Improving Cross-Lingual Representations by Distilling Multilingual Encoders into Monolingual Components

Master Thesis

presented by
Minh Duc Bui
Matriculation Number 1557406

submitted to the
Data and Web Science Group
Prof. Dr. Simone Paolo Ponzetto
University of Mannheim

Mai 2022

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Natural language processing (NLP) has made significant progress in recent years, achieving impressive performances across diverse tasks. However, these advances are focused on just a tiny fraction of the 7000 languages in the world, e.g., English, where sufficient amounts of text in the respective language are available (high-resource languages). Nevertheless, when the situation arises where text data in a language is scarce, language technologies fail. These languages are called low-resource languages, e.g., Swahili, Basque, or Urdu; see Figure 1.1 for a more apparent distinction between high-, mid-, and low-resource languages. This leaves low-resource languages and, therefore, most languages understudied, which further increases the digital language divide[1] on a technological level. Being able to develop technologies for low-resource languages is vital for scientific, social, and economic reasons, e.g., Africa and India are the hosts of around 2000 low-resource languages and are home to more than 2.5 billion inhabitants [Magueresse et al., 2020]. "Opening" the newest NLP technologies for low-resource languages can help bridge the gap, e.g., digital assistants, or help reduce the discrimination against speakers of non-English languages [Tatman, 2017, Rabinovich et al., 2018, Zhiltsova et al., 2019].

To improve language technologies for low-resource languages, the field of Cross-Lingual Representation Learning is focused on creating high-quality representations for these languages by gaining benefit from abundant data in another language via a shared representation space. As static word representations gained in popularity, many multilingual embedding methods have been presented [Mikolov et al., 2013b, Hermann and Blunsom, 2014, Hu et al., 2020]. The idea behind these

[1] http://labs.theguardian.com/digital-language-divide/

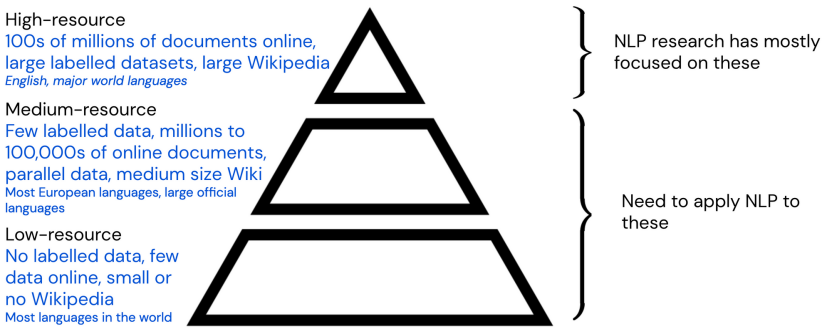


Figure 1.1: A conceptual view of the NLP resource hierarchy categorised in availability of task-specific labels and availability of unlabeled language-specific text. Taken from [Ruder et al., 2019].

methods is to induce embeddings[2] such that the embeddings for two languages are aligned, i.e., word translations, e.g., *cat* and *Katze*, have similar representations. Recently, however, large pre-trained language models, the so-called transformer models, took static word embedding methods over in virtually every aspect, partly because these models induce context-dependent word representations, capturing the rich meaning of a word better [Peters et al., 2018, Howard and Ruder, 2018, Radford and Narasimhan, 2018, Devlin et al., 2019]. E.g., mBERT and XLM-R, transformer-based multilingual masked language models pre-trained on text in (approximately) 100 languages, can obtain impressive performances for a variety of cross-lingual transfer tasks [Pires et al., 2019, Conneau et al., 2020]. Even though these models were not trained with any cross-lingual objectives, they still produce representations that can generalize well across languages for a wide range of downstream tasks [Wu and Dredze, 2019, Conneau et al., 2020]. To analyze the cross-lingual transfer ability of multilingual models, the model is first fine-tuned on annotated data of a downstream task and then evaluated in the zero or few-shot scenario, i.e., evaluated with the fine-tuned models in the target language [Hu et al., 2020] with no or few additional labeled target language data.

As impressive as these multilingual transformers might seem, low-resource languages still perform sub-par to high-resource languages [Wu and Dredze, 2019, Conneau et al., 2020], partly due to the fact of a smaller pre-training corpus [Conneau et al., 2020], the curse of multilinguality [Conneau et al., 2020] and the importance of vocabulary curation and size [Chung et al., 2020, Artetxe et al., 2020]. E.g., the curse of multilinguality argues assuming that the model capacity stays

[2]Word embeddings and word representation are interchangeable in our thesis.

constant that adding more languages leads to better cross-lingual performance on low-resource languages up until a point where the overall performance on monolingual and cross-lingual benchmarks degrades. Intuitively explained, adding more languages to the model has two effects: (1) Positive cross-lingual transfer, especially for low-resource languages, and (2) lower per-language capacity, which then, in turn, can degrade the overall model performance. These two effects of *capacity dilution* and positive transfer need to be carefully traded against each other. The model either needs to have a large model capacity[3] or is specialized (constrained) towards a subset of languages beforehand. For these reasons, it is hard to create a single model that can effectively represent a diverse set of languages. One solution is to create language-specific models (monolingual models) with language-specific vocabulary and model parameters [Virtanen et al., 2019, Antoun et al., 2020], but in return, monolingual models need enough text to pre-train the model on the language modeling task, which is typically not available for low-resource languages. Additionally, we can not benefit from any cross-lingual transfer from related languages, making it harder to create an adequate representation for low-resource languages [Pires et al., 2019, Lauscher et al., 2020].

1.2 Research Objective & Contribution

In this thesis, we explore how one can alleviate the issues of big multilingual transformers for low-resource languages, especially the curse of multilinguality, see previous Section 1.1. Specifically, our two main objectives are: (1) Improving the cross-lingual alignment for low-resource languages and (2) improving cross-lingual downstream task performance for low-resource languages. We utilize Knowledge Distillation (KD) by distilling the multilingual model into language-specialized (also called monolingual) language models. We make the following contributions:

- We propose a novel setup to distill multilingual transformers into monolingual components. Based on the setup, we propose two KD strategies: One for improving the alignment between two languages and one to improve cross-lingual downstream task performance. We call the former `MonoAlignment` and the latter `MonoShot`.

- `MonoAlignment` uses a distillation strategy to distill multilingual transformer models into smaller monolingual components which have an improved aligned representation space between a high-resource language and a low-resource language. We demonstrate the effectiveness by distilling XLM-

[3] Here: Measured in the number of free parameters in the model.

R and experimenting with aligning English with Turkish, Swahili, Urdu, and Basque.

- We compare `MonoAlignment` to other Knowledge Distillation strategies showing that it outperforms them in the retrieval task for low-resource languages.
- Our work suggests that an increase in the cross-lingual alignment of a multilingual transformer model does not necessarily translate into an increase in cross-lingual downstream task performance.
- Therefore, we propose `MonoShot`, another Knowledge Distillation strategy to distill multilingual transformer models into smaller monolingual components but which have a strong cross-lingual downstream performance in the zero- and few-shot settings.
- We show that `MonoShot` performs best among many different Knowledge Distillation strategies, albeit still lacks behind the teacher performance. However, it outperforms models built upon the teacher architecture but is trimmed down to the same size as the distilled components and initialized from parts of the teacher.
- We demonstrate an effective fine-tuning strategy for the zero-shot scenario for aligned monolingual models and compare it against many other strategies.

To conduct our research, we will draw inspiration from the field of Cross-Lingual Representation Learning, Knowledge Distillation, and Parameter-Efficient Fine-tuning. Following different Knowledge Distillation strategies, such as from DistilBert [Sanh et al., 2020] or TinyBert [Jiao et al., 2020], we distill the aligned cross-lingual representation space of the multilingual transformer model XLM-R [Conneau et al., 2017] into smaller monolingual students. To fine-tune aligned monolingual models in a zero-shot scenario, we study the field of parameter-efficient fine-tuning, i.e., Adapters [Houlsby et al., 2019, Pfeiffer et al., 2021], BitFit [Zaken et al., 2021] and Sparse Fine-Tuning [Guo et al., 2020]. Finally, we evaluate the general-purpose cross-lingual representation of our monolingual models in the retrieval, classification, structured prediction, and question-answering task.

1.3 Thesis Structure

The thesis is divided into five chapters: Chapter 2 provides the preliminaries and our review of related literature. Specifically, we explore the history of Cross-

Lingual Representation Learning, study current state-of-the-art methods and their limitations. Next, we give an overview of Knowledge Distillation methods in NLP and structure relevant methods by categorizing them into which parts of the model architecture they distill from. Since we introduce a novel distillation setup to induce aligned monolingual students, we explore related distillation setups to create multilingual students. Finally, we briefly explore parameter-efficient fine-tuning methods. In Chapter 3, we motivate and explain our proposed monolingual setup, detail our training procedure, and outline different KD losses and zero-shot fine-tuning approaches for our monolingual setup. We then explain which loss and zero-shot fine-tuning approach our final proposed methods `MonoAlignment` and `MonoShot` utilize. In Chapter 4, we provide experimental results of our approaches and compare them to relevant baselines. Subsequently, we analyze different aspects of our approaches based on further experiments. Finally, Chapter 5 concludes our work and discusses various future works.

Chapter 2

Preliminaries & Related Work

In this chapter, we will motivate and explore the history & current state-of-the-art as well as challenges of Cross-Lingual Representation Learning (Section 2.1). Additionally, we discuss the origin of Knowledge Distillation and the recent developments in the context of NLP (Section 2.2). As we will utilize techniques from the field of parameter-efficient fine-tuning to later fine-tune our monolingual students on a downstream task, we will briefly discuss some relevant methods in Section 2.3.

In Section 2.1.1, we explain why Cross-Lingual Representation Learning is needed in today's age and motivate why we can leverage shared knowledge across languages. As word representations are the basis of Cross-Lingual Representation Learning and modern NLP, we explore the concept of word representations (Section 2.1.2), give a brief history of architectures to construct these representations (Section 2.1.3), discuss cross-lingual word representations (Section 2.1.4 and 2.1.5) and analyze current challenges in multilingual transformers (Section 2.1.6), the state of the art in cross-lingual representations. In the next Section 2.2 we discuss various aspects of Knowledge Distillation, specifically in Section 2.2.1 we discuss why Knowledge Distillation is important and how the "vanilla" Knowledge Distillation setup is constructed. We then discuss Knowledge Distillation in the context of NLP (Section 2.2.2) and give an overview of recent relevant approaches by categorizing them into which parts of the transformer they distill from (Section 2.2.3). As our thesis utilizes a novel distillation setup to induce aligned monolingual students, we analyze different distillation setups to induce multilingual students in Section 2.2.4. Subsequently, we discuss challenges in KD in Section 2.2.5. In the final Section 2.3, we will discuss parameter-efficient fine-tuning techniques such as Adapters (Section 2.3.1) and Sparse-Fine-Tuning (Section 2.3.2), specifically Diff-Pruning and BitFit.

2.1 Cross-Lingual Representation Learning

In this section, we will explore Cross-Lingual Representation Learning by first motivating the approach (Section 2.1.1) and exploring the concept of word representations (Section 2.1.2). As modern NLP and Cross-Lingual Representation Learning are based on word representations, we thoroughly explore the history of architectures to construct these (Section 2.1.3). Finally, we briefly outline static cross-lingual word representation approaches (Section 2.1.4) and discuss contextualized cross-lingual word representation approaches more in-depth as they currently are the state-of-the-art and the basis of this thesis (Section 2.1.5).

2.1.1 Motivation

The latest NLP technology relies on pre-training on massive amounts of text in the respective language in an unsupervised fashion, producing fixed-size sequence or word representations that can be used to fine-tune on a task with sufficient labeled data. In most cases, both data sources are needed to meet the performance of state-of-the-art NLP approaches. However, the lack of both data sources will highly degrade the performance of these methods, posing a fundamental problem in scaling low-resource languages. Cross-lingual representation techniques try to alleviate the issue of data scarcity for low-resource languages by inducing an aligned representation across languages, i.e., language-agnostic language representations. The idea is to transfer lexical, syntactic, and semantic knowledge across languages that can be used for cross-lingual downstream tasks. This gives rise to two advantages: (1) Transferring lexical knowledge across languages enables us to reason about the semantics of words in multilingual contexts and is a vital source of knowledge for multilingual systems such as machine translation [Artetxe et al., 2017, Qi et al., 2018, Lample et al., 2018], multilingual search, and question answering [Vulić and Moens, 2015]. (2) More importantly, given a downstream task, models can utilize the joint representation space by training on a high-resource language such as English, where labeled data exists, to then use the acquired task knowledge to transfer it to other (low-resource) languages. The hope is that the model can generalize lexical properties and relations across languages [Plath, 2002]. Ultimately, cross-Lingual representations can also be seen as a type of transfer learning which can help us understand why transferring knowledge across languages works.

Relation to Transfer Learning. Transfer Learning is a sub-field in Machine Learning that focuses on reusing the acquired knowledge from past related tasks to help the learning process of solving a new task. Cross-Lingual Representation Learning, therefore, is a type of transfer learning, specifically similar to domain adaption; see Figure 2.1 for a taxonomy of transfer learning in NLP. Viewing

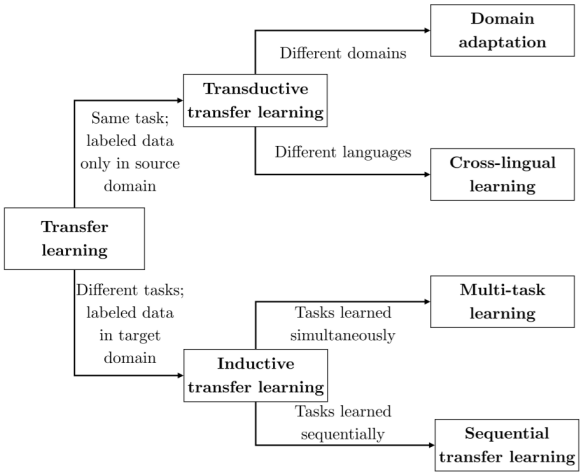


Figure 2.1: A taxonomy for transfer learning for NLP. Given a downstream task, Cross-Lingual Representation Learning utilizes the joint representation space by fine-tuning on a high-resource language to then use the acquired task knowledge to transfer it to other (low-resource) languages. Adopted from [Ruder, 2019].

cross-lingual representation as a form of transfer learning can help us understand in which cases knowledge from one to another language can be transferred:

1. Transfer Learning works well when the *underlying structures are similar to each other*. Languages share many aspects on many different levels, e.g., on a lexical level, languages can incorporate words from another language (loanwords) and have words from the same origin (cognate). On a syntactical level, languages might have a similar structure of sentences, and on a semantic level languages, languages are built upon a so-called natural semantic metalanguage, see Goddard [2006] for a more in-depth analysis.

2. Transfer Learning fails when the source and target settings are vastly different. In our cross-lingual setting, transferring any knowledge from languages that are not related in any way is hard, i.e., languages that are not typologically or etymologically related.

In summary, languages must be in some way related; otherwise, we can not transfer knowledge across languages. Languages in the same language family share much more than unrelated languages. This is also reflected in the performance of cross-lingual methods [Lample and Conneau, 2019, Lauscher et al., 2020]. Furthermore, even when languages come from different language families, Goddard

[2006] argues that, on a semantic level, languages are built upon a natural semantic metalanguage, therefore, share a connection.

2.1.2 Constructing Word Representations

The question remains how one can exploit shared structures across languages to build a cross-lingual representation. As word representations, i.e., words represented as real-valued vectors, are the basis of modern NLP and cross-lingual representations, we first discuss different approaches to built word representations.

Bag-of-Words. The simplest solution to represent words in a vector form is to use the bag-of-words approach, which describes the occurrence of words within a document. However, the bag-of-word approach has major problems such as losing information about word order and semantics, being highly dimensional (curse of dimensionality), and can not represent out-of-vocabulary tokens.

Distributed Word Representation. To improve upon bag-of-words, distributed word representations were utilized, which represent words (or more generally, tokens[1]) as distributed representations of lower dimensionality, trained to capture syntactic and semantic relationships between words. The approach is motivated by the underlying idea that "a word is characterized by the company it keeps", called the *distributional hypothesis* [Harris, 1954]. This means that words that occur in the same context are semantically related. The general approach to generating distributed word embeddings is by computing word co-occurrence statistics based on unlabeled free text (unsupervised).

Language Models. The most prominent way to make use of the distributional hypothesis to create distributed word representations is by using language models (LMs), which is *the* backbone of modern NLP. LMs first gained momentum when Collobert and Weston [2008] showed the effectiveness of applying neural models to language modeling to create semantic word embeddings for downstream tasks. It was the start point of the modern approach to NLP: Pre-train neural models to create word representations for downstream tasks. Formally, an LM, given a sequence of tokens $\{t_1, ..., t_m\}$ with length m, outputs a probability distribution over the sequence of tokens, i.e., the likelihood of the tokens occurring jointly:

[1] A tokenizer splits the text into smaller units called tokens. Tokens can be words, characters, or subwords. In our thesis, we mostly use the term word representations to illustrate concepts better. Only when necessary do we explicitly state tokens. Nevertheless, all presented approaches can be generalized to any token-level.

$$P(t_1, ..., t_m) \overset{\text{(chain rule)}}{=} P(t_1) \cdot P(t_2|t_1) \cdot ... \cdot P(t_m|t_1, ..., t_{m-1}) = \prod_{i=1}^{m} P(t_i|t_{i:i-1}).$$

The chain rule allows us to calculate the joint probability of the tokens in the sequence as conditional probabilities of a token given their respective previous tokens. Even then, this becomes quickly intractable due to combinatorial explosion of the possible number of previous tokens. To avoid this problem, typically we leverage the Markov assumption, i.e., it is assumed that the probability of $P(t_m|t_1, ..., t_{m-1})$ depends only on the previous $n - 1 << m$ tokens:

$$P(t_m|t_1, ..., t_{m-1}) \approx P(t_m|t_{m-(n-1)}, ..., t_{m-1})$$
$$\implies P(t_1, ..., t_m) \approx \prod_{i=1}^{m} P(t_i|t_{i-(n-1):i-1}).$$

As the joint probability of tokens now only depend on the product of probabilities of the form

$$P(t_i|t_{i-(n-1)}, ..., t_{i-1}),$$

called n-grams, we need to estimate the n-grams which can be done with the maximum likelihood estimate (MLE) on the training corpus. In practice, models are trained to either predict the subsequent tokens (directional) or to predict the missing token given the surrounding context of the word (bidirectional).

To evaluate the performance of a language model, the usual metric is the *Perplexity* [Jelinek et al., 1977], which is defined as the inverse probability of sequences on a validation set normalized by the number of tokens:

$$PP(t_1, ..., t_{m-1}) = \sqrt[m]{\frac{1}{P(t_1, t_2, ..., t_m)}} \overset{\text{(chain rule)}}{=} \sqrt[m]{\frac{1}{\prod_{i=1}^{m} P(t_i|t_{i:i-1})}}.$$

A lower perplexity indicates a better model.

The following subsection will explore different model architectures that utilize language modeling to create powerful word embeddings.

2.1.3 Evolution of Architectures

We outline the evolution of (neural) architectures in NLP to induce strong word representations that can be utilized for downstream tasks, such as Natural Language

Understanding. For now, we restrict ourselves to inducing monolingual word representations using the language modeling task on *monolingual* text corpus.

Feedforward Neural Networks. Mikolov et al. [2013a,b] introduce an efficient way of learning high-quality word vectors for millions of words in the vocabulary from a large amount of unlabeled text data in an unsupervised fashion. The released word embeddings not only capture semantic and syntactic information but also learn relationships between words[2], e.g., *Paris - France + Italy = Rome*. They dub their approach word2vec and give two novel model architectures: Skip-Gram (SG) and Continuous Bag-of-Words (CBOW).

Both CBOW and SG architectures are based on a simple feedforward neural network. The CBOW method computes the probability of the current token based on the context tokens within a window W of k neighboring tokens. On the other hand, the SG computes the probability of surrounding tokens within a window W of k neighboring tokens given the current token. The network encodes each token t_i into a center token e_i and context token c_i which correspond to the $i-$th row of the center token matrix $E^{|V|\times d}$ and context token matrix $E^{|V|\times d}$, where the V is the size of the vocabulary and d the token vector embedding size. Given a center token t, SG estimates the likelihood of seeing a context token w conditioned on the given center token with the softmax function:

$$P(c_w|e_t) = \frac{\exp e_t^T c_w}{\sum_{i=1}^{N} \exp e_t^T c_i}, \tag{2.1}$$

where e_t denote the embedding for the center token t and c_w the embedding for the context token w in the window $c_w \in W$. Given a text corpus $\{t_1, ..., t_T\}$ of length T and assuming that the context words are independently generated given any center word, we learn the model parameters (token embeddings) by maximizing the likelihood function over the text corpus which is eqivalent to minimizing the negative log-likelihood:

$$\max_{e,c} \prod_{t=1}^{T} \prod_{w\in W_t} P(c_w|e_t) \Leftrightarrow \min_{e,c} -\sum_{t=1}^{T} \sum_{w\in W_t} \log\left(P(c_w|e_t)\right). \tag{2.2}$$

For a downstream task, the final embedding for token t is either the center token or calculated as the element-wise average or the sum of its center and context representations. Therefore, the word2vec objective in Equation (2.2) directly uses the language modeling task to generate effective word embeddings.

[2]Relationships are defined by subtracting two words vectors, and the result is added to another word.

Even though word2vec is very effective in creating powerful word representations, there are some considerable drawbacks: First, the denominator in Equation (2.1) sums over the entire vocabulary, slowing down the calculating of the softmax. There are some approaches such as hierarchical softmax and negative sampling to overcome this [Mikolov et al., 2013b]. Still, there are two major conceptually disadvantages of word2vec representations: First, they can not embed any tokens outside the vocabulary, and second, they do not account for the linguistic morphology of a word, e.g., the representations of "eat" and "eaten" are learned separately (no parameter-sharing) based on its context they appear on.

To solve the above issues, Bojanowski et al. [2016] introduce FastText, a new embedding method. FastText extends the idea of word2vec by using the internal structure of a word to improve the word representations of word2vec. Instead of constructing representations for words, FastText learns representations for n-grams of characters which are then used to build word representations by summing the bag-of-character n-grams up. E.g., for $n = 3$, the word *"artificial"* is represented by *<ar, art, rti, tif, ifi, fic, ici, ial, al>* where the angular brackets indicate the beginning and end of the word. This allows us to better represent and capture the meaning of suffixes and prefixes. Furthermore, words that do not appear during training can be represented by breaking the word into n-grams to get its representation.

The released representations of FastText and word2vec became famous because of their ease of use and effectiveness in a variety of NLP problems [Lample et al., 2016, Kiros et al., 2015, Kusner et al., 2015]. Furthermore, these (monolingual) representations can be used to construct a cross-lingual representation space by mapping representations of multiple languages into a shared space (see Section 2.1.4).

However, word2vec and FastText have several drawbacks: Each word has a static word representation. Consequently, both methods can not correctly capture phrases and polysemy of words. Furthermore, during training, we only consider the context of a word, leading to a similar representation of a word and its anatomy since both appear in a similar context. Another drawback is that we only consider a fixed-size window of context words for conditioning the language model. A more natural way to learn representation is to allow a variable amount of context words.

Recurrent Neural Network (RNN). A RNN is a class of artificial neural networks that are specialized to process sequential data, e.g., natural language text. RNNs are capable of conditioning the model to an arbitrary number of words in the sequence. Figure 2.2 depicts the architecture of an uni-directional RNN where each vertical box is a hidden layer at a time-step t. At each time step t, the hidden layer gets two inputs: the output of the previous layer h_{t-1} and the input at that time-step x_t,

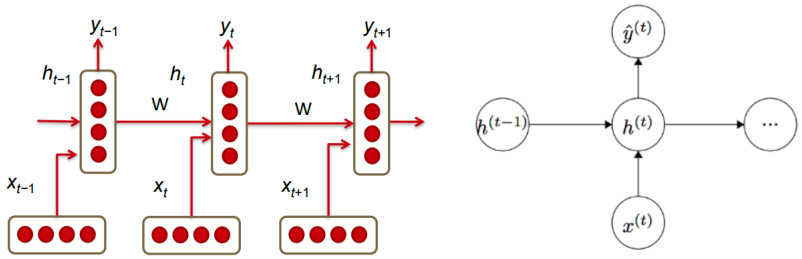


Figure 2.2: A Recurrent Neural Network (RNN). *(Left)* Three time-steps are shown. *(Right)* The inputs and outputs to a neuron of a RNN. Images taken from the Stanford CS224n course.

see Figure 2.2 *(right)*. To produce the output features h_t and to obtain a prediction output $\hat{y}$ of the next word, we utilize the weight matrices $W^{(hh)}, W^{(hx)}, W^{(S)}$ as follows:

$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$$
$$\hat{y}_t = \text{softmax}\left(W^{(S)}h_t\right)$$

Notice that the weights $W^{(hh)}, W^{(hx)}$ are applied repeatedly at each time step, therefore sharing the weights across time steps. This allows the model to process sequences of arbitrary length. Furthermore, the model size does not increase with longer input sequences. In theory, RNNs can use information from any steps from the past. However, in practice, this is difficult as the vanishing and exploding gradients become a big issue with long sequences [Hochreiter, 1998, Bengio et al., 1994] which then makes the model insensitive to past inputs. To alleviate these issues, we mention some heuristic solutions: Clipping the gradient to a small number whenever they explode [Pascanu et al., 2012], initialization of $W^{(hh)}$ as the identity matrix since it helps avoid the vanishing gradients [Le et al., 2015] and using the Rectified Linear Units (ReLU) instead of the sigmoid function [Agarap, 2018]. However, one of the most important extensions to solve the vanishing gradient problem is the so-called long-short term memory (LSTM) [Hochreiter and Schmidhuber, 1997, Gers et al., 2000], which is a sub-architecture for the hidden layer of an RNN. The LSTM unit introduces a gating mechanism that selectively propagates only a subset of relevant information across time steps and consequently mitigates the vanishing gradient problem.

RNNs and LSTMs started to dominate NLP, either performing competitively or outperforming existing state-of-the-art on various tasks [Sutskever et al., 2011,

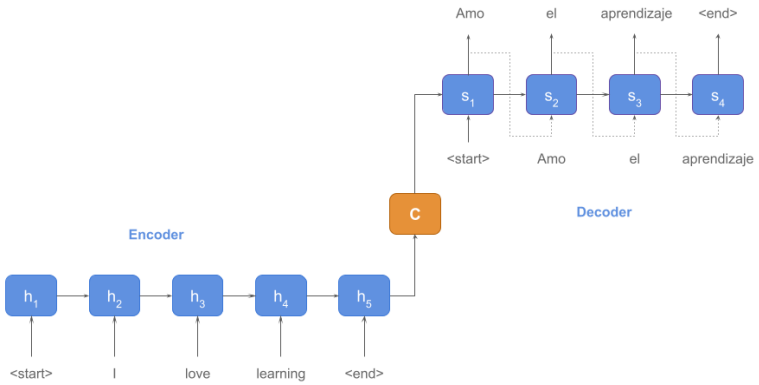


Figure 2.3: Encoder-decoder architecture of an RNN on a machine translation task. The encoder produces a thought vector C, representing the source sentence. The decoder then unfolds state C to produce an output sequence.

Mikolov et al., 2011, Sutskever et al., 2014]. One particular interesting architecture emerged to address tasks where a output sequence was needed such as machine translation: The *encoder-decoder* architecture. The architecture was first proposed by Hinton and Zemel [1993] and was then later used in the context of NLP [Kalchbrenner and Blunsom, 2013, Sutskever et al., 2014], see Figure 2.3[3]. The encoder part takes a sequence as an input and outputs a single vectorized representation of the whole sequence (called thought vector), which the decoder takes as an input to generate a sequence.

However, since the thought vector has a fixed size representation form, and the decoder only depends on the thought vector, the representative power of the (unidirectional) RNN encoder-decoder architecture for sequences is naturally limited. All the information about the input has to be encoded into the fixed-size thought vector, which becomes increasingly difficult for long sequences.

Attention. To improve upon the above described shortcoming of encoder-decoders, Bahdanau et al. [2014] introduces the concept of *attention*. The attention module allows the decoder to re-access and select encoder hidden states at decoding time, see Figure 2.4[4]. Following the numbers in the Figure 2.4, the attention module can be explained by the following: (1) The decoder's hidden state and (2) the intermediate hidden states of the encoder are being fed into the attention module. (3)

[3]Figure taken from https://www.baeldung.com/cs/nlp-encoder-decoder-models

[4]Figure taken from https://project-archive.inf.ed.ac.uk/ug4/20201880/ug4_proj.pdf

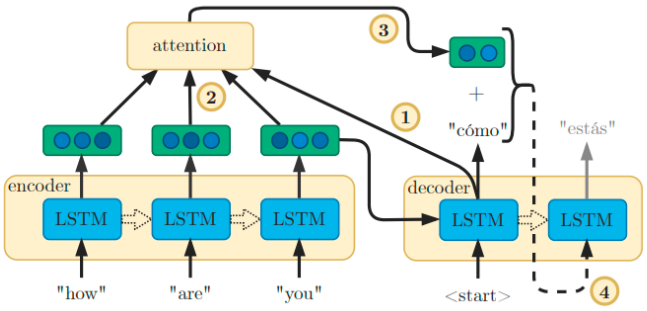


Figure 2.4: An encoder-decoder model for machine translation with added attention mechanism.

The attention module then selects relevant information from the hidden states of the encoder based on the decoder's hidden state and calculates the context vector. Finally, (4) the decoder takes the context vector and the last output word ("cómo") as the input and outputs the next word. Galassi et al. [2021] gives an exhaustive overview of different attention implementations. However, we restrict ourselves to the common attention mechanism used in transformers, explained in the next paragraph.

First, the decoder state $\boldsymbol{h}_D \in \mathbb{R}^{1\times d}$ is embed to a *query* $\boldsymbol{q} \in \mathbb{R}^{1\times d_k}$ using a learnable weight matrix $W_Q \in \mathbb{R}^{d\times d_k}$:

$$\boldsymbol{q} = \boldsymbol{h}_D W_Q$$

and each encoder state $\boldsymbol{h}_E^{(i)}$, where i denotes the encoder time-step, is stacked to a encoder state matrix $\boldsymbol{H}_E$ and is used to produce the *key* matrix $\boldsymbol{K}$ and *value* matrix $\boldsymbol{V}$:

$$\boldsymbol{K} = \boldsymbol{H}_E W_K, \quad \boldsymbol{V} = \boldsymbol{H}_E W_V,$$

where $W_K \in \mathbb{R}^{d\times d_k}, W_V \in \mathbb{R}^{d\times d_v}$ are learnable weights. We then calculate the *attention weights* which computes how relevant a single key $\boldsymbol{k}^{(i)}$ vector is for the query $\boldsymbol{q}$:

$$w = \boldsymbol{q}\boldsymbol{K}^T.$$

Commonly, the weights are normalized to a probability distribution using the softmax function which are then used to create the *context vector* $\boldsymbol{c}$ by taking the

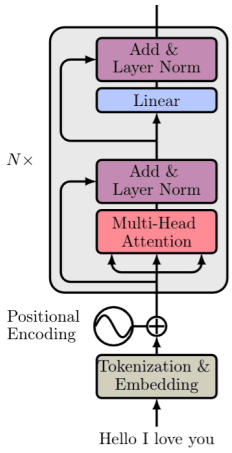


Figure 2.5: A transformer encoder block, adopted from Vaswani et al. [2017].

weighted average of the values:

$$\boldsymbol{c} = \sum_i a^{(i)} \boldsymbol{v}^{(i)} \quad \text{where} \quad a^{(i)} = \text{softmax}(w)_i.$$

During training, we optimize the weights W_Q, W_K, W_V which then improves the selective focus of the attention module. We can summarize the (dot-product) Attention with:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(QK^T\right) V. \tag{2.3}$$

Notice that we extended the vector query to a matrix query Q where each row represents one query vector.

Transformers. Even though attention solves the issue of restrictive expressiveness, RNNs have another main architectural drawback: RNNs are slow since they are sequential and therefore hard to parallelize. A new model architecture based solely on attention mechanisms and fully parallelised was proposed by Vaswani et al. [2017], called *Transformers*, an encoder-decoder model. In this thesis, our models only rely on the encoder part of the model, which is why we omit the description of the decoder. We visualize the architecture of the encoder in Figure 2.5.

First, the sequence is tokenized, then the model embeds each token in the input sequence with a token embedding layer, then adds a positional encoding[5] depending on the position of the token in the sequence. These representations

[5]Notice that without the positional encoding, the transformer has no notion of word order.

are then routed N times through separate (self-)attention and feedforward sub-networks. The core difference between the attention module described above and self-attention is that the query matrix $\boldsymbol{Q}$ is generated from tokens of the input sequence and can attend to all other tokens of the same sequence, including itself, to generate its new representation. Furthermore, they do not use the dot-product attention (2.3) but the *scaled dot-product attention*:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \tag{2.4}$$

Additionally they utilize *multiple heads* (multiple self-attention layers), which split the queries, keys, and values matrices Q, K, V along the embedding dimensions with $d_k = d_v = d/h$ where h is the number of heads. Subsequently, they apply the self-attention independently, each having its own parameters. The advantage of multi-heads is that tokens can jointly attend to multiple tokens in the sequence. Each head produces its own output and gets concatenated, once again projected, resulting in the final values

$$\text{MultiHead}(Q, V, K) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.5}$$

where $W^O \in \mathbb{R}^{hd_v \times d}$ is the projection matrix. Finally, the result is fed into a feedforward neural network. Additionally, the architecture uses dropout, residual connections, and layer normalization to stabilize and improve training.

Using the transformer architecture has improved upon RNNs in many ways: Through multi-heads, the total computational complexity per layer is much lower, and through their ability to parallelize many computations, the scalability of transformers far exceeds RNNs. Therefore, stacking transformer blocks to increase the representative model capacity can be done efficiently. Furthermore, the path length between long-range dependencies in the network is reduced from $O(n)$ to $O(1)$ as self-attention allows access to the input directly.

Another critical aspect of transformers is the pre-training and fine-tuning paradigm: The general procedure is to pre-train on a language modeling task on huge training text, which is possible because of the high parallelizability of transformers, e.g., Radford and Narasimhan [2018] train on the next word prediction task on a corpus with over 7000 books. Given a downstream task, the whole pre-trained model (coupled with a task head) is then fine-tuned on the task dataset.

2.1.4 Cross-Lingual Static Word Representations

In the previous Section 2.1.3, we outlined different architectures to induce word representations. However, we restricted ourselves to inducing monolingual word

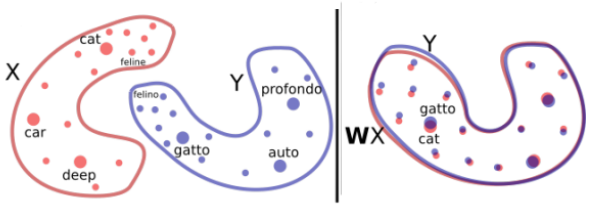


Figure 2.6: Illustration of projection-based methods. X and Y are monolingual spaces and W the projection matrix. Adopted from Conneau et al. [2017].

representations by pre-training on monolingual text corpora (with the language modeling task, i.e., predicting the next word). Since monolingual word embeddings pre-train in each language independently, therefore only learning monolingual distributional information, they can not capture semantic relations between words across languages. The fundamental idea behind cross-lingual word representations is to create an aligned representation space for word representations from across multiple languages. This section briefly discusses how to extend static word embeddings (induced by, e.g., word2vec and FastText) to create a cross-lingual space.

FastText and word2vec induce static word embeddings, i.e., they do not consider the context of a word in their representation. Therefore phrases and polysemy of words can not be correctly captured, which prohibits the effectiveness of an aligned space across languages with static embeddings. Nonetheless, we discuss two popular approaches: (1) Projection-based models and (2) Bilingual Embeddings.

Projection-based methods. Projection-based methods rely on independently trained monolingual word vector spaces in the source language $\boldsymbol{X}_{L1}$ and target language $\boldsymbol{X}_{L2}$, that post-hoc align the monolingual spaces into one cross-lingual word embedding $\boldsymbol{X}_{CL}$, see Figure 2.6. The alignment is based on word translation pairs D, which can be obtained by existing dictionaries or by inducing it automatically. The former is a supervised method and the latter an unsupervised approach that usually assumes (approximately) isomorphism between monolingual spaces. Typically, a supervised projection-based method uses a pre-obtained dictionary D containing word pairs for finding the alignment [Mikolov et al., 2013b, Huang et al., 2015]. Unsupervised methods induce the dictionary using different strategies such as adversarial learning [Conneau et al., 2017], similarity-based heuristics [Artetxe et al., 2018], PCA [Hoshen and Wolf, 2018], and optimal transport [Alvarez-Melis and Jaakkola, 2018].

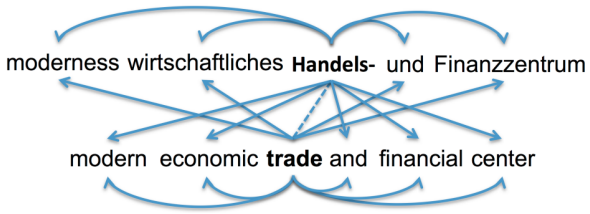


Figure 2.7: The BiSkip-gram predicts within language and cross-lingually based on the alignment information. Image taken from Luong et al. [2015].

Projection-based methods construct an aligned monolingual subspace $\boldsymbol{X}_S$ and $\boldsymbol{X}_T$, where the aligned rows are translations of each other. Mikolov et al. [2013b] learns the projection $\boldsymbol{W}_{L1}$, by minimizing the Euclidean distance between the linear projection of $\boldsymbol{X}_S$ onto $\boldsymbol{X}_T$:

$$\boldsymbol{W}_{L1} = \arg\min \|\boldsymbol{X}_S\boldsymbol{W} - \boldsymbol{X}_T\|$$

which can be further improved by constraining $\boldsymbol{W}_{L1}$ to be an orthogonal matrix Xing et al. [2015].

The induced cross-lingual space performs well for related languages on the BLI task but degrades when the language pair is distant [Vulić et al., 2019]. Furthermore, Glavas et al. [2019] show that the BLI is not necessarily correlated to downstream performance.

Bilingual Embeddings. Bilingual Embeddings induce the cross-lingual space by jointly learning representations from scratch. In general, the general joint objective can be expressed as:

$$\alpha(Mono_1 + Mono_2) + \beta Bi$$

where $Mono_1$ and $Mono_2$ are monolingual models, aiming to capture the clustering structure of each language, whereas the bilingual component, Bi, encodes the information that ties the two monolingual spaces together [Klementiev et al., 2012, Luong et al., 2015]. The hyperparameters α and β weight the influence of the monolingual components and the bilingual component.

One popular choice is the BiSkip-gram model which extends the Skip-gram model (see Section 2.1.3) by predicting words crosslingually rather than just monolingually, see Figure 2.7. However, the approach is expensive in terms of supervision as the BiSkip-gram approach is based on a parallel corpus. Furthermore, for low-resource languages, this level of supervision is, in some cases, impossible to acquire.

2.1.5 Cross-Lingual Contextualized Word Representations

Transformers and RNNs produce contextualized word embeddings, i.e., they encode the same word differently depending on its context. We already discussed in Section 2.1.3 why transformers improve upon RNNs from an architectural standpoint and how it benefits the pre-training. This section will first dive into one of the most popular transformer models and its pre-training tasks. We then explore multilingual transformers and, more importantly, the XLM-R model as it builds the foundation of our thesis.

BERT. Before exploring multilingual transformer models, we introduce the perhaps most popular transformer: *BERT* (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) [Devlin et al., 2019]. BERT is a sequence encoder-only model, which slightly modifies the Transformer architecture [Vaswani et al., 2017] in the following ways: As it only consists of an encoder, the model allows information to flow bidirectionally, creating bidirectional representations.

Specifically, the BERT encoder is composed of L layers, each layer l with M self-attention heads, where a self-attention head (m, l) has a *key*, *query* and *value* encoders. Each one is calculated by a linear layer:

$$\begin{aligned}\boldsymbol{Q}^{m,l}(\boldsymbol{x}) &= \boldsymbol{W}_q^{m,l}(\boldsymbol{x}) + \boldsymbol{b}_q \\ \boldsymbol{K}^{m,l}(\boldsymbol{x}) &= \boldsymbol{W}_k^{m,l}(\boldsymbol{x}) + \boldsymbol{b}_k \\ \boldsymbol{V}^{m,l}(\boldsymbol{x}) &= \boldsymbol{W}_v^{m,l}(\boldsymbol{x}) + \boldsymbol{b}_v\end{aligned}$$

where the input $\boldsymbol{x}$ is the output representation embedding layer for the first encoder layer, and for the rest of the layers, $\boldsymbol{x}$ is the output representation of the former encoder layer. These are then fed into the scaled dot-product attention (2.4) (which does not introduce any new parameters) and concatenated by the projection (2.5), outputting $\boldsymbol{h}_1^l$. The output is then again fed into the MLP with the layer norm component of the transformer block:

$$\begin{aligned}\boldsymbol{h}_2^l &= \text{Dropout}(\boldsymbol{W}_{m_1}^l \cdot \boldsymbol{h}_1^l + \boldsymbol{b}_{m_1}^l) \\ \boldsymbol{h}_3^l &= \boldsymbol{g}_{LN_1}^l \odot \frac{(\boldsymbol{h}_2^l + \boldsymbol{x}) - \mu}{\sigma} + \boldsymbol{b}_{m_1}^l \\ \boldsymbol{h}_4^l &= \text{GELU}(\boldsymbol{W}_{m_2}^l \cdot \boldsymbol{h}_3^l + \boldsymbol{b}_{m_2}^l) \\ \boldsymbol{h}_5^l &= \text{Dropout}(\boldsymbol{W}_{m_3}^l \cdot \boldsymbol{h}_4^l + \boldsymbol{b}_{m_3}^l) \\ \text{out}^l &= g_{LN_2}^l \odot \frac{(\boldsymbol{h}_5^l + \boldsymbol{h}_3^l) - \mu}{\sigma} + \boldsymbol{b}_{LN_2}^l.\end{aligned}$$

The model parameters Θ are therefore constructed from the weight matrices $\boldsymbol{W}_{(\cdot)}^{l,(\cdot)}$, bias vectors $\boldsymbol{b}_{(\cdot)}^{l,(\cdot)}$ and vectors $\boldsymbol{g}_{(\cdot)}^l$.

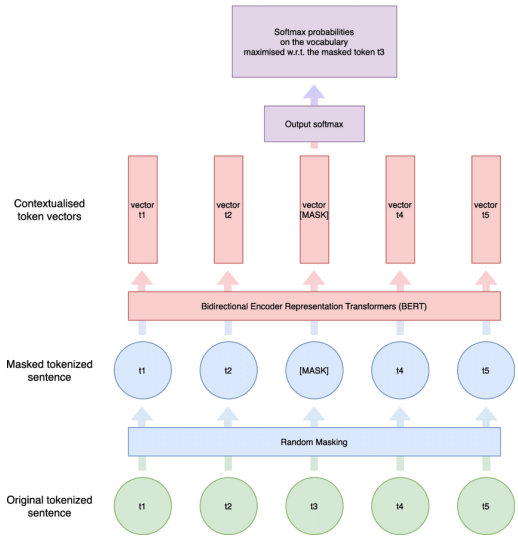


Figure 2.8: Masked Language Modeling task in BERT. Masked tokens get replaced with a special token `[MASK]`. Taken from Torregrossa et al. [2021].

Furthermore, the pre-training task is not the next word prediction anymore but consists of two novel pre-training tasks: (1) The masked language modeling task (MLM) and the (2) next-sentence prediction (NSP) task. MLM masks tokens of the input sequence, and the task is to predict the original token based on the masked sequence, see Figure 2.8. The masking happens by selecting 15% tokens, then 80% are masked, or 10% replaced by a random token or 10% left unchanged. On the other hand, NSP asks the model to predict whether the input, which consists of two concatenated sentences, if they are consecutive to one another. Specifically, they construct sentence pairs by taking 50% actual sentence pairs that are consecutive and 50% artificial sentence pairs that are not consecutive. Additionally, while tokenizing the input sequence, BERT inserts special tokens, such as the `[CLS]` token and `[SEP]`, where the former is always inserted at the start of a sequence, and the latter separates two sentences allowing to process sentence pairs. The pre-trained BERT model can then be fine-tuned end-to-end by adding one additional output layer, which either takes the token representations for token level tasks or the `[CLS]` representation for classification tasks as an input. BERT achieves state-of-the-art for a wide range of tasks, such as question answering and language inference [Devlin et al., 2019] and marks the start of modern NLP with transformers. A noteworthy modification of the BERT model is *RoBERTa* (Robustly Optimized BERT Pretraining Approach) [Liu et al., 2019b]: First, they remove the NSP task,

use bigger batch sizes & longer sequences, and use a dynamic masking strategy, i.e., the masking pattern is generated every time a sequence is fed to the model. RoBERTa outperforms BERT on GLUE, RACE, and SQuAD.

mBERT. The idea to extend BERT to multiple languages is to concatenate multiple monolingual corpora to then jointly pre-train on it. As this massive multilingual corpus from many languages has an enormous vocabulary size and a large number of out-of-vocabulary tokens, BERT/mBERT uses a *subword-based tokenizer*. The idea is to split rare words into smaller meaningful subwords, e.g., "papers" is split into "paper" and "s". The model then learns that the word "papers" is formed using the word "paper" with a slightly different meaning but the same root word. There are many different implementations of this idea, such as WordPiece [Wu et al., 2016], BPE [Sennrich et al., 2015] and SentencePiece [Kudo and Richardson, 2018]. The original BERT/mBERT implementation uses WordPiece. To encode text from multiple languages, the subword tokenizer creates its vocabulary on the concatenated text. The multilingual version of BERT, dubbed *mBERT*, pre-trains on 104 languages and surprisingly learns strong cross-lingual representations that generalize well to other languages via zero-shot transfer [Pires et al., 2019, Wu et al., 2019] without any explicit supervision.

This ability can be explained by three factors [Pires et al., 2019]: (1) The subword tokenizer maps common subwords across languages which act as anchor points for learning an alignment, e.g., "DNA" [6] has a similar meaning even in distantly related languages. The anchor points are similar to the seed dictionary in the projection-based approach (see Section 2.1.4). (2) This effect is then reinforced and distributed to other non-overlapping tokens by jointly training across multiple languages forcing co-occurring tokens also to be mapped to a shared space. (3) mBERT learns cross-lingual representations deeper than simple vocabulary memorization, generalizing across languages. However, recent works [Wu and Dredze, 2019, K et al., 2020] show that a shared vocabulary is not required to create a strong cross-lingual representation. [K et al., 2020] additionally demonstrate that word order plays an important role.

XLM. Lample and Conneau [2019] introduce a new unsupervised method for learning cross-lingual representations, called XLM (cross-lingual language models). XLM builds upon BERT and makes the following adjustment: First, they include the Translation Language Modeling (TLM) task into the pre-training. Each training sample consisting of pairs of parallel sentences (source and target sentence) is randomly masked. To predict a masked word, the model is then allowed to either attend to the surrounding source words or the target translation, encourag-

[6] "DNA" is indeed a subword in mBERT [Wu and Dredze, 2019].

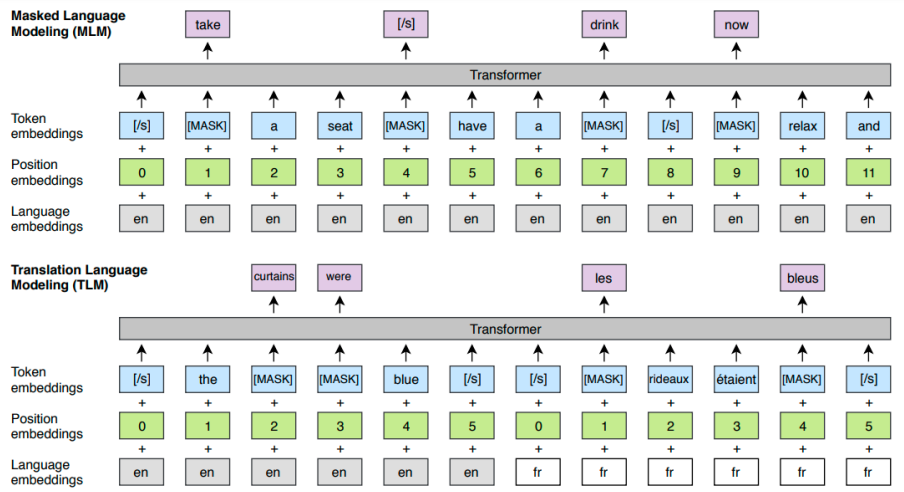


Figure 2.9: Cross-lingual language model pretraining. Image taken from Lample and Conneau [2019].

ing the model to align the source and target representations, see Figure 2.9. They then choose to drop the NSP task and only alternate training between MLM and TLM. Furthermore, the model receives a language ID to its input (similar to positional encoding), helping the model learn the relationship between related tokens in different languages. XLM uses the subword tokenizer BPE [Sennrich et al., 2015] which learns the splits on the concatenation of sentences sampled randomly from the monolingual corpora. Furthermore, XLM samples according to a multinomial distribution with probabilities:

$$q_i = \frac{p_i^\alpha}{\sum_{j=1}^{N} p_i^\alpha} \quad \text{with} \quad p_i = \frac{n_i}{\sum_{k=1}^{N} n_k} \tag{2.6}$$

where i denotes the index of the language and n_i the the number of sentences in the text corpora of the language with the index i. XLM uses $\alpha = 0.5$.

XLM outperforms mBERT on XNLI [Conneau et al., 2018] in 15 languages. However, XLM handles fewer languages than mBERT, is based on a larger model, and uses a high amount of supervision as it needs parallel sentences during pretraining. Therefore the difference may not be so significant in reality. Furthermore, acquiring parallel sentences for low-resource languages is problematic, making the model unsuitable for such scenarios.

XLM-R. Conneau et al. [2020] propose to take a step back and drop the TLM task and only pre-train in RoBERTa fashion with the MLM task on a huge, multilin-

gual dataset. They dub their multilingual model XLM-RoBERTa (XLM-R). They crawled a massive amount of text, over 2.5TB of data in 100 languages. Additionally, they changed the vocabulary size of RoBERTa to 250000 tokens compared to RoBERTa's 50000 tokens. They employ the subword tokenizer SentencePiece [Kudo and Richardson, 2018] with an unigram language model [Kudo, 2018]. They use the same sampling strategy (2.6) as XLM, but utilize $\alpha = 0.3$. Furthermore, XLM-R does not use language IDs, which will allow XLM-R to better deal with code-switching. Conneau et al. [2020] provide two models: XLM-R_{Base} ($L = 12, H = 768, A = 12$, 270M params) and XLM-R ($L = 24, H = 1024, A = 16$, 550M params).

Conneau et al. [2020] show that XLM-R sets a new State-of-the-Art on numerous cross-lingual tasks. Compared to mBERT and XLM, XLM-R provides substantial gains in classification, sequence labeling, and question answering *without any explicit cross-lingual supervision*.

2.1.6 Challenges in Multilingual Transformers

As XLM-R and the concept of multilingual transformers build the basis of our thesis, we will further analyze weaknesses and how our approach tries to alleviate them.

Low-Resource Languages. Even though multilingual LMs do not use any explicit cross-lingual signals, they still create multilingual representations [Pires et al., 2019, Wu and Dredze, 2019], which can be used for cross-lingual downstream tasks. By releasing a new multi-task benchmark for evaluating the cross-lingual generalization, called XTREME[7], which covers nine tasks and 40 languages, Hu et al. [2020] showed that even though models achieve human performance on many tasks in *English*, there is a sizable gap in the performance of cross-lingually transferred models, especially in *low-resource languages*. Additionally, Wu and Dredze [2019], Lauscher et al. [2020] showed that multilingual transformers pre-trained via language modeling significantly underperform in resource-lean scenarios and for distant languages. Furthermore, the literature [Pires et al., 2019, Wu and Dredze, 2019, K et al., 2020] focused on evaluating languages that were from the same language family or with large corpora in pre-training, languages such as German, Spanish or French. For example, K et al. [2020] investigate Hindi, Spanish, and Russian, which are from the same language family, Indo-European, and have a large corpus from Wikipedia. This concern is raised by multiple sources [Lauscher et al., 2020, Wu and Dredze, 2019], which show that the performance drops huge for distant target languages and target languages that have small pre-

[7] https://sites.research.google/xtreme

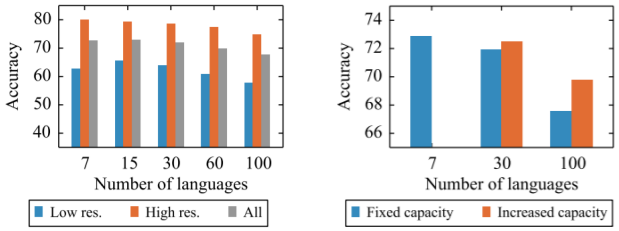


Figure 2.10: Trade-off between number of languages in pre-training vs. XNLI performance. *(Left)* From 7-15 languages, the model benefits from positive cross-lingual transfer, but degrades after the curse of multilingualty kicks in. *(Right)* Adding more model capacity can alleviate some of the curse. Images taken from Conneau et al. [2020].

training corpora. Furthermore, Lauscher et al. [2020] show empirically that for massively multilingual transformers, pre-training corpora sizes affect the zero-shot performance in higher-level tasks. In contrast, the results in lower-level tasks are more impacted by typological language proximity.

As multilingual transformers struggle with low-resource languages, we investigate if our approach is capable of improving in these scenarios. To strengthen the performance of low-resource languages, we try to avoid the curse of multilinguality.

Curse of Multilinguality. Conneau et al. [2020] experiment with different settings for the XLM-R model, showing that scaling the batch size, training data, training time and shared vocabulary improve performance on downstream task. More importantly, they show that for a fixed model capacity, adding more languages to the pre-training lead to better cross-lingual performance on low-resource languages till the per-language capacity is too low (*capacity dilution*), after which the overall performance on monolingual and cross-lingual benchmarks degrades. They call this the *curse of multilinguality*, see Figure 2.10. Adding more languages to the pre-training ultimately has two important effects: (1) Positive cross-lingual transfer, especially for low-resource languages, and (2) lower per-language capacity, which then, in turn, can degrade the overall model performance. Multilingual transformer models have to carefully balance these two effects of *capacity dilution* and positive transfer. Adding more capacity to the model can alleviate some of the curse but is not a solution for moderate-size models.

Furthermore, the allocation of the model capacity across languages is influenced by the training set size, the size of the shared subword vocabulary, and the rate at which the model samples training instances from each language during pre-

training. Conneau et al. [2020] show that sampling batches of low-resource languages improve performance on low-resource languages and vice-versa. XLM-R uses a rate of $\alpha = 0.3$, which still leaves some room for improvement on low-resource languages.

Our approach introduces language-specific students that allocate 100% of model parameters to one language, avoiding the capacity dilution while still benefiting from the acquired cross-lingual knowledge from XLM-R by distilling from the XLM-R model.

2.2 Knowledge Distillation

Knowledge Distillation is a technique to transfer knowledge from a well-trained teacher model to a student model, typically smaller than the teacher. During training, Knowledge Distillation allows the student model to access the learned knowledge from the teacher model, providing more information to the student. Thus, the student can achieve similar or even better performance than the teacher.

As our thesis is based on multilingual transformers, we restrict our review to Knowledge Distillation for transformers.

2.2.1 Motivation & Preliminaries

Motivation. Bucila et al. [2006] first introduced the concept of Knowledge Distillation as a way to compress models by transferring the learned knowledge to a smaller model. In their work, the primary goal was to compress a large complex ensemble into smaller, faster models without significant performance loss. First, they train the ensemble model (also called *teacher*) and then transfer the acquired knowledge to a smaller model (also called *student*) by mimicking the teacher's behavior. Thus the idea of Knowledge Distillation[8] was born.

Knowledge Distillation is especially beneficial in the case of complex, over-parameterized teachers. From an optimization perspective, Du and Lee [2018], Soltanolkotabi et al. [2018] show that high capacity models (i.e., the teacher) can find a good local minimum due to over-parameterization. Therefore, using these over-parameterized models to guide a lower capacity model during training can facilitate optimization. The state-of-the-art multilingual models are massive pre-trained transformers consisting of millions of parameters which some argue are over-parameterized but need the capacity during pre-training [Hao et al., 2019, Conneau et al., 2020, Dufter and Schütze, 2021]. Our approach uses Knowledge

[8] Knowledge Distillation can also be referred to as teacher-student Knowledge Distillation.

Distillation to precisely distill these transformer models to induce cross-lingual knowledge into students.

Vanilla Knowledge Distillation. The most straightforward approach to Knowledge Distillation is to distill from the output layer by matching the output of the teacher and student [Bucila et al., 2006, Ba and Caruana, 2013, Hinton et al., 2015]. Initially, Bucila et al. [2006] used the teacher to make predictions on an unlabeled dataset producing labels for the student to mimic. The created labels are referred to as *hard labels*, and the dataset used to train the student is called *transfer dataset*. As hard labels only transfer information about the highest class probability, Ba and Caruana [2013] propose to use the logits of the teacher, called *soft labels*. The idea is that the soft labels of a well-trained teacher provide additional supervisory signals of the inter-class similarities to the student. They directly used the logits z by minimizing the squared difference between the logits produced by the teacher and the logits produced by the student:

$$\mathcal{L}_{\text{logits}} = \frac{1}{2N} \sum_{x \in \mathcal{X}} \| z_x^{TE} - z_x^{ST} \|_2^2, \tag{2.7}$$

where N is the number of examples, z_x^{TE} the logit output of the teacher and z_x^{ST} of the logit output of the student.

One problem that may arise in well-trained teachers is that they are overconfident and thus almost always predict classes (tokens) with very high confidence. The learned similarities between similar classes (tokens) reside in the ratios of very small probabilities in the soft targets - These, however, have very little influence on the cost function (2.7) during distillation. To leverage the information residing in these small probabilities Hinton et al. [2015] propose to *soften* the distribution by modifying the predicted probability distribution of the teacher. Hinton et al. [2015] introduce a variable called temperature into the final softmax:

$$\sigma(z_x; T)_i := p_i = \frac{\exp\left(z_{x,i}/T\right)}{\sum_j \exp\left(z_{x,j}/T\right)}, \tag{2.8}$$

where T denotes the temperature, which is normally set to 1 for the softmax function. A higher temperature T causes a softer probability distribution over tokens, resulting in a higher entropy in the probability distribution. Hinton et al. [2015] then use the same temperature T when training the student to match these soft targets. Furthermore, Hinton et al. [2015] showed that matching logits is a special case of the modified softmax in (2.8). Using the Kullback–Leibler divergence between teacher and student, we get the *(unscaled) Hinton loss function*:

$$\sum_{x \in \mathcal{X}} D_{KL}\left(\sigma(z_x^{TE}; T) \| \sigma(z_x^{ST}; T)\right).$$

To compensate for the changed magnitude of the gradients caused by the soft targets scale, Hinton et al. [2015] multiply the loss function with T^2 to get the *Hinton loss function*:

$$\mathcal{L}_H = T^2 \cdot \sum_{x \in \mathcal{X}} D_{KL}\left(\sigma(z_x^{TE}; T) \| \sigma(z_x^{ST}; T)\right). \tag{2.9}$$

This allows us to change the temperature while experimenting without changing the relative contributions of the soft targets. The Hinton loss (2.9) is the vanilla setup that is mainly used and referred to as (vanilla) Knowledge Distillation.

The main difference between different Knowledge Distillation strategies is how one mimics the teacher's behavior. Formally, we can define a transformation f^{TE} and f^{ST} of the teacher and student network inputs, respectively, to some informative representation for transfer. Jiao et al. [2020] calls these transformations *behavior functions*. Knowledge Distillation can then be defined in general as minimizing the objective function

$$\mathcal{L}_{\text{KD}} = \sum_{x \in \mathcal{X}} \mathcal{L}\left(f^{TE}(x), f^{ST}(x)\right),$$

where $\mathcal{L}(\cdot)$ is a loss function evaluating the difference between a teacher and student networks, x the (text) input, and $\mathcal{X}$ denotes the training set. Behavior functions can be defined for any type of representation of the input, e.g., the logits of the final output or some intermediate representations in the network, which the student then mimics.

2.2.2 Brief History: Knowledge Distillation for Transformers

Knowledge Distillation methods first found their use in the area of Computer Vision. As AlexNet [Krizhevsky et al., 2012], one of the largest neural networks at that time, completely outperformed other methods on ImageNet [Deng et al., 2009], the trend towards bigger models started in that area. To keep the models to a relatively acceptable size while still having the same performance as the big models, a variety of model compression techniques were developed, such as Knowledge Distillation, see the survey of Gou et al. [2021] for a more in-depth look into Knowledge Distillation in Computer Vision.

Task-Specific Distillation. Sun et al. [2019] was one of the first known successes at using Knowledge Distillation for BERT at the fine-tuning stage. They introduce *Patient Knowledge Distillation (PKD)* which uses the intermediate representations of BERT for a more effective distillation (additional to the Hinton Loss). This

is motivated by previous findings of Romero et al. [2015] showing that distilling intermediate representations can serve as hints for the student during training, improving the final performance. Similar to PKD, XtremeDistil [Mukherjee and Awadallah, 2020] also distills from intermediate representations but additionally utilizes parameter projection that is agnostic of teacher architecture. Subsequently, inspired by the findings that attention weights of BERT capture linguistic knowledge and that BERT becomes more complex in higher layers [Clark et al., 2019a], *Stacked Internal Distillation (SID)* [Aguilar et al., 2020] additionally distills the attention probabilities of the teacher and from lower layers first. However, these works used distillation for fine-tuned models on specific downstream tasks (*task-specific distillation*). In our thesis, we want to produce a general-purpose student that can be fine-tuned on any downstream task. While we are not focusing on task-specific distillation [Liu et al., 2019a, Turc et al., 2019, Tang et al., 2019, Kaliamoorthi et al., 2021] or multi-task distillation [Tan et al., 2019, Clark et al., 2019b, Liu et al., 2020], we still want to mention important work in this field.

General-Purpose Students. To produce general-purpose students, distillation happens during the pre-training tasks, mainly the masked language modeling task. In this task, the transformer model is trained to predict the original token for each masked token by maximizing the estimated probability for this token. The standard loss function is to minimize the cross-entropy between the transformer's predicted distribution and the one-hot distribution of all tokens. We denote the loss as the masked language modeling loss $\mathcal{L}_{\text{MLM}}$ [Devlin et al., 2019] as

$$\mathcal{L}_{\text{MLM}} = -\sum_{x \in \mathcal{X}} \left(\sum_{i} l_{x,i} \log(p_{x,i}) \right), \tag{2.10}$$

where $l_{x,i}$ is the ground-truth and $p_{x,i}$ the predicted probability for i-th token in the text input x. Typically a transformer applies a softmax function to obtain $p_{x,i}$, denoted by $\sigma(z_x) = p_x = (p_{x,1}, ..., p_{x,C})$, where σ is the softmax function and $z_x = (z_{x,1}, ..., z_{x,C})$ the logit output of the text input x.

DistilBert [Sanh et al., 2020] extends the work of Sun et al. [2019] by distilling during pre-training on a large-scale corpus with a soft-label distillation loss and a cosine embedding loss to construct a general-purpose student. Furthermore, they initialize the student from the teacher by taking one layer out of two. Similar to SID [Aguilar et al., 2020], TinyBert [Jiao et al., 2020] introduce additional attention distillation to produce general-purpose students. Specifically, they distill self-attention distributions Attention(Q, K, V) (2.4). TinyBert further uses task distillation with data augmentation to strengthen performance on downstream tasks. They show that their 6 layer model performs on par with its teacher BERT on the GLUE benchmark [Wang et al., 2019]. Instead of using shallower students, MobileBERT

[Sun et al., 2020] uses thinner students by introducing bottleneck structures which have been shown to be more effective [Turc et al., 2019]. Compared to TinyBert, they outperform it on GLUE and SQuAD with a similar-sized MobileBERT while not utilizing any task-distillation or data augmentations during fine-tuning. However, MobileBERT constructs another teacher network and many architectural modifications to help facilitate training. MiniLM [Wang et al., 2020] only distills the self-attention module with an added loss function: They align the relation between values in the self-attention module, which is calculated via the multi-head scaled dot-product between values V. Furthermore, they only distill from the last transformer layer of the teacher. Compared to TinyBert and MobileBert, MiniLM alleviates the difficulties in layer mapping between the teacher and student models, and the layer number of our student model can be more flexible. Khanuja et al. [2021] introduce MergeDistil, which uses multiple mono- or multilingual teachers to distill into one general-purpose student to leverage language-specific LM.

To further dive deeper into previous works and build a foundation for further exploration for our thesis, we distinguish previous works into which parts of the transformer they distill from.

2.2.3 Transformer Components Distillation

This section will explore different behavior functions and their corresponding loss function to transfer knowledge from a transformer teacher into a transformer student. We categorize the possible parts of the transformer that we can distill from into: *Final output layer*, *hidden layer representations*, *attention* and *embedding distillation*. We summarize our findings in Table 2.1.

Final Output Layer. One simple idea to induce knowledge from a teacher during pre-training is to use the predicted distribution over all tokens of the teacher model as *soft targets* for training the student. The hope is that soft targets are much more informative than hard targets as they can reveal similarities between tokens, e.g., the masked token is ”dog” and our well-trained teacher model would give a high probability to ”dog”, but also to ”cat” as both could make sense in the masked sentence. This information about similarities between tokens can then be transferred to our students. One problem that may arise in well-trained teachers is that they almost always predict the correct token with very high confidence.

Almost all works use the hard and soft targets to distill knowledge from the teacher. PKD, SID, and DistilBERT use the cross-entropy loss to distill from the soft and hard targets, while XtremeDistil uses the mean-squared error to align soft targets. In some implementations, the Kullback Leibler loss is used to align soft targets. During pre-training, TinyBert does not use any output layer distillation

Distillation Name	Embedding	Hidden Layer Distill		Final Output Layer	
		Attention	Hidden Representation	Soft Targets	Hard Targets
Hinton Loss [Hinton et al., 2015]	–	–	–	**CE**	**CE**
PKD [Sun et al., 2019]	–	–	**MSE** (Uniform)	**CE**	**CE**
SID [Aguilar et al., 2020]	–	**KL** (Uniform)	**COS** (Uniform)	**CE**	**CE**
DistilBert [Sanh et al., 2020]	**COS**	–	–	**CE**	**CE**
TinyBert [Jiao et al., 2020]	**MSE**	**MSE** (Uniform)	**MSE** (Uniform)	–	–
XtremeDistil [Hu et al., 2020]	**SVD**	–	**KL** (Uniform)	**MSE**	**CE**
MiniLM [Hu et al., 2020]	–	**KL** (Last)	–	–	–

Table 2.1: Categorizing of previous works into different transformer parts distillation during pre-training. Notice that TinyBert does not utilize soft or hard targets during pre-training, only in their second distillation stage (task-distillation). Furthermore, we specify the loss function (**KL**: Kullback Leibler Loss; **MSE**: Mean-squared error; **COS**: Cosine Embedding Loss; **CE**: Cross-entropy loss; SVD: Singular Value Decomposition.

as Jiao et al. [2020] argue that their goal is to primarily learn the intermediate structures of BERT. Furthermore, they conduct experiments showing that output layer distillation does not bring additional improvements on downstream tasks.

Hidden Layer Representation Distillation. Using only logits to transfer cross-lingual knowledge from the teacher can be problematic as they only serve as one "task-specific" knowledge, in this case, masked language modeling. Romero et al. [2015] show that additional hints from the *intermediate layers* can improve the training process and the final performance of the student.

As one has to decide which layer of the teacher to distill from (typically, the student is smaller than the teacher), we have to define a mapping function. For this purpose, we assume that the teacher model has N Transformer layers and the student M Transformer layers, where $M \leq N$. The index 0 corresponds to the embedding layer, $M+1$ the index of the student's prediction layer, and $N+1$ the index of the teacher's prediction layer. The mapping function is then defined as

$$n = g(m), \qquad 0 < m \leq M, 0 < g(m) \leq N \tag{2.11}$$

between indices from student layers to teacher layers, where the teacher's $g(m)$-th layer is distilled into the m-th layer of the student model. Typically, there are four different mapping strategies:

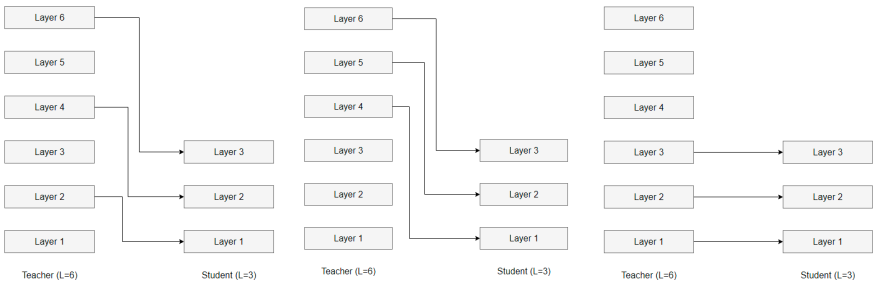


Figure 2.11: Given a teacher with 6 layers and a student with 3 layers, we visualize different mapping functions: *(Left)* Uniform, *(Middle)* Top and *(Right)* bottom.

(1) *Uniform*: We distill uniformly over the teacher layers. E.g. assume 12 layer teacher and 6 layer student, then the mapping function is $g(m) = m * 2$.

(2) *Top*: We distill from the top layers of the teacher: $g(m) = m + N - M$.

(3) *Bottom*: We distill from the bottom layers of the teacher: $g(m) = m$.

(4) *Last*: We distill only from the last layer of the teacher into the last layer of the student: $g(M) = N$.

For a visualization of different mapping functions, see Figure 2.11. We discuss which mapping functions in previous works were tested and which performed best.

PKD [Sun et al., 2019] focuses on extracting intermediate representations but only for the `[CLS]` token, reasoning that BERT's original implementation [Devlin et al., 2019] mostly uses the output from the last layer's `[CLS]` token to perform predictions for downstream tasks. The hope is that if the student can imitate the representation of `[CLS]` in the teacher's intermediate layers for any given input, the generalization ability can be similar to the teacher. The additional loss is then defined as the mean squared error between the normalized hidden states of the `[CLS]` token in all layers:

$$\mathcal{L}_{\text{MSECLS}} = \sum_{x \in \mathcal{X}} \sum_{m=1}^{M} \text{MSE}\left(\text{CLS}_m^{ST}(x), \text{CLS}_{g(m)}^{TE}(x)\right)^2,$$

where $\text{CLS}_m^{ST}(x) \in \mathbb{R}^d$ and $\text{CLS}_{g(m)}^{TE}(x) \in \mathbb{R}^d$ extracts the `[CLS]` token for the input x in the layer m for student and $g(m)$ teacher respectively. PKD was tested with the mapping function *top* and *uniform*, where the strategy *uniform* showed better performance. Similar to PKD, SID Aguilar et al. [2020] also uses hidden

vector representations for the [CLS] token to additionally align internal representations, but opted to use cosine similarity

$$\mathcal{L}_{\text{cos}CLS}(x) = 1 - cos\left(\text{CLS}_m^{ST}(x), \text{CLS}_{g(m)}^{TE}(x)\right)$$

with the *uniform* mapping strategy. The obvious problem with focusing on the [CLS] is that it only works for tasks that use the [CLS] for prediction such as tasks in the GLUE dataset [Wang et al., 2019].

Jiao et al. [2020] extended the idea of distilling hidden layer representations to student representations with any arbitrary number of hidden dimensions with their proposed model *TinyBert*. Let $H_m^{ST} \in \mathbb{R}^{L\times d'}$ and $H_{g(m)}^{TE} \in \mathbb{R}^{L\times d}$ refer to the hidden states in layer m of student and teacher networks respectively, L the sequence length and the scalars d', d denote the hidden sizes of the student and teacher models respectively. Instead of assuming that the hidden dimension is the same for the teacher and student $d' = d$, they allow for $d \neq d'$ in general. This allows for creating thinner models by using a smaller hidden dimension $d' < d$ than the teacher. They achieve this by introducing a learnable linear transformation matrix $W \in \mathbb{R}^{d'\times d}$ to transform the hidden states of the student network into the same space as the teacher network's states, i.e., $H_m^{ST}W \in \mathbb{R}^{L\times d}$. TinyBert transfers the knowledge which resides in the hidden layers output representation from *each token* in the sequence by minimizing the MSE across all tokens:

$$\mathcal{L}_{\text{Hid}_{\text{MSE}}}(\,\cdot\,; m) = \text{MSE}(H_m^{ST}W_m, H_{g(m)}^{TE}), \tag{2.12}$$

where the matrices $H_m^{ST} \in \mathbb{R}^{L\times d'}$ and $H_{g(m)}^{TE} \in \mathbb{R}^{L\times d}$ can have different hidden sizes $d \neq d'$. Furthermore, they experimented with all three mapping functions and concluded that *uniform* works best in their case. Notice that we can also align hidden representations with the cosine loss

$$\mathcal{L}_{\text{COS}hidn} = 1 - \cos(H_m^{ST}W_m, H_{g(m)}^{TE}), \tag{2.13}$$

with the matrices $H_m^{ST} \in \mathbb{R}^{L\times d'}$ and $H_{g(m)}^{TE} \in \mathbb{R}^{L\times d}$.

XtremeDistil also uses a projection to make all output spaces compatible. The projection however is non-linear and they use the KL-divergence:

$$\mathcal{L}_{\text{KL}hidn}(\,\cdot\,; m) = D_{KL}\left(\textit{Gelu}\left(H_m^{ST}W_m + b_m\right) \| H_{g(m)}^{TE}\right),$$

where W_m is the learnable projection matrix, b_m the learnable bias term and *Gelu* (Gaussian Error Linear Unit) is the non-linear projection function.

As we have seen, different mapping functions were used across previous work. It might not be useful to distill from intermediate representations as in Jiao et al.

[2020], Mukherjee and Awadallah [2020], Aguilar et al. [2020], especially when the student is a network of lower representational capacity. Yang et al. [2021] showed that using only the last hidden layer (*last* strategy) for matching representation results in the best performance, albeit experiments were conducted with Convolutional Neural Networks. To further align the last hidden representation, Yang et al. [2021] use the teacher's pre-trained Softmax Regression (SR) classifier to first pass the input x to the teacher network resulting in the output

$$\sigma(z_x^{TE}) = \sigma(W_{N+1}^{TE} H_N^{TE}),$$

where W_{N+1}^{TE} is the weight matrix of the softmax classifier head. Moreover, they feed the same input to the student to get the last hidden representation H_M^{ST} and input the representation to the teacher's SR classifier to obtain

$$\sigma(z_x^{ST}) = \sigma(W_{N+1}^{TE} H_M^{ST}).$$

They then optimize the loss

$$L_{SR} = -\sigma\left(W_{N+1}^{TE} H_M^{TE}\right) \log\left(\sigma(W_{N+1}^{TE} H_M^{ST})\right),$$

while keeping the classifier's parameter W_{N+1}^{TE} frozen. This loss is aligning the last hidden representations since if $\sigma(z_x^{TE}) = \sigma(z_x^{ST})$ (which the loss is optimizing to) then this implies that $H_N^{TE} = H_M^{ST}$.

Attention Distillation. Clark et al. [2019a] found that the attention weights of the transformer model BERT can capture linguistic knowledge, which can be used to transfer linguistic knowledge into our student. Motivated by this, Jiao et al. [2020] additionally uses attention distillation for TinyBert, which uses the mean squared error to align the teacher and students matrices of the multi-head attention. Let the matrices $Q, K, V \in \mathbb{R}^{L \times d_k}$ denote the queries, keys and values, where d_k is the dimension of keys. As already discussed, the attention $\text{Attention}(Q, K, V)$ (2.4) is calculated with

$$A = \frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{L \times L}, \quad \text{Attention}(Q, K, V) = \sigma(A)V \in \mathbb{R}^{L \times d_k}, \quad (2.14)$$

where d_k acts as a scaling factor. We then calculate the attention loss with

$$\mathcal{L}_{\text{MSEAtt}} = \frac{1}{h} \sum_{i=1}^{h} \text{MSE}(A_i^{ST}, A_i^{TE}), \quad (2.15)$$

where h is the number of attention heads, $A_i \in \mathbb{R}^{L \times L}$ refers to the attention matrix before applying the softmax, the index i corresponds to the i-th head of teacher or

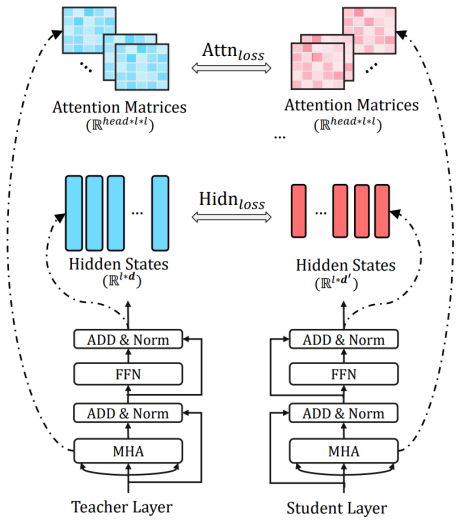


Figure 2.12: Visualization of a attention and intermediate hidden representation distillation. Taken from [Jiao et al., 2020].

student, L is the input text length, and MSE$(\cdot)$ means the mean squared error loss function. They argue that using A instead of Attention(Q, K, V) (2.4) show faster convergence rate and better performances. We visualize the TinyBert attention and intermediate hidden representation distillation in Figure 2.12.

Nonetheless Aguilar et al. [2020] make use of the attention matrix Attention(Q, K, V) (2.4) and chose the KL-divergence loss. For a given head in a transformer, the loss can be formulated as

$$\mathcal{L}_{\text{KLAtt}} = \frac{1}{L}\sum_{i}^{L} \texttt{Att}(Q^{TE}, K^{TE}, V^{TE})_i \log\left(\frac{\texttt{Att}(Q^{TE}, K^{TE}, V^{TE})_i}{\texttt{Att}(Q^{ST}, K^{ST}, V^{ST})_i}\right),$$

where $\texttt{Att}(Q, K, V)_i = \text{Attention}(Q, K, V)_i \in \mathbb{R}^{d_k}$ describe the i-th row of the attention probability matrix.

MiniLM [Wang et al., 2020] additionally aligns the relation between values in the self-attention module, which is calculated via the multi-head scaled dot-product between values V^{ST} and V^{TE}.

Embedding Distillation. Multiple lines of work indicate that the embedding layer is important for a successful cross-lingual representation [Pires et al., 2019, Wu and Dredze, 2019, Dufter and Schütze, 2021]. Therefore, it is also important to distill from our multilingual teacher's embedding layer to our students.

DistilBert uses the cosine embedding loss to align the embeddings of teacher and student:

$$\mathcal{L}_{\text{COS}embed} = 1 - \cos(E_m^{ST}, E^{TE}).$$

where the matrices E^{ST} and E^{TE} refer to the embeddings of student and teacher networks, respectively. Instead of using the cosine loss, TinyBert uses the MSE:

$$\mathcal{L}_{\text{Emb}_{\text{MSE}}ed} = \text{MSE}(E^{ST}W_e, E^{TE}). \tag{2.16}$$

Again, TinyBert allows for a mismatch between dimensions of E^{ST} and E^{TE} by using a learnable linear transformation W_e.

XtremeDistil [Mukherjee and Awadallah, 2020] uses Singular Value Decomposition (SVD) to project the word embeddings of the teacher to a lower-dimensional space for the student since they use the same WordPiece vocabulary.

2.2.4 Distillation Setup Strategies

This thesis introduces a novel distillation setup to induce aligned monolingual students, which is why we review different distillation setup strategies for transformers in this section. We restrict our literature review to a "fixed" teacher at the distillation time and not an, e.g., jointly trained teacher-student setup, such as in Jin et al. [2019]. Furthermore, we focus on the general distillation stage, where we only perform distillation on the Masked Language Modeling objective. Finally, the task is to distill knowledge from the multilingual teacher(s) to our student while MLM pre-training on multiple monolingual text corpora, i.e., multilingual corpus, obtaining a general-purpose student.

One Teacher, One student. In this setup, only one teacher exists and is distilled into one student. Reimers and Gurevych [2020] use parallel data to facilitate strong cross-lingual sentence representations by training the student model such that (1) identical sentences in different languages are close and (2) the original source language from the teacher model SBERT [Reimers and Gurevych, 2019] are adopted and transferred to other languages. They do so by minimizing the mean squared error of (1) the source sentence embedding of the teacher with the target sentence embedding using parallel data and (2) the source sentence embedding of the teacher and the student, see Figure 2.13 for a visualization.

Furthermore, all discussed monolingual distillation approaches can be extended to create a multilingual student straightforwardly: Distill a multilingual teacher into a student during the MLM task on a multilingual corpus. Since the teacher is already multilingual and the representations aligned [Pires et al., 2019, Wu and Dredze, 2019], the cross-lingual knowledge can then be distilled into one student.

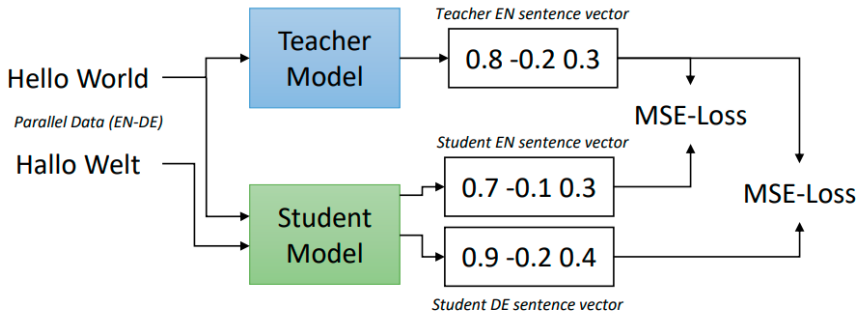


Figure 2.13: Visualization of the distillation strategy of Reimers and Gurevych [2020]. Given parallel data (here: English and German), the student model is trained such that the sentence embeddings for the English and German sentences are close to the teacher English sentence vector. Adopted from Reimers and Gurevych [2020].

Consequently, every monolingual distillation approach can be directly applied, e.g. PKD [Sun et al., 2019], DistilBert [Sanh et al., 2020] or TinyBert [Jiao et al., 2020]. We can generalize the distillation loss as a *convex combination* or a *linear combination* of all chosen loss functions. E.g. DistilBert uses a convex combination of the masked language modeling loss $\mathcal{L}_{\text{MLM}}$ (2.10), the Hinton Loss $\mathcal{L}_H$ (2.9) and the cosine embedding loss $\mathcal{L}_{\text{COS}embed}$ (2.13):

$$\mathcal{L}_{\text{DB}} = \alpha \cdot \mathcal{L}_H + \beta \cdot \mathcal{L}_{MLM} + \gamma \cdot \mathcal{L}_{\text{COS}embed}(\,\cdot\,;\,\cdot\,;M),$$

where M is the index of the last hidden layer, $\alpha, \beta, \gamma \in [0, 1]$ with $\alpha + \beta + \gamma = 1$.

In this work, we will briefly discuss and conduct experiments with bilingual students (see Section 4.4.4 and Section 4.5.3)

Multiple Teacher, One Student. Language-specific language models may perform better given a sizable pre-training data volume than multilingual teachers in their respective language but are, in turn, monolingual and do not use any positive language transfer. Distilling multiple task-agnostic language-specific teachers into one task-agnostic student can help the student be competitive or outperform the individual language-specific LMs while still being multilingual. Khanuja et al. [2021] merges multiple monolingual or multilingual pre-trained LMs into a single task-agnostic multilingual student using task-agnostic Knowledge Distillation, which they call *MergeDistill*. The difficulty is that each LM can have its own vocabulary. Khanuja et al. [2021] use the union of all teacher LM vocabularies for the

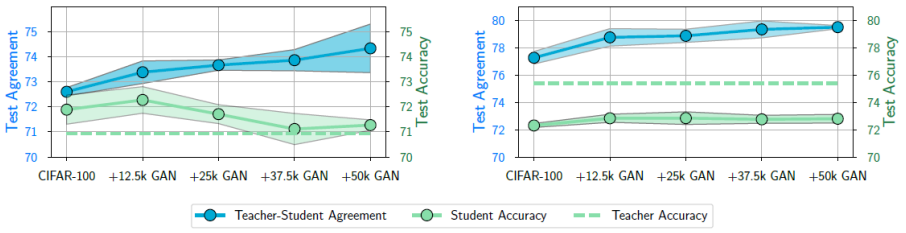
student vocabulary. They use a vocab mapping step *teacher* $\rightarrow$ *student*, converting each teacher token index to its corresponding student token index. They first tokenize and predict for each language using their respective teacher LM and get the top-k logits for each masked word. For distillation, they then use the Hinton Loss $\mathcal{L}_H$ (2.9) and MLM loss $\mathcal{L}_{\text{MLM}}$ (2.10). Interestingly, their experiments show that due to the shared multilingual representations, the student is able to perform in a zero-shot manner on related languages that the teacher does not cover.

One Teacher, Multiple Students. In this setup, we have one multilingual teacher and want to distill into multiple mono-, bi- or multilingual students, for which the representations for each language are aligned across students. In this work, we will focus on distilling into multiple monolingual students. To the best of our knowledge, this is the first work that investigates distilling from a multilingual teacher into monolingual students sharing a representation space.

2.2.5 Challenges

The trend towards bigger and bigger models in NLP is fueled by the high generalization power of the learned representations. Smaller models lack the inductive biases to learn these representations from the training data alone but may have the capacity to represent these solutions [Ba and Caruana, 2013, Stanton et al., 2021]. We discussed several methods such as DistilBert [Sanh et al., 2020] and TinyBert [Jiao et al., 2020] that achieve similar performances on some downstream tasks as the teacher with just a fraction of the total parameter number. In the previous sections, we discussed different Knowledge Distillation strategies to induce knowledge into the student, e.g., the effects of distilling different transformer components into the student. However, we highlight two open challenges in regards to Knowledge Distillation in general (Fidelity vs. Generalization) and our thesis (KD for Monolingual Students).

Fidelity vs. Generalization. Recently, Stanton et al. [2021] show that while Knowledge Distillation can improve the generalization abilities of students, there often remains a low fidelity, i.e., the ability of a student to match a teacher's predictions. Previous works [Furlanello et al., 2018, Mobahi et al., 2020] already show that in self-distillation, the student can improve generalization. This can only happen by virtue of failing at the distillation procedure: The student fails to match the teacher, see Figure 2.14 (a). These experiments, however, hold true for students that have the *same model capacity* as the teacher. Often there is a significant disparity in generalization between large teacher models and smaller students. Importantly, Stanton et al. [2021] then show that for these larger teacher models, improvements in fidelity translate into improvements in generalization, see Figure



(a) Self-distillation (b) Ensemble distillation

Figure 2.14: The effect of enlarging the CIFAR-100 distillation dataset with GAN-generated samples. The shaded region corresponds to $\mu \pm \sigma$, estimated over three trials. (a) The teacher and student have the same model capacity. Student fidelity increases as the dataset grows, but the test accuracy decreases. (b) The teacher has a larger model capacity than the student. Student fidelity again increases when the dataset grows, but the test accuracy now also slightly increases. Figure taken from Stanton et al. [2021].

2.14 (b).

KD for Monolingual Students. As this is the first work that explores distilling multilingual encoders into monolingual components (to the best of our knowledge), the question remains which parts of the teacher are important to distill from to improve (1) alignment and (2) cross-lingual downstream task performance. Another open question is whether sharing between students and weight initialization from the teacher improves (1) and (2). Finally, as we have multiple students, we can not utilize the default approach to solve cross-lingual downstream tasks: Fine-tuning one multilingual model in the source language and evaluating/train with the same model in the target language. We must explore fine-tuning strategies for multiple (monolingual) students for cross-lingual downstream tasks. In the following subsection 2.3 we review methods that can potentially be used.

2.3 Parameter-Efficient Fine-Tuning

In this section, we will briefly cover Adapters (Section 2.3.1) and Sparse-Fine-Tuning (Section 2.3.2), specifically Diff-Pruning and BitFit. These techniques will help us fine-tune our monolingual students on cross-lingual downstream tasks in a zero- and few-shot manner (Section 3.3).

Motivation. The standard approach to solving a new task with a pre-trained transformer is by adding a task-head (e.g., a linear classification layer) on top of the pre-trained transformer (encoder) and minimizing the task loss end-to-end. However,

this approach results in a completely new unique, large model making it harder to track what significantly changed during fine-tuning and therefore making it also hard to transfer acquired task-specific knowledge (*modularity*). The latter is important in our monolingual setup as we want to transfer the acquired task-specific knowledge by our source student into our target student, see Section 3.3 for more details. Ideally, transferring the acquired task-specific knowledge still matches the results of fully fine-tuning one model.

The first work that we explore is Adapters [Rebuffi et al., 2017, Houlsby et al., 2019] which inserts a small subset of trainable task-specific parameters between layers of a model and only changes these during fine-tuning, keeping the original parameters frozen. We then discuss sparse fine-tuning, which only changes a subset of the pre-trained model parameters. Specifically, we consider Diff-Pruning [Guo et al., 2020], which adds a sparse, task-specific difference-vector to the original parameters, and BitFit [Zaken et al., 2021], which enforces sparseness by only fine-tuning the bias terms and the classification layer.

2.3.1 Adapters

Adapters were initially proposed for computer vision to adapt to multiple domains [Rebuffi et al., 2017] but were then used as an alternative lightweight training strategy for pre-trained transformers in NLP [Houlsby et al., 2019]. Adapters introduce additional parameters to a pre-trained transformer, usually small, bottleneck feed-forward networks inserted at each transformer layer. Adapters enable us to keep the pre-trained parameters of the model fixed and only fine-tune the newly introduced parameters on either a new task [Houlsby et al., 2019, Stickland and Murray, 2019, Pfeiffer et al., 2021] or new domains [Bapna et al., 2019]. Adapters perform either on par or slightly below full fine-tuning [Houlsby et al., 2019, Stickland and Murray, 2019, Pfeiffer et al., 2021]. Importantly, adapters learn task-specific representations which are compatible with subsequent transformer layers [Pfeiffer et al., 2020a].

Placement & Architecture. Most work insert adapters at each layer of the transformer model, the architecture and placement of adapters are, however, non-trivial: Houlsby et al. [2019] experiment with different adapter architectures and empirically validated that using a two-layer feed-forward neural network with a bottleneck worked well, see Figure 2.15 (*Right*). This simple down- and up-projection with a non-linearity has become the common adapter architecture. The placement and number of adapters within each transformer block are still debated. Houlsby et al. [2019] place the adapter at two positions: One after the multi-head attention and one after the feed-forward layers. Stickland and Murray [2019] just use one adapter after the feed-forward layers, which Bapna et al. [2019] adopted and

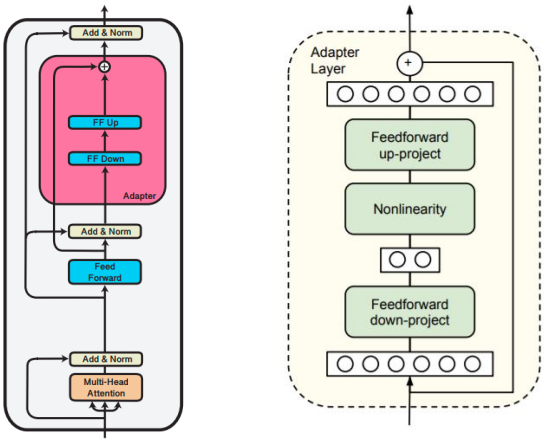


Figure 2.15: (*Left*) Proposed placement of the adapter within transformer block: After feed-forward neural network Pfeiffer et al. [2021]. (*Right*) Proposed adapter architecture [Houlsby et al., 2019, Pfeiffer et al., 2021]. Images are taken from Pfeiffer et al. [2021] and Houlsby et al. [2019].

.

extends by including a layer norm [Ba et al., 2016] after the adapter. Pfeiffer et al. [2021] test out different adapter positions and adapter architectures jointly and came to the conclusion to use the same adapter architecture as Houlsby et al. [2019] but only places the adapter after the feed-forward neural network, see Figure 2.15 (*Left*).

Modularity of Representations. One important property of Adapters is that they learn task-specific knowledge within each adapter component. The reason is that they are placed within a frozen transformer block layer, forcing the adapters to learn an output representation compatible with the subsequent layer of the transformer model. This results in modular adapters, meaning that they can be either stacked on top of each other or replaced dynamically [Pfeiffer et al., 2020b]. This modularity of adapters can be used to fine-tune multiple monolingual aligned students on cross-lingual downstream tasks: Instead of fine-tuning the whole student model on the source language, we insert and fine-tune the adapters, which then can be inserted into the monolingual student corresponding to the target language (see Section 3.3.3).

2.3.2 Sparse Fine-Tuning

Sparse fine-tuning (SFT) only fine-tunes a small subset of the original pre-trained model parameters at each step, effectively fine-tuning in a parameter efficient way. The fine-tuning procedure can be describes as

$$\Theta_{\text{task}} = \Theta_{\text{pretrained}} + \delta_{\text{task}}$$

where Θ_{task} is the task-specific parameterization of the model after fine-tuning, $\Theta_{\text{pretrained}}$ is the set of pretrained parameters which is fixed and δ_{task} is called the task-specific diff vector. We call the procedure sparse if δ_{task} is sparse. As we only have to store the nonzero positions and weights of the diff vector, the method is parameter efficient. Method generally differ in the calculation of δ_{task} and its induced sparseness.

Methods. Guo et al. [2020] introduce Diff Pruning, which determines δ_{task} by adaptively pruning the diff vector during training. To induce this sparseness, they utilize a differentiable approximation of the L_0-norm penalty [Louizos et al., 2017]. Zaken et al. [2021] induce sparseness by only allowing non-zero differences in the bias parameters (and the classification layer) of the transformer model. The method, called BitFit, and Diff-Pruning, both can match the performance of fine-tuned baselines on the GLUE benchmark [Guo et al., 2020, Zaken et al., 2021]. Ansell et al. [2021] learn sparse, real-valued masks based on a simple variant of the Lottery Ticket Hypothesis [Frankle and Carbin, 2018]: First, they fine-tune a pre-trained model for a specific task or language, then select the subset of parameters that change the most which correspond to the non-zero values of the diff vector δ_{task}. Then, the authors set the model to its original pre-trained initialization and re-tune the model again by only fine-tuning the selected subset of parameters. The diff vector δ_{task} is therefore sparse.

Comparison to Adapters. In contrast to Adapters, Sparse fine-tuning (SFT) does not modify the architecture of the model but restricts its fine-tuning to a subset of model parameters [Guo et al., 2020, Zaken et al., 2021]. As a result, SFT is much more expressive, as they are not constricted to just modifying the output of Transformer layers with shallow MLP but can directly modify the pre-trained model's embedding and attention layers [Ansell et al., 2021]. Similar to Adapters, Ansell et al. [2021] show that their sparse fine-tuning technique has the same concept of modality found in Adapters [Pfeiffer et al., 2020b]. Again this modularity can be used in our monolingual setup to fine-tune on cross-lingual downstream tasks.

Chapter 3

Monolingual Setup & Students

In this main chapter, we present our two approaches: `MonoAlignment` and `MonoShot`. The former is a method to improve the cross-lingual alignment, while the latter tries to improve cross-lingual representations. During the masked language modeling task (monolingual setup), both methods distill multilingual encoders into language-specific components.

Specifically, in Section 3.1 we first motivate why our monolingual setup is advantageous (Section 3.1.2) and specify what obstacles one has to overcome (Section 3.1.3). In Section 3.2 we then go into detail how the training procedure of the monolingual setup is constructed (Section 3.2.1), which Knowledge Distillation losses we test out (Section 3.2.2), what to share across students (Section 3.2.3) and how to initialize the students (Section 3.2.4). In Section 3.3, we then investigate how to effectively leverage cross-lingual knowledge in the monolingual setup during zero-shot fine-tuning. Finally, in Section 3.4, we explain our proposed methods `MonoAlignment` and `MonoShot`.

3.1 Motivation & Problem Formulation

As already in Section 2.1.4 described, the current state of the art to induce cross-lingual representations is by using multilingual pre-trained encoders [Pires et al., 2019, Wu and Dredze, 2019, Conneau et al., 2020]. This thesis aims to improve these general-purpose cross-lingual representations by distilling cross-lingual knowledge into monolingual components. However, we first must understand how the strength of general-purpose cross-lingual representations can be evaluated (Section 3.1.1). We then discuss current problems in state-of-the-art cross-lingual representations and how we try to solve these issues with our approach (Section 3.1.2).

3.1.1 Measuring Cross-Lingual Representations

To capture the model's cross-lingual generalization ability, we distinguish between two cross-lingual abilities: The cross-lingual alignment of our students and the cross-lingual downstream task performance[1].

Cross-Lingual Alignment. One important way to probe a cross-lingual representation is by finding translations, either on the word-level [Conneau et al., 2017, Glavas et al., 2019, Vulić et al., 2019, Guzman et al., 2019] or sentence level [Artetxe and Schwenk, 2019]. The idea is that translations are close to each other in the cross-lingual space, which can be evaluated by finding the nearest neighbor using cosine similarity and calculating the error rate. We use the the `Tatoeba` dataset (Retrieval task) [Artetxe and Schwenk, 2019] to evaluate the cross-lingual alignment of our students.

Cross-Lingual Downstream Tasks. Previous works show that only evaluating translations has a weak correlation with other downstream tasks such as language understanding tasks [Glavas et al., 2019]. For instance, Mohammad et al. [2016], Smith et al. [2016] show that translations can alter the sentiment of a text, effectively showing that translations can not capture cultural differences resulting in a degradation of classification performances. Furthermore, later in our thesis, we find the same poor correlation between cross-lingual alignment and cross-lingual downstream task performance (see Section 4.5.1). Therefore we need more than translations to evaluate the model's cross-lingual generalization ability. One standard benchmark that has been established recently is the XTREME dataset that covers classification, structured prediction (Struct. Pred.), Question & Answering (QA), and retrieval tasks, each requiring a different level of reasoning [Hu et al., 2020]. As not all tasks have validation and test sets for our target languages Turkish and Swahili, we choose `XNLI` (classification) [Conneau et al., 2018], `Wikiann` dataset as a NER task (Struct. Pred.) [Pan et al., 2017] and `XCOPA` dataset (QA) [Ponti et al., 2020], see Section 4.1 for more details on each task.

3.1.2 Motivation

Multilingual LM. As we already discussed the challenges of multilingual transformers in Section 2.1.6, we shortly mention them here again: Hu et al. [2020] show that the multilingual LMs have a significant cross-lingual transfer gap[2], especially in low-resource and distant languages [Wu and Dredze, 2019, Lauscher

[1] In our thesis, we do not categorize a retrieval task as a downstream task as there is no further fine-tuning process.

[2] Measured in the difference between performance on the English test set and the average performance on the other languages.

et al., 2020]. Because of the curse of multilinguality, positive cross-lingual transfer and capacity dilution has to be carefully traded off against each other. Increasing model capacity can help alleviate the curse of multilinguality. However, it is not practical as the memory and computational power are already at a high level.

Language-Specific LM. To avoid the issues caused by the capacity dilution and the curse of multilinguality, practitioners often prefer to have a language-specific LM that works well on a subset of languages that they are interested in, e.g., AraBERT [Antoun et al., 2020], CamemBERT [Martin et al., 2020] and Finnish Bert [Virtanen et al., 2019]. However, these language-specific models are only monolingual, not aligned with other languages, and in particular, do not use any positive cross-lingual transfer. Therefore creating these language-specific LM for low-resource languages without any positive cross-lingual transfer is problematic as we do not have enough monolingual text. Furthermore, they are not suitable for cross-lingual downstream tasks. We have abundant labeled data in a high-resource language but none in the target language (zero-shot scenario).

Our Approach. To obtain the gained positive cross-lingual transfer of the multilingual LM *and* to alleviate the issue of capacity dilution, we propose to distill the multilingual LM into language-specific students (see Section 3.2 for more details). We hope to *reduce the capacity dilution* by allocating language-specific parameters (students), but *still gaining from positive cross-lingual transfer* of the multilingual LM by distillation.

3.1.3 Problem Formulation

We only focus on the state-of-the-art transformer architecture, so we design distillation methods specifically for transformer models. To benefit from cross-lingual Knowledge Distillation and to assign language-specific students, there are four primary considerations: (1) The distillation loss $\mathcal{L}_{\text{model}}$, (2) what is shared across students, (3) the initialization of the students embedding and layers and (4) the zero-shot fine-tuning strategies.

Distillation Loss. How can we construct the distillation loss $\mathcal{L}_{\text{model}}$ such that we benefit from the cross-lingual knowledge of the teacher the most? The question is which teacher components to distill from and what kind of loss functions we have to use. We define the general distillation loss $\mathcal{L}_{\text{model}}$ as follows: Given a teacher model with N transformer layers and a student with M transformer layers. We assume that student transformers are either smaller or equal in size of the teacher $M \leq N$. The index 0 corresponds to the embedding layer, $M+1$ the index of the student's prediction layer and $N+1$ the index of the teacher's prediction layer. If we distill from the teacher's layers, we have to choose which layers of the teacher

we distill to which layer of the student. For this purpose, we already defined the mapping function $n = g(m)$ (2.11) in Section 2.2.3. As we will introduce two more mapping functions, we redefine this mapping function as $g(m) = g_{\text{loss}}(m)$ for clarification. Formally, we then define the general distillation loss as

$$\mathcal{L}_{\text{model}} = \sum_{x \in \mathcal{X}} \sum_{m=0}^{M+1} \lambda_m \mathcal{L}_{m-\text{layer}} \left(f_m^S(x), f_{g_{\text{loss}}(m)}^T(x) \right),$$

where $\mathcal{L}_{m-\text{layer}}$ corresponds to the loss function of the mth layer (including embedding and prediction layer), $f_m(x)$ the behavior function induced from the m-th layers and λ_m is the importance scalar (hyperparameter) of the m-th layer's distillation. In Section 2.2.3, we already gave an extensive overview of different component distillation loss functions. In Section 3.2.2, we give an overview of all constructed distillation losses.

Sharing across Students. Another aspect of our setup is if there is a need to share components across students. Given our student models with M number of layers, we define a subset of student layer indices $g_{share} \subseteq \{0, 1, ..., M\}$ that indicates which layers are shared across students. E.g. $g_{share} = \{0\}$ corresponds to sharing the embedding layer across all students. Sharing components across students can be an important factor in having a stronger alignment between representation spaces of student models or stronger cross-lingual downstream task performance. We give an overview of sharing strategies in Section 3.2.3. Finally, we analyze their impact on the cross-lingual downstream task performance in Section 4.4.3 and on the cross-lingual alignment in Section 4.5.2.

Initialization of Students. Another option is the initialization of our students. We either randomly initialize our students or initialize the weights from the teacher (assuming that the student has the same architecture). Typically, our student is smaller than the teacher, and therefore, we need another mapping function $n = g_{\text{init}}(m)$ between indices from student layers to teacher layers, where the $g_{\text{init}}(m)$-th layer of the teacher is initialized for the m-th layer of the student model. As initialization can give us a good starting point to find a strong cross-lingual representation, we discuss two initialization strategies in Section 3.2.4. Subsequently, we study their effect on the cross-lingual downstream task performance in Section 4.4.3 and on the cross-lingual alignment in Section 4.5.2. .

Zero-Shot Fine-Tuning Strategies. In cross-lingual downstream tasks, the default setting to evaluate multilingual models is by fine-tuning the model on the source language and evaluating the same fine-tuned model on the target language. As we have multiple monolingual students, we can not utilize the same setting. In Section 3.3, we discuss various fine-tuning strategies.

3.2 General-Purpose KD into Monolingual Components

This section will explore how our approach uses different Knowledge Distillation methods to distill from a multilingual teacher into multiple monolingual (language-specific) students. The goal is to distill the cross-lingual knowledge from the teacher such that the cross-lingual representation space across students is either more aligned or lends itself for cross-lingual downstream tasks (or both).

3.2.1 Monolingual Setup

This section discusses the distillation setup for our monolingual student distillation. Our approach aims to distill a strong cross-lingual representation from a multilingual encoder into monolingual students. Additionally, we want our students to be general-purpose, which is why we distill the knowledge during the pre-training task: Masked Language Modeling.

Teacher & Student Models. As we focus on multilingual transformers, we choose the state-of-the-art model XLM-R [Conneau et al., 2020] as our teacher. XLM-R has proven to have a very strong cross-lingual representation across many languages but still struggles in low-resource languages (see Section 2.1.6). Our student has the same architecture and vocabulary size as the teacher. We only vary the number of transformer layers of the student.

Training Procedure. We use Knowledge Distillation during the masked language modeling task and utilize the same masking dynamic as XLM-R (see Section 2.1.5). As we want to create language-specific students, we only distill instances into a student from the desired language(s). E.g., if we want to create a Turkish student, we only distill Turkish instances into the Turkish student. Specifically, at each distillation step, given students with a total of K languages, we optimize our students as follows:

1 We create batches that only contain samples from one language for each language. Each of the K batches are equal in size.

2. All created batches are then fed into the teacher, and the teacher's output is saved for later use in the distillation process.

3. We then loop through all batches and only feed each batch to the respective language-specific student. Finally, we optimize the chosen distillation loss $\mathcal{L}_{\text{model}}$ for each student-batch pair sequentially.

Algorithm 1 shows the pseudocode of one distillation step in the training procedure.

Algorithm 1 Distillation Step: Training Procedure for CLKD

Require: Teacher Model T. $S_1, S_2, ..., S_P$ Student Models, each Student has set of target languages $L_1 = \{...\}, ..., L_P = \{...\}$.

```
L = L_1 ∪ ... ∪ L_P                                  ▷ Set of Total Languages
N ← |L|                                              ▷ Number Languages
B_1, ..., B_N ← SampleBatchForEachLanguage(L)        ▷ Batch for each Language
for lang in L do                                     ▷ Loop through all Target Languages
    B_current ← GetBatch(B_1, ..., B_N, lang)        ▷ Batch for current Language
    Out_T ← GetOutput(T, B_current)
    for S_current in StudentsForLanguage(lang) do    ▷ Loop through Students
                                                     ▷ Corresponding to Language
        Out_S ← GetOutput(S_current, B_current)
        Loss ← CalculateLoss(Out_T, Out_S, B_current)
        S_current ← Optimize(Loss, S_current)
    end for
end for
```

3.2.2 Distillation Loss

In this section, we study various versions of the general distillation loss $\mathcal{L}_{\text{model}}$ (3.1.3) to distill multilingual encoders into monolingual components. The alignment across students only stems from the cross-lingual ability of the teacher that we are distilling from. We hope to distill the shared and aligned representation space from the multilingual teacher to our students without any explicit alignment. Furthermore, we then hope to achieve the same or even better cross-lingual downstream performance than the teacher. We already discussed and organized previous distillation losses in Section 2.2.3. We summarize all tested distillation losses in Table 3.1.

Hinton. We first test out distilling from only the last output layer of the teacher. We therefore use the same loss as in Hinton et al. [2015], a weighted average of the (scaled) Hinton loss (2.9) and the MLM loss (2.10) with the correct token:

$$\mathcal{L}_{\text{Hinton}}^{\alpha,\beta} = \alpha \cdot \mathcal{L}_{\text{MLM}} + \beta \cdot \mathcal{L}_{\text{H}},$$

where $\alpha, \beta \in [0, 1]$ and $\alpha + \beta = 1$.

Hinton+Hid$_{\text{MSE}}$. To incorporate intermediate layer representations, we match the intermediate representations of the teacher and student. To match these representations, use the mean squared error $\mathcal{L}_{\text{Hid}_{\text{MSE}}}$. The final loss function is a linear combination of the masked language modeling loss $\mathcal{L}_{\text{MLM}}$ (2.10), the Hinton Loss $\mathcal{L}_H$

Distillation Name	Emb	Hidden Layer Distill		Output Layer	
		Attention	Hidden Repr.	Soft	Hard
Hinton	–	–	–	KL	CE
Hinton+Hid$_{\text{MSE}}$ (Uniform)	–	–	MSE (Uniform)	KL	CE
Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$ (Uniform)	–	MSE (Uniform)	MSE (Uniform)	KL	CE
Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Uniform)=MonoShot	MSE	MSE (Uniform)	MSE (Uniform)	KL	CE
Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Uniform)	MSE	MSE (Uniform)	MSE (Uniform)	–	–
Hinton+Hid$_{\text{COS}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Uniform)	MSE	MSE (Uniform)	COS (Uniform)	KL	CE
Hinton+Hid$_{\text{COS}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Bottom)	MSE	MSE (Bottom)	COS: Bottom	KL	CE
Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Top)	MSE	MSE (Uniform)	MSE (Uniform)	KL	CE
Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Bottom)	MSE	MSE (Top)	MSE (Top)	KL	CE
Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ (Last)=MonoAlignment	MSE	MSE (Bottom)	MSE (Bottom)	KL	CE

Table 3.1: Loss Functions: We categorize the possible distillations of different parts of the transformer into: Embedding (Emb), Hidden Layer, and Output Layer, see Section 2.2.3 for more details. The distillation name is constructed from two parts: The first part is the distillation loss name (see Section 3.2.2), and the second part, which is denoted in brackets, is the mapping strategy (see Section 2.2.3). The top part of the table denotes the different distillation losses that we study based on the uniform mapping strategy. Additionally, we study the cosine loss function to align hidden representations. The bottom part denotes the study of different mapping strategies based on the Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$ loss.

(2.9) and aligning the hidden representation with the mean squared error $\mathcal{L}_{\text{Hid}_{\text{MSE}}}$ (2.12)

$$\mathcal{L}^{\alpha,\beta,\gamma}_{\text{Hinton+Hid}_{\text{MSE}}} = \mathcal{L}^{\alpha,\beta}_{\text{Hinton}} + \gamma \cdot \sum_{m=1}^{M} \mathcal{L}_{\text{Hid}_{\text{MSE}}}(\,\cdot\,; m), \tag{3.1}$$

where m is the index of the student layer, $\alpha, \beta, \gamma \in [0, 1]$ with $\alpha + \beta + \gamma = 1$. Notice that the loss function $\mathcal{L}^{\alpha,\beta,\gamma}_{\text{Hinton+Hid}_{\text{MSE}}}$ (3.1) is similar to the loss function of PKD [Sun et al., 2019], except that in this loss we consider all tokens, not just the [CLS] token.

Hinton+Hid$_{\text{MSE}}$+ATT$_{\text{MSE}}$. Furthermore, we test the gain from incorporating the mean-squared error attention loss $\mathcal{L}_{\text{ATT}_{\text{MSE}}}$ (2.14). The full loss is then calculated as

$$\mathcal{L}^{\alpha,\beta,\gamma,\delta}_{\text{Hinton+Hid}_{\text{MSE}}+\text{ATT}_{\text{MSE}}} = \mathcal{L}^{\alpha,\beta,\gamma}_{\text{Hinton+Hid}_{\text{MSE}}} + \delta \sum_{m=1}^{M} \mathcal{L}_{\text{ATT}_{\text{MSE}}}(\,\cdot\,; m) \tag{3.2}$$

where m is the index of the student layer, $\alpha, \beta, \gamma, \delta \in [0, 1]$ with $\alpha + \beta + \gamma + \delta = 1$. Again, notice that the $\mathcal{L}_{\text{Hinton+Hid}_{\text{MSE}}+\text{ATT}_{\text{MSE}}}$ loss (3.2) is similar to SID's loss [Aguilar et al., 2020], except that we use the mean squared error and consider all tokens, not just the [CLS] token.

Hinton+Hid$_{\text{MSE/COS}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$. We additionally use the mean-squared embedding loss $\mathcal{L}_{\text{Emb}_{\text{MSE}}}$. Using the equations (2.9), (2.12), (2.15) and (2.16), we can define the full loss as

$$\mathcal{L}^{\alpha,\beta,\gamma,\delta,\epsilon}_{\text{Hinton+Hid}_{\text{MSE}}\text{+ATT}_{\text{MSE}}\text{+Emb}_{\text{MSE}}} = \mathcal{L}^{\alpha,\beta,\gamma,\delta}_{\text{Hinton+Hid}_{\text{MSE}}\text{+ATT}_{\text{MSE}}} + \epsilon \cdot \mathcal{L}_{\text{Emb}_{\text{MSE}}}(\,\cdot\,)$$

with $\alpha, \beta, \gamma, \delta, \epsilon \in [0, 1]$ with $\alpha + \beta + \gamma + \delta + \epsilon = 1$. Furthermore, we study the cosine similarity to align hidden representations:

$$\mathcal{L}^{\alpha,\beta,\gamma,\delta,\epsilon}_{\text{Hinton+Hid}_{\text{COS}}\text{+ATT}_{\text{MSE}}\text{+Emb}_{\text{MSE}}} = \mathcal{L}^{\alpha,\beta,\gamma,\delta}_{\text{Hinton+Hid}_{\text{COS}}\text{+ATT}_{\text{MSE}}} + \epsilon \cdot \mathcal{L}_{\text{Emb}_{\text{MSE}}}(\,\cdot\,).$$

This loss is similar to the one used during task-distillation in TinyBert [Jiao et al., 2020].

Hid$_{\text{MSE/COS}}$+ATT$_{\text{MSE}}$+Emb$_{\text{MSE}}$. We additionally study the TinyBert loss function used during the general distillation stage (distilling on MLM task). The loss consists of $\mathcal{L}_{\text{Hid}_{\text{MSE}}}$ (2.12), $\mathcal{L}_{\text{ATT}_{\text{MSE}}}$ (2.15) and $\mathcal{L}_{\text{Emb}_{\text{MSE}}}$ (2.16). Consequently, the full loss can be defined as

$$\mathcal{L}^{\alpha,\beta,\gamma}_{\text{Hid}_{\text{MSE}}\text{+ATT}_{\text{MSE}}\text{+Emb}_{\text{MSE}}} = \alpha \cdot \sum_{m=1}^{M} \mathcal{L}_{\text{Hid}_{\text{MSE}}}(\,\cdot\,; m) + \beta \cdot \sum_{m=1}^{M} \mathcal{L}_{\text{ATT}_{\text{MSE}}}(\,\cdot\,; m) + \gamma \cdot \mathcal{L}_{\text{Emb}_{\text{MSE}}}(\cdot)$$

with $\alpha, \beta, \gamma \in [0, 1]$ with $\alpha + \beta + \gamma = 1$.

We study the effect of using different loss functions on the cross-lingual downstream task performance in Section 4.4.2 and on the cross-lingual alignment in Section 4.5.1. The second part of the distillation loss is from which layers of the teacher we distill. As already listed in Section 2.2.3, we study four different mappings: Uniform, Top, Bottom, and Last. To limit the computational cost of running all experiments with these mapping functions, we restrict our experiments with different mapping functions on the `Hinton+Hid`$_{\texttt{MSE/COS}}$`+ATT`$_{\texttt{MSE}}$`+Emb`$_{\texttt{MSE}}$ distillation loss.

3.2.3 Sharing Across Students

We further study the effect of sharing components across students. Given student models with M number of layers and where the $M + 1$ corresponds to the masked language modeling head (MLM Head), we define $g_{share} \subseteq \{0, 1, ..., M, M + 1\}$ as a subset of student layer indices that indicates which layers are shared across students. We study three different scenarios:

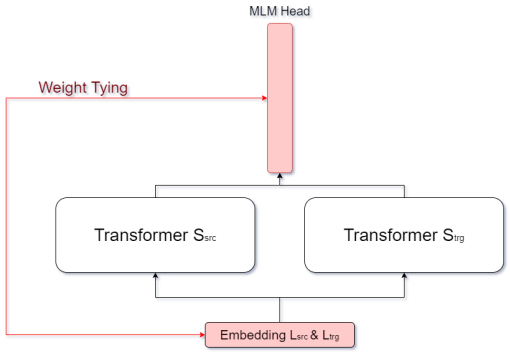


Figure 3.1: Default setting of sharing components across students.

No Sharing (No Share). To give full flexibility to the student, we share nothing between them: $g_{share} = \{\}$. Therefore we allocate the full amount of parameters in a student to one language.

Embedding Sharing (Emb Share). It is hypothesized that the embedding layer is a significant factor in the cross-lingual ability of multilingual transformers [Wu et al., 2019, Artetxe and Schwenk, 2019, Dufter and Schütze, 2021]. E.g., Dufter and Schütze [2021] show that shared special tokens and shared position embeddings are essential factors. We, therefore, experiment with sharing embeddings across students, $g_{share} = \{0\}$, which then are jointly trained by all students during distillation.

Everything Shared (Bilingual/Multilingual). The last option is a special case where sharing everything between students: $g_{share} = \{0, 1, ..., M, M+1\}$ lead to having one student. As we share everything, we essentially have just one student that is then bi- or multilingual. We study this case in Section 4.4.4 and Section 4.5.3

MLM Head Tying. Transformer language models utilize weight tying between the decoder of the masked language modeling head and the embedding layer. This leads to a significantly smaller parameter number, and some works indicate an improvement in the language modeling and translation task [Press and Wolf, 2016, Pappas et al., 2018]. We investigate both tying and having an independent embedding layer and decoder in our work.

Our default setting is first to share the embeddings between students *and* tie the weight of the embedding to the MLM head. This results in also sharing the MLM head between students, i.e,. $g_{share} = \{0, M+1\}$, see Figure 3.1. In Section 4.4.3

and in Section 4.5.2 we investigate what effects the embedding sharing and MLM head tying setting have.

3.2.4 Initialization of Students

The last remaining option that we study is the initialization of our students. We study the effect of two different initialization strategies for the students.

Random Initialization. We study students that are initialized from scratch.

Full Teacher Initialization. Finally, we initialize all student weights from the teacher. The teacher layers are uniformly selected.

As a default, we always use the full teacher initialization option and study the effect of random initialization in Section 4.4.3 and in Section 4.5.2.

3.3 Zero-Shot Fine-Tuning for Monolingual Students

We evaluate our cross-lingual representations across students by zero- and few-shot downstream tasks. Fine-tuning monolingual students in a zero- and few-shot scenario is not trivial: In the extreme case, we share nothing across students, and each is monolingual. Therefore utilizing the default fine-tuning paradigm of multilingual models on cross-lingual downstream tasks, i.e., fine-tuning the model on the source language and evaluating the same fine-tuned model on the target language, is not possible. This section explores different zero-shot (and few-shot) fine-tuning strategies.

For clarification, we assume we have two students, one for the source language L_{src} and one for the target language L_{trg}. Both were distilled after the training procedure in Section 3.1.3 for the respective language. We then denote the student corresponding to the source language as the source student S_{src} and the one corresponding to the target language as the target student S_{trg}. The question remains how one can leverage the knowledge from the source labeled data into the target student.

3.3.1 Full Student Fine-Tuning

Full Fine-tuning (`FULL`). Naively, one can fully fine-tune either source or target model and use the same fine-tuned model to do zero-shot. We experiment with the following:

Source (`SRC`). Fully fine-tuning the source model and using the fine-tuned source model for evaluation on the target language.

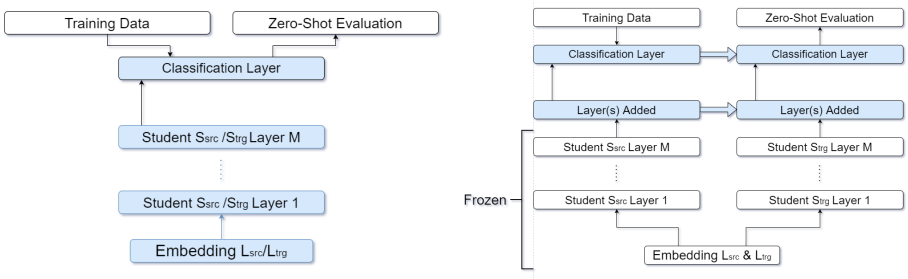


Figure 3.2: Zero-Shot Fine-Tuning Strategies: (*Left*) Full Fine-tuning (`FULL`). (*Right*) Freezing Source Model with one layer added (`FREEZE - Add 1 Layer`). Blue color indicates which components are being fine-tuned.

Target (`TRG`). Fully fine-tuning target model and using the fine-tuned target model for evaluation on the target language.

The approach gives us the full model capacity to fine-tune on the downstream task. We hope that by sharing some components, e.g., embedding sharing (Section 3.2.3), the source and target model have cross-lingual abilities such that fine-tuning in the source language will automatically align the target language to the task. We visualize the full fine-tuning strategy in Figure 3.2 (*Left*).

3.3.2 Freeze Source Student

Freezing Source Model (`FREEZE`). Another naive solution to keep the source and target model aligned is to just fine-tune the task-specific source head on the source language while keeping the source transformer model frozen. Additionally, we can add (randomly initialized) transformer layers on top of the frozen transformer model to increase model capacity. We then initialize the added layers and target head from the source finetuned added layers and head and evaluate zero-shot. We visualize this strategy in Figure 3.2 (*Right*).

3.3.3 Adapters

Task Adapters (`ADAPT`). To introduce parameters that enables us to transfer task-specific knowledge into our target model we use adapters. The idea is that adapters capture task-specific knowledge during fine-tuning and as source and target model are somewhat aligned, we can utilize the modularity of adapters (see Section 2.3.1) and transfer these to the target model. We denote the parameters of the source model as Θ_{src}, target model as Θ_{trg}, source adapters as ϕ_{src} and target adapters

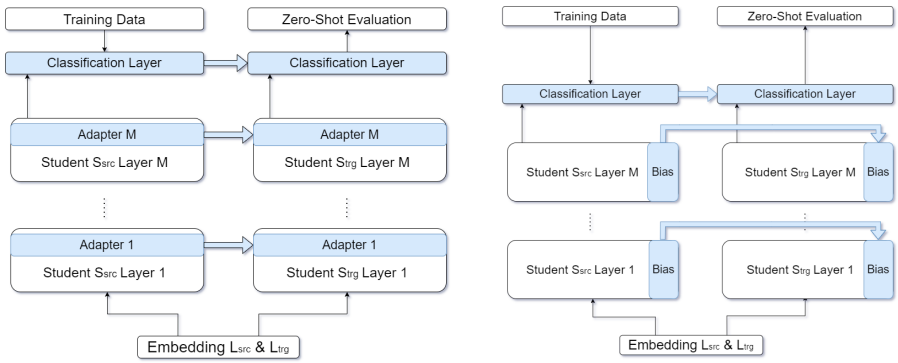


Figure 3.3: Zero-Shot Fine-Tuning Strategies: (*Left*) Task Adapters on the source model (`ADAPT - SRC`). (*Right*) BitFit on the source model (`BitFit - SRC`). Blue color indicates which components are being fine-tuned.

as ϕ_{trg}. We insert the same adapters into our source model and target model, i.e., $\phi_{src} = \phi_{trg}$. We utilize the same placement and architecture as Pfeiffer et al. [2021], see Section 2.3.1 for a more in-depth analysis. We then only fine-tune the adapter parameters and the classification layer while freezing all other parameters $\Theta_{src}, \Theta_{trg}$. We investigate three different fine-tuning strategies:

Source (`SRC`). We fine-tune the adapters based on the source model and transfer the fine-tuned adapters and classification layer into our target model. We then evaluate with the target model, see Figure 3.3 (*Left*).

Target (`TRG`). We fine-tune the adapters based on the target model. In this case, we do not need to transfer any components since we evaluate with the same fine-tuned model - with the target model.

Joint (`JOINT`). We fine-tune the adapters jointly with the source and target model, i.e., calculate the loss on the source and target model with the same adapter and jointly train them based on the averaged loss. E.g. `ADAPT+JOINT` would mean that we insert the same adapters into the source and target model, input a training instance into both models, calculate the loss for each and average it to then jointly train the adapters based on the averaged loss. See Figure 3.4 for a visualization.

3.3.4 BitFit

BitFit (`BitFit`). Another way to introduce task-specific parameters is by using BitFit [Zaken et al., 2021], where we only fine-tune the bias parameters and the

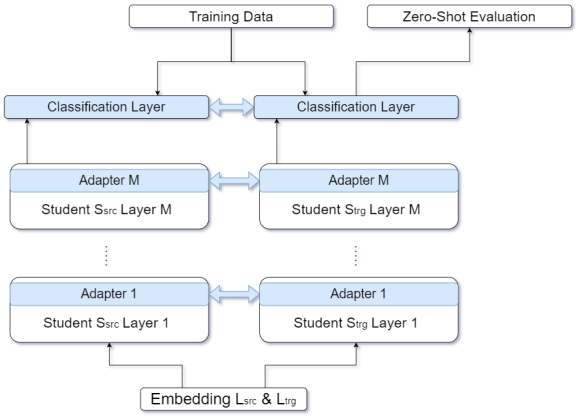


Figure 3.4: Zero-Shot Fine-Tuning Strategies: Joint training of adapters in source and target model. The blue color indicates which components are being fine-tuned.

task head while freezing the rest of the source model. Similar to adapters, we then transfer the fine-tuned bias terms and task head in to our target model. Specifically, as we already denoted in Section 2.1.5, the model parameters Θ are constructed from the weight matrices $\boldsymbol{W}_{(\cdot)}^{l,(\cdot)}$, bias vectors $\boldsymbol{b}_{(\cdot)}^{l,(\cdot)}$ and vectors $\boldsymbol{g}_{(\cdot)}^{l}$. In the BitFit scenario, we only fine-tune the bias terms $\boldsymbol{b}_{(\cdot)}^{l,(\cdot)}$ while keeping $\boldsymbol{W}_{(\cdot)}^{l,(\cdot)}$ and $\boldsymbol{g}_{(\cdot)}^{l}$ frozen.

As in the adapters scenario, we can either fine-tune the bias terms of the source model (`SRC`) and transfer to the target model or fine-tune the bias terms of the target model (`TRG`) directly, or jointly train them (`JOINT`).

3.3.5 Sparse Fine-Tuning

Sparse Fine-Tuning (`SFT`). This strategy is inspired by the idea of sparse fine-tuning [Ansell et al., 2021]. At each fine-tuning step of the source model, we calculate the difference vector ϕ_t at time step t which is the difference between the updated parameters of the source transformer model Θ_{src}^{t+1} at time step $t+1$ and the original parameter Θ_{src}^{t} at time step t. We then add the difference vector ϕ_t to the target model parameters Θ_{trg}^{t} to get the updated target model parameters Θ_{trg}^{t+1}. Only the embeddings are frozen (and shared) across students.

3.3.6 Task Distillation

We further investigate task-specific distillation to try to improve downstream task performance. We outline the task-specific distillation for zero-shot downstream tasks: First, we fine-tune the teacher model on the downstream task with the labeled source instances. We distill from the fine-tuned teacher by providing additional teacher predictions on the source instances. The student then fine-tunes on the teacher's predictions and labels of the source instances. Notice that we utilize the zero-shot fine-tuning strategies listed in the previous sections in this stage. Finally, we evaluate the fine-tuned student on the target instances.

We only distill from the prediction layer of the teacher. The reason is that we have many different zero-shot fine-tuning strategies where, in general, we can not utilize other intermediate representations of the teacher. E.g., when we fine-tune with adapters, we do not change any layer representations of the student and therefore can not incorporate layer distillation. We choose to employ a simple distillation strategy that can be applied to all zero-shot fine-tuning strategies: Minimizing the mean squared difference between the logits produced by the teacher and the logits produced by the student:

$$\mathcal{L}_{\text{logits}} = \frac{1}{N} \sum_{x \in \mathcal{X}} \| z_x^{TE} - z_x^{ST} \|_2^2,$$

where N is the number of examples, z_x^{TE} the logit output of the fine-tuned teacher and z_x^{ST} of the logit output of the student. Together with the MLM loss we get the task distillation loss

$$\mathcal{L}_{\text{task}} = \alpha \cdot \mathcal{L}_{\text{logits}} + \beta \cdot \mathcal{L}_{\text{MLM}}.$$

In our experiments, we choose $\alpha = 0.5$ and $\beta = 0.5$.

3.4 MonoAlignment & MonoShot

In the previous sections, we outlined different alternatives for the main components of the monolingual setup, e.g., distillation loss and what to share across students. In this section, we describe which strategies our approaches `MonoAlignment` and `MonoShot` utilize. Again, `MonoAlignment` is specialized to maximize the cross-lingual alignment of our monolingual students measured in the performance of the retrieval task. In contrast, `MonoShot` tries to maximize the cross-lingual downstream task performance.

Both approaches utilize embedding sharing across students, MLM head tying (see Figure 3.1) and full initialization from the teacher (see Section 3.2.4). Furthermore, both uses the same base distillation loss $\texttt{Hinton+Hid}_{\texttt{MSE}}\texttt{+ATT}_{\texttt{MSE}}\texttt{+Emb}_{\texttt{MSE}}$

(see Section 3.2.2). The main difference between these two approaches is that `MonoAlignment` uses the layer distillation mapping `Last` while `MonoShot` utilizes `Uniform`, i.e. `MonoAlignment` distills from the last layer of the teacher into the last layer of the student while `MonoShot` distills uniformly from the teacher layers into the corresponding student layer. Furthermore, `MonoShot` employs the `FULL TRG` zero-shot fine-tuning strategy (see Section 3.3.1), i.e., fully fine-tuning the target model and using the fine-tuned target model for evaluation.

We provide an heuristic argument of our approaches by comparing them to many different alternatives. In Section 4.4.2 and Section 4.5.1 we lay out a thoroughly comparison of all the alternative distillation strategies listed in Section 3.2.2 to `MonoShot` and `MonoAlignment`. Furthermore, we study all alternative zero-shot fine-tuning strategies (Section 3.3) coupled with `MonoShot` in Section 4.4.1.

Chapter 4

Experiments

In this chapter, we present and analyze the results of `MonoAlignment` for cross-lingual alignment and `MonoShot` for cross-lingual downstream tasks. We first specify the implementation for the teacher/student architecture, the dataset used for the cross-lingual alignment, and the selected downstream tasks in Section 4.1. In Section 4.2, we describe the training and fine-tuning setups. Additionally, we define their hyperparameters, e.g. the learning rate and batch size. We then report the results of our alignment method and downstream task method in Section 4.3. Finally, we split the analyzes of our proposed methods into the analyzes of the downstream task performance (Section 4.4) and the cross-lingual alignment (Section 4.5). Specifically, we analyze different zero-shot fine-tuning strategies for `MonoShot` (Section 4.4.1) and evaluate the impact on the downstream task performance and cross-lingual alignment for the following scenarios: Alternative Knowledge Distillation strategies (Section 4.4.2 and Section 4.5.1), an ablation study (Section 4.4.3 and Section 4.5.2) and changing to the bilingual setup (Section 4.4.4 and Section 4.5.3).

4.1 Model and Dataset

Teacher. As we already discussed, we use XLM-R as our teacher and utilize the implementation XLM-R_{Base} from huggingface[1]. The model XLM-R_{Base} has $L = 12$ layers, a hidden size of $H = 768$, and $A = 12$ self-attention heads. XLM-R_{Base} has a large vocabulary size of 250K and has $12,250$M parameters in total.

Student. The student uses the same architecture as the teacher. We only vary the number of layers L and keep everything the same as the teacher's architecture. We

[1] https://huggingface.co/xlm-roberta-base

Task	Corpus	|Train|	|Dev|	|Test|	Task	Metric
Classification	XNLI	392,702	2,490	5,010	NLI	Acc.
Struct. pred.	NER	20,000	10,000	1,000-10,000	NER	F1
CSR	SIQA & XCOPA	33,410	100	500	MCC	Acc.
Retrieval	Tatoeba	-	-	1,000	Sent. Retrieval	Acc.

Table 4.1: Downstream Tasks and their characteristics. For NER, sizes are in sentences. Struct. pred.: structured prediction. Sent. retrieval: sentence retrieval. Adopted from Hu et al. [2020].

choose $L = 6$ numbers of layers.

Training Set. We train (distill) on the recreated CC-100[2] corpus from [Conneau et al., 2020, Wenzek et al., 2020] which contains monolingual text for over 100 languages, including our relevant languages *English (en)*, *Turkish (tr)*, *Swahili (sw)*, *Urdu (ur)* and *Basque (eu)*. The dataset was constructed by scraping web pages in many languages from January to December 2018. Our source language English has a data size of 82 GB, whereas our target languages have the following data sizes: Turkish has 5.4 GB, Swahili has 332 MB, Urdu has 884 MB, and Basque has 488 MB in total. Among the target languages, Turkish has the highest data size by far. Notice that we do not use any cross-lingual data during training.

Downstream Evaluation Tasks. For downstream evaluation, we use tasks from 4 different categories, each with one task, see Table 4.1. Each category requires different levels of syntax or semantics understanding, e.g., words, phrases, and sentences. Furthermore, each task includes typologically diverse languages making it suitable to evaluate general-purpose cross-lingual representation and the generalization capabilities. All tasks are trained in a zero-shot and few-shot cross-lingual transfer scenario. Zero-shot means we only use labeled English data and unlabeled target language data during training. We do not use any additional data such as human or machine-generated parallel text. In the few-shot scenario, we are given additional labeled examples in the target language that we can use to train our models. We describe each task in the following paragraph:

Retrieval: To evaluate our cross-lingual alignment, we use the `Tatoeba` dataset [Artetxe and Schwenk, 2019], consisting of up to 1000 English-aligned sentence pairs for 122 languages. The task is to find the correct translation by finding the nearest neighbor for each sentence in the other language according to a similarity score and computing the accuracy. We analyze the cosine similarity and BERTScore [Zhang et al., 2019] as the similarity score. To

[2]Publicly available at https://data.statmt.org/cc-100/.

calculate the BERTScore, we use all token representation of each layer. We refer to Zhang et al. [2019] for the exact calculation of BERTScore. We study the retrieval for XX - English, where XX is the target language.

Classification: To evaluate sentence representations, we choose the cross-lingual XNLI dataset [Conneau et al., 2018] for Natural Language Inference (NLI). The dataset is an extension of the existing English datasets for NLI [Bowman et al., 2015, Williams et al., 2017] to several languages. The validation and test set was translated into 14 languages by professional translators. Specifically, the task is to classify a sentence pair consisting of a premise and a hypothesis whether they are either an entailment, contradiction, or neutral relationship. To fine-tune, we first encode the sentence pair and use the final hidden vector of the first input token ([CLS]) as the aggregate representation. We then add a classification layer with layer weights $W \in \mathbb{R}^{3\times H}$, where 3 is the number of labels and H is the hidden representation size. The loss is the standard cross-entropy loss.

Structured prediction: To evaluate structural predictions, we use the NER task. For this purpose, we chose the Wikiann [Pan et al., 2017] dataset. The dataset consists of automatically labeled named entities LOC, PER and ORG in Wikipedia by using a combination of knowledge base properties, cross-lingual and anchor links, self-training, and data selection. We use the same train, dev, and test split strategy as Rahimi et al. [2019]. Furthermore, we deploy an identical training strategy as in the classification task, except we use the individual token representation and its corresponding label to calculate the loss.

Commonsense Reasoning. To evaluate the cross-lingual commonsense reasoning (CSR) performance, we use the XCOPA [Ponti et al., 2020] dataset for evaluation, a cross-lingual multiple-choice classification (MCC) dataset. The dataset is a translation of the English validation and test of COPA [Roemmele et al., 2011] and was re-annotated into 11 target languages. Each instance represents a premise, a question (question type of "What was the CAUSE?" or "What happened as a RESULT?"), and two alternatives. As (X)COPA has a minimal training size in English (400 instances), we follow the same setup as Ponti et al. [2020]: Use SIQA [Sap et al., 2019], another MCC dataset, to fine-tune models with labeled English instances and use XCOPA to evaluate in a different language. An instance in SIQA[3] consists of a premise, a question, and three possible answers. To solve this task, we concatenate each

[3] SIQA has unconstrained questions.

Stage	Batch Size	Training Time XNLI / NER / XCOPA	Optimizer	LR	Weight Decay	LR Scheduler
General-Purpose Distillation	16	500,000 Steps	AdamW	2e-4	0.01	Linear+10% Warm-Up
Zero-Shot Fine-Tuning	64	10 / 20 / 20 Epochs	AdamW	2e-5	0.05	Linear+10% Warm-Up
Few-Shot Fine-Tuning	32	50 / 50 / 50 Epochs	AdamW	2e-5	0.05	-

Table 4.2: Hyperparameters of each training stage.

candidate answer with the corresponding question and passage[4]. We then encode each sequence and pass the resulting `[CLS]` representations through a fully-connected layer, which is used to predict the correct answer. The predicted answer corresponds to the triple with the highest probability.

4.2 Experimental Setup

This section outlines the hyperparameters of all training and evaluation procedures, such as batch size and learning rate (LR). Particularly, we define the hyperparameters of the general-purpose distillation in Section 4.2.1, the zero-shot fine-tuning in Section 4.2.2 and the few-Shot fine-tuning in Section 4.2.3. Furthermore, we summarize the hyperparameters in Table 4.2.

4.2.1 General-Purpose Distillation

As we already discussed in Section 3.2, we use the following default setting: The layers of the students are initialized uniformly by the teacher, we share the embeddings between students and tie the embedding with the decoder of the masked language modeling head if not otherwise stated.

We train (distill) on the CC-100 Corpus, see Section 4.1. Since the training is computational and memory intensive, we choose a batch size of only 16. Furthermore, we chose the standard optimizer AdamW [Loshchilov and Hutter, 2019] with a learning rate of 0.0002 with weight decay 0.01. We adjust the learning rate with a linear scheduler with warm-up steps of $50,000$. In total, we train the models with $500,000$ steps.

4.2.2 Zero-Shot Downstream Task Fine-tuning

We consider the cross-lingual zero-shot setup as a downstream task. We first fine-tune on the English dataset and then provide the system with samples in the

[4] As we use XLM-R models, we concatenate as follows: <s> context </s> </s> question </s> </s> choice </s>

target language without any annotations. We only have one model in the bilingual/multilingual setup that we then fine-tune on the source language and evaluate with the same model in the target language. This poses a challenge in our monolingual student setup as we have separate models for each language. Our proposed method `MonoShot` deploys the `FULL TRG` zero-shot fine-tuning strategy (see Section 3.3.1). In Section 4.4.2 we compare and analyze the fine-tuning strategies listed in Section 3.3.

We optimize all strategies with the standard AdamW optimizer [Loshchilov and Hutter, 2019] with a weight decay of 0.05 and a linear learning rate schedule with 10% warm-up steps. We choose a batch size of 64 and a high learning rate of 0.0001 for parameter-efficient fine-tuning strategies as they introduce new (randomly) initialized parameters. For the full fine-tuning strategy `FULL` we utilize a (standard) lower learning rate of $2e-5$ as all parameters are initialized from the pre-training. We fine-tune for 10 epochs for XNLI and 20 epochs for NER and SIQA/XCOPA.

4.2.3 Few-Shot Downstream Task Fine-tuning

In the few-shot scenario, we are given n numbers of labeled instances in the target language. First, we use the same fine-tuning strategy as in the zero-shot scenario to leverage the task-specific knowledge from the source language. In the bilingual/multilingual setup, we then fine-tune the bilingual/multilingual model on the additional labeled instances. In the monolingual setup, we only fine-tune the target model on these.

Again, we optimize with the standard AdamW optimizer [Loshchilov and Hutter, 2019] with a weight decay of 0.05 and a linear learning rate schedule with 10% warm-up steps with a learning rate of $2e-5$. We choose a batch size of 32. We fine-tune for 50 epochs for XNLI, NER and SIQA/XCOPA.

4.3 Results

In this section, we present the results of our proposed method `MonoAlignment` to improve cross-lingual alignment and `MonoShot` for downstream tasks. First, we select relevant baselines in Section 4.3.1 and compare them to our proposed alignment method on the cross-lingual alignment task in Section 4.3.2. Finally, we report and compare the performance of `MonoShot` in the cross-lingual zero-shot (Section 4.3.3) and few-shot downstream tasks (Section 4.3.4)

4.3.1 Baselines

We select two subsets of baselines: One subset for the alignment task and one for the cross-lingual downstream tasks.

Alignment Baselines. For the alignment task, we process all instances with a pre-trained 12-layer XLM-R model. During the retrieval task, we consider the following layer representations:

- XLM-$R_{\texttt{BASE}}$: All 12 layers.
- XLM-R_6-`ALIGNMENT` (`Uniform`): Only every second layer representation, i.e. the 1th, 3th, 5th, 7th, 9th and 11th layer.
- XLM-R_4-`ALIGNMENT` (`Top`): Only the top 4 layers.
- XLM-R_4-`ALIGNMENT` (`Middle`): Only the middle 4 layers, i.e. the 5th, 6th, 7th and 8th layer.
- XLM-R_4-`ALIGNMENT` (`Bottom`): Only the bottom 4 layers.

Furthermore, we analyze single layer representations for the retrieval task.

Downstream Task Baselines. For the cross-lingual downstream tasks, we analyze the following baselines:

- XLM-$R_{\texttt{BASE}}$: A pre-trained 12 layers XLM-R model which we used as the teacher.
- XLM-R_6 (`Uniform`): A 6 layer XLM-R model where the layers are initialized by every second layer of the pre-trained XLM-$R_{\texttt{BASE}}$ model.
- XLM-R_6 (`Top`): A 6 layer XLM-R model where the layers are initialized by the 6 top layers of the pre-trained XLM-$R_{\texttt{BASE}}$ model.
- XLM-R_6 (`Bottom`): A 6 layer XLM-R model where the layers are initialized by the first 6 layers of the pre-trained XLM-$R_{\texttt{BASE}}$ model.

Notice that the XLM-R_6-`ALIGNMENT` baselines are not separate models but just different subsets of layer representations of XLM-$R_{\texttt{BASE}}$. On the other hand, XLM-R_6 are separate 6 layer models, initialized from different layers of XLM-$R_{\texttt{BASE}}$. All XLM-R models correspond to the implementation `xlm-roberta-base` from huggingface[5].

[5] https://huggingface.co/xlm-roberta-base

Strategy	BERTScore					Cosine Similarity				
	tr-en	sw-en	ur-en	eu-en	Average	tr-en	sw-en	ur-en	eu-en	Average
XLM-R_{BASE}	0.310	0.139	0.143	0.166	0.190	0.545	**0.151**	0.332	0.276	0.326
XLM-R_6-ALIGNMENT (Uniform)	0.242	0.131	0.098	0.138	0.152	0.501	0.146	0.306	0.256	0.302
XLM-R_4-ALIGNMENT (Top)	0.509	0.146	**0.280**	0.163	0.274	0.601	0.092	0.342	0.213	0.312
XLM-R_4-ALIGNMENT (Middle)	**0.528**	**0.164**	0.240	**0.227**	**0.290**	**0.682**	0.146	**0.352**	**0.302**	**0.371**
XLM-R_4-ALIGNMENT (Bottom)	0.135	0.014	0.029	0.053	0.058	0.369	0.123	0.152	0.150	0.199
MonoAlignment-XLM-R_6	0.478	**0.280**	**0.428**	**0.314**	**0.375**	0.494	**0.223**	0.287	0.261	0.316

Table 4.3: We report the accuracy of the retrieval task on the test set of Tatoeba (XTREME) based on the similarity scores BERTScore and Cosine Similarity. We average across language pairs in column Average for each similarity score. The bold numbers in the upper table indicate the best accuracy for the baselines in each language pair for each similarity score. The bold numbers in the bottom table indicate if our approach outperforms the best baseline in the respective language pair and similarity score.

4.3.2 Cross-Lingual Alignment

In this part, we study the induced cross-lingual alignment of MonoAlignment by analyzing the performance of the retrieval task on the Tatoeba dataset and comparing them to our selected baselines. We report the performance on the Tatoeba test set on the target languages Turkish, Swahili, Urdu, and Basque in Table 4.3. Notice that in this section, we do not report alignments of different Knowledge Distillation strategies but have a separate section dedicated to analyzing different strategies (Section 4.5.1).

Comparing XLM-R Baselines. First, we observe that XLM-R_4-ALIGNMENT (Middle) performs best among all other baselines and XLM-R_4-ALIGNMENT (Bottom) performs notably worse. We explain this behavior in the following two paragraphs by looking at using individual layer representations for the retrieval task. Finally, we notice that the retrieval with the cosine similarity in almost all language pairs and baselines performs better than with the BERTScore.

Comparing MonoAlignment. With the similarity score BERTScore our approach MonoAlignment-XLM-R_6 reaches an average accuracy of 0.375 and outperforms all other baselines, improving the best baseline XLM-R_4-ALIGNMENT (Middle) by $+0.085$. On the other hand, when we only consider the Cosine Similarity, our method is inferior to the baseline XLM-R_4-ALIGNMENT (Middle) by -0.055.

The strength of our method is using the similarity score BERTScore for retrieval: First, we observe that using MonoAlignment-XLM-R_6 with the BERTScore yields higher accuracy than using it with the Cosine Similarity - except for the tr - en language pair. Secondly, when we compare MonoAlignment-XLM-R_6 paired with BERTScore, it *outperforms every baseline - regardless of whether we*

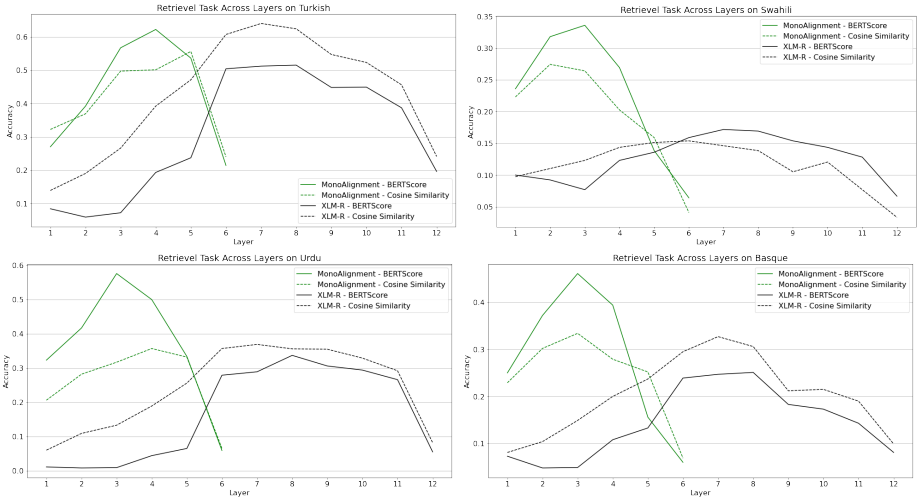


Figure 4.1: Comparison of different model sentence representations with the cosine similarity and the BERTScore for the Tatoeba retrieval task on the `tr-en` *(top-left)*, `sw-en` *(top-right)*, `ur-en` language pair *(bottom-left)* and `eu-en` language pair *(bottom-right)*.

pair the baseline with BERTScore or Cosine Similarity. This holds true for every language pair except the `tr - en` language pair. We hypothesize that since the teacher already has a strong cross-lingual alignment for the language pair `tr - en`, further improving the alignment in an unsupervised fashion is challenging. On the other hand, our method excels in improving the alignment between the languages that have a weak alignment in the teacher, e.g., the pairs `sw - en`, `ur - en` and `eu - en`. On average, our approach `MonoShot` beats the teacher by $+0.185$ with BERTScore. Interestingly, while for the teacher model, the cosine similarity performs better than BERTScore (on average), for our method `MonoAlignment` it is the opposite.

Individual Layer Representation. To further analyze the alignment of our method, we compare individual layer representations for retrieval of `MonoAlignment`-XLM-R_6 and the teacher XLM-R_{BASE}, see Figure 4.1. When we look at the layers in isolation, we first make the observation that the bottom layers of the XLM-R_{BASE} model have the worse alignment performance revealing that representations in early layers are not aligned well. Moreover, the middle layers perform the best among other layers. The alignment then decreases in deeper layers. This behavior is also reflected in our selected baselines. Looking at the retrieval across layers for `MonoAlignment`-XLM-R_6 we immediately see that BERTScore is virtually

Strategy	XNLI		NER		XCOPA		
	tr	sw	tr	sw	tr	sw	Average
XLM-R_{Base}	0.717	0.644	0.693	0.632	0.562	0.544	0.632
XLM-R_6 (Uniform)	0.647	0.554	0.559	0.590	0.538	0.536	0.571
XLM-R_6 (Bottom)	0.662	0.598	0.621	0.588	0.508	0.548	0.588
XLM-R_6 (Top)	0.394	0.350	0.343	0.409	0.510	0.488	0.416
`MonoShot`-XLM-R_6	0.679	0.625	**0.650**	**0.627**	**0.555**	**0.556**	**0.615**
`MonoShot`-XLM-R_6+TD	**0.681**	**0.634**	0.640	0.616	0.552	0.528	0.601

Table 4.4: Results on XNLI, NER, and XCOPA for different Knowledge Distillation strategies target languages $\{tr, sw\}$ on the dev set. The best results across 6 layer models are in-bold. In column Average, we average across all downstream tasks and languages.

outperforming the cosine similarity in every layer. Additionally, we notice that the early layers of `MonoAlignment`-XLM-R_6 are much better aligned than the teacher. Only the last layer of the student and teacher behave similarly in the retrieval task - This is expected as we distill the representation of the last layer of the teacher into the last layer of the student. Importantly, we see that the best performing student layer (3th or 4th layer) with BERTScore is considerably outperforming the best-performing teacher layer with either BERTScore or Cosine Similarity.

Interpretation. We hypothesize that through the high model capacity, XLM-R_{BASE} spreads out the lexical semantics of tokens throughout layers and therefore performs poorly paired with BERTScore. This results in a better retrieval performance by mean-pooling the representations to calculate the cosine similarity to capture the sentence's overall meaning. Through the distillation into a smaller model and pre-training on the MLM task, we compress the overall token representations into a much smaller space. We, therefore, build up stronger lexical semantics per token representation. We investigate this assumption in Section 4.5.1 more.

Conclusively, `MonoAlignment` coupled with BERTScore outperforms the cross-lingual alignment of the teacher considerably in low-resource languages such as Swahili, Urdu, and Basque.

4.3.3 Zero-Shot Results

We evaluate `MonoShot` on the test set of each task along with the selected baselines in Section 4.3.1. We report the zero-shot downstream task performance in Table 4.4.

Comparing MonoShot. `MonoShot`-XLM-R_6 reaches a score of 0.615 on average, outperforming every XLM-R_6 model. Compared to XLM-R_6 (Top), which performs the best among the XLM-R_6 models, we have a performance increase of $+0.027$ on average. Even when we look at each task and target language, `MonoShot`-XLM-R_6 consistently exceeds any XLM-R_6 baseline. Nonetheless, when comparing to the teacher XLM-R_{Base}, we still see that `MonoShot` underperforms -0.031 points on average. We only see an increase in accuracy on XCOPA. However, this effect can be explained by the high variance on XCOPA since the sample size of the test set is very small.

We hypothesize that the model capacity is too low to learn the task efficiently to reach the same performance as the teacher. Therefore we also conduct task-distillation with XLM-R_{Base} model as the teacher. However, we see a decrease of -0.014 in performance. Since Jiao et al. [2020] show that they can significantly improve their downstream task performance with task distillation, we hypothesize that only using the logits as additional information is not enough.

Initialization of XLM-R_6. The results show that among the XLM-R_6 models initialized from different layers of the XLM-R_{Base} model, XLM-R_6 (Top) performs worse than any other 6 layer model with an average score of 0.416. Top layers are more specialized towards the pre-training task [Clark et al., 2019a], i.e., MLM, and therefore are not optimal for cross-lingual transfer tasks. On the other hand, bottom layers are more task-agnostic and lend themselves to cross-lingual transfer tasks. This is indicated in the performance of XLM-R_6 (Bottom), which is the best performing XLM-R_6 model. XLM-R_6 (Uniform) is a mix of XLM-R_6 (Top) and XLM-R_6 (Bottom), which is reflected in its performance, outperforming the former by $+0.048$ and underperforming the latter by -0.017.

Conclusively, our proposed `MonoShot`-XLM-R_6 is a strong 6 layer model for cross-lingual downstream tasks but falls short in comparison to the XLM-R_{Base} model.

4.3.4 Few-Shot Results

We further investigate the cross-lingual few-shot downstream task performances of `MonoShot`. For this purpose we additionally fine-tune the trained models from the previous Section 4.3.3 on a few labeled instances in the target language. In the monolingual setup, we only fine-tune the trained (monolingual) target model. We report the results for Turkish and Swahili in Table 4.5.

We first make some general observations: In the XNLI task, providing the model with a few Turkish samples does not significantly improve performance for any method. However, we see a more significant improvement for the more low-

Strategy - Turkish	XNLI - Turkish				NER - Turkish				XCOPA - Turkish			
	k=0	k=10	k=100	k=500	k=0	k=10	k=100	k=500	k=0	k=10	k=50	k=100
XLM-R_{Base}	0.717	0.701	0.717	0.727	0.693	0.742	0.822	0.853	0.562	0.586	0.590	0.636
XLM-R_6 (Uniform)	0.647	0.638	0.643	0.657	0.559	0.657	0.772	0.813	0.538	0.510	**0.540**	0.544
XLM-R_6 (Bottom)	0.662	**0.663**	0.668	0.671	0.621	0.677	0.771	0.819	0.548	0.508	0.522	**0.562**
XLM-R_6 (Top)	0.394	0.438	0.476	0.545	0.394	0.378	0.582	0.680	0.510	0.488	0.498	0.524
MonoShot-XLM-R_6	**0.679**	0.654	**0.680**	**0.684**	**0.650**	**0.699**	**0.787**	**0.823**	**0.555**	**0.544**	0.532	0.558

Strategy - Swahili	XNLI - Swahili				NER - Swahili				XCOPA - Swahili			
	k=0	k=10	k=100	k=500	k=0	k=10	k=100	k=500	k=0	k=10	k=50	k=100
XLM-R_{Base}	0.644	0.654	0.652	0.688	0.632	0.620	0.838	0.891	0.544	0.544	0.580	0.566
XLM-R_6 (Uniform)	0.554	0.543	0.572	0.585	0.590	**0.668**	0.806	0.850	0.536	0.514	0.528	0.560
XLM-R_6 (Bottom)	0.598	0.592	0.612	0.614	0.588	0.639	0.827	0.871	0.548	0.508	0.522	0.562
XLM-R_6 (Top)	0.394	0.377	0.413	0.432	0.409	0.529	0.734	0.830	0.488	0.488	0.494	0.510
MonoShot-XLM-R_6	**0.625**	**0.654**	**0.648**	**0.667**	**0.627**	0.644	**0.810**	**0.890**	**0.556**	**0.544**	**0.532**	**0.588**

Table 4.5: We report the few-shot performance of the selected baselines and MonoShot on the test set for Turkish (*top*) and Swahili (*bottom*). Bold numbers indicate the best performing 6 layer model.

resource language, Swahili. E.g., with $k = 500$ samples on XNLI, the XLM-R model improves its Turkish performance by $+0.01$ accuracy, but for Swahili, the increase is much higher with $+0.044$. Furthermore, the few labels increase the performance considerably for the NER task. On the other hand, XCOPA is still hard to learn even when providing the system with a few labeled instances.

Comparing MonoShot. We see the same pattern then in the zero-shot scenario: XLM-R_6(Bottom) performs the best across the XLM-R_6 model baselines. Our approach MonoShot-XLM-R_6 exceeds all XLM-R_6 baselines. Again, the teacher still far surpasses our approach MonoShot-XLM-R_6. Only on the NER task in the Swahili language do we get very close to the teacher.

4.4 Analysis of Downstream Task Performance

In the previous Section 4.3, we presented the results of our two distillation approaches MonoAlignment and MonoShot. In this section, we dive into the analysis of both methods' cross-lingual downstream task performance. We first note that the zero-shot fine-tuning of monolingual students consists of two main training procedures: The general Knowledge Distillation to obtain general-purpose students and the zero-shot fine-tuning technique. To find the best performing[6] combination, however, one has to fully evaluate every distillation strategy (10 strategies) coupled with the different fine-tuning strategies (12 strategies) in 2 different languages. This is unfeasible as we are restricted in computational power and time. Therefore, to further analyze MonoShot and its cross-lingual downstream task

[6]Measured in performance in zero-shot cross-lingual downstream tasks.

Strategy	Adapters			BitFit			SFT	FREEZE			FULL	
	SRC	TRG	JOINT	SRC	TRG	JOINT		+0 Layer	+2 Layer	+4 Layer	SRC	TRG
XNLI - TR	0.565	0.664	0.672	0.510	0.579	0.579	0.621	0.449	0.600	0.646	0.360	**0.684**
NER - TR	0.575	0.622	0.631	0.387	0.499	0.501	0.628	0.340	0.482	0.492	0.453	**0.653**
XCOPA - TR	0.61	**0.66**	0.65	0.56	0.58	0.55	0.59	0.60	0.61	0.55	0.55	**0.66**
XNLI - SW	0.526	**0.616**	0.610	0.476	0.536	0.544	0.585	0.389	0.500	0.542	0.425	0.614
NER - SW	0.553	0.636	0.622	0.356	0.366	0.348	0.622	0.0	0.483	0.540	0.541	**0.651**
XCOPA - SW	0.60	0.66	**0.67**	0.53	0.60	0.58	0.60	0.60	0.61	0.60	0.54	0.64
Avg	0.572	0.643	0.643	0,470	0.527	0.517	0.608	0.396	0.548	0.562	0.478	**0.650**

Table 4.6: Results on XNLI, NER and XCOPA for different zero-shot fine-tuning strategies coupled with the distillation strategy of MonoShot on Turkish (*Top Half Table*) and Swahili (*Bottom Half Table*) reported on the dev set. The best results across all fine-tuning strategies are in-bold. We average across the target languages and downstream tasks. Our proposed method MonoShot deploys the FULL (TRG) strategy.

performance, we split the discussion into two parts: (1) First, we discuss alternative zero-shot fine-tuning techniques in Section 4.4.1 such as Adapters and BitFit. (2) Secondly, we discuss different Knowledge Distillation strategies and their impact onto the cross-lingual downstream task performance based on the same zero-shot fine-tuning strategy as MonoShot in Section 4.4.2.

4.4.1 Alternative Zero-Shot Fine-Tuning Strategies

We first study different zero-shot fine-tuning strategies: Adapters, BitFit, SFT, freezing layers, and fully fine-tuning the source or target model, see Section 3.3 for more details on each strategy. As we already discussed, we are computational restricted and therefore only evaluate each fine-tuning strategy based on pre-trained (distilled) students with the MonoShot Knowledge Distillation strategy, see Section 3.2.2 for more details on the distillation strategy. We report the performance of all fine-tuning strategies on the target languages Turkish and Swahili in Table 4.6. We average across all downstream tasks and languages to select the best zero-shot fine-tuning strategy. Note that MonoShot fully fine-tunes the target model in the zero-shot setting, which corresponds to the FULL TRG strategy.

Bitfit. BitFit is not performing on par with any other method. Training the bias terms of the source models and transferring them to the target model (SRC) only reaches an average score of 0.470. While training the bias terms directly in the target model (TRG) increases the performance by +0.057, it still lacks far behind other methods. We observe the same for jointly training the bias terms (JOINT); however, the average increase is less. We explain the poor behavior as follows: In BitFit, we have to share the bias terms across source and target models. As we have to choose which bias terms we want to share, e.g., the source model's bias terms,

we inevitably have to destroy any information that resides in the pre-trained bias terms in the other model. This results in information loss and may be the reason why BitFit underperforms for our setup.

Adapters. Training adapters exceeds BitFit considerably in every training setup (`SRC`, `TRG` and `JOINT`). E.g., training Adapters only with the target model results in an average score of 0.643, an increase of $+0.116$ in comparison to `BitFit TRG`. In contrast to BitFit, Adapters do not destroy any pre-trained parameters but add new randomly initialized parameters to the model, which might explain the increase in performance. Furthermore, training jointly (`JOINT`) or only with the target model (`TRG`) results in the second-best performing zero-shot fine-tuning strategy after the `Full TRG` strategy. It is, however, the best parameter-efficient fine-tuning strategy, as we only add 3.6% parameters per task. By contrast, the `Full TRG` strategy needs to fine-tune 100% of the parameters per task.

SRC vs. TRG vs. JOINT. We notice that training Adapters/bias terms solely with the source model (`SRC`) and then transferring the fine-tuned components into the target model is not competitive to training directly with the target model (`TRG`) or training jointly (`JOINT`). This hints that the source and target model are not perfectly aligned and may represent different information at each layer. Therefore the modularity of the adapters and bias terms can not fully be utilized as in Section 3.3.3 and Section 3.3.4 hypothesized. It is necessary to consider the target model during the fine-tuning process.

SFT. In our modified version of Sparse Fine-Tuning, the target model takes the same updates during the fine-tuning of the source model. We hope to induce the acquired task-specific knowledge into the target model. However, as we already saw in the `Joint vs. SRC/TRG` setup, the source and target model are not perfectly aligned. Therefore, STF lacks behind other methods that consider both models while fine-tuning. Nonetheless, when we compare SFT to `Adapters SRC` and `BitFit SRC`, all methods that do not consider the target model, SFT comes out on top by $+0.036$ and $+0.138$ respectively. This may indicate that using the full model capacity to fine-tune on the task is more valuable.

FREEZE. When we only train the task-head and keep everything else frozen (+0 Layer), the model can not capture the task. The reason is that the task head, which is the only part being fine-tuned, is not complex enough to create a powerful class separation. This argument is supported by the fact that increasing the complexity by adding layers that can be fine-tuned also increases performance on downstream tasks. E.g., adding 4 layers outperforms adding no layers by $+0.166$. However, adding 4 layers to the model is still lacking behind other methods, even though we add and increase a substantial amount of parameters to the model.

FULL. In the case of fine-tuning the full source model and using the fine-tuned source model to evaluate in the target language, the source model loses most cross-lingual knowledge and only reaches an average score of 0.478. It, therefore, can not perform in the target language and only excels at the source language which it was distilled and fine-tuned for (but still lacks behind the teacher - we do not report the English performance here as it is not our goal). However, directly fine-tuning the target model performs the best across all zero-shot fine-tuning strategies with an average score of 0.650. `FULL TRG` also performs best across many tasks and languages, e.g., for XNLI in Turkish, it reaches a score of 0.684, an increase of the second-best performing strategy (`Adapters TRG`) by $+0.02$. It shows that the target model can train on English instances and retain knowledge about the target language outperforming all other methods. We attribute the high performance of `FULL TRG` to the utilization of the full model capacity while having a high degree of target-language-specific parameters. Compared to the other methods, fully fine-tuning uses a much higher number of parameters. Additionally, except for SFT, FULL is the only method to modify the embedding layer of the models, which might help learn the task as most parameters reside in the embedding layer. Furthermore, compared to the `FULL SRC` strategy, which "forgets" most of the target language knowledge, the `FULL TRG` strategy uses the target language-specific model for fine-tuning, which helps to keep this knowledge.

In conclusion, the `FULL TRG` fine-tuning strategy results in the best score on average and is also the leading strategy on many tasks and languages. We, therefore, show that `MonoShot` which utilizes `FULL TRG` uses the most effective zero-shot fine-tuning strategy.

4.4.2 Alternative Knowledge Distillation Strategies

In the previous section, we found that fully fine-tuning directly on the target model `FULL TRG` - the zero-shot fine-tuning strategy that `MonoShot` deploys - worked well across all tasks and languages. This section will analyze how different Knowledge Distillation strategies influence the cross-lingual zero-shot downstream performance. We report the results in Table 4.7.

Different Component Distillation. We first analyze the effect of distilling different parts of the teacher on downstream task performance. Only distilling from the logits via the Hinton Loss (`Hinton`) underperforms on all tasks with an average score of 0.552. This suggests that only distilling information from the logits is not enough to create a cross-lingual representation that can be used for downstream tasks. The reason might be that imitating the logits can be done in many ways through different layer parameterizations that are not optimal for downstream task

Strategy	XNLI		NER		XCOPA		
	tr	sw	tr	sw	tr	sw	AVG
Hinton	0.506	0.533	0.595	0.609	0.53	0.54	0.552
Hinton+Hid_{MSE} (Uniform)	0.648	0.604	0.625	0.644	0.64	0.64	0.634
Hinton+Hid_{MSE}+ATT_{MSE} (Uniform)	0.663	0.608	0.652	0.642	0.63	0.67	0.644
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)=MonoShot	**0.684**	0.614	**0.654**	0.644	**0.66**	0.64	**0.649**
Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.678	0.614	0.621	0.636	0.59	0.58	0.620
Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.660	0.608	0.644	0.646	0.63	**0.68**	0.645
Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Bottom)	0.641	0.575	0.605	0.601	0.61	0.63	0.610
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Top)	0.666	**0.616**	0.647	**0.654**	0.65	0.62	0.642
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Bottom)	0.653	0.577	0.622	0.605	0.59	0.65	0.616
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)=MonoAlignment	0.571	0.574	0.604	0.544	0.63	0.57	0.582

Table 4.7: We report on the dev set of XNLI, NER, and XCOPA for different Knowledge Distillation strategies in the target languages $\{tr, sw\}$. The best results across all distillation strategies are in-bold. In column AVG, we average across downstream tasks and target languages.

performance.

Additional information about intermediate representations via distilling the hidden states (Hinton+Hid_{MSE} (Uniform)) greatly improves performance across all downstream tasks. On average, the method increases by $+0.082$ points in comparison to Hinton. The results are consistent across languages, which also confirm the findings of numerous works [Romero et al., 2015, Sun et al., 2019, Jiao et al., 2020, Sanh et al., 2020]. Specifically, intermediate representations serve as hints for the student during distillation to improve downstream task performance.

We furthermore incorporate the attention distribution with the distillation loss Hinton+Hid_{MSE}+ATT (Uniform). The loss shows a slight improvement by $+0.01$ on average. It was shown that attention weights learned by the teacher capture substantial linguistic knowledge [Clark et al., 2019b] which encourages our students to learn these linguistic hints and, in turn, improves downstream tasks.

Aligning the embeddings between teacher and student with the distillation loss Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform) minimally increases performance by 0.005. Since we initialize the student embedding from the teacher embedding, we hypothesize that the student already has a strong embedding representation that is not being improved by a further alignment through the mean squared error. Nevertheless, other works [Pires et al., 2019, Wu and Dredze, 2019, Dufter and Schütze, 2021] have shown that the embedding is an important factor for the cross-lingual ability of the model. Notice that Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform) which distills every component of the teacher corresponds to the distillation strategy deployed by MonoShot.

We finally study the effect of the MLM task by removing the Hinton Loss from MonoShot (Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)) and we observe a de-

crease in performance by -0.029 on average. This shows that the MLM task is an important factor in inducing better general-purpose students for cross-lingual downstream tasks.

Different Loss Functions. We discuss the choice of the loss function to align hidden representations. We analyze two popular choices: Mean Squared Error (MSE) and Cosine Similarity (COS). We compare `Hinton+` Hid_{MSE} $+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ `(Uniform)` vs `Hinton+` Hid_{COS} $+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ `(Uniform)` and `Hinton+` Hid_{MSE} $+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ `(Bottom)` vs `Hinton+` Hid_{COS} $+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ `(Bottom)`. In both comparisons, we see a slight decline in performance when switching from aligning the hidden representations with mean squared error to the cosine similarity. `Hinton+`$\text{Hid}_{\text{COS}}+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ `(Uniform)` loses on average -0.004 points in comparison to the mean squared error variant and the other loss `Hinton+`$\text{Hid}_{\text{COS}}+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ `(Bottom)` loses -0.006. However, the decline is only minimal. We conclude that both loss functions are a good choice to induce strong cross-lingual downstream task performance.

Different Mapping Functions. We furthermore discuss different mapping functions which determine from which layers of the teacher we distill from. We discuss the loss function `Hinton+`$\text{Hid}_{\text{MSE}}+\text{ATT}_{\text{MSE}}+\text{Emb}_{\text{MSE}}$ with the mapping functions: `Uniform`, `Top`, `Bottom` and `Last`.

We observe that only distilling from the last layer of the teacher into the last layer of the student (`Last`) has only 0.582 points on average and is substantially worse than other mapping functions. Notice that this distillation strategy corresponds to the distillation strategy of `MonoAlignmnet`, which, as we know, creates a strong token alignment representation (see Section 4.3.2). Although only taking information from the last layer creates a strong token alignment, it does not lend itself for cross-lingual downstream tasks. In Section 4.5.1 we discuss the correlation of cross-lingual alignment with downstream task performance.

Distilling from the bottom layers of the teacher (`Bottom`) reaches 0.616 on average and is -0.026 points worse than distilling from the top layers (`Top`). This somehow contradicts our first observations in Section 4.3.3 where 6 layer models (XLM-R_6) initialized from bottom layers performed better than top layers. We argue that because the upper part of the teacher is more suitable for the retrieval task (see Section 4.3.2) and therefore more language-agnostic, this could positively influence the distillation and the performance in a cross-lingual setting.

The best-performing mapping function is uniformly distilling (`Uniform`) with 0.645 points on average. This is in line with the findings of Jiao et al. [2020] in the monolingual case.

We thoroughly investigated different strategies to distill cross-lingual knowledge from the teacher. The best strategy to maximize cross-lingual downstream task

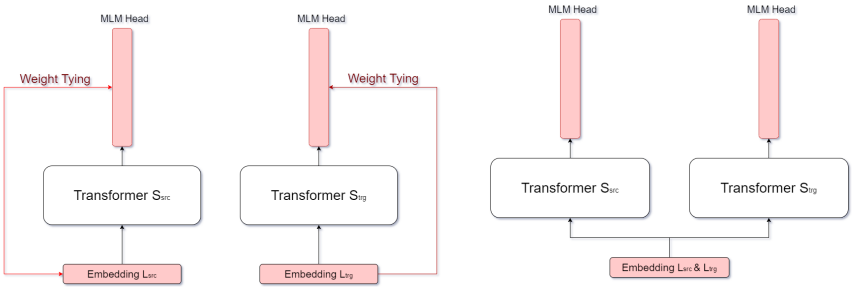


Figure 4.2: *(Left)* w/o Emb Sharing: Independent Student Models. *(Right)* w/o Output Tie: We remove the tying between the MLM head and Embedding Layer. Notice that in the default setting, tying results in sharing the MLM head between students.

Strategy	XNLI		NER		XCOPA		
	tr	sw	tr	sw	tr	sw	AVG
MonoShot-XLM-R_6	**0.684**	**0.614**	**0.654**	**0.644**	**0.66**	0.64	**0.649**
MonoShot-XLM-R_6 w/o Emb Sharing	0.660	0.614	0.645	0.635	0.65	0.61	0.635
MonoShot-XLM-R_6 w/o Teacher Init	0.622	0.596	0.598	0.597	0.55	0.65	0.602 0
MonoShot-XLM-R_6 w/o Output Tie	0.654	0.604	0.590	0.630	0.53	**0.67**	0.613

Table 4.8: Ablation study on embedding sharing, teacher initialization and output tying. We report results on the dev set of XNLI, NER, XCOPA for the target languages $\{tr, sw\}$.

performance is Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform), i.e., distilled from every component of the teacher, choosing the teacher layers to distill from uniformly and using the mean squared error to align the hidden representations of teacher and student. Conclusively, we showed that MonoShot performs the best among different Knowledge Distillation strategies.

4.4.3 Ablation Study

All studies have been conducted by (1) sharing the embeddings across our monolingual students, (2) initialization the weights of our students from the teacher weights, and (3) we tied the embedding layer with the decoder of the masked language modeling head. In this section, we evaluate how these settings influence the downstream task performance of MonoShot. We visualize the different settings that we analyze in Figure 4.2 (see Figure 3.1 for the default setting). We report the results on the dev set in Table 4.8.

Effect of Embedding Sharing. We remove the sharing of the embedding layer

between students, see Figure 4.2 *(Left)*. For the Turkish language, we see a substantial decline in all downstream tasks. On the other hand, Swahili maintains the same performance in XNLI and loses minimally in NER and XCOPA tasks. Without sharing embeddings across students, we lose -0.014 average points. This suggests that when we share embeddings across students, we gain a stronger cross-lingual representation for downstream tasks. Notice that our students are independent of each other in this scenario, i.e., we do not share anything across students. Therefore we show that sharing components across students is essential.

Effect of Teacher Initialization. In our default setting, we use the weights from the teacher (selected uniformly across layers). We study the effects of training our students from scratch, i.e., initializing the weights randomly. We see a strong decrease across all tasks and languages (except for Swahili in XCOPA), losing -0.045 points on average. As our training time during distillation is relatively short in comparison to the training times of XLM-R_{Base}, we can not efficiently train students from scratch. The amount of parameters, especially in the embedding layer, is too high.

Effect of Output Tying. We remove the weight tying between the MLM head and embedding layer. Through the weight untying, we also do not share the MLM head between students anymore; see Figure 4.2 *(Right)*. This has a negative effect across most tasks and languages. On average, we lose -0.036 points without the output tying. Through the output tying, we first reduce the number of parameters massively and secondly improve downstream task performance, which is in line with other works [Press and Wolf, 2016].

Conclusively, our chosen default setting is performing the best compared to the settings we analyzed in this section.

4.4.4 Bilingual Setup

To see the effects of sharing everything across students, we conduct experiments with one bilingual student distilled with the same Knowledge Distillation as our approach `MonoShot` and experiment with a subset of different distillation strategies. We report the results in Table 4.9

MonoShot in Bilingual Setup. We use the same Knowledge Distillation strategy as in `MonoShot` but in the bilingual setup. This strategy then reaches an average score of 0.656 across downstream tasks, which is a slight increase of $+0.007$ in comparison to `MonoShot`. This already shows that sharing components across students helps to create a stronger cross-lingual representation for downstream tasks.

Strategy	XNLI		NER		XCOPA		
	tr	sw	tr	sw	tr	sw	AVG
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)=`MonoShot`	0.684	0.614	0.654	0.644	0.66	0.64	0.649
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)=`MonoAlignment`	0.571	0.574	0.604	0.544	0.63	0.57	0.582
Bilingual-Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.684	**0.636**	**0.668**	0.647	0.66	0.64	0.656
Bilingual-Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Uniform)	**0.696**	0.631	0.643	**0.658**	**0.68**	0.64	**0.658**
Bilingual-Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)	0.666	0.613	0.619	0.607	0.63	**0.66**	0.633

Table 4.9: Results on the dev set of XNLI, NER, and XCOPA for different Knowledge Distillation strategies on the target languages $\{tr, sw\}$. The best results across all distillation strategies are in-bold. In column AVG, we average across downstream tasks and target languages.

While `MonoShot` assigns more capacity to one language, the cross-lingual representation loses some of its strong downstream performance. We hypothesize that we also have a much higher degree of freedom because of the higher model capacity per language. Consequently, we can create a cross-lingual representation space that is not optimal for downstream tasks more easily. On the other hand, although the bilingual setup has a smaller capacity per language, we theorize that the model benefits more through the sharing and joint training of the parameters.

Different Knowledge Distillation Strategies. To see if the same best performing Knowledge Distillation strategy in the monolingual setup holds up in the bilingual setup, we compare different Knowledge Distillation strategies in the bilingual setup. Notice that we do not experiment with all distillation strategies as in the previous Section 4.4.2 since we do not have the computational and time resources to do so. We select promising strategies that worked well in the monolingual setup.

We see that by changing from the mean squared error to the cosine similarity loss to align the hidden representations, the performance increases to an average score of 0.658, a slight increase in performance. In the monolingual student case, changing to the cosine similarity loss decreased performance slightly. However, both changes are not significant. Nonetheless, we see the same drop in performance when we distill from the last layer of the teacher into the last layer of the bilingual student. However, what is noticeable is that the drop is not as substantial as in the monolingual setup - Again showing that through sharing and jointly training, we create a stronger cross-lingual representation space.

Comparison to Teacher. As the bilingual setup works better for cross-lingual downstream tasks than the monolingual setup, we compare the best performing distillation strategy in the bilingual setup (Bilingual-Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Uniform)) to the teacher, see Figure 4.10. In XNLI, the teacher substantially outperforms the bilingual student in every language, e.g., in Swahili by $+0.021$. On the NER task, the teacher again outperforms the bilingual student by $+0.05$ in the

Strategy	XNLI		NER		XCOPA		
	tr	sw	tr	sw	tr	sw	Average
XLM-R_{Base}	**0.717**	**0.644**	**0.693**	0.632	0.562	**0.544**	**0.632**
Bilingual-Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.691	0.623	0.643	**0.634**	**0.586**	0.532	0.618

Table 4.10: We report the best performing bilingual setup (selected via the dev set) with the teacher on the test set of XNLI, NER, and XCOPA with the target languages $\{tr, sw\}$. The best results across methods are in-bold. In the column Average, we average across all downstream tasks and languages.

Turkish language. On the other hand, in Swahili, the bilingual student minimally surpasses the teacher by $+0.002$. In this task, the bilingual setting shows promising results on the very low-resource language Swahili with substantially fewer parameters than the teacher. Future work should explore more low-resource languages for the bilingual setting on the NER task. Finally, on XCOPA, the bilingual student exceeds the teacher in Turkish by $+0.024$ and performs below the teacher in Swahili by -0.012. Again, these results on XCOPA should be explored more as the test size is tiny, and results have a high variance.

Still, the bilingual setup performs below the teacher by -0.014 points on average.

4.5 Analysis of Cross-Lingual Alignment

In this section, we analyze the cross-lingual alignment of `MonoAlignment`. For this reason, we analyze the effect of using different Knowledge Distillation strategies on the performance of the Tatoeba retrieval task and how it affects the sentence representations across layers (Section 4.5.1). Additionally, we study the correlation of downstream tasks with the cross-lingual alignment. In Section 4.5.2, we analyze the same ablation study as in Section 4.4.3 but for the cross-lingual alignment. Finally, we experiment with the bilingual setup for cross-lingual alignment in Section 4.5.3.

We report the results on the alignment task with the average downstream task performance in Table 4.11.

4.5.1 Alternative Knowledge Distillation Strategies

In this part, we study how different Knowledge Distillation strategies affect cross-lingual alignment.

Distill different Components. When we only distill from the logits by using the Hinton strategy (*Hinton*), we get a high accuracy of 0.297 with the BERTScore, but

Strategy	Downstream Tasks	BERTScore			Cosine Similarity		
	Average	tr-en	sw-en	Average	tr-en	sw-en	Average
XLM-R	0.659	0.310	0.139	0.225	**0.545**	0.151	0.348
Hinton	0.552	0.404	0.190	0.297	0.351	0.079	0.215
Hinton+Hid_{MSE} (Uniform)	0.634	0.218	0.136	0.177	0.407	0.103	0.255
Hinton+Hid_{MSE}+ATT (Uniform)	0.649	0.227	0.139	0.183	0.411	0.100	0.256
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)=MonoShot	0.649	0.229	0.128	0.179	0.407	0.108	0.258
Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.620	0.201	0.139	0.183	0.405	0.103	0.256
Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.645	0.202	0.118	0.160	0.416	0.097	0.257
Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Bottom)	0.610	0.124	0.115	0.120	0.331	0.103	0,217
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Top)	0.642	0.205	0.167	0.186	0.408	0.113	0.267
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Bottom)	0.616	0.139	0.105	0.122	0.326	0.118	0.222
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)=MonoAlignment	0.582	**0.478**	**0.280**	**0.379**	0.494	**0.223**	**0.359**

Table 4.11: First, we report the average downstream task performance (XNLI, NER, and XCOPA) across languages on the dev set. We then state the accuracy based on the BERTScore and Cosine Similarity on the test set of Tatoeba (XTREME). We average across languages in column AVG for each similarity score. Bold numbers indicate the best performing strategy for each language pair and similarity score.

a low accuracy of 0.215 with the cosine similarity compared to other methods. This already shows that BERTScore and cosine similarity does not necessarily correlate. To calculate the cosine similarity between two sentences, we have to average across all tokens and layers, which models the overall meaning of a sentence. On the other hand, BERTScore is a more fine-grained approach to comparing two sentences, calculating similarities between single token representations. We hypothesize that because the Hinton loss only takes the logits and the MLM task into account, the (contextualized) token representations of the teacher have to be *condensed down into a smaller model resulting in a better lexical semantics token representation*. On the other hand, the teacher is twice as deep as the student, which then allows to *spread meaningful token representations more out and may aggregate to a better representation* of the sentence with the cosine similarity.

This effect is diminished when we take every second of hidden representations of the teacher into the distillation strategy (Hinton+Hid_{MSE} (Uniform)) - We now mimic the spreading out of meaningful token representations in our student - losing -0.12 accuracy points with BERTScore but improving the accuracy by 0.03 with cosine similarity. Distilling from the Attention and Embedding only slightly affects the retrieval task on both metrics.

Our above reasoning is supported by the performance of the different mapping functions: (Uniform), (Top), (Bottom) and (Last). When we couple every layer of the student with one of the teachers - either uniformly, top or only bottom layers, we force every layer to learn the same property of the teacher layer representation resulting in low average accuracy of around $0.222 - 0.267$ with BERTScore. When we, however, only distill from the last layer of the teacher,

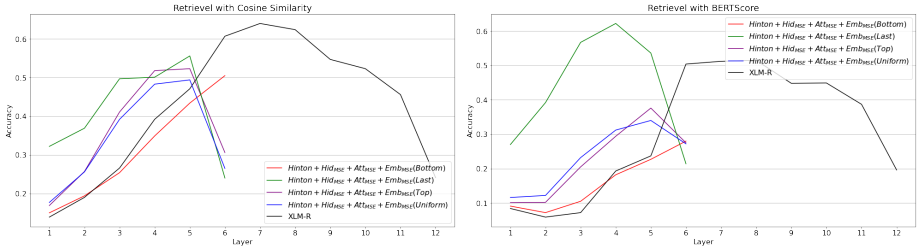


Figure 4.3: Comparison of the alignment of different layer representations with *(left)* the cosine similarity and *(right)* the BERTScore for the Tatoeba retrieval task on the `English-Turkish` language pair.

we force the smaller student to learn the same contextualized token representation used in the MLM task as the teacher, which in turn then compresses all previous representations into the smaller student's capacity. This leads to a stronger lexical semantics contextualized token representation. `MonoAlignment` reaches an average accuracy of 0.314 (averaged across 4 target languages, see Table 4.3 in the result Section) outperforming the teacher by $+0.185$ with BERTScore.

Finally, we investigate using the cosine similarity to align hidden representations instead of the mean squared error. We see a slight decrease compared to the mean squared error on both BERTScore and cosine similarity. E.g. using `Hinton+Hid`$_{\text{COS}}$`+ATT`$_{\text{MSE}}$`+Emb`$_{\text{MSE}}$ `(Uniform)` instead of the mean squared error version, we observe a slight decrease of -0.019 using BERTScore.

Alignment Across Layers. To further understand the effects of mapping functions, we visualize the alignment across different layer representations, see Figure 4.3. As already seen in Table 4.11, distilling only from the bottom layers (`Hinton+Hid`$_{\text{MSE}}$`+ATT`$_{\text{MSE}}$`+Emb`$_{\text{MSE}}$ `(Bottom)`) shows the worse alignment whereas uniformly and top layer distillation show similar improvement. The Figure can explain these effects of different mapping functions: The bottom half of the XLM-R teacher, especially the first three layers, have low alignment, which then is induced into the students by the `Hinton+Hid`$_{\text{MSE}}$`+ATT`$_{\text{MSE}}$`+Emb`$_{\text{MSE}}$ `(Bottom)` distillation strategy and explains the overall low alignment. In comparison to the top half of the teacher, the alignment is better than the lower part, which is reflected in the `Hinton+Hid`$_{\text{MSE}}$`+ATT`$_{\text{MSE}}$`+Emb`$_{\text{MSE}}$ `(Top)` strategy.

We also observe that for `Hinton+Hid`$_{\text{MSE}}$`+ATT`$_{\text{MSE}}$`+Emb`$_{\text{MSE}}$ `(Last)` (corresponds to `MonoAlignment`) earlier layers are much better aligned using the BERTScore. Again, this confirms our hypothesis that we get a better lexical semantic representation throughout layers by compressing the token representations.

Strategy	BERTScore			Cosine Similarity		
	tr-en	sw-en	Average	tr-en	sw-en	Average
MonoShot-XLM-R_6	**0.229**	**0.128**	**0.179**	0.407	0.108	0.258
MonoShot-XLM-R_6 w/o Emb Sharing	0.190	0.108	0.149	**0.413**	**0.149**	**0.281**
MonoShot-XLM-R_6 w/o Teacher Init	0.171	0.100	0.136	0.394	0.133	0.264
MonoShot-XLM-R_6 w/o Output Tie	0.186	0.115	0.151	0.402	0.133	0.268

Table 4.12: Ablation study on embedding sharing, teacher initialization and output tying. We report results on the retrieval task on Tatoeba for the languages tr - en and sw - en. We average across languages in column Average for each similarity score. Bold numbers indicate the best performing strategy for each language pair and similarity score.

Correlation Downstream Tasks with Cross-Lingual Alignment. Surprisingly, there is no positive correlation between the performance on downstream tasks and alignment for either BERTScore and cosine similarity. One would expect that we perform better on cross-lingual downstream tasks when the alignment is better between monolingual students. E.g. our best aligned model distilled with Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)=MonoAlignment has an retrieval accuracy of 0.379 with BERTScore averaged on tr-en and sw-en but only reaches a downstream task score of 0.582. On the other hand, Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)=MonoShot only has an retrieval accuracy of 0.179 with BERTScore but reaches the best downstream task score of 0.649, outperforming MonoAlignment by $+0.066$. The same pattern can be observed for the retrieval with the cosine similarity.

4.5.2 Ablation Study

We do the same ablation study as in Section 4.4.3 to study the effects on the cross-lingual alignment. We however do not conduct the ablation study for MonoAlignment but for MonoShot. The reason is that we would have to rerun each ablation study for MonoAlignment which is computationally expensive. Furthermore, we hypothesize that the effects on the cross-lingual alignment observed for MonoShot are highly correlated to MonoAlignment. We summarize the results in Table 4.12.

Effect of Embedding Sharing. Interestingly, not sharing the embedding across students increases the alignment between students with the cosine similarity. When we do not share the embeddings between the students, we have a higher degree of freedom to mimic the teacher. As the teacher already has a high alignment with the cosine similarity, we hypothesize that the students can mimic the alignment with the mean squared error much better than when we compress the students to a lower

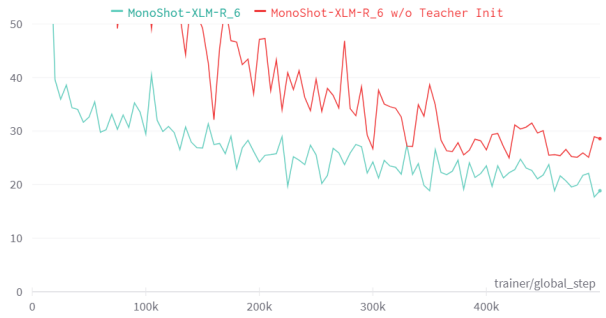


Figure 4.4: Visualization of the MLM task performance for Turkish during distillation for MonoShot-XLM-R_6 and MonoShot-XLM-R_6 w/o Teacher Init. The performance is measured in perplexity. MonoShot-XLM-R_6 w/o Teacher Init is not performing as well as with initializing weights from the teacher.

model capacity by sharing the embeddings between students. On the other hand, alignment with BERTScore decreases. This is in line with our hypothesis that we force the students to create stronger lexical token representations by compressing the model (by sharing the embeddings).

Effect of Teacher Initialization. Without teacher initialization, the alignment with the cosine similarity is in the same range as when we distill from scratch. This shows that the alignment with the cosine similarity can be created by mimicking the teacher layer with the mean squared error. We optimize the alignment between the representations of all tokens of the teacher and students with the mean squared error. Since the cosine similarity calculates the mean of all token representations first, we efficiently can create a strong cross-lingual alignment with the cosine similarity. However, BERTScore does decrease in performance (both having the same model capacity), showing that a well-initialized student influences BERTScore positively. We hypothesize that the BERTScore is much more influenced by the MLM task, which shows poor performance without any teacher initialization, see Figure 4.4.

Effect of Output Tying. We see the same behavior as in the w/o Emb Sharing scenario. Again, we explain these effects by the model capacity. Without the output tying, the students have a much higher degree of freedom, and therefore the BERTScore decreases as the model capacity is not getting compressed as much.

Conclusively, our chosen default setting has the best alignment with BERTScore compared to the settings that we analyzed in this section.

Strategy	Downstream Tasks	BERTScore			Cosine Similarity		
	Average	tr-en	sw-en	Average	tr-en	sw-en	Average
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)=MonoShot	0.649	0.229	0.128	0.179	0.407	0.108	0.258
Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)=MonoAlignment	0.582	**0.478**	**0.280**	**0.379**	0.494	**0.223**	**0.359**
Bilingual-Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.659	0.214	0.136	0.175	0.425	0.110	0.268
Bilingual-Hinton+Hid_{COS}+ATT_{MSE}+Emb_{MSE} (Uniform)	0.661	0.205	0.136	0.171	0.421	0.118	0.270
Bilingual-Hinton+Hid_{MSE}+ATT_{MSE}+Emb_{MSE} (Last)	0.633	0.408	0.228	0.318	**0.509**	0.159	0.334

Table 4.13: First, we report the average downstream task performance (XNLI, NER, and XCOPA) across languages on the dev set. We then state the accuracy based on the BERTScore and Cosine Similarity on the test set of Tatoeba (XTREME). We average across languages in column Average for each similarity score. Bold numbers indicate the best performing strategy for each language pair and similarity score.

4.5.3 Bilingual Setup

In Section 4.4.4 we analyzed the downstream performance of different Knowledge Distillation for the bilingual setup. We extend the analysis to the induced cross-lingual alignment of the bilingual setup. We report the performance on the Tatoeba retrieval task in Table 4.13.

MonoAlignment in Bilingual Setup. Using the same Knowledge Distillation as MonoAlignment in the bilingual setup reaches an average accuracy of 0.318 with BERTScore and 0.334 with the cosine similarity. Surprisingly, it performs below the monolingual setup in both scores, by -0.061 and -0.025, respectively. This hints that our monolingual setup creates a stronger alignment than the bilingual setup, even though we only share the embedding and MLM head during the distillation. We, therefore, show that the monolingual setup positively attributes to the high alignment with BERTScore. On the other hand, the downstream task performance is notable stronger on the downstream task on average.

Chapter 5

Conclusion & Future Work

We presented a novel setup to distill multilingual transformers into monolingual components in this work. This setup assumes that we can circumvent the curse of multilinguality by assigning more model capacity per language by having language-specific students. Based on the novel setup, we propose two Knowledge Distillation: A distillation strategy to create students that have a strong cross-lingual alignment, called `MonoAlignment` and another distillation strategy to induce a cross-lingual representation that lends itself for zero-shot downstream tasks, called `MonoShot`. We conduct various experiments with the XLM-R model as the teacher, distilling into 6 layer student models.

We evaluated our approach `MonoAlignment`-XLM-R_6 on the Tateoba (XTREME) dataset [Hu et al., 2020] which is used to evaluate the cross-lingual alignment of models. We choose the low-resource languages on `Turkish`, `Swahili`, `Urdu` and `Basque` and show that `MonoAlignment`-XLM-R_6 exceeds the alignment performance of the teacher model XLM-R [Conneau et al., 2020] and other baselines significantly (on average). The strong alignment of our model is mostly driven by strong token representations, measured by the high alignment with BERTScore. Our experiments indicate that this effect is caused by the smaller model capacity of the student model. Since the only intermediate representation that `MonoAlignment`-XLM-R_6 mimics is the last layer of the teacher, which additionally takes the MLM task into account, the (contextualized) token representations of the teacher have to be condensed down into a smaller model resulting in a better lexical semantics token representation. On the other hand, the teacher has a bigger model capacity than the student, which allows for spreading meaningful token representations more out. Furthermore, our experiments show that when we use the bilingual setup with the same distillation strategy as `MonoAlignment`-XLM-R_6, i.e., distilled into just one bilingual student, the strong alignment declines indicating that

the separate monolingual students are an important factor. We further show that `MonoAlignment`-XLM-R_6 show the strongest alignment compared to different Knowledge Distillation strategies.

Future work could explore `MonoAlignment` for multilingual transformers that are trained to have a strong retrieval performance. E.g., Reimers and Gurevych [2020] construct a multilingual version of SBERT [Reimers and Gurevych, 2019] using Knowledge Distillation with parallel data. The resulting multilingual transformer model has a strong performance on various retrieval tasks. The question remains if we can improve their model via our unsupervised distillation method. As `MonoAlignment` can be applied to any transformer model, the extension to Reimers and Gurevych [2020] work is straightforward.

Finally, we evaluate our second approach `MonoShot`-XLM-R_6 for cross-lingual zero and few-shot downstream tasks on the target languages `Turkish`, `Swahili`. For this purpose, we use the XNLI [Conneau et al., 2018], Wikiann and SIQA [Sap et al., 2019] with XCOPA [Ponti et al., 2020] dataset. `MonoShot`-XLM-R_6 exceeds other models that have the same small capacity as the trained students showing a strong cross-lingual representation for downstream tasks for its model capacity. However, when we consider the teacher (XLM-R) model, which has a much higher model capacity, `MonoShot`-XLM-R_6 performs below the teacher on average. The same behavior can be seen in the few-shot scenario. Nonetheless, our analysis shows that the distillation strategy that is being deployed by `MonoShot` outperforms other strategies. Furthermore, we conduct experiments with various zero-shot strategies and show that `MonoShot`'s strategy works best.

One limiting aspect of our experiments with `MonoShot` is that we used small models. This alone can already be the reason why we fail to exceed the teacher. Future work could either scale the teacher up to a higher capacity so that the student can also be scaled up or only scale the student up. We then may have the suitable capacity to learn the downstream task properly while benefiting from the higher language per model capacity. Furthermore, we used the same vocabulary for the student as the teacher, which is built up from 100 languages and therefore is suboptimal for especially low-resource languages. Future work can explore using a more specialized vocabulary for the selected languages and expanding distillation strategies that can accompany different vocabulary sizes between teacher and student.

Appendix A

Program Code / Resources

All source code used during this thesis is available on Github under the link https://github.com/MinhDucBui/Master-Thesis and instructions on how to run the code. Specifically, the code is split into three projects: (1) Cross-Lingual Knowledge Distillation, (2) Trident, and (3) Trident-Xtreme.

(1) Cross-Lingual Knowledge Distillation (clkd). This project contains all distillation done throughout the thesis. Furthermore, the project is set up to use any distillation loss easily, any number of students, and languages per student. The project also allows to change the teacher and student architecture and to choose between monolingual, bilingual, or multilingual distillation setup. Furthermore, we constructed config files for each distillation loss and each setup used in this thesis. Each hyperparameter can be adjusted by changing the respective config file.

Finally, we implemented Adversarial Learning (Similar to [Keung et al., 2020]) and Hierarchical Contrastive Learning [Wei et al., 2021] to strengthen the alignment across students. Both were not used in this thesis as they did not yield any improvements. We hypothesize that one has to find the correct hyperparameters, which we leave for future work to explore.

Our Cross-Lingual Knowledge Distillation project can be found here https://github.com/MinhDucBui/clkd.

(2) Trident (trident). This project is a generic framework to train and evaluate (deep learning) models. We adjusted the implementation from https://github.com/fdschmidt93/trident to accompany our needs of multiple students that interact with each other during training. Our implementation can be found here: https://github.com/MinhDucBui/trident

(3) Trident-Xtreme (trident-xtreme). Trident-Xtreme is a project implementing

a modular deep learning stack to train and evaluate models with Trident. We adjust the implementation from https://github.com/fdschmidt93/trident-xtreme to allow us to train and evaluate our students on XNLI, XCOPA, NER, and the retrieval task. Our implementation can be found here: https://github.com/MinhDucBui/trident-xtreme.

Nevertheless, our entire code base with a PDF version of this thesis and an instruction on how to run the code is also contained in the USB stick attached to this thesis.

Bibliography

Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018. URL https://arxiv.org/abs/1803.08375.

Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations, 2020.

David Alvarez-Melis and Tommi Jaakkola. Gromov-Wasserstein alignment of word embedding spaces. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1881–1890, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1214. URL https://aclanthology.org/D18-1214.

Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. Composable sparse fine-tuning for cross-lingual transfer, 2021. URL https://arxiv.org/abs/2110.07560.

Wissam Antoun, Fady Baly, and Hazem Hajj. AraBERT: Transformer-based model for Arabic language understanding. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, pages 9–15, Marseille, France, May 2020. European Language Resource Association. ISBN 979-10-95546-51-1. URL https://aclanthology.org/2020.osact-1.2.

Mikel Artetxe and Holger Schwenk. Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond. *Transactions of the Association for Computational Linguistics*, 7:597–610, nov 2019. doi: 10.1162/tacl_a_00288. URL https://doi.org/10.1162%2Ftacl_a_00288.

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation, 2017. URL https://arxiv.org/abs/1710.11041.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. A robust self-learning method for fully unsupervised cross-lingual mappings of word embeddings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 789–798, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1073. URL https://aclanthology.org/P18-1073.

Mikel Artetxe, Sebastian Ruder, Dani Yogatama, Gorka Labaka, and Eneko Agirre. A call for more rigor in unsupervised cross-lingual learning. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7375–7388, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.658. URL https://aclanthology.org/2020.acl-main.658.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Lei Jimmy Ba and Rich Caruana. Do deep nets really need to be deep?, 2013. URL https://arxiv.org/abs/1312.6184.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. URL https://arxiv.org/abs/1409.0473.

Ankur Bapna, Naveen Arivazhagan, and Orhan Firat. Simple, scalable adaptation for neural machine translation, 2019.

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 2:157–66, 1994.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2016. URL https://arxiv.org/abs/1607.04606.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1075. URL https://aclanthology.org/D15-1075.

Cristian Bucila, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD '06*, 2006.

Hyung Won Chung, Dan Garrette, Kiat Chuan Tan, and Jason Riesa. Improving multilingual models with language-clustered vocabularies. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4536–4546, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.367. URL https://aclanthology.org/2020.emnlp-main.367.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D. Manning. What does bert look at? an analysis of bert's attention, 2019a.

Kevin Clark, Minh-Thang Luong, Urvashi Khandelwal, Christopher D. Manning, and Quoc V. Le. Bam! born-again multi-task networks for natural language understanding, 2019b.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390177. URL https://doi.org/10.1145/1390156.1390177.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data, 2017. URL https://arxiv.org/abs/1710.04087.

Alexis Conneau, Guillaume Lample, Ruty Rinott, Adina Williams, Samuel R. Bowman, Holger Schwenk, and Veselin Stoyanov. Xnli: Evaluating cross-lingual sentence representations. In *EMNLP*, 2018.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Simon S. Du and Jason D. Lee. On the power of over-parametrization in neural networks with quadratic activation, 2018.

Philipp Dufter and Hinrich Schütze. Identifying necessary elements for bert's multilinguality, 2021.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. 2018. doi: 10.48550/ARXIV.1803.03635. URL https://arxiv.org/abs/1803.03635.

Tommaso Furlanello, Zachary C. Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks, 2018. URL https://arxiv.org/abs/1805.04770.

Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32 (10):4291–4308, oct 2021. doi: 10.1109/tnnls.2020.3019893. URL https://doi.org/10.1109%2Ftnnls.2020.3019893.

Felix Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12:2451–71, 10 2000. doi: 10.1162/089976600300015015.

Goran Glavas, Robert Litschko, Sebastian Ruder, and Ivan Vulic. How to (properly) evaluate cross-lingual word embeddings: On strong baselines, comparative analyses, and some misconceptions, 2019.

Cliff Goddard. Natural semantic metalanguage. 2006.

Jianping Gou, Baosheng Yu, Stephen J. Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, mar 2021. doi: 10.1007/s11263-021-01453-z. URL https://doi.org/10.1007%2Fs11263-021-01453-z.

Demi Guo, Alexander M. Rush, and Yoon Kim. Parameter-efficient transfer learning with diff pruning, 2020. URL https://arxiv.org/abs/2012.07463.

Francisco Guzman, Peng-Jen Chen, Myle Ott, Juan Pino, Guillaume Lample, Philipp Koehn, Vishrav Chaudhary, and Marc'Aurelio Ranzato. The flores evaluation datasets for low-resource machine translation: Nepali–english and sinhala–english. pages 6100–6113, 01 2019. doi: 10.18653/v1/D19-1632.

Yaru Hao, Li Dong, Furu Wei, and Ke Xu. Visualizing and understanding the effectiveness of bert, 2019. URL https://arxiv.org/abs/1908.05620.

Zellig S. Harris. Distributional structure. *¡i¿WORD¡/i¿*, 10(2-3):146–162, 1954. doi: 10.1080/00437956.1954.11659520. URL https://doi.org/10.1080/00437956.1954.11659520.

Karl Moritz Hermann and Phil Blunsom. Multilingual models for compositional distributed semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 58–68, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1006. URL https://aclanthology.org/P14-1006.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

Geoffrey E. Hinton and Richard S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *NIPS*, 1993.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998. doi: 10.1142/S0218488598000094.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Yedid Hoshen and Lior Wolf. Non-adversarial unsupervised word translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 469–478, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1043. URL https://aclanthology.org/D18-1043.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp, 2019.

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification, 2018.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. Xtreme: A massively multilingual multi-task benchmark for evaluating cross-lingual generalization, 2020.

Kejun Huang, Matt Gardner, Evangelos Papalexakis, Christos Faloutsos, Nikos Sidiropoulos, Tom Mitchell, Partha P. Talukdar, and Xiao Fu. Translation invariant word embeddings. In *Proceedings of the 2015 Conference on Empirical*

Methods in Natural Language Processing, pages 1084–1088, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1127. URL https://aclanthology.org/D15-1127.

Frederick Jelinek, Robert L. Mercer, Lalit R. Bahl, and J. Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *Journal of the Acoustical Society of America*, 62, 1977.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding, 2020.

Xiao Jin, Baoyun Peng, Yichao Wu, Yu Liu, Jiaheng Liu, Ding Liang, Junjie Yan, and Xiaolin Hu. Knowledge distillation via route constrained optimization, 2019.

Karthikeyan K, Zihan Wang, Stephen Mayhew, and Dan Roth. Cross-lingual ability of multilingual bert: An empirical study, 2020.

Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *EMNLP*, 2013.

Prabhu Kaliamoorthi, Aditya Siddhant, Edward Li, and Melvin Johnson. Distilling large language models into tiny and effective students using pqrnn, 2021.

Phillip Keung, Yichao Lu, and Vikas Bhardwaj. Adversarial learning with contextual embeddings for zero-resource cross-lingual classification and ner, 2020.

Simran Khanuja, Melvin Johnson, and Partha Talukdar. Mergedistill: Merging pre-trained language models using distillation, 2021.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors, 2015. URL https://arxiv.org/abs/1506.06726.

Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. Inducing crosslingual distributed representations of words. In *Proceedings of COLING 2012*, pages 1459–1474, Mumbai, India, December 2012. The COLING 2012 Organizing Committee. URL https://aclanthology.org/C12-1089.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. doi: 10.1145/3065386.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates, 2018. URL https://arxiv.org/abs/1804.10959.

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing, 2018. URL https://arxiv.org/abs/1808.06226.

Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *ICML*, 2015.

Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining, 2019.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition, 2016. URL https://arxiv.org/abs/1603.01360.

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Phrase-based & neural unsupervised machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5039–5049, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1549. URL https://aclanthology.org/D18-1549.

Anne Lauscher, Vinit Ravishankar, Ivan Vulić, and Goran Glavaš. From zero to hero: On the limitations of zero-shot language transfer with multilingual Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4483–4499, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.363. URL https://aclanthology.org/2020.emnlp-main.363.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *ArXiv*, abs/1504.00941, 2015.

Linqing Liu, Huan Wang, Jimmy Lin, Richard Socher, and Caiming Xiong. Mkd: a multi-task knowledge distillation approach for pretrained language models, 2020.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Improving multi-task deep neural networks via knowledge distillation for natural language understanding, 2019a.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019b.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization, 2017. URL https://arxiv.org/abs/1712.01312.

Thang Luong, Hieu Pham, and Christopher D. Manning. Bilingual word representations with monolingual quality in mind. In *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pages 151–159, Denver, Colorado, June 2015. Association for Computational Linguistics. doi: 10.3115/v1/W15-1521. URL https://aclanthology.org/W15-1521.

Alexandre Magueresse, Vincent Carles, and Evan Heetderks. Low-resource languages: A review of past work and future challenges, 2020.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. Camembert: a tasty french language model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020. doi: 10.18653/v1/2020.acl-main.645. URL http://dx.doi.org/10.18653/v1/2020.acl-main.645.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, J.H. Cernocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. pages 5528 – 5531, 06 2011. doi: 10.1109/ICASSP.2011.5947611.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013a.

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation, 2013b.

Hossein Mobahi, Mehrdad Farajtabar, and Peter L. Bartlett. Self-distillation amplifies regularization in hilbert space, 2020. URL https://arxiv.org/abs/2002.05715.

Saif Mohammad, Mohammad Salameh, and Svetlana Kiritchenko. How translation alters sentiment. *J. Artif. Intell. Res. (JAIR)*, 55:95–130, 01 2016. doi: 10.1613/jair.4787.

Subhabrata Mukherjee and Ahmed Awadallah. Xtremedistil: Multi-stage distillation for massive multilingual models, 2020.

Xiaoman Pan, Boliang Zhang, Jonathan May, Joel Nothman, Kevin Knight, and Heng Ji. Cross-lingual name tagging and linking for 282 languages. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1946–1958, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1178. URL https://aclanthology.org/P17-1178.

Nikolaos Pappas, Lesly Miculicich, and James Henderson. Beyond weight tying: Learning joint input-output embeddings for neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 73–83, Brussels, Belgium, October 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-6308. URL https://aclanthology.org/W18-6308.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2012. URL https://arxiv.org/abs/1211.5063.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, 2018.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. Adapterhub: A framework for adapting transformers, 2020a. URL https://arxiv.org/abs/2007.07779.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. Mad-x: An adapter-based framework for multi-task cross-lingual transfer, 2020b. URL https://arxiv.org/abs/2005.00052.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. Adapterfusion: Non-destructive task composition for transfer learning, 2021.

Telmo Pires, Eva Schlinger, and Dan Garrette. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1493. URL https://aclanthology.org/P19-1493.

Warren J. Plath. Early Years in Machine Translation: Memoirs and Biographies of Pioneers. *Computational Linguistics*, 28(4):554–559, 12 2002. ISSN 0891-2017. doi: 10.1162/089120102762671990. URL https://doi.org/10.1162/089120102762671990.

Edoardo Maria Ponti, Goran Glavaš, Olga Majewska, Qianchu Liu, Ivan Vulić, and Anna Korhonen. Xcopa: A multilingual dataset for causal commonsense reasoning, 2020. URL https://arxiv.org/abs/2005.00333.

Ofir Press and Lior Wolf. Using the output embedding to improve language models, 2016. URL https://arxiv.org/abs/1608.05859.

Ye Qi, Devendra Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. When and why are pre-trained word embeddings useful for neural machine translation? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 529–535, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-2084. URL https://aclanthology.org/N18-2084.

Ella Rabinovich, Yulia Tsvetkov, and Shuly Wintner. Native language cognate effects on second language lexical choice. *Transactions of the Association for Computational Linguistics*, 6:329–342, 2018. doi: 10.1162/tacl_a_00024. URL https://aclanthology.org/Q18-1024.

Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

Afshin Rahimi, Yuan Li, and Trevor Cohn. Massively multilingual transfer for ner, 2019. URL https://arxiv.org/abs/1902.00193.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters, 2017.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019. URL https://arxiv.org/abs/1908.10084.

Nils Reimers and Iryna Gurevych. Making monolingual sentence embeddings multilingual using knowledge distillation. pages 4512–4525, 01 2020. doi: 10.18653/v1/2020.emnlp-main.365.

Melissa Roemmele, Cosmin Bejan, and Andrew Gordon. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. 01 2011.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2015.

Sebastian Ruder. Neural transfer learning for natural language processing. 2019. URL http://ruder.io/thesis/neural_transfer_learning_for_nlp.pdf.

Sebastian Ruder, Anders Søgaard, and Ivan Vulić. Unsupervised cross-lingual representation learning. In *Proceedings of ACL 2019, Tutorial Abstracts*, pages 31–38, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1454. URL https://aclanthology.org/D19-1454.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2015. URL https://arxiv.org/abs/1508.07909.

Laura Smith, Salvatore Giorgi, Rishi Solanki, Johannes Eichstaedt, H. Andrew Schwartz, Muhammad Abdul-Mageed, Anneke Buffone, and Lyle Ungar. Does 'well-being' translate on Twitter? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2042–2047, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1217. URL https://aclanthology.org/D16-1217.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D. Lee. Theoretical insights into the optimization landscape of over-parameterized shallow neural networks, 2018.

Samuel Stanton, Pavel Izmailov, Polina Kirichenko, Alexander A. Alemi, and Andrew Gordon Wilson. Does knowledge distillation really work?, 2021.

Asa Cooper Stickland and Iain Murray. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning, 2019.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. Patient knowledge distillation for bert model compression, 2019.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices, 2020.

Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. pages 1017–1024, 01 2011.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Xu Tan, Yi Ren, Di He, Tao Qin, Zhou Zhao, and Tie-Yan Liu. Multilingual neural machine translation with knowledge distillation, 2019.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. Distilling task-specific knowledge from bert into simple neural networks, 2019.

Rachael Tatman. Gender and dialect bias in YouTube's automatic captions. In *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pages 53–59, Valencia, Spain, April 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-1606. URL https://aclanthology.org/W17-1606.

François Torregrossa, Allesiardo Robin, Vincent Claveau, Nihel Kooli, and Guillaume Gravier. A survey on training and evaluation of word embeddings. *International Journal of Data Science and Analytics*, 11:1–19, 03 2021. doi: 10.1007/s41060-021-00242-8.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Well-read students learn better: On the importance of pre-training compact models, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. Multilingual is not enough: Bert for finnish, 2019.

Ivan Vulić and Marie-Francine Moens. Monolingual and cross-lingual information retrieval models based on (bilingual) word embeddings. In *Proceedings*

of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '15, page 363–372, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336215. doi: 10.1145/2766462.2767752. URL https://doi.org/10.1145/2766462.2767752.

Ivan Vulić, Goran Glavaš, Roi Reichart, and Anna Korhonen. Do we really need fully unsupervised cross-lingual embeddings?, 2019. URL https://arxiv.org/abs/1909.01638.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding, 2019.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020.

Xiangpeng Wei, Rongxiang Weng, Yue Hu, Luxi Xing, Heng Yu, and Weihua Luo. On learning universal representations across languages, 2021.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://aclanthology.org/2020.lrec-1.494.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. A broad-coverage challenge corpus for sentence understanding through inference, 2017. URL https://arxiv.org/abs/1704.05426.

Shijie Wu and Mark Dredze. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 833–844, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1077. URL https://aclanthology.org/D19-1077.

Shijie Wu, Alexis Conneau, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. Emerging cross-lingual structure in pretrained language models, 2019. URL https://arxiv.org/abs/1911.01464.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016. URL https://arxiv.org/abs/1609.08144.

Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1006–1011, Denver, Colorado, May–June 2015. Association for Computational Linguistics. doi: 10.3115/v1/N15-1104. URL https://www.aclweb.org/anthology/N15-1104.

Jing Yang, Brais Martinez, Adrian Bulat, and Georgios Tzimiropoulos. Knowledge distillation via softmax regression representation learning. In *ICLR2021*, 2021.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models, 2021. URL https://arxiv.org/abs/2106.10199.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with bert, 2019. URL https://arxiv.org/abs/1904.09675.

Alina Zhiltsova, Simon Caton, and Catherine Mulway. Mitigation of unintended biases against non-native english texts in sentiment analysis. In *AICS*, 2019.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Master-/Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Mannheim, den 28.05.2022 Unterschrift