Rethinking the Design of Sequence-to-Sequence Models for Efficient Machine Translation

Maha Elbayad

THÈSE

Poutr obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité: Mathématiques et Informatique

Arrêté ministériel : 25 Mai 2016

Présentée par

Maha Elbayad

Thèse dirigée par **Laurent Besacier**, Professeur, UGA

et codirigée par **Jakob Verbeek**, INRIA/Facebook AI Research

préparée au sein du **Laboratoire d'informatique de Grenoble**

dans **l'Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Une Alternative aux Modèles Neuronaux Séquence-à-Séquence Pour la Traduction Automatique

Rethinking the Design of Sequence-to-Sequence Models for Efficient Machine Translation

Thèse soutenue publiquement le **22 juin 2020**,

devant le jury composé de :

M. François YVON	
Senior Researcher, LIMSI-CNRS	Président
M. Hermann NEY	
Professor, RWTH Aachen University	Rapporteur
M. Holger SCHWENK	
Research scientist, Facebook AI research	Rapporteur
Mme. Marine CARPUAT	
Associate professor, University of Maryland	Examinatrice
M. Laurent BESACIER	
Professor, Université Grenoble Alpes	Directeur de thèse
M. Jakob VERBEEK	
Resarch scientist, INRIA/Facebook AI research	Co-directeur de thèse

To my parents
Abderrahmane and Milouda

Acknowledgments

First, and foremost, I would like to express my deepest gratitude to my advisors Jakob Verbeek and Laurent Besacier. My PhD contributions would not have been possible without their endless support and understanding allowing me to adjust my research agenda to my own interests. I remain indebted to them for the guidance and encouragement they provided me since day one.

I would like to thank my thesis committee members. I am very honored and humbled by their review of my work. I would also like to thank Michael Auli for his generous care and invaluable mentorship since I interned with him at Facebook AI research.

During my PhD studies, I had the pleasure to interact with a wonderful group of researchers, engineers and fellow PhD students. I would like to thank the GETALP and THOTH teams in particular for the moments we've spent together.

Last, but not least, I would like to thank my family for their unconditional love and support. I thank my brothers and sisters for their faith in me. I dedicate this thesis to my parents to whom I owe more than words could ever express.

Abstract

Rethinking the Design of Sequence-to-Sequence Models
for Efficient Machine Translation

In recent years, deep learning has enabled impressive achievements in Machine Translation. Neural Machine Translation (NMT) relies on training deep neural networks with large number of parameters on vast amounts of parallel data to learn how to translate from one language to another. One crucial factor to the success of NMT is the design of new powerful and efficient architectures. State-of-the-art systems are encoder-decoder models that first encode a source sequence into a set of feature vectors and then decode the target sequence conditioning on the source features. In this thesis we question the encoder-decoder paradigm and advocate for an intertwined encoding of the source and target so that the two sequences interact at increasing levels of abstraction. For this purpose, we introduce Pervasive Attention, a model based on two-dimensional convolutions that jointly encode the source and target sequences with interactions that are pervasive throughout the network. To improve the efficiency of NMT systems, we explore online machine translation where the source is read incrementally and the decoder is fed partial contexts so that the model can alternate between reading and writing. We investigate deterministic agents that guide the read/write alternation through a rigid decoding path, and introduce new dynamic agents to estimate a decoding path for each sample. We also address the resource-efficiency of encoder-decoder models and posit that going deeper in a neural network is not required for all instances. We design depth-adaptive Transformer decoders that allow for anytime prediction and sample-adaptive halting mechanisms to favor low cost predictions for low complexity instances and save deeper predictions for complex scenarios.

Résumé

Une Alternative aux Modèles Neuronaux Séquence-à-Séquence
Pour la Traduction Automatique

L'apprentissage profond a permis des avancées significatives dans le domaine de la traduction automatique. La traduction automatique neuronale (NMT) s'appuie sur l'entrainement de réseaux de neurones avec un grand nombre de paramètres sur une grand quantité de données parallèles pour apprendre à traduire d'une langue à une autre. Un facteur primordial dans le succès des systèmes NMT est la capacité de concevoir des architectures puissantes et efficaces. Les systèmes de pointe sont des modèles encodeur-décodeurs qui, d'abord, encodent une séquence source sous forme de vecteurs de caractéristiques, puis décodent de façon conditionne la séquence cible. Dans cette thèse, nous remettons en question le paradigme encodeur-décodeur et préconisons de conjointement encoder la source et la cible afin que les deux séquences interagissent à des niveaux d'abstraction croissants. À cette fin, nous introduisons Pervasive Attention, un modèle basé sur des convolutions bidimensionnelles qui encodent conjointement les séquences source et cible avec des interactions qui sont omniprésentes dans le réseau neuronal. Pour améliorer l'efficacité des systèmes NMT, nous étudions la traduction automatique simultanée où la source est lue de manière incrémentielle et le décodeur est alimenté en contextes partiels afin que le modèle puisse alterner entre lecture et écriture. Nous améliorons les agents déterministes qui guident l'alternance lecture / écriture à travers un chemin de décodage rigide et introduisons de nouveaux agents dynamiques pour estimer un chemin de décodage adapté au cas-par-cas. Nous abordons également l'efficacité computationnelle des modèles NMT et affirmons qu'ajouter plus de couches à un réseau de neurones n'est pas requis pour tous les cas. Nous concevons des décodeurs Transformer qui peuvent émettre des prédictions à tout moment dotés de mécanismes d'arrêt adaptatifs pour allouer des ressources en fonction de la complexité de l'instance.

Contents

Contents **v**

Notations **ix**

1 Introduction **1**

1.1 Context and Motivation 1

1.2 Contributions 4

1.3 Thesis Outline 5

1.4 Publications 6

1.5 Open-source Implementation 7

2 Background **9**

2.1 Data for Neural Machine Translation 10

2.2 Token Representations 11

2.3 Encoding Sequences 12

2.3.1 Convolutional Encoders 12

2.3.2 Recurrent Encoders 14

2.3.3 Attention-based Encoders 15

2.4 Language Models 16

2.4.1 Recurrent Language Models 17

2.4.2 Convolutional Language Models 18

2.4.3 Attention-based Language Models 18

2.5 Sequence-to-sequence Models 19

2.5.1 Conditioned Auto-regressive Generation 20

2.5.2 Training Sequence-to-Sequence Models 22

2.6 Baseline Architectures . . . 23
2.6.1 Bi-LSTM . . . 24
2.6.2 ConvS2S . . . 24
2.6.3 Transformer . . . 25
2.7 Discussion . . . 27
2.8 Conclusion . . . 31

3 Pervasive Attention **33**
3.1 Introduction . . . 34
3.2 Pervasive Attention Model . . . 35
3.2.1 Input Representation . . . 35
3.2.2 Convolutions . . . 36
3.2.3 Skip Connections . . . 38
3.2.4 Building Layers . . . 41
3.2.5 Aggregation for Sequence Prediction . . . 44
3.2.6 Inference . . . 46
3.2.7 Sequence-to-sequence properties . . . 47
3.3 Experiments . . . 47
3.3.1 Experimental Setup . . . 48
3.3.2 Ablation Results . . . 50
3.3.3 Comparison to State-of-the-art . . . 55
3.4 Analysis and Discussion . . . 56
3.5 Related Work . . . 57
3.6 Conclusion . . . 61

4 Online Neural Machine Translation **63**
4.1 Introduction . . . 64
4.2 A Common Framework for Online Decoding . . . 65
4.3 Sequence-to-Sequence models for Online Decoding . . . 67
4.3.1 Online Transformer . . . 68
4.3.2 Online Pervasive Attention . . . 69
4.4 Online Decoding Agents . . . 72
4.4.1 Deterministic Agents . . . 73

4.4.2 Dynamic Agents . . . 76
4.5 Experiments . . . 78
4.5.1 Experimental Setup . . . 79
4.5.2 Latency metrics . . . 80
4.5.3 Offline Models . . . 82
4.5.4 Online Models with Deterministic Agents . . . 82
4.5.5 Online Models with Dynamic Agents . . . 88
4.5.6 Qualitative analysis . . . 90
4.6 Related Work . . . 95
4.7 Conclusion . . . 97

5 Cost Effective Decoding **99**
5.1 Introduction . . . 100
5.2 Anytime Structured Prediction . . . 101
5.2.1 Transformer with Intermediate Classifiers . . . 102
5.2.2 Aligned Training . . . 105
5.2.3 Mixed Training . . . 105
5.2.4 Balancing the Losses . . . 106
5.3 Adaptive Depth Estimation . . . 107
5.3.1 Sequence-specific Depth . . . 108
5.3.2 Token-specific Depth . . . 111
5.4 Experiments . . . 114
5.4.1 Experimental Setup . . . 114
5.4.2 Training Multiple Output Classifiers . . . 115
5.4.3 Adaptive Depth Estimation . . . 117
5.4.4 Scaling the Adaptive-depth Models . . . 121
5.4.5 Qualitative Results . . . 123
5.5 Related Work . . . 125
5.6 Conclusion . . . 128

6 Conclusion **129**
6.1 Summary . . . 129
6.2 Future work . . . 131

Bibliography **135**

Appendices **159**

A ConvNet's activations and skip-connections **161**

B FLOPs approximation **163**

C Online Neural Machine Translation: Additional Experiments **167**

D Online Neural Machine Translation: Numerical Results **171**

E Loss Smoothing for Language Models **185**

Notations

$\boldsymbol{x}$	A sequence of tokens
$\lvert\boldsymbol{x}\rvert$	The length of the sequence $\boldsymbol{x}$
x_i	The i-th element of $\boldsymbol{x}$
$\boldsymbol{x}_{i:j}$	A slice of $\boldsymbol{x}$ starting from i and ending with j (i and j included)
$\boldsymbol{x}_{\leq i}$	First i tokens of $\boldsymbol{x}$
$\mathcal{A}$	A set
$\mathbb{R}$	The set of real numbers
$\lvert\mathcal{A}\rvert$	Cardinality of the set $\mathcal{A}$
$\mathcal{A} \times \mathcal{B}$	Cartesian product of $\mathcal{A}$ and $\mathcal{B}$
$\mathcal{A}^n$	The n-ary Cartesian power of the set $\mathcal{A}$
$\{0, 1\}$	The set containing 0 and 1
$[1..n]$	The $\{1, 2, \ldots, n\}$ set
$(0, 1)$	The open interval
$[0, 1]$	The closed interval
A	A matrix (more generally a tensor)
A_i	i-th row of the matrix A
A_{ij}	Element i, j of the matrix A
$\mathbb{R}^{m \times n}$	The set of real-valued $m \times n$ matrices
$\mathrm{A}^\top$	Transpose of the matrix A
$\mathrm{A} \odot \mathrm{B}$	Hadamard (element-wise) product of the matrices A and B
$\mathrm{softmax}(\mathrm{A})$	Applying softmax-normalization to the columns of A
a	A vector
$\mathbf{0}$	The null vector
$\mathbf{1}$	The all-ones vector
a_i	i-th element of the vector a
$a^\top$	Transpose of the vector a

$a \odot b$	Hadamard product of the vectors a and b
$\text{softmax}(a)$	Applying softmax-normalization to the vector a
$\begin{bmatrix} a \\ b \end{bmatrix}, [a, b]^\top$	Concatenating the two vectors a and b
$\log(x)$	Natural logarithm of x
$\text{sigmoid}(x)$	Logistic sigmoid, $\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}$
$\lceil x \rceil$	The ceiling of x (the least integer greater than or equal to x)
$\lfloor x \rfloor$	The floor of x (greatest integer less than or equal to x)
p	The probability distribution of a random variable
$x \sim p$	Sampling from the distribution p
$\nabla_\theta x$	Gradient of x w.r.t. θ
$[\![x = y]\!]$	Iverson bracket; 1 if the proposition $x = y$ is true, 0 otherwise
$\mathcal{U}(\mathcal{A})$	The uniform distribution over the set $\mathcal{A}$

Chapter 1

Introduction

1.1 Context and Motivation

There are about 7000 human languages spoken worldwide (Eberhard et al., 2020), being able to communicate flawlessly between all of them is an age-old dream of mankind. In science fiction works, this dream is often materialized with the *universal translator*. Some variants of this device are described as telepathically reading the brain patterns of the speaker to instantly translate any alien language. Other, more plausible, collect monolingual databases to *decipher* new languages.

To realize this dream of seamless translation, the research field of Machine Translation (MT) aims to teach machines how to automatically translate across languages. Although the recent success of Machine translation is founded on modern computing capabilities, the field's foundations date back to the late 1940s. Inspired by the success of cryptography and recent advances in information theory (Shannon, 1948), Warren Weaver in his memorandum (Weaver, 1949) made the famous claim – When I look at an article in Russian, I say, "This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode." – His memorandum was one of the first publications to instigate research in mechanical translation suggesting that methods from cryptography might be useful for translation.

During the first decade of MT research, three basic approaches to MT were discussed: direct, transfer and interlingual (Hutchins, 2007). Direct translation models (Reifler, 1952; Oettinger, 1954; Dostert, 1955) used large bilingual dictionaries with unique target-side

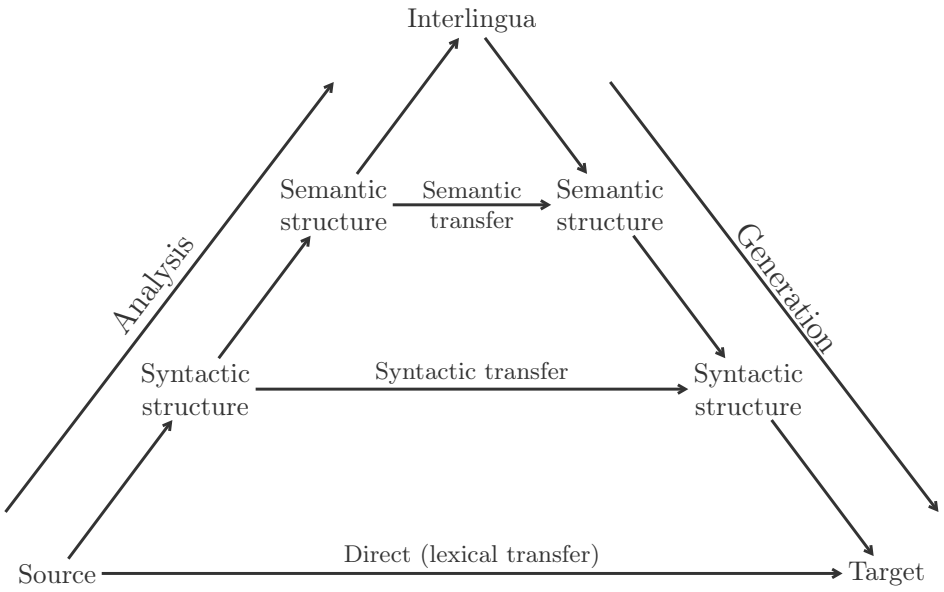


Figure 1.1: Schematization of MT systems with the Vauquois triangle (Vauquois et al., 1965)

equivalents of source words and translation was mostly a one-to-one mapping maintaining the order of the source utterance. However, language must be considered as a system of words or particles influencing one another, to this end, indirect approaches were developed, namely, transfer and interlingual. Transfer MT models (Boitet and Nedobejkine, 1980; Bennett and Slocum, 1985) translate in three stages: analysis, transfer and generation. For transfer a set of linguistic rules is built to cast a wide net for the complexities of language. The other indirect method is interlingual. Interlingual MT models are based on language-neutral representations that abstracts away from the characteristics of the language itself to focus on semantics. The translation process uses an *interlingua* as a pivot by encoding the source into interlingua then decoding the target. These different methodologies can be illustrated with the Vauquois triangle in Figure 1.1 where they only differ in the depth of analysis and abstraction of the language. Although the intuition behind interlingua has its roots in the 1950s (Weaver, 1949), research into interlingua systems was less popular and fell short of the language-neutral ideal.

Up until the late 1980s, MT systems were mostly transfer rule-based. These systems in their attempt to cover all the exceptions of language became cumbersome and exacting to the point of contradiction. With the development of computational resources new statistical *data-driven* approaches for MT emerged. Statistical MT (SMT) systems learn their parameters

from translation frequencies in parallel corpora and model the translation task as a search for the most probable translation of a given utterance. With $\boldsymbol{x}$ the source utterance and $\boldsymbol{y}$ a candidate translation, the goal is to find $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \mid \boldsymbol{x})$.

SMT systems (Brown et al., 1988, 1993) are based on the Bayesian noisy channel model (Shannon, 1948) where Bayes rule is used to transform the problem of maximizing $p(\boldsymbol{y} \mid \boldsymbol{x})$ into maximizing $p(\boldsymbol{x} \mid \boldsymbol{y})p(\boldsymbol{y})$. Therefore, the SMT system maximizes $p(\boldsymbol{x} \mid \boldsymbol{y})$ with a translation model and $p(\boldsymbol{y})$ with a language model. Simplifying assumptions are made to learn each model; the language model is a Markov chain of order n and the translation model assumes the words to be independent from one another.

State-of-the-art SMT systems up to the surge of neural-based MT were phrase-based (Och et al., 1999; Koehn et al., 2003) *i.e.* they alleviate the strong assumption of independence in the translation model and estimate probabilities over phrases. To improve their translation quality, SMT systems used log-linear models with $\log p(\boldsymbol{x} \mid \boldsymbol{y})$ and $\log p(\boldsymbol{y})$ as features. This means optimizing the scaled features in the log-space $\lambda_1 \log p(\boldsymbol{x} \mid \boldsymbol{y}) + \lambda_2 \log p(\boldsymbol{y})$ (Och and Ney, 2002). Log-linear models allowed for the inclusion of other features, all weighted with optimal weights λ_i (Och et al., 2004). However, with many components, optimizing the SMT systems became more difficult. In fact, each component must be optimized separately before combining all of them together.

Neural Machine Translation (NMT) is a relatively new approach that addresses the limits of SMT. The use of neural networks for MT was suggested decades ago (Allen, 1987; Pollack, 1990; Chrisman, 1991; Forcada and Ñeco, 1997; Castano and Casacuberta, 1997), but it was not until the increase of computing power that NMT proved its potential (Kalchbrenner and Blunsom, 2013). Similar to SMT, NMT is also data-driven, however, instead of having disjoint building blocks, NMT has a single map modeled as a neural network responsible for the entire translation process.

SMT and NMT systems can be viewed as an extension of transfer-based MT systems of the 1980s with transfer rules that are not explicitly enumerated but inferred from monolingual and parallel data. Where SMT systems have different components for different types of semantic and syntactic transfer rules, NMT goes a step further and abstracts away from humanly interpretable rules by embedding language in high-dimensional vector spaces that

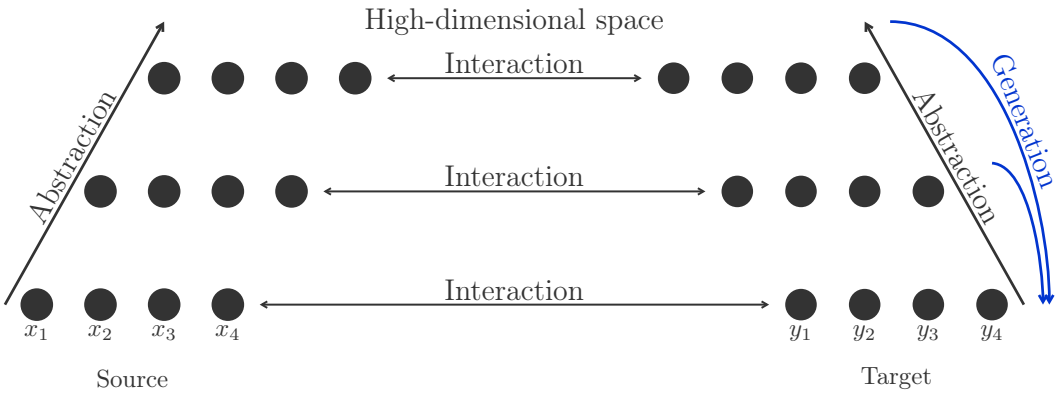


Figure 1.2: A schematization of our view of NMT systems

are potentially language-independent (Johnson et al., 2017).

In state-of-the-art NMT systems (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017) the analysis step of the Vauquois triangle (see Figure 1.1) is modeled with an encoder model and the generation with a decoder model. In this thesis, we aim to re-assess the two-steps encoder-decoder NMT paradigm. Our view of NMT models, illustrated in Figure 1.2, put emphasis on the interaction between the source and target sequences at increasing levels of abstraction where the two sequences are jointly encoded in interwoven computational graphs. We consider the challenging task of online or simultaneous machine translation as a particular case of MT that requires an efficiently designed and controlled interaction between the source and the target sequences. We also endeavor in choosing the right level of abstraction for generation (the blue arrows in Figure 1.2) that will optimize a trade-off between the translation's quality and its computational cost.

1.2 Contributions

This thesis studies the problem of modeling neural machine translation systems and proposes alternative computational graphs to improve:

(a) The translation quality.
(b) The translation delay or lagging in the case of online machine translation.
(c) The translation's computational cost.

(a) To address the first point, we introduce Pervasive Attention (Chapter 3), an NMT model with a computational graph different from existing encoder-decoder models. In Pervasive attention, the source and the target communicate and interact throughout the encoding process towards abstract features. To this end, our NMT model uses two-dimensional convolutional neural networks to process a grid of features where every position represents an interaction between a target and a source tokens.

(b) To improve the translation's delay in online NMT systems (Chapter 4), we first setup a common framework for online sequence-to-sequence models that will allow us to train existing deterministic decoders that alternate between reading the source and writing the target in a pre-determined fashion, and dynamic decoders that condition their decoding path on the current input. We first prove the effectiveness of the deterministic online decoders and their ability to perform well outside the delay range they were optimized for. We then adapt Pervasive Attention models for the task of online translation with both a deterministic and a dynamic decoding strategy.

(c) For the last point, we introduce depth-adaptive NMT systems based on the Transformer model (Chapter 5). We present simple yet effective methods to have MT models capable of generating output tokens at different stages of decoding. and propose a variety of halting mechanisms to determine the amount of computation required by each input. This work was done during an internship at Facebook AI research (Menlo Park, CA) hosted by Michael Auli.

1.3 Thesis Outline

To present our contributions we structure this thesis in six chapters:

Chapter 2: Background. We provide in this chapter an overview of background knowledge on the subject of Machine Translation relevant to the understanding of this thesis.

Chapter 3: Pervasive Attention. We describe in this chapter the building blocks of our Pervasive Attention models. For each block we motivate different design choices, and compare their effect on the translation's quality in an ablation study. We then compare our model to state-of-the-art sequence-to-sequence models on a few MT benchmarks.

Chapter 4: Online Neural Machine Translation. We introduce in this chapter our proposed framework for modeling and training online NMT models. We adapt Transformer and Pervasive Attention models for online decoding and propose dynamic decoding agents devised for Pervasive Attention models. We also study in this chapter the existing deterministic wait-k decoding models to better understand their full potential as well as their limits, and compare our dynamic Pervasive-Attention-based decoding agents to the deterministic variety.

Chapter 5: Cost-effective Decoding. We prepare in this chapter the Transformer model for the setup of anytime prediction where the NMT model is required to emit a prediction at multiple exit points of the decoder. We then describe our novel halting mechanisms to model the amount of computation required by every input and suggest an easy way to supervise these mechanisms.

Chapter 6: Conclusion. The final chapter lists our conclusions where we summarize our results and findings and provide an outlook to future research directions.

1.4 Publications

Several contributions presented in this dissertation relate to the following peer-reviewed articles:

1) Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018a. Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction. In *Proc. of CoNLL*
2) Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020b. Depth-Adaptive Transformer. In *Proc. of ICLR*

The following preprint is discussed:

3) Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020a. Improved Training Techniques for Online Neural Machine Translation

The following publication is related, but is not extensively discussed in this thesis (the publication is covered in Appendix E):

4) Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018b. Token-level and Sequence-level Loss Smoothing for RNN Language Models. In *Proc. of ACL*

1.5 Open-source Implementation

Pervasive Attention models introduced in Chapter 3 of this thesis and Online NMT models described in Chapter 4 are implemented in our open-source code available at: `https://github.com/elbayadm/attn2d`.

Chapter 2

Background

The task of machine translation can be viewed as a learning problem. As defined in Mitchell (1997), "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." Our models are based on neural networks that *experience* large amounts of data in order to learn how to translate by proxy of improving a performance measure. In the following, we will provide background knowledge on (a) the type of data used to train a Neural Machine Translation (NMT) model, (b) the type of models we can use to translate, (c) how these models are trained and (d) how their final performance is evaluated.

Contents

2.1 Data for Neural Machine Translation 10
2.2 Token Representations 11
2.3 Encoding Sequences 12
2.4 Language Models 16
2.5 Sequence-to-sequence Models 19
2.6 Baseline Architectures 23
2.7 Discussion 27
2.8 Conclusion 31

2.1 Data for Neural Machine Translation

NMT systems are usually trained with parallel corpora collected from bilingual texts such as news stories, translated lectures or transcribed talks. A parallel corpus is a set of paired tokenized utterances $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i \in \mathcal{I}}$ where $\mathcal{I}$ is an index set. We will refer to $\boldsymbol{x}$ as the source and to $\boldsymbol{y}$ as the target and drop the superscript (i) when processing a single sample $(\boldsymbol{x}, \boldsymbol{y})$. The paired sequences have possibly different lengths $|\boldsymbol{x}|$ and $|\boldsymbol{y}|$. From this set of sequences two vocabularies (set of occurring tokens) are built for each language. The utterances in the corpus can be tokenized into individual words, characters or segments. When using words as tokens, the vocabularies are very large and are consequently truncated *i.e.* we remove low-frequency tokens and replace them in the sequences with a special token $<unk>$. Translation is, however, an open-vocabulary problem and a fixed word-vocabulary limits the capability of the model to generalize to unseen words. Tokenizing into characters on the other hand solves the issue of out-of-vocabulary words but lead to very long sequences that are hard to model.

Alternatively, Sennrich et al. (2016) segment the corpus into *subwords* with the byte pair encoding compression algorithm (Gage, 1994), BPE in short. Subwords are initialized as characters and are iteratively refined by merging the most frequent pair of subwords into a new one. With this segmentation, we can represent rare and unseen words with significantly smaller vocabularies. This type of tokenization is referred to as BPE and can be learned as two independent encodings, one for each language, or as a joint BPE on the bitexts.

In a given experiment, the data is tokenized and divided into three parts: a training set to train the model on, a development set to tune the model's hyper-parameters and a test set on which the system if evaluated and compared to existing work. We pad an end of sequence marker $</s>$ at the end of each sequence (source and target) and include $</s>$ as a special token in the vocabularies. A beginning of sequence marker $<s>$ is also added to the vocabulary but is not considered part of the sequence (*c.f.* §2.5.1). As such, a pair $(\boldsymbol{x}, \boldsymbol{y})$ is formatted as:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_{|\boldsymbol{x}|-1}, </s>), \quad \boldsymbol{y} = (y_1, y_2, \ldots, y_{|\boldsymbol{y}|-1}, </s>). \tag{2.1}$$

Abstracting away from the particular task of machine translation, the general setup of a learning problem is to learn a mapping of the input to an output label. In Natural Language Processing (NLP), the input is a sequence of tokens from a vocabulary $\mathcal{V}$ and the labels are either atomic, one for each token (part-of-speech tagging, named entity recognition, *etc.*) or global *i.e.* a label for the whole sequence (sentiment classification, entailment, *etc.*) Two major problems arise, the first is the discreteness of the input, ill-suited for statistical classifiers, and the second is the input's variable length.

We will briefly introduce continuous token representations that are well-disposed for learning in §2.2 and then look in §2.3 at sequence models used to contextualize and transform these atomic representations into sequence-level feature vectors. Special attention is given to language models in §2.4, and the whole NMT framework is put in place in §2.5. Three NMT architectures, used throughout this thesis as baselines, are described in §2.6. Finally, we discuss a few key characteristics of sequence-to-sequence models in §2.7

2.2 Token Representations

A token from a vocabulary $\mathcal{V}$ is represented with a *feature vector*. Each dimension of this vector is supposed to represent a feature with a possibly interpretable property. Conventionally, the vocabulary $\mathcal{V}$ is indexed and a token is represented with its index in $\mathcal{V}$ or equivalently with a one-hot vector in $\mathbb{R}^{|\mathcal{V}|}$ in which all entries are zero except the one corresponding to the token's index. These one-hot representations are sparse and will yield a poorly trained statistical classifier. Besides, they do not take into account the distributional hypothesis of language (Harris, 1954) stating that the meaning of a word can be inferred from the contexts in which it is used.

Many algorithms were proposed to learn representations that leverage this hypothesis, some are clustering-based (Brown et al., 1992), others are based on the co-occurrence matrix (Ritter and Kohonen, 1989; Dumais et al., 1988), but, the most commonly used are *distributed* representations also called token embeddings. These embeddings are typically the first layer in a neural language model and are learned jointly with the other parameters (Bengio et al., 2001; Schwenk and Gauvain, 2002; Bengio et al., 2003; Mnih and Hinton, 2007; Collobert

and Weston, 2008). Instead of training a language model (predicting the next token in a sequence), recent works shifted to a Cloze task, where the model learns to fill in a few masked tokens (Mikolov et al., 2013a,b; Devlin et al., 2019) or to detect tokens replaced by incorrect, yet plausible, fakes (Clark et al., 2020).

A set of d-dimensional embeddings form a matrix $\mathrm{E} \in \mathbb{R}^{d \times \mathcal{V}}$ where the i-th column of E is the feature vector associated with the i-th token in $\mathcal{V}$. Through the one-hot vector w of a token, the dense feature vector is retrieved with $\mathrm{E}w$.

2.3 Encoding Sequences

Let $\boldsymbol{x} = (x_1, \ldots, x_{|\boldsymbol{x}|})$ be a sequence of tokens from a vocabulary $\mathcal{V}$ and let E be an embedding matrix of $\mathcal{V}$. The goal of a sequence model is to learn how to combine the token embeddings $(\mathrm{E}x_1, \ldots, \mathrm{E}x_{|\boldsymbol{x}|})$ of $\boldsymbol{x}$ into a fixed-size representation $f(\boldsymbol{x})$.

When encoding sequences of fixed lengths, concatenating the embeddings is a valid choice (Bengio et al., 2003), but if the task requires fixed-size representations for sequences of arbitrary length, a naive approach would ignore the sequence's temporal nature and encode it as an *averaged bag of words*:

$$f(\boldsymbol{x}) = \frac{1}{|\boldsymbol{x}|} \sum_i \mathrm{E}x_i. \tag{2.2}$$

More interestingly, f could be designed to pay attention to the input's structure. In the following we will look at three majors architectures used to encode an input sequence: convolutional, recurrent and attention-based.

2.3.1 Convolutional Encoders

The one-dimensional convolution is an operator with a filter $\omega \in \mathbb{R}^k$, k is called the width of the convolution. Let v be a sequence of length t with elements in $\mathbb{R}$. The output of the convolution is simply the dot product of the filter ω with every k-gram of v. If $v_{i-k':i+k'}$ is

the k-gram centered on v_i with k' tokens on each side ($k' = \lfloor k/2 \rfloor$), the i-th output is

$$(v \star \omega)_i = \omega^\top v_{i-k':i+k'}. \tag{2.3}$$

A *narrow* convolution with $k \leq t$ yields a shorter output $(v \star \omega) \in \mathbb{R}^{t-k+1}$, but the operator can be *widened* with zero-padding *i.e.* adding zeroes in out-of-range positions (Kalchbrenner et al., 2014). The normal convolution, narrow or wide, applies a filter to k-grams *i.e.* k contiguous terms in the sequence, however, the filter can be *dilated* (Yu and Koltun, 2016) to expand its reach. With a dilation factor l, we skip elements from the sequence with a step l. The dilated convolution operator, denoted with $\star_l$, is defined as:

$$(v \star_l \omega)_i = \sum_{j=-k'}^{k'} w_j v_{i-jl}. \tag{2.4}$$

We are rather interested in multi-dimensional sequences, and so the convolution operator is modified to combine the input features with a bank of filters. In this case we will denote with V the input matrix in $\mathbb{R}^{d_{\text{in}} \times t}$:

$$\mathrm{V} = \begin{pmatrix} | & | & | \\ v_1 & \dots & v_t \\ | & | & | \end{pmatrix}. \tag{2.5}$$

The number of output features d_{out} is dictated by the number of filters $\omega \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}} \times k}$. The j-th channel of the output is evaluated as:

$$(\mathrm{V} \star \omega)_j = \sum_{l=1}^{d_{\text{in}}} \mathrm{V}_l \star \omega_{jl}. \tag{2.6}$$

Time-Delay Neural Networks (TDNN). Time-delay networks (Waibel et al., 1989; Hinton, 1990), were adopted as sequence models for natural language tasks in Collobert and Weston (2008). A TDNN layer is a one-dimensional narrow convolution applied to the matrix of embeddings:

$$\mathrm{S} = (\mathrm{E}x_1, \dots, \mathrm{E}x_{|\boldsymbol{x}|}) = \begin{pmatrix} | & | & | \\ \mathrm{E}x_1 & \dots & \mathrm{E}x_{|\boldsymbol{x}|} \\ | & | & | \end{pmatrix} \in \mathbb{R}^{d \times |\boldsymbol{x}|}. \tag{2.7}$$

Typical to convolutional networks (LeCun et al., 1998), these layers are stacked to gradually grow the encoder context. To capture salient segments of $\boldsymbol{x}$ and encode it into a fixed-size representation, Collobert and Weston (2008) added a max-over-time pooling layer after the TDNN. Let $\mathrm{S}^{\mathrm{N}} \in \mathbb{R}^{d \times t}$ be the output of the TDNN after N convolutions, the max-pooled output is evaluated as:

$$\forall k \in [1..d],\ \text{Max-pool}(\mathrm{S}^{\mathrm{N}})_k = \max_{1 \leq i \leq t} \mathrm{S}^{\mathrm{N}}_{ki}. \tag{2.8}$$

Dynamic Convolutional Neural Network (DCNN). With TDNNs, the use of narrow convolutions heavily constrain the minimum length that the model can handle, in fact, the input sequence has to be larger than the filter at every layer. Kalchbrenner et al. (2014) proposed a better control of the sequence length throughout the network with DCNNs. They used wide convolutions and dynamic k-max pooling that selects the k most active positions over time.

2.3.2 Recurrent Encoders

Recurrent neural networks (Rumelhart et al., 1986; Jordan, 1986; Elman, 1990), RNN for short, process the sequence tokens one at a time recursively. In its vanilla form, the RNN encodes the input vectors $(\mathrm{E}x_1, \ldots, \mathrm{E}x_{|\boldsymbol{x}|})$ recursively as follows:

$$s_0 = \mathbf{0}, \quad \forall i \in [1..|\boldsymbol{x}|],\ s_i = \text{nonlin}\left(\mathrm{W}\begin{bmatrix}\mathrm{E}x_i \\ s_{i-1}\end{bmatrix}\right), \tag{2.9}$$

where nonlin is non-linear differentiable activation function.

Due to the recursive nature of Eq. (2.9), the state s_i is an encoding of the prefix $\boldsymbol{x}_{\leq i}$ and the final state encodes the full sequence $\boldsymbol{x}$.

In practice, RNNs with gated units (LSTMs (Hochreiter and Schmidhuber, 1997; Gers et al., 1999) and GRUs (Cho et al., 2014)) replace the vanilla version for their proven ability to capture long-term dependencies in language modeling. Without loss of generality, we describe any and all RNNs with an update function g applied recursively to the feature vectors:

$$s_0 = \mathbf{0}, \quad \forall i \in [1..|\boldsymbol{x}|],\ s_i = g(\mathrm{E}x_i, s_{i-1}). \tag{2.10}$$

In some NLP tasks such as sequence tagging, the labels are assumed to be conditioned on the entire sequence $\boldsymbol{x}$. However, the previously introduced recurrent network is uni-directional *i.e.* the state s_i is encoding only the left context $\boldsymbol{x}_{\leq i}$. Bi-directional recurrent networks (Schuster and Paliwal, 1997) exploit the context on both sides. Two separate recurrent layers scan $\boldsymbol{x}$ forwards and backwards and the output is the concatenation of their states. RNNs can also be stacked (El Hihi and Bengio, 1996) by simply feeding the outputs of one RNN to another. In multiple tasks stacked RNNs outperform single-layer RNNs, similar to the stacking of convolutional layers in the previous section, this allows the model to induce representations at growing levels of abstraction proportional to the layer's depth.

2.3.3 Attention-based Encoders

Self-attention for sentence embedding (Lin et al., 2017) is a mechanism relating tokens within a sequence to each other in order to compute a representation of the full sequence. In this form, the only role of self-attention is to aggregate a sequence of states S computed with another sequence model, *e.g.* a recurrent network, into a fixed-size vector:

$$\text{Attention}(\mathrm{S}) = \mathrm{S} \cdot \text{softmax}(\text{nonlin}(\mathrm{S}^\top \mathrm{W}_1)\mathrm{W}_2) \in \mathbb{R}^{d \times r}, \tag{2.11}$$

with weights $\mathrm{W}_1 \in \mathbb{R}^{d \times d}$ and $\mathrm{W}_2 \in \mathbb{R}^{d \times r}$. The softmax operator (Bridle, 1990), applied column-wise, normalizes the column vectors into probability distributions. For a matrix $\mathrm{X} \in \mathbb{R}^{m \times n}$:

$$\forall (i,j) \in [1..m] \times [1..n], \quad \text{softmax}(\mathrm{X})_{ij} = \frac{e^{x_{ij}}}{\sum_{i'=1}^{m} e^{x_{i'j}}}. \tag{2.12}$$

The parameter r allows for computing multiple representations and possibly stacking the attention modules for more complex features. To encode the sequence in a single vector, r is set to 1.

Rather than using attention on top of recurrent models, Vaswani et al. (2017) introduced a fully attentional model where the input embeddings are updated with an attention mechanism from the get-go. Their approach is based on a *key-value-query* dot-product attention bearing similarity to Memory Networks (Weston et al., 2015; Sukhbaatar et al., 2015; Miller et al., 2016). The memory in the form of *key-value* pairs is a representation of the context *i.e.* the

set of token embeddings to be encoded. Each input at a given position queries the memory and assigns relevance probabilities to its content through a measure of similarity to the keys. The relevance probabilities are then used to *read* the memory by taking a weighted sum of its values.

Formally, with $\mathrm{S} = (\mathrm{E}x_1, \ldots, \mathrm{E}x_{|\boldsymbol{x}|})$ the matrix of input embeddings and $\mathrm{W_Q}, \mathrm{W_K}, \mathrm{W_V}$ maps for the queries, keys and values respectively. The attention output is:

$$\text{Attention}(\mathrm{S}) = (\mathrm{W_V S}) \cdot \text{softmax}\left(\frac{1}{\sqrt{d}}(\mathrm{W_K S})^\top (\mathrm{W_Q S})\right). \tag{2.13}$$

For a fixed-size representation, these new features can simply be pooled or a special token is added to the sequence and its state is used as the sequence-level representation (Devlin et al., 2019).

2.4 Language Models

Let $\boldsymbol{y} = (y_1, \ldots, y_{|\boldsymbol{y}|})$ be a sequence of tokens from a vocabulary $\mathcal{V}$. The role of a language model is to estimate the joint probability $p(\boldsymbol{y})$ *i.e.* to assess the *plausibility* or *fluency* of $\boldsymbol{y}$. To reduce the difficulty of modeling $p(\boldsymbol{y})$, we leverage the order of the sequence and assume that the distribution of a given token y_t is conditioned on the history $\boldsymbol{y}_{<t}$. With this assumption, the joint probability $p(\boldsymbol{y})$ is broken down by the chain rule as follows:

$$p(\boldsymbol{y}) = \prod_{t=1}^{|\boldsymbol{y}|} p(y_t \,|\, \boldsymbol{y}_{<t}). \tag{2.14}$$

We refer to $\boldsymbol{y}_{<t} = (y_1, \ldots, y_{t-1})$ as the context or the history. Each product term $p(y_t \,|\, \boldsymbol{y}_{<t})$ is the model's conditional probability for the t-th y_t token given the history $\boldsymbol{y}_{<t}$. For the first term ($t = 1$), the only available context is the special token $<s>$.

A general dependence on all previous tokens grows exponentially in complexity with respect to the length of the sequence. This problem, referred to as the curse of dimensionality, leads to the consideration of the much stronger $(n-1)$-order Markov assumption.

Models with this assumption, also called n-gram models, condition each token's probability on the history of the most recent $(n-1)$ tokens. In practice, n-gram models table the

conditional probabilities $y_t \mid y_{t-n+1:t-1}$ and explicitly enumerate n-grams occurring in a training corpus. As a result, the chosen n is about 4 to 6 tokens and the models generalize poorly to unseen contexts.

Bengio et al. (2003) fought this curse of dimensionality with distributed representations. More precisely, they associated distributed feature vectors or embeddings with each word in the vocabulary then learned to transform the vectors of a given sequence into a joint probability.

Formally, let $\mathrm{E} \in \mathbb{R}^{d \times |\mathcal{V}|}$ be the matrix of d-dimensional embeddings. A neural language model is tasked with modeling $p(y_t \mid \boldsymbol{y}_{<t})$ as a mapping of the context vectors $(\mathrm{E}y_1, \ldots, \mathrm{E}y_{t-1})$:

$$p(y_t \mid \boldsymbol{y}_{<t}) = \mathrm{softmax}(\mathrm{W}_p f(\mathrm{E}y_1, \ldots, \mathrm{E}y_{t-1})). \tag{2.15}$$

The map W_p guarantees the final output is $|\mathcal{V}|$-dimensional and the softmax operator normalizes it into a probability distribution with:

$$\forall x \in \mathbb{R}^{|\mathcal{V}|},\ \mathrm{softmax}(x)_i = \frac{e^{x_i}}{\sum_{i'=1}^{|\mathcal{V}|} e^{x'_i}}. \tag{2.16}$$

The map f in Bengio et al. (2003) is a simple feed-forward network with, as input, the concatenation of the last $n-1$ tokens of the history:

$$x = [\mathrm{E}y_{t-n+1}, \ldots, \mathrm{E}y_{t-1}]^\top, \quad p(y_t \mid \boldsymbol{y}_{<t}) = \mathrm{softmax}(\mathrm{W}x + b + \mathrm{U} \cdot \tanh(\mathrm{H}x + d)), \tag{2.17}$$

with weights $\mathrm{W}, \mathrm{U}, \mathrm{H}$ and biases b, d. Although this work addresses the curse of dimensionality as it pertains to the handling of unseen n-grams in the training corpus, its capacity is still limited by the design of the feed-forward network where n is specified *ad hoc*. In other words, we can generalize better to unseen n-grams but we cannot use contexts larger than n tokens.

2.4.1 Recurrent Language Models

Mikolov et al. (2010) capture the effect of longer context by introducing more structure to the map f with Recurrent Neural Networks (RNN). As seen in §2.3.2, the RNN states encode the context recursively:

$$h_0 = 0, \quad \forall t \in [1..|\boldsymbol{y}|],\ h_t = \mathrm{nonlin}\left(\mathrm{W}\begin{bmatrix}\mathrm{E}y_t \\ h_{t-1}\end{bmatrix}\right). \tag{2.18}$$

The state h_{t-1} is an encoding of the prefix $(\mathrm{E}y_1, \ldots, \mathrm{E}y_{t-1})$ and we can predict the next token y_t as in Eq. (2.15) with:

$$p(y_t \mid \boldsymbol{y}_{<t}) = \mathrm{softmax}(\mathrm{W}_p h_{t-1}). \tag{2.19}$$

2.4.2 Convolutional Language Models

The first convolutional decoders for language modeling where used in ByteNet (Kalchbrenner et al., 2016b). To prevent leaking signal from future tokens, ByteNet uses masked one-dimensional convolution (Oord et al., 2016b). The convolution at step i in Eq. (2.3) encodes the k-gram centered on y_t. We can stop the leaking future signal by zeroing out the right side of the filter, effectively convolving $k' = \lfloor k/2 \rfloor$ tokens from the history plus the current token $\mathrm{E}y_t$:

$$\forall j > k', \omega_j = 0. \tag{2.20}$$

Similar to recurrent uni-directional architectures, the output of stacked masked convolutions at step $t-1$ encodes only the prefix $\boldsymbol{y}_{<t}$. Models with the particular property of encoding only past tokens are suitable for *auto-regressive* generation (*c.f.* §2.5.1).

ByteNet also uses multiplicative units (Kalchbrenner et al., 2017) to incorporate LSTM-like gates in the layers and wrap convolutional layers with residual connections (He et al., 2016b). Multiple of these layers are then stacked to expand the encoded context. The particularity of ByteNet resides in its encoder-decoder stacking with dynamic folding, more on this in §2.5.

ConvS2S (Gehring et al., 2017) introduces another fully convolutional language model. With this architecture the language model is equipped with an mechanism to attend over the source states. This architecture is detailed in §2.6.2.

2.4.3 Attention-based Language Models

Similar to masked convolution, the attention operator described in Eq. (2.13) can be made auto-regressive by limiting the memory access of each query to its predecessors *i.e.* the past

context. With $\mathrm{H} = (\mathrm{E}y_1, \ldots, \mathrm{E}y_{|\boldsymbol{y}|})$ the matrix of token embeddings, at the j-th position, scores associated with future keys $\{\mathrm{E}y_i \,|\, i > j\}$ will be masked.

$$\mathrm{M}_{ij} = \begin{cases} 0, & i \leq j \\ -\infty, & \text{otherwise,} \end{cases} \tag{2.21}$$

$$\text{Scores} = \frac{1}{\sqrt{d}}(\mathrm{W}_\mathrm{K}\mathrm{H})^\top(\mathrm{W}_\mathrm{Q}\mathrm{H}), \tag{2.22}$$

$$\text{Attention}(\mathrm{H}) = (\mathrm{W}_\mathrm{V}\mathrm{H}) \cdot \text{softmax}\left(\text{Scores} + \mathrm{M}\right). \tag{2.23}$$

Self-attention and its masked version are the building blocks of the state-of-the-art Transformer model (Vaswani et al., 2017). The Transformer model is described in §2.6.3.

2.5 Sequence-to-sequence Models

A sequence-to-sequence model is a conditioned generation framework also called encoder-decoder (Kalchbrenner and Blunsom, 2013; Cho et al., 2014; Sutskever et al., 2014). This model is capable of generating arbitrary length output sequences that are contextually appropriate given a source sequence. Sequence-to-sequence models are widely applied for a range of tasks like machine translation, speech recognition, summarization, question answering and image captioning. The most common pipeline for sequence-to-sequence models is to use an encoder that takes an input sequence and creates a contextualized representation of it. This representation is then passed to a decoder which generates an output sequence.

We have discussed in §2.3 existing models capable of encoding a sequence and in §2.4 language models or decoders capable of generating variable length sequences. In the following, we will look at how these two modules are combined to build a sequence-to-sequence model.

Let us consider a pair of source-target sequences $(\boldsymbol{x}, \boldsymbol{y})$ embedded with matrices E_x and E_y respectively. For conditioned generation, we model the conditional probability $p(\boldsymbol{y} \,|\, \boldsymbol{x})$ instead of $p(\boldsymbol{y})$. Following the same reasoning, this probability is factorized as:

$$p(\boldsymbol{y} \,|\, \boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{y}|} p(y_t \,|\, \boldsymbol{y}_{<t}, \boldsymbol{x}). \tag{2.24}$$

Each term $p(y_t \,|\, \boldsymbol{y}_{<t}, \boldsymbol{x})$ is not only a function of the prefix $\boldsymbol{y}_{<t}$ but also the source $\boldsymbol{x}$. Consequently, the map f introduced in Eq. (2.15) takes an additional input:

$$p(y_t \,|\, \boldsymbol{y}_{<t}, \boldsymbol{x}) = \text{softmax}(\text{W}_p f(\boldsymbol{y}_{<t}, \boldsymbol{x})). \tag{2.25}$$

In the language models previously discussed, we represented the prefix $\boldsymbol{y}_{<t}$ with token embeddings $(\text{E}_y y_1, \ldots, \text{E}_y y_{|\boldsymbol{y}|})$ that are auto-regressively encoded into fixed-size states. To inject $\boldsymbol{x}$ into the input we can encode it into a fixed-size vector c (*e.g.* the last hidden state of an RNN or the pooled features of a ConvNet) and then include c as additional input in every atomic update of the language model (Kalchbrenner and Blunsom, 2013; Cho et al., 2014). Alternatively, with recurrent architectures, this feature-vector can be used to initialize the language model states *i.e.* $h_o = c$ in Eq. (2.18) (Sutskever et al., 2014).

Rather than cramming the full source sequence in a single vector c, Bahdanau et al. (2015), inspired by alignment models in statistical machine translation, interfaced the encoder and decoder with an attention module. At every time-step t, this module combines the source states $(s_1, \ldots, s_{|\boldsymbol{x}|})$ into a context vector c_t to be included in the decoder state update. c_t is simply a linear combination of $(s_1, \ldots, s_{|\boldsymbol{x}|})$ with weights estimated by a feed-forward network f_{att}. Formally:

$$\forall j \in [1..|\boldsymbol{x}|],\ \epsilon_{tj} = f_{\text{att}}(h_{t-1}, s_j) = w^\top \tanh(\text{U}h_{t-1} + \text{V}s_j), \tag{2.26}$$

$$c_t = \sum_{j=1}^{|\boldsymbol{x}|} \alpha_{tj} s_j, \quad \alpha_t = \text{softmax}(\epsilon_t) \tag{2.27}$$

with w, U and V the parameters of f_{att}. In their analysis of attention mechanisms for machine translation, Luong et al. (2015) found that modeling f_{att} as a simple dot-product of the two vectors is a better choice. With this mechanism, the decoder decides which parts of the source sequence to focus on in order to predict y_t. The score α_{tj} could be interpreted as a soft alignment between y_t and x_j (see a visualization of these weights in Figure 2.1).

2.5.1 Conditioned Auto-regressive Generation

The ultimate goal of a sequence-to-sequence model is to generate the most likely target sequence $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \,|\, \boldsymbol{x})$. Unfortunately, unlike in Markov models where we can find

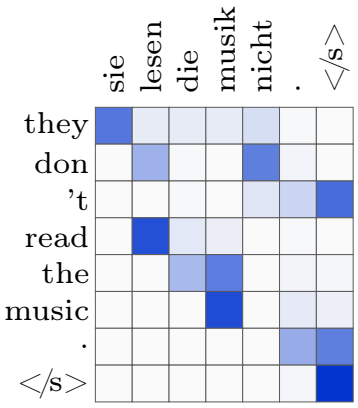


Figure 2.1: Visualisation of the attention weights α_{tj} for a German to English translation.

the most likely sequence with dynamic-programming algorithms (*e.g.* with Viterbi algorithm in $O(|\boldsymbol{y}|\mathcal{V}^2)$ time), with neural language models, there are no algorithms to find the optimal output sequence in subexponential time, instead we resort to approximations.

To generate a sequence $\widetilde{\boldsymbol{y}}$, also called a hypothesis, we first encode the source sequence $\boldsymbol{x}$ and initiate the prefix with $\widetilde{y}_0 = <s>$. From this input we estimate the output distribution $p(\widetilde{y}_1 \,|\, \widetilde{y}_0, \boldsymbol{x})$ and sample the next token from it. The sampled token is embedded and added to the model's input for the next step. We continue sampling tokens until the end of sequence marker $</s>$ is sampled or a length limit is reached. This technique of generation is auto-regressive *i.e.* output at a time-step is conditioned on previous outputs and can only be used for language models trained accordingly.

The straight-forward approach to approximate $\hat{\boldsymbol{y}}$ is *greedy decoding*. We previously decomposed $p(\boldsymbol{y} \,|\, \boldsymbol{x})$ into a product of local scores $p(y_t \,|\, \boldsymbol{y}_{<t}, \boldsymbol{x})$ and so greedy decoding optimizes each term individually by simply taking $\widetilde{y}_t = \arg\max_{y_t} p(y_t \,|\, \widetilde{\boldsymbol{y}}_{<t})$ at every step.

Another option is beam-search (Lowerre and Reddy, 1980). Also used in statistical machine translation (Koehn et al., 2003; Tillmann and Ney, 2003), beam-search finds a better scoring sequence by keeping track of b partial hypotheses. At every decoding step, the successors of the hypotheses are scored and only the best b remain in the beam. The hypotheses are usually scored with their log-likelihood. However, beam-search favors short sequences as a negative log-probability is added with each new token penalizing long hypotheses. To normalize the length in the finalized hypotheses (ones that emitted $</s>$), Wu et al. (2016)

re-score the hypotheses with:

$$\text{score}(\widetilde{\boldsymbol{y}}) = \frac{\log p(\widetilde{\boldsymbol{y}} \,|\, \boldsymbol{x})}{\text{length-norm}(\widetilde{\boldsymbol{y}})}, \quad \text{length-norm}(\widetilde{\boldsymbol{y}}) = \left(\frac{\mu + |\widetilde{\boldsymbol{y}}|}{\mu + 1}\right)^{\alpha}. \tag{2.28}$$

μ is often set to 0 and α, referred to as the *length penalty*, is optimized on a development set. Depending on the choice of α we will either favor shorter sequences if $\alpha < 1$ or longer ones with $\alpha > 1$.

To measure the quality of the generated sequence $\widetilde{\boldsymbol{y}}$ w.r.t. to the gold standard $\boldsymbol{y}$, the metric of choice is often BLEU (Papineni et al., 2002). BLEU measures n-gram precisions, n ranging from 1 to 4, and reports their geometric average. To compute an n-gram precision, we divide the number of correct n-grams by the total number of n-grams in the hypothesis. Evaluating a precision score per sequence leads to noisy scores, and so, the precision is macro-averaged by dividing the sum of correct n-grams by their total number in the overall corpus. The final score is multiplied by a brevity penalty to control the length of the candidates. BLEU is a precision score and so the higher the score the better.

Another evaluation metric is the perplexity over unseen sequences. Perplexity measures how well the model fits a set of samples; a lower perplexity indicates a better fit. For a set of N pairs $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{1 \le i \le \mathrm{N}}$, we feed the source $\boldsymbol{x}^{(i)}$ as well as the ground truth prefix $\boldsymbol{y}^{(i)}_{<t}$ and estimate the likelihood of correctly predicting the next token $y^{(i)}_t$. The perplexity PP is evaluated as:

$$\text{PP} = 2^{-\ell}, \quad \ell = \frac{1}{\sum_i |\boldsymbol{y}^{(i)}|} \sum_i \log p(\boldsymbol{y}^{(i)} \,|\, \boldsymbol{x}^{(i)}). \tag{2.29}$$

2.5.2 Training Sequence-to-Sequence Models

We train an NMT model by estimating its parameters θ with regularized Maximum Likelihood Estimation (MLE). The parameters of a sequence-to-sequence model θ include all of the model's parameters from the encoder to the decoder and everything in-between.

Given a training set of N pairs $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{1 \le i \le \mathrm{N}}$, we solve the following problem:

$$\theta^* = \arg\max_{\theta \in \Theta} \sum_{i=1}^{\mathrm{N}} \log p(\boldsymbol{y}^{(i)} \,|\, \boldsymbol{x}^{(i)}; \theta) + \Omega(\theta), \tag{2.30}$$

where Ω is a regularization function such as the widely used L_2-regularization, also called weight decay, with $\Omega(\theta) = \|\theta\|^2$. Interestingly, all of the modules previously described are differentiable w.r.t. their parameters and so the model can be trained end-to-end with back-propagation (Rumelhart et al., 1986) and gradient-based optimization. In practice, the training data is batched into small sets of samples and θ is optimized with mini-batch stochastic gradient descent. In all of our experiments the reported batch-size is the number of source tokens that the model *sees* before every update, possibly accumulating gradients over multiple backward passes.

For gradient-based optimization, we use Adam (Kingma and Ba, 2015). This algorithm computes adaptive learning rates from estimates of first and second moments of the gradients in order to achieve a better convergence speed.

The base learning rate follows an inverse square-root schedule with a warm-up phase (Vaswani et al., 2017). The learning rate is initialized at lr_0 then linearly increased throughout $\mathrm{T_w}$ steps up to $\mathrm{lr_{max}}$, afterwards, the learning rate is decreased proportionally to the inverse square-root of the update step t:

$$\mathrm{lr}(t) = \min\left(\mathrm{lr}_0 + \frac{\mathrm{lr_{max}} - \mathrm{lr}_0}{\mathrm{T_w}} \cdot t,\ \mathrm{lr_{max}}\sqrt{\frac{\mathrm{T_w}}{t}}\right). \tag{2.31}$$

We use Fairseq (Ott et al., 2019), a toolkit for sequence modeling based on PyTorch (Paszke et al., 2017), to train all of our models.

2.6 Baseline Architectures

In the rest of this thesis we will consider an LSTM-based model with an attention mechanism (Bahdanau et al., 2015) as an archetype of recurrent architectures. For convolutional architectures, we will use the ConvS2S model (Gehring et al., 2017) and for attention-based models, the state-of-the-art Transformer (Vaswani et al., 2017). We will describe in the following the components and particularities of each of these models. The hyper-parameters described below will be specified in the experiments sections of subsequent chapters.

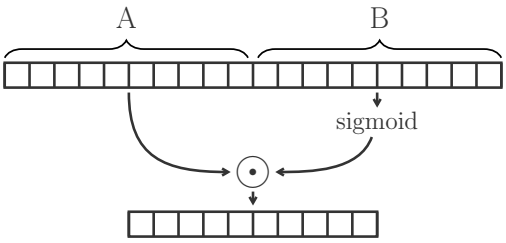


Figure 2.2: Gated linear unit (GLU).

2.6.1 Bi-LSTM

In Bi-LSTM models, the encoder is a bi-directional LSTM and the decoder is a uni-directional LSTM with dot-product attention. The encoder hidden state size is twice the size of the embeddings ($d_{\text{enc}} = 2d_{\text{emb}}$) and the decoder hidden state has the same size as the embedding ($d_{\text{dec}} = d_{\text{emb}}$). For regularization, we apply dropout (Srivastava et al., 2014) to the input embeddings and to the output of every layer with a rate p_{drop}.

2.6.2 ConvS2S

ConvS2S is a fully convolutional encoder-decoder model, the first of its kind to outperform recurrent architectures. On each side, the input is a sum of token and position embeddings. These inputs are processed with a stack of N convolutional layers. The encoder and decoder are interfaced with a dot-product attention. Each of these components is detailed below:

Position embedding. These position embeddings encode the absolute position of a token in the sequence *i.e.* for x_i in $\boldsymbol{x}$ we embed the index i. Similar to token embeddings, they form a matrix P with columns representing each position.

Convolutional layers. In ConvS2S, each convolution is gated (Meng et al., 2015; Dauphin et al., 2017) *i.e.* it yields double the requested channels then use one half to gate the other.

$$\text{Conv}(\text{X}) = [\text{A}, \text{B}]^\top, \tag{2.32}$$

$$\text{Gated-conv}(\text{X}) = \text{GLU}(\text{Conv}(\text{X})) = \text{A} \odot \text{sigmoid}(\text{B}), \tag{2.33}$$

where $\odot$ is the Hadamard product (element-wise) and GLU is the gated linear unit, a non-linearity using half of the input channels to scale the other half as illustrated in Figure 2.2.

The convolutions are also surrounded with residual connections (He et al., 2016b). If the layer map is F then the final output is the following:

$$\mathrm{Y} = \gamma(\mathrm{X} + \mathrm{F}(\mathrm{X})), \tag{2.34}$$

with γ a scaling factor set to $\sqrt{0.5}$ to stabilize the feature variance.

Attention. Each layer of the decoder is equipped with a dot-product attention module. In its most generic form, this module has three input matrices, the keys $\mathrm{K} \in \mathbb{R}^{d \times |\boldsymbol{x}|}$, the values $\mathrm{V} \in \mathbb{R}^{d \times |\boldsymbol{x}|}$ and the queries $\mathrm{Q} \in \mathbb{R}^{d \times |\boldsymbol{y}|}$ and it produces a matrix in $\mathbb{R}^{d \times |\boldsymbol{y}|}$:

$$\mathrm{Attention}(\mathrm{K}, \mathrm{V}, \mathrm{Q}) = \mathrm{V} \cdot \mathrm{softmax}(\mathrm{K}^\top \mathrm{Q}). \tag{2.35}$$

ConvS2S particularly places a large importance on the input embeddings as they are integrated into the attention values V and queries Q. Let us consider S^0 and S^N to be the initial and final states of the encoder respectively, and let us denote with H^n the output of the decoder's n-th layer and with H^0 its initial input. The n-th layer performs the following transformations:

$$\widetilde{\mathrm{H}}^n = \gamma\left(\mathrm{H}^{n-1} + \text{Gated-conv}(\mathrm{H}^{n-1})\right), \tag{2.36}$$

$$\mathrm{Q} = \gamma(\mathrm{W}_\mathrm{Q}\widetilde{\mathrm{H}}^n + \mathrm{H}^0), \quad \mathrm{K} = \mathrm{S}^\mathrm{N}, \quad \mathrm{V} = \gamma(\mathrm{S}^\mathrm{N} + \mathrm{S}^0), \tag{2.37}$$

$$\mathrm{O} = \mathrm{Attention}(\mathrm{K}, \mathrm{V}, \mathrm{Q}), \quad \mathrm{H}^n = \gamma(\mathrm{W}_\mathrm{O}\mathrm{O} + \widetilde{\mathrm{H}}^n). \tag{2.38}$$

Regularization. For regularization, dropout masks are applied to the input sum of token and position embeddings and to the output of each convolution before the gating and the residual connection with a rate p_{drop}.

2.6.3 Transformer

This sequence-to-sequence model is based entirely on attention. The input to the Transformer encoder (*resp.* decoder) is a sum of token and position embeddings. This input is processed with a stack of N blocks. Each encoder block has 2 sub-blocks, multi-head self-attention and a feed-forward network. The decoder has an additional sub-block, a multi-head attention to interact with the source states. The key elements of this architecture are described below.

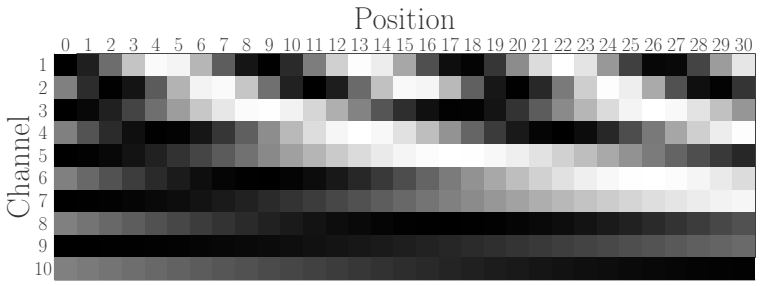


Figure 2.3: Sinusoidal positional embeddings. Values in the $[-1, 1]$ range with larger values in darker shades.

Position embedding. Similar to ConvS2S, Transformer uses position embeddings, but instead of being learned, these are fixed and are based on sinusoids of different frequencies. If P is the embedding matrix with columns in $\mathbb{R}^d$ associating the i-th position to its i-th column, then:

$$\omega_k = 10000^{-2k/d}, \quad \mathrm{P}_{j,i} = \begin{cases} \sin\left(\omega_k i\right), & \text{if } j = 2k \\ \sin\left(\frac{\pi}{2} + \omega_k i\right), & \text{if } j = 2k+1. \end{cases} \tag{2.39}$$

Each dimension of the positional encoding corresponds to a sinusoid. The wavelengths $2\pi/\omega_k$ progress geometrically resembling alternating bits in continuous space (*c.f.* Figure 2.3).

Multi-head attention. Multi-heading consists of concatenating the outputs of n_{H} different heads, producing vector representations in $\mathbb{R}^{n_{\mathrm{H}} d_h}$. An attention head receives key-value pairs (K, V) and queries Q, and outputs d_h-dimensional columns:

$$\mathrm{Attention}(\mathrm{K}, \mathrm{V}, \mathrm{Q}) = (\mathrm{W_V V}) \cdot \mathrm{softmax}\left(\frac{1}{\sqrt{d_h}} \left(\mathrm{W_K K}\right)^\top (\mathrm{W_Q Q})\right), \tag{2.40}$$

where the weights $(\mathrm{W_K}, \mathrm{W_V}, \mathrm{W_Q})$ map the keys, values and queries to $\mathbb{R}^{d_h}$. Except from the normalizing scale $\frac{1}{\sqrt{d_h}}$, this attention is similar to the usual dot-product attention. In the case of self-attention, $(\mathrm{K}, \mathrm{V}, \mathrm{Q})$ are all set to the encoder (*resp.* decoder) hidden states, with the appropriate masking in the decoder side (*c.f.* §2.4.3). For encoder-decoder interfacing the keys and values are the final encoder states and the queries are the decoder states.

Feed-forward network. The second sub-block is a point-wise feed-forward network FF consisting of two affine transformations with a non-linear function, ReLU, in between. ReLU

(Nair and Hinton, 2010) is defined as $\mathrm{ReLU}(x) = \max(0, x)$.

$$\mathrm{FF}(x) = \mathrm{W}_2 \,\mathrm{ReLU}(\mathrm{W}_1 x + b_1) + b_2. \tag{2.41}$$

The matrix $\mathrm{W}_1 \in \mathbb{R}^{d_{\mathrm{FF}} \times d}$ maps to a larger space, usually $d_{\mathrm{FF}} = 4d$, and $\mathrm{W}_2 \in \mathbb{R}^{d \times d_{\mathrm{FF}}}$ maps back to $\mathbb{R}^d$.

Skip connections and normalization. Multi-head (self-)attention and the feed-forward network are surrounded with a residual connection followed with a layer-normalization (Ba et al., 2016). This normalized skip-connection is referred to as AddNorm. With F a map and LN the layer-normalization operator,

$$\mathrm{AddNorm}(x) = \mathrm{LN}(x + \mathrm{F}(x)). \tag{2.42}$$

LN computes statistics over the channels for each sample and for each time-step, and normalizes the activations. LN has usually two learnable affine transformation parameters (β, γ). For an activation vector $a \in \mathbb{R}^d$:

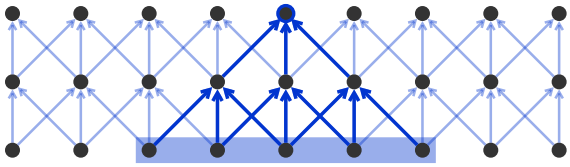
$$\mu = \frac{1}{d}\sum_{k=1}^{d} a_k, \quad \sigma^2 = \frac{1}{d}\sum_{k=1}^{d}(a_k - \mu)^2, \quad \mathrm{LN}(a) = \frac{1}{\sigma}\gamma \odot (a - \mu\mathbf{1}) + \beta. \tag{2.43}$$

Compared to the rigid scaling of Eq. (2.34), this normalization scheme flexibly conditions its output for the next layer.

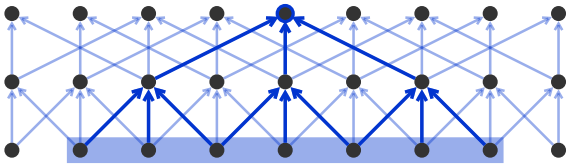
Regularization. For regularization, dropout masks are applied to the input sum of token and position embeddings and to the output of each layer (FF and multihead-attention) before the residual connection with a rate p_{drop}.

2.7 Discussion

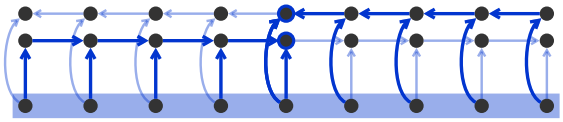
An interesting concept when looking at sequence models is that of the *receptive-field*. Borrowed from computer vision, the receptive-field is the size of the context window captured in a given state vector. This notion is often paired with the amount of *time* and *computation* needed to reach an unbounded receptive-field.



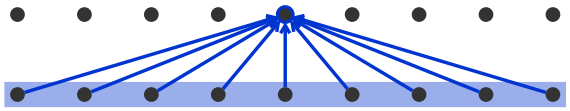
(a) A stack of two normal convolutions.



(b) A dilated convolution on top of a normal convolution.



(c) A bi-directional RNN.



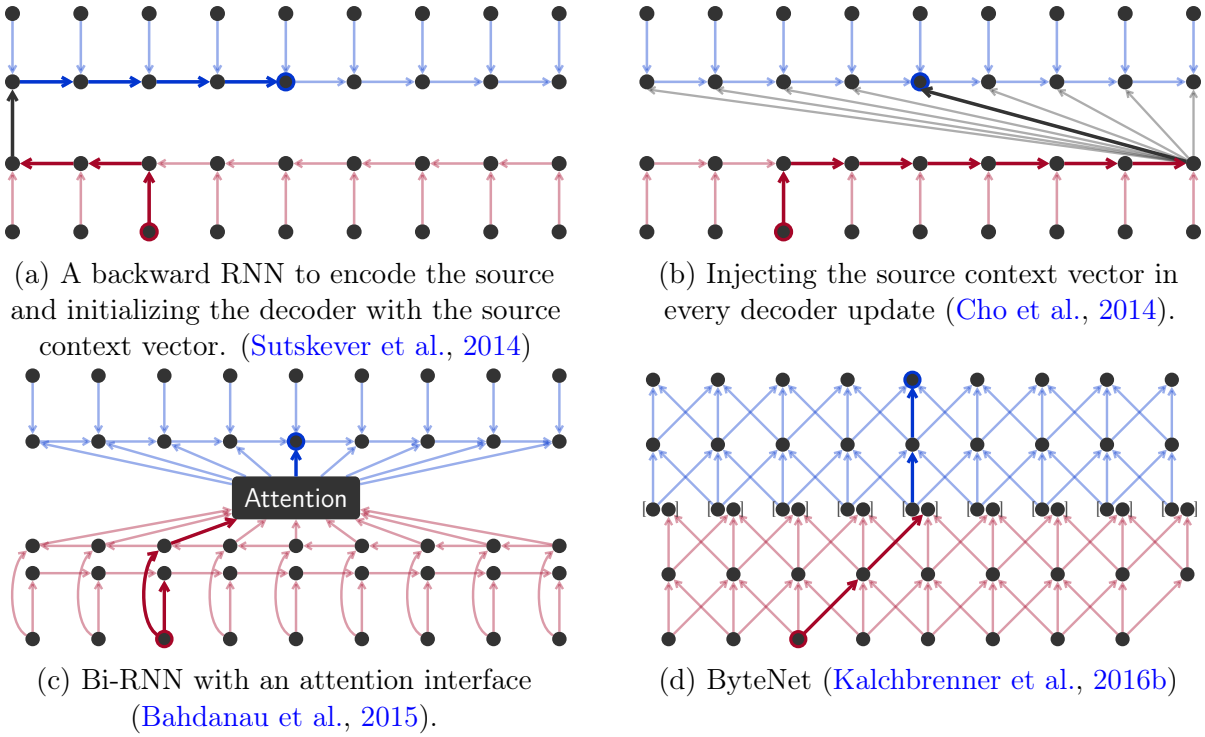
(d) Self-attention.

Figure 2.4: The receptive-field growth in sequence models: for a select position, we highlight the receptive-field at the input level and the paths leading to it.

In a convolutional encoder with d-dimensional states, a single convolution with a k-wide filter has a receptive-field of k tokens and costs $O(kd^2 \cdot |\boldsymbol{x}|)$ in $O(1)$ sequential steps, meaning that convolving a sequence can be performed in parallel. Stacking convolutions widens the receptive-field linearly with the depth *i.e.* after N convolutions, $1 + \mathrm{N}(k-1)$ tokens are captured in the states of the ultimate layer. Dilating the convolutions helps speeding up the growth of the receptive-field, in fact, with well chosen dilation factors the receptive-field can grow exponentially with the depth N (Kalchbrenner et al., 2016b).

In Figure 2.4a, stacking two normal convolutions with $k = 3$ gives a receptive-field of 5. With the same amount of computation, dilating the top convolution, as in Figure 2.4b, gives a receptive-field of 7. Besides the size of the receptive-field, it is important to measure the path between a *teaching signal i.e.* the deepest state at a given time-step directly linked to a classifier, and the input token captured in its receptive-field. This path plays an important role in the model's ability to capture long-range dependencies, since the training signal needs to traverse it forward and backward (Hochreiter et al., 2001). The path in convolutional encoders is of constant length N as long as $|\boldsymbol{x}| \leq 1 + \mathrm{N}(k-1)$. In the examples of Figures 2.4a and 2.4b each path from the top blue state to its inputs consists of two edges. However, more layers are needed to cover the full sequence.

Recurrent encoders have an unbounded receptive field after only a single layer. If bi-directional, the concatenation of forward and backward states at a time-step encodes all



(a) A backward RNN to encode the source and initializing the decoder with the source context vector. (Sutskever et al., 2014)

(b) Injecting the source context vector in every decoder update (Cho et al., 2014).

(c) Bi-RNN with an attention interface (Bahdanau et al., 2015).

(d) ByteNet (Kalchbrenner et al., 2016b)

Figure 2.5: Interfacing the encoder and decoder. The encoder in red and the decoder in blue. In each architecture we highlight the path from x_3 to the teaching signal associated with y_5.

future and past tokens (*c.f.* Figure 2.4c). This unbounded receptive field comes at the cost of computation time; the recursions characteristic of RNNs are not parallelizable and cost $O(d^2 \cdot |\boldsymbol{x}|)$ in $O(|\boldsymbol{x}|)$ sequential steps. The recursion also affects the path between the teaching signal and the input, reaching a maximum length of $O(|\boldsymbol{x}|)$.

Self-attention has an unbounded receptive field accessed in a single step and parallelizable computation across time-steps (*c.f.* Figure 2.4d). The computation cost is $O(d \cdot |\boldsymbol{x}|^2)$ in $O(1)$ sequential steps. The quadratic cost w.r.t. the sequence length $|\boldsymbol{x}|$ is especially taxing for long sequences.

In sequence-to-sequence models, the only teaching signals come from the decoder. Therefore, we measure the path from a target-side state h_t back to an input in the prefix $\boldsymbol{y}_{<t}$ and the path from h_t back to a source token x_j. The first path is the one discussed for encoders, the second path however, depends on the way the encoder and the decoder are interfaced.

Model	RF	Computation	Sequential	$\boldsymbol{y}$-path	$\boldsymbol{x}$-path
RNN (Cho et al., 2014)	∞	$O(\mathrm{N}d^2\lvert\boldsymbol{x}\rvert+\mathrm{N}d^2\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N}(\lvert\boldsymbol{x}\rvert+\lvert\boldsymbol{y}\rvert))$	$O(\lvert\boldsymbol{y}\rvert)$	$O(\lvert\boldsymbol{x}\rvert)$
RNN (Sutskever et al., 2014)	∞	$O(\mathrm{N}d^2\lvert\boldsymbol{x}\rvert+\mathrm{N}d^2\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N}(\lvert\boldsymbol{x}\rvert+\lvert\boldsymbol{y}\rvert))$	$O(\lvert\boldsymbol{y}\rvert)$	$O(\lvert\boldsymbol{x}\rvert+\lvert\boldsymbol{y}\rvert)$
RNN + attention (Luong et al., 2015)	∞	$O(\mathrm{N}d^2\lvert\boldsymbol{x}\rvert+\mathrm{N}d^2\lvert\boldsymbol{y}\rvert+d\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N}(\lvert\boldsymbol{x}\rvert+\lvert\boldsymbol{y}\rvert))$	$O(\lvert\boldsymbol{y}\rvert)$	$O(1)$
ByteNet (Kalchbrenner et al., 2016b)	$O(k2^{\mathrm{N}})$	$O(\mathrm{N}kd^2\lvert\boldsymbol{x}\rvert+\mathrm{N}kd^2\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N})$	$O(\log_k(\lvert\boldsymbol{y}\rvert))$	$O(log_k(\lvert\boldsymbol{y}\rvert\lvert\boldsymbol{x}\rvert))$
ConvS2S (Gehring et al., 2017)	$O(k\mathrm{N})$	$O(\mathrm{N}kd^2\lvert\boldsymbol{x}\rvert+\mathrm{N}kd^2\lvert\boldsymbol{y}\rvert+\mathrm{N}d\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N})$	$O(\lvert\boldsymbol{y}\rvert/k)$	$O(1)$
Transformer (Vaswani et al., 2017)	∞	$O\big(\mathrm{N}d\lvert\boldsymbol{x}\rvert^2+\mathrm{N}d\lvert\boldsymbol{y}\rvert^2+\mathrm{N}d\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert\big)$	$O(\mathrm{N})$	$O(1)$	$O(1)$

Table 2.1: Properties of sequence-to-sequence models.

When using uni-directional recurrent networks, if the source is encoded in a single vector c that is injected in every update, signals from a source token x_j need to travel the remaining of the source sequence before reaching a target state h_t *i.e.* $|\boldsymbol{x}| - j + 1$ edges. If c is used to initialize the decoder states this distance is $|\boldsymbol{x}| - j + 1 + t$. Sutskever et al. (2014) opted for a backward recursion to encode $\boldsymbol{x}$ as they found it to improve over a forward encoder. With this change, the path is $t + j$ long which is particularly short for the initial time-steps. With an attention-mechanism in between, the path length from a target-side state to a source token is reduced to $O(1)$.

A particular interface is introduced in ByteNet (Kalchbrenner et al., 2016b) where the decoder is placed on top of the encoder. Since the two sequences are of different lengths, the encoder is parameterized to generate $a|\boldsymbol{x}| + b$ context vectors with wide convolutions and (a, b) chosen to approximate the length of target sequences in the corpus. With that, the convolutional decoder is fed the concatenation of a context vector from the encoder and a target token embedding. This design, with the use of dilated convolutions, reduces all paths to a logarithmic length.

Table 2.1 summarizes the discussed properties for a few key architectures. RF refers to the receptive-field and the $\boldsymbol{y}$-path and $\boldsymbol{x}$-path to the maximum length of the path from a teaching signal to a token in $\boldsymbol{y}$ or a token in $\boldsymbol{x}$ respectively. Each model has N processing layers in its encoder and decoder. It is worth noting that recurrent architectures are usually shallow in comparison to convolutional ones. The model's dimension, *i.e.* size of the embeddings and

hidden states, is d and convolutional layers use a filter of width k.

2.8 Conclusion

In this chapter, we have covered the background knowledge needed to understand the context of this thesis. We first described the task of neural machine translation as a source-conditioned language model and went on to review existing sequence models to encode the source and language models to generate the target. We proceeded with describing the training and testing of NMT models and detailed the baseline models that will be used in the rest of this thesis. We emphasized some properties of NMT models we deem central to the understanding of their effectiveness. These properties will guide our design choices and motivate our contributions in the following chapters.

Chapter 3

Pervasive Attention

We present in this chapter our novel NMT model dubbed Pervasive Attention. This sequence-to-sequence model addresses the issue of source-target interaction in existing encoder-decoder models by jointly encoding the two sequences with a two-dimensional convolutional neural network. We propose different design choices for each building block of Pervasive Attention and study their effects on the quality of translation. Experiments on a set of MT benchmarks show results competitive with state-of-the-art encoder-decoder models.

Some of the contributions presented in this chapter are published in:

> Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018a. Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction. In *Proc. of CoNLL*

Contents

3.1 Introduction 34
3.2 Pervasive Attention Model 35
3.3 Experiments 47
3.4 Analysis and Discussion 56
3.5 Related Work 57
3.6 Conclusion 61

3.1 Introduction

Sequence-to-sequence models, introduced in Chapter 2, achieve state-of-the-art performances in a plethora of tasks, including, but not limited to, speech synthesis (Oord et al., 2016a; Wang et al., 2017; Arik et al., 2017), image captioning (Vinyals et al., 2015; Xu et al., 2015; Yao et al., 2017a), machine comprehension and question answering (Shen et al., 2017; Chen et al., 2017; Devlin et al., 2019), semantic parsing (Liang et al., 2017) and machine translation (Ng et al., 2019). These models are based on processing a source sequence with an encoder and using its output to condition the generation of the target sequence by a separate decoder. We raise two limitations of the encoder-decoder design:

Limited capacity. The attention mechanism interfacing the encoder and the decoder is limited to weighting the source representations with basic similarity scores. In fact, Michel et al. (2019) established that the multiple heads in Transformer layers are more important in the encoder-decoder attention layers than in self-attention ones *i.e.* a complex similarity measure is needed to model the source-target interaction. We speculate that enriching the similarity measure of the interface is crucial to improving sequence-to-sequence models.

Late interaction. Be it a single representation or a set of feature vectors, the encoding of the source is completed before advancing to the decoder. The decoder has only access to high-level source features. In deep RNN-based encoder-decoder models, the attention mechanism connects the last layers in the encoder and decoder. In other variants of this recurrent model, Wu et al. (2016) connects the last encoder layer with the first layer in the decoder. The evaluated context is then injected in all subsequent decoder layers. Although this choice was made to improve the training parallelism, it allows for the decoder to interact with the source from an early stage. ConvS2S and Transformer also score the ultimate encoder states evaluating different contexts at every decoding level. These state-of-the-art models share the property of an early interaction on the decoder side. We argue that we can push this further and have the two encoder-decoder models interwoven.

We propose to build a grid of pairwise interactions between the source and target tokens and process this two-dimensional input with a deep convolutional network (ConvNet). This

way, the source and target sequences meet from the starting point and are *co-encoded* in increasing abstraction levels throughout the ConvNet. Moreover, this interaction or attention is *pervasive* in the model and is more rich and complex in its modeling of $(\boldsymbol{x}, \boldsymbol{y})$ as a pair.

We will describe in §3.2 our co-encoding NMT model dubbed *Pervasive Attention*, detailing its different building blocks. In the experiments section §3.3, we start with the results of an ablation study on IWSLT'14 De→En to tune our design choices (§3.3.2), afterwards, we compare to state-of-the-art encoder-decoder models on IWSLT'14 De→En, IWSLT'15 en→Vi, WMT'16 En→Ro and WMT'15 De→En (§3.3.3). We will end the chapter with a discussion of the reported results and a few comments on recent works related to ours (§3.4 and §3.5).

3.2 Pervasive Attention Model

Let us consider a neural machine translation task with a source-side vocabulary $\mathcal{V}_x$ and a target side vocabulary $\mathcal{V}_y$, and let $(\boldsymbol{x}, \boldsymbol{y})$ be a pair of source-target sequences. Our model consists of building a grid representation of the pair $(\boldsymbol{x}, \boldsymbol{y})$ and then processing this grid with a resolution-preserving convolutional neural network. The final features of this network are aggregated in order to have a fixed-size representation per target token. An overview of the Pervasive Attention model is provided in Figure 3.1.

3.2.1 Input Representation

To build the input tensor of our model, we consider embedding matrices $\mathrm{E}_x \in \mathbb{R}^{d \times |\mathcal{V}_x|}$ and $\mathrm{E}_y \in \mathbb{R}^{d \times |\mathcal{V}_y|}$ for the source and target-side respectively. The input to our model is a three-dimensional tensor $\mathrm{H}^0 \in \mathbb{R}^{|\boldsymbol{y}| \times |\boldsymbol{x}| \times d}$ corresponding to a $|\boldsymbol{y}| \times |\boldsymbol{x}|$ grid where at each position (t, j) we concatenate the embedding of the source word x_j with the embedding of the target word y_t and project the concatenation to $\mathbb{R}^d$ with the matrix $W_0 \in \mathbb{R}^{d \times 2d}$.

$$\forall t, j \in [1..|\boldsymbol{y}|] \times [1..|\boldsymbol{x}|],\ \mathrm{H}^0_{tj} = \mathrm{W}_0 \begin{bmatrix} \mathrm{E}_y y_t \\ \mathrm{E}_x x_j \end{bmatrix} \in \mathbb{R}^d. \tag{3.1}$$

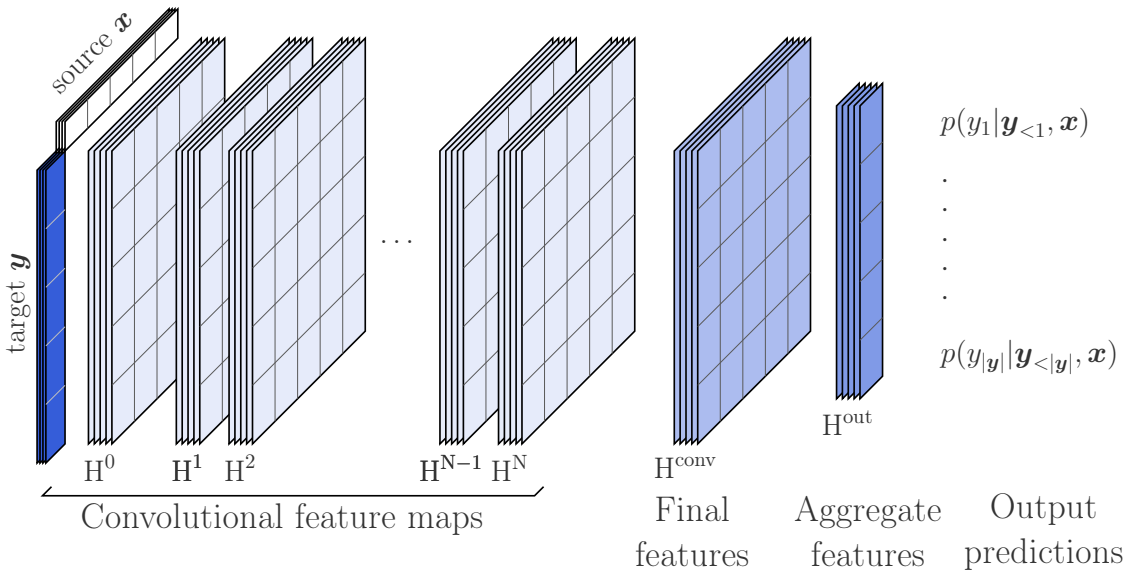


Figure 3.1: Overview of the Pervasive Attention model.

Note that we fix here a dimension d characteristic of the model and set $d_{\text{emb}} = d$ for both embedding matrices. We can alternatively have embeddings of different sizes and control with W_0 the dimension of the ConvNet's input.

3.2.2 Convolutions

The input features H^0 are processed with a ConvNet *i.e.* a set of stacked convolutional layers to jointly encode at each position (t, j) the source $\boldsymbol{x}$ and the target prefix $\boldsymbol{y}_{\leq t}$. The main operator in these layers is a causal two-dimensional convolution. In the following we describe this operator and detail the different ConvNet designs considered in this work.

Causal convolutions. As seen in §2.4.2, auto-regressive convolutional language models use masked convolutions. This type of convolution, also called *causal*, guarantees that the input at time t is convolved only with elements from time t and earlier. Tied with this property is that the convolution yields an output of the same length as its input. With two-dimensional convolutions for sequence-to-sequence tasks we have two time axes, one along the source $\boldsymbol{x}$ and the other along the target $\boldsymbol{y}$. For the task of offline translation where the source sequence $\boldsymbol{x}$ is fully available at the beginning of decoding, we will only tend to the causality along the

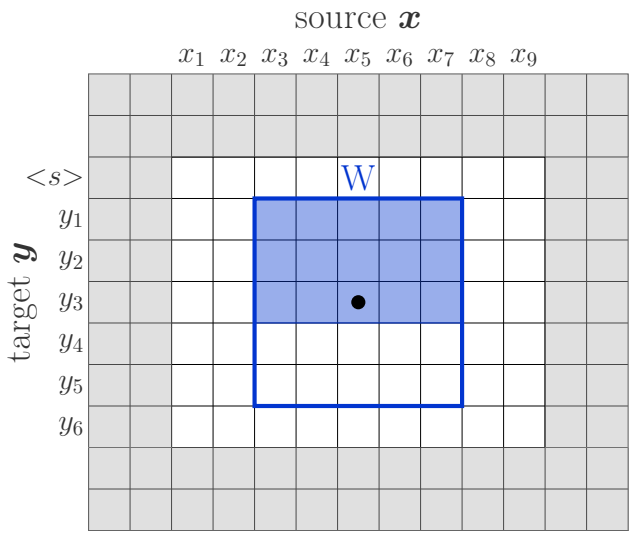


Figure 3.2: Illustration of a causal two-dimensional convolution with filter $\mathrm{W} \in \mathbb{R}^{5\times5}$. At the position marked with •, the convolution only includes signals from the highlighted blue area. The rest of W is zeroed out. The gray positions are zero-paddings to maintain the input resolution; with $k_x = k_y = 5$ the grid is padded twice on each side.

y-axis. With that, let $\mathrm{W} \in \mathbb{R}^{k_y \times k_x}$ be a convolution filter:

a) The input at position (t, j) of the grid is convolved only with *past* inputs in (t', j') where $1 \leq t' \leq t \leq |\boldsymbol{y}|$ and $1 \leq j' \leq |\boldsymbol{x}|$. This is achieved through zeroing out the corresponding elements of W:

$$\forall i \in \left[\left\lceil \frac{k_y}{2} \right\rceil .. k_y\right], \forall j \in [1..k_x], \mathrm{W}_{ij} = 0. \tag{3.2}$$

b) The convolved output has the same dimension as the input along both axes. Maintaining the y-resolution is necessary to yield the $|\boldsymbol{y}|$ outputs required. However, for the x-axis we chose to preserve the resolution in order to keep the grid interpretable and to avoid dimensionality issues we may face with short source sequences.[(1)]

The input dimension is maintained with zero paddings of lengths $k_y - 1$ and $k_x - 1$ to the y and x axes respectively.

With a stack of N causal convolutions, the receptive field is grown as a function of the filter size $k_y \times k_x$ and the depth N. In fact, after N convolutions, the features at a given position

[(1)] The input has to be wider than the filter, but with stacked convolutions that reduce the input dimension, we risk reducing the input beyond the size of the filter especially with short sequences.

(t, j) encode $\mathrm{N}\lfloor k_x/2 \rfloor$ source tokens to the left and to the right of x_j and $\mathrm{N}\lfloor k_y/2 \rfloor$ target tokens preceding y_t. Our two-dimensional causal convolution is illustrated in Figure 3.2.

3.2.3 Skip Connections

Another key aspect of a ConvNet is the skip connections used to aggregate the outputs of its early layers to be consumed by later ones. In computer vision, these connections are critical to the training of very deep networks with hundreds of layers. In NLP, however, the architectures are wide and shallow with only 6 blocks in the state-of-the-art Transformer. We hypothesize that a good NLP model should understand the sequence at increasing levels of abstraction. Skip connections provide a direct access to the previous levels of abstraction so that the next map could infer deeper features without having to *remember* or *store* previous ones.

Let $\mathrm{F}_1, \ldots, \mathrm{F}_\mathrm{N}$ be the maps associated with N layers each outputting H^n. We will denote with $\widetilde{\mathrm{H}}^{n-1}$ the input of the n-th layer and with $\mathrm{H}^{\mathrm{conv}}$ the ultimate output of the ConvNet. We detail in the following two main types of skip connections, residual and dense. For residual connections, we distinguish four variants depending on what follows the residual summation and how the final output $\mathrm{H}^{\mathrm{conv}}$ is prepared.

Residual connections. First proposed in He et al. (2016c), these skip connections are widely adopted in vision and NLP models and consist of simply taking the sum of the features:

$$\forall n \in [1..\mathrm{N}], \quad \widetilde{\mathrm{H}}^0 = \mathrm{H}^0, \quad \mathrm{H}^n = \mathrm{F}_n\left(\widetilde{\mathrm{H}}^{n-1}\right), \quad \widetilde{\mathrm{H}}^n = \mathrm{H}^n + \widetilde{\mathrm{H}}^{n-1}, \quad \mathrm{H}^{\mathrm{conv}} = \sum_{n=0}^{\mathrm{N}} \mathrm{H}^n. \tag{3.3}$$

Residual-norm connections. A particular version of residual connections is addNorm, notably used in Transformer-based architectures (Vaswani et al., 2017). It follows each summation in the path with a layer-normalization (LN, see §2.6.3).

$$\forall n \in [1..\mathrm{N}], \quad \widetilde{\mathrm{H}}^0 = \mathrm{H}^0, \quad \mathrm{H}^n = \mathrm{F}_n\left(\widetilde{\mathrm{H}}^{n-1}\right), \quad \widetilde{\mathrm{H}}^n = \mathrm{LN}\left(\mathrm{H}^n + \widetilde{\mathrm{H}}^{n-1}\right). \tag{3.4}$$

Suppose that the features are normalized and that each layer-normalization operator amounts to an element-wise linear transformation with weights γ_n ($\gamma_0 = \mathbf{1}$). Then addNorm changes

the simple summation of residual connections to one with geometric-like weights.

$$\forall n \in [1..\mathrm{N}], \quad \widetilde{\mathrm{H}}^n = \sum_{k=0}^{n} \left(\bigodot_{k'=k}^{n} \gamma_{k'} \right) \odot \mathrm{H}^k, \quad \mathrm{H}^{\mathrm{conv}} = \sum_{n=0}^{\mathrm{N}} \left(\bigodot_{k=n}^{\mathrm{N}} \gamma_k \right) \odot \mathrm{H}^n, \tag{3.5}$$

where $\odot$ is the Hadamard product.

Residual-cumulative connections. In order to explicitly include mid-level abstractions in the final features $\mathrm{H}^{\mathrm{conv}}$, we propose to accumulate the residual outputs of the ConvNet.

$$\forall n \in [1..\mathrm{N}], \quad \widetilde{\mathrm{H}}^0 = \mathrm{H}^0, \quad \mathrm{H}^n = \mathrm{F}_n\left(\widetilde{\mathrm{H}}^{n-1}\right), \quad \widetilde{\mathrm{H}}^n = \gamma\left(\widetilde{\mathrm{H}}^{n-1} + \mathrm{H}^n\right), \tag{3.6}$$

$$\mathrm{H}^{\mathrm{conv}} = \frac{1}{\sqrt{\mathrm{N}+1}} \sum_{n=0}^{\mathrm{N}} \widetilde{\mathrm{H}}^n, \tag{3.7}$$

where we add a variance stabilizing multiplier $\gamma = 1/\sqrt{2}$ after each summation.[(2)] The final summation in Eq. (3.7) is also stabilized by the multiplier $1/\sqrt{\mathrm{N}+1}$. Explicitly, the inputs of the ConvNet's layers and the final aggregate $\mathrm{H}^{\mathrm{conv}}$ are weighted sums of the feature maps $(\mathrm{H}^0, \mathrm{H}^1, \ldots, \mathrm{H}^{\mathrm{N}})$:

$$\forall n \in [0..\mathrm{N}], \quad \widetilde{\mathrm{H}}^n = \sum_{k=0}^{n} \alpha_{n,k} \mathrm{H}^k, \quad \alpha_{n,0} = \gamma^n, \quad \alpha_{n,k} = \gamma^{n-k+1} \ (\forall k \in [1..n]), \tag{3.8}$$

$$\mathrm{H}^{\mathrm{conv}} = \frac{1}{\sqrt{\mathrm{N}+1}} \sum_{n=0}^{\mathrm{N}} \beta_n \mathrm{H}^n, \quad \beta_0 = \frac{(1-\gamma^{N+1})}{(1-\gamma)}, \quad \beta_n = \frac{\gamma(1-\gamma^{N-n+1})}{(1-\gamma)} \ (\forall n \in [1..\mathrm{N}]). \tag{3.9}$$

At fixed n, the weights $\alpha_{n,k}$ are strictly increasing with the index k *i.e.* far away early features are assigned lower weights. The weights β_n have the opposite monotonicity and so, early layers are assigned higher weights.

Residual-gated connections. We experimented with a parameterized information flow in the ConvNet with the help of gating mechanisms similar to *Highway Networks* (Srivastava et al., 2015). We particularly separate the encoding flow from that of aggregating the features into a final representation. Let us consider two gates: *forward* f_n and *snapshot* s_n. The layer's signal committed to the final features $\mathrm{H}^{\mathrm{conv}}$ is controlled with the *snapshot* gate. This

[(2)] Assuming the maps F_n surrounded by the residual connections maintain their inputs' variance, then this multiplier guarantees the variance remains the same after the summation.

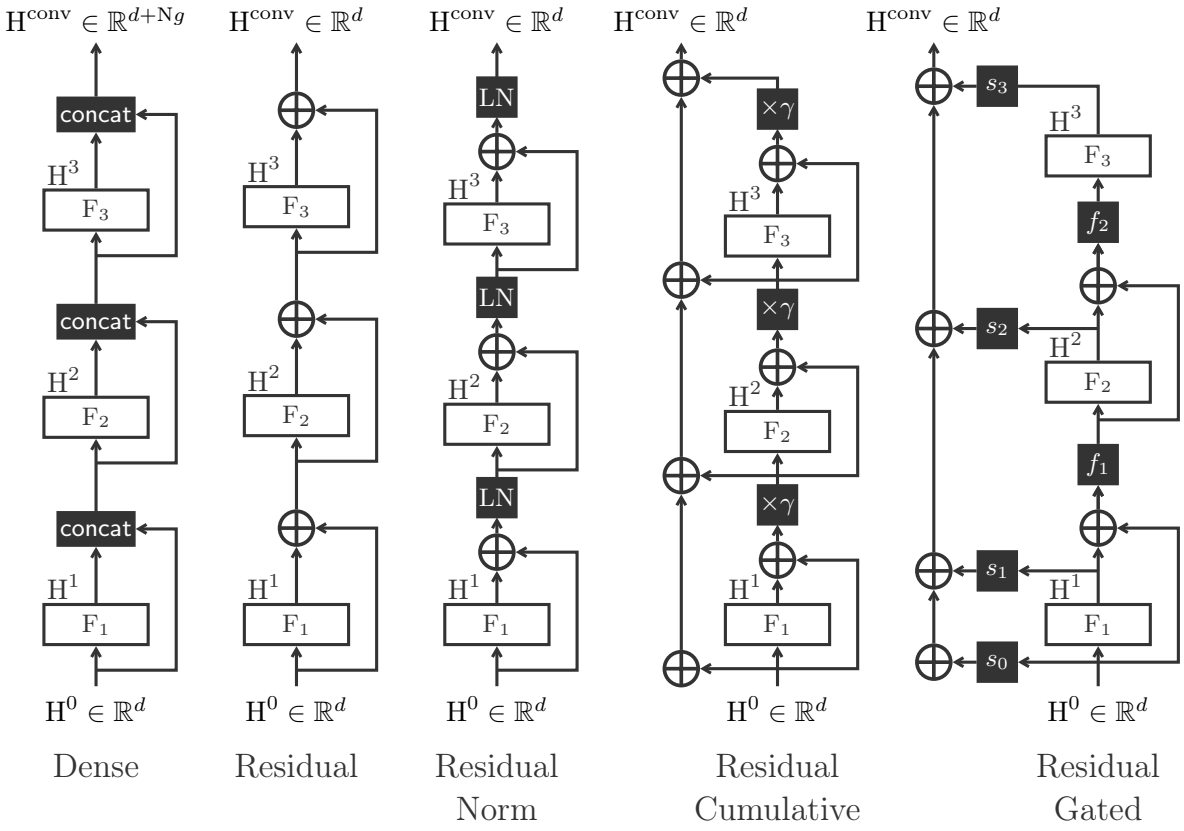


Figure 3.3: Skip connections for Pervasive Attention. Each ConvNet is illustrated with N = 3 blocks.

gate does not affect the features fed to the next layer in the ConvNet. The *forward* gate scales the output of the residual connection preparing it for the next block.

$$\forall n \in [0..\mathrm{N}], \quad \widetilde{\mathrm{H}}^0 = \mathrm{H}^0, \quad \mathrm{H}^n = \mathrm{F}_n(\widetilde{\mathrm{H}}^{n-1}), \quad \widetilde{\mathrm{H}}^n = f_n\left(\widetilde{\mathrm{H}}^{n-1} + \mathrm{H}^n\right), \tag{3.10}$$

$$\mathrm{H}^{\mathrm{conv}} = \sum_{n=0}^{\mathrm{N}} s_n(\mathrm{H}^n). \tag{3.11}$$

The gates can be better visualized in Figure 3.3. Beside controlling the information flow, the motivation behind gating the skip connections is to visualize and interpret the contribution of every layer or level of abstraction into the final features. We experimented with a few options for the *forward* and *snapshot* gates and settled on constant gates that barely affect the model's complexity and are easily interpretable. In the following the gates are channel-wise

multipliers comparable to the affine transformation that follows layer-normalization:

$$\forall n \in [1..\mathrm{N}],\ s_n, f_n \in \mathbb{R}^d, \quad s_n(\mathrm{H}^n) = s_n \odot \mathrm{H}^n, \quad f_n(\widetilde{\mathrm{H}}^{n-1} + \mathrm{H}^n) = f_n \odot (\widetilde{\mathrm{H}}^{n-1} + \mathrm{H}^n) \quad (3.12)$$

Dense connections:. Introduced in Huang et al. (2017), they change the summations of residual connections to concatenations.

$$\forall n \in [1..\mathrm{N}],\ \mathrm{H}^n = \mathrm{F}_n\left([\mathrm{H}^0, \mathrm{H}^1, \ldots, \mathrm{H}^{n-1}]^\top\right), \quad (3.13)$$

$$\mathrm{H}^{\mathrm{conv}} = [\mathrm{H}^1, \mathrm{H}^2, \ldots, \mathrm{H}^{\mathrm{N}}]^\top. \quad (3.14)$$

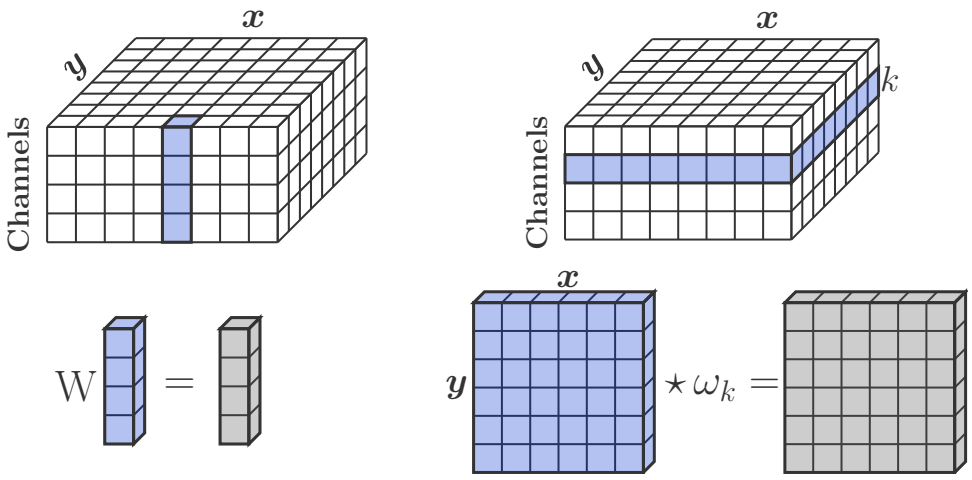
Figure 3.3 provides a schematic overview of the residual and dense skip connections.

3.2.4 Building Layers

Depending on the skip connection, the layer map F_n should be appropriately designed. We will describe in the following the layers used with residual and dense skip connections.

Residual layer. The only constraint on a residual layer is that it yields an output of the same resolution and the same number of features as its input. To mimic a Transformer block where layers alternate between self-attention and position-wise feed forward maps, we follow each convolution with a position-wise feed-forward layer. This design separates the temporal interactions between tokens modeled with the convolutional layer from the position-wise transformations applied to each token independently with the feed-forward network. To strengthen the separation between the temporal interactions and the point-wise transformation, we use a depth-wise separable convolution (Chollet, 2017). With that, we first apply a point-wise convolution *i.e.* using 1-wide filters (or equivalently a position-wise affine map) and follow it with a causal-convolution applied depth-wise. A depth-wise convolution operates on each channel independently mapping them one-to-one instead of the usual summation over channels (see §2.3.1). The depth-wise separable convolution is illustrated in Figure 3.4.

The feed-forward layer, similar to the one in Transformer blocks, is built with two affine maps interleaved with a ReLU non-linearity (Nair and Hinton, 2010). Ultimately, the maps



(a) Point-wise convolution (b) Depth-wise convolution

Figure 3.4: Depth-wise separable convolution.

F_n take the form:

$$\mathrm{F}_n = \begin{cases} \mathrm{Conv}(k_y \times k_x, d \to d) \circ \mathrm{Conv}(1 \times 1, d \to d) & \text{if } n \equiv 1 \pmod 2 \\ \mathrm{Linear}(d_{\mathrm{FF}} \to d) \circ \mathrm{ReLU} \circ \mathrm{Linear}(d \to d_{\mathrm{FF}}) & \text{if } n \equiv 0 \pmod 2 \end{cases} \tag{3.15}$$

Although we define the ConvNet as alternating layers, we will use the same block count as Transformer *i.e.* N blocks are effectively 2N layers surrounded with residual connections.

Dense layer. Since the feature maps are to be concatenated, each layer is forced to produce a small number of features. We refer to this number of features by the growth rate and denote it with g. Unlike residual-norm skip connections where we normalize the residual sum, it's common in computer vision architectures to normalize the features within the layer map. In this case, F_n first normalizes its input then applies ReLU followed by a linear map to $\mathbb{R}^{4g}$. This map controls the computational cost and memory footprint of the block by reducing the growing number of features to $4g$. In fact, with d the number of the ConvNet's input feature maps, the n-th layer receives $d+(n-1)g$ feature maps. A second $\mathrm{ReLU} \circ \mathrm{Norm}$ follows, and the last step is the main two-dimensional causal convolution with $k_y \times k_x$ filters. The composite function $\mathrm{Conv} \circ \mathrm{ReLU} \circ \mathrm{Norm}$ used twice[(3)] within the layer is the residual

[(3)] the first linear map is equivalent to a point-wise convolution

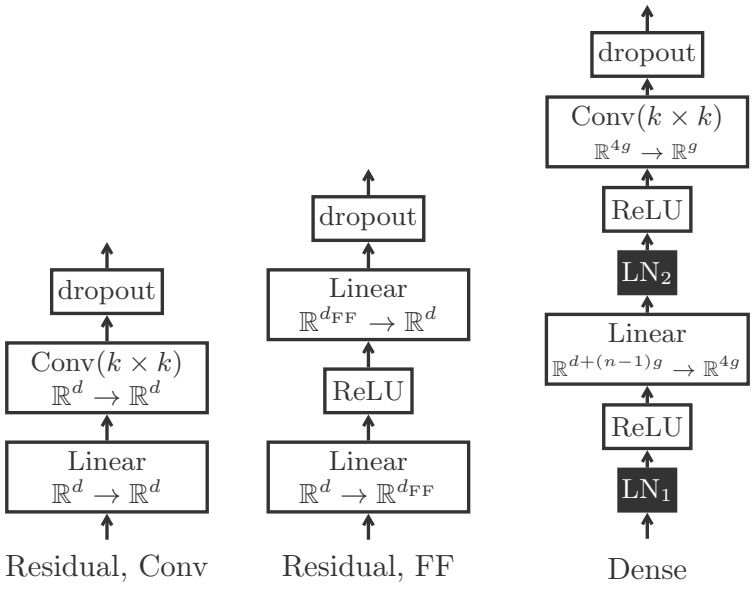
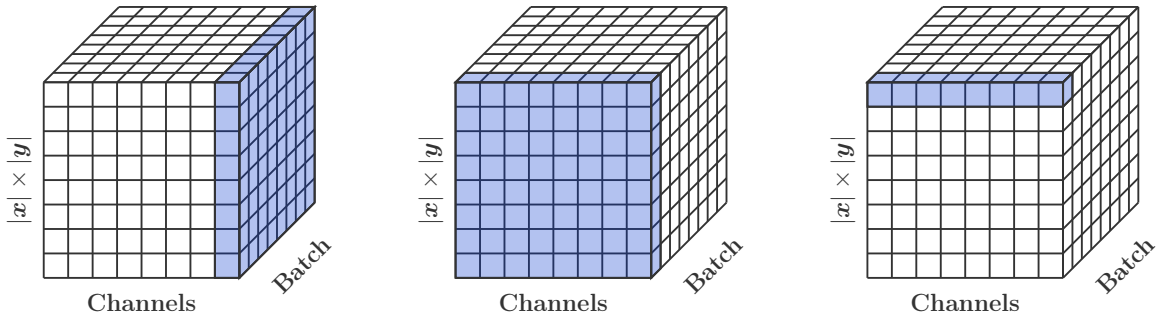


Figure 3.5: Layers for Pervasive Attention: Residual and dense layers. For regularization, each map is followed with a dropout mask.

unit proposed in He et al. (2016c) shown to be much easier to train with better generalization. Both dense and residual layers are illustrated in Figure 3.5.

On normalization. ConvNets with dense or residual connections were introduced for vision tasks and conventionally used batch-normalization (Ioffe and Szegedy, 2015). However, this normalization scheme is not fitting for auto-regressive language modeling so we swap it with layer-normalization (Ba et al., 2016). Different from the adaptation of layer-normalization in vision architectures (*e.g.* in Wu and He (2018)) where it is assumed to normalize across all channels and positions for a given sample (see Figure 3.6b), for Pervasive Attention, we normalize position-wise across channels to prohibit leakage from future positions to past ones (see Figure 3.6c).

For a fair comparison between ConvNets, we force the production of d features so that we can have equally sized classifiers $\mathrm{W}_p = \mathrm{E}_y^\top \in \mathbb{R}^{|\mathcal{V}| \times d}$. With dense skip connections, and to avoid an aggressive compression from $\mathbb{R}^{d+\mathrm{N}g}$ to $\mathbb{R}^d$, we stack a few linear layers ultimately mapping to $\mathbb{R}^d$.



(a) Batch-normalization (b) Layer-normalization (CV) (c) Layer-normalization (NLP)

Figure 3.6: Normalization schemes for Pervasive Attention.

3.2.5 Aggregation for Sequence Prediction

To compute a distribution over the output vocabulary $\mathcal{V}_y$, we need to first yield a fixed-size representation for each token in $\boldsymbol{y}$. However, the ConvNet final features $\mathrm{H}^{\mathrm{conv}}$ are in $\mathbb{R}^{|\boldsymbol{y}|\times|\boldsymbol{x}|\times d}$ which means that, for every target position, we have to aggregate the $|\boldsymbol{x}|$ $\mathbb{R}^d$-vectors into a fixed-size representation. We will denote this output with $\mathrm{H}^{\mathrm{out}} \in \mathbb{R}^{|\boldsymbol{y}|\times d}$. Each token y_t will be predicted with $\mathrm{H}^{\mathrm{out}}_{t-1}$.

Max-pooling. The simplest aggregation approach is to max-pool the $|\boldsymbol{x}|$ vectors *i.e.* for every channel $k \in [1..d]$, we select the most active position:

$$\forall t \in [1..|\boldsymbol{y}|],\ \forall k \in [1..d],\ \mathrm{H}^{\mathrm{out}}_{t-1,k} = \max_{1\leq j\leq|\boldsymbol{x}|} \mathrm{H}^{\mathrm{conv}}_{t-1,j,k}. \tag{3.16}$$

Average-pooling. Instead of selecting the most active position, we can take the average of the $|\boldsymbol{x}|$ vectors. We scale the average with the inverse square-root of the source length to stabilize the variance of the pooled features:

$$\forall t \in [1..|\boldsymbol{y}|],\ \forall k \in [1..d],\ \mathrm{H}^{\mathrm{out}}_{t-1,k} = \frac{1}{\sqrt{|\boldsymbol{x}|}} \sum_{1\leq j\leq|\boldsymbol{x}|} \mathrm{H}^{\mathrm{conv}}_{t-1,j,k}. \tag{3.17}$$

Self-attention. With average-pooling, all positions contribute equally to the final features. As an alternative, we can use self-attention to weight the $|\boldsymbol{x}|$ vectors differently:

$$\forall t \in [1..|\boldsymbol{y}|],\ \mathrm{H}^{\mathrm{out}}_{t-1} = (\mathrm{H}^{\mathrm{conv}}_{t-1})^\top \mathrm{softmax}\left(\left(\mathrm{H}^{\mathrm{conv}}_{t-1}\mathrm{W}_1\right) w_2\right), \tag{3.18}$$

with $\mathrm{W}_1 \in \mathbb{R}^{d\times d}$ and $w_2 \in \mathbb{R}^d$.

Gated max-pooling. For a fair comparison with self-attention, we prefix max-pooling with a gated-linear unit to add more weights and a non-linearity:

$$\forall t \in [1..|\boldsymbol{y}|],\ \forall k \in [1..d],\ \mathrm{H}^{\mathrm{out}}_{t-1,k} = \max_{1 \leq j \leq |\boldsymbol{x}|} \mathrm{GLU}(\mathrm{H}^{\mathrm{conv}}_{t-1,j,k}\,\mathrm{W}), \tag{3.19}$$

with $\mathrm{W} \in \mathbb{R}^{d \times 2d}$ and GLU the gated linear unit introduced in §2.6.2.

The aggregated features $\mathrm{H}^{\mathrm{out}}$ are then transformed to predictions over the output vocabulary $\mathcal{V}_y$ with a softmax-normalized linear map of weights $\mathrm{W}_p \in \mathbb{R}^{|\mathcal{V}_y| \times d}$. Thus, the probability distribution for every target token is obtained as:

$$\forall t \in [1..|\boldsymbol{y}|],\ p(y_t \,|\, \boldsymbol{y}_{<t}, \boldsymbol{x}) = \mathrm{softmax}(\mathrm{W}_p \mathrm{H}^{\mathrm{out}}_{t-1}). \tag{3.20}$$

To reduce the number of the model's parameters we can tie the weights of the input embeddings E_y with those of the final projection W_p:

$$\forall t \in [1..|\boldsymbol{y}|],\ p(y_t \,|\, \boldsymbol{y}_{<t}, \boldsymbol{x}) = \mathrm{softmax}(\mathrm{E}_y^{\top} \mathrm{H}^{\mathrm{out}}_{t-1}). \tag{3.21}$$

If the translation task uses shared vocabularies $\mathcal{V} = \mathcal{V}_x = \mathcal{V}_y$, a further reduction can be achieved by tying the three matrices E_x, E_y and W_p.

By-product alignments. If we use an attention mechanism to aggregate $\mathrm{H}^{\mathrm{conv}}$, we can look at the softmax-normalized attention scores as soft alignments between the source and target sequences:

$$\alpha_{tj} = \mathrm{softmax}\left(\left(\mathrm{H}^{\mathrm{conv}}_{t-1} \mathrm{W}_1\right) w_2\right)_j. \tag{3.22}$$

With average or max-pooling, there are no explicit alignment scores. Nonetheless, in the case of max-pooling, we can leverage the pooled positions to obtain some *implicit* alignments. In fact, for a given target position i, the max-pooling operator assigns the d channels to different source positions. Let us define a_{tj} as the number of channels that voted for x_j being aligned with y_t. An implicit alignment between $\boldsymbol{x}$, $\boldsymbol{y}$ is inferred by normalizing the counts a_{tj}:

$$a_{tj} = \left| \left\{ k \in [1..d] \,|\, j = \arg\max_{j \in [1..|\boldsymbol{x}|]} (\mathrm{H}^{\mathrm{conv}}_{t-1,j,k}) \right\} \right|, \quad \alpha_{tj} = \frac{a_{tj}}{\sum_{j=1}^{|\boldsymbol{x}|} a_{tj}}. \tag{3.23}$$

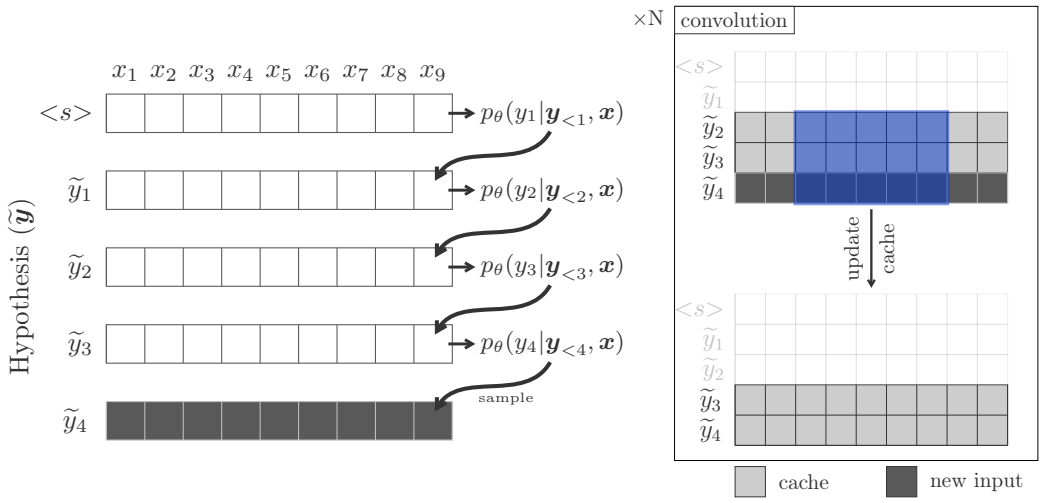


Figure 3.7: Decoding with Pervasive Attention. The left-hand panel shows the stacking of rows in the grid after every new target token. The right-hand panel illustrates the caching of activations in every two-dimensional convolution.

3.2.6 Inference

To generate a hypothesis $\widetilde{\boldsymbol{y}}$ conditioning on the source $\boldsymbol{x}$ we initiate the two-dimensional grid with a single row corresponding to the special token <s>. The ConvNet processes this single row to emit an output distribution $p_\theta(y_1 \,|\, y_{<1}, \boldsymbol{x})$ for the next token. The emitted distribution is then used either greedily or with beam-search to select a candidate $\widetilde{y}_1$ and add another row to the grid. We keep packing rows in the grid until the model decides to stop by generating the special token </s>.

If implemented naively, at each decoding step the ConvNet has to process the growing grid from scratch, making the decoding process costly in terms of computation. To alleviate this cost we cache the activations of each causal-convolution and only process the newly decoded input with a cache of the last $\left\lfloor \frac{k_y}{2} \right\rfloor$ rows. In Figure 3.7, with $k_y = 5$, we retrieve the two cached rows (right panel), to this cache, we add the recent input coming from the preceding map then convolve with the filter (in blue). Once the new convolution output is evaluated, we update the cache.

Model	RF	Computation	Sequential	$\boldsymbol{y}$-path	$\boldsymbol{x}$-path
RNN + attention (Luong et al., 2015)	∞	$O(\mathrm{N}d^2\lvert\boldsymbol{x}\rvert+\mathrm{N}d^2\lvert\boldsymbol{y}\rvert+d\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N}(\lvert\boldsymbol{x}\rvert+\lvert\boldsymbol{y}\rvert))$	$O(\lvert\boldsymbol{y}\rvert)$	$O(1)$
ConvS2S (Gehring et al., 2017)	$O(k\mathrm{N})$	$O(\mathrm{N}kd^2\lvert\boldsymbol{x}\rvert+\mathrm{N}kd^2\lvert\boldsymbol{y}\rvert+\mathrm{N}d\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N})$	$O(\lvert\boldsymbol{y}\rvert/k)$	$O(1)$
Transformer (Vaswani et al., 2017)	∞	$O\big(\mathrm{N}d\lvert\boldsymbol{x}\rvert^2+\mathrm{N}d\lvert\boldsymbol{y}\rvert^2+\mathrm{N}d\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert\big)$	$O(\mathrm{N})$	$O(1)$	$O(1)$
Pervasive Attention	$O(k\mathrm{N})$	$O(\mathrm{N}dk^2\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert+\mathrm{N}d^2\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert)$	$O(\mathrm{N})$	$O(1)$	$O(1)$

Table 3.1: Properties of Pervasive Attention compared to state-of-the-art sequence-to-sequence models.

3.2.7 Sequence-to-sequence properties

We characterized sequence-to-sequence models in §2.7 with a few properties, namely, the receptive-field (RF), the number of sequential steps, the amount of computation performed, and the $\boldsymbol{y}$-path and $\boldsymbol{x}$-path defined as the maximum length of the path from a teaching signal to a token in $\boldsymbol{y}$ or a token in $\boldsymbol{x}$ respectively. With N layers in the model and features in $\mathbb{R}^d$, Pervasive Attention with k-wide filters has a receptive field of $O(k\mathrm{N})$, N sequential steps from the stacking of layers and constant $\boldsymbol{x}$-path and $\boldsymbol{y}$-path due to the residual skip connections. In terms of computation, Pervasive Attention is more expensive to train because of the term $\mathrm{N}d^2\lvert\boldsymbol{x}\rvert\lvert\boldsymbol{y}\rvert$ induced by the linear projections of the two-dimensional grid.

3.3 Experiments

We run our first experiments on the IWSLT'14 De→En dataset (Cettolo et al., 2014) which contains transcripts of TED talks aligned at the sentence level. After ablation experiments, we validate our best models on IWSLT'15 En→Vi (Luong and Manning, 2015) and attempt to scale Pervasive Attention to large benchmarks with experiments on WMT'16 En→Ro[4] and WMT'15 De→En.[5]

[4] http://www.statmt.org/wmt16/translation-task.html
[5] http://www.statmt.org/wmt15/translation-task.html

3.3.1 Experimental Setup

IWSLT'14 De→En. We replicate the setup of Edunov et al. (2018) and remove sentences longer than 175 words and pairs with length-ratio exceeding 1.5. We train on 160K pairs, develop on 7283 held out pairs and test on TED dev2010+tst2010-2013 (6750 pairs). All data is tokenized and lower-cased using the standard scripts from the Moses toolkit (Koehn et al., 2007). For open-vocabulary translation, we segment sequences using byte pair encoding (Sennrich et al., 2016) with 10K merge operations on the merged bi-texts. This results in a German and English vocabularies of 8.8K and 6.6K types respectively.

IWSLT'15 En→Vi. We use the setup of Luong and Manning (2015). We train on 133K pairs, develop on TED tst2012 (1553 pairs) and test on TED tst2013 (1268 pairs). Apart from tokenizing the corpus, no other preprocessing step was done. The vocabularies are of 17K and 7.7K words in English and Vietnamese respectively.

WMT'16 En→Ro. We use the data provided by Lee et al. (2018) where we train on 608K pairs, develop on newsdev-2016 (1999 pairs) and report test results on newstest-2016 (1999 pairs). We use a joint source-target vocabulary of 35K BPE types.

WMT'15 De→En. The training data is taken from the Europarl corpus (Koehn, 2005), the News Commentary (Tiedemann, 2012) and the Common Crawl corpus (Smith et al., 2013). We reproduce the setup of Ma et al. (2019a); Arivazhagan et al. (2019) with a joint vocabulary of 32K BPE types. We train on 4.5M pairs, develop on newstest2013 (3000 pairs) and test on newstest15 (2169 pairs).

We report in Table 3.2 statistics relative to the length of the tokenized bi-texts for all four benchmarks.

Optimization details. To train our models we optimize label-smoothed maximum likelihood (Szegedy et al., 2016) with a smoothing rate $\epsilon = 0.1$. The parameters are updated using Adam ($\beta_1, \beta_2 = 0.9, 0.98$) with a learning rate that follows an inverse square-root schedule. We train for a total of 50K updates and evaluate with the check-pointed weights corresponding to the lowest (best) loss on the development set. We generate the model's

Dataset	Split	#Pairs	$\|\boldsymbol{x}\|$	$\|\boldsymbol{y}\|$	$\|\boldsymbol{y}\|/\|\boldsymbol{x}\|$
IWSLT'14 De→En	Train	160K	(7, 19, 58)	(7, 19, 56)	(0.8, 1.0, 1.3)
	Dev	7283	(6, 19, 59)	(7, 19, 56)	(0.8, 1.0, 1.3)
	Test	6750	(6, 19, 54)	(6, 18, 52)	(0.7, 1.0, 1.3)
IWSLT'15 En→Vi	Train	133K	(6, 16, 47)	(7, 20, 59)	(0.9, 1.2, 1.7)
	Dev	1553	(6, 19, 59)	(7, 19, 46)	(0.9, 1.2, 1.7)
	Test	1268	(6, 18, 49)	(7, 22, 61)	(0.9, 1.3, 1.7)
WMT'16 En→Ro	Train	608K	(8, 24, 53)	(8, 25, 54)	(0.8, 1.0, 1.3)
	Dev	1999	(9, 26, 59)	(9, 28, 63)	(0.8, 1.0, 1.4)
	Test	1999	(8, 24, 54)	(8, 26, 58)	(0.8, 1.1, 1.4)
WMT'15 De→En	Train	4.5M	(10, 26, 64)	(9, 25, 62)	(0.6, 1.0, 1.6)
	News-commentary	215K	(8, 29, 63)	(8, 27, 57)	(0.7, 0.9, 1.2)
	Europarl	1.9M	(8, 27, 64)	(8, 26, 62)	(0.7, 1.0, 1.3)
	Common-crawl	2.4M	(11, 26, 64)	(10, 25, 63)	(0.5, 1.0, 1.9)
	Dev	3000	(7, 24, 57)	(7, 22, 54)	(0.7, 0.9, 1.3)
	Test	2169	(7, 24, 56)	(7, 23, 53)	(0.7, 1.0, 1.3)

Table 3.2: Length and length-ratio statistics in four datasets after BPE tokenization (except from IWSLT'15 En→Vi with word tokens). For each variable we measure the percentiles at 5%, 50% and 95%.

Hyper-parameter	IWSLT'14 De→En	IWSLT'15 En→Vi	WMT'16 En→Ro	WMT'15 De→En
lr_{max}	0.002	0.002	0.001	0.0015
Warm-up T_{w}	4000	4000	4000	2000
Batch-size (tokens)	8K	8K	29K	460K
Epochs	78	100	76	140
Beam-width	5	5	5	5
Length-penalty	1	1	0.5	0.6

Table 3.3: Training and evaluation hyper-parameters for the four benchmarks. The number of epochs is equivalent to 50K updates with the selected batch-size.

hypotheses with length-penalized beam-search and measure the word-level tokenized BLEU score with respect to the references using Moses's *multi-bleu.pl*.[(6)] Training and evaluation hyper-parameters of each dataset are detailed in Table 3.3.

IWSLT'14 De→En baselines. We compare the Pervasive Attention models to three

[(6)] https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

baseline encoder-decoder models: Bi-LSTM, ConvS2S and Transformer. For a fair assessment of the different architectures, we fix a few design choices. Namely, we use input embeddings of size $d = 256$ (both source and target) and tie the target embeddings E_y with the pre-softmax projection weights W_p. After a limited search of the hyper-parameters' space, focused mainly on the depth, the kernel width and the number of attention heads when relevant, we selected the following baselines:

- Bi-LSTM: Two layers in the encoder and decoder with a dropout rate $p_{\text{drop}} = 0.3$.
- ConvS2S: 16-layers encoder and decoder with 3-wide filters and $p_{\text{drop}} = 0.3$.
- Transformer *small*: Encoder and decoder with $\mathrm{N} = 6$ blocks, $n_{\mathrm{H}} = 4$ attention heads, feed-forward network with $d_{\mathrm{FF}} = 1024$ and a dropout rate $p_{\text{drop}} = 0.3$.

IWSLT'15 En→Vi baselines. We compare to Transformer *small* ($d = 256$, $p_{\text{drop}} = 0.2$) and Transformer *base* ($d = 512$, $p_{\text{drop}} = 0.3$). We use a smaller d_{FF} for the *base* model (1024 instead of 2048) following the setup of Ma et al. (2020). With large d_{FF} the model overfits the small dataset. We tie the decoder matrices E_y and W_p in both baselines.

WMT'16 En→Ro baselines. We compare to Transformer *small* ($d = 256$) and *base* ($d = 512$) widely used for this dataset (Lee et al., 2018; Ma et al., 2019b). For both models we use a dropout rate of $p_{\text{drop}} = 0.2$. Since we are using a joint source-target dictionary, we tie the three embedding matrices $\mathrm{E}_x = \mathrm{E}_y = \mathrm{W}_p$.

WMT'15 De→En baselines. We compare to Transformer *base* ($d = 512$). We set the dropout rate to $p_{\text{drop}} = 0.2$. We tie the matrices $\mathrm{E}_x, \mathrm{E}_y$ and W_p in this benchmark as well.

3.3.2 Ablation Results

In this section we aim to study the effect of four main hyper-parameters: the ConvNet connectivity, the kernel size ($k_y \times k_x$), the depth of the model N and the aggregation operator reducing the grid after the ConvNet. To allow ourselves a broad exploration of the hyperparameter space, we only train on pairs shorter than 75 tokens.

Skip connections. First we compare each of the ConvNets to select which is better suited for Pervasive Attention. We train five Pervasive Attention models with $\mathrm{N} = 14$ layers, filters

Skip connections	#Params	PPL	BLEU(b=1)	BLEU(b=5)
Dense	**7.1M**	5.59	33.37	34.45
Dense w/o LN	**7.1M**	5.71	32.47	33.69
Residual	12.8M	5.31	33.56	34.78
Residual-norm	12.8M	5.77	32.95	33.83
Residual-cumulative	12.8M	**5.15**	34.22	**35.24**
Residual-gated	12.8M	**5.15**	**34.25**	35.20

Table 3.4: IWSLT'14 De→En development set: accuracies (greedy and beam-search decoding) and perplexity with different skip connections. All models have N=14 layers, filters in 11×11 and aggregate with max-pooling.

of size 11×11 and a simple max-pooling to aggregate the features. Each model uses one of the skip connections detailed in §3.2.3.

In the results of Table 3.4, aside from residual-norm, dense connections are outperformed by residual connections. One advantage of dense ConvNets is their small number of parameters due to the generation of a few channels per layer. In this experiment, we use a growth-rate of $g = 48$. Although each block generates only g channels, the inputs grow as we advance in the architecture. The memory bottleneck of the n-th dense layer is the first composite $\text{Conv} \circ \text{ReLU} \circ \text{Norm}$ mapping form $\mathbb{R}^{d+(n-1)g}$ to constant $\mathbb{R}^{4g}$. We alleviate the memory cost with gradient checkpointing (Chen et al., 2016) to the detriment of training speed as we double the forward-pass computational cost. Residual connections without normalization are thus the better choice for their accuracy and their manageable memory consumption. Since residual skip connections, aside from residual-norm, do not normalize their activations, we tried removing the layer-normalizations from the DenseNet as well. However, the performance deteriorates without the normalizations in each dense layer. The results of Table 3.4 also show a clear advantage of accumulating the outputs of the residual skip connections instead of predicting with their final output. Accumulating these features accelerates the convergence and explicitly links all of the features to the training signal. Mid-level features with cumulative and gated residual skip-connections are tuned down to focus on the deepest and shallowest set of features. See Appendix A for a profiling of the features in the ConvNet with varied skip connections.

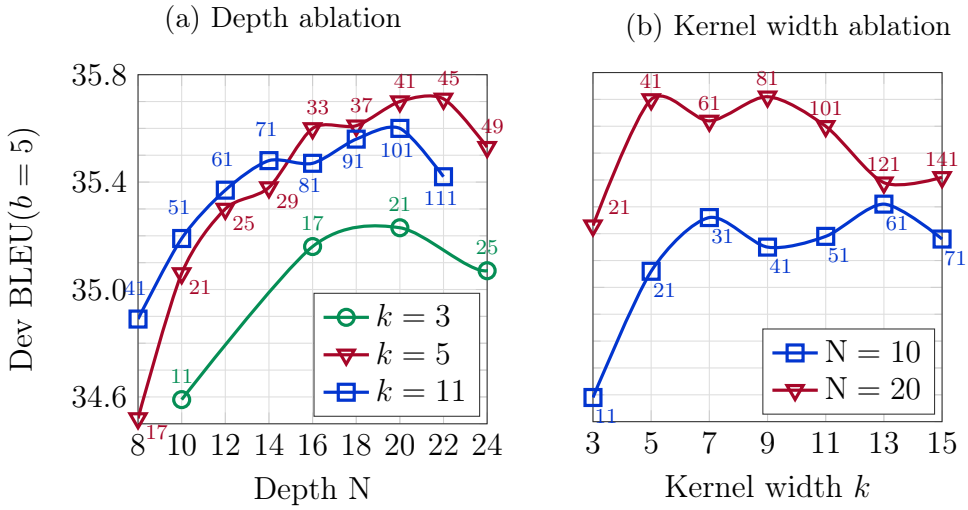


Figure 3.8: IWSLT'14 De→En development set: accuracies (beam-search decoding) with varying kernel widths k and depths N. Each marker is annotated wit the size of the target-side receptive field.

The ConvNet's receptive field. To study the effect of the ConvNet receptive field, we train models under different depths and kernel sizes. We use residual-cumulative skip connections with the max-pooling aggregator. The receptive field of these models is $1 + \mathrm{N}(k_x - 1)$ in source tokens and $1 + \mathrm{N}\lfloor k_y/2 \rfloor$ in target tokens. We only use square kernels ($k = k_x = k_y$) to limit our search space. Both k and N grow the receptive field linearly. However, the depth N has a stronger impact on the number of the ConvNet parameters. In fact, with features in $\mathbb{R}^d$ and a $4d$-wide feed-forward layer, the residual ConvNet has $k\mathrm{N}d^2 + 9\mathrm{N}d^2 + 5\mathrm{N}d$ free parameters.

Results in Figure 3.8 show that with $k = 11$ starting from a minimal depth $\mathrm{N} = 14$ with a target-side receptive field of 71 tokens, the performances plateaues then the model start overfitting the training data. On IWSLT'14 De→En, a receptive field of this size covers the full source and target sequences for all pairs except from a few outliers (see Table 3.2). The minimal $k = 3$ filters grow the receptive field one token at a time and require stacking more layers. Even with $\mathrm{N} = 24$ these models falter behind wider filters. With $k = 5$ filters, we can grow the receptive field at a slow pace and outperform shallower networks with wider filters. Take for example the three setups with an equal receptive field of 41 target tokens, $(\mathrm{N} = 8, k = 11)$, $(\mathrm{N} = 20, k = 5)$ and $(\mathrm{N} = 10, k = 9)$. The three models rank in

Aggregation	#Params	PPL	BLEU(b=1)	BLEU(b=5)
Max-pooling	12.8M	5.15	34.51	35.48
Average-pooling	12.8M	5.28	33.85	34.84
Gated max-pooling	12.9M	5.12	**34.61**	**35.65**
Self-attention	12.9M	**5.00**	34.48	35.60

Table 3.5: Performance on the development set with different aggregation mechanisms. All models have N=14 layers, filters in 11×11 and residual-cumulative skip connections.

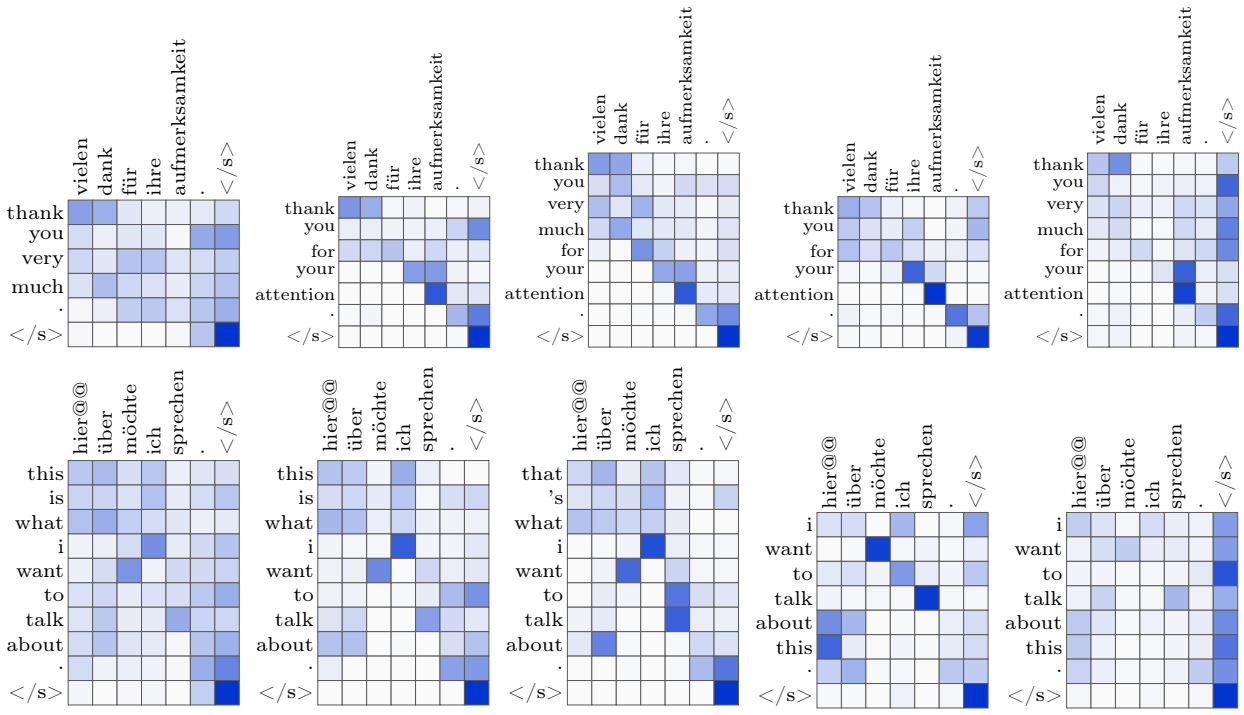
accuracy according to their depths. However, the increase in performance comes at the cost of computation and memory.

Aggregation for sequence prediction. We examine in this section the aggregation method used to reduce the ConvNet's ultimate features, $\mathrm{H}^{\mathrm{conv}}$, into fixed-size representations. We train four Pervasive Attention models with $\mathrm{N} = 14$ layers, filters in $\mathbb{R}^{11\times11}$ and residual-cumulative skip connections. In each model we reduce the source dimension differently.

Results in Table 3.5 show that between the two weightless aggregators, max and average pooling, max-pooling the source positions is the better choice. When considering parameterized aggregators, self-attention has a clear advantage in terms of perplexity. Accuracy-wise, gated max-pooling outperforms self-attention.

In the examples of Figure 3.9, the soft alignments explicitly obtained with the softmax-normalized scores of self-attention have a high entropy. On the opposite side, max-pooling yields low-entropy alignments.

To better assess the quality of the soft alignments, we encode the full development set in teacher forcing mode *i.e.* we feed the true prefix $\boldsymbol{y}_{<t}$ to predict the next target token similarly to how we measure the perplexity. Afterwards, we evaluate the entropies and coverages of the soft alignments. Whether implicit or explicit, let α_{tj} be the alignment score between y_t and x_j. For every target token, we evaluate the entropy of the probability distribution α_t denoted with $\mathrm{H}(\alpha_t)$. To evaluate the alignment's coverage, we measure the total mass assigned to each source position and denote it with $\mathrm{C}(\alpha, j)$. These measures are then averaged on the corpus level. A low entropy means the attention mechanism is confident in its consideration



(a) Self-attention (b) Max-pooling (c) Gated (d) Bi-LSTM (e) Transformer

Figure 3.9: IWSLT'14 De→En development set: Examples of alignments generated by Pervasive attention with self-attention, max-pooling and gated max-pooling aggregators. We include for reference the alignments of the Bi-LSTM and Transformer (last block) baselines.

of the source positions and a high coverage means that no source position is overlooked.

$$\mathrm{H}(\alpha_t) = -\sum_{\substack{j=1 \\ \alpha_{tj}>0}}^{|\boldsymbol{x}|} \alpha_{tj} \log(\alpha_{tj}), \quad \mathrm{H} = \frac{\sum_{\boldsymbol{y}} \sum_{t=1}^{|\boldsymbol{y}|} \mathrm{H}(\alpha_t)}{\sum_{\boldsymbol{y}} |\boldsymbol{y}|}, \tag{3.24}$$

$$\mathrm{C}(\alpha, j) = \min(1, \sum_{t=1}^{|\boldsymbol{y}|} \alpha_{tj}), \quad \mathrm{C} = \frac{\sum_{\boldsymbol{x}} \sum_{j=1}^{|\boldsymbol{x}|} \mathrm{C}(\alpha, j)}{\sum_{\boldsymbol{x}} |\boldsymbol{x}|}. \tag{3.25}$$

In Table 3.6, we verify that Pervasive Attention (PA) with self-attention has high entropy alignments compared to max-pooling. Self-attention has however a better coverage of the source sequence. For reference, we include in Table 3.6 the entropies and coverages of alignments obtained with the Bi-LSTM and Transformer baselines. Since Transformer has 6 blocks, each with an encoder-decoder attention module, we average the heads in each block and report 6 values. For an in-depth analysis of the different attention heads in a

Model	Entropy (H)	Coverage (C)
Pervasive Attention, Max-pooling	1.17	0.78
Pervasive Attention, Gated max-pooling	1.14	0.76
Pervasive Attention, Self-attention	2.06	**0.82**
Bi-LSTM	**1.11**	0.78
Transformer	2.71, 2.47, 2.79 1.73, 1.48, 1.54	0.77, 0.79, 0.79 0.75, 0.71, 0.62

Table 3.6: IWSLT'14 De→En development set: Entropies and coverages of soft alignments obtained with Pervasive Attention and two baseline encoder-decoder models.

Transformer model, we refer to Tang et al. (2018), Voita et al. (2019) and Michel et al. (2019). Generating better alignments (low entropy and high coverage) is not a quality we require in a good translation model; the alignment themselves are a by-product we did not optimize. The Bi-LSTM baseline has better alignments than Transformer but the later improves the decoding accuracy by up to 3 BLEU points (see Table 3.7).

3.3.3 Comparison to State-of-the-art

In this section, we train our Pervasive Attention (PA) models on the full training set of IWSLT'14 DE→En and compare them to the baseline encoder-decoder models in Table 3.7. Pervasive Attention has N = 14 layers, filters in 11×11, residual-cumulative skip connections and self-attention for aggregation. We train all models with three different seeds and report the mean and standard-deviation of BLEU accuracies and perplexities. With equally sized features and classifiers, Pervasive Attention outperforms the recurrent, convolutional and attention-based baselines. IWSLT'14 En→Vi being of comparable size to IWSLT'14 De→En, we train the same model on this dataset. We also train the same model on the larger benchmark of WMT'16 En→Ro. The results in Table 3.8 show that Pervasive Attention is competitive on IWSLT'15 En→Vi outperforming Transformer *base* with 30% less parameters. On WMT'16 En→Ro, Pervasive Attention achieves comparable accuracies on the test set with considerably fewer parameters than Transformer models.

Next, we move to the WMT'15 De→En benchmark. We use $d = 512$ and $d_{\text{FF}} = 2048$ similar

		Development			Test	
Model	#Params	PPL	BLEU(b=1)	BLEU(b=5)	BLEU(b=1)	BLEU(b=5)
Bi-LSTM	**8.5M**	7.55±0.02	30.48±0.08	31.71±0.08	29.71±0.18	31.03±0.16
ConvS2S	19.5M	6.74±0.03	32.44±0.08	33.55±0.01	31.51±0.06	32.66±0.05
Transformer	15.0M	5.07±0.04	34.68±0.15	35.64±0.13	33.54±0.10	34.53±0.10
Pervasive Attention	12.9M	**4.98±0.01**	**34.95±0.05**	**36.00±0.07**	**33.55±0.02**	**34.67±0.03**

Table 3.7: IWSLT'14 De→En: Performances on the development and test sets.

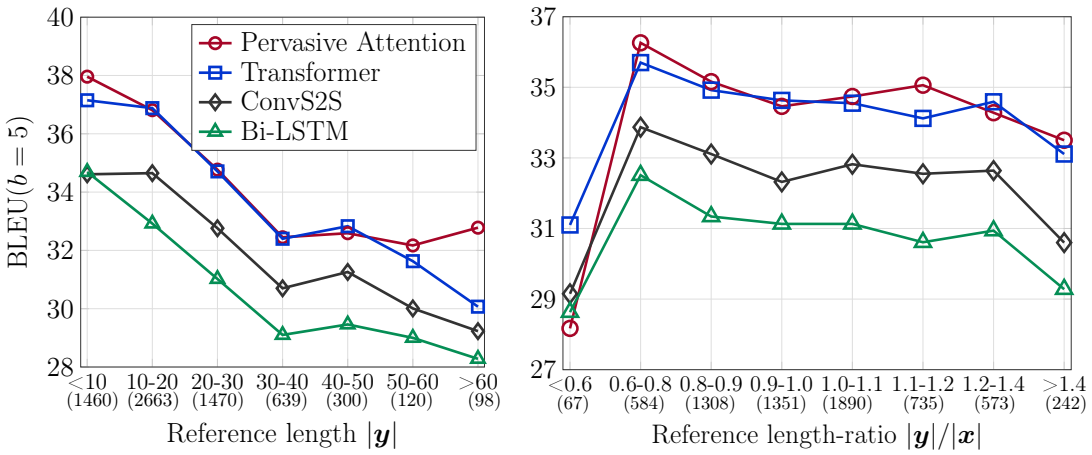
			Development		Test	
		#Params	PPL	b=1	b=5	b=1	b=5
IWSLT'15 En→Vi	Transformer *base*	44.2M	**8.23**	25.80	26.75	29.33	30.17
	Transformer *small*	17.4M	8.70	25.35	26.12	28.56	29.20
	Pervasive Attention	**15.2M**	8.64	**26.29**	**27.17**	**29.81**	**30.49**
WMT'16 En→Ro	Transformer *base*	62.0M	6.07	33.63	34.06	32.56	33.26
	Transformer *small*	20.0M	6.13	**33.83**	**34.37**	**32.95**	33.46
	Pervasive Attention	**17.9M**	**6.03**	33.69	34.25	32.87	**33.54**
WMT'15 De→En	Transformer *small*	19.3M	5.50	29.01	29.84	29.58	30.49
	Transformer *base*	60.5M	**4.38**	**30.76**	**31.55**	**31.96**	**32.80**
	Pervasive Attention	**49.0M**	5.33	28.05	29.00	28.78	29.61

Table 3.8: Performances on the development and test sets of IWSLT'15 En-Vi, WMT'16 En→Ro and WMT'15 De→En.

to Transformer *base* and train N = 16 layers with filters in $\mathbb{R}^{11\times11}$ and residual-cumulative skip connections. We aggregate the ConvNet's ultimate features with self-attention. Results in Table 3.8 show that Pervasive Attention is less successful in learning from large noisy datasets like WMT'15 De→En.

3.4 Analysis and Discussion

To diagnose the strengths and weaknesses of Pervasive Attention, we breakdown the IWSLT'14 De→En test set into buckets according to the reference length ($|\boldsymbol{y}|$) and length-ratio ($|\boldsymbol{y}|/|\boldsymbol{x}|$).



(a) BLEU score in length buckets (b) BLEU score in length-ratio buckets

Figure 3.10: IWSLT'14 De→En test set: BLEU scores in buckets of length and length-ratio. The size of each bucket is noted under the bucket label in the x-axis.

In Figure 3.10a, we note that Pervasive Attention is capable of translating both short and long sequences, successfully outperforming all three baselines. It is particularly better than the Transformer baseline in the extremity buckets $|\boldsymbol{y}| < 10$ and $|\boldsymbol{y}| > 60$. In Figure 3.10b, we see that Pervasive Attention is worse than three baselines on pairs with $|\boldsymbol{y}|/|\boldsymbol{x}| < 0.6$. Pairs in this bucket are usually mistranslated or misaligned (see examples in Table 3.9) and the model generating the shorter hypothesis scores the higher. On average, Bi-LSTM and Transformer generate shorter hypotheses with a corpus length-ratio[(7)] of 0.97 compared to the convolutional models with a ratio of 0.99. With two equally bad hypotheses, BLEU score with its brevity penalty will favor shorter ones.

The issue with outlier length-ratios is a plausible explanation for the low performance on WMT'15 De→En, particularly with the noisy ratios in common-crawl (see Table 3.2).

3.5 Related Work

The idea of building a 2D grid from parallel sequences is used in different NLP tasks especially for scoring parallel texts. This includes works on semantic matching and paraphrase

[(7)] Evaluated as the total number of generated tokens divided by the total number of reference tokens.

Source	die beiden rie@@fen mich an .
Reference ($\|\boldsymbol{y}\|/\|\boldsymbol{x}\|$=0.57)	they called me .
PA (BLEU=34.57)	the two of them called me .
TF (BLEU=53.73)	they called me up .
Source	das hat damit zu tun , dass wir zurück@@gehen in das vier@@zehn@@te jahrhundert .
Reference ($\|\boldsymbol{y}\|/\|\boldsymbol{x}\|$=0.12)	century .
PA (BLEU=10.12)	that has to do with the fact that we go back to the four@@th century .
TF (BLEU=12.57)	that has to do with us going back to the fourth century .
Source	wir hatten das vo@@gt -@@zeitalter , wir hab hatten das masch@@in@@en@@zeitalter und wir beschrei@@ten nun das so genan@@nte cyber@@-@@zeitalter .
Reference ($\|\boldsymbol{y}\|/\|\boldsymbol{x}\|$=0.31)	the question is , what will come next ?
PA (BLEU=6.13)	we had the vo@@cal age , we 've had the machine age , and we 're now descri@@bing the so-@@called cyber@@-@@age .
TF (BLEU=6.43)	we had this bird age , we had the machine age , and we now described this called cy@@ber age .

Table 3.9: IWSLT'14 De→En: Misaligned or mistranslated samples with outlier length-ratios from the test set.

identification (Hu et al., 2014; He and Lin, 2016; Wan et al., 2016; Levy and Wolf, 2017). For semantic matching, Hu et al. (2014) apply 1D convolutions to each sequence separately before a series of 2D convolutions and max-poolings on a grid of the two sequences. The convolutions are then followed by a feed-forward network to estimate the matching score. They interestingly highlighted the desirable property of letting the sequences *meet* before their representations mature. He and Lin (2016); Wan et al. (2016) first encode the sequences with Bi-LSTMs then evaluate pairwise similarities between the words of the two sequences to build an interaction grid. While He et al. (2016a) process the grid with a two-dimensional ConvNet, Wan et al. (2016) directly use k-max pooling to aggregate and then score the pair. Similarly, for sequence alignment, Levy and Wolf (2017) use LSTM hidden states as token representations and, similar to our work, concatenate pairwise representations and feed their input grid to a 2D ConvNet followed by a softmax to estimate soft-alignment probabilities. Recently in question-answering, Raison et al. (2018) weaved two Bi-LSTMs, one along the context dimension and the other along the question dimension in order to identify a response span in the context.

More related to our work on machine translation, Kalchbrenner et al. (2016a) proposed the *reencoder* network where a Grid LSTM processes both sequences along its first and second dimension, allowing the model to re-encode the source sequence as it advances along the

target dimension. They also observed that such a structure implements an implicit form of attention. Wu et al. (2018) used a ConvNet over the 2D source-target representation, but only as a discriminator in an adversarial training setup. Similar to semantic matching models, they do not use masked convolutions, since their ConvNet is used to predict if a given source-target pair is a human or machine translation. Concurrently with our work, Bahar et al. (2018) used a 2D-LSTM layer to jointly process the source and target sequences with a similar two-dimensional layout. He et al. (2018) proposed a Transformer with layer-wise coordination where the source and target are concatenated into a single sequence that is processed by the same stack of Transformer blocks. The sub-block with encoder-decoder interaction is removed and the attention over source tokens is integrated in self-attention. Fonollosa et al. (2019) also proposed a joint processing of the source-target concatenation with additional locality constraints in self-attention. With this joint encoding, the decoder has access to the source features at various levels of abstraction instead of only attending to high-level source representations.

Recently, Wu et al. (2019) revisited convolutional sequence-to-sequence models and proposed light-weight and dynamic convolutions. With light-weight convolutions, self-attention in a Transformer block is replaced with depth-wise separable uni-dimensional convolutions. Those convolutions are then made dynamic by predicting input-dependent kernels at every time-step. Wu et al. (2019) use the same number of blocks as in Transformer models and progressively widen the convolution filters up to $k = 31$. Some incremental updates of the Transformer model include Zhao et al. (2019a) where only top-k scored positions are selected in self-attention to reduce the alignments' entropy, Lu et al. (2019) where each self-attention module is surrounded by two feed-forward networks and Zhao et al. (2019b) where both self-attention and depth-wise separable convolutions are used within each block. We list in Table 3.10 the results of recent works that experimented with IWSLT'14 De→En.

	BLEU	#Params	$\lvert\mathcal{V}_{\text{EN}}\rvert$	d
MUSE (Zhao et al., 2019b)	**36.30**	-	32K	512
Top-k Attention(Zhao et al., 2019a)	35.60	-	32K	512
Macaron Transformer (Lu et al., 2019)	35.40	42.8M	6.6K	512
Local joint self-attention (Fonollosa et al., 2019)	35.70	26.8M	31K	256
Join self-attention (He et al., 2018)	35.07	19.1M	31K	512
DynamicConv (Wu et al., 2019)	35.20	35.3M	6.6K	512
LightConv (Wu et al., 2019)	34.80	34.7M	6.6K	512
Varational attention (Deng et al., 2018)	33.69	57.0M	8.9K	512
ConvS2S Risk[S] (Edunov et al., 2018)	32.84	8.8M	8.9K	256
Transformer *small*	34.53	15.0M	6.6K	256
Pervasive Attention	34.72	14.1M	6.6K	256

Table 3.10: State-of-the-art accuracies on IWSLT'14 De→En test set. For each model we report the total number of free parameters and the size of the final classifier ($\mathbb{R}^d \rightarrow \mathbb{R}^{|\mathcal{V}_{EN}|}$).

	BLEU	#Params	$\lvert\mathcal{V}_{\text{Ro}}\rvert$	d
Ma et al. (2019b) - Transformer *base* - 5 ckps	32.92	$5 \times$ 44.2M	40K	512
Lee et al. (2018) - Transformer *base*	32.40	44.2M	40K	512
Gu et al. (2017a) - Transformer *base*	31.91	44.2M	40K	512
Gehring et al. (2017)	30.02	111.5M	40K	512
Transformer *base*	33.26	62.0M	35K	512
Transformer *small*	33.46	20.0M	35K	256
Pervasive Attention	**33.54**	17.9M	35K	256

Table 3.11: State-of-the-art accuracies on WMT'16 En→Ro test set (newstest-2016).

	BLEU	#Params	$\lvert\mathcal{V}_{\text{Vi}}\rvert$	d
Provilkov et al. (2019) - Transformer *base*	**33.27**	48.2M	BPE-4K	512
Nguyen and Salazar (2019) - Transformer *base*	32.79	52.3M	BPE-8K	512
Luong et al. (2015)	26.90	-	Words-7.7K	500
Transformer *base*	30.17	44.2M	Words-7.7K	512
Pervasive Attention	30.49	15.2M	Words-7.7K	256

Table 3.12: State-of-the-art accuracies on IWSLT'15 En→Vi test set (TED tst2013).

	BLEU	#Params	$\lvert\mathcal{V}_{\text{En}}\rvert$	d
Arivazhagan et al. (2019) - RNMT+	30.50	-	32K	512
Ma et al. (2019a) - Transformer *base*	31.00	-	32K	512
Lee et al. (2017) - convolutional	25.83	-	32K	512
Firat et al. (2016) - LSTM	24.20	-	30K	1200
Transformer *base*	32.80	60.5M	32K	512
Pervasive Attention	29.61	49.0M	32K	512

Table 3.13: State-of-the-art accuracies on WMT'15 De→En test set (newstest-2015).

3.6 Conclusion

In this chapter we presented our work on Pervasive Attention, a sequence-to-sequence model to jointly encode the source and target sequences with a two-dimensional ConvNet. With this joint encoding, the source and target sequences meet at an early stage of encoding allowing each of them to influence the encoding of the other. We proposed different design choices for the core ConvNet focusing on the impact of the design on the receptive field and the abstraction level of the final features. Different methods to aggregate the final features were examined as to their effect on the translation quality and the informativeness of the explicit source-target alignments. Experiments on IWSLT'14 De→En show a clear advantage of a gradual and slow increase of the receptive field with small 5×5 filters and the importance of accumulating features coming from different encoding levels. Comparison to state-of-the-art encoder-decoder models on IWSLT'14 De→En, IWSLT'15 En→Vi and WMT'16 En→Ro prove the competitiveness of Pervasive Attention models. On the large and noisy WMT'15 De→En benchmark, Pervasive Attention show some sensitivity to misaligned training pairs. Future works on Pervasive Attention models should focus on solving this issue by means of data filtering or curriculum learning.

Chapter 4

Online Neural Machine Translation

In this chapter, we propose a framework for online machine translation where a sequence-to-sequence model alternates between reading the source and writing the target. We build on existing wait-k decoding models (Dalvi et al., 2018; Ma et al., 2019a) and prove them to be a strong baseline that can match state-of-the-art dynamic models. We propose to use Pervasive Attention models with their characteristic two-dimensional decoding grid for the task of online translation. With an oracle based on dynamic programming and likelihood scores, we outperform Transformer-MILk (Ma et al., 2020) on the IWSLT'15 En→Vi benchmark.

Some of the contributions presented in this chapter are published in:

> Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020a. Improved Training Techniques for Online Neural Machine Translation

Contents

4.1 Introduction 64
4.2 A Common Framework for Online Decoding 65
4.3 Sequence-to-Sequence models for Online Decoding 67
4.4 Online Decoding Agents 72
4.5 Experiments 78
4.6 Related Work 95
4.7 Conclusion 97

4.1 Introduction

Sequence-to-Sequence models used in the previous chapters read the entire source sequence $\boldsymbol{x}$ before decoding the target $\boldsymbol{y}$. Target tokens are produced one at a time conditioned on the full source and the previously decoded prefix. This type of conditioned generation is dubbed *offline* decoding. In this chapter we move to *online* decoding. Online (equivalently simultaneous or real-time) decoding alternates between reading source tokens for context and writing target tokens. This type of decoding is desirable for applications such as real-time speech translation (Fügen et al., 2007) and as-you-type machine translation. In such scenarios, the translation model or the interpreter must emit a translation in the target language while simultaneously listening to or reading and comprehending the source input.

In an offline translation task, models are only evaluated for the quality of their outputs (*e.g.* with BLEU). Online models, on the other hand, are required to strike a balance between translation quality and latency or delay (Mieno et al., 2015). Online decoding with partial contexts is extremely challenging and a lower latency generally means decoding with smaller, insufficient contexts, therefore, outputting low quality translations. Nonetheless, when looking at the attention maps of an offline model, we notice that the decoder is only attending to a subset of the source segments and that an online decoding strategy could be devised without degrading the translation quality. In the example of Figure 4.1, the alignment is only monotonic in the first half but we can easily find an appropriate decoding path that reads the context aligned with the target token before decoding it.

In this chapter, we propose in §4.2 a universal framework for online sequence-to-sequence models with a latent variable $\boldsymbol{z}$ modeling the growing context sizes. We model $\boldsymbol{z}$ with an agent and we distinguish between two variants: *deterministic* and *dynamic*. Deterministic agents include the recently proposed wait-k models (Dalvi et al., 2018; Ma et al., 2019a) that first read k tokens from the source then alternate between producing a target token and reading another source token. In contrast to the rigid wait-k decoding paths, dynamic agents are data-adaptive and dynamically switch between reading the source and writing the target. We adapt two offline sequence-to-sequence models for the task of online translation, attention-based Transformer §4.3.1 and our own convolutional Pervasive Attention §4.3.2. We

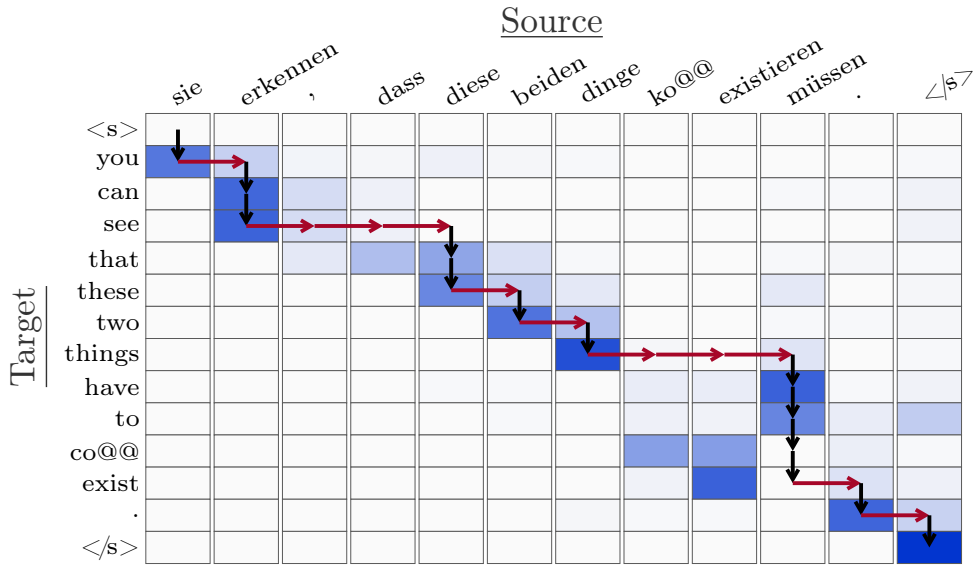


Figure 4.1: Example of an online decoding path on top of the attention map of a Pervasive Attention model.

will investigate in §4.4.1 different training objectives and encoding strategies to improve the wait-k agents and we will introduce in §4.4.2 dynamic agents devised for Pervasive Attention models. In §4.5, we experiment on IWSLT'14 De↔En, IWSLT'15 En↔Vi and the larger WMT'15 De↔En. At last, we will discuss recent works related to ours in §4.6.

4.2 A Common Framework for Online Decoding

Let $(\boldsymbol{x}, \boldsymbol{y})$ be a pair of source-target sequences of respective lengths $|\boldsymbol{x}|$ and $|\boldsymbol{y}|$. Online decoding consists of executing a sequence of interleaved reading and writing operations, consuming tokens from the source $\boldsymbol{x}$ and producing tokens for the target $\boldsymbol{y}$. Let $\boldsymbol{z}$ be the sequence of context sizes where z_t is the number of source tokens read when decoding y_t. The decoder predicts y_t conditioning on the target prefix $\boldsymbol{y}_{<t}$ and the partial source context $\boldsymbol{x}_{\leq z_t}$. We will refer to $\boldsymbol{z}$ as the decoding path. These decoding paths can be represented on a $|\boldsymbol{y}| \times |\boldsymbol{x}|$ grid, where we advance from the top left to the bottom right in a total of $|\boldsymbol{x}|$ read steps and $|\boldsymbol{y}|$ write steps (see Figure 4.1).

A translation sequence-to-sequence model with parameters θ will estimate $p_\theta(\boldsymbol{y} \,|\, \boldsymbol{x}, \boldsymbol{z})$

and an *agent* with parameters φ will model $p_\varphi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$, the likelihood of a decoding path $\boldsymbol{z}$ given the inputs $(\boldsymbol{x}, \boldsymbol{y})$. The conditional probability $p_\theta(\boldsymbol{y} \mid \boldsymbol{x})$ conventionally modeled by sequence-to-sequence models is now modified to include the variable $\boldsymbol{z}$ with:

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{z}) = \prod_{t=1}^{|\boldsymbol{y}|} p_\theta(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq z_t}, \boldsymbol{z}_{\leq t}). \tag{4.1}$$

Let $\mathcal{Z}$ be the set of possible decoding paths. To evaluate the likelihood of $\boldsymbol{y}$ given $\boldsymbol{x}$ we simply marginalize the joint probability $p_{\theta,\varphi}(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$ over $\mathcal{Z}$:

$$\log p_{\theta,\varphi}(\boldsymbol{y} \mid \boldsymbol{x}) = \log \sum_{\boldsymbol{z} \in \mathcal{Z}} p_{\theta,\varphi}(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}). \tag{4.2}$$

For any choice of a distribution q, we can lower-bound the likelihood with:

$$\log p_{\theta,\varphi}(\boldsymbol{y} \mid \boldsymbol{x}) \geq \mathrm{LB}(\boldsymbol{x}, \boldsymbol{y}, q) \tag{4.3}$$

$$\mathrm{LB}(\boldsymbol{x}, \boldsymbol{y}, q) = \log p_{\theta,\varphi}(\boldsymbol{y} \mid \boldsymbol{x}) - \mathrm{D}_{\mathrm{KL}}(q(\boldsymbol{z}) \,\|\, p_\varphi(\boldsymbol{z} \mid \boldsymbol{y}, \boldsymbol{x})) \tag{4.4}$$

$$= \mathrm{H}(q) + \sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log p_{\theta,\varphi}(\boldsymbol{z}, \boldsymbol{y} \mid \boldsymbol{x}), \tag{4.5}$$

where D_{KL} is the KL-divergence and H is the entropy. With the Expectation-Maximization (EM) algorithm (Dempster et al., 1977), we will alternate between computing the lower bound LB in the E-step and maximizing this bound in the M-step. The tightest lower-bound in the E-step is the one minimizing the KL-divergence $\mathrm{D}_{\mathrm{KL}}(q(\boldsymbol{z}) \,\|\, p_\varphi(\boldsymbol{z} \mid \boldsymbol{y}, \boldsymbol{x}))$ *i.e.* we estimate $q(\boldsymbol{z}) = p_\varphi(\boldsymbol{z} \mid \boldsymbol{y}, \boldsymbol{x})$ given the current model parameters. In the M-step we hold q fixed and solve the following optimization problem:

$$\theta^*, \varphi^* = \arg\max_{\theta,\varphi} \mathrm{LB}(\boldsymbol{x}, \boldsymbol{y}, q) \tag{4.6}$$

$$= \arg\max_{\theta,\varphi} \mathrm{H}(q) + \sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log p_{\theta,\varphi}(\boldsymbol{z}, \boldsymbol{y} \mid \boldsymbol{x}). \tag{4.7}$$

The entropy $\mathrm{H}(q)$ being a constant w.r.t. θ and φ, we maximize the Q-function or the expected complete data log-likelihood:

$$\mathrm{Q}(\boldsymbol{x}, \boldsymbol{y}, q; \theta, \varphi) = \sum_{\boldsymbol{z}} q(\boldsymbol{z}) \log p_{\theta,\varphi}(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x}). \tag{4.8}$$

To factorize the complete data log-likelihood $\log p_{\theta,\varphi}(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$, we will formulate the decoding path as a sequence of READ/WRITE actions. Each sub-path $(t, z_t) \rightarrow (t{+}1, z_{t+1})$ corresponds

to $z_{t+1} - z_t$ READs followed by a single WRITE. We will denote with ρ_{tj} the probability of reading another token at position (t, j).

$$\rho_{tj} = p_\varphi(\text{READ} \mid \boldsymbol{x}_{\leq j}, \boldsymbol{y}_{\leq t}). \tag{4.9}$$

With these binary decisions along the way, we will estimate the transition probability $p_\varphi(z_{t+1} \mid z_t)$ and factorize $\log p_\varphi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y})$ as:

$$p_\varphi(z_{t+1} \mid z_t) = \left((1 - \rho_{tz_{t+1}}) \prod_{l=z_t}^{z_{t+1}-1} \rho_{tl}\right) [\![z_{t+1} \geq z_t]\!], \tag{4.10}$$

$$\log p_\varphi(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{|\boldsymbol{z}|-1} \log \left((1 - \rho_{tz_{t+1}}) \prod_{l=z_t}^{z_{t+1}-1} \rho_{tl}\right) \tag{4.11}$$

$$= \sum_{t=1}^{|\boldsymbol{z}|-1} \left(\log(1 - \rho_{tz_{t+1}}) + \sum_{l=z_t}^{z_{t+1}-1} \log \rho_{tl}\right). \tag{4.12}$$

Ultimately the complete data log-likelihood is developed as:

$$\log p_{\theta,\varphi}(\boldsymbol{y}, \boldsymbol{z}|\boldsymbol{x}) = \sum_{t=1}^{|\boldsymbol{y}|} \log p_\theta(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq z_t}, \boldsymbol{z}_{\leq t}) + \sum_{t=1}^{|\boldsymbol{z}|-1} \left(\log(1 - \rho_{tz_{t+1}}) + \sum_{l=z_t}^{z_{t+1}-1} \log \rho_{tl}\right). \tag{4.13}$$

In the following, we will describe how we modify translation models to decode along a path $\boldsymbol{z}$ and estimate the terms $p_\theta(y_t \mid \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq z_t})$. In §4.3.1, we start with Transformer models and by extension any encoder-decoder model with an attention interface, then we move to Pervasive Attention models in §4.3.2.

4.3 Sequence-to-Sequence models for Online Decoding

To adapt sequence-to-sequence models for decoding along a path $\boldsymbol{z}$ we need to provide an encoding of the partially read source $\boldsymbol{x}_{\leq z_t}$ at every time-step t. We first describe our adaptation of the attention-based Transformer and then our own convolutional Pervasive Attention model.

4.3.1 Online Transformer

An encoder block of Transformer models consists of a multi-head self-attention followed by a point-wise feed-forward network. The self-attention module of an encoder block is bi-directional *i.e.* the state at a given position includes signals from past as well as future time-steps. Transformer wait-k models introduced in Ma et al. (2019a) and used in Zheng et al. (2019b,a) maintain the bi-directionality of the encoder and re-encode the source context after each new READ. Recall that an attention module updates its input states $\mathrm{S} = (s_1, \dots, s_{|\boldsymbol{x}|})$ of d-dimensional features with:

$$\text{Attention}(\mathrm{S}) = (\mathrm{W_V S}) \cdot \text{softmax}\left(\frac{1}{\sqrt{d_h}} (\mathrm{W_K S})^\top (\mathrm{W_Q S})\right), \tag{4.14}$$

where $(\mathrm{W_K}, \mathrm{W_V}, \mathrm{W_Q})$ are the weights of the key, value and query mappings. If we drop the matrix form, then the i-th state is updated as:

$$\text{Attention}(\mathrm{S})_i = \sum_{j=1}^{|\boldsymbol{x}|} \alpha_{ij} \mathrm{W_V} s_j, \quad \alpha_{ij} = \text{softmax}(e_i)_j, \quad e_{ij} = \frac{1}{\sqrt{d}} (\mathrm{W_K} s_j)^\top (\mathrm{W_Q} s_i). \tag{4.15}$$

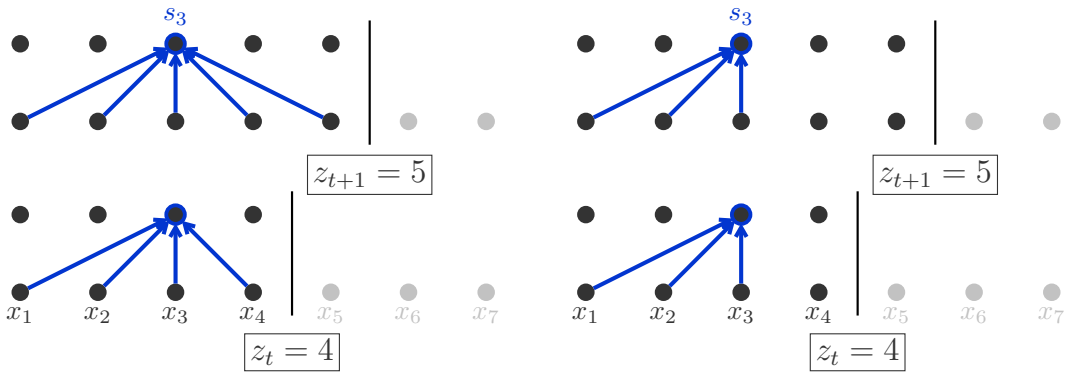
In bi-directional wait-k models, given z_t, the attention energies e_{ij} are modified with the following M^{z_t} mask:

$$\alpha_{ij} = \text{softmax}(e_i + \mathrm{M}_i^{z_t})_j, \quad \mathrm{M}_{ij}^{z_t} = \begin{cases} 0, & i, j \leq z_t \\ -\infty, & \text{otherwise.} \end{cases} \tag{4.16}$$

With this masking, the attention weights α_{ij} are zeroed out for any position beyond z_t. For the encoder to remain bi-directional every new source token that we read has to be injected in all the past source states. This means that we need to re-encode the source every time we read a new token (see an illustration of the computational graph in Figure 4.2a).

To avoid multiple forward passes for each value of z_t, we simply use a uni-directional encoder (see Figure 4.2b). The energy masking to make the encoder uni-directional is similar to how we mask the decoder for auto-regressive generation (§2.4.3). This mask is independent from the context size z_t:

$$\alpha_{ij} = \text{softmax}(e_i + \mathrm{M}_i)_j, \quad \mathrm{M}_{ij} = \begin{cases} 0, & j \leq i \\ -\infty, & \text{otherwise.} \end{cases} \tag{4.17}$$



(a) Bi-directional online encoder (b) Uni-directional online encoder

Figure 4.2: Computational graph of s_3 in the encoder of an Online Transformer between two consecutive steps with different context sizes.

For the task of online translation, the model has to simultaneously read and write. In this paradigm, we argue that the model need not remember the past source tokens that were partially translated. To test this hypothesis, we consider a locally constrained uni-directional encoder. This encoder only attends to the last Δ tokens instead of the full history:

$$\alpha_{ij} = \text{softmax}(e_i + \mathrm{M}_i^{\Delta})_j, \quad \mathrm{M}_{ij}^{\Delta} = \begin{cases} 0, & i - \Delta < j \leq i \\ -\infty, & \text{otherwise.} \end{cases} \tag{4.18}$$

In the decoder side, at every time-step t, the encoder-decoder attention modules are fed the available z_t source representations $(s_1, \dots s_{z_t})$. With h_{t-1} the current decoder state at a given block, the attention energies are evaluated as:

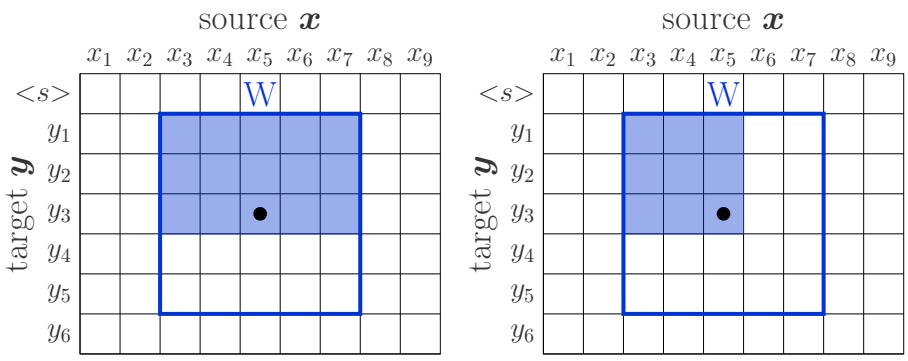
$$\forall j \in [1..z_t], \quad e_{tj} = \frac{1}{\sqrt{d}} (\mathrm{W_K} s_j)^\top (\mathrm{W_Q} h_{t-1}). \tag{4.19}$$

With the source signal truncated to $\boldsymbol{x}_{\leq z_t}$ in the computational graph of the decoder hidden state h_{t-1}, we can predict the next token output probability as usual with:

$$p(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq z_t}) = \text{softmax}(\mathrm{W}_p h_{t-1}). \tag{4.20}$$

4.3.2 Online Pervasive Attention

In Pervasive Attention models, the encoded context is managed with two-dimensional convolutions. We have discussed in §3.2.2 how the filters are masked to stop leakage from



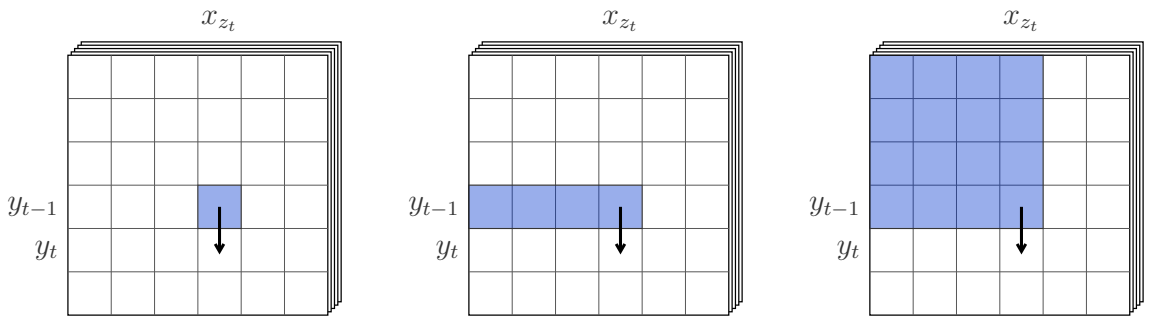
(a) Bi-directional (b) Uni-directional

Figure 4.3: Illustration of a causal two-dimensional convolution with filter $\mathrm{W} \in \mathbb{R}^{5\times5}$. At the position marked with •, the convolution only includes signals from the highlighted blue area. The rest of W is zeroed out.

future target positions. In a similar fashion, we mask the filters in the source direction as well, in order for the encoding to be uni-directional (see Figure 4.3 for an illustration of the bi-directional and uni-directional convolution filters).

Stacking N convolutions with $k \times k$ filters progressively grows the receptive field. In online Pervasive Attention, the ConvNet's ultimate features $\mathrm{H}^{\mathrm{conv}}$ at a given position (t, j) encode $\mathrm{N}\,\lfloor k/2 \rfloor$ source tokens preceding x_j and $\mathrm{N}\,\lfloor k/2 \rfloor$ target tokens preceding y_t. With that, a joint encoding of $\boldsymbol{x}_{\leq z_t}$ and $\boldsymbol{y}_{<t}$ can be aggregated from the partial grid $\mathrm{H}^{\mathrm{conv}}_{<t,\leq z_t}$ *i.e.* cells in the top-left of position $(t-1, z_t)$.

Unlike offline Pervasive Attention where we need a single representation per target position, in the online task, we would like to make a prediction at every position where the model would be asked to write. When predicting y_t at position $(t-1, z_t)$ of the grid, the straightforward solution would be to make a prediction based on the cell activations $\mathrm{H}^{\mathrm{conv}}_{t-1,z_t}$ (see Figure 4.4a). However, as we did in the offline model, we could consider the activations of all the source history $\boldsymbol{x}_{\leq z_t}$. This amounts to aggregating the positions to the left of $(t-1, z_t)$ *i.e.* $\mathrm{H}^{\mathrm{conv}}_{t-1,\leq z_t}$ (see Figure 4.4b). Pushed to the limit, we could aggregate all of the *past* positions on both axes *i.e.* $\mathrm{H}^{\mathrm{conv}}_{<t,\leq z_t}$ (see Figure 4.4c). Initial experiments with cell, row and grid features show a better accuracy with predicting the next token with an aggregate of the row features (Figure 4.4b). Similar to the offline model, we could explore different methods to evaluate the aggregate denoted with $\mathrm{H}^{\mathrm{out}}$.



(a) Cell prediction (b) Row prediction (c) Grid prediction

Figure 4.4: Position-wise aggregation of features in Pervasive Attention models. At every position $(t-1, z_t)$ we consider three different areas to aggregate in order to predict the next target token.

Max-pooling. In this, we simply max-pool the z_t vectors *i.e.* for every channel $k \in [1..d]$, we select the most active position:

$$\forall t \in [1..|\boldsymbol{y}|],\ \forall k \in [1..d],\ \mathrm{H}^{\mathrm{out}}_{t-1,z_t,k} = \max_{1 \leq j \leq z_t} \mathrm{H}^{\mathrm{conv}}_{t-1,j,k}. \tag{4.21}$$

Gated max-pooling. We prefix max-pooling with a gated-linear unit:

$$\forall t \in [1..|\boldsymbol{y}|],\ \forall k \in [1..d],\ \mathrm{H}^{\mathrm{out}}_{t-1,z_t,k} = \max_{1 \leq j \leq z_t} \mathrm{GLU}(\mathrm{H}^{\mathrm{conv}}_{t-1,j,k}\,\mathrm{W}), \tag{4.22}$$

with $\mathrm{W} \in \mathbb{R}^{d \times 2d}$ and GLU the gated linear unit introduced in §2.6.2.

Self-attention. We can use self-attention to weight the z_t vectors of the context:

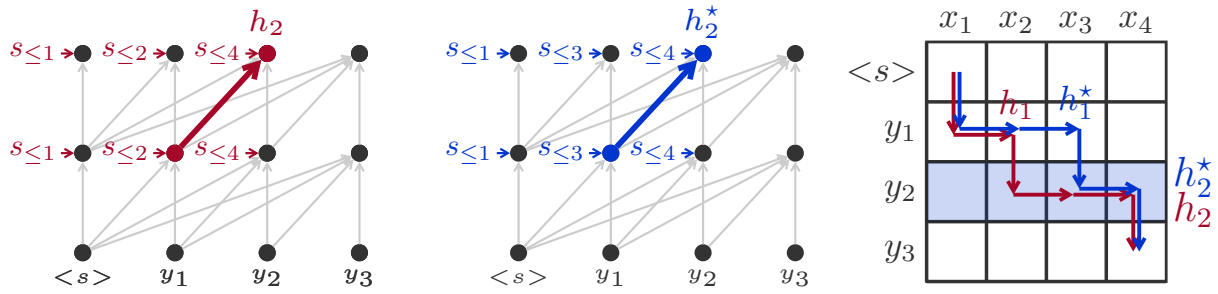
$$\forall t \in [1..|\boldsymbol{y}|],\ \mathrm{H}^{\mathrm{out}}_{t-1,z_t} = (\mathrm{H}^{\mathrm{conv}}_{t-1,\leq z_t})^\top \mathrm{softmax}\left(\left(\mathrm{H}^{\mathrm{conv}}_{t-1,\leq z_t}\mathrm{W}_1\right) w_2\right), \tag{4.23}$$

with $\mathrm{W}_1 \in \mathbb{R}^{d \times d}$ and $w_2 \in \mathbb{R}^d$.

The aggregated features $\mathrm{H}^{\mathrm{out}}$ are then transformed to predictions over the output vocabulary $\mathcal{V}_y$ with a softmax-normalized linear map of weights $\mathrm{W}_p \in \mathbb{R}^{|\mathcal{V}_y| \times d}$. The output probability for y_t after reading z_t source tokens is:

$$p(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq z_t}) = \mathrm{softmax}(\mathrm{W}_p \mathrm{H}^{\mathrm{out}}_{t-1,z_t}). \tag{4.24}$$

One major difference between Pervasive Attention and Transformer architectures is that in Pervasive Attention, the output hidden state at a given position (t, z_t) is independent



(a) Transformer (b) Pervasive Attention

Figure 4.5: Following two different paths leading to $(y_3, z_t = 4)$, the hidden state h_2 used to predict y_3 is dependent on the path with a Transformer decoder ($h_2 \neq h_2^\star$). With Pervasive Attention, the state at given position is independent from the path leading to it ($h_2 = h_2^\star$).

from the decoding path $\boldsymbol{z}$. To evaluate the output state $\mathrm{H}^{\mathrm{out}}_{t,z_t}$, we will always aggregate the row $\mathrm{H}^{\mathrm{conv}}_{t,\leq z_t}$ (see Figure 4.5b). In the receptive field of the aggregated area we match the tokens of the target prefix $\boldsymbol{y}_{<t}$ to the tokens of the source context read so far $\boldsymbol{x}_{\leq z_t}$ *i.e.* we implicitly re-encode the prefix $\boldsymbol{y}_{<t}$ to account for the previously unread source tokens. With Transformer we can end up with different states depending on the followed path. This is due to the interleaving of encoder-decoder interaction and self-attention in a stack of N decoder blocks. The encoder-decoder interaction injects a source context into the early decoder hidden states and self-attention updates these contextualized hidden states (see colored edges in Figure 4.5a).

4.4 Online Decoding Agents

As introduced in §4.2, an agent estimates the likelihood of a decoding path $\boldsymbol{z}$ by emitting reading probabilities $p_\varphi(\mathrm{READ} \mid \boldsymbol{x}_{\leq j}, \boldsymbol{y}_{\leq t})$ as we advance in the decoding grid. We will consider two types of agents, deterministic where we follow the same path for any input $(\boldsymbol{x}, \boldsymbol{y})$, and dynamic where we estimate ρ_{tj} given the available prefixes from $\boldsymbol{x}$ and $\boldsymbol{y}$. Depending on the agent type, we can decide on the estimation of q in the E-step and simplify the Q-function to train the agent and the underlying translation model.

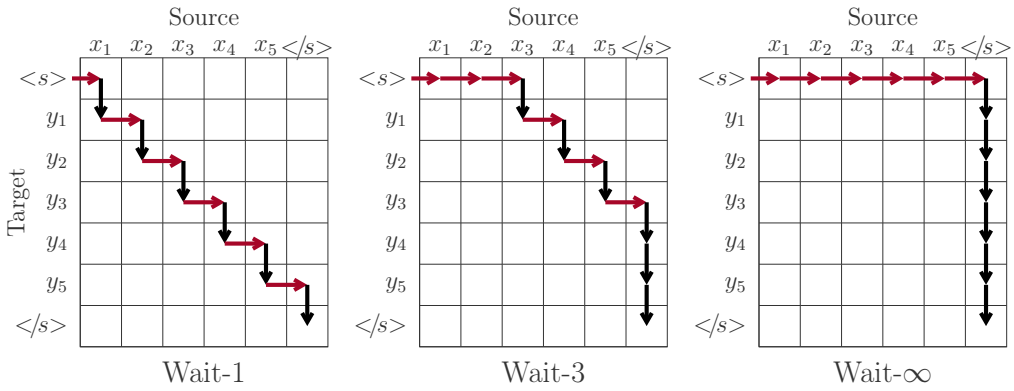


Figure 4.6: Wait-k decoding as a sequence of reads (red) and writes (black) over a source (horizontal) and target (vertical) grid. After reading the first k source tokens, the decoder alternates between reads and writes. In Wait-∞, also called Wait-until-End (WUE), decoding the source is fully read before any write operation.

4.4.1 Deterministic Agents

For our deterministic agents we use *wait-k* paths (Dalvi et al., 2018; Ma et al., 2019a). We start by reading k source tokens then we alternate between reading and writing a single token at a time, until either the full source has been read, or the target generation has been terminated. The latent variable $\boldsymbol{z}^{\text{wait-}k}$ and its associated READ/WRITE decisions for a wait-k path are evaluated as:

$$\forall t \in [1..|\boldsymbol{y}|], \quad z_t^{\text{wait-}k} = \text{clamp}_{[1..|\boldsymbol{x}|]}(k + t - 1)\,, \tag{4.25}$$

$$\forall (t, j) \in [1..|\boldsymbol{y}|-1] \times [1..|\boldsymbol{x}|-1],\ \rho_{tj} = p_{\text{wait-}k}(\text{READ} \mid \boldsymbol{x}_{\leq j}, \boldsymbol{y}_{\leq t}) = [\![j < z_t^{\text{wait-}k}]\!], \tag{4.26}$$

where $\text{clamp}_{[a,b]}(x) = \max(a, \min(x, b))$. Examples of wait-$k$ paths are shown in Figure 4.6. The offline model is recovered with $k = \infty$ so that we read the full source sequence before decoding.

To train online models with deterministic decoding paths, the EM algorithm is reduced to the M-step where q is the prior distribution on $\mathcal{Z}$:

$$\text{Q}(\boldsymbol{x}, \boldsymbol{y}, q; \theta) = \mathbb{E}_q[\log p_\theta(\boldsymbol{y} \mid \boldsymbol{z}, \boldsymbol{x})]\,. \tag{4.27}$$

As the prior on $\mathcal{Z}$, we consider either a single wait-k path as in Ma et al. (2019a) or multiple paths.

Optimizing a single decoding path. In this setup, we optimize a single wait-k decoding path and so the prior is a Dirac distribution centered on $\boldsymbol{z}^{\text{wait-}k}$ *i.e.* $q(\boldsymbol{z}) = [\![\boldsymbol{z} = \boldsymbol{z}^{\text{wait-}k}]\!]$. The Q-function is simply the decoding loss along $\boldsymbol{z}^{\text{wait-}k}$:

$$\mathrm{Q}(\boldsymbol{x}, \boldsymbol{y}, q; \theta) = \log p_\theta(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{z}^{\text{wait-}k}) = \sum_{t=1}^{|\boldsymbol{y}|} \log p_\theta(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq z_t^{\text{wait-}k}}, \boldsymbol{z}_{<t}^{\text{wait-}k}). \tag{4.28}$$

Note that the dependency on $\boldsymbol{z}_{<t}^{\text{wait-}k}$ is only relevant for the Transformer model where the path history matters.

Optimizing along multiple paths. Instead of optimizing a single decoding path, we can jointly optimize multiple deterministic paths. The additional loss terms may provide a richer training signal, and potentially yield models that could perform well in different regimes. Due to the dependence of the decoder hidden states on the full decoding path $\boldsymbol{z}_{<t}$ in the transformer-based model, it is not possible to improve over simply training in parallel across a limited set of paths. We consider a set of wait-k paths $\mathcal{Z} = \{\boldsymbol{z}^{\text{wait-}k}, \forall k \in K\}$ and choose a uniform prior over this set:

$$q(\boldsymbol{z}) = \frac{1}{|\mathrm{K}|} \sum_{k \in \mathrm{K}} [\![\boldsymbol{z} = \boldsymbol{z}^{\text{wait-}k}]\!]. \tag{4.29}$$

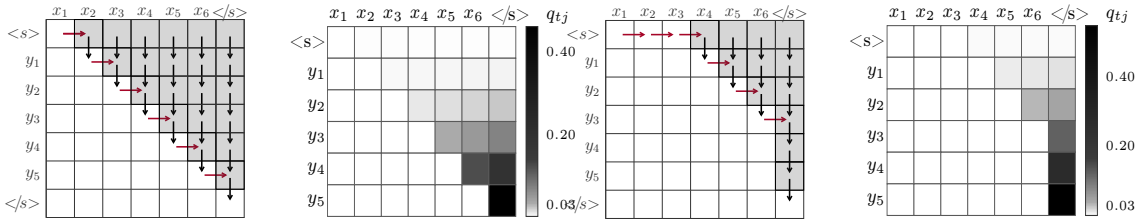
During training we encode the source sequence once, and then sample M paths $\boldsymbol{z} \sim q$ to decode along. Formally we optimize the approximated Q-function:

$$\mathrm{Q}(\boldsymbol{x}, \boldsymbol{y}, q; \theta) = \mathbb{E}_q[\log p_\theta(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{z})] \approx \frac{1}{\mathrm{M}} \sum_{\substack{m=1 \\ \boldsymbol{z} \sim q}}^{\mathrm{M}} \log p_\theta(\boldsymbol{y} | \boldsymbol{x}, \boldsymbol{z}). \tag{4.30}$$

To cover all possible wait-k paths for an input $(\boldsymbol{x}, \boldsymbol{y})$, we set K to be $[1..|\boldsymbol{x}|]$ with $|\boldsymbol{x}|$ the source length in the training batch. We will refer to this training with $k \in [1..|\boldsymbol{x}|]$. In practice, we use $\mathrm{M} = 1$ per training epoch.

With Pervasive Attention, we can leverage more training signals, in fact, the grid nature of the model allows us to efficiently compute the output distributions $p_\theta(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq j})$ all over the grid in a single forward pass of the ConvNet. Let $\mathcal{Z}$ be the set of decoding paths within an area of interest, typically paths to the right of a wait-k path (see examples in Figure 4.7). The grid has a total of $\binom{|\boldsymbol{x}|+|\boldsymbol{y}|}{|\boldsymbol{x}|}$ paths[(1)] but after removing the ones that cross the k-th

[(1)] Choosing $|\boldsymbol{y}|$ WRITEs in a sequence of $(|\boldsymbol{x}| + |\boldsymbol{y}|)$ READ/WRITE actions.



(a) $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-2}}$ (b) $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-4}}$

Figure 4.7: Optimizing along multiple paths for Pervasive Attention. In the two examples $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-2}}$ and $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-4}}$, we illustrate in the decoding grid the boundary wait-k path and the writing positions above it. Any path in which we write within the grayed area is optimized. In the q_{tj} heatmap, we show the weights q_{tj} evaluated with Eq. (4.35).

diagonal, only Z_k remain with[(2)]:

$$\mathrm{Z}_k(|\boldsymbol{y}|, |\boldsymbol{x}|) = \binom{|\boldsymbol{y}| + |\boldsymbol{x}| - k}{|\boldsymbol{y}|} - \binom{|\boldsymbol{y}| + |\boldsymbol{x}| - k}{|\boldsymbol{y}| + 1}. \tag{4.31}$$

The first term enumerates all the paths starting with k READs from which we subtract paths crossing the diagonal of the $(|\boldsymbol{y}|, |\boldsymbol{x}| - k)$-sized matrix counted via the Reflection principle (Bertrand, 1907).

When we develop the Q-function with a uniform prior over $\mathcal{Z}$, the likelihood of decoding y_t with $z_t = j$ is weighted by q_{tj}, the probability in $\mathcal{Z}$ of using this context size.

$$\mathrm{Q}(\boldsymbol{x}, \boldsymbol{y}, q; \theta) = \frac{1}{|\mathcal{Z}|} \sum_{\boldsymbol{z} \in \mathcal{Z}} \log p_\theta(\boldsymbol{y} \,|\, \boldsymbol{z}, \boldsymbol{x}) \tag{4.32}$$

$$= \frac{1}{|\mathcal{Z}|} \sum_{t=1}^{|\boldsymbol{y}|} \sum_{j=1}^{|\boldsymbol{x}|} \left(\log p_\theta(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq j}) \sum_{\boldsymbol{z} \in \mathcal{Z}} [\![z_t = j]\!] \right) \tag{4.33}$$

$$= \sum_{t=1}^{|\boldsymbol{y}|} \sum_{j=1}^{|\boldsymbol{x}|} q_{tj} \log p_\theta(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq j}). \tag{4.34}$$

The context size $z_t = j$ corresponds to passing through the edge $(t-1, j) \to (t, j)$. With the constraint of remaining above the k-th diagonal:

$$q_{tj} = \frac{\mathrm{Z}_k(t-1, j)}{\mathrm{Z}_k(|\boldsymbol{y}|, |\boldsymbol{x}|)} [\![j \geq z_t^{\text{wait-}k}]\!]. \tag{4.35}$$

[(2)] We assume that $k < |\boldsymbol{x}|$ otherwise the only remaining path is wait-until-the-end.

Empirically, using this uniform prior over $\mathcal{Z}$ leads to small weights assigned to *difficult* regions of the grid. Such regions include the area near the diagonal where we decode with a smaller context and positions at the beginning of decoding with a short prefix to build on (see the heatmaps of q_{tj} in Figure 4.7). To avoid biasing the training objective, we opted for a binary $q_{tj} = [\![j \geq z_t^{\text{wait-}k}]\!]$ to simply select all writing positions in the area of interest. We will refer to this training with $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-}k}$.

4.4.2 Dynamic Agents

Instead of following a pre-determined decoding path, here we adapt the decoding path dynamically to the context read and written so far. Recall that an agent estimates the reading probabilities $\rho_{tj} = p_\varphi(\text{READ} \,|\, \boldsymbol{x}_{\leq j}, \boldsymbol{y}_{\leq t})$ across the grid in order to estimate the likelihood of a monotonic decoding path $\boldsymbol{z}$ and that in the M-step of the training, we optimize the Q-function:

$$\text{Q}(\boldsymbol{x}, \boldsymbol{y}, q; \theta, \varphi) = \mathbb{E}_q[\log p_\theta(\boldsymbol{y} \,|\, \boldsymbol{z}, \boldsymbol{x}) + \log p_\varphi(\boldsymbol{z} \,|\, \boldsymbol{y}, \boldsymbol{x})] \,, \tag{4.36}$$

$$\mathbb{E}_q[\log p_\varphi(\boldsymbol{z} \,|\, \boldsymbol{y}, \boldsymbol{x})] = \mathbb{E}_q \left[\sum_{t=1}^{|\boldsymbol{z}|-1} \log(1 - \rho_{t z_{t+1}}) + \sum_{l=z_t}^{z_{t+1}-1} \log \rho_{tl} \right] . \tag{4.37}$$

Instead of estimating the posterior $p_\varphi(z|\boldsymbol{x}, \boldsymbol{y})$ in the E-step as outlined in §4.2, here we use an oracle to find the optimal $\boldsymbol{z}^\star$ given $(\boldsymbol{x}, \boldsymbol{y})$ and choose $q(x) = [\![z = \boldsymbol{z}^\star]\!]$ to optimize the Q-function. This is similar to the works of Zheng et al. (2019a) and Arthur et al. (2020) where an oracle produces READ/WRITE actions to guide the translation model.

Supervising the agent with an Oracle. To find the optimal decoding path $\boldsymbol{z}^\star$, we devise an oracle that leverages the emissions of the current model $p_\theta(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$. At every position (t, j) of the decoding grid we estimate the *writing* and *reading* costs W_{tj} and R_{tj} respectively:

$$\text{W}_{tj} = -\log p_\theta(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq j}), \quad \text{R}_{tj} = \lambda \frac{j}{|\boldsymbol{x}|} \quad (\lambda \geq 0). \tag{4.38}$$

The reading cost R_{tj} is designed to discourage reading the source at once especially when the writing cost at early positions is high.

In a forward pass, we estimate the lowest cost of any path leading to a position (t, j) and denote it with $\overrightarrow{\mathrm{C}}_{tj}$. Then, in a backward pass, we estimate the lowest cost of any path from a position (t, j) to the end point $(|\boldsymbol{y}|, |\boldsymbol{x}|)$ and denote it with $\overleftarrow{\mathrm{C}}_{tj}$. Both costs are evaluated recursively as follows:

$$\forall t \in [1..|\boldsymbol{y}|], \forall j \in [1..|\boldsymbol{x}|], \overrightarrow{\mathrm{C}}_{t1} = \sum_{t' \leq t} \mathrm{W}_{t'1}, \quad \overrightarrow{\mathrm{C}}_{1j} = \sum_{j' \leq j} \mathrm{R}_{1j'}, \tag{4.39}$$

$$\forall t \in [2..|\boldsymbol{y}|], \forall j \in [2..|\boldsymbol{x}|], \quad \overrightarrow{\mathrm{C}}_{tj} = \min\left(\overrightarrow{\mathrm{C}}_{t-1,j} + \mathrm{W}_{tj},\ \overrightarrow{\mathrm{C}}_{t,j-1} + \mathrm{R}_{tj}\right), \tag{4.40}$$

$$\forall t \in [1..|\boldsymbol{y}|], \forall j \in [1..|\boldsymbol{x}|], \overleftarrow{\mathrm{C}}_{t|\boldsymbol{x}|} = \sum_{t' > t} \mathrm{W}_{t'|\boldsymbol{x}|}, \quad \overleftarrow{\mathrm{C}}_{|\boldsymbol{y}|j} = \sum_{j' > j} \mathrm{R}_{tj'}, \tag{4.41}$$

$$\forall t \in [1..|\boldsymbol{y}| - 1], \forall j \in [1..|\boldsymbol{x}| - 1], \quad \overleftarrow{\mathrm{C}}_{tj} = \min\left(\overleftarrow{\mathrm{C}}_{t+1,j} + \mathrm{W}_{t+1,j},\ \overleftarrow{\mathrm{C}}_{t,j+1} + \mathrm{R}_{t,j+1}\right). \tag{4.42}$$

To get $\boldsymbol{z}^\star$, we start from the top-left corner of the grid $(t, j) = (1, 1)$ and advance by picking the next position with the lowest aggregate cost $\mathrm{C} = \overrightarrow{\mathrm{C}} + \overleftarrow{\mathrm{C}}$ *i.e.* we decide to read if $\mathrm{C}_{t,j+1} < \mathrm{C}_{t+1,j}$ otherwise we write.

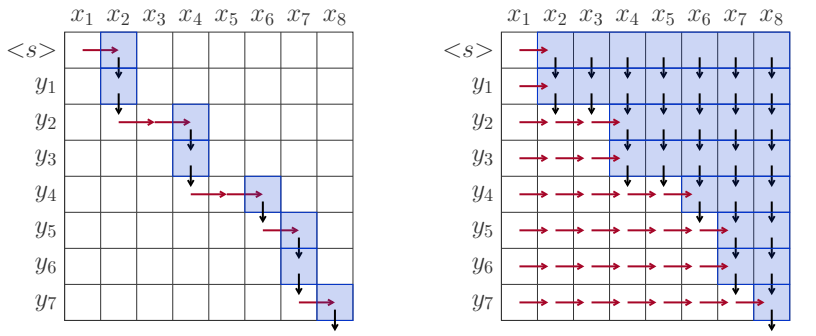
Our oracle and by extension this dynamic agent is only viable for Pervasive Attention models where we can easily estimate the writing costs across the grid.

Modeling the dynamic agent. To model the reading probabilities ρ_{tj} we use a ConvNet, similarly to Pervasive Attention models, to jointly encode the prefixes $\boldsymbol{x}_{\leq j}$ and $\boldsymbol{y}_{\leq t}$. The ultimate ConvNet features are then mapped to $[0, 1]$ with a sigmoid-activated affine form. Once the target $\boldsymbol{z}^\star$ is selected we maximize the Q-function term supervising the agent:

$$\mathbb{E}_q[\log p_\varphi(\boldsymbol{z} \,|\, \boldsymbol{y}, \boldsymbol{x})] = \sum_{t=1}^{|\boldsymbol{z}|-1} \log(1 - \rho_{tz^\star_{t+1}}) + \sum_{l=z^\star_t}^{z^\star_{t+1}-1} \log \rho_{tl}. \tag{4.43}$$

Initial experiments show that we get better READ/WRITE accuracies if we train the agent to read left of $\boldsymbol{z}^\star$ and write right of $\boldsymbol{z}^\star$ allowing the agent to catch up after surpassing the ideal context *i.e.* we optimize:

$$\sum_{t=1}^{|\boldsymbol{z}|-1} \sum_{j=1}^{|\boldsymbol{x}|} [\![j \geq z^\star_{t+1}]\!] \log(1 - \rho_{tj}) + [\![j < z^\star_{t+1}]\!] \log \rho_{tj}. \tag{4.44}$$



(a) Optimal path $\boldsymbol{z}^\star$ (b) READ/WRITE supervision

Figure 4.8: Illustration of the oracle optimal path $\boldsymbol{z}^\star$ and the inferred labels supervising the agent. In the M-step, we set $q_{tj} = 1$ in the highlighted blue positions to train the translation model.

When jointly training the agent and the online MT model, updating the MT model's parameters entails that the oracle will also update its estimate of the optimal path $\boldsymbol{z}^\star$. We found this joint training to be better than freezing the MT model when training the agent. We also found that training the model to decode right of $\boldsymbol{z}^\star$ leads to better results compared to using $q = [\![\boldsymbol{z} = \boldsymbol{z}^\star]\!]$ in the decoding objective. All in all, the Q-function takes the form:

$$\mathrm{Q}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}^\star; \theta, \varphi) = \alpha \sum_{t=1}^{|\boldsymbol{y}|} \sum_{j=1}^{|\boldsymbol{x}|} [\![j \geq z_t^\star]\!] \log p_\theta(y_t | \boldsymbol{y}_{<t}, \boldsymbol{x}_{\leq j}) \tag{4.45}$$

$$+ \beta \sum_{t=1}^{|\boldsymbol{z}|-1} \sum_{j=1}^{|\boldsymbol{x}|} [\![j \geq z_{t+1}^\star]\!] \log(1 - \rho_{tj}) + [\![j < z_{t+1}^\star]\!] \log \rho_{tj}, \tag{4.46}$$

where (α, β) are hyper-parameters balancing the two terms.

4.5 Experiments

We first experiment on the IWSLT'14 De↔En (Cettolo et al., 2014) and IWSLT'15 En↔Vi datasets (Luong and Manning, 2015). Then, we scale up to the larger benchmark of WMT'15 De→En.[3]

[3] http://www.statmt.org/wmt15/translation-task.html

	IWSLT'14 De↔En		IWSLT'15 En↔Vi		WMT'15 De→En	
	PA	TF	PA	TF	PA	TF
Embedding d	256	256	256	512	512	512
Feed-forward network d_{FF}	1024	1024	1024	1024	2048	2048

Table 4.1: The sizes of the embeddings (d) and the feed-forward features of Pervasive Attention (PA) and Transformer (TF) models trained in each benchmark.

4.5.1 Experimental Setup

IWSLT'14 De↔En. We replicate the setup of Edunov et al. (2018) and remove sentences longer than 175 words and pairs with length-ratio exceeding 1.5. We train on 160K pairs, develop on 7283 held out pairs and test on TED dev2010+tst2010-2013 (6750 pairs). All data is tokenized and lower-cased using the standard scripts from the Moses toolkit (Koehn et al., 2007). We segment sequences using byte pair encoding with 10K merge operations on the merged bi-texts. This results in a German and English vocabularies of 8.8K and 6.6K types respectively.

IWSLT'15 En↔Vi. We use the setup of Luong and Manning (2015). We train on 133K pairs, develop on TED tst2012 (1553 pairs) and test on TED tst2013 (1268 pairs). Apart from tokenizing the corpus, no other pre-processing step was done. The vocabularies are of 17K and 7.7K words in English and Vietnamese respectively.

WMT'15 De↔En. The training data is taken from the Europarl corpus (Koehn, 2005), the News Commentary (Tiedemann, 2012) and the Common Crawl corpus (Smith et al., 2013). We reproduce the setup of Ma et al. (2019a); Arivazhagan et al. (2019) with a joint vocabulary of 32K BPE types. We train on 4.5M pairs, develop on newstest2013 (3000 pairs) and test on newstest15 (2169 pairs).

Baselines. In each benchmark we train Pervasive Attention (PA) and Transformer (TF) models. Pervasive Attention models use residual-cumulative skip connections and stack $\text{N} = 14$ layers with convolution filters in $\mathbb{R}^{11\times11}$. Unless otherwise mentioned, we default to aggregating the final features with max-pooling. We train Transformer *small* on IWSLT'14

De↔En, a modified *base* on IWSLT'15 En↔Vi and Transformer *base* on WMT'15 De→En. The hyper-parameters of the trained models in each benchmark are given in Table 4.1.

For offline comparison, we train each model for offline decoding with uni-directional and bi-directional encoding of the source sequence. Each model is decoded greedily and with beam-search (b=5) and we measure case-sensitive [(4)] tokenized BLEU with *multi-bleu.pl*[(5)] after reversing the byte-pair encoding.

We only evaluate with greedy decoding for online models. We measure the translation quality with BLEU and the decoding latency with the three metrics discussed in the following section. Online models trained for wait-k decoding will be evaluated for decoding along wait-k_{eval} with $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$ regardless of the training k.

4.5.2 Latency metrics

We measure the decoding latency with Average Proportion (AP) (Cho and Esipova, 2016), Average Lagging (AL) (Ma et al., 2019a), and Differentiable Average Lagging (DAL) (Cherry and Foster, 2019; Arivazhagan et al., 2019).

AP (Cho and Esipova, 2016) measures the decoding latency by averaging the absolute delays of the decoded target tokens. Each target token incurs a delay of z_t, the size of the source context it used.

$$\text{AP}(\boldsymbol{z}) = \frac{1}{|\boldsymbol{y}|} \sum_{t=1}^{|\boldsymbol{y}|} \frac{z_t}{|\boldsymbol{x}|}. \tag{4.47}$$

AP ranges between 0 and 1 where an offline evaluation (wait-∞) reaches AP=1. As noted in previous works (Alinejad et al., 2018; Ma et al., 2019a; Arivazhagan et al., 2019), AP is unfavorable to short sequences. With $|\boldsymbol{x}| = |\boldsymbol{y}| = n$, A wait-1 path has an AP of $\frac{n+1}{2n}$ ranging from 0.75 for $n = 2$ to 0.5 for $n \to +\infty$. Besides this bias to long sequences, AP of the most aggressive wait-k decoding path is higher than 0.5 which means that we will not use half of the AP's range.

(4) except for IWSLT'14 De↔En where the data is lower-cased.

(5) https://github.com/moses-smt/mosesdecoder/blob/master/scripts/generic/multi-bleu.perl

Ma et al. (2019a) introduced average lagging (AL) that measures the average rate by which the decoding lags behind *the ideal simultaneous policy* wait-0. AL is designed so that for pairs with equal lengths ($|\boldsymbol{x}| = |\boldsymbol{y}|$), a wait-$k$ decoding path has an AL of k *i.e.* lagging k tokens behind the ideal wait-0. For such a pair, AL is evaluated as:

$$\text{AL} = \frac{1}{\tau(\boldsymbol{x}, \boldsymbol{z})} \sum_{t=1}^{\tau(\boldsymbol{x},\boldsymbol{z})} z_t - (t-1), \quad \tau(\boldsymbol{x}, \boldsymbol{z}) = \min\{t \,|\, z_t = |\boldsymbol{x}|\} \tag{4.48}$$

where $\tau(\boldsymbol{x}, \boldsymbol{z})$ is the *cut-off* step when we finish reading the source $\boldsymbol{x}$. For the more realistic $|\boldsymbol{x}| \neq |\boldsymbol{y}|$, the ideal context size (t-1) is scaled by the length-ratio $|\boldsymbol{x}|/|\boldsymbol{y}|$, in which case AL takes the form:

$$\text{AL} = \frac{1}{\tau(\boldsymbol{x}, \boldsymbol{z})} \sum_{t=1}^{\tau(\boldsymbol{x},\boldsymbol{z})} z_t - \frac{|\boldsymbol{x}|}{|\boldsymbol{y}|}(t-1). \tag{4.49}$$

The summands beyond the cutoff in Eq. (4.49) will decrease the average lagging when they should not, so the straightforward solution is to ignore them completely. And so, AL assumes the tail after the cutoff $\tau(\boldsymbol{x}, \boldsymbol{z})$ is generated instantly without any effect on the delay.

More recently Cherry and Foster (2019) and Arivazhagan et al. (2019) proposed Differentiable Average Lagging (DAL) where they include the lagging terms beyond the cutoff τ. To avoid artificially decreasing AL, they enforce a minimum delay of $|\boldsymbol{x}|/|\boldsymbol{y}|$ for writing any target token. This minimum delay is accounted for with the modified context sizes $\boldsymbol{z}'$.

$$\text{DAL} = \frac{1}{|\boldsymbol{y}|} \sum_{t=1}^{|\boldsymbol{y}|} z'_t - \frac{|\boldsymbol{x}|}{|\boldsymbol{y}|}(t-1), \quad z'_t = \begin{cases} z_t, & \text{if } t = 1 \\ \max\left(z_t, z'_{t-1} + \frac{|\boldsymbol{x}|}{|\boldsymbol{y}|}\right), & \text{otherwise.} \end{cases} \tag{4.50}$$

Arivazhagan et al. (2019) were also motivated by the differentiability of DAL after the removal of arg min from the definition. With that, DAL can be used to regularize the training objectives of a trainable decoding agent.

Other metrics include the Consecutive Wait (CW) introduced in Gu et al. (2017b) that measures the average number of source tokens read between two consecutive writes *i.e.* the average of the increments $z_t - z_{t-1}$.

In this chapter, we will report Average Lagging (AL) for a direct comparison with existing works (Ma et al., 2019a; Zheng et al., 2019a; Arivazhagan et al., 2019; Ma et al., 2020), and

Encoder	Uni-directional				Bi-directional			
Decoding	BLEU(b=1)		BLEU(b=5)		BLEU(b=1)		BLEU(b=5)	
Architecture	PA	TF	PA	TF	PA	TF	PA	TF
IWSLT'14 En→De	**26.81**	26.58	27.54	27.70	27.23	27.46	27.96	**28.27**
IWSLT'14 De→En	32.40	**32.81**	33.62	33.79	33.43	33.64	34.36	**34.56**
IWSLT'15 En→Vi	**29.22**	28.90	30.20	29.65	29.81	29.33	**30.49**	30.17
IWSLT'15 Vi→En	**26.81**	25.73	27.96	26.92	27.43	28.09	28.04	**29.00**
WMT'15 De→En	28.08	**31.14**	29.28	32.27	28.78	31.96	29.61	**32.80**

Table 4.2: Offline evaluation of the Pervasive Attention (PA) and Transformer (TF) with either a uni-directional or a bi-directional encoder. We evaluate using greedy decoding and beam-search. For each task we highlight the best overall performance as well as the best uni-directional greedy decoding (best of the two first columns).

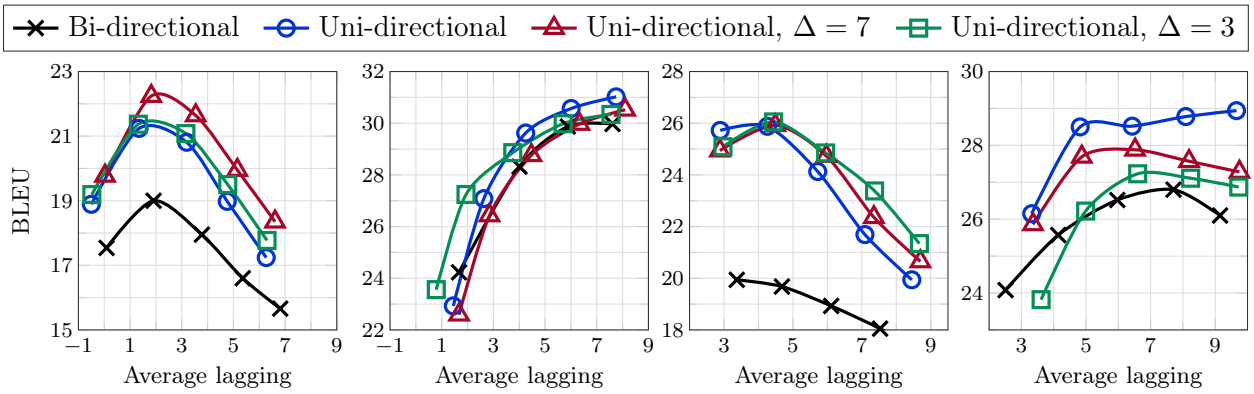
include Average Proportion (AP) and Differentiable Average Lagging (DAL) measures in Appendix D.

4.5.3 Offline Models

Table 4.2 reports offline performance of Pervasive Attention (PA) and Transformer (TF) models with both a uni-directional encoder and a bi-directional encoder. Each model is evaluated with greedy decoding then with beam-search. Overall and as expected, bi-directional encoders in the offline setup are better than their uni-directional counterparts. The gain on PA is of 0.65 on average while for the Transformer models (TF) the addition of bi-directionality improves BLEU by 1.1 on average. The first two columns of Table 4.2 show that Pervasive Attention (PA) is very competitive with Transformer (TF) on the small datasets particularly with the uni-directional encoding of the source.

4.5.4 Online Models with Deterministic Agents

Impact of the encoder's receptive field. The offline comparison clearly favors bi-directional encoders. We will verify in this section if the superiority of bi-directional encoders



(a) De→En $k = 1$ (b) De→En $k = 7$ (c) En→Vi $k = 1$ (d) En→Vi $k = 7$

Figure 4.9: IWSLT'14 De→En and IWSLT'15 En→Vi: Impact of the encoder's receptive field on online Transformer models for wait-k decoding ($k \in \{1, 7\}$). Each model is evaluated along wait-k_{eval} with $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$.

holds in the context of online translation. For the Transformer model, we initially consider a bi-directional encoder similar to Ma et al. (2019a); Zheng et al. (2019a), in which case the encoder states have to be updated after each read. With a bi-directional encoder, both the training and decoding are very costly. Second, we consider a uni-directional encoder and then we constrain the encoder receptive field by masking all previously read tokens but the last Δ. The standard self-attention in the encoder has an unlimited receptive field ($\Delta = \infty$).

Figure 4.9 shows the effect of the encoder receptive field on the quality of online decoding. We look at two particular wait-k paths: $k = 1$ and $k = 7$. On the extreme case of $k = 1$ limiting the receptive field helps improve the decoding quality. On the other hand, with $k = 7$ the unlimited receptive field gives the best BLEU. On both cases however, the bi-directional encoder under-performs. We conclude then that the superiority of bi-directional encoders offline does not carry over to online translation. Furthermore, we argue that limiting the encoder memory makes the task of the wait-k decoders easier, particularly with small k.

In the remainder of this chapter, we use uni-directional encoders for both TF and PA architectures. Other than the uni-directionality of the Transformer encoder, we investigate in Appendix C the importance of the <s> and </s> markers in the online encoding of the source.

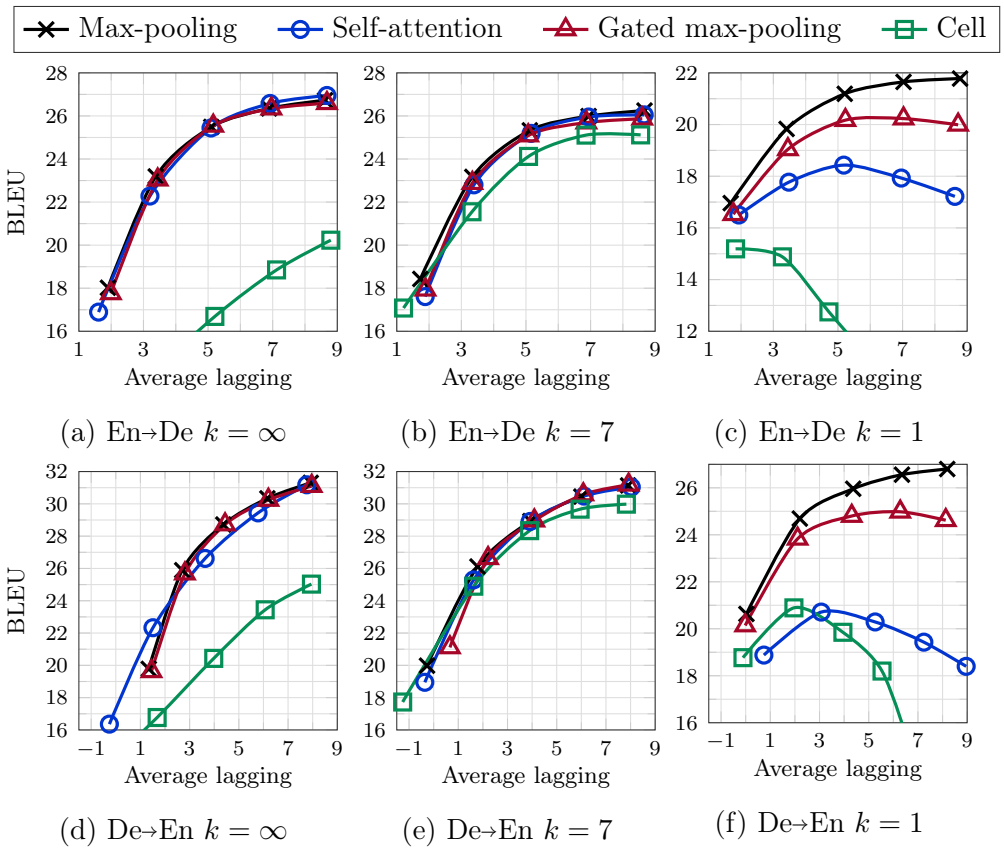


Figure 4.10: IWSLT'14 De↔En: Training Pervasive Attention models for wait-k decoding ($k \in \{1, 7, +\infty\}$) with different aggregations of the ConvNet's features. Each model is evaluated along wait-k_{eval} with $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$.

Ablation of Pervasive Attention models. We present in this section an ablation of the aggregation methods for Pervasive Attention models. This study is similar to §3.3.2 except that we focus on the features' aggregation in uni-directional models trained for wait-k decoding. Among the 3 different pool of features we considered (see Figure 4.4), pooling the full grid led to mediocre BLEU scores (less than 10 across all latency regimes) and is thus omitted in the panels of Figure 4.10. In the results of Figure 4.10, predicting with the cell features performed worse than pooling the row of features up to z_t. With row-pooling, and unlike with offline models, simple max-pooling outperformed its gated version as well as self-attention, particularly when trained for the difficult wait-1 path. In the following we will default to row max-pooling for Pervasive Attention models.

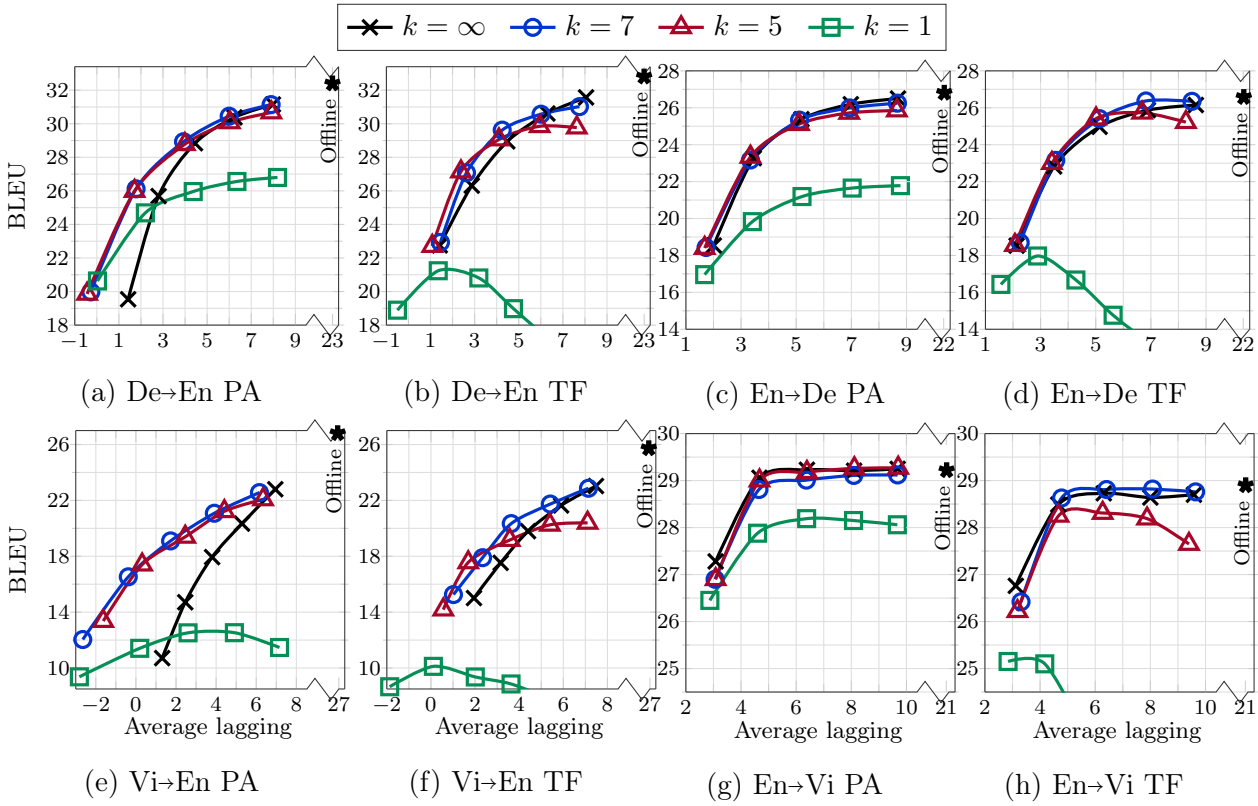
Pervasive attention and Transformer for online translation. In this section, we evaluate the wait-k online decoding for different TF and PA models, trained for different wait-k decoding paths. We denote with $k = \infty$ the wait-until-end training where the full source is read before decoding. In each figure the offline results are added for reference with their latency AL $= |\boldsymbol{x}|$.

We show in Figure 4.11 the performance of models trained for wait-k decoding across a range of latencies. We observe that for both architectures across the 4 tasks, models trained on wait-7 generalize well on other evaluation paths. Thus, unlike Ma et al. (2019a), we note that we can train a single model to use in different latency regimes *i.e.* we do not have to use $k_{\text{eval}} = k$ to achieve better BLEU scores. This generalization to other wait-k paths is notably stronger with PA models compared to TF models that drop in performance far from the training path (*e.g.* $k = 1$ and $k_{\text{eval}} = 7$). Overall, for tasks where PA performs better offline the model consistently outperforms TF online and vice versa.

Joint training on multiple paths. In the previous section, we found that training on a particular wait-k generalizes to all the other paths. To avoid tuning k and finding the optimal path for each dataset, we consider jointly optimizing multiple paths. For TF models we use $k \in [1..|\boldsymbol{x}|]$ and for PA $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$.

Results in Figure 4.12 show that this joint optimization, for both architectures and on two datasets, manages to achieve comparable results to training on a specific path that we have to manually select.

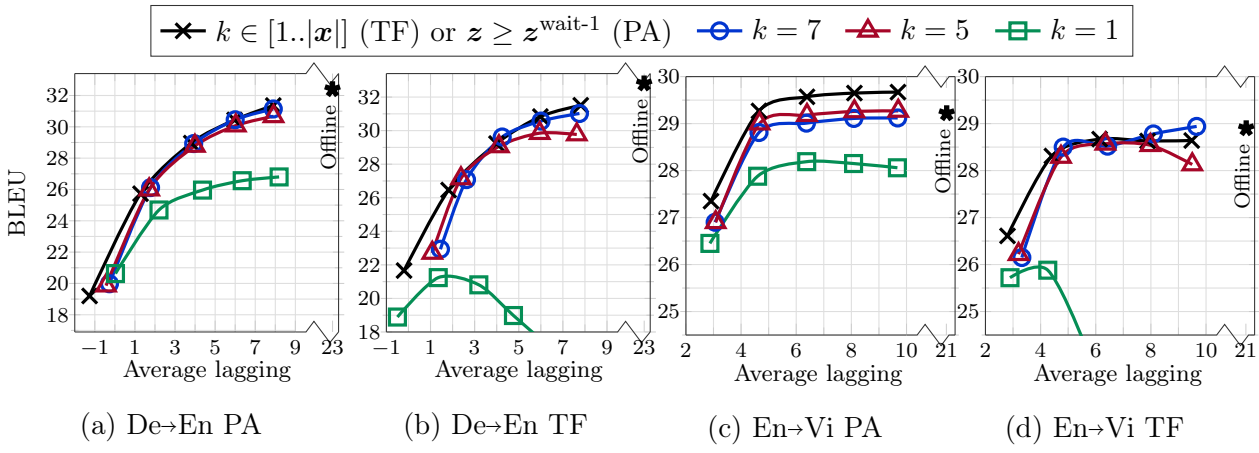
Experiments on the WMT15 De→En corpus. In Figure 4.13a we present our results on the WMT15 De→En task using transformer *base* models. The results confirm observations made on IWSLT, namely, (i) a single low-latency model ($k = 7$) performs better or comparable to using separate models trained specifically for the evaluation path $k = k_{\text{eval}}$. (ii) if jointly training on multiple paths, we can achieve comparable results to training on different paths and then selecting the best model. The main difference w.r.t. the IWSLT results is observed in the model trained for offline decoding ($k = \infty$). Whereas on IWSLT it was competitive with the best models trained for online decoding, for this larger corpus, this is no longer the case and the offline model performs significantly worse than the ones specifically trained for



(a) De→En PA (b) De→En TF (c) En→De PA (d) En→De TF

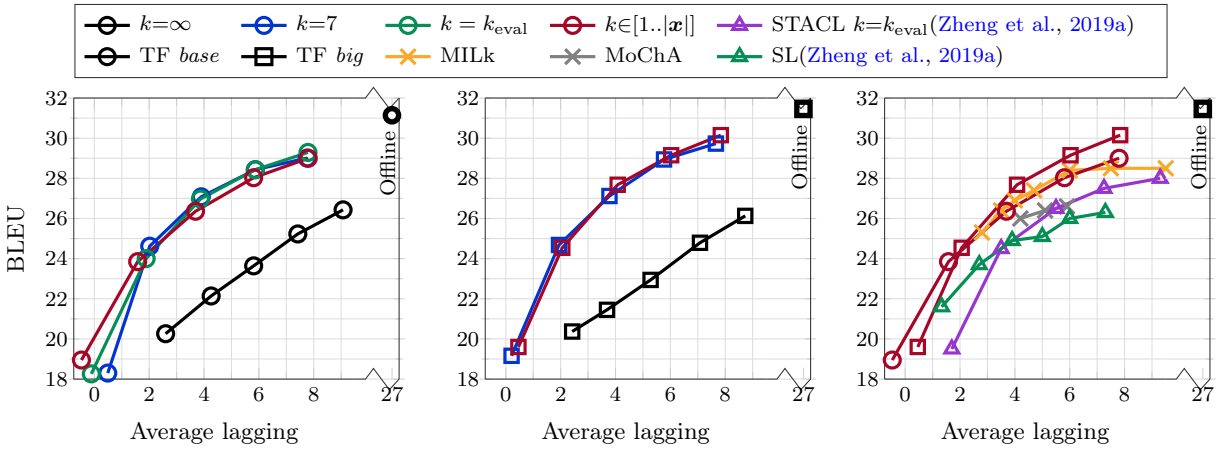
(e) Vi→En PA (f) Vi→En TF (g) En→Vi PA (h) En→Vi TF

Figure 4.11: IWSLT'14 De↔En and IWSLT'15 En↔Vi: wait-k online decoding with Pervasive attention (a,c,e,g) and Transformer (b,d,f,h). Each model is evaluated along wait-k_{eval} with $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$.



(a) De→En PA (b) De→En TF (c) En→Vi PA (d) En→Vi TF

Figure 4.12: IWSLT De→En and En→Vi: TF and PA models jointly trained on multiple paths. Each model is evaluated along wait-k_{eval} with $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$.



(a) Transformer *base* (b) Transformer *big* (c) State-of-the-art results

Figure 4.13: Experimental results on WMT'15 De→En. Wait-k models are evaluated along wait-k_{eval} with $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$.

online decoding. This is probably due to the domain of the training and evaluation data; IWSLT datasets are based on sentence-level aligned transcriptions of TED talks (spoken language) while WMT data consists of noisily aligned news stories and web crawled data. Ours observations on the Transformer *base* wait-k models apply for the *big* variant as well (see Figure 4.13b).

Comparison to state-of-the-art. Figure 4.13c compares our results with state-of-the-art performances on WMT 15 De→En. Namely, the deterministic STACL wait-k (Ma et al., 2019a) and the dynamic supervised adaptive policy (SL) Zheng et al. (2019a), MoChA (Raffel et al., 2017) and MILk (Arivazhagan et al., 2019). The first two models are based on Transformer *base* while the last two use RNMT+ (Chen et al., 2018), a recurrent architecture. For a fair comparison with RNMT+ based MILk and MoChA, we also train Transformer *big*, being the most comparable in terms of offline performances to RNMT+.

Our simple uni-directional wait-k model trained on multiple paths establishes a new state of the art for this task, significantly improving over STACL and SL across the full range of latency levels. It also matches the performance of MILk with a single trained model instead of having a different model trained for each latency regime.

4.5.5 Online Models with Dynamic Agents

In this section we train Pervasive Attention models with dynamic agents on IWSLT'14 De↔En and IWSLT'15 En↔Vi datasets. We consider three different pre-trainings or initializations for our dynamic models: translation models trained for offline decoding ($k = \infty$), models trained for decoding along the $\boldsymbol{z}^{\text{wait-7}}$ path ($k = 7$) and models trained to jointly optimize paths above the $\boldsymbol{z}^{\text{wait-1}}$ path ($\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$). These models are pre-trained for 50K updates (see §4.5.4) before initializing both the dynamic agent and the translation model. We jointly optimize the two branches with $\alpha = \beta = 1$ for another 10K updates.

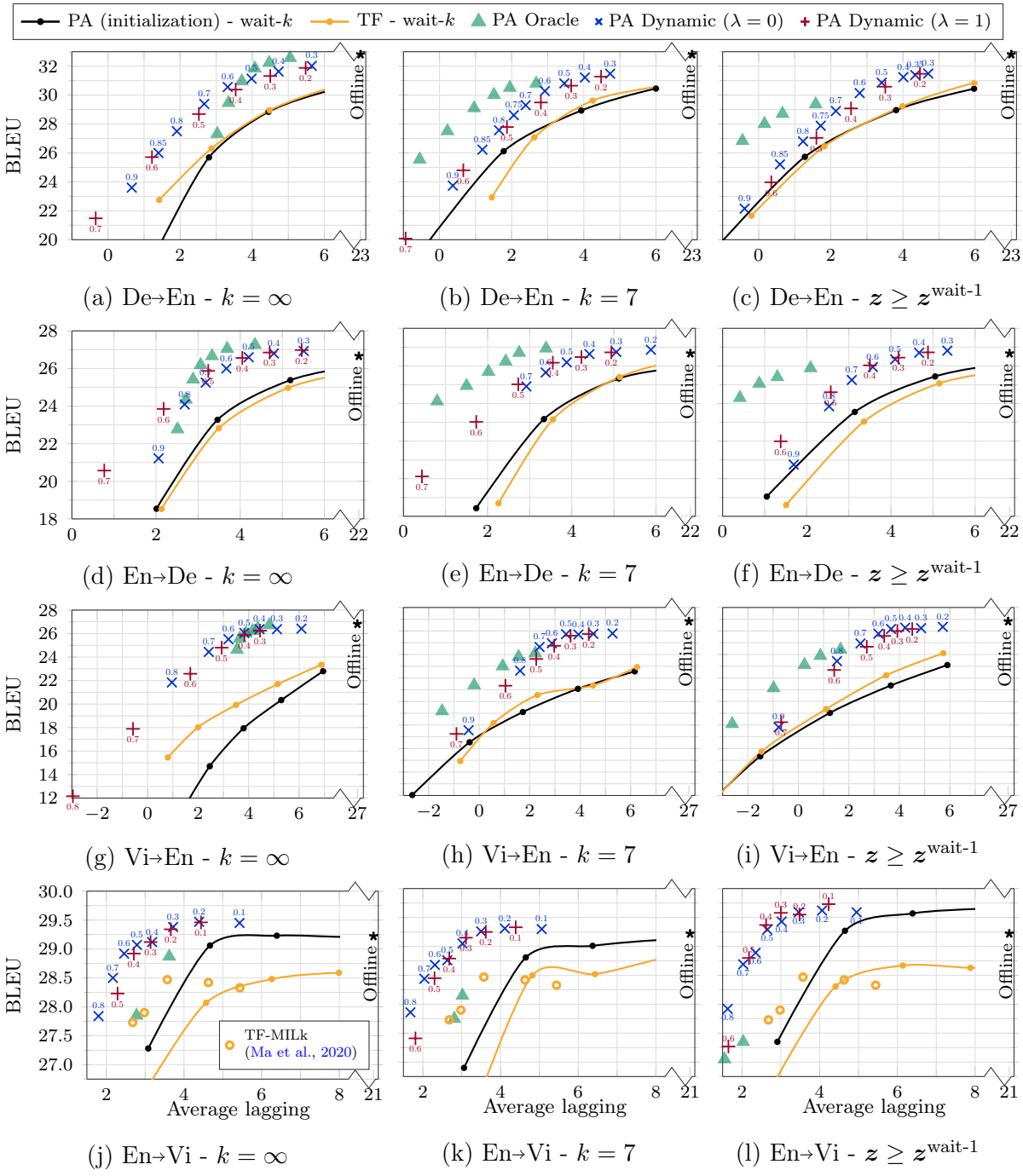
In the results of Figure 4.14, we first assess the quality of the oracle by decoding along its optimal paths $\boldsymbol{z}^{\star}$. The paths are estimated given the true target $\boldsymbol{y}$ and the full source $\boldsymbol{x}$ forwarded in the MT model in teacher-forcing mode (see Algorithm 1). Afterwards, we decode a hypothesis by simply following $\boldsymbol{z}^{\star}$ as we would with a deterministic path.

Models trained for low-latency ($k = 7$ and $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$ in the last two columns of Figure 4.14) have more potential than the offline model with our oracle (the first column of Figure 4.14) and are a priori a better choice for pre-training our dynamic agents. On the En→Vi task, the oracle is particularly deficient operating to the left of the deterministic wait-k decoding without improving the BLEU scores. This is probably due to the quality of the translation and proportionally the oracle's writing costs plateauing right after $k_{\text{eval}} = 2$. Under such circumstances, the cost of a path is a poor indicator of the translation's quality.

Algorithm 1 Pseudo-code for decoding along the oracle's $\boldsymbol{z}^{\star}$

Input: Source-target sequences $(\boldsymbol{x}, \boldsymbol{y})$.
Output: A hypothesis $\widetilde{\boldsymbol{y}}$.

Forward $(\boldsymbol{x}, \boldsymbol{y})$ in the translation model to estimate the writing costs.
Estimate the grid costs $\overrightarrow{\mathrm{C}}$ and $\overleftarrow{\mathrm{C}}$.
Infer $\boldsymbol{z}^{\star}$ from the READ/WRITE costs.
▷ Decode along $\boldsymbol{z}^{\star}$:
$t = 0, \quad \widetilde{y}_t = <s>$.
while $\widetilde{y}_t \neq </s>$ **do**
    $\widetilde{y}_t = \arg\max_{y_t} p(y_t \,|\, \widetilde{\boldsymbol{y}}_{<t}, \boldsymbol{x}_{\leq z^{\star}_t})$.
    $t\,{+}{+}$.
end while



(a) De→En - $k = \infty$ (b) De→En - $k = 7$ (c) De→En - $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$

(d) En→De - $k = \infty$ (e) En→De - $k = 7$ (f) En→De - $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$

(g) Vi→En - $k = \infty$ (h) Vi→En - $k = 7$ (i) Vi→En - $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$

(j) En→Vi - $k = \infty$ (k) En→Vi - $k = 7$ (l) En→Vi - $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$

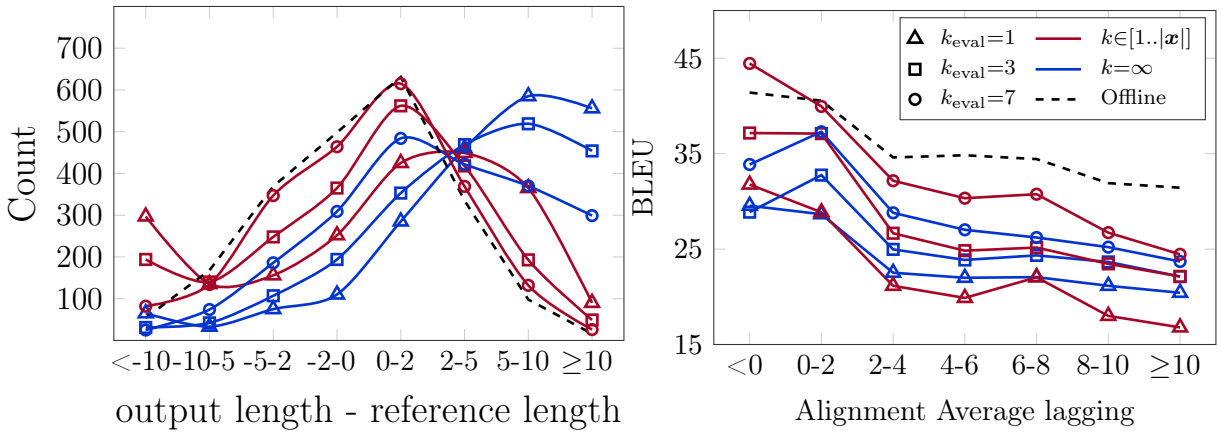
Figure 4.14: IWSLT'14 De↔En and IWSLT'15 En↔Vi: For different initializations, we report (a) the performance of the initialization along wait-k_{eval} for $k_{\text{eval}} \in \{1, 3, 5, 7, 9\}$, (b) the performance of an analogous TF model along the same wait-k_{eval}, (c) the performance of the oracle under different reading costs λ, and (d) the performance of two dynamic agents supervised with the oracle using $\lambda = 0$ or $\lambda = 1$. Each dynamic agent is evaluated with different thresholds τ to decide whether to read or write. Details of the hyper-parameters can be found in Appendix D.

In each task (De→En, En→De, Vi→En and En→Vi), we fine-tune six dynamic agents with the three different initializations and under two regularizers $\lambda \in \{0, 1\}$ for the supervising oracle (λ as defined in §4.4.2 weights the reading cost). A dynamic agent is evaluated with various thresholds $\tau \in [0, 1]$ to binarize the reading probability ρ_{tj} into a READ/WRITE decision. With a higher threshold τ, the model is encouraged to read less and achieve a lower lagging. The results in Figure 4.14 show that, regardless of the initialization, we can improve the translation quality at comparable delays to wait-k decoding by up to 3 BLEU points. With the $k = \infty$ initialization, the dynamic agents catch up with the oracle (see panels a,d and g of Figure 4.14), however, with $k = 7$ and $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$, the dynamic agents shift to the right (higher delay) and improve the translation quality over the initialization in the AL mid-range (De→En: [3,5], En→De: [4, 6] and Vi→En: [2, 6]). When increasing the threshold τ to force a lower lagging, models trained with $\lambda = 1$ appear to be more malleable and can cover a wide range of delays compared to $\lambda = 0$. Models for the difficult direction De→En (German is verb-final and important context might be delayed until the end of the source) fall back to the initial wait-k performance when forcing a translation with AL$\leq$ 2. On the En→Vi task, and although the oracle's performance is worse than the wait-k initialization, supervising a dynamic agent with this oracle manages to outperform the wait-k models. Note that since we are jointly optimizing the agent and the underlying translation model, the oracle's supervising labels are changing after every update. In the last row of panels in Figure 4.14, we include results from the recent Ma et al. (2020) using monotonic multihead-attention with infinite lookback (a.k.a MILk) for Transformer models. Our dynamic agents outperform Transformer-MILk by about one BLEU point.

4.5.6 Qualitative analysis

4.5.6.1 WMT'15 De→En:

We use the `compare-mt` toolkit (Neubig et al., 2019) to compare the outputs of our different systems on the test set of WMT'15 De→En. We look more closely at the following properties:



(a) Output length distribution

(b) Quality per level of difficulty

Figure 4.15: WMT'15 De→En: Qualitative analysis of wait-k agents.

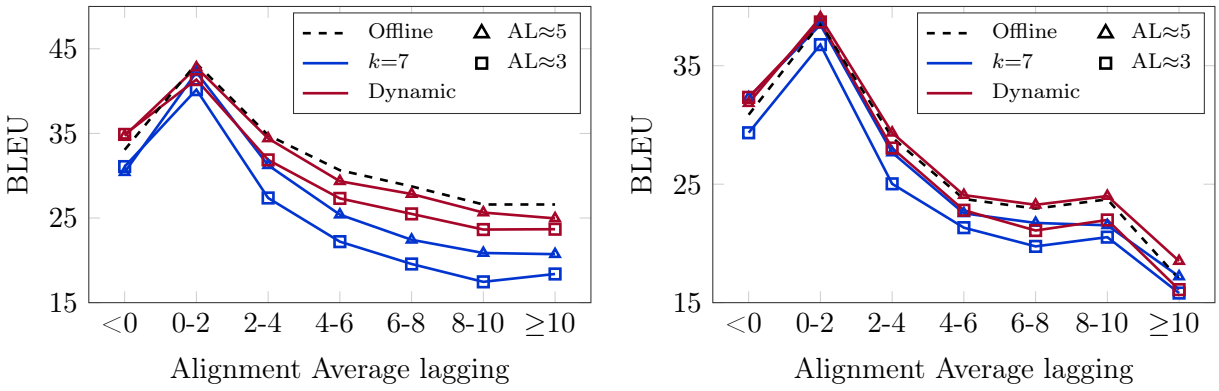
Length distribution. The distribution of the difference between the length of the system's output and the length of the reference. When evaluated in low latency regimes (k_{eval}=1), the model trained in the offline setup (k=∞) tends to generate lengthy translations with mainly repeated n-grams. Figure 4.15a shows the length difference distribution for the k=∞ and the k∈[1..|$\boldsymbol{x}$|] models where the multiple-paths model matches the offline distribution while k=∞ is skewed towards long outputs.

Sentence-BLEU per bucket of difficulty. To measure the difficulty of a pair, we first estimate source-target alignments with `fast-align` (Dyer et al., 2013) and then infer a reference decoding path. The reference decoding path $\boldsymbol{z}$ is non-decreasing and guarantees that at a given decoding position t, z_t is larger or equal than all the source positions aligned with t. Concretely, given an alignment map from the target to the source $\mathcal{A}$ (*i.e.* for each t, $\mathcal{A}(t)$ is a list of source indices aligned with t), we evaluate z_t recursively as:

$$\forall t \in [1..|\boldsymbol{y}|],\ a_t = \max(\mathcal{A}(t)),$$
$$z_1 = a_1, \quad \forall t \in [2..|\boldsymbol{y}|],\ z_t = \max(z_{t-1}, a_t)$$

The difficulty is finally measured as the Average Lagging (AL) of the reference $\boldsymbol{z}$.

We measure the average Sentence-BLEU per bucket of difficulty (Figure 4.15b). While BLEU of offline model naturally decreases with increasing source-target divergence, the performance loss is more important for the online models (k=∞ and k∈[1..|$\boldsymbol{x}$|]), the multiple-



(a) De→En (b) En→De

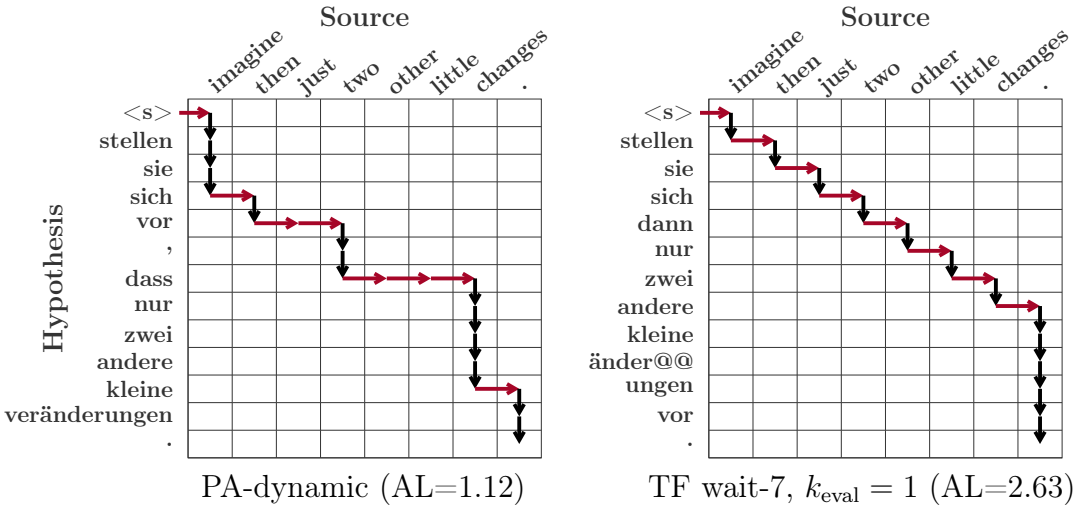
Figure 4.16: IWSLT'14 De↔En: Quality per level of difficulty

paths model being more robust. Several cases are a verb-final sentence in German for which online MT struggles to predict its translation and eventually omits it. This method to extract challenging sentences could be used more systematically to identify weaknesses of low latency NMT.

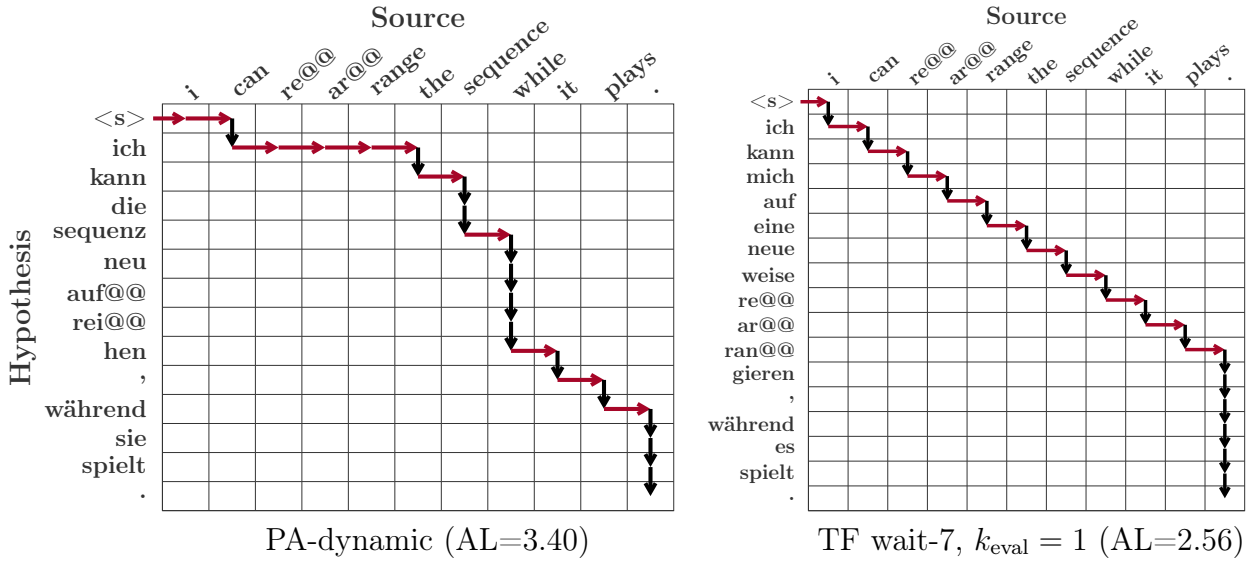
4.5.6.2 IWSLT'14 De↔En:

In this section, we compare dynamic agents to deterministic wait-7 agents on IWSLT'14 De↔En in increasing levels of difficulty. We pick an operating point of the dynamic agent that achieves a lagging (AL) comparable to a wait-k decoder. For En→De, the dynamic agent is at (BLEU=26.98, AL=5.46) and (BLEU=26.17, AL=3.55) compared to wait-7 at (BLEU=25.41, AL=5.14) and (BLEU=23.15, AL=3.55). For De→En, the dynamic agent is at (BLEU=31.63, AL=4.73) and (BLEU=29.39, AL=2.67) compared to wait-7 at (BLEU=29.62, AL=4.25) and (BLEU=27.08, AL=2.63).

In Figure 4.16, we note that dynamic agents on En→De outperform the offline model across all difficulty ranges. However, on De→En, the dynamic model under-performs in the buckets of difficulty larger than 4. Compared to wait-k, dynamic agents achieve better accuracies across the board.

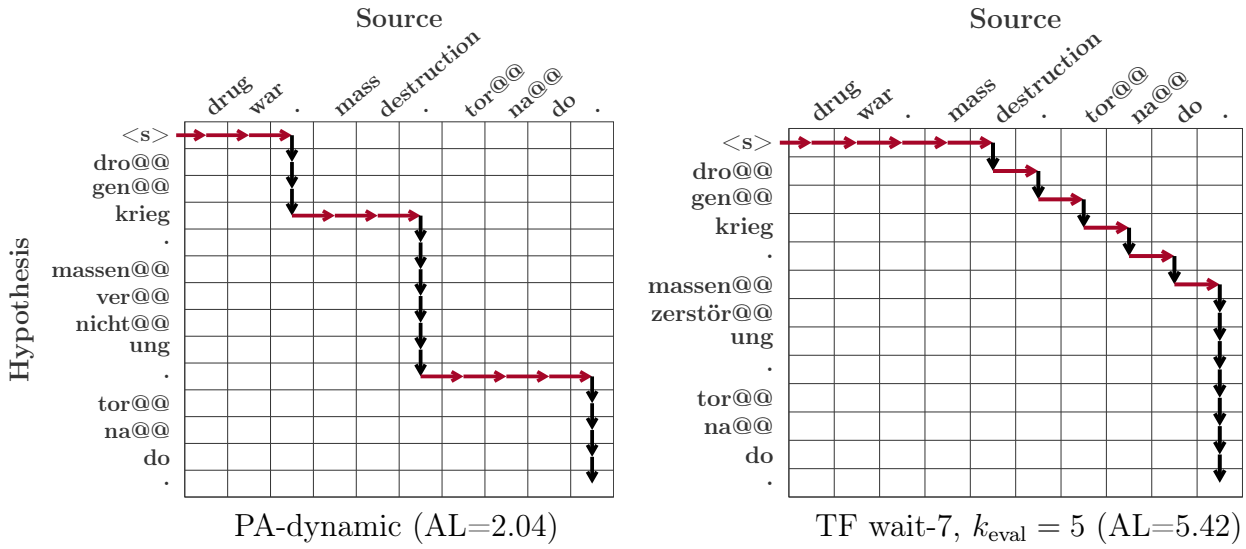


(a) Reference: stellen sie sich dann nur zwei andere kleine änder@@ungen vor .

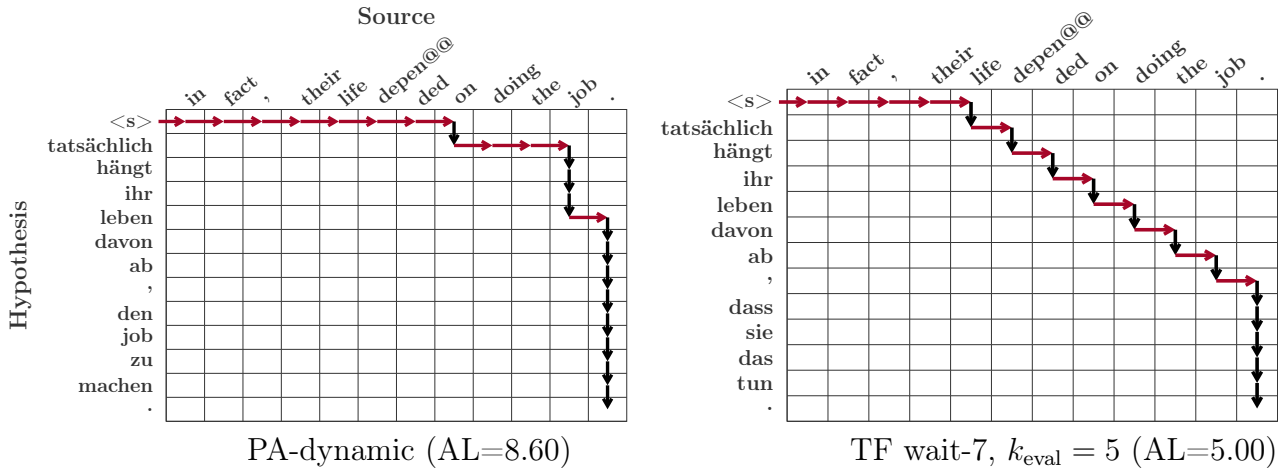


(b) Reference: ich kann die sequenz während sie ab@@läuft um@@stellen .

Figure 4.17: IWSLT'14 En→De: Examples decoded with a dynamic agent ($k = \infty$ initialization, $\lambda = 1$ and evaluated with $\tau = 0.7$, BLEU=20.57, AL=0.77). The same examples are decoded with TF wait-7, $k_{\text{eval}} = 1$ (BLEU=18.69, AL=2.26).



(a) Reference: verschiedene sprecher : dro@@gen@@krieg . massen@@ver@@nicht@@ungs@@waffen . tor@@na@@do .



(b) Reference: denn ihr leben hing davon ab , ob sie den job richtig machten .

Figure 4.18: IWSLT'14 En→De: Examples decoded with a dynamic agent ($k = \infty$ initialization, $\lambda = 1$ and evaluated with $\tau = 0.2$, BLEU=26.98, AL=5.46). The same examples are decoded with TF wait-7, $k_{\text{eval}} = 5$ (BLEU=25.41, AL=5.14).

In the examples of Figures 4.17 and 4.18, we can see that compared to the rigid wait-k decoding, the dynamic agent effectively adapts its decoding path to the input. In 4.17a, the word 'imagine' is translated into the 3-words phrase 'stellen sie sich' while the wait-1 decoder is forced to read one more token after each write inflating its delay. In 4.17b, the agent advances to read the object 'sequence' omitted in the wait-1 translation. Similarly in 4.18b, the agent reads a large portion of the source to understand the key component of the sentence *i.e.* 'doing the job'. With the wait-5, 'doing the job' is mistranslated into 'dass sie das tun' (that they do that). In the example of Figure 4.18a, the agent is segmenting the source and translating a segment at a time. This leads to a considerably lower delay without affecting the translation's quality.

4.6 Related Work

Online statistical MT. After pioneering work from Fügen et al. (2007), Yarmohammadi et al. (2013) and He et al. (2015) proposed methods to increase the alignment monotonicity for statistical machine translation. To allow for a more flexible segmentation, Grissom II et al. (2014) introduced a trainable segmentation module that decides whether to write or read tokens based on the prediction of the final verb and the next token in the source sequence. This work was tailored specifically for translating verb-final languages (*e.g.*, German and Japanese) to verb-medial languages (*e.g.*, English) and used reinforcement learning to optimize the segmentation policy. Similarly, Oda et al. (2015) use a syntax-based statistical translation system to find the optimal segmentation strategy via greedy search and dynamic programming.

Early neural models. One of the first works on online translation to use attention-based sequence-to-sequence models is that of Cho and Esipova (2016), which uses manually designed, non-trainable, waiting criteria that dictate whether the model should READ or WRITE. Jaitly et al. (2016) reads equally-sized chunks of the source sequence and generates output sub-sequences of variable lengths, each ending with a special token marking the end of writing. For training, a single segmentation is chosen to optimize the likelihood of the full output

sequence.

Attempts to learn READ/WRITE policies. Another line of research tries to learn a read/write policy for online decoding. Replicating the HMM word alignment model used in statistical machine translation (Vogel et al., 1996), the segment-to-segment neural transducer model of Yu et al. (2016) integrates a latent alignment variable into an LSTM-based sequence-to-sequence model where the transition probabilities are conditioned on the encoder-decoder hidden states. During training the alignment is marginalized out with a forward-backward algorithm (Rabiner, 1989). However, their approach is used as an alternative to attention-based models, not for low-latency translation. Luo et al. (2017) introduce a recurrent sequence-to-sequence model with binary stochastic decision variables to either emit next output token or advance in encoding the source sequence with a uni-directional LSTM that jointly encodes the partially generated output sequence. Stochastic decisions are optimized using a standard policy gradient RL approach.

Gu et al. (2017b) propose a trainable agent emitting read/write decisions modelled as an RNN fed with the encoder and decoder current hidden states. In their framework, a left-to-right recurrent sequence-to-sequence model is first pre-trained on the full bi-texts and then fine-tuned with policy gradient to optimize a reward balancing translation quality and latency.

Wait-k decoders and their improvements. Dalvi et al. (2018) used a static decoding algorithm that starts with k read operations then alternates between blocks of l write operations and l read operations. This approach outperforms the information based criteria of Cho and Esipova (2016) and allows complete control of the translation delay. More recently, Ma et al. (2019a) trained a sequence-to-sequence model based on the transformer architecture (Vaswani et al., 2017) with an integrated "wait-k" agent that first reads k source tokens then alternate single read-writes, similar to Dalvi et al. (2018) but using $l=1$. Wait-k approaches were found most effective by Zheng et al. (2019a) when trained for the specific k that is used to generate translations. This, however, requires training separate models for each potential value of k used for translation. As an alternative, Zheng et al. (2019b,a) both learn an adaptive policy to produce read/write decisions in a pre-trained offline translation model.

Zheng et al. (2019b) train the model for decoding along two wait-k paths, but unlike our multi-wait-k, they only optimize along the two boundary paths. Zheng et al. (2019a) use supervised training based on an oracle read/write sequence derived from the pre-trained offline translation model. However, read/write probabilities have to be tuned to achieve different latency levels and models do not perform much better than wait-k models. Zheng et al. (2019c) also investigate beam search decoding for wait-k models using a target language model look-ahead.

State-of-the-art monotonic attention. Raffel et al. (2017) propose an alternative to attention using monotonic alignments. Whereas attention requires access to the full source sequence to compute weights, monotonic alignments enable linear time computation of weights and online decoding. Chiu and Raffel (2018) proposed MoChA that extends monotonic attention so that the decoder attends to a chunk of the encoder states rather than focusing only on the last one. Arivazhagan et al. (2019) with MILk opts for an infinite lookback rather than a limited chunk in order to boost the translation quality. Recently, Ma et al. (2020) adapted MILk's monotonic attention for multi-headed Transformer decoders.

4.7 Conclusion

In this chapter, we introduced a common framework for online sequence-to-sequence models with the sequence of context sizes $\boldsymbol{z}$ as a latent variable. In the first part, we adapt Transformer and Pervasive Attention models for the task of online decoding by filtering out future source signal. We showed that, unlike in offline translation, uni-directional encoders perform better than bi-directional ones in online translation. Attempts at limiting the size of the past context to a few tokens prove to be helpful for low-latency training regimes (*e.g.* training wait-1 decoders) with no to little effect on the mid-range delays (*e.g.* training wait-7 decoders). With Pervasive Attention models, aggregating the features with max-pooling is especially efficient for training low delays (*e.g.* wait-1). Overall, we showed that simple wait-k decoding agents can be very effective and match the state-of-the-art dynamic agents such as MoChA and MILk. We also proved that we do not need to train wait-k models for every latency regime as some values of k generalize better in a wide range of laggings. In the second part,

we showed the potential of Pervasive Attention models in the task of online translation, not only under deterministic policies where we can easily optimize multiple decoding paths, but also with dynamic decoding agents supervised with likelihood-based oracles. The grid-nature of the features in Pervasive Attention allows for encoding all the paths at once which leads to an easier comparison between paths and a straight-forward optimization of multiple decoding paths. With our Pervasive attention based dynamic agents we were able to outperform or match the offline models on IWSLT'14 De↔En and IWSLT'15 En↔Vi with up to 70% less lagging.

Chapter 5

Cost Effective Decoding

In this chapter we introduce our anytime structured prediction models for Neural Machine Translation. We prepare Transformer models for anytime prediction *i.e.* yielding an output at different stages of the network with classifiers plugged at multiple exit points. We propose a number of different halting mechanisms to predict the required network depth and train them under the supervision of oracles. We obtain promising trade-offs between speed and accuracy reducing the computational cost by up to 75% without hindering the translation's quality.

The contributions presented in this chapter are published in:

> Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020b. Depth-Adaptive Transformer. In *Proc. of ICLR*

Contents

5.1 Introduction 100

5.2 Anytime Structured Prediction 101

5.3 Adaptive Depth Estimation 107

5.4 Experiments 114

5.5 Related Work 125

5.6 Conclusion 128

5.1 Introduction

We described in §2.6.3 the Transformer model, state-of-the-art in a wide range of NLP tasks (Devlin et al., 2019; Yang et al., 2019; Liu et al., 2019b; Ng et al., 2019). Models based on the Transformer architecture can amount to billions of parameters. For instance, the large version of BERT (Devlin et al., 2019) has 24 blocks and 1024-dimensional features with 16 attention heads amounting to 340M parameters. For translation, the winning entry of the WMT'19 news machine translation task in English-German used an ensemble totaling two billion parameters (Ng et al., 2019). With these large models, training becomes very expensive in terms of computation and memory usage; not only we need to store the model's parameters but also all of the intermediate activations. Some interesting techniques like gradient check-pointing can help reduce the memory usage but at the cost of a much slower training. Computation-wise, a single block is $O(d^2|\boldsymbol{x}| + d|\boldsymbol{x}|^2)$ expensive. With $|\boldsymbol{x}|$ the length of the sequence, computing the $|\boldsymbol{x}| \times |\boldsymbol{x}|$ similarity scores between d-dimensional features costs $O(d|\boldsymbol{x}|^2)$. Additionally, the position-wise key, value and query maps each costs $O(d^2|\boldsymbol{x}|)$. This cost is a huge burden both during training and test time. Larger models are more beneficial for hard examples, where the increased complexity usually leads to better results. However, for easy examples such as translating "Thank you" into "Merci", using these models would be uncalled-for. Current evaluation pipelines for sequence-to-sequence models apply the same amount of computation regardless of whether the input is easy or hard.

In this chapter, we explore Transformer decoders that adapt the amount of computation to each input in order to achieve a satisfying trade-off between speed and accuracy at inference time. We revise the *adaptive computation time* (ACT) algorithm (Graves, 2016), introduced to dynamically halt a computational cycle with recurrent neural networks, in order to adapt the computation in a uni-directional computation graph. Before using ACT, our decoder has to be capable of emitting output distributions at multiple stages of the computation graph. We will describe in §5.2 the different methods we explored to effectively train a decoder with intermediate classifiers. In §5.3, we will detail the halting mechanisms we investigated to control the amount of computation in the decoder, and outline the training pipeline of these mechanisms. Experiments on IWSLT'14 De→En (Cettolo et al., 2014) as well as WMT'14

En→Fr translation tasks reported in §5.4 show that we can match the performance of well tuned baseline models at up to 76% less computation. We will end the chapter with an examination of existing work related to ours (§5.5).

5.2 Anytime Structured Prediction

In this section we will address the problem of anytime prediction as it pertains to the design of the model. Algorithms for *Anytime prediction*, studied in Zilberstein (1996), should be capable of emitting a prediction at anytime and should provide increasingly better results given more computation time with diminishing returns. In order to emit a prediction at any computation stage, the decoder is equipped with auxiliary classifiers and losses at different exit points. Multiple works in computer vision have used early auxiliary classifiers in deep convolutional network. In Lee et al. (2015); Szegedy et al. (2015); Shen et al. (2016), these auxiliary losses were used to speed-up convergence and regularize the model by increasing the discriminativeness of the intermediate features. In Zamir et al. (2016) they were used for curriculum learning (Bengio et al., 2009). More related to our work, they are used for anytime prediction or early exiting Bolukbasi et al. (2017); Teerapittayanon et al. (2016); Wang et al. (2018); Huang et al. (2018). The later, state-of-the-art in anytime prediction for visual object recognition, raised the concern that these auxiliary losses negatively affect the late classifiers. In fact, feature extraction in ConvNets depends on how many layers are left until classification and, with auxiliary losses, the early features are prepared for short-term prediction. Their proposed model, MSDNet, uses dense skip connections to directly reach for low-level features. To train MSDNet, Huang et al. (2018) found that weighting all the losses equally works well in practice. Teerapittayanon et al. (2016) assigned more weights to early losses to encourage earlier exits. Recently, Hu et al. (2019) remarked that early losses have larger gradients that dominate the rest. They proposed adaptive loss balancing, where each loss has a weight inversely proportional to its empirical mean.

These conclusions and intuitions are based on convolutional architectures for object recognition and may not translate to sequence modeling with attention-based models. In §5.2.1 we will introduce auxiliary classifiers to the Transformer decoder and outline two

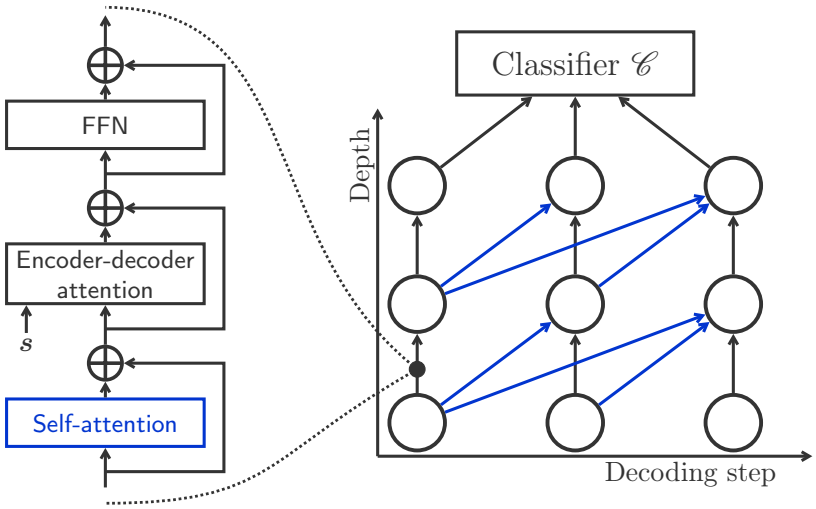


Figure 5.1: Transformer decoder: dependencies through-time (in blue), characteristic of the self-attention sub-block, require the outputs of the previous layer at earlier steps. A decoder block between two consecutive representations is developed on the side. The additional input for the encoder-decoder attention is the sequence of source hidden states $\boldsymbol{s} = (s_1, \ldots, s_{|\boldsymbol{x}|})$.

schemes for training these classifiers, aligned (§5.2.2) and mixed (§5.2.3). We will then explore a few variations of loss scaling and gradient manipulation to improve the overall performance of the model in §5.2.4.

5.2.1 Transformer with Intermediate Classifiers

Recall from §2.6.3 that a Transformer model (Vaswani et al., 2017) has N stacked blocks in both the encoder and decoder branches. Each block has several sub-blocks surrounded by residual skip-connections. The first sub-block is a multi-head dot-product self-attention and the second a position-wise fully connected feed-forward network. For the decoder, there is an additional sub-block after the self-attention to add source context via another multi-head attention. A decoder block is illustrated in Figure 5.1.

Let $(\boldsymbol{x}, \boldsymbol{y})$ be a pair of source-target sequences. The source $\boldsymbol{x}$ is processed with the encoder yielding a sequence of representations $\boldsymbol{s} = (s_1, \ldots, s_{|\boldsymbol{x}|})$. Next, the decoder generates $\boldsymbol{y}$ step-by-step. For every new token y_t input to the decoder, the N decoder blocks process it

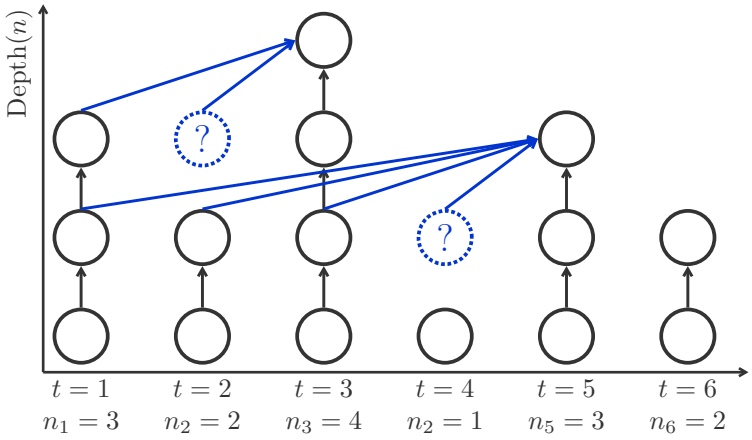


Figure 5.2: Missing inputs for self-attention. Hidden states marked with ? were not computed because the model exited earlier. These missing states are required in the computational graph of subsequent steps (blue edges).

to evaluate the increasingly deep sequence of hidden states $(h_t^n)_{1\leq n\leq \mathrm{N}}$:

$$h_t^0 = \mathrm{E}y_t, \quad h_t^n = \mathrm{block}_n(h_{\leq t}^{n-1}, \boldsymbol{s}) \ \ (\forall n \in [1..\mathrm{N}]), \tag{5.1}$$

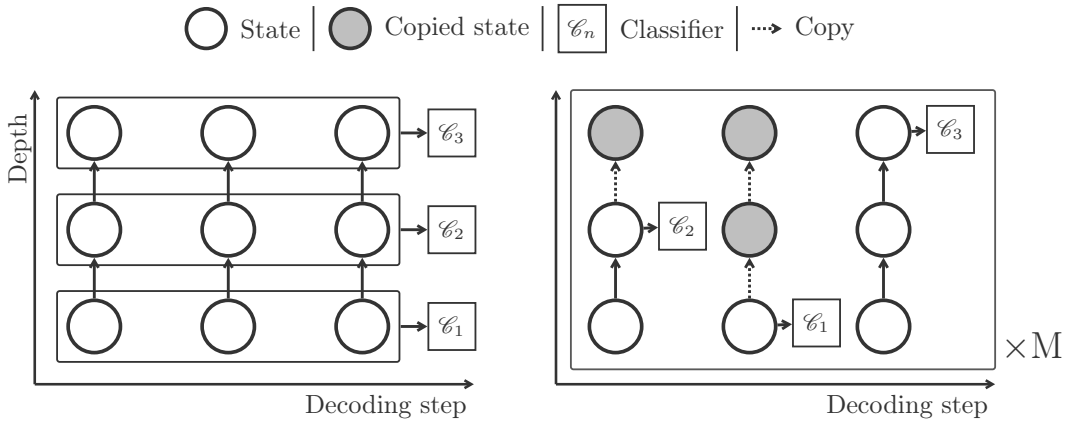
where block_n is the mapping associated with the n-th block and E the target-side embedding matrix. The output distribution for predicting the next token is computed by feeding the activations of the last decoder block h_t^{N} into a softmax-normalized output classifier W:

$$p(y_{t+1} \,|\, h_t^{\mathrm{N}}) = \mathrm{softmax}(\mathrm{W}h_t^{\mathrm{N}}). \tag{5.2}$$

In a standard Transformer decoder, an output classifier is attached to the top of the decoder network. However, for anytime prediction, the model needs to be interruptible *i.e.* capable of emitting a prediction at different exit points. We place our exit points after each block and attach an output classifier $\mathscr{C}_n$ parameterized by W_n to the output h_t^n of block_n. The N classifiers can be parameterized with independent weights $(\mathrm{W}_n)_{1\leq n\leq \mathrm{N}}$ or we can share these weights across the N blocks.

$$\forall n \in [1..\mathrm{N}], \ p(y_{t+1} \,|\, h_t^n) = \mathrm{softmax}(\mathrm{W}_n h_t^n). \tag{5.3}$$

Note that block_n requires not only h_t^{n-1}, the output of the previous block at the current time-step t, but all of its past outputs $h_{<t}^{n-1}$. These past outputs are required by the self-attention sub-block (see Figure 5.1), with its characteristic dependencies through-time. Such



(a) Aligned training (b) Mixed training

Figure 5.3: Training regimes for decoder networks able to emit outputs at any layer. Aligned training optimizes all output classifiers $\mathscr{C}_n$ simultaneously assuming all previous hidden states for the current layer are available. Mixed training samples M paths of random exits at which the model is assumed to have exited; missing previous hidden states are copied from below.

dependencies are present in most sequence models and bring about a particular issue to the design of anytime predictors. If we make a collective decision for the whole sequence, then all of the necessary input at a given time-step is available. However, if we treat each token of the sequence independently when it comes to the exit decision, some inputs will potentially be missing. In the example of Figure 5.2, at each step t the model decides to stop after the n_t-th block. The decision to exit after $n_2 = 2$ in the second step obstructs the computation in the following step at the fourth block. To address the issue of missing inputs, we simply *copy* the last computed state to all subsequent blocks. These copied features will consequently *interact* with high-level features in the self-attention sub-block. Note that each self-attention will still apply its own key and value projections to the copied state.

For the model to appropriately handle the misaligned features at test time, we speculate that it must be exposed to such features in training time as well. Thus, we consider two possible ways to train an anytime decoder (see Figure 5.3): *Aligned* where we simply avoid mixing the states and assume all tokens will exit at the same depth, and *mixed* where, for each input sequence $\boldsymbol{y}$, we sample $|\boldsymbol{y}|$ random exit decisions for the model to follow, thus exposing the model to misaligned states.

5.2.2 Aligned Training

For aligned training, we assume all the hidden states $h_1^{n-1}, \ldots, h_t^{n-1}$ are available so we can properly go through self-attention. In this training mode, we optimize decoding along N constant-depth paths each associated with a classifier $\mathscr{C}_n$ and a loss term $\mathrm{NLL}(n)$ (see Figure 5.3a). The term $\mathrm{NLL}(n)$ is the negative log-likelihood of $\boldsymbol{y}$ given $\boldsymbol{x}$ when we always exit after the n-th block:

$$\mathrm{NLL}(n) = -\sum_{t=1}^{|\boldsymbol{y}|} \log p(y_t \,|\, h_{t-1}^n) = -\sum_{t=1}^{|\boldsymbol{y}|} \text{log-softmax}(\mathrm{W}_n h_{t-1}^n). \tag{5.4}$$

We jointly optimize the N paths with a compound loss $\mathcal{L}_{dec}(\boldsymbol{x}, \boldsymbol{y})$ which is a weighted average of N terms w.r.t. to $(\omega_1, \ldots \omega_{\mathrm{N}})$ in the N$-$1-simplex[(1)]:

$$\mathcal{L}_{dec}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=1}^{\mathrm{N}} \omega_n \, \mathrm{NLL}(n). \tag{5.5}$$

5.2.3 Mixed Training

Aligned training does not expose the model to misaligned states coming from different blocks and interacting within a self-attention operator. This creates a mismatch between training and testing. Mixed training reduces this discrepancy by training the model to use hidden states from different blocks of previous time-steps for self-attention. We sample M different exit sequences $\left(n_1^{(m)}, \ldots n_{|\boldsymbol{y}|}^{(m)}\right)_{1 \leq m \leq \mathrm{M}}$ and for each one we evaluate the negative log-likelihood loss:

$$\mathrm{NLL}(n_1, \ldots n_{|\boldsymbol{y}|}) = -\sum_{t=1}^{|\boldsymbol{y}|} \log p(y_t \,|\, h_{t-1}^{n_t}) = -\sum_{t=1}^{|\boldsymbol{y}|} \text{log-softmax}(\mathrm{W}_{n_t} h_{t-1}^{n_t}). \tag{5.6}$$

When $n_t < \mathrm{N}$, we copy the last evaluated hidden state h_t^n to the subsequent layers so that the self-attention of future time steps can function as usual (see Figure 5.3b). The optimized loss is an average of the M losses.

$$\mathcal{L}_{dec}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{\mathrm{M}} \sum_{m=1}^{\mathrm{M}} \mathrm{NLL}\left(n_1^{(m)}, \ldots, n_{|\boldsymbol{y}|}^{(m)}\right). \tag{5.7}$$

(1) The N$-$1-simplex is defined as $\Delta^{\mathrm{N}-1} = \{(\omega_1, \ldots, \omega_{\mathrm{N}}), \sum_{n=1}^{\mathrm{N}} \omega_n = 1 \wedge \omega_n \geq 0 \, \forall n\}$

5.2.4 Balancing the Losses

Sensitivity to the weights. Prior work in computer vision studied the effect of the weights $(\omega_n)_{1\leq n\leq \mathrm{N}}$ on the quality of the classifiers (Teerapittayanon et al., 2016; Huang et al., 2018; Hu et al., 2019), each reaching a different conclusion. In this work, we consider five different constant scales for the aligned training: $\omega_n = 1/\mathrm{N}$ *i.e.* equally weighted losses, $\omega_n \propto n$ and $\omega_n \propto n^{1/2}$ assigning higher weights to later exits and $\omega_n \propto n^{-1/2}$ and $\omega_n \propto n^{-1}$ assigning more weights to the early exits.

We empirically found that uniform weights achieve higher accuracies on average compared to the other weighting schemes that tend to favor a range of exits (see §5.4.2).

Manipulating the gradients. Two major concerns arise from the joint training of N classifiers plugged at varying depths of the same neural network. The first is that early classifiers receive early, close to raw, features leading to larger losses and consequently larger gradients. The second concern is that subsets of the model's parameters receive more training signals than other. In fact, the first block is trained by N loss terms while the last block is optimized by a single loss term. With these two factors, the parameters of block$_n$, denoted with θ_n wind up possessing much larger gradients in comparison to the subsequent blocks.

To balance the gradients of each block in the decoder, we regulate the gradients due to each loss term as they back-propagate through different sections of the computation graph. Let us consider an aligned training with equal weights ($\omega_n = 1/\mathrm{N}$). When updating θ_n, the parameters of the n-th block, we use the compound gradients:

$$\nabla_{\theta_n}\mathcal{L}_{dec} = \frac{1}{\mathrm{N}}\sum_{n'=n}^{\mathrm{N}} \nabla_{\theta_n}\mathrm{NLL}(n'). \tag{5.8}$$

We are interested in calibrating the ratio of the n-th gradient to the subsequent $\mathrm{N}-n$ gradients when updating θ_n. To this end, we scale the gradient $\nabla_{\theta_n}\mathrm{NLL}(n)$ by $\gamma_n = \gamma(\mathrm{N}-n)$ during the update of θ_n. Note that this scaling only affects θ_n as we revert back to the normal scale before back-propagating to the previous blocks. The choice $\gamma_n = \gamma(\mathrm{N}-n)$ allows us to tune the aforementioned ratio at γ:1. Figure 5.4 illustrates the computation graph under gradient manipulation. In Algorithm 2 we show how this gradient manipulation is practically achieved

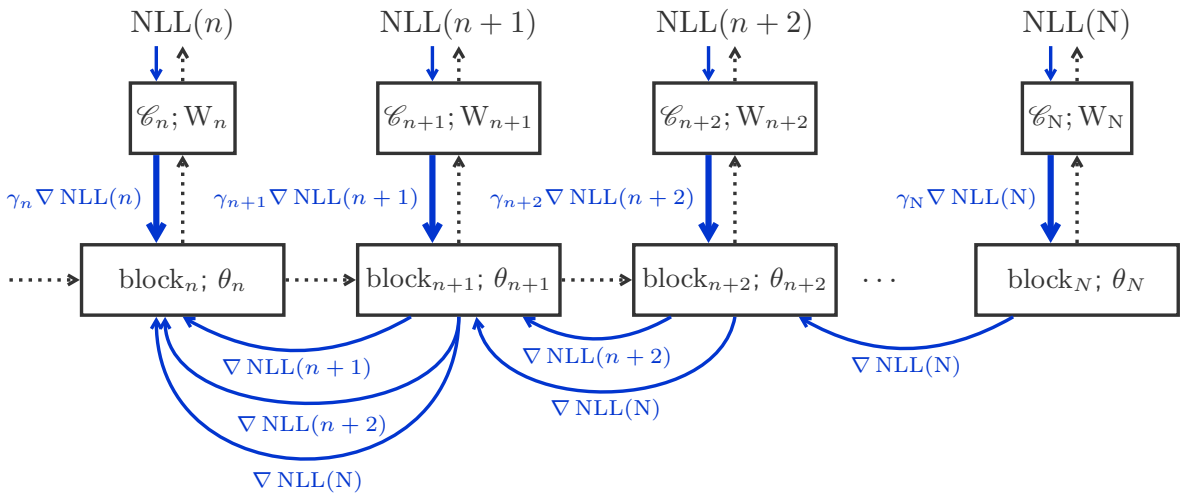


Figure 5.4: Illustration of the computation graph with gradient manipulation. The forward pass flow is shown in dotted edges and the backward pass in blue edges. The manipulated edges are the thick edges of the graph where the gradient is scaled by γ_n before entering the n-th block.

by simply tweaking the forward-pass.

Algorithm 2 Pseudo-code for gradient manipulation in the forward pass

for $n \in 1..\mathrm{N}$ **do**
  $h^n = \mathrm{block}_n(h^{n-1})$.
  $h^n = \textsc{scale\_gradient}\,(h^n, \gamma_n)$.
  $p(\boldsymbol{y} \mid h^n) = \mathrm{softmax}\,(\mathrm{W}_n h^n)$
  ▷ Revert gradients flowing from the next block to normal scale.
  if $n < \mathrm{N}$ **then** $h^n = \textsc{scale\_gradient}\left(h^n, \frac{1}{\gamma_{n+1}}\right)$
end for
function SCALE_GRADIENT(Tensor x, scale γ)
  return $\gamma x + (1-\gamma)\textsc{stop\_gradient}(x)$.
end function
▷ STOP_GRADIENT in PyTorch by detaching the variable from the graph with x.detach().

5.3 Adaptive Depth Estimation

Now that our decoder is equipped with auxiliary classifiers that allow it to be interruptible, we focus on the decision to be made at every exit point, either to continue or to exit. In this

work, we consider a variety of mechanisms to predict the decoder block at which the model will stop and output the next token, or when it should exit to achieve a good speed-accuracy trade-off. We consider two approaches: *sequence-specific depth* decodes all the tokens in $\boldsymbol{y}$ by exiting at the same block (§5.3.1) while *token-specific depth* determines an independent exit for each individual token (§5.3.2).

In both paradigms, we model the distribution of exiting at time-step t with a parametric distribution $q_t \in \mathcal{Q}$ where $q_t(n)$ is the probability of forwarding y_t up to the n-th block and then emitting a prediction with $\mathscr{C}_n$. The parameters of q_t are optimized with the supervision of an oracle distribution q_t^* by minimizing the cross-entropy $\mathrm{H}(q_t^*, q_t)$. At the sequence-level, the following compound loss is optimized:

$$\mathcal{L}_{exit}(\boldsymbol{x}, \boldsymbol{y}) = \sum_{t=1}^{|\boldsymbol{y}|} \mathrm{H}(q_t^*, q_t)\,. \tag{5.9}$$

Depending on the family of distributions considered $\mathcal{Q}$, the exit loss ($\mathcal{L}_{\text{exit}}$) can be back-propagated to the encoder and or decoder parameters. After pre-training the encoder-decoder model with the auxiliary prediction losses, we optimize $\mathcal{L}_{exit}$ jointly with the decoding loss (Eq. (5.5)). The compound loss is balanced with a hyper-parameter α:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = \mathcal{L}_{dec}(\boldsymbol{x}, \boldsymbol{y}) + \alpha \mathcal{L}_{\text{exit}}(\boldsymbol{x}, \boldsymbol{y}), \tag{5.10}$$

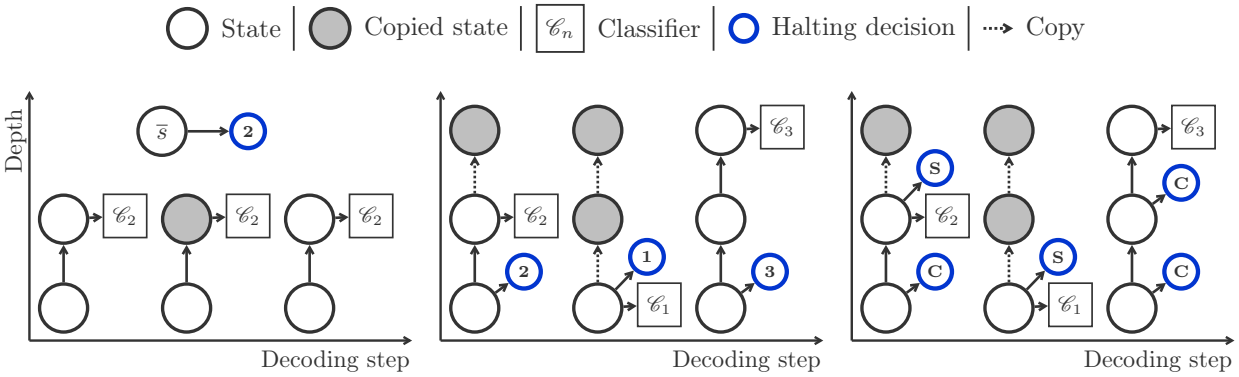
In the following we describe for each approach the family of distributions $\mathcal{Q}$ we consider when optimizing $\mathcal{L}_{exit}$ (see Figure 5.5) and the supervising oracle q_t^*.

5.3.1 Sequence-specific Depth

For sequence-specific depth, the exit distribution $q \in \mathcal{Q}$ and the oracle distribution q^* are independent of the time-step so we drop the subscript t. We choose $\mathcal{Q}$ as the family of softmax-normalized linear classifiers on the average source hidden states. That is:

$$\overline{s} = \frac{1}{|\boldsymbol{x}|} \sum_{t=1}^{|\boldsymbol{x}|} s_t, \quad q(n \,|\, \boldsymbol{x}) = \mathrm{softmax}(\mathrm{W}_q \overline{s} + b_q), \tag{5.11}$$

where W_q and b_q are the weights and biases of the linear classifier. Any other sequence-level representation of the source can be used to condition the classifier; initial experiments with



(a) Sequence-specific depth (b) Token-specific - Multinomial (c) Token-specific - Geometric-like

Figure 5.5: Variants of the adaptive depth prediction classifiers. Sequence-specific depth uses a multinomial classifier to choose an exit for the entire output sequence based on the encoder output s (left panel). It then outputs a token at this depth with classifier $\mathscr{C}_n$. The token-specific multinomial classifier determines the exit after the first block and proceeds up to the predicted depth before outputting the next token (middle panel). The token geometric-like classifier (right panel) makes a binary decision after every block to dictate whether to continue (C) to the next block or to stop (S) and emit an output distribution.

average-pooling, max-pooling and a special-token representation (Devlin et al., 2019) show a slight superiority of the average state. Inference with sequence-specific depth is illustrated in Figure 5.5a.

For q^*, we consider two oracles to determine which of the N blocks should be selected for exit. The first is based on the sequence likelihood and the second looks at an aggregate of the correctly predicted tokens at each block. Oracles based on test metrics such as BLEU are feasible but expensive to compute since we would need to decode every training sentence N times. We leave this for future work.

Likelihood-based. For this oracle, we evaluate the likelihoods of the target sequence $\boldsymbol{y}$ given the source $\boldsymbol{x}$ if decoded with the different predictions emitted at all the exit points. The exit achieving the highest likelihood (or equivalently log-likelihood) is said to be the ideal exit and q^* is simply the discrete distribution represented with the Kronecker delta

centered on it.

$$\mathrm{LL}(n) = \sum_{t=1}^{|\boldsymbol{y}|} \log p(y_t \mid h_{t-1}^n), \quad n^*(\boldsymbol{x}, \boldsymbol{y}) = \arg\max_n \mathrm{LL}(n), \tag{5.12}$$

$$q^*(n \mid \boldsymbol{x}, \boldsymbol{y}) = [\![n = n^*(\boldsymbol{x}, \boldsymbol{y})]\!], \tag{5.13}$$

where we use the Iverson brackets to represent the delta distribution:

$$[\![n = n^*]\!] = \begin{cases} 1, & \text{if } n = n^* \\ 0, & \text{otherwise.} \end{cases} \tag{5.14}$$

If we assume that we are likely to get better results with increasing depths, then this oracle will, more often than not, settle on the latest exit. Therefore, instead of scoring the exits purely on their accuracies, we add a regularization term to penalize late exits ($\lambda \geq 0$):

$$n^*(\boldsymbol{x}, \boldsymbol{y}) = \arg\min_n \left(\mathrm{LL}(n) - \lambda n \right). \tag{5.15}$$

Note that q^* as an oracle is only used in training time; it requires the full training input $(\boldsymbol{x}, \boldsymbol{y})$ in order to score the exits.

Correctness-based. In the previous oracle, we relied on the likelihood as a measure of the quality of an exit. However, likelihood ignores how the classifier is sampled during evaluation. The correct token could be sampled by virtue of being the argmax of the distribution, but the confidence of the model and therefore the likelihood score is low. This is often the case if the model is undecided between synonyms or different word orders.

Instead, in this oracle we score the exits by the number of *correct* predictions made. For each block, we count the number of correctly predicted tokens over the sequence and choose the block with the most number of correct tokens. If tied, the earliest highest scoring block is selected. Similarly to the previous oracle, a regularization term controls the trade-off between speed and accuracy.

$$\mathrm{C}(n) = \left| \{ t \mid y_t = \arg\max_{\widetilde{y}_t} p(\widetilde{y}_t \mid h_{t-1}^n) \} \right|, \tag{5.16}$$

$$n^*(\boldsymbol{x}, \boldsymbol{y}) = \arg\max_n \mathrm{C}(n) - \lambda n, \quad q^*(n \mid \boldsymbol{x}, \boldsymbol{y}) = [\![n = n^*(\boldsymbol{x}, \boldsymbol{y})]\!]. \tag{5.17}$$

5.3.2 Token-specific Depth

The token-specific approach can choose a different exit at every time-step. We consider three options for the family of distributions $\mathcal{Q}$. The first is basic confidence thresholding, where after each block a prediction is made with the classifier $\mathscr{C}_n$ and depending on its quality we decide whether to accept it or wait for a better prediction with deeper features. The second is softmax-normalized linear classifiers on the first decoder hidden state h_t^1. And the third is geometric distributions but with different success rates estimated from the current decoder hidden state h_t^n. The two last models are dubbed *multinomial* and *geometric-like*.

Confidence thresholding. Here, we base our exit policy on the quality of the predictions made at each exit point. The quality of a prediction is assessed through the classifier's confidence *i.e.* its maximum score

$$\text{Confidence}(n,t) = \max_{\widetilde{y}_{t+1}\in\mathcal{V}} p(\widetilde{y}_{t+1} \,|\, h_t^n). \tag{5.18}$$

If the confidence exceeds a pre-determined threshold τ_n, we accept the prediction and exit. With this exit policy, the thresholds $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{\mathrm{N}-1})$ are simply tuned on the development set in order to meet the required accuracy-speed trade-off. Concretely, for a large number of iterations (in the order of 10k), we sample a sequence of thresholds $\boldsymbol{\tau} \sim \mathcal{U}(0,1)^{\mathrm{N}-1}$ and decode the development set with the associated exit policy. We then evaluate the accuracy as a BLEU score and the computational cost of each sampled $\boldsymbol{\tau}$. At the end we pick the best performing thresholds, that is $\boldsymbol{\tau}$ with the highest accuracy in various budget ranges.

Multinomial. Similar to the distributions considered for sequence-specific depth, here an affine map with parameters (W_q, b_q) projects the first block hidden state to $\mathbb{R}^{\mathrm{N}}$, and a softmax operator normalizes the scores into a proper distribution.

$$q_t(n \,|\, \boldsymbol{x}, \boldsymbol{y}_{<t}) = \text{softmax}(\mathrm{W}_q h_t^1 + b_q). \tag{5.19}$$

The use of h_t^1 as a feature for the classifier is justified by the fact that the first block is mandatory; the earliest exit being right after this block.

During inference, the decoder computes it first-level state and the classifier estimates q_t, the probability of each exit. The most probable exit $n_t^* = \arg\max q_t(n \,|\, \boldsymbol{x}, \boldsymbol{y}_{<t})$ is selected and

the decoder continues its forward pass up to n_t^* before emitting a prediction. This inference is illustrated in Figure 5.5b.

Geometric-like. Instead of committing to an exit decision right after the first block, with the geometric-like variant an exit probability is estimated after each block. At time-step t, the n-th exit probability, denoted with χ_t^n is estimated as an affine transformation of the current decoder hidden state h_t^n.

$$\forall n \in [1..\mathrm{N}{-}1],\ \chi_t^n = \mathrm{sigmoid}(w_q^\top h_t^n {+} b_q), \tag{5.20}$$

where w_q and b_q are parameters of the affine map. χ_t^n is the chance of successfully exiting right after the n-th block. Similar to a geometric distribution, $q_t(n)$ is the probability that the previous $n-1$ trials fail and the n-th succeeds. That is:

$$q_t(n \,|\, \boldsymbol{x}, \boldsymbol{y}_{<t}) = \begin{cases} \chi_t^n \prod_{n'=1}^{n-1} (1 - \chi_t^{n'}), & \text{if } n < \mathrm{N}, \\ \prod_{n'=1}^{\mathrm{N}-1} (1 - \chi_t^{n'}), & \text{otherwise.} \end{cases} \tag{5.21}$$

During inference, we advance in the decoder and after each block we estimate the exit probability χ_t^n. If this exit probability exceeds a pre-determined threshold τ_n then an output prediction is made and emitted, otherwise we continue in the decoder network. If no exit decision is made throughout the decoder forward pass then we default to exiting after the integral N blocks. These thresholds $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_{\mathrm{N}-1})$ are tuned on the development set to achieve the desired accuracy-speed trade-off.

As with sequence-specific depth, in the case of multinomial and geometric-like exit distributions, a supervising oracle is needed to estimate the parameters of $q \in \mathcal{Q}$. We consider two oracles, the first is based on a likelihood score and the second on the correctness count.

Likelihood-based. At each time-step t, we choose the block whose prediction has the highest likelihood. A regularization term is added to penalize late exits ($\lambda \geq 0$):

$$\mathrm{LL}(n, t) = \log p(y_t \,|\, h_{t-1}^n), \tag{5.22}$$

$$n_t^*(\boldsymbol{x}, \boldsymbol{y}) = \arg\max_n \mathrm{LL}(n, t) - \lambda n, \quad q_t^*(n|\boldsymbol{x}, \boldsymbol{y}) = [\![n = n_t^*(\boldsymbol{x}, \boldsymbol{y})]\!]. \tag{5.23}$$

As it is, this oracle is myopic, focusing on a single time-step. A decision at time-step t, in particular, could affect later predictions. We therefore consider smoothing the likelihood scores with an RBF kernel. Instead of only looking at future time-steps, we consider a symmetric kernel including past time-steps too.

$$\kappa(t, t') = e^{-\frac{|t-t'|^2}{\sigma}}, \quad \widetilde{\mathrm{LL}}(n, t) = \sum_{t'=1}^{|\boldsymbol{y}|} \kappa(t, t')\, \mathrm{LL}(n, t'). \tag{5.24}$$

The size of the surrounding context included in this smoothing is controlled with the kernel width σ. If $\sigma \to +\infty$, we revert to scoring each exit with the sequence-level likelihood used in sequence-specific depth. On the other end of the spectrum, if $\sigma \to 0$ then we are scoring with the likelihood of the current step. As with other scores, these smoothed likelihoods are regularized and q^* is centered on the highest scoring exit.

$$n_t^*(\boldsymbol{x}, \boldsymbol{y}) = \arg\min_n \widetilde{\mathrm{LL}}(n, t) - \lambda n. \tag{5.25}$$

Correctness-based. Similar to the likelihood-based oracle we can look at the correctness of the prediction at time-step t and smooth the scores by integrating surrounding positions. In which case, q_t^* is evaluated as follows:

$$\mathrm{C}(n, t) = [\![y_t = \arg\max_{\widetilde{y}_t} p(\widetilde{y}_t \mid h_{t-1}^n)]\!], \tag{5.26}$$

$$\kappa(t, t') = e^{-\frac{|t-t'|^2}{\sigma}}, \quad \widetilde{\mathrm{C}}(n, t) = \sum_{t'=1}^{|\boldsymbol{y}|} \kappa(t, t')\ \mathrm{C}(n, t), \tag{5.27}$$

$$n_t^*(\boldsymbol{x}, \boldsymbol{y}) = \arg\max_n \widetilde{\mathrm{C}}(n, t) - \lambda n, \tag{5.28}$$

$$q_t^*(n|\boldsymbol{x}, \boldsymbol{y}) = [\![n = n_t^*(\boldsymbol{x}, \boldsymbol{y})]\!]. \tag{5.29}$$

An adaptive model is based on the choice of the family modeling the exit distribution and the oracle supervising the training of said distribution. With the different choices introduced in this section, we can train six different combinations summarized in Table 5.1.

Nomenclature	Exit distribution	Oracle score
Seq-LL(λ)	Sequence-specific	$\mathrm{LL}(n) - \lambda n$
Seq-C(λ)	Sequence-specific	$\mathrm{C}(n) - \lambda n$
Tok-LL(λ, σ) Geometric-like	Token-specific, Geometric-like	$\widetilde{\mathrm{LL}}(n,t) - \lambda n$
Tok-LL(λ, σ) Multinomial	Token-specific, Multinomial	$\widetilde{\mathrm{LL}}(n,t) - \lambda n$
Tok-C(λ, σ) Geometric-like	Token-specific, Geometric-like	$\widetilde{\mathrm{C}}(n,t) - \lambda n$
Tok-C(λ, σ) Multinomial	Token-specific, Multinomial	$\widetilde{\mathrm{C}}(n,t) - \lambda n$

Table 5.1: Summary of exit distributions and oracles for adaptive depth estimation

5.4 Experiments

In this section we first evaluate our methods for training a Transformer model with multiple output classifiers then compare the different models for adaptive depth estimation. Initial experiments were run on IWSLT'14 En→De and our contributions were then validated on the large benchmark of WMT'14 En→Fr.

5.4.1 Experimental Setup

IWSLT'14 De→En. We replicate the setup of Edunov et al. (2018) and remove sentences longer than 175 words and pairs with length ratio $\max(|\boldsymbol{x}|/|\boldsymbol{y}|, |\boldsymbol{y}|/|\boldsymbol{x}|)$ exceeding 1.5 were removed from the original data. A development set of 7K pairs is extracted from the training set and final results are reported on a test set of 6.5K pairs obtained by concatenating dev2010 and tst2010-2013. We tokenized and lower-cased all data using the standard scripts from the Moses toolkit (Koehn et al., 2007). For open-vocabulary translation, we segment sequences using byte pair encoding with 10K merge operations on the merged bi-texts. This results in a German and English vocabularies of around 8.8K and 6.6K types respectively.

We train a *small* variant of Transformer with $\mathrm{N} = 6$ blocks in the encoder and decoder, $n_{\mathrm{H}} = 4$ attention heads and a per-position feed-forward network with $d_{\mathrm{FF}} = 1024$. We set the dropout rate to $p_{\mathrm{drop}} = 0.3$. The encoder's embedding dimension is 512 and the decoder's

is 256. Embeddings are untied with 6 different output classifiers $\{\mathrm{W}_n\}_{1\leq n\leq 6}$. We evaluate with a single checkpoint, a beam of width 5 and a length penalty of 1.

WMT'14 En→Fr. This large benchmark has 35.5M training pairs from which 26K are held out for development. Final results are reported on the test set of newstest14. The vocabulary consists of 44K joint BPE types. We train a Transformer *big* architecture and tie the embeddings of the encoder, the decoder and the output classifiers $(\mathrm{W}_n)_{1\leq n\leq 6}$. We average the last ten checkpoints and use a beam of width 4 and length penalty of 0.6.

For both benchmarks we evaluate a model by measuring tokenized BLEU. Models are optimized with Adam (Kingma and Ba, 2015). We train for 50K updates[(2)] on 128 GPUs with a batch size of 460K tokens for WMT'14 En→Fr and on 2 GPUs with 8K tokens per batch for IWSLT'14 De→En. To stabilize training, we re-normalize the gradients if their norm exceeds $g_{\text{clip}} = 3$.

To train adaptive exits models, we first pre-train the encoder-decoder model *i.e.* $\alpha = 0$ in the compound loss of Eq. (5.10). The output classifiers are jointly trained under the aligned paradigm (see §5.2.2) for 50K updates. This pre-trained model serves as initialization for all of our adaptive models as we optimize the exit distribution parameters and finetune the encoder-decoder with $\alpha = 1$. This finetuning stage lasts for another 3K updates.

5.4.2 Training Multiple Output Classifiers

We first compare the two training regimes for anytime prediction (§5.2.1). Aligned training performs self-attention on aligned states and mixed training exposes self-attention to hidden states from different blocks. As baselines we show six separate standard Transformers with $\mathrm{N} \in [1..6]$ decoder blocks. All models are trained with an equal number of updates and mixed training with $\mathrm{M} = 6$ paths is most comparable to aligned training since the number of losses per sample is identical. Two criteria are selected for evaluation, in the first we simply evaluate each exit $n \in [1..6]$ independently. To do so, we force all samples to exit at the same point similar to a baseline with exactly n decoder blocks. In the second evaluation criterion, we sample a random exit $n \sim \mathcal{U}([1..6])$ for every decoded token. Evaluation with sampled

[(2)] The equivalent of 78 epochs on IWSLT'14 En→De and 17 epochs on WMT'14 En→Fr.

	Uniform	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	Average
Baseline	-	34.17	35.27	35.58	35.70	35.55	35.93	35.37
Aligned ($\omega_n = 1/\text{N}$)	**35.48**	34.11	**35.45**	**35.84**	**36.07**	**36.08**	**36.18**	**35.62**
Mixed M = 1	34.12	32.86	34.26	34.47	34.48	34.55	34.49	34.19
Mixed M = 3	35.09	33.89	35.15	35.40	35.45	35.46	35.54	35.15
Mixed M = 6	35.30	**34.18**	35.35	35.80	35.89	35.82	35.94	35.50

Table 5.2: IWSLT'14 De→En development set: aligned vs. mixed training. We report BLEU scores for a uniformly sampled exit $n \sim \mathcal{U}([1..6])$ at each token, a fixed exit $n \in [1..6]$ for all tokens, as well as the average BLEU over the fixed exits. As baseline we show six standard Transformer models with 1-6 blocks.

exits tests the robustness to mixed hidden states and the fixed exit evaluation simulates an ideal setting where all previous states are available.

To select the best pre-training strategy for anytime prediction, we compare the models based on their performance on a development set. Table 5.2 shows that aligned training outperforms mixed training both for fixed exits as well as for randomly sampled exits. The latter is surprising since aligned training never exposes the self-attention mechanism to hidden states from other blocks. We suspect that this is due to the residual connections which *copy* features from lower blocks to subsequent layers and which are ubiquitous in Transformer models. Aligned training also performs very competitively to the individual baseline models.

Aligned training is conceptually simple and fast. We can process a training sample with N exits in a single forward/backward pass while M passes are needed for mixed training. On our largest setup, WMT'14 En→Fr, the training time of an aligned model with six output classifiers increases only marginally by about 1% compared to a baseline with a single output classifier, keeping everything else equal. In the remainder of this work, we use the aligned mode to pre-train our anytime prediction models.

Balancing the losses. With the aligned model, we experiment with different weights for scaling the output classifier losses. Instead of uniform weighting, we bias towards a range of exits, early or late, by assigning higher weights to their losses. Table 5.3 shows that weighing the classifiers equally provides the best performance on average. Adding more weight to the first exit with $\omega_n \propto n^{-1}$ gives the best performance at that exit. In fact unless this early

	Uniform	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	Average
Baseline	-	34.17	35.27	35.58	35.70	35.55	35.93	35.37
$\omega_n = 1/\mathrm{N}$	35.48	34.11	**35.45**	**35.84**	36.07	36.08	36.18	**35.62**
$\omega_n \propto n$	35.28	32.19	35.00	35.79	35.98	**36.22**	**36.27**	35.24
$\omega_n \propto n^{1/2}$	35.35	33.30	35.23	35.80	35.92	36.05	36.10	35.40
$\omega_n \propto n^{-1/2}$	**35.62**	34.51	35.40	35.70	35.84	35.82	35.85	35.52
$\omega_n \propto n^{-1}$	35.30	**34.69**	35.32	35.49	35.71	35.76	35.78	35.46

Table 5.3: IWSLT'14 De→En development set: aligned training with different weights (ω_n).

exit is given a large weight, its performance is worse than a baseline with a single decoder block *i.e.* the deep supervision hurts the first block. Our conclusions on the first block do not translate to the other early exits $n = 2$ or $n = 3$. These exits are barely affected by the weight shifting from early to late. Adding more weight to the late exits improves their performance by up to 0.3 BLEU points. However, this comes at the cost of the early classifiers where we loose up to 2 BLEU points. A possible explanation of this dynamic is that, unlike with convolutions, the first block can abruptly compute sequence-level features from the granular token embeddings with unbounded self-attention. If a classifier is put on top of the first block, then its features have to be coarse to improve the classifier's accuracy. If optimized w.r.t. late supervision, the first self-attention can choose to compute finer features.

Gradient manipulation. Recall that with gradient manipulation we regulate at $\gamma : 1$ the ratio of the n-th gradient to the subsequent $\mathrm{N}-n$ gradients before updating the parameters of the n-th block. With that, $\gamma > 1$ favors the block's own classifier when updating its parameters and $\gamma < 1$ relies on deeper signal.

Table 5.4 shows that $\gamma = 1.1$ benefits the first exit, achieving the highest BLEU so far with a single block, to the detriment of later exits. Once more, similar to loss balancing, optimizing with regular, unaffected gradients has the best performance on average.

5.4.3 Adaptive Depth Estimation

Next, we train models with adaptive depth estimation, all initialized with an aligned model with equal weights. These models are compared in terms of accuracy (BLEU) and com-

	Uniform	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	Average
Baseline	-	34.17	35.27	35.58	35.70	35.55	35.93	35.37
$\emptyset$	**35.48**	34.11	**35.45**	**35.84**	**36.07**	**36.08**	**36.18**	**35.62**
$\gamma = 0.3$	35.14	33.66	34.66	35.34	35.70	35.97	36.03	35.23
$\gamma = 0.5$	35.44	34.77	35.36	35.58	35.63	35.66	35.63	35.44
$\gamma = 0.7$	34.88	34.64	35.07	35.12	35.23	35.38	35.34	35.13
$\gamma = 0.9$	34.87	34.81	35.26	35.28	35.32	35.41	35.46	35.26
$\gamma = 1.1$	35.13	**34.93**	35.20	35.30	35.29	35.26	35.33	35.22

Table 5.4: IWSLT'14 De→En development set: aligned training with different gradient manipulation ratios $\gamma : 1$.

	Uniform	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	Average
Baseline		**33.39**	34.32	34.56	34.51	34.43	34.77	34.33
Aligned	34.38	33.20	**34.35**	**34.80**	**34.91**	**34.95**	**34.86**	**34.51**

Table 5.5: IWSLT'14 De→En test set: BLEU scores of N independent standard transformers and the anytime prediction model.

putational cost. We measure the latter as the average exit per output token (AE). We average at the corpus level and so, for sequence-specific depth, the exit is weighted with the length of the hypothesis $\widetilde{\boldsymbol{y}}$. As baselines we use again six separate standard Transformers with N $\in$ [1..6] with a single output classifier. We also include the scores of the pre-trained anytime prediction model for fixed exits $n \in [1..6]$. The BLEU scores of these two baselines are additionally reported in Table 5.5.

We train the six combinations described in Table 5.1 with different hyper-parameters λ and σ when relevant. These hyper-parameters are tuned on the development set for a range of budgets, we specifically partition the full range of costs $[1, 6]$ into 0.5-wide segments. At the end, we report results on the test set.

Figure 5.6 shows that the aligned model (blue line) can match the accuracy of a standard 6-block Transformer (black line) at half the number of layers ($n = 3$) by always exiting at the third block. The aligned model outperforms the baseline for $n \in [2..6]$.

For token specific halting mechanisms, the geometric-like classifiers in Figure 5.6b achieve

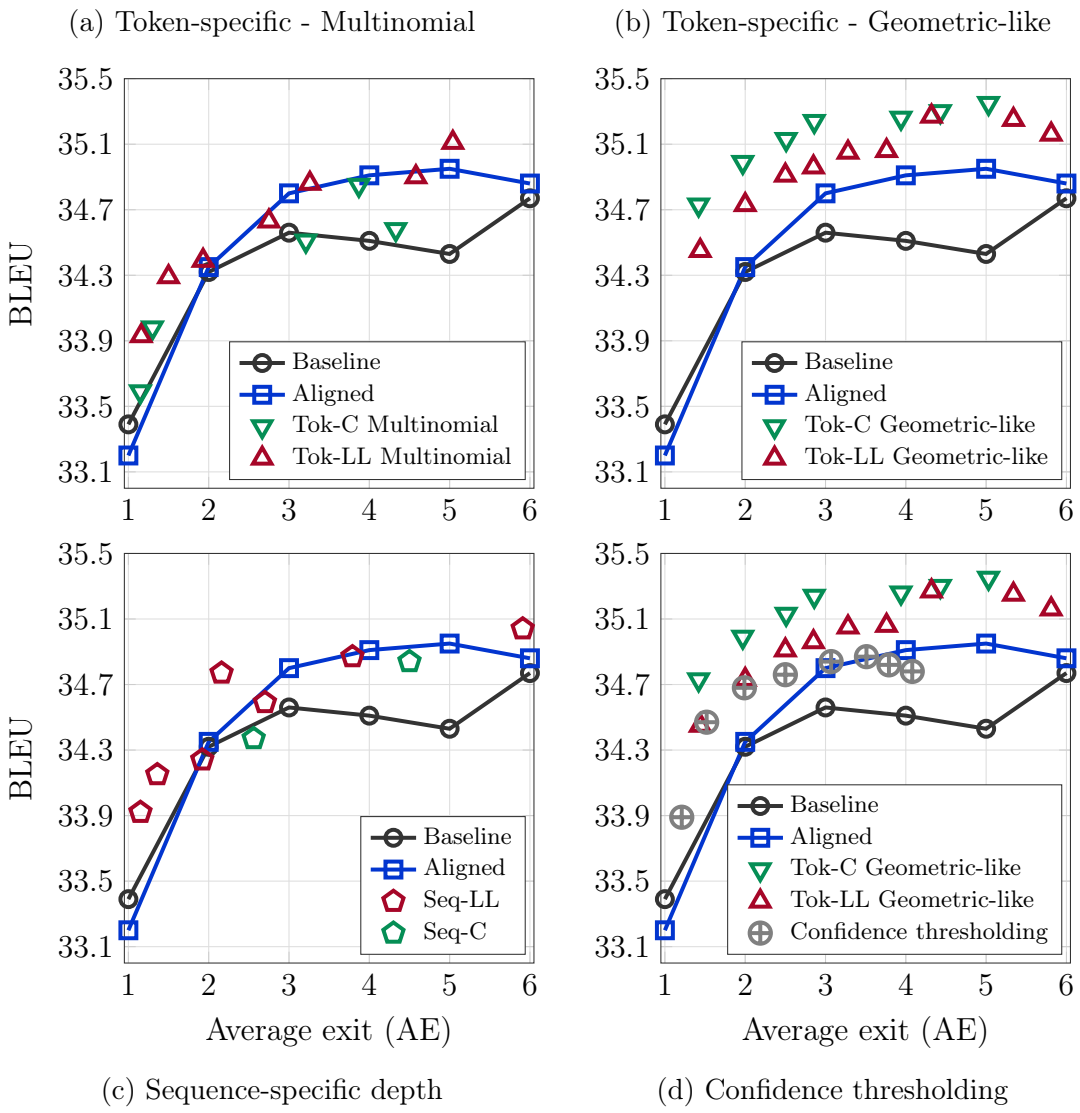
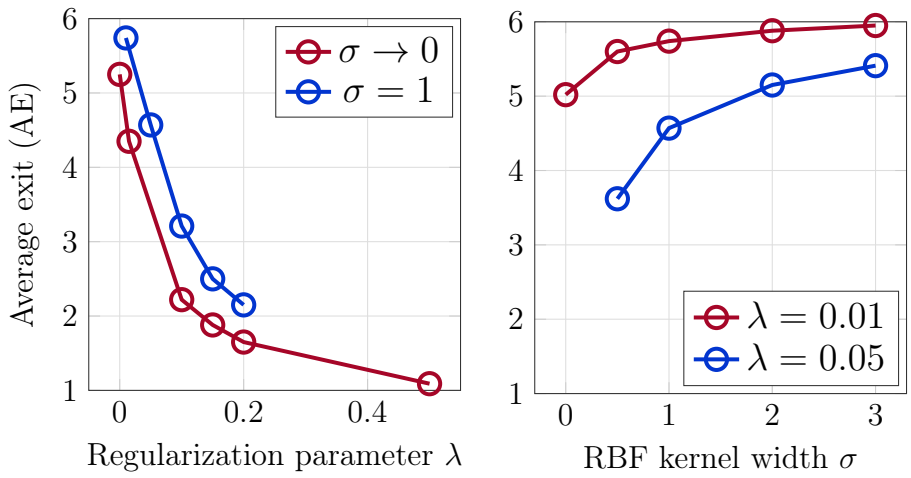


Figure 5.6: IWSLT14 De→En test set: trade-off between speed (average exit or AE) and accuracy (BLEU) for depth-adaptive methods.

a better speed-accuracy trade-off than the multinomial classifiers in Figure 5.6a. This is largely due the cascaded decisions and the additional tuning step to select the thresholds $(\tau_1, \ldots, \tau_5)$. With geometric-like classifiers, the correctness oracle (Tok-C) outperforms the likelihood oracle (Tok-LL) in 5.6b but the trend is less clear for multinomial classifiers. At the sequence-level, Figure 5.6c shows that likelihood is the better oracle. Note that if no marker is present in a budget range, it means that the model does not operate at this range



(a) Effect of λ on AE (b) Effect of σ on AE

Figure 5.7: IWSLT'14 De→En development set: effect of the hyper-parameters σ and λ on the average exit (AE).

despite considering different values for λ and σ.

The leftmost Tok-C geometric-like point achieves 34.73 BLEU at AE = 1.42 which corresponds to similar accuracy as the $n = 6$ baseline at 76% fewer decoding blocks. The best accuracy of the aligned model is 34.95 BLEU at exit 5 and the best comparable Tok-C geometric-like configuration achieves 34.99 BLEU at AE = 1.97, or 61% fewer decoding blocks. When fixing the budget to two decoder blocks, Tok-C geometric-like with AE = 1.97 achieves BLEU 34.99, a 0.64 BLEU improvement over the baseline (BLEU=34.32) and the aligned model (BLEU=34.35) at $n = 2$. Confidence thresholding (Figure 5.6d) performs very well but cannot outperform Tok-C geometric-like. Moreover, its accuracy dips after the exit $n = 3$.

Ablation of hyper-parameters. In this section, we look at the effect of the two main hyper-parameters: λ the regularization scale (see Eq. (5.23)), and the RBF kernel width σ used to smooth the scores (see Eq. (5.24)). We train Tok-LL Geometric-like models and evaluate them with thresholds τ_n defaulting to 0.5 *i.e.* we exit if χ_t^n exceeds the half-way point 0.5. Figure 5.7a shows that higher values of λ naturally lead to lower exits. Figure 5.7b shows the effect of σ for two values of λ. In both curves, we see that with wide kernels *i.e.* coarse scores close to the sequence-level, late exits are favored.

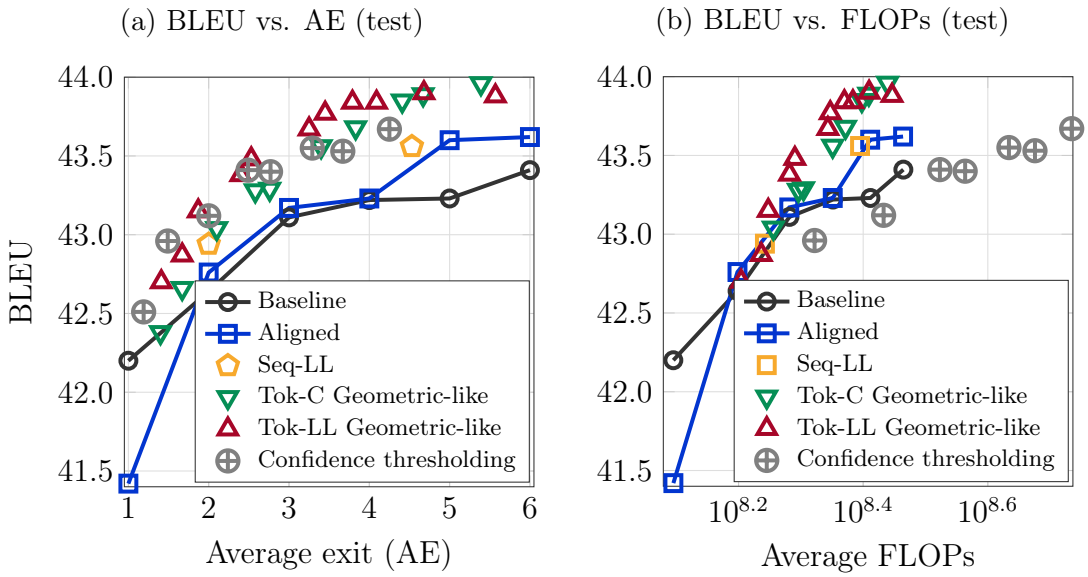


Figure 5.8: WMT'14 En→Fr test set: trade-off between speed (average exit or FLOPs) and accuracy (BLEU) for depth-adaptive methods.

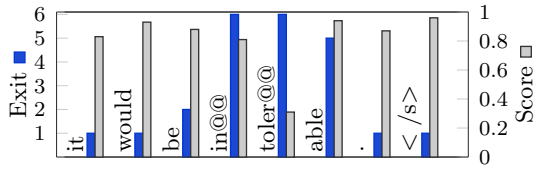
	$n=1$	$n=2$	$n=3$	$n=4$	$n=5$	$n=6$	Average
Baseline	**42.20**	42.64	43.11	43.22	43.23	43.41	**42.97**
Aligned	41.42	**42.76**	**43.17**	**43.23**	**43.60**	**43.62**	**42.97**

Table 5.6: WMT'14 En→Fr test set: BLEU scores of N independent standard transformers and the anytime prediction model.

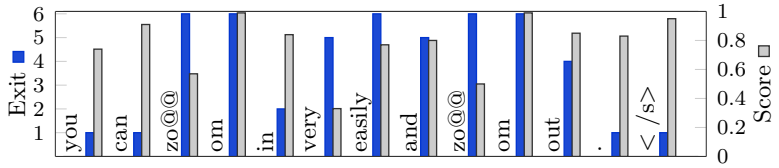
5.4.4 Scaling the Adaptive-depth Models

Finally, we take the best performing models form the IWSLT'14 En→De benchmark and test them on the large WMT'14 En→Fr benchmark. Results on the test set (Figure 5.8a and Table 5.6) show that, even in a very large-scale setup, adaptive depth estimation improves on a Transformer baseline. The sequence-specific depth approach improves only marginally over the baseline. Tok-LL geometric-like can match the best baseline result of BLEU 43.41 ($n = 6$) by using only AE = 2.40 which corresponds to 40% of the decoder blocks. We also match the best aligned result of BLEU 43.62 ($n = 6$) with AE = 3.25. In this setup, Tok-LL geometric-like slightly outperforms the Tok-C counterpart. Confidence thresholding

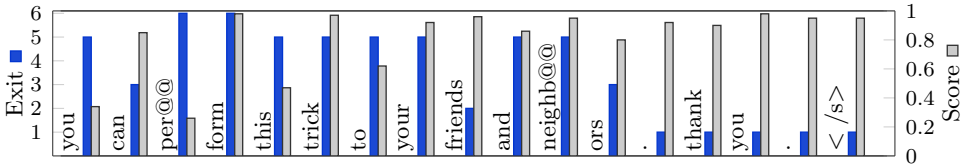
matches the accuracy of the $n = 6$ baseline with AE 2.5 or 59% fewer decoding blocks. However, confidence thresholding requires computing the output classifier at each block to determine whether to exit or continue. This is a large overhead since output classifiers predict $|\mathcal{V}|$ =44K types for this benchmark (§5.4.1). To better account for this, we measure the average number of FLOPs per output token (the breakdown of the computation cost is available in Appendix B). Figure 5.8b shows that the Tok-LL geometric-like approach provides a better trade-off when the overhead of the output classifiers is considered. In fact, with features in $\mathbb{R}^d$, the decision with a geometric-like distribution has a cost in $O(d)$ compared to $O(d\,|\mathcal{V}|)$ with confidence thresholding.



(a) **Src:** es wäre uner@@trä@@glich .
Ref: i couldn 't bear it .

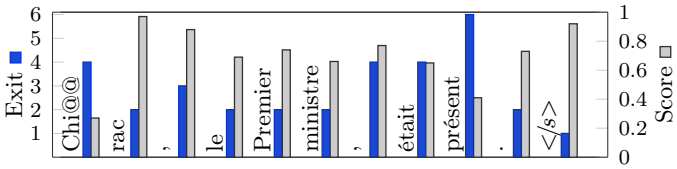


(b) **Src:** man kann sehr leicht rein und raus@@zo@@omen .
Ref: you can zo@@om around very simply .

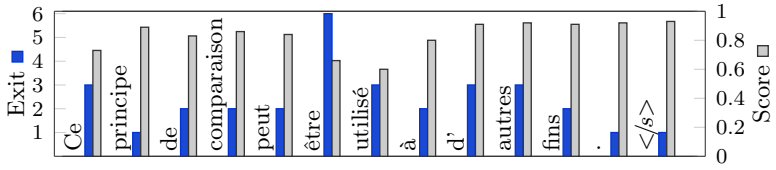


(c) **Src:** diesen trick können sie ihren freunden und nachbarn vor@@führen . danke .
Ref: there is a trick you can do for your friends and neighb@@ors . thanks .

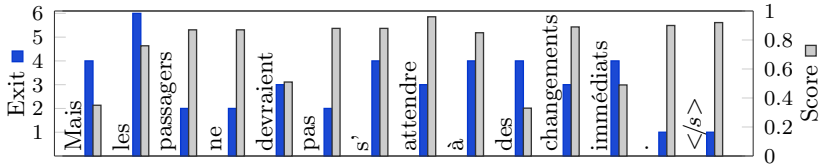
Figure 5.9: Examples from the IWSLT'14 De→En test set with Tok-LL geometric-like depth estimation. Token exits are in blue and model scores (probabilities) are in gray. The '@@' are due to BPE or subword tokenization. For each example the source (**Src**) and the reference (**Ref**) are provided in the caption.



(a) **Src:** Chi@@rac , the Prime Minister , was there .
Ref: Chi@@rac , Premier ministre , est là .



(b) **Src:** This principle of comparison can be used for other purposes .
Ref: Ce principe de comparaison peut avoir d' autres applications .



(c) **Src:** But passengers shoul@@dn't expect changes to happen immediately .
Ref: Mais les passagers ne devraient pas s' attendre à des changements immédiats .

Figure 5.10: Examples from the WMT'14 En→Fr test set with Tok-LL geometric-like depth estimation. For details see Figure 5.9.

5.4.5 Qualitative Results

Besides the advantageous accuracy-speed trade-offs we can attain with adaptive-depth estimation, the exits distribution for a given sample can give an insight into what a Transformer decoder considers to be a difficult task. In this section, for each hypothesis $\widetilde{\boldsymbol{y}}$, we will look at the sequence of selected exits $(n_1, \ldots, n_{|\widetilde{\boldsymbol{y}}|})$ and the probability scores $(p_1, \ldots p_{|\widetilde{\boldsymbol{y}}|})$ with $p_t = p(\widetilde{y}_t | h_{t-1}^{n_t})$ *i.e.* the confidence of the model in the sampled token at the estimated exit.

Figures 5.9 and 5.10 show a few hypotheses from IWSLT'14 De→En and the WMT'14 En→Fr test sets, respectively. For each hypothesis we state the exits and the probability scores. In Figure 5.10a, predicting 'présent' (meaning 'present') is hard. A straightforward translation is 'était là' but the model chooses 'present' which is also appropriate. In Figure 5.10c, the model uses more computation to predict the definite article 'les' since the source has omitted the article for 'passengers'.

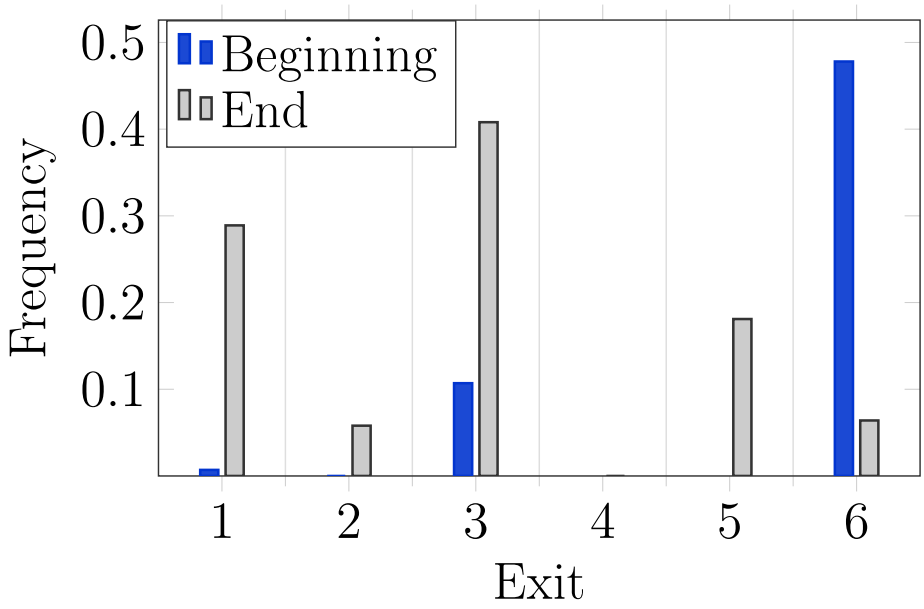


Figure 5.11: WMT'14 En→Fr test set: exit distributions in the beginning (relative-position: rpos<0.1) and near the end (rpos>0.9) of the hypotheses of three models.

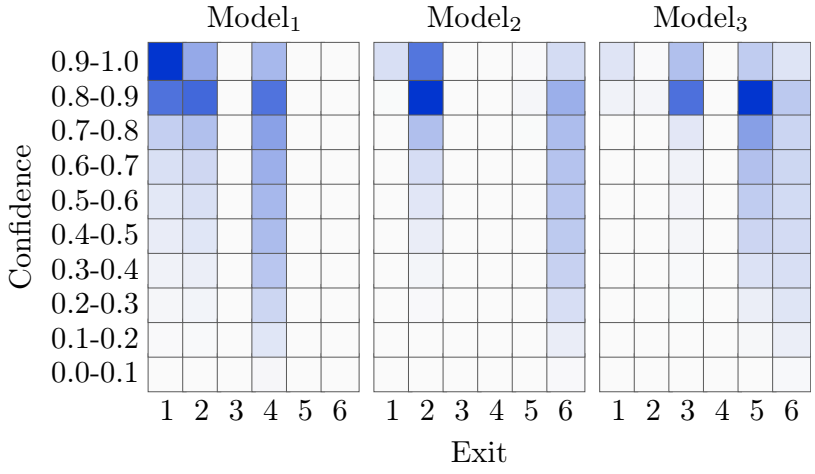


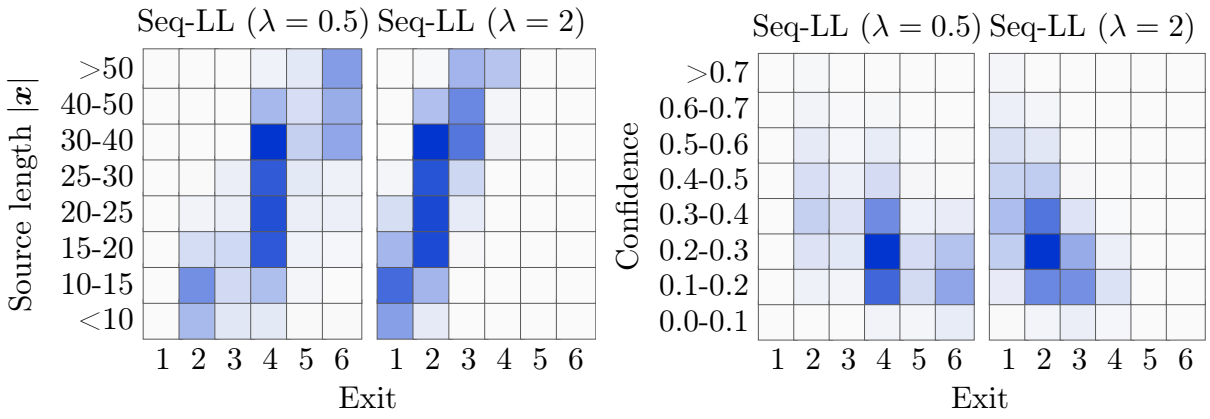
Figure 5.12: WMT'14 En→Fr test set: joint histogram of the exits and the confidence scores for three models.

A clear trend in both benchmarks is that the model requires less computation near the end of decoding to generate the end of sequence marker </s> and the preceding full-stop when relevant. In Figure 5.11, we show the distribution of the exits at the beginning and near the end of test set hypotheses. We consider the beginning of a sequence to be the first 10% of its tokens and the end its last 10%. The exit distributions are shown for three models on WMT'14 En→Fr. Model_1 has an average exit of AE = 2.53, Model_2 exits at AE = 3.79 on average and Model_3 with AE = 4.68. Within the same models, deep exits are used at the beginning of the sequence and early exits are selected near the end. For heavily regularized models such as Model_1 with AE = 2.53, the disparity between beginning and end is less severe as the model exits early most of the time. We also observe a correlation between the model's probability scores and the amount of computation, particularly in low-cost models. We show in Figure 5.12, for the aforementioned three models, the joint histogram of the scores and the selected exit. In both Model_1 and Model_2, early exits ($n \leq 2$) are used in the high confidence range $[0.8 - 1]$ and deeper exits ($n \geq 4$) are used in the low-confidence range $[0 - 0.5]$. Model_3 has a high average exit (AE = 4.68) so most tokens exit late, however, in low confidence ranges the model does not exit earlier than $n = 5$.

At the sequence level, we look at the correlation between the model's confidence evaluated as the average score $\frac{1}{|\boldsymbol{y}|}\sum_t p_t$ and the chosen exit in Figure 5.13b. We observe a similar behavior to token-specific exits where a higher confidence is associated with an early exit. We also note a correlation between the source sequence length ($|\boldsymbol{x}|$) and the chosen exit. In the joint histogram of Figure 5.13a, longer sequences exit late in the decoder and short ,*easy*, sequences exit earlier.

5.5 Related Work

Anytime prediction is less common in natural language processing. Until recently with the surge of pre-trained Transformer-based language representations (Devlin et al., 2019; Radford et al., 2019; Yang et al., 2019), NLP architectures were *tiny* in comparison to computer vision networks. Consequently, there was little interest in slicing the models to improve inference speed. To our knowledge, the only existing work in adaptive computation for structured



(a) (exit, source length) (b) (exit, confidence)

Figure 5.13: WMT'14 En→Fr test set: joint histogram of (exit, confidence score) and (exit, source length) for two sequence-level models, Seq-LL $\lambda = 0.5$ and $\lambda = 2$.

prediction is the ACT algorithm (Graves, 2016). ACT builds off of boosting techniques (Schapire, 2003; Grubb and Bagnell, 2012) and estimates for an input the amount of iterations the same RNN cell need to be re-applied. Recently, Universal Transformers (Dehghani et al., 2019) used ACT to control the iterative application of a single Transformer block. Dehghani et al. (2019) did only estimate dynamic amount of computation for language modeling tasks and fixed the number of iterations for large-scale machine translation. Besides, in their machine translation experiments, they used a wider block *i.e.* large embeddings and classifiers in order to pack as many weights as an entire standard Transformer in a single block.

In our work we do not increase the size of individual layers and tie the auxiliary classifiers in the large WMT'14 En→Fr benchmark, so that the number of parameters remains unchanged. Our work considers a variety of mechanisms to estimate the depth or amount of computation needed on a sample basis.

In computer vision literature, different depth estimation methods were proposed. Huang et al. (2018) consider two evaluation setups. The first in *anytime prediction* where a finite computational budget B is available for each test example. The sample is forwarded through the network until the budget is depleted and the most recent prediction is emitted. The second is *budgeted batch classification* with a budget B allocated for a set of samples $\mathcal{D}$. In this scenario a test sample traverses the network and exits after the n-th classifier if its

confidence exceeds a pre-determined threshold. The confidence of a classifier is measured as the maximum value of its output distribution. To determine the thresholds, Huang et al. (2018) assumes that a sample will exit at any given point n with a fixed probability across all exit points denoted with $q \in [0, 1]$. This means that the n-th exit point will process a sample with a probability $q_n = \frac{1}{Z}(1-q)^{n-1}q$, where Z is a normalizing constant. With cost_n denoting the cost of processing an input up to the n-th classifier, the estimated budget of classifying all the samples in $\mathcal{D}$ is $|\mathcal{D}|\,\mathbb{E}_{q_n}[\text{cost}_n]$. q is determined by solving $|\mathcal{D}|\,\mathbb{E}_{q_n}[\text{cost}_n] \leq B$ and the thresholds are approximated on a development set so that the exit probability q is met. A similar approach is used in BranchyNet (Teerapittayanon et al., 2016) where the prediction confidence is measured with the entropy of the output distribution instead. The thresholds are tuned to select a satisfying accuracy-speed trade-off. This is similar to our confidence thresholding model where the tuning was also based on the resulting trade-off between accuracy and speed. This approach turns out to be extremely expensive in the case of large output spaces. Bolukbasi et al. (2017) train decision functions to minimize the prediction time under a tolerated classification error margin. The n-th decision function takes as input the network's features by the n-th exit point as well as the entropy of the prediction. These decision functions are trained iteratively starting from the deepest one.

The previously mentioned ACT (Graves, 2016) trains a halting mechanism in order to break form a loop of iterative updates with an RNN cell. With each update, the halting signal is accumulated and once it exceeds a fixed threshold 1-ϵ, a prediction is emitted. The halting signal is an affine projection of the current feature followed by a sigmoid non-linearity. If χ^n is the signal evaluated after the n-th iteration, then the prediction cost is $\min\{n' \mid \sum_{n=1}^{n'} \chi^n \leq 1 - \epsilon\}$. The RNN and the halting module parameters are jointly trained in order to minimize the decoding loss plus a regularization term $\Omega = 1 - \sum_{n=1}^{\text{cost}} \chi^n$. Our geometric-like distribution is more intuitive than accumulating halting signals as we can interpret each term as an exit probability. Our supervision with an oracle allows us to effectively train these kinds of exit distributions.

5.6 Conclusion

In this chapter we presented our work on anytime structured prediction. We introduced simple but effective methods to render sequence models capable of making predictions at different points in the network. We compared two different training paradigms to address the potential issues of misaligned states in self-attention and established that Transformer models are robust to mixed-level features. We studied in the second part a number of different mechanisms to predict the required network depth and found that a well tuned geometric-like estimation of the exit on a token basis, obtains the best trade-off between speed and accuracy. Our Results show that the number of decoder layers can be reduced by more than three quarters at no loss in accuracy compared to a well tuned Transformer baseline.

Chapter 6

Conclusion

In this dissertation, we develop alternative methods for Neural Machine Translation (NMT) questioning the prevalent paradigm in state-of-the-art sequence-to-sequence models. We reconsider the source-target interaction in the encoding pipeline of NMT systems and build a framework for online NMT where the model alternates between reading and writing. Asides from the source-target interaction, we explore the trade-off between the translation's computational cost and its quality and introduce an efficient NMT model with anytime-prediction capabilities. We will conclude this thesis with a summary of our contributions (§6.1) and a discussion of the remaining challenges and open directions of inquiry (§6.2).

6.1 Summary

Chapter 3: Pervasive Attention. We revised the shallow attention modules in encoder-decoder models and introduced Pervasive Attention, a sequence-to-sequence model that jointly encodes the source and target sequences with a two-dimensional ConvNet. Pervasive Attention enhances the interaction between the source and target sequences by letting the encoding of one sequence affect the other. We proposed different design choices pertaining to the interconnectivity of the main two-dimensional convolutions and found that accumulating features from different levels of abstraction helps. We also established that slowly growing the receptive field with 5×5 kernels achieves better results than abruptly integrating the full context with wide filters. To deal with the generative aspect of Pervasive Attention, we explored different methods to aggregate the ConvNet's features into fixed-size representations and studied their effect on the translation's quality as well

as the source-target alignments inferred from the grid. We finally established that the quality of our model's translations on small to midsize benchmarks is competitive with state-of-the-art encoder-decoder models.

Chapter 4: Online Neural Machine Translation. In the context of online machine translation, the model is equipped with an agent to decide whether to read from the source or write to the target and is judged by both the quality and lagging of its translations. We proposed a framework for modeling and training online NMT systems with either deterministic or dynamic decoding agents. We adapted Transformer models for online decoding and highlighted the finding that "what works best for the offline task is not guaranteed to work as well online", in particular, encoding the source context in a uni-directional or bi-directional manner. We studied the deterministic wait-k decoding models, especially their ability to operate outside the range of delays they were optimized for, and the potential of training a multitasking wait-k agent well tuned for a variety of laggings. We also introduced dynamic decoding agents that leverage the interaction grid of Pervasive Attention models. These agents were jointly trained with the underlying NMT model and were supervised with a likelihood-based oracle using dynamic programming to estimate the optimal decoding path. We proved deterministic wait-k agents to be a strong baseline in online translation, moreover, we were able to match the quality of offline models on a few NMT benchmarks with 70% less lagging with our dynamic Pervasive Attention agents. We demonstrated the disposition of Pervasive Attention models to the task of online translation thanks to its grid of features akin to a decoding grid.

Chapter 5: Cost-effective Decoding. We tackled the problem of anytime structured prediction with Transformer models that adapt the amount of computation to each input in order to achieve a trade-off between speed or computational cost and accuracy. We first equipped our decoders with the capability of emitting output distributions at multiple stages of decoding then compared two different training paradigms, mixed and aligned, to address the potential issues of misaligned states in self-attention. We established that Transformer models are robust to mixed-level features so that we can copy low-level activations to subsequent blocks without hindering the output's quality. We proposed a number of halting mechanisms to predict the required network depth and found that

estimating the exit on a token basis with a geometric-like distribution updated after each block, obtains the best trade-off between speed and accuracy. Experiments on translation benchmarks of different scales show that we can effectively match the performance of baseline models with less than 76% of the computation.

6.2 Future work

In our attempt to answer our research questions just as many new questions arose. We discuss in the following some future perspectives and open questions based on the work presented in this thesis.

Analysis and interpretability A valid criticism of NMT, and by extension most of the inference processes based on deep learning techniques, is their lack of transparency. These processes are described as a black box *i.e.* we can observe feature activation but we cannot understand the actual phenomenon behind it. Methods for interpreting and visualizing neural models are explored much more in vision (Szegedy et al., 2014; Simonyan et al., 2014; Zeiler and Fergus, 2014; Mahendran and Vedaldi, 2015; Bau et al., 2017) than in NLP. These visualization techniques consist mainly in finding inputs that maximize the activation of a neuron in order to capture humanly interpretable information about the neuron's role and how it might contribute to the final decision. Recently, the question of analyzing and interpreting NLP models has gained traction with the BlackboxNLP workshop (Linzen et al., 2018, 2019). One popular method of analysis consists of evaluating the quality of the features learned by an NMT model in other NLP tasks (Shi et al., 2016; Hill et al., 2017; Adi et al., 2017; Belinkov et al., 2017; Raganato and Tiedemann, 2018). Although this method has its merits, it does not allow us to know more about the inner workings of the model and how the information flows in the computational graph up until outputting the final decision. Another prominent research direction is to assess the quality of the attention mechanism in a encoder-decoder model (Tang et al., 2018; Jain and Wallace, 2019; Boito et al., 2019; Voita et al., 2019; Vig and Belinkov, 2019). Conclusions on the interpretability of attention weights were contradictory and varied from one task to the other depending on the importance of the attention to the

model's prediction (Vashishth et al., 2020) and how it was trained (Pruthi et al., 2019). Other works go beyond the attention mechanism and provide a breakdown of the input's contribution (source and target) in each decoding step (Li et al., 2016; Ding et al., 2017) and more broadly a quantification of the interaction between different neurons in the trained model (Dalvi et al., 2019; Bau et al., 2019; Li et al., 2020).

These methods of *post-hoc* interpretability (Montavon et al., 2018) aims at understanding what a trained model predicts. In our experiments, we find that slight changes of the architecture's design, the learning rate schedule, the number of training updates and the characteristics of the training data and how its pre-processed and batched are important factors with a considerable effect on what the model learns and how it performs. Understanding how the signal is propagated in the computational graph throughout the training phase could offer clues about the model's inner workings and how to effectively design and train better models.

Neural architecture search We discussed in the previous point the interpretability of NMT models. There are different motivations behind interpreting a model's decisions and ours is to *debug* our models and improve their designs. Another tool for discovering better neural architectures is Neural Architecture Search (NAS). Works on NAS propose solutions based on reinforcement learning (Zoph and Le, 2017; Baker et al., 2016; Cai et al., 2018) or evolutionary algorithms (Real et al., 2017; Liu et al., 2018; Pham et al., 2018; Brock et al., 2017; Real et al., 2019; Liu et al., 2019a) to automate the process of architecture design. Most of these works focus on computer vision tasks and are limited in NLP to the search of the optimal recurrent cell for language modeling (Zoph and Le, 2017; Pham et al., 2018). We posit that NAS is a viable framework for finding more than recurrent cells in NLP. In the context of NMT, pairing NAS with a graph view of the source-target sequences and a clear-cut separation between position-based context (*e.g.* with convolution) and content-based context(*e.g.* with self-attention) can help us find better interpretable models for the source-source, target-target and source-target interactions. Besides the aforementioned interaction of tokens, this argument holds for other aspects addressed in this thesis, namely, the depth or level of abstraction appropriate for generation and the ideal context size in online NMT.

Unordered decoding In this thesis, and like other conventional sequence-to-sequence models, we generate target tokens auto-regressively *i.e.* step by step conditioning the generation on the history. This sequential decoding inhibits the parallelization of inference in convolutional and attention-based models. Recent works propose non-autoregressive decoding by iteratively refining a sequence of target tokens (Gu et al., 2017a; Lee et al., 2018). These methods, although fast, produce low-quality translations and require explicit length modeling that remains fixed throughout the refinement process. A new research direction on the rise is insertion-deletion based decoding that allows for flexible generation orders and partially parallel inference without diminishing the translation's quality (Stern et al., 2019; Welleck et al., 2019; Gu et al., 2019b,a; Emelianenko et al., 2019; Ruis et al., 2019). In the context of online machine translation, allowing the decoder to write without order restrictions will most likely mitigate the issues of duplication and omission that occurs when the decoder is asked to write with insufficient context. With that, the evaluation metrics of translation lagging (AP, AL and DAL) need to accommodate insertions and deletions to fairly measure the delay of refined translations. This will also allow for beam-search decoding in Online MT where the partially outputted candidate might be substituted with a more likely translation in the beam.

Speech recognition and speech translation Another future direction is to validate Pervasive Attention models on other sequence transduction tasks, in particular automatic speech recognition (ASR). Similar to MT, sequence-to-sequence models with attention mechanisms have proven their worth in ASR with recurrent (Bahdanau et al., 2016; Chan et al., 2016), convolutional (Sercu et al., 2016; Zhang et al., 2017) and Transformer-based architectures (Sperber et al., 2018; Salazar et al., 2019; Mohamed et al., 2019). We expect the monotonically aligned ASR data to be an appropriate fit for Pervasive Attention. Another task to consider is end-to-end automatic speech translation (AST) (Bérard et al., 2018; Weiss et al., 2017) where models with strong segmentation and alignment capabilities are a plus.

Bibliography

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-Grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. In *Proc. of ICLR*.

Ashkan Alinejad, Maryam Siahbani, and Anoop Sarkar. 2018. Prediction Improves Simultaneous Neural Machine Translation. In *Proc. of EMNLP*.

Robert B Allen. 1987. Several Studies on Natural Language and Back-Propagation. In *Proceedings of the IEEE First International Conference on Neural Networks*.

Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. 2018. Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering. In *Proc. of CVPR*.

Sercan Arik, Mike Chrzanowski, Adam Coates, Gregory Frederick Diamos, Andrew Gibiansky, Yongguo Kang, Xiongmin Li, John Miller, Andrew Ng, Jonathan Raiman, Shubho Sengupta, and Mohammad Shoeybi. 2017. Deep Voice: Real-Time Neural Text-to-Speech. In *Proc. of ICML*.

Naveen Arivazhagan, Colin Cherry, Wolfgang Macherey, Chung-Cheng Chiu, Semih Yavuz, Ruoming Pang, Wei Li, and Colin Raffel. 2019. Monotonic Infinite Lookback Attention for Simultaneous Machine Translation. In *Proc. of ACL*.

Philip Arthur, Trevor Cohn, and Gholamreza Haffari. 2020. Learning Coupled Policies for Simultaneous Machine Translation. In *ArXiv preprint*.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *ArXiv preprint*.

Parnia Bahar, Christopher Brix, and Hermann Ney. 2018. Towards Two-Dimensional Sequence to Sequence Model in Neural Machine Translation. In *Proc. of EMNLP*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proc. of ICLR*.

Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. 2016. End-to-End Attention-Based Large Vocabulary Speech Recognition. In *Proc. of ICASSP*.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing Neural Network Architectures Using Reinforcement Learning. In *Proc. of ICLR*.

Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. 2019. Identifying and Controlling Important Neurons in Neural Machine Translation. In *Proc. of ICLR*.

David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. 2017. Network Dissection: Quantifying Interpretability of Deep Visual Representations. In *Proc. of CVPR*.

Yonatan Belinkov, Lluís Màrquez, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. 2017. Evaluating Layers of Representation in Neural Machine Translation on Part-of-Speech and Semantic Tagging Tasks. In *Proc. of IJCNLP*.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. In *Proc. of NeurIPS*.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2001. A Neural Probabilistic Language Model. In *Proc. of NeurIPS*.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proc. of ICML*.

Winfield S Bennett and Jonathan Slocum. 1985. The LRC Machine Translation System. *Computational linguistics.*

Alexandre Bérard, Laurent Besacier, Ali Can Kocabiyikoglu, and Olivier Pietquin. 2018. End-to-End Automatic Speech Translation of Audiobooks. In *Proc. of ICASSP.*

Joseph Bertrand. 1907. *Calcul des Probabilités*. Gauthier-Villars.

Christian Boitet and Nicolas Nedobejkine. 1980. Russian-French at GETA: Outline of the Method and Detailed Example. In *Proc. of COLING.*

Marcely Zanon Boito, Aline Villavicencio, and Laurent Besacier. 2019. Empirical Evaluation of Sequence-to-Sequence Models for Word Discovery in Low-resource Settings. In *Proc. of INTERSPEECH.*

Tolga Bolukbasi, Joseph Wang, Ofer Dekel, and Venkatesh Saligrama. 2017. Adaptive Neural Networks for Efficient Inference. In *Proc. of ICML.*

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating Sentences from a Continuous Space. In *Proc. of CoNLL.*

John S. Bridle. 1990. Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition. In *Neurocomputing.*

Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. 2017. Smash: One-Shot Model Architecture Search through Hypernetworks. In *Proc. of ICLR.*

Peter Brown, John Cocke, S Della Pietra, V Della Pietra, Frederick Jelinek, Robert Mercer, and Paul Roossin. 1988. A Statistical Approach to Language Translation. In *Proc. of COLING.*

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics.*

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based N-gram Models of Natural Language. *CL.*

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient Architecture Search by Network Transformation. In *Proc. of AAAI*.

Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. 2016. An Analysis of Deep Neural Network Models for Practical Applications. *ArXiv preprint*.

Asuncion Castano and Francisco Casacuberta. 1997. A Connectionist Approach to Machine Translation. In *Fifth European Conference on Speech Communication and Technology*.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. 2014. Report on the 11th IWSLT evaluation campaign. In *Proc. of IWSLT*.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. 2016. Listen, Attend and Spell: A neural Network for Large Vocabulary Conversational Speech Recognition. In *Proc. of ICASSP*.

Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. 2015. Learning to Search Better than Your Teacher. In *Proc. of ICML*.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proc. of ACL*.

Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Mike Schuster, Noam Shazeer, Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. 2018. The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation. In *Proc. of ACL*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. *ArXiv preprint*.

Colin Cherry and George Foster. 2019. Thinking Slow about Latency Evaluation for Simultaneous Machine Translation. *ArXiv preprint*.

Chung-Cheng Chiu and Colin Raffel. 2018. Monotonic Chunkwise Attention. In *Proc. of ICLR*.

Kyunghyun Cho and Masha Esipova. 2016. Can Neural Machine Translation Do Simultaneous Translation? *ArXiv preprint.*

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of EMNLP.*

François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *Proc. of CVPR.*

Lonnie Chrisman. 1991. Learning Recursive Distributed Representations for Holistic Computation. *Connection Science.*

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *Proc. of ICLR.*

Ronan Collobert and Jason Weston. 2008. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning. In *Proc. of ICML.*

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James Glass. 2019. What is One Grain of Sand in the Desert? Analyzing Individual Neurons in Deep NLP Models. In *Proc. of AAAI.*

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, and Stephan Vogel. 2018. Incremental Decoding and Training Methods for Simultaneous Translation in Neural Machine Translation. In *Proc. of NAACL-HLT.*

Hal Daumé, John Langford, and Daniel Marcu. 2009. Search-Based Structured Prediction. *Machine Learning.*

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language Modeling with Gated Convolutional Networks. In *Proc. of ICML.*

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal Transformers. In *Proc. of ICLR.*

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*.

Yuntian Deng, Yoon Kim, Justin Chiu, Demi Guo, and Alexander M. Rush. 2018. Latent Alignment and Variational Attention. In *Proc. of NeurIPS*.

Michael Denkowski and Alon Lavie. 2014. Meteor Universal: Language Specific Translation Evaluation for Any Target Language. In *Proc. of the Workshop on Statistical Machine Translation*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *Proc. of NAACL-HLT*.

Yanzhuo Ding, Yang Liu, Huanbo Luan, and Maosong Sun. 2017. Visualizing and Understanding Neural Machine Translation. In *Proc. of ACL*.

Leon E Dostert. 1955. The Georgetown-IBM Experiment. *Machine translation of languages. John Wiley & Sons, New York*.

Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. 1988. Using Latent Semantic Analysis to Improve Access to Textual Information. In *Proc. of the SIGCHI conference on Human factors in computing systems*.

Chris Dyer, Victor Chahuneau, and Noah A. Smith. 2013. A Simple, Fast, and Effective Reparameterization of IBM Model 2. In *Proc. of NAACL-HLT*.

David M. Eberhard, Simons Gary F., and Fennig (eds.) Charles D. 2020. Ethnologue: Languages of the World.

Sergey Edunov, Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Classical Structured Prediction Losses for Sequence to Sequence Learning. In *Proc. of NAACL-HLT*.

Salah El Hihi and Yoshua Bengio. 1996. Hierarchical Recurrent Neural Networks for Long-Term Dependencies. In *Proc. of NeurIPS*.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018a. Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction. In *Proc. of CoNLL*.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018b. Token-level and Sequence-level Loss Smoothing for RNN Language Models. In *Proc. of ACL*.

Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2020a. Improved Training Techniques for Online Neural Machine Translation.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. 2020b. Depth-Adaptive Transformer. In *Proc. of ICLR*.

Jeffrey L. Elman. 1990. Finding Structure in Time. *Cognitive science*.

Dmitrii Emelianenko, Elena Voita, and Pavel Serdyukov. 2019. Sequence Modeling with Unconstrained Generation Order. In *Proc. of NeurIPS*.

Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*.

Marzieh Fadaee, Arianna Bisazza, and Christof Monz. 2017. Data Augmentation for Low-Resource Neural Machine Translation. In *Proc. of ACL*.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-Way, Multilingual Neural Machine Translation with a Shared Attention Mechanism. In *Proc. of NAACL-HLT*.

José AR Fonollosa, Noe Casas, and Marta R Costa-jussà. 2019. Joint Source-Target Self Attention with Locality Constraints. *ArXiv preprint*.

Mikel L Forcada and Ramón P Ñeco. 1997. Recursive Hetero-Associative Memories for Translation. In *International Work-Conference on Artificial Neural Networks*.

Christian Fügen, Alex Waibel, and Muntsin Kolss. 2007. Simultaneous Translation of Lectures and Speeches. *Machine translation*.

Philip Gage. 1994. A New Algorithm for Data Compression. *C Users J.*

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional Sequence to Sequence Learning. In *Proc. of ICML.*

Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to Forget: Continual Prediction with LSTM. In *Proc. of ICANN.*

Alex Graves. 2016. Adaptive Computation Time for Recurrent Neural Networks. In *ArXiv preprint.*

Alvin Grissom II, He He, Jordan Boyd-Graber, John Morgan, and Hal Daumé III. 2014. Don't Until the Final Verb Wait: Reinforcement Learning for Simultaneous Machine Translation. In *Proc. of EMNLP.*

Alex Grubb and Drew Bagnell. 2012. Speedboost: Anytime Prediction with /niform Near-Optimality. In *Proc. of AISTATS.*

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017a. Non-Autoregressive Neural Machine Translation. In *Proc. of ICLR.*

Jiatao Gu, Qi Liu, and Kyunghyun Cho. 2019a. Insertion-Based Decoding with Automatically Inferred Generation Order. *TACL.*

Jiatao Gu, Graham Neubig, Kyunghyun Cho, and Victor OK Li. 2017b. Learning to Translate in Real-time with Neural Machine Translation. In *Proc. of EACL.*

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019b. Levenshtein Transformer. In *Proc. of NeurIPS.*

Zellig S. Harris. 1954. Distributional Structure. *WORD.*

He He, Jordan Boyd-Graber, and Hal Daumé III. 2016a. Interpretese vs. Translationese: The Uniqueness of Human Strategies in Simultaneous Interpretation. In *Proc. of NAACL-HLT.*

He He, Alvin Grissom II, John Morgan, Jordan Boyd-Graber, and Hal Daumé III. 2015. Syntax-based Rewriting for Simultaneous Machine Translation. In *Proc. of EMNLP.*

Hua He and Jimmy Lin. 2016. Pairwise Word Interaction Modeling with Deep Neural Networks for Semantic Similarity Measurement. In *Proc. of NAACL-HLT*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016b. Deep Residual Learning for Image Recognition. In *Proc. of CVPR*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016c. Identity Mappings in Deep Residual Networks. In *Proc. of ECCV*.

Tianyu He, Xu Tan, Yingce Xia, Di He, Tao Qin, Zhibo Chen, and Tie-Yan Liu. 2018. Layer-Wise Coordination between Encoder and Decoder for Neural Machine Translation. In *Proc. of NeurIPS*.

Felix Hill, Kyunghyun Cho, Sébastien Jean, and Yoshua Bengio. 2017. The Representational Geometry of Word Meanings Acquired by Neural Machine Translation Models. *Machine Translation*.

Geoffrey E. Hinton. 1990. Connectionist Learning Procedures. In *Machine learning*.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*.

Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. 2014. Convolutional Neural Network Architectures for Matching Natural Language Sentences. In *Proc. of NeurIPS*.

Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J. Andrew Bagnell. 2019. Learning Anytime Predictions in Neural Networks via Adaptive Loss Balancing. In *Proc. of AAAI*.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens Van Der Maaten, and Kilian Q. Weinberger. 2018. Multi-scale Dense Convolutional Networks for Efficient Prediction. In *Proc. of ICLR*.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2017. Densely Connected Convolutional Networks. In *Proc. of CVPR*.

John Hutchins. 2007. Machine Translation: A Concise History. *Computer aided translation: Theory and practice*.

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proc. of ICML*.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep Unordered Composition Rivals Syntactic Methods for Text Classification. In *Proc. of ACL*.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not Explanation. In *Proc. of NAACL-HLT*.

Navdeep Jaitly, Quoc V. Le, Oriol Vinyals, Ilya Sutskever, David Sussillo, and Samy Bengio. 2016. An Online Sequence-to-Sequence Model Using Partial Conditioning. In *Proc. of NeurIPS*.

Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. *TACL*.

Michael I. Jordan. 1986. Serial Order: a Parallel Distributed Processing Approach. Technical Report. *San Diego: University of California, Institute for Cognitive Science*.

Nal Kalchbrenner and Phil Blunsom. 2013. Recurrent Continuous Translation Models. In *Proc. of EMNLP*.

Nal Kalchbrenner, Ivo Danihelka, and Alex Graves. 2016a. Grid Long Short-Term Memory. In *Proc. of ICLR*.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016b. Neural Machine Translation in Linear Time. *ArXiv preprint*.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *Proc. of ACL*.

Nal Kalchbrenner, Aäron van den Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. 2017. Video Pixel Networks. In *Proc. of ICML*.

Andrej Karpathy and Li Fei-Fei. 2015. Deep Visual-Semantic Alignments for Generating Image Descriptions. In *Proc. of CVPR*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proc. of ICLR*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. of ACL*.

Philipp Koehn. 2005. Europarl: A parallel Corpus For Statistical Machine Translation. In *MT summit*.

Philipp Koehn, Marcello Federico, Wade Shen, Nicola Bertoldi, Ondrej Bojar, Chris Callison-Burch, Brooke Cowan, Chris Dyer, Hieu Hoang, Richard Zens, et al. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proc. of ACL*.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical Phrase-Based Translation. In *Proc. of NAACL-HLT*.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. 2017. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *IJCV*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. of NeurIPS*.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. 2017. Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations. In *Proc. of ICLR*.

Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. 2016. Ask me anything: Dynamic memory networks for natural language processing. In *Proc. of ICML*.

Rémi Leblond, Jean-Baptiste Alayrac, Anton Osokin, and Simon Lacoste-Julien. 2018. SeaRnn: Training RNNs with Global-Local Losses. In *Proc. of ICLR*.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. 2015. Deeply-Supervised Nets. In *Proc. of AISTATS*.

Jason Lee, Kyunghyun Cho, and Thomas Hofmann. 2017. Fully Character-level Neural Machine Translation without Explicit Segmentation. *TACL*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proc. of EMNLP*.

Dor Levy and Lior Wolf. 2017. Learning to Align the Source Code to the Compiled Object Code. In *Proc. of ICML*.

Jian Li, Xing Wang, Baosong Yang, Shuming Shi, Michael R. Lyu, and Zhaopeng Tu. 2020. Neuron Interaction Based Representation Composition for Neural Machine Translation. In *Proc. of AAAI*.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and Understanding Neural Models in NLP. In *Proc. of NAACL-HLT*.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proc. of ACL*.

Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. In *Proc. of ECCV*.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A Structured Self-Attentive Sentence Embedding. In *Proc. of ICLR*.

Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors. 2018. *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Tal Linzen, Grzegorz Chrupała, Yonatan Belinkov, and Dieuwke Hupkes, editors. 2019. *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018. Progressive Neural Architecture Search. In *Proc. of ECCV*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019a. Darts: Differentiable Architecture Search. In *Proc. of ICLR*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A Robustly Optimized BERT Pretraining Approach. *ArXiv preprint*.

Bruce T. Lowerre and Raj Reddy. 1980. The HARPY Speech Understanding System. *Trends in Speech Recognition*.

Jiasen Lu, Caiming Xiong, Devi Parikh, and Richard Socher. 2017. Knowing When to Look: Adaptive Attention via a Visual Sentinel for Image Captioning. In *Proc. of CVPR*.

Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Understanding and Improving Transformer from a Multi-Particle Dynamic System Point of View. *ArXiv preprint*.

Yuping Luo, Chung-Cheng Chiu, Navdeep Jaitly, and Ilya Sutskever. 2017. Learning Online Alignments with Continuous Rewards Policy Gradient. In *Proc. of ICASSP*.

Minh-Thang Luong and Christopher D. Manning. 2015. Stanford Neural Machine Translation Systems for Spoken Language Domains. In *Proc. of IWSLT*.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proc. of EMNLP*.

Mingbo Ma, Liang Huang, Hao Xiong, Renjie Zheng, Kaibo Liu, Baigong Zheng, Chuanqiang Zhang, Zhongjun He, Hairong Liu, Xing Li, Hua Wu, and Haifeng Wang. 2019a. STACL: Simultaneous Translation with Implicit Anticipation and Controllable Latency using Prefix-to-Prefix Framework. In *Proc. of ACL*.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019b. FlowSeq: Non-Autoregressive Conditional Sequence Generation with Generative Flow. In *Proc. of EMNLP*.

Xutai Ma, Juan Pino, James Cross, Liezl Puzon, and Jiatao Gu. 2020. Monotonic Multihead Attention. In *Proc. of ICLR*.

Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding Deep Image Representations by Inverting Them. In *Proc. of CVPR*.

Fandong Meng, Zhengdong Lu, Mingxuan Wang, Hang Li, Wenbin Jiang, and Qun Liu. 2015. Encoding Source Language with Convolutional Neural Network for Machine Translation. In *Proc. of ACL*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are Sixteen Heads Really Better than One? In *Proc. of NeurIPS*.

Takashi Mieno, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Speed or Accuracy? A Study in Evaluation of Simultaneous Speech Translation. In *Proc. of INTERSPEECH*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. In *ICLR Workshop Papers*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent Neural Network Based Language Model. In *Proc. of INTERSPEECH*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed Representations of Words and Phrases and Their Compositionality. In *Proc. of NeurIPS*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-Value Memory Networks for Directly Reading Documents. In *Proc. of EMNLP*.

Thomas M. Mitchell. 1997. *Machine Learning*. McGraw-Hill, Inc.

Andriy Mnih and Geoffrey Hinton. 2007. Three New Graphical Models for Statistical Language Modelling. In *Proc. of ICML*.

Abdelrahman Mohamed, Dmytro Okhonko, and Luke Zettlemoyer. 2019. Transformers with Convolutional Context for ASR. *ArXiv preprint*.

Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. 2018. Methods for Interpreting and Understanding Deep Neural Networks. *Digital Signal Processing*.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proc. of ICML*.

Graham Neubig, Zi-Yi Dou, Junjie Hu, Paul Michel, Danish Pruthi, and Xinyi Wang. 2019. compare-mt: A Tool for Holistic Comparison of Language Generation Systems. In *Proc. of NAACL, Demo Track*.

Nathan Ng, Kyra Yee, Alexei Baevski, Myle Ott, Michael Auli, and Sergey Edunov. 2019. Facebook FAIR's WMT19 News Translation Task Submission. In *Proc. of WMT*.

Toan Q Nguyen and Julian Salazar. 2019. Transformers Without Tears: Improving the Normalization of Self-attention. In *Proc. of IWSLT*.

Mohammad Norouzi, Samy Bengio, zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. 2016. Reward Augmented Maximum Likelihood for Neural Structured Prediction. In *Proc. of NeurIPS*.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. 2004. A Smorgasbord of Features for Statistical Machine Translation. In *Proc. of NAACL-HLT*.

Franz Josef Och and Hermann Ney. 2002. Discriminative Training and Maximum Entropy Models for Statistical Machine Translation. In *Proc. of ACL*.

Franz Josef Och, Christoph Tillmann, and Hermann Ney. 1999. Improved Alignment Models for Statistical Machine Translation. In *Proc. of EMNLP*.

Yusuke Oda, Graham Neubig, Sakriani Sakti, Tomoki Toda, and Satoshi Nakamura. 2015. Syntax-Based Simultaneous Translation Through Prediction of Unseen Syntactic Constituents. In *Proc. of ACL*.

Anthony G Oettinger. 1954. *A Study for the Design of an Automatic Dictionary*. Harvard University.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016a. WaveNet: a Generative Model for Raw Audio. In *ISCA Speech Syntesis Workshop*.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. 2016b. Pixel Recurrent Neural Networks. In *Proc. of ICML*.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A Fast, Extensible Toolkit for Sequence Modeling. In *Proc. of NAACL-HLT*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a Method for Automatic Evaluation of Machine Translation. In *Proc. 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 *Proc. of NeurIPS, Autodiff Workshop*.

Mattis Paulin, Jerome Revaud, Zaid Harchaoui, Florent Perronnin, and Cordelia Schmid. 2014. Transformation Pursuit for Image Classification. In *Proc. of CVPR*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proc. of EMNLP*.

Gabriel Pereyra, George Tucker, Jan Chorowski, Łukasz Kaiser, and Geoffrey Hinton. 2017. Regularizing Neural Networks by Penalizing Confident Output Distributions. In *Proc. of ICLR*.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. In *Proc. of ICML*.

Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. 2014. Dropout Improves Recurrent Neural Networks for Handwriting Recognition. In *Frontiers in Handwriting Recognition*.

Jordan B Pollack. 1990. Recursive Distributed Representations. *Artificial Intelligence*.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2019. BPE-Dropout: Simple and Effective Subword Regularization. *ArXiv preprint*.

Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C Lipton. 2019. Learning to Deceive with Attention-Based Explanations. *ArXiv preprint*.

Lawrence R. Rabiner. 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proc. of the IEEE*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. In *Technical report, OpenAI*.

Colin Raffel, Minh-Thang Luong, Peter J Liu, Ron J Weiss, and Douglas Eck. 2017. Online and Linear-Time Attention by Enforcing Monotonic Alignments. In *Proc. of ICML*.

Alessandro Raganato and Jörg Tiedemann. 2018. An Analysis of Encoder Representations in Transformer-Based Machine Translation. In *Proc. of EMNLP Workshop BlackboxNLP*.

Martin Raison, Pierre-Emmanuel Mazaré, Rajarshi Das, and Antoine Bordes. 2018. Weaver: Deep Co-Encoding of Questions and Documents for Machine Reading. *ArXiv preprint*.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2016. Sequence Level Training with Recurrent Neural Networks. In *Proc. of ICLR*.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized Evolution for Image Classifier Architecture Search. In *Proc. of AAAI*.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. 2017. Large-Scale Evolution of Image Classifiers. In *Proc. of ICML*.

Erwin Reifler. 1952. Mechanical Translation with a Pre-editor and Writing for Mechanical Translation. In *Conference on Mechanical Translation*.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. 2017. Self-Critical Sequence Training for Image Captioning. In *Proc. of CVPR*.

Helge Ritter and Teuvo Kohonen. 1989. Self-Organizing Semantic Maps. *Biological cybernetics*.

Laura Ruis, Mitchell Stern, Julia Proskurnia, and William Chan. 2019. Insertion-Deletion Transformer. In *Proc. of WNGT*.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Representations by Back-Propagating Errors. *nature*.

Julian Salazar, Katrin Kirchhoff, and Zhiheng Huang. 2019. Self-Attention Networks for Connectionist Temporal Classification in Speech Recognition. In *Proc. of ICASSP*.

Robert E. Schapire. 2003. The Boosting Approach to Machine Learning: An Overview. In *Nonlinear Estimation and Classification*. Springer.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing.*

Holger Schwenk and Jean-Luc Gauvain. 2002. Connectionist Language Modeling for Large Vocabulary Continuous Speech Recognition. In *Proc. of ICASSP.*

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proc. of ACL.*

Tom Sercu, Christian Puhrsch, Brian Kingsbury, and Yann LeCun. 2016. Very Deep Multilingual Convolutional Neural Networks for LVCSR. In *Proc. of ICASSP.*

Claude E. Shannon. 1948. A Mathematical Theory of Communication. *Bell System Technical Journal.*

Li Shen, Zhouchen Lin, and Qingming Huang. 2016. Relay Backpropagation for Effective Learning of Deep Convolutional Neural Networks. In *Proc. of ECCV.*

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2017. ReasoNet: Learning to Stop Reading in Machine Comprehension. In *Proc. of SIGKDD.*

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does String-Based Neural MT Learn Source Syntax? In *Proc. of EMNLP.*

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *ICLR Workshop Papers.*

Jason Smith, Herve Saint-Amand, Magdalena Plamadă, Philipp Koehn, Chris Callison-Burch, and Adam Lopez. 2013. Dirt Cheap Web-Ccale Parallel Text from the Common Crawl. In *Proc. of ACL.*

Matthias Sperber, Jan Niehues, Graham Neubig, Sebastian Stüker, and Alex Waibel. 2018. Self-Attentional Acoustic Models. In *Proc. of INTERSPEECH.*

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Training very Deep Networks. In *Proc. of NeurIPS*.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion Transformer: Flexible Sequence Generation via Insertion Operations. In *Proc. of ICML*.

Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. 2015. End-to-End Memory Networks. In *Proc. of NeurIPS*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proc. of NeurIPS*.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going Deeper With Convolutions. In *Proc. of CVPR*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proc. of CVPR*.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. *Proc. of ICLR*.

Gongbo Tang, Rico Sennrich, and Joakim Nivre. 2018. An Analysis of Attention Mechanisms: The Case of Word Sense Disambiguation in Neural Machine Translation. In *Proc. of WMT*.

Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. 2016. Branchynet: Fast Inference via Early Exiting from Deep Neural Networks. In *Proc. of ICPR*.

Jörg Tiedemann. 2012. Parallel Data, Tools and Interfaces in OPUS. In *Proc. of LREC*.

Christoph Tillmann and Hermann Ney. 2003. Word Reordering and a Dynamic Programming Beam Search Algorithm for Statistical Machine Translation. *Computational Linguistics*.

Shikhar Vashishth, Shyam Upadhyay, Gaurav Singh Tomar, and Manaal Faruqui. 2020. Attention Interpretability Across NLP Tasks. *ArXiv preprint*.

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 *Proc. of NeurIPS*.

Bernard Vauquois, Gerard Veillon, and Jean Veyrunes. 1965. Syntaxe et Interprétation. In *Proc. of COLING*.

Ramakrishna Vedantam, C. Lawrence Zitnick, and Devi Parikh. 2015. CIDEr: Consensus-Based Image Description Evaluation. In *Proc. of CVPR*.

Jesse Vig and Yonatan Belinkov. 2019. Analyzing the Structure of Attention in a Transformer Language Model. In *Proc. of BlackBoxNLP*.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and Tell: A Neural Image Caption Generator. In *Proc. of CVPR*.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. HMM-Based Word Alignment in Statistical Translation. In *Proc. of COLING*.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. 2019. Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned. In *Proc. of ACL*.

Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. 1989. Phoneme Recognition Using Time-Delay Neural Networks. *IEEE transactions on acoustics, speech, and signal processing*.

Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. 2016. A Deep Architecture for Semantic Matching with Multiple Positional Sentence Representations. In *Proc. of AAAI*.

Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. 2018. Skipnet: Learning Dynamic Routing in Convolutional Networks. In *Proc. of ECCV*.

Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. 2017. Tacotron: Towards end-to-end speech synthesis. In *Proc. of INTERSPEECH*.

Warren Weaver. 1949. Translation. Memorandum.

Ron J. Weiss, Jan Chorowski, Navdeep Jaitly, Yonghui Wu, and Zhifeng Chen. 2017. Sequence-to-Sequence Models Can Directly Translate Foreign Speech. In *Proc. of INTERSPEECH*.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. Non-Monotonic sequential Text Generation. In *Proc. of ICML*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory Networks. *Proc. of ICLR*.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. Pay Less Attention with Lightweight and Dynamic Convolutions. In *Proc. of ICLR*.

Lijun Wu, Yingce Xia, Fei Tian, Li Zhao, Tao Qin, Jianhuang Lai, and Tie-Yan Liu. 2018. Adversarial Neural Machine Translation. In *Proc. of ACML*.

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*.

Yuxin Wu and Kaiming He. 2018. Group Normalization. In *Proc. of ECCV*.

Ziang Xie, Sida I Wang, Jiwei Li, Daniel Lévy, Aiming Nie, Dan Jurafsky, and Andrew Y Ng. 2017. Data Noising as Smoothing in Neural Network Language Models. In *Proc. of ICLR*.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proc. of ICML*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Proc. of NeurIPS*.

Zhilin Yang, Ye Yuan, Yuexin Wu, William W Cohen, and Russ R Salakhutdinov. 2016. Encode, Review, and Decode: Reviewer Module for Caption Generation. In *Proc. of NeurIPS*.

Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. 2017a. Boosting Image Captioning with Attributes. In *Proc. of ICCV*.

Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. 2017b. Boosting Image Captioning with Attributes. In *Proc. of ICCV*.

Mahsa Yarmohammadi, Vivek Kumar Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. 2013. Incremental Segmentation and Decoding Strategies for Simultaneous Translation. In *Proc. of NAACL-HLT*.

Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. 2016. Image Captioning with Semantic Attention. In *Proc. of CVPR*.

Fisher Yu and Vladlen Koltun. 2016. Multi-Scale Context Aggregation by Dilated Convolutions. In *Proc. of ICLR*.

Lei Yu, Jan Buys, and Phil Blunsom. 2016. Online Segment to Segment Neural Transduction. In *Proc. of EMNLP*.

Amir R. Zamir, Te-Lin. Wu, Lin Sun, William Shen, , Jitendra Malik, and Silvio Savarese. 2016. Feedback Networks. *ArXiv preprint*.

Matthew D Zeiler and Rob Fergus. 2014. Visualizing and Understanding Convolutional Networks. In *Proc. of ECCV*.

Yu Zhang, William Chan, and Navdeep Jaitly. 2017. Very Deep Convolutional Networks for End-to-End Speech Recognition. In *Proc. of ICASSP*.

Guangxiang Zhao, Junyang Lin, Zhiyuan Zhang, Xuancheng Ren, Qi Su, and Xu Sun. 2019a. Explicit Sparse Transformer: Concentrated Attention Through Explicit Selection. *ArXiv preprint*.

Guangxiang Zhao, Xu Sun, Jingjing Xu, Zhiyuan Zhang, and Liangchen Luo. 2019b. MUSE: Parallel Multi-Scale Attention for Sequence to Sequence Learning. *ArXiv preprint*.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019a. Simpler and Faster Learning of Adaptive Policies for Simultaneous Translation. In *Proc. of EMNLP*.

Baigong Zheng, Renjie Zheng, Mingbo Ma, and Liang Huang. 2019b. Simultaneous Translation with Flexible Policy via Restricted Imitation Learning. In *Proc. of ACL*.

Renjie Zheng, Mingbo Ma, Baigong Zheng, and Liang Huang. 2019c. Speculative Beam Search for Simultaneous Translation. In *Proc. of EMNLP*.

Shlomo Zilberstein. 1996. Using Anytime Algorithms in Intelligent Systems. *AI magazine*.

Barret Zoph and Quoc V Le. 2017. Neural Architecture Search with Reinforcement Learning. In *Proc. of ICLR*.

Appendices

Appendix A

ConvNet's activations and skip-connections

In this appendix we analyze the impact of the skip connections on Pervasive Attention. We first compare the convergence speed in Figure A.1, where we observe that models with gated and cumulative skip connections converge faster.

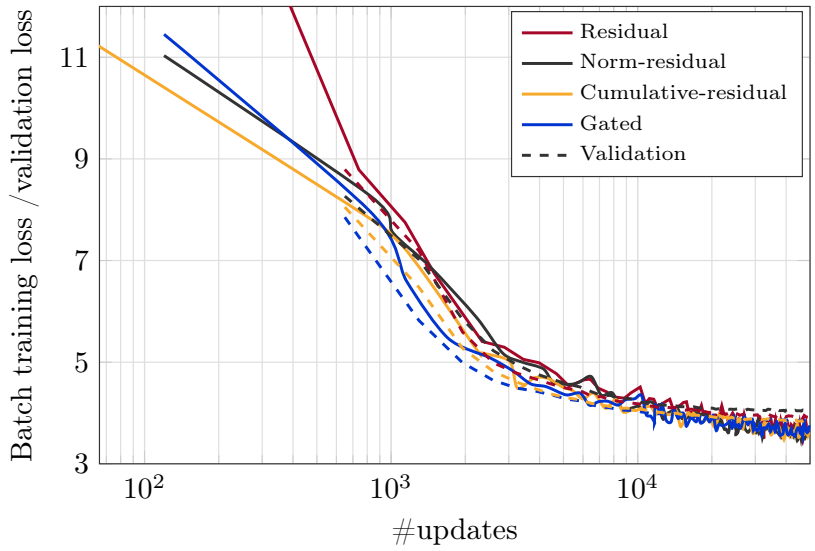


Figure A.1: Training (full) and validation (dashed) losses with different skip connections

To better understand the improvement achieved with cumulative skip connections, we profile the activations within the ConvNet using either standard residual skip connections or the cumulative and gated variant. For each model, we forward the development set in batches and in teacher-forcing mode. For each batch, we evaluate the mean and standard deviation of the activations at multiple points of the architecture then report the mean of the means and deviations at the corpus level. We observe that the variance of the final features

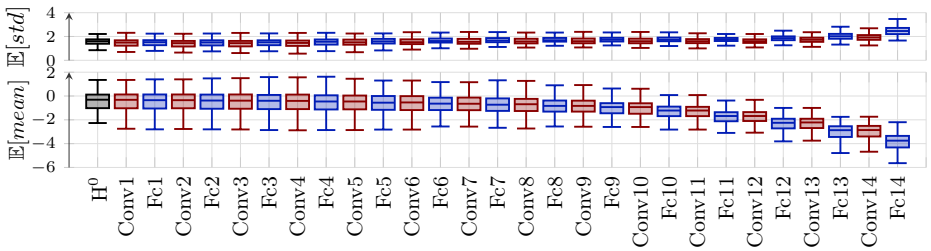


Figure A.2: Residual

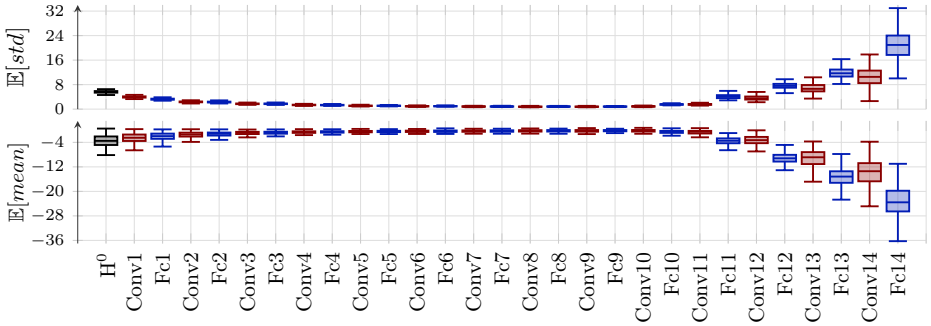


Figure A.3: Residual-cumulative

(a) forward

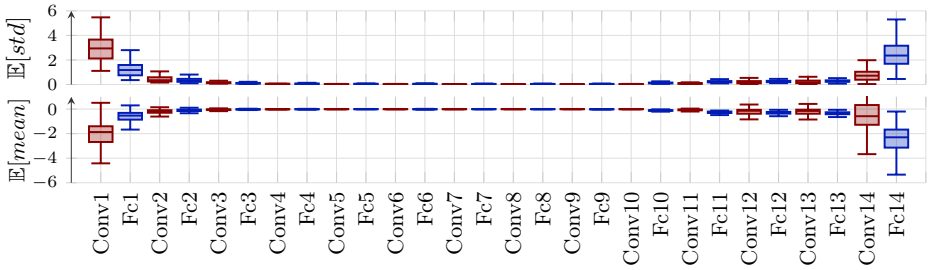


Figure A.4: Residual-gated

with accumulated outputs is higher than with standard residual skip connections *i.e.* the classifier's input is more *separable* as it occupies a larger space. We can also visualise the lesser importance accorded to mid-level features in both the gated (forward) and cumulative ConvNets. The features' profile is U-shaped with the activations getting larger as we approach the end of the ConvNet which means that the nearest and earliest (input embeddings) set of features are more important and contribute more to the input of the upcoming layer.

Appendix B

FLOPs approximation

Timing the decoding is a noisy estimation of the speedup we can achieve with adaptive depth estimation. With a large batch size applying the same operator on a large matrix is faster on GPU than different operators applied to chunks of the same matrix. With adaptive exits samples within a batch are split depending on the chosen exit, hence we apply different operators to smaller matrices. Indexing and splitting also causes a slight overhead. To abstract away from the indirect costs of computation, we resort to counting the theoretical amount of floating point operations (FLOPs). This section details the computation of the FLOPs we report in §5.4.4. The per token FLOPs are for the decoder network only since we use an encoder of the same size for all models. We breakdown the FLOPs of every operation in Algorithm 3. We omit non-linearities, normalizations and residual connections with a first order cost. The main operations we account for are dot-products and by extension matrix-vector products since those represent the vast majority of FLOPs .

	Parameters
d_{dec}	decoder embedding dimension.
d_{enc}	encoder embedding dimension.
d_{FF}	The feed-forward network dimension.
$\lvert\boldsymbol{x}\rvert$	source length.
t	Current time-estep ($t \geq 1$).
$\lvert\mathcal{V}\rvert$	output vocabulary size.

Operation	FLOPs
Dot-product (d)	$2d - 1$
Linear $d_{\text{in}} \rightarrow d_{\text{out}}$	$2d_{\text{in}}d_{\text{out}}$

Table B.1: FLOPs of basic operations, key parameters and variables for the FLOPs estimation.

With this breakdown, the total computational cost at time-step t of a decoder block that

we actually go through, denoted with FC, is:

$$\mathrm{FC}(\boldsymbol{x}, t) = 12d_{\mathrm{dec}}^2 + 4d_{\mathrm{FF}}d_{\mathrm{dec}} + 4td_{\mathrm{dec}} + 4|\boldsymbol{x}|d_{\mathrm{dec}} + 4[\![\mathrm{FirstCall}]\!]|\boldsymbol{x}|d_{\mathrm{dec}}d_{\mathrm{enc}},$$

where the cost of mapping the source's keys and values is incurred the first time the block is called (flagged with FirstCall). This occurs at $t = 1$ for the baseline model but it is input-dependent with depth adaptive estimation and may never occur if all tokens exit early.

If skipped, a block still has to compute the keys and value of the self-attention block so the self-attention of future time-steps can function. We will denote this cost with FS and we have $\mathrm{FS} = 4d_{\mathrm{dec}}^2$.

Depending on the halting mechanism, an exit prediction cost, denoted with FP, is added:

$$\begin{aligned}
\text{Sequence-specific depth:} \quad & \mathrm{FP}(t, q(t)) = 2[\![t = 1]\!]\mathrm{N}d_{\mathrm{dec}} \\
\text{Token-specific Multinomial:} \quad & \mathrm{FP}(t, q(t)) = 2\mathrm{N}d_{\mathrm{dec}} \\
\text{Token-specific Geometric-like:} \quad & \mathrm{FP}(t, q(t)) = 2d_{\mathrm{dec}}q(t) \\
\text{Confidence thresholding:} \quad & \mathrm{FP}(t, q(t)) = 2q(t)|\mathcal{V}|d_{\mathrm{dec}}
\end{aligned}$$

For a set of source sequences $\{\boldsymbol{x}^{(i)}\}_{i\in\mathcal{I}}$ and generated hypotheses $\{\boldsymbol{y}^{(i)}\}_{i\in\mathcal{I}}$, the average flops per token is:

$$\begin{aligned}
\text{Baseline (N blocks):} \quad & \frac{1}{\sum_i |\boldsymbol{y}^{(i)}|} \sum_i \sum_{t=1}^{|\boldsymbol{y}^{(i)}|} \left(\mathrm{N}\ \mathrm{FC}(\boldsymbol{x}^{(i)}, t) + 2|\mathcal{V}|d_{\mathrm{dec}}\right) \\
\text{Adaptive depth:} \quad & \frac{1}{\sum_i |\boldsymbol{y}^{(i)}|} \sum_i \sum_{t=1}^{|\boldsymbol{y}^{(i)}|} q(t)\mathrm{FC}(\boldsymbol{x}^{(i)}, t) + (\mathrm{N} - q(t) - 1)\mathrm{FS} \\
& \qquad + \mathrm{FP}(t, q(t)) + 2|\mathcal{V}|d_{\mathrm{dec}}
\end{aligned}$$

In the case of confidence thresholding the final output prediction cost ($2|\mathcal{V}|d_{\mathrm{dec}}$) is already accounted for in the exit prediction cost FP.

Algorithm 3 Adaptive decoding with a geometric-like exit distribution

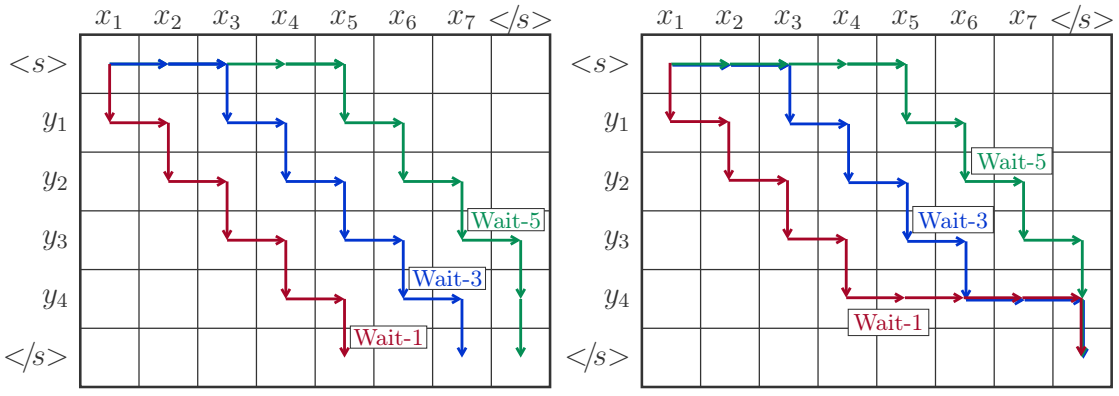
Input: source codes $\boldsymbol{s}$, incremental state
Initialization: $t = 1$, $\widetilde{y}_1 = <s>$
for $n \in [1..\mathrm{N}]$ **do**
    FirstCall[n] = True. ▷ A flag signaling if $\boldsymbol{s}$'s keys and values should be evaluated.
end for
while $\widetilde{y}_t \neq </s>$ **do**
    Embed the last output token $\widetilde{y}_t$.
    for $n \in [1..\mathrm{N}]$ **do**
        ▷ **Self-attention.**
        - Map the input into a key (k) and value (v). FLOPS=$4d^2_{\text{dec}}$
        - Map the input into a query q. FLOPS=$2d^2_{\text{dec}}$
        - Score the memory keys with q to get the attention weights α. FLOPS=$4td_{\text{dec}}$
        - Map the attention output. FLOPS=$2d^2_{\text{dec}}$
        ▷ **Encoder-Decoder interaction.**
        if FirstCall[n] **then**
            Map $\boldsymbol{s}$ into keys and values for the n-th block. FLOPS=$4|\boldsymbol{x}|d_{\text{enc}}d_{\text{dec}}$
            FirstCall[n] = False
        end if
        - Map the input into a query q. FLOPS=$2d^2_{\text{dec}}$
        - Score the memory keys with q to get the attention weights α. FLOPS=$4|\boldsymbol{x}|d_{\text{dec}}$
        - Map the attention output. FLOPS=$2d^2_{\text{dec}}$
        Feed-forward network. FLOPS=$4d_{\text{dec}}d_{\text{FF}}$
        Estimate the halting probability χ^n_t. FLOPS=$2d_{\text{dec}}$
        if $\chi^n_t > \tau_n$ **then**
            Exit the loop (Line 8)
        end if
    end for
    if $n < \mathrm{N} - 1$ **then**
        ▷ **Skipped blocks.**
        for $n_s \in [n + 1..\mathrm{N} - 1]$ **do**
            Copy and map the copied state into a key (k) and value (v). FLOPS=$4d^2_{\text{dec}}$
        end for
    end if
    Project the final state and sample a new output token. FLOPS=$2|\mathcal{V}|d_{\text{dec}}$
    t++
end while

Appendix C

Online Neural Machine Translation: Additional Experiments

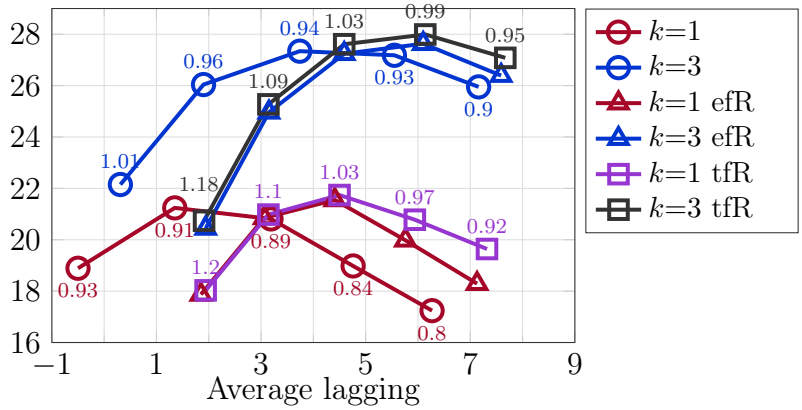
We investigate in this section the cause of the negative lagging that wait-k decoders fall into especially when decoding along the wait-1 path. Both AL and DAL are interpreted as the source steps by which we lag behind an ideal translator with values between 0 and $|\boldsymbol{x}|$. However, in cases where the system finishes decoding prematurely before the full source $|\boldsymbol{x}|$ is read, the lagging can be negative (better than the ideal translator). For example with $|\boldsymbol{x}| = 2|\boldsymbol{y}|$, the terms $z_t - \frac{|\boldsymbol{x}|}{|\boldsymbol{y}|}(t-1)$ on the wait-1 decoding path are equal to $2-t$ making for an average lagging of $2 - \frac{|\boldsymbol{y}|+1}{2}$, negative for any hypothesis with more than 3 tokens. In these scenarios the usually mediocre translation quality is paired with a better than ideal delay.

Finish reading the source $\boldsymbol{x}$. To counter this premature end of decoding, we force the model to *finish reading* the source sequence. This condition can be enforced either in training (tfR) or in evaluation time (efR). If included in training, we simply modify $\boldsymbol{z}$ so that $z_{|\boldsymbol{y}|}^{\text{wait-}k} = |\boldsymbol{x}|$ (see paths in Figure C.1b). During evaluation, efR amounts to ignoring the prediction of </s> until the source is fully read. The results in Figure C.1c show that models trained with the *finish reading* condition perform comparably to simply evaluating standard wait-k models with evaluation-time *finish reading*. Both methods score a positive AL but the standard decoding achieves better quality-delay trade-offs; the standard approach with $k_{\text{eval}} = 3$ has a comparable delay to *finish reading* models with $k_{\text{eval}} = 1$ with considerably better translation quality (+3.21 with $k = 1$ and +5.3 with $k = 3$). Forcing decoders to *finish reading* yields predictably larger length-ratios (total system tokens / total reference tokens) which might explain the lower BLEU scores. On the other hand, models that *finish reading*



(a) Standard wait-k

(b) Wait-k with *finish reading*



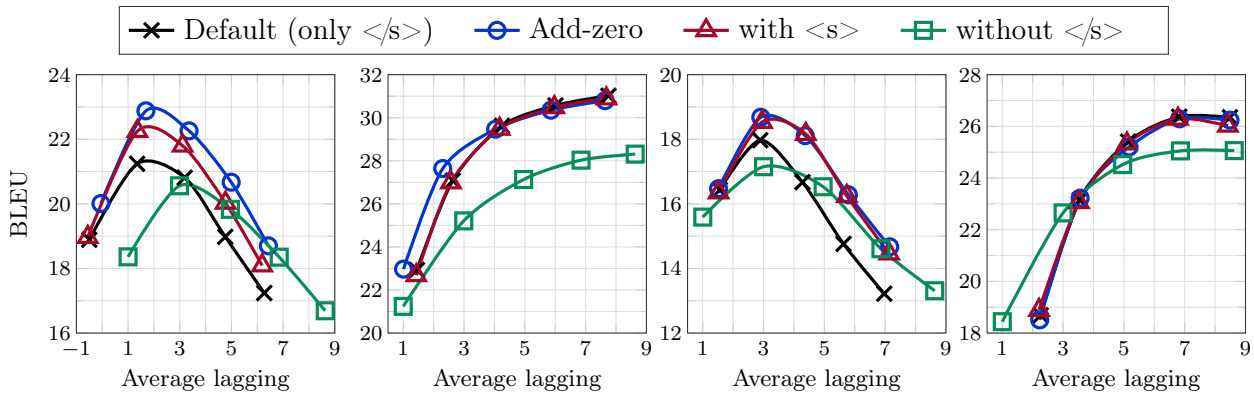
(c) IWSLT'14 De→En: Performance of Online Transformer models trained for wait-k decoding ($k \in \{1, 3\}$). The annotations near the markers show the system length-ratio.

Figure C.1: Training for wait-k decoding without reading the full source (left) and with the added condition of *finish reading* (right) in either training (tfR) or evaluation time (efR).

are marginally better in the less interesting high latency region (AL > 4.5).

In chapter 4, we default to decoding and training along the standard wait-k paths without the *finish reading* condition.

The source's <s> and </s> markers. In offline sequence-to-sequence models, depending on the implementation, we either append an end of sequence marker to the source $\boldsymbol{x}$ or keep it as-is. Zheng et al. (2019b) uses OpenNMT (Klein et al., 2017) and mention that the addition of </s> to the source helped improve the training of their wait-k models. In Fairseq (Ott et al., 2019), the source is encoded with </s> by default. When removing $</s>$, the accuracies across latency ranges fall by up to 8 BLEU points (not included in the graphs of



(a) De→En $k = 1$ (b) De→En $k = 7$ (c) En→De $k = 1$ (d) En→De $k = 7$

Figure C.2: Training Online Transformer wait-k models with different source markers.

Figure C.2). These models fail to learn when to stop decoding and generate longer hypotheses (length-ratio > 1.1). In Figure C.2, we assist the decoders trained without $</s>$ by forcing the end of decoding if and only if we have generated $|\boldsymbol{x}|$ target tokens. With such decoding strategy we improve the BLEU scores but they are still behind models with source $</s>$.

We also experimented with prefixing the source with $<s>$ allowing the decoder to not *read* any source token. When attending to $<s>$, its hidden state does not carry any information about the source sequence and the decoder hidden state will only encode the target decoded so far. Similarly, we add an artificial key-value pair to the source hidden states in the encoder-decoder attention mechanism. This artificial pair has a constant key and a zero value. If picked, this pair similar to the <s>token, does not include any source information and allow the decoder to skip the attention over the source tokens. We denote this setup with *add-zero*.

The results in Figure C.2 show a considerable improvement (up to 1.5 BLEU points) with either options over the default encoding, particularly for $k = 1$. We argue that allowing the model to skip attending over the source in online translation is helpful when the source token aligned with the target token is not yet read. In chapter 4, we keep the source encoding in its default format (only $</s>$).

Appendix D

Online Neural Machine Translation: Numerical Results

We provide in this appendix detailed results of the experiments in Chapter 4.

		IWSLT'14 De→En					IWSLT'15 En→Vi				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = 1$ bi-directional	BLEU	17.54	19.00	17.95	16.60	15.66	19.94	19.68	18.93	18.05	17.03
	AP	0.55	0.64	0.72	0.77	0.81	0.66	0.72	0.78	0.83	0.86
	AL	0.09	1.91	3.78	5.36	6.81	3.37	4.68	6.11	7.53	8.94
	DAL	2.44	4.12	5.99	7.72	9.28	4.51	5.78	7.16	8.58	9.99
$k = 1$ uni-directional	BLEU	18.89	21.24	20.81	18.98	17.24	25.72	25.88	24.13	21.69	19.94
	AP	0.50	0.60	0.68	0.74	0.79	0.63	0.71	0.77	0.81	0.85
	AL	-0.50	1.35	3.19	4.76	6.27	2.89	4.26	5.72	7.09	8.45
	DAL	1.52	3.22	5.06	6.90	8.65	3.43	4.75	6.21	7.66	9.10
$k = 1$ uni-directional $\Delta = 7$	BLEU	19.77	22.24	21.64	19.95	18.35	24.94	25.92	24.73	22.36	20.65
	AP	0.52	0.61	0.69	0.75	0.80	0.63	0.71	0.78	0.82	0.86
	AL	0.03	1.82	3.54	5.15	6.61	2.88	4.49	5.97	7.35	8.70
	DAL	1.68	3.36	5.17	6.99	8.71	3.66	5.12	6.55	7.97	9.38
$k = 1$ uni-directional $\Delta = 3$	BLEU	19.19	21.37	21.08	19.50	17.77	25.09	26.05	24.85	23.38	21.34
	AP	0.50	0.60	0.68	0.74	0.79	0.63	0.71	0.77	0.82	0.86
	AL	-0.48	1.32	3.16	4.78	6.30	2.97	4.44	5.95	7.37	8.67
	DAL	1.56	3.26	5.10	6.94	8.67	3.66	5.05	6.53	7.98	9.34
$k = 7$ bi-directional	BLEU	17.16	24.23	28.32	28.87	29.97	24.08	25.57	26.52	26.80	26.10
	AP	0.47	0.62	0.72	0.78	0.83	0.63	0.71	0.78	0.84	0.87
	AL	-1.48	1.67	4.01	5.88	7.61	2.51	4.14	5.97	7.71	9.15
	DAL	1.87	3.58	5.33	7.11	8.81	3.44	4.91	6.59	8.27	9.68
$k = 7$ uni-directional	BLEU	22.93	27.08	29.62	30.57	31.02	26.15	28.50	28.52	28.78	28.94
	AP	0.58	0.65	0.72	0.78	0.83	0.65	0.73	0.79	0.84	0.88
	AL	1.45	2.63	4.25	6.01	7.75	3.32	4.82	6.43	8.09	9.66
	DAL	2.11	3.44	5.19	7.00	8.74	3.97	5.43	7.04	8.68	10.22
$k = 7$ uni-directional $\Delta = 7$	BLEU	22.59	26.43	28.76	29.97	30.52	25.86	27.69	27.88	27.57	27.28
	AP	0.59	0.65	0.73	0.79	0.84	0.65	0.73	0.80	0.85	0.88
	AL	1.67	2.85	4.47	6.33	8.10	3.36	4.87	6.52	8.19	9.75
	DAL	2.35	3.70	5.46	7.31	9.01	4.04	5.53	7.17	8.82	10.35
$k = 7$ uni-directional $\Delta = 3$	BLEU	23.56	27.24	28.86	29.98	30.34	23.82	26.22	27.23	27.11	26.87
	AP	0.56	0.63	0.71	0.77	0.82	0.67	0.74	0.80	0.85	0.88
	AL	0.79	1.94	3.75	5.70	7.55	3.61	4.99	6.60	8.24	9.73
	DAL	1.90	3.34	5.16	7.00	8.74	4.47	5.71	7.29	8.92	10.33

Table D.1: Numerical results of Figure 4.9 with AP, AL and DAL latency metrics.

		IWSLT'14 De→En					IWSLT'14 En→De				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = \infty$ Max-pooling	BLEU	19.78	25.89	29.72	30.34	31.30	18.02	23.19	25.49	26.35	26.73
	AP	0.55	0.64	0.72	0.78	0.83	0.58	0.66	0.74	0.80	0.85
	AL	1.34	2.70	4.37	6.16	7.92	1.90	3.38	5.10	6.89	8.62
	DAL	1.42	3.18	5.11	6.99	8.74	2.06	3.66	5.47	7.29	9.01
$k = \infty$ Gated max-pooling	BLEU	19.63	25.68	28.72	30.24	31.10	17.76	23.04	25.54	26.34	26.59
	AP	0.65	0.64	0.72	0.78	0.83	0.58	0.67	0.74	0.80	0.85
	AL	1.45	2.81	4.44	6.20	7.96	1.99	3.44	5.16	6.96	8.68
	DAL	1.51	3.20	5.11	6.98	8.74	2.14	3.70	5.50	7.33	9.05
$k = \infty$ Self-attention	BLEU	16.36	22.33	26.62	29.44	31.18	16.89	22.28	25.44	26.58	26.95
	AP	0.51	0.61	0.70	0.77	0.83	0.57	0.66	0.74	0.80	0.85
	AL	-0.26	1.51	3.63	5.78	7.76	1.61	3.20	5.09	6.92	8.68
	DAL	1.30	3.13	5.08	6.98	8.73	1.99	3.61	5.49	7.33	9.06
$k = \infty$ Cell	BLEU	13.60	16.77	20.43	23.44	25.03	8.72	13.40	16.69	18.85	20.23
	AP	0.51	0.61	0.71	0.78	0.83	0.46	0.63	0.74	0.81	0.85
	AL	-0.48	1.68	3.98	6.07	7.95	-0.25	2.81	5.21	7.12	8.80
	DAL	2.04	3.62	5.31	7.09	8.82	2.44	4.26	5.86	7.56	9.18
$k = 7$ Max-pooling	BLEU	19.98	26.12	28.93	30.45	31.14	18.43	23.17	25.33	25.99	26.52
	AP	0.51	0.62	0.71	0.78	0.83	0.57	0.66	0.74	0.80	0.85
	AL	-0.27	1.78	3.93	6.00	7.90	1.73	3.34	5.12	6.94	8.67
	DAL	1.40	3.20	5.12	7.00	8.77	2.10	3.70	5.50	7.33	9.05
$k = 7$ Gated max-pooling	BLEU	21.11	26.63	28.96	30.58	31.18	17.93	22.88	25.09	25.69	25.87
	AP	0.54	0.63	0.71	0.78	0.83	0.58	0.66	0.74	0.80	0.85
	AL	0.66	2.21	4.09	6.08	7.92	1.89	3.34	5.07	6.87	8.60
	DAL	1.53	3.21	5.13	6.99	8.75	2.12	3.67	5.48	7.29	9.01
$k = 7$ Self-attention	BLEU	18.95	25.29	28.91	30.48	31.04	17.61	22.80	25.19	25.95	26.05
	AP	0.51	0.61	0.71	0.78	0.83	0.58	0.66	0.74	0.80	0.85
	AL	-0.36	1.62	3.92	6.11	8.03	1.88	3.39	5.16	6.94	8.66
	DAL	1.41	3.23	5.15	7.04	8.80	2.15	3.71	5.53	7.34	9.06
$k = 7$ Cell	BLEU	17.73	24.90	28.34	29.67	29.98	17.08	21.55	24.12	25.11	25.12
	AP	0.46	0.61	0.71	0.78	0.83	0.55	0.66	0.74	0.80	0.85
	AL	-1.26	1.63	3.90	5.95	7.83	1.21	3.33	5.08	6.86	8.55
	DAL	1.51	3.29	5.16	7.01	8.77	2.22	3.74	5.48	7.29	8.97
$k = 1$ Max-pooling	BLEU	20.63	24.69	25.96	26.56	26.80	18.02	23.19	25.49	26.35	26.73
	AP	0.52	0.63	0.72	0.79	0.84	0.58	0.66	0.74	0.80	0.85
	AL	0.03	2.20	4.37	6.35	8.19	1.90	3.38	5.10	6.89	8.62
	DAL	1.62	3.38	5.26	7.11	8.85	2.06	3.66	5.47	7.29	9.01
$k = 1$ Gated max-pooling	BLEU	20.14	23.83	24.80	24.98	24.62	17.76	23.04	25.54	26.34	26.59
	AP	0.52	0.63	0.72	0.79	0.84	0.58	0.67	0.74	0.80	0.85
	AL	-0.02	2.11	4.31	6.28	8.13	1.99	3.44	5.16	6.96	8.68
	DAL	1.62	3.40	5.31	7.16	8.90	2.14	3.70	5.50	7.33	9.05
$k = 1$ Self-attention	BLEU	18.88	20.71	20.29	19.43	18.40	16.89	22.28	25.44	26.58	26.95
	AP	0.54	0.66	0.75	0.81	0.85	0.57	0.66	0.74	0.80	0.85
	AL	0.74	3.07	5.26	7.24	8.96	1.61	3.20	5.09	6.92	8.68
	DAL	2.32	4.27	6.21	8.08	9.76	1.99	3.61	5.49	7.33	9.06
$k = 1$ Cell	BLEU	18.77	20.89	19.85	18.20	14.61	8.72	13.40	16.69	18.85	20.23
	AP	0.51	0.62	0.71	0.76	0.80	0.46	0.63	0.74	0.81	0.85
	AL	-0.11	1.96	3.95	5.54	6.78	-0.25	2.81	5.21	7.12	8.80
	DAL	1.69	3.46	5.33	7.12	8.83	2.44	4.26	5.86	7.56	9.18

Table D.2: Numerical results of Figure 4.10 with AP, AL and DAL latency metrics.

		Transformer					Pervasive Attention				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = 1$	BLEU	18.89	21.24	20.81	18.98	17.24	20.63	24.69	25.96	26.56	26.80
	AP	0.50	0.60	0.68	0.74	0.79	0.52	0.63	0.72	0.79	0.84
	AL	-0.50	1.35	3.19	4.76	6.27	0.03	2.20	4.37	6.35	8.19
	DAL	1.52	3.22	5.06	6.90	8.65	1.62	3.38	5.26	7.11	8.85
$k = 3$	BLEU	22.15	26.04	27.34	27.18	25.95	20.61	26.16	28.45	29.37	29.84
	AP	0.54	0.62	0.70	0.77	0.81	0.50	0.62	0.71	0.78	0.83
	AL	0.31	1.90	3.74	5.55	7.16	-0.30	1.89	4.11	6.13	7.99
	DAL	1.69	3.29	5.12	6.95	8.69	1.46	3.27	5.17	7.04	8.79
$k = 5$	BLEU	22.71	27.15	29.06	29.85	29.77	19.68	25.99	28.78	30.08	30.67
	AP	0.56	0.64	0.71	0.78	0.82	0.50	0.61	0.71	0.78	0.83
	AL	1.08	2.38	4.10	5.95	7.63	-0.44	1.71	3.99	6.03	7.91
	DAL	1.95	3.39	5.17	6.99	8.72	1.42	3.22	5.15	7.01	8.76
$k = 7$	BLEU	22.93	27.08	29.62	30.57	31.02	19.98	26.12	28.93	30.45	31.14
	AP	0.58	0.65	0.72	0.78	0.83	0.51	0.62	0.71	0.78	0.83
	AL	1.45	2.63	4.25	6.01	7.75	-0.27	1.78	3.93	6.00	7.90
	DAL	2.11	3.44	5.19	7.00	8.74	1.40	3.20	5.12	7.00	8.77
$k = 9$	BLEU	22.63	26.69	29.59	31.00	31.54	19.91	26.09	28.79	30.34	31.23
	AP	0.59	0.65	0.72	0.78	0.83	0.52	0.62	0.71	0.78	0.83
	AL	1.72	2.88	4.36	6.18	7.88	0.06	1.96	4.04	6.04	7.92
	DAL	2.26	3.54	5.25	7.05	8.76	1.43	3.23	5.13	7.01	8.77
$k = \infty$	BLEU	22.75	26.31	28.95	30.61	31.57	19.78	25.89	29.72	30.34	31.30
	AP	0.57	0.65	0.72	0.79	0.83	0.55	0.64	0.72	0.78	0.83
	AL	1.42	2.87	4.48	6.30	8.05	1.34	2.70	4.37	6.16	7.92
	DAL	1.94	3.43	5.22	7.06	8.79	1.42	3.18	5.11	6.99	8.74
$k \in [1..\lvert\boldsymbol{x}\rvert]$ or $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	21.66	26.46	29.33	30.84	31.51	19.20	25.73	29.96	30.44	31.36
	AP	0.51	0.62	0.71	0.78	0.83	0.47	0.60	0.71	0.78	0.83
	AL	-0.20	1.82	3.99	5.98	7.81	-1.26	1.28	3.81	5.97	7.90
	DAL	1.57	3.27	5.14	6.99	8.74	1.38	3.23	5.15	7.03	8.78

Table D.3: IWSLT'14 De→En: numerical results of Figures 4.11 and 4.12.

		Transformer					Pervasive Attention				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = 1$	BLEU	16.43	17.97	16.67	14.76	13.22	18.02	23.19	25.49	26.35	26.73
	AP	0.57	0.65	0.71	0.76	0.81	0.58	0.66	0.74	0.80	0.85
	AL	1.56	2.89	4.28	5.63	6.97	1.90	3.38	5.10	6.89	8.62
	DAL	2.12	3.53	5.19	6.95	8.67	2.06	3.66	5.47	7.29	9.01
$k = 3$	BLEU	17.78	22.10	23.68	22.38	20.97	18.12	22.34	23.91	24.34	24.58
	AP	0.59	0.66	0.73	0.79	0.83	0.57	0.66	0.74	0.80	0.85
	AL	2.00	3.32	4.85	6.36	7.81	1.54	3.27	5.10	6.95	8.68
	DAL	2.31	3.65	5.33	7.07	8.75	2.14	3.74	5.54	7.36	9.07
$k = 5$	BLEU	18.56	23.00	25.41	25.73	25.22	18.40	23.34	25.11	25.72	25.85
	AP	0.59	0.67	0.74	0.80	0.84	0.57	0.66	0.74	0.80	0.85
	AL	2.07	3.41	5.01	6.68	8.27	1.68	3.34	5.10	6.92	8.65
	DAL	2.37	3.73	5.42	7.17	8.86	2.12	3.70	5.52	7.33	9.04
$k = 7$	BLEU	18.69	23.15	25.41	26.37	26.35	18.43	23.17	25.33	25.99	26.52
	AP	0.60	0.67	0.74	0.80	0.85	0.57	0.66	0.74	0.80	0.85
	AL	2.26	3.55	5.14	6.82	8.49	1.73	3.34	5.12	6.94	8.67
	DAL	2.53	3.82	5.50	7.25	8.94	2.10	3.70	5.50	7.33	9.05
$k = 9$	BLEU	18.82	23.05	25.27	26.27	26.67	18.36	23.23	25.36	26.03	26.32
	AP	0.60	0.67	0.74	0.80	0.85	0.58	0.67	0.74	0.80	0.85
	AL	2.29	3.59	5.19	6.88	8.54	1.92	3.45	5.17	6.97	8.69
	DAL	2.54	3.85	5.53	7.29	8.97	2.16	3.74	5.53	7.34	9.06
$k = \infty$	BLEU	18.53	22.82	24.97	25.87	26.15	18.02	23.19	25.49	26.35	26.73
	AP	0.59	0.67	0.74	0.80	0.85	0.58	0.66	0.74	0.80	0.85
	AL	2.13	3.49	5.14	6.89	8.63	1.90	3.38	5.10	6.89	8.62
	DAL	2.33	3.74	5.51	7.31	9.04	2.06	3.66	5.47	7.29	9.01
$k \in [1..\lvert\boldsymbol{x}\rvert]$ or $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	18.60	23.04	25.07	25.80	26.09	19.05	23.55	25.44	26.17	26.39
	AP	0.56	0.66	0.74	0.80	0.85	0.55	0.66	0.74	0.80	0.85
	AL	1.51	3.36	5.15	6.96	8.68	1.05	3.14	5.05	6.92	8.68
	DAL	2.23	3.77	5.53	7.34	9.07	2.08	3.72	5.52	7.34	9.06

Table D.4: IWSLT'14 En→De: numerical results of Figure 4.11.

		Transformer					Pervasive Attention				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = 1$	BLEU	25.72	25.88	24.13	21.69	19.94	26.45	27.88	28.19	28.15	28.06
	AP	0.63	0.71	0.77	0.81	0.85	0.63	0.72	0.79	0.84	0.88
	AL	2.89	4.26	5.72	7.09	8.45	2.87	4.60	6.38	8.08	9.67
	DAL	3.43	4.75	6.21	7.66	9.10	3.52	5.17	6.94	8.64	10.20
$k = 3$	BLEU	25.83	27.59	27.45	27.15	26.18	27.31	29.24	29.15	29.13	29.11
	AP	0.64	0.72	0.79	0.83	0.87	0.63	0.72	0.79	0.84	0.88
	AL	3.10	4.58	6.21	7.74	9.22	3.07	4.70	6.41	8.10	9.70
	DAL	3.77	5.18	6.78	8.30	9.76	3.59	5.26	6.98	8.67	10.24
$k = 5$	BLEU	26.22	28.29	28.57	28.54	28.13	26.90	29.00	29.18	29.26	29.27
	AP	0.64	0.72	0.79	0.84	0.88	0.63	0.72	0.79	0.84	0.88
	AL	3.20	4.74	6.35	7.98	9.50	3.08	4.67	6.39	8.11	9.70
	DAL	3.82	5.31	6.92	8.53	10.00	3.61	5.22	6.97	8.68	10.24
$k = 7$	BLEU	26.15	28.50	28.52	28.78	28.94	26.90	28.81	29.01	29.11	29.12
	AP	0.65	0.73	0.79	0.84	0.88	0.63	0.72	0.79	0.84	0.88
	AL	3.32	4.82	6.43	8.09	9.66	3.06	4.65	6.37	8.09	9.68
	DAL	3.97	5.43	7.04	8.68	10.22	3.56	5.19	6.94	8.64	10.21
$k = 9$	BLEU	25.59	27.90	28.38	28.72	28.78	26.84	29.22	29.29	29.48	29.47
	AP	0.65	0.72	0.79	0.84	0.88	0.64	0.72	0.79	0.84	0.88
	AL	3.24	4.73	6.41	8.09	9.64	3.09	4.68	6.41	8.11	9.70
	DAL	3.86	5.31	7.02	8.68	10.19	3.62	5.23	6.99	8.68	10.24
$k = \infty$	BLEU	26.54	28.07	28.48	28.59	28.58	27.28	29.06	29.23	29.21	29.25
	AP	0.64	0.72	0.79	0.84	0.88	0.64	0.72	0.79	0.84	0.88
	AL	3.00	4.57	6.26	7.99	9.58	3.08	4.67	6.39	8.10	9.68
	DAL	3.50	5.10	6.82	8.54	10.09	3.59	5.22	6.96	8.66	10.21
$k \in [1..\lvert\boldsymbol{x}\rvert]$ or $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	26.61	28.31	28.67	28.63	28.64	27.35	29.27	29.57	29.65	29.67
	AP	0.63	0.71	0.78	0.84	0.88	0.63	0.72	0.79	0.84	0.88
	AL	2.80	4.41	6.14	7.88	9.48	2.91	4.65	6.39	8.11	9.69
	DAL	3.32	4.90	6.64	8.38	9.94	3.56	5.21	6.97	8.66	10.22

Table D.5: IWSLT'14 En→Vi: numerical results of Figures 4.11 and 4.12.

		Transformer					Pervasive Attention				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = 1$	BLEU	8.29	9.77	9.07	8.48	7.53	9.37	11.41	12.52	12.53	11.47
	AP	0.42	0.51	0.60	0.66	0.72	0.43	0.55	0.65	0.72	0.78
	AL	-2.83	-1.03	1.21	2.89	4.72	-2.82	0.19	2.60	4.91	7.16
	DAL	1.44	3.19	5.21	6.98	8.83	1.76	3.80	5.51	7.47	9.51
$k = 3$	BLEU	12.37	16.10	17.21	17.42	16.58	12.09	15.69	17.69	19.03	19.88
	AP	0.48	0.56	0.64	0.70	0.75	0.44	0.54	0.63	0.70	0.77
	AL	-1.18	0.60	2.37	4.24	6.10	-2.18	-0.26	2.07	4.13	6.40
	DAL	1.58	3.27	5.16	6.98	8.92	1.36	3.19	5.13	6.95	8.84
$k = 5$	BLEU	13.56	17.43	19.29	20.68	20.35	13.36	17.40	19.39	21.22	22.07
	AP	0.47	0.54	0.63	0.70	0.76	0.46	0.55	0.64	0.71	0.77
	AL	-1.38	0.08	2.32	4.25	6.38	-1.63	0.31	2.46	4.41	6.34
	DAL	1.79	3.24	5.28	7.03	9.05	1.39	3.19	5.12	6.98	8.79
$k = 7$	BLEU	14.93	18.16	20.55	21.36	22.93	12.04	16.53	19.10	21.08	22.56
	AP	0.50	0.56	0.64	0.71	0.76	0.43	0.53	0.62	0.70	0.76
	AL	-0.75	0.57	2.31	4.51	6.25	-2.65	-0.38	1.73	3.91	6.16
	DAL	1.66	3.26	5.14	7.04	8.79	1.26	3.07	5.02	6.94	8.79
$k = 9$	BLEU	14.56	17.75	20.48	21.70	22.96	12.56	16.28	19.66	21.94	22.89
	AP	0.52	0.58	0.65	0.71	0.77	0.45	0.54	0.63	0.70	0.76
	AL	0.06	1.12	2.69	4.64	6.48	-1.97	-0.22	1.88	4.02	6.07
	DAL	1.78	3.30	5.10	7.04	8.90	1.29	3.09	5.02	6.94	8.78
$k = \infty$	BLEU	15.46	18.03	19.93	21.70	23.35	10.71	14.71	17.94	20.34	22.80
	AP	0.54	0.60	0.67	0.73	0.77	0.54	0.61	0.67	0.73	0.78
	AL	0.80	2.00	3.50	5.15	6.88	1.31	2.46	3.80	5.29	6.95
	DAL	1.65	3.20	5.11	6.98	8.79	1.34	3.08	5.02	6.95	8.79
$k \in [1..\lvert\boldsymbol{x}\rvert]$ or $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	10.19	15.74	19.35	22.23	24.11	10.99	15.33	19.03	21.36	23.10
	AP	0.38	0.50	0.60	0.69	0.75	0.39	0.50	0.61	0.69	0.76
	AL	-3.93	-1.46	1.10	3.48	5.74	-3.68	-1.51	1.25	3.66	5.90
	DAL	1.23	3.06	5.01	6.93	8.77	1.20	3.10	5.03	6.97	8.78

Table D.6: IWSLT'14 Vi→En: numerical results of Figure 4.11.

		Transformer *base*					Transformer *big*				
	k_{eval}	1	3	5	7	9	1	3	5	7	9
$k = 7$	BLEU	18.30	24.62	27.08	28.43	29.01	19.16	24.68	27.12	28.94	29.73
	AP	0.54	0.60	0.67	0.74	0.79	0.53	0.60	0.67	0.73	0.78
	AL	0.49	2.02	3.90	5.87	7.76	0.21	1.94	3.78	5.77	7.66
	DAL	2.03	3.38	5.23	7.11	8.94	1.95	3.40	5.23	7.10	8.93
$k = \infty$	BLEU	20.26	22.14	23.64	25.23	26.43	20.37	21.45	22.94	24.79	26.13
	AP	0.60	0.66	0.72	0.77	0.81	0.59	0.65	0.71	0.76	0.81
	AL	2.60	4.26	5.81	7.42	9.07	2.42	3.69	5.28	7.08	8.73
	DAL	3.55	5.05	6.70	8.29	9.88	3.74	5.07	6.69	8.32	9.93
$k \in [1..\lvert\boldsymbol{x}\rvert]$	BLEU	18.95	23.85	26.36	28.03	29.00	19.60	24.54	27.68	29.15	30.15
	AP	0.50	0.59	0.67	0.73	0.79	0.53	0.60	0.68	0.74	0.79
	AL	-0.47	1.58	3.69	5.81	7.80	0.46	2.06	4.08	6.03	7.84
	DAL	1.70	3.34	5.21	7.11	8.95	1.90	3.43	5.27	7.15	8.96
$k = k_{\text{eval}}$	BLEU	18.26	24.00	26.97	28.43	29.29					
	AP	0.51	0.60	0.67	0.74	0.79					
	AL	-0.11	1.89	3.87	5.87	7.79					
	DAL	1.78	3.37	5.25	7.11	8.92					

Table D.7: WMT'15 De→En: numerical results of Figure 4.13.

	k_{eval}	1	3	5	7	9
Initialization $k = \infty$	BLEU	19.54	25.69	28.84	30.38	31.15
	AP	0.56	0.64	0.72	0.78	0.83
	AL	1.42	2.80	4.45	6.24	7.98
	DAL	1.47	3.21	5.12	7.00	8.75

	λ	4	3	2	1	0
Oracle	BLEU	29.46	30.96	31.85	32.24	32.57
	AP	0.65	0.66	0.68	0.69	0.72
	AL	3.34	3.71	4.07	4.47	5.05
	DAL	6.31	6.70	7.10	7.54	8.28

	τ	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 1$	BLEU	21.48	25.71	28.69	30.38	31.32	31.88
	AP	0.54	0.60	0.64	0.68	0.71	0.75
	AL	-0.34	1.22	2.52	3.54	4.50	5.48
	DAL	3.43	4.20	4.99	5.77	6.67	7.71

	τ	0.9	0.85	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 0$	BLEU	23.60	26.00	27.50	29.39	30.56	31.14	31.63	32.02	32.24
	AP	0.55	0.58	0.60	0.63	0.66	0.69	0.72	0.76	0.79
	AL	0.66	1.40	1.91	2.67	3.32	3.98	4.73	5.65	6.86
	DAL	2.62	3.05	3.47	4.25	4.97	5.72	6.56	7.59	8.93

Table D.8: IWSLT'14 De→En: numerical results of panel (a) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k=7$	BLEU	19.98	26.12	28.93	30.45	31.14
	AP	0.51	0.62	0.71	0.78	0.83
	AL	-0.27	1.78	3.93	6.00	7.90
	DAL	1.40	3.20	5.12	7.00	8.77

	λ	4	3	2	1	0
Oracle	BLEU	27.51	29.10	30.02	30.51	30.80
	AP	0.53	0.56	0.58	0.60	0.64
	AL	0.22	0.96	1.51	1.95	2.68
	DAL	4.04	4.57	5.08	5.54	6.30

	τ	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda=1$	BLEU	20.08	24.80	27.79	29.49	30.65	31.28
	AP	0.52	0.58	0.62	0.65	0.68	0.72
	AL	-0.94	0.66	1.87	2.81	3.65	4.48
	DAL	3.01	3.70	4.36	5.01	5.72	6.54

	τ	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda=0$	BLEU	23.74	27.57	28.61	29.30	30.28	30.81	31.22	31.48
	AP	0.54	0.59	0.61	0.62	0.64	0.67	0.69	0.72
	AL	0.37	1.65	2.06	2.39	2.92	3.46	4.03	4.73
	DAL	2.38	3.18	3.53	3.86	4.48	5.09	5.74	6.50

Table D.9: IWSLT'14 De→En: numerical results of panel (b) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	19.20	25.73	28.96	30.44	31.36
	AP	0.47	0.60	0.71	0.78	0.83
	AL	-1.26	1.28	3.81	5.97	7.90
	DAL	1.38	3.23	5.15	7.03	8.78

	λ	4	3	2	1	0
Oracle	BLEU	24.57	26.84	28.01	28.71	29.37
	AP	0.45	0.49	0.52	0.54	0.60
	AL	-1.39	-0.45	0.16	0.66	1.58
	DAL	3.15	3.76	4.27	4.73	5.64

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda=1$	BLEU	15.20	19.99	23.97	27.04	29.08	30.58	31.47
	AP	0.44	0.51	0.57	0.61	0.64	0.68	0.71
	AL	-2.57	-1.01	0.35	1.60	2.56	3.51	4.47
	DAL	2.42	3.03	3.70	4.35	5.00	5.74	6.62

	τ	0.9	0.85	0.8	0.75	0.7	0.6	0.5	0.4	0.35	0.3
Dynamic $\lambda=0$	BLEU	22.15	25.21	26.80	27.88	28.90	30.14	30.86	31.24	31.39	31.49
	AP	0.51	0.55	0.58	0.60	0.61	0.64	0.67	0.69	0.71	0.72
	AL	-0.39	0.59	1.23	1.72	2.14	2.80	3.41	4.01	4.34	4.70
	DAL	2.19	2.67	3.04	3.43	3.79	4.45	5.08	5.76	6.11	6.51

Table D.10: IWSLT'14 De→En: numerical results of panel (c) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k = \infty$	BLEU	18.54	23.27	25.38	26.19	26.50
	AP	0.58	0.67	0.74	0.80	0.85
	AL	2.01	3.46	5.19	6.98	8.69
	DAL	2.17	3.71	5.53	7.35	9.06

	λ	6	5	4	3	2	1	0
Oracle	BLEU	22.77	24.35	25.42	26.19	26.66	27.05	27.26
	AP	0.62	0.63	0.64	0.65	0.66	0.68	0.71
	AL	2.51	2.71	2.88	3.06	3.33	3.69	4.35
	DAL	5.34	5.50	5.69	5.89	6.16	6.55	7.30

	τ	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 1$	BLEU	21.48	25.71	28.69	30.38	31.32	31.88
	AP	0.54	0.60	0.64	0.68	0.71	0.75
	AL	-0.34	1.22	2.52	3.54	4.50	5.48
	DAL	3.43	4.20	4.99	5.77	6.67	7.71

	τ	0.9	0.85	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 0$	BLEU	23.60	26.00	27.50	29.39	30.56	31.14	31.63	32.02	32.24
	AP	0.55	0.58	0.60	0.63	0.66	0.69	0.72	0.76	0.79
	AL	0.66	1.40	1.91	2.67	3.32	3.98	4.73	5.65	6.86
	DAL	2.62	3.05	3.47	4.25	4.97	5.72	6.56	7.59	8.93

Table D.11: IWSLT'14 En→De: numerical results of panel (d) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k = 7$	BLEU	18.43	23.17	25.33	25.99	26.25
	AP	0.57	0.66	0.74	0.80	0.85
	AL	1.73	3.34	5.12	6.94	8.67
	DAL	2.10	3.70	5.50	7.33	9.05

	λ	5	4	3	2	1	0
Oracle	BLEU	24.12	24.94	25.69	26.24	26.70	26.93
	AP	0.56	0.59	0.61	0.63	0.64	0.67
	AL	0.80	1.50	2.02	2.43	2.75	3.39
	DAL	4.26	4.69	5.08	5.43	5.75	6.42

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 1$	BLEU	15.96	20.12	23.02	25.02	26.17	26.47	26.72
	AP	0.51	0.57	0.62	0.65	0.68	0.71	0.74
	AL	-1.14	0.44	1.73	2.73	3.55	4.23	4.94
	DAL	2.92	3.51	4.10	4.65	5.22	5.85	6.60

	τ	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 0$	BLEU	24.91	25.65	26.19	26.63	26.74	26.86	26.91
	AP	0.65	0.67	0.69	0.72	0.74	0.77	0.81
	AL	2.92	3.38	3.88	4.42	5.06	5.88	7.11
	DAL	4.12	4.65	5.20	5.81	6.55	7.46	8.77

Table D.12: IWSLT'14 En→De: numerical results of panel (e) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	19.05	23.55	25.44	26.17	26.39
	AP	0.55	0.66	0.74	0.80	0.85
	AL	1.05	3.14	5.05	6.92	8.68
	DAL	2.08	3.72	5.52	7.34	9.06

	λ	5	4	3	2	1	0
Oracle	BLEU	20.94	22.89	24.30	25.06	25.41	25.88
	AP	0.45	0.49	0.52	0.55	0.57	0.62
	AL	-1.30	-0.41	0.41	0.87	1.30	2.09
	DAL	3.15	3.66	4.21	4.58	4.98	5.73

	τ	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 1$	BLEU	18.66	21.99	24.60	26.03	26.44	26.72
	AP	0.55	0.60	0.64	0.68	0.70	0.73
	AL	-0.02	1.38	2.57	3.50	4.18	4.88
	DAL	3.29	3.89	4.49	5.08	5.70	6.44

	τ	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda = 0$	BLEU	20.75	23.85	25.25	25.94	26.35	26.70	26.81	26.86
	AP	0.59	0.63	0.65	0.68	0.70	0.73	0.75	0.78
	AL	1.69	2.53	3.07	3.57	4.10	4.67	5.34	6.19
	DAL	3.00	3.73	4.31	4.88	5.47	6.14	6.90	7.85

Table D.13: IWSLT'14 En→De: numerical results of panel (f) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k = \infty$	BLEU	10.71	14.71	17.94	20.34	22.80
	AP	0.54	0.61	0.67	0.73	0.78
	AL	1.31	2.46	3.80	5.29	6.95
	DAL	1.34	3.08	5.02	6.95	8.79

	λ	5.0	4.0	3.0	2.0	1.0	0.0
Oracle	BLEU	24.63	25.58	26.03	26.24	26.43	26.72
	AP	0.64	0.64	0.65	0.66	0.67	0.69
	AL	3.54	3.65	3.89	4.16	4.36	4.82
	DAL	6.47	6.59	6.86	7.14	7.38	7.92

	τ	0.8	0.7	0.6	0.5	0.4	0.3
Dynamic $\lambda = 1$	BLEU	12.16	17.89	22.58	24.80	25.88	26.25
	AP	0.44	0.53	0.59	0.63	0.66	0.68
	AL	-2.96	-0.58	1.69	2.93	3.83	4.45
	DAL	2.91	3.55	4.24	4.83	5.41	6.00

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 0$	BLEU	21.84	24.44	25.52	26.05	26.37	26.37	26.42	26.57
	AP	0.56	0.61	0.63	0.66	0.68	0.71	0.74	0.79
	AL	0.96	2.42	3.20	3.83	4.43	5.12	6.09	7.71
	DAL	3.52	4.26	4.80	5.30	5.93	6.69	7.72	9.54

Table D.14: IWSLT'15 Vi→En: numerical results of panel (g) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k=7$	BLEU	12.04	16.53	19.10	21.08	22.56
	AP	0.43	0.53	0.62	0.70	0.76
	AL	-2.65	-0.38	1.73	3.91	6.16
	DAL	1.26	3.07	5.02	6.94	8.79

	λ	4	3	2.0	1	0
Oracle	BLEU	19.18	21.39	23.00	23.80	24.11
	AP	0.47	0.51	0.55	0.58	0.61
	AL	-1.47	-0.20	0.93	1.56	2.22
	DAL	3.66	4.34	5.11	5.67	6.27

	τ	0.8	0.7	0.6	0.5	0.4	0.3
Dynamic $\lambda=1$	BLEU	12.16	17.89	22.58	24.80	25.88	26.25
	AP	0.44	0.53	0.59	0.63	0.66	0.68
	AL	-2.96	-0.58	1.69	2.93	3.83	4.45
	DAL	2.91	3.55	4.24	4.83	5.41	6.00

	τ	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda=0$	BLEU	17.55	22.62	24.64	24.95	25.73	25.70	25.76	25.79
	AP	0.51	0.57	0.60	0.62	0.64	0.66	0.69	0.72
	AL	-0.42	1.62	2.39	2.88	3.44	3.93	4.52	5.28
	DAL	2.64	3.47	3.96	4.40	4.89	5.40	6.03	6.82

Table D.15: IWSLT'15 Vi→En: numerical results of panel (h) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	10.99	15.33	19.03	21.36	23.10
	AP	0.39	0.50	0.61	0.69	0.76
	AL	-3.68	-1.51	1.25	3.66	5.90
	DAL	1.20	3.10	5.03	6.97	8.78

	λ	5	4	3	2.0	1	0
Oracle	BLEU	15.80	18.09	21.14	23.12	23.89	24.42
	AP	0.36	0.40	0.46	0.51	0.54	0.59
	AL	-3.69	-2.62	-0.98	0.24	0.87	1.67
	DAL	2.53	2.90	3.72	4.55	5.09	5.83

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda=1$	BLEU	12.14	18.24	22.68	24.66	25.58	26.00	26.17
	AP	0.42	0.51	0.58	0.62	0.64	0.66	0.69
	AL	-3.20	-0.68	1.42	2.71	3.40	3.92	4.51
	DAL	2.60	3.27	3.90	4.46	4.92	5.41	6.05

	τ	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2
Dynamic $\lambda=0$	BLEU	17.81	23.43	24.94	25.77	26.17	26.28	26.24	26.35
	AP	0.50	0.58	0.61	0.63	0.65	0.68	0.70	0.73
	AL	-0.78	1.53	2.46	3.17	3.67	4.23	4.86	5.72
	DAL	2.54	3.47	4.05	4.63	5.15	5.71	6.38	7.33

Table D.16: IWSLT'15 Vi→En: numerical results of panel (i) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k = \infty$	BLEU	27.28	29.06	29.23	29.21	29.25
	AP	0.64	0.72	0.79	0.84	0.88
	AL	3.08	4.67	6.39	8.10	9.68
	DAL	3.59	5.22	6.96	8.66	10.21

	λ	7	0
Oracle	BLEU	27.85	28.87
	AP	0.64	0.68
	AL	2.78	3.62
	DAL	5.14	6.23

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 1$	BLEU	16.91	22.59	26.37	28.23	28.92	29.12	29.34	29.46
	AP	0.48	0.55	0.60	0.63	0.64	0.67	0.69	0.73
	AL	-1.39	0.33	1.53	2.29	2.71	3.14	3.66	4.44
	DAL	2.54	3.08	3.52	3.93	4.31	4.71	5.23	6.04

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 0$	BLEU	27.84	28.50	28.92	29.07	29.12	29.38	29.47	29.45
	AP	0.60	0.62	0.63	0.65	0.67	0.70	0.72	0.77
	AL	1.80	2.17	2.46	2.79	3.18	3.72	4.39	5.43
	DAL	3.24	3.58	3.91	4.28	4.70	5.21	5.91	7.06

Table D.17: IWSLT'15 En→Vi: numerical results of panel (j) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $k = 7$	BLEU	26.90	28.81	29.01	29.11	29.12
	AP	0.63	0.72	0.79	0.84	0.88
	AL	3.06	4.65	6.37	8.09	9.68
	DAL	3.56	5.19	6.94	8.64	10.21

	λ	1	0
Oracle	BLEU	27.75	28.15
	AP	0.65	0.66
	AL	2.81	3.02
	DAL	5.29	5.57

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 1$	BLEU	17.48	23.72	27.41	28.45	28.79	29.15	29.25	29.33
	AP	0.49	0.56	0.60	0.63	0.64	0.66	0.69	0.72
	AL	-1.20	0.68	1.81	2.30	2.68	3.11	3.62	4.39
	DAL	2.49	3.05	3.46	3.80	4.16	4.58	5.12	5.92

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 0$	BLEU	27.86	28.44	28.68	28.76	29.05	29.26	29.31	29.30
	AP	0.59	0.61	0.62	0.64	0.66	0.69	0.71	0.75
	AL	1.68	2.04	2.31	2.63	3.02	3.50	4.11	5.06
	DAL	3.05	3.39	3.69	4.05	4.45	4.92	5.55	6.54

Table D.18: IWSLT'15 En→Vi: numerical results of panel (k) in Figure 4.14.

	k_{eval}	1	3	5	7	9
Initialization $\boldsymbol{z} \geq \boldsymbol{z}^{\text{wait-1}}$	BLEU	27.35	29.27	29.57	29.65	29.67
	AP	0.63	0.72	0.79	0.84	0.88
	AL	2.91	4.65	6.39	8.11	9.69
	DAL	3.56	5.21	6.97	8.66	10.22

	λ	1	0
Oracle	BLEU	27.05	27.35
	AP	0.55	0.58
	AL	1.55	2.03
	DAL	4.14	4.57

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 1$	BLEU	17.48	23.19	27.27	28.80	29.37	29.58	29.55	29.73
	AP	0.48	0.55	0.60	0.62	0.64	0.66	0.68	0.72
	AL	-1.41	0.34	1.65	2.18	2.62	3.00	3.48	4.23
	DAL	2.43	2.97	3.43	3.73	4.07	4.44	4.93	5.73

	τ	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
Dynamic $\lambda = 0$	BLEU	27.92	28.69	28.89	29.29	29.43	29.59	29.62	29.59
	AP	0.59	0.61	0.63	0.64	0.66	0.68	0.71	0.75
	AL	1.63	2.03	2.35	2.65	3.02	3.46	4.06	4.95
	DAL	3.02	3.41	3.76	4.09	4.45	4.90	5.50	6.45

Table D.19: IWSLT'15 En→Vi: numerical results of panel (l) in Figure 4.14.

Appendix E

Loss Smoothing for Language Models

Despite the effectiveness of neural language models, their maximum likelihood estimation suffers from two limitations. It treats all sentences that do not match the ground truth as equally poor, ignoring the structure of the output space. Second, it suffers from *exposure bias*: during training tokens are predicted given ground-truth sequences, while at test time prediction is conditioned on generated output sequences. To overcome these limitations, we smooth the training loss to tolerate outputs close to the target w.r.t. a similarity measure at the sequence-level or at the token-level. Our experiments on the tasks of image captioning and machine translation show that we can significantly improve the models' accuracy with loss smoothing.

The contributions presented in this appendix are published in:

> Maha Elbayad, Laurent Besacier, and Jakob Verbeek. 2018b. Token-level and Sequence-level Loss Smoothing for RNN Language Models. In *Proc. of ACL*

Contents

E.1 Introduction 186
E.2 Loss Smoothing for Language Models 187
E.3 Experiments 192
E.4 Related Work 198
E.5 Conclusion 200

E.1 Introduction

Neural language models are typically trained by maximizing the likelihood of the target sentence given an encoded source (text, image, speech). Maximum likelihood estimation (MLE) has two main limitations. First, the training signal only differentiates the ground-truth target output from all other outputs. It treats all other output sequences as equally incorrect, regardless of their semantic proximity from the ground-truth target. While such a *zero-one* loss is probably acceptable for coarse grained classification of images, *e.g.* across a limited number of basic object categories (Everingham et al., 2010) it becomes problematic as the output space becomes larger and some of its elements become semantically similar to each other. This is in particular the case for tasks that involve natural language generation (captioning, translation, speech recognition) where the number of possible outputs is practically unbounded. For natural language generation tasks, evaluation measures typically do take into account structural similarity, *e.g.* based on n-grams, but such structural information is not reflected in the MLE criterion. The second limitation of MLE is that training is based on predicting the next token given the input and preceding ground-truth output tokens, while at test time the model predicts conditioned on the input and the so-far generated output sequence. Given the exponentially large output space of natural language sentences, it is not obvious that the trained language model generalizes well beyond the relatively sparse distribution of ground-truth sequences used during MLE optimization. This phenomenon is known as *exposure bias* (Ranzato et al., 2016; Bengio et al., 2015).

MLE minimizes the KL divergence between a target Dirac distribution on the ground-truth sentence(s) and the model's distribution. In this work, we build upon the loss smoothing approach by Norouzi et al. (2016), which smooths the Dirac target distribution over similar sentences, increasing the support of the training data in the output space. We make the following main contributions:

a) We propose a token-level loss smoothing approach, using word-embeddings, to achieve smoothing among semantically similar terms, and we introduce a special procedure to promote rare tokens.

b) For sequence-level smoothing, we propose to use restricted token replacement vocabu-

laries, and a *lazy* evaluation method that significantly speeds up training.

c) We experimentally validate our approach on the MSCOCO image captioning task and the WMT'14 English to French machine translation task, showing that on both tasks combining token-level and sequence-level loss smoothing improves results significantly over maximum likelihood baselines.

In the remainder of this chapter, we present our token-level and sequence-level approaches in §E.2. Experimental evaluation results based on image captioning and machine translation tasks with recurrent language models are laid out in §E.3. We end the chapter with a review of existing methods related to our work (§E.4).

E.2 Loss Smoothing for Language Models

Let us recall that a conditional language model estimates the probability of an output sequence $\boldsymbol{y}$ given an input $\boldsymbol{x}$, $p(\boldsymbol{y} \,|\, \boldsymbol{x})$ factorized as:

$$p_\theta(\boldsymbol{y} \,|\, \boldsymbol{x}) = \prod_{t=1}^{|\boldsymbol{y}|} p_\theta(y_t \,|\, \boldsymbol{x}, \boldsymbol{y}_{<t}), \tag{E.1}$$

with θ the model parameters. The input $\boldsymbol{x}$ is a source sequence or an image in the contexts of machine translation and image captioning, respectively.

Depending on the architecture (see §2.4), the prefixes $\boldsymbol{y}_{<t}$ and the context $\boldsymbol{x}$ are encoded into hidden states $(h_t)_{1 \leq t \leq |\boldsymbol{y}|}$ topped with a softmax-normalized linear layer to predict the next token in $\boldsymbol{y}$. In standard teacher-forced training, the hidden states will be computed by forwarding the ground truth sequence $\boldsymbol{y}$ *i.e.* the model is predicting the next token given the ground truth history and not its own past predictions. In this chapter, to demarcate the ground truth output sequence, we will denote it with $\boldsymbol{y}^\star$ and denote the hidden states evaluated in teacher-forcing mode with $h_t^\star$.

Maximum likelihood estimation (MLE) of the model parameters θ amounts to minimizing the loss

$$\ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x}) = -\log p_\theta(\boldsymbol{y}^\star \,|\, \boldsymbol{x}) = -\sum_{t=1}^{|\boldsymbol{y}|} \log p_\theta(y_t^\star \,|\, h_t^\star). \tag{E.2}$$

This loss can equivalently be expressed as the KL-divergence between a Dirac delta distribution centered on the target output (with $\delta_a(x) = 1$ at $x = a$ and 0 otherwise) and the model's distribution, either at the sequence-level or at the token-level:

$$\ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x}) = \mathrm{D}_{\text{KL}}(\delta_{\boldsymbol{y}^\star} \,\|\, p_\theta(\boldsymbol{y} \,|\, \boldsymbol{x})) \tag{E.3}$$

$$= \sum_{t=1}^{|\boldsymbol{y}|} \mathrm{D}_{\text{KL}}\big(\delta_{y_t^\star} \,\|\, p_\theta(y_t \,|\, h_t^\star)\big)\,. \tag{E.4}$$

Loss smoothing approaches considered in this work consists in replacing the Dirac on the ground-truth sequence with distributions with larger support. These distributions can be designed in such a manner that they reflect which deviations from ground-truth predictions are preferred over others.

E.2.1 Sequence-level Loss Smoothing

The reward augmented maximum likelihood approach of Norouzi et al. (2016) consists in replacing the sequence-level Dirac δ_{y^*} in Eq. (E.3) with a distribution

$$r(\boldsymbol{y} \,|\, \boldsymbol{y}^\star) \propto \exp r(\boldsymbol{y}, \boldsymbol{y}^\star)/\tau, \tag{E.5}$$

where $r(\boldsymbol{y}, \boldsymbol{y}^\star)$ is a reward function that measures the quality of sequence $\boldsymbol{y}$ w.r.t. $\boldsymbol{y}^\star$, *e.g.* metrics used for evaluation of natural language processing tasks can be used, such as BLEU (Papineni et al., 2002) or CIDEr (Vedantam et al., 2015). The temperature parameter τ controls the concentration of the distribution around $\boldsymbol{y}^\star$. When $m > 1$ ground-truth sequences are paired with the same input $\boldsymbol{x}$, the reward function can be adapted to fit this setting and be defined as $r(\boldsymbol{y}, \{\boldsymbol{y}^{\star(1)}, \ldots, \boldsymbol{y}^{\star(m)}\})$. The sequence-level smoothed loss function is then given by

$$\ell_{\text{Seq}}(\boldsymbol{y}^\star, \boldsymbol{x}) = \mathrm{D}_{\text{KL}}(r(\boldsymbol{y} \,|\, \boldsymbol{y}^\star) \,\|\, p_\theta(\boldsymbol{y} \,|\, \boldsymbol{x})) = \mathrm{H}(r(\boldsymbol{y} \,|\, \boldsymbol{y}^\star)) - \mathbb{E}_r[\log p_\theta(\boldsymbol{y} \,|\, \boldsymbol{x})]\,, \tag{E.6}$$

where the entropy term $\mathrm{H}(r(y \,|\, y^*))$ does not depend on the model parameters θ.

In general, expectation in Eq. (E.6) is intractable due to the exponentially large output space, and replaced with a Monte-Carlo approximation:

$$\mathbb{E}_r[-\log p_\theta(\boldsymbol{y} \,|\, \boldsymbol{x})] \approx -\sum_{l=1}^{L} \log p_\theta(\boldsymbol{y}^l \,|\, \boldsymbol{x}). \tag{E.7}$$

Stratified sampling. Norouzi et al. (2016) show that when using the Hamming or edit distance as a reward, we can sample directly from $r(\boldsymbol{y} \mid \boldsymbol{y}^\star)$ using a stratified sampling approach. In this case sampling proceeds in three stages. (i) Sample a distance d from $\{0, \dots, |\boldsymbol{y}|\}$ from a prior distribution on d. (ii) Uniformly select d positions in the sequence to be modified. (iii) Sample the d substitutions uniformly from the token vocabulary.

Importance sampling. For a reward based on BLEU or CIDEr , we cannot directly sample from $r(\boldsymbol{y} \mid \boldsymbol{y}^\star)$ since the normalizing constant, or partition function, of the distribution is intractable to compute. In this case we can resort to importance sampling. We first sample L sequences $\boldsymbol{y}^l$ from a tractable proposal distribution $q(\boldsymbol{y} \mid \boldsymbol{y}^\star)$. We then compute the importance weights

$$\omega_l \approx \frac{r(\boldsymbol{y}^l \mid \boldsymbol{y}^\star)/q(\boldsymbol{y}^l \mid \boldsymbol{y}^\star)}{\sum_{k=1}^{L} r(\boldsymbol{y}^k \mid \boldsymbol{y}^\star)/q(\boldsymbol{y}^k \mid \boldsymbol{y}^\star)}, \tag{E.8}$$

where $r(\boldsymbol{y}^k \mid \boldsymbol{y}^\star)$ is the un-normalized reward distribution in Eq. (E.5). We finally approximate the expectation by reweighing the samples in the Monte Carlo approximation as

$$\mathbb{E}_r[-\log p_\theta(\boldsymbol{y} \mid \boldsymbol{x})] \approx -\sum_{l=1}^{L} \omega_l \log p_\theta(\boldsymbol{y}^l \mid \boldsymbol{x}). \tag{E.9}$$

In our experiments we use a proposal distribution based on the Hamming distance, which allows for tractable stratified sampling, and generates sentences that do not stray away from the ground truth.

We propose two modifications to the sequence-level loss smoothing of Norouzi et al. (2016): sampling to a restricted vocabulary (described in the following paragraph) and lazy sequence-level smoothing (described in section E.2.3).

Restricted vocabulary sampling. In the stratified sampling method for Hamming and edit distance rewards, instead of drawing from the large vocabulary $\mathcal{V}$, containing typically in the order of 10^4 words or more, we can restrict ourselves to a smaller subset $\mathcal{V}_{\text{sub}}$ more adapted to our task. We considered three different possibilities for $\mathcal{V}_{\text{sub}}$.

a) $\mathcal{V}$: the full vocabulary from which we sample uniformly (default), or draw from our token-level smoothing distribution defined below in Eq. (E.10).

b) $\mathcal{V}_{\text{refs}}$: uniformly sample from the set of tokens that appear in the ground-truth sentence(s) associated with the current input.

c) $\mathcal{V}_{\text{batch}}$: uniformly sample from the tokens that appear in the ground-truth sentences across all inputs that appear in a given training mini-batch.
Uniformly sampling from $\mathcal{V}_{\text{batch}}$ has the effect of boosting the frequencies of words that appear in many reference sentences, and thus approximates to some extent sampling substitutions from the uni-gram statistics of the training set.

E.2.2 Token-level Loss Smoothing

While the sequence-level smoothing can be directly based on performance measures of interest such as BLEU or CIDEr, the support of the smoothed distribution is limited to the number of samples drawn during training. We propose smoothing the token-level Diracs $\delta_{y_t^\star}$ in Eq. (E.4) to increase its support to similar tokens. Since we apply smoothing to each of the tokens independently, this approach implicitly increases the support to an exponential number of sequences, unlike the sequence-level smoothing approach. This comes at the price, however, of a naive token-level independence assumption in the smoothing.

We define the smoothed token-level distribution, similar as the sequence-level one, as a soft-max over a token-level reward function,

$$r(y_t \,|\, y_t^\star) \propto \exp r(y_t, y_t^\star)/\tau, \tag{E.10}$$

where τ is again a temperature parameter. As a token-level reward $r(y_t, y_t^\star)$ we use the cosine similarity between y_t and $y_t^\star$ in a semantic word-embedding space. In our experiments we use GloVe (Pennington et al., 2014); preliminary experiments with word2vec (Mikolov et al., 2013a) yielded somewhat worse results.

Promoting rare tokens. We can further improve the token-level smoothing by promoting rare tokens. To do so, we penalize frequent tokens when smoothing over the vocabulary, by subtracting $\beta\,\text{freq}(y_t)$ from the reward, where $\text{freq}(\cdot)$ denotes the term frequency and β is a non-negative weight. This modification encourages frequent tokens into considering the rare ones. We experimentally found that it is also beneficial for rare tokens to boost frequent

ones, as they tend to have mostly rare tokens as neighbors in the word-embedding space. With this in mind, we define a new token-level reward as:

$$r^{\text{freq}}(y_t, y_t^\star) = r(y_t, y_t^\star) - \beta \min\left(\frac{\text{freq}(y_t)}{\text{freq}(y_t^\star)}, \frac{\text{freq}(y_t^\star)}{\text{freq}(y_t)}\right), \tag{E.11}$$

where the penalty term is strongest if both tokens have similar frequencies.

E.2.3 Combining Losses

In both loss smoothing methods presented above, the temperature parameter τ controls the concentration of the distribution. As τ gets smaller the distribution peaks around the ground-truth, while for large τ the uniform distribution is approached. We can, however, not separately control the spread of the distribution and the mass reserved for the ground-truth output. We therefore introduce a second parameter $\alpha \in [0, 1]$ to interpolate between the Dirac on the ground-truth and the smooth distribution. Using $\bar{\alpha} = 1 - \alpha$, the sequence-level and token-level loss functions are then defined as

$$\ell^{\alpha}_{\text{Seq}}(\boldsymbol{y}^\star, \boldsymbol{x}) = \alpha \ell_{\text{Seq}}(\boldsymbol{y}^\star, \boldsymbol{x}) + \bar{\alpha} \ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x}) = \alpha \mathbb{E}_r[\ell_{\text{MLE}}(y, x)] + \bar{\alpha} \ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x}) \tag{E.12}$$

$$\ell^{\alpha}_{\text{Tok}}(\boldsymbol{y}^\star, \boldsymbol{x}) = \alpha \ell_{\text{Tok}}(\boldsymbol{y}^\star, \boldsymbol{x}) + \bar{\alpha} \ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x}) \tag{E.13}$$

To benefit from both sequence-level and token-level loss smoothing, we also combine them by applying token-level smoothing to the different sequences sampled for the sequence-level smoothing. We introduce two mixing parameters α_1 and α_2. The first controls to what extent sequence-level smoothing is used, while the second controls to what extent token-level smoothing is used. The combined loss is defined as

$$\begin{aligned} \ell^{\alpha_1,\alpha_2}_{\text{Seq, Tok}}(\boldsymbol{y}^\star, \boldsymbol{x}, r) &= \alpha_1 \mathbb{E}_r[\ell_{\text{Tok}}(\boldsymbol{y}, \boldsymbol{x})] + \bar{\alpha}_1 \ell_{\text{Tok}}(\boldsymbol{y}^\star, \boldsymbol{x}) \\ &= \alpha_1 \mathbb{E}_r[\alpha_2 \ell_{\text{Tok}}(\boldsymbol{y}, \boldsymbol{x}) + \bar{\alpha}_2 \ell_{\text{MLE}}(\boldsymbol{y}, \boldsymbol{x})] + \bar{\alpha}_1(\alpha_2 \ell_{\text{Tok}}(\boldsymbol{y}^\star, \boldsymbol{x}) + \bar{\alpha}_2 \ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x})). \end{aligned} \tag{E.14}$$

In our experiments, we use held out validation data to set mixing and temperature parameters.

Lazy sequence smoothing. Although sequence-level smoothing is computationally efficient compared to reinforcement learning approaches (Ranzato et al., 2016; Rennie et al., 2017), it is slower compared to MLE. In particular, we need to forward each of the samples $\boldsymbol{y}^l$ in

Algorithm 4 Sequence-level smoothing algorithm

Input: $\boldsymbol{x}, \boldsymbol{y}^\star$.
Output: $\ell^\alpha_{\text{seq}}(\boldsymbol{y}^\star, \boldsymbol{x})$.
  Encode x.
  Forward $\boldsymbol{y}^\star$ in the graph to compute the hidden states $h^\star_t$.
  Compute the MLE loss $\ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x})$.
  for $l \in [1..L]$ **do**
    Sample $\boldsymbol{y}^l \sim r(\cdot \,|\, \boldsymbol{y}^\star)$
    if Lazy **then**
      Compute $\ell(\boldsymbol{y}^l, \boldsymbol{x}) = -\sum_t \log p_\theta(y^l_t \,|\, h^\star_t)$
    else
      Forward $\boldsymbol{y}^l$ in the graph to get its hidden states h^l_t.
      Compute $\ell(\boldsymbol{y}^l, \boldsymbol{x}) = \ell_{\text{MLE}}(\boldsymbol{y}^l, \boldsymbol{x})$
    end if
  end for
  $\ell^\alpha_{\text{Seq}}(\boldsymbol{y}^\star, \boldsymbol{x}) = \bar{\alpha}\ell_{\text{MLE}}(\boldsymbol{y}^\star, \boldsymbol{x}) + \frac{\alpha}{L}\sum_l \ell(\boldsymbol{y}^l, \boldsymbol{x})$

the decoder in teacher-forcing mode so as to compute its hidden states h^l_t, which are used to compute the sequence MLE loss as

$$\ell_{\text{MLE}}(\boldsymbol{y}^l, \boldsymbol{x}) = -\sum_{t=1}^{|\boldsymbol{y}|} \log p_\theta(y^l_t \,|\, h^l_t). \tag{E.15}$$

To speed up training, and since we already forward the ground truth sequence in the decoder to evaluate the MLE part of $\ell^\alpha_{\text{Seq}}(\boldsymbol{y}^\star, \boldsymbol{x})$, we propose to use the same hidden states $h^\star_t$ to compute both the MLE and the sequence-level smoothed loss. In this case:

$$\ell_{\text{lazy}}(\boldsymbol{y}^l, x) = -\sum_{t=1}^{|\boldsymbol{y}|} \log p_\theta(y^l_t \,|\, h^\star_t) \tag{E.16}$$

In this manner, we only have a single instead of $L + 1$ decoder forwards-passes. We provide the pseudo-code for training in Algorithm 4.

E.3 Experiments

In this section, we compare sequence prediction models trained with maximum likelihood (MLE) with our token and sequence-level loss smoothing on two different tasks: image captioning and machine translation. Our methods are architecture-independent, and so,

			Without attention			**With attention**		
Loss	Reward	$\mathcal{V}_{sub}$	BLEU-1	BLEU-4	CIDEr	BLEU-1	BLEU-4	CIDEr
MLE			70.63	30.14	93.59	73.40	33.11	101.63
MLE + γH			70.79	30.29	93.61	72.68	32.15	99.77
Tok	Glove sim		71.94	31.27	95.79	73.49	32.93	102.33
Tok	Glove sim r^{freq}		72.39	31.76	97.47	74.01	33.25	102.81
Seq	Hamming	$\mathcal{V}$	71.76	31.16	96.37	73.12	32.71	101.25
Seq	Hamming	$\mathcal{V}_{batch}$	71.46	31.15	**96.53**	73.26	**32.73**	101.90
Seq	Hamming	$\mathcal{V}_{refs}$	**71.80**	**31.63**	96.22	**73.53**	32.59	**102.33**
Seq, lazy	Hamming	$\mathcal{V}$	70.81	30.43	94.26	73.29	32.81	101.58
Seq, lazy	Hamming	$\mathcal{V}_{batch}$	71.85	31.13	**96.65**	73.43	32.95	**102.03**
Seq, lazy	Hamming	$\mathcal{V}_{refs}$	**71.96**	**31.23**	95.34	**73.53**	**33.09**	101.89
Seq	CIDEr	$\mathcal{V}$	71.05	30.46	94.40	73.08	32.51	101.84
Seq	CIDEr	$\mathcal{V}_{batch}$	71.51	31.17	95.78	**73.50**	**33.04**	**102.98**
Seq	CIDEr	$\mathcal{V}_{refs}$	**71.93**	**31.41**	**96.81**	73.42	32.91	102.23
Seq, lazy	CIDEr	$\mathcal{V}$	71.43	**31.18**	**96.32**	73.55	**33.19**	**102.94**
Seq, lazy	CIDEr	$\mathcal{V}_{batch}$	71.47	31.00	95.56	73.18	32.60	101.30
Seq, lazy	CIDEr	$\mathcal{V}_{refs}$	**71.82**	31.06	95.66	**73.92**	33.10	102.64
Tok-Seq	Hamming	$\mathcal{V}$	70.79	30.43	96.34	73.68	32.87	101.11
Tok-Seq	Hamming	$\mathcal{V}_{batch}$	72.28	31.65	96.73	73.86	33.32	102.90
Tok-Seq	Hamming	$\mathcal{V}_{refs}$	72.69	32.30	98.01	73.56	33.00	101.72
Tok-Seq	CIDEr	$\mathcal{V}$	70.80	30.55	96.89	73.31	32.40	100.33
Tok-Seq	CIDEr	$\mathcal{V}_{batch}$	72.13	31.71	96.92	73.61	32.67	101.41
Tok-Seq	CIDEr	$\mathcal{V}_{refs}$	**73.08**	**32.82**	**99.92**	**74.28**	**33.34**	**103.81**

Table E.1: MS-COCO 's test set evaluation measures.

can be used with different types of decoders. In our experiments we use recurrent language models.

E.3.1 Image Captioning

E.3.1.1 Experimental setup.

We use the MS-COCO dataset (Lin et al., 2014), which consists of 82k training images each annotated with five captions. We use the standard splits of Karpathy and Fei-Fei (2015), with 5k images for validation, and 5k for test. The test set results are generated via beam search (beam size 3) and are evaluated with the MS-COCO captioning evaluation tool. We

	BLEU-1		BLEU-4		METEOR		ROUGE-L		CIDEr	
	c5	c40	c5	c40	c5	c40	c5	c40	c5	c40
Vinyals et al. (2015) +	71.3	89.5	30.9	58.7	25.4	34.6	53.0	68.2	94.3	94.6
Xu et al. (2015)	70.5	88.1	27.7	53.7	24.1	32.2	51.6	65.4	86.5	89.3
You et al. (2016) +	73.1	90.0	31.6	59.9	25.0	33.5	53.5	68.2	94.3	95.8
Yang et al. (2016) +	72.0	90.0	31.3	59.7	25.6	34.7	53.3	68.6	96.5	96.9
Lu et al. (2017) +	74.8	92.0	33.6	63.7	26.4	35.9	55.0	70.5	104.2	105.9
Rennie et al. (2017) +†	78.1	93.7	35.2	64.5	27.0	35.5	56.3	70.7	114.7	116.7
Yao et al. (2017b) +†∘	78.7	93.7	35.6	65.2	27.0	35.4	56.4	70.5	116	118
Anderson et al. (2018) +†∘	80.2	95.2	36.9	68.5	27.6	36.7	57.1	72.4	117.9	120.5
Ours: Tok-Seq CIDEr	72.6	89.7	30.2	58.3	25.5	34.0	53.5	68.0	96.4	99.4
Ours: Tok-Seq CIDEr +	74.9	92.4	34.3	64.7	26.5	36.1	55.2	71.1	103.9	104.2

Table E.2: MS-COCO 's server evaluation . (+) for ensemble submissions, (†) for submissions with CIDEr optimization and (∘) for models using additional data.

report CIDEr and BLEU scores on this internal test set. We also report results obtained on the official MS-COCO server that additionally measures METEOR (Denkowski and Lavie, 2014) and ROUGE-L (Lin, 2004).

We experiment with both non-attentive LSTMs (Vinyals et al., 2015) and the ResNet baseline of the state-of-the-art top-down attention (Anderson et al., 2018). The MS-COCO vocabulary consists of 9,800 words that occur at least 5 times in the training set.

E.3.1.2 Results and discussion

Restricted vocabulary sampling. In this section, we evaluate the impact of the vocabulary subset from which we sample the modified sentences for sequence-level smoothing. We experiment with two rewards: CIDEr , which scores w.r.t. all five available reference sentences, and Hamming distance reward taking only a single reference into account. For each reward we train our (Seq) models with each of the three subsets previously described $\mathcal{V}$, $\mathcal{V}_{\text{refs}}$ and $\mathcal{V}_{\text{batch}}$.

From the results in Table E.1 we note that for the inattentive models, sampling from $\mathcal{V}_{\text{refs}}$ or $\mathcal{V}_{\text{batch}}$ has a better performance than sampling from the full vocabulary on all metrics. In fact, using these subsets introduces a useful bias to the model and improves performance.

This improvement is most notable using the CIDEr reward that scores candidate sequences w.r.t. to multiple references, which stabilizes the scoring of the candidates.

With an attentive decoder, no matter the reward, re-sampling sentences with words from $\mathcal{V}_{\text{ref}}$ rather than the full vocabulary $\mathcal{V}$ is better for both reward functions, and all metrics.

Lazy training. From the results of Table E.1, we see that lazy sequence-level smoothing is competitive with exact non-lazy sequence-level smoothing, while requiring roughly equivalent training time as MLE.

Overall. For reference, we include in Table E.1 baseline results obtained using MLE, and our implementation of MLE with entropy regularization (MLE+γH) (Pereyra et al., 2017), as well as the RAML approach of Norouzi et al. (2016) which corresponds to sequence-level smoothing based on the Hamming reward and sampling replacements from the full vocabulary (Seq, Hamming, $\mathcal{V}$)

We observe that entropy smoothing is not able to improve performance much over MLE for the model without attention, and even deteriorates for the attention model. We improve upon RAML by choosing an adequate subset of vocabulary for substitutions. We also report the performances of token-level smoothing, where the promotion of rare tokens boosted the scores in both attentive and non-attentive models. For sequence-level smoothing, choosing a task-relevant reward with importance sampling yielded better results than plain Hamming distance. Moreover, we used the two smoothing schemes (Tok-Seq) and achieved the best results with CIDEr as a reward for sequence-level smoothing combined with a token-level smoothing that promotes rare tokens improving CIDEr from 93.59 (MLE) to 99.92 for the model without attention, and improving from 101.63 to 103.81 with attention.

Qualitative results. In Figure E.1 we showcase captions obtained with MLE and our three variants of smoothing *i.e.* token-level (Tok), sequence-level (Seq) and the combination (Tok-Seq). We note that the sequence-level smoothing tend to generate lengthy captions overall, which is maintained in the combination. On the other hand, the token-level smoothing allows for a better recognition of objects in the image that stems from the robust training of the classifier *e.g.* the carrots in the second example or the 'cement block' in the third one.

Ground truth:
1) an airplane on the tarmac and the glass passageway leading to its door
2) a commercial airplane hooked up to a gate at the airport
3) a jumbo jet plane with a boarding walkway attached to it
4) an airplane sitting at the gate of an airport
5) a large plain is at a airport terminal

Generated:
- Baseline: a plane is parked on the side of the road
- Seq: an airplane is parked on the runway at the airport
- Tok: an airplane sitting on the tarmac at an airport
- Tok-Seq: a large airplane sitting on top of an airport tarmac

Ground truth:
1) carrots, grapes, cheese, bread and butter in a plastic container
2) some different foods are laying out on a table
3) a tray with bread, cheese, fruits and vegetables
4) the lunch provides all the major food groups
5) a trey with some fruit inside of it

Generated:
- Baseline: a tray with a variety of foods in it
- Seq: a tray with a variety of fruits and vegetables
- Tok: a lunch box with carrots carrots and carrots
- Tok-Seq: a tray of food that includes a sandwich and some vegetables

Ground truth:
1) a child on a skateboard is performing a trick on a slab
2) a man that is standing on a skateboard on wood
3) a kid riding a skateboard across a couple of blocks
4) a person on a skateboard on a wooden rail
5) a kid that is skateboarding on some wood block

Generated:
- Baseline: a man riding a skateboard down a sidewalk
- Seq: a man riding a skateboard down a sidewalk
- Tok: a man riding a skateboard on top o a cement block
- Tok-Seq: a man riding a skateboard down a wooden bench

Figure E.1: Examples of generated captions with the baseline MLE and our models with attention.

Comparison to the state of the art. We compare our model to state-of-the-art systems on the MS-COCO evaluation server in Table E.2. We submitted a single model (Tok-Seq, CIDEr , $\mathcal{V}_{\text{refs}}$) as well as an ensemble of five models with different initializations trained on the training set plus 35k images from the dev set (a total of 117k images) to the MS-COCO server. The three best results on the server (Rennie et al., 2017; Yao et al., 2017b; Anderson et al., 2018) are trained in two stages where they first train using MLE, before switching to policy gradient methods based on CIDEr. Anderson et al. (2018) reported an increase of 5.8% of CIDEr on the test split after the CIDEr optimization. Moreover, Yao et al. (2017b) uses additional information about image regions to train the attributes classifiers, while Anderson et al. (2018) pre-trains its bottom-up attention model on the Visual Genome dataset (Krishna et al., 2017). Lu et al. (2017); Yao et al. (2017b) use

Loss	Reward	$\mathcal{V}_{sub}$	WMT'14	IWSLT'14
MLE	-	-	30.03	27.55
tok	Glove sim	-	30.16	27.69
tok	Glove sim r^{freq}	-	30.19	27.83
Seq	Hamming	$\mathcal{V}$	30.85	27.98
Seq	Hamming	$\mathcal{V}_{batch}$	31.18	28.54
Seq	BLEU-4	$\mathcal{V}_{batch}$	31.29	28.56
Tok-Seq	Hamming	$\mathcal{V}_{batch}$	31.36	28.70
Tok-Seq	BLEU-4	$\mathcal{V}_{batch}$	31.39	28.74

Table E.3: Tokenized BLEU score on WMT'14 En-Fr evaluated on the news-test-2014 set and tokenized case-insensitive BLEU on IWSLT'14 De-En.

the same CNN encoder as ours (ResNet-152), (Vinyals et al., 2015; Yang et al., 2016) use Inception-v3 (Szegedy et al., 2016) for image encoding and Rennie et al. (2017); Anderson et al. (2018) use Resnet-101, both of which have similar performances to ResNet-152 on ImageNet classification (Canziani et al., 2016).

E.3.2 Machine Translation

E.3.2.1 Experimental setup.

For this task we validate the effectiveness of our approaches on two different datasets. The first is WMT'14 English to French, in its filtered version, with 12M sentence pairs obtained after dynamically selecting a "clean" subset of 348M words out of the original "noisy" 850M words (Bahdanau et al., 2015; Cho et al., 2014; Sutskever et al., 2014). The second benchmark is IWSLT'14 German to English consisting of around 150k pairs for training. In all our experiments we use the attentive model of (Bahdanau et al., 2015) To assess the translation quality we report the BLEU-4 metric.

Source	I think it's conceivable that these data are used for mutual benefit.
Target	J'estime qu'il est concevable que ces données soient utilisées dans leur intérêt mutuel.
MLE	Je pense qu'il est possible que ces données soient utilisées à des fins réciproques.
Tok-Seq	Je pense qu'il est possible que ces données soient utilisées pour le bénéfice mutuel.
Source	However, given the ease with which their behaviour can be recorded, it will probably not be long before we understand why their tails sometimes go one way, sometimes the other.
Target	Toutefois, étant donné la facilité avec laquelle leurs comportements peuvent être enregistrés, il ne faudra sûrement pas longtemps avant que nous comprenions pourquoi leur queue bouge parfois d'un côté et parfois de l'autre.
MLE	Cependant, compte tenu de la facilité avec laquelle on peut enregistrer leur comportement, il ne sera probablement pas temps de comprendre pourquoi leurs contemporains vont parfois une façon, parfois l'autre.
Tok-Seq	Cependant, compte tenu de la facilité avec laquelle leur comportement peut être enregistré, il ne sera probablement pas long avant que nous ne comprenons la raison pour laquelle il arrive parfois que leurs agresseurs suivent un chemin, parfois l'autre.
Source	The public will be able to enjoy the technical prowess of young skaters, some of whom, like Hyeres' young star, Lorenzo Palumbo, have already taken part in top-notch competitions.
Target	Le public pourra admirer les prouesses techniques de jeunes qui, pour certains, fréquentent déjà les compétitions au plus haut niveau, à l'instar du jeune prodige hyérois Lorenzo Palumbo.
MLE	Le public sera en mesure de profiter des connaissances techniques des jeunes garçons, dont certains, à l'instar de la jeune star américaine, Lorenzo, ont déjà participé à des compétitions de compétition.
Tok-Seq	Le public sera en mesure de profiter de la finesse technique des jeunes musiciens, dont certains, comme la jeune star de l'entreprise, Lorenzo, ont déjà pris part à des compétitions de gymnastique.

Table E.4: WMT'14 En→Fr examples

E.3.2.2 Results and analysis

We present our results in Table E.3. On both benchmarks, we improve on both MLE and RAML approach of Norouzi et al. (2016) (Seq, Hamming, $\mathcal{V}$): using the smaller batch-vocabulary for replacement improves results, and using importance sampling based on BLEU-4 further boosts results. In this case, unlike in the captioning experiment, token-level smoothing brings smaller improvements. The combination of both smoothing approaches gives best results, similar to what was observed for image captioning, improving the MLE BLEU-4 from 30.03 to 31.39 on WMT'14 and from 27.55 to 28.74 on IWSLT'14. The outputs of our best model are compared to the MLE in some examples showcased in Figure E.4.

E.4 Related Work

Previous work aiming to improve the generalization performance of conventional language models can be roughly divided into three categories: those based on regularization, data augmentation, and alternatives to maximum likelihood estimation.

Regularization techniques are used to increase the smoothness of the function learned by

the network, *e.g.* by imposing an ℓ_2 penalty on the network weights, also known as weight decay. More recent approaches mask network activations during training, as in dropout (Srivastava et al., 2014) and its variants adapted to recurrent models (Pham et al., 2014; Krueger et al., 2017). Instead of masking, batch-normalization (Ioffe and Szegedy, 2015) rescales the network activations to avoid saturating the network's non-linearities. Instead of regularizing the network parameters or activations, it is also possible to directly regularize based on the entropy of the output distribution (Pereyra et al., 2017).

Data augmentation techniques improve the robustness of the learned models by applying transformations that might be encountered at test time to the training data. In computer vision, this is common practice, and implemented by, *e.g.*, scaling, cropping, and rotating training images (LeCun et al., 1998; Krizhevsky et al., 2012; Paulin et al., 2014). In natural language processing, examples of data augmentation include input noising by randomly dropping some input tokens (Iyyer et al., 2015; Bowman et al., 2016; Kumar et al., 2016), and randomly replacing words with substitutes sampled from the model (Bengio et al., 2015). Xie et al. (2017) introduced data augmentation schemes for RNN language models that leverage n-gram statistics in order to mimic Kneser-Ney smoothing of n-grams models. In the context of machine translation, Fadaee et al. (2017) modify sentences by replacing words with rare ones when this is plausible according to a pre-trained language model, and substitutes its equivalent in the target sentence using automatic word alignments. This approach, however, relies on the availability of additional monolingual data for language model training.

The *de facto* standard way to train RNN language models is maximum likelihood estimation (MLE) (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015). The sequential factorization of the sequence likelihood generates an additive structure in the loss, with one term corresponding to the prediction of each output token given the input and the preceding ground-truth output tokens. In order to directly optimize for sequence-level structured loss functions, such as measures based on n-grams like Bleu or Cider, Ranzato et al. (2016) use reinforcement learning techniques that optimize the expectation of a sequence-level reward. In order to avoid early convergence to poor local optima, they pre-train the model using MLE.

Leblond et al. (2018) build on the learning to search approach to structured prediction

(Daumé et al., 2009; Chang et al., 2015) and adapts it to RNN training. The model generates candidate sequences at each time-step using all possible tokens, and scores these at sequence-level to derive a training signal for each time step. This leads to an approach that is structurally close to MLE, but computationally expensive.

Norouzi et al. (2016) introduce a reward augmented maximum likelihood (RAML) approach, that incorporates a notion of sequence-level reward without facing the difficulties of reinforcement learning. They define a target distribution over output sentences using a soft-max over the reward over all possible outputs. Then, they minimize the KL divergence between the target distribution and the model's output distribution. Training with a general reward distribution is similar to MLE training, except that we use multiple sentences sampled from the target distribution instead of only the ground-truth sentences.

In our work, we build upon the work of Norouzi et al. (2016) by proposing improvements to sequence-level smoothing, and extending it to token-level smoothing. Our token-level smoothing approach is related to the label smoothing approach of Szegedy et al. (2016) for image classification. Instead of maximizing the probability of the correct class, they train the model to predict the correct class with a large probability and all other classes with a small uniform probability. This regularizes the model by preventing over-confident predictions. In natural language generation with large vocabularies, preventing such *narrow* over-confident distributions is imperative, since for many tokens there are nearly interchangeable alternatives.

E.5 Conclusion

We investigated the use of loss smoothing approaches to improve over maximum likelihood estimation of RNN language models. We generalized the sequence-level smoothing RAML approach of Norouzi et al. (2016) to the token-level by smoothing the ground-truth target across semantically similar tokens. For the sequence-level, which is computationally expensive, we introduced an efficient lazy evaluation scheme, and introduced an improved re-sampling strategy. Experimental evaluation on image captioning and machine translation demonstrates the complementarity of sequence-level and token-level loss smoothing, improving over both the maximum likelihood and RAML.