Understanding and exploiting subspace organization in contextual word embeddings

David Yenicelik
ETH Zürich

A dissertation submitted to ETH Zürich
in partial fulfilment of the requirements for the degree of
Master of Science ETH in Computer Science

Under the supervision of
Florian Schmidt
Yannic Kilcher
Prof. Dr. Thomas Hofmann

Eidgenössische Technische Hochschule Zürich
Data Analytics Lab
Institute of Machine Learning
CAB F 42.1, Universitätsstrasse 6, 8006, Zürich
Zürich 8006
Switzerland

Email: yedavid@ethz.ch

May 4, 2020

Declaration

I David Yenicelik of ETH Zürich, being a candidate for the M.Sc. ETH in Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14,235

Signed:

Date:

Abstract

Write a summary of the whole thing. Make sure it fits in one page.

How can we arrive at meaning embeddings? Compared to context-embeddings, these should encode meaning better and exclusively.

Better represent what a computer means, can be used for interpretability, downstream tasks, better unbiased embeddings which do not entail sentiment but meaning only with the words.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the goals of Natural Language Understanding *NLU* is to formalize written text in a way, such that linguistic features can be captured through computational models, which can then again be used for downstream machine learning tasks. The most popular methodology in NLU is the use of vectors that represent written text, including word-vectors, sentence-vectors and more. These vectors are often referred to as *embedding* vectors, as they embed more complex concepts into a vector representation. Optimally, the algebra of the space the we operate in, including the various operations such as multiplication and addition should have meaningful relations to the concepts captured by these embeddings.

Language models *LM* are a generalization of NLU models that can both generate word-embeddings, sentence embeddings, but also intermediate representations which can be used for downstream tasks. Most commonly, language models take into consideration the context that a word token appears and, and includes this in the calculation when creating the embedding.

Because embeddings - due to their strong usage in downstream tasks - can be considered as the fundamental unit of machine learning in the domain

of natural language, any shortcomings will propagate to the performance of the target task. Thus, improving the way these embeddings have strong implications for any subtasks in NLU, including but not limited to named entity recognition *NER*, sentiment analysis and translation between languages. However, most of the modern LMs are so complex that these are considered black-box models.

Although this work does not try to imminently push the performance of these language models, we believe that providing a better understanding of the underlying principles will pave the way for future research to better track these issues. For the sake of focus, and because we believe that meaning is the most important linguistic feature for communication, we will focus on investigating how modern LMs capture semantics.

1.0.1 Main Contributions

Our main contribution lies in conducting experiments which test how semantics is captured in one of the most popular language models. Because discriminative tasks are simpler than generative ones, first we analyse the linear separability of semantics within embedding vectors produced by BERT. Then we check how natural this separability is entailed, by trying to match an implicit generative distribution onto it through clustering. We then investigate to what effect introducing additional embedding vectors inside modern language models has. We aim to use simple and easy to understand models for best interpretability of the results.

In conclusion, we show that semantics in modern language models is not interpretable using simple linear models, implying that model complex language models are still required for downstream tasks. We also show that linguistic features such as sentiment are much more strongly captured in modern language models, allowing for language stereotypes to easily manifest itself, leading to stronger cases of bias in language models.

Chapter 2

Background

[30] was one of the early works that argue that that there is inherent structure in language, and that this structure can be formalized. They also mention that the relation between the linguistic representation in terms of sounds and tokens are related to the meaning that the representation entail. This representation is often captured in form of a word token w, such as `bank` or `cat`. However, despite the obvious relationship, the distinction between distributional structure of the language and semantics is not a formal one. For the above case, the token `bank` may have different meanings $m_{\text{bank-financial instution}}$, $m_{\text{bank-sea bank}}$, whereas the token `cat` will have a more straight-forward semantic interpretation of $m_{\text{cat-animal}}$. [30] argues that there is a parallel semantic structure, and argues that this is not a one-to-one relation between vocabulary and different semantic classes. Generally, one of the man views of this paper is that the meaning of a word is defined by the context it carries. The formalization of language is not a trivial problem to consider, as this also touches base on a philosophical level [31], [88], posing the question on the nature of the relation between though and (language)-representation.

2.1 Linguistic Features

There is a vast number of linguistic features, going from phonological features, morphological and syntactic features to semantic features. This is not an exhaustive summary, but focusing on the properties of languages relevant to this work.

We will first look at language, specifically at properties that are manifested within languages, called linguistic features. These make a formal analysis of our topic easier.

Polysemy describes the phenomenon that a word token w may have multiple meanings. For the above example the number of meanings that `cat` can entail is $|M_{\text{cat}}| = |\{m_{\text{cat-animal}}\}| = 1$, whereas the number of meanings that `bank` can entail is $|M_{\text{bank}}| = |\{m_{\text{bank-financial instution}}, m_{\text{bank-sea bank}}\}| = 2$. These are just examples, but the true numbers vary depending on the granularity chosen by humans (which again is a non-trivial question of when one concept stops, and another distinct concept starts) and the language viewed. This definition and investigation is often left to linguists to decide.

Part of Speech *PoS* implies the category of syntactic function of a word. These categories include nouns, verbs and adjectives, which follow certain grammatical rules. There is a number of such classes, including adjectives *ADJ*, adposition *ADP*, adverbs *ADV*, auxiliary *AUX*, conjunction *CONJ*, coordinating conjunction *CCONJ*, determiner *DET*, interjection noun *INTJ*, noun *NOU*, numeral *NUM*, particle *PART*, pronoun*PRON*, proper noun *PROPN*, punctuation *PUNCT*, subordinating conjunction *SCONJ*, symbol *SYM*, verb *VERB*, as is defined by [1]. In this work, we will be using the the spaCy python package [34] whenever we need to apply computation on word-tokens to identify the underlying part of speech class. Within this work, we focus on intuitively interpretable aspects, and thus limit our work on nouns, adjectives, adverbs and verbs.

Other linguistic features are recorded in [83] and include

Coarse-Grained Categories	Fine-Grained Categories
Lexical Semantics	Lexical Entailment, Morphological Negation, Factivity, Symmetry/Collectivity, Redundancy, Named Entities, Quantifiers
Predicate Argument Structure	Core Arguments, Prepositional Phrases, Ellipsis Implicits, Anaphora/Coreference Active/Passive, Nominalization, Genitives/Partitives, Datives, Relative Clauses, Coordination Scope, Intersectivity, Restrictivity
Logic	Negation, Double Negation, Intervals/Numbers, Conjunction, Disjunction, Conditionals, Universal, Existential Temporal, Upward Monotone Downward Monotone, Non-Monotone
Knowledge	Common Sense, World Knowledge

Table 2.1: Taken from [83] Table to test captions and labels

2.1.1 Tokens and n-grams

The most trivial basic unit of language is considered to be a token w. In the above sections, we have implicitly made the assumption that this token w is always a word. However, there is a variety of ways that sentences can be encoded into a sequence of tokens. In a sentence such as

```
The man travelled down the road.
```

this would constitute the basic units of computation to be the set of

$$\{\text{the}, \text{man}, \text{travelled}, \text{down}, \text{road}\}$$

The way a sentence is split up into indiviudal basic units of language is referred to as *tokenization*. As such, the sentence would be tokenized into the sequence (assuming lower-casing is part of the tokenization)

$$[the, man, travelled, down, the, road]$$

It is important to note that there are also other ways to interpret language tokens. I will provide insights on different formats, as the language models that we will be using use other tokenization methods.

Character s can be considered as another base unit of language. For the sentence (**??**), the set of basic units would correspond to the latin alphabet plus whitespace $a - z$"" (" " denotes whitespace), and the above sentence would be tokenized into the sequence

$$[t, he, "", m, a, n, "", t, r, a, v, e, l, l, e, d, "", d, o, w, n, "", t, h, e, "", r, o, a, d]$$

Choosing characters as the basic token levels has shown considerable success and popularity in language modelling [78] and translation [42].

Byte pair encodings [24] are another popular mechanism to tokenize sentences, and also shows success in the case of language modelling and translation [71]. Here, the most frequent pair of bytes in a sequence is replaced with the most frequent pair of bytes in a sequence with a single, unused byte.

Depending on what corpus this tokenizer is trained on, the above sentence could be tokenized into the following sequence

$$[the., "", ma., n., "", t., rav., e., lled., "", do., w., n., "", the., "", r., oa., d.]$$

The tokenization of the sentence above implies that the corpus that the tokenizer was trained on includes as a majority of word-occurences the sequence of characters *the*, as well as *lled* and *do*. The . (dot) at the end of each character denotes the end of the byte-pair. The main idea behind this tokenization is to construct a vocabulary of frequent subwords.

WordPiece tokenizer [89] follows a similar idea as the byte-pair encoding, and constructs a vocabulary given a corpus by maximizing the entropy. The main difference to the byte-pair encoding is that the vocabulary naturally includes a set of the most commonly appearing words in the respective language, and is then aggregated by introducing additional subword units to cover unseen character sequences.

The tokenization of (**??**) would then look as follows (depending on which implementation and version of the WordPiece tokenizer is used)

$$[the, man, travel, \#\#led, down., the, ro, \#\#ad.]$$

where we can see that words such as `the` and `down` are not further divided into tokens, and that suffixes such as `##led` are introduced, because these are rather unregular patterns, which however allow for better generalization.

2.2 Word Embeddings

Because in the last few decades, computing power has gradually increased, and popularity in using computational methods have surged, we will go over the different word-representations through which we allow computers to execute operations on.

Word-Embeddings In general, we want to find a mapping

$$w \mapsto (x_w, b_w) \in \mathcal{X}^{d+1} \tag{2.1}$$

where w is a token representation from a vocabulary $w \in V$ and where this token representation is transformed into a $d+1$-dimensional vector representation x_w, and a bias-term b_w that is specific to the token w. Whenever we talk about *word-vectors*, *(word)-embeddings*, or *feature-vectors*, we will refer to the image of the above map. For convenience and unless stated otherwise, we will assume that x_w absorbs the bias term b_w as an additional

vector-element. Also, for simplicity and unless otherwise stated, we will use the euclidean real-valued vector-space $\mathbb{R}^{d+1}$ to described the resulting embedding vectors. Please note, however, that the choice of the embedding space $\mathcal{X}$ is not fixed in general, and as such, work in other spaces such as hyperbolic spaces [25] have also been conducted.

Distance: Our goal is to build an embedding space where the distance between word-embeddings correspond to the relation between words (i.e. the semantics entailed by their tokens). Formally, we introduce the concept of *distance* $d : \mathcal{X} \times \mathcal{X} \mapsto [0, \inf)$, which should capture the relation between different elements. For a set of elements $x, y, z \in \mathcal{X}$, following properties must hold for a mapping to be a valid distance metric:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = 0 \iff x = y$ (identity, i.e. if two embedding vectors are identitical, they must capture the same underlying word instance)
3. $d(x, y) \leq d(x, z) + d(z, y)$ (triangle inequality)

One consequence of the above rules is that for words a, b, c, each having a word embedding x, y, z respectively, $d(x, y) < d(x, z) \iff$ word instance b is conceptually closer to word a than word c. For convenience, whenever I input a, b, c into the distance function d, the word-embeddings for the corresponding word-tokens shall be used. Also, please notice that I left out the notion of symmetry for distance measures.

Learning a distance A popular paradigm in machine learning is to maximize a certain probability distribution given some data $\mathbf{X}$. In the context of word vectors, we want to maximize the probability that w occurs in the context window of $w\prime$ through some parameters θ. Implicitly, this corresponds to minimizing the distance between w and w', while keeping the distance between w and all other words constant. We call $w\prime$ a *context word* for w.

$$p\left(w|w'\right) \tag{2.2}$$

and in the work we focus at, the probabilistic maximization problem corresponds to linear minimization problem

$$\forall w, w' : \qquad \max p\left(w|w'\right) \iff \min d(w|w') \tag{2.3}$$

Please note that we do not generalize to formalizing a dual-relationship between the two problem-statements, however.

Exploiting the distributional structure of language: Our goal is to learn distance between words by exploiting the distributional structure of a set of sentences, which make up a *corpus*. One of the early formalizations of representing the distributional structure of words through was expressed in [5], which argues that a sequential statistical model can be constructed to estimate this true posterior. In it's most naive form, this would imply that we can estimate the probability of a word $w^{(t)}$ after words $w^{(t-1)}, \ldots, w^{(1)}$ as

$$p(\mathbf{w}) = \prod_{t=1}^{T} p\left(w^{(t)}|w^{(t-1)}, \ldots, w^{(1)}\right) \tag{2.4}$$

where $\mathbf{w} = w^{(1)}, \ldots, w^{(T)}$ is the sequence of words, $p(\mathbf{w})$ the probability of this sentence occurring. This corresponds to a simple autoregressive model. One could for example use (hidden) markov models to model this relation, as we could fix the *context size* n (in this case, only looking at previously occurring words) and include the markov assumption of

$$p\left(w^{(t)}|w^{(t-1)}, \ldots, w^{(1)}\right) = p\left(w^{(t)}|w^{(t-1)}, \ldots, w^{(t-n+1)}\right) \tag{2.5}$$

Fixing the context window to n words for each computation (i.e. convolution), introduces the concept of *n-grams*. *n-grams* can also occur in relation to character, where n characters are fed in to some model at a certain timestep t.

Next to a context which only consists of the *previous* n words, one can also regard a context which includes both the previous *and subsequent* R words.

$$p(\mathbf{w}) = \prod_{t=1}^{T} \prod_{\triangle \in \mathcal{I}} p\left(w^{(t)} | w^{(t+\Delta)}\right) \tag{2.6}$$

whose probability we wish to estimate (and maximize if this is in the given training dataset), $\mathcal{I} = \{-R, \ldots, -1, 1, \ldots, R\}$ is the context window.

Loss Objective The above function is a probability estimate of the true underlying distribution. However, given that computational power is limited, we are usually interested in learning a parametrized model which captures this underlying distribution. One way to learn this parametrized model is to minimize a loss by backpropagating the gradients of the parameters. In that case, the loss function would look as follows.

$$\mathcal{L}(\theta; \mathbf{w}) = -\prod_{t=1}^{T} \prod_{\triangle \in \mathcal{I}} p_\theta\left(w^{(t)} | w^{(t+\Delta)}\right) \tag{2.7}$$

By minimizing the mean loss over all sentences in a big-enough corpus $\mathcal{C}$, we can arrive at a reliable estimator model with parameters θ, given the model is complex enough and generalizes well.

Specifically, if p_θ is a parametric probability density function, one can then optimize for the most optimal $\hat{\theta} = \text{argmin}_\theta \mathcal{L}(\theta; \mathbf{w})$ using a maximum likelihood estimation approach. One would usually minimize the log-loss as this allows for thee gradients to be more easily computed and avoids numerical issues.

Intuitively the above models flow the idea expressed by [30] very well, speaking that the meaning of a word is captured by its neighborhood. However, the above equations follow a *continuous bag of words CBOW* approach, which aims at predicting a target word w^t given some context words w'. However,

it is also possible to follow a *skip-gram SG* approach, where the aim is to predict context words w' given a target word w^t. Both methods result in the same type of model, which is merely trained differently. The skip-gram model is found to work better with rare words, whereas the CBOW model seems to have slightly better accuracy on words that are occuring more frequently.

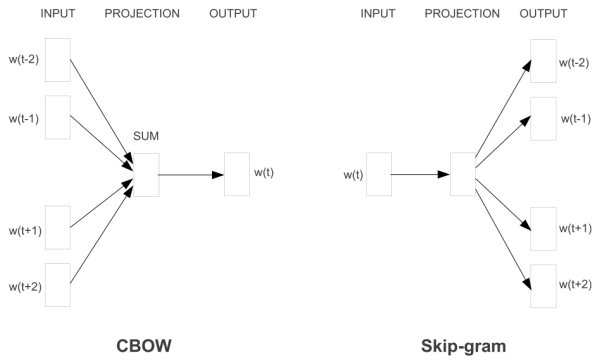


Figure 2.1: Figure taken from [51]. The CBOW architecture predicts the current word based on the context. The Skip-gram predicts surrounding words given the current word.

Using a skip-gram approach, (2.2) for example would be reformulated into

$$\mathcal{L}(\theta; \mathbf{w}) = \prod_{t=1}^{T} \prod_{\triangle \in \mathcal{I}} p_\theta \left(w^{(t+\Delta)} | w^{(t)} \right) \tag{2.8}$$

Although we have mentioned that the probability density distribution p is often modelled through a parametrized function (due to the sheer amount of data within corpora), we have not presented methods on how this parametrization can be achieved. In the following section, we will show how vectors for x_w and b_w can be found, and other ways to parametrize the probability density function p. We will not go into too much detail as to how these models are trained, as all of them can be trained by applying a gradient-based optimization on the loss term.

2.2.1 Static Word Embeddings

Here we will talk about word-embeddings where each word-token only has a single embedding x_w, i.e. there is a bijection between word-tokens and word-embeddings. Specifically, the mapping (2.1) is not a probabilistic function with a latent random factor, but rather a deterministic one. We will go over the basics of static word embeddings only, as the focus of this work lays in context embeddings which are presented in the subsequent section.

Basic Model

The first model we are going to look at is a most basic model which fulfills the properties of the distance metrics shown in (2.2).

Here, we can introduce a *log-bilinear* model where the log-probability is define as

$$\log p_\theta(w|w') = \langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + b_w + \text{ const.} \tag{2.9}$$

To arrive at the actual probability, we can exponentiate the log-probability as such

$$p_\theta\left(w|w'\right) = \frac{\exp\left[\langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + b_w\right]}{Z_\theta\left(w'\right)}$$

where $Z_\theta\left(w'\right) := \sum_{v \in \mathcal{V}} \exp\left[\langle \mathbf{x}_v, \mathbf{x}_{w'} \rangle + b_v\right]$ is a normalization constant such that the probability mass sums to 1, and the model parameters entail the word-embeddings $\theta = (x_w, b_w) \in \mathbb{R}^{d+1}$

Because we have $\theta = \forall w \in \mathcal{V} : \{x_w, b_w\}$, we can use a gradient-based loss-minimizing method which minimizes the cost function for θ, for a word w and all their possible context words $w' = w^{(t+\Delta)}$ within a context size R.

$$\mathcal{L}(\theta; \mathbf{w}) = \sum_{t=1}^{T} \sum_{\Delta \in \mathcal{I}} \log p_\theta\left(w^{(t+\Delta)}|w^{(t)}\right)$$

.

This model basic model comes with certain drawbacks. Amongst others the distance would be minimized if all word-embeddings would collapse onto a single point. There is no term that forces unrelated words to move away from each other, a property that we are interested in as unrelated words should form embedding vectors that are far from each other.

Word2Vec

One of the most prominent of word vector models is proposed by [51, 52]. Here, a neural network with a single embedding layer can be trained to transform one-hot-vectors $\in \{0,1\}^{|\mathcal{V}|}$ which represents a word w in vocabulary $\mathcal{V}$ into a latent vector representation $x_w \in \mathbf{R}^{d+1}$ using a neural network - which boils down to a linear embedding matrix - W both a continuous bag of word and also a continuous skip-gram approach. The skip-gram approach is preferred in practice.

Specifically, the loss-function that is optimized looks as follows

$$\mathcal{L}(\theta; \mathbf{w}) = \sum_{t=1}^{T} \sum_{\Delta \in \mathcal{I}} [\tag{2.10}$$

$$b_{w^{t+\Delta}} + \langle \mathbf{x}_{w^{(t+\Delta)}}, \mathbf{x}_{w^{(t)}} \rangle \tag{2.11}$$

$$- \log \sum_{v \in \mathcal{V}} \exp \left[\langle \mathbf{x}_v, \mathbf{x}_{w^{(t)}} \rangle + b_v \right] \tag{2.12}$$

As one can see, the loss function takes as input the bilinear loss from the basic model, and complements this by adding a regularizing term , such that random samples are not put next to each other. This idea is referred to as *negative sampling*.

Backpropagation is used to optimize over the networks weights. Also, highly frequent words are optionally subsampled and their frequency is re-weighted

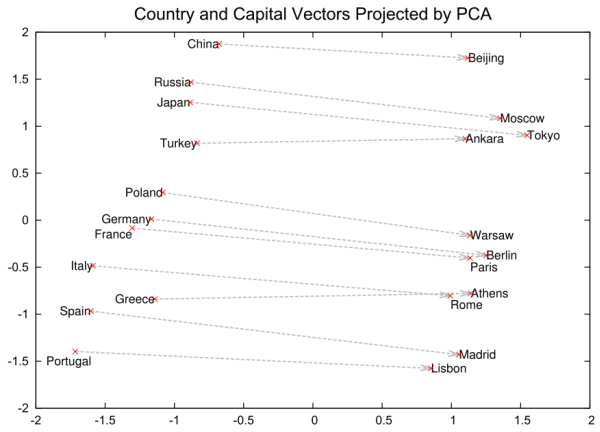


Figure 2.2: Figure taken from [52]. A 2-dimensional PCA projection of the 1000-dimensional skip-gram vectors of countries and their capital cities. The proposed model is able to automatically organize concepts and learn implicit relationships between them. No supervised information was provided about what a capital city means.

using the formula

$$P\left(w_i\right) = 1 - \sqrt{\frac{t}{f\left(w_i\right)}}$$

where $\dot{f()}$ is the function that returns the frequency of a word w_i.

GloVe

For the global vectors for word representation *GloVe* [58], the authors follow a more straight-forward matrix factorization approach.

First, a global co-occurence matrix is created $\mathbf{N} = (n_{ij}) \in \mathbb{N}^{|\mathcal{V}||\mathcal{C}|}$ where each entry n_{ij} is determined by the number of occurrences of word $w_i \in \mathcal{V}$ in context $w_j \in \mathcal{C}$. Given that the vocabulary size can exceed multiple thousand items, this practically results in a sparse matrix.

$$\mathcal{H}(\theta;\mathbf{N}) = \sum_{i,j} f\left(n_{ij}\right)\left(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\log \tilde{p}_\theta\left(w_i|w_j\right)}_{\text{model}}\right)^2 \tag{2.13}$$

$$= \sum_{i,j} f\left(n_{ij}\right)\left(\log n_{ij} - \langle x_i, y_j\rangle\right)^2 \tag{2.14}$$

$$\tag{2.15}$$

where $f(n_{ij})$ is the weighting function which assigns a weight for each entry in the co-occurence matrix based on the co-occurence of words w_i and w_j. In the second line, we also again use a bilinear probability density model $\tilde{p}_\theta\left(w_i|w_j\right) = \exp\left[\langle \mathbf{x}_i, \mathbf{y}_j\rangle + b_i + c_j\right]$. The constants b_i, c_j are left out and are assumed to be absorbed in the embedding vectors.

A popular choice for the weighting function is

$$f(n) = \min\left\{1, \left(\frac{n}{n_{\max}}\right)^\alpha\right\}$$

with $\alpha \in (0, 1]$. The motivation behind this is that frequent words do not receive a word-frequency which is considered too high (there is a cutoff at some point) and small counts are considered noise and slowly cancelled out.

Further extensions which include other basic units of include for example the fastText embeddings [8].

Gaussian Embeddings

Gaussian embeddings is the only type of distribution representation that we cover here. [6] already used distributional representations to embed concepts from natural language processing into embeddings.

[81] considers creating a Gaussian representation for each word token, resulting in Gaussian word embeddings. This can be interpreted as a probabilistic and continuous extension to the otherwise common discrete point vectors. Each word is represented by a Gaussian distribution in high-dimensional

space, with the aim to better capture uncertainty when doing algebraic operations on the word-embeddings, resulting in a model which expresses the relation between words and their representations better. Because the probabilistic framework is more flexible, it can also express asymmetric relations more naturally than dot-products or cosine similarities do. Specifically, the newly emerging embedding for word w can be modelled by the tuple

$$\theta_w = (\mu_w, \Sigma_w)$$

and the probability density function of the word embedding x_w is expressed by

$$p(x; w) = \mathcal{N}(x; \mu_w, \Sigma_w) \tag{2.16}$$

where x is a point in the embedding space.

Given that the probability density function is given by a gaussian distribution, the loss functions which is minimized is adapted accordingly:

$$L_m(w, c_p, c_n) = -min(0, m - E_\theta(w, c_p) + E_\theta(w, c_n) \tag{2.17}$$

where E is the energy function which calculates the distance measure between word w, a positive context word c_p (a word that co-occurs with the word w, and c_n), and a negative-sampled context word c_n (a word that does not co-occur with the word w, usually randomly sampled from the set of all words in the vocabulary $\mathcal{V}$).

[81] proposes two different ways to compute the energy function, using a symmetric similarity, and an asymmetric similarity measure.

The energy function can be **symmetric**. In this case, the authors use an inner product of two gaussian distributions defined as:

$$E_\theta(w, c) = \int_{x \in \mathcal{R}^d} f(x)g(x)dx \tag{2.18}$$

$$= \int_{x \in \mathcal{R}^d} \mathcal{N}(x; \mu_w, \Sigma_w)\mathcal{N}(x; \mu_c, \Sigma_c)dx \tag{2.19}$$

$$= \mathcal{N}(0; \mu_w - \mu_c, \Sigma_w + \Sigma_c) \tag{2.20}$$

which results in a nice closed form representation.

For numerical feasibility and easy of differentiation, usually the log-loss $\log E_\theta(w, c_p, c_n)$ is optimized the for a given dataset with $w \in \mathcal{V}, c_p, c_n \in \mathcal{C}$ (usually $\mathcal{V} = \mathcal{C}$, unless the set of context words is different.

The energy function can also be **asymmetric**. In this case, the authors refer to the KL-divergence, resulting in the following energy function minimized.

$$-E(w_i, c_j) = D_{KL}(c_j || w_i) \tag{2.21}$$

$$= \int_{x \in \mathcal{R}^d} \mathcal{N}(x; \mu_{w_i}, \Sigma_{w_i}) \log \frac{\mathcal{N}(x; \mu_{c_j}, \Sigma_{c_j})}{\mathcal{N}(x; \mu_{w_i}, \Sigma_{w_i})} dx \tag{2.22}$$

$$= \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_i^{-1}\Sigma_j\right) + (\mu_i - \mu_j)^\top \Sigma_i^{-1}(\mu_i - \mu_j) - d - \log \frac{\det(\Sigma_j)}{\det(\Sigma_i)}\right) \tag{2.23}$$

Because of the loss function, this can entail information such as "y entails x", without necessarily implying "x entails y" (a property that symmetric loss functions cannot model).

Assuming training was conducted by one of the two presented models, one can use the resulting model parameters for two words x_{w_1}, x_{w_2} to calculate the uncertainty - which intuitively models the similarity between the two words w_1 and w_2. This is analogous to calculating the dot product between the two distributions $P(z = x^T y)$ through the close-form formulation of

$$\mu_z = \mu_x^T \mu_y \tag{2.24}$$

$$\Sigma_z = \mu_x^T \Sigma_x \mu_x + \mu_y^T \Sigma_y \mu_y + \operatorname{tr}(\Sigma_x \Sigma_y) \tag{2.25}$$

We then get an uncertainty bound, where c denotes the number of standard deviations away from the mean. This uncertainty bound can be interpreted as a hard cutoff to where one concept end, allowing to make calculations between embeddings.

$$\mu_x^\top \mu_y \pm c\sqrt{\mu_x^\top \Sigma_x \mu_x + \mu_y^\top \Sigma_y \mu_y + \operatorname{tr}(\Sigma_x \Sigma_y)} \tag{2.26}$$

We can learn the parameters Σ and μ for each of these embeddings using a simple gradient-based approach, where we set constraints on

$$\forall i \quad \|\mu_i\|_2 \leq C \tag{2.27}$$

$$\forall i \quad mI < \Sigma_i < MI \tag{2.28}$$

These constraints can be used by using a laplacian regularizer in the loss function which is then optimized using a gradient based approach. The method shows competitive scores to the Skip-Gram model, although usually only with minor improvements depending on the benchmark-dataset.

2.2.2 Context Embeddings

Context embeddings follow the idea of the static word embeddings. In contrast to static word embeddings, while context embeddings also produce embeddings for words, the produced embedding vector vary depending on the context $\mathbf{c}$ that a word w carries.

Specifically, the mapping for word embeddings (2.1) is adapted in such a way to also take as input the previous context words $c_{\text{previous}} = w^{(t)}, \ldots, w^{(t-d+1)}$, and subsequent context word $c_{\text{subsequent}} = w^{(t-d-1)}, \ldots, w^{(1)}$ for a given word of interest w.

$$(c_{\text{previous}}, w, c_{\text{subsequent}}) \mapsto (x_w) \in \mathcal{X}^{d+1} \tag{2.29}$$

As an example, while static word embeddings would produce the same vector x_1 for all occurrences of the word *bank*, the context embedding would produce different embeddings x_1 and x_2 respectively for the two sentences:

```
I withdrew some cash at the bank's ATM.
I walked down the sea bank.
```

Up to some exceptions, any deviation of the context will result in a deviation of the resulting context embedding.

The underlying model is a many-to-one sequence estimator, in general modelling the probability of

$$p(w^{(t-d)}; c_{\text{previous}}, w^{t-d}, c_{\text{subsequent}}) = p(w^{(t-d)}; w^{(t)}, \ldots, w^{(t-d+1)}, w^{(t-d)}, w^{(t-d-1)}, \ldots, w^{(1)}) \tag{2.30}$$

instead of independently querying the probability $p(w^{(t-d)})$ without any context words.

There are different ways to achieve modelling this probability, from simple markov models, to LSTMs to transformers. We will go over the most prominent models of the past few years, discussing their underlying mechanism.

ELMo

The simplest idea of context embeddings, which take into account more than just a single word, but all tokens from the context $\mathbf{c} = (c_{\text{previous}}, c_{\text{subsequent}})$is the *Embeddings from Language Models* (ELMo) model proposed by [60].

The most basic architecture of ELMo is an extension of the Long Short-Term Memory *LSTM* architecture. LSTMs are sequential recurrent neural networks [32]. LSTMs are an extension of the recurrent neural network *RNN* [68], but introduces a mechanism to solve the problem of error in the back-propagating gradient, and a "sotrage" mechanism which allows part of the RNN cell to be non-changed throughout backpropagation update mechanisms.

The internal cell of the LSTM includes the following mechanisms

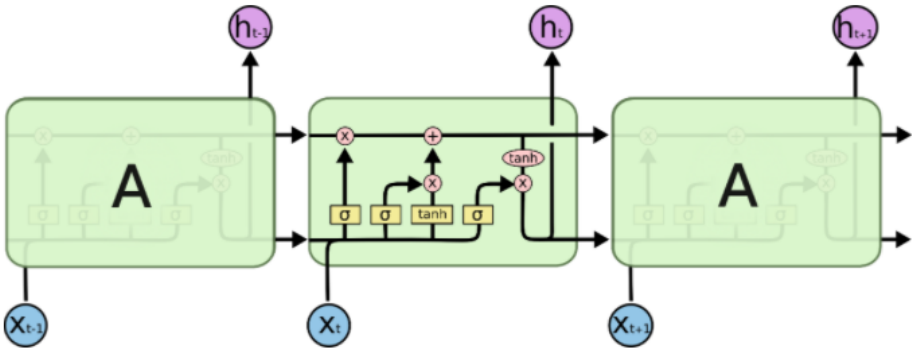


Figure 2.3: Figure taken from [2]. The internals of the LSTM cell, and the recurrent flow which is repeated for three consecutive timesteps $t-1, t, t+1$. The LSTM produces an hidden representation at every steep h_{t-1}, h_t, h_{t+1} given some inputs x_{t-1}, x_t, x_{t+1}.

Notice that the flow has a direction, and that in this case, the flow moves towards a positive direction of the sequence (i.e. from $t-1=0$ to $t+1=2$ for example).

Specifically, LSTMs [32] are good at estimating the probability of word $w^{(t)}$ occurring given a previous history of words $w^{(t-1)}, \dots w^{(1)}$. In practice, the LSTM is a popular choice for sequence modeling tasks, as it is able to capture very well the following probability distribution.

$$p\left(w_1, w_2, \dots, w_T\right) = \prod_{k=1}^{T} p\left(w_k | w_{k-1}, \dots, w_2, w_1\right) \tag{2.31}$$

When we apply two LSTM layers, one that reads the sequence in the positive direction - as decpicted in (**??**) - and another one that reads the sequence in a negative direction (i.e. reads in the sequence $[\dots, x_{t+1}, x_t, x_{t-1}, \dots]$ in this particular direction), the resulting architecture is called bidirectional LSTM. In its underlying architecture, ELMo is using bidirectional Long Short-Term Memory architectures *bi-LSTM*. Specifically, a possible loss function results in

$$\sum_{k=1}^{T} \Big(\log p \left(w_k | w_{k-1}, \dots, w_1; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s \right) \tag{2.32}$$

$$+ \log p \left(w_k | w_{k+1}, \dots, w_T; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s \right) \Big) \tag{2.33}$$

where p is an LSTM model parametrized by it's weights θ, including softax heads to arrive at a valid probability distribution amongst words in the vocabulary $\mathcal{V}$.

Given that the LSTM is a sequential model, it outputs as many hidden representations as there are timesteps T that are given as input to the model. Each LSTM layer produces one set of such hidden representations. ELMo concatenates the hidden representations for both of the bidireectional LSTM layers to arrive at a single hidden representation. Within ELMo, each one of these hidden representations are interpreted as one of the context embeddings, which can then be used for downstream tasks. As such, we retrieve an embedding vector $x_{w^i} = [h_{w^i}^{\text{backward}}, h_{w^i}^{\text{forward}}]$ for word w^i Each context em-

beddings outputs a vector representing x_k for the word token $w^{(k)}$, given its context $w_1, \ldots, w_{k-1}w_{k+1}, \ldots, w_N$.

Formally, one would pre-train this model on a huge corpus in an unsupervised fashion. This pre-trained language model would then be extended by another linear layer, for example, to be adapted to a task at hand which has less data. This downstream task could be a named entity recognition, part of speech tagging etc. The gradients of the ELMo biLSTM weights are backpropagated and updated for the final fine-tuning process.

The Transformer Architecture

Although ELMo is able to crerate context embeddings, the performance is limited to LSTMs which can only capture timestep-directed sequences.

Specifically, the LSTM used within ELMo is a sequential timestep model which covers a storage mechanism within the weights it uses, resulting in a 1-step markovian probability-assumption

$$\begin{aligned} p(w_1, \ldots, w_T) = p(w_T|w_{T-1}; h_{T-1}, \theta) \\ \ldots p(w_t|w_{t-1}; h_{t-1}, \theta) p(w_{t-2}|w_{t-2}; h_{t-2}, \theta) \\ \ldots p(w_0; \mathbf{0}, \theta) \end{aligned} \tag{2.34}$$

where h_i is the forward-propagation vector outputted by the LSTM cell at timestep i, which are supposed to capture some concept of storage together with the models weights θ. Although intuitively h_{T-1} is supposed to capture information from many timesteps back, there is no explicit information flow between the LSTM cell at timestep $t+1$ and $t-1$ (i.e. any two cells which are longer than 2 steps away).

This is something where the transformer architecture comes in. The first transformer architecture for sequence modelling of text was introduced in [80]. The transformer architecture introduces the concept of attention for

modern language models. It takes the full sample sentence including all timesteps, and calculates a correlation between the hidden representations of each timestep. Internally, it calculates hidden representations, uses the attention mechanism to rank similarity between the hidden representations, and then outputs a sample predicted sentence using the output probabilities.

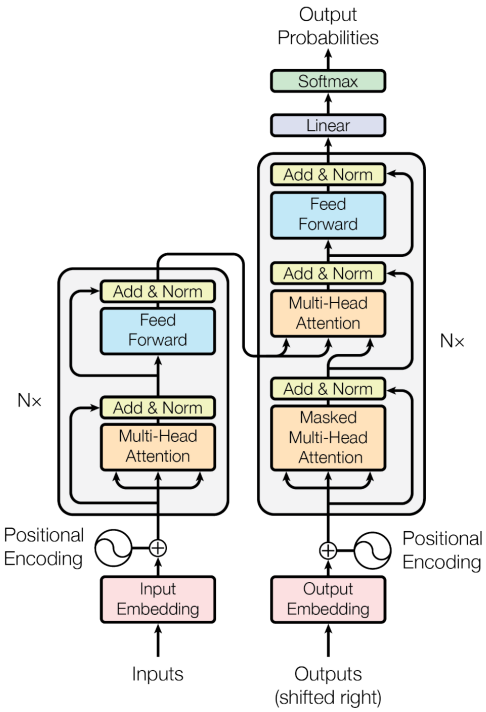


Figure 2.4: Figure taken from [80]. The transformer module architecture. The transformer encapsulates multiple attention layers.

In the context of sentences, it is able to take as input a full sequence of words. For each word, a vector representation is computed using an embedding layer. For the input sequence, this results in a set of vectors V which represents the values, and K which represents the keys. One can then take set of query vectors Q and calculate the inner product $\langle K, Q \rangle$. One then applies a softmax this set of dot-products to arrive at a notion of which two vector are similar to each other. In the case of self-attention, the query vectors Q are equivalent to K. This allows for the model to be more attentive between two elements,

and allows the model to focus exploit relationships between different tokens within a token-sequence better.

This has implications for the complexity of the model. Instead of calculating the raw probability of (**??**), and even using a markovian assumption because computation power is expensive, the transformer architecture implicitly calculates the joint probability of an entire sequence of words, thus decreasing the step of computations through which information flows.

We will not go into further detail of how and why the transformer architecture works better than the LSTM to keep focus. All of the below presented models use the transformer architecture as a building block instead of the LSTM used in ELMo. The following architectures are all based on the transformer module.

BERT

As introduced in [19], the Bidirectional Encoder Representations from Transformers model (BERT) combines the above concepts of forward and backwards sequential models with attention, as marked in the transformer architecture above.

Although there are different version of BERT published, the base model of BERT consists of 12 attention layers, which output a 768-dimensional hidden representation of the input sequence, which on the final layer is outputted to work with any target task.

The difference between BERT and ELMo (and GPT which is introduced in the next section) is depicted in the following figure.

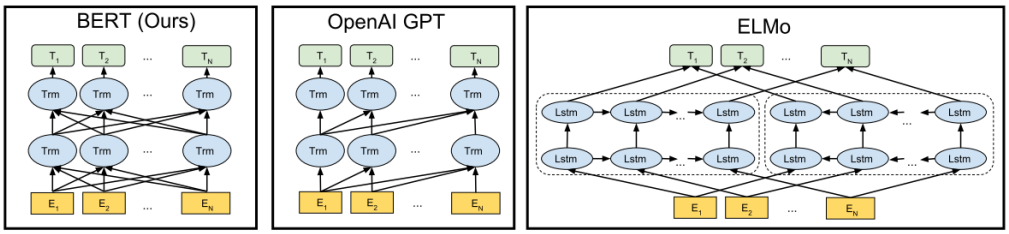


Figure 2.5: Figure taken from [19]. BERT uses a bidirectional transformer, which is not limited to reading in all the input from only left to right or right to left. OpenAI GPT (next section) uses a left-to-right Transformer, while ELMo is using a bidirectional LSTM which naturally captures a direction.

As stated in the transformed architecture, the main advantage in BERT is that the information flow between any two tokens is shortened because there is not a intermediate latent representation laying between the flow of hidden representations of any two timesteps. Specifically, the attention mechanism allows for a direct comparison of hidden states h_t and h_k with $(|t-k|>1)$, whereas the paradigm of the LSTM would be to capture any relation between the two hidden representations in the weights of the LSTM, as well as the ongoing hidden representation which is passed on at every timestep (i.e. h_{t-1} would have to contain the information about h_k where $k < t-1$). by using multiple attention layers of on top each other, BERT is a deep model, allowing for more complex patterns to be trained.

BERT is trained using two phases. The first phase is BERT **pre-training**, where the model is trained using a masked language model approach. Here, about 15% of words in a sample are replaced with the "mask" token, and the goal of the model is to predict the word which was replaced by the `[MASK]` token.

```
[CLS] The man went to [MASK] store. [SEP]
```

Figure 2.6: Example from [19]. An input sentence which where 15% of the tokens are replaced with the [MASK] token. During pre-training, the weights of the BERT model are optimized in such a way to predict the true underlying words. The word to be predicted is *the*

The second pre-training task which is trained analogously to the masked language model prediction is the **next sentence prediction** task. Here, the language model is given two sentences, and is supposed to predict whether the second sentence is the continuation of the first one. 50 % of the training set here consists of randomly sampled sentences, and the other 50 % of the training set consists of the actual next sentence for a corpus.

```
[CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
```

Figure 2.7: Example from [19]. Two input sequences which where 15% of the tokens are replaced with the [MASK] token. During pre-training, the weights of the BERT model are optimized in such a way to predict the true underlying words. In this case, the second sentence is a continuation of the first one, and thus the label would be *isNext*.

BERT is also a pre-trained model which is trained on the scraped english wikipedia corpus.

BERT introduces itself as a pre-trained language model. Specifically, it shall be used to fine-tune on specific downstream tasks.

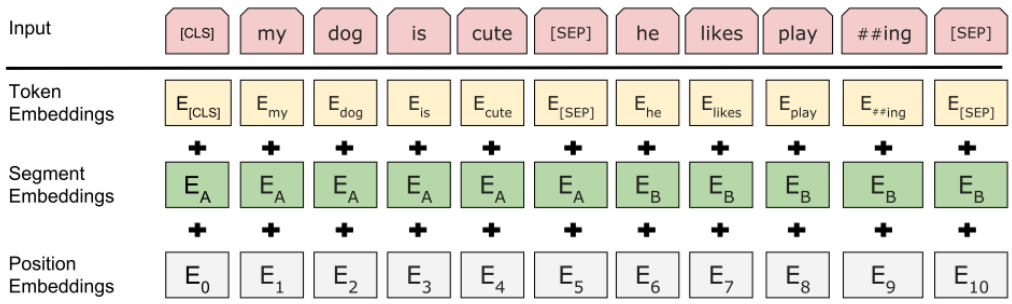


Figure 2.8: Figure taken from [19]. BERT takes as input multiple tokens, including a position embedding, the token embedding and the segment embedding. This allows BERT to distinguish between the location of the word within a sentence, and which word token was provided and which sentence the word token is a part of.

The BERT model takes a sequence of size (up to) 512 elements as input.

Multiple versions of BERT are provided, including $BERT_{LARGE}$ which includes , and $BERT_{BASE}$ which includes

Ever since BERT came out, a wide variety of similar models came out that mimick the main logic of BERT, and improve upon it, such as DistillBERT [69].

Numerous works have been published on making BERT representations more compact, for it to be used for on-device text-processing applications, such as more compact sentence-representations [72].

GPT and GPT-2

First proposed in [62] and further extended in [63].

[62] introduces the concept of *generative pre-training* and discriminative fine-tuning *generative pre-training*.

Unsupervised pre-training consists of an unsupervised corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$ and use the common objective of

$$L(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{u-1}; \theta) \tag{2.35}$$

where k describes a context window, and the conditional probability P is modeled using a neural network wth parameters θ. The transformer decoder [44] is used which is a variant of the transformer [80], which introduces a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers to produce an output distribution over target tokens as such:

$$h_0 = UW_e + W_p \tag{2.36}$$

$$h_l = \text{transformer\_block}(h_{l-1}) \forall i \in [1, n] \tag{2.37}$$

$$P(u) = \text{softmax}(h_n {W_e}^T) \tag{2.38}$$

where $U = (u_{-k}, \ldots, u_{-1})$ is the context vector of tokens, n is the number of layers, W_e is the token embedding matrix, and W_p is the position embedding matrix.

Supervised fine-tuning consists of leveraging a labeled dataset $\mathcal{C}$, where each instance consists of a sequence of input tokens, $x^1, \ldots, x^m$ along with a label y. The inputs are passed through the pre-trained model to obtain the final transformer block's activation ${h_l}^m$ which is then fed into an added linear output layer with parameters W_y to predict y.

$$P\left(y|x^1, \ldots, x^m\right) = \text{softmax}\left(h_l^m W_y\right) \tag{2.39}$$

which then gives the following objective to maximize:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P\left(y|x^1, \ldots, x^m\right) \tag{2.40}$$

As suggested in [66] and [59], an auxiliary language modeling loss (unsupervised training) is added which accelerates convergence and improves generalization

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C}) \tag{2.41}$$

with a regularization parameter λ.

GPT-2 improves on the first version of GPT. Firs, the training dataset uses a crawled text of all outbound texts from Reddit resulting in the WebText corpus. The crawling als prioritizes documents by quality.

The second major modification is in input representations. GPT-2 uses a Byte Pair Encoding (BPE) scheme, [70], which operates on unicode code points as the most fundamentla unit of language. The BPE tokenization is modified by avoiding merging across character categories for any byte

sequence (i.e. for words that occur in many variations, such as `dog.`, `dog!`, and `dog?`) This is similar to the character level rnns (cite jason).

The base model of GPT is modified by moving the layer normalization [4] layer to the input of each sub-block and an additional layer normalization after the final self-attention block. Also, initialization was modified to account for the accumulation of the residual path with model depth. Also, vocabulary is expanded to 50257 tokens.

2.3 GLUE benchmark dataset

The GLUE benchmark dataset was first introduced by [83]. Although additional work has been done through the introduction of SuperGLUE [82] to train for an even bigger variety of language tasks, after most NLU models started surpassing non-expert human performance on the initially-proposed GLUE-tasks, we will only use at the standard GLUE benchmarking dataset, as our base model will be BERT, which does not perform on superhuman leves with the GLUE benchmark yet.

Corpus	**|Train|**	**|Test|**	**Task**	**Domain**
CoLA	8.5k	1k	acceptability	misc
SST-2	67k	1.8k	sentiment	movie reviews
MRPC	3.7k	1.7k	paraphrase	news
STS-B	7k	1.4k	sentence similarity	misc.
QQP	364k	391k	paraphrase	social QA questions
MNLI	393k	20k	NLI	misc
QNLI	105k	5.4k	QA/NLI	Wikipedia
RTE	2.5k	3k	NLI	news, Wikipedia
WNLI	634	146	coreference/NLI	fiction books

Figure 2.9: Hello

We use an accuracy score for all benchmarking datasets. Because MRPC

and QQP have a dominant class, the F1-score is used here. Finally, STS-B is judged using a pearson and spearman correlation due to a ranking being produced, and CoLa is assessed using the Matthews correlation.

Single Sentence Tasks

The Corpus of Linguistic Acceptability **CoLA** [86] consists of englihs sentences, where the task of the model is to predict whether or not the sentence is a valid english sentence. The Matthews correlation evaluates the model from a score of -1 to 1, where 0 corresponds to uniformly random guessing. The test set includes out of domain sentences.

```
label:1
The professor talked us into a stupor.

label:0
The professor talked us.
```

The Stanford Sentiment Treebank **SST-2** [73] consists of sentences from moview reviews and human annotations of their sentiment. This is a binary classificaiton (positive / negative) task.

```
label:1
comes from the brave , uninhibited performances

label:0
excruciatingly unfunny and pitifully unromantic
```

Similarity and paraphrase tasks

The Microsoft Research Paraphrase Corpus **MRPC** [20] is a corpus of sentence pairs automatically extracted from online news sources, with human annotations whether the sentences in the pair are semantically equivalent. Here, an F1-score is taken, because the class-sets are imbalanced.

```
quality:1
Prosecutors filed a motion informing Lee they intend to seek the death penalt
```

```
He added that prosecutors will seek the death penalty.

quality:0
A former teammate , Carlton Dotson , has been charged with the murder.

His body was found July 25 , and former teammate Carlton Dotson has been charged in
```

The Quora Question Pairs **QQP** [37] dataset is a collection of question and answer pairs from the website Quora. The task it to check whether a pair of questions are semantically equivalent. Again, the F1-score is taken as a measure because the class-sets are unbalanced.

```
My Galaxy ace is hang?
Why are the people on Staten Island are racist?
0

Where can I learn to invest in stocks?
How can I learn more about stocks?
1
```

The Semantic Textual Similarity Benchmark **STS-B** [12] is a corpus of pairs of news headlines, video and image captions. Each pair is annotated with a similarity score from 1 to 5, and the task is to predict these scores. The evaluation is done using Pearson and Spearman correlation, as the determining factor is the relative ranking amongst scores.

```
The man is riding a horse.
A man is riding on a horse.
5.000

A man is playing a guitar.
A girl is playing a guitar.
2.800
```

Inference Tasks

The Multi-Genre Natural Language Inference Corpus **MNLI** [87] [9] is a crowd-sourced collection of sentence pairs with extual entailment annota-

tions. For each premise and hypothesis sentence, the task is to predict whether the premise entails the hypothesis, contradit the hypothesis, or neither. Thus, this is a 3-class classification problem. The matched MNLI version tests on data which is in the same domain as the training set, while the mis-matched MLNI tests on a training set which is cross-domain. The MNLI contains parse-trees, which is the reason we do not display these here.

The Stanford Question Answering Dataset is a question-answering dataset [64]. The authors of GLUE use this dataset, to create the Question answering NLI **QNLI**, by removing the requirements that the model selects the exact answer. On top of that, they also remove the simplifying assumption that the answer is always present in the input and that lexical overlap is a reliable cue. The task is to determine whether the context sentence contains the answer to the question.

```
Who did the children work beside?
In many cases, men worked from home.
not_entailment
```

```
How many alumni does Olin Business School have worldwide?
Olin has a network of more than 16,000 alumni worldwide.
entailment
```

The Recognizing Textual Entailment **RTE** dataset comes from a series of annual textual entailment challenges. Data is combined from RTE1, RTE2, RTE3 and RTE5 [18] [29] [7] [27]. The classes to be predicted by the model include *neutral*, *contradiction* and *no entailment*.

```
Oil prices fall back as Yukos oil threat lifted
Oil prices rise.
not_entailment
```

```
Money raised from the sale will go into a trust for Hepburn's family.
Proceeds go to Hepburn's family.
entailment
```

The Winograd NLI **WNLI** dataset uses the original Winograd Schema Chal-

lenge dataset [43], which is a reading comprehension task where the reader must read a sentence with a pronoun and select the referent of that pronoun from a list of choices. Sentence pairs are constructed by replacing the ambiguous pronoun with each possible referent. The task is to predict if the sentence with the pronoun substituted is entailed by the original sentence.

Some example sentences include

```
Bob was playing cards with Adam and was way ahead.
If Adam hadn't had a sudden run of good luck, he would have won.
Adam would have won.
label:0
```

```
Mark told Pete many lies about himself, which Pete included in his book.
He should have been more truthful.
Mark should have been more truthful.
label:1
```

2.4 WordNet

The online lexical database WordNet was original introduced in [53]. WordNet is a semantic reference system similar to a thesaurus whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs and adjectives are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. WordNet 1 contains of 95,600 different word forms (51,500 simple words and 44,100 collocations), and include a total of 70,100 word meanings, or sets of synonyms. Compared to a standard dictionary, WordNet divides the lexicon into five categories, namely nouns, verbs, adjectives, adverbs and function words. The authors argue that the rest of language is probably stored separately as part of the syntactic component of language.

The most ambitious feature of WordNet, however, is to attempt to organize lexical information in terms of word meanings.

The problem with alphabetical thesaurus is redundant entries. If Word w_a

and word w_b are synonyms, the pair should be entered twice. The problem with a topical thesaurus is that two look-ups are required, first on an alphabetical list and again in the thesaurus proper.

Lexical matrix: Word are often referred to both the utterance and to its associated concept. As such, [53] specify the difference between the two concepts as "word form", which refers to the physical utterance or inscription, and "word meaning", which refers to the lexicalized concept that a form can be used to express.

The following table captures the concept of a lexical matrix, which encodes word-forms (columns), word-meanings (rows), and the existence of the expression of the word-meanings through the corresponding word-form (cell). If multiple entries exist for a single column, then the single word-form encode mulitple meanings, implying that the word-form is polysemous (it encodes multiple word-forms). If multiple entries exist for a single row, the two words express the same underlying concept, and thus the two words-forms are synonoms.

Word			Word	Forms	
Meanings	F_1	F_2	F_3	$\ldots$	F_n
M_1	$E_{1,1}$	$E_{1,2}$			
M_2		$E_{2,2}$			
M_3			$E_{3,3}$		
$\vdots$				$\ddots$	
M_m					$E_{m,n}$

Figure 2.10: Figure taken from [53]. Word-forms F_1, and F_2 are synonyms of each other, as they share one word meaning M_1. Word-form F_2, as it entails more than one meaning, namely M_1 and M_2.

WordNet also includes synonym sets referred to as *synsets*, which are a collection of word-forms that together determine a single meaning. Thus, a meaning is represented as a synset. WordNet is organized by semantic relations. Thus, WordNet includes a set of semantic relations. WordNet defines

synonyms as a set of words, where one word is interchangable for another. This implies that in terms of substituatbiliy, WordNet clusters words into nouns, verbs, adjectives and adverbs, as the part of speech must be valid at the position of the sentence.

Although the authors mention that synonyms should be best thought of a continuum along which similarity of meaning can be graded, the authors decide to determine similarity in terms of a binary event, something which is either present, or not present.

This is mainly handcrafted by linguistics over the last 2 decades. Meanings have different levels of detail. The two tables depicted below show examples of how WordNet 3.1 introduces different semantic classes for the words `bank` and `was`

Part of Speech	Definition
noun	(sloping land (especially the slope beside a body of water)) "they pulled the canoe up on the bank"; "he sat on the bank of the river and watched the currents"
noun	depository financial institution, bank, banking concern, banking company (a financial institution that accepts deposits and channels the money into lending activities) "he cashed a check at the bank"; "that bank holds the mortgage on my home"
noun	(an arrangement of similar objects in a row or in tiers) "he operated a bank of switches"
verb	(tip laterally) "the pilot had to bank the aircraft"
verb	(do business with a bank or keep an account at a bank) "Where do you bank in this town?"

Figure 2.11: Example output for WordNet 3.1 noun propositions for the word "bank". In total, 18 different concepts are recorded.

Part of Speech	Definition
noun	Washington, Evergreen State, WA, Wash. (a state in northwestern United States on the Pacific)
verb	(have the quality of being; (copula, used with an adjective or a predicate noun)) "John is rich"; "This is not a good answer" bank"; "that bank holds the mortgage on my home"
verb	(an arrangement of similar objects in a row or in tiers) "he operated a bank of switches"
verb	(form or compose) "This money is my only income"; "The stone wall was the backdrop for the performance"; "These constitute my entire belonging"; "The children made up the chorus"; "This sum represents my entire income for a year"; "These few men comprise his entire army"
verb	(work in a specific place, with a specific subject, or in a specific function) "He is a herpetologist"; "She is our resident philosopher"

Figure 2.12: Example output for WordNet 3.1 noun propositions for the word "was". In total, 18 different concepts are recorded.

In this work, we will often use WordNet as this is one of the few resources that quantify language semantics.

2.4.1 SemCor dataset

The SemCor 3.0 corpus is a collection of the Brown corpora [22], where each noun, adjective and verb is tagged with the WordNet 3.0 senses. It was part of the early WordNet project, initially introduced in [54].

Some example snippets look as follows, where SemCor includes sentences, and each sentence is annotated by WordNet senses.

```
A Texas halfback who does n't even know the team 's plays,
 Eldon_Moritz, ranks fourth in Soutwest_conference scoring after three games.
```

The corresponding part-of-speech labels and wordnet sense ids are

```
DT, NN, NN, WP, VBZ, RB, RB, VB, DT, NN,
POS, NN, NNP, VB, JJ, IN, NNP, VP, IN, JJ, NN
```

```
None, 1, 1, None, 0, 1, 7, None, 1,
None, 3, 1, 1, 1, None, 1, 1, 1, 1
```

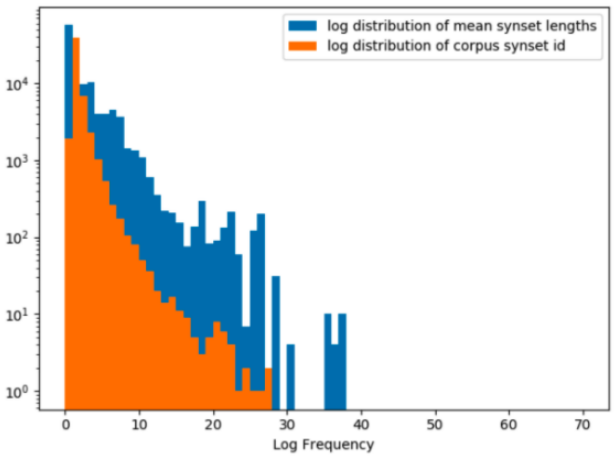


Figure 2.13: Shows that the SemCor data is biased. Words with a low WordNet sense index (i.e. close to 0) occur much more often than words that have a high WordNet sense index (i.e. above 5). The x-axis shows the WordNet sense index for a chosen word, while the y-axis shows the log-frequency within SemCor. This is a cumulative plot over all words with WordNet senses within SemCor 3.0. The skew could be a natural effect of how word lower WordNet indecies are assigned to more commonly used words.

2.5 Metric Learning and Disentanglement

Althogh difference metric spaces can be considered, we will be refering to Euclidean spaces for simplicity and unless otherwise stated.

Metric learning is the concept of learning a new space in which distances between data samples

Distance metric learning approaches the problem of learning a similarity metric.

We introduce two sets of data-sample pairs, S and D:

$$S = \{(x_i, x_j) : x_i and x_j should be similar\} \tag{2.42}$$

$$D = \{(x_i, x_j) : x_i and x_j should be dissimilar\} \tag{2.43}$$

As is also mentioned in [55], the proximity relation can also be captured through *relation triplets*

$$R = \{(x_i, x_j, j_l) : x_i is more similar to x_j than x_l\} \tag{2.44}$$

$$\tag{2.45}$$

The general notion of a distance in the euclidean space can be defined as

$$d_E(\boldsymbol{x_i}, \boldsymbol{x_j}) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top (\boldsymbol{x}_i - \boldsymbol{x}_j)} \tag{2.46}$$

or equivalents as the l_2-norm

$$d_E(\boldsymbol{x}_i, \boldsymbol{x}_j) = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \tag{2.47}$$

Most distance learning techniques propose different ways of learning a Mahalanobis distance [48]

$$d_M(\boldsymbol{x}_i - \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{A}^{-1} (\boldsymbol{x}_i - \boldsymbol{x}_j)} \tag{2.48}$$

Because calculating d^2 is computationally simpler, and d^2 maintains most mathematical properties as calculating d, often d^2 is used for downstream calculations (i.e. optimization, distance measure), rather than d.

In the case of the Mahalanobis distance, A is a linear operator such as a matrix. However, A can also be generalized to a nonlinear operator by using

the following formulation of the distance metric

$$d_M^2(x_i - x_j) = (x_i - x_j)^\top A^{-1}(x_i - x_j) \tag{2.49}$$
$$d_M^2(x_i - x_j) = (L(x_i - x_j))^\top (L(x_i - x_j)) \tag{2.50}$$
$$d_M(x_i - x_j) = \|L(x_i - x_j)\|_2 \tag{2.51}$$
$$d_M(x_i - x_j) = \|Lx_i - Lx_j\|_2 \tag{2.52}$$

where $A = L^T L$ and L can now be any function $L \in \mathbf{R}^n -> \mathbf{R}^d$. Whenever, $d < n$, the data is projected to a lower dimensional space d, which results in dimensionality reduction, which however breaks the convexity property if L is not invertible. In comparison to a dimensionality reduction such as PCA, it has been obsereved that learning a similarity measure can sometimes be more effective [13].

Specifically, the data is projected into another The authors in [55] argue that due to this property of measuring similarity between pairs, the metric learning models are able to generalize better across classes.

Non-Deep Metric Learning

Although we focus on deep metric learning during our experiments, due to the additional flexibility that deep neural networks provide, a good understanding of the underlying mechanisms is useful.

[77] provides a good introduction into the literature of metric learning algorithms.

We first define by a proper definition of a distance

Definition 1 *Let X be a non empty set. A distance or metric over X is a map $d : X \times X -> \mathbb{R}$ that satisfies the following properties:*

1. *Coincidence: $d(x, y) = 0 \iff x = y$, for all $x, y \in X$*
2. *Symmetry: $d(x, y) = d(y, x)$ for all $x, y \in X$*

3. *Triangle inequality:* $d(x, z) \leq d(x, y) + d(y, z)$ *for all* $x, y \in X$

The ordered pair (X, d) *is called a metric space. We will be assuming a euclidean metric space unless otherwise specified.*

Because the coincidence property is not always important to us, but because word-embeddings are more interested in measuring relative distances, we will also consider mappings konwn as *pseudoinstances* which relax the coincidence property to merely fulfill $d(x, x) = 0$. In the d dimensional euclidean space, the set of *Mahalanobis distances*, which is parametrized by positive semidefinite matrices are useful.

Using above similarity, dissimilarity and relation triplets, an optimal distance d from a set of distances $\mathcal{D}$ can be found

$$\min_{d \in \mathcal{D}} l(d, S, D, R) \tag{2.53}$$

where l is one of the loss metrics shown below.

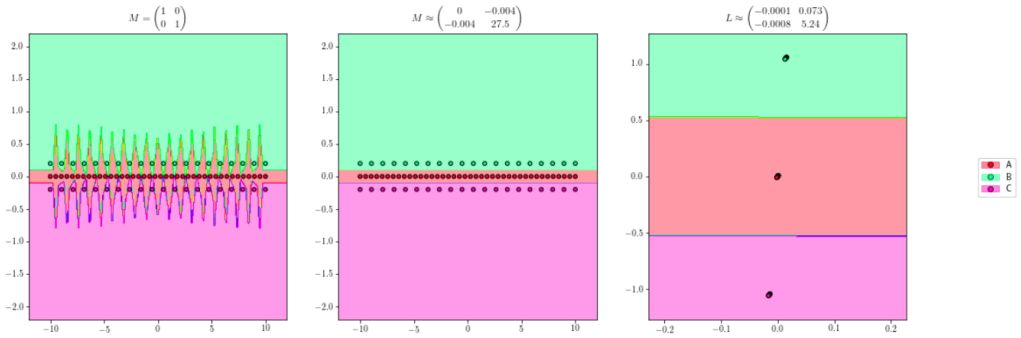


Figure 2.14: Taken from [77]. The individual tiles show the kNN prediction regions by color for every point in the image. Using the unmodified euclidean distancee, this would result in classification regions on the left. The reader can see, learning an appropriate distance, the classification is much more effective (middle). Finally, the dimensionality is also reducable with this dimension while still matching the classification accuracy (right).

Although we will not go into too much detail, we will outline some algorithms that [77] mentions in their work.

We will first start with dimensionality reduction techniques, then move to nearest neighbors classifiers, follow with nearest centroids classifiers and finally go over some information theory based algorithms. Notice that for almost each one of these methods, kernelized versions are available.

Dimensionality Reduction techniques include principal component analysis *PCA*, LDA ANMM

Deep Metric Learning

Besides the non-deep proposed in the work above models , in[39] also introduces the deep generalization to these frameworks.

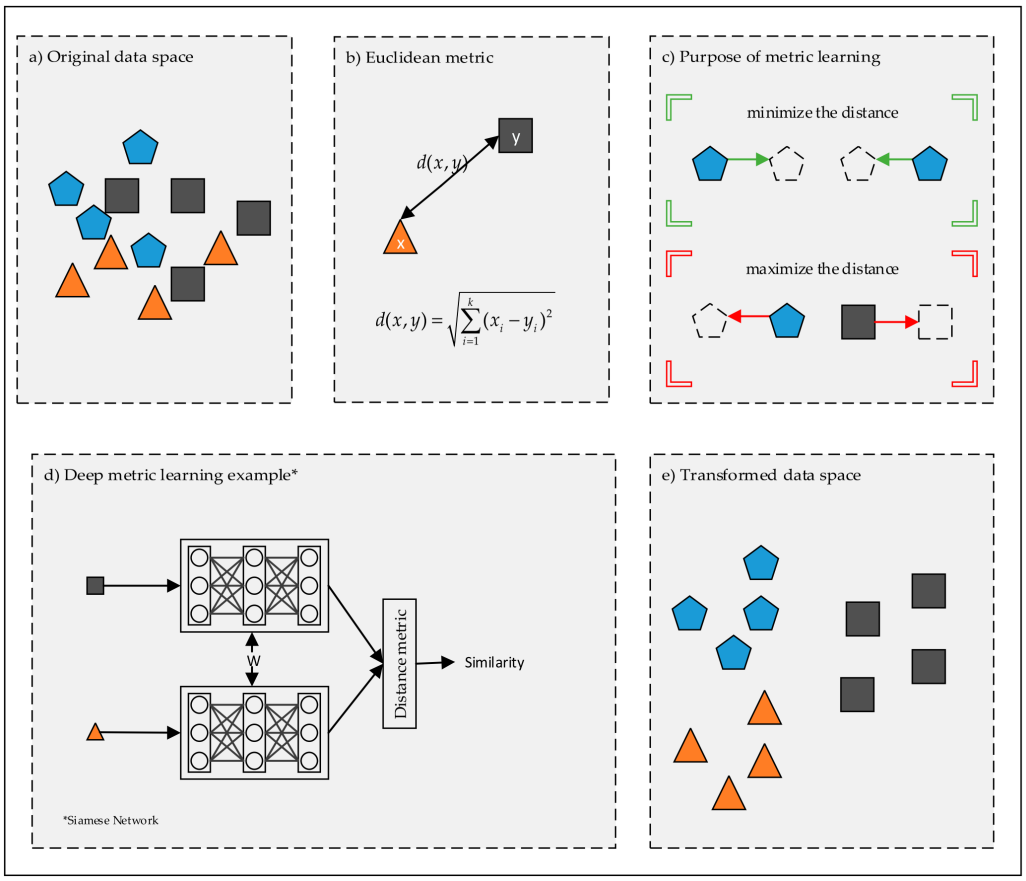


Figure 2.15: Taken from [39]. An illustration of deep metric learning. The space is transformed in such a way, that similar object are closer to each other, and dissimilar objects are moved away from each other.

The most prominent model is the siamese network [10], [15], [28], where the network consists of two identical sub-networks joined at their outputs. We provide an example the architecture looks that looks as follows:

```
class Siamese:

    def __init__(input_dim, latent_dim):
        self.fully_connected = nn.Linear(input_dim, latent_dim)

    def forward(input1, input2):
        latent_1 = self.fully_connected(input1)
```

```
latent_2 = self.fully_connected(input2)

distance = torch.sqrt(
    torch.sum((latent_1 - latent_2) * (latent_1 - latent_2))
)

return distance, latent_1, latent_2
```

Two samples are ingested by the neural network simultaneously. Both samples are passed through identical weights. Training occurs by minimizing the contrastive loss.

$$L_{\text{Contrastive}} = (Y)\frac{1}{2}\left(D_W\right)^2 + (1-Y)\frac{1}{2}\left\{\max\left(0, m - D_W\right)\right\}^2 \tag{2.54}$$

where we follow the convention that $Y = 1$ whenever the given sample-pairs are of the same class, and $Y = 0$ whenever the given sample-pairs are from different classes, and D_W is the output of the above neural network. Training can be highly unstable, and as such, the learning rate must be properly set, batches must have a balanced amount of both similar, and dissimilar pairs. Finally, training can fail if a single batch includes multiple domains of data.

For the triplet networks [33] which makes use of relation triplets, including an input X, a point similar to the input X^p and a point dissimilar to the input X^n, the following loss can be used for minimization.

$$L_{\text{Triplet}} = \max\left(0, \left\|G_W(X) - G_W\left(X^p\right)\right\|_2 - \left\|G_W(X) - G_W\left(X^n\right)\right\|_2 + \alpha\right) \tag{2.55}$$

where α forms a distance margin, similar to the length of the support vector in SVMs.

As the authors suggest, using a triplet logic results in more stable training and bigger margins between similarity classes. Analogous logic goes for

quadrupel loss etc., but it is unsure to what extend these extensions increase performance. This general idea can also be extended to angular distance (using cosine distances), and non-euclidean spaces, however, we will not go into further detail on this as this is not the focus of this work.

Although we will not go into further detail, other loss functions include histogram loss [79], the structured loss [76] n-pair loss [74], magnet loss [67], angular loss [84], quadruple loss [56], clustering loss [75], hierarchical triplet loss [26] and the multi-similarity loss [85].

Chapter 3

Related Work

Talke about [**?**]

Talk about community detection Make a subsection "Clustering for semantic structures in word and text embeddings"

3.1 Clustering for Semantics

One of our goals is to identify semantic clusters within a given set of context embeddings. Implicitly, this requires the algorithm to solve a multi-modality detection problem, which in itself is hard. This implies that we cannot use algorithms such as k-means [45, 47], which requires the number of clusters to be identified beforehand. Instead, we turn our attention to algorithms, which both find clustering assignments, but also the best number of clusters that the data can be clustered by.

Another constraint we put on ourselves is that the clustering methodology should not allow for the space to be projected into another space. Specifically, clustering should be conducted in the span of whatever set of points X is provided. There should be no additional function f applied to the points X which transform the space in which this clustering takes place. This means that we do not use algorithms which include dimensionality reduction

algorithms, such as autoencoders or other neural network based algorithms unless otherwise stated.

We will later on use different clustering algorithms to evaluate how well semantics is interpretable inside BERT vectors. We will give a short introduction on the following clustering algorithms.

Affinity Propagation was first introduced by [23]. Affinity propagation takes as input measures of similarity between pairs of data points. Real-values messages for multiple iterations are exchanged between data point until clusters start to emerge. Initialization happens by assigning the same class to close-enough points, message passing and the choice of which candidate sample to take on happens as defined by a *preference* threshold.

DBScan is a density based clustering algorithm [21] which initiates by finding representative samples that are in regions of high density, and expands clusters from there on. Due to the density criteria, the resulting clusterings don't have to be convex as is the case with k-means for example. **HDBScan** is a hierarchical extension of the DBScan algorithm [11]. **Optics** [50] is another extension of DBScan which keeps the cluster hierarchy for a variable neighborhood radius.

MeanShift is a centroid-based clustering algorithm [16]. After cluster-centers are randomly initialized, the centroids are updated by the mean of the cluster until convergence in the cluster-assignments is reached. A final post-processing step removes near-duplicate clusters.

The Chinese Whispers algorithm is one of the more relevant clustering algorithms used in this survey.

Here, we also introduce the chinese whispers algorithm. The chinese whispers algorithm was used to cluster for word-senses in the context of static word embeddings, such as in [57].

3.2 Structure inside BERT

(How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings)

going down the drain of "geometry" of BERT and ELMo.

could also go down the drain of bias (we would prefer to have uniform space over gender etc.)

-¿ does projection into some subspace which has same metric properties perhaps not make it asitropic?

pretty ok summary of what kind of properties we want from word-embeddings... (https://devopedia.org/word-embedding)

Especially in Named Entity Recognition (NER), there is a lot of use for static word-embeddings. I guess this is because we need static embeddings which represent the individual clusters?

-¿ Using pooling for some

-¿ Character level operation

-¿ Perhaps make good sense to work towards a word-embeddings where different vectors are close to each other?

-¿ perhaps find a metric space warping the vectors, s.t. an isotropic representation is achieved?

-¿ Perhaps tokenization is a big problem, but perhaps other architecture..? but retraining is too difficult.. probably best to just stick to BERT? one way or the other, we need good word-embeddings derived from good language models to form a probabilistic prediction of the concept

-¿ Could perhaps also try to make an adversarial autoencoder after the BERT layer (or continue training, s.t. a second loss is mimized as a downstream task?)

-¿ Perhaps distilling with "correct" tokens? i.e. another network which copies

BERT, but instead of outputting ##end, it outputs one of most frequent 20k words

-¿ thesaurus using a (set of) words. a little like sentence-generation, but generating most-probable examples

-¿ Everyone just averages token-embeddings..

-¿ perhaps fitting a GMM to the contextualized representations of BERT may give a good probability space..?

-¿ Perhaps make sense to apply MUSE to this?

-¿ Artetxe 2019 uses language models to generate embeddings. we also do this, but do it using 1) better language models, and 2) better

-¿ QUESTION: Which factors (mapping algo, embedding) is delimiting in automated embedding matching

-¿ perhaps create a GMM for each concept, based on how many modals we identify? how to estimate the number of clusters? by graph-clustering perhaps! (this could be very consuming)

-¿ Adversarial autoencoder on BERTs last models to enforce it to some better distribution

3.2.1 Other methods

Although the above presented methods are the prevailent methods for word-embeddings, there are also other methods which do not clearly fit into one of the above categories.

Generating "static" word-embeddings through contextual embeddings

Some work has been done in extracting word-embeddings from contextual language models like BERT or ELMo.

CITE (BERT WEARS GLOVES: DISTILLING STATIC EMBEDDINGS FROM PRETRAINED CONTEXTUAL REPRESENTATIONS)

(1) Uses *pooling* between BERT tokens to arrive at a single representation between words.

Here, sentences are split by space (tokenized). Words are tokenized further into a subword as defined by WordPiece (Wu et al. 2016).

The defined pooling operations looks as follows to arrive at the word from the individual subwords:

$$\mathbf{w}_c = f\left(\mathbf{w}_c^1, \ldots, \mathbf{w}_c^k\right); f \in \{\min, \max,\ \text{mean}, \text{last}\ \}$$

where we have subwords $w^1, \ldots, w^k$ such that cat $\left(w^1, \ldots, w^k\right) = w$

Why would any of these pooling operations result in a meaninigful source-word? This is just squishing tokens together!

-¿ This is a major limitation for which we may need to use ELMo -¿ However this may be needed for "unseen concepts" (which are unseen words...) -¿ Perhaps check what fasttext does...?

(2) Uses *context combination* to map from different contexts $c_1, \ldots, c_n$ to a single static embedding w that is agnostic of context.

Proposed are two ways to represent context.

Decontextualization For a single word-context, we siimply feed-in the word by itself to the model.

Aggregated combine w in multiple contexts. n sentences are sampled from the dictionary $\mathcal{D}$. From the multiple sampled words, we then apply pooling to arrive at a single representation that aggregates the different tokens into one.

$$\mathbf{w} = g\left(\mathbf{w}_{c_1}, \ldots, \mathbf{w}_{c_n}\right); g \in \{\min, \max, \text{ mean }\}$$

This post extracts (token?) word-embeddings: (https://towardsdatascience.com/nlp-extract-contextualized-word-embeddings-from-bert-keras-tf-67ef29f60a7b)

This seems to be a way to extract embeddings for tokens from BERT (https://github.com/img embedding)

(-¿How can we create a (parametric) probability density from a point-cloud distribution?)

perhaps not necessarily interpretable in standard euclidean space (https://www.kdnuggets.com features-interbertible.html) original (https://medium.com/thelocalminima/are-bert-features-interbertible-250a91eb9dc)

Perhaps we can mask all but the target token to arrive at one vector per token (and then combine them somehow...). But how do they extract the singular word-embeddings...?

(-¿ you could be like "acquiring bedeutung" is a big problem in many tasks. especially useful when we try to map one concept to another. we look at the NLP task for concreteness)

Generally, really good critique on this paper:

(https://openreview.net/forum?id=SJg3T2EFvr)

usually, we have sentence-embeddings, and do not look at word-embeddings.

(-¿ we don't want to add more and more context. we want a model which contains the polysemy of different contexts, which could allow for probability maximization..., otherwise we have to look at bigger and bigger documents to build more accurate language models, which becomes infeasible at some point. (although this would be the way humans work, because they live in context as well)

This blog aims to generate word-embeddings (and sentence-embeddings) from the BERT model. (https://mccormickml.com/2019/05/14/BERT-word-

embeddings-tutorial/)

create word-vectors by taking out what BERT predicts at the nth token. create word-vectors by concatenating or summing multiple layer's outputs.

the cosine similarity between these vectors seem pretty well-done!

(-¿ Does it make sense to use BERT and then on calculate word-embeddings through an extended fully-connected model)

-¿ ELMo may provide a better tokenizer, maybe better ot use this? What about GPT? ELMo uses moses tokenizer which seems word-level enough

-¿ How to solve this tokenization problem....

-¿ Can also analyze only words that exist.

3.2.2 Attention mechanism

Some analysis has been done with looking at the similarity between the produced context-vectors for biLM [61].

3.2.3 From token-vectors to word-vectors

[49] analyse the social biases that are part of the context embeddings and show that depending on the language model (ELMo, BERT), the amount of social bias deviates. Because context embeddings capture more than just semantics, bias is a natural implication of context vectors. A by-product of their research includes aggregating token-embeddings (i.e. aggregating the tokens *hav* and *###ing* to result at the word *having*). Although not very extensive on the topic of aggregation, the authors use techniques of mean-pooling, max-pooling, and last-pooling to determine arrive at a single context-vector, if the given token is intrinsically split-up by the language-model tokenizer (incl. BERT, ELMo, GPT).

Finally, evaluation on the corpora also yield inclusion of human-like biases [38] by names, races, male-vs-female.

3.2.4 Bias in BERT vectors

3.2.5 Change of meanings over time

[35] analyse how meanings quantitatvely using corpora from different historical epochs, including the 1890s, 1940s, 1960s, 1970s, 1990s and 2000s. Specifically, they measure the similarity scores for given context-vectors based on differently trained corpora. They demonstrate through examples of the word *gay* and *alien* that the similarity between context-vectors change.

3.2.6 Clinical concept extration

The task of concept-extration is heavily applied in the field in the clinical domain.

[90] trains a LSTM-Conditional-Random-Field to extract concepts. They used annotated corpora from doctors notes to train the model and model this as a named-entity-recognition task.

3.2.7 Discovering for semantics

[14] apply a co-clustering framework that discovery multiple semantic and visual senses of a given noun phrase. They use an structure-EM-like algorithm to achive this.

3.2.8 Embeddings for translation

A lot of work is done in the field of analysing embeddings for translation tasks, to further mitigate the black-box behvaior of neural network.

This includes [40] who use singular value canonical correlation analysis to compare hidden representations of language models between different languages. To arrive at a sentence-representation, they do mean-pooling over the outputted embedding-vectors of a sentence. They find interesting behavior in the arrangement of context-embedding in the Transfoerm-Big architecture proposed in the [80] paper, which is also inherently used in BERT

and GPT. Interestingly, they show that different languages do not project into similar spaces, thus making zero-shot learning tasks between languages difficult.

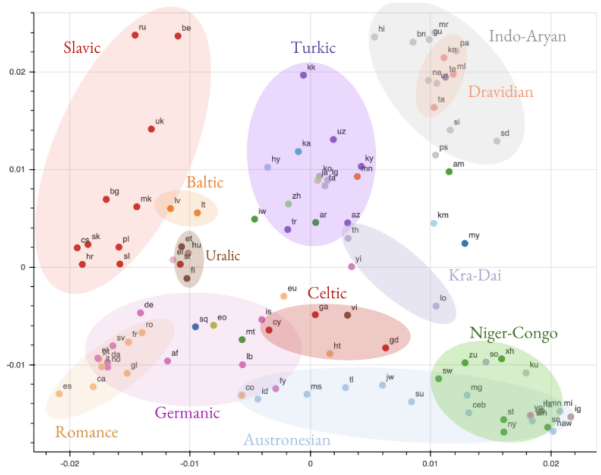


Figure 3.1: From [40], visualizing clustering of the encoder representations of all languages, based on ther SVCCA similarity.

[41] uses a cyclic loss to apply unsupervised machine translation suign monolingual corpora only. Here, sentence-embeddings are learned over time, which follow the cyclic loss constraint, and minimize the sunsupervised translation loss.

Other work tries to disentangle the different linguisitc, visual and audio-based features by supplying multi-modal input at once [46]. A translation model is learned which can not only translate between different domains, but also between different modalities (i.e. from image to audio, audio to text, and any such combination).

Chapter 4

Understanding the semantic subspace organization in BERT

We begin our analysis by trying to understanding how BERT organized its semantic subspace.

For this, we conduct three experiments. The experiments have varying difficulty for the algorithm to understand the subspace organization for a given word w. We start with a supervised algorithm which should build a discriminative model of the subspace, where labels are different semantic classes as defined by WordNet. The second experiment aims to identify a similar subspace organization through an unsupervised algorithm, implicitly identify a multimodal structure within the subspace at which we look at. Finally, we check how part-of-speech relates with semantics within the BERT model, hoping to simplify subsequent work by introducing the assumption that part of speech strongly correlates to semantics within the BERT model.

4.1 On the Linear Separability of meaning within sampled BERT vectors

The first experiment takes a word of interest w, samples a set of sentences S from a corpus (in this case, the above mentioned SemCor corpus and the news corpus [**?**] which does not have any WordNet annotated classes). To see if there is any structure within BERT vectors w.r.t. the different meaning of one word, we ask ourselves whether or not different part of the meaning are at different locations of the embedding space produced by BERT. We test this hypothesis proactively by training a linear classifier which learns a separating hyperplane between the different WordNet classes for w.

4.1.1 Experiment setup

To produce a context embedding, we pass through a sequence of words $[w_1, \ldots, w_i, \ldots w_T]$ sampled from S, where $w_i = w$ (i.e. w_i coincides with the word of interest). The output of BERT is a set of hidden representations which are interpreted as the context embeddings $[x_{w_1}, \ldots, x_{w_i}, \ldots, x_{w_T}]$, where x_{w_i} corresponds to the context embedding of w_i. Repeating this n times, this results in a feature matrix X, where each row corresponds to the sampled sentence, and the columns correspond to the latent dimensions of the context embedding. In all of our experiments, we set $n = 500$, unless otherwise stated. However, there are cases (especially for the SemCor dataset), where we cannot find n number of sentence which include our word of interest w. In that case, we sample as many sentences as available in the corpus, making $n = min(S_{\text{available } w}, 500)$. We make sure however to conduct our investigation with words that occur often enough and take 30 occurrences per WordNet class as a lower cap. Because this experiment requires w to have multiple WordNet classes, we restrict the choice of w to be polysemous as defined by WordNet. Due to the restricted resources of SemCor, this limits our analysis to the following set of words. Although these words are not the most intuitive polysemous words (as would `bank` be, for example), the limited "obviousness" should allow this experiment to be stricter w.r.t. not

rejecting the hypothesis.

Words used to test BERT-sampled vectors for linear separability of lexical features		
was	is	be
are	more	one
first	only	time

To arrive at uniform conditions for each sampled matrix of embedding vectors X, we apply a standard scalar such that the data is normalized around 0.0 with standard deviation 1.0. We also allow for dimensionality reduction using all sampled words. The reason for this is to have a stricter set of requirements for the separating hyperplane to be drawn. If the dimensionality of the data d is high, but the number of samples n is low, then the system of linear equations is underdetermined, as there is an infinite set of solutions possible to find a separating hyperplane. Thus, projecting X to a lower dimensionality works like a regularizer in that we restrict the set of hyperplanes to be drawn to be much more limited. On a separate note, this also tests if the semantic notion of the context vectors is kept at a lower dimension using a simple model such as PCA.

In summary, the experiment setup is captured by the following algorithm.

Algorithm 1: Checks sampled BERT vectors for linear interpretability by meaning

Input: A target word w_{target}; The latent dimensionality k for PCA;

Result: Accuracy of a logistic regression classifier

$\mathbf{D}, \mathbf{y} \leftarrow$ sample up to 500 sentences (as much as available) from the SemCor corpus which include the word w_{target}, along with the corresponding WordNet meaning-id;

$\mathbf{X} \leftarrow BERT(\mathbf{D})$ i.e. pass each sentence through BERT and retrieve the resulting word embedding x_w as defined in the above section;

$\mathbf{X}, \mathbf{y} \leftarrow oversample(\mathbf{X}, \mathbf{y})$ such that we don't have dominating classes;

$\mathbf{X}_{\text{train}}, \mathbf{X}_{\text{test}}, \mathbf{y}_{\text{train}}, \mathbf{y}_{\text{test}} \leftarrow trainTestSplit(\mathbf{X}, \mathbf{y}, testProportion = 0.4)$;

$\mathbf{X}_{\text{train}} \leftarrow StandardScaler(\mathbf{X}_{\text{train}})$ such that all the data is normalized;

$\mathbf{X}_{\text{train}} \leftarrow PCA(\mathbf{X}_{\text{train}}, k)$ such that all the data is projected to a lower latent dimensionality k;

$model \leftarrow LogisticRegression(\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}})$;

$\hat{\mathbf{y}}_{\text{test}} \leftarrow model.predict(\mathbf{X}_{\text{test}})$;

$accuracy, confusionMatrix \leftarrow loss(\mathbf{y}_{\text{test}}, \hat{\mathbf{y}}_{\text{test}})$;

return $accuracy, confusionMatrix$;

We use standard sklearn [**?**] implementations for the Standard Scalar, PCA and Logistic Regression. We use the Hugginface transformers library [**?**] for the this BERT model, and also all subsequent modification of BERT. The loss function we use is a binary simple $l1$ manhatten loss function, where the loss for a correct sample is 0, and for an incorrect class assignment is 1. We apply 5-fold cross validation, and measure the mean accuracy as well as the standard deviation of the accuracy. We also note the variance kept after projecting PCA on the lower dimensionality k to understand how a simple dimensionality reduction technique can affect the separation between semantic classes.

4.1.2 Results

We run the above experiment for a set of different k. The variance shown below is a fraction of 1.. A "variance kept" value of 1. corresponds to no

information loss in terms of eigenvalues left out.

We show the result for 3 of the experiments, for the words *was*, *is* and *one* respectively, as these are the words that have the highest number of occurrences within the SemCor dataset.

Table 4.1: Mean and standard deviation of the accuracy of a linear classifier trained on the 2 most common classes of WordNet meanings for the word *was*.

dimensionality	variance kept	accuracy (mean / ± stddev)
10	0.27	0.82/±0.03
20	0.41	0.81/±0.04
30	0.50	0.85/±0.03
50	0.63	0.92/±0.03
75	0.73	0.94/±0.02
100	0.81	0.95/±0.02

Table 4.2: Mean and standard deviation of the accuracy of a linear classifier trained on the 2 most common classes of WordNet meanings for the word *is*.

dimensionality	variance kept	accuracy (mean / ± stddev)
2	0.09	0.57/±0.02
10	0.29	0.82/±0.03
20	0.42	0.82/±0.04
30	0.51	0.83/±0.03
50	0.72	0.85/±0.04
75	0.78	0.84/±0.04
100	0.79	0.85/±0.03

Table 4.3: Mean and standard deviation of the accuracy of a linear classifier trained on the 2 most common classes of WordNet meanings for the word *one*.

dimensionality	variance kept	accuracy (mean / ± stddev)
2	0.10	0.55/±0.10
3	0.14	0.51/±0.05
10	0.34	0.59/±0.08
20	0.50	0.76/±0.03
30	0.62	0.77/±0.02
50	0.76	0.83/±0.06
75	0.87	0.87/±0.05
100	0.94	0.87/±0.05

There are many permutations We now also employ multi-class classificiation.

Table 4.4: Mean and standard deviation of the accuracy of a linear classifier trained on the the 4 most common classes of WordNet meanings for the word *was*.

dimensionality	variance kept	accuracy (mean / ± stddev)
2	0.08	0.38/±0.03
3	0.11	0.38/±0.04
10	0.28	0.65/±0.03
20	0.43	0.76/±0.04
30	0.53	0.83/±0.03
50	0.67	0.93/±0.01
75	0.77	0.95/±0.01
100	0.83	0.95/±0.01

We get very similar accuracies for "time", "made", "thought". However, these tables are left out as we do not deem these to be statistically significant. To make sure that no inconsistencies happened during testing, we also analysed the confusion matrices, which all were balanced across class-pairs,

as the transpose elements were usually off by only 1-4 samples, and the majority of the samples were correctly predicted (and thus on the diagonal of that matrix) .

The reader can see that for a learned hyperplane, the classification accuracy reaches more than 75% for dimensions of more than 20. This is also the case for multiclass classification problems, which removes the possibility that overfitting could have occurred for lower dimensions (due to the sheer amount of permutations that a hyperplane can take place in higher dimensions), which is also supported by the number of samples which is higher than the reduced dimensionality k.

4.2 On the Clusterability of meaning within sampled BERT vectors

Modality-detection can tell us a lot about the nature of a dataset. It can tell us what the predominant categories of a dataset are. Given a set of clusters amongst vectors $x_1, \ldots, x_n$ organized in a matrix X, the different clusters can be interpreted as the different modes of the data. This partitioning can be interpreted similarly to what principal components are, and what different categories are possible for these categories.

There are different ways to check if a set of vectors is clusterable, this includes theoretical ways to determine clusterability, as for example depicted in [**?**, **?**]. However, we are most interested in the practical clusterability of BERT, and as such, we decide to follow a brute-force approach to find clusters within BERT.

Because the experiment on linear separability has shown us, that different semantics are at different locations of the embedding space, we now want to investigate how "obvious" this clustering can be found, and whether a clustering can be found in the first place, or if the transitioning between the semantic categories is a smooth transition, or constitutes a hard partinioning amongst context vectors.

This experiment can be seen as an extension to the previous experiment. In the previous experiment, we try to train a discriminative model. In this experiment, we try to come up with a quasi-generative model which is able to detect modalities of the underlying true distribution, theoretically allowing us to generate new data samples.

4.2.1 Experiment setup

Initially, we tried to cluster the matrix of embedding vectors X using standard approaches with no hyperparameter optimization. However, all of the algorithms we manually tried, and with the default hyperparameters would fail and return -1 or similar messages, indicating that no clustering was found. We suspect that this is the case as the vectors produced by BERT are very smoothly distributed across the span of X.

As such, we had to adapt our experiment setup to be more sophisticated and implement a fast brute-force using random search and [**?**] and Bayesian optimization [**?**]. We use the Pytorch Adaptive Experimentation platform [**?**] which automatically chooses which of the two policies to use, given the dimensionality and number of parameters to search for. Specifically, our experiment setup is precisely described by the following algorithm. We introduce two algorithm. First algorithm calculates the clustering overlap between the predicted clustering $\hat{y}$ and the true cluster labels as defined by the WordNet semantic class labels y. The input to this algorithm is the clustering model that we want to use, which is one of the algorithms introduced in section (3.1) and the chinese whispers algorithm introduced above, a possible dimensionality to which the embeddings are reduced to using a predefined dimensionality reduction algorithm such as PCA, or UMAP, and finally the target word w for which we want to calculate this overlap score.

Algorithm 2: Checks sampled BERT vectors for clusters by meaning

Input: w_{target}: The target word whose BERT vectors we want to cluster;
$DimRed$: The dimensionality reduction method;
k : The latent dimensionality for the dimensionality reduction method;
$ClusterAlgorithm$: The clustering algorithm to use;

Result: The associated cluster-id with each sampled sentence, and the adjusted random index.

$\mathbf{D}, \mathbf{y} \leftarrow$ sample up to 500 sentences from the SemCor corpus which include the word w_{target}, along with the corresponding WordNet meaning-id, and compensate more sentences by sampling from the news corpus if less than 500 sentences are available in the SemCor sentence. We set y = -1 whenever no labeling information is available, which is the case if we don't sample data from the SemCor corpus.;

$\mathbf{X} \leftarrow BERT(\mathbf{D})$ i.e. pass each sentence through BERT and retrieve the resulting word embedding x_w as defined in the above section;

$\mathbf{X}, \mathbf{y} \leftarrow oversample(\mathbf{X}, \mathbf{y})$ such that we don't have dominating classes (all except for $y = -1$).;

$\mathbf{X} \leftarrow StandardScaler(\mathbf{X})$ such that all the data is normalized;

$\mathbf{X} \leftarrow DimRed(\mathbf{X}, k)$ such that all the data is projected to a lower latent dimensionality k;

$model \leftarrow ClusterAlgorithm(\mathbf{X})$;

$\hat{\mathbf{y}} \leftarrow model.predict(\mathbf{X})$;

$score \leftarrow AdjustedRandomIndex(\hat{\mathbf{y}}[\mathbf{semcor}], \mathbf{y}[\mathbf{semcor}])$;

return $score, \hat{\mathbf{y}}$;

Running this over all possible clustering algorithms, and taking the mean across a set of words, we hope to get a reliable estimate of which clustering approach generalizes best across words. This is depicted in the following algorithm

Algorithm 3: Algorithm to find the best model with the best fitting parameter configuration.

Input: w_{target}: The target word whose BERT vectors we want to cluster;

Result: The model which has the highest adjusted random index score between its predicted clustering, and the true underlying clustering as given by the SemCor WordNet class labels.

```
for clusterModelClass in clusterModels:
    for modelConfiguration in clusterModelClass.hyperparNext() :
        for w, X, numberOfSenses, trueClustering, knownIndices
        in crossvalidationDataIterator:
            score = Algorithm2(w, *configuration)
            if score > maxScore:
                maxScore = score
                bestModel = clusterModelClass
                bestConfig = modelConfiguration
return maxScore, bestModel, bestConfig
```

Because SemCor only had limited resources, and often a certain number of samples (more than 10) must be provided per word for the score to be significant enough, we were again limited to choosing some of the most occurring words within SemCor. Due to the limited size of SemCor, we also decided to only use words where all of the WordNet classes are present.

This limits us to the following choices for the target words w, with number of WordNet classes in paranthesis next to it: `have` (20), `live` (19), `report` (13), `use` (13), `test` (13), `know` (12), `write` (10), `tell` (9), `state` (11), `limit` (9), `allow` (10), `enter` (9), `concern` (7), `learn` (6), `seek` (6), `final` (5), `central` (3), `critic` (3), `topic` (2), `obvious` (1), `kitchen` (1), `pizza` (1). Because one of the implicit goals of the clustering algorithm is detecting the number of modes, we uniformly pick words `was`, `thought`, `made`, `only`, `central`, `pizza`. Cross-validating over a different number of cluster classes scores our algorithm in the multi-modality detection aspect of the task. Due to the limited size of the SemCor dataset, we aggregate the matrix of context vectors X

with sentences from an unsupervised news corpus as well. Because we don't have labels for the unsupervised corpus, however, we only apply scoring on the samples for which we have the WordNet labels (i.e. sentences sampled from SemCor).

The scoring scheme we use is the random adjusted index ARI [65], [36] which compares to what extent two clusterings are identical. This is necessary, because clusteres across clustering episodes will not produce the same cluster labels when run repeatedly. The ARI is computed by going through all possible permutations of the cluster-labels, and normalizing the score of the match for each permutation by a normalizing value. Because all possible permutations are taken into account, the ARI is *adjusted for chance*.

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \tag{4.1}$$

where $n_{i,j}$ is the number of items that are present in both clustering assignment, $a_i = \sum_j n_{i,j}$, $b_j = \sum_i n_{i,j}$ for two clustering a and b.

The resulting score is between $[-1.0, 1.0]$, where a score of 0 implies completely assignment of cluster-labels into random buckets, -1.0 implies a negative correlation, and 1.0 implies perfect similarity.

4.2.2 Results

To keep the presentation of the results concise, I will be going over the most meaningful results. This is the case for when we set $k = 20$.

Although initially it was apparent that adding additional dimensions to the dimensionality reduction technique would improve the clustering per-

formance we soon realized that introducing the chinese whispers algorithm would show that trimming down the dimensionality to $k = 20$ achieved the best results. We assume that this is the case as the 20 principal components with biggest eigenvalues capture enough about semantics such that the rest of the vectors can be left out. To keep the discussion focused, please refer to section (A.1) to see the results for $k = [50, 100]$ for different clustering methods, exluding the adapted chinese whispers algorithm.

The next two tables show results where we keep the dimensionality at $k = 100$ and keep the sample size at $n = 500$ and $n = 10000$.

Table 4.5: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 100$ and $n = 500$.

Clustering Model	**ARI Score**
Affinity Propagation	0.168
Chinese Whispers	**0.298**
DBScan	0.201
HDBScan	0.242
MeanShift	0.167
Optics	0.167

Table 4.6: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 100$ and $n = 1000$.

Clustering Model	ARI Score
Affinity Propagation	0.170
Chinese Whispers	0.249
DBScan	**0.260**
HDBScan	0.234
MeanShift	0.167
Optics	0.197

We quickly realize (together with the results in the appendix) that the sample does not have much effect on the clustering performance. This is also

the case because the corpora are exhaustive, and especially SemCor does not provide enough additional labels to make adding additional samples meaningful. Because of this, we change the sample size at either $n = 500$ and $n = 1000$ for subsequent trials.

Table 4.7: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 20$ and $n = 1000$.

Clustering Model	**ARI Score**
Affinity Propagation	0.165
Chinese Whispers	**0.349**
DBScan	0.167
HDBScan	0.273
MeanShift	0.226
Optics	0.167

So far, we had not added the `[SEP]` token to the sentence that BERT would input. The following table includes the `[SEP]` at the end of the sentence.

Table 4.8: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 20$ and $n = 1000$.

Clustering Model	**ARI Score**
Affinity Propagation	0.316
Chinese Whispers	**0.457**
DBScan	0.170
HDBScan	0.298
MeanShift	0.251
Optics	0.167

It is interesting to see that adding the `[SEP]` significantly increases the performance of the clustering algorithms. However, this is also expected, as BERT is trained using the `[SEP]` token. Adding 1 or more `[PAD]` at the end of the sentence does not improve the score any further.

Qualitative evaluation

So far, we have looked at the Adjusted Random Index Score as a way to measure how well the datasamples were clustered by meaning. However, because this is an arbitrary score, it is also important to do a qualitative analysis, and go over examples to see what this clustering scheme results in. The below clusterings form partitions of the sentences to be clustered.

Table 4.9: A famous equation

- - - -

She swooped him up into her arms and kissed him madly

- - - -

I place my son in her arms and I pray that it somehow comforts her perfect babies . . . into the loving arms of middle class+ Americans

which he falls back into her arms like a baby

Sometimes I took it into my arms and felt its surprising heft

- - - -

Figure 4.1: A famous equation

- - - -

and shuttle robotic arms of a solar array and truss

- - - -

a contingent of young arms that will allow us to win now

By and large, those arms remained as fictional as those in "The War ..."

and extensive use of robotic arms operating at their limits

staff of strong young arms that might have tamed the National League East

- - - -

(4.2)

Figure 4.2: A famous equation

- - - -

this country is so polarized that people spring to arms against any proposal

- - - -

At least they are carrying arms to protect themselves

His organization issued a call to arms

he shoves it in the faces of his comrades in arms

people who had taken up arms against the United States

Mostly non-Arab rebels took up arms in early 2003

- - - -

(4.3)

Figure 4.3: A famous equation

- - - -

The classic years of thearms race, the 1950s and ’60s before

- - - -

that concerns over nuclear arms proliferation in the Middle East

Russian adherence to anotherarms control treaty

he will press for peace and an eventualarms cut for the states

payments to the companies that suppliedarms to Iraq were often delayed

Mr. Safar denies any wrongdoing, including anyarms dealings

- - - -

(4.4)

Figure 4.4: A famous equation

- - - -

leaned back in his chair and, with arms crossed,

- - - -

God, fully in name, is at the bottom with his arms out wide.

If you feel yourself falling, spread your arms

Agamemnon, arms raised ... barely contained violence

Mr. James sat with his arms folded, his head lowered

she felt tears in her eyes and held her arms out in simple joy

- - - -

(4.5)

Figure 4.5: A famous equation

NOW MOVING OVER TO THE BANK EXAMPLE

- - - -

heavy withdrawals from the British bank Northern Rock reignited concern

- - - -

credit card and other consumer loans, forcing the bank to set aside $3.4 billion

by a mainland Chinese commercial bank in a U.S. bank

Investors fear the bank will be forced to write down

investors hoped that the bank had disclosed the

- - - -

(4.6)

Figure 4.6: A famous equation

- - - -

I would expect the bank by the trail to the left of the road to have bee

- - - -

The current slowed and swirled alongside a mud bank where cows had trodden to the water

But the bank had positioned itself well

provide plant scientists and farmers with a bank of genes

Upstairs at the large bank of cashiers

- - - -

(4.7)

Figure 4.7: A famous equation

- - - -

of their family's naturalization — bank deposit by bank deposit

- - - -

12 that the company might suffer a run on the bank because of mortgage concerns

Third, per several bank managers of major national banks

Local party bosses gained broad powers over state bank lending, taxes

government contracts, and a web of bank accounts

Prince Bandar's Washington bank accounts

- - - -

(4.8)

Figure 4.8: A famous equation

- - - -

to lift some of the mystery surrounding the central bank and improve communications with

- - - -

many economists had predicted that the bank would not cut its rate

expectations that the central bank will raise interest

a hint that the central bank plans to hold rates

China's central bank has stepped up its already huge pu

fertilizer prices in African countries, but that the bank itself had often failed to recognize

- - - -

(4.9)

Figure 4.9: A famous equation

- - - -

citing challenges for the investment bank and the potential for an above-average credit burd

- - - -

93 percent drop in profits at its investment bank last week

UBS said it did not expect its investment bank to return to profitability

defrauded by the investment bank in 1998 when

said in an interview that the investment bank approached him last month

said the leaders of Citigroup's investment bank and alternative

- - - -

(4.10)

Figure 4.10: A famous equation

EXAMPLES FOR KEY CLUSTERING

- - - -

Many of the key Arab states

- - - -

in two months and Australia's key S&P ASX 200 shed 1.9 percent

Wall Street rebounded Wednesday after key earnings reports from JPMorgan Cha

The Democratic candidate hires a key strategist

[CLS] Mr. Jones "quickly established a good rapport with key donors"

able to meet the two key officials in the government

- - - -

(4.11)

Figure 4.11: A famous equation

- - - -

former president of Trinity College, who played a key role in designing the test

- - - -

seen in the West as a key yardstick of the fairness of an election

treat ... as a key factor in its decisions about regulatory

which policy makers have called the key test for deciding whether to lower inte

9. 11. as a key element in pitch meetings

- - - -

(4.12)

Figure 4.12: A famous equation

- - - -

Interstate 5 is a key route connecting Southern and Northern Ca

- - - -

A key piece of new functionality for Ops Center

Youssef Squali at Jefferies & Co. says two key factors are driving the stock up

transforming connection with believers is a key element of evangelical Christianity

What would you say was the key element of your management style that allow

is a key indicator of retailer performance

in the West the key players were not a small group of intellectua

- - - -

(4.13)

Figure 4.13: A famous equation

- - - -

times change and technology advances, the key to the city symbolizes

- - - -

And an official, five-and-three-quarters-inch-long gold-plated pewter key to prove it

but it is small enough to fit onto a key chain

In it lay three keys on a key chain in the shape of a red

- - - -

(4.14)

Figure 4.14: A famous equation

- - - -

The Red Sox will now have all their key players from their 2007 championship team

- - - -

Mike Green scored 12 points and had a key assist in overtime as No. 22 Butler beat Virgini

or taking the chance of losing a key player to injury

But they never led, could not get a key basket at crucial times and played like a team

but Cam Long stole the ball near the top of the key and ran out the clock

- - - -

(4.15)

Figure 4.15: A famous equation

- - - -

Three of their key players played more than 4

- - - -

A key for the Giants on Sunday

chemical reactions on solid surfaces, which are key to understanding questions

but is she part of the conspiracy or the key to Sim's salvation?

whose ability to play on a sprained ankle against the Eagles key to that matchup

Horses have been the lifelong key to satisfying the real femini

Connecticut cornerback said the key to defeating Louisville woul

- - - -

(4.16)

Figure 4.16: A famous equation

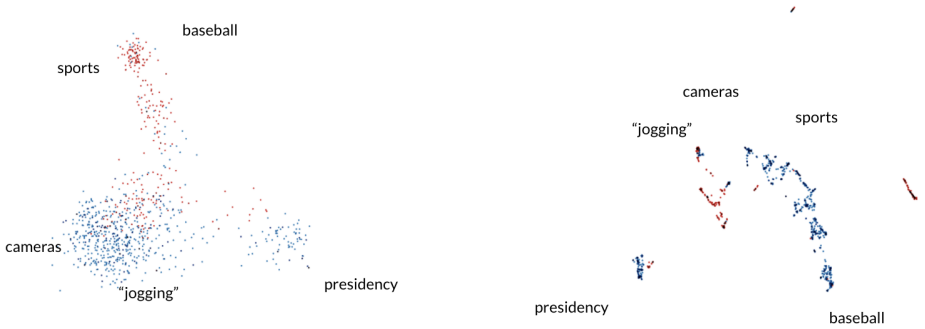
4.3 Correlation between Part of Speech and Context within BERT

4.3.1 Motivation

Because there are more resources in NLP with Part of Speech (PoS) tags, as compared to semantics, we want to analyse to what extent BERT sees similarities between PoS and, and because we assume a strong correlation between PoS and semantics, we analyse to what extent this is visible within BERT vectors.

4.3.2 Experiment setup

We test the hypothesis "semantics implies PoS" by conductin the following experiment. For a chosen target word w_t, we fixate one of the wordnet meanings. We then sample n sentences for the target word w_t where w_t has



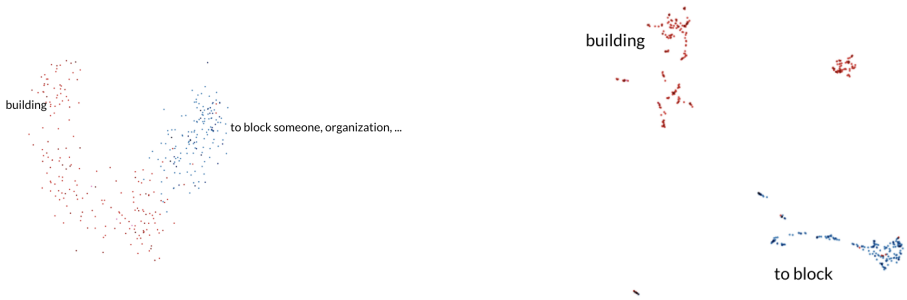
(a) 1a (b) 1b

Figure 4.17: plots of....

semantic meaning m in the occurring sentence. After we have sampled all the sentences, we determine the PoS for the target word w_t. We then calculate the percentage occurrence of the majority PoS class and record this as a percentage. If all of the sampled target words w_t for all the sentences have the same assigned PoS tag, then the score results in a value of 1.0. If the dominant PoS tag occurs only half the time, this number decreases to 0.5. Please notice that in this experiment, we only view simple PoS tags (i.e. "noun", "verb", "adjective", "pronoun"), and not the more complex ones listed above.

4.3.3 Results

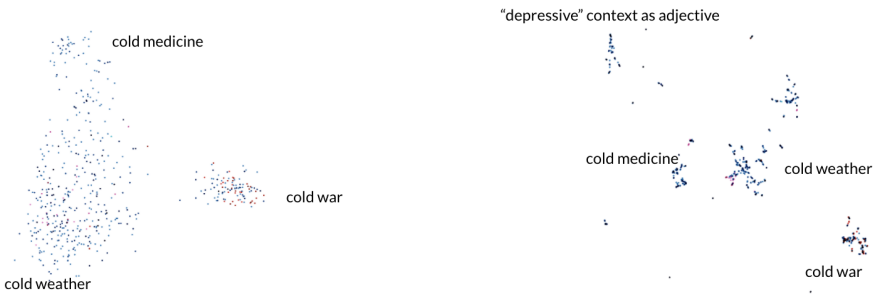
It is apparent that there is a strong relation between PoS and meaning. Especially "erstarrte" Verben are a strong part of this



(a) 1a

(b) 1b

Figure 4.18: plots of....



(a) 1a

(b) 1b

Figure 4.19: plots of....

Chapter 5

Exploiting subspace organization of semantics of BERT embeddings

The general pipeline of how BERT processes sentences is as follows. We have a sentence s which includes a set of words $w_1, \ldots, w_n$. This sequence of words is now given as input to the BERT model. First, this sequence of words is split into a set of tokens $t_1, \ldots, t_m$, where $m \geq n$. Now these tokens, which are in the vocabulary of the BERT tokenizer, are converted to indicies, which correspond to index of each individual embedding inside BERT.
We introduce *split-words*, for which we will be generating more specific embeddings. In particular, The idea behind this is that introducing more specialized embeddings for certain tokens will allow to model more complex distributions.
Because the non-modified (vanilla) BERT model uses a certain workflow, we will shortly introduce BERTs pipeline.

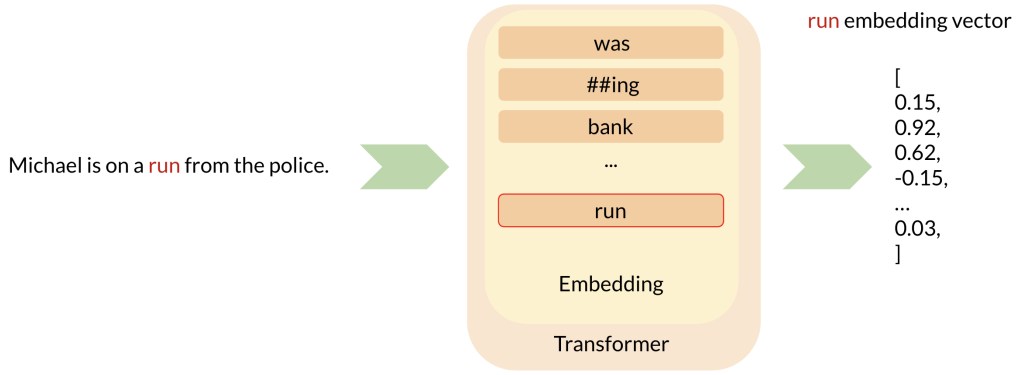


Figure 5.1: The BERT model takes as input a sentence s. The sentence s is converted to a sequence of BERT tokens $t_1, \ldots, t_m$ as defined in a given vocabulary V. Each item in the vocabulary V has a corresponding embedding vector inside the embedding layer of the transformer. This embedding vector is used by the intermediate layers of the transformer, and thus affects the downstream pipeline of the transformer for any subsequent layers of the transformer.

5.0.1 BERnie PoS

Motivation

We want to start with a simple model first. Because we have seen in the above section, that there is a strong correlation between semantics and part-of-speech, the initial idea is to introduce additional embedding vectors which are able to capture semantics (rather than just purely word tokens) better. In this case, part-of-speech is used as a proxy for semantics.

Experiment setup

BERnie PoS introduces one new embedding vector for each possible PoS configuration of the split-words.
As an example, instead of having a single embedding vector for the word *run*, we introduce two new embeddings *run_ VERB* and *run_ NOUN*, which both replace the intial *run*-embedding. We will refer to *run_ VERB* and *run_ NOUN* as *sub-embeddings*, as these exploit the subspace structure of the context embeddings sampled for the word *run*. As for the tokenizer, we

also introduce a mechanism which turns any occurrence of *run* into one of the sub-embeddings.
Specifically, our desired pipeline would not look as follows.

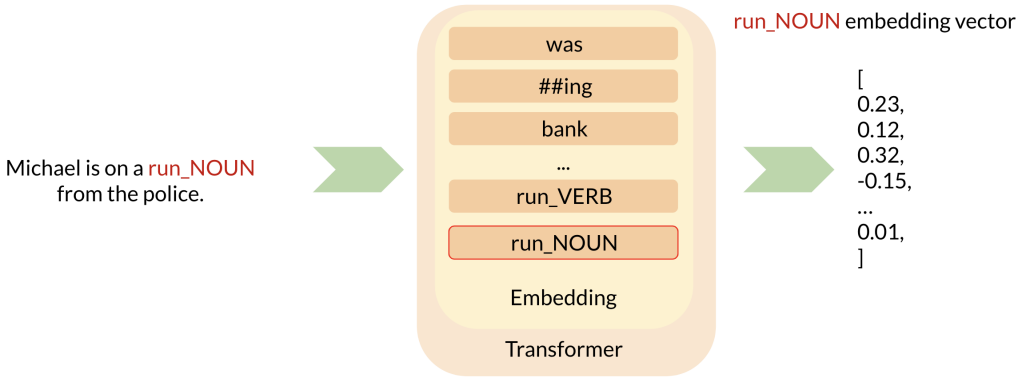


Figure 5.2: The modified pipeline. The BERnie model takes as input a sentence s. The sentence s is converted to a sequence of BERT tokens t_1, ..., t_m as defined in a given vocabulary V. For each target token t_{target}, we make the token more specific by converting the token to a more specialized token-representation, which specifies the part-of-speech information as part of the token. In this case, *run* becomes run_VERB. Again, each item in the vocabulary V has a corresponding embedding vector inside the embedding layer of the transformer. This embedding vector is used by the intermediate layers of the transformer, and thus affects the downstream pipeline of the transformer for any subsequent layers of the transformer.

To identify whether the occurring *run* in an example sentence is a verb or a noun, we use the spaCy part-of-speech tagger [1], which claims 92.6 % accuracy for this task.

5.0.2 BERnie Meaning

Motivation

Similar to the above model, we are introducing new BERT embeddings. This time however, we do not introduce new tokens by part-of-speech, as we did in the previous section, but rather by semantics. For this, we use the a similar methodology as the model from the section 5.0.1. However, instead of replacing word by their part-of-speech specialization, we replace words

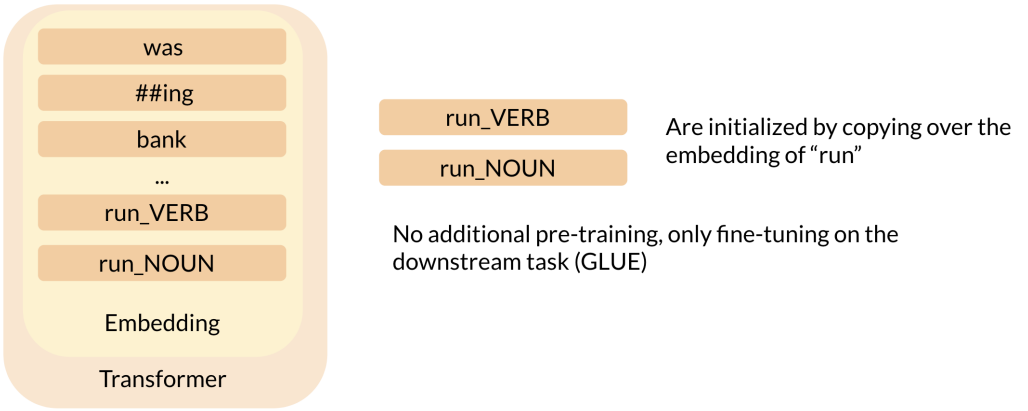


Figure 5.3: Inside the embedding layer of BERT, we introduce more specific embeddings *run_ VERB* and *run_ NOUN*. The BERT model should intuitively now capture more expressiveness, as the model size increased. The original *run* embedding is removed.

by some semantic specialization. The idea here is to adhere more strongly to the subspace structure as seen in section 4.3.
Thus, we introduce the clustering methodology introduced in 4.2.

Experiment setup

Again, we start with the standard BERT model, whose pipeline looks as follows.

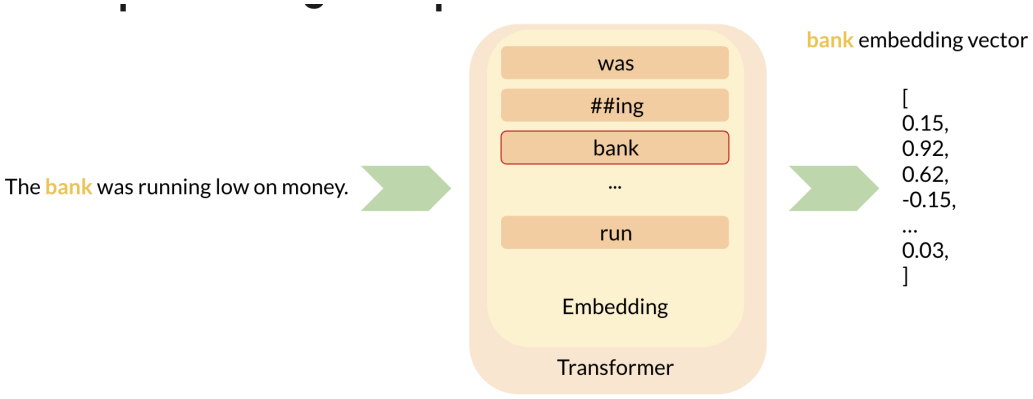
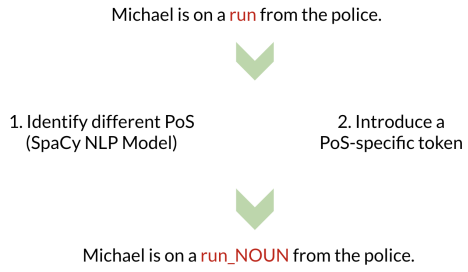
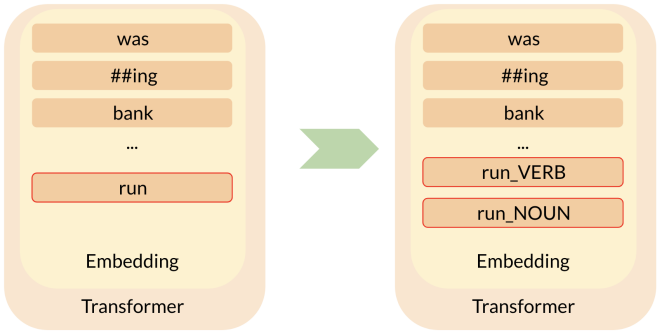


Figure 5.5



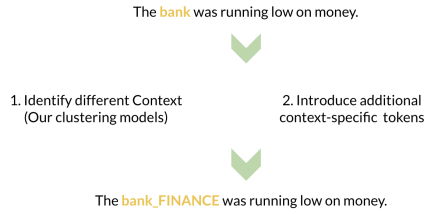
(a) 1a: The part-of-speech modified tokenizer pipeline



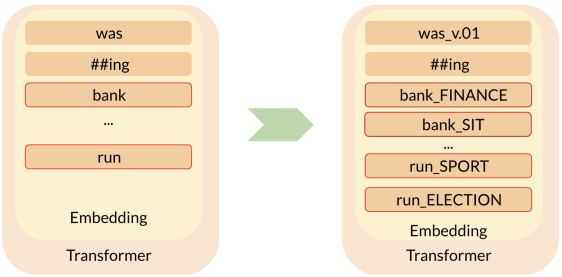
(b) 1b: The part-of-speech modified BERT embedding

Figure 5.4: plots of....

We now modify the tokenizer in such a way, that ambigious and polysemous words are replaced with a more specific token. As an example, we want to replace the word *bank* with a token, which captures whether or not the *bank* that we refer to implies a 1) financial institution, or 2) a river bank.
We use the our clustering approach from 4.2 as the intermediate model to distinguish on which bank the sentence refers to. We also introduce new embedding vectors that the new tokens correspond to.



(a) 1a: The part-of-speech modified tokenizer pipeline



(b) 1b: The part-of-speech modified BERT embedding

Figure 5.6: plots of....

After we have concluded these changed, we arrive at the following model, which has a modified tokenizer, and a modified BERT model. We call the corresponding model **BERnie**.

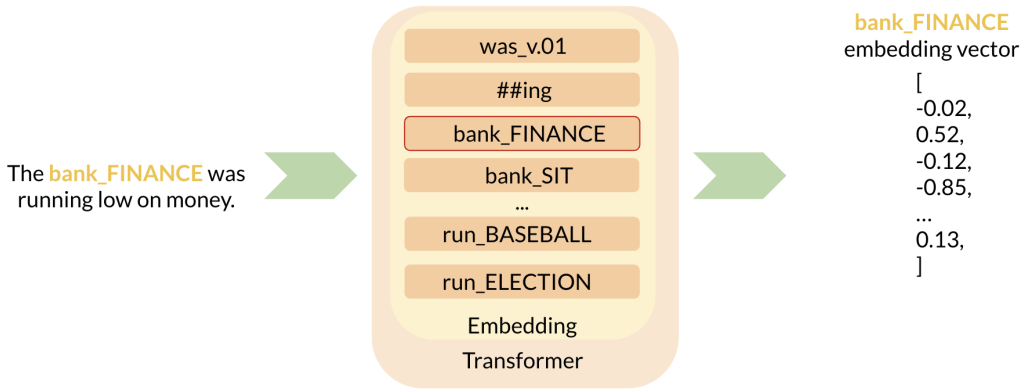


Figure 5.7: The BERT model takes as input a sentence s. The sentence s is converted to a sequence of BERT tokens $t_1, \ldots, t_m$ as defined in a given vocabulary V. Each item in the vocabulary V has a corresponding embedding vector inside the embedding layer of the transformer. This embedding vector is used by the intermediate layers of the transformer, and thus affects the downstream pipeline of the transformer for any subsequent layers of the transformer.

Results

We now do a short evaluation of the introduced BERnie PoS and BERnie model. To understand on a wide scale, we use the GLUE benchmarks to understand what effect the extension of the embedding layer on points of high variance has.

One can see that in general, the BERnie models performs slightly worse.

All values are the average of two runs. Weights which are instantiated specifically for the GLUE tasks are once instantiated with a random seed of 42, and once with a random seed of 101.

Although most experimental results are similar between standard BERT and the modified BERnie PoS and BERnie models, the WNLI and RTE experiments are considerably deviating in performance through the addition of the additional embedding vectors. For the BERnie model, WNLI performs **10.56%** better than standard BERT, and RTE performs **-7.40%** worse than the standard BERT implementation. Similar results are observable for the difference between the unmodified BERT and the BERnie PoS model with **3.52%** improvement for WLNI and **-3.79%** decline in performance for the

		BERT	BERnie PoS	BERnie
CoLA	Accuracy	**0.5739**	0.5263	0.5457
MRPC	Accuracy	**0.8223**	0.8199	0.8064
	F1	**0.8778**	0.8614	0.8684
	Mixed	0.8501	**0.8579**	0.8374
SST-2	Accuracy	0.9214	0.9203	**0.9266**
STS-B	Correlation	**0.8841**	0.8615	0.8574
	Pearson	**0.8860**	0.8621	0.8587
	Spearman	**0.8822**	0.8601	0.8567
QNLI	Accuracy	**0.9126**	0.9090	0.9020
RTE	Accuracy	**0.6462**	0.6083	*0.5722*
WNLI	Accuracy	*0.35915*	0.3947	**0.4649**
MNLI	Accuracy	0.8453		0.8334
SNLI	Accuracy	**0.8461**		0.8367
QQP	Accuracy	**0.9113**		0.9042
	F1	**0.8809**		0.8732
	Mixed	**0.8961**		0.8887

Table 5.1: asj

RTE task.
Also, compared to BERnie PoS, the standard BERnie model amplifies the performence-difference to BERT in both the negative and positive direction.

5.0.3 BERnie Meaning with additional pre-training

Motivation

In section 5.0.2, we can see that extending the BERT model generally leads to worse performance.
We assume that the BERnie require additional pre-training, as we are adding more specific embedding vectors, without fine-tuning them. We assume that this is necessary, as this is how the weights of BERT are also trained in the original paper [19] through the masked language model approach.

Experiment setup

We evaluate the models of the previous experiments.

Table 5.2: Mean and standard deviation of the accuracy of a linear classifier trained on the the 4 most common classes of WordNet meanings for the word *was*.

dimensionality	variance kept	accuracy (mean / ± stddev)
2	0.08	0.38/±0.03
3	0.11	0.38/±0.04
10	0.28	0.65/±0.03
20	0.43	0.76/±0.04
30	0.53	0.83/±0.03
50	0.67	0.93/±0.01
75	0.77	0.95/±0.01
100	0.83	0.95/±0.01

WNLI is a comprehension task where a model must read a sentence with a pronoun, and select the referent of that pronoun form a list of choices.
Each one is contingent on contextual information provided by a single word or phrase in the sentence. To convert the problem into sentence pair classification, we construct sentence pairs by replacing the ambiguous pronoun with each possible referent.
To train an ablation study, whether or not additional pre-training improves the instantiated word-vectors, we conduct the following experiment. We instantiate the additional embeddings as described in the previous section. We then run one full epoch of training with the same training parameters as used for GLUE on the news.corpus.2007 dataset using a masked languag emodel methodology . We save this model and load it for the GLUE tasks. BERnie full Pre-training trains the entire embeddings, whereas BERnie partial Pre-Training fixates all embedding vectors except the ones for the ones that were newly added.

Table 5.3: Mean of mulitple experiments for BERnie with addiitonally trained embeddings and weights.

		BERnie	BERnie full Pre	BERnie partial Pre
CoLA	Accuracy	0.5457	0.5418	0.5731
SST-2	Accuracy	0.9266	0.9169	0.9266
QNLI	Accuracy	0.9020	0.5374	0.6700
RTE	Accuracy	0.5722	0.4784	0.4729
WNLI	Accuracy	0.4649	0.4225	0.4507

We suspect a bug in loading the model appropriately for the MRPC task, and thus ommited the results for this task.

5.1 Compressing the non-lexical out

Motivation

Experiment setup

We conduct three experiments.
First, we use a simple algorithm to what extent we can disentangle the semantic space within a single sampled word.
The, we use a simple algorithm to what extent we can disentangle the semantic space within multiple sampled words, where we assume that the underlying semantics should again be mutually exclusive (i.e. the sampled words are not independent towards each other.
Finally, we use a simple algorithm to what extent we can disentangle the semantic space within multiple sampled words, where we don't inject any assumptions upon the underlying semantics of the words, i.e. the words are independent towards each other.

Results

Chapter 6

Conclusion

Bibliography

[1] spacy: Part-of-speech tagging. `https://spacy.io/api/annotation#pos-tagging`. Accessed: 2020-04-13.

[2] Understanding lstms. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2020-05-02.

[3] David Alvarez-Melis and Tommi S. Jaakkola. Gromov-wasserstein alignment of word embedding spaces. 2018.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv*, 2016.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155, 2003.

[6] Yoshua Bengio, Holger Schwenk, Jean-Sebastien Senecal, Morin Frederic, and Jean-Luc Gauvain. Neural probabilistic language models. *Innovations in Machine Learning*, pages 137–186, 2006.

[7] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*, 2009.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. 2017.

[9] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[10] Jane Bromley, Isabelle Guyon, and Yann LeCun. Signature verification using a "siamese" time delay neural network. 1994.

[11] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clusteering based on hierarchical density estimates. *PAKDD 2013: Advances in Knowledge Discovery and Data Mining*, pages 160–172, 2013.

[12] Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. *arXiv preprint arXiv:1708.00055*, 2017.

[13] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. Large scale online learning of image similarity through ranking. *JMLR*, 2010.

[14] Xinlei Chen, Alan Ritter, Abhinav Gupta, and Tom Mitchell. Sense discovery via co-clustering on images and text. 2019.

[15] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. 2005.

[16] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 603–619, 2002.

[17] Alexis Conneau, Guillaume Lample, Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. 2017.

[18] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer, 2005.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Arxiv*, 2018.

[20] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[21] Matrin Ester, Hans-Peter Kriegel, Joerg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters. *KDD-96 Proceedings*, 1996.

[22] W. N. Francis and H. Kucera. A standard corpus of present-day edited american english, for use with digital computers. 1964.

[23] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. 2007.

[24] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 1994.

[25] Octavian-Eugen Ganea, Gary Becigneul, and Thomas Hofmann. Hyperbolic entailment cones for learning hierarchical embeddings. 2018.

[26] Weifeng Ge, Weilin Huang, Dengke Dong, and Matthew R. Scott. Deep metric learning with hierarchical triplet loss. 2018.

[27] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics, 2007.

[28] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. 2006.

[29] Bar-Roy Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second pascal recognising

textual entailment challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, volume 6, pages 6–4. Venice, 2006.

[30] Zellig Harris. Distributional structure. *Word*, 10, 1954.

[31] W. F. Hegel. Encyclopedia of the philosophical science: Science of logic. 1817.

[32] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Arxiv*, 1997.

[33] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. 2014.

[34] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[35] Renfen Hu, Shen Li, and Shichen Liang. Diachronic sense modeling with deep contextualized word embeddings: An ecological view. 2019.

[36] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, pages 193–218, 1985.

[37] Shankar Iyer, Nikhil Dandekar, and Kornel Csernai. First quora dataset release: Question pairs, 2017.

[38] Sophie Jentzsch, Constantin Rothkopf, and Patrick Schramowski Kristian Kersting. On measuring social biases in sentence encoders. *AIES 2019 - Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 37–44, 2019.

[39] Mahmut Kaya and Hasan Sakir Bilge. Deep metric learning : A survey. *MDPI Symmetry*, 2019.

[40] Sneha Kudugunta, Ankur Bapna, Isaac Caswell, Naveen Arivazhagan, and Orhan Firat. Investigating multilingual nmt representations at scale. 2018.

[41] Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. Unsupervised machine translation using monolingual corpora only. *ICLR*, 2018.

[42] Jason Lee, Kyunghyun Cho, and Thomas Hofmann. Fully character-level neural machine translation without explicit segmentation. 2017.

[43] Hector Levesque, Ernest Davis, and Leora Morgenstern. The winograd schema challenge. In *Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2012.

[44] Peter J. Liu, Mohammad Saleh, Etienne Pot†, Ben Goodrich, Ryan Sepassi, Łukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. 2018.

[45] S. P. Lloyd. Least squares quantization in pcm. *Technical Report RR-5497 Bell Lab*, 1957.

[46] Shuang Ma, Daniel Mcduff, and Yale Song. M3 d-gan: Multi-modal multi-domain translation with universal attention. 2019.

[47] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. *L. M. Le Cam and J. Neyman (Eds.), Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, pages Vol. 1, pp. 281–297, 1967.

[48] P. C. Mahalanobis. On the generalized distance in statistics. *Proc. Nat. Inst. Sci.*, 1936.

[49] Chandler May, Alex Wang, Shikha Bordia, Samuel R. Bowman, and Rachel Rudinger. On measuring social biases in sentence encoders. 2019.

[50] Ankerst Mihael, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *ACM SIGMOD Record 28, no. 2*, pages 49–60, 1999.

[51] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

[52] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[53] George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, pages 235–244, 1990.

[54] George A. Miller, Martin Chodorow, Shari Landes, Claudia Leacock, and Robert G. Thomas. Using a semantic concordance for sense identification. pages 240–243, 1994.

[55] Panagiotis Moutafis, Mengjun Leng, and Ioannis A. Kakadiaris. An overview and empirical comparison of distance metric learning methods. *IEEE Transactions on Cybernetics*, 2017.

[56] Jiazhi Ni, Jie Liu, Chenxin Zhang, Dan Ye, and Zhirou Ma. Fine-grained patient similarity measuring using deep metric learning. 2017.

[57] Maria Pelevina, Nikolay Arefyev, Chris Biemann, and Alexander Panchenko. Making sense of word embeddings. *Journal of Classification*, 2016.

[58] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. *EMNLP*, page 1532–1543, 2014.

[59] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Po. Semi-supervised sequence tagging with bidirec- tional language models. *ACL*, 2017.

[60] Matthew E Peters, Mark Neumann, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *Arxiv*, 2018.

[61] Matthew E Peters, Mark Neumann, Luke Zettlemoyer, and Wen-Tau

Yih. Dissecting contextual word embeddings: Architecture and representation. 2018.

[62] Alec Radford, Karthk Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *Arxiv*, 2018.

[63] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *Arxiv*, 2019.

[64] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[65] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, pages 846–850, 1971.

[66] Marek Rei. Semi-supervised multitask learning for sequence labeling. 2017.

[67] Oren Rippel, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev. Metric learning with adaptive density discrimination. 2016.

[68] David Rumelhart, Geoffrey Hinton, and Ronald Williams. Learning internal representations by error propagation. 1985.

[69] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. 2019.

[70] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv*, 2015.

[71] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. 2016.

[72] Dinghan Shen, Pengyu Cheng, Dhanasekar Sundararaman, Xinyuan Zhang, Qian Yang, Meng Tang, Asli Celikyilmaz, and Lawrence Carin. Learning compressed sentence representations for on-device text processing. 2019.

[73] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[74] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. 2016.

[75] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. 2017.

[76] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep

metric learning via lifted structured feature embedding. *CVPR*, 2016.
[77] Juan Luis Suarez, Salvador Garcia, and Francisco Herrera. A tutorial on distance metric learning: Mathematical foundations, algorithms and experiments. 2019.
[78] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. 2011.
[79] Evgeniya Ustinova and Victor Lempitsky. Learning deep embeddings with histogram loss. *NIPS*, 2016.
[80] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Neural Information Processing Systems*, 2017.
[81] Luke Vilnis and Andrew McCallum. Word representations via gaussian embedding. *CoRR*, abs/1412.6623, 2014.
[82] Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *NeurIPS*, 2019.
[83] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *ICLR*, 2019.
[84] Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. Deep metric learning with angular loss. 2017.
[85] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R. Scott. Multi-similarity loss with general pair weighting for deep metric learning. 2019.
[86] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.
[87] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.
[88] Ludwig Wittgenstein. Philosophical investigations. 1953.
[89] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, and Mohammad Norouzi et. al. Google's neural machine translation system: Bridging the gap between human and machine translation. 2016.
[90] Henghui Zhu, Ioannis Ch Paschalidis, and Amir Tahmasebi. Clinical concept extraction with contextual word embedding. 2018.

Appendix A

More Results

A.1 Finding the best clustering model

Table A.1: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 50$ and $n = 500$.

Clustering Model	ARI Score
Affinity Propagation	0.001
DBScan	0.000
HDBScan	**0.328**
MeanShift	0.004
Optics	0.000

Table A.2: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 50$ and $n = 1000$.

Clustering Model	ARI Score
Affinity Propagation	0.000
DBScan	0.139
HDBScan	**0.271**
MeanShift	0.005
Optics	0.070

Table A.3: The maximum ARI score achieved during hyperparameter optimization for the different models as described by experiment for $k = 100$ and $n = 1000$.

Clustering Model	ARI Score
Affinity Propagation	0.000
DBScan	**0.215**
HDBScan	0.359
MeanShift	0.003
Optics	0.000

Appendix B

Using word vectors in translation

Although word-vectors can be used for a variety of tasks, we will focus on some varieties of how these vectors are used for machine translation (of human languages) tasks.
Amongst the most prominent method is the MUSE model introduced by [17]. Here, a mapping W is found using either an unsupervised methodology by minimizing the batch-size cosine-distance between sampled word-vectors between two languages A and B, or a supervised methodology is devised using the procrustes algorithm.

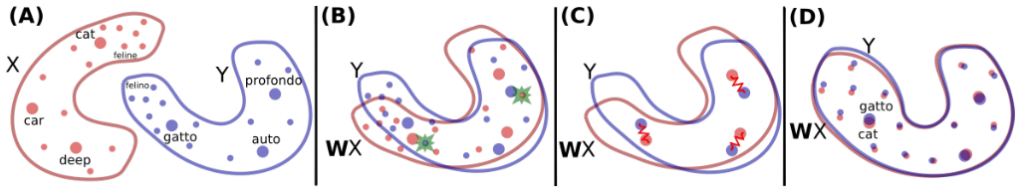


Figure B.1: Taken from [17]. Each

Specifically, the following loss-function optimizes over the space of possible mapping matrices W^*:

$$W^\star = \underset{W \in O_d(\mathbb{R})}{\operatorname{argmin}} \|WX - Y\|_{\mathrm{F}} = UV^T, \text{ with } U\Sigma V^T = \mathrm{SVD}\left(YX^T\right) \tag{B.1}$$

Unsupervised training is conducted using a discriminator and a mapping function, where the discriminator's task is to identify the origin-distribution, and the mapping function is supposed to fool the discriminator, resulting in

a mapping function which maps all word-emebddings into a common global embedding space.
A generalization of these word vectors to point-vector probability-distributions are captured by [3]. Here, the point-vectors are interpreted as point-delta-distributions in the $d+1$ dimensional space. Common optimal transport methods are coupled with the Gromov-Wasserstein loss to find an orthogonal mapping which maps from vector-space X to vector space Y, and thus from the word-token of one language to another. One way to achieve this is to use the supervised loss-metric and solve the procrustes problem, finding correspondences between the columns of X and columns of Y through:

$$\min_{\mathbf{P} \in O(n)} \|\mathbf{X} - \mathbf{P}\mathbf{Y}\|_F^2 \tag{B.2}$$

with $O(n) = \left\{\mathbf{P} \in \mathbb{R}^{n \times n} | \mathbf{P}^\top \mathbf{P} = \mathbf{I}\right\}$. The resulting algorithm achieves similar performance to MUSE while requiring only a fraction of the computational cost, being able to get competitive results on CPU.
The unsupervised approach deals with finding a transportation map T that fulfills

$$\inf_T \left\{ \int_{\mathcal{X}} c(\mathbf{x}, T(\mathbf{x})) d\mu(\mathbf{x}) | T_{\#}\mu = \nu \right\} \tag{B.3}$$

,
where μ and ν are the point-delta distributions describing the word-embeddings in the probability space. The relaxed Kantorovich's formulation is then finally used to reduce this problem to finding a set of transportation plans in a polytopes.

$$\Pi(\mathbf{p}, \mathbf{q}) = \left\{\Gamma \in \mathbb{R}_+^{n \times m} | \Gamma \not\Vdash_n = \mathbf{p}, \Gamma^\top \mathbf{1}_n = \mathbf{q}\right\}$$

Finally, the cost function

$$\min_{\Gamma \in \Pi(\mathbf{p}, \mathbf{q})} \langle \Gamma, \mathbf{C} \rangle$$

is minimized to find an optimal transportation plan.
Another extension is the work by

Word2Vec

BERT conditions on the rest of the input sentence.
BERT uses words, subwords and individual characters (in total 30'000) that are then used as tokens.
Idea is to do the following: Concepts (and thus words), are represented across multiple contexts. We can create probabilistic word-embeddings by sampling from a corpus the context of a specific word. From multiple samples of the context-embedding-vector, we take the mean and stddev, or create a GMM if there are multimodal logic (we can check this multimodality by runniing some sort of rejection sampling algorithm). Then we have a probability density function (one for each language), and can map one into another.
Perhaps we could split up too high-variance embeddings to multimodal embeddings, depending on their use-cases.
This allows for interpretability in polysemy sentences.
Not using more complex flows implies that the flow itself is not the bottleneck (they probably tried out other flows as well).
Are the individual word-embeddings going to be one big blob with high variance, or is it going to be a multi-modal distribution...?
Another task we may look at is, from a given word-embedding, sample it's context. Not entirely sure how to do this with BERT and co.
At what point do we overfit, and at what point do we generalize?

Artetxe bilingual token matching through unsupervised machine translation

- Input is cross-lignual word-embeddings - Build an unsupervised phrase-based statistical machine translation system - Derive phrase-based embeddings from input-word-embeddings by taking top 400'000 bigrams and 400'000 trigrams -¿ take arithmetic mean of word-embedding - score top 100 close phrases using softmax cosine similarity - generation of synthetic parallel corpus using this approach - Then use FastAlign to use parallel corpus to align words -