UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Extracting Text Representation from Pretrained Language Models via Edit Distance-based Loss Function

Author:
Yurii ANTENTYK

Supervisor:
Serhii HAVRYLOV

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2024

Declaration of Authorship

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Extracting Text Representation from Pretrained Language Models via Edit Distance-based Loss Function

by Yurii ANTENTYK

Abstract

Transformers have proven themselves as versatile architecture for a wide range of NLP tasks. Because of unsupervised pre-training on large text corpus and a drastic increase in a number of parameters, both encoder and decoder architectures were able to generalize well enough to show emergent abilities and achieve SOTA results in many downstream tasks by prompting or fine-tuning.

This makes pre-trained language models tempting to be used for sentence embedding, as the meaningful fixed-vector representation of text is crucial for good performance of such tasks as text similarity, semantic search, etc. The topic has been extensively explored, and the most successful approaches can be viewed as different variations of extracting sentence representations from the hidden states of language models, which have been fine-tuned using domain-specific datasets.

On the other hand, other works try to extract the embedding vector from the pre-trained language model without altering its parameters, which opens the way to turn the pre-trained language model into an embedding model without fine-tuning. This is done by optimising the latent reparameterised sentence space, which is then used as additional context while decoding the original sentence. While showing promising recoverability results, this approach has been shown to suffer from exposure bias, a discrepancy between the distribution of sequences that were observed and generated by the model.

This work aims to study the way to condition a pre-trained language model with a latent variable as well as to mitigate the exposure bias by incorporating Optimal Completion Distillation loss, an alternative to Maximum Likelihood Estimation, which minimises the edit distance between a sampled text from the model and a ground truth sentences.

Acknowledgements

I would like to thank my dear mother for supporting my through this journey. Special gratitude goes to my supervisor Serhii Havrylov, for his guidance, advices and unfailing help in difficult moments.

Contents

List of Figures

List of Tables

List of Abbreviations

NLP	**N**atural **L**anguage **P**rocessing
RNN	**R**ecurrent **N**eural **N**etwork
LSTM language model	**L**ong **s**hort-**t**erm memory language model
MLE	**M**aximum **L**ikelihood **E**stimation
ELMo	**E**mbeddings from **L**anguage **M**odels
BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
MLP	**m**ulti**l**ayer **p**erceptron
SNLi dataset	**S**tanford **N**atural **L**anguage **i**nference dataset
SBERT	**S**entence BERT
BEIR	**Be**nchmarking-**IR**
OCD	**O**ptimal **C**ompletion **D**istillation
RL	**R**einforcement **L**earning
OC policy	**O**ptimal **C**ompletion policy
KL divergence	**K**ullback-**L**eiber divergence
BOS token	**b**eginning **o**f **s**equence token
EOS token	**e**nd **o**f **s**equence token

Chapter 1

Introduction

1.1 Embeddings in Pre-Transformer Era

The technique of encoding categorical entities into fixed-vector embeddings is still used in various domains, such as recommender systems, graph representation learning, speech processing, natural language processing, etc. In Natural Language Processing (NLP), in particular, the problem of learning highly informative dense embeddings for representing words / sentences / documents has been the subject of long-term research with many architectures and approaches. While early methods (word2vec Mikolov et al., 2013, glove Brochier, Guille, and Velcin, 2019) directly optimize similarities/dissimilarities between word embeddings based on their surrounding context, more recent approaches like variational autoencoders Kingma and Welling, 2014 explore the idea of latent variable modelling, which assumes the existence of a latent space, from which the observed data might be fully reconstructed. Together with the introduction of recurrent neural networks, this has laid the foundation of sequence-to-sequence architecture Sutskever, Vinyals, and Le, 2014, which introduced an encoder-decoder architecture for learning a mapping between sequences of arbitrary lengths . The original paper tackled the problem of machine translation, while others have used the same architecture for language modelling in a semi-supervised fashion, achieving comparable performance in text classification Dai and Le, 2015 and sentence generation tasks Bowman et al., 2016.

Because Recurrent Neural Networks (RNNs) were known to suffer from a vanishing gradient problem Pascanu, Mikolov, and Bengio, 2013, they could not utilize the bigger context when training, limiting their ability to capture long-term dependencies Bengio, Simard, and Frasconi, 1994 and deal with longer sequences Cho et al., 2014. The introduction of the attention mechanism Bahdanau, Cho, and Bengio, 2015 has allowed the decoder to access the outputs of the encoder at every timestamp, thus reducing the gradient path length and dependency on the latent variable to model the output sequence.

1.2 Learning Embeddings using Transformers

The idea of using attention mechanism for temporal dynamics modelling was further developed into a transformer architecture Vaswani et al., 2017, where self and cross-attention are one of the few mechanics used to capture the semantics and temporal structure of the input data. Despite implicit temporal dynamics modelling, transformer-based architectures like T5 Raffel et al., 2020, BERT Devlin et al., 2019, GPT Brown et al., 2020 and their derivatives still hold state-of-the-art results on multiple information retrieval downstream tasks, such as zero-shot text search, question answering, fact-checking, etc.

This success owed much to the unsupervised pretraining and fine-tuning paradigm. The particular implementation of the transformer model was first trained to do language modelling on a large corpus of unlabeled data. These checkpoints were made public, which simplified the process of replicating the experiments and adapting the pre-trained model to solve the downstream task in an end-to-end fashion. The process of fine-tuning takes much less time and computational resources compared to training from scratch, with some works claiming to achieve comparable results by fine-tuning only the biases of the model Zaken, Goldberg, and Ravfogel, 2022.

Even though input sequence embedding is not explicitly modelled by transformer architecture, learning fixed-vector representations is still important for a few NLP downstream tasks, such as semantic search, semantic text similarity and natural language inference. Embeddings also lie at the core of vector databases, which enable search for the relevant documents based on the semantics of the query, as opposed to a traditional keyword-based search. This is made possible through techniques like approximate nearest neighbours or locality-sensitive hashing, which assume that a meaningful fixed-vector representation can be derived from the document or a query.

With previous works, attempts have been made to tackle the problem of learning distributed representation for text using transformers in different ways. This included solving semantic search via prompting (cross-encoder setup in Muennighoff, 2022), fine-tuning (bi-encoder setup in Muennighoff, 2022, Reimers and Gurevych, 2019) or altering the cross-attention mechanics of the transformer to learn sentence embeddings explicitly Wang, Reimers, and Gurevych, 2021. However, with the number of parameters of transformer models reaching hundreds of billions Brown et al., 2020, this poses challenges to the fine-tuning approaches at this scale.

Additionally, as the computational capabilities of edge devices increase, employing embeddings at the edge presents new hurdles, such as the need to research the methods for compressing Pansare et al., 2022 or binarization Shen et al., 2019 to save memory space and computational resources (with dedicated binarization approaches needed to achieve comparable accuracy as opposed to binarizing the existing floating-point models/embeddings).

With the increasing number of parameters, the researchers observed the emergent abilities in Large Language Models, such as in-context learning, instruction following, and step-by-step reasoning Zhao et al., 2023, meaning that these models have been able to capture the structure of the human language well enough to successfully generalize in out-of-domain observations. This makes the idea of finding a way to extract the sequence embedding without altering the pre-trained model appealing and worth researching. This idea was explored with the Long short-term memory (LSTM) language model Subramani, Bowman, and Cho, 2019. The authors showed the existence of the reparameterized sentence space and presented the way to learn the sentence embedding without altering language model parameters, showing promising recoverability results. Later, a similar approach was explored Subramani and Suresh, 2020 on transformer-based language models. However, despite the almost perfect recoverability of the sentences, the associated representations are limited in use. For example, they fall short of lexical methods tailored to semantic similarity tasks and methods that finetune on natural language inference datasets Subramani, Suresh, and Peters, 2022. We assume that this issue is due to the fact that transformer-based language models are extremely powerful decoders and there is no incentive for them to use the content of the vector representation when the sentence context can be used instead. Hence, using the loss proposed in this paper that creates this incentive should result in better representations.

The rest of the thesis is organised in the following manner: chapter 2 introduces the objective of language modelling, as well as detailed review of recurrent and transformer based architectures with the focus on embedding strategies. In chapter 3 we formulate the list of research questions, introduce the chosen approach of extraction of the embeddings from a pre-trained language model as well as the objective of optimal completion distillation. In 4 we describe the architecture of the language model we will be experimenting with as well as the technical details of encorporating the embedding vector into the model. In chapter 5 we outline the setup for the experiments, introduce the recoverability metrics as well as provide results of the experiments. In 6 we analyze and discuss the results of the experiments. In 7 we suggest the directions for future work on the topic. Finally, in 8 we highlight our contributions and summarize the findings, presented in the thesis.

Chapter 2

Related Works

2.1 Language Modelling Setup

Language modelling task aims to assign high probabilities to the sentences from the given dataset $\mathcal{D}$. Usually it is done by optimizing (eq. 2.1) likelihood function of the dataset $\mathcal{D}$ given parametric model $p_\theta(x)$, where $x = (x_1, x_2, \dots x_{|x|})$ is a sample from the dataset $\mathcal{D}$ and $|x|$ is the length of a sentence. Such approach to learning is known as a Maximum Likelihood Estimation (MLE).

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \, \mathbb{E}_{x \sim \mathcal{D}} \log p_\theta(x) \tag{2.1}$$

Parametric model $p_\theta(x)$ is usually decomposed using general product rule in a way that conditions each token from the sentence by its previous context (eq. 2.2).

$$p_\theta(x) = \prod_{t=1}^{|x|} p_\theta(x_t | x_{<t}) \tag{2.2}$$

In practice, after pre-training a language model different methods of sampling may be used. One of the most popular methods is a beam search Freitag and Al-Onaizan, 2017. Because greedily picking the most likely continuation at each timestamp might not be optimal, beam search keeps k most likely branches at every timestamp and tries to expand each of them, leaving the k most probable branches at the next step. The choice of k is a compromise between speed and correctness of choosing the most likely continuation sequence (with $k = 1$ corresponding to greedily picking the next most probable token and $k = \infty$ corresponding to an exhaustive search of all possible continuations).

2.2 Getting Embeddings from Language Models

With multiple works that used RNNs for language modelling in the aforementioned fashion, Embeddings from Language Models (ELMo, Peters et al., 2018) was the first to propose a way of forming sequence embeddings from the trained language model, as opposed to contextless embedding per word like word2vec of glove. The authors trained forward and backward multi-layer LSTM language models using a large text corpus and suggested several options for using the hidden states of the LSTMs as embedding, with the simplest one being the concatenation of the top-layer hidden states of the forward and backward LSTMs. The fine-tuned ELMo has set state-of-the-art results for many downstream tasks at the time, such as question answering, textual entailment, semantic role labelling, etc.

Like ELMo, Bidirectional Encoder Representations from Transformers (BERT, Devlin et al., 2019) builds upon the ideas of language modelling and pre-training using an encoder-only transformer. Because encoder self-attention is not masked, it allows the model to look into future timestamps, which essentially breaks the idea of language modelling. To address this, the authors suggest the idea of a masked language model objective, which adds noise to the input sentence and trains the model to restore the original token, not including the loss between unchanged tokens. Another contribution of BERT is adding two special tokens: [CLS] and [SEP]. [CLS] is prepended to each input sequence during pre-training, allowing the corresponding output to be used for a classification downstream task, such as entailment or sentiment analysis. [SEP] is used as a separator between two sentences that are fed into BERT during pre-training. This allows using of BERT for downstream tasks that use text pairs because apart from the masked language model, BERT was also pre-trained on the next sentence prediction problem, which means that it can also be used as a cross-encoder. Similarly to ELMo, the output of the hidden states of the transformer at each position can be used as embeddings. The authors have compared the feature-based approach with fine-tuning on the named entity recognition task, with the concatenation of the last four hidden layers as features achieving results comparable to fine-tuning.

2.3 Modelling Embeddings for Semantic Text Similarity

Other works aim to learn meaningful sentence embeddings explicitly in a supervised fashion by solving semantic text similarity task, the objective of which is to determine the degree of similarity (in either categorical or continuous manner) between pairs of input sentences. InferSent Conneau et al., 2017 uses the shared sentence encoder to produce fixed-size embeddings for the pair of sentences, which are then combined and used as input to an multilayer perceptron (MLP) to train on the SNLI dataset Bowman et al., 2015, which consists of 570k sentence pairs, manually annotated into three categories: entailment, contradiction, neutral. The authors experiment with multiple encoder architectures and embedding strategies, with the best one being a bidirectional LSTM with embedding obtained via max-pooling between hidden states at different timestamps. Glove word embeddings are used as input features into the BiLSTM. The authors also evaluate the trained model on downstream tasks in a feature-based approach, where sentence embeddings are used as features without fine-tuning the original model, with new state-of-the-art results for some datasets at the time.

Sentence BERT (SBERT Reimers and Gurevych, 2019) addresses the problem of computational inefficiency of BERT when it comes to clustering or nearest-neighbour search since it is primarily used as a cross-encoder for a sentence pair. To account for this, the authors fine-tune BERT in a siamese bi-encoder setup, with mean pooling of the hidden states used as embeddings. Authors report SBERT embeddings being better than the competitors in zero-shot setup across multiple semantic text similarity datasets, with averaged BERT embeddings often being worse than average glove embeddings.

2.4 Unsupervised Approaches

With the abundance of unlabeled textual data, efforts have been made to learn sentence embedding without supervision. For example, TSDAE Wang, Reimers, and Gurevych, 2021 uses an autoencoder approach with a transformer encoder and decoder. It modifies the key and value of the cross-attention mechanism to only access the sentence embedding. Together with introducing noise into the input and expecting the decoder to produce the original sequence, this puts the bottleneck on the embedding and forces the encoder to produce an informative embedding vector.

Similarly to SBERT, cpt-text Neelakantan et al., 2022 uses an encoder transformer in a bi-encoder setup, optimising the cosine similarity between embeddings of pairs of sentences. The model is trained with in-batch negatives, with the pair of consecutive sentences in the unlabeled text corpus considered as a positive and two random sentences considered as a negative match. The last hidden state of the encoder transformer is used as a sentence embedding.

2.5 Tackling Semantic Search

Transformer Language Models have also been used in the problem of semantic search, which is stated as follows: given the set of documents d and a query q, determine the sorted set $d*$ of documents that are relevant to the given query (with most relevant document coming first). Using the neural networks, the problem is solved in a supervised fashion, with the datasets containing the pairs of documents and queries with different levels of annotation (e.g. relevant vs. non-relevant, highly relevant vs partially relevant vs non-relevant, or even an integer score). Hence, the training can be done in a classification or regression setup.

Because of high computational cost, cross-encoder transformer models often cannot be used to compare the query with every document in the dataset. Despite that, transformers are used as a part of the multi-stage retrieval system, with a traditional bag of words model like BM25 Robertson and Zaragoza, 2009 selecting the initial set of relevant documents and a transformer-based model performing re-ranking of the selected subset of documents w.r.t. the given query. The combination of BM25 and MiniLM Wang et al., 2020, a knowledge-distilled counterpart of BERT, is reported to have the best zero-shot performance across most of the datasets in Benchmarking-IR (BEIR Thakur et al., 2021) at the time of submission.

The limitation of using the transformer language model is addressed in SGPT Muennighoff, 2022. The authors fine-tune the GPT-based encoder in an end-to-end fashion to solve the semantic search problem. In a bi-encoder setup, documents and query are encoded in a siamese fashion using a 5.8B parameters GPT encoder. Embeddings are calculated via a position-weighted mean pooling. The model is fine-tuned in a supervised manner using contrastive learning with in-batch negatives. Only biases of the GPT model are being fine-tuned. The model set new state-of-the-art results on five subsets of the BEIR dataset compared to other approaches.

2.6 Learning Embeddings Directly from the Model

Contrary to the approaches mentioned above, other works Subramani, Bowman, and Cho, 2019; Subramani, Suresh, and Peters, 2022; Subramani and Suresh, 2020 study the problem of existence of the sequence representation space, the vectors from which may be used to condition the pre-trained language model to recover the original sequence without additional fine-tuning or architecture changes, with the hypothesis being that the information that is needed to generate the sequence is already encoded in the language model parameters. Doing this may allow turning a pre-trained language model into a universal embedding model without fine-tuning. This is done by incorporating the projected context vector into the hidden state at every timestamp of the LSTM language model Subramani, Bowman, and Cho, 2019 or by altering the embedding of the first token of the decoder transformer model Subramani, Suresh, and Peters, 2022. The context vector is then optimized to maximize the conditional log-likehood of the language model reconstructing the original input, given the context. This approach shows promising recoverability results, however Subramani, Bowman, and Cho, 2019 reports difficulties of recovering larger sequences, with the recoverability lacking at the point when the language model begins generating incorrect words.

Chapter 3

Methodology

In this chapter we identify the gaps in the existing literature and formulate a list of research questions. We then describe our chosen method for extracting a vector representation from a fixed, pre-trained language model (steering) as well as optimal completion distillation (OCD) objective function.

3.1 Gap Analysis

3.1.1 Averaging May Produce Bad Results

Given a pre-trained language model (either LSTM or transformer), there is no definitive answer on how to use its hidden layer activations to form the resulting embedding for the input sentence. The exact combination of hidden layers is not the same across different downstream tasks and needs thorough validation and fine-tuning. Without addressing this point, the quality may degrade significantly. For example, embeddings retrieved from averaging the activations of the output layer of BERT perform worse than Glove embeddings on semantic text similarity benchmarks as reported in Reimers and Gurevych, 2019. A similar problem is present in the ELMo language model Peters et al., 2018, where the authors recommend learning a separate set of layer weight coefficients for a particular downstream task for best performance. This approach can no longer be regarded as universal, with the resulting embeddings being different for each downstream task. Also, there seems to be no clear mathematical ground, which justifies the strategy of forming the resulting embeddings.

3.1.2 Limited use of Transformers in Some Downstream Tasks

Despite transformers bringing advancements to multiple downstream tasks, they are still limited in use in case an explicit embedding vector for the input sentence is required (e.g. semantic search). Current state-of-the-art approaches either alter the architecture to model embeddings explicitly Wang, Reimers, and Gurevych, 2021, use transformers only for reranking Muennighoff, 2022 or specifically train the model for embedding purposes Cer et al., 2018. These approaches don't help in case we want to harness the knowledge of pre-trained large language models. Hence, the procedure of extracting the embeddings without altering the model's weights and architecture sounds appealing, useful and worth researching.

3.1.3 Learning Embeddings Directly from the Model is Experimental and Challenging

Despite its advantages and mathematical ground, the approach for learning sentence embedding without fine-tuning the model described in is rather experimental and not widely used. In particular, in Subramani, Bowman, and Cho, 2019 it was presented as a proof of concept, without an extensive evaluation on multiple benchmarks. Without this, further work is needed to estimate whether it can compete with traditional approaches in terms of computational complexity and performance. In addition to this, the approach has challenges with learning embeddings for longer sequences as reported in Subramani, Bowman, and Cho, 2019, which may become a problem when trying to apply it in real-life scenarios. Furthermore, additional research is needed to estimate whether this approach easily transfers to other model architectures.

3.2 Research Questions

Given the research gaps in the existing literature, we formulate the next list of questions, that will be studied in this work:

1. *How well/easy does the approach of learning embeddings directly from the pre-trained language model work?* Since there is no code associated with Subramani, Bowman, and Cho, 2019, it's important to know whether the details outlined in the paper is enough to reproduce it.

2. *What is the quality of the resulting embeddings? How does it compare to traditional approaches?* To understand the potential of the approach, further evaluation is needed.

3. *How can the problem of reconstructing longer sequences may be addressed?* Since the main advantage of transformer language model is to be able to tackle longer sequences, it's crucial to estimate the boundaries of the sequence length, which can be successfully reconstructed using the resulting embedding as well as suggest a potential solution to this problem.

3.3 Steering

In NLP, a large volume of tasks involve usage of a fixed-dimensional vector representation. Available approaches usually encode the source sequence into a fixed-length vector using a special encoder that has to be learnt from the data. Instead of learning an encoder to obtain a representation, we consider a question whether it's possible to extract a vector directly from a pre-trained language model without any additional data or modification to the architecture of the language model.

Namely, we define a sentence representation vector of a given sentence x as $z_\theta(x) \in \mathbb{R}^d$, where θ denotes parameters of a pre-trained language model and d corresponds to the hidden dimensionality of the language model.

This vector is obtained by maximizing the probability of a given sentence x, conditioned of $z_\theta(x)$, while keeping the language model architecture unchanged:

$$z_\theta(x) = \underset{z \in \mathbb{R}^d}{\operatorname{argmax}} \log p_\theta(x|z) \tag{3.1}$$

Since the weights of the pre-trained language model remain frozen, the technique of incorporating the conditioning on z depends on a particular model architecture. For example, in Subramani, Bowman, and Cho, 2019 authors add $z_\theta(x)$ to the hidden states of the LSTM language model on every timestamp (see 3.1). During inference, $z_\theta(x)$ is also used as an initial hidden state to steer the model to generate the desired sentence x. Similar idea is used in Subramani and Suresh, 2020, where authors experimented with adding $z_\theta(x)$ to the hidden states of encoder transformer architecture at different locations.

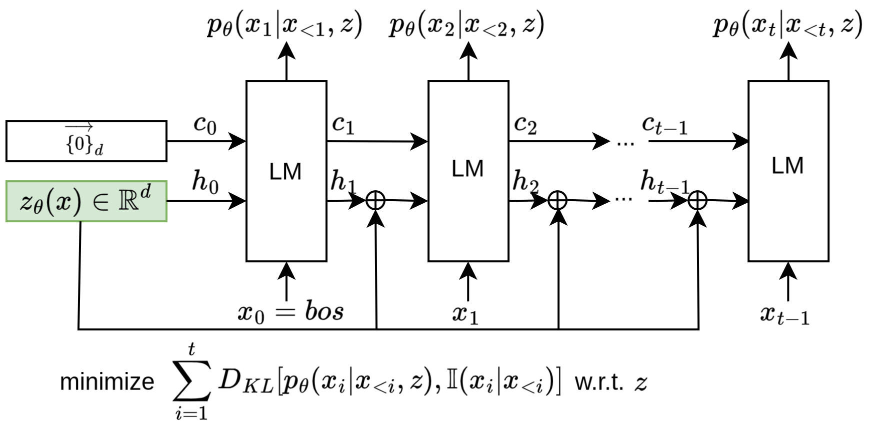

$p_\theta(x_1|x_{<1}, z)$
$p_\theta(x_2|x_{<2}, z)$
$p_\theta(x_t|x_{<t}, z)$
$\overrightarrow{\{0\}}_d$
$z_\theta(x) \in \mathbb{R}^d$
c_0
h_0
LM
c_1
h_1
LM
c_2
h_2
c_{t-1}
h_{t-1}
LM
$x_0 = bos$
x_1
x_{t-1}
minimize $\sum_{i=1}^{t} D_{KL}[p_\theta(x_i|x_{<i}, z), \mathbb{I}(x_i|x_{<i})]$ w.r.t. z

FIGURE 3.1: Steering approach described in Subramani, Bowman, and Cho, 2019. Embedding vector z is added to hidden state across all timestamps to incorporate the conditioning of the input sentence x on the embedding vector z. z is then trained to maximize the probability of input sentence x. (*bos* stands for *b*egining *o*f *s*equence)

The aforementioned method of learning embeddings comes with its own set of shortcoming and advantages, with the biggest upside being its independence from requiring any additional data to find the embedding for a particular sentence. It also eliminates the need for finetuning the language model for different downstream tasks and allows for the reuse of the single embedding across them. It's important to acknowledge that the method also has its limitations. In particular, each sentence requires an optimization problem (3.1) to be solved, which may require significant amount of computational resources. It's also worth noting that the problem of determining the optimal embedding vector for different sentences is embarrassingly parallel, with computation possible on a distributed system with very little intercommunication.

3.4 Exposure Bias

Despite the advantages, the quality of the resulting embeddings still need further assessment. For example, in Subramani, Suresh, and Peters, 2022 the embeddings produced by steering a pre-trained GPT2 model were reported to outperform Glove or the ones that are obtained by taking the activations from the last layer of GPT2. On the other hand, in Subramani, Bowman, and Cho, 2019 the steering embeddings obtained from LSTM language model are reported to have difficulties with recovering longer sentences, which still leaves room for improvement.

We hypothesize that such subpar vectors are produced due to the phenomenon which is known as *exposure bias*. Exposure bias is a problem that occurs when the model is trained with ground-truth data, but has to generate its own data during inference. This can lead to errors or degradation in the quality of the generated text. A powerful autoregressive decoder has sufficient capacity to achieve good data likelihood without using provided vector representation given the ground-truth context, thus not all available information in the sentence is compressed into the vector form when optimizing eq (3.1).

The problem is closely related to the issue of distributional shift in behavior cloning, where the distribution of the observed states/actions produced by running a policy p_θ (sentences, generated from the pre-trained model) differs from an expert one, which the policy was initially trained on (sentences from the train dataset). In particular, LSTM steering embeddings in Subramani, Bowman, and Cho, 2019 are reported to fail in achieving further recovery of the target sentence, once an error arises at a specific position during generation.

3.5 Cross Entropy and Optimal Completion Distillation

The reason exposure bias exists is partially due to the way maximum likelihood is being optimized when training the language model (solving eq. (2.1)) as well as training embeddings via steering (solving eq. (3.1)). MLE is equivalent to cross-entropy loss between the model's probability distribution and the ground truth data. Let's consider the language modelling setup described in 2.1. For an input sentence x and given timestamp t, we minimize the cross-entropy (and thus KL-divergence) between $p_\theta(x_t|x_{<t})$ and $\mathbb{I}(x_t|x_{<t})$, where $p_\theta(x_t|x_{<t})$ is a probability distribution of the next token produced by the language model and $\mathbb{I}(x_t|x_{<t})$ is a degenerate distribution where all probability is assigned to x_t. Note that there is no conditioning on the sentence generated by the model itself $\hat{x}_{<t}$, which allows for distributional shift and makes the exposure bias possible. The same reasoning applies to using cross-entropy loss to find steering embedding $z_\theta(x)$ while solving eq. (3.1) (see 3.1).

One of the ways to mitigate the distributional shift in control is to use a dataset aggregation technique (DAGGER Ross, Gordon, and Bagnell, 2011), where trajectories, sampled from a pre-trained policy (and hence being out of training distribution), are assessed by an expert to determine the best action in a current state. This new data is then used to iteratively improve the initial policy. In the case of learning steering embeddings, we need to define a way to determine the optimal action x_t^*, which is reported by the expert, based on the sampled trajectory $\hat{x}_{<t}$ and expected tragectory x. This idea was explored in Sabour, Chan, and Norouzi, 2018, where the choice of optimal x_t^* in language modelling setup is determined to minimize the edit distance between $[\hat{x}_{<t}, x_t^*, \hat{x}^*]$ and x, where $\hat{x}^*$ denotes optimal suffix(es) that are yet to be generated by the model.

In this work we would like to apply similar approach to learning steering embeddings and propose an alternative objective function that eliminates the exposure bias issue:

$$z_\theta(x) = \underset{z \in \mathbb{R}^d}{\operatorname{argmin}} \, \mathbb{E}_{p_\theta(\hat{x}|z)} \left[D_{edit}(\hat{x}, x) \right] \tag{3.2}$$

Here, $\hat{x}$ represents a sentence sampled from the language model and x is a sentence of interest for which one wants to obtain a representation vector. $D_{edit}(\hat{x}, x)$ is the Levenshtein distance or edit distance between these two sentences. An important property of this objective is that ground-truth sentence is only used to compare it with the sampled sentence, thus the only way for the language model to know which tokens to generate is to use a provided vector z. Because there is no information leakage from the ground-truth sentence context during generation, the optimization process should lead to a representation that is more informative in comparison to the one obtained from eq. (3.1).

Inspecting the eq. (3.2) one can notice that it presents a challenge for direct optimization, because the Levenshtein distance is non-differentiable function. One possible way to optimize it is using reinforcement learning (RL) approach, e.g. using the REINFORCE Williams, 1992 algorithm. Tokens generated so far during the sampling process correspond to the agent's state, the agent's action corresponds to the symbol that should be produced next. Despite the recent successes of deep RL, obtaining acceptable levels of performance often requires an almost prohibitively large amount of experience to be acquired by the agent. Another, more practical way, to optimize this loss is to use policy distillation approach. If the expert policy exists, the learning efficiency of the RL task can be drastically improved. In Sabour, Chan, and Norouzi, 2018 the authors noticed that for the Levenshtein distance dynamic programming can be used to efficiently compute such expert policy, which they call the optimal completion (OC) policy $\pi_{\text{OC}}(\cdot|\hat{x}_{<t}, x)$. In other words, given the generated prefix $\hat{x}_{<t}$ at time step t and the original sequence x, we can determine what symbol should be generated to minimise the edit distance (see 3.2). In NLP, such expert policies are known as dynamic oracles Goldberg and Nivre, 2012. There is great interest in the RL field in methods that enable knowledge transfer to agents based on already trained policies Rusu et al., 2015 or human examples Abbeel and Ng, 2004. One of the most successful techniques for knowledge transfer is policy distillation Ross, Gordon, and Bagnell, 2010, where an agent is trained to match the state-dependent probability distribution over actions provided by a teacher/oracle. We use policy distillation as a proxy to optimize the eq. (3.2).

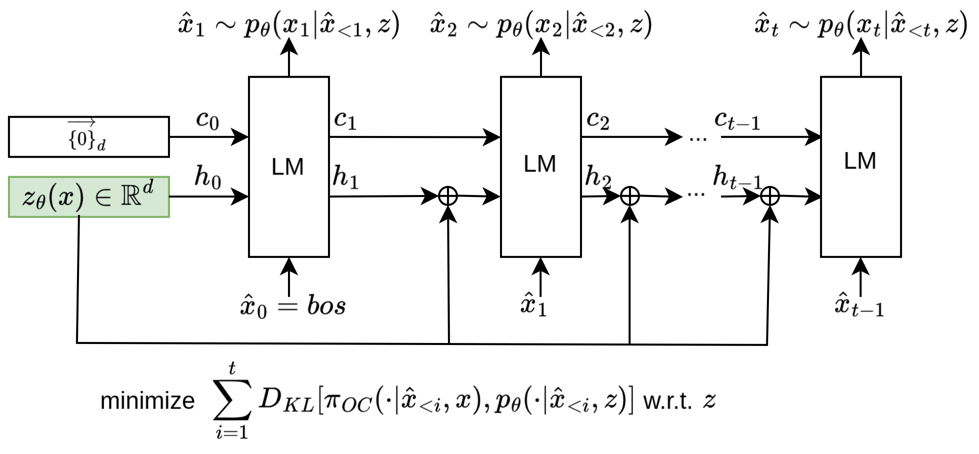

$\hat{x}_1 \sim p_\theta(x_1|\hat{x}_{<1}, z)$
$\hat{x}_2 \sim p_\theta(x_2|\hat{x}_{<2}, z)$
$\hat{x}_t \sim p_\theta(x_t|\hat{x}_{<t}, z)$
$\overrightarrow{\{0\}}_d$
$z_\theta(x) \in \mathbb{R}^d$
c_0
h_0
LM
c_1
h_1
c_2
h_2
c_{t-1}
h_{t-1}
$\hat{x}_0 = bos$
$\hat{x}_1$
$\hat{x}_{t-1}$
minimize $\sum_{i=1}^{t} D_{KL}[\pi_{OC}(\cdot|\hat{x}_{<i}, x), p_\theta(\cdot|\hat{x}_{<i}, z)]$ w.r.t. z

FIGURE 3.2: Steering approach from Subramani, Bowman, and Cho, 2019 + Optimal Completion Distillation. We hypothesize that the quality of the embedding z may be improved by conditioning on the samples from the language model and using optimal completion policy π_{OC}, which determines optimal completion tokens based on the edit distance between sampled trajectory $\hat{x}_{<i}$ and expected trajectory x.

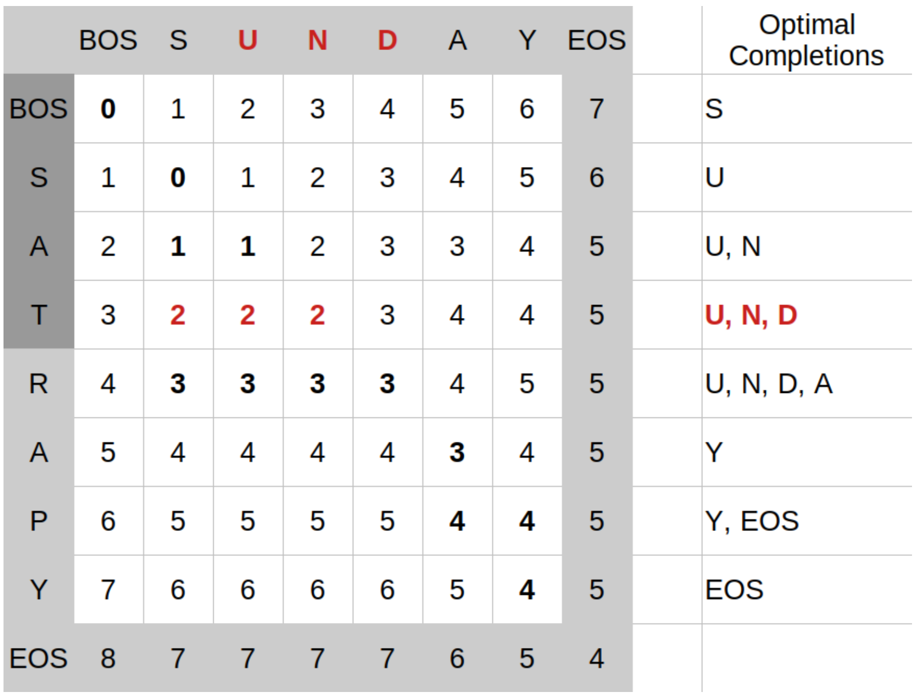
Usually, policy distillation is done by following updates in the gradient-like direction:

$$\mathbb{E}_{\hat{x}\sim\rho}\left[\sum_{t=1}^{|\hat{x}|}\nabla_\theta D_{\mathrm{KL}}\left[\pi_{\mathrm{oc}}(\cdot|\hat{x}_{<t},x),p_\theta(\cdot|\hat{x}_{<t},z)\right]\right] \tag{3.3}$$

The distribution ρ is known as a control policy and it is used to generate trajectories over which the distillation process is performed. D_{KL} is the Kullback–Leibler divergence. In our work we use pre-trained language model as a control policy $\rho = p_\theta$.

3.6 Calculating Optimal Completion Policy

To understand how π_{OC} is calculated, let us draw a connection between optimal completion and the notion of optimal Q values in reinforcement learning. Given the state-action pair (s, a) the optimal Q value $Q^*(s, a)$ represents a maximum possible future reward, assuming the agent will take optimal actions thereafter. In our case the state s corresponds to the prefix $\hat{x}_{<t}$ sampled by the model and action a corresponds to picking a particular token $\hat{x}_t$ at time t. The reward value should be a measure of similarity between sampled and ground truth sequences. Negative edit distance was chosen in Sabour, Chan, and Norouzi, 2018 as a common task metric. So for prefix $\hat{x}_{<t}$ and every possible token $\hat{x}_t$ from the language model vocabulary $\mathcal{V}$ we need to find the maximum possible future reward, which translates to finding the smallest edit distance $D_{edit}([\hat{x}_{<t}, \hat{x}_t, \hat{x}^*], x)$, where x is a target sequence and $\hat{x}^*$ is the optimal suffix that is yet to be generated by the model.



	BOS	S	U	N	D	A	Y	EOS	Optimal Completions
BOS	**0**	1	2	3	4	5	6	7	S
S	1	**0**	1	2	3	4	5	6	U
A	2	**1**	**1**	2	3	3	4	5	U, N
T	3	**2**	**2**	**2**	3	4	4	5	**U, N, D**
R	4	**3**	**3**	**3**	**3**	4	5	5	U, N, D, A
A	5	4	4	4	4	**3**	4	5	Y
P	6	5	5	5	5	**4**	**4**	5	Y, EOS
Y	7	6	6	6	6	5	**4**	5	EOS
EOS	8	7	7	7	7	6	5	4	

FIGURE 3.3: Edit distance matrix for generated sequence "SATRAPY" (vertical) and target sequence "SUNDAY" (horizontal). Set of optimal completion tokens for each generated prefix is determined based on the continuation of the target prefixes, which have the minimum edit distance with the sampled prefix at the current timestamp. The figure is an adaptation from Sabour, Chan, and Norouzi, 2018. (*BOS* and *EOS* stand for *b*egining and *e*nd *o*f *s*equence respectively)

Let's consider a particular sampled prefix $\hat{x}_{<t} = "SAT"$ (see 3.3). Recall the dynamic programming algorithm for calculating the edit distance between every prefix of $\hat{x}$ and x:

$$D_{edit}(\hat{x}_{<i}, x_{<j}) = min \begin{cases} D_{edit}(\hat{x}_{<i}, x_{<j-1}) + 1 \\ D_{edit}(\hat{x}_{<i-1}, x_{<j}) + 1 \\ D_{edit}(\hat{x}_{<i-1}, x_{<j-1}) + \mathbb{I}(\hat{x}_i \neq x_j) \end{cases} \tag{3.4}$$

It's easy to see that the edit distance $D_{edit}([\hat{x}_{<t}, \hat{x}_t], x)$ is lower bounded by $m_t = \min_j D_{edit}(\hat{x}_{<t}, x_{<j})$ regardless of the next generated token $\hat{x}_t$ because of how the values in edit distance matrix are calculated. It is also shown in Sabour, Chan, and Norouzi, 2018 that the smallest possible edit distance $D_{edit}([\hat{x}_{<t}, \hat{x}_t, \hat{x}^*], x)$ is achieved by following the suffixes of the target sequence, which correspond to $D_{edit}(\hat{x}_{<t}, x_{<j}) = m_t$ (see optimal completions in 3.3). Hence, we can define $Q^*(\hat{x}_{<t}, \hat{x}_t) = -m_t$ if $\hat{x}_t$ is an optimal completion and $Q^*(\hat{x}_{<t}, \hat{x}_t) = -m_t - 1$ otherwise. Then by applying softmax with low enough temperature we can define $\pi_{OC}(\cdot|\hat{x}_t, x)$ as a distribution, where all probability is equally distributed between optimal completion tokens.

It is worth mentioning that time complexity of computing policy $\pi_{OC}(\cdot|\hat{x}_t, x)$ is comparable to the time complexity of MLE in most cases. In MLE, we calculate D_{KL} for every timestamp of the target sequence, which yields $\mathcal{O}(|x| \cdot |\mathcal{V}|)$ time complexity. For $\pi_{OC}(\cdot|\hat{x}_t, x)$ we first calculate $D_{edit}(\hat{x}, x)$ and then calculate D_{KL} for every timestamp of the sampled sequence, yielding $\mathcal{O}(|x| \cdot |\hat{x}| + |\hat{x}| \cdot |\mathcal{V}|)$. Because the length of the generated and target sequences is usually smaller than the vocabulary size of the language model, we can conclude that time complexity of both approaches is the same (additionally assuming that $|x| \approx |\hat{x}|$).

Chapter 4

Model

In this chapter we describe the architecture of the language model we will be experimenting with as well as the strategies to incorporate learning an embedding vector from a pre-trained language model.

4.1 Architecture

Similarly to Subramani, Bowman, and Cho, 2019, *Long Short Term Memory (LTSM)* network (Hochreiter and Schmidhuber, 1997) is used to solve the task of language modelling. It's a type of Recurrent Neural Network (RNN) that has shown effectiveness with input data of different lengths. This effectiveness is achieved by merging an input vector with a state vector to create a new state vector that will be used for the next input.

Because basic RNNs struggled to capture long-term dependencies in data, the LSTM network was introduced. Its notable feature is the additional cell state, responsible for managing long-term dependencies. This cell state can be altered or retained using "forget" and "update" gates. Predictions are then based on a combination of the cell state, hidden state, and current input vector (see 4.1).

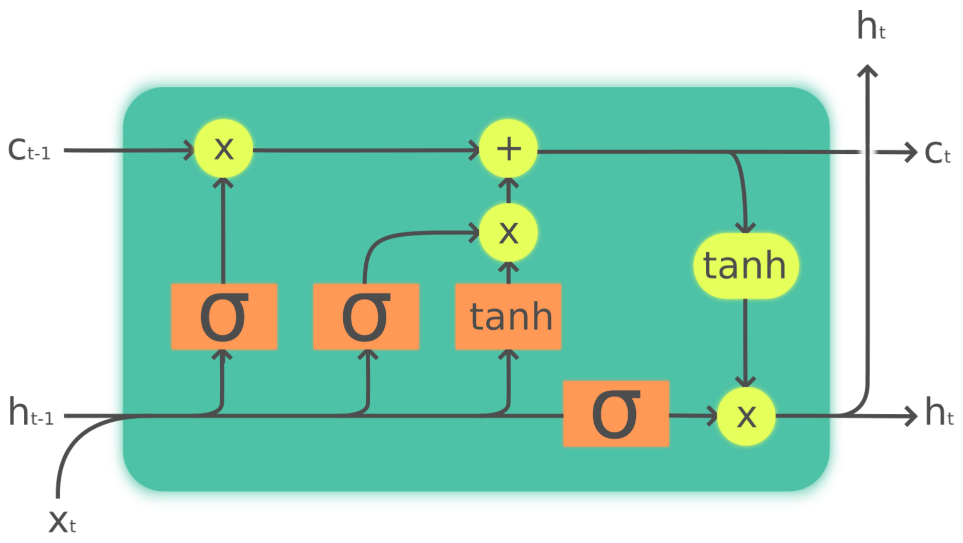


FIGURE 4.1: LSTM cell. Prediction for time stamp t is based on previous hidden state h_{t-1}, previous cell state c_{t-1} and current input x_t. Cell state is changed via "update" and "forget" gates. (original picture "LSTM Cell" by Guillaume Chevalier, licensed under CC BY 4.0).

The full architecture of the LSTM language model is shown on fig 4.2. For each token x_i of the input sequence x we first convert it to a dense representation

$emb_{x_i} \in \mathbb{R}^{d_{emb}}$. The resulting dense vectors are then passed through a one-directional (possibly multi-layer) LSTM network and the last layer hidden states $h_i^{out} \in \mathbb{R}^{d_{model}}$ are received. The final layer of the model is a linear transformation that projects the hidden states h_i^{out} to the dimensionality of the language model vocabulary size ($proj_{h_i^{out}} \in \mathbb{R}^{\mathcal{V}}$). After applying softmax we receive a probability distribution of the next token conditioned on the previous input $p_\theta(x_i|x_{<t})$ (where θ are the parameters of the language model). Additionally, to prevent overfitting and improve generalization capabilities of the model we use dropout after emb_{x_i} and $proj_{h_i^{out}}$.

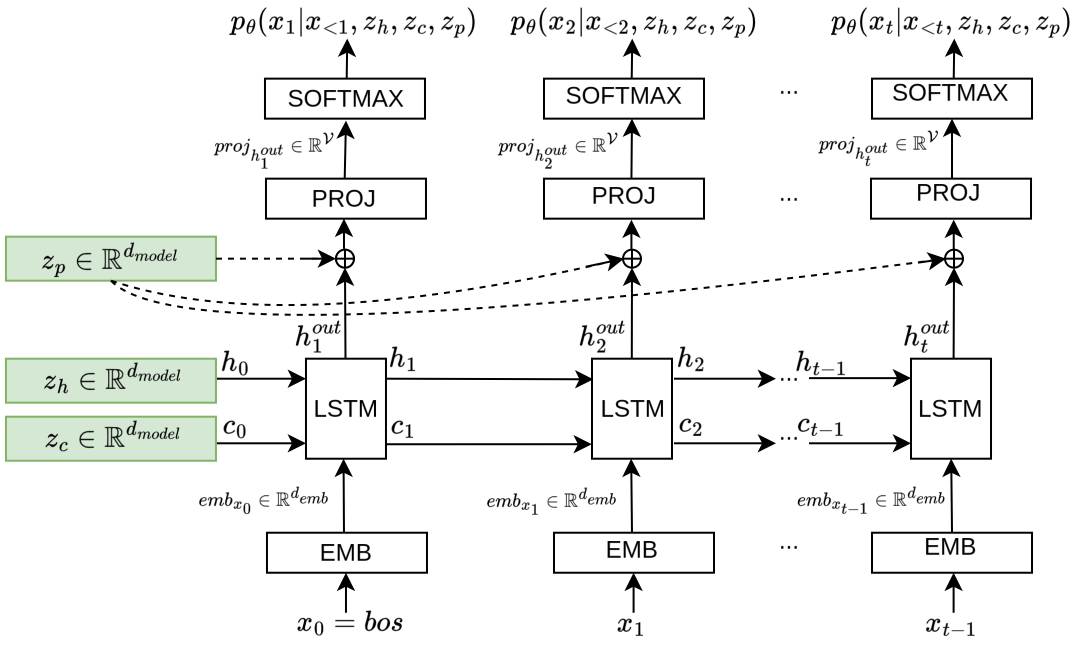

$p_\theta(x_1|x_{<1}, z_h, z_c, z_p)$
$p_\theta(x_2|x_{<2}, z_h, z_c, z_p)$
$p_\theta(x_t|x_{<t}, z_h, z_c, z_p)$
SOFTMAX
$proj_{h_1^{out}} \in \mathbb{R}^{\mathcal{V}}$
$proj_{h_2^{out}} \in \mathbb{R}^{\mathcal{V}}$
$proj_{h_t^{out}} \in \mathbb{R}^{\mathcal{V}}$
PROJ
$z_p \in \mathbb{R}^{d_{model}}$
$z_h \in \mathbb{R}^{d_{model}}$
$z_c \in \mathbb{R}^{d_{model}}$
h_1^{out}
h_2^{out}
h_t^{out}
h_0
h_1
h_2
h_{t-1}
c_0
c_1
c_2
c_{t-1}
LSTM
$emb_{x_0} \in \mathbb{R}^{d_{emb}}$
$emb_{x_1} \in \mathbb{R}^{d_{emb}}$
$emb_{x_{t-1}} \in \mathbb{R}^{d_{emb}}$
EMB
$x_0 = bos$
x_1
x_{t-1}
...

FIGURE 4.2: Visualization of the 1-layer LSTM language model (white cells) as well as the chosen strategy of incorporating steering vectors into the frozen pre-trained language model (green cells). Dropout layers after *emb* and *proj* layers of the language model are omitted for clarity.

4.2 Steering

To study the conditioning of the pre-trained LSTM language model on a steering vector z, we suggest three possible locations, where such vector can be injected (see green cells in 4.2):

1. independently adding embedding vector $z_p \in \mathbb{R}^{d_{model}}$ to the hidden states of the last LSTM layer before projection layer
2. providing embedding vector $z_h \in \mathbb{R}^{L \cdot d_{model}}$ as the first hidden state to the LSTM (where L is the number of layers in LSTM)
3. providing embedding vector $z_c \in \mathbb{R}^{L \cdot d_{model}}$ as the first cell state to the LSTM

The final steering vector z for a particular sequence x is a concatenation of z_p, z_h, z_c. Since conditioning on z_p, z_h and z_c is independent of each other, we can also drop one of the locations (e.g. consider z to be a concatenation of z_p and z_h).

Chapter 5

Experiments

In this chapter we describe the setup for the language modelling and training steering embeddings experiments as well as present the target metrics and experiments results.

5.1 Word-Level Language Modelling on Penn Treebank Dataset

To begin with, we train an LSTM language model on Penn Treebank dataset release 2 CDROM Prasad et al., 2008. It consists of a million words of 1989 Wall Street Journal material, with part-of-speech tagging information. The sentences are preprocessed in the following manner:

- all the punctuation is removed
- sentences are converted to lowercase
- all numbers are replaced with a special N token
- top 10k most frequent words are kept, all the other are replaced with <unk> token

In our experiments we use Penn Treebank solely for language modelling purposes, so only sentences part of the dataset is used. We utilize word-level tokenization to preprocess the text data as a standard tokenization procedure for this particular dataset, subsequently training a word-level LSTM language model. This approach involves segmenting the text into individual words, each of which is treated as a distinct token.

We use cross-entropy as a loss function to optimize the language model parameters. For particular sentence x we minimize:

$$\mathcal{L}(x) = -\sum_{i=1}^{|x|} log p_\theta(x_i|x_{<i})$$

where $p_\theta(x_i|x_{<i})$ is the predicted probability of the i-th word given the previous words in the input sequence. We report perplexity as the main metric for a particular split:

$$PP(split) = \frac{1}{|split|} \sum_{x \in split} exp(\mathcal{L}(x))$$

The language model is optimized via gradient descent using Adam optimizer. To achieve optimal results with simple LSTM architecture, we experiment with cosine warmup learning rate scheduler as well as different combinations of dropout probability and weight decay L2 regularization. Because single-layer LSTM model is powerful enough for such relatively small dataset as Penn Treebank, we did not observe an improvement in model's generalization abilities when using greater number of layers. Hence we only consider single-layer LSTMs in our language model experiments (see 5.1).

Exp Name	**Model Config**	**Training Config**	**PP Train**	**PP Validation**	**PP Test**
ptb-word-lst-002	d_model=512 d_emb=256 num_layers=1 p_dropout=0.15	num_epochs=16 batch_size=1024 lr = 1e-2 lr_scheduler=None weight_decay=0.0	28.8	960.6	527.4
ptb-word-lstm-003	d_model=512 d_emb=256 num_layers=1 p_dropout=0.15	num_epochs=16 batch_size=1024 lr = 1e-2 lr_scheduler=None weight_decay=1e-4	30.7	269.6	257
ptb-word-lstm-005	d_model=512 d_emb=256 num_layers=1 p_dropout=0.15	num_epochs=32 batch_size=1024 lr = 1e-1 lr_scheduler=CosineWarmup warmup epochs=4 weight_decay=1e-3	150.9	221.2	211.2
ptb-word-lstm-006	d_model=512 d_emb=256 num_layers=1 p_dropout=0.15	num_epochs=32 batch_size=1024 lr = 1.25e-2 lr_scheduler=CosineWarmup warmup_epochs=4 weight_decay=1e-3	119.4	**211.7**	203.9
ptb-word-lstm-008	d_model=256 d_emd=256 num_layers=1 p_dropout=0.15	num_epochs=32 batch_size=1024 lr = 1.25e-2 lr_scheduler=CosineWarmup warmup_epochs=4 weight_decay=1e-3	154.5	225.06	216.4

TABLE 5.1: Experiment results for different configurations of word-level LSTM language model trained on Penn Treebank Dataset. The model from **ptb-word-lstm-006** was chosen for training steering embeddings on.

5.2 Training Steering Embeddings on Penn Treebank Word-Level LSTM

Given a pre-trained language model from **ptb-word-lstm-006** (see 5.1), we train the steering embeddings for Penn Treebank dataset using the approach described in 4.2. We experiment with different combinations of injection locations (z_p, z_h, z_c) as well as different optimization procedures (solving 3.1 via maximum likelihood vs. solving 3.2 via optimal completion distillation).

To estimate the training progress, we report perplexity of the split, conditioned on the trained embedding vectors:

$$PP(split|z_{split}) = -\frac{1}{|split|} \sum_{x \in split} \sum_{i=1}^{|x|} log p_\theta(x_i|x_{<i}, z_\theta(x))$$

Since both MLE and OCD optimization procedures aim to achieve perfect reconstruction of the original sentence, we also report the reconstruction metrics discussed in Subramani, Bowman, and Cho, 2019, namely exact match, BLEU and prefix match.

Given a pre-trained language model with parameters θ, an input sequence x, a sentence steering embedding $z_\theta(x)$ and a sequence $\hat{x}$, which was sampled by the language model conditioned on $z_\theta(x)$, we want to measure, how well the model is able to recover the original sequence. For this we calculate the following metrics:

Exact match is defined as the average number of matching tokens between x and $\hat{x}$:

$$exact\_match(x, \hat{x}) = \sum_{t=1}^{|x|} \mathbb{I}(x_t = \hat{x}_t)/|x|$$

While being suitable for perfect-recoverability setup, the exact match may degrade significantly even when the differences between x and $\hat{x}$ are minor (e.g. $exact\_match((x_1, x_2, \ldots, x_n), (x_n, x_1, \ldots, x_{n-1})) = 0$ given unique tokens).

To alleviate this, *BLEU* Papineni et al., 2002 metric is used as an alternative, initially proposed to evaluate the quality of machine translations by averaging the precision of n-grams of different lengths between candidate and reference sentences, thus allowing us to capture both the accuracy and fluency of the reconstruction.

Additionally, we might be interested to investigate the limits of the perfect reconstruction length, with *prefix match* metric calculating the position of the first discrepancy between x and $\hat{x}$:

$$prefix\_match(x, \hat{x}) = \underset{t}{\operatorname{argmax}}(t \cdot \mathbb{I}(x_{<t} = \hat{x}_{<t}))$$

The results of the experiments are presented in table 5.2. The best values of reconstruction metrics are achieved by using all three embedding locations together with Maximum Likelihood Estimation.

The best training configuration for each combination of embedding locations and optimization procedure was obtained via grid search. During hyperparameter tuning we found that embeddings trained with OCD optimization procedure require lower learning rate and greater number of epochs to converge. Hence the training config is different for *005-phc-ocd* as opposed to other experiments mentioned in 5.2 (see 6.1 for further discussion on this matter).

Exp Name	**000-phc**	002-ph	003-hc	005-phc-ocd
Embedding Locations	z_p, z_h, z_c	z_p, z_h	z_h, z_c	z_p, z_h, z_c
Training Config	lr=1e-1 n_epochs=512	lr=1e-1 n_epochs=512	lr=1e-1 n_epochs=512	lr=1e-2 n_epochs=1024
Optimization Procedure	MLE	MLE	MLE	OCD
PP Train | z	2.164	3.089	26.4	-
PP Validation | z	2.586	4.071	36.7	9.6
PP Test | z	2.511	3.945	33.7	8.32
exact_match Train	0.66	0.43	0.34	-
exact_match Validation	0.61	0.38	0.33	0.45
exact_match Test	0.62	0.37	0.34	0.46
prefix_match Train	12.88	7.56	5.19	-
prefix_match Validation	11.33	6.29	4.99	7.44
prefix_match Test	11.47	6.27	5.11	7.67
BLEU score Train	0.68	0.44	0.25	-
BLEU score Validation	0.62	0.36	0.24	0.44
BLEU score Test	0.63	0.37	0.25	0.46

TABLE 5.2: Comparison between different configurations of steering embeddings trained on Penn Treebank dataset from a pre-trained LSTM language model from **ptb-word-lstm-006**

5.3 Estimating the Quality of MLE Steering Embeddings

To estimate the quality of MLE steering embeddings, we cross-evaluate them on a Stanford Sentiment Treebank (SST-2) dataset, which contains a collection of 70k movie reviews together with binary sentiment labels. We use MLE steering embeddings from **000-phc** as features and train a logistic regression. We report the resulting accuracy.

We compare the resulting accuracy with traditional embedding approach, where embedding of the sentence it determined via max/mean pooling of the hidden/cell states of the LSTM part of the language model. We make sure the pooling embedding vector has the same dimensionality as the steering embedding for the comparison of logistic regression models to be fair. The results are presented in table 5.3.

Embedding Type	Accuracy Train	Accuracy Validation
Steering MLE 000-phc	62.37	61.58
Mean-pooling of the hidden states	**66.99**	**67.08**
Max-pooling of the hidden states	63.5	59.17

TABLE 5.3: The results of cross-evaluation of steering MLE embeddings versus traditional embedding approaches on SST2 dataset

Similarly to SST-2, we also cross-evaluate the MLE steering embeddings from **000-phc** on the Stanford Natural Language Inference (SNLI) corpus. The SNLI dataset contains more than 570,000 annotated pairs of sentences, with each pair being categorized as entailment, contradiction, or neutral. This extensive collection is specifically created to evaluate a model's capacity to comprehend and deduce connections between sentences, serving as a rigorous standard for assessing the quality of the embeddings in tasks related to understanding natural language. The results are presented in table 5.4.

Embedding Type	Accuracy Train	Accuracy Validation	Accuracy Test
Steering MLE 000-phc	50.65	50.33	49.88
Mean-pooling of the hidden states	**54.71**	**54.87**	**54.32**
Max-pooling of the hidden states	51.71	51.19	51.92

TABLE 5.4: The results of cross-evaluation of steering MLE embeddings versus traditional embedding approaches on SNLI dataset

Chapter 6

Discussion

6.1 Difficulties of Learning OCD Steering Embeddings

Contrary to our hypothesis, learning steering embeddings using optimal completion distillation objective did not improve the likelihood of the input sentence, when conditioning the language model with the steering vector (see 5.2). Despite training OCD embeddings for more number of epochs, the likelihood of the input sentence remained bigger as opposed to the embedding trained with MLE approach. We also noticed that OCD required lower learning rate to converge as opposed to MLE.

Because OCD is trained on sequences sampled from the model, these sequences change throughout the process of optimizing the embedding vector $z_\theta(x)$. This forces embedding vector $z_\theta(x)$ to store the information about optimal completions of multiple generated sequences, effectively reducing the embedding vector capacity to learn how to reconstruct the original sentence x from $z_\theta(x)$. This may also explain why OCD requires lower learning rate to converge, since with bigger learning rate generated sequences $\hat{x}$ might have been changing too quickly, possibly turning into a sequence of "random" sentences for a single embedding vector $z_\theta(x)$ to learn.

It is also worth noting that despite time complexity of OCD is the same as MLE, there are some performance considerations that should be taken into account. Because OCD is trained on sequences generated from the model, sampling is required. This greatly limits our ability to use fused implementation of the LSTM, effectively degrading performance. In addition to this, edit distance matrix is calculated on cpu, which requires transferring model's generated sequence from gpu to cpu and π_{OC} values from cpu back to gpu. Since latency associated with transferring data to the gpu may become a bottleneck, custom CUDA implementation of edit distance algorithm may be required for the OCD to work optimally.

6.2 On the quality of the resulting embeddings

We have shown, that the quality of the resulting MLE steering embeddings is comparable to the ones that are obtained by max-pooling the hidden states of the LSTM. However, it falls behind when comparing to mean-pooling of the hidden states. We suspect it may be related to the nature of SST2 dataset itself, since it is straightforward enough that only a keyword detector is needed to understand the sentiment of a sentence. And because the steering embedding $z_\theta(x)$ contains information about the order of the words in a sentence, it falls behind in estimating sentiment.

Another explanation of why MLE steering embeddings perform worse compared to mean pooling may be because of how z_h and z_c steering vectors are incorporated into the LSTM language model (see 4.2). Because z_h and z_c are only injected on the first timestamp, the usage of information from these vectors on later timestamps is fairly limited due to the lack of direct injection, which limits the amount of knowledge from the input sequence that could be "learnt into" z_h and z_c. As a confirmation of this hypothesis, from the results of training steering embeddings with MLE in 5.2 we can see that z_p injection location is crucial for reconstruction metrics to be high.

Chapter 7

Future Work

7.1 Exploring Alternative Injection Locations

As discussed in 6.2, the current way of injecting z_h and z_c parts of the steering vector into the pre-trained LSTM language model may be suboptimal. To account for this, we may try to inject z_h and z_c at every timestamp (similarly to how it is done for z_p in 4.2). Doing this should improve z_h and z_c parts of the resulting embedding vector and potentially lead to better evaluation results.

7.2 Altering Optimal Completion Objective Function

To account for the difficulties in learning OCD steering embeddings described in 6.1, we suggest two possible directions for future work:

1. focusing on optimal completions of the first tokens of the sentence instead of all tokens. Because generated sequences $\hat{x}$ is subject to change when optimizing $z_\theta(x)$ with OCD, it may be suboptimal to include the optimal completions later in the sequence into the loss function, since the end of the generated sequence is more likely to change in the next iteration. Hence, reducing the number of prefixes we optimize $p_\theta(\cdot|\hat{x}_t, z)$ for will help to focus on early corrections of the generated sequence.

2. as a continuation of the previous point, another way to reduce the number of sequences for vector $z_\theta(x)$ to learn is to combine MLE and OCD objective functions. Doing this will ensure that the likelihood $p_\theta(x|z)$ will remain high as well as allow for the model to find optimal completion token once the model has made a mistake during reconstruction.

7.3 Further Evaluation of Steering Embeddings

Another direction for future works would be to further evaluate the quality of the steering embeddings on more complex downstream tasks. Sent-eval benchmark Conneau and Kiela, 2018 may be used for this purpose. It offers a collection of seventeen downstream tasks as well as the requirements for probing architectures, that use sentence embedding in a feature-based approach, such as classification, semantic text similarity, semantic relatedness and others. The evaluation metric depends on the downstream task, accuracy is used for classification tasks and Pearson and Spearman correlations (and their combination) are reported for tasks where relation between pairs of sentences needs to be estimated.

7.4 Steering Transformers

Since significant portion of current research and computational power in the field of machine learning is dedicated to the Transformer architecture, investigation of steering strategies for this architecture would be a natural next step. Despite the fact that it was explored in Subramani, Suresh, and Peters, 2022, we were unable to reproduce the results mentioned in the paper. We have found that learning steering embedding vector using MLE is sensitive to learning rate, steering vector injection location and the way $z_\theta(x)$ is initialized.

7.5 Direct Optimization of the Optimal Completion Objective Function

One of the question not considered in this work is whether it's feasible to optimize eq. (3.2) directly. Since Levenstein distance is not differentiable function, it's not feasible to calculate the gradient with respect to it. One of the possible approaches might be to use policy gradient, where the gradient is estimated in Monte Carlo fasion via sampling, which provides an unbiased estimator. However, it's important to assess the issue of high variance to evaluate its impact on the reliability and stability of the results. One of the possible options for reducing the variance of an estimator is to incorporate optimal completion distillation as a form of control variates, though the precise mathematical formulation and practical implementation of this method requires additional investigation.

Another approach to directly solving eq. (3.2) would be to modify the definition of the Levenstein distance itself to allow for differentiation. In this case, the properties of such new distance function should be thoroughly assessed before proceeding. It's also possible to consider other distance functions. An important property for such a function should be an existence of the completion policy, which should output the set of extension tokens for any prefix sampled from the langauge model.

Chapter 8

Conclusions

In this work we explored an alternative approach for learning embeddings from pre-trained language model without altering its architecture or parameters. We formulated the objective of learning steering embeddings via maximum likelihood estimation (eq. 3.1) as well as presented the results of its particular implementation on a pre-trained LSTM language model. From our experiments we conclude the following:

1. for the LSTM language model injection of steering embedding should be done at every timestamp
2. injecting embedding vector into the projection layer (see z_p in 4.2) is crucial for steering approach to work (i.e. steering embeddings having good recoverability results)
3. the quality of the resulting embeddings may be improved further by injecting additional steering vector at LSTM hidden/cell states at every timestamp

We state that there is no standardized approach for injecting steering vectors if we were to consider different language model architecture, with no guarantee that successful injection location for one architecture automatically leads to a success in a completely different architecture.

We also showed that even with not entirely optimal injection approach, the quality of the steering embeddings is comparable to the ones obtained by taking max-pooling of the activations of the pre-trained LSTM language model, with a potential for achieving even better quality as described in 7.1. However, the low number of downstream tasks steering embeddings were evaluated on does not allow us to make a definitive statement on their general applicability and quality.

Independently to Subramani, Bowman, and Cho, 2019 we verified, that the limit of perfect reconstruction of the original sequence using steering embeddings appears to be quite low (see prefix_match results in 5.2). To account for this, we suggested learning steering embeddings using an alternative objective function (eq. 3.2) via Optimal Completion Distillation (OCD) loss, which focuses on distilling knowledge of optimal completion for the sequence, that was generated from model, conditioned on the steering embedding (see 3.2).

From our experiments we conclude that OCD loss is more challenging to optimize as opposed to MLE objective. OCD does not improve recoverability out of the box and needs additional modifications to work properly (see 7.2).

Bibliography

Abbeel, P. and A. Ng (2004). “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. URL: https://api.semanticscholar.org/CorpusID:207155342.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1409.0473.

Bengio, Yoshua, Patrice Y. Simard, and Paolo Frasconi (1994). “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5 2, pp. 157–66. URL: https://api.semanticscholar.org/CorpusID:206457500.

Bowman, Samuel R. et al. (2015). “A large annotated corpus for learning natural language inference”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*. Ed. by Lluís Màrquez et al. The Association for Computational Linguistics, pp. 632–642. DOI: 10.18653/V1/D15-1075. URL: https://doi.org/10.18653/v1/d15-1075.

Bowman, Samuel R. et al. (2016). “Generating Sentences from a Continuous Space”. In: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, CoNLL 2016, Berlin, Germany, August 11-12, 2016*. Ed. by Yoav Goldberg and Stefan Riezler. ACL, pp. 10–21. DOI: 10.18653/V1/K16-1002. URL: https://doi.org/10.18653/v1/k16-1002.

Brochier, Robin, Adrien Guille, and Julien Velcin (May 2019). “Global Vectors for Node Representations”. In: *The World Wide Web Conference*. WWW ’19. ACM. DOI: 10.1145/3308558.3313595. URL: http://dx.doi.org/10.1145/3308558.3313595.

Brown, Tom B. et al. (2020). “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Cer, Daniel et al. (2018). “Universal Sentence Encoder”. In: *CoRR* abs/1803.11175. arXiv: 1803.11175. URL: http://arxiv.org/abs/1803.11175.

Cho, Kyunghyun et al. (2014). “On the Properties of Neural Machine Translation: Encoder-Decoder Approaches”. In: *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014*. Ed. by Dekai Wu et al. Association for Computational Linguistics, pp. 103–111. DOI: 10.3115/V1/W14-4012. URL: https://aclanthology.org/W14-4012/.

Conneau, Alexis and Douwe Kiela (2018). “SentEval: An Evaluation Toolkit for Universal Sentence Representations”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12,* *2018*. Ed. by Nicoletta Calzolari et al. European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2018/summaries/757.html.

Conneau, Alexis et al. (2017). "Supervised Learning of Universal Sentence Representations from Natural Language Inference Data". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Association for Computational Linguistics, pp. 670–680. DOI: 10.18653/V1/D17-1070. URL: https://doi.org/10.18653/v1/d17-1070.

Dai, Andrew M and Quoc V Le (2015). "Semi-supervised Sequence Learning". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/7137debd45ae4d0ab9aa953017286b20-Paper.pdf.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/V1/N19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

Freitag, Markus and Yaser Al-Onaizan (2017). "Beam Search Strategies for Neural Machine Translation". In: *Proceedings of the First Workshop on Neural Machine Translation*. Association for Computational Linguistics. DOI: 10.18653/v1/w17-3207. URL: http://dx.doi.org/10.18653/v1/W17-3207.

Goldberg, Yoav and Joakim Nivre (2012). "A Dynamic Oracle for Arc-Eager Dependency Parsing". In: *International Conference on Computational Linguistics*. URL: https://api.semanticscholar.org/CorpusID:1195002.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Kingma, Diederik P. and Max Welling (2014). "Auto-Encoding Variational Bayes". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1312.6114.

Mikolov, Tomás et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1301.3781.

Muennighoff, Niklas (2022). "SGPT: GPT Sentence Embeddings for Semantic Search". In: *CoRR* abs/2202.08904. arXiv: 2202.08904. URL: https://arxiv.org/abs/2202.08904.

Neelakantan, Arvind et al. (2022). "Text and Code Embeddings by Contrastive Pre-Training". In: *CoRR* abs/2201.10005. arXiv: 2201.10005. URL: https://arxiv.org/abs/2201.10005.

Pansare, Niketan et al. (2022). "Learning Compressed Embeddings for On-Device Inference". In: *Proceedings of Machine Learning and Systems 2022, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*. Ed. by Diana Marculescu, Yuejie Chi, and Carole-Jean Wu. mlsys.org. URL: https://proceedings.mlsys.org/paper_files/paper/2022/hash/72988287eb4acead9fe584bff6c488c5-Abstract.html.

Papineni, Kishore et al. (July 2002). "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for*

Computational Linguistics. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040.

Pascanu, Razvan, Tomás Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. Vol. 28. JMLR Workshop and Conference Proceedings. JMLR.org, pp. 1310–1318. URL: http://proceedings.mlr.press/v28/pascanu13.html.

Peters, Matthew E. et al. (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*. Ed. by Marilyn A. Walker, Heng Ji, and Amanda Stent. Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/V1/N18-1202. URL: https://doi.org/10.18653/v1/n18-1202.

Prasad, Rashmi et al. (Jan. 2008). "The Penn Discourse TreeBank 2.0". In.

Raffel, Colin et al. (2020). "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *J. Mach. Learn. Res.* 21, 140:1–140:67. URL: http://jmlr.org/papers/v21/20-074.html.

Reimers, Nils and Iryna Gurevych (2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*. Ed. by Kentaro Inui et al. Association for Computational Linguistics, pp. 3980–3990. DOI: 10.18653/V1/D19-1410. URL: https://doi.org/10.18653/v1/D19-1410.

Robertson, Stephen and Hugo Zaragoza (Jan. 2009). "The Probabilistic Relevance Framework: BM25 and Beyond". In: *Foundations and Trends in Information Retrieval* 3, pp. 333–389. DOI: 10.1561/1500000019.

Ross, Stéphane, Geoffrey J. Gordon, and Drew Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. Ed. by Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík. Vol. 15. JMLR Proceedings. JMLR.org, pp. 627–635. URL: http://proceedings.mlr.press/v15/ross11a/ross11a.pdf.

Ross, Stéphane, Geoffrey J. Gordon, and J. Andrew Bagnell (2010). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics*. URL: https://api.semanticscholar.org/CorpusID:103456.

Rusu, Andrei A. et al. (2015). "Policy Distillation". In: *CoRR* abs/1511.06295. URL: https://api.semanticscholar.org/CorpusID:1923568.

Sabour, Sara, William Chan, and Mohammad Norouzi (2018). "Optimal Completion Distillation for Sequence Learning". In: *ArXiv* abs/1810.01398. URL: https://api.semanticscholar.org/CorpusID:52909749.

Shen, Dinghan et al. (2019). "Learning Compressed Sentence Representations for On-Device Text Processing". In: *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Ed. by Anna Korhonen, David R. Traum, and Lluís Màrquez.

Association for Computational Linguistics, pp. 107–116. DOI: 10.18653/V1/P19-1011. URL: https://doi.org/10.18653/v1/p19-1011.

Subramani, Nishant, Samuel R. Bowman, and Kyunghyun Cho (2019). "Can Unconditional Language Models Recover Arbitrary Sentences?" In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al., pp. 15232–15242. URL: https://proceedings.neurips.cc/paper/2019/hash/48c8c3963853fff20bd9e8bee9bd4c07-Abstract.html.

Subramani, Nishant and Nivedita Suresh (2020). "Discovering Useful Sentence Representations from Large Pretrained Language Models". In: *ArXiv* abs/2008.09049. URL: https://api.semanticscholar.org/CorpusID:221186910.

Subramani, Nishant, Nivedita Suresh, and Matthew E. Peters (2022). "Extracting Latent Steering Vectors from Pretrained Language Models". In: *ArXiv* abs/2205.05124. URL: https://api.semanticscholar.org/CorpusID:248693452.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Zoubin Ghahramani et al., pp. 3104–3112. URL: https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2
Abstract.html.

Thakur, Nandan et al. (2021). "BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models". In: *CoRR* abs/2104.08663. arXiv: 2104.08663. URL: https://arxiv.org/abs/2104.08663.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al., pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Wang, Kexin, Nils Reimers, and Iryna Gurevych (Nov. 2021). "TSDAE: Using Transformer-based Sequential Denoising Auto-Encoderfor Unsupervised Sentence Embedding Learning". In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Ed. by Marie-Francine Moens et al. Punta Cana, Dominican Republic: Association for Computational Linguistics, pp. 671–688. DOI: 10.18653/v1/2021.findings-emnlp.59. URL: https://aclanthology.org/2021.findings-emnlp.59.

Wang, Wenhui et al. (2020). "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. Ed. by Hugo Larochelle et al. URL: https://proceedings.neurips.cc/paper/2020/hash/3f5ee243547dee91fbd053c1c4a845aa
Abstract.html.

Williams, Ronald J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8, pp. 229–256. URL: https://api.semanticscholar.org/CorpusID:2332513.

Zaken, Elad Ben, Yoav Goldberg, and Shauli Ravfogel (2022). "BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*. Ed. by Smaranda Muresan, Preslav Nakov, and Aline Villavicencio. Association for Computational Linguistics, pp. 1–9. DOI: 10.18653/V1/2022.ACL-SHORT.1. URL: https://doi.org/10.18653/v1/2022.acl-short.1.

Zhao, Wayne Xin et al. (2023). *A Survey of Large Language Models*. arXiv: 2303.18223 [cs.CL].