi Yang, Zhigang Ma, Feiping Nie, Xiaojun Chang, and Alexander G Hauptmann. 2015. Multi-class active learning by uncertainty sampling with diversity maximization. *International Journal of Computer Vision*, 113(2):113–127.

Yongxin Yang and Timothy M Hospedales. 2016. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*.

Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. 2018. Improving neural machine translation with conditional sequence generative adversarial nets. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1346–1355.

Xiao-Tong Yuan, Xiaobai Liu, and Shuicheng Yan. 2012. Visual classification with multitask joint sparse representation. *IEEE Transactions on Image Processing*, 21(10):4349–4360.

Jiajun Zhang and Chengqing Zong. 2016. Exploiting source-side monolingual data in neural machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545.

Yu Zhang, Ying Wei, and Qiang Yang. 2018a. Learning to multitask. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5771–5782.

Zhirui Zhang, Shujie Liu, Mu Li, Ming Zhou, and Enhong Chen. 2018b. Joint training for neural machine translation models with monolingual data. In *Thirty-Second AAAI Conference on Artificial Intelligence*.