From Context Embeddings to Meaning 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

April 13, 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.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Motivation

- Natural Language Understanding (NLU) lies in the intersection of formalising human text into a computer-readable format. - Computer-readble units must be numerical, thus we represent words and meanings by vectors - The relationships between vectors should cover underlying relations between the word meaning - Language models are a generalization of word vectors - Word embeddings such as Word2Vec and language models such as BERT are used for other tasks, which are refered to as ”downstream” tasks, due to their nature of using up these word-embeddings and language models

- Because these are the most basic units of text, any shortcomings and properties will propagate over to any downstream task - Some properties include that static word vectors like word2vec form a bijection between discrete vectors and word tokens. However, because a single word can entail multiple meanings, such as the polysemous word ”bank” ((1) financial instution, (2) a sitting bench), this results in a lossy compression - Other language models like context embeddings entail too much information, and also include other linguistic features such as semantic information, relatenedness to unrelated concepts. These properties can easily introduce bias into any downstream tasks. - We conjecture that many language tasks incl. translation will benefit most from meaning information

- Our general approach is to start with a complex language model that outputs context embeddings, and find signals / vectors that entail meaning.
- Although this work is dedicated to the domain of natural language understanding, the principles analysed in this work should generalize to other domains with similar structural properties as well, where we want to denoise some embedding space to some select properties.

Chapter 3

Background

TODO: Take some more from here

- Some of the main points behind [?] are that there is inherent structure in language, and that this structure can be formalized. - [?] mentions that the relation between the linguistic representation in terms of sounds and tokens are related to the meaning that the representation entail. - However, despite the obvious relationship, the distinction between distributional structure of the language and meaning is not always clear. and that there is a ”parallel” meaning structure, and argues that there is not a one-to-one relation between vocabulary and classification of meaning. - Also argues that the meaning of a word is entailed by it’s environment, and the words that it occurs with. - Other viewpoints (CITE HOFMANNs ”teachers”) In the end, language captures the evolution of human thought.

- cite paper ”meaning is classified by its context”

Chronologically, the following data structures are manifestations of the above idea of defining a word by it’s neighbourhood.

3.1 Linguistic Features

Polysemy: a word may have multiple meanings. in a cross-lingual embedding space, this feeling is amplified. there's some work in multi-sense embedding. this should enable to capture more fine-grained embeddings embeddings.

Over the past decades, linguists have tried to inject structure into language. Similar to features in machine learning, there are properties in language. There is a vast number of linguistic features, going from phonological features, morphological and syntactic features to semantic features. To keep this analysis concise, we will give a short introduction of what features that are relevant for the topic of finding suitable word embeddings.

Lexical features include word classes, such as nouns, verbs and adjectives, which follow certain grammatical rules in western languages. Whenever we are looking to parse for such lexical features, we use the python spaCy package [**?**]. As defined in [**?**], some of the linguistic features that are available to us include but are not limited to

1. adjectives *ADJ*
2. adposition *ADP*
3. adverbs *ADV*
4. auxiliary *AUX*
5. conjunction *CONJ*
6. coordinating conjunction *CCONJ*
7. determiner *DET*
8. interjection noun *INTJ*
9. noun *NOU*
10. numeral *NUM*
11. particle *PART*

12. pronoun*PRON*
13. proper noun *PROPN*
14. punctuation *PUNCT*
15. subordinating conjunction *SCONJ*
16. symbol *SYM*
17. verb *VERB*

These features can alter as we look at different languages

3.2 Word Embeddings

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

$$w^t \mapsto (x_w, b_w) \in \mathcal{X}^{d+1} \tag{3.1}$$

where w^t is a token representation from a vocabulary $w^t \in V$ and where this token representation is transformed into a d-dimensional vector representation x_w, and a bias-term b_w that is specific to the token w^t. Whenever we will 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 $mathbbR^{d+1}$ 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 have also been conducted.

Distance Now our goal is to build an embedding space where the relationship between words are meaningful. Specifically, we want to incorporate the notion of *meaning* into these word-embeddings, which should follow following properties. 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 our distance metric to be a valid on:

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 Often in machine learning, one wants 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.

$$p\left(w|w'\right) \tag{3.2}$$

and

$$\forall w, w' : \qquad \max p\left(w|w'\right) \iff \min d(w|w') \tag{3.3}$$

Going from the distributional structure of sentences to learning distances between words. One of the early formalizations of representing the distributional structure of words through was expressed in [?], 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 $w^{(t-1)}, \dots, w^{(1)}$ as

$$\begin{aligned} p(\mathbf{w}) &= p(w^{(t)}, \dots, w^{(1)}) && (3.4) \\ &= p\left(w^{(t)} | w^{(t-1)}, \dots, w^{(1)}\right) && (3.5) \end{aligned}$$

where $\mathbf{w} = w^{(1)}, \dots, w^{(T)}$ is the sequence of words, $p(\mathbf{w})$ the probability of this sentence occurring.

Because inference for the above equation would have been computationally infeasible, the notion of *n-grams* was introduced to estimate this conditional probability, following a markov assumption.

$$p\left(w^{(t)} | w^{(t-1)}, \dots, w^{(1)}\right) \approx p\left(w^{(t)} | w^{(t-1)}, \dots, w^{(t-n+1)}\right) \tag{3.6}$$

Often, the above equation will be simplified even further using the independence of words assumption

However, because we are only interested in the distance between two words at a time (and not the full seqence) [?] simplify the above function using the assumption of marginalized independence of words amongst each other w.r.t. a target word (t) (we can do this because we marginalize over all possible combinations of target words and context words).

$$p(\mathbf{w}) = \prod_{t=1}^{T} \prod_{\Delta < t} p\left(w^{(t)} | w^{(t-\Delta)}\right) \tag{3.7}$$

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 so-called *context window* which includes all the words that R index units to the left, and R index units to the right of the word of interest $w^(t)$.

However, we do not need to only look at only the previous words . One can consider the full neighbourhood of a word. If we describe this by a loss-measure we would like to minimize, this would result in

$$\mathcal{L}(\theta; \mathbf{w}) = \prod_{t=1}^{T} \prod_{\triangle \in \mathcal{I}} p_\theta \left(w^{(t)} | w^{(t+\Delta)} \right) \tag{3.8}$$

where $\mathbf{w} = w^{(1)}, \ldots, w^{(T)}$ is the sequence of words 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 so-called *context window* which includes all the words that R index units to the left, and R index units to the right of the word of interest $w^(t)$.

If p_θ is a parametric function, one can then optimize for the most optimal $\hat{\theta} = \text{argmax}_\theta \mathcal{L}(\theta; \mathbf{w})$ using a maximum likelihood estimation approach.

In general, one does not necessarily need to interpret $\mathbf{w}$ as a sequence of word-tokens, but can also interpret $w^{(1)}, \ldots, w^{(T)}$ as *n-grams*, which are triplets of n-characters forming a token-unit. Generally, the definition of a token is open for interpretation. We will generally assume that a single word is a token unless otherwise stated.

Intuitively the above models fllw the idea expressed by [**?**] 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. A continuous-bag-of-words approach is an approach where we want to predict a target word w^t given some context words w'. However, it is also possible to follow a *skip-gram* approach. Here, we want to predict context words w'

given a target word w^t.

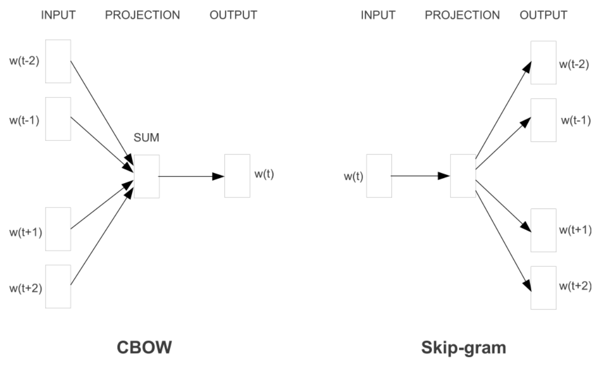


Figure 3.1: Figure taken from [?]. The CBOW architecture predicts the current word based on the context. The Skip-gram predicts surrounding words given the current word.

Using a skip-gram approach, (3.2) for example would be reformulated into

$$\mathcal{L}(\theta; \mathbf{w}) = \prod_{t=1}^{T} \prod_{\triangle \in \mathcal{I}} p_\theta \left(w^{(t+\Delta)} | w^{(t)} \right) \tag{3.9}$$

The question now is, how do we parametrize the distribution $p_\theta(w, w')$, as well as the word feature vectors x_w and b_w. The following will show a few methods of how this can be achieved. In the following methods shown, we will aim to provide a loss function that we try to minimize, and interesting properties for each method. However, we will not go into too much details, as to how the loss function is optimized, as almost all of these methods can be solved using gradient-based methods.

3.2.1 Static Word Embeddings

Here we will talk about word-embeddings where each word-token only has a single x_w. Specifically, the mapping (3.2) is not a probabilistic function with an implicit random factor, but rather a deterministic one.

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 (3.2).

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

$$\log p_\theta(w|w') = \langle \mathbf{x}_w, \mathbf{x}_{w'} \rangle + b_w + \text{ const.} \tag{3.10}$$

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 $\theta = (x_w, b_w) \in \mathbb{R}^{d+1}$

Word2Vec

During training, the above basic model comes with certain drawbacks. The distance would be minimized if all word-embeddings would collapse onto a single point. Also, 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.

One of the most prominent example of word vectors manifests itself in the work of [**?**] and [**?**]. 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 $w \in \mathbf{R}^{d+1}$ using 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

$$\mathcal{L}(\theta; \mathbf{w}) = \sum_{t=1}^{T} \sum_{\Delta \in \mathcal{I}} [\tag{3.11}$$

$$b_{w^{t+\Delta}} + \langle \mathbf{x}_{w^{(t+\Delta)}}, \mathbf{x}_{w^{(t)}} \rangle \tag{3.12}$$

$$- \log \sum_{v \in \mathcal{V}} \exp \left[\langle \mathbf{x}_v, \mathbf{x}_{w^{(t)}} \rangle + b_v \right] \tag{3.13}$$

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

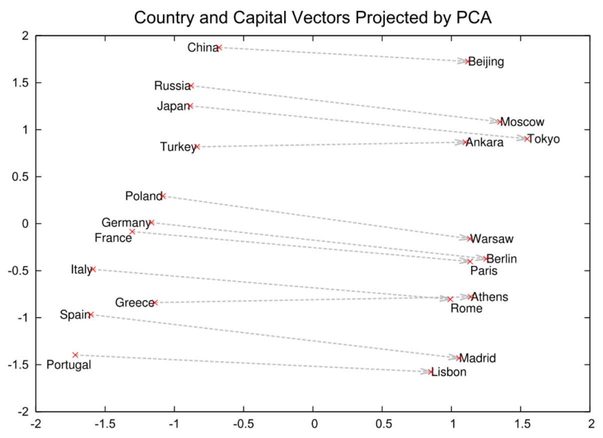


Figure 3.2: Figure taken from [**?**]. A 2-dimensional PCA projection of the 1000-dimensional skip-gram vectors of countries and their capital cities. The proposed model is able to automatically organize concepts and learn implicit relationships between them. No supervised information was provided about what a capital city means.

GloVe

For the global vectors for word representation (GloVe) [**?**], 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.

$$\mathcal{H}(\theta;\mathbf{N}) = \sum_{i,j} f\left(n_{ij}\right)\big(\underbrace{\log n_{ij}}_{\text{target}} - \underbrace{\log \tilde{p}_\theta\left(w_i|w_j\right)}_{\text{model}}\big)^2 \tag{3.14}$$

$$= \sum_{i,j} f\left(n_{ij}\right)\left(\log n_{ij} - \langle x_i, y_j\rangle\right)^2 \tag{3.15}$$

$$\tag{3.16}$$

where $f(n_{ij})$ is the weighting function which assigns a weight for each entry in the co-occurence matrix. 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 weighting which is ”too high” (there is a cutoff at some point) and small counts are considered noise and slowly cancelled out.

Gaussian Embeddings

Gaussian Embeddings have been first proposed in the context of words, although prior work has been adapted in embedding matrix-rows into a mean and standard deviation [?]

It is a continuous probabilistic relaxation of 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 and a representation and it's relationships. It can also express asymmetric relations more naturally than dot-products or cosine similarity, and enables for better-parametrized rule between decision boundaries.

Training is done using an energy function which is incorporated within a loss function that we try to minimize. The energy function describes similarity between two items. The authors propose the following energy functions to derive a Gaussian word-embedding.

$$L_m(w, c_p, c_n) = max(0, m - E_\theta(w, c_p) + E_\theta(w, c_n) \tag{3.17}$$

Here, w is the word we want to sample, c_p is a "positive" context word, i.e. a word that co-occurs with the word w, and c_n is a "negative" context word, i.e. a word that does not co-occur with the word w. Usually the negative context words is sampled randomly from the corpus. The loss function reminds of a hinge-loss in logistic regression.

The authors propose two possible ways to learn the mean and variance of the Gaussian embeddings. They argue that the empirical covariance is not the most effective method of deriving the words as Gaussian embeddings. This does not allow for inclusion between ellipsoids

Symmetric similarity: expected likelihood or probability product kernel We can use any kernel (which is symmetric by definition) to derive at an energy function. For two Gaussians $f(x)$, $g(x)$, the inner product is defined as:

$$
\begin{aligned}
E_\theta(w,c) &= \int_{x\in\mathcal{R}^d} f(x)g(x)dx && (3.18)\\
&= \int_{x\in\mathcal{R}^d} \mathcal{N}(x;\mu_w,\Sigma_w)\mathcal{N}(x;\mu_c,\Sigma_c)dx && (3.19)\\
&= \mathcal{N}(0;\mu_w-\mu_c,\Sigma_w+\Sigma_c) && (3.20)
\end{aligned}
$$

For numerical feasibility and easy of differentiation, we usually maximize the $\log E_\theta(w,c)$ for a given dataset with $w\in\mathcal{W}, c\in\mathcal{C}$.

Asymmetric divergence: KL-Divergence We can use more directional supervision to exploit directional supervision, such as a knowledge graph.

Following energy-function is optimized:

$$
\begin{aligned}
-E(w_i,c_j) &= D_{KL}(c_j\|w_i) && (3.21)\\
&= \int_{x\in\mathcal{R}^d} \mathcal{N}(x;\mu_{w_i},\Sigma_{w_i})\log\frac{\mathcal{N}(x;\mu_{c_j},\Sigma_{c_j})}{\mathcal{N}(x;\mu_{w_i},\Sigma_{w_i})}dx && (3.22)\\
&= \frac{1}{2}\left(\operatorname{tr}\left(\Sigma_i^{-1}\Sigma_j\right)+\left(\mu_i-\mu_j\right)^\top\Sigma_i^{-1}\left(\mu_i-\mu_j\right)-d-\log\frac{\det\left(\Sigma_j\right)}{\det\left(\Sigma_i\right)}\right) && (3.23)
\end{aligned}
$$

Because of the loss function, this can entail information such as ”y entails x” as a soft form of inclusion between two datasets (if KL divergence is used). If a symmetric loss function is used, then this would most likely lead to overlap (IS THIS TRUE...???)

Uncertainty calculation: In contrast to the empirical standard deviation as an uncertainty measure, we can now calculate the uncertainty of the inner product (i.e. the distribution $P(z=x^Ty)$ using the following formula

$$\mu_z = \mu_x^T \mu_y \Sigma_z = \mu_x^T \Sigma_x \mu_x + \mu_y^T \Sigma_y \mu_y + \operatorname{tr}(\Sigma_x \Sigma_y) \tag{3.24}$$

We then get an uncertainty bound, where c denotes the number of standard deviations away from the mean.

$$\mu_x^\top \mu_y \pm c\sqrt{\mu_x^\top \Sigma_x \mu_x + \mu_y^\top \Sigma_y \mu_y + \operatorname{tr}(\Sigma_x \Sigma_y)} \tag{3.25}$$

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

$$\|\mu_i\|_2 \leq C, \forall i \tag{3.26}$$

$$mI < \Sigma_i < MI \tag{3.27}$$

The method shows competitive scores to the Skip-Gram model, although usually only with minor improvements depending on the benchmark-dataset.

3.2.2 Context Embeddings

Now that we have seen a few methods that aim to capture one embedding x_w for each word-token, we not turn out attention to context embeddings.

Context embeddings are more modern applications of embeddings which

A single transformer takes as input a sequence x and outputs softmax probabilites. In the context of sentences, it is able to take as input a full sequence of words. 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 withouth any explicit conditioning

$$p(w^{(t-d)}) = p(w^{(t)}, \ldots, w^{(t-d+1)}, w^{(t-d-1)}, \ldots, w^{(1)}) \tag{3.28}$$

where $p(w^{(t-d)})$ is the probability of the target word $w^{(t-d)}$ which we want to calculate.

This modifies our initial ask of calculating $p(w, w')$ to not including only a single context word w', but rather a wider context
(i.e. $w^{(t)}, \ldots, w^{(t-d+1)}, w^{(t-d-1)}, \ldots, w^{(1)}$). Theoretically, and because we want to arrive at some distance measure, $p(w, w')$ can still be calculated by marginalizing out all the context-variables except the one of interest. Specifically, if we are interested in word w, we can marginalize out all input words except w'. Practically this is infeasible due to the high space, and as such, we make use of some other mechanisms that we will talk about in the "BERT" section below.

Because the following models not merely static word embeddings, but language models, we shall provide a short comparison between for downstream tasks between the individual language models after a short introduction on the details of each language models.

ELMo

The simplest idea of context embeddings, which take into account more than just a single word, but all tokens from a predefined context is the *Embeddings from Language Models* (ELMo) model proposed in [?].

In its underlying architecture, ELMo is using LSTMs. LSTMs are sequential recurrent neural networks .

Specifically, LSTMs [?] 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.

$$p\left(w_1, w_2, \dots, w_N\right) = \prod_{k=1}^{N} p\left(w_k | w_{k-1}, \dots, w_2, w_1\right) \tag{3.29}$$

The ELMo language model now makes use of a backward LSTM, which does not evaluate the probability of a certain token occuring with its *past* words (i.e. the context with context tokens $w^{(k)}$ with $k < t$ for a word $w^{(t)}$ whose probability we want to estimate), but rather considers only the *future context*. Specifically, this results in

$$p\left(w_1, w_2, \dots, w_N\right) = \prod_{k=1}^{N} p\left(w_k | w_{k+1}, w_{k+2}, \dots, w_N\right) \tag{3.30}$$

A bidirectional language model (biLM) combines the above two LSTM models, thus jointly maximizing the log likelihood of the forwards, as well as backward directions.

$$\sum_{k=1}^{N} \left(\log p \left(w_k | w_{k-1}, \ldots, w_1; \Theta_x, \vec{\Theta}_{LSTM}, \Theta_s \right) \right. \tag{3.31}$$

$$\left. + \log p \left(w_k | w_{k+1}, \ldots, w_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s \right) \right) \tag{3.32}$$

Because the output of a sequential neural network - which the LSTM is part of - includes $2L + 1$ output representations, we receive the following set of vectors, one for each input token, which we now consider the *context embeddings*. Each context embeddings outputs a vector representing x_k for the word token $w^{(k)}$, given its context $w_1, \ldots, w_{k-1} w_{k+1}, \ldots, w_N$.

$$R_k = \left\{ \mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, \ldots, L \right\} \tag{3.33}$$

$$= \left\{ \mathbf{h}_{k,j}^{LM} | j = 0, \ldots, L \right\} \tag{3.34}$$

Notice that one can stack the above biLM such that the output of one module is taken as input to another module. The biLM module is stacked twice ELMo. If a downstream task is trained end-to-end, the all output vectors can be stacked together and provided as input to a final downstrea-specific neural network module.

The Transformer Architecture

All of the below presented models use the transformer architecture which initially was presented in [?], which shows how far simple *attention* modules can go.

The following architectures are all based on the transformer module.

BERT

As introduced in [?], the Bidirectional Encoder Representations from Transformers model (BERT) combines the above concepts of forward and back-

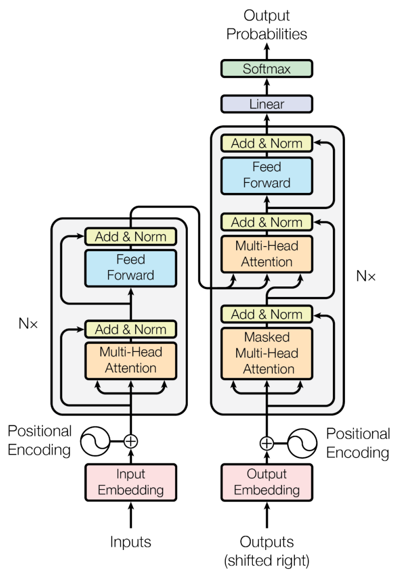


Figure 3.3: Figure taken from [?]. The transformer module architecture. The transformer encapsulates multiple attention layers.

wards sequential models with attention, as marked in the transformer architecture above.

[?] argue that the main limitations is that standard language models are unidirection, and that .

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

BERT is pre-trained on multiple gigabytes of text .

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

GPT and GPT-2

First proposed in [?] and further extended in [?].

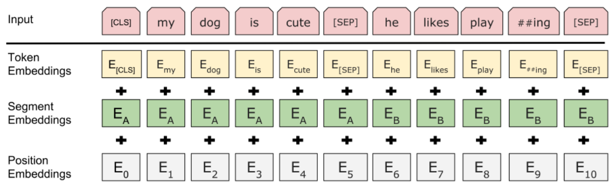

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.4: Figure taken from [?]. 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.

3.2.3 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:

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

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/imgarylai/b
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/2019/
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.

3.3 Resources and Datasets

WordNet

SemCor dataset

News dataset

GLUE benchmark dataset

Chapter 4

Related Work

4.1 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

4.2 Metric Learning and Disentanglement

4.3 Zero shot and One shot learning

4.4 Clustering Algorithms

4.5 Applications of word vector

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 -

Chapter 5

Analysing the current state of the art

Upadhyay et al. argue that the choice of data is more important than the actual algorithm.

Definitely also look into this, [Analyzing the Limitations of Cross-lingual Word Embedding Mappings] seems to be an analysis of the difficulties etc.

5.1 On the Linear Separability of meaning within sampled BERT vectors

5.1.1 Motivation

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.

5.1.2 Experiment setup

Let us regard a word w in sentence s is indexed by i. When passed through the BERT model, the word w produces an embedding x_w. When we sample

BERT embeddings for a single word w for $n = 500$ sentences.

Let us restrict the choice of words w on polysemous and ambigious words which carry multiple meanings. Specifically, w is chosen in such a way that each meaning has more than 30 samples in each class when chosen from within the SemCor dataset.

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

When we sample Specifically, the experiment setup looks as follows.

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

```
**Input:** A target word $w_{\text{target}}$; 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 $w_{\text{target}}$, 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 a binary

First of all, we sample $n = 500$ sentences from the news corpus for a target word $\hat{w}$, which has index i in sentence s.

Each sentence, we pass through the

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

We sample $n = 500$ vectors from BERT. This is the vector at the output embedding layer of BERT, which has the same location as the input to the BERT model.

We apply t-fold cross validation, and measure the mean accuracy as well as the standard deviation of the accuracy. We also note down the variance kept after projecting PCA on the lower dimensionality k. This is the sum of the first k largest normalized eigenvalues.

5.1.3 Results

The number of words we can use to realiably estimate a planar separation of semantics within the sampled BERT vectors is small.

We run the above experiment for a set of different k to build up an intuition of how well different dimensionalites still capture the meaning in different locations of the vector space.

Table 5.1: 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 5.2: 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 5.3: 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 5.4: 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.

5.2 On the Clusterability of meaning within sampled BERT vectors

5.2.1 Motivation

5.2.2 Experiment setup

The general algorithm to cluster the dataset is shown below.

```
Algorithm 2: Checks sampled BERT vectors for clusters by meaning
Input:  $w_{\text{target}}$: 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 $w_{\text{target}}$, 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 $k$;
$model \leftarrow ClusterAlgorithm(\mathbf{X})$ ;
$\hat{\mathbf{y}} \leftarrow model.predict(\mathbf{X})$ ;
$score \leftarrow AdjustedRandomIndex(\hat{\mathbf{y}}, \mathbf{y})$ ;
return $score, \hat{\mathbf{y}}$;
```

However, because some simple algor.

We had a few constraints. When clustering, we are not given the number of cluster to be found. Thus, the algorithm must solve the multi-modal detection problem intrinsically, which is considered a hard problem in machine learning, as it falls in the same category as global probability density estimation. Also, because we use the lexical distinction of semantics as defined in

WordNet, our algorithm needs to adapt to the granularity that was defined by linguists.

Because we want to evaluate how well our clustering method can mimick the semantic definitions in WordNet, but also generalize to unseen words, we train on an unsupervised dataset. We then test our resulting clustering on a labeled datasetet using the random adjusted index [?], [?].

The adjusted random index calculates the overlap and as such the similarity between two clustering assignments. Specifically,

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \tag{5.1}$$

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.

where 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. The adjusted random index is adjusted for chance, by taking all possible permutations of the labels that the clustering algorithm can have.

Although we will not go into too much detail with the algorithm descriptions here, we will

The general algorithm to cluster the dataset is shown below.

1. s

model	ARI)
Affinity Propagation	0.001
DBScan	0.000
HDBScan	0.328
MeanShift	0.004
Optics	0.000

5.2.3 Results

Repeating the experiment with 1000 datapoint does not change the results by much.

model	ARI)
Affinity Propagation	0.000
DBScan	0.139
HDBScan	0.271
MeanShift	0.005
Optics	0.070

Repeating the experiment with 1000 items and projecting PCA to 100 results in

model	ARI)
Affinity Propagation	0.000
DBScan	0.215
HDBScan	0.359
MeanShift	0.003
Optics	0.000

We see that including more dimensions make the clustering better w.r.t. the adjusted random score.

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 [?].

We now apply forceful clustering using Bayesian Optimization.

100 latent dimensions, 500 samples

model	ARI)
Affinity Propagation	0.168
Chinese Whispers	0.298
DBScan	0.201
HDBScan	0.242
MeanShift	0.167
Optics	0.167

100 components, 1000 samples

model	ARI)
Affinity Propagation	0.170
Chinese Whispers	0.249
DBScan	0.260
HDBScan	0.234
MeanShift	0.167
Optics	0.197

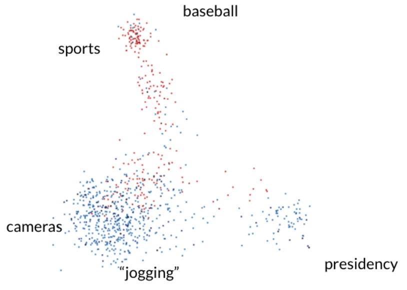
Projecting to lower dimensions increases accuracy strongly. 20 components, 1000 samples

model	ARI)
Affinity Propagation	0.165
Chinese Whispers	0.349
DBScan	0.167
HDBScan	0.273
MeanShift	0.226
Optics	0.167

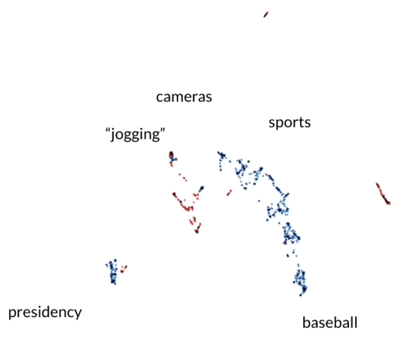
Projecting to lower dimensions increases accuracy strongly. and now we add a SEP token 20 components, 1000 samples

We now analyse the embeddings manually

model	ARI
Affinity Propagation	0.316
Chinese Whispers	0.457
DBScan	0.170
HDBScan	0.298
MeanShift	0.251
Optics	0.167

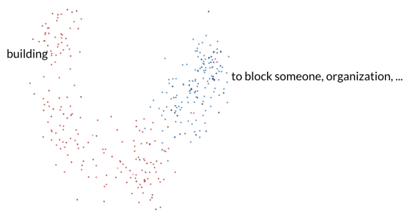


(a) 1a

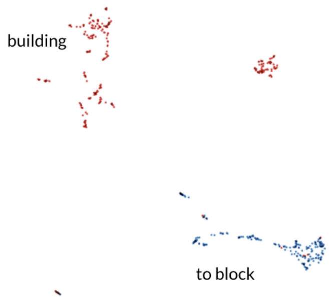


(b) 1b

Figure 5.1: plots of....

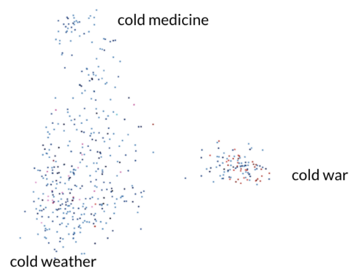


(a) 1a

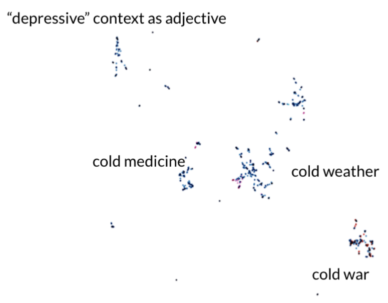


(b) 1b

Figure 5.2: plots of....



(a) 1a



(b) 1b

Figure 5.3: plots of....

5.3 Correlation between Part of Speech and Context within BERT

5.3.1 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.

5.3.2 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 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.

5.3.3 Results

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

Chapter 6

Our Method

We introduce *split-words*, for which we will be generating more detailed embeddings. 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.

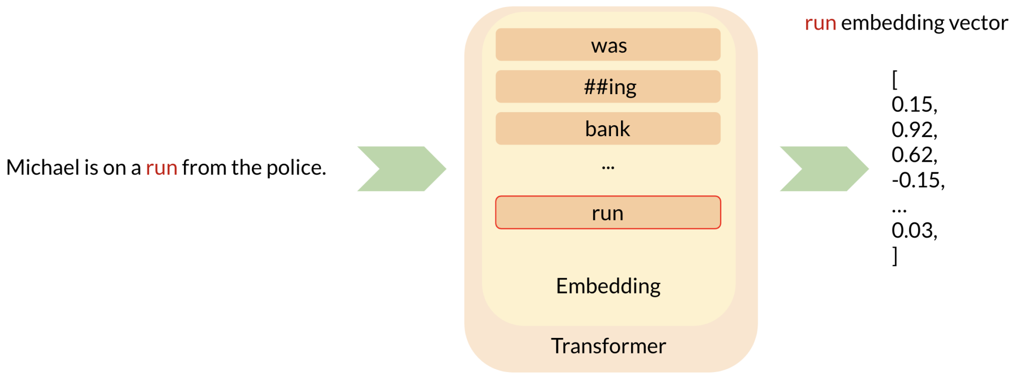


Figure 6.1

6.0.1 BERnie PoS

Motivation

Experiment setup

BERnie PoS introduces one new embedding vector for each possible PoS configuration of the split-words.

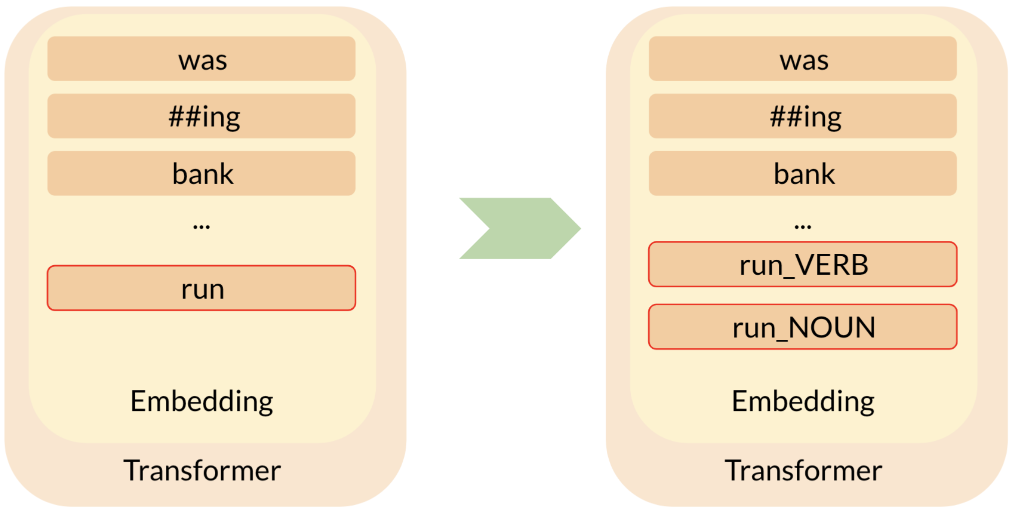


Figure 6.2

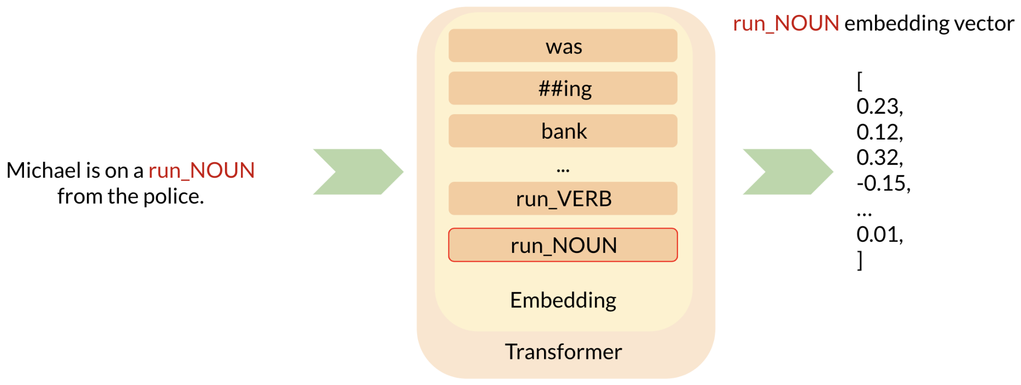


Figure 6.3

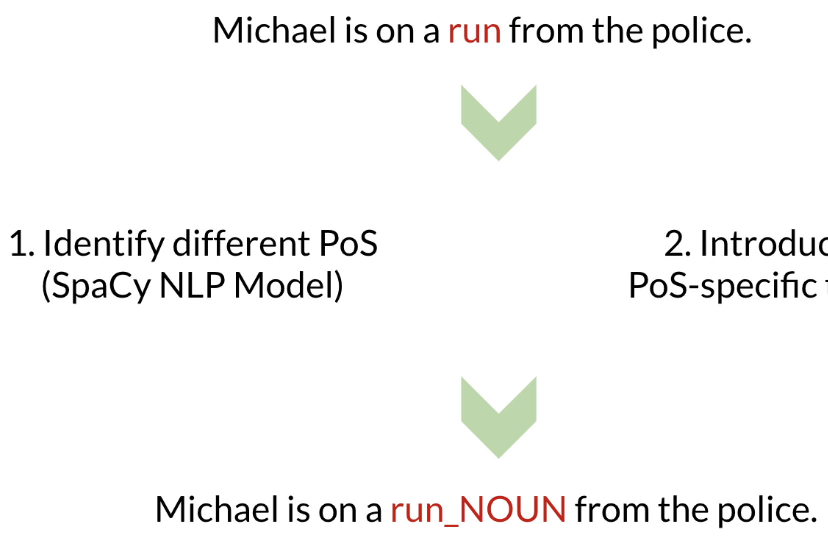

Michael is on a run from the police.
1. Identify different PoS (SpaCy NLP Model)
2. Introduce a PoS-specific token
Michael is on a run_NOUN from the police.

Figure 6.4

6.0.2 BERnie Meaning

Motivation

Experiment setup

6.0.3 BERnie Meaning with additional pre-training

Motivation

Experiment setup

6.0.4 Compressing the non-lexical out

Motivation

Experiment setup

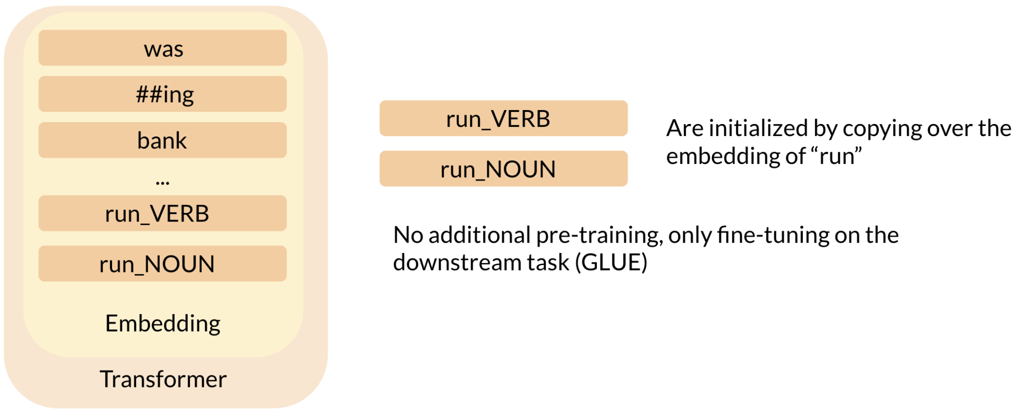

was
##ing
bank
...
run_VERB
run_NOUN
Embedding
Transformer
run_VERB
run_NOUN
Are initialized by copying over the embedding of "run"
No additional pre-training, only fine-tuning on the downstream task (GLUE)

Figure 6.5

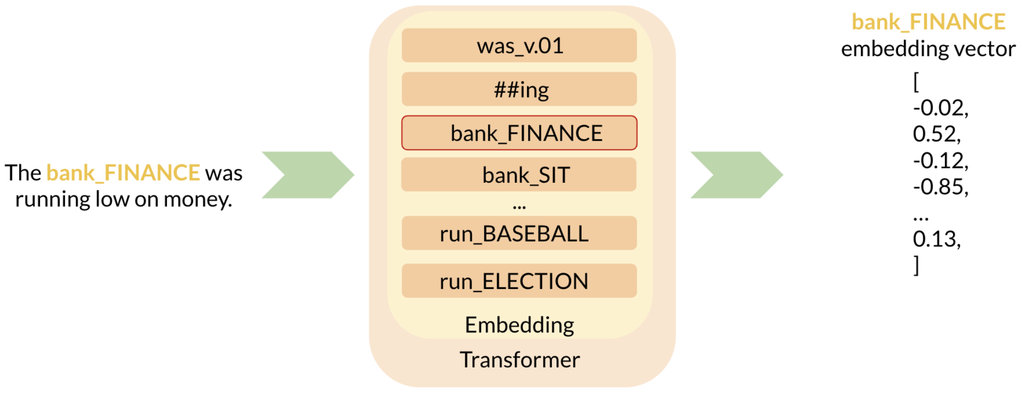


Figure 6.6

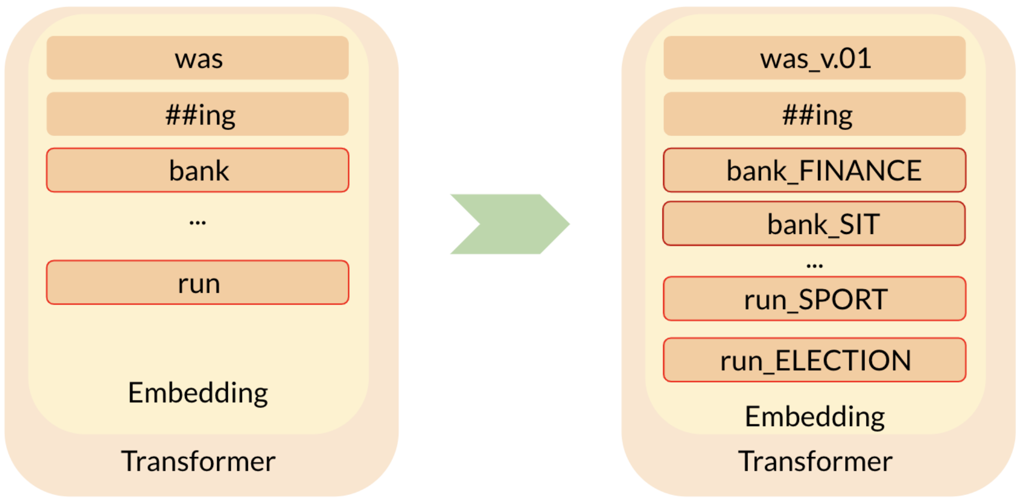


Figure 6.7

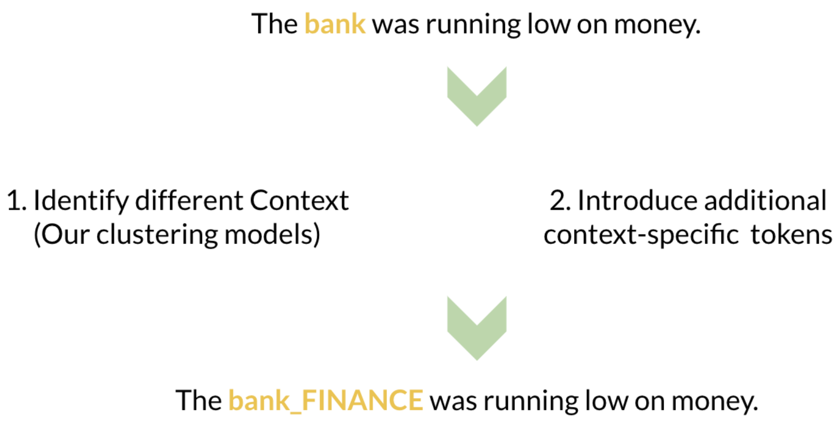

The bank was running low on money.
1. Identify different Context (Our clustering models)
2. Introduce additional context-specific tokens
The bank_FINANCE was running low on money.

Figure 6.8

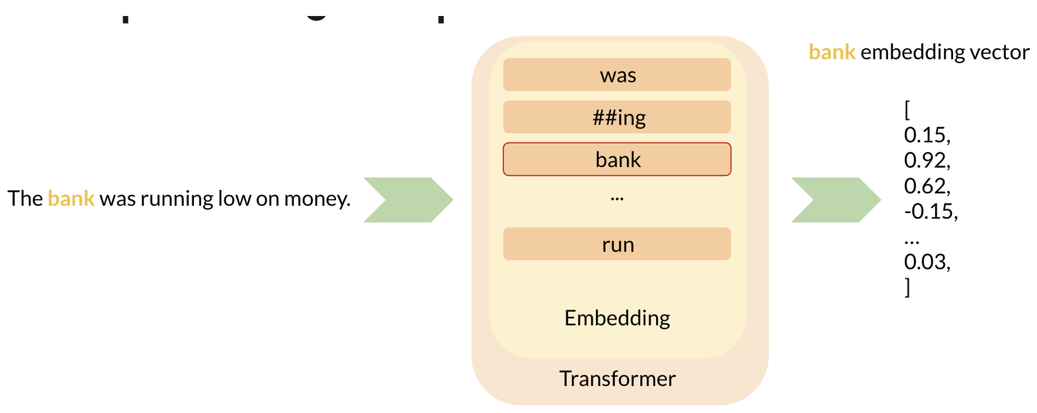

The bank was running low on money.
was
##ing
bank
...
run
Embedding
Transformer
bank embedding vector
[
0.15,
0.92,
0.62,
-0.15,
...
0.03,
]

Figure 6.9

Chapter 7

Further Work

Can be used for more unstructured data, like graphs.

Chapter 8

Evaluation

Chapter 9

Conclusion