Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

Faculty of Design Computer Science Media

Master's Thesis Computer Science

Unsupervised Neural Text Search using Retrieval-Augmented Language Modeling

Submitted by	Viola Campos
Date of submission	07.06.2021
Supervisor	Dr. Adrian Ulges
Second Supervisor	Dr. Dirk Krechel

Selbstständigkeitserklärung

Ich erkläre hiermit,

- dass ich die vorliegende Abschlussarbeit selbstständig angefertigt,
- keine anderen als die angegebenen Quellen benutzt,
- die wörtlich oder dem Inhalt nach aus fremden Arbeiten entnommenen Stellen, bildlichen Darstellungen und dergleichen als solche genau kenntlich gemacht und
- keine unerlaubte fremde Hilfe in Anspruch genommen habe.

Wiesbaden, 07.06.2021

Viola Campos

Erklärung zur Veröffentlichung

Hiermit erkläre ich mein Einverständnis mit den im folgenden aufgeführten Verbreitungsformen dieser Abschlussarbeit:

Verbreitungsform	**Ja**	**Nein**
Einstellung der Arbeit in die Bibliothek der RheinMain University of Applied Sciences	×	
Veröffentlichung des Titels der Arbeit im Internet	×	
Veröffentlichung der Arbeit im Internet	×	

Wiesbaden, 07.06.2021

Viola Campos

Für Malu, Lua und Fabio.

Abstract

Information retrieval is a core task in many real-world applications, such as digital libraries, expert finding, web search, and question answering. Recent work in information retrieval is often based on deep neural language models, usually pre-trained on unstructured text followed by a supervised fine-tuning step. In this context, a common problem is to gather annotated training data, which is time consuming and expensive.

This thesis presents a scheme for self-supervised training of a transformer-based neural text retriever. The approach is based on training a two-stage autoencoder model consisting of a retrieval and a generator component. The training objective is reconstruction of an input text by first retrieving a set of related documents and then conditioning on them to generate the original input. To simultaneously improve retrieval, the usefulness of each document for reconstruction serves as a training signal to train the retriever. The idea is inspired by *retrieval-augmented language modeling*, a recent thread of work that seeks to integrate neural text retrieval into model pre-training.

To facilitate the above multi-task training, it was essential to start training with a retriever component that is capable, to some degree, of identifying relevant documents. To achieve this, the retriever is pre-trained on pairs of queries and supposedly related documents, sampled using heuristics. In ablation studies, the individual components of the system proposed in this work were examined in detail. First, different approaches for pre-training the BERT based retrieval component are compared and empirically analyzed, examining strategies for sampling negative training samples and investigating the effect of representation size. Several approaches for constructing and training a combined retrieve-and-generate model were compared and evaluated for their ability to refine the retriever by self-supervised training. Based on the insights from the ablation studies, two model variants were developed, trained and evaluated on the passage retrieval task of MS MARCO, an open passage retrieval benchmark. The best model achieves 24.6% MRR@10 on MS MARCO, outperforming the BM25 baseline by 5.9%. Further, it was demonstrated that the objective of self-supervised reconstruction actually improves retriever accuracy, depending on the model architecture by up to 10% MRR@10 on MS MARCO compared to the pre-trained model.

Contents

1. Introduction

A core task for language technology is to retrieve relevant information from a corpus of unstructured text. The most common formulation of the problem is text search, where a user typically enters a query and the retrieval system produces a list of texts such as web pages, scientific papers, news articles or tweets, ordered by estimated relevance with respect to the query. With vast amounts of written text available in wikipedia, news articles, scientific papers, books, emails, tweets, etc., the need for efficient techniques to access desired information is huge. Furthermore, text retrieval is not only important for ad hoc search, but forms a core component of many *natural language processing (NLP)* applications such as question-answering or recommender systems.

While conventional *information retrieval (IR)* systems use heuristic keyword matching approaches to find relevant documents for a query, the introduction of neural network-based technologies into IR is currently leading to a paradigm shift. In October 2019, Google Search announced in a blog post[1] the introduction of a neural model called *Bidirectional Encoder Representations from Transformers (BERT)* [Devlin et al., 2019] into its web search as 'the biggest leap forward in the past five years, and one of the biggest leaps forward in the history of Search.' BERT belongs to a family of deep neural models (called *transformers*) that process words in relation to all other words in a sentence. Transformers are commonly pre-trained on a huge text corpus to generate contextualized representations for queries and documents and fine-tuned on various target tasks what has led to new state-of-the-art in retrieval as well as in many other NLP tasks.

When applying transformer models in information retrieval, pre-training is performed in a self-supervised manner on unstructured text, followed by a supervised fine-tuning step. In this process, it is often challenging to obtain sufficient training data for supervised fine-tuning. Usually, the training data consists of pairs of queries and a corresponding list of documents containing the required information, typically extracted from a heuristic search

[1] `https://www.blog.google/products/search/search-language-understanding-bert/`, retrieved May 2021

and hand-labeled as relevant by human annotators. This process is time-consuming and labor-intensive, hence many datasets and applications lack such annotations. This is a particular problem for domain-specific data, e.g. in the medical or legal domain, where typical open-access corpora, used by researchers to pre-train and fine-tune models, such as Wikipedia, may be very different from the target corpora. The main problem here are vocabulary differences between the pre-training corpus and the target domain, such that words that are important in the target domain are rarely or not at all encountered during pre-training, which prevents the model from learning the associated facts [Lin et al., 2020]. The pre-trained model is only of limited use for the target task and would require large amounts of labeled training data for domain-specific fine-tuning.

This work explores techniques to mitigate the issue and proposes a scheme for self-supervised training of a neural retriever. Such a system is expected to be of high practical value, as many companies have access to huge corpora of unstructured domain-specific text data, where the possibility to search for useful information without the effort of annotating training examples would be valuable. Current approaches to overcome the data bottleneck in supervised learning are mostly based on distant supervision [Rudra and Anand, 2020] and data augmentation [Lu et al., 2020], where labeled examples are gathered synthetically, either using heuristics or by creating additional examples from an existing set of training examples. This work takes a different approach, training a two-stage autoencoder model consisting of a retrieval component and a generator component. The training objective is to reconstruct a query text by first retrieving a set of related texts and then conditioning on them to generate the original. The usefulness of each document for reconstruction serves as a training signal to improve the retriever.

The idea is inspired by *retrieval-augmented language modeling*, a very recent thread of work that seeks to integrate neural text retrieval into model pre-training [Guu et al., 2020, Lewis et al., 2020a, Lewis et al., 2020b]. Based on the observation that large language representation learners like BERT [Devlin et al., 2019], RoBERTa [Liu et al., 2019] and T5 [Raffel et al., 2020] have been shown to store an amazing amount of linguistic knowledge, acquired from the massive text corpora they are trained on, retrieval-augmented models intend to capture knowledge in a more modular and interpretable way. By splitting language modeling into a retrieval and a prediction step, models like REALM [Guu et al., 2020] can retrieve documents from a large document corpus that help to inform its prediction and use the information during inference, achieving new state-of-the-art in question answering. While prior works [Miller et al., 2016, Chen et al., 2017, Khandelwal et al., 2020b, Min et al., 2019, Asai et al., 2020] already showed the benefit of adding a retrieval step to

a neural network, REALM is the first approach to employ a learned retriever instead of a heuristic approach and describes end-to-end training for the model. Similar approaches that integrate retrieval into text generation instead of language modeling are presented in [Lewis et al., 2020a, Lewis et al., 2020b].

This work exploits the fact that these models combine the retrieval and the generation component into an end-to-end differentiable model to provide a method for self-supervised retrieval. While a retrieval-augmented generation model is trained to self-supervisedly reconstruct an input from retrieved texts, the retriever jointly learns to retrieve beneficial texts. As argued above, existing pre-trained models have weaknesses in domain-specific retrieval. Nevertheless, they store useful linguistic knowledge, which is leveraged in the proposed training scheme by initializing the models as already pre-trained transformers that will be further refined by self-supervised training. Specifically, BERT [Devlin et al., 2019] is used as retriever and BART [Lewis et al., 2019] as generator.

The thesis is organized as follows. Chapter 2 provides an overview over important concepts and technologies. After introducing the Transformer model and the underlying attention mechanism, the pre-trained models used in the experiments are presented in more detail. Further, different approaches for neural retrieval are compared. Chapter 3 continues with an overview over related work in the area of retrieval-augmented language modeling. Chapter 4 presents the proposed approach for self-supervised retrieval training and gives a comparison of two different models. Chapter 5 and Chapter 6 describe the experimental setup and the experiments performed on a publicly available dataset to evaluate the approach, respectively. Finally, Chapter 7 gives a conclusion and an outlook on interesting open problems.

2. Background

This work builds on several concepts and technologies used in natural language processing, information retrieval and deep neural networks. The following chapter will introduce the relevant concepts for this work.

The chapter starts with a brief overview of tokenization methods for textual input in Section 2.1. Section 2.2 gives a general introduction into the concept of *attention* before focusing on attention mechanisms in the context of natural language processing. In Section 2.3 the Transformer model [Vaswani et al., 2017] is introduced, which provides the basis for all neural models investigated and used in this work. In addition, the example of the transformer is used to characterize sequence-to-sequence problems and natural language generation in more detail, which will be important for the generator component later. Section 2.4 presents BERT [Devlin et al., 2019] and BART [Lewis et al., 2019], the pre-trained language models used in the implementation. Since various systems for text retrieval and ranking will be introduced either as baselines or as components of the final model, Section 2.5 provides an overview of these approaches.

2.1. Input representation

To process sequential text, it must first be divided into meaningful units or *tokens*. An intuitive approach is to treat individual words as tokens, which is done in models that train semantic word embeddings such as Word2Vec [Mikolov et al., 2013a] and GloVe [Pennington et al., 2014]. A drawback of this approach is the limited vocabulary size of these pre-trained models, which results in rare or unseen words being mapped to a special 'unknown' token. In order to be able to represent arbitrary words even with a limited vocabulary of tokens, subword tokenization schemes have been introduced that reduce the vocabulary space by splitting words into subwords. A popular tokenization algorithm is *Byte Pair Encoding (BPE)* [Sennrich et al., 2016], which performs a statistical analysis of a large corpus of training data to discover frequently occurring subwords. Starting from

tokens of length 1, BPE iteratively merges the most frequent pair of consecutive tokens to produce longer tokens that are added to the vocabulary. Pairs crossing word boundaries are not considered for efficiency. This process is repeated until the desired number of merge operations is completed or until the desired vocabulary size is achieved. The advantage of BPE tokenization (and related methods) is that a relatively small vocabulary (e.g., 30,000 subwords) is sufficient to model large, naturally-occurring corpora that may have millions of unique tokens if a simple method like tokenization by spaces is applied. A similar approach is *WordPiece* [Wu et al., 2016], which was originally proposed for Japanese or Korean segmentation in Google's speech recognition system and uses a slightly modified version of the byte pair encoding algorithm. BPE and WordPiece are used to represent inputs and outputs in the popular Transformer model and its variants, which will be introduced later in this Chapter.

2.2. Attention mechanisms

Attention has become an important concept in neural networks since its first introduction in [Bahdanau et al., 2014], supporting a remarkably large number of applications in natural language processing [Galassi et al., 2020], speech [Cho et al., 2015] and computer vision [Wang and Tax, 2016]. It is inspired by the way information is processed in the human sensory system: we are able to focus our attention on certain interesting objects and ignore irrelevant input in a manner that assists in perception. When inspecting a visual scene, for example, the optic nerve receives information of the order of 10^8 bits per second [Zhang et al., 2020a], which is far beyond what the brain can fully process. Thus, the ability of directing attention to only a fraction of information of interest allows the brain to allocate resources more smartly. Similarly, in various tasks involving the computational processing of language, speech or vision, some parts of the input are more important than others: In translation or summarization tasks, certain keywords in the input sequence will be more relevant than others for predicting the next word of the output sequence. In an image captioning problem, some regions of the input image will be more relevant than others for generating the caption. Figure 2.1 illustrates an example of an image captioning system where the areas of the image that were considered relevant for generating individual words in the description are color-coded.

Key idea of attention mechanisms is to incorporate this concept of relevance by allowing the model to dynamically pay attention only to certain parts of the input that help in effectively performing the task at hand.

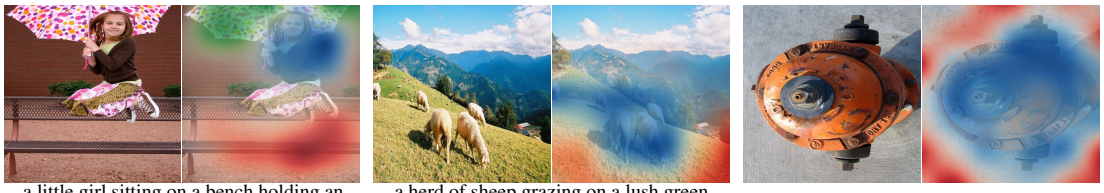


Figure 2.1.: Visualization of generated captions and attended image regions from [Lu et al., 2017]. Same colors show at which image region the network focused its attention during generation of the underlined words.

The current popularity and rapid progress of attention-based neural networks is mainly due to three reasons. First, these models are currently state-of-the-art for multiple tasks, especially in the field of NLP[1]. Second, as a useful side-effect, attention might provide insights into the inner workings of neural models by inspecting which parts of the inputs are considered relevant to a particular output. This potential for improving interpretability of neural networks is considered valuable as the growing impact of machine learning based applications on human life increases research interest in fairness, accountability, and transparency of such models [Chaudhari et al., 2019, Bender et al., 2021]. Finally, they help to overcome some shortcomings of former recurrent systems such as performance degradation with increasing input length or the lack of parallelization, resulting from sequential processing of the input. Attention was first introduced in the field of natural language processing in [Bahdanau et al., 2014] and [Luong et al., 2015] for neural machine translation. Until then, recurrent neural networks (RNNs) were the go-to choice for sequence-to-sequence problems like translation or summarization [Sutskever et al., 2014] due to their sequential nature. Traditionally, these models consist of an encoder and a decoder part where an RNN-based encoder sequentially processes each word in the input sequence and encodes the information into a vector of float values, the *hidden state*. In each step, the hidden state is updated based on the current token and the previous state. Once the entire input sequence has been processed, the hidden state is sent to the decoder to generate the output sequence, again token by token based on the respective hidden state. The main bottleneck of recurrent sequence-to-sequence models is the need to compress the entire content of the source sequence into a fixed-size representation, making the model vulnerable to forgetting long-time dependencies.

Attention alleviates the issue by allowing the model to focus on the relevant parts of the

[1] `http://nlpprogress.com`, retrieved May 2021

input sequence as needed. With attention, not only the last but all previous hidden states are passed from the encoder to the decoder. Each state is the result of processing an input token and can therefore be associated with that token, allowing the decoder to generate a context vector for each decoding step as a weighted sum of all hidden states. The weights are based on learned *alignment scores* which measure the relevance of each input token to generate the next token. Figure 2.2 exemplary visualizes these scores for a sentence which is translated from English to French.

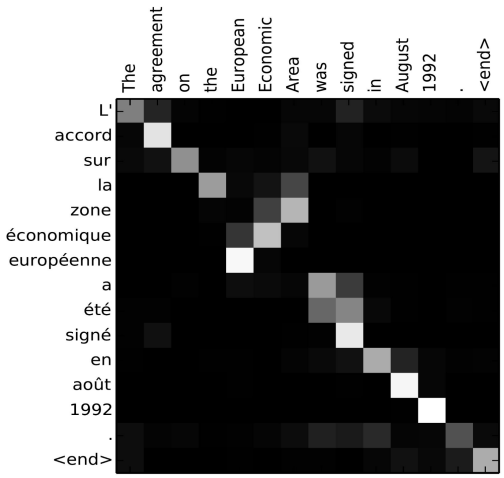


Figure 2.2.: Matrix of alignment scores for a generated translation from English to French from [Bahdanau et al., 2014]. Lighter color denotes higher correlation between source and target words.

2.3. Transformer

Although the attention-based encoder-decoder models described in the last section significantly improved the performance of translation applications, the inherent recurrent architecture prevents effective parallelization which remained a major drawback. To address this problem, in [Vaswani et al., 2017] the authors proposed the *Transformer* architecture, which enabled attention-based sequence-to-sequence modeling without sequential processing and recurrent connections. This work was highly influential, Transformer architecture has become the state-of-the-art approach for many NLP tasks, with multiple variants adopted for a wide variety of problems in recent years (for further details see [Galassi et al., 2020]).

The models and approaches explored in this work are based on the Transformer as well.

The retrieval pre-training procedures proposed in Chapter 4 use two transformer models pre-trained on language-modeling tasks: BERT [Devlin et al., 2019], and BART [Lewis et al., 2019]. Both exploit novel pre-training objectives while the model architecture is based on the key concepts developed for the original Transformer. These concepts such as *self-attention* and *multi-head attention* are discussed in the following section.

2.3.1. Self-attention

The Transformer [Vaswani et al., 2017] relies on a mechanism called *self-attention*. At a high level, self-attention describes the relationship between words in a sentence or a close context. Figure 2.3 exemplary illustrates how self-attention assists in finding information that would otherwise be difficult to detect for a neural network. Self-attention considers all word pairs in a sequence and measures how strongly a word correlates with - or *attends to* - other words, allowing inference of further information in case of ambiguity.

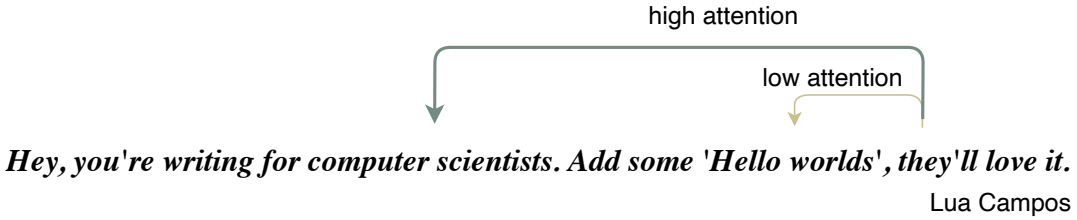


Figure 2.3.: Self-attention allows to determine what the word 'they' refers to.

During the processing of each word of the input sequence, self-attention allows the model to find clues at other positions of the input which provide a better understanding of the current word.

The first step in calculating self-attention for an input sequence is to represent each token in the sequence as a vector of fixed length $\mathbf{x}_i \in \mathbb{R}^d$, where d denotes the dimension of the vector. A sequence of l tokens is mapped to l embedding vectors $\mathbf{x}_1, \ldots, \mathbf{x}_l$ accordingly. Note that these initial token embeddings are learned during pre-training. Each of the input vectors $\mathbf{x}_i$ is projected to a *query* vector $\mathbf{q}_i$, *key* vector $\mathbf{k}_i$ and a *value* vector $\mathbf{v}_i$ through

three trainable weight matrices $\mathbf{W}^Q \in \mathbb{R}^{d \times d_q}, \mathbf{W}^K \in \mathbb{R}^{d \times d_k}, \mathbf{W}^V \in \mathbb{R}^{d \times d_v}$:

$$\begin{aligned}\mathbf{q}_i &= \mathbf{W}^Q \mathbf{x}_i, \\ \mathbf{k}_i &= \mathbf{W}^K \mathbf{x}_i, \\ \mathbf{v}_i &= \mathbf{W}^V \mathbf{x}_i, \ \forall i \in \{1, \dots, l\}.\end{aligned}$$

The idea is to think of the queries and keys as abstractions for calculating attention. Each query vector $\mathbf{q}_i$ is compared to all key vectors $\mathbf{k}_1, \dots, \mathbf{k}_l$. The more similar a key $\mathbf{k}_j$ is to the query $\mathbf{q}_i$, the more relevant is the token at position j (corresponding to the key) to the current token i (corresponding to the query). To calculate a contextualized output representation for token i, an output vector $\mathbf{x}'_i$ is defined as a weighted sum of all value vectors $\mathbf{v}_1, \dots, \mathbf{v}_l$ where the weight assigned to each value is determined by the similarity between $\mathbf{q}_i$ and the respective key vectors $\mathbf{k}_1, \dots, \mathbf{k}_l$. This reflects the idea of attention: the output for a token includes parts of all other tokens in the sentence, focusing on relevant parts while fading out unimportant ones.

Scaled dot-product attention. To compute the attention weights for the weighted sum of values, a scoring function is needed that is in some way able to calculate the relevance of each token with respect to the current token. [Vaswani et al., 2017] choose an operation denoted as *scaled dot-product attention* due to its computational efficiency. The weights are calculated as dot products scaled by the square root of the key dimension d_k which helps to stabilize the gradients during backpropagation according to [Vaswani et al., 2017]. Then, a softmax function is applied to normalize the scores. Thus, the attention score $\alpha_{i,j}$ for a query and a key vector $\mathbf{q}_j, \mathbf{k}_i$ is calculated as:

$$a_{ij} = \text{softmax}\Big(\frac{\mathbf{q}_i \cdot \mathbf{k}_j^\top}{\sqrt{d_k}}\Big) = \frac{\exp(\frac{\mathbf{q}_i \cdot \mathbf{k}_j^\top}{\sqrt{d_k}})}{\sum_{j=1}^{l} \exp(\frac{\mathbf{q}_i \cdot \mathbf{k}_j^\top}{\sqrt{d_k}})} \tag{2.1}$$

In practice, self-attention is calculated in matrix form for faster processing. The embeddings are packed into a matrix $\mathbf{X} \in \mathbb{R}^{l \times d}$ which is multiplied with the weight matrices $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ to obtain query, key and value matrices $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$. The matrix of attention outputs is then computed as:

$$\mathbf{X}' = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\Big(\frac{\mathbf{Q} \cdot \mathbf{K}^\top}{\sqrt{d_k}}\Big)\mathbf{V} \tag{2.2}$$

where the softmax is applied row-wise to the similarity scores.

Multi-Head Self-Attention. As a further enhancement, [Vaswani et al., 2017] proposes to linearly project the input $\mathbf{X}$ to h different sets of queries, keys and values using different learned projection matrices $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V$ for $i \in \{1, \ldots, h\}$ and to perform the attention function in parallel on each of these projections, or *heads*. The mechanism is called *multi-head attention* and allows to perform attention in different 'representation subspaces' focusing on different aspects of the input. Figure 2.4 illustrates the process.

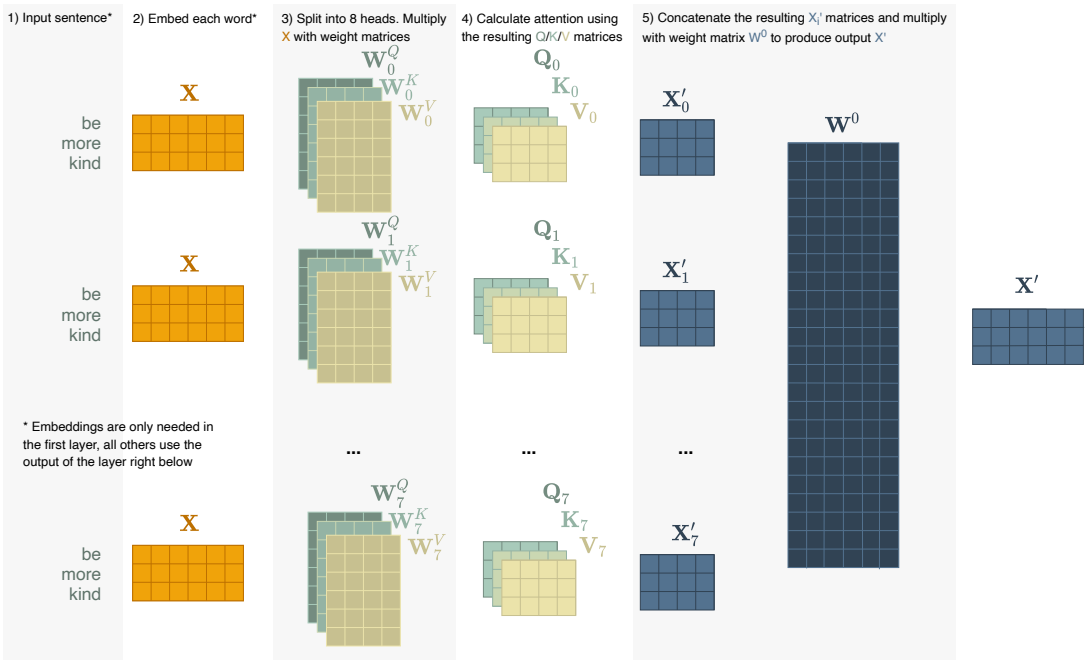


Figure 2.4.: Multi-head self attention. [Alammar, 2018] The attention outputs $\mathbf{X}'_i$ are computed according to Equation 2.2 using 8 different sets of projection matrices $(\mathbf{W}_0^Q, \mathbf{W}_0^K, \mathbf{W}_0^V) \ldots (\mathbf{W}_7^Q, \mathbf{W}_7^K, \mathbf{W}_7^V)$ in parallel. The resulting output matrices $\mathbf{X}'_i$ are concatenated and projected to the original output dimension, so that $\mathbf{X}'$ can serve as input to the next layer.

For each of the h heads, the attention output is computed according to Equation 2.2, the resulting output matrices $\mathbf{X}'_i$ are concatenated and linearly projected to the original dimension d through a trainable weight matrix $\mathbf{W}^O \in \mathbb{R}^{h \cdot d_V \times d}$:

$$\mathbf{X}_i' = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{softmax}\Big(\frac{\mathbf{Q}_i \cdot \mathbf{K}_i^\top}{\sqrt{d_k}}\Big)\mathbf{V}_i \tag{2.3}$$

$$\mathbf{X}' = \text{concat}(\mathbf{X}_1', \ldots, \mathbf{X}_h') \cdot \mathbf{W}^O, \tag{2.4}$$

where $\mathbf{X}_i' \in \mathbb{R}^{l \times d_V}$ and $\mathbf{X}' \in \mathbb{R}^{l \times d}$. The dimension of the query, key and value vectors is commonly chosen as $d_K = d_V = d/h$ for all heads. The reason for applying a reduced size for each head is to keep the computation effort nearly constant, independent of the number of heads.

The resulting output matrix $\mathbf{X}'$ captures information from all attention heads, improving the contextualized representation.

Positional Encodings. Due to the fact that the model only relies on self-attention and contains no recurrence, the order of the words in an input sequence is not taken into account so far. In contrast to recurrent neural networks, where positional information is automatically derived from successive processing of input data, the Transformer needs to inject additional information about the relative or absolute position of the tokens within a sequence. To make use of the word order, a special vector, the *positional encoding* is added to the embeddings. These encodings follow a special pattern that is hypothesized in [Vaswani et al., 2017] to be easily learned by the model to attend to relative positions.

2.3.2. Transformer-based Encoder-Decoder Models

After introducing the basic building blocks of the transformer in the last section, the following section addresses the construction of the model. The Transformer uses an encoder-decoder architecture which is common practice for sequence-to-sequence problems like translation. Assuming that inputs are represented as a sequence of embedding vectors $\mathbf{X}_{1:l} = (\mathbf{x}_1, \ldots, \mathbf{x}_l)$, where $\mathbf{X}_{1:l}$ denotes a sequence of l tokens, a sequence-to-sequence problem can be solved by finding a mapping f from an input sequence $\mathbf{X}_{1:l}$ to a sequence of target vectors $\mathbf{Y}_{1:m}$ where the sequence length m is initially unknown.

$$f : \mathbf{X}_{1:l} \rightarrow \mathbf{Y}_{1:m}$$

To achieve this, the model consists of an encoder and a decoder component, whereby the encoder maps the sequence of input vectors $\mathbf{X}$ to a sequence of contextualized representations or *hidden states* $\overline{\mathbf{X}} = (\overline{\mathbf{x}}_1, \ldots, \overline{\mathbf{x}}_l)$:

$$f_{\text{enc}} : \mathbf{X}_{1:l} \rightarrow \overline{\mathbf{X}}_{1:l}$$

Given the encoder output $\overline{\mathbf{X}}$, the decoder generates an output sequence $\mathbf{Y} = (\mathbf{y}_1, \ldots, \mathbf{y}_m)$ of arbitrary length m depending on the input. Actually, the decoder defines a conditional probability distribution of output sequences given the sequence of encoded input representation:

$$p_{\text{dec}}(\mathbf{Y}_{1:m} | \overline{\mathbf{X}}_{1:l})$$

Using chain rule, the probability distribution for a target sequence $\mathbf{Y}_{1:m}$ can be factorized into a product of conditional distributions for the next token $\mathbf{y}_i$ given the hidden states and all previous tokens $\mathbf{Y}_{0:i-1}$:

$$p_{\text{dec}}(\mathbf{Y}_{1:m} | \overline{\mathbf{X}}) = \prod_{i=1}^{m} p_{\text{dec}}(\mathbf{y}_i | \mathbf{Y}_{0:i-1}, \overline{\mathbf{X}}) \tag{2.5}$$

Thus, if the decoder is able to model the conditional distribution of the next target vector, given the encoded input and all previous output vectors $p_{\text{dec}}(\mathbf{y}_i | \mathbf{Y}_{0:i-1}, \overline{\mathbf{X}})$ for all tokens in $\mathbf{Y}_{1:m}$, then it can model the distribution of any target vector sequence given the hidden state $\overline{\mathbf{X}}$ by simply multiplying all conditional probabilities. This formulation allows the decoder to generate the output sequence *auto-regressively*. Starting from an empty sequence, the output is generated one token after another whereby the decoder input is updated in each step to use the previously generated tokens as additional input. Note that a special *beginning of sentence (BOS)* vector $\mathbf{y}_0$ is introduced in Equation 2.5 to denote the empty sequence in the first step.

Language modeling head. Actually, the outputs of the decoder are outputs of an attention layer. This means, the decoder creates an encoded sequence of float vectors $\overline{\mathbf{Y}}$ instead of the expected conditional probability distribution. To transform $\overline{\mathbf{Y}}$ into probabilities, the stack of decoder layers is followed by a dense linear layer, the *language modeling (LM) head* and a softmax layer. The LM head maps the encoded sequence of target vectors $\overline{\mathbf{Y}}$ to a sequence of *logit* vectors $\mathbf{L} = (\mathbf{l}_1, \ldots, \mathbf{l}_m) \in \mathbb{R}^{m \times v}$ where the dimensionality of each logit vector $\mathbf{l}_i$ corresponds to the number of tokens in the vocabulary v. These logits realize the

mapping to the tokens in the vocabulary: applying a softmax operation to a logit vector $\mathbf{l}_i$, yields a probability distribution that gives, for each token, its probability of being the next token in the output sequence. These distributions are computed for each position in the sequence and define the conditional distribution from Equation 2.5.

The weight matrix of the language modeling head is often initialized as the transpose of the matrix of word embeddings [Press and Wolf, 2017]. Intuitively, this means that for each position $i \in \{1, \ldots, l\}$, the LM head determines the dot product between output vector $\overline{\mathbf{y}}_i$ and each token embedding in the vocabulary. The logit vector thus gives the similarity scores between $\overline{\mathbf{y}}_i$ and each token embedding.

In the following, the architecture to realize the transformer-based encoder and decoder is described in more detail. Section 2.3.3 shows how self-attention is used to generate contextualized encodings $\overline{\mathbf{X}}$ for the input sequence in the encoder and Section 2.3.4 describes how the decoder models the probability distribution for a target sequence $p_{\mathsf{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:l})$. Finally, the auto-regressive generation of an actual output sequence is described.

2.3.3. Encoder

The transformer-based encoder is composed of a stack of identical encoder blocks. The input to the first block is the list of token embeddings for the input sequence with added positional encoding, in the following blocks the output of the previous layer serves as input until the last encoder block outputs the contextualized representations $\overline{\mathbf{X}}$. Figure 2.5 provides an overview over the sublayers.

The key component of each of these blocks is a multi-head self-attention layer that relates every input vector $\mathbf{x}_i$ to all input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_l$ and by doing so, generates a refined, contextualized representation $\mathbf{x}'_i$ of the same dimension, as described in Section 2.3.1. A residual connection is employed around the self-attention layer, followed by a layer-normalization step [Ba et al., 2016]. The normalized attention output is fed to a fully connected feed-forward neural network, which is applied to each position separately and identically. Again, the feed-forward network is surrounded by a residual connection and followed by a layer-normalization.

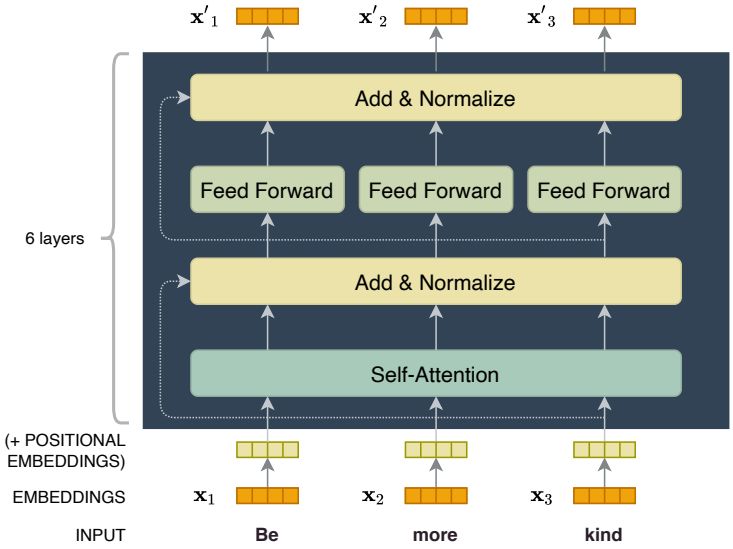


Figure 2.5.: Structure of an encoder block. Each block consists of a self-attention and point-wise feed-forward layer, where both sub-layers are surrounded by residual connections and followed by layer normalization.

2.3.4. Decoder

As described in Section 2.3.2, the decoder models the conditional probability distribution of a target sequence $\mathbf{Y}$, given the contextualized encoded inputs $\overline{\mathbf{X}}$ as the product of conditional distributions of the next target vector, according to Equation 2.5. In each step, the encoder generates probabilities for the next token in the sequence, conditioned on the encoded input and the previously generated target tokens.

To do so, the decoder uses a similar architectural design to the encoder also consisting of a stack of identical decoder blocks. The original Transformer uses 6 layers in both encoder and decoder. A decoder block employs similar sub-layers to the encoder - a multi-head self-attention layer and a point-wise feed-forward layer, both surrounded by residual connections and followed by layer normalization. In addition, a third sub-layer is inserted in the decoder block: the *encoder-decoder attention* or *cross-attention layer*, which performs multi-head attention over the output of the encoder stack. Again, residual connections and layer normalization are used. Figure 2.6 provides an overview.

Since the objective of the decoder is language generation, the self-attention layer in the decoder is only allowed to attend to earlier positions in the target sequence. This is imple-

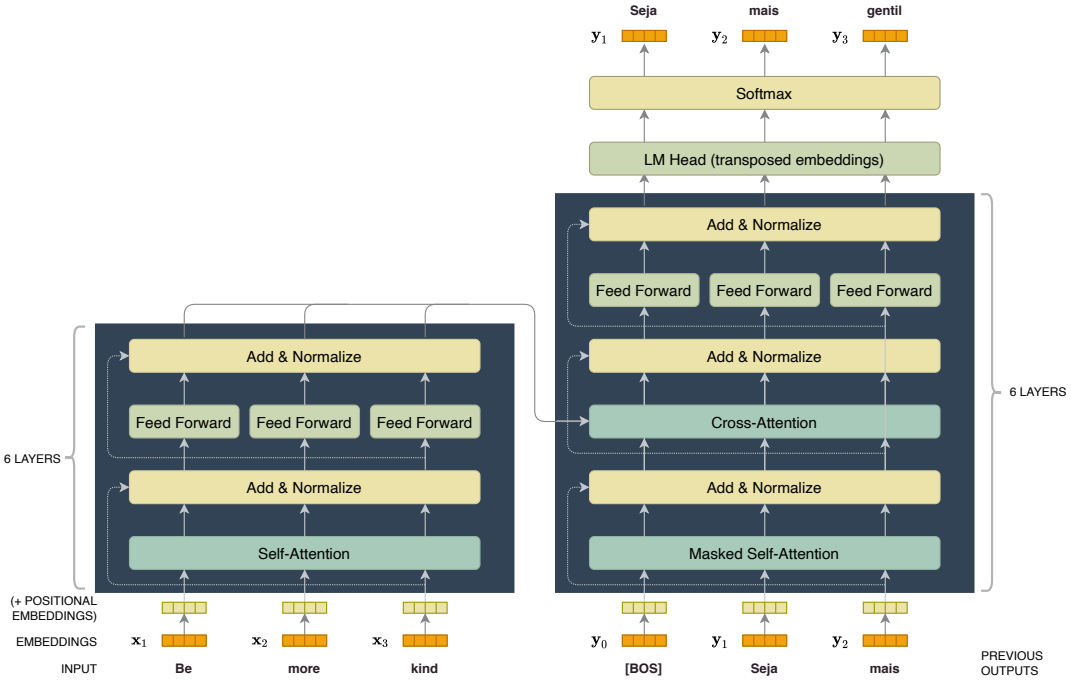


Figure 2.6.: Architecture of the transformer-based encoder (left) and decoder (right), exemplarily illustrated for translation. Compared to the encoder block, the decoder block contains an additional sublayer, performing cross-attention between queries created from the underlying block and keys created from the encoder outputs. The self-attention layer masks out future positions to realize forward modeling. The Figure illustrates the 3^{rd} step during auto-regressive generation: From the encoder outputs $\overline{\mathbf{X}}$ and the first input vectors $\mathbf{y}_{0:2}$, a prediction for $\mathbf{y}_3$ is modeled.

mented by masking out the following tokens setting their attention scores to $-\infty$ before the softmax operation.

The cross-attention layer performs multi-head attention as described in Section 2.3.1, except that the matrix of queries $\mathbf{Q}$ is created from the decoder layer below while the keys $\mathbf{K}$ and values $\mathbf{V}$ are taken from the output of the encoder. Thus, for a target sequence $\mathbf{Y}$, the cross-attention in each head $i \in \{1, \ldots, h\}$ is computed as

$$\mathbf{Q}_i = \mathbf{W}_i^Q \cdot \mathbf{Y} \tag{2.6}$$

$$\mathbf{K}_i = \mathbf{W}_i^K \cdot \overline{\mathbf{X}} \tag{2.7}$$

$$\mathbf{V}_i = \mathbf{W}_i^V \cdot \overline{\mathbf{X}} \tag{2.8}$$

$$\text{Cross-Attention}(\mathbf{Q}_i, \mathbf{K}i, \mathbf{V}_i) = \text{softmax}\Big(\frac{\mathbf{Q}_i \cdot \mathbf{K}_i^\top}{\sqrt{d_k}}\Big) \cdot \mathbf{V}_i \tag{2.9}$$

Training The training objective for the Transformer is *forward language modeling*. Given an input sequence $\mathbf{X}$ and a corresponding target sequence $\mathbf{Y}$, the model is trained to traverse $\mathbf{Y}$ left-to-right and predict the next token $\mathbf{y_i}$ for each position $i-1$, based on the encoded input $\overline{\mathbf{X}}$ and the part of the target sequence $\mathbf{Y}_{0:i-1}$ that has already been read.

The model is trained on a labeled training dataset, where pairs of input and corresponding target sequences are presented to the network and the likelihood to generate the target for a given input is maximized.[Vaswani et al., 2017] used pairs of english and german respectively english and french sentences. Recall that the model output is a sequence of logit vectors $\mathbf{L} = (\mathbf{l}_1, \ldots, \mathbf{l}_m) \in \mathbb{R}^{m \times v}$ where each logit $\mathbf{l}_i$ models the next token probability distribution from Equation 2.5 over all tokens in the vocabulary for the corresponding position i. To maximize this probability for a given token sequence, the *language modeling loss* is defined as the negative log likelihood of the sequence which is minimized during training. This loss is summed over all tokens of the sequence

$$\text{Loss}_{LM} = \sum_{i=1}^{l} -\log p(\mathbf{y_i} | \mathbf{Y}_{0:i-1}, \overline{\mathbf{X}}), \tag{2.10}$$

and minimized during training using stochastic gradient descent.

Besides the *forward language modeling* objective, various further pre-training objectives were developed for transformer models. Examples are *masked language modeling (MLM)*

and *next sentence prediction* introduced in BERT [Devlin et al., 2019], or *text infilling* and *sentence permutation*, developed for BART. For a more detailed description of other objectives, refer to Section 2.4.

Auto-regressive text generation Aim of this work is to repurpose a retriever-augmented generation model for self-supervised retriever training. Nevertheless, in the experiments in Section 6, not only the performance of the retriever is evaluated, but also the quality of the generated text sequences. Generation of an actual text means to sample a token sequence with high probability from the conditional distribution $p_{\text{dec}}(\mathbf{Y}|\overline{\mathbf{X}})$. This is done in an *auto-regressive* way. The model outputs one token after another and in each step, the new token becomes part of the model's input in the next step. Since the sequence length m is initially unknown, a special *end-of-sequence (EOS)* token is introduced allowing to determine m on-the-fly as the time step when the EOS token is generated. Obviously, the space of possible sequences is large, so that during inference, an efficient algorithm is needed for decoding the probability distribution into a concrete sequence of output tokens. Some common methods are briefly presented below.

Greedy search. As the name suggests, greedy search selects the token with the highest probability as next token at each time step i.

$$\mathbf{y}_i = \text{argmax}_y \, p_{\text{dec}}(\mathbf{y}|\mathbf{Y}_{1:i-1}, \overline{\mathbf{X}})$$

As with other greedy algorithms, a drawback is that the search misses high probability tokens hidden behind low probabilities. Moreover, the algorithm is prone to getting stuck in repetitive loops where a short sequence of tokens is repeated endlessly as shown in [Vijayakumar et al., 2018].

Beam search. To reduce the risk of getting trapped in a local optimum, beam search keeps the n most likely hypothesizes at every time step during generation until all of them reach the EOS token. The hypothesis with the highest overall probability is chosen as output. Depending on the number of beams n, beam search is more likely to find a sequence with high probability but repetitive generation remains a concern since most of the model's attention is on the most recent few tokens [Yang et al., 2018].

Temperature sampling. [Holtzman et al., 2020] state a additional weakness of greedy and beam search: Following a distribution of high probability next words is not a suitable representation of human language due to the absence of 'surprising' next words. To address this problem, methods were introduced into language generation, which sample next words from the conditional probability distribution.

$$\mathbf{y}_i \sim p_{\text{dec}}(\mathbf{y}|\mathbf{Y}_{1:i-1}, \overline{\mathbf{X}})$$

To prevent the model from sampling extremely unlikely tokens too often, the adjusting *temperature* parameter is introduced: low temperatures increase the models confidence in its top choices while infinite temperature corresponds to uniform sampling.

Top k sampling. [Fan et al., 2018] introduce a sampling scheme, which selects the k most likely next tokens and redistributes the probability mass among those k tokens. This eliminates more unlikely candidates by setting the probabilities for every other token to zero.

Top p (nucleus) sampling. Instead of sampling from the most likely k words, top p sampling defines a probability p as bound and chooses from the smallest set of words whose cumulative probability exceeds this probability. The method provides a better response to the probability distribution of the next token: It avoids choosing unlikely next tokens when only few tokens share most of the probability mass while it preserves variety when the most likely tokens have low confidence [Holtzman et al., 2020].

2.4. Pre-trained language models

Transformer-based models became recently extremely popular in NLP, mainly because of their applicability in *transfer learning*. Classical supervised training in machine learning requires a large number of training examples which are often not directly available. Transfer learning extends this approach to a two stage process: in the *pre-training* phase, the model learns general representation for a source task unsupervised on a large text corpus. In the *fine-tuning* phase, these representations are adapted to a specific target task with supervised training, but requiring far less training samples than training from scratch.

In the field of NLP, transfer learning first gained attention with unsupervised pre-training of *word embeddings*, dense real-valued vector representations of the distribution of words. These word vectors came from systems such as word2vec [Mikolov et al., 2013b] and GloVe [Pennington et al., 2014] and later LSTM models such as context2vec [Melamud et al., 2016] and could be shown to improve performance across a diverse range of downstream tasks when used as initialization for the fine-tuning step.

Even more influential was the introduction of neural networks pre-trained on language modeling objectives like ULMFiT [Howard and Ruder, 2018], ELMo [Peters et al., 2018] or the Transformer [Vaswani et al., 2017]. Given a sequence of words, language modeling (LM) aims to predict the probability for the next word, which was found to be a powerful pre-training task, requiring the model to learn syntactic and semantic information simultaneously. Furthermore, language models have been shown to be versatile and capable of learning both sentence and word representations with a variety of objective functions [Ruder et al., 2019]. The original Transformer [Vaswani et al., 2017] was followed by a variety of transformer-based neural language models, that can be broadly categorized into two groups from an architectural perspective. Models like T5 [Raffel et al., 2019], Bart [Lewis et al., 2019], Pegasus [Zhang et al., 2020b] or Marge [Lewis et al., 2020a] explore different pre-training objectives for transformer-based encoder-decoder models and adopt the model architecture largely unchanged. A second line of work replaces the original encoder-decoder architecture with a single-stack architecture. Examples include BERT [Devlin et al., 2019] and its variants [Liu et al., 2019, Lan et al., 2020, Sanh et al., 2020], which employ a single stack of Transformer encoder layers to generate embeddings, and GPT-2/3 [Radford et al., 2019, Brown et al., 2020], using a stack of masked self-attention layers for auto-regressive text-generation. Today, reference implementations and model checkpoints pre-trained on massive datasets are publicly available for most of these models[2], allowing to use the pre-trained language models as components during development of neural language processing systems.

In this work, two pre-trained models will be used und further refined: BERT [Devlin et al., 2019] as a transformer-based model for the creation of contextualized embeddings and BART as a *sequence-to-sequence* model for language generation. In the following, these models are presented in more detail.

[2] See e.g. huggingface, a popular open source library for NLP.

2.4.1. Masked language modeling: BERT

Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] was developed by Google AI and is currently one of the most popular systems for generation of contextualized word or document embeddings. BERT obtained state-of-the-art results in numerous benchmarks, while it is still an open question why it performs so remarkably well. Actually, there exist more than 150 works in the emerging field of *BERTology*, investigating what kind of information BERT learns about linguistic structure from the unsupervised language modeling task (see [Rogers et al., 2020] for a survey). BERT consists of a stack of transformer encoder layers as depicted in Figure 2.5 and is pre-trained on language modeling. The model is presented in two sizes, both larger than the initial Transformer: BERT$_{BASE}$ is constructed of 12 layers, while BERT$_{LARGE}$ uses 24. The hidden size d is 768 and 1024 respectively and the number of attention heads h is set to 12 for BERT$_{BASE}$ and 16 for BERT$_{LARGE}$. A special classification token [CLS] is prepended to each input sentence before the input is processed as described in Section 2.3.3. The [CLS] token is embedded to a vector $\mathbf{v}_0$ whose contextualized representation $\overline{\mathbf{v}}_0$ serves as an encoded representation for the input sequence. As the name suggests, BERT is trained in a *bidirectional* way. To enable bi-directional training, BERT adopts the *masked language modeling* task where randomly selected tokens are replaced by a special [MASK] token and the model learns to predict the masked token. 10% of the training samples use a random token instead of the [MASK] token and in another 10% of the training data the selected token remains unchanged. The pre-training process includes a second task, *next sentence prediction*, to train the model on relationships between sentences. Given two sentences, the task is to predict whether the second sentence follows the first in the original text. Both tasks are pre-trained on a combination of English Wikipedia and the BooksCorpus [Zhu et al., 2015].

2.4.2. Seqence-to-sequence: BART

BART [Lewis et al., 2019] was developed by Facebook AI as a denoising autoencoder for pre-training sequence-to-sequence models. BART is particular effective when fine-tuned for text-generation tasks like abstractive dialogue, question-answering, translation and summarization. The model uses the standard Transformer encoder-decoder architecture as shown in Figure 2.6 in two model sizes. The base model uses 6 layers in the encoder and decoder, a hidden size of $d = 768$ and 8 attention heads, BART$_{LARGE}$ uses 12 layers in each, hidden size of $d = 1024$ and 16 attention heads. Following common practice in language model pre-training, BART is trained on a large corpus of text where text passages are corrupted

with different noising function and the model learns to reconstruct the original text. The transformations used for corrupting the original text can be seen as generalization of BERTs masked language modeling and next sentence prediction objectives. Additional to the *token masking* task, where a single token is masked, BART introduces *token deletion* and *text infilling*, where spans of arbitrary length (including $l = 0$) are replaced with a single [MASK] token. Additional tasks on larger text passages are *sentence permutation* where a document is divided into sentences which are shuffled in random order and *document rotation* where documents are rotated so that a randomly chosen token becomes the first token in the corrupted document. For all tasks, the corrupted text is processed by the model as described in Section 2.3.2 using the original text as target for the autoregressive decoder. The model is trained by optimizing the negative log likelihood of the original.

2.5. Representations for text retrieval and ranking

The following section discusses different approaches for retrieval and ranking. Section 2.5.1 addresses retrieval models using sparse bag-of-word models to represent texts while Section 2.5.2 discusses learning based approaches using dense representations.

The goal of text ranking is to generate an ordered list of texts retrieved from a corpus containing relevant information regarding a query for a particular task. The most common formulation of text ranking is search, where the search engine outputs a ranked list of texts like web pages, scientific papers, news articles or tweets ordered by estimated relevance with respect to the query. Nevertheless, text retrieval and ranking is also a core component of many language processing applications such as question answering or recommendation systems. A major challenge for ranking models is that relevance, as a key concept in IR, is often vaguely defined and difficult to estimate. While classical information retrieval focused on heuristic weights for sparse bag-of-words representation [Jones, 2004] to estimate relevance between document and query, more recent work introduce transformer-based neural into IR. Usually, these neural retrieval systems use a multi-stage architecture, where a first-phase retriever selects a set of candidates that are ranked in a second step using a more sophisticated but computationally expensive procedure. In this work, both BM25 as a sparse retrieval model and dense retrieval using BERT will be used. In the following, a brief overview over the different approaches is given.

2.5.1. Sparse retrieval models

Classical information retrieval systems used sparse bag-of-word models to represent queries and documents. Typically, each query and each document is represented as a vector $\mathbf{q}, \mathbf{d} \in \mathbb{R}^v$ where v is the vocabulary size and the values at each vector position are derived from the number of occurrences of the corresponding token in the text. To denote the relevance between a query and a document vector, the most straightforward approach is to use their dot product $\mathbf{q} \cdot \mathbf{d}$ as relevance score. To improve the ranking, more sophisticated ranking functions assign weights to terms. The most popular term-weighting scheme is *TF-IDF (term frequency - inverse document frequency)*. Let $f(q_i, \mathbf{d})$ denote the raw count of term q_i in document $\mathbf{d}$. The *term frequency* is then defined as the occurrences of a term q_i in a document $\mathbf{d}$ adjusted by the document length:

$$\mathsf{TF}(q_i, \mathbf{d}) = \frac{f(q_i, \mathbf{d})}{\sum_{t \in \mathbf{d}} f(t, \mathbf{d})} \tag{2.11}$$

The *inverse document frequency* $\mathsf{IDF}(q_i)$ is the inverse of the fraction of documents in which the search term q_i occurs in the corpus, penalizing terms that are common. The intuition here is that rare keywords are more important for the search results and should be given greater consideration than frequent words.

$$\mathsf{IDF}(q_i) = \log\Big(\frac{N}{\sum_{\mathbf{d}: q_i \in \mathbf{d}} 1}\Big), \tag{2.12}$$

where N is the total number of documents in the corpus.

The product of both is denoted as $\mathsf{TF\text{-}IDF}(q_i)$, measuring the frequency of term q_i in a document $\mathbf{d}$, normalized by the fraction of documents containing the term:

$$\mathsf{TF\text{-}IDF}(q_i) = \mathsf{TF}(q_i, \mathbf{d}) \cdot \mathsf{IDF}(q_i, \mathbf{d}) \tag{2.13}$$

The retrieval function actually used for sparse retrieval in this work is *BM25* [Robertson and Zaragoza, 2009], which is defined for a query $\mathbf{q}$, tokenized into terms $q_1, \ldots, q_n$ and a document $\mathbf{d}$ as:

$$score(\mathbf{d}, \mathbf{q}) = \sum_{i=1}^{n} \mathsf{IDF}(q_i, \mathbf{d}) \cdot \frac{f(q_i, \mathbf{d}) \cdot (k1 + 1)}{f(q_i, \mathbf{d}) + k1 \cdot \Big(1 - b + b \cdot \frac{|\mathbf{d}|}{l_{avgd}}\Big)}, \tag{2.14}$$

where $|\mathbf{d}|$ denotes the number of terms in $\mathbf{d}$ and l_{avgd} is the average document length, again based on the number of terms per document. The fraction $\frac{|\mathbf{d}|}{l_{avgd}}$ in the denominator of Equation 2.14 causes longer documents (longer than the average) to reduce the BM25 score, while shorter documents contribute to a higher score. The ratio of the document length is multiplied by a free parameter b, which serves as a scaling factor for the discussed effect. If b becomes larger, the effect on the score is increased, while it is decreased for small values. The default value for b is 0.75, which will be used in this work as well. Finally, the variable $k1$ determines the term frequency saturation. It defines an asymptotic upper bound for the effect of the term frequency $f(q_i, \mathbf{d})$ on the score. While for terms that rarely appear in the document, a higher term frequency significantly increases the score of the document, above a certain frequency further increases no longer have an impact.

BM25 is chosen as strong baseline for the experiments. [Luan et al., 2021] shows that BM25 often outperforms a dual encoder based on BERT, particularly on longer documents or task that requires precise detection of word overlap.

2.5.2. Learned dense representations

While practical IR applications continue to be dominated by sparse systems, current research focuses on neural ranking models (often denoted as *learning to rank (LTR)*). Among them, a recent approach has emerged that fine-tunes deep pre-trained language models like BERT [Devlin et al., 2019] to estimate relevance. By using contextualized representations for queries and documents, these models are better at handling vocabulary mismatch between query and document [Mitra and Craswell, 2018].

Transformer based ranking models can be roughly divided into three groups according to their query-document matching paradigm [Lin et al., 2020]: cross-encoders following an interaction-focused approach, dual encoder that follow a representation-based approach and systems that combine both paradigms. Figure 2.7 illustrates the differences. In this work, the dual encoder approach will be used.

Interaction-focused approaches using cross-encoders For each pair of query and passage, both texts are tokenized, concatenated using a special separator token and the resulting sequence is fed to a transformer-based language-model to generate a contextualized representation. Finally the [CLS] vector is used as input to a linear classification layer to compute the probability of the passage being relevant. By performing full (cross) self-attention

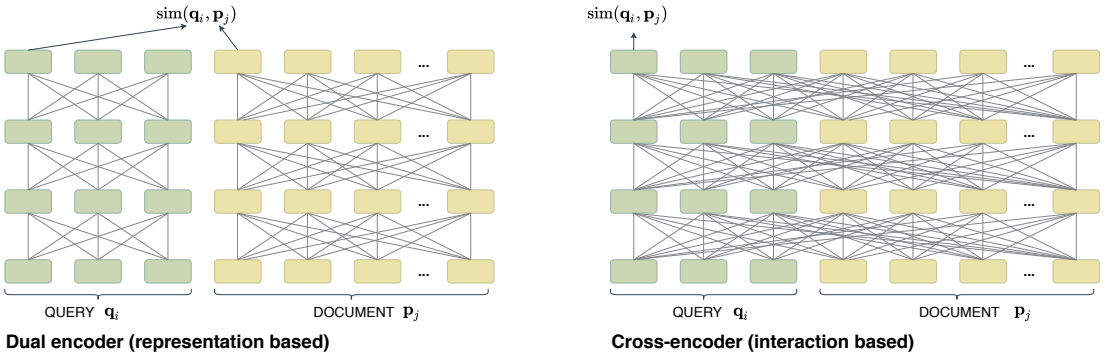


Figure 2.7.: Schematic diagrams illustrating query-document matching approaches in neural IR. Adapted from [Khattab and Zaharia, 2020]

over query and document, the representation can model fine-granular local interactions between the two. While bag-of-words models depend on term overlap, representation-based models deal well with lexical mismatch between query and passage, leveraging the implicit knowledge of the language model. Neural ranking models based on BERT [Devlin et al., 2019], such as [Dai and Callan, 2019, MacAvaney et al., 2019a, Nogueira and Cho, 2020, Akkalyoncu Yilmaz et al., 2019] have achieved state-or-the-art results on various benchmarks. However, cross-encoder based retrieval requires extensive computation on each candidate document, BERT-based models in the literature are 100 to 1000 times more computationally expensive than prior models according to [Hofstätter and Hanbury, 2019]. Therefore, the approach is only feasible for re-ranking in two-stage architectures.

Representation based ranking with dual encoders Dual encoder [Huang et al., 2013, Reimers and Gurevych, 2019, Gillick et al., 2019, Karpukhin et al., 2020] provide an alternative that leverages the advantages of transformer-based representations without the computational effort of full cross-attention. The embeddings for queries and documents are computed independently and a retrieval score is computed for each document as the inner product between its encoding and that of the query. Since the document embeddings can be computed offline, dual encoder can be easily applied to very large document collections.

3. Related Work

The following chapter addresses *retrieval-augmented language modeling*, the starting point of this thesis, and gives an overview of current work in the field.

It has been shown that large scale transformer-based language models store a surprising amount of factual information [Radford et al., 2019, Petroni et al., 2019, Jiang et al., 2020] which can be extracted for knowledge intensive NLP tasks such as open domain question answering. However, to improve their accuracy, models get larger and larger. Current state-of-the-art contains billions of parameters to store all the information needed in the model weights, rendering training and querying the model expensive. To capture information in a more modular, flexible and interpretable way, a line of work evolved, that follows a different approach to improve performance of pre-trained language models. The idea is to give the system access to an external knowledge source, such as Wikipedia, to perform an intermediate retrieval step during inference. The input sequence is used to retrieve a set of relevant documents from the knowledge base which are used as additional information to generate the actual output. Thus the model has two sources of knowledge: the knowledge stored in the parameters of the model (parametric memory) and the knowledge stored in the corpus from which the supporting passages are retrieved (non-parametric memory). These two sources complement each other by combining 'closed-book' or parametric-only approach with 'open-book' or retrieval-based methods. This thread of work forms the basis for the unsupervised retriever pre-training proposed in this work. The following is an overview over related work in the field, with a more detailed presentation of work relevant to our approach such as REALM [Guu et al., 2020], RAG [Lewis et al., 2020b], and MARGE [Lewis et al., 2020a].

Retrieval-augmented language modeling. The first works to integrate text retrieval and text ranking directly into neural network training are [Miller et al., 2016] and [Chen et al., 2017]. Here, non-learned retrievers are used to select informative documents from large document collections to transfer open-domain question answering to reading tasks.

Khandelwal et al. propose a k-Nearest-Neighbor Language model (kNN-LM) in [Khandelwal et al., 2020b], a method that adds a retrieval step to language model pre-training. Here, the nearest neighbors in the pre-trained LM-embedding space are looked up to generate target word predictions. The method achieves good results in terms of perplexity, but the authors do not evaluate on downstream tasks. Other language models that access external knowledge, implementing retrieval in a heuristic fashion are [Min et al., 2019] and [Asai et al., 2020]. Both models are fine-tuned and evaluated on Open-Domain Question Answering (Open-QA).

The direct precursor of REALM is [Lee et al., 2019], which describes the idea to jointly learn a retriever and reader for Open-QA for the first time. To determine which documents from the corpus are relevant for a given query, the retriever pre-computes vector embeddings for documents and query using BERT-style transformers and determines the relevance score between document and query as the inner product of the embeddings. To efficiently find the approximate top k documents, the embeddings are pre-computed and a Maximum Inner Product Search (MIPS) algorithm is used. The retriever is trained by treating the retrieved documents as latent variables and optimizing the marginal likelihood. Here, an additional pre-training step is introduced to train the embeddings for documents and queries. During end-to-end training only the embeddings of the queries are updated, the precomputed and indexed document embeddings remain unchanged. REALM [Guu et al., 2020] extends this approach to train both query and document embeddings by periodically re-embedding and re-indexing the document base during end-to-end training. Further contributions involve better pre-training methods and computational tricks, yielding impressive empirical results on Open-QA. [Balachandran et al., 2021] presents a study of REALM and proposes several enhancements for the training and inference setup leading to further Open-QA accuracy improvements without changes in the model design.

A different approach for augmenting language model pre-training with information retrieval is presented in [Wu et al., 2020]. Wu et al. state that in large corpora, many words only appear rarely, which leads to poorly optimized embeddings for these words. They propose to maintain a dictionary which collects context information (sentences) for such rare words. When a word is masked and predicted again during masked language model training, the stored sentence is used to enhance the semantics of the current sentence. The authors argue that by employing cross-sentence information for rare words, the model provides a better data utilization which reduces training time significantly.

Retrieval-augmented generation. A second line of work investigates retrieval augmented text generation and combines a differentiable retrieval component with a sequence-to-sequence model. The first work to take this approach is *Retrieval Augmented Generation (RAG)* [Lewis et al., 2020b]. It describes an end-to-end differentiable model which combines a pre-trained neural information-retrieval component (DPR [Karpukhin et al., 2020]) with a sequence-to-sequence generator (BART [Lewis et al., 2019]). The model behaves like a standard sequence-to-sequence model. It accepts a textual sequence as input and outputs a corresponding sequence. In an intermediary step, RAG retrieves a set of top-k documents relevant to the input from a given knowledge base, which are then concatenated with the original inputs and fed to the generator to produce latent outputs per document. To integrate knowledge from all retrieved documents, the individual predictions are marginalized either on a per-document basis or a per-token basis where different documents are responsible for different tokens. Weighting the latent documents with their retrieval scores during marginalization, allows both the retriever and generator to be trained jointly.

In [Izacard and Grave, 2020b] Gautier Izacard and Edouard Grave develop an approach for open domain question answering which they call *Fusion-In-Decoder*. The method follows two steps: in a first step, passages with supporting information for the input query are retrieved using either BM25 [Robertson et al., 1995] or DPR [Karpukhin et al., 2020] as sparse or dense retrieval component, respectively. Then, the retrieved passages serve as additional input to a sequence-to-sequence model to generate the answer. The authors use T5 [Raffel et al., 2020] as the generative model, an encoder-decoder model pre-trained on a combination of several unsupervised and supervised tasks. Each retrieved passage is concatenated with the question and processed independently by the T5 encoder. Finally the resulting representations of all passages are concatenated and fed to the T5 decoder which performs attention over the resulting representation of all passages. According to [Izacard and Grave, 2020b], processing the context passages independently in the encoder and jointly in the decoder improves the scalability of the model. In [Izacard and Grave, 2020a], a follow-up work to [Izacard and Grave, 2020b], the authors propose a procedure which uses their Fusion-in-Decoder model to refine a retriever without strong supervision. Similar to [Izacard and Grave, 2020b], the model implements a two-step process to generate an answer for a given question by first selecting support passages from a knowledge base which are then processed by the reader to generate the answer. The work aims to train the retriever module to find the most relevant passages without strong supervision. For this purpose, the authors assume that the attention scores of the reader are a good proxy for the usefulness of a passage - the more the tokens in a text segment are attended to, the more relevant this segment is to answer the question. In an iterative training process, the

attention scores of the reader are aggregated to form a synthetic relevance score for each input document and the retriever is trained to estimate these scores.

In *MARGE, a Multilingual Autoencoder that Retrieves and Generates* [Lewis et al., 2020a], the authors present a model that jointly trains retrieval and generation on an unsupervised multi-lingual paraphrasing task. The model is trained to reconstruct a target text by first retrieving a set of related texts in different languages and then conditioning on them to maximize the likelihood of generating the original. Similar to [Izacard and Grave, 2020b], MARGE uses a conventional sequence-to-sequence Transformer model [Vaswani et al., 2017], that encodes each retrieved text passage separately and performs cross attention on the concatenated encoded sequences in the decoder to generate the target. To retrieve related texts, the normalized output of a certain encoder layer is used as embedding for queries and documents and the relevance score is computed as cosine similarity between these representations. These retrieval scores are used in the decoder to bias the cross attention to the most relevant retrieved documents which allows to train retrieval and generation model jointly by minimizing the reconstruction loss.

Retrieval-augmented models for other tasks. The models presented so far applied retrieval-augmented procedures to language modeling [Khandelwal et al., 2020b], and question-answering [Guu et al., 2020, Lewis et al., 2020b, Lewis et al., 2020a, Izacard and Grave, 2020b], which is an obvious choice as a knowledge-intensive task. The following works use non-parametric models for tasks such as machine translation and conversational tasks. In [Khandelwal et al., 2020a] the authors propose to combine pre-trained neural machine translation (NMT) model with token-level k-nearest-neighbor (kNN) retrieval to improve the translation accuracy. [Zheng et al., 2021] further refines the approach by using an adaptive retrieval method dynamically determining the number of k for each target token.

[Shuster et al., 2021] explores retrieval-augmented generation for open-domain knowledge-grounded dialogue. The authors state as their main findings that retrieval-augmented models significantly reduce the knowledge 'hallucination' problem [Maynez et al., 2020] in conversational agents, where plausible looking statements are generated that are factually incorrect. Further, the authors hypothesize that the retrieval step helps in generalizing beyond the training data to previously unseen distributions.

4. Approach

This work investigates retrieval-augmented language modeling with the objective of improving the retrieval component. Current neural retrieval models [Nogueira and Cho, 2019, Yang et al., 2019, Nogueira et al., 2019] are based on pre-trained language models and pre-trained in a supervised manner. This has only become possible since large amounts of training data, annotated for retrieval [Craswell et al., 2020] have become available in recent years. In practice, however, suitable labeled training data may not be available for domain-specific applications. Annotation of training data is typically performed by human annotators and as such an expensive and time consuming task. Since training a neural network requires a certain amount of labeled training samples, this can only be obtained at considerable expense. To get rid of this requirement, this work seeks ways to train an IR system in an unsupervised manner. Based on an end-to-end differentiable system, consisting of a neural retriever and a neural generator, a procedure is suggested to learn a retrieval system without any supervision in terms of pairs of queries and related passages.

The approach is inspired by MARGE, a retrieval-augmented language model which is pre-trained by multi-lingual paraphrasing. MARGE is trained to reconstruct a given target text by first retrieving a set of related texts in various languages and then conditioning on them to maximize the likelihood to generate the original text. For more details, refer to [Lewis et al., 2020a]. In this work, the multi-lingual aspect of the original paper is discarded, allowing to train much smaller models than MARGE which contains about 960M parameters. Instead, it is investigated, to what extent the idea of reconstructing a text from a set of related passages can be used to train the retrieval component without supervision.

The key intuition is to train the retriever using a performance-based signal from the generation process: retrieval of a passage that increases the probability of reconstructing the original input is considered helpful and should be rewarded while uninformative retrieval should be penalized. Thus, the retrieval function is learned indirectly while optimizing reconstruction which significantly improves the retrieval accuracy as shown by the experiments in Section 6.3.2. The result of the training process is a pre-trained retrieval model

that can later be fine-tuned for downstream tasks.

4.1. Architecture

This Section discusses the architecture of the proposed models. Figure 4.1 provides an overview of the individual components.

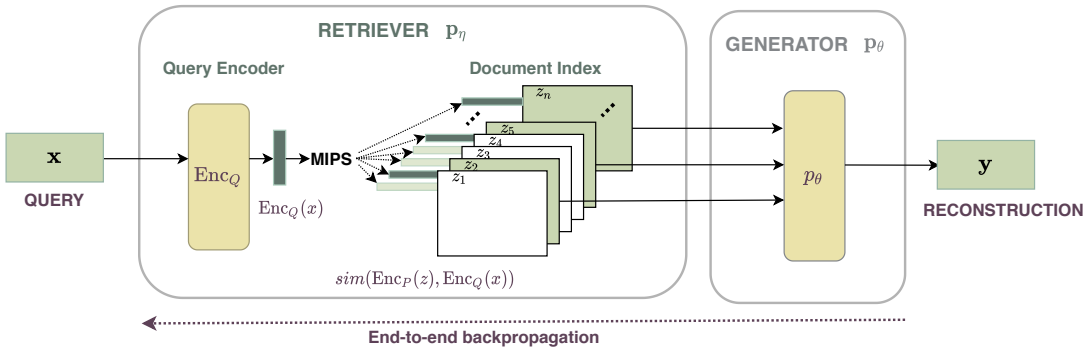


Figure 4.1.: Overview of the model architecture. The retriever selects a set of related passages for a query, which are used by the generator to reconstruct the original query. Both components are trained jointly end-to-end to maximize the likelihood of reconstructing the original query.

The self-supervised pre-training is modeled as a two-step process which can be described as *retrieve-and-generate*. The model accepts an input text $\mathbf{x}$ and retrieves possibly helpful documents $\{\mathbf{z}_1, \ldots, \mathbf{z}_k\}$ from a knowledge corpus $\mathcal{Z}$. This corpus is usually a collection of unstructured textual information such as Wikipedia, news articles or domain-specific data, split into text segments of equal length. In the following, the terms document or passage refer to such chunks of text. Retrieving a document from the corpus is modeled as sampling from a probability distribution $p(\mathbf{z}|\mathbf{x})$ where the probability to draw a support document $\mathbf{z}$ for a given input $\mathbf{x}$ should reflect the similarity between $\mathbf{x}$ and $\mathbf{z}$. Then, in a second step, the model computes the likelihood $\mathcal{L}(\mathbf{x}|\mathbf{z}_1, \ldots, \mathbf{z}_k)$ to reconstruct the original input $\mathbf{x}$ from a set of documents $(\mathbf{z}_1, \ldots \mathbf{z}_k)$ with highest retrieval probabilities. To jointly train retrieval and reconstruction, the retrieval scores $p(\mathbf{z}_i|\mathbf{x})$ are integrated into the computation of the likelihood. While the model learns to generate better texts, the retriever simultaneously learns to retrieve those documents that are helpful for the reconstruction. Basically, the approach models a denoising autoencoder, since the reconstruction of an input $\mathbf{x}$ is indirectly conditioned on $\mathbf{x}$, but with an intermediate bottleneck formed by the retrieved documents and relevance scores.

As shown in Figure 4.1, the model consists of two key components: the *passage retriever* with parameters η, which models $p_\eta(\mathbf{z}|\mathbf{x})$, and the *generator* parametrized by θ, which reconstructs the original text $\mathbf{x}$ given a set of support documents $(\mathbf{z}_1, \ldots, \mathbf{z}_k)$ together with their relevance scores $p_\eta(\mathbf{z}_i|\mathbf{x})$ from the retriever. These components are implemented as two separate neural networks, a pre-trained BERT$_{base}$ model [Devlin et al., 2019] serves as retrieval component while a modified BART$_{base}$ model [Lewis et al., 2019] is used as generator. While retrieval-augmented language models like MARGE [Lewis et al., 2020a] or REALM [Guu et al., 2020] were trained from scratch using massive computational resources (e.g. ≈ 4700 GPU days for MARGE), this work uses a setting where both components are pre-trained using language modeling objectives, which hopefully allows to leverage knowledge already present in the parametric memory of the neural networks. Section 6.3.1 presents the experiments that lead to this architectural decision.

Training the model end-to-end poses a chicken-and-egg problem: the reconstruction model cannot be effectively updated if none of the support documents contains relevant information while the retriever needs the signal from the generator to retrieve better documents. This means that starting the training from a randomly initialized model can easily lead into a vicious circle, where mostly uninformative passages are drawn randomly from the huge knowledge base and the model is not able to learn. To avoid this cold-start problem, the retriever is pre-trained on synthetic pairs of queries and related passages, where BM25 was used to find the highest ranked passage for each query in the knowledge base $\mathcal{Z}$ to construct the training samples as illustrated in Figure 4.2. Further details on pre-training can be found in Section 4.2.3.

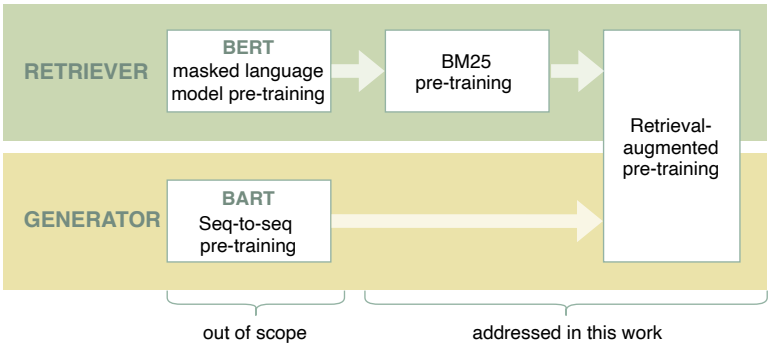


Figure 4.2.: Schematic representation of the training steps. BART and BERT pre-training is not considered in this work, the implementation is based on open source pre-trained models.

4.2. Dense passage retrieval

In this section, the *retrieval component* of the system is addressed. After an overview of the structure of the query and document encodings in Section 4.2.1, the following Section 4.2.2 deals with indexing and retrieval during training and inference. Finally, Section 4.2.3 describes the pre-training of the retriever.

The goal of the retriever is to index all passages of a given collection of text passages $\mathcal{Z}$ in a low-dimensional and continuous space, such that the top k passages relevant to an input query $\mathbf{x}$ can be efficiently retrieved. Since it is desirable to draw support documents from an informative knowledge base, it is important to consider that $\mathcal{Z}$ can become very large (e.g. 8 million passages in our experiments) while k is typically small, such as 5 - 100 passages. To retrieve from the huge knowledge base with reasonable effort, a dual-encoder architecture as described in Section 2.5.2 is employed allowing to precompute the representations in embedding space. Furthermore, the top k passages are determined with an approximate nearest neighbor (ANN) search as a trade-off between desired accuracy and resource requirements.

4.2.1. Query and Document Encoding

The retriever uses two dense encoders $\text{Enc}_Q(\cdot)$ and $\text{Enc}_P(\cdot)$ to embed queries and passages to d-dimensional real-valued vectors. The similarity between a query $\mathbf{x}$ and a passage $\mathbf{z}$ is defined as the dot product between their embedding vectors:

$$\text{sim}(\mathbf{x}, \mathbf{z}) = \text{Enc}_Q(\mathbf{x}) \cdot \text{Enc}_P(\mathbf{z})^\top \tag{4.1}$$

The retrieval distribution is the softmax over all relevance scores.

$$p(\mathbf{z}|\mathbf{x}) = \frac{e^{\text{sim}(\mathbf{x},\mathbf{z})}}{\sum_{\mathbf{z}'} e^{\text{sim}(\mathbf{x},\mathbf{z}')}}$$

It should be noted that there are more expressive models to estimate relevance between a query $\mathbf{x}$ and a passage $\mathbf{z}$ in neural information retrieval, using *interaction-focused* approaches which model word- and phrase-level relationships across $\mathbf{x}$ and $\mathbf{z}$ instead of computing a single score between their representations. Examples are [Mitra et al., 2017] using CNNs/MLPs, [Xiong et al., 2017] using kernels or [Nogueira and Cho, 2019] using the

transformer architecture and performing cross-attention on $\mathbf{x}$ and $\mathbf{z}$. While interaction-focused models tend to achieve higher accuracy in retrieval tasks [Guo et al., 2020], they are computationally expensive. In this work, a dual encoder architecture with a decomposable similarity function was chosen since it allows to precompute the passage embeddings and index the embeddings in advance, which significantly reduces the computational effort per query. This is key for an efficient training process.

Similar to the findings in [Karpukhin et al., 2020], our experiments showed that other decomposable similarity functions such as Euclidian L2 or cosine distance perform comparable or worse to the simpler dot product. Therefore, the dot product was chosen as similarity function.

The encoders are implemented using a pre-trained BERT-style Transformer (base, uncased) [Devlin et al., 2019]. Input texts are prepended with a special [CLS] token for classification, tokenized into wordpieces using the tokenization of the pretrained BERT_{base} model and passed through a BERT_{base} network. The output vector corresponding to the [CLS] token is used as a pooled representation of the input sequence. Finally, a linear layer with no activations projects the 768-dimensional BERT output to the desired embedding size d. While both the query encoder $\text{Enc}_Q(\cdot)$ and the passage encoder $\text{Enc}_P(\cdot)$ share the same BERT network, different linear projection layers are used for query and passage embeddings:

$$\text{Enc}_Q(\mathbf{x}) = \boldsymbol{W_Q}\text{BERT}_{\texttt{CLS}}(\mathbf{x}) \in \mathbb{R}^{b \times d} \tag{4.2}$$

$$\text{Enc}_P(\mathbf{z}) = \boldsymbol{W_P}\text{BERT}_{\texttt{CLS}}(\mathbf{z}) \in \mathbb{R}^{b \times d}, \tag{4.3}$$

where $\mathbf{x}$ and $\mathbf{z}$ denote minibatches of b queries and passages, respectively. Using the same network for query and document embeddings is useful since the network generates meaningful embeddings for queries and passages with high lexical overlap from the start, as shown in [Lewis et al., 2020a, Wieting and Kiela, 2019].

4.2.2. Offline indexing / Maximum inner product search

In the proposed architecture, a large-scale neural retrieval step is included into each training-step. This is obviously computational challenging since the retriever may have to consider

millions of candidate documents and the system needs to backpropagate through its decisions. To address this issue, the embeddings for each document are computed in advance, cached and updated asynchronously and infrequently.

Since the similarity score $\text{sim}(\mathbf{x}, \mathbf{z})$ is defined as an inner product, a Maximum Inner Product Search (MIPS) algorithm can be employed to efficiently find the top k documents. Inner product search has been widely studied and approximate MIPS algorithms [Ram and Gray, 2012, Shrivastava and Li, 2014, Shen et al., 2015], which scale sub-linearly in running time and storage space with the number of documents, are commonly used. In this work, FAISS [Johnson et al., 2019] is used to index the precomputed embeddings. FAISS[1] is an open source library for efficient similarity search and clustering of dense vectors.

However, the precomputed data structure will no longer be consistent with $p_\eta(\mathbf{z}|\mathbf{x})$ if the parameters η of the passage encoder $\text{Enc}_P(\cdot)$ are updated during training. Hence, the search index becomes "stale" after every gradient update. To resolve this with reasonable effort, the index is refreshed periodically by re-embedding and re-indexing all documents every few thousand training steps. In between, the stale index is only used to select the top k documents, which appears not to be crucial for training according to [Guu et al., 2020]. The retrieval probability $p_\eta(\mathbf{z}|\mathbf{x})$ and its gradient is recomputed using the current parameters η for these top k documents after retrieving them. Thereby, $p_\eta(\mathbf{z}|\mathbf{x})$ is approximated as softmax over the k documents. Guu et al. use a similar approach in [Guu et al., 2020] and show empirically that the procedure results in stable optimizations as long as the index gets refreshed at a sufficiently frequent rate.

4.2.3. Distantly supervised retriever pre-training

As stated earlier, the *retrieve-and-generate* training suffers from a cold-start problem if started with a freshly initialized retrieval component. The untrained encoder is not yet able to produce meaningful embeddings $\text{Enc}_Q(\mathbf{x})$ and $\text{Enc}_P(\mathbf{z})$ and the retrieved passages $\mathbf{z}$ are likely unrelated to $\mathbf{x}$. To avoid this issue, a pre-training step is introduced where the retriever learns embeddings that enable end-to-end training. The training data, in order to allow unsupervised training, is generated using heuristics. More precisely, BM25 (using the implementation from Apache Lucene[2]) is used to find the highest ranked passage for each query in the knowledge base $\mathcal{Z}$ as positive passage for the query.

[1] `https://github.com/facebookresearch/faiss`
[2] `https://lucene.apache.org/`

Training the encoders $\text{Enc}_Q(\cdot)$ and $\text{Enc}_P(\cdot)$ so that the dot-product similarity from Equation 4.1 becomes a good ranking function for retrieval is basically a *metric learning* problem [Kulis et al., 2012]. The encoder is supposed to map textual sequences to a vector space such that relevant pairs of queries and passages have a smaller distance (i.e., higher similarity) than the irrelevant ones in embedding space. To train the encoder to distinguish between relevant and irrelevant passages with respect to a given query, a training scheme similar to [Karpukhin et al., 2020] is used: The embeddings are optimized for maximizing the inner product between the query and the relevant passage vector, with an objective comparing all pairs of questions and passages in a batch.

Let $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{z}_1^+, \mathbf{z}_{1,1}^-, \ldots, \mathbf{z}_{1,n}^-), \ldots (\mathbf{x}_b, \mathbf{z}_b^+, \mathbf{z}_{b,1}^-, \ldots, \mathbf{z}_{b,n}^-)\}$ be the training data provided as a minibatch containing b instances. Each instance contains one query $\mathbf{x}_i$ and one relevant (positive) passage $\mathbf{z}_i^+$, along with n irrelevant (negative) passages $\mathbf{z}_{i,j}^-$. The loss function is optimized as the negative log likelihood of the positive passage. For each instance i the negative log likelihood is computed as

$$\mathcal{L}_i(\mathbf{x}_i, \mathbf{z}_i^+, \mathbf{z}_{i,1}^-, \ldots, \mathbf{z}_{i,n}^-) = -\log \frac{e^{\text{sim}(\mathbf{x}_i, \mathbf{z}_i^+)}}{e^{\text{sim}(\mathbf{x}_i, \mathbf{z}_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(\mathbf{x}_i, \mathbf{z}_{i,j}^-)}} \tag{4.4}$$

and the loss values are averaged over the b instances.

In-batch negatives In fact, the minibatches are structured in a somewhat different way for efficiency reasons. A batch contains b queries and each one is associated with a relevant passage. After encoding, this results in two $b \times d$-dimensional matrices $\mathbf{Q}$ and $\mathbf{P}$ containing b d-dimensional question and passage embeddings. The dot product of these matrices $\mathbf{S} = \mathbf{Q} \cdot \mathbf{P}^T$ is a $(b \times b)$ matrix of similarity scores, where each row corresponds to a query paired with b passages. Thus, n^2 query - passage pairs $(\mathbf{x}_i, \mathbf{z}_j)$ are obtained for each batch, where a pair $(\mathbf{x}_i, \mathbf{z}_j)$ is a positive example if $i = j$, and a negative one otherwise. This effectively creates b training instances per batch with one positive and $b - 1$ negative passages for each question.

The strategy of using in-batch negatives has been used in [Gillick et al., 2019] or [Karpukhin et al., 2020]. The fact that the number of training samples is effectively increased by reusing memory and computations makes it an efficient strategy for learning a dual-encoder model.

4.3. Retrieval-augmented Generation

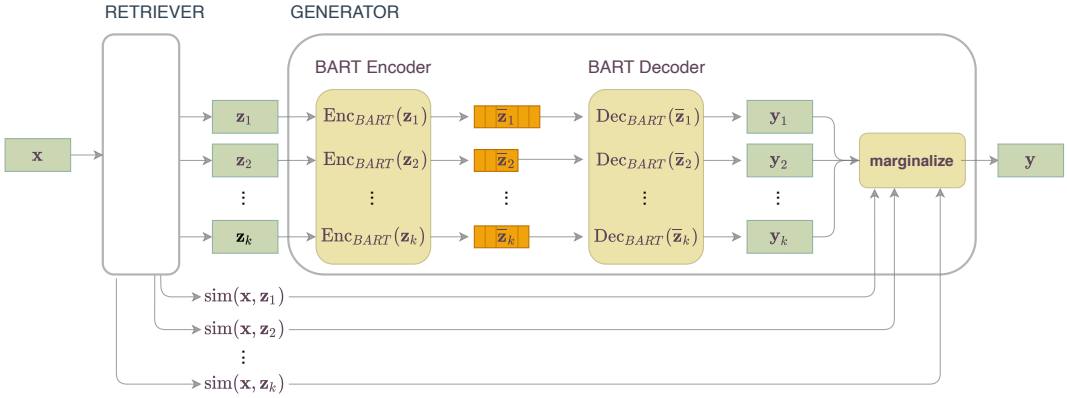
To develop a fully differentiable system which jointly learns retrieval and generation, the neural retriever described above is combined with a sequence-to-sequence transformer in a probabilistic model. The retriever provides relevant passages for a given input and the generator conditions solely on the retrieved passages to reconstruct the original text sequence. Generation is performed in a autoregressive way: a current token x_i is generated based on the previous $i-1$ tokens $\mathbf{x}_{1:i-1}$ and the retrieved passages $(\mathbf{z}_1, \ldots, \mathbf{z}_n)$. Please refer to Section 2.3.2 for further details on the generation process. The following sections present two different approaches to construct the *retrieve-and-generate* model: the first system marginalizes over retrieved passages in a top-k approximation to compute the likelihood to reconstruct a document, similar to [Guu et al., 2020, Lewis et al., 2020b]. From now on, this approach will be called *RUMBArt (Retrieval aUgmented Marginalized BArt)* while the second one - called *ACROBArT (retrieval Augmented CROss-attention BArT)* - uses an attention-based mechanism like [Lewis et al., 2020a] which uses the retrieval scores to bias attention to more relevant documents during generation. Figure 4.3 illustrates the different architectures.

4.3.1. RUMBArt: marginalization over support documents

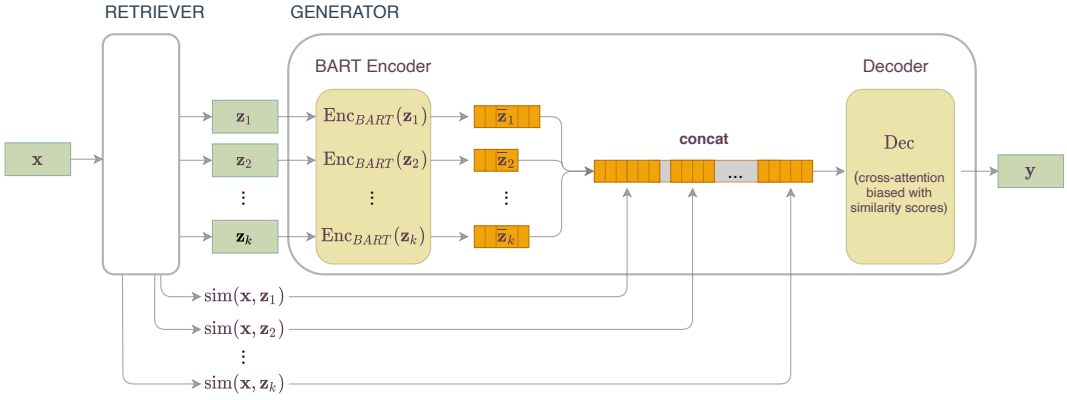
RUMBArt allows to train the retrieve-and-generate model end-to-end by treating the retrieved passage $\mathbf{z}$ as a latent variable. The overall likelihood of reconstructing an input $\mathbf{x}$ from retrieved support documents $\mathbf{z}_1 \ldots \mathbf{z}_k$ is obtained by marginalizing over all possible documents in $\mathcal{Z}$ according to Equation 4.5. In the following, the generated output sequence is denoted by $\mathbf{y}$ for clarity, where our training goal is to approximate $\mathbf{x}$ by $\mathbf{y}$ as closely as possible.

$$p(\mathbf{y}|\mathbf{x}) = \sum_{\mathbf{z} \in \mathcal{Z}} p_\eta(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{y}|\mathbf{z}) \tag{4.5}$$

Since a summation over all documents in the knowledge corpus $\mathcal{Z}$ is not feasible, the marginal probability $p(\mathbf{y}|\mathbf{x})$ is approximated by instead summing over the k documents with largest probability $p_\eta(\mathbf{z}|\mathbf{x})$ as shown in Equation 4.6. Assuming that only few documents



(a) RUMBArt. The retriever forwards a set of support documents $\mathbf{z}_1, \ldots \mathbf{z}_k$ and the corresponding retrieval scores to the generator. An output $\mathbf{y}_i$ is generated for each $\mathbf{z}_i$ and the model output is calculated as a marginalized sum weighting the outputs $\mathbf{y}_i$ by their retrieval scores $\text{sim}(\mathbf{x}, \mathbf{z}_i)$.



(b) ACROBArT. The generator is an encoder-decoder model, which processes the support documents $\mathbf{z}_1, \ldots \mathbf{z}_k$ individually in the encoder. The encoder outputs $\overline{\mathbf{z}}_i$ are concatenated and processed in the decoder, where the retrieval scores are included into the attention mechanism.

Figure 4.3.: Generator architectures for both model variants.

in $\mathcal{Z}$ are relevant for a given input, while most documents have near zero probability, this approximation is reasonable.

$$p(\mathbf{y}|\mathbf{x}) \approx \sum_{i=1}^{k} p_\eta(\mathbf{z}_i|\mathbf{x}) p_\theta(\mathbf{y}|\mathbf{z}_i) \tag{4.6}$$

More precisely, the retriever determines the top k passages with maximum retrieval scores for the query and passes each of these passages as input to the generator. Here, for each of the passages, the probability for generating the target token sequence $\mathbf{y}$ conditioned on

the respective passage $\mathbf{z}_i$ is computed as the product of the probabilities for all individual tokens:

$$p_\theta(\mathbf{y}|\mathbf{z}_i) = \prod_{j=1}^{n} p_\theta(\mathbf{y}_j|\mathbf{z}_i, \mathbf{y}_{1:j-1}) \tag{4.7}$$

where n is the number of tokens in $\mathbf{y}$. Finally, the overall output probability is approximated as a weighted sum over the k individual output probabilities:

$$\begin{aligned} p(\mathbf{y}|\mathbf{x}) &\approx p_\theta(\mathbf{y}|\mathbf{z}_1, \ldots, \mathbf{z}_k, p_\eta(\mathbf{z}_1|\mathbf{x}), \ldots, p_\eta(\mathbf{z}_k|\mathbf{x})) \\ &= \sum_{i=1}^{k} p_\eta(\mathbf{z}_i|\mathbf{x}) \prod_{j=1}^{n} p_\theta(\mathbf{y}_j|\mathbf{z}_i, \mathbf{y}_{1:j-1}) \end{aligned} \tag{4.8}$$

Algorithm 1 describes the forward propagation for a batch of b query strings for RUMBArt in detail. Here, l_Q denotes the maximum token length of the tokenized query strings, l_P is the length of the tokenized passages. All token sequences are assumed to be padded to a maximum length for simplicity.

Training The generator component is initialized as BART_{base} while a BERT_{base} model serves as retriever, pre-trained as described in Section 4.2.3. After this initialization step, the retriever and generator are trained jointly without any further supervision on what passages should be retrieved. Therefore, minibatches of randomly selected queries $\mathcal{D} = \{\mathbf{x}_1, \ldots \mathbf{x}_b\}$ serve as training data. The retriever draws the top-k passages for each query $\mathbf{x}_i$ from the current MIPS index and re-embeds these passages using the current model weights to calculate the actual similarity scores. For each of the passages, the generator computes the probability of generating the original input and marginalizes these probabilities as described above. The negative logarithm of the marginalized likelihood $p(\mathbf{y}|\mathbf{x})$ defined in Equation 4.8 then serves as the loss function that is minimized using stochastic gradient descent with Adam [Kingma and Ba, 2017]:

$$Loss = -\log p_(\mathbf{y}|\mathbf{x}_1) \tag{4.9}$$

Algorithm 1 Forward pass in RUMBArt end-to-end training

procedure $\mathrm{FORWARD}(query)$
/* RETRIEVE */
/* (1) get top-k passages from current MIPS index */
$\mathbf{x} \in \mathbb{N}^{b \times l_Q} \leftarrow \text{tokenize}_{BERT}(query)$
$\mathbf{x\_enc} \in \mathbb{R}^{b \times d} \leftarrow \text{Enc}_Q(\mathbf{x})$
$passages \leftarrow$ top-k entries for $\text{Enc}_Q(\mathbf{x})$ in MIPS index

/* (2) Recompute embeddings and similarity with current Encoder */
$\mathbf{z} \in \mathbb{R}^{b \times k \times l_P} \leftarrow \text{tokenize}_{BERT}(passages)$
$\mathbf{z\_enc} \in \mathbb{R}^{b \times k \times d} \leftarrow Enc_P(\mathbf{z})$
$\mathbf{sim}(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{b \times k} \leftarrow \mathbf{z\_enc} \cdot \mathbf{x\_enc}$
$\mathbf{p}_\eta(\mathbf{z}|\mathbf{x}) \leftarrow \text{softmax}(\mathbf{sim}(\mathbf{x}, \mathbf{z}), dim = 1)$

/* GENERATE */
/* (3) Compute probability to generate the original query for each input */
$\mathbf{z} \in \mathbb{R}^{b \times k \times l_P} \leftarrow \text{tokenize}_{BART}(passages)$
$\mathbf{y} \in \mathbb{R}^{b \times l_Q} \leftarrow \text{tokenize}_{BART}(query)$
$\mathbf{p}_\theta(\mathbf{y}|\mathbf{z}) \in \mathbb{R}^{b \times k \times l_Q} \leftarrow \text{BART}(\mathbf{z}, target = \mathbf{y})$

/* (4) Marginalize probabilities over passages according to Equation 4.8 */
$\mathbf{p}(\mathbf{y}|\mathbf{x}) \in \mathbb{R}^b \leftarrow \sum_{i=1}^{k} \mathbf{p}_\eta(\mathbf{z_i}|\mathbf{x}) \mathbf{p}_\theta(\mathbf{y}|\mathbf{z_i})$
return $\mathbf{p}(\mathbf{y}|\mathbf{x})$
end procedure

The training algorithm is explained in Algorithm 2. Since both, the retriever and the generator are differentiable neural networks, the gradient of $-\log p(\mathbf{y}|\mathbf{x})$ can be computed with respect to the model parameters η and θ, and optimized using stochastic gradient descent. During training, the MIPS index is updated at regular intervals.

4.3.2. ACROBArT: Generation using retrieval biased cross-attention

In contrast to RUMBArt, where each passage $\mathbf{z}_i$ is fed to the generator separately to compute $p_\theta(\mathbf{y}|\mathbf{z}_i)$ before marginalization, ACROBArT generates a single generator output for the top k retrieved passages as shown in Figure 4.3b. The retriever is constructed exactly as described in Section 4.2. Given a query $\mathbf{x}$, the retriever calculates $\text{Enc}_Q(\mathbf{x})$ and retrieves a set of top k support passages $(\mathbf{z}_1 \dots \mathbf{z}_k)$ together with their retrieval scores $p_\eta(\mathbf{z}_i|\mathbf{x})$ from the pre-computed MIPS index. The reconstruction model then computes the likelihood of target document $\mathbf{y}$ as:

Algorithm 2 End-to-end training procedure

procedure TRAIN(queries $\mathcal{X}$, passages $\mathcal{Z}$)
/* Initialize */
  $model.retriever \leftarrow$ BERT$_{base}$, pre-trained as described in Section 4.2.3
  $model.generator \leftarrow$ BART$_{base}$
  for all $\mathbf{z} \in \mathcal{Z}$ **do**
    Add Enc$_P(\mathbf{z})$ to MIPS index
  end for
  $epoch \leftarrow 0$
  /* Train */
  while $epoch <$ max number of epochs **do**
    $\mathcal{X} =$ shuffle$(\mathcal{X})$
    for all batches $\{x_i\}_{i=1}^{b} \in \mathcal{X}$ **do**
      $\mathbf{p}(\mathbf{y}|\mathbf{x}) \leftarrow model.forward(batch)$
      $loss \leftarrow \sum_{i=1}^{b} -\log \mathbf{p}(\mathbf{y_i}|\mathbf{x_i})$
      compute gradients in backwards step
      update model parameters η and θ
      /* Every refresh_period steps update MIPS index */
      if number of steps % refresh_period $= 0$ **then**
        discard MIPS index
        for all $\mathbf{z} \in \mathcal{Z}$ **do**
          Add Enc$_P(\mathbf{z})$ to MIPS index
        end for
      end if
    end for
    $epoch \leftarrow epoch + 1$
  end while
end procedure

$$\mathcal{L}_\theta = -\log p_\theta(\mathbf{y}|\mathbf{z}_1, \ldots, \mathbf{z}_k, p_\eta(\mathbf{z}_1|\mathbf{x}), \ldots, p_\eta(\mathbf{z}_k|\mathbf{x}))$$

just like above. By selecting the top results, i.e. those considered most relevant to a query $\mathbf{x}$ by the retriever, the model obtains the best available information to reconstruct $\mathbf{x}$.

Like in RUMBArt, the generator model is a bidirectional seq-to-seq Transformer (BART$_{base}$), composed of an encoder Enc$_{BART}$ and a decoder Dec$_{BART}$. Please refer to Section 2.3.2 for details on the encoder-decoder architecture. In a first step, the encoder encodes the k input passages $(\mathbf{z}_1 \ldots \mathbf{z}_k)$ individually as $\overline{\mathbf{z}}_i =$ Enc$_{BART}(\mathbf{z}_i)$ and concatenates the resulting embeddings $(\overline{\mathbf{z}}_1 \ldots \overline{\mathbf{z}}_k)$ to a global representation $\overline{\mathbf{Z}}$ of dimension $(\sum_j |\mathbf{z}_j|) \times d$ where $|\mathbf{z}_j|$ is the token length of the jth passage and d is the dimension of the hidden representations

of the model:

$$\overline{\mathbf{Z}} \in \mathbb{R}^{(\sum_j |z_j|) \times d} = \overline{\mathbf{z}}_1 + \!\!+ \, \overline{\mathbf{z}}_2 + \!\!+ \ldots + \!\!+ \, \overline{\mathbf{z}}_k, \tag{4.10}$$

where $+\!\!+$ denotes concatenation of the output vectors along the first dimension.

The decoder takes this representation as input and continues as a regular autoregressive model, alternating self-attention, cross-attention and a feed-forward module for each layer (see Section 2.3.1 for details). Recall, that the encoder output $\overline{\mathbf{Z}}$ is only processed during cross-attention. If $\mathbf{Y} \in \mathbb{R}^{m \times d}$ denotes the output of the previous self-attention layer in the decoder, the cross-attention operation consists of the following operations: First, queries $\mathbf{Q}$, keys $\mathbf{K}$ and values $\mathbf{V}$ are computed by applying linear transformations following common practice [Vaswani et al., 2017]:

$$\begin{aligned} \mathbf{Q} &= \mathbf{W}^Q \cdot \mathbf{Y}, \\ \mathbf{K} &= \mathbf{W}^K \cdot \overline{\mathbf{Z}}, \\ \mathbf{V} &= \mathbf{W}^V \cdot \overline{\mathbf{Z}}. \end{aligned}$$

The model performs multi-head attention, so the computation is performed in parallel using different projection matrices. The indices for the different heads are omitted here and in the following for simplicity.

Cross-Attention (Basic). Then a matrix of cross-attention probabilities between the target text $\mathbf{y}$ and the concatenated support passages $\overline{\mathbf{Z}}$ is obtained by computing the dot-product between query $\mathbf{Q}$ and key $\mathbf{K}$ and normalizing over the elements of $\overline{\mathbf{Z}}$:

$$\alpha = \text{softmax}(\mathbf{Q} \cdot \mathbf{K}^\top) \in \mathbb{R}^{m \times \sum_j |\mathbf{z}_j|}$$

where the softmax is applied row-wise to the scores.

Cross-Attention: Retrieval-Enhanced. To jointly train both model components, the similarity scores need to be included into the generation process in such a way that it is possible to calculate the gradient of the overall loss $\mathcal{L}_\theta$ with respect to the retrieval parameters η and the generator parameters θ and optimize them by means of gradient descent. This is achieved by using the retrieval scores as additive bias for the cross-attention operation, so that the decoder will pay more attention to passages with a larger retrieval score. The actual biased cross-attention values are computed as:

$$\alpha = \text{softmax}(\mathbf{Q} \cdot \mathbf{K}^\top + \beta \cdot p_\eta(\mathbf{z}_j|\mathbf{x})) \in \mathbb{R}^{|\mathbf{x}| \times \sum_j |\mathbf{z}_j|}, \tag{4.11}$$

where β is a trainable parameter that weights the importance of the similarity scores for the reconstruction process. Since the generator is a multi-head attention model, the operations described here are performed in parallel with different linear transformations in each attention head. The complete forward pass for a batch of b queries is described in Algorithm 3. The maximum token length of the tokenized query strings is denoted as l_Q, the length of the tokenized passages is l_P. For simplicity, all token sequences are assumed to be padded to a maximum length and the attention mechanism is presented for a single head.

Given that the use of more relevant input sequences improves the likelihood of reconstructing the input query $\mathbf{x}$, minimizing the reconstruction loss should simultaneously improve the quality of the retriever. Unlike RUMBArt, ACROBArT allows the generator to attend to tokens from different input passages during generation. Section 6.3.2 presents a comparison of both models with a focus on the self-supervised training capabilities of their retrieval components.

Algorithm 3 Forward pass in ACROBArT end-to-end training

procedure $\mathrm{FORWARD}(queries)$
/* RETRIEVE */
/* (1) get top-k passages from current MIPS index */
$\mathbf{x} \in \mathbb{R}^{b \times l_Q} \leftarrow \text{tokenize}_{BERT}(queries)$
$\mathbf{x\_enc} \in \mathbb{R}^{b \times d} \leftarrow \text{Enc}_Q(\mathbf{x})$
$passages \leftarrow$ top-k entries for $\mathbf{x\_enc}$ in MIPS index

/* (2) Recompute embeddings and similarity with current encoder */
$\mathbf{z} \in \mathbb{R}^{b \times k \times l_P} \leftarrow \text{tokenize}_{BERT}(passages)$
$\mathbf{z\_enc} \in \mathbb{R}^{b \times k \times d} \leftarrow Enc_P(\mathbf{z})$
$\mathbf{sim}(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{b \times k} \leftarrow \mathbf{z\_enc} \cdot \mathbf{x\_enc}$
$\mathbf{p}_\eta(\mathbf{z}|\mathbf{x}) \leftarrow \text{softmax}(\mathbf{sim}(\mathbf{x}, \mathbf{z}), dim = 1)$

/* GENERATE */
/* (3) Compute BART encodings for each passage */
$\mathbf{z} \in \mathbb{R}^{b \times k \times l_P} \leftarrow \text{tokenize}_{BART}(passages)$
$\mathbf{y} \in \mathbb{R}^{b \times l_Q} \leftarrow \text{tokenize}_{BART}(queries)$
$\overline{\mathbf{z}} \in \mathbb{R}^{b \times k \times l_P \times d} \leftarrow \text{Enc}_{BART}(\mathbf{z})$
/* (4) Perform cross-attention according to equation 4.11 */
$\mathbf{p}_\theta(\mathbf{y}|\mathbf{z}) \in \mathbb{R}^{b \times l_Q} \leftarrow \text{Dec}(\overline{\mathbf{z}}, \mathbf{p}_\eta(\mathbf{z}|\mathbf{x}), target = \mathbf{y})$
return $\mathbf{p}_\theta(\mathbf{y}|\mathbf{x})$
end procedure

5. Experimental Setup

This chapter describes the data used for the experiments and the basic setup. The goal of this work is unsupervised training of passage retrieval on unstructured text. However, in order to evaluate the quality of the developed models, the experiments are performed on a dataset for which labelled ground truth is available. Therefore, all models are trained and evaluated on MS MARCO [Bajaj et al., 2018], a publicly available collection of datasets focused on deep learning in search.

5.1. The MS Marco Dataset

The *MAchine Reading COmprehension dataset* was originally introduced by Microsoft in 2016 for reading comprehension. A revised version adapted for retrieval was published in 2018 [Bajaj et al., 2018]. The dataset includes about 8.8M passages extracted from 3.5M web documents, which were gathered from Microsoft's *Bing* results to 1M real-world search queries. Each query is associated to a set of about 10 candidate passages. The queries and candidate passages were presented to human editors who formulated answers to the queries and marked those passages as 'positive', that contained useful and necessary information to answer the question. If the information required to answer the query was not present in the candidate passages, the query does not contain an answer and accordingly no passages were marked as positive. A quantitative analysis shows that of the 808k queries in the training set, no answer could be given for 278k queries. Of the remaining 530k answerable queries, one passage was marked as positive for 505k, while two or more positive passages were annotated for 25k queries. Figure 5.1 provides a quantitative overview of the data. Note that the human formulated answers from the MS Marco dataset are not used in this work. Evaluation and several supervised pre-training experiments are performed on the queries and the candidate passages labeled as positive.

It is important to note that passages have not been marked explicitly as irrelevant. Hence, any unlabeled passage is assumed to be negative, though the dataset might contain unla-

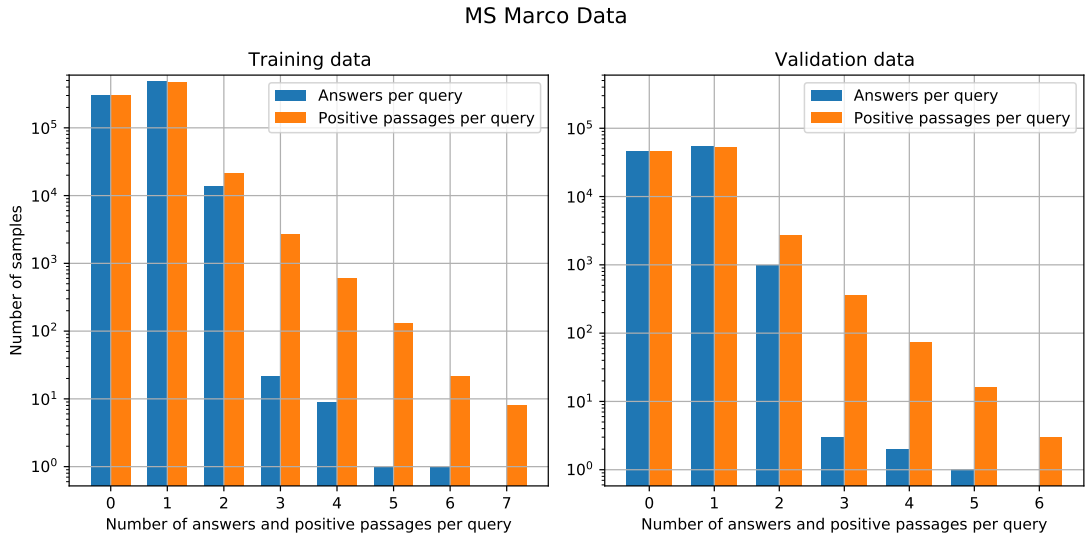


Figure 5.1.: Queries and related answers and support passages in MS Marco (logarithmic scale).

beled passages that are actually more relevant for a query than the labeled one. Table 6.2 will show some examples of such 'false negative' passages.

The validation and evaluation sets each consist of 101k queries, with relevance judgements of the evaluation set being held back by Microsoft. Ranking results on the evaluation set can only be obtained by submission to the competition organizers. Following common practice, the official validation split is re-purposed as local evaluation set. The held-out evaluation set is used to test different training approaches as well as baselines in the following chapter. In a preprocessing step, all queries were filtered out that could not be answered and for which accordingly no positive passage was labelled.

Evaluation metrics Following the official MS Marco evaluation, the *mean reciprocal rank* cut at the 10th position (MRR@10) is used to measure effectiveness of the retriever, allowing to compare the results to current state-of-the-art in neural retrieval. The reciprocal rank of a retrieval result to a query is the multiplicative inverse of the rank of the first positive passage: 1 for first place, $\frac{1}{2}$ for second place and so on. The mean reciprocal rank is calculated according to Equation 5.1 as the average of the reciprocal ranks for a set of

queries Q:

$$MRR@k = \frac{1}{|Q|}\sum_{i=1}^{|Q|} RR_i@k \quad \text{with} \quad RR_i@k = \begin{cases} \frac{1}{\text{rank}_i} & \text{if rank}_i \leq k \\ 0 & \text{otherwise,} \end{cases} \tag{5.1}$$

where rank_i denotes the rank position of the first relevant passage for the ith query. Additionally, the top-k *recall* (Recall@k) is reported, which is the percentage of queries for which the set of top-k retrieved passages contains at least one passage labeled as relevant.

5.2. Implementation details

The following technical setup is used for the practical experiments.

Software. All models are implemented using Python 3.8 and PyTorch 1.7. The pre-trained BERT$_{base}$ and BART$_{base}$ models originate from the huggingface transformers library[1] [Wolf et al., 2020], a popular open source library for natural language processing. The training code is structured with pyTorch Lightning[2], an open source framework developed for AI research projects aiming to decouple research from engineering code. Logging and monitoring of the training runs is done with weights and biases[3]. Furthermore, the implementation uses the FAISS library[4] for passage indexing and maximum inner product search. For the BM25 based pre-training and BM25 baseline the elasticsearch implementation[5] from apache lucene is used.

Hardware and hyperparameters. A disadvantage of the two-stage retrieve-and-generate model is the memory consumption of the model. Training is performed on a RTX A6000 graphic card with 48GB memory to be able to train the BERT$_{base}$ - BART$_{base}$ network on GPU. Unless stated otherwise, a batch size of $b = 32$ is chosen for the retriever pre-training, while the end-to-end training uses batches of size $b = 16$. To achieve a larger effective batch size, gradients were accumulated over 10 batches. The learning rate is set as $lr = 1\text{e} - 04$ and linearly annealed to $lr = 1\text{e}{-}07$. The first 1000 steps are defined as

[1] https://huggingface.co/transformers/
[2] https://www.pytorchlightning.ai/
[3] https://wandb.ai/
[4] https://github.com/facebookresearch/faiss
[5] https://www.elastic.co/de/elasticsearch/

warm-up period where the learning rate linearly increases to the initial value. The MIPS index is rebuild every 5000 steps. To reduce the effort of re-embedding the knowledge base $\mathcal{Z}$ during training, most experiments are performed on a reduced subset of $100,000$ MS MARCO queries and related candidate passages.

6. Experiments

This chapter describes the experiments that were conducted to develop and empirically evaluate RUMBArt and ACROBArT. Following a bottom-up approach, the retrieval and generation component are investigated independently before evaluating the combined *retrieve-then-generate* model. In particular, the following questions are addressed:

- What are efficient pre-training schemes for the retrieval component?
- To what extent can pre-trained models reduce the training effort and what are suitable architectures?
- Even if the focus in training is on the retrieval component, what is the quality of the generator?
- What are advantages and disadvantages of the different approaches for retrieval-augmented generation from Section 4.3, especially with regard to retrieval?

6.1. Passage Retrieval Pre-Training

Two stages of retriever training are involved in our training procedure: a pre-training step to initialize the retrieval component such that the retriever draws reasonably meaningful passages from the beginning, and as a second step, the actual end-to-end training to further refine the retriever.

The first part of this chapter focuses on the pre-training step for the retrieval component. Given that the retriever will be further optimized in the second training step, the pre-training actually only needs to facilitate this *retrieve-and-generate* training. Nevertheless, we expect the final accuracy to improve when training is initialized with a better pre-trained retrieval model. In the following, different methods for improving retrieval accuracy during pre-training are discussed and empirically evaluated. Although the goal of this work is to develop a completely unsupervised retriever pre-training, the following experiments will

be conducted on supervised data in anticipation of findings, that could be transferred to unsupervised training.

6.1.1. Positive and negative passages

The retrieval component is pre-trained on training samples consisting of a query q_i, a relevant (positive) passage p_i^+ and several irrelevant (negative) passages $p_{i,j}^-$ by optimizing the likelihood of the positive passage as described in Section 4.2.3. MS Marco as the underlying data for the experiments provides labels for positive 'gold' passages, but does not explicitly identify negative passages, which is often the case for retrieval datasets. Without further knowledge about the data, all passages other than positive ones have to be considered irrelevant, i.e., negative passages are selected from an extremely large pool.

In practice, according to [Gillick et al., 2019, Karpukhin et al., 2020, Gao et al., 2021], the selection of negative passages during training has been found to be decisive for learning a high-quality dense encoder. Emphasizing challenging negative passages, which are difficult to distinguish from the positive passage, forces the model to exploit context rather than simply learning word overlap. In [Karpukhin et al., 2020], the authors combine negative passages randomly sampled from the corpus with one additional challenging negative passage per query, where the 'hard' negative is selected as a passage with large BM25 score regarding the query. Gao, Dai and Callan propose to sample non-relevant negative documents from the target retriever [Gao et al., 2021]. During training, the retriever is used in its current state to retrieve a set of top ranked passages from the entire corpus for a query. From this set, n non-relevant passages are sampled as 'hard' negative examples.

In the following, different types of such 'hard negatives' as well as strategies to incorporate them into training are evaluated.

Random randomly chosen passages from the corpus

BM25 passages with top BM25 score meaning high lexical overlap

MS Marco passages related to a query but not labeled as relevant by an expert

Retriever current nearest neighbors in the embedding space (except the labeled positives)

As opposed to the standard 1-of-N training setting, where each query in the batch is combined with one positive and its own set of n negative passages, the in-batch negative training setting described in Section 4.2.3 is examined. Here, a minibatch contains pairs

of query and related gold passage, with the gold passages of the remaining queries serving as negative passages for each query in the batch. Negative passages obtained as in-batch negatives are denoted as *gold* in the following. All experiments use an embedding dimension of $d = 256$ and a batchsize of $b = 32$. After each epoch, the training data is shuffled and new sets of negative passages are sampled. The model is trained by minimizing the negative log likelihood of the positive passage using an Adam optimizer [Kingma and Ba, 2017] with a constant learning rate of 10^{-4}. The tokenized passages are padded to a maximum length of $l = 256$ tokens and dropout is applied throughout the model with the default rate of 0.1.

Table 6.1 summarizes the results of different pre-training schemes on the MS Marco dataset. The top block shows a 1-of-N setting, where each query in the batch is combined with its own set of one positive and n negative passages. The irrelevant passages are chosen as described above, comparing random passages, high rated BM25 passages, 'negative' MS Marco passages (BING search results for the query not labeled as relevant) and passages with a high retrieval score in the current retriever state.

The second block examines in-batch negative training and combinations of in-batch negatives with additional 'hard' negative samples. For each query in a batch, an additional hard negative passage is added, with these additional passages also serving as negative passages for all other queries in the batch. For the batchsize of $b = 32$ used in the experiments, this results in 31 negative passages per query in the standard setting and $31 + 32$ negative passages when additional 'hard' negatives are used.

The experiments show that in-batch negative training significantly improves the results. Rather than generating new negative samples for every query, each passage already in the batch is reused for each query. Thus, more pairs are generated allowing to train larger training samples using the same resources, which might contribute to the good model performance. As a result, with in-batch negatives, accuracy improves as the batch size grows. Incorporating additional 'hard' negatives can further improve the accuracy although the effect is found to be weaker than expected. The best results are obtained with negative MS Marco passages.

The [Gao et al., 2021] approach of selecting unlabeled passages that are currently highly ranked by the retriever as negatives looks promising, as the retriever learns to decrease the rank of over-ranked results. In fact, this approach achieved the lowest accuracy in the practical experiments: training in the 1-to-N setting with retriever negatives prevented the retriever from learning, while additional retriever negatives in the in-batch negatives setting

Type	# neg.	in-batch	MRR@10	Recall@10	Recall@20
Random	5	-	30.4	53.5	65.5
BM25	5	-	57.1	76.1	81.2
MS Marco	5	-	13.6	30.7	37.3
Retriever	5	-	0.0	0.2	0.2
Gold	31	✓	69.9	87.2	91.0
Gold + MS Marco	31 + 32	✓	70.0	87.8	92.3
Gold + Retriever	31 + 32	✓	41.3	61.7	70.6
Gold + BM25	31 + 32	✓	68.7	87.2	91.9

Table 6.1.: Comparison of supervised retrieval pre-training schemes, measured as top-10 mean reciprocal rank and recall in percent on MS Marco (evaluation set).

resulted in significantly reduced accuracy. This may be due to the fact that the approach expects only one positive passage per query, which is not the case for the MS Marco data. Passages that are assumed to be negative may actually be relevant to a query. Table 6.2 lists selected examples from the MS MARCO dataset, which confirm this assumption. Thus, the network receives conflicting training signals and is unable to learn good representations. Whether the approach performs better when explicit negative passages can be identified needs further research and is beyond the scope of this work.

The results of the experiments suggest that, contrary to the statements of [Karpukhin et al., 2020] and [Gao et al., 2021], the use of dedicated hard negatives does not make a decisive difference in our setting. In-batch negative training, on the other hand, has proven to be very effective. Therefore, for the further pre-training experiments in-batch negative training without additional hard negatives will be used.

6.1.2. Effect of representation size

The encoded representation for query and context $\text{Enc}_Q(\cdot)$ and $\text{Enc}_P(\cdot)$ is obtained by projecting the output of the BERT network to an arbitrary representation size d as described in Section 4.2.1. Obviously, the choice of the embedding dimension d determines the size of the indexed encoded corpus and thus the memory footprint of the system. On the other hand, a larger representation size intuitively increases the amount of information that can be encoded in the embeddings. To verify this intuition, the relation between embedding size d and retrieval quality is investigated by performing identical training runs with different values for d. The retriever is trained in a supervised setting using the labeled support

Query	Labeled positive passage	Top-1 neural retrieval
has jeopardy been cancelled	Are drugs and divorce killing the Kardashians? Ratings for E! reality show are so low its in 'jeopardy of being cancelled'. How much longer will the world be interested in Keeping Up With the Kardashians? The show looks to be on the brink of being cancelled as less and less people tune in, per RadarOnline.	The original Jeopardy!series premiered on March 30, 1964, as a daytime program on NBC. With Art Fleming as host and Don Pardo as announcer, that series continued to air until January 3, 1975, and also spawned a weekly syndicated version that aired within the 1974-1975 season.
how much is the stamp to send a card	As of June 2014, the United States Postal Service states that the cost to send a standard sized postcard is 34 cents. At this rate, the postcard can be 6 inches long, 4 1/4 inches high and 0.16 inches thick.	How Much Does A Postage Stamp Cost For 2017. Effective January 22, 2017 the cost of a first class (1) ounce letter sent through the United States Post Office increased two cents from $0.47 to $0.49 for letters mailed within the United States. Reversing the $0.2 price drop from $0.49 to $0.47 effective April 10, 2016.
how much protein should i take per day for working out	And more. So, here's the daily protein recommendations you need to know: 1 If you do NOT workout and do NOT have any real diet or fitness related goal (like building muscle, losing fat, etc.), you should eat between 0.5-0.7 grams of protein per pound of body weight per day.	1 She'd simply multiply 130 by 1-1.2 and get a daily protein intake of between 130-156 grams per day. Now let's say a 180lb man wanted to build muscle, or maintain muscle while losing fat, or improve strength/performance. He'd do 180 x 1-1.5 and get a daily protein intake of between 180-270 grams per day.

Table 6.2.: Sample queries and passages from MS MARCO. The table lists examples where the best passage found by the neural model is not labeled as positive and has similar or higher relevance for the query than the labeled one. Sample passages are retrieved with RUMBArt.

passage from MS Marco as positive passages with in-batch negatives and no additional 'hard' negative passages. To reduce the effort of re-embedding the knowledge base $\mathcal{Z}$ during training, only a subset of $100,000$ MS MARCO passages is used as knowledge base. Figure 6.1 illustrates the effect of different embedding dimensions on the accuracy of the retrieval model.

For small embedding dimensions from $d = 64$ to $d = 256$ the retrieval accuracy increases almost linearly with embedding dimension, suggesting that the increased expressiveness of larger embedding vectors can be exploited directly to achieve higher retrieval accuracy. The improvement in quality for embedding dimension $d = 512$ compared to $d = 256$ is only marginal, while noticeably better accuracy is achieved with $d = 768$. Since the main objective of this work is to evaluate what retrieval accuracy can be achieved with an unsupervised procedure, an embedding dimension of 768 is chosen for the final models. However, depending on the requirements, the choice of representation size can be used as trade-off between the resources required and the accuracy achieved.

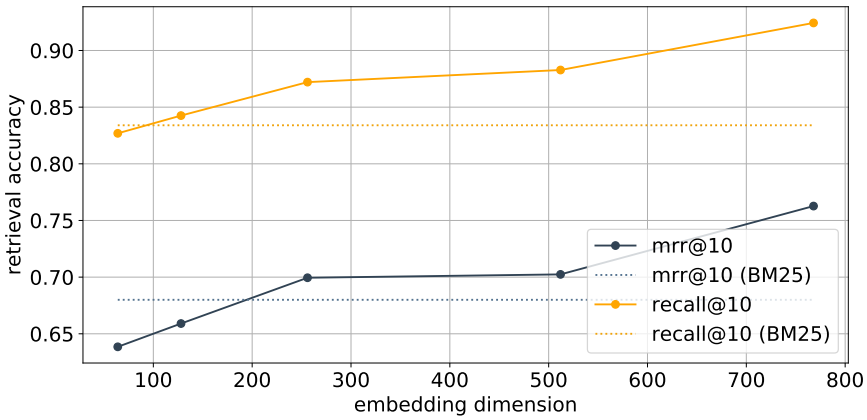


Figure 6.1.: Retrieval accuracy measured as MRR@10 and recall@10 for different embedding dimensions.

6.1.3. Unsupervised Pre-Training

In all experiments conducted so far, the labeled gold passages from MS Marco are used as positive passages during training. Since the aim of this work is to develop a retrieval training procedure for situations where labeled data is not available, we evaluate the effects of distantly-supervised training on retrieval accuracy by using BM25 to estimate the positive passage. For each query, Lucene-BM25 is used to select the best ranked passage out of the knowledge base $\mathcal{Z}$ which is then used as positive sample during training. Negative samples are provided as in-batch negatives and the embedding dimension is $d = 768$. Table 6.3 compares the performance of the retriever when pre-trained using the supervised and the distantly-supervised BM25 setting. The BM25 baseline is added for reference. The distantly-supervised pre-training achieved a surprisingly high accuracy: unsupervised pre-

training reduces the MRR@10 by only 7% compared to supervised pre-training.

Pre-Training	**positive samples**	MRR@10	Recall@5	Recall@10	Recall@50
None (BM25 baseline)		68.0	78.9	83.4	90.7
Supervised	MS Marco	76.3	88.8	92.4	97.7
Unsupervised	BM 25	69.1	83.2	88.1	96.5

Table 6.3.: Comparison of supervised and unsupervised retrieval pre-training. The table shows MRR and Recall in % for retrieval from 100.000 candidate passages.

6.2. Text Generation

This section evaluates the generation component of the system. For both model architectures, the generated reconstructions for exemplary input queries are visually inspected and compared to the queries.

The output texts are generated auto-regressively as described in Section 2.3.4. Since the output probability distribution is modeled as product of conditional probabilities, generation must be successive, with each step adding the newly generated token to the model input for the next step. For each step, the generator outputs a logit vector indicating, for each token in the vocabulary, its probability of being the next token $p_\theta(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \overline{\mathbf{X}})$, depending on the input and the sequence already generated. To sample token sequences with high overall probability from the large space of possible sequences, greedy decoding, beam search with 4 beams and nucleus sampling with $p = 0.92$ are used as decoding methods. Table 6.4 shows examples for RUMBArt. The table shows the final reconstruction, generated by greedy decoding the marginalized output logits and additionally displays the top-3 retrieved passages each with queries generated from the single passage. Queries are decoded using greedy search for comparison to the marginalized reconstructions and with beam search and nucleus sampling to assess whether the decoding method influences the quality of the generated queries.

The quality of the marginalized queries was found to be worse than the queries generated from a single passage. Obviously, the marginalized output is not directly interpretable as logit output which can be decoded into token ids. The same observation can be found in [Lewis et al., 2020b], where the authors propose the following decoding method: each output $\mathbf{y}_i$ generated from a single passage $\mathbf{z}_i$ is decoded into a candidate token sequence using a conventional decoding method. Each of these sequences is then used as target

for marginalized generation and the model calculates the negative log likelihood for the sequence. The most likely candidate is chosen as output.

Note that the examples in Table 6.4 were generated with an earlier version of RUMBArt. The currently best model for retrieval generates queries of poorer quality which comes as no surprise since the training was designed and optimized for the retrieval objective.

Table 6.5 compares reconstructions of the two model variants. For RUMBArt, in addition to the marginalized output, the reconstruction from the top passage is given. Since ACROBArT is able to attend to important tokens from all input passages, it was expected to generate reconstructions of higher quality. This intuition could not be demonstrated in our experiments. Again, the training objective was to improve the retriever, which might be one reason for limited performance on the generation task.

6.3. Retrieval-Augmented Generation

After the retrieval and the generation component were investigated individually in the last sections, the following section examines the complete retrieve-and-generate model. Section 6.3.1 presents architectural decisions regarding the usage of pre-trained language models, Section 6.3.2 compares RUMBArt and ACROBArT as different strategies for retrieval-augmented generation and evaluates the approaches.

6.3.1. Encoder model

Current approaches in retrieval augmented language modeling can be categorized into two architectural approaches. Systems like REALM [Guu et al., 2020] or RAG [Lewis et al., 2020b] use two distinct neural networks for retrieval and generation while MARGE [Lewis et al., 2020a] uses a classical Transformer model [Vaswani et al., 2017] and directly exploits its encoder-decoder architecture by additionally using the encoder stack for retrieval.

For this work, both approaches were studied. Of particular interest was the question of the compatibility of the different architectures with pre-trained language models and the extent to which such a system can exploit implicit knowledge of such models.

Query	average number of lightning strikes per day		
Reconstruction	how lightning of lightning strikes day		
Top-1 retrieval	Although many lightning flashes are simply cloud-to-cloud, there are as many as 9,000,000 reported lightning strikes that damage buildings, trees, and other objects every year. Worldwide, it is estimated that of an annual 1.4 billion lightning bolts, 25% (more than 350 million) will strike the Earth's surface or objects on the surface. The vast majority of these strikes, however, occur in the tropics, and in unpopulated areas. 100 times per second; Lightning can strike over a thousand times in one storm. So, lightning strikes the earth over a million times a day. Globally, 8,640,000 lightning strikes per day.		
Generation	Greedy: how often can you see lightning	Beam search: how often can you see lightning	Nucleus sampling: how often can you see lightning
Top-2 retrieval	anywhere from 20,000 to 200,000 amps can be found in a lightning bolt, It all depends on the size of the storm, the humidity of the air, the temperature, the charge in the ground and a number of other factors. Type your answer here...		
Generation	Greedy: how many in bolt	Beam search: how many in bolt	Nucleus sampling: how many in bolt
Top-3 retrieval	However, given that on average over 1,000 tornadoes hit the United States each year, that means that 20 can be expected to be violent and possibly one might be incredible (EF-5). United States Tornado Frequency and Tornado Alley		
Generation	Greedy: how many tornadoes in us	Beam search: how many tornadoes in us	Nucleus sampling: how many tornadoes in us
Query	define preventive		
Reconstruction	meaning preventive		
Top-1 retrieval	Adjective [edit]preventive (comparative more preventive, superlative most preventive) 1 Preventing, hindering, or acting as an obstacle to. Carried out to deter military aggression		
Generation	Greedy: meaning of preventive	Beam search: meaning of preventive	Nucleus sampling: meaning of preventive
Top-2 retrieval	A corrective action refers to action taken to correct a situation that has already occured, whereas a preventative action foresees that a non-conformity is likely to occur and takes pre-emptive action in order to stop it from eventuating.		
Generation	Greedy: what is a corrective action	Beam search: what is a corrective action	Nucleus sampling: what is a corrective action
Top-3 retrieval	Princeton's WordNet (0.00 / 0 votes) Rate this definition: dangerous, unsafe (adj) involving or causing danger or risk; liable to hurt or harm a dangerous criminal; a dangerous bridge; unemployment reached dangerous proportions dangerous, grave, grievous, serious, severe, life-threatening (adj) causing fear or anxiety by threatening great harm		
Generation	Greedy: meaning of danger	Beam search: meaning of danger	Nucleus sampling: what is a danger zone for potentially hazardous foods

Table 6.4.: Sample queries, generated with RUMBArt. The table shows for randomly selected queries (1) the reconstruction, generated by greedy decoding the marginalized output and (2) the top-3 retrievals each with a set of queries, generated from the respective passage and decoded using different decoding methods.

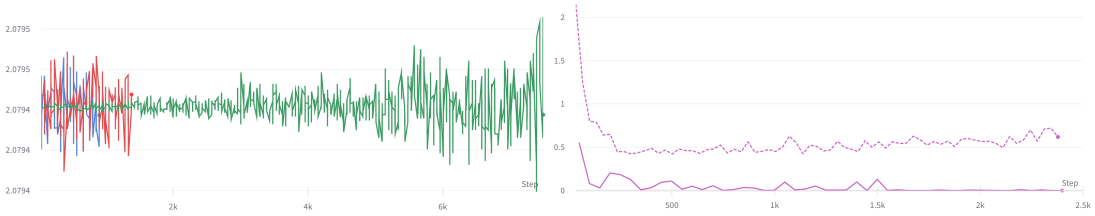
Query	Marginalized reconstruction	Marge-like reconstruction
average number of lightning strikes per day	(1) how lightning of lightning strikes day (2) how often can you see lightning	what number of lightning lightning lightning lightning
define preventive	(1) meaning preventive (2) meaning of preventive	what prevention prevention
how much are postage stamps right now	(1) how much does a stamps cost (2) current price of stamps	how much much postage postage postage stamps
symptoms of air pollution exposure	(1) what causes of air pollution pollution what are the effects of air pollution	what of air air pollution
different types of memory of a computer	(1) what is of RAM memory is is is is is is. (2) what is memory used computer	what types of of of a memory
how long to recover from wisdom teeth removal	(1) how long does it after wisdom teeth (2) how long recovery wisdom removal	how long to to removal teeth removal teeth

Table 6.5.: Examples of query reconstructions generated with RUMBArt and ACROBArT. For RUMBArt, (1) the marginalized generation and (2) the generation for the top-1 passage is given. All passages are sampled using greedy encoding.

Single model with integrated retriever: BART In a first step, MARGE's approach of using the encoder of a sequence-to-sequence transformer model simultaneously for retrieval was transferred to an encoder-decoder model already pre-trained with a different objective. For the tests, $\text{BART}_{\text{base}}$ was chosen as the pre-trained sequence-to-sequence model and the BART encoder was refined to generate good embeddings for the retrieval step. Hence, the retrieval pre-training described in Section 4.2.3 was conducted on a model initialized as $\text{BART}_{\text{base}}$, using the output vector corresponding to the `[CLS]` token in different encoder layers to generate embeddings $\text{Enc}_Q(\mathbf{x})$ and $\text{Enc}_P(\mathbf{z})$ according to Equation 4.3.

In this set of experiments, the BART encoder failed to train suitable embeddings for retrieval. Figure 6.2a shows that the training loss for 3 different layers remains at a high level. During training, the retriever was found to repeatedly draw the same few passages for all queries in a batch. It was initially suspected that this effect might be due to large differences in the norm of the embedding vectors. Since the similarity scores are calculated as dot products, vectors with large vector norms would more often obtain large scores than others. To verify the hypothesis, it was tested whether using cosine similarity as distance measure mitigates the issue, which could not be shown, even when similarity was calculated with normalized vectors, the problem remained.

Inspection of the generated embedding vectors shows that the retriever tends to generate similar embeddings for many queries, independent of the input. The effect could be observed in all encoder layers. Apparently, pre-training with the objective of maximizing the negative log likelihood of the positive passage seems to get stuck in a local optimum during gradient descent optimization, when the retriever is initialized as a BART encoder.



(a) BART encoder for retrieval. Loss for retrieval pre-training from Section 4.2.3 using the encoder output from layer 3 (blue), layer 4 (red) and layer 5 (green). The loss remains at a high level for all layers.

(b) BERT as retrieval model. Loss for retrieval pre-training from Section 4.2.3 using a $\text{BERT}_{\text{base}}$ model on train (solid) and validation data (dashed).

Figure 6.2.: Comparison of loss curves using BART and BERT as retrieval model.

Two distinct models for retrieval and generation The setup to explore the approach of using two distinct pre-trained neural networks as retrieval and generation component was the following: $\text{BART}_{\text{base}}$ was used as generator while an additional $\text{BERT}_{\text{base}}$ model served as retrieval component. Using BERT for retrieval is a common choice, BERT's effective use in neural IR has been demonstrated in many works, e.g. [Nogueira et al., 2019, Yang et al., 2019, MacAvaney et al., 2019b] or [Karpukhin et al., 2020]. Initially, the retrieval pre-training from Section 4.2.3 was tested for the $\text{BERT}_{\text{base}}$ model, using the output for the `[CLS]` token in the final layer to generate embeddings. The experiments showed that, in contrast to the BART encoder, the BERT model could be successfully fine-tuned for retrieval. Figure 6.2b shows an exemplary loss curve.

As a consequence, the two-component architecture with a BERT model as retriever and a BART model as generator is used for further experiments. Apparently, the principle used in MARGE [Lewis et al., 2020a] to create good embeddings for retrieval on the fly in a certain encoder layer cannot be easily transferred to a pre-trained language generation model like BART and it is indeed necessary to train the model from scratch. It would be interesting to investigate, whether combinations of two single-stack models (like e.g. BERT and GPT [Radford et al., 2019]) are more efficient.

6.3.2. Strategies for text generation from multiple input documents

The final step in the proposed pre-training scheme is to further improve the pre-trained retriever with the retrieve-and-generate training described in Section 4.3.

To evaluate RUMBArt and ACROBArT, the models presented in Section 4.3, both variants were trained to reconstruct randomly chosen queries from MS MARCO where the MS MARCO passages served as knowledge base. For both models, the retriever component was initialized as BERT_{base} model, pre-trained with the method described in Section 6.1, the generator is initialized as pre-trained BART_{base} model. The embedding dimension for the retrieval component is set to $n = 768$. For both models, a parameter search was performed using weights and biases[1], where the influence of most hyperparameters on the accuracy of the retriever was found to be rather moderate. For a visualization of the results see Figure 6.3 and 6.4. The models are trained for 15 epochs using a batch size of $b = 16$, accumulating gradients of 10 batches, using a linearly annealing learning rate of $lr = 1\text{e}{-}04$ and retrieving $k = 5$ documents for each query.

[1] https://wandb.ai/site

Comparison to BM25 baseline. The evaluation results on MS MARCO for RUMBArt, ACROBArT and the retriever pre-trained with distant supervision are reported in Table 6.6. The table shows results for two different evaluation settings: to establish comparability with the other results in this chapter, the models are evaluated in the reduced test setting used in all experiments, where retrieval is performed from a reduced set of 100,000 candidates. For a realistic interpretation of the results, the models are evaluated on the official passage retrieval task in MS MARCO. For this purpose, the official evaluation script of the TREC organizers[2] is used.

The results are compared against BM25 as baseline, which is measured with the elasticsearch implementation[3] from apache lucene with standard parameters.

Training	**Retriever**	Test setting		Official
		MRR@10	Recall@10	MRR@10
None	BM25	68.0	83.4	18.7
BM25 retrieval pre-training	BERT	69.1	88.1	1.0
RUMBArt	BERT	82.8	94.8	24.6
ACROBArT	BERT	78.4	92.5	20.6

Table 6.6.: Ranking results for unsupervised training schemes, measured as top-10 mean reciprocal rank and recall in percent. Results are reported for retrieval from 100.000 candidates (test setting) and for reranking on the official MS MARCO evaluation set.

Table 6.6 shows, that RUMBArt reaches 4% better MRR@10 than ACROBArT. This might be due to the fact that ACROBArT allows the generator to attend to tokens from different input passages during generation, which might result in a weaker training signal for the retriever. If an informative passage is present in the set of support passages during inference, the generator can attend to the relevant tokens, where the added retrieval score seems to be of minor importance. RUMBArt's approach to multiply the generator results with the retrieval scores seems to work better in this context.

[2] `https://github.com/usnistgov/treceval`
[3] https://www.elastic.co/de/elasticsearch/

Figure 6.3.: Parameter search for RUMBArt.

Figure 6.4.: Parameter search for ACROBArT.

7. Conclusions

This thesis proposes a novel scheme for self-supervised pre-training of a dual-encoder model for text retrieval and ranking. Two model variants were developed and evaluated on MS MARCO, achieving 20.6% and 24.6% MRR@10 on the MS MARCO passage ranking task, both outperforming the BM25 baseline while trained without supervision. In ablation studies, individual aspects and components of the model were examined in detail.

All experiments were conducted on a dataset for retrieval where explicit queries were available. Although the annotated related passages were ignored during training, the set of queries was used. The use-case of having an unstructured knowledge base and unrelated queries (e.g. from a ticketing system) may well occur in practice, but the training should also work without queries, e.g. by reconstructing sentences from the knowledge base. Alternatively, the generator component of the trained system can be used to generate synthetic queries for training since it was shown that the marginalized model is capable of creating reasonable queries. [Lu et al., 2020] uses a similar approach by training a dual encoder model on pairs of documents and synthetic queries with promising results.

An interesting question is whether the approach is suited as pre-training method for few-shot learning. Presumably, the pre-trained retrieval model can be further fine-tuned with limited labelled training data. Evaluation of the approach could be the subject of further research.

A. Symbols and Acronyms

Symbol	Meaning
d	Dimension of hidden state / embeddings.
h	Number of heads in multi-head attention layer.
l	The token length of an input sequence.
v	The size of the vocabulary.
$\mathbf{x}_i$	Embedding vector of shape d representing the ith token of the input.
$\mathbf{X} \in \mathbb{R}^{l \times d}$	The input sequence where each token has been mapped to an embedding.
$\overline{\mathbf{X}} \in \mathbb{R}^{l \times d}$	Encoded input sequence / last encoder hidden state.
$\mathbf{X}_{1:n} \in \mathbb{R}^{n \times d}$	Input sequence of length n.
$\mathbf{Y}_{1:m} \in \mathbb{R}^{m \times d}$	The target sequence of length m.
d_k, d_v	Dimension of the query / key resp. the value vectors. Often $d_k = d_v$.
$\mathbf{W}^Q \in \mathbb{R}^{d \times d_k}$	The query weight matrix.
$\mathbf{W}^K \in \mathbb{R}^{d \times d_k}$	The key weight matrix.
$\mathbf{W}^V \in \mathbb{R}^{d \times d_v}$	The value weight matrix.
$\mathbf{W}_i^Q, \mathbf{W}_i^K \in \mathbb{R}^{d \times d_k/h}$	The query / key weight matrices per head.
$\mathbf{W}_i^V \in \mathbb{R}^{d \times d_v/h}$	The value weight matrix per head.
$\mathbf{W}^O \in \mathbb{R}^{h \cdot d_v \times d}$	The output weight matrix.
$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}^Q \in \mathbb{R}^{l \times d_k}$	The Query matrix.
$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}^K \in \mathbb{R}^{l \times d_k}$	The Key matrix.
$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}^V \in \mathbb{R}^{l \times d_v}$	The Value matrix.
a_{ij}	The scalar attention score between query $\mathbf{q_i}$ and key $\mathbf{k_j}$.
$\mathbf{x} = \{x_1, \ldots, x_n\}$	Input query, consisting of n tokens.
$\mathbf{y} = \{y_1, \ldots, y_m\}$	The tokens for the ground truth label of length m
$\mathbf{Z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$	The set of k passages (documents) retrieved for a query $\mathbf{x}$
$\mathcal{Z}$	The knowledge corpus. Usually unstructured data, such as Wikipedia
$\text{Enc}_Q(\mathbf{x}_i)$	The representation of a query in the retrieval mechanism
$\text{Enc}_P(\mathbf{z}_j)$	The representation of a support passage in the retrieval mechanism

Symbol	Meaning
$\mathbf{p}_\eta(\mathbf{z}_j \vert \mathbf{x}_i)$	The retrieval probability of selecting a passage $\mathbf{z}_j$ for a query $\mathbf{q}_i$
$\mathbf{p}_\theta(\mathbf{y}_j \vert \mathbf{z}_1, \ldots, \mathbf{z}_m, y_1, \ldots, y_{j-1})$	The probability of outputting a token $\mathbf{y}_j$ given a set of passages $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\}$ and the previous tokens $\{y_1, \ldots, y_{j-1}\}$.
$\mathbf{L} \in \mathbb{R}^{l \times v}$	The logit vector which models $\mathbf{p}_\theta(\mathbf{y}_j \vert \mathbf{z}_1, \ldots, \mathbf{z}_m, y_1, \ldots, y_{j-1})$ over the tokens of the vocabulary.
$\vert X \vert$	Cardinality of a set X
$\mathsf{sim}(\mathbf{x}, \mathbf{z})$	Similarity between query $\mathbf{x}$ and passage $\mathbf{z}$

Bibliography

[Akkalyoncu Yilmaz et al., 2019] Akkalyoncu Yilmaz, Z., Yang, W., Zhang, H., and Lin, J. (2019). Cross-domain modeling of sentence-level evidence for document retrieval. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3490–3496, Hong Kong, China. Association for Computational Linguistics.

[Alammar, 2018] Alammar, J. (2018). The illustrated transformer [blog post].

[Asai et al., 2020] Asai, A., Hashimoto, K., Hajishirzi, H., Socher, R., and Xiong, C. (2020). Learning to retrieve reasoning paths over wikipedia graph for question answering.

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

[Bajaj et al., 2018] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., and Wang, T. (2018). Ms marco: A human generated machine reading comprehension dataset.

[Balachandran et al., 2021] Balachandran, V., Vaswani, A., Tsvetkov, Y., and Parmar, N. (2021). Simple and efficient ways to improve REALM. *CoRR*, abs/2104.08710.

[Bender et al., 2021] Bender, E. M., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA. Association for Computing Machinery.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners.

[Chaudhari et al., 2019] Chaudhari, S., Mithal, V., Polatkan, G., and Ramanath, R. (2019). An Attentive Survey of Attention Models. *arXiv e-prints*, page arXiv:1904.02874.

[Chen et al., 2017] Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading wikipedia to answer open-domain questions.

[Cho et al., 2015] Cho, K., Courville, A., and Bengio, Y. (2015). Describing multimedia content using attention-based encoder-decoder networks. *IEEE Transactions on Multimedia*, 17(11):1875–1886.

[Craswell et al., 2020] Craswell, N., Mitra, B., Yilmaz, E., Campos, D., and Voorhees, E. M. (2020). Overview of the trec 2019 deep learning track. In *Text REtrieval Conference (TREC)*. TREC.

[Dai and Callan, 2019] Dai, Z. and Callan, J. (2019). Deeper text understanding for ir with contextual neural language modeling. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding.

[Fan et al., 2018] Fan, A., Lewis, M., and Dauphin, Y. (2018). Hierarchical neural story generation.

[Galassi et al., 2020] Galassi, A., Lippi, M., and Torroni, P. (2020). Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–18.

[Gao et al., 2021] Gao, L., Dai, Z., and Callan, J. (2021). Rethink training of bert rerankers in multi-stage retrieval pipeline.

[Gillick et al., 2019] Gillick, D., Kulkarni, S., Lansing, L., Presta, A., Baldridge, J., Ie, E., and García-Olano, D. (2019). Learning dense representations for entity retrieval. In Bansal, M. and Villavicencio, A., editors, *Proceedings of the 23rd Conference on*

Computational Natural Language Learning, CoNLL 2019, Hong Kong, China, November 3-4, 2019, pages 528–537. Association for Computational Linguistics.

[Guo et al., 2020] Guo, J., Fan, Y., Pang, L., Yang, L., Ai, Q., Zamani, H., Wu, C., Croft, W. B., and Cheng, X. (2020). A deep look into neural ranking models for information retrieval. *Information Processing & Management*, 57(6):102067.

[Guu et al., 2020] Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. (2020). Realm: Retrieval-augmented language model pre-training.

[Hofstätter and Hanbury, 2019] Hofstätter, S. and Hanbury, A. (2019). Let's measure run time! extending the ir replicability infrastructure to include performance aspects.

[Holtzman et al., 2020] Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. (2020). The curious case of neural text degeneration.

[Howard and Ruder, 2018] Howard, J. and Ruder, S. (2018). Universal language model fine-tuning for text classification.

[Huang et al., 2013] Huang, P.-S., He, X., Gao, J., Deng, l., Acero, A., and Heck, L. (2013). Learning deep structured semantic models for web search using clickthrough data. pages 2333–2338.

[Izacard and Grave, 2020a] Izacard, G. and Grave, E. (2020a). Distilling knowledge from reader to retriever for question answering. *CoRR*, abs/2012.04584.

[Izacard and Grave, 2020b] Izacard, G. and Grave, E. (2020b). Leveraging passage retrieval with generative models for open domain question answering. *CoRR*, abs/2007.01282.

[Jiang et al., 2020] Jiang, Z., Xu, F. F., Araki, J., and Neubig, G. (2020). How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

[Johnson et al., 2019] Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*.

[Jones, 2004] Jones, K. S. (2004). A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 60(5):493–502.

[Karpukhin et al., 2020] Karpukhin, V., Oğuz, B., Min, S., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*.

[Khandelwal et al., 2020a] Khandelwal, U., Fan, A., Jurafsky, D., Zettlemoyer, L., and Lewis, M. (2020a). Nearest neighbor machine translation. *CoRR*, abs/2010.00710.

[Khandelwal et al., 2020b] Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L., and Lewis, M. (2020b). Generalization through memorization: Nearest neighbor language models.

[Khattab and Zaharia, 2020] Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In Huang, J., Chang, Y., Cheng, X., Kamps, J., Murdock, V., Wen, J., and Liu, Y., editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 39–48. ACM.

[Kingma and Ba, 2017] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[Kulis et al., 2012] Kulis, B. et al. (2012). Metric learning: A survey. *Foundations and trends in machine learning*, 5(4):287–364.

[Lan et al., 2020] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations.

[Lee et al., 2019] Lee, K., Chang, M.-W., and Toutanova, K. (2019). Latent retrieval for weakly supervised open domain question answering.

[Lewis et al., 2020a] Lewis, M., Ghazvininejad, M., Ghosh, G., Aghajanyan, A., Wang, S., and Zettlemoyer, L. (2020a). Pre-training via paraphrasing. *arXiv preprint arXiv:2006.15020*.

[Lewis et al., 2019] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

[Lewis et al., 2020b] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2020b). Retrieval-augmented generation for knowledge-intensive nlp tasks.

[Lin et al., 2020] Lin, J., Nogueira, R., and Yates, A. (2020). Pretrained transformers for text ranking: Bert and beyond.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

[Lu et al., 2020] Lu, J., Ábrego, G. H., Ma, J., Ni, J., and Yang, Y. (2020). Neural passage retrieval with improved negative contrast. *CoRR*, abs/2010.12523.

[Lu et al., 2017] Lu, J., Xiong, C., Parikh, D., and Socher, R. (2017). Knowing when to look: Adaptive attention via a visual sentinel for image captioning.

[Luan et al., 2021] Luan, Y., Eisenstein, J., Toutanova, K., and Collins, M. (2021). Sparse, dense, and attentional representations for text retrieval.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation.

[MacAvaney et al., 2019a] MacAvaney, S., Yates, A., Cohan, A., and Goharian, N. (2019a). Cedr. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[MacAvaney et al., 2019b] MacAvaney, S., Yates, A., Cohan, A., and Goharian, N. (2019b). Cedr: Contextualized embeddings for document ranking. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1101–1104.

[Maynez et al., 2020] Maynez, J., Narayan, S., Bohnet, B., and McDonald, R. T. (2020). On faithfulness and factuality in abstractive summarization. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J. R., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 1906–1919. Association for Computational Linguistics.

[Melamud et al., 2016] Melamud, O., Goldberger, J., and Dagan, I. (2016). context2vec: Learning generic context embedding with bidirectional lstm. In *Proceedings of the 20th SIGNLL conference on computational natural language learning*, pages 51–61.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality.

[Miller et al., 2016] Miller, A., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. (2016). Key-value memory networks for directly reading documents.

[Min et al., 2019] Min, S., Chen, D., Hajishirzi, H., and Zettlemoyer, L. (2019). A discrete hard em approach for weakly supervised question answering.

[Mitra and Craswell, 2018] Mitra, B. and Craswell, N. (2018). An introduction to neural information retrieval. *Found. Trends Inf. Retr.*, 13(1):1–126.

[Mitra et al., 2017] Mitra, B., Diaz, F., and Craswell, N. (2017). Learning to match using local and distributed representations of text for web search. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1291–1299.

[Nogueira and Cho, 2019] Nogueira, R. and Cho, K. (2019). Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

[Nogueira and Cho, 2020] Nogueira, R. and Cho, K. (2020). Passage re-ranking with bert.

[Nogueira et al., 2019] Nogueira, R., Yang, W., Cho, K., and Lin, J. (2019). Multi-stage document ranking with bert.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

[Petroni et al., 2019] Petroni, F., Rocktäschel, T., Lewis, P., Bakhtin, A., Wu, Y., Miller, A. H., and Riedel, S. (2019). Language models as knowledge bases?

[Press and Wolf, 2017] Press, O. and Wolf, L. (2017). Using the output embedding to improve language models.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

[Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer.

[Ram and Gray, 2012] Ram, P. and Gray, A. G. (2012). Maximum inner-product search using cone trees. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 931–939.

[Reimers and Gurevych, 2019] Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In Inui, K., Jiang, J., Ng, V., and Wan, X., editors, *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*, pages 3980–3990. Association for Computational Linguistics.

[Robertson et al., 1995] Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., Gatford, M., et al. (1995). Okapi at trec-3. *Nist Special Publication Sp*, 109:109.

[Robertson and Zaragoza, 2009] Robertson, S. E. and Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

[Rogers et al., 2020] Rogers, A., Kovaleva, O., and Rumshisky, A. (2020). A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

[Ruder et al., 2019] Ruder, S., Peters, M. E., Swayamdipta, S., and Wolf, T. (2019). Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18.

[Rudra and Anand, 2020] Rudra, K. and Anand, A. (2020). Distant supervision in bert-based adhoc document retrieval. In d'Aquin, M., Dietze, S., Hauff, C., Curry, E., and Cudré-Mauroux, P., editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 2197–2200. ACM.

[Sanh et al., 2020] Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

[Sennrich et al., 2016] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units.

[Shen et al., 2015] Shen, F., Liu, W., Zhang, S., Yang, Y., and Tao Shen, H. (2015). Learning binary codes for maximum inner product search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4148–4156.

[Shrivastava and Li, 2014] Shrivastava, A. and Li, P. (2014). Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). *arXiv preprint arXiv:1405.5869*.

[Shuster et al., 2021] Shuster, K., Poff, S., Chen, M., Kiela, D., and Weston, J. (2021). Retrieval augmentation reduces hallucination in conversation. *CoRR*, abs/2104.07567.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Vijayakumar et al., 2018] Vijayakumar, A. K., Cogswell, M., Selvaraju, R. R., Sun, Q., Lee, S., Crandall, D., and Batra, D. (2018). Diverse beam search: Decoding diverse solutions from neural sequence models.

[Wang and Tax, 2016] Wang, F. and Tax, D. M. J. (2016). Survey on the attention based rnn model and its applications in computer vision.

[Wieting and Kiela, 2019] Wieting, J. and Kiela, D. (2019). No training required: Exploring random encoders for sentence classification. *arXiv preprint arXiv:1901.10444*.

[Wolf et al., 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

[Wu et al., 2020] Wu, Q., Xing, C., Li, Y., Ke, G., He, D., and Liu, T.-Y. (2020). Taking notes on the fly helps bert pre-training.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil,

N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

[Xiong et al., 2017] Xiong, C., Dai, Z., Callan, J., Liu, Z., and Power, R. (2017). End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, pages 55–64.

[Yang et al., 2019] Yang, W., Zhang, H., and Lin, J. (2019). Simple applications of bert for ad hoc document retrieval.

[Yang et al., 2018] Yang, Y., Huang, L., and Ma, M. (2018). Breaking the beam search curse: A study of (re-)scoring methods and stopping criteria for neural machine translation.

[Zhang et al., 2020a] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. (2020a). *Dive into Deep Learning*. `https://d2l.ai`.

[Zhang et al., 2020b] Zhang, J., Zhao, Y., Saleh, M., and Liu, P. (2020b). Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR.

[Zheng et al., 2021] Zheng, X., Zhang, Z., Guo, J., Huang, S., Chen, B., Luo, W., and Chen, J. (2021). Adaptive nearest neighbor machine translation. *CoRR*, abs/2105.13022.

[Zhu et al., 2015] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.