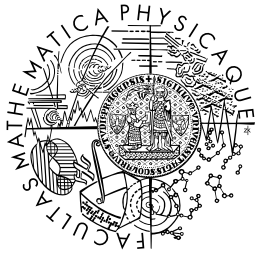


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Ondřej Cífka

Continuous Sentence Representations in Neural Machine Translation

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	RNDr. Ondřej Bojar, Ph.D.
Study programme:	Computer Science
Study branch:	Computational Linguistics

Prague 2018

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ..... signature of the author

Title: Continuous Sentence Representations in Neural Machine Translation

Author: Ondřej Cífka

Institute: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Ondřej Bojar, Ph.D., Institute of Formal and Applied Linguistics

Abstract: Recent advances in natural language processing using neural networks have given rise to numerous methods of obtaining continuous-space vector representations of textual data that can be exploited for various applications. One of these methods is to use internal representations learned by neural machine translation (NMT) models. However, the attention mechanism in modern NMT systems removes the single point in the neural network from which the source sentence representation can be extracted. In this thesis, we propose and empirically evaluate novel ways to remove this limitation. We review existing methods of obtaining sentence representations and evaluating them, and present novel intrinsic evaluation metrics. Next, we describe our modifications to attention-based NMT architectures that allow extracting sentence representations. In the experimental section, we analyze these representations and evaluate them using a wide range of metrics with a focus on meaning representation. The results suggest that the better the translation quality, the worse the performance on these tasks. We also observe no performance gains from using multi-task training to control the representations.

Keywords: sentence representation neural machine translation

Contents

Introduction

In recent years, deep learning techniques have transformed many areas of computer science and artificial intelligence. One of the fields that benefit from the advances in deep learning is machine translation (MT), where remarkable improvements in translation quality have been enabled by neural machine translation (NMT) models.

An important feature of deep neural networks is their ability to automatically learn continuous representations of data. While such representations are more nuanced than hand-crafted features and allow to capture more complex dependencies present in the data, they are not directly interpretable. It is therefore unclear whether the superior results achieved by NMT systems are due to more abstract, meaning-oriented representations, or simply thanks to the ability to learn a more accurate but superficial mapping.

This thesis aims to shed some light on this question by exploring the representations learned by NMT systems. Our task is complicated by the fact that the most widely used sentence representations are fixed-dimension vectors, while in modern NMT systems using attention, such representations are not available. For this reason, previous research aiming to evaluate sentence representations in NMT has avoided these attention-based systems. In the present work, we propose novel sequence-to-sequence architectures that address this issue. Our most important contribution is the introduction of *compound attention*, which modifies the widely used attention mechanism to make it compatible with a fixed-size sentence representation.

We evaluate the representations using a wide range of automatic metrics with a focus on meaning representation. We find that most of the metrics correlate negatively with translation quality.

We also experiment with multi-task training as a way to steer the sentence representations toward state-of-the-art sentence embeddings, but we do not observe a consistent improvement in the evaluation metrics as a result of this training.

The thesis is structured as follows. In Chapter 1, we present different criteria that can be used for evaluating sentence representations and in Chapter 2, we review existing techniques for obtaining such representations. In Chapter 3, we give a background of the most popular NMT models and continue in Chapter 4 by describing our proposed modifications to these models. Chapter 5 details our experiments and presents their results.

Related work

The properties of continuous sentence representations have always been of interest to researchers working on neural machine translation. In the first works on RNN sequence-to-sequence models, Cho et al. (2014b) and Sutskever et al. (2014) provided visualizations of the phrase and sentence embedding spaces and observed that they capture some semantic and syntactic structure.

Hill et al. (2016) perform a systematic evaluation of sentence representation in different models, including NMT, by applying them to various sentence classification tasks and by relating semantic similarity to closeness in the representation

space.

Shi et al. (2016) investigate the syntactic properties of representations learned by NMT systems by predicting sentence- and word-level syntactic labels (e.g. tense, part of speech) and by generating syntax trees from these representations.

Schwenk and Douze (2017) aim to learn language-independent sentence representations using NMT systems with multiple source and target languages. They evaluate primarily by similarity scores of the learned representations for similar sentences (within or across languages).

All of the above work builds on non-attentive models and deliberately avoids using attention for reasons already stated.

1. Evaluating sentence representations

This chapter describes possible techniques for evaluating the quality of sentence representations. Some of the techniques are inspired by work on evaluating word embeddings. For a recent survey of word embedding evaluation methods, see Bakarov (2018).

1.1 Extrinsic evaluation

When on the quest for a universal sentence representation, the most natural evaluation criterion is the performance of our sentence embeddings on different NLP tasks. This type of evaluation is sometimes referred to as *extrinsic*. (Also, this is related to the area of *transfer learning*, where features pre-trained on one task are used for a possibly related but different task.) We could therefore base our evaluation protocol on as many tasks as possible, with complexity ranging from e.g. sentiment classification to machine translation.

However, tasks that require complex prediction models can be expected to be less reliable as indicators of the quality of representations. For example, we might choose to train a neural machine translation model with our source-language sentence embeddings as inputs, and evaluate its translation quality using a metric such as BLEU. There are two issues with this approach. Firstly, besides evaluating the sentence representation, our metric would also measure the ability of our NMT model to make use of the information encoded in this representation, which can vary greatly depending on the model architecture and the optimization method. Consequently, we might learn more about the fitness of our particular training setup to the given representation than about the representation itself. Secondly, machine translation is difficult to evaluate (naturally occurring sentences can have billions of correct translations, as shown by Dreyer and Marcu, 2012; Bojar et al., 2013) and all currently known automatic metrics are but poor proxies to human judgment. This also applies to other sequence prediction and structured prediction tasks. In contrast, simple sentence-level classification and regression tasks are straightforward to evaluate and permit the use of linear prediction models, which are easier to optimize and can be interpreted geometrically.

1.1.1 SentEval

Recently, a set of such tasks has emerged as a commonly used benchmark for sentence embeddings (see e.g. Kiros et al., 2015; Conneau et al., 2017a). An open-source tool implementing this benchmark has been released under the name *SentEval*.[1]

Most of the tasks included in this benchmark are of semantic nature. A notable example is **natural language inference** (NLI), a classification task where the goal is to predict for a pair of sentences – a *premise* and a *hypothesis* – whether

[1] `https://github.com/facebookresearch/SentEval`

the premise implies the hypothesis (*entailment*), denies it (*contradiction*), or neither (*neutral*). To predict a label from a pair of sentence embeddings u and v, the embeddings are concatenated with their element-wise product $u \odot v$ and absolute element-wise difference $|u - v|$. The resulting feature vector $(u, v, u \odot v, |u - v|)$ is then used as input to a logistic regression classifier. The fact that the premise and the hypothesis are processed independently and only combined just before the final prediction makes this task very challenging. It means that the underlying model needs to be able to parse the semantics of each sentence and represent it in a way that allows meanings to be compared using simple mathematical operations.

Another relevant task is **semantic relatedness** or **similarity**.[2] Datasets for this task consist of pairs of sentences labeled with a real-valued similarity score (from human raters), and the goal is to predict the score given the sentence pair. SentEval follows the setup of Tai et al. (2015) where the features are the element-wise product of the respective embeddings and their absolute difference, i.e. $(u \odot v, |u - v|)$. A softmax layer is used to predict a distribution of integer scores from these features, and the expected value of this distribution is computed to obtain the similarity score.

SentEval also includes a metric that measures the similarity between u and v geometrically instead of relying on a prediction model. This is discussed in Section 1.2.3.

Table 1.1 describes all SentEval classification tasks (on sentence pairs as well as single sentences) and Table 1.2 lists the similarity tasks. See Dolan et al. (2004) for details on the MRPC task and Hill et al. (2016) for the remaining tasks.

1.2 Intrinsic evaluation

Apart from providing a performance metric, linear prediction models also have a straightforward geometric interpretation: they attempt to find subspaces or 'directions' in the representation space that correspond to certain linguistic phenomena. Geometric properties like this can also be investigated more directly using so-called *intrinsic evaluation*. In general, intrinsic evaluation consists in defining properties that are deemed desirable and measuring the degree to which these properties hold; it may not always be clear whether such properties are helpful for downstream tasks and the metrics should therefore be checked for correlation with extrinsic ones.

1.2.1 Analogy completion with offset vectors

One technique, commonly applied to word embeddings, is evaluation using *analogies* (Mikolov et al., 2013d). It is based on the (purported) observation that certain semantic and syntactic relations between words can be expressed using simple vector arithmetics. A typical analogy question has the form '*a* is to *b* as *c* is to *d*' where *d* is unknown, e.g. '*man* is to *woman* as *king* is to __', the correct

[2]Semantic *relatedness* and *similarity* are distinct concepts: similarity refers simply to the degree of semantic equivalence, while relatedness is a more general term encompassing different types of semantic relations. SentEval contains both similarity tasks (STS) and a relatedness task (SICK-R). In the following, we use the expression 'similarity tasks' to refer to all of them.

Name	Cl.	Data size		Task type and example
		train	test	
MR	2	11k	—	sentiment (movies) *an idealistic love story that brings out the latent 15-year-old romantic in everyone.* (+)
CR	2	4k	—	product review polarity *no way to contact their customer service.* (−)
SUBJ	2	10k	—	subjectivity *a little weak – and it isn't that funny.* (subjective)
MPQA	2	11k	—	opinion polarity *was hoping* (+), *breach of the very constitution* (−)
SST2	2	68k	2k	sentiment (movies) *contains very few laughs and even less surprises* (−)
SST5	5	10k	2k	sentiment (movies) *it's worth taking the kids to.* (4)
TREC	6	5k	500	question type *What was Einstein's IQ?* (NUM)
MRPC	2	4k	2k	semantic equivalence *Lawtey is not the first faith-based program in Florida's prison system. / But Lawtey is the first entire prison to take that path.* (−)
SNLI	3	559k	10k	natural language inference *Two doctors perform surgery on patient. / Two surgeons are having lunch.* (contradiction)
SICK-E	3	5k	5k	natural language inference *A group of people is near the ocean / A crowd of people is near the water* (entailment)

Table 1.1: SentEval classification tasks with examples. 'Train' includes validation data. Tasks without a test set use 10-fold cross-validation.

Name	Data size		Method
	train	test	
SICK-R	5k	5k	regression
STSB	7k	1k	regression
STS12	—	3k	cosine similarity
STS13	—	2k	cosine similarity
STS14	—	4k	cosine similarity
STS15	—	9k	cosine similarity
STS16	—	9k	cosine similarity

Table 1.2: SentEval semantic similarity (STS) and relatedness (SICK-R) tasks. 'Train' includes validation data.

answer being *queen*. Mikolov et al. assume that the relation that underlies this analogy corresponds to a vector offset in the embedding space, so the analogy can be expressed as

$$v_b - v_a \approx v_d - v_c, \tag{1.1}$$

where v_w is the embedding vector of a word w. Hence, to answer the analogy question, we can compute $v^* = v_c + v_b - v_a$ and find the word d^* whose embedding is most similar to v^* in terms of cosine similarity:

$$d^* = \arg\max_d \cos(v_d, v^*) = \arg\max_d \frac{v_d^\top v^*}{\|v_d\|\|v^*\|}. \tag{1.2}$$

Although this idea has become very popular and was probably one of the things that made word embeddings an attractive topic, recent research has shown that it has a number of hidden flaws. Firstly, Levy and Goldberg (2014) point out that for normalized vectors (as in Mikolov et al.), the method is equivalent to seeking a word d which is close to b and c in the embedding space while also being distant from a:

$$d^* = \arg\max_d \Big(\cos(v_d, v_b) + \cos(v_d, v_c) - \cos(v_d, v_a)\Big). \tag{1.3}$$

In this reframing, it becomes clear that this method can be somewhat successful in solving analogies even if they do not exactly correspond to vector offsets.

More importantly, Levy and Goldberg also mention the fact (omitted by Mikolov et al.) that the method does not search among the 3 vectors that appear in the analogy question; that is, the argmax in Eqs. (1.2) and (1.3) is actually over $d \notin \{a, b, c\}$. Rogers et al. (2017) demonstrate that this trick is in fact crucial for achieving a decent accuracy: in the 'honest' version of the method which searches over all words in the vocabulary, the retrieved word d^* is almost always equal to either b or c. For example, the word closest to '*king* − *man* + *woman*' is not *queen* but *king* again. This suggests that *queen* is already close to *king* in the embedding space (as is *woman* to *man*) and the method simply exploits this fact.

In addition to these flaws, there are two issues with applying this technique to sentence embeddings. Firstly, while we can reasonably assume a closed, finite vocabulary, we can never enumerate all possible sentences (or compute distances to them), and (1.2) is therefore infeasible. We can of course approximate the solution on a sufficiently small sample, but the result will vary hugely depending on its size, with performance declining as we add more data points.

The second issue is a lack of datasets for this task. Guu et al. (2017) tackle this problem by generating an artificial sentence analogy dataset with the help of existing sets of word analogies. The idea is to mine a text corpus for pairs of sentences that differ in words which are part of a word analogy.

1.2.2 Domain alignment

The just described approach consists in finding the offset vector for a pair of data points and checking whether it holds for another pair which is in some sense analogous. A possible generalization of this technique (illustrated in Fig. 1.1) is to learn a general linear or affine transformation of embeddings from one *domain*

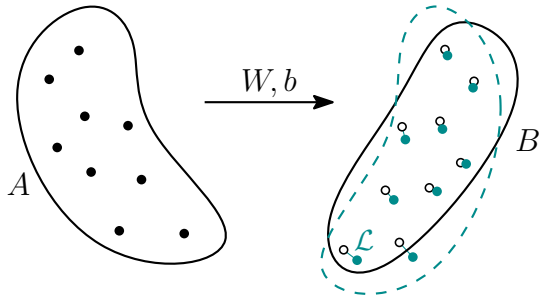


Figure 1.1: Learning an affine mapping from domain A to domain B. W (weight matrix) and b (bias vector) are the parameters of this mapping.

to another. By domains, we broadly mean distinct sets of sentences such that the sentences in each set share a common characteristic and there exists a meaningful and reasonably well-defined mapping from one set to the other. For example, we could attempt to find a mapping from informal to formal sentences, from Twitter posts to newspaper headlines, from negative movie reviews to positive. If we found such a mapping, we could conclude that the two domains have a similar structure in the embedding space – in other words, that the representation is consistent across domains. A major obstacle is once again the lack of parallel data that would pair sentences from the different domains.

In a bilingual context, we can consider languages to be the domains, and search for a transformation that would translate words or sentences from a source language to a target language. In this case, the necessary data is readily available. Mikolov et al. (2013b) learn a 'translation matrix' W that maps word embeddings from one language to another:

$$\arg\min_{W} \sum_{i=1}^{n} \|W v_{a_i} - v_{b_i}\|^2. \tag{1.4}$$

To translate a given word a from the source language, its embedding is multiplied by W and the closest target-language embedding is retrieved, again using cosine similarity as the metric. (Note that since the training objective (1.4) is based on L_2 distance, it might be more appropriate to use L_2 distance also for retrieving the translation. See Section 1.2.5 for a discussion of the relationship between L_2 distance and cosine similarity.)

For entire sentences, we cannot expect to be able to accurately map the source- and target-language embedding spaces to each other using something as simple as an affine transformation, unless the embeddings were trained jointly with this goal in mind (as in Schwenk and Douze, 2017). However, as hinted above, we can use the same technique in a monolingual embedding space to find a mapping between two domains in the same language, provided that we have the parallel data necessary to learn the mapping.

The technique as just described is *supervised* because of the need for parallel data. One could also learn the mapping in an *unsupervised* way (using unpaired data from both domains) by means of adversarial training. Again, a similar approach has been applied to word translation (Conneau et al., 2017b).

1.2.3 Semantic similarity as vector similarity

A natural property to demand from a sentence representation space is that embeddings of semantically similar sentences lie close to each other, and conversely, embeddings of semantically dissimilar sentences lie far apart. Such a sentence representation would permit to measure semantic similarity directly using a suitable vector similarity metric, such as cosine similarity. A simple way of quantifying this property on semantic similarity datasets (used for word representations since Harris, 1954) is to compute the similarity scores for all test pairs and correlate them with the human judgments using Pearson or Spearman correlation. This method is adopted by the SentEval benchmark (see Section 1.1).

In the case of word embeddings, Faruqui et al. (2016) discuss several problems with this evaluation (e.g. unclear definition, low correlation with extrinsic metrics and lack of statistical significance) and argue against using it.

1.2.4 Clusters of paraphrases

Another way of investigating the relation between semantic and geometrical proximity is by looking at paraphrases, i.e. sentences with identical or near-identical meanings. Suppose we possess a corpus where sentences are grouped by their meaning, so that each sentence is a paraphrase of every other sentence in the same group. If our goal is to be able to distinguish between meanings geometrically, then each group of sentences should constitute a distinct, coherent cluster in the embedding space. To assess the extent to which this property holds, we can either gauge it directly using a suitable clustering validation measure, or by indirect means such as a classification task.

Internal clustering measures. Internal clustering validation measures are used for evaluating the quality of clustering, i.e. assignment of data points to clusters, in cases where the correct assignment is not known. Usually, these measures combine two criteria: the compactness of the clusters and their mutual separation. In our setting, the situation is reversed (the assignment to clusters is fixed and the points themselves are not) but our criteria are the same. However, we need to be careful and choose a measure that is comparable when the positions of the data points are different; for example, it needs to be invariant to scaling.

One such commonly used metric is the *Davies-Bouldin index* (Davies and Bouldin, 1979), defined as follows. For every pair of clusters, we compute the ratio R_{ij} of their combined scatter $S_i + S_j$ and the L_2 distance of their centroids d_{ij}:

$$R_{ij} = \frac{S_i + S_j}{d_{ij}}. \tag{1.5}$$

Here, S_i is the scatter of the i^{th} cluster, i.e. the average L_2 distance of its members to its centroid. This can be interpreted as measuring the mutual 'entangledness' of the two clusters. Then, we pick the maximal (i.e. worst-case) value for each cluster, and average the results:

$$\text{DB} = \frac{1}{N} \sum_{i=1}^{N} \max_{j \neq i} R_{ij}. \tag{1.6}$$

Therefore, the lower the Davies-Bouldin index, the better the mutual separation of the clusters.

Many other internal clustering validation measures exist; see Liu et al. (2010) for a detailed study.

Paraphrase retrieval. We would like to verify whether nearest neighbors in the sentence embedding space correspond to paraphrases. A straightforward way, originally proposed for evaluating the extraction of synonymous words by Curran and Moens (2002), is to retrieve the k nearest neighbors for each sentence and calculate precision, i.e. the average fraction of paraphrases among the k neighbors. It could also be useful to calculate recall (the fraction of retrieved paraphrases out of all paraphrases). Note that for a particular value of k, it usually does not make sense to report both: precision should be used when k is less than the number of paraphrases present in the corpus for each sentence, and recall is appropriate when k is larger.

This technique suffers from a similar problem as the analogy completion technique described in Section 1.2.1: for datasets where it is not guaranteed that different clusters correspond to different meanings, it can happen that a highly relevant sentence from a different cluster will be closer than a loose paraphrase from the same cluster, but will not be actually counted as a paraphrase. A similar effect is noted by Schwenk and Douze (2017), who use a nearest neighbor approach to retrieve translations using multilingual sentence embeddings.

Classification into clusters. For $k = 1$, the paraphrase retrieval technique can be thought of as 1-NN classification (with leave-one-out cross-validation). The idea can be extended to other classification methods: choose a subset of embeddings from each cluster, train a classifier on them and evaluate on the rest of the embeddings. Especially well-suited for this purpose are classifiers like LDA and softmax classifiers that can easily handle a large number of classes.

1.2.5 Choosing a distance metric

All evaluation methods presented in this section relied on a similarity measure or a distance metric. Common choices are cosine similarity and L_2 distance, respectively. While both in principle serve the same purpose, each has its advantages and drawbacks. For the sake of comparing these two measures, let us replace cosine similarity with *cosine distance*, defined as

$$d(u, v) = 1 - \cos(u, v) = 1 - \frac{u^\top v}{\|u\|\|v\|}. \tag{1.7}$$

Contrary to its name, cosine distance does not satisfy the formal definition of a distance metric, violating the requirement that two vectors have zero distance only if they are identical.[3] This is because cosine distance essentially normalizes its inputs, making all vectors with the same direction indistinguishable.

In spite of this drawback, cosine distance has become the de-facto standard for word embeddings. This could be because it is computationally inexpensive, and

[3]Moreover, cosine distance violates the triangle inequality, but this not an issue when it is only used to rank vectors by distance to a fixed point.

because word embeddings are often normalized, in which case the cosine distance is proportional to the square of the L_2 distance:

$$\begin{aligned} \|u - v\|^2 &= (u - v)^\top (u - v) \\ &= u^\top u - 2u^\top v + v^\top v \\ &= \|u\| - 2u^\top v + \|v\| \\ &= 2(1 - \cos(u, v)). \end{aligned} \tag{1.8}$$

This means that for ranking normalized vectors, cosine distance and L_2 distance will give the same results.

A reason why cosine distance might work well in practice is that the norms of embedding vectors may not strongly reflect semantics. For example, it has been shown (Schakel and Wilson, 2015) that the norm of a word2vec embedding (Mikolov et al., 2013a,c) is largely determined by the word's frequency and the diversity of contexts in which it is used. In this case, using L_2 distance would actually do harm. It is not clear whether this can be generalized to any neural representation of words or sentences; however, Schwenk and Douze (2017) report that cosine distance consistently works better even for sentence embeddings obtained from NMT.

To conclude this discussion, different distance metrics should be further investigated, and we cannot give a clear recommendation as to which one to use for a given application. In our experiments, we employ both metrics in parallel wherever it seems appropriate.

2. Representations of sentence meaning

This chapter focuses on different methods of obtaining representations of sentences and their meaning. We begin with a brief detour into symbolic representations in Section 2.1 and continue with methods of combining symbolic and vector-space representations in Section 2.2. The rest of the chapter (Section 2.3) describes architectures of neural sentence encoders and methods of using them to learn vector-space representations.

2.1 Symbolic representations

In formal semantics, propositions are commonly represented as logical forms (Montague, 1970), usually in the language of first-order (predicate) logic. Simple natural language expressions are represented either as symbols of this formal language (terms, predicates, logical operators and quantifiers) or as lambda abstractions. This allows to build this representation according to the *principle of compositionality*, which states that the meaning of a compound expression (e.g. a phrase) is a function of the meanings of its constituents.

In contrast to continuous vector representations of sentences, logical forms are discrete objects of variable size (depending on the complexity of the represented sentence). They are well suited for automatic reasoning over a knowledge base.

The process of converting a natural language sentence to a logical form is called *semantic parsing*. Grammars that adhere to the principle of compositionality (such as Combinatory Categorial Grammar, see Section 2.2) build a symbolic representation of meaning together with the syntactic structure. However, a description of methods used for semantic parsing is outside the scope of this thesis.

2.2 Compositional distributional semantics

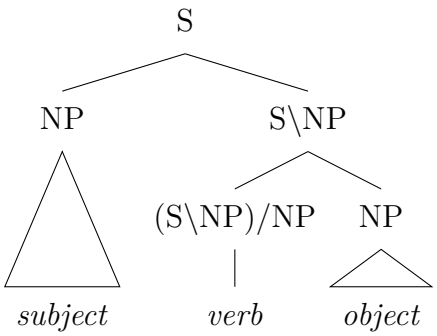
While learning vector-space sentence representations is closely linked to the recent advances in deep learning, vector-space representations of words have a longer history, having first been studied within the framework of *distributional semantics* (Deerwester et al., 1990; Lund and Burgess, 1996; Schütze, 1998). Based on the idea that words occurring in similar contexts have similar meanings (the so-called *distributional hypothesis*; Harris, 1954), distributional approaches form word vectors by collecting word co-occurrence statistics.

Compositional distributional semantics (CDS) refers to a set of recent efforts to unify distributional and formal semantics by building sentence representations from word vectors using a composition operation. CDS is motivated by the distributional hypothesis and aims to extend it from words to phrases. As such, it is somewhat opposed to deep learning techniques, which simply optimize sentence representations towards performance on a particular task or set of tasks. Nevertheless, some neural network architectures (namely recursive neural networks;

see Section 2.3.1) bear similarities to CDS models.

In the simplest of CDS models (Mitchell and Lapata, 2008, 2010), the composition operation is realized as addition (possibly weighted), element-wise multiplication, or a combination of both.

A much more sophisticated theory of compositional distributional semantics, the so-called *categorial framework* (Coecke et al., 2010; Maillard et al., 2014), is based on the formalism of *combinatory categorial grammar* (CCG; Steedman, 2000). In CCG, each phrase is assigned a syntactic type that directly encodes its ability to combine with other phrases. There is a small number of primitive types – such as S (sentence) and NP (noun phrase) – and these are used to derive complex types. A complex type is denoted as X/Y or $X\backslash Y$, which means that a phrase of this type can combine with an adjacent phrase of type Y and this operation (called *application*) will yield a phrase of type X. The direction of the slash determines whether Y should appear to the left (backslash) or to the right (forward slash) of the complex type. For example, intransitive verbs in English would have the type S\NP ('an S missing an NP on the left') because they can take a noun phrase (a subject) from the left to form a sentence. Transitive verbs could be denoted as (S/NP)\NP, requiring an object to the right and a subject to the left:



In accordance with the principle of compositionality, the operations in CGG have an explicit semantic interpretation, allowing to associate each phrase with a formal representation of its meaning. The idea that allows to replace this formal representation with a distributional one is to pair each syntactic type with a distinct *tensor space* in which the distributional representations live. Primitive types such as S and NP are associated with spaces of first-order tensors (vectors) and tensor spaces of complex types are obtained using the *tensor product*. For example, the tensor space corresponding to the category of intransitive verbs S\NP would be the tensor product space $\mathsf{S} \otimes \mathsf{NP}$, whose elements are matrices (second-order tensors). These higher-order tensors are then treated as linear maps that can be applied to other tensors by means of the *tensor contraction* operation (a generalization of matrix multiplication). For instance, a vector $u \in \mathsf{NP}$ (representing a noun phrase) can be multiplied by a matrix $A \in \mathsf{S} \otimes \mathsf{NP}$ (representing an intransitive verb or a verb phrase), yielding a sentence vector $Au \in \mathsf{S}$. For more details on the different operations in CCG and how they are realized in this framework, see Maillard et al. (2014).

We have given a very condensed account of how meanings of phrases are represented in the categorial framework and how they are combined to form the representations of complex phrases. We still owe an explanation of how to obtain

the tensor representations of words. The framework, however, is rather abstract and doesn't give a concrete recipe. One option is to compute the vector representations of primitive types first (in the standard distributional way, from word co-occurrence counts) and then use linear regression to learn the representations of complex types. There is an obvious issue with this method: because the category of sentences S is necessarily a primitive type in CCG, we would need to be able to compute representations of entire sentences before we could find the representations of categories such as verbs. There have been attempts to remedy this problem (Grefenstette et al., 2011; Kartsaklis et al., 2012), but to our knowledge, no complete and practically useful implementation of the framework exists at the time of this writing.

2.3 Deep learning methods

In this section, we will give an account of different approaches to learning sentence representations, based on supervised or unsupervised deep learning methods.

2.3.1 Encoder architectures

All deep neural networks compute many intermediate numerical representations of the input. Our focus here is on representations that (1) have a fixed size and (2) completely separate the input layer from the output layer, and thus capture all information relevant for producing the output. In networks where such a representation exists, the 'subnet' that computes it is referred to as the *encoder*.

In neural networks, words are commonly represented as real-valued vectors called *word embeddings*. We can either learn these vectors as parameters of the network or use embeddings obtained using algorithms such as word2vec (Mikolov et al., 2013a) or GloVe (Pennington et al., 2014) or by means of distributional semantics (see Section 2.2). The job of the encoder is to combine the sequence of word embeddings for a given sentence into one vector.[1]

As in compositional distributional semantics, the simplest way to combine word embeddings is using a simple mathematical operation such as addition or averaging. In *deep averaging networks* (Iyyer et al., 2015), averaging of word embeddings is followed by one or more feed-forward layers. Such approaches are generally inadequate for meaning representation since they ignore the syntactic structure of sentences. Despite this defect, they have been shown to outperform more complex models in some out-of-domain scenarios (Wieting et al., 2015).

Recurrent neural networks

The most commonly used encoder architectures for text and sequential data in general are *recurrent neural networks* (RNNs). An RNN consumes the input

[1]To reduce the vocabulary size and to deal with rare words, sentences are often presented to the network as sequences of *subword units* (Sennrich et al., 2016b) or even individual characters. The encoder architectures described in this section are largely oblivious to the choice of input units, and we will therefore assume that we are working with words. However, encoders that make use of external linguistic knowledge such as syntax are usually more naturally applied to words than to subword units.

sequence (i.e. the sequence of word embeddings) from left to right, updating its hidden state in each step. In general, an RNN cell receives the input x_t at time t and its previous hidden state h_{t-1} and uses them to compute the next hidden state h_t. Formally,

$$h_t = f(x_t, h_{t-1}), \tag{2.1}$$

where f is a function whose parameters are learned from data and shared across all time steps. The last encoder state h_T (produced after consuming all T input words) can then be considered a representation of the sentence. More sophisticated ways of computing a sentence representation from the RNN states will be discussed later.

Different types of RNN cells – i.e. different implementations of the function f from Eq. (2.1) – have been proposed, the most popular one being *long short-term memory* (LSTM; Hochreiter and Schmidhuber, 1997; Gers et al., 2000, 2002), designed to capture long-range dependencies and to have stable gradients for training by back-propagation. One of the most general variants of LSTMs (Gers et al., 2002) is composed of a so-called *memory cell* c_t and three *gates*: the *input gate*, *forget gate* and *output gate*, whose activations are denoted by i_t, f_t and o_t, respectively. The value of the hidden state h_t is computed as follows (formulas adapted from Graves et al., 2013):

$$i_t = \sigma(W_i[x_t, h_{t-1}, c_{t-1}] + b_i), \tag{2.2}$$
$$f_t = \sigma(W_f[x_t, h_{t-1}, c_{t-1}] + b_f), \tag{2.3}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c[x_t, h_{t-1}] + b_c), \tag{2.4}$$
$$o_t = \sigma(W_o[x_t, h_{t-1}, c_t] + b_o), \tag{2.5}$$
$$h_t = o_t \odot \tanh(c_t). \tag{2.6}$$

The hidden state of an LSTM, in the sense of Eq. (2.1), is therefore not just the vector h_t, but the tuple (h_t, c_t).

The LSTM cell was greatly simplified by Cho et al. (2014b), resulting in the *gated recurrent unit* (GRU). The hidden state and the memory cell are merged into one vector h_t and there are only two gates: the *update gate* and the *reset gate*, denoted by z_t and r_t, respectively. The GRU cell operates as follows:

$$r_t = \sigma(W_r[x_t, h_{t-1}]), \tag{2.7}$$
$$z_t = \sigma(W_z[x_t, h_{t-1}]), \tag{2.8}$$
$$\tilde{h}_t = \tanh(W_{\tilde{h}}[x_t, r_t \odot h_{t-1}]), \tag{2.9}$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t. \tag{2.10}$$

Both gates essentially allow to drop information from the previous hidden state and replace it with new information.

Many other variations on LSTM exist. Józefowicz et al. (2015) empirically evaluated more than ten thousand different RNN architectures and found one that outperforms LSTM and GRU on some but not all tasks.

RNNs can also be used to process the input from right to left. In *bidirectional recurrent neural networks* (BRNN or BiRNN; Schuster and Paliwal, 1997), a left-to-right (forward) and a right-to-left (backward) RNN are used to encode the

input independently:

$$\overrightarrow{h_t} = \overrightarrow{f}(x_t, \overrightarrow{h_{t-1}}), \tag{2.11}$$

$$\overleftarrow{h_t} = \overleftarrow{f}(x_t, \overleftarrow{h_{t+1}}). \tag{2.12}$$

A representation of the sentence can then be obtained by concatenating the final hidden states in both directions: $[\overrightarrow{h_T}, \overleftarrow{h_1}]$. For some purposes, it is useful to concatenate the forward and backward hidden state at each time step and treat them as one RNN state: $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$.

Deep recurrent neural networks (Graves et al., 2013; Zhou et al., 2016) make use of multiple RNNs stacked on top of each other, with the hidden states of each layer becoming the inputs to the next layer. This enables the encoder to build increasingly abstract representations.

As mentioned above, the usual way to obtain a vector-space sentence representation using an RNN is to use the last encoder state h_T. Another option is to combine all encoder states by taking the average or maximum over time (sometimes called *pooling*; Collobert and Weston, 2008; Schwenk and Douze, 2017).

Inner attention

Instead of a simple average, we can compute one or more weighted averages of the inputs using an *inner attention*[2] mechanism (Liu et al., 2016; Li et al., 2016b; Lin et al., 2017). This allows the encoder to explicitly decide which parts of the sentence are important for constructing the sentence embedding. The mechanism is usually applied on top of the states of a bidirectional RNN, but in theory, it could also be applied to word embeddings.

A concrete implementation of inner attention will be described in detail in Section 4.1.1.

Recursive neural networks

Recurrent neural networks can be considered a special case of *recursive neural networks*, where the computation graph has a general tree structure. Such models are more linguistically adequate since instead of building the sentence representation in a linear fashion, they compose it from units with more well-defined meanings. Moreover, they enable long-range dependencies to be captured by a much smaller number of compositions.

The tree structure can be based on syntax, i.e. a dependency or constituency tree obtained using a parser (Socher et al., 2011, 2013), or learned by the model in a supervised way (Socher et al., 2010) or an unsupervised way (Yogatama et al., 2016). The structure could also be completely independent of the sentence (except for its length), e.g. a balanced binary tree with word embeddings as leaves. More generally, the encoder may have the structure of any directed acyclic graph (DAG), such as in grConv (*gated recursive convolutional network*; Cho et al., 2014a) or AdaSent (Zhao et al., 2015).

[2]In the literature, this and similar approaches have been termed *inner attention*, *self-attention* and *single-time attention*. We find the term *self-attention* too heavily overloaded and even misleading, and *single-time attention* somewhat cumbersome.

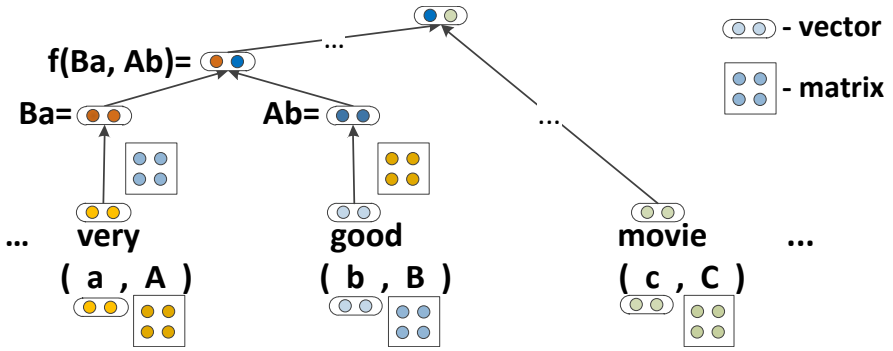


Figure 2.1: The recursive matrix-vector model (image from Socher et al., 2012, edited). The vector representation p of the phrase *very good* is computed as $f(Ba, Ab)$ where (a, A) is the vector-matrix representation of *very*, (b, B) is the representation of *good*, and f is a learned non-linear function. The matrix representation P of the phrase (not shown in the figure) is computed as $f_M(A, B)$ where f_M is a learned linear function.

In the most basic variants of recursive neural networks, the same operation is applied at every tree node, as in Socher et al. (2010) or more recently in Tai et al. (2015). Hermann and Blunsom (2013), whose approach is rooted in the formalism of combinatory categorial grammar (CCG; see Section 2.2), introduce distinct weight matrices for different combinatory rules or even different CCG categories.

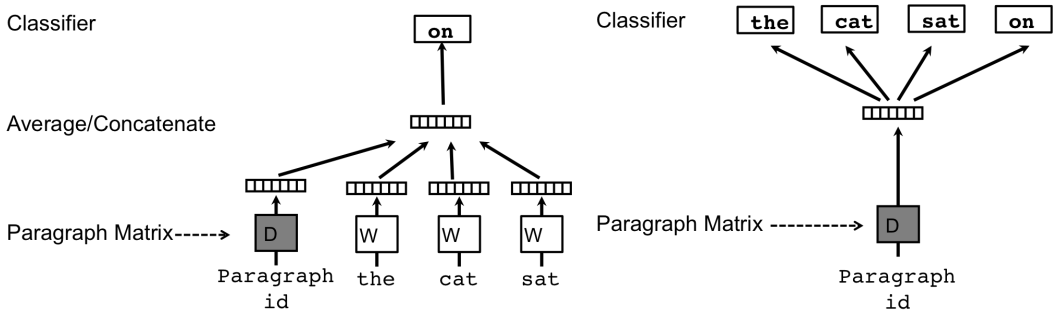
An even more fine-grained approach is taken by Socher et al. (2012), who use a different weight matrix for every word and, by composition, every phrase. Each word or phrase is therefore represented by a vector (which can be thought of as representing the actual meaning of the phrase) and a matrix (determining how it combines with other phrases). The composition operation is always binary. When forming a compound phrase, the matrix associated with each constituent is 'applied' to the other constituent's vector representation, and the results are combined to form the vector representation of the phrase; similarly, the matrix representation of the phrase is computed by combining the constituents' matrices. See Fig. 2.1 for details.

Convolutional neural networks

Convolutional neural networks (CNN) use convolution with a set of learned filters to compute intermediate representations. For text, the convolutions are usually computed along the time dimension, which corresponds to taking the dot product of a filter with each n-gram (n-tuple of adjacent word embeddings). Multiple convolutional layers can be applied consecutively, interleaved with pooling operations (e.g. max pooling) which serve to reduce the representation to a fraction of the original size.

The most common way to obtain a fixed size representation from a CNN is to take the maximum over time (Collobert et al., 2011). Most other methods described above (recurrent or recursive neural networks, inner attention) can in principle be used as well.

Notable examples of using convolutional networks to learn sentence representations are Kalchbrenner et al. (2014); Semeniuta et al. (2017); Conneau et al.



(a) PV-DM. The task is to predict the next word given the preceding words.

(b) PV-DBOW. The task is to predict words randomly sampled from the sentence.

Figure 2.2: Paragraph Vector models (images from Le and Mikolov, 2014). In the case of sentences, the 'paragraph matrix' D contains sentence embeddings as rows. W is the word embedding matrix.

(2017a). The aforementioned grConv and AdaSent can be seen as hybrids between recursive and convolutional networks.

2.3.2 Unsupervised methods

We now turn to methods for training neural networks for meaning representation, mostly employing the encoders described in the previous section. By unsupervised methods, we mean approaches that make use of unstructured text data without additional labeling. These methods can be further divided into two categories: those that work on isolated sentences alone and those that need to access some context (usually adjacent sentences) during training.

Paragraph Vector (Doc2Vec)

We start with a model by Le and Mikolov (2014) which is atypical in that it does not have an encoder, and instead learns sentence embeddings as parameters, i.e. as rows of a sentence embedding matrix D. Despite the name Paragraph Vector (or Doc2Vec[3]), it is applicable to texts of any length, ranging from phrases to entire documents.

Similarly to word embedding algorithms like CBOW and Skip-Gram (Mikolov et al., 2013a), this technique learns representations that are good at predicting the distribution of words in a context. Specifically, the PV-DM model (where DM stands for *distributed memory*) is trained on the task of predicting the next word in a sentence given the context of a fixed number of preceding words and the sentence embedding. In a variant of the model called PV-DBOW (*distributed bag of words*), the task is to guess words randomly sampled from the sentence, using only the sentence embedding as input. Both models are depicted in Fig. 2.2.

An important difference from word embedding algorithms is that in order to use the model, we need to be able to compute representations of sentences unseen in the training data. This is done by forming a new sentence embedding matrix D

[3]The popular name Doc2Vec refers to a concrete implementation: `https://github.com/RaRe-Technologies/gensim/blob/master/gensim/models/doc2vec.py`

for these unseen sentences and optimizing it using gradient descent while holding the rest of the parameters fixed.

Autoencoders

A popular class of models used for the unsupervised learning of representations, not only for text but also for other modalities like images and audio, are *autoencoders*. Their encoder is followed by a *decoder*, which is trained jointly with the encoder to reconstruct the input from the internal representation. The usual choice of decoder architecture for text is an RNN. (See Section 3.1 for an explanation of RNN decoders.)

Usually, learning to blindly copy the input will not result in useful features being captured. It is therefore common practice to deliberately constrain the representation in some way. One possibility is to make the dimension of the representation less than that of the input, resulting in a so-called *undercomplete* autoencoder. Note that this is always the case with sentences, since their length is potentially unlimited.

Another option is to *regularize* the model by adding a penalty term to the loss function. This term can, for example, encourage the sentence embedding vectors to follow an imposed prior distribution. This is the case with *variational autoencoders* (VAE; Kingma and Welling, 2013). In VAEs, the representation vector is treated as a latent variable z with a prior distribution $p(z)$ (usually a standard Gaussian). The decoder models the conditional distribution $p(x|z)$ and the encoder models the so-called *variational posterior* $q(z|x)$. We would like to maximize the marginal log-likelihood $\log p(x)$, but this is intractable. Instead, we train the model by maximizing a *variational lower bound*[4] on the log-likelihood:

$$\mathbb{E}_{q(z|x)}[\log p(x|z)] - \mathrm{KL}(q(z|x) \,||\, p(z)) \leq \log p(x), \tag{2.13}$$

i.e. our loss function becomes:

$$\mathcal{L}_{\mathrm{VAE}} = -\mathbb{E}_{q(z|x)}[\log p(x|z)] + \mathrm{KL}(q(z|x) \,||\, p(z)). \tag{2.14}$$

The first term is the expected negative log-probability of reconstructing the input x given a latent vector z sampled from the posterior distribution. During training, this expectation is estimated by encoding the input x to obtain the parameters of the posterior $q(z|x)$, drawing a sample z from the posterior and conditioning the decoder on this sample to compute $\log p(x|z)$. The second term is the KL divergence from the prior $p(x)$ to the posterior $p(x|z)$. This term effectively acts as a regularizer, pushing the posterior closer to the prior.

Variational autoencoders suffer from serious issues when applied to text, especially when used with a recurrent decoder. In particular, the KL term may lead the encoder to make the posterior distribution almost identical to the prior, resulting in little or no information being encoded in the latent variable. This is because a recurrent decoder is autoregressive, predicting the next word given all the previous words in the sentence. The decoder can therefore achieve a relatively low reconstruction error just by knowing the beginning of the sentence; this allows the posterior distribution to become very 'uncertain', and thereby push the KL

[4]Also known as the *evidence lower bound* or *ELBO*.

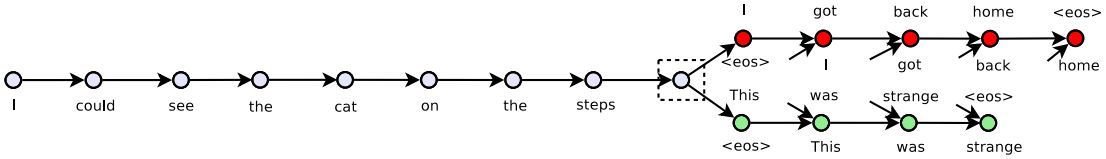


Figure 2.3: The Skip-Thought model (image from Kiros et al., 2015). Two RNN decoders are involved: one for reconstructing the previous sentence (shown in red), one for reconstructing the following sentence (shown in green).

divergence term close to zero. In other words, the encoder tends to sacrifice too much reconstruction accuracy for the sake of minimizing the KL term. Bowman et al. (2016) propose tricks to overcome this issue, but these do not seem to be sufficient for a decent reconstruction performance (Cífka et al., 2018).

Other regularization techniques can be used. For example, Cífka et al. (2018) investigate the effect of adding fixed-variance noise to the representation, normalizing the representation to a unit sphere, and recurrent dropout, and show that these techniques have a similar effect as the KL term in VAEs while allowing for more control over this effect and making the model easier to train. However, the focus of that work is on the generative properties of the models rather than on the representations themselves.

A different way to make the job of an autoencoder harder is to corrupt the input in some way and train the model to reconstruct the original, uncorrupted data. Autoencoders trained using this method are called *denoising autoencoders*. Common ways of corrupting text include randomly dropping or masking words (Hill et al., 2016) or permuting them (Lample et al., 2017).

Skip-Thought

Another model inspired by word embedding algorithms is Skip-Thought (Kiros et al., 2015). Like the Skip-Gram model, which is trained to predict words from the context, Skip-Thought is trained to generate the sentences adjacent to a given sentence. An RNN encoder is used to encode the input sentence and the obtained representation is given as input to two different RNN decoders: one for predicting the previous sentence and one for predicting the following sentence. As with autoencoders, the model is not expected to be able to reconstruct the sentences perfectly, but rather to encode features that help reconstructing them. The architecture of the model is shown in Fig. 2.3.

2.3.3 Supervised methods

By supervised methods, we mean methods that make use of auxiliary supervised tasks. They are supervised in the sense that they need labeled or structured data, not in the sense that we prescribe what the embeddings should look like.

Classification and regression

There are many NLP tasks that could be used for learning sentence representations. One of the first successful attempts is due to Tai et al. (2015), who train a

recursive neural network (a dependency tree LSTM) for sentiment classification, and evaluate the learned representations on semantic relatedness.

Kiela et al. (2017) learn 'visually grounded' sentence representations on the COCO image captioning dataset (Lin et al., 2014), which contains multiple captions for each image. The grounding is ensured by training on two tasks: (a) mapping caption representations to image representations from ResNet (He et al., 2015) and (b) predicting other captions for the same image. The model is trained either on one task only or jointly on both.

The *InferSent* model (Conneau et al., 2017a) is trained on the natural language inference (NLI) task, which is a three-way classification task on pairs of sentences. The setup is similar to that of the SNLI evaluation in the SentEval benchmark (as described Section 1.1.1): the sentence embeddings u, v, their element-wise product $u \odot v$ and their absolute element-wise difference $|u - v|$ are concatenated and fed into a classifier. For details on the NLI task, see Section 1.1.1.

In principle, any sentence-level classification or regression task (e.g. from the SentEval benchmark) could be used to learn sentence embeddings. However, their quality will depend on the nature and difficulty of the task.

Sequence prediction

Another family of models that can be used for learning sentence embeddings are sequence-to-sequence models. In Section 2.3.2, we saw some examples of sequence-to-sequence models applied to learning sentence representations in an unsupervised way. An obvious choice of supervised task for this purpose is machine translation. Previous research in this direction has been discussed in Related work above.

Note that some sequence-to-sequence models are not suitable for this purpose since they do not include a global fixed-size sentence representation. Specifically, this is the case with current attention-based models, which are the most common choice for machine translation. This problem is central to this thesis and we will tackle it in the following chapters.

3. Neural machine translation

In this chapter, we give a background of the predominant models used in neural machine translation (NMT), to the extent needed to explain our proposed modifications to these models.

3.1 RNN encoder-decoder

The basis of most current NMT systems is the RNN *encoder-decoder* architecture (Cho et al., 2014b; Sutskever et al., 2014), which combines an RNN encoder (usually bidirectional; see Section 2.3.1) with an RNN decoder. Like most other neural architectures for modelling sequential data, the RNN decoder operates by predicting the next symbol in a sequence given the previous symbols and context (which, in this case, is the entire input sentence, encoded as the embedding v). The next-word distribution at position t is computed from the hidden state s_t of the RNN at time t (i.e. after having consumed the $t-1$ preceding words) using a softmax layer:

$$\begin{aligned} p(y_t = w | v, y_1, \ldots, y_{t-1}) &= \operatorname{softmax}(W s_t)_w \\ &= \operatorname{softmax}\Big(W g([y_{t-1}, v], s_{t-1})\Big)_w, \end{aligned} \tag{3.1}$$

where W is the *output projection* matrix, the function g is implemented by the RNN cell, and the softmax function is defined as

$$\operatorname{softmax}(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}. \tag{3.2}$$

The decoder is conditioned on the sentence embedding v via its initial state. Optionally, the sentence embedding can also be given as input to the RNN cell at every time step (i.e. the function g may or may not depend on v). The zeroth target word y_0 (needed to compute the first state s_1) is set to the special symbol $\langle \textsc{bos} \rangle$ (beginning of sequence); likewise, the special symbol $y_{T'+1} = \langle \textsc{eos} \rangle$ is used to signal the end of the sentence.

The encoder and the decoder are trained jointly to maximize the probability of the target sequence given the source sequence:

$$\begin{aligned} \max p(y|x) &= \max \prod_{t=1}^{T'+1} p(y_t | x_1, \ldots, x_T, y_1, \ldots, y_{t-1}) \\ &= \max \sum_{t=1}^{T'+1} \log p(y_t | x_1, \ldots, x_T, y_1, \ldots, y_{t-1}). \end{aligned} \tag{3.3}$$

Once the model is trained, we apply it to a new source sequence x and we search for the sequence y for which the probability in Eq. (3.3) is the highest. Since the exact solution cannot be found efficiently, we have to use either *greedy search* (taking the argmax of the softmax distribution at time t and feeding it as input to the RNN cell at time $t+1$) or *beam search* (keeping a list of k best hypotheses and extending them one symbol at a time).

3.1.1 Attention

A component crucial for the success of NMT models is the *attention mechanism* (Bahdanau et al., 2014), which allows the decoder to dynamically shift focus between different input positions and not rely on the encoder to pack the whole sentence into a fixed-dimension vector. For our purposes, this also means that no compact representation of the whole sentence remains available in the network.

The attention mechanism computes a set of weights $(\alpha_{ti})_{t=1,i=1}^{T',T}$ which can be interpreted as a soft alignment between the source and the target sentence. At time t, the weights $(\alpha_{ti})_{i=1}^{T}$ are computed by

$$\alpha_{ti} = \mathrm{softmax}\Big(a(s_{t-1}, h_1), \ldots, a(s_{t-1}, h_T)\Big)_i, \tag{3.4}$$

where a is an *alignment model*, realized as a feed-forward network jointly trained with the rest of the system. The weights $\alpha_{t1}, \ldots, \alpha_{tT}$ are used for combining the encoder states $h_1, \ldots, h_T$ into a context vector c_t, which is used to update the decoder state:

$$c_t = \sum_{i=1}^{T} \alpha_{ti} h_i, \tag{3.5}$$

$$s_t = g([y_{t-1}, c_t], s_{t-1}). \tag{3.6}$$

Other choices for the alignment model a are possible: Luong et al. (2015) propose to use a simple dot product $s_{t-1}^\top h_T$ or, more generally, a learned bilinear form $s_{t-1}^\top W_a h_T$. Luong et al. also introduce *local attention* which predicts a position p_t in the source sentence; the context vector c_t is then computed as a weighted average of the states near position p_t.

3.2 Transformer

The Transformer (Vaswani et al., 2017) is a recent model based entirely on feed-forward layers and attention. It consists of an encoder and a decoder, each formed by stacking $N = 6$ identical layers. Each layer is composed of one or two multi-head attention sub-layers and a feed-forward sub-layer. The input and output of each sub-layer is a sequence of vectors of dimension d_{model}.

The Transformer attention is defined as operating on three sets of vectors: queries, keys and values. Each set of vectors is packed into one matrix, denoted Q, K or V, respectively. For each query in Q, the attention mechanism computes a probability distribution over the keys in K and uses this distribution for weighting the corresponding values from V. The concrete version of attention used in the Transformer is *scaled dot-product attention*, computed as (using the notation from Vaswani et al.)

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3.7}$$

where d_k is the dimensionality of the keys.

This attention mechanism serves two different purposes in the Transformer:

1. Each encoder and decoder layer contains an attention sub-layer at its input. Q, K and V are computed from the output of the previous layer. This application of attention is termed *self-attention*.

   To keep the decoder autoregressive, masking is employed in the decoder self-attention to prevent each position from attending to the following positions.

2. Each decoder layer contains an additional attention sub-layer that operates on the output of the encoder. Here, Q comes from the decoder self-attention sub-layer and K and V are computed from the output layer of the encoder.

Instead of setting Q, K and V directly to the output of the respective sub-layer, the Transformer computes the queries, keys and values using different linear projections with matrices W^Q, W^K and W^V, respectively. In each layer, this is done h times in parallel, each time with different matrices; this is called *multi-head attention*:

$$\begin{aligned} \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V), \\ \text{MultiHead}(Q, K, V) &= [\text{head}_1, \ldots, \text{head}_h]W^O. \end{aligned} \tag{3.8}$$

The matrix W^O projects the concatenated outputs of all attention heads back to d_{model} dimensions.

We omit some important details (positional encoding, residual connections) because the Transformer architecture is marginal to our work. For these details, we refer the reader to Vaswani et al.

4. Proposed models

In this section, we propose new model architectures for sequence-to-sequence learning which are designed to provide continuous vector-space representations of sentences.

Table 4.1 summarizes all the examined configurations of RNN-based models. The architectures differ in (a) which encoder states are considered in subsequent processing (column ①), (b) how they are combined (column ②), and (c) how they are used in the decoder (column ⑥). Column ③ indicates whether a sentence-level representation is available in the model. The first three rows (ATTN, FINAL and FINAL-CTX) correspond roughly to the standard sequence-to-sequence models, Bahdanau et al. (2014), Sutskever et al. (2014) and Cho et al. (2014b), respectively. The last row (ATTN-ATTN) is our main proposed architecture, compound attention, described here in Section 4.1.

In addition to RNN-based models, we modify the Transformer model (Vaswani et al., 2017) in a similar way, see Section 4.3.

4.1 Compound attention

Our compound attention model incorporates attention in both the encoder and the decoder. Its architecture is depicted in Fig. 4.1.

4.1.1 Encoder with inner attention

First, we process the input sequence $x_1, x_2, \ldots, x_T$ using a bi-directional recurrent network with GRU cells:

$$\overrightarrow{h_i} = \overrightarrow{\mathrm{GRU}}(x_t, \overrightarrow{h_{i-1}}), \tag{4.1}$$

$$\overleftarrow{h_i} = \overleftarrow{\mathrm{GRU}}(x_t, \overleftarrow{h_{i+1}}), \tag{4.2}$$

$$h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]. \tag{4.3}$$

We denote by u the combined number of units in the two RNNs, i.e. the dimensionality of h_i.

Next, our goal is to combine the states of the encoder $H = (h_1, h_2, \ldots, h_T)$ into a vector of fixed dimensionality that represents the entire sentence. For this purpose, we employ inner attention (Liu et al., 2016; Li et al., 2016b) to compute several weighted averages of the encoder states (Lin et al., 2017). The main motivation for incorporating these multiple 'views' of the state sequence is that it removes the need for the RNN cell to accumulate the representation of the whole sentence as it processes the input, and therefore it should have more capacity for modelling local dependencies.

Specifically, we fix a number r, the number of *attention heads*, and compute an $r \times T$ matrix A of attention weights α_{ji}, representing the importance of position i in the input for the j^{th} attention head. We then use this matrix to compute r weighted sums of the encoder states, which become the rows of a new matrix M:

$$M = AH. \tag{4.4}$$

	①	②	③	④	⑤	⑥
ATTN*	all	—	✗	✓	✗	—
FINAL†	final	—	✓	✗	✗	init
FINAL-CTX‡	final	—	✓	✗	✗	init+ctx
∗POOL	all	pooling	✓	✗	✗	init
∗POOL-CTX	all	pooling	✓	✗	✗	init+ctx
ATTN-CTX	all	inner att.	✓	✗	✗	init+ctx
ATTN-ATTN♥	all	inner att.	✓	✗	✓	input for att.

*Bahdanau et al. †Sutskever et al. ‡Cho et al. ♥compound attention

Table 4.1: Different RNN-based sequence-to-sequence architectures and their properties. Legend:

① encoder states used
② states combined using...
  – pooling = mean (AVGPOOL) or maximum (MAXPOOL)
③ sentence embedding available
④ decoder attends to encoder states
⑤ decoder attends to parts of sentence embedding
⑥ sentence embedding used in...
  – init = initial decoder state
  – ctx = context vector, i.e. input for the decoder cell
  – input for att. = input for decoder attention

A vector representation of the source sentence (the 'sentence embedding') can be obtained by flattening the matrix M. In our experiments, we project the encoder states $h_1, h_2, \ldots, h_T$ down to a given dimensionality before applying Eq. (4.4), so that we can control the size of the representation.

Following Lin et al. (2017), we compute the attention matrix A by feeding the encoder states to a two-layer feed-forward network:

$$A = \text{softmax}(U \tanh(WH^\top)), \tag{4.5}$$

where W and U are weight matrices of dimensions $d \times u$ and $r \times d$, respectively (d is the number of hidden units); the softmax function is applied along the second dimension, i.e. across the encoder states.

4.1.2 Attentive decoder

In vanilla sequence-to-sequence models with a fixed-size sentence representation (FINAL and FINAL-CTX), the decoder is conditioned on this representation via the initial RNN state or via the input of the RNN cell. We propose to instead leverage the structured sentence embedding by applying attention to its components. This is no different from the classical attention mechanism as described in Section 3.1.1, except that it acts on this fixed-size representation instead of the sequence of encoder states.

In the t^{th} decoding step, the attention mechanism computes a distribution $\{\beta_{tj}\}_{j=1}^{r}$ over the r components of the structured representation. This is then used to weight these components to obtain the context vector c_t, which in turn is used to update the decoder state. Again, we can write this in matrix form as

$$C = BM, \tag{4.6}$$

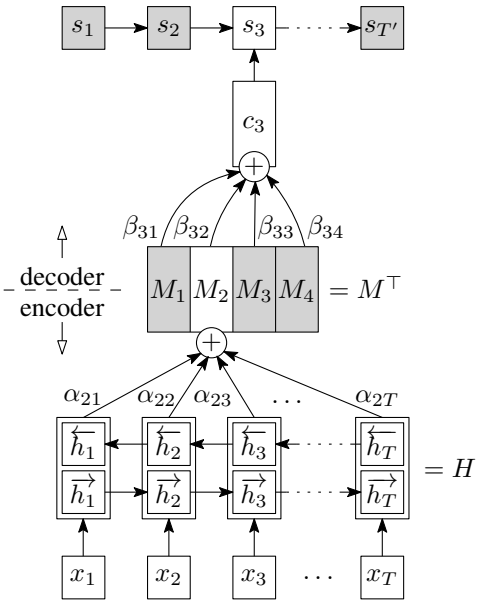


Figure 4.1: An illustration of compound attention with 4 attention heads. The figure shows the computations that result in the decoder state s_3 (in addition, each state s_t depends on the previous target token y_{t-1}, which is not reflected here to keep the figure uncluttered). Note that the matrix M is the same for all positions in the output sentence and it can thus serve as the source sentence representation.

where $B = (\beta_{tj})_{t=1,j=1}^{T',r}$ is the attention matrix and $C = (c_i, c_2, \ldots, c_{T'})$ are the context vectors.

Note that by combining Eqs. (4.4) and (4.6), we get

$$C = (BA)H. \tag{4.7}$$

Hence, the composition of the encoder and decoder attentions (the 'compound attention') defines an implicit alignment between the source and the target sequence. From this viewpoint, our model can be regarded as a restriction of the widely used ATTN model.

The decoder uses a conditional GRU cell (cGRU_{att}; Sennrich et al., 2017), which consists of two consecutively applied GRU blocks. The first block processes the previous target token y_{t-1}, while the second block receives the context vector c_t and predicts the next target token y_t.

4.2 Constant context

Compared to the FINAL model, the compound attention architecture described in the previous section undoubtedly benefits from the fact that the decoder is presented with information from the encoder (i.e. the context vectors c_t) in every decoding step. To investigate this effect, we include baseline models where we replace all context vectors c_t with the entire sentence embedding (indicated by the suffix '-CTX' in Table 4.1). Specifically, we provide either the flattened matrix M (for models with inner attention; ATTN-CTX), the final state of the encoder

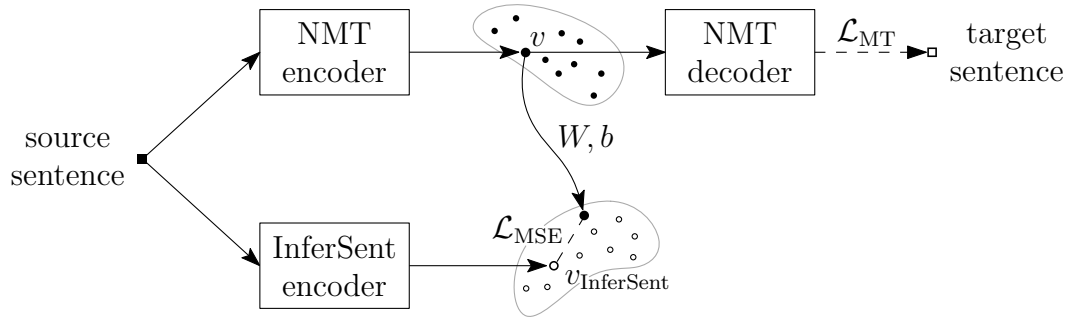


Figure 4.2: Multi-task training where the auxiliary task is to predict InferSent embeddings using linear regression. The InferSent encoder is pre-trained and is not updated during the multi-task training.

(FINAL-CTX), or the result of mean- or max-pooling (∗POOL-CTX) as a constant input to the decoder cell.

4.3 Transformer with inner attention

Like the ATTN model, the Transformer (TRF) does not provide a fixed-dimension sentence representation. In order to introduce this representation into the model, we modify it by adding (multi-head) inner attention after the last encoder layer. The matrix M above then serves as the query matrix Q for the decoder attention.

Our variation on the Transformer model corresponds to the ATTN-ATTN row in Table 4.1 and we refer to it as TRF-ATTN-ATTN.

4.4 Multi-task models

The models described so far are trained for translation only. To encourage the models to perform better on our evaluation tasks (see Section 1.1), we could directly train them on these tasks in a multi-task setting (in addition to translation). If we had labels for these tasks in our training parallel corpus, this could be done in a straightforward way by adding a term for each task to our loss function:

$$\mathcal{L} = \mathcal{L}_{\text{MT}} + \alpha_{\text{MR}}\mathcal{L}_{\text{MR}} + \alpha_{\text{CR}}\mathcal{L}_{\text{CR}} + \alpha_{\text{SUBJ}}\mathcal{L}_{\text{SUBJ}} + \dots \tag{4.8}$$

Note that this would work only for tasks that take single sentences, and not sentence pairs (like the NLI task), as input. However, we don't possess the labels and it might not even be possible to assign them since our MT data is out-of-domain for most of the tasks (e.g. assessing review polarity or question type makes little sense for most sentences in a general machine translation corpus).

What we have is the training data for each task. We could therefore alternate the different datasets and objectives (MT and other tasks) during training. This still comes with technical difficulties, especially for tasks like NLI where each training example consists of two sentences. For this reason, we do not pursue this method here.

Another option is to 'ask for help' from a sentence embedding model which is known to achieve good results in the evaluations. One such model, trained on the

NLI task, is InferSent (Conneau et al., 2017a). We use the pre-trained InferSent model[1] to encode the source side of our parallel corpus. Then, when training our MT model, we project our MT embeddings to the InferSent embedding space and compute a MSE (mean squared error) objective, which we add to our loss function:

$$\mathcal{L} = \mathcal{L}_{\text{MT}} + \alpha\mathcal{L}_{\text{MSE}} = \mathcal{L}_{\text{MT}} + \alpha\|Wv + b - v_{\text{InferSent}}\|^2. \tag{4.9}$$

Here, α is a hyperparameter, W is a learned weight matrix and b is a learned bias vector. The setup illustrated in Fig. 4.2.

[1] `https://github.com/facebookresearch/InferSent`

5. Experiments

5.1 Training

We trained English-to-German and English-to-Czech NMT models using Neural Monkey[1] (Helcl and Libovický, 2017a). In the following, we use the code of the target language, i.e. *de* or *cs*, to distinguish these models. We chose English as the source language mainly because most current sentence embedding models, as well as datasets and tools for evaluating them, are only available for English.

The *de* models were trained on the Multi30K multilingual image caption dataset (Elliott et al., 2016), extended by Helcl and Libovický (2017b), who acquired additional parallel data using back-translation (Sennrich et al., 2016a) and perplexity-based selection (Yasuda et al., 2008). This extended dataset contains 410k sentence pairs, with an average sentence length of 12 ± 4 tokens in English. We train each model for 20 epochs with a batch size of 32. We truecased the training data as well as all data we evaluate on. For German, we employed Neural Monkey's reversible pre-processing scheme, which expands contractions and performs morphological segmentation of determiners. We used a vocabulary of at most 30k tokens for each language (no subword units).

The *cs* models were trained on CzEng 1.7 (Bojar et al., 2016).[2] We used byte-pair encoding (BPE) with a vocabulary of 30k sub-word units, shared for both languages (the original vocabulary size for English is 1.9M). For English, the average sentence length is 15 ± 19 BPE tokens. We performed 1 training epoch with a batch size of 128 on the entire training section (57M sentence pairs).

All models were optimized using Adam (Kingma and Ba, 2014) with a learning rate of 10^{-4}. We employed L_2 regularization with a weight of 10^{-8} and gradient norm clipping with a threshold of 1.0. The maximum input length was set to 30 tokens and the maximum output length to 35 tokens; longer sentences were truncated.

The datasets for both *de* and *cs* models come with their respective development and test sets of sentence pairs, which we use for the evaluation of translation quality. (We use 1k randomly selected sentence pairs from CzEng 1.7 dtest as a development set. For evaluation, we use the entire etest.)

5.1.1 InferSent multi-task training

For a subset of the models, we implemented the multi-task training method described in Section 4.4 to 'ground' the sentence representations in the InferSent embeddings. For these models, we doubled the gradient clipping threshold to 2.0. The rest of the hyperparameters are kept unchanged.

Adding another term to the loss function could alter the learning dynamics or otherwise regularize the model. To check whether the performance differences caused by using this method are entirely due to such effects or whether training with InferSent targets is genuinely helpful, we include variants of these models where we randomly shuffle the target embeddings in the entire training set. In

[1] `https://github.com/ufal/neuralmonkey`

[2] `http://ufal.mff.cuni.cz/czeng/czeng17`

doing so, we weaken the model by making it learn a random, useless function instead of letting it fully focus on translation. We hypothesize that this (ridiculous) form of regularization might still make the representations perform better at some other tasks.

5.2 Representation evaluation

We evaluate our learned representations with classification and similarity tasks from SentEval (Section 1.1.1), by examining clusters of paraphrase representations, and using the supervised domain alignment technique from Section 1.2.2. The rest of this section details the last two techniques.

5.2.1 Paraphrases

We evaluate the representation of paraphrases using the method described in Section 1.2.4. We use two data sources for this purpose: COCO and HyTER Networks.

COCO (Common Objects in Context; Lin et al., 2014) is an object recognition and image captioning dataset, containing 5 captions for each image. We extracted the captions from its validation set to form a set of $5 \times 5\text{k} = 25\text{k}$ sentences grouped by the source image. The average sentence length is 11 tokens and the vocabulary size is 9k token types.

It should be noted that while we treat captions belonging to one image as being synonymous, this is often far from the truth. As can be seen from Fig. 5.1, the images are sometimes fairly complex and some of the captions may pay more attention to certain details, while other captions may be too vague or even incorrect. There is also the issue that we mentioned in Section 1.2.4, namely that captions of different images may be synonymous (e.g. when the images depict the same thing), but we treat them as completely unrelated.

HyTER Networks (Dreyer and Marcu, 2014) are large finite-state networks representing a subset of all possible English translations of 102 Arabic and 102 Chinese sentences. The networks were built by a number of human annotators based on reference sentences in Arabic, Chinese and English. Each network contains up to hundreds of thousands of possible translations of a given source sentence. We randomly generated 500 translations for each source sentence, obtaining a corpus of 102k sentences grouped into 204 clusters of 500. The average length of the sentences is 28 tokens and the vocabulary size is 11k token types. Since the sentences in each cluster are accurate translations of one source sentence, they can be safely considered semantically equivalent. Moreover, sentences in different clusters are guaranteed to have different meanings.

For every model, we encode each dataset to obtain a set of sentence embeddings with cluster labels. We then compute the metrics described in Section 1.2.4: cluster classification accuracy, paraphrase retrieval accuracy, and Davies-Bouldin (DB) index.

To compute cluster classification accuracy, we remove 1 point (in the case of COCO) or half of the points (in the case of HyTER) from each cluster, and fit

[3] `http://cocodataset.org/#explore?id=43411`
`http://cocodataset.org/#explore?id=366199`

- A person laughing on the telephone near a lot of treats
- A woman talking on her cell phone in an old building near a table displaying baked goods.
- A happy girl stands in front of a table covered with deserts.
- A woman standing in front of a table of baked goods.
- A bunch of different food siting out at a store.

- A cat hides underneath the cover of blankets.
- Small cat under the blankets on a bed.
- A very cute cat hiding under a blanket.
- That looks like it may be hiding under something.
- I am not sure what this image is.

Figure 5.1: Examples of images[3] with captions from the COCO dataset. The images were picked to showcase captions that are incomplete or unspecific.

an LDA classifier on the rest. We then compute the accuracy of the classifier on the removed points.

For paraphrase retrieval, we follow the approach described in Section 1.2.4 with $k = 1$, i.e. we consider only the single nearest neighbor. We try L_2 and cosine distance for finding the nearest neighbor and report accuracy (precision) for both options.

We compute the DB index according to Eq. (1.6). As already mentioned, a lower value indicates better cluster separation. Since all of our other metrics behave conversely (higher values are better), we report the inverse of the DB index (DB^{-1}) to maintain this property.

5.2.2 Domain alignment

We also evaluate the sentence embeddings using the supervised domain adaptation technique outlined in Section 1.2.2. Due to the lack of quality parallel data for this experiment, we use a rather simple dataset consisting of news article headlines and summaries from a sentence compression corpus (Filippova and Altun, 2013). The full dataset[4] contains data from about 200k news articles. For each article, the headline and the first sentence were acquired. Next, an extractive summary (a compression) of the first sentence was generated by identifying the words in the sentence that match content words in the headline, then pruning the parse tree of the sentence while preserving these content words. As a result, the summaries are very similar to the headlines, often differing only in the main verb and punctuation. See Fig. 5.2 for examples.

We use only the compression-headline pairs from this dataset, namely 18k pairs for training plus 2k pairs as a held-out set, and 20k pairs for testing. We

[4] `https://github.com/google-research-datasets/sentence-compression`

headline	Bank of America opens outreach center in Henderson, US
compression	Bank of America has opened an outreach center in Henderson, US.
full sentence	Bank of America has opened an outreach center in Henderson, US, to serve its mortgage customers.
headline	Al Qaeda essentially defeated in Iraq:
compression	Al Qaeda is essentially defeated in Iraq.
full sentence	CIA chief Michael Hayden, in an interview published Friday, said Al Qaeda is essentially defeated in Iraq, Saudi Arabia and on the defensive elsewhere, including the Afghanistan-Pakistan border.
headline	Apple to open first Hong Kong store this quarter
compression	Apple Inc. will open its first Hong Kong store this quarter.
full sentence	Apple Inc. said it will open its first Hong Kong retail store this quarter and is targeting another Shanghai store opening by the end of the year.

Figure 5.2: A sample from the sentence compression dataset. In our experiments, we only use the compressions and the headlines.

chose not to use the full sentences because these tend to be very long, especially when they include quotations.

Our domain alignment experiment consists in trying to learn an affine transformation that maps embeddings of compressions to embeddings of headlines (see Fig. 5.3a). We fit a linear regression on the embeddings using Adam (Kingma and Ba, 2014), stopping when the loss on the held-out set no longer improves. At test time, we transform the embedding of each sentence compression using the learned mapping and retrieve the closest embedding (using L_2 or cosine distance) from the mixed pool of 20k compressions and 20k headlines. Because the retrieved embedding may happen to be the same as the original (untransformed) embedding (see Fig. 5.3b), we optionally exclude this original embedding. In this way, we obtain two sets of accuracies, which we refer to as 'incl.' and 'excl.'.

Note that the excl. and incl. variants are analogous to the original (Mikolov et al., 2013d) and 'honest' (Rogers et al., 2017) version of the word analogy evaluation, respectively (see Section 1.2.1), and we are therefore more interested in the latter.

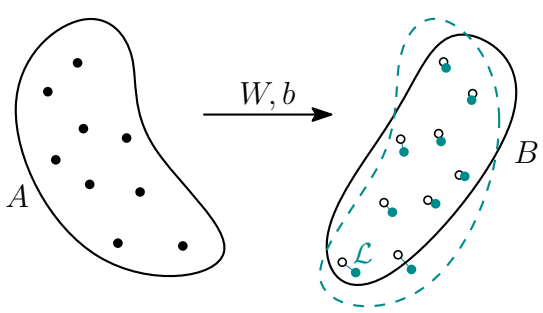
5.3 Results

5.3.1 Translation quality

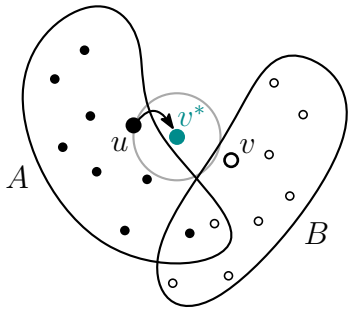
We estimate translation quality of the various models using single-reference case-sensitive BLEU (Papineni et al., 2002) on translations decoded using greedy search.

Tables 5.1 and 5.2 provide the results on the two datasets. The *cs* dataset is much larger and the training takes much longer. We were thus able to experiment with only a small subset of the possible model configurations.

The columns 'Size' and 'Heads' specify the total size of sentence representation and the number of heads of encoder inner attention.



(a) Learning an affine mapping from domain A to domain B. W (weight matrix) and b (bias vector) are the parameters of this mapping. (A reproduction of Fig. 1.1 for convenience.)



(b) An example where the prediction v^* lies closer to the original embedding u than to the target v.

Figure 5.3: Supervised domain alignment. In our experiments, domains A and B correspond to sentence compressions and headlines, respectively.

Model			BLEU	
Name	Size	Heads	dev	test
de-ATTN	—	—	37.6	36.2
de-TRF	—	—	38.2	36.1
de-ATTN-ATTN	2400	12	36.2	34.8
de-ATTN-ATTN	1200	12	35.6	34.3
de-ATTN-ATTN	600	8	35.4	33.7
de-ATTN-ATTN	600	12	35.3	33.4
de-ATTN-ATTN	1200	6	35.0	33.2
de-ATTN-ATTN	600	6	35.1	33.2
de-TRF-ATTN-ATTN	600	3	32.3	30.1
de-ATTN-ATTN	600	3	31.4	29.4
de-ATTN-CTX	1200	12	30.6	29.2
de-ATTN-CTX	600	12	29.8	29.1
de-ATTN-CTX	600	8	29.8	28.9
de-ATTN-CTX	600	6	29.5	28.8
de-TRF-ATTN-ATTN	2400	12	30.6	28.5
de-MAXPOOL-CTX	600	—	27.8	28.1
de-FINAL-CTX	600	—	28.1	26.9
de-ATTN-CTX	600	3	27.8	26.9
de-AVGPOOL-CTX	600	—	27.1	26.5
de-ATTN-ATTN	600	1	27.2	26.0
de-TRF-ATTN-ATTN	600	6	26.5	25.8
de-TRF-ATTN-ATTN	1200	12	26.6	25.3
de-FINAL	600	—	23.9	23.8

Table 5.1: Translation quality of *de* models.

Model			BLEU		Human judgment (%)	
Name	Size	Heads	dev	test	> others	≥ others
cs-ATTN	—	—	22.8	22.2	50.9	93.8
cs-ATTN-ATTN	1000	8	19.1	18.4	42.5	88.6
cs-ATTN-ATTN	4000	4	18.4	17.9	—	—
cs-ATTN-ATTN	1000	4	17.5	17.1	—	—
cs-ATTN-CTX	1000	4	16.6	16.1	31.7	77.9
cs-FINAL-CTX	1000	—	16.1	15.5	—	—
cs-ATTN-ATTN	1000	1	15.3	14.8	27.3	71.7
cs-FINAL	1000	—	11.2	10.8	—	—
cs-AVGPOOL	1000	—	11.1	10.6	—	—
cs-MAXPOOL	1000	—	5.4	5.4	2.7	13.1

Table 5.2: Translation quality of *cs* models. The human judgment results show how often a model was judged better (>) or at least as good (≥) as all other models on a particular sentence pair.

In both cases, the best performing model is ATTN (Bahdanau et al.), followed by Transformer (*de* only) and our ATTN-ATTN (compound attention). It is worth noting that ATTN-ATTN presents a significant improvement over ATTN-CTX (+3.7 BLEU on average). The non-attentive FINAL (Sutskever et al.) is the worst, except cs-MAXPOOL.

For 5 selected *cs* models, we also performed manual evaluation on 200 sentence pairs using WMT-style pairwise ranking (as used in WMT 2011). The results, shown in Table 5.2, confirm the automatic evaluation results.

We also checked the relation between BLEU and the number of heads and representation size. While there are many exceptions, the general tendency is that the larger the representation or the more heads, the higher the BLEU score. The Pearson correlation between BLEU and the number of heads is 0.87 for *cs* and 0.31 for *de*.

5.3.2 SentEval

Due to the large number of SentEval tasks, we present the results abridged in two different ways: (1) by reducing the number of tasks reported and (2) by only showing our best performing setups and comparing them to other approaches. The full results can be found in Appendix A.

For these and all following tasks, we computed the out-of-vocabulary (OOV) rate and the perplexity of a 4-gram language model (LM) trained on the English side of the respective parallel corpus (CzEng 1.7 or extended Multi30K) and evaluated on all available data for the given task. The OOV rate and perplexity are shown in the respective results tables to aid in explaining the observed differences (in particular, between *cs* and *de* models).

Table 5.3 shows all of our models evaluated on a subset of the tasks. As the baseline for the classification tasks, we assign the most frequent class to all test examples. (For MR, CR, SUBJ, and MPQA, where there is no distinct test set, the most frequent class is established on the whole collection. For the other tasks,

Model			Class. accuracy (%)			Avg. sim.
Name	Size	Heads	SNLI	SICK-E	Avg.$^+$	
Most frequent baseline			56.7	34.3	48.19	—
InferSent	4096	—	(83.7)	86.4	81.7	.70
GloVe-BOW	300	—	66.0	78.2	75.8	.59
cs-FINAL-CTX	1000	—	70.2	82.1	74.4	.60
cs-ATTN-ATTN	1000	1	69.3	80.8	73.4	.54
cs-FINAL	1000	—	69.2	81.1	73.2	.60
cs-MAXPOOL	1000	—	68.5	81.7	73.0	.60
cs-AVGPOOL	1000	—	67.8	79.7	72.4	.50
cs-ATTN-CTX	1000	4	66.0	79.5	72.2	.45
cs-ATTN-ATTN	4000	4	65.2	78.0	71.2	.39
cs-ATTN-ATTN	1000	4	64.6	78.0	70.8	.39
cs-ATTN-ATTN	1000	8	63.2	76.6	70.0	.36
de-MAXPOOL-CTX	600	—	68.0	78.8	67.1	.50
de-ATTN-CTX	1200	12	65.0	77.4	66.7	.52
de-ATTN-CTX	600	8	64.0	75.7	65.8	.51
de-AVGPOOL-CTX	600	—	65.2	77.5	65.6	.48
de-ATTN-CTX	600	12	61.9	76.0	65.5	.50
de-FINAL	600	—	64.7	77.0	65.3	.47
de-ATTN-CTX	600	3	63.3	76.0	65.3	.50
de-ATTN-ATTN	600	1	63.8	76.9	64.8	.50
de-ATTN-ATTN	600	3	61.5	74.7	64.5	.47
de-FINAL-CTX	600	—	62.6	76.2	64.5	.48
de-ATTN-ATTN	1200	6	59.6	72.3	64.3	.41
de-TRF-ATTN-ATTN	600	3	61.4	72.5	63.9	.49
de-ATTN-ATTN	1200	12	58.2	72.5	63.4	.43
de-ATTN-ATTN	2400	12	59.8	73.9	63.2	.41
de-TRF-ATTN-ATTN	2400	12	59.0	71.2	63.0	.46
de-ATTN-ATTN	600	6	57.5	70.9	62.6	.40
de-ATTN-ATTN	600	8	55.6	68.6	62.1	.39
de-TRF-ATTN-ATTN	600	6	59.5	71.0	61.9	.45
de-ATTN-ATTN	600	12	55.2	70.5	61.5	.40
de-TRF-ATTN-ATTN	1200	12	58.2	68.8	61.1	.46
de-ATTN-CTX	600	6	62.9	68.7	61.0	.43
LM perplexity (*cs*)			190.6	299.4	1150.2	1224.2
% OOV (*cs*)			0.3	0.2	2.3	2.6
LM perplexity (*de*)			38.8	65.0	3578.2	2010.6
% OOV (*de*)			1.5	1.7	17.8	16.2

Table 5.3: Abridged results of the SentEval benchmark. Full results in Appendix A. 'Avg.$^+$' is the average of all 10 classification tasks (see Table A.1), 'Avg. sim.' averages all 7 similarity tasks (see Table A.2). Note that InferSent was trained on the NLI task using a superset of the SNLI dataset.

Model			Class. accuracy (%)						
Name	Size	H.	MR	CR	SUBJ	MPQA	SST2	SST5	TREC
Most frequent baseline			50.0	63.8	50.0	68.8	49.9	23.1	18.8
InferSent	4096	—	**81.5**	**86.7**	92.7	**90.6**	**85.0**	**45.8**	88.2
Hill et al. en→fr†	2400	—	64.7	70.1	84.9	81.5	—	—	82.8
SkipThought-LN†	—	—	79.4	83.1	**93.7**	89.3	82.9	—	88.4
GloVe-BOW	300	—	77.0	78.2	91.1	87.9	81.0	44.4	82.0
cs-FINAL-CTX	1000	—	68.7	77.4	88.5	85.5	73.0	38.2	88.6
cs-ATTN-ATTN	1000	1	68.2	76.0	86.9	84.9	72.0	35.7	**89.0**

Model			Class. accuracy (%)				Correl. (P./S.)	
Name	Size	H.	MRPC	SICK-E	SNLI	Avg.	SICK-R	STSB
Most frequent baseline			66.5	56.7	34.3	48.19	—	—
InferSent	4096	—	76.6	**86.4**	(83.7)	**81.7**	**.88/.83**	**.76/.75**
Hill et al. en→fr†	2400	—	**96.1**	—	—	—	—	—
SkipThought-LN†	—	—	—	79.5	—	—	.85/ —	—
GloVe-BOW	300	—	72.3	78.2	66.0	75.8	.80/.72	.64/.62
cs-FINAL-CTX	1000	—	71.8	82.1	70.2	74.4	.82/.76	.74/.74
cs-ATTN-ATTN	1000	1	70.7	80.8	69.3	73.4	.81/.76	.73/.73

Model			Correlation (Pearson/Spearman)					
Model	Size	H.	STS12	STS13	STS14	STS15	STS16	Avg.
InferSent	4096	—	**.59/.60**	**.59/.59**	**.70/.67**	**.71/.72**	**.71/.73**	**.70**
SkipThought-LN†	—	—	—	—	.44/.45	—	—	—
GloVe-BOW	300	—	.52/.53	.50/.51	.55/.56	.56/.59	.51/.58	.59
cs-FINAL-CTX	1000	—	.51/.53	.44/.44	.52/.50	.62/.61	.57/.58	.60
cs-ATTN-ATTN	1000	1	.46/.49	.32/.33	.45/.44	.53/.52	.47/.48	.54

Table 5.4: A comparison of state-of-the-art SentEval results with our best models and the Glove-BOW baseline. 'H.' is short for 'Heads'. Reprinted results are marked with †, others are our measurements.

the class is learned from the training set.)

The *de* models are generally worse, most likely due to the higher OOV rate and the overall simplicity of the training sentences. On *cs*, we see a clear pattern that more heads hurt the performance. The *de* set has more variations to consider but the results are less conclusive.

For the similarity results, it is worth noting that cs-ATTN-ATTN performs very well with 1 attention head but fails miserably with more heads. Otherwise, the relation to the number of heads is less clear.

Table 5.4 compares our strongest models with the state of the art on all tasks. Besides InferSent, we include SkipThought as evaluated by Conneau et al. (2017a), and the NMT-based embeddings by Hill et al. (2016) trained on the English-French WMT15 dataset (this is the best result reported by Hill et al. for NMT). As a baseline, we include bag-of-words embeddings obtained by averaging GloVe word vectors (GloVe-BOW).

We see that the supervised InferSent clearly outperforms all other models in all tasks except for MRPC and TREC. Results by Hill et al. are always lower than our best setups, except MRPC and TREC again. On classification tasks, our models are outperformed even by GloVe-BOW, except for the NLI tasks (SICK-E and SNLI) where cs-FINAL-CTX is better.

5.3.3 Paraphrase scores

Table 5.5 provides our measurements based on sentence paraphrases. We found that for paraphrase retrieval, cosine similarity worked better than L_2 distance in most cases. For this reason, we only give the cosine-based results here. Again, the full results can be found in Appendix A.

This evaluation seems less stable and discerning than the previous two, but we can again confirm the victory of InferSent followed by our non-attentive *cs* models. *cs* and *de* models are no longer clearly separated. This might be because of the aforementioned issues with using the COCO dataset as a paraphrase corpus (Section 5.2.1).

The HyTER tasks, especially the paraphrase retrieval (NN) task, are clearly easy to solve. This is probably due to the relatively low number of clusters (204). Since most of the sentences in each cluster probably share at least some words which are unique to that cluster (e.g. named entities), it might be possible to separate the clusters simply based on vocabulary. This would explain the almost-perfect accuracy achieved by most models including GloVe-BOW.

5.3.4 Domain alignment

Domain alignment results on the compression-headline dataset are shown in Table 5.6. First of all, in the incl. column, L_2 distance tends to give higher scores than cosine similarity. We can probably relate this to the fact that the loss function for learning the alignment is also L_2-based. In the following, we will therefore only consider the L_2-based values in each column.

The highest accuracy in each row is always in the excl. column – this is by definition because the task is made easier by eliminating one of the wrong answers. However, the differences between incl. and excl. are surprisingly large,

Model			HyTER			COCO		
Name	Size	Heads	Cls.	NN	DB^{-1}	Cls.	NN	DB^{-1}
InferSent	4096	—	99.99	100.00	0.579	31.58	26.21	0.367
GloVe-BOW	300	—	99.94	100.00	0.654	34.28	19.72	0.352
cs-FINAL-CTX	1000	—	99.92	100.00	0.406	23.20	16.07	0.346
cs-MAXPOOL	1000	—	99.86	100.00	0.447	21.76	16.34	0.348
de-ATTN-CTX	600	8	98.11	99.90	0.348	21.64	17.32	0.349
cs-FINAL	1000	—	99.91	100.00	0.439	22.40	14.63	0.340
de-ATTN-CTX	1200	12	98.88	99.91	0.347	20.06	16.68	0.348
de-MAXPOOL-CTX	600	—	98.42	99.90	0.343	21.54	15.62	0.341
de-ATTN-CTX	600	3	97.81	99.87	0.328	19.74	16.43	0.343
de-ATTN-CTX	600	12	97.79	99.89	0.360	20.22	16.10	0.344
de-ATTN-CTX	600	6	98.11	99.86	0.358	20.44	15.57	0.342
de-ATTN-ATTN	600	1	97.70	99.73	0.352	19.74	16.26	0.340
de-AVGPOOL-CTX	600	—	97.72	99.60	0.312	20.04	14.27	0.337
cs-ATTN-ATTN	1000	1	99.88	99.91	0.347	21.54	11.50	0.331
de-ATTN-ATTN	600	3	97.42	99.75	0.314	17.36	14.35	0.333
de-FINAL	600	—	97.01	99.30	0.305	19.88	12.40	0.328
de-FINAL-CTX	600	—	96.65	99.70	0.323	17.22	12.84	0.333
de-TRF-ATTN-ATTN	600	3	95.79	99.64	0.315	15.76	14.04	0.340
cs-AVGPOOL	1000	—	99.80	99.99	0.387	17.90	8.61	0.311
de-ATTN-ATTN	1200	12	97.15	99.65	0.283	12.18	11.97	0.330
de-ATTN-ATTN	1200	6	98.05	99.80	0.289	11.90	10.69	0.327
de-ATTN-ATTN	2400	12	98.69	99.77	0.287	10.26	10.94	0.326
cs-ATTN-CTX	1000	4	99.75	99.74	0.287	14.60	7.54	0.318
de-ATTN-ATTN	600	6	96.03	99.71	0.287	12.22	10.59	0.323
de-TRF-ATTN-ATTN	2400	12	95.82	99.03	0.307	5.66	14.53	0.339
de-ATTN-ATTN	600	8	95.32	99.73	0.275	10.22	10.58	0.325
de-ATTN-ATTN	600	12	95.16	99.64	0.278	9.62	10.47	0.323
de-TRF-ATTN-ATTN	600	6	90.24	98.44	0.313	9.06	13.64	0.332
de-TRF-ATTN-ATTN	1200	12	90.71	98.22	0.301	7.06	13.70	0.333
cs-ATTN-ATTN	4000	4	99.54	98.98	0.252	11.52	5.51	0.303
cs-ATTN-ATTN	1000	4	99.26	98.93	0.253	10.84	5.20	0.299
cs-ATTN-ATTN	1000	8	99.41	98.09	0.243	10.24	4.64	0.287
LM perplexity / % OOV (*cs*)			668.5 / 1.2			238.5 / 0.1		
LM perplexity / % OOV (*de*)			3354.8 / 19.3			86.3 / 1.9		

Table 5.5: Paraphrase evaluation on HyTER and COCO – 'Cls.' is the cluster classification accuracy, 'NN' is the nearest-neighbor paraphrase retrieval accuracy and DB^{-1} is the inverse Davies-Bouldin index. 'NN' columns are based on cosine similarity; L_2-based accuracies are included in the full results in Table A.3. Ordered according to the average performance (see full results).

Model			% Accuracy (L_2/cos)	
Name	Size	Heads	incl.	excl.
cs-AVGPOOL	1000	—	69.1/66.7	85.7/87.8
cs-ATTN-ATTN	1000	8	67.5/68.4	71.9/73.7
cs-ATTN-ATTN	1000	4	60.0/63.0	74.2/78.0
cs-ATTN-ATTN	1000	1	60.0/57.3	90.7/92.6
de-FINAL	600	—	56.1/64.7	57.4/66.2
cs-ATTN-ATTN	4000	4	53.8/60.2	72.2/79.4
de-MAXPOOL-CTX	600	—	51.5/27.3	61.3/69.9
cs-ATTN-CTX	1000	4	51.5/56.0	82.5/86.5
cs-FINAL	1000	—	51.2/51.0	94.4/95.4
de-FINAL-CTX	600	—	45.1/37.5	53.2/70.3
cs-MAXPOOL	1000	—	43.5/22.5	91.9/94.3
de-AVGPOOL-CTX	600	—	38.9/29.8	58.6/68.2
InferSent	4096	—	36.3/33.7	97.4/97.2
de-ATTN-ATTN	2400	12	36.3/32.1	42.0/61.9
de-ATTN-ATTN	1200	12	34.2/27.9	47.6/66.7
de-ATTN-ATTN	600	12	33.4/25.6	40.2/61.2
de-ATTN-CTX	1200	12	32.3/29.7	62.8/74.7
de-ATTN-ATTN	600	6	32.1/28.1	40.7/61.5
de-ATTN-CTX	600	6	32.0/29.6	60.0/72.5
de-ATTN-ATTN	600	3	32.0/30.4	55.3/70.0
cs-FINAL-CTX	1000	—	31.9/34.6	95.4/96.2
de-ATTN-ATTN	600	1	31.5/30.0	64.4/71.9
de-ATTN-ATTN	1200	6	31.0/29.6	41.0/62.3
de-ATTN-CTX	600	12	30.4/26.4	60.2/72.6
de-ATTN-ATTN	600	8	30.3/27.6	37.8/63.0
de-ATTN-CTX	600	3	30.1/30.6	61.6/72.7
de-ATTN-CTX	600	8	24.4/24.7	62.7/74.3
de-TRF-ATTN-ATTN	600	3	23.7/32.6	49.0/55.7
de-TRF-ATTN-ATTN	2400	12	21.6/26.0	37.8/40.7
de-TRF-ATTN-ATTN	600	6	20.1/25.8	40.6/44.8
de-TRF-ATTN-ATTN	1200	12	19.8/27.3	38.1/42.7
GloVe-BOW	300	—	10.5/15.7	87.5/88.7
LM perplexity / % OOV (*cs*)			1501.3 / 1.8	
LM perplexity / % OOV (*de*)			6439.4 / 28.5	

Table 5.6: Domain alignment results. Ordered by incl. (L_2).

namely 24.8 ± 18.0 on average. This means that in 24.8 % of the cases, the predicted embedding was closest to the source embedding – compare this to the 38.2 % cases (= average of the incl. column) where the prediction was closest to the target. This suggests that the source and target embeddings are already close to each other (which is not that surprising given the fact that the headlines and the compressions are mutually similar).

For some of the models, the incl. results are reasonably high (50–60 %), which means that it was possible to align the domains fairly accurately. Interestingly, most *cs* models outperformed InferSent significantly, unlike in all other evaluations. We will also see later that the correlation of incl. with other metrics is generally negative. We explain this by the fact that the mapping which we try to learn preserves the meaning of the sentence and transforms its surface structure in a rather deterministic (and subtle) way. Sentence embeddings that tend to capture more of this surface structure are then more suitable for learning the mapping, scoring higher in this evaluation. Models with strong focus on meaning (or models like GloVe-BOW which have limited access to the surface structure) will map both sentences from each pair close to each other (which will lead to a high excl. score), but the mapping from one domain to the other might be less predictable (and the incl. score therefore lower).

Another clear pattern is that in models of the same type and representation size, the incl. score is often constant for different numbers of attention heads, but the excl. score decreases as the number of heads increases. E.g. for de-ATTN-ATTN with 600 dimensions, the excl. scores are 64.4, 55.3, 40.7, 37.8 and 40.2 for 1, 3, 6, 8 and 12 heads, respectively, while the incl. score remains between 30.3 and 33.4. This means that increasing the number of attention heads doesn't necessarily help learn a better mapping, but it does push the sentences in each pair far apart in the embedding space.

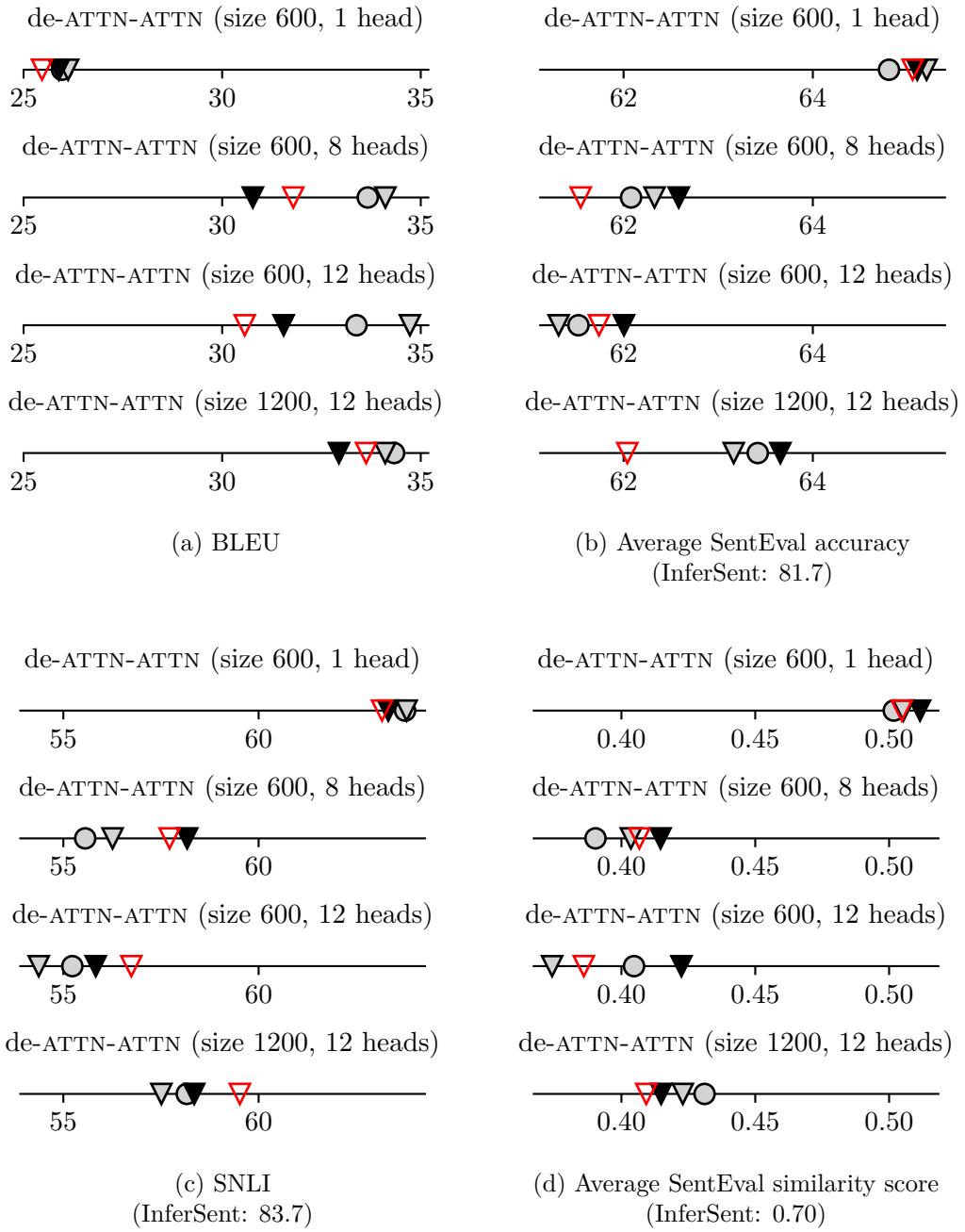
The generally poor performance of *de* models can again be attributed to the discrepancy between the training data and the (news domain) test data, testified by a high perplexity and OOV rate.

5.3.5 InferSent multi-task training

Figs. 5.4 and 5.5 show results on selected evaluation tasks for multi-task variants of *de* and *cs* models, respectively. For each base (MT-only) model, we show its multi-task variant with and without shuffled regression targets (as described in Section 5.1.1) and a variant where the regression task weight α is set to 0. This last variant is included because it differs slightly from the base model due to different initialization and perhaps the different gradient clipping threshold. On the other hand, all multi-task variants, including the one with $\alpha = 0$, are initialized and trained identically.

For *de* models (Fig. 5.4), the most consistent result is that multi-task training always harms the translation quality (BLEU), but it does not seem to matter whether or not the targets are shuffled. In three out of four setups, multi-task training yielded a slight improvement in SNLI and average SentEval accuracy. For SNLI, however, the same or higher improvement was achieved by the models with shuffled targets.

By looking at the regression losses plotted in Fig. 5.6, we can tell that the



(a) BLEU

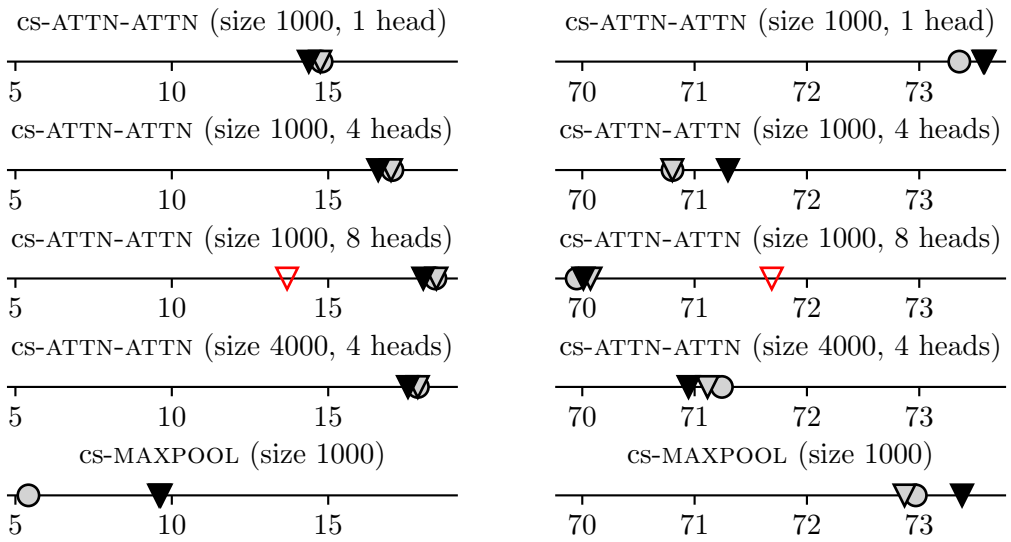
(b) Average SentEval accuracy
(InferSent: 81.7)

(c) SNLI
(InferSent: 83.7)

(d) Average SentEval similarity score
(InferSent: 0.70)

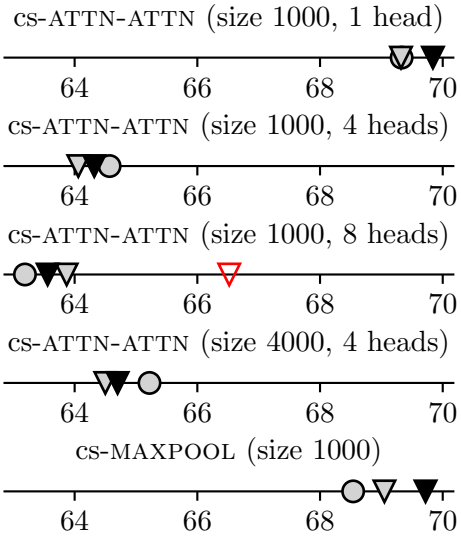
Figure 5.4: Results of selected *de* models and their multi-task variants. Each line shows the following variants of one base model (α is the weight of the regression task):

▼ – $\alpha = 100$ ▽ – $\alpha = 100$, shuffled targets
▽ – $\alpha = 0$ ● – no multi-task (MT only)

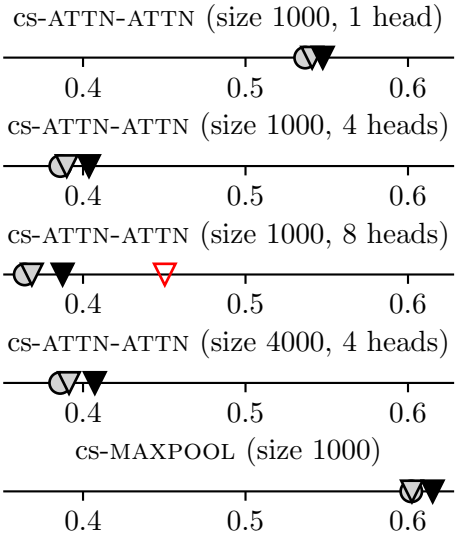


(a) BLEU

(b) Average SentEval accuracy



(c) SNLI



(d) Average SentEval similarity score

Figure 5.5: Results of selected *cs* models and their multi-task variants. Each line shows the following variants of one base model (α is the weight of the regression task):

▼ – $\alpha = 100$ ▽ – $\alpha = 100$, shuffled targets
▽ – $\alpha = 0$ ○ – no multi-task (MT only)

The only model with the ▽ variant is the one with 8 attention heads.

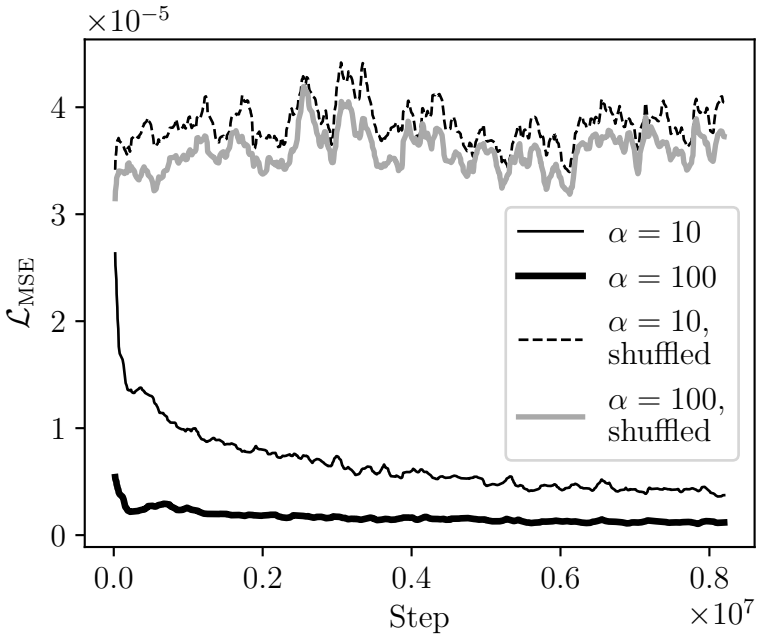


Figure 5.6: Representation regression loss $\mathcal{L}_{\text{MSE}}$ (smoothed) in multi-task variants of the *de*-ATTN-ATTN model (size 600, 8 heads) with and without shuffled regression targets.

model is in fact able to make some sense of the 'real' InferSent embeddings, reaching $\mathcal{L}_{\text{MSE}} < 1$, as opposed to the 'fake' (shuffled) ones, where $\mathcal{L}_{\text{MSE}} > 3$.

For *cs* models (Fig. 5.5), there is a very slight but consistent improvement on the similarity tasks and an equally slight decrease in BLEU.

We trained only one *cs* model with shuffled targets and it yielded a comparatively dramatic improvement on all the selected tasks (along with a substantial drop in BLEU). The regression loss and BLEU of this model are plotted in Fig. 5.7. Interestingly, the model was apparently more successful in minimizing the regression loss (approx. 10^{-5}, compared to $4 \cdot 10^{-5}$ in the *de* model) at the cost of a reduced translation quality.

Fig. 5.8 shows the relative improvements on all tasks. For the *cs* models, the differences are small but mostly positive. For the *de* models, the most important thing to notice is that the pattern is very irregular, but similar between models with and without shuffled targets.

In sum, we do not have enough evidence to conclude that joint regression training with InferSent targets is useful. Quite the contrary – it seems that the few observed improvements are, for the most part, caused by 'blind' regularization which is a side effect of the multi-task training.

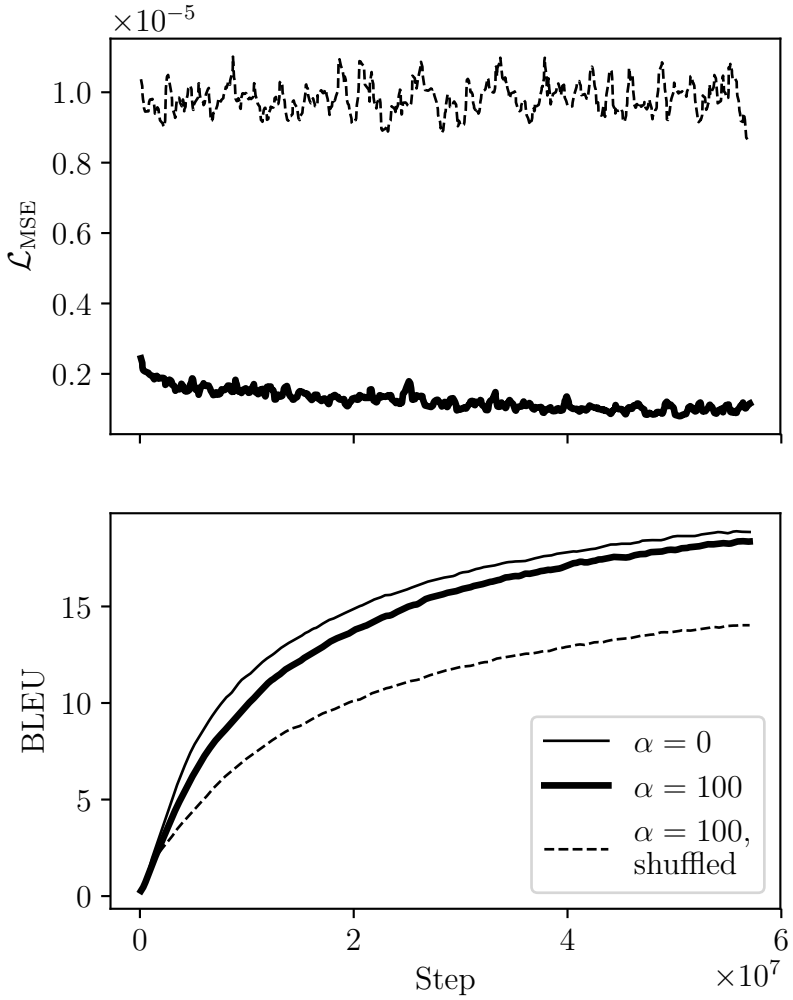


Figure 5.7: Representation regression loss $\mathcal{L}_{\text{MSE}}$ and BLEU (both smoothed) in multi-task variants of the *cs*-ATTN-ATTN model (size 1000, 8 heads) with and without shuffled regression targets. BLEU was computed on the validation set. We do not show $\mathcal{L}_{\text{MSE}}$ for $\alpha = 0$ because the loss is ignored during training.

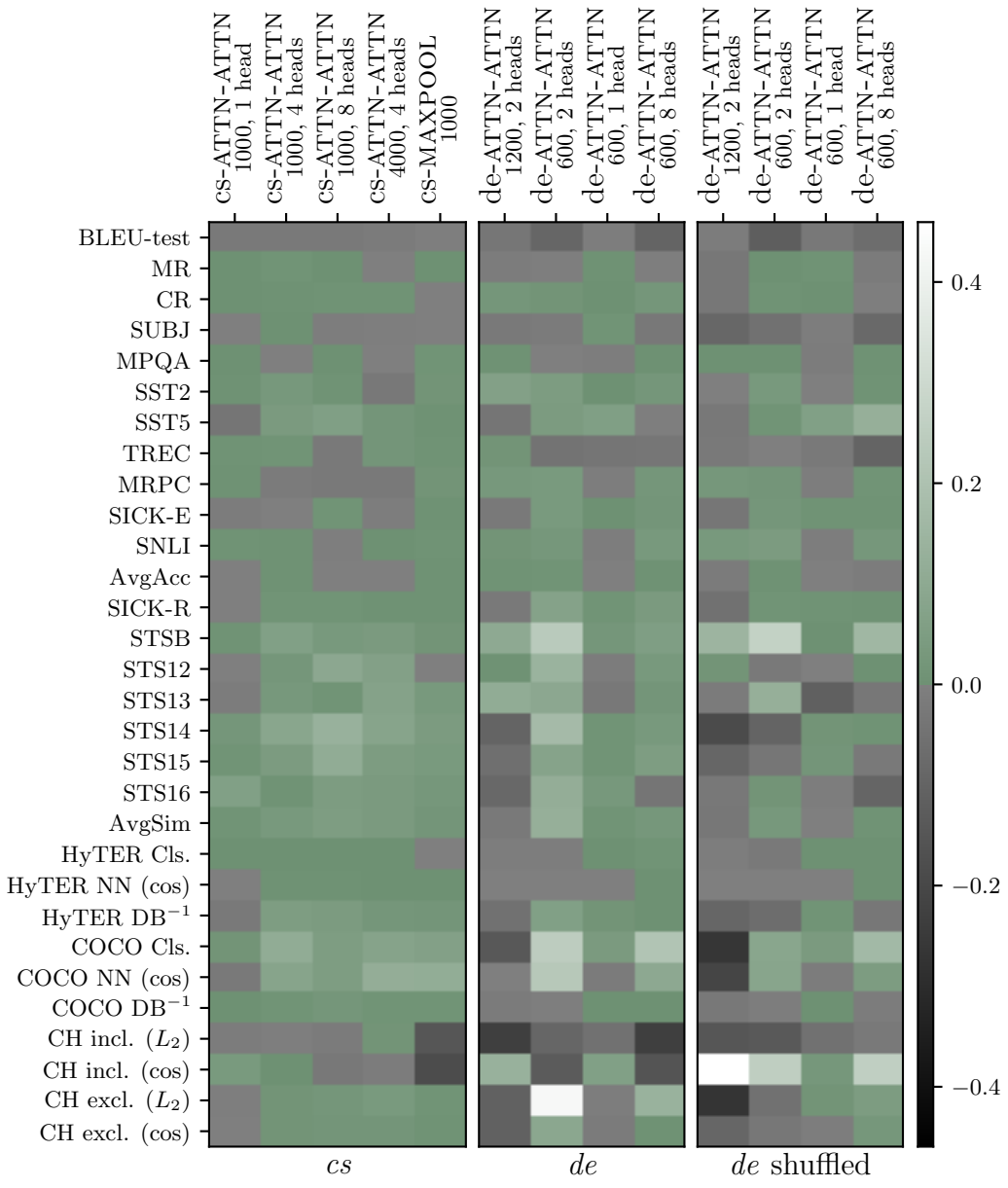


Figure 5.8: Relative differences in the performance of multi-task models with $\alpha = 100$ vs. $\alpha = 0$. Positive differences (cases where the multi-task training improved the result) are shown in shades of green.

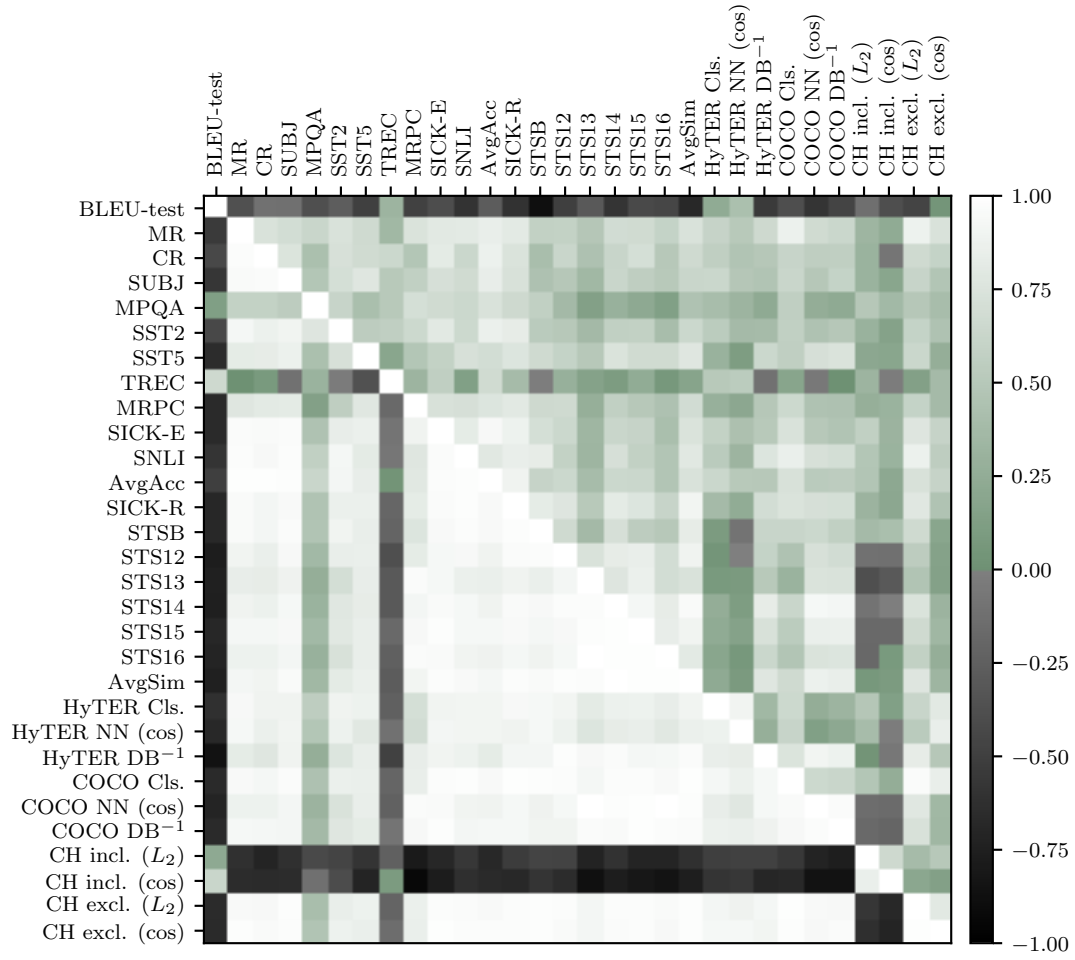


Figure 5.9: Pearson correlations between results on different tasks. Upper triangle: *de* models, lower triangle: *cs* models. Positive correlations are shown in shades of green. Results of similarity tasks are Pearson correlations (Spearman correlations are omitted). AvgAcc and AvgSim stand for average SentEval accuracy and similarity score, respectively. CH stands for compression-headline alignment.

5.4 Discussion

5.4.1 Correlations

To assess the relations between the various metrics, we plotted a heatmap of their Pearson correlations in Fig. 5.9. As one example, Fig. 5.10 details BLEU scores vs. average SentEval accuracy in *cs* models.

A good sign is that on the *cs* dataset, most metrics of representation are positively correlated (the pairwise Pearson correlation is 0.61 ± 0.57 on average), the outliers being TREC (-0.16 ± 0.16 correlation with the other metrics on average) and the 'incl.' variants of domain alignment (-0.62 ± 0.18 on average).

On the other hand, most representation metrics correlate with BLEU negatively (-0.51 ± 0.39) on *cs*. The pattern is less pronounced but still clear also on the *de* dataset.

A detailed understanding of what the learned representations contain is diffi-

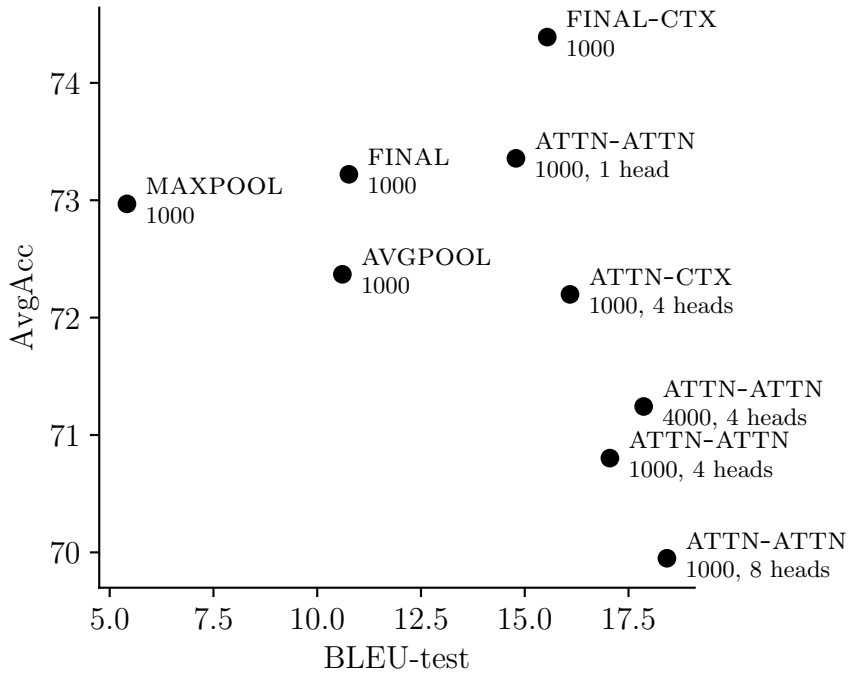


Figure 5.10: BLEU vs. average SentEval accuracy for *cs* models.

cult. We can only speculate that once the NMT model has sufficient capability to follow the source sentence superficially, it will use it and spend its capacity on closely matching the target sentences rather than on deriving a more abstract representation. We assume that this can be a direct consequence of NMT being trained for cross entropy: putting the exact word forms in exact positions as the target sentence requires. Performing well in single-reference BLEU is not an indication that the system understands the meaning but rather that it can maximize the chance of producing the n-grams required by the reference.

5.4.2 Attention interpretation

To interpret the compound attention mechanism, we visualized the induced alignments between source and target sentences while separating the individual attention heads – see Fig. 5.11 for an example and Appendix A for more examples. We noticed that each head tends focus on one segment of the sentence. While one would hope that the segments correspond to some meaningful units (e.g. syntactic functions like subject, predicate or object), we failed to find any such relation for ATTN-ATTN and for *cs* models in general. Instead, the heads divide the source sentence more or less equidistantly, regardless of its length. For example, note how in Fig. 5.11b, the word *osteoporosis*, consisting of 5 tokens, was split between 7 attention heads because the sentence is short, while in the longer sentence in Fig. 5.11a, most heads span 3 tokens. Moreover, the order of the heads' positions in the input sentence seems to be the same for almost all sentences, as documented by the plot in Fig. 5.11c.

We observed a different situation in de-ATTN-CTX models, where the distribution of attention weights for each head was much flatter and we were often able to identify a head focusing on the main verb, as in Fig. 5.12. This might be

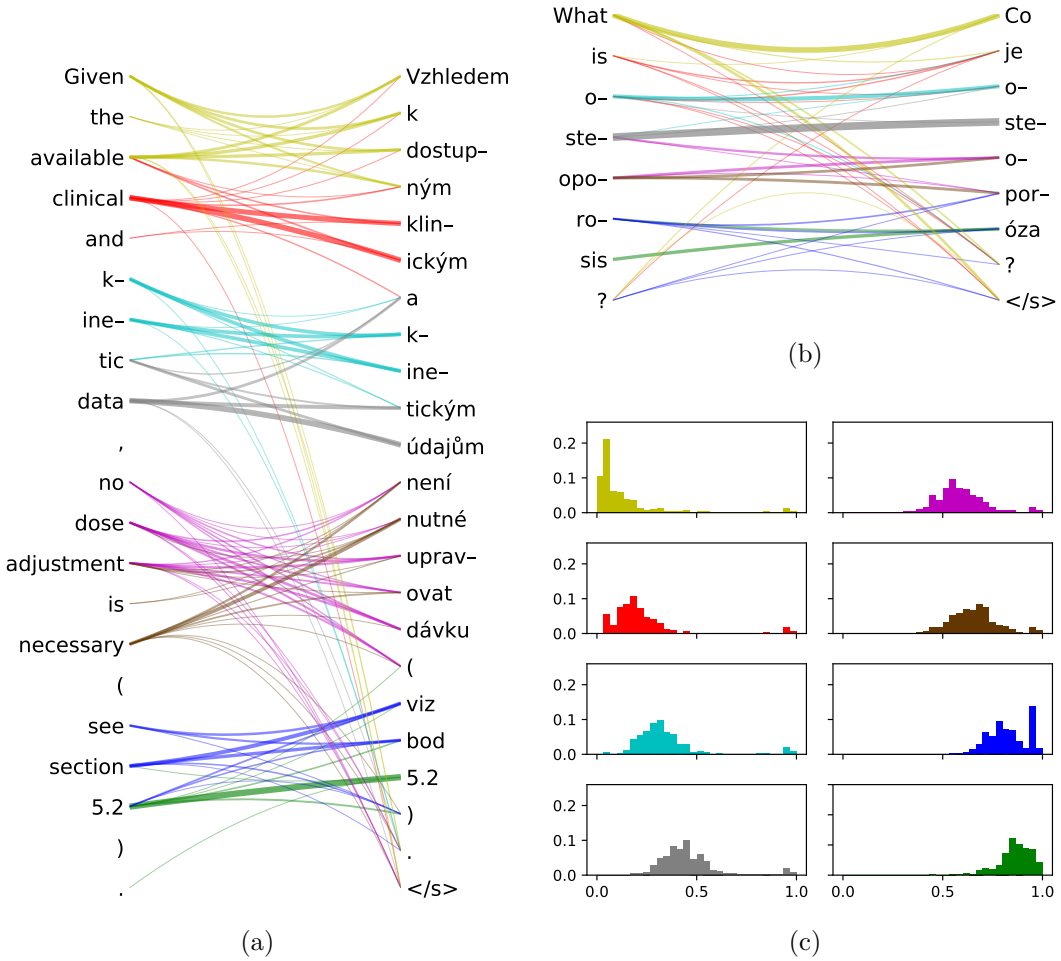


Figure 5.11: Compound attention in the cs-ATTN-ATTN model with 8 attention heads and embedding size of 1000. Best viewed in color. (a) & (b) Induced alignments between source sentences and outputs. Each color represents a different head. The stroke width indicates the alignment weight, with weights ≤ 0.01 pruned out. (c) Inner attention weight by relative position in the source sentence, averaged over the dev set (sentences with less than 8 tokens excluded). Each plot corresponds to one attention head.

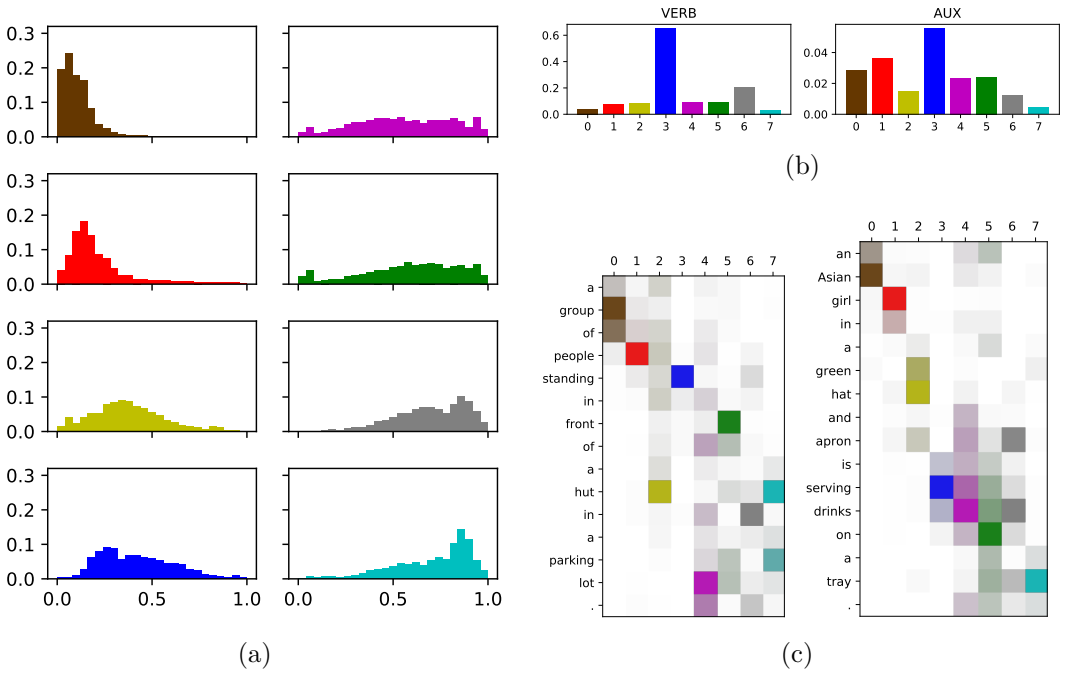


Figure 5.12: Inner attention in the de-ATTN-CTX model with 8 attention heads and embedding size of 600. Best viewed in color. (a) Attention weight by relative position in the source sentence, averaged over the dev set (sentences with less than 8 tokens excluded). Each plot corresponds to one attention head. (b) Total attention weight allotted to verbs (VERB) and auxiliary verbs (AUX) by each head (averaged over the dev set). (c) Attention weights for each head for concrete input sentences.

because the sentences from the Multi30K dataset are simpler compared to CzEng and the position of the main verb is more predictable. However, by looking at the alignments in Fig. 5.12, we can confirm that the 'verb' head (number 3, shown in blue) picks the verb even if it is in an unusual position. The other heads seem to be more position-oriented, but also not entirely.

These observations can help explain why multi-headed models score lower in most representation evaluations: if attention heads are assigned based on position, then semantically close sentences that differ in word order will have very different representations element-wise. For example, consider the following sentences and assume a model with 3 attention heads which splits the sentences in the following way (as delimited by the brackets):

(a) [good food]$_1$ [but]$_2$ [horrible service]$_3$

(b) [horrible service]$_1$ [but]$_2$ [good food]$_3$

Although the strings (a) and (b) have similar meanings, they will inevitably lie far apart in the embedding space of this model. This is because in both cases, e.g. the first component of the embedding will be an average of the two leftmost encoder states, but the inputs to these states will be *good food* in (a) and *horrible service* in (b). In contrast, a model with only 1 attention head will always take a weighted average of the whole sentence, and therefore there is at least some chance that the embeddings of (a) and (b) will lie in each other's proximity.

5.4.3 Dimension importance

We also attempted to interpret individual dimensions of the sentence embeddings based on the *representation erasure* technique of Li et al. (2016a). This method computes the *importance* of dimension d of an internal representation as the relative difference in performance caused by erasing dimension d:

$$I(d) = \frac{1}{|\mathcal{X}|} \sum_{(x,y)\in\mathcal{X}} \frac{S(x,y) - S(x,y,\neg d)}{S(x,y)}, \tag{5.1}$$

where $S(x,y)$ is the log-likelihood of the model computed on an example x for which the correct label is y, and $S(x,y,\neg d)$ is the same quantity when dimension d of the internal representation is erased (set to zero). The importances are then visualized in a plot.

We apply a similar method to the SentEval classification tasks. Our approach differs from that of Li et al. in two regards. Firstly, while Li et al. work with pre-trained networks, we re-run all evaluations from scratch after erasing a given dimension, which requires re-training the classifiers. This might allow the classifiers to compensate for the missing dimension.

Secondly, the SentEval benchmark does not report the per-example loss necessary for evaluating Eq. (5.1). For this reason, we instead compute the importance of dimension d as the actual or relative difference in overall accuracy when the dimension is erased:

$$I_a(d) = \mathrm{Acc} - \mathrm{Acc}_{\neg d}, \tag{5.2}$$

$$I_r(d) = \frac{I_a(d)}{\mathrm{Acc}}. \tag{5.3}$$

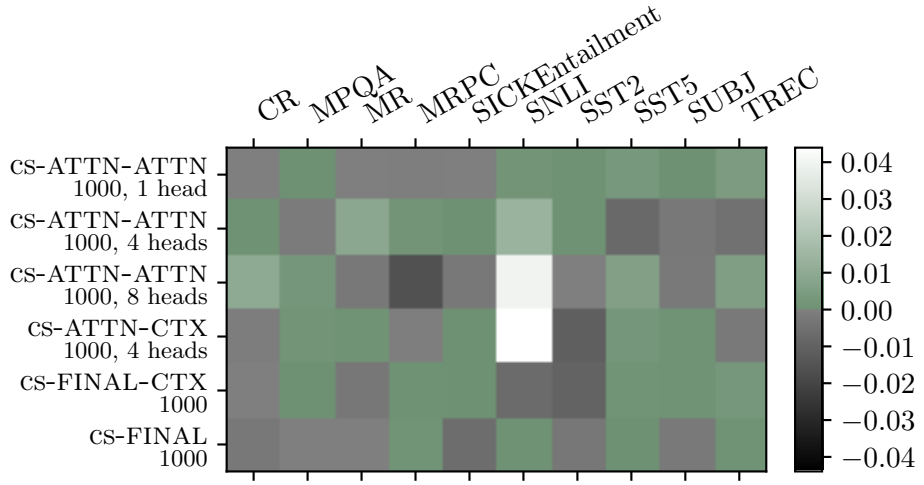


Figure 5.13: Dimension importances (averaged over all dimensions) in different models when evaluated on SentEval classification tasks. Positive importances are shown in shades of green.

Due to the high number of dimensions, we needed to adjust the hyperparameters of the SentEval classifiers to reduce computation time. Specifically, we use the speed-optimized configuration suggested by the authors[5] while setting some options to even more radical values: `optim='rmsprop'` (RMSProp optimizer instead of Adam), `kfold=2` (2-fold cross-validation), `batch_size=256`, `epoch_size=2` and `tenacity=3`.

Examples of the visualizations are included in Appendix A. The results for different models and tasks are summarized in Fig. 5.13 where we computed the actual (non-normalized) differences and averaged them over all dimensions to obtain a single number for each model and task. In general, we find the results hard to interpret. One thing to notice is that for some model/task combinations, most dimensions have a large negative importance, which means that erasing these dimensions improved the performance on that task. This result is yet to be explained, but we suspect that it might be related to the reduced training time of the classifiers. On the other hand, the classifiers are always initialized identically, so we can eliminate random initialization as the culprit.

To obtain reliable results, the experiments should be run repeatedly with different random initializations and the results averaged. Also, the training time of the classifiers should perhaps be extended. However, both options are very computationally expensive. Another option would be to use the representation erasure technique to evaluate the importance of attention heads rather than single dimensions, which would require a much lower number of runs.

[5] `https://github.com/facebookresearch/SentEval#senteval-parameters`

Conclusion

We explored and extensively evaluated continuous sentence representations in NMT systems. In order to apply our methods to the widely used attentive models, we devised a variation on these models that again provides a single meeting point with a continuous representation of the source sentence.

While our proposed 'compound attention' leads to translation quality not much worse than the fully attentive model, it generally does not perform well in meaning representation. Quite on the contrary, we found that the higher the BLEU score, the worse the performance in meaning representation evaluations.

We believe that this observation is important for representation learning where bilingual MT now seems less likely to provide useful data, but perhaps more so for MT itself, where the struggle towards a high single-reference BLEU score (or even worse, cross entropy) leads to systems that refuse to consider the meaning of the sentence.

Future work

We benchmarked our models against InferSent, which uses GloVe word vectors. The effect of pre-trained word embeddings should be looked into; it is entirely possible that our sentence representations would benefit from them substantially. For models based on sub-word units, an option such as BPEmb (Heinzerling and Strube, 2017) could be used.

The new compound attention architecture should also be tested on multilingual NMT (as in Schwenk and Douze, 2017). The sentence representations could then function as a form of interlingua in machine translation. It is also expected that this would make the representations more robust and improve their performance on other tasks.

There are also several possible improvements and extensions to the evaluation protocol. For tasks where the evaluation is based on nearest neighbor search (i.e. paraphrase retrieval and domain alignment), it should be investigated how the number and the character of 'competing' sentences affects the results. In the case of HyTER, which appears to be a very easy dataset to evaluate on, more informative results could be obtained by running the evaluation many times on a very small sample of points from each cluster. Our domain alignment evaluation method also calls for a better dataset, ideally one where the mapping between the domains needs to capture a specific change in meaning. As for the representation erasure technique, its reliability should be further investigated.

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

Amir Bakarov. A survey of word embeddings evaluation methods. *CoRR*, abs/1801.09536, 2018.

Ondrej Bojar, Matous Machácek, Ales Tamchyna, and Daniel Zeman. Scratching the surface of possible translations. In *TSD*, 2013.

Ondřej Bojar, Ondřej Dušek, Tom Kocmi, Jindřich Libovický, Michal Novák, Martin Popel, Roman Sudarikov, and Dušan Variš. CzEng 1.6: Enlarged Czech-English Parallel Corpus with Processing Tools Dockered. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 231–238, Cham / Heidelberg / New York / Dordrecht / London, 2016. Masaryk University, Springer International Publishing. ISBN 978-3-319-45509-9.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. In *CoNLL*, 2016.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST@EMNLP*, 2014a.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN Encoder-Decoder for statistical machine translation. In *EMNLP*, 2014b.

Ondřej Cífka, Aliaksei Severyn, Enrique Alfonseca, and Katja Filippova. Eval all, trust a few, do wrong to none: Comparing sentence generation models. *CoRR*, abs/1804.07972, 2018.

Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *CoRR*, abs/1003.4394, 2010.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *ICML*, 2008.

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

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, 2017a.

Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *CoRR*, abs/1710.04087, 2017b.

James R. Curran and Marc Moens. Improvements in automatic thesaurus extraction. 2002.

David L. Davies and Donald W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1:224–227, 1979.

Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41: 391–407, 1990.

William B. Dolan, Chris Quirk, and Chris Brockett. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *COLING*, 2004.

Markus Dreyer and Daniel Marcu. HyTER: Meaning-equivalent semantics for translation evaluation. In *HLT-NAACL*, 2012.

Markus Dreyer and Daniel Marcu. HyTER networks of selected OpenMT08/09 sentences. Linguistic Data Consortium, Philadelphia, 2014. LDC2014T09.

Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. Multi30K: Multilingual English-German image descriptions. *CoRR*, abs/1605.00459, 2016.

Manaal Faruqui, Yulia Tsvetkov, Pushpendre Rastogi, and Chris Dyer. Problems with evaluation of word embeddings using word similarity tasks. In *RepEval@ACL*, 2016.

Katja Filippova and Yasemin Altun. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1481–1491, 2013.

Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with LSTM. *Neural computation*, 12 10:2451–71, 2000.

Felix A. Gers, Nicol N. Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research*, 3:115–143, 2002.

Alex Graves, Abdel rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

Edward Grefenstette, Mehrnoosh Sadrzadeh, Stephen Clark, Bob Coecke, and Stephen Pulman. Concrete sentence spaces for compositional distributional models of meaning. In *Proceedings of the Ninth International Conference on Computational Semantics*, pages 125–134. Association for Computational Linguistics, 2011.

Kelvin Guu, Tatsunori B Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *CoRR*, abs/1709.08878, 2017.

Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

Benjamin Heinzerling and Michael Strube. Bpemb: Tokenization-free pre-trained subword embeddings in 275 languages. *CoRR*, abs/1710.02187, 2017.

Jindřich Helcl and Jindřich Libovický. Neural Monkey: An open-source tool for sequence learning. *The Prague Bulletin of Mathematical Linguistics*, 107(1): 5–17, 2017a.

Jindřich Helcl and Jindřich Libovický. CUNI system for the WMT17 multimodal traslation task. 2017b.

Karl Moritz Hermann and Phil Blunsom. The role of syntax in vector space models of compositional semantics. In *ACL*, 2013.

Felix Hill, Kyunghyun Cho, and Anna Korhonen. Learning distributed representations of sentences from unlabelled data. In *HLT-NAACL*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 8:1735–80, 1997.

Mohit Iyyer, Varun Manjunatha, Jordan L. Boyd-Graber, and Hal Daumé. Deep unordered composition rivals syntactic methods for text classification. In *ACL*, 2015.

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *ICML*, 2015.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL*, 2014.

Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, and Stephen Pulman. A unified sentence space for categorical distributional- compositional semantics: Theory and experiments. In *Proceedings of 24th International Conference on Computational Linguistics (COLING): Posters*, 2012.

Douwe Kiela, Alexis Conneau, Allan Jabri, and Maximilian Nickel. Learning visually grounded sentence representations. *CoRR*, abs/1707.06320, 2017.

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

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 3294–3302, Cambridge, MA, USA, 2015. MIT Press.

Guillaume Lample, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. *CoRR*, abs/1711.00043, 2017.

Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

Omer Levy and Yoav Goldberg. Linguistic regularities in sparse and explicit word representations. In *CoNLL*, 2014.

Jiwei Li, Will Monroe, and Daniel Jurafsky. Understanding neural networks through representation erasure. *CoRR*, abs/1612.08220, 2016a.

Peng Li, Wei Li, Zhengyan He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. Dataset and neural recurrent sequence labeling model for open-domain factoid question answering. *CoRR*, abs/1607.06275, 2016b.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017.

Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. Understanding of internal clustering validation measures. *2010 IEEE International Conference on Data Mining*, pages 911–916, 2010.

Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. Learning natural language inference using bidirectional LSTM model and inner-attention. *CoRR*, abs/1605.09090, 2016.

Kevin Lund and Curt Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior research methods, instruments, & computers*, 28(2):203–208, 1996.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015.

Jean Maillard, Stephen Clark, and Edward Grefenstette. A type-driven tensor-based semantics for ccg. In *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, 2014.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a.

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168, 2013b.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013c.

Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, 2013d.

Jeff Mitchell and Mirella Lapata. Vector-based models of semantic composition. In *ACL*, 2008.

Jeff Mitchell and Mirella Lapata. Composition in distributional models of semantics. *Cognitive science*, 34 8:1388–429, 2010.

Richard Montague. English as a formal language. *Linguaggi nella Società e nella Tecnica*, 1970.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *ACL 2002, Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, 2002.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

Anna Rogers, Aleksandr Drozd, and Bofang Li. The (too many) problems of analogical reasoning with word vectors. In *Proceedings of the 6th Joint Conference on Lexical and Computational Semantics (*SEM 2017)*, pages 135–148, 2017.

Adriaan M. J. Schakel and Benjamin J. Wilson. Measuring word significance using distributed representations of words. *CoRR*, abs/1508.02297, 2015.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Hinrich Schütze. Automatic word sense discrimination. *Computational Linguistics*, 24:97–123, 1998.

Holger Schwenk and Matthijs Douze. Learning joint multilingual sentence representations with neural machine translation. In *Rep4NLP@ACL*, 2017.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. In *EMNLP*, 2017.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *CoRR*, abs/1511.06709, 2016a.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016b. Association for Computational Linguistics.

Rico Sennrich, Orhan Firat, Kyunghyun Cho, Alexandra Birch, Barry Haddow, Julian Hitschler, Marcin Junczys-Dowmunt, Samuel Läubli, Antonio Valerio Miceli Barone, Jozef Mokry, and Maria Nadejde. Nematus: a toolkit for neural machine translation. In *EACL*, 2017.

Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural MT learn source syntax? In *EMNLP*, 2016.

Richard Socher, Christopher D. Manning, and Andrew Y. Ng. Learning continuous phrase representations and syntactic parsing with recursive neural networks. 2010.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*, 2011.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. Semantic compositionality through recursive matrix-vector spaces. In *EMNLP-CoNLL*, 2012.

Richard Socher, A. V. Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. 2013.

Mark Steedman. *The Syntactic Process*. MIT Press, Cambridge, MA, USA, 2000. ISBN 0-262-19420-1.

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

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *ACL*, 2015.

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

John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2015.

Keiji Yasuda, Ruiqiang Zhang, Hirofumi Yamamoto, and Eiichiro Sumita. Method of selecting training data to build a compact and efficient translation model. In *IJCNLP*, 2008.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. *CoRR*, abs/1611.09100, 2016.

Han Zhao, Zhengdong Lu, and Pascal Poupart. Self-adaptive hierarchical sentence model. In *IJCAI*, 2015.

Jie Zhou, Ying Cao, Xuguang Wang, Peng Li, and Wei Xu. Deep recurrent models with fast-forward connections for neural machine translation. *TACL*, 4:371–383, 2016.

A. Attachments

Below, we give the full results of different evaluations:

- Table A.1 – SentEval classification tasks
- Table A.2 – SentEval similarity tasks
- Table A.3 – paraphrase-based evaluation

Figs. A.1 and A.2 show some additional examples of alignments from the compound attention models. Finally, we give examples of sentence embedding visualizations using the representation erasure technique in Fig. A.3.

Model			% Accuracy										
Name	Size	Heads	MR	CR	SUBJ	MPQA	SST2	SST5	TREC	MRPC	SICK-E	SNLI	Avg.
Most frequent baseline			50.0	63.8	50.0	68.8	49.9	23.1	18.8	66.5	56.7	34.3	48.19
Hill et al. en→fr†	2400	—	64.7	70.1	84.9	81.5	—	—	82.8	96.1	—	—	—
InferSent†	4096	—	81.1	86.3	92.4	90.2	84.6	—	88.2	76.2	86.3	(84.5)	—
InferSent	4096	—	81.5	86.7	92.7	90.6	85.0	45.8	88.2	76.6	86.4	83.7	81.7
GloVe-BOW	300	—	77.0	78.2	91.1	87.9	81.0	44.4	82.0	72.3	78.2	66.0	75.8
cs-FINAL-CTX	1000	—	68.7	77.4	88.5	85.5	73.0	38.2	88.6	71.8	82.1	70.2	74.4
cs-ATTN-ATTN	1000	1	68.2	76.0	86.9	84.9	72.0	35.7	89.0	70.7	80.8	69.3	73.4
cs-FINAL	1000	—	67.9	75.7	87.6	84.7	72.5	36.2	86.0	71.4	81.1	69.2	73.2
cs-MAXPOOL	1000	—	67.4	75.2	86.9	84.3	70.3	37.5	85.8	72.1	81.7	68.5	73.0
cs-AVGPOOL	1000	—	66.5	74.1	86.5	85.0	71.9	36.7	85.4	70.0	79.7	67.8	72.4
cs-ATTN-CTX	1000	4	66.5	74.8	85.7	84.7	70.1	36.1	88.2	70.4	79.5	66.0	72.2
cs-ATTN-ATTN	4000	4	64.9	72.7	84.3	85.1	70.1	33.5	88.8	69.7	78.0	65.2	71.2
cs-ATTN-ATTN	1000	4	64.0	72.6	84.6	84.2	67.9	33.2	89.0	70.1	78.0	64.6	70.8
cs-ATTN-ATTN	1000	8	62.9	71.7	83.6	84.2	67.0	34.2	86.2	69.8	76.6	63.2	70.0
de-MAXPOOL-CTX	600	—	60.0	69.2	77.0	73.1	61.4	32.4	80.2	70.7	78.8	68.0	67.1
de-ATTN-CTX	1200	12	61.1	70.0	77.3	71.7	63.5	32.4	78.4	69.8	77.4	65.0	66.7
de-ATTN-CTX	600	8	60.5	68.5	77.0	72.1	62.0	31.1	77.0	70.1	75.7	64.0	65.8
de-AVGPOOL-CTX	600	—	59.5	67.5	75.6	72.5	64.1	29.3	74.6	70.8	77.5	65.2	65.6
de-ATTN-CTX	600	12	59.7	68.4	77.0	71.2	61.2	30.9	78.0	71.1	76.0	61.9	65.5
de-FINAL	600	—	59.9	65.9	76.2	72.7	61.5	31.4	73.0	70.7	77.0	64.7	65.3
de-ATTN-CTX	600	3	60.3	67.0	75.4	72.7	60.6	30.4	77.0	69.9	76.0	63.3	65.3
de-ATTN-ATTN	600	1	60.0	66.5	72.8	72.2	61.7	29.5	74.2	70.5	76.9	63.8	64.8
de-ATTN-ATTN	600	3	60.7	67.5	74.1	71.8	60.6	30.1	75.0	69.5	74.7	61.5	64.5
de-FINAL-CTX	600	—	58.9	66.2	73.1	71.9	61.0	29.2	75.8	70.3	76.2	62.6	64.5
de-ATTN-ATTN	1200	6	58.7	65.9	75.4	72.3	61.0	29.7	78.4	70.1	72.3	59.6	64.3
de-TRF-ATTN-ATTN	600	3	58.8	64.9	76.2	71.7	60.3	30.4	72.0	71.2	72.5	61.4	63.9
de-ATTN-ATTN	1200	12	58.6	66.9	74.1	70.7	60.8	29.5	75.8	67.1	72.5	58.2	63.4
de-ATTN-ATTN	2400	12	57.4	66.0	74.0	70.9	58.5	27.7	76.0	67.7	73.9	59.8	63.2
de-TRF-ATTN-ATTN	2400	12	56.9	65.3	74.4	71.2	61.2	30.5	74.0	66.1	71.2	59.0	63.0
de-ATTN-ATTN	600	6	57.4	64.8	72.4	71.8	59.5	27.2	76.0	68.6	70.9	57.5	62.6
de-ATTN-ATTN	600	8	57.5	64.5	71.7	71.8	58.8	28.1	77.4	67.0	68.6	55.6	62.1
de-TRF-ATTN-ATTN	600	6	57.8	64.6	72.0	70.8	59.3	29.2	65.6	69.1	71.0	59.5	61.9
de-ATTN-ATTN	600	12	56.0	65.6	73.1	70.5	57.6	28.6	74.0	64.1	70.5	55.2	61.5
de-TRF-ATTN-ATTN	1200	12	56.6	64.9	71.4	71.0	56.7	29.6	66.2	67.9	68.8	58.2	61.1
de-ATTN-CTX	600	6	58.4	63.9	72.9	70.6	57.4	29.6	58.6	66.5	68.7	62.9	61.0
LM perplexity (*cs*)			1362.5	736.4	1059.0	3213.3	2099.1	1340.8	338.2	863.0	299.4	190.6	1150.2
% OOV (*cs*)			4.2	2.5	3.6	0.9	3.4	4.2	0.6	3.5	0.2	0.3	2.3
LM perplexity (*de*)			3776.8	2639.3	3137.7	8740.0	5003.3	3519.2	3790.8	5070.7	65.0	38.8	3578.2
% OOV (*de*)			22.8	13.1	21.0	27.9	24.4	23.3	16.7	25.6	1.7	1.5	17.8

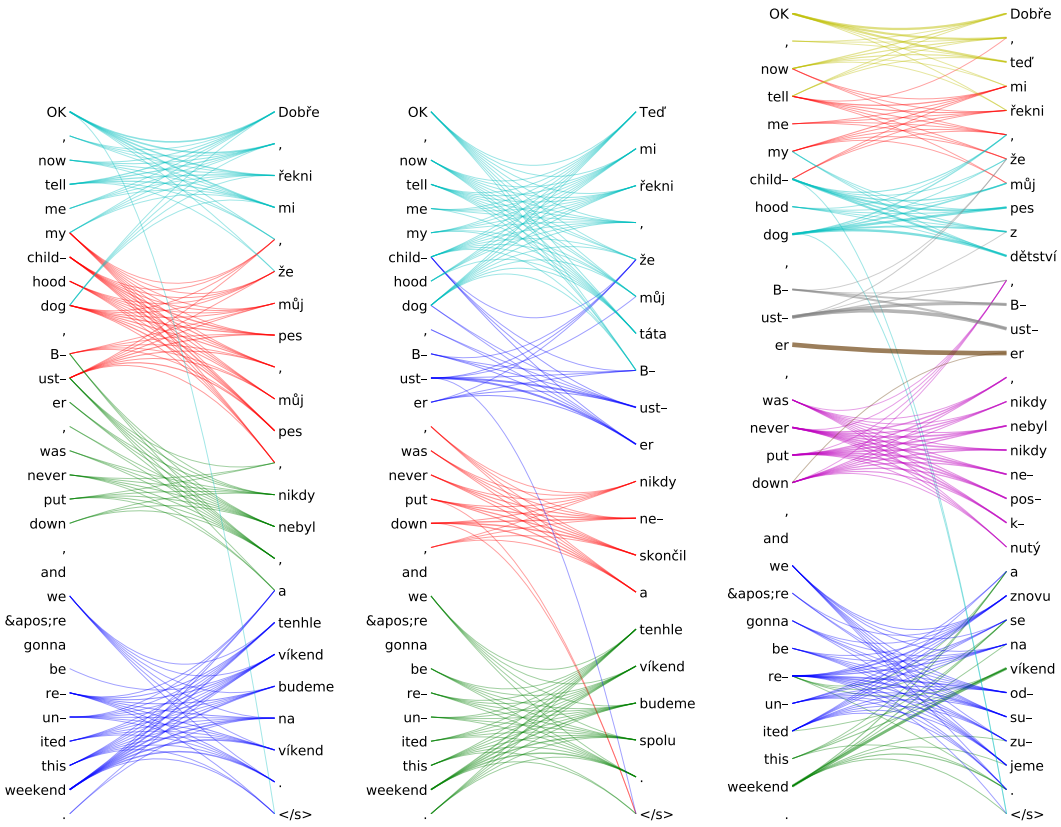
Table A.1: SentEval classification accuracies. Reprinted results are marked with †, others are our measurements. Perplexity and OOV rate as in Table 5.3.

Model			Correlation (Pearson/Spearman)							
Model	Size	Heads	SICK-R	STSB	STS12	STS13	STS14	STS15	STS16	Avg.
InferSent†	4096	—	.88/–	—	—	—	.70/.67	—	—	—
InferSent	4096	—	.88/.83	.76/.75	.59/.60	.59/.59	.70/.67	.71/.72	.71/.73	.70
cs-MAXPOOL	1000	—	.81/.75	.72/.71	.52/.53	.47/.47	.54/.53	.61/.61	.58/.58	.60
cs-FINAL	1000	—	.80/.74	.74/.75	.54/.56	.42/.43	.55/.53	.60/.59	.55/.56	.60
cs-FINAL-CTX	1000	—	.82/.76	.74/.74	.51/.53	.44/.44	.52/.50	.62/.61	.57/.58	.60
GloVe-BOW	300	—	.80/.72	.64/.62	.52/.53	.50/.51	.55/.56	.56/.59	.51/.58	.59
cs-ATTN-ATTN	1000	1	.81/.76	.73/.73	.46/.49	.32/.33	.45/.44	.53/.52	.47/.48	.54
de-ATTN-CTX	1200	12	.76/.70	.52/.51	.46/.49	.31/.31	.50/.50	.58/.57	.51/.52	.52
de-ATTN-CTX	600	8	.75/.68	.52/.50	.47/.49	.30/.31	.52/.52	.56/.56	.48/.49	.51
de-ATTN-ATTN	600	1	.74/.67	.56/.55	.46/.48	.30/.31	.48/.48	.53/.53	.46/.47	.50
de-ATTN-CTX	600	3	.72/.65	.53/.52	.45/.48	.34/.35	.48/.48	.55/.54	.46/.46	.50
de-ATTN-CTX	600	12	.75/.68	.51/.49	.46/.47	.28/.29	.51/.50	.54/.54	.48/.48	.50
cs-AVGPOOL	1000	—	.78/.72	.70/.70	.47/.49	.29/.30	.38/.39	.44/.44	.41/.43	.50
de-MAXPOOL-CTX	600	—	.77/.71	.61/.60	.46/.48	.26/.28	.46/.46	.51/.52	.40/.42	.50
de-TRF-ATTN-ATTN	600	3	.70/.63	.53/.52	.47/.48	.31/.31	.47/.47	.52/.51	.47/.47	.49
de-AVGPOOL-CTX	600	—	.76/.69	.59/.58	.44/.46	.25/.27	.45/.45	.50/.49	.41/.42	.48
de-FINAL-CTX	600	—	.73/.66	.57/.55	.44/.47	.25/.27	.43/.43	.52/.51	.44/.44	.48
de-FINAL	600	—	.73/.66	.62/.60	.41/.44	.22/.24	.43/.43	.47/.47	.44/.44	.47
de-ATTN-ATTN	600	3	.67/.62	.50/.49	.44/.47	.27/.28	.43/.44	.50/.49	.45/.45	.47
de-TRF-ATTN-ATTN	2400	12	.66/.59	.50/.49	.42/.42	.28/.28	.46/.45	.51/.51	.44/.45	.46
de-TRF-ATTN-ATTN	1200	12	.61/.58	.51/.50	.44/.46	.26/.28	.43/.43	.50/.50	.47/.47	.46
de-TRF-ATTN-ATTN	600	6	.66/.59	.51/.49	.44/.45	.27/.28	.43/.43	.50/.51	.39/.41	.45
cs-ATTN-CTX	1000	4	.74/.70	.64/.64	.35/.38	.26/.27	.31/.31	.44/.44	.39/.40	.45
de-ATTN-ATTN	1200	12	.63/.58	.40/.39	.40/.43	.28/.29	.40/.41	.50/.49	.42/.41	.43
de-ATTN-CTX	600	6	.60/.57	.47/.47	.37/.38	.23/.26	.42/.43	.47/.48	.42/.44	.43
de-ATTN-ATTN	2400	12	.58/.59	.40/.39	.41/.44	.22/.25	.39/.39	.47/.47	.39/.38	.41
de-ATTN-ATTN	1200	6	.66/.60	.39/.39	.39/.42	.21/.23	.37/.37	.46/.45	.40/.39	.41
de-ATTN-ATTN	600	12	.59/.55	.40/.39	.39/.43	.24/.25	.37/.37	.46/.46	.39/.38	.40
de-ATTN-ATTN	600	6	.61/.56	.39/.38	.40/.43	.22/.23	.36/.36	.45/.45	.38/.37	.40
de-ATTN-ATTN	600	8	.57/.52	.37/.36	.38/.41	.24/.25	.35/.36	.46/.44	.38/.36	.39
cs-ATTN-ATTN	1000	4	.70/.66	.57/.56	.29/.32	.22/.21	.25/.25	.35/.35	.34/.34	.39
cs-ATTN-ATTN	4000	4	.72/.67	.57/.56	.29/.32	.22/.22	.24/.24	.36/.35	.32/.32	.39
cs-ATTN-ATTN	1000	8	.70/.65	.54/.52	.28/.31	.20/.20	.22/.22	.31/.32	.32/.33	.36
LM perplexity (*cs*)			299.4	1338.8	697.2	2783.9	1716.8	995.6	737.8	1224.2
% OOV (*cs*)			0.2	3.6	2.9	2.6	3.3	2.5	3.0	2.6
LM perplexity (*de*)			65.0	1301.4	1621.0	5041.8	2364.6	1096.7	2583.5	2010.6
% OOV (*de*)			1.7	19.6	18.5	23.5	19.9	13.3	17.2	16.2

Table A.2: Similarity scores (Pearson/Spearman). 'Avg.' averages both correlation coefficients for all tasks. Perplexity and OOV rate as in Table 5.3.

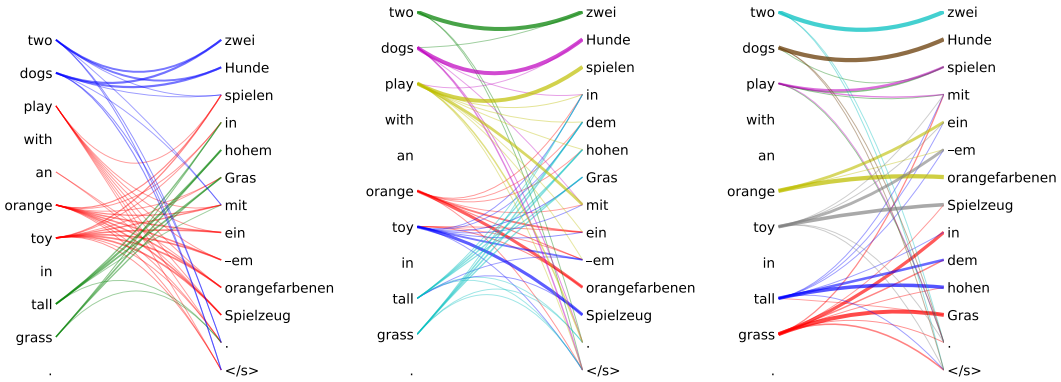
Model			HyTER			COCO			Avg.
Name	Size	Heads	Cls.	NN (L_2/cos)	DB^{-1}	Cls.	NN (L_2/cos)	DB^{-1}	
InferSent	4096	—	99.99	100.00/100.00	0.579	31.58	25.28/26.21	0.367	48.0
GloVe-BOW	300	—	99.94	100.00/100.00	0.654	34.28	20.29/19.72	0.352	46.9
cs-FINAL-CTX	1000	—	99.92	100.00/100.00	0.406	23.20	15.74/16.07	0.346	44.5
cs-MAXPOOL	1000	—	99.86	100.00/100.00	0.447	21.76	15.01/16.34	0.348	44.2
de-ATTN-CTX	600	8	98.11	99.86/ 99.90	0.348	21.64	15.40/17.32	0.349	44.1
cs-FINAL	1000	—	99.91	100.00/100.00	0.439	22.40	14.31/14.63	0.340	44.0
de-ATTN-CTX	1200	12	98.88	99.85/ 99.91	0.347	20.06	14.92/16.68	0.348	43.9
de-MAXPOOL-CTX	600	—	98.42	99.89/ 99.90	0.343	21.54	14.65/15.62	0.341	43.8
de-ATTN-CTX	600	3	97.81	99.77/ 99.87	0.328	19.74	15.28/16.43	0.343	43.7
de-ATTN-CTX	600	12	97.79	99.84/ 99.89	0.360	20.22	14.54/16.10	0.344	43.6
de-ATTN-CTX	600	6	98.11	99.79/ 99.86	0.358	20.44	14.48/15.57	0.342	43.6
de-ATTN-ATTN	600	1	97.70	99.71/ 99.73	0.352	19.74	14.95/16.26	0.340	43.6
de-AVGPOOL-CTX	600	—	97.72	99.59/ 99.60	0.312	20.04	13.49/14.27	0.337	43.2
cs-ATTN-ATTN	1000	1	99.88	99.91/ 99.91	0.347	21.54	11.15/11.50	0.331	43.1
de-ATTN-ATTN	600	3	97.42	99.64/ 99.75	0.314	17.36	13.35/14.35	0.333	42.8
de-FINAL	600	—	97.01	99.14/ 99.30	0.305	19.88	11.41/12.40	0.328	42.5
de-FINAL-CTX	600	—	96.65	99.66/ 99.70	0.323	17.22	12.06/12.84	0.333	42.3
de-TRF-ATTN-ATTN	600	3	95.79	99.61/ 99.64	0.315	15.76	13.20/14.04	0.340	42.3
cs-AVGPOOL	1000	—	99.80	99.99/ 99.99	0.387	17.90	8.36/ 8.61	0.311	41.9
de-ATTN-ATTN	1200	12	97.15	99.47/ 99.65	0.283	12.18	11.09/11.97	0.330	41.5
de-ATTN-ATTN	1200	6	98.05	99.74/ 99.80	0.289	11.90	9.84/10.69	0.327	41.3
de-ATTN-ATTN	2400	12	98.69	99.65/ 99.77	0.287	10.26	9.96/10.94	0.326	41.2
cs-ATTN-CTX	1000	4	99.75	99.72/ 99.74	0.287	14.60	7.52/ 7.54	0.318	41.2
de-ATTN-ATTN	600	6	96.03	99.62/ 99.71	0.287	12.22	9.92/10.59	0.323	41.1
de-TRF-ATTN-ATTN	2400	12	95.82	99.05/ 99.03	0.307	5.66	13.85/14.53	0.339	41.1
de-ATTN-ATTN	600	8	95.32	99.59/ 99.73	0.275	10.22	9.56/10.58	0.325	40.7
de-ATTN-ATTN	600	12	95.16	99.52/ 99.64	0.278	9.62	9.59/10.47	0.323	40.6
de-TRF-ATTN-ATTN	600	6	90.24	98.39/ 98.44	0.313	9.06	12.98/13.64	0.332	40.4
de-TRF-ATTN-ATTN	1200	12	90.71	98.21/ 98.22	0.301	7.06	13.10/13.70	0.333	40.2
cs-ATTN-ATTN	4000	4	99.54	98.89/ 98.98	0.252	11.52	5.54/ 5.51	0.303	40.1
cs-ATTN-ATTN	1000	4	99.26	98.90/ 98.93	0.253	10.84	5.16/ 5.20	0.299	39.9
cs-ATTN-ATTN	1000	8	99.41	98.17/ 98.09	0.243	10.24	4.51/ 4.64	0.287	39.4
LM perplexity / % OOV (cs)				668.5 / 1.2			238.5 / 0.1		
LM perplexity / % OOV (de)				3354.8 / 19.3			86.3 / 1.9		

Table A.3: Paraphrase evaluation on HyTER and COCO – 'Cls.' is the cluster classification accuracy, 'NN' is the nearest-neighbor paraphrase retrieval accuracy and DB^{-1} is the inverse Davies-Bouldin index. 'Avg.' is simply the average of each row. Perplexity and OOV rate as in Table 5.3.



(a) Size 1000, 4 heads. (b) Size 4000, 4 heads. (c) Size 1000, 8 heads.

Figure A.1: Example alignments from cs-ATTN-ATTN models.



(a) Size 600, 3 heads. (b) Size 1200, 6 heads. (c) Size 600, 8 heads.

Figure A.2: Example alignments from de-ATTN-ATTN models.

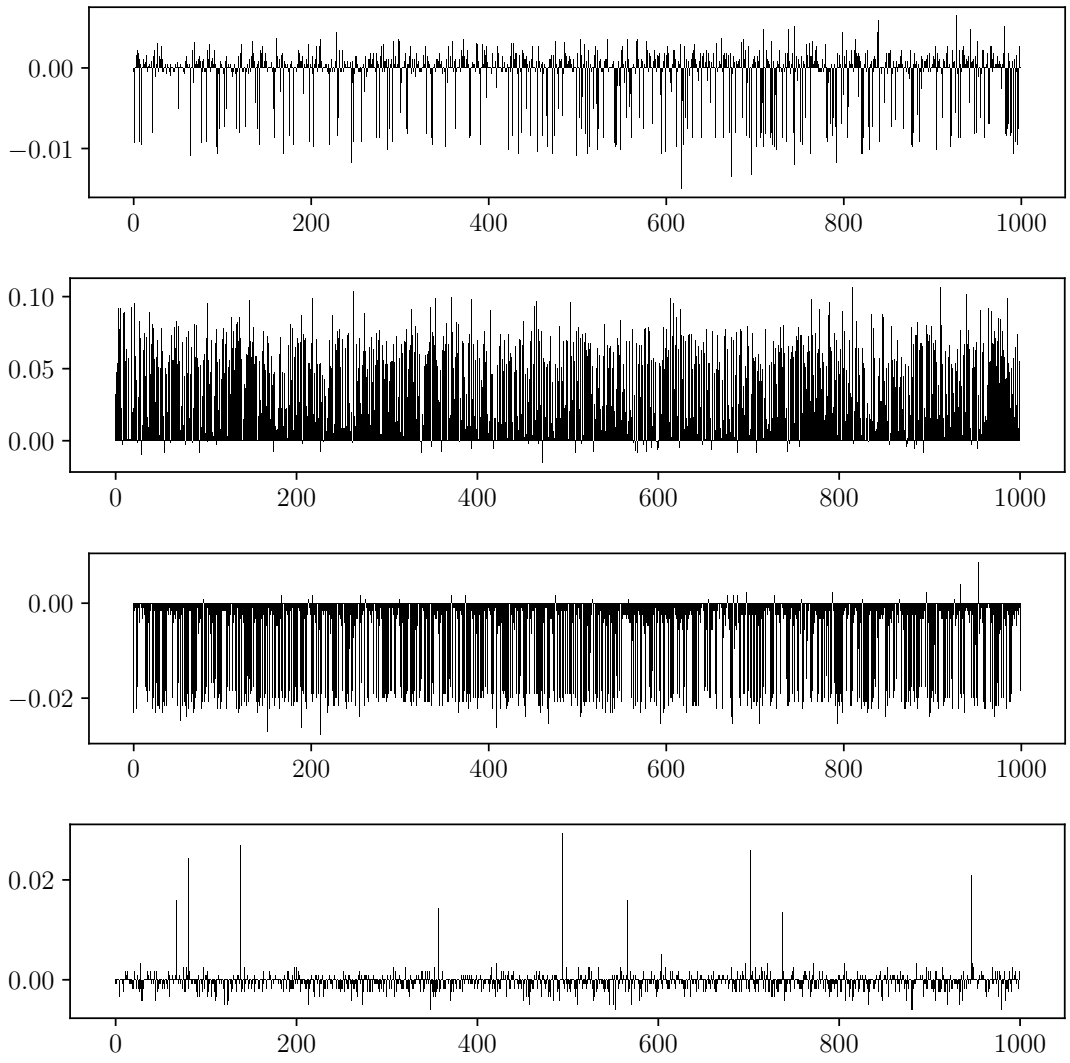


Figure A.3: Sentence embeddings from the cs-ATTN-CTX (4 attention heads, size 1000) visualized using representation erasure on CR, SNLI, SST2 and MRPC from SentEval (top to bottom). Each vertical bar shows the relative importance of one dimension of the embedding.