TÉCNICO LISBOA

Dense Template Retrieval for Customer Support

Tiago Manuel Reis Mesquita

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Bruno Emanuel da Graça Martins
Dra. Mariana S. C. Almeida

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Bruno Emanuel da Graça Martins
Member of the Committee: Prof. João Miguel da Costa Magalhães

November 2021

This work was created using LaTeX typesetting language
in the Overleaf environment (www.overleaf.com).

Acknowledgments

Firstly, I would like to thank my supervisors, Prof. Bruno Martins and Mariana Almeida, for the valued guidance they provided throughout the entirety of of my M.Sc. research project. Our meetings and exchanged messages were an integral part of this thesis, providing the answers to my questions, the direction to guide my research, the knowledge to tackle the next challenges and the motivation and confidence to keep working. I'd also like to thank you for the active participation on the elaboration of this report and the accompanying paper, where you were always available to read and review my preliminary versions. Without you, this work would not have been possible.

I would also like to express my gratitude to the team at Cleverly, for welcoming me and making this collaboration possible. In particular, I would like to thank Inês and Lourenço, for their insightful reading groups, of which I was lucky to attend, and Pedro for providing all the necessary hardware support. I would like to thank André Godinho, who I had the pleasure to accompany trough most of our theses. Watching you accomplishing your thesis' goals was truly inspiring. I would also like to thank everyone at Zendesk, that was involved in the review and approval of our paper, for submission in such short notice.

I'd also like to thank all of my close friends, for always being there. João Barata and João Daniel, thank you for reminding me that even though we're in different journeys, we are in them together.

Last but definitely not least, a special thank you for my family. Thank you for the emotional and financial support that allowed me to truly immerse myself in this thesis. I hope my work reflects your effort.

Abstract

Templated answers are used extensively in customer support scenarios, providing an efficient way to cover a plethora of topics, with an easily maintainable small collection of templates. Still, the number of templates is often too high for an agent to search. Automatically suggesting the correct template for a given question can improve the service efficiency, reducing costs and leading to a better customer satisfaction. In this work, we adapt the dense retrieval framework for the customer support scenario, modifying the commonly used in-batch negatives technique to support unpaired sampling of queries and templates. We also propose a novel loss that extends the typical query-centric similarity, exploiting other similarity relations in the training data. Experiments on private and public datasets show that our approach achieves considerable improvements in terms of performance and training speed.

Keywords

Information Retrieval, Customer Support, Template Retrieval, Dense Retrieval

Resumo

Respostas pré-formatadas são utilizadas extensivamente em cenários de suporte ao cliente, constituindo um método eficiente de cobrir uma infinidade de tópicos, com uma coleção de modelos de resposta de fácil manutenção. Ainda assim, o número de modelos é geralmente grande de mais para um agente pesquisar. A sugestão automática do modelo correto, para uma determinada questão, pode melhorar a eficiência do serviço, reduzindo custos e levando a uma melhor satisfação do cliente. Neste trabalho, adaptamos a estrutura de recuperação de informação com representações densas, para o cenário de suporte ao cliente, modificando a técnica frequentemente usada de *in-batch negatives* para suportar a amostragem não pareada de questões e modelos de resposta. Também propomos uma nova função de custo que estende a típica similaridade centrada nas questões, explorando outras relações de similaridade nos dados de treino. Experiências em dados privados e públicos mostram que a nossa abordagem consegue melhorias consideráveis, tanto em desempenho como velocidade de treino.

Palavras Chave

Recuperação de Informação, Suporte ao Cliente, Recuperação de Modelos de Resposta, Recuperação de Informação com Representações Densas

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI	Artificial Intelligence
ADORE	Algorithm for Directly Optimizing Ranking pErformance
ANCE	Approximate Nearest Neighbor Negative Contrastive Estimation
ANN	Approximate Nearest Neighbour
BERT	Bidirectional Encoder Representations from Transformers
BM25	Best Match 25
BoW	Bag-of-Words
CBoW	Continuous Bag-of-Words
DeepCT	Deep Contextualized Term Weighting
DPR	Dense Passage Retriever
CLEAR	Complementing Lexical Retrieval with Semantic Residual Embedding
FF	Feed-Forward
FFNN	Feed-Forward Neural Network
GD	Gradient Descent
IDF	Inverse Document Frequency
IR	information retrieval
MLM	Masked Language Modeling
MTL	multi-task learning
MRR	Mean Reciprocal Rank
NN	Neural Networks
NNS	Nearest Neighbours Search

NLL	negative log likelihood
NLP	Natural Language Processing
NSP	Next Sentence Prediction
LTRe	Learning To Retrieve
QTR	Query Term Recall
PAIR	PAssage-centric sImilarity Relations
RepBERT	Representation-focused BERT
ReLU	Rectified Linear Unit
RNN	Recursive Neural Networks
seq2seq	sequence-to-sequence
SOTA	state of the art
STAR	Stable Training Algorithm for dense Retrieval
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
tanh	Hyperbolic Tangent

1

Introduction

Contents

This thesis was developed in the context of a collaboration between INESC-ID and two companies: Zendesk[1], that provides a platform for digital custommer support, and Cleverly[2], that offers an Artificial Intelligence (AI) layer with features for optimizing customer support processes. Our work explores neural approaches for document retrieval, in order to support the development of a new feature that leverages AI to improve customer support.

1.1 Problem Definition

Customer support is an essential complement for any company supplying services or products, guaranteeing their intended use. As such, the effectiveness of this interaction has direct impact in customer's perception and satisfaction. Ideally, the communication would be fully conducted by an human agent, (fully automated agents are, for the most part, still unable to compete), offering tailored help for each customer. However, this is unfeasible in most scenarios, as the sheer amount of requests far outweigh the human power.

In the case of email support, a common way to increase the efficiency of customer support agents is with the use of templates, pre-written responses that answer a plethora of known requests, simplifying the customer support agents' process of replying to a choice of the most appropriate template, and standardizing the replies to a reduced set of possible answers. Template answers not only severally improve the throughput of human agents, they also assure uniformity in the handling of different customers, as any requests, with the same underlying problem, should be handled with the same template. Although the use of templates facilitates agents work, finding the right template can still be a cumbersome, particularly for unexperienced agents since depending on the complexity of the product or service, several distinct questions may arise, each with its own template. This provides a great opportunity for automated agents to simplify the process, by providing a pre-selection of the most likely correct templates, from which a human operator can select the most appropriate, as seen in figure 1.1.

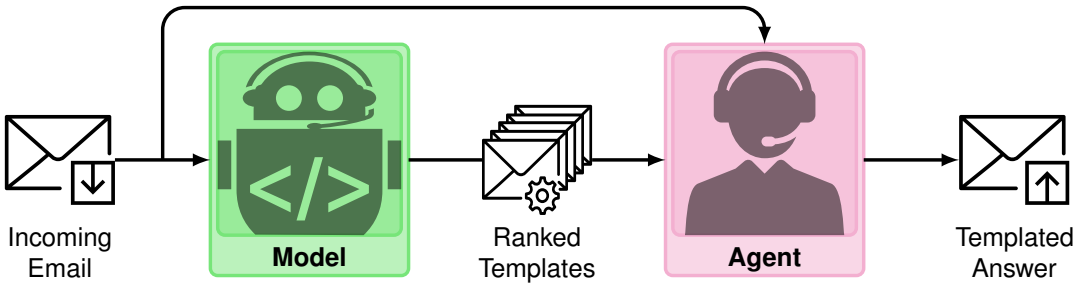


Figure 1.1: Overview of the proposed template suggestion framework.

[1] https://www.zendesk.com/
[2] https://www.cleverly.ai/

If effective, the described framework severely simplifies the choice of the template, facilitating agent's work by reducing reply times, accelerating the learning curve of new agents, helping agents to focus on more added valued tasks, and overall providing a better support at reduced costs.

Despite this, literature on the topic is somewhat scarce, likely due to a unavailability of public data, with most work being conducted privately, under in-house datasets. Particularly, the described scenario differs from the ones in the literature, as it features small collections of templates, data that is generally multilingual, lengthier texts, and an emphasis on efficiency and speed to support real-time utilization.

1.2 Objective

This thesis focuses on Cleverly's AI feature of suggesting templates of replies to agents. Considering this application, and the recent advances in large pre-trained language models (Devlin et al., 2019; Vaswani et al., 2017), together with their successful use in question-answering (Karpukhin et al., 2020; Qu et al., 2021) and information retrieval (Xiong et al., 2021; Zhan et al., 2021, 2020a), the main goal of this research was:

> *to study the feasibility of using state of the art (SOTA) dense retrieval models to retrieve templates, helping customer support agents answer incoming requests*

Motivated by this main objective, the thesis explores the practical differences between template retrieval and common information retrieval (IR) scenarios in the literature, and how they affect the efficacy and efficiency of the prevailing training techniques. In particular, we aim to improve and adapt the existing techniques, to better address the specific challenges of the problem at hand, such as:

i The relation between queries and templates is strict many-to-one, as opposed to other common retrieval tasks as discussed by Nguyen et al. (2016);

ii Template collections are relatively small and generally in the order of hundreds, although also dynamically updated over time;

iii The frequency of use of each template differs considerably, resulting in skewed distributions of training examples;

iv The length of the queries (e.g., emails from costumers) tends to be relatively long, severely limiting the available batch space;

v Models must provide template rankings in real-time, maximizing the customer support efficiency;

vi Training speed is highly regarded, as it facilitates quick deployment;

vii Data is generally multilingual, so we need a solution applicable to a vast range of languages.

1.3 Methodology

To achieve the objectives defined in the previous section, we started by surveying the recent contributions in the field of IR with transformer-based models, collecting the techniques and design principles that guided our implementation and experiments. Given real-time and computation constrains in the template suggestion problem, research mostly focused on bi-encoder models (Yates et al., 2021), which at prediction time only need to compute dense representations of queries and make fast comparisons with pre-computed representations of template candidates, reaching fast retrieval times and reduced computational costs. We discard cross-encoder models that, despite often achieving higher retrieval performance (Yates et al., 2021) can have problems processing long queries and/or documents, failing to take advantage of pre-computed template representations.

Following our research, we proceeded with the implementation and adaptation of the collected techniques for the scenario of customer support with templates. The design of the testing framework focused heavily on modularity, supporting models that combine different combinations of the surveyed techniques. Its implementation leveraged PyTorch[3], together with pre-trained models from HuggingFace Transformers[4] and Sentence-Transformers[5].

Finally, we moved onto the evaluation and experimentation of the produced models, comparing them across the different methods and datasets. Experiments were mostly conducted on four private datasets, kindly provided by Cleverly and Zendesk. These datasets featured anonymized real-world customer support interactions, where answer templates were carefully selected by human operators. They combined differently sized template collections, number of training examples and languages, providing a representative evaluation of model performance, across a range of possible real scenarios. To assure the reproducibility of the results, and in the absence of a public dataset of customer support that met our criteria, we also created a new dataset, using the data available in the Twitter Customer Support Corpus, from Kaggle[6]. To test the models, we split each dataset into training, validation and test splits. The test split was extracted temporally, corresponding to the most recent interactions, mimicking the real scenario, where the model is used to suggest templates for future queries. Training and validation were composed from the remaining examples, in a stratified split.

To evaluate the performance of the tested models, we used metrics that capture the quality of the produced rankings, namely Mean Reciprocal Rank (MRR) and recall.

[3] https://pytorch.org/
[4] https://huggingface.co/transformers/
[5] https://www.sbert.net/
[6] https://www.kaggle.com/thoughtvector/customer-support-on-twitter

1.4 Contributions

The main ideas and research contributions of this M.Sc thesis work are the following:

- Proposal of a novel model for semi-automatic selection of answer templates, leveraging dense retrieval over the template collection;
- Comparison of the proposed model, with classic and recent dense retrieval approaches in the task of template retrieval for customer support;
- Creation and release [7] of a corpus for template retrieval based on the *Customer Support on Twitter* dataset that is available on Kaggle, in order to motivate further research and benchmarking on the topic of information retrieval for customer support;
- Proposal of a new in-batch sampling strategy, that preserves the distributions of queries and templates to better select the information within batches, while exploring all possible query-template pairs in a batch. Experiments showed that this technique was key in combating the skewed distributions resultant of the strict many-to-one relation between queries and templates, not only achieving better performance, but also faster training;
- Proposal of a new loss function that exploits not only query-template similarity relations, but also query-query and template-template relations, yielding better representations for retrieval. Experiments showed that this loss consistently outperformed the former, despite utilizing the same information available within the batch;

On a final note, this thesis contributed innovative techniques with positive results, which motivated the submission of a paper to ECIR 2022[8], taking place in Stavanger, Norway.

1.5 Organization of the Dissertation

The rest of this dissertation is organized as follows: Chapter 2 recaps basic concepts related to deep learning with neural networks, along with the basis for understanding the state of the art in NLP. Chapter 3 examines related work on neural ranking and retrieval. Chapter 4 describes the proposed model and techniques. Chapter 5 reports the evaluation of the proposed techniques. Finally, Chapter 6 discusses the main conclusions from this work and possible ideas for future developments.

[7] https://github.com/t-mesq/twitter-apple-cs-hdbscan
[8] https://ecir2022.org/

2

Fundamental Concepts

Contents

This section holds the fundamentals for understanding the state of the art (SOTA) in information retrieval (IR), namely the essentials for BERT (Devlin et al., 2019) based neural models (Section 2.1) and text representation (Section 2.2).

2.1 Neural Models

IR has seen a vast variety of neural models in the recent past with considerably different approaches. Recently, however, models based on BERT's (Devlin et al., 2019) architecture have taken the lead, generally with no opposition, and as such only this types of models will be considered. The topics addressed in this section are the absolute fundamentals to understand the BERT model.

Artificial Neural Networks (NN) are models based on their biological counterpart, the connections in the brain. Likewise, they are able to perform supervised learning, that is, given a set of inputs and their respective outputs, learn the underlying mapping function. Formally, a input $\mathbf{x}$ is a vector of dimension N_x and the output $\mathbf{y}$ is a vector of dimension N_y.

2.1.1 Feed-Forward Models

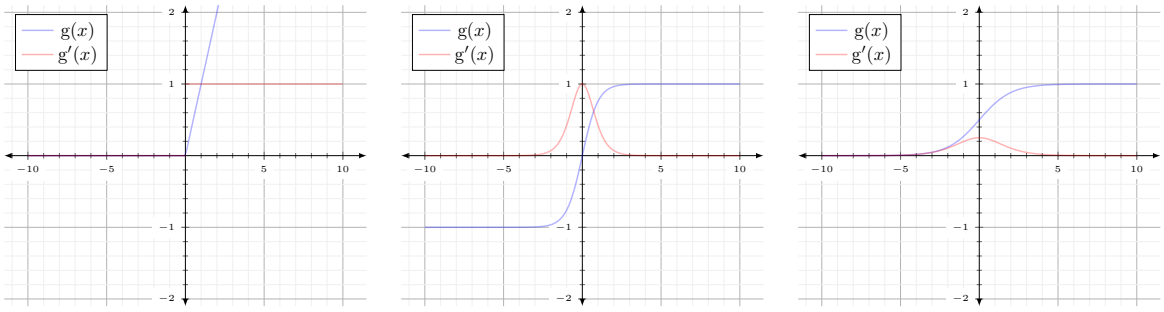
The simplest case for a NN maps the input x into the output y with an affine transformation:

$$\mathbf{y} = \mathbf{x}\mathbf{W} + \mathbf{b}, \tag{2.1}$$

where $\mathbf{W}$ is a weights matrix, of dimensions $N_x \times N_y$ and $\mathbf{b}$ is the bias vector, of dimension N_y. The biological intuition behind this formula is that each output dimension has a corresponding neuron, that receives $\mathbf{x}$ as an input. The weight $w_{i,j}$ characterizes how much the dimension x_i stimulates the neuron responsible for y_j and the bias b_j is then the predisposition for stimulus of said neuron. From the formula alone, it is immediate that it can only learn linear functions. For more complex problems however, one could consider composing several of this layers, where a layer l_n is given as input to the layer l_{n+1}. However, this formally corresponds to the same affine model of Equation 2.1:

$$\begin{aligned} \mathbf{y} &= (\mathbf{x}\mathbf{W^1} + \mathbf{b^1})\mathbf{W^2} + \mathbf{b^2} \\ &= \mathbf{x}(\mathbf{W^1}\mathbf{W^2}) + (\mathbf{b^1}\mathbf{W^2} + \mathbf{b^2})) \\ &= \mathbf{x}\mathbf{W} + \mathbf{b}, \end{aligned} \tag{2.2}$$

with $\mathbf{W^n}$ and $\mathbf{b^n}$, representing the weight matrix and bias vector of layer l_n, respectively. Thankfully the solutions to this problem is also found in nature. A biological neuron only activates after the stimulus surpasses a certain threshold. This functionality can be achieved by passing the output of the layer to a specialized function that attenuates values below a certain threshold (usually 0 or 0.5) and augments



(a) ReLU activation $(\max(0, x))$ **(b)** tanh activation $(\tanh(x))$ **(c)** sigmoid activation $\left(\frac{1}{e^{-x}+1}\right)$

Figure 2.1: Common activation functions $(\mathrm{g}(x))$ and their respective 1st derivative $(\mathrm{g}'(x))$

values above it. Some examples that fit this properties are sigmoid, Rectified Linear Unit (ReLU) and Hyperbolic Tangent (tanh), and their plots and formulas can be seen in Figure 2.1.

Finally, the revised Equation (2.1) becomes:

$$\mathbf{y} = \mathrm{g}(\mathbf{x}\mathbf{W} + \mathbf{b}), \tag{2.3}$$

where $g(z)$ is the activation function. This entire process can be easily visualized as a network like the one in Figure 2.2, where each layer n corresponds to the output vector at each intermediate step, composed of nodes correspondent with each of its features. The connections between each layer correspond to each individual weight in the matrix $\mathbf{W^n}$, representing the influence of each feature, in the next layer. The weights in the bias vector $\mathbf{b^n}$, also influence the next layer and can be thought of as an hidden node with its own set of weighs (this is usually implicit but represented here to aid visualization). The output of each layer is implicitly activated as in accordance with Equation 2.3.

We now have a model that can theoretically reproduce any complex function within a small error (Hornik, 1991), however, we still need a way to find the weights and biases needed to reproduce it, that is, some algorithm that can learn those parameters from a training set.

2.1.2 Training with Gradient Descent

Gradient Descent (GD) is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. In other words, it uses the derivative of the function in a given point, to get a new point, closer to a minimum of the function. This process is repeated for the new point and so on, until it converges, in a local or possibly global minimum. By considering a differentiable function that represents the the dissimilarity between the distribution predicted by the NN, and the true distribution, we can effectively train the model's parameters (weights and bias) over GD, since minimizing said function,

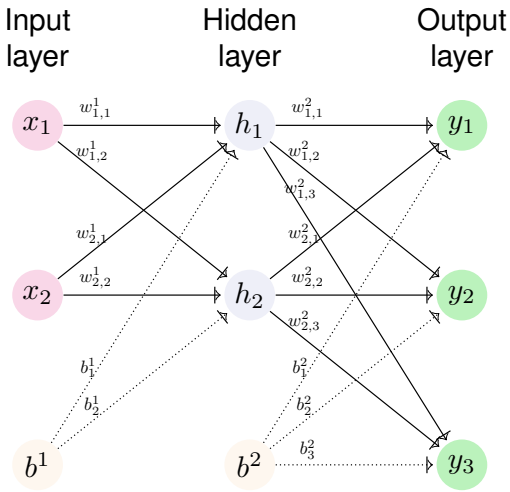


Figure 2.2: Graphical representation of a multi-layer perceptron (feed-forward), with an input layer of dimension 2, an hidden layer of dimension 2 and an output layer of dimension 3

approximates the distributions. Such a function is called a loss function and different variants exist, according to the needs of the problem to approximate. For instance, Mean Squared Error (MSE) can be used for regression problems, and the Cross Entropy (Log Loss) can be used for classification.

Since the loss function only applies to the output layer of the NN, the one that outputs the actual predictions, we need a way to propagate the error to the weights and bias prior to it, which is accomplished through the back-propagation algorithm (Rumelhart et al., 1986). The algorithm starts by performing a forward pass, where the NN takes the input and processes it across all layers until the output is produced. The output is fed to the loss function, comparing it to the ground truth, and outputting the prediction error. The error is then propagated back through the NN, updating the parameters of each node. The update is given by the partial derivatives of the loss function relative to the parameters of the previous layers. In order to minimize the loss, this gradient is subtracted to the parameters, as gradient descent update, formally:

$$\theta = \theta - \eta \nabla_\theta \mathrm{L}(\theta), \tag{2.4}$$

where θ are the parameters of the model $\{\mathbf{W}, \mathbf{b}\}$, $\mathrm{L}(\cdot)$ is the loss function and η is the learning rate, an hyper-parameter responsible for the size of the learning step. If this hyper-parameter is too small, it will lead to a really slow convergence, since each update is very small, whilst values too big will lead to oscillations, since the model will constantly over estimate θ which could cost convergence.

There are 3 main variants of GD, differing in the amount of data required to compute the gradient of the loss function:

A – Batch Gradient Descent Batch gradient descent is the default model where in each step, the gradient of the loss function is calculated over the entire training set and the parameters are adjusted accordingly. In general this is the smoothest, but its costly updates, requiring the full dataset to be loaded into memory, make it prohibitive for large datasets.

B – Stochastic Gradient Descent Stochastic gradient descent is the most commonly used, as it is computationally cheapest. In each step, a single training example is used to estimate the gradient and update the parameters accordingly. This makes each update considerably more irregular, as training examples may vary tremendously, making careful consideration regarding the learning rate crucial for achieving convergence.

C – Mini-Batch Gradient Descent Mini-batch gradient descent tries to join the best of both previous methods. The data set is first divided into smaller batches of size $k \ll N$. Then, in each step, the loss is calculated for every example in a batch and the update is performed similarly to the batch variant. This is then repeated for each batch. This way the batch computations can be performed efficiently trough matrix operations, leading to much stabler gradients than the stochastic variant. The hyper-parameter k can be adjusted according to the data set size, making it easily adaptable to the data and the hardware.

2.1.3 Seq2Seq Architectures

A sequence-to-sequence (seq2seq) model takes an input sequence of items (words, letters, features of an images. . . etc) and outputs another sequence of items. In the case of NLP problems, they effectively solve the problem of texts having arbitrary length, as they're treated as sequences of words or subwords.

In order to process sequences, seq2seq models have an encoder-decoder architecture, where in general terms, a Encoder model processes a sequence, generating a fixed representation of it, commonly a vector or a set of vectors known as context. Then a Decoder model, usually very similar in architecture, albeit in reverse order, produces the output sequence, from said context. In order for this to work, both (encoder and decoder) models are trained at the same time, with the decoder trying to predict what the encoder encoded. Although encoder-decoder's history is deeply intertwined with Recursive Neural Networks (RNN) models[1], other approaches have surfaced, namely by using fixed length input sequences, with padding for the unused positions. Such a model will be addressed in the next section.

[1] RNN models will not be addressed on this thesis, but the survey from Sherstinsky (2020) is recommended for interested readers.

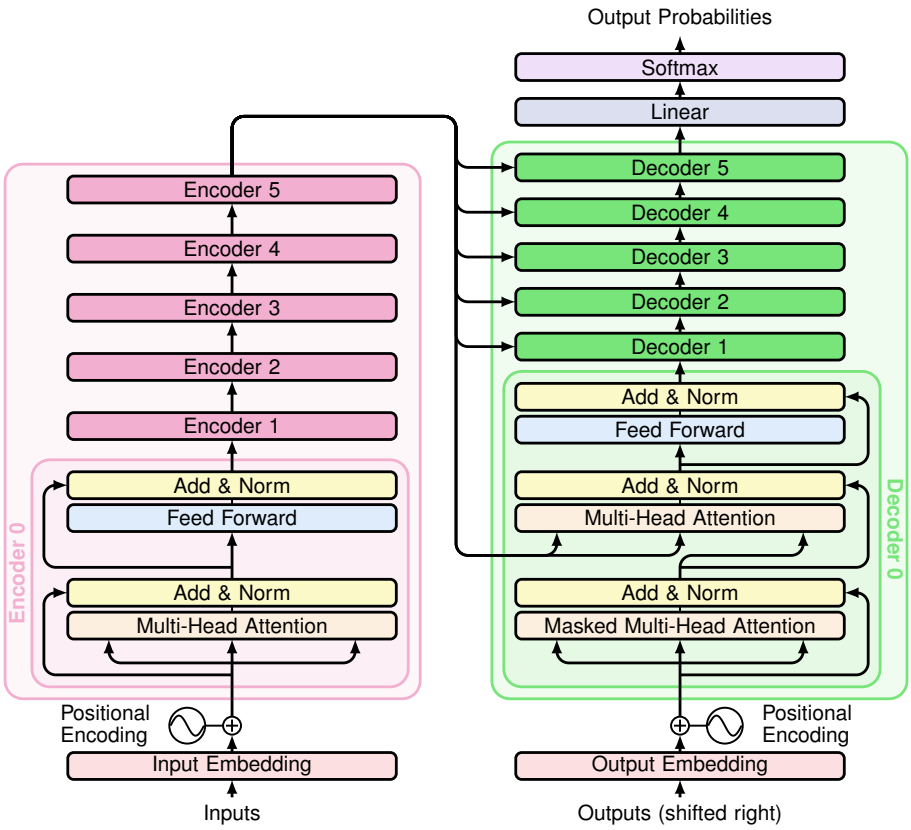


Figure 2.3: The Transformer - model architecture

2.1.4 The Transformer Architecture

The Transformer (Vaswani et al., 2017) is a neural model based on the encoder-decoder (i.e., sequence-to-sequence) architecture . As such it contains an encoder part followed by a decoder.

Like many Natural Language Processing (NLP) models, the Transformer processes words (note that for simplicity, we will consider words, however, although the inputs can be of any granularity, including subwords) as embeddings, that is, vectors of size N that represent them. When receiving a sentence, the model converts all words to embeddings, ending up with a vector of embeddings, each of size d_{model}. The full architecture can be seen in Figure 2.3 and each of its components, namely encoder and decoder layers, each composed of attention and Feed-Forwards (FFs) blocks. Each of these will be addressed in detail on the following sections.

2.1.4.A Encoder

A transformer's encoder is composed of 6 sequential smaller encoder layers, with exactly the same structure (but different parameters), meaning their outputs and inputs have the exact same format (vectors of embeddings of size d_{model}). The structure of each encoder layer has a Attention layer followed by

a common position wise Feed-Forward Neural Network (FFNN) layer (the same FF network is applied to each embedding), with both layers sharing the exact same input and output formats. There are also residual connections (He et al., 2016) in both the Attention and FFNN layers, where the input is posteriorly summed to the output, being then normalized. Joining all of this, for each encoder receiving the vector of embeddings $\mathbf{x_1}$:

$$\begin{aligned} \mathbf{x_2} &= \text{LayerNorm}(\mathbf{x_1} + \text{MultiHeadAttention}(\mathbf{x_1})) \\ \mathbf{x_3} &= \text{LayerNorm}(\mathbf{x_2} + \text{FFNN}(\mathbf{x_2})), \end{aligned} \tag{2.5}$$

where $\mathbf{x_3}$ is the output of the encoder, $\text{MultiHeadAttention}(\cdot)$ will be addressed in the following section (2.1.4.B) and $\text{LayerNorm}(\cdot)$ (Ba et al., 2016) is applied to each embedding v according, formulated as:

$$\text{LayerNorm}(v) = \gamma \frac{v - \mu}{\sigma} + \beta \qquad \mu = \frac{1}{d}\sum_{k=1}^{d} v_k \qquad \sigma^2 = \frac{1}{d}\sum_{k=1}^{d} (v_k - \mu)^2 \tag{2.6}$$

with μ, σ corresponding to the mean and standard deviation of the elements in v, $d = d_{model}$ and γ and β are hyperparameters correspondent to the scale and bias vector, respectively.

2.1.4.B Attention

Let us now focus on the attention mechanism which tries to mimic the humans ability of the same name. When reading a sentence, for instance, we tend to focus on a limited group of words, in order to get the full meaning of each word. This way we can tackle the ambiguity of the words themselves, inferring their meaning, through their context. In order to achieve this programmatically, a model needs to learn to combine words in the sentence, in order to generate contextualized embeddings. Note that attention is by no means exclusive to text processing, as it can be applied to any situation where a model may benefit from learning to dynamically extract relevant information, from specific points along a sequence of data, for instance a sequence of pixels in an image (computer vision). For the purposes of this thesis, however, we will be solely focusing on its application over sequences of tokens (words or subwords).

Scaled dot-product attention is the attention mechanism used in transformers. To understand the process, let us consider a simplified version of it, where for an embedding $\mathbf{x_1}$ we perform the dot-product wit all embeddings (self included) and get a vector of scores, one per embedding. In passing this vector through a softmax activation layer, we obtain a normalized (L1 norm) vector of weights. These can then be used to weight each embedding, when combining them. The softmax operation normalizes the weights, guaranteeing that they're all positive and add up to 1, formally:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{2.7}$$

for $i = 1, \ldots, K$, where $\mathbf{z}$ is a vector with K dimensions.

The described process appears to capture the essence of attention, however as you probably realized, it has no learnable parameters, defeating the entire purpose. Rather than using the embeddings directly, a more correct approach would be to multiply them with a trainable matrix of weights that would generate vectors of size d_k. This way, the model is able to learn interesting relations between embeddings, the so called attention. Furthermore, we can have different matrices for computing the attention layer for the embedding $\mathbf{x_i}$: matrix $\mathbf{W_Q}$ that generates the query vector $\mathbf{q_i}$ from $\mathbf{x_i}$; a matrix $\mathbf{W_K}$ to be applied to all embeddings to be scored $\mathbf{X}$, generating $\mathbf{K}$, a matrix of key vectors; finally a $\mathbf{W_V}$, to be applied to $\mathbf{X}$, generating the vectors to be combined according to the weights, $\mathbf{V}$, a matrix composed of value vectors. In practice, we can compute the attention of the entire input $\mathbf{X}$ simultaneously, computing the dot-product with $\mathbf{W_Q}$, generating $\mathbf{Q}$, a matrix where each row is the vector $\mathbf{q_i}$ correspondent with the embedding $\mathbf{x_i}$. In essence we can describe this process as:

$$\begin{aligned} \mathbf{Q} &= \mathbf{X}\mathbf{W_Q}, \\ \mathbf{K} &= \mathbf{X}\mathbf{W_K}, \\ \mathbf{V} &= \mathbf{X}\mathbf{W_V}, \\ \mathrm{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) &= \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K^T}}{\sqrt{d_k}}\right)\mathbf{V}, \end{aligned} \tag{2.8}$$

which neatly condenses the described process. The only addition here is the division by $\sqrt{d_k}$ needed for scaling purposes, since the dot-product may lead to high variance for larger values of d_k, pushing the softmax function into regions where it has extremely small gradients.

So far we've assumed $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ with the same sizes as the original embeddings, d_{model}. This assumption, however, is not fundamental and $\mathbf{W_Q}$, $\mathbf{W_K}$ and $\mathbf{W_V}$ can act as projection matrices, generating vectors of reduced dimensionality, reducing the computational cost of the attention mechanism.

Multi-head Attention makes use of this idea, by performing, in parallel, multiple attention functions, each with its own projection matrices. Each function generates a $\mathbf{V}$ matrix, with d_v dimensional vectors. These can then be concatenated and once again projected, with a forth matrix, $\mathbf{W_O}$, producing the final values, of embedding size. Intuitively, multi-head attention allows the model to learn different representation subspaces, exploring different interactions and contexts, something which single attention fails to achieve. We can formally define this process as:

$$\mathrm{MultiHeadAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{Concat}\left(\mathrm{head}_1,\ldots,\mathrm{head}_h\right)\mathbf{W_O}, \tag{2.9}$$

where

$$\mathrm{head}_i = \mathrm{Attention}\left(\mathbf{W_{Qi}},\mathbf{W_{Ki}},\mathbf{W_{Vi}}\right), \tag{2.10}$$

and h is the number of attention heads and the projections are parameter matrices $\mathbf{W_{Qi}} \in \mathbb{R}^{d_{model}\times d_k}$, $\mathbf{W_{Ki}} \in \mathbb{R}^{d_{model}\times d_k}$, $\mathbf{W_{Vi}} \in \mathbb{R}^{d_{model}\times d_v}$ and $\mathbf{W_{Oi}} \in \mathbb{R}^{hd_v\times d_{model}}$.

Following the multi-head attention layer, a fully-connected feed-forward network is applied to each embedding independently, although with the exact same model (and parameters), changing parameters only for each encoder or decoder layer. This model is pretty straightforward, consisting in only 2 linear transformations, with a ReLU activation in between. Formally:

$$\mathrm{FFN}(\mathbf{x}) = \max\left(0, \mathbf{x}\mathbf{W^1} + \mathbf{b^1}\right)\mathbf{W^2} + \mathbf{b^2}, \tag{2.11}$$

where $\mathbf{W^n}$ and $\mathbf{b^n}$ are the weight matrix and bias vector of the FF layer n, respectively. The input and output layers have dimensionality of $d_{model} = 512$ whilst the inner layer has higher dimensionality, $d_{ff} = 2048$, for the proposed model.

This completes the architecture of the inner encoders and subsequently of the entire encoder block. Contrarily to other encoder-decoder models, like the aforementioned RNNs, the transformer architecture, as it was defined so far, lacks any awareness of the order of the input sequence (like recursion), a property that is undoubtedly essential in most NLP tasks. As such, a need to introduce some sort of information relative to the position of the tokens in the sequence emerges.

Positional Encoding solves the aforementioned problem with position aware vectors that can be added to the embeddings. This way, embeddings corresponding to the same token in a sequence, have slight differences, corresponding with the absolute position they occupy. Many function can generate this positional representations, although sine and cosine are chosen, given their periodicity, guaranteeing nice variation throughout the entire distribution of dimensions:

$$\begin{aligned} \mathrm{PE}_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_{model}}\right), \\ \mathrm{PE}_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_{model}}\right), \end{aligned} \tag{2.12}$$

where pos is the token's position in the sequence and i corresponds to the i$^{\text{th}}$ vector dimension, so that odd and even dimensions get considerably different positional encodings. These encodings can also be learned by the model, although with no noticeable improvements, as well as poorer generalization for unseen cases as Vaswani et al. (2017) noted.

2.1.4.C Decoder

The decoder architecture follows the encoder for the most part, with few variations. Like other encoder-decoder models, the output sequence prediction is iterative, meaning the sequence is generated token by token. As tokens are generated, they are turned into embeddings and are fed into the decoder architecture, along with their positional encoding. Like the encoder, each of the sub-layers is followed by a residual connection , followed by a layer normalization. First it performs multi-head attention, although with the particularity that only embeddings prior to the token being predicted are considered. This is

achieved through a ”mask” in the softmax step of the attention, that nullifies all posterior positions. After this step, contrarily to the encoder, we have another layer of attention, this time however, we have ”encoder-decoder” attention, where the queries matrix $\mathbf{Q}$ is obtained from the encoder's output embeddings, whilst the keys and values matrices, $\mathbf{K}$ and $\mathbf{V}$ respectively, are obtained by the output embeddings. Besides the previous variations, the attention function remains the same. Like in the encoder, the output is then passed to a FFNN with the exact same structure (different parameters). Formally, for a vector of embeddings $\mathbf{x_1}$ we have:

$$\begin{aligned}
\mathbf{x_2} &= \mathrm{LayerNorm}(\mathbf{x_1} + \mathrm{MaskedMultiHeadAttention}(\mathbf{x_1})), \\
\mathbf{x_3} &= \mathrm{LayerNorm}(\mathbf{x_2} + \mathrm{EncoderDecoderMultiHeadAttention}(\mathbf{x_2})), \\
\mathbf{x_4} &= \mathrm{LayerNorm}(\mathbf{x_3} + \mathrm{FFNN}(\mathbf{x_3})).
\end{aligned} \tag{2.13}$$

Finally the output embedding (current position being predicted) is passed to a simple linear layer, with output dimension equal to the size of the vocabulary. This is followed by a softmax function which computes the predicted probabilities for each word in the vocabulary, corresponding to each of the dimensions. Here the most probable token is picked (or tokens for e.g. beam search), and the process is repeated with the new token, appending it to the current output sequence, until a special, end-of-sequence token is returned.

Training the transformer, and any seq2seq model for that matter, requires specific training to predict the output sequence, since at each decoding step, the transformer must correctly predict each individual token correctly. To achieve this, the model is trained for the following loss:

$$\mathcal{L} = \sum_{i=1}^{M} \log P\left(yi \mid y_{<i}, x\right), \tag{2.14}$$

where x is an input text and y, an expected output text, both as sequences of tokens $(x_0, \ldots, x_M)$ and $(y_0 \ldots, y_M)$, respectively, and $P\left(y_i|x\right)$ is the probability assigned by the model at the i-th decoding step to a token y_i, given the input x. At training time, even if the model predicts the wrong token at the $(i-1)$-th decoding step, the correct token sequence $q_{<i}$ is provided as input to the decoder, in the next step. This training scheme is called teacher forcing or maximum likelihood learning, being quite common in text generation tasks.

The transformer architecture brought a change of paradigm, exposing attention as a full-fledged neural model, along with state of the art results. It also vastly improved computational and time efficiency, by encoding the input sequence in a single iteration, whilst remaining mostly paralellizable, since the feed-forward networks are parallel, running concurrently for each sequence position, along with each attention head. The great results suggest excellent sentence comprehension, proving that attention can effectively capture the context of each embedding. Such results could be useful to other NLP tasks

that generally[2] don't follow the encoder-decoder architecture, such as classification tasks or IR. Looking back at the encoder-decoder architecture, along with the transformer model, the transformer's encoder shows great promise, effectively encoding the sequence's meaning into the returned representations. Furthermore, seeing as it maintains the embeddings structure, retrieving outputs consistent with the input embeddings, although deeply contextualized, hence able to be used by other models. In the next section we will discuss a model that uses the encoder architecture effectively, achieving state of the art performance in several NLP tasks.

2.1.5 The BERT Transformer Encoder Model

Bidirectional Encoder Representations from Transformers (BERT) is based on the Transformer, closely matching its encoder architecture, albeit with considerable scaling, doubling the number of inner encoder layers (d_{model}) to 12 (for the base version). This comes from Devlin et al. (2019)'s realization that the attention mechanism in a Transformer's encoder holds the potential for a truly bidirectional representation model. Note that up to then most representation models were unidirectional, analyzing term interactions either forwards or backwards. Even models that claimed bidirectionally, like ELMo (Peters et al., 2018), are in actuality a fusion of unidirectional models, arguably a less elegant solution.

Another key feature of the encoder design, is that it maintains the words representations throughout the whole architecture, implying that throughout the model, the representations are being "enriched" so that upper layers can leverage new knowledge, culminating in outputted representations that are extremely rich and contextualized. Such contextualized representations can be useful to other models, namely for tasks that don't follow the structure of seq2seq models.

2.1.5.A Transfer Learning

The notion of leveraging knowledge from other tasks, is called transfer learning, and is based around the idea that most NLP tasks share a common background, a general purpose knowledge. The training designed around the idea of capturing this general purpose knowledge is called pretraining, and the subsequent training on a specific task, is referred to as fine-tuning.

One of the keys for transfer learning lies on the vocabulary itself. A model relies on the knowledge gathered during pretraining, to approach the fine-tuning task. If vocabularies mismatch, this transfer is obviously affected. One of the limitations of the common approaches lies on the need to generate the vocabulary, after the training dataset, which limits the known vocabulary, to the words present in the data. If new words appear in the fine-tuning dataset, there's no elegant solution to approach them, usually resulting on the use of a special embedding, *_unk_*, reserved for all infrequent words. Even word

[2] this used to be the case, but recent studies like T5 (Raffel et al., 2020), challenged this notion, achieving comparable results in these tasks.

embedding models designed around the idea of supporting unseen terms, like FastText (Bojanowski et al., 2017), do not guarantee that the model is able to generalize the word formation process during pretraining, which injures its capacity for fine-tuning. The solution to achieve this, is to provide support for unseen words on a model basis, and not on a vocabulary basis. This is the reasoning behind the WordPiece method that BERT uses, (Wu et al., 2016), seeking to balance vocabulary size and ability to handle out-of-vocabulary words by splitting longer words into subwords (e.g., *playing* becomes *play* and *##ing*, where *##* indicates that the subword is part of another word). This way, BERT develops knowledge behind these wordpieces, being effectively capable of handling unseen cases.

Achieving bidirectional representations goes beyond the architectural design, as the common pre-training task, of predicting the following text (Language modeling prediction), given the current sequence, assumes unidirectionality. The unidirectionality guarantees that during training, the model can only rely exclusively on the known tokens, in order to make predictions. A bidirectional model, however, has access to all tokens when making a prediction, making such a task, trivial, and therefore, unfit for training.

2.1.5.B Pretraining Tasks

To counter the model's bidirectionally, the authors opted to apply a Masked Language Modeling (MLM) training objective, inspired on the Cloze task (Taylor, 1953). Specifically, for a given sequence of tokens from unlabeled text, 15% of the input tokens are selected at random, to be masked, that is, replaced with a special `[MASK]` token. During training, the model is tasked with predicting the original tokens, having to rely on the right and left contexts. Token prediction is performed as in the transformer model, by processing the corresponding embedding with a linear layer with softmax activation, outputting a distribution over the vocabulary space. Although MLM achieves the desired bidirectionally, the authors fear that `[MASK]` tokens create a mismatch, as they do not appear in the fine-tuning scenario. To mitigate the mismatch, out of the original 15%, 80% are masked, 10% are replaced with a random token and 10% remain unchanged. This way, BERT is able to transfer knowledge gained from the mask `[MASK]` token, to virtually any other.

To further improve performance on question answering and natural language inference tasks, BERT is also trained for Next Sentence Prediction (NSP). This objective takes as input two sequences of tokens, A and B, separated by a special `[SEP]` token, that can be composed of more than one natural sentences. Then given A and B, with B following A 50% of the time, BERT must predict whether B follows A or not. The positive cases are obtained by extracting consecutive sentences from a text while the negative examples are collected by pairing segments from different documents. The prediction is done by appending a special token, `[CLS]`, to the input sequence and applying a simple linear layer with softmax activation to the corresponding output embedding, performing classification over two classes, "IsNext" and "NotNext", correspondent to the assessment over the precedence of the 2 sentences.

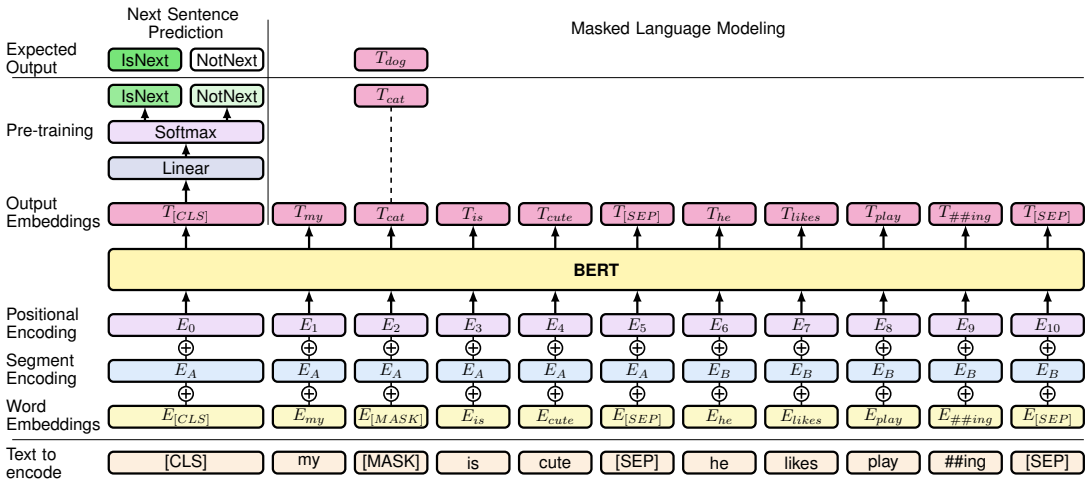


Figure 2.4: Token encoding and pe-training tasks in BERT

Both objectives, MLM and NLP can be seen in the upper portion of Figure 2.4, and constitute the pre-training process of BERT, performed on a large corpus, specifically on a collection of texts from Wikipedia and the Toronto Book Corpus (TBC).

2.1.5.C Input Processing

To accommodate these tasks, along with the fine-tuning's, BERT's input needs to undergo some pre-processing steps, before being used by the model, as seen in the lower portion of Figure 2.4. The first step consists on tokenizing the input segments with the WordPiece method, followed by adding a special token, `[CLS]`, to the start of every input sequence, that can later be used in classification tasks, including theNSP objective, paired with special `[SEP]` tokens, used to separate the segments. The resulting tokens have to pass through three different layers, consistent with the transformer's preprocessing, in order to produce the final input: The first, the word embedding layer, transforms tokens into vector representations, according to the WordPiece method; The second encodes information pertaining to theNSP task, indicating to which segment of the input segment pair every token belongs; Finally, the last adds position embeddings akin to Vaswani et al. (2017). Each layer creates a vector of dimension 768 for each token and the final result is the sum of the representations provided by each layer, generating a matrix $\mathbf{E} = [\mathbf{e_1}, \ldots, \mathbf{e_N}] \in \mathbb{R}^{N \times 768}$, where N is the length of the input sequence with all the special tokens added.

2.1.5.D Finetuning Tasks

Finally, let us address how BERT leverages the word representations for classification. BERT deeply contextualizes any tokens of the input sequence, meaning that an input token, that appears in 2 different

sequences, will be outputted with 2 considerably different representations. The contextualization suggests that BERT "injects" information referring to the sequence , on a token level, making each token representative of its original sequence. Such a property can be extremely useful for a simple classification model (with input equal to the embedding size, of 768), that classifies the sequence, based on token information, leaving 2 problems: it isn't clear which tokens can be used, since a sequence is composed of several tokens and all of them contain information regarding the sequence; there's no clear way to address classification problems that include more than one sequence. To address these problems, Devlin et al. (2019) include the aforementioned special token, `[CLS]`, that is included in the beginning of any input. The model is than trained, so that any classification problem, including ones with multiple input sequences, separated by `[SEP]` tokens, are performed on the output of this token, specifically trough a linear layer, followed by softmax activation, similar to the one used for the NSP objective.

2.1.6 BERT Variants

Similarly to transformer, BERT struck massive interest int the NLP community, originating several models inspired in its architecture and training schemes. Here we will lightly describe a few examples, representative of the range of approaches.

2.1.6.A RoBERTa

RoBERTa (Liu et al., 2019b) aimed at improving BERT's pre-training, specifically with further tuning of the hyperparameters and the training set size. In this study, Liu et al. (2019b) found that BERT was considerably undertrained and propose some modifications: longer training with bigger batches, to make use of multiple GPUs and consequently, more training data, to accommodate the considerable increase of training steps; removal of theNSP objective, found to offer less improvements than further training with the revised MLM objective; pre-training over bigger sequences, to better match the fine-tuning scenarios. With these changes, RoBERTa is able to improve on BERT's results in a vast amount of tasks, namely on GLUE (Wang et al., 2018) and SQuAD (Rajpurkar et al., 2018, 2016).

2.1.6.B ELECTRA

ELECTRA (Clark et al., 2020) also improved on BERT's pre-training. Contrarily to RoBERTa, that compensates BERT's undertraining with more training, ELECTRA aims at maintaining the same steps, but improving the quality of the pretraining task. To achieve this, Clark et al. (2020) proposed a new task, *replaced token detection*, where rather than trying to predict the masked token, based on its surroundings, the model is tasked with distinguishing real input tokens from plausible but synthetically generated replacements, produced by a smaller MLM model (like BERT). This way, all input tokens train the model,

rather than just the masked token, explaining the efficiency of ELECTRA. This approach is so effective, that it performs comparably to RoBERTa, in the GLUE benchmark, at less than ¼of the computational cost. For the smaller version of the model, the authors trained on the same pretraining data as BERT, however, for the bigger models, the authors used the dataset used for XLNet (Yang et al., 2020), which extends BERT pretraining dataset, with text from the web.

2.1.6.C ALBERT

ALBERT (Lan et al., 2019) aimed to improve the scalability of BERT, by considerably reducing the number of parameters. In order to accomplish this, Lan et al. (2019) propose two techniques: factorizing embedding parameterization, specifically in 2 smaller matrices, effectively separating the embedding size from the transformer hidden size, allowing to scale the model without scaling the embeddings; enforcing cross-layer parameter sharing, where both the attention and FFN's parameters, are shared across every layer. The combinations of these techniques results in a reduction of $18\times$ the number of parameters of the comparable BERT-large, whilst resulting on a roughly $1.7\times$ faster training. Another contribution of this work was the introduction of a new task, *sentence order prediction* (SOP), similar to BERT'sNSP, but the negative examples are picked just like the positive ones, but with the order swapped. This way, the model is forced to learn sentence coherence, whereas inNSP, topic prediction is a conflicting signal, originated from random picking. ALBERT was able to achieve new SOTA performance at the time, in GLUE, SQUAD and RACE (Lai et al., 2017).

2.1.6.D mBERT

mBERT (Devlin, 2018) simply aims to adapt BERT to languages other than English. BERT's pretraining consisted exclusively of English text, making the model ineffective at leveraging this knowledge, for finetuning tasks on other idioms. One possibility would be to pretrain it on language specific data, this however is unpractical, given the sheer amount of idioms, and the costs associated with pretraining. A more general solution is to produce a multilingual BERT that has been pre-trained on a mixture of many languages, learning knowledge shared between idioms and leveraging it for finetuning on tasks of any language contained in the mixture. The multilingual approach has also been used for RoBERTa, yielding XLM-RoBERTa (Conneau et al., 2020).

2.1.7 The T5 Model for Multi-Task Learning

T5 (Raffel et al., 2020) is a variant of the original Transformer, that aims to unify every NLP task in a text-to-text framework. In practice, this means that for any task, the model receives an input sequence of tokens, and must output a specific sequence of tokens in return. The text-to-text adaptation is trivial for

naturally seq2seq tasks, like translation or summarization tasks, but for tasks like classification, there's no clear approach. For these cases , the authors suggested training the model, to predict specific tokens, correspondent with each class. For example, for a task of sentiment analysis, where a model is trained to predict whether a sentence reveals a positive or negative sentiment, the model outputs either the token "positive" or "negative", accordingly. Note that the model is free to predict any other token, unlike typical approaches for classification, where the architectural design enforces a one-to-one correspondence with the output classes (e.g., BERT). Despite this, the authors find that implicit enforcing through training is sufficient, and the model never outputted unexpected tokens during testing.

The text-to-text framework also adds a task-dependent prefix to the input sequence. This is meant to "signal" the model, of the task being performed. For example, a translation task, from English to German, originally simply composed of an input text to be translated, `[input]`, becomes:

$$\text{translate English to German: } \texttt{[input]}, \tag{2.15}$$

with "translate English to German:" being the literal string. This formulation enables the model to be trained on a multitude of tasks, simultaneously, a training technique known as multi-task learning (MTL) (Caruana, 1997), firstly adopted to NLP by Collobert and Weston (2008). MTL is meant to to teach the model general-purpose knowledge, shared by most tasks, similarly to a weak supervision objective, in pre-training. Despite this, the authors also use a weak-supervision pre-training objective, specifically an adaptation of BERT's MLM to seq2seq, trained alongside the other tasks.

T5 was able to achieve state of the art (SOTA) (at the time) performance in a multitude of tasks, including classification tasks, beating its BERT counterparts. This, alongside the release of the pre-trained models, by the authors, motivated lots of research, namely in expanding this unified approach, to untested NLP problems. These includes ranking, cases that will be discussed further in Chapter 3.

Similarly to BERT, T5 has also explored multilingual training, yielding mT5 (Xue et al., 2020).

2.2 Text Representation Models

Raw text format is often unfit for NLP approaches, with most machine learning models requiring generalized data formats, namely vectors of features. The process of representing text as a vector is in itself an open problem. In this section we will cover different approaches to create text representations.

2.2.1 Sparse Models

When presented with the task of vectorizing a passage of text, perhaps the most intuitive approach is to consider features at a word level. In essence, each word in a vocabulary Γ, with size n, is a dimension

and every piece of text can be represented by such a vector, a representation also known as Bag-of-Words (BoW), since any order or positional information in the text is lost. This is what confers the sparsity of the representation, as most documents lack the majority of words in the vocabulary, corresponding to values of zero. Formally, a document d_i belonging to a collection D, can be represented by a vector $\mathbf{v_i}$ = $\{w_{1,i}, w_{2,i}, ..., w_{n,i}\}$ where $w_{i,j}$ corresponds to the feature weight for word w_i from Γ, for a document d_j.

Intuitively, the weight of a feature should represent it's relevance in a document, with a higher weights corresponding to higher relevance. Judging a word's semantic relevance can be challenging, even for a human, so an appropriate estimator is preferred. Statistical relevance can be a cheap, decent approximation, where a word's importance is based purely on how frequently it is used throughout the collection. The simplest case we may consider is a binary valued vector, correspondent to word incidences. Here, the weight $w_{i,j}$ is 1 if the word w_i is present in d_j, else 0. It's easy to see why this approach is an insufficient representation for most use cases. Not only it assumes every word has the same relevance, it completely neglects word frequency in a document, so a word appearing 10 times will have the same importance as a word with a single presence.

2.2.1.A TF

Term Frequency (TF) roughly solves some of the aforementioned problems, measuring relevance based on how frequent a term is inside a document. The weight $w_{i,j}$ is simply the number of occurrences of the term w_i in document d_j, although in practice this value is still passed to a scaling function, usually the logarithm function, in order to gradually reduce the importance of further repetitions. Formally:

$$\mathrm{TF}(w, d) = \log(f_{w,d} + 1), \tag{2.16}$$

where w and d correspond to a word and a document respectively, and $f_{w,d}$ the number of occurrences (frequency) of w in d. Although a definitive improvement over the previous binary approach, it still suffers from some problems, most noticeably frequent irrelevant words, such as sentence connectors, that appear frequently in most documents, whilst posing no contribution to a documents relevance. This problem could be tackled with the removal of stop-words, a common NLP approach that aims to remove irrelevant, overly frequent words from the documents in the collection. Such a solution has its own problems however, as these lists of words are often domain specific and expensive to generate. Removing stop-words might also lead to overcompensation, by removing terms that can have an impact, even if minimal.

2.2.1.B IDF

Inverse Document Frequency (IDF) presents itself as an alternative to removing stop-words, by scoring terms based on how frequent they are throughout the collection. The bigger the set of documents in which a term appears, the lesser its importance, hence the ”inverse”. The intuition here it that if a word appears in most documents, it's presence does not add much information. By contrast, a term that appears in a very specific group of documents, is likely to be extremely relevant in that context. To achieve this behavior for a word w_i we can simply divide the total number of documents —D— by the number of documents containing the term. The logarithm function is applied to properly scale the score. Formally:

$$\mathrm{IDF}(w, D) = \log \frac{|D|}{|\{d \in D : w \in d\}|}, \tag{2.17}$$

where w and D correspond to a word and the set of documents in a corpus respectively.

2.2.1.C TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) (Equation 2.18) applies both previous notions, in order to achieve a better relevance score. This way, the more frequent and exclusive a document's terms are, the more relevant they are to a query containing them. The TF-IDF score for a word $w_{i,j}$ in a document d_j is then simply the product of both frequency scores. Formally:

$$\mathrm{TFIDF}(w, d) = \mathrm{TF}(w, d)\mathrm{IDF}(w, D), \tag{2.18}$$

where w and d correspond to a word and a document respectively. Some variations may apply depending on the collection's properties. The most common approach is to include some sort of document length normalization. The original TF-IDF does not consider a document's length, despite longer documents having a natural tendency to having higher term frequencies. By applying normalization, differently lengthened documents can be compared more equally, generally leading to better results.

2.2.1.D BM25

Best Match 25$^{\text{th}}$ iteration (BM25) (Crestani et al., 1998; Robertson et al., 1995; Robertson and Zaragoza, 2009), could be considered a lengthy variation of previous model, although it proposes enough changes to be considered its own model, with its own variations. The origin behind this model is deeply rooted in probabilistic IR, offering concrete statistical significance behind each element of the scoring function, contrasting with the more intuitive nature behind TF-IDF. Like the previous model, BM25 can also be split into an IDF part, paired with a TF part. The IDF section is a common variation on the original,

where 1 is added to prevent negative log values, consistent with a probabilistic approach (no negative probabilities). The TF part is where we find most of the new functionality, which can be condensed into 2 main ideas: the original TF score grows infinitely, which isn't really in accordance with reality, where after a few repetitions, the statistical value added by another one is negligible, and also that the variance in document length plays a big role in the performance of the model and therefore need further adjusting. For these, 2 hyper-parameters are introduced: k_1 responsible for the first, controls the term frequency at which the score function stabilizes, producing the same scores for higher frequencies and b, responsible for the second, that controls the weight of the normalization with the average document size, avgdl, where 0 corresponds to no normalization and 1 to a full length normalization. Formally:

$$\begin{aligned} \mathrm{BM25}(w,d) &= \mathrm{IDF}(w,d)\frac{f_{w,d}(k_1+1)}{f_{w,d}+k_1\left(1-b+b\frac{|D|}{\mathrm{avgdl}}\right)}, \\ \mathrm{IDF}(w,d) &= \log\left(\frac{|D|-n_w+0.5}{n_w+0.5}+1\right), \\ n_w &= |\{d\in D : w\in d\}|. \end{aligned} \tag{2.19}$$

where w and d correspond to a word and a document respectively, and $f_{w,d}$ the number of occurrences (frequency) of w in d. The BM25 procedure marks the peak of frequency-based indexing approaches, still seeing much use in the current scene, given its unmatched efficiency and reliable performance. Interestingly, one of its most prevalent uses, is its use as a filtering step, considerably reducing the search space for costlier and more powerful methods, namely transformer-based. This interaction will be addressed in Chapter 3.

2.2.2 Dense Models and Word Embeddings

So far we've looked at text representation models. Although effective in less demanding problems, harder cases often require finer granularity: Rather than representing a full text trough a vector, represent each word as a vector, and a text as a sequence of said vectors. This way models can capture relations on a word level, allowing interpretations at a semantic and syntactic level, relying on the words positions and contexts.

When translating words into vectors, a naïve approach, perhaps motivated by the sparse models, would be to use one-hot encoding: binary vectors of size V, for a vocabulary with V words, where each position corresponds to a word in the vocabulary, and is set to 0, with the exception of the position correspondent with the represented word, which is set to 1. The inadequacy of this approach is immediate: not only is the the high dimensionality mostly unnecessary, it fails to exploit the properties of the vector space in any meaningful way, since proper vectorial word representation should capture a word's meaning. This would imply that related words are packed closely in the vector space whilst unrelated words are distant. Furthermore, this condition guarantees that each vector dimension corresponds to a certain

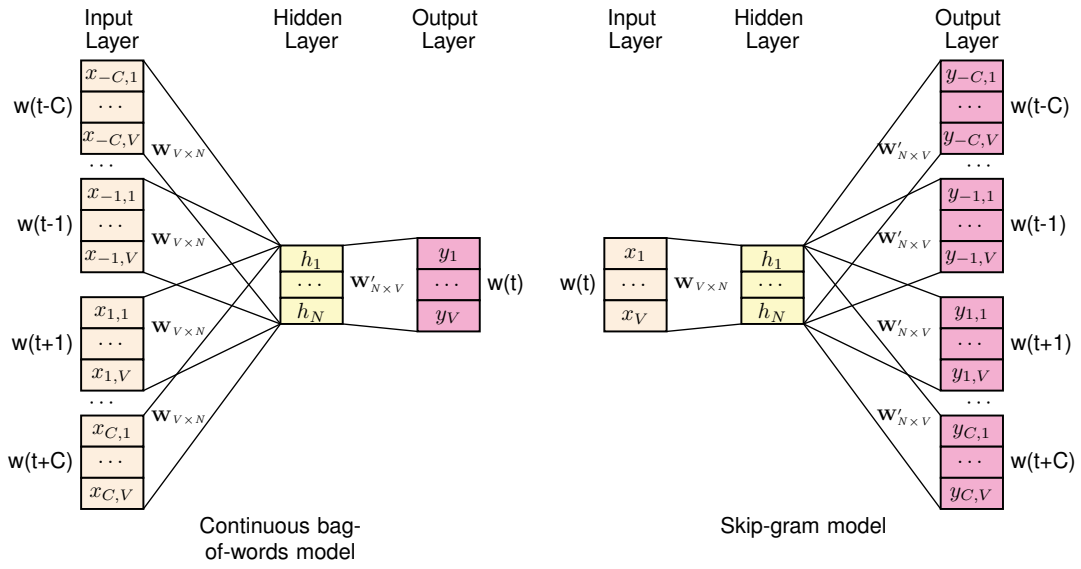


Figure 2.5: Architectures of both methods used in the word2vec model.

word property or context, and that words may closely relate in some dimensions whilst differing in others. An intuitive way to think of this is to consider a specific example: Given the vector representations for the word *king*, $\mathbf{e_{king}}$, *man*, $\mathbf{e_{man}}$, and *woman*, $\mathbf{e_{woman}}$, we'd have that $\mathbf{e_{king}} - \mathbf{e_{man}} + \mathbf{e_{woman}} = \mathbf{e_{queen}}$, where $\mathbf{e_{queen}}$ is the vector representation of *queen*. While it is hard for us to define the set of contexts that *man* and *king* share, it is intuitive that if we remove them from *king* and add the the context of the word *woman*, we obtain *queen*. In this example it's clear how the vectors capture the notion of gender, although they can capture more complex and less intuitive notions. A vectorial word representation is also often called a word embedding and will be term used from now on.

The contexts of a word are hard to define, however we can deduce them from it's neighbouring words, as they share their contexts (Harris, 1954). By extracting words that occur together in vast amounts of text, we can feed this information into a neural model, capable of extracting word embeddings from it. Different models have been proposed for this task:

2.2.2.A Word2Vec

Word2Vec (Mikolov et al., 2013) is perhaps the most well known and motivated massive improvements in neural approaches for NLP problems, proving to be an effective way to translate words into vectors. Mikolov et al. (2013) proposed 2 methods to generate embeddings, namely the Continuous Bag-of-Words (CBoW) and the Skip-Gram approaches.

The CBoW model, as seen on the left portion of Figure 2.5, is composed of a simple feed-forward network with a single hidden layer. The input and output layers share their format, vectors $\mathbf{x}$ and $\mathbf{y}$ respectively, of size V, where V is the number of words in the vocabulary. The hidden layer, $\mathbf{h}$, has size

N, with N correspondent to the desired number of embedding dimensions. This way, the model inputs and outputs one-hot encoded vectors and, ideally, the hidden layer corresponds to the embedding of the outputted word. To achieve this, the model is trained to predict a word $\mathbf{w}(t)$, given its context $[\mathbf{w}(t-C), \ldots, \mathbf{w}(t+C)]$. The prediction is repeated for every word in the surroundings, C antecedent words and C subsequent words, producing $2 \times C$ vectors: $\mathbf{x_{-C}}, \ldots, \mathbf{x_C}$. This can be optimized by performing the first computation (up to the hidden layer) for each word, averaging the hidden layer output and finally resuming the last part of the model for this vector, $\mathbf{h}$. The output vector is then activated trough a softmax, outputting a distribution of probabilities over the vocabulary and the loss is calculated over the one-hot encoded vector correspondent with the expected word. After training, we have a model capable of condensing the information from multiple contexts in a vector ($\mathbf{h}$), and predict the word that shares them from it. A calculated consequence of using one-hot encoded vectors is that when calculating $\mathbf{h_c}$, from $\mathbf{x_c}$, the one-hot encoded vector for a word c, we get:

$$\mathbf{h_c} = \mathbf{x_c}\mathbf{W} = \mathbf{W_c}, \tag{2.20}$$

where $\mathbf{W}$ corresponds to weight matrix used from the input layer to the hidden layer, with dimensions $V \times N$ and $\mathbf{W_c}$, the row of $\mathbf{W}$ matching the position of c, corresponding to a vector of size N. Similarly, the same is true for the output side, since the predicted word is also one-hot encoded, but for the columns of $\mathbf{W}'$, the weight matrix from the hidden layer to the output layer. This essentially means that both matrices contain embeddings for the entire vocabulary, with the ones present in the input matrix being referred to as input vectors and the ones from the output matrix, as output vectors (not to be mistaken with the actual inputs and outputs of the model). In the first one, words that appear in the same context (of a target word) are closer, whilst in the output vector, word distances reflect the similarity between their contexts (note that the distances considered here are based on the cosine similarity). Although both definitions appear very similar, in practice the difference between them can be impactful, depending heavily on the use case.

The skip-gram model shares most of the previous features, but with the inverse model: rather then predicting words based on their context, predicting context based on the word. The structure is pretty much the inverse, with the notable exception of the handling of multiple contexts. Whilst CBOW computes a vector for each context and averages them, the skip-gram model computes the same context output vector, but calculates the loss over all the expected context words (in a sense averaging the loss). The properties mentioned above still apply but the roles of the embedding vectors (input and output) are reversed. In the Figure 2.5 we can see both architectures, along with the formats of each layer.

3

Related Work

Contents

This section explores some of the different approaches that have been used for solving information retrieval problems. In particular, Section 3.1 starts with an historical overview over information retrieval, then Sections 3.2 and 3.3 deepens related work with transformer based retrieval.

3.1 Ranking Before Transformers

Before discussing related work exploring transformer models, we will lightly discuss the historical approaches that motivated them, namely improvements on the BoW models and the generalized deep-learning ranking architectures.

3.1.1 Frequency-Based Indexing and BM25

Although many approaches for IR (Harman, 2019) precede BM25, none have continuously stand the test of time quite like it, as it is still used as a starting point for many text ranking approaches (see Section 3.2). The reason for this lies in its simplicity, maintaining relatively decent performance in most use cases, and the fastest inference times. But BM25's effectiveness is often insufficient for most IR tasks, given its reliance on exact matching, requiring that the terms used in a query, to match those of the relevant documents.

That vocabulary mismatch problem (Furnas et al., 1987) is the biggest limitation of BM25 and one of the biggest challenges in IR in general. Some approaches have surfaced, attempting to combat this problem trough query expansion, enriching the queries representations trough lexical-semantic relations (Voorhees, 1994) or co-occurring relevant terms (Xu and Croft, 2000). Other approaches use document expansion, with similar strategies, but adapted towards documents representations (Salton and Buckley, 1988). Although these approaches achieved some performance gains, they still rely on exact matching, which fails to capture semantic or context level similarities.

3.1.2 Deep Learning-Based Ranking

The unsupervised approaches discussed above had clear limits, so deep-learning models were the obvious next step, in order to achieve greater effectiveness. Here we'll be taking a brief overview on models that serve as precursors for architectural details that will be later explored in transformer based models, on the next section. For a more comprehensive analysis we suggest the surveys of Onal et al. (2018), Mitra and Craswell (2018) and Xu et al. (2020).

Most pre-BERT neural ranking models generally follow one of 2 designs. On the one hand, **representation-based** models replicate the ideas of inverted indexes, by independently learning semantic vector representations of queries and documents, later compared trough similarity functions, such as

cosine relevance, in order to assess relevance. On the other hand, **interaction-based** models compare query and document representations directly, usually trough a similarity matrix, learning to generate a relevance score from this interaction. In the next sections, we will see how these 2 designs were adapted in order to leverage transformer-based models.

3.2 Interaction-Based Transformer Architectures

Perhaps the simplest and most straightforward way to leverage transformers for ranking, is to turn ranking into a relevance classification problem. This relevance classification task is simply judging whether a document is relevant for a given query, or not, and its probability can then be used as a score, to rank each document. Leveraging classification for ranking is not immediate and has practical limitations. However, it has been thoroughly studied under the *Probability Ranking Principle* since Robertson (1977), and it has been successfully implemented in transformer based models, as we'll see in this section.

3.2.1 Simple Relevance Classification with MonoBERT

Relevance classification is formally described as:

$$P(\text{Relevant} = 1 | d_i, q), \tag{3.1}$$

where d_i corresponds to the a candidate text i and q, to a query. Given this formulation, ranking with BERT is seemingly trivial, sharing the same conditions as other tested use cases, like some of the proposed on Devlin et al. (2019). Unlike those cases, however, relevance classification for the purposes of ranking involves considerably more computations, since each query is classified over the entire collection of documents, in multiple inferences. This makes a brute-force approach unviable, as the inference time grows linearly with the collection size. Nogueira and Cho (2019) addresses this problem by proposing a multi-stage architecture, that starts with the full set of text candidates (the full data set), and iteratively filters this set with different models, culminating in a final ranking, composed of the top-k documents. Specifically, in their work, they propose the simplest of this architectures: the first stage is performed using keyword search, namely with BM25, retrieving a ranked top-k ($k = 1000$), an approach often called candidate generation, initial retrieval or first-stage retrieval; followed by the earlier described ranking with relevance classification, performed only over this top-k, essentially *reranking* them. The described approach is consistent with *retrieval-and-rerank*, a concept that can be traced back all the way to Simmons (1965) and has been widely used.

Nogueira and Cho (2019)'s use of BERT is very consistent with one of the tested approaches for classification in Devlin et al. (2019) (specifically for 2 inputs), with the input as follows:

$$[\texttt{[CLS]}, q, \texttt{[SEP]}, d_i, \texttt{[SEP]}], \tag{3.2}$$

for a query q and a candidate text d_i, both tokenized as a sequence of BERT tokens. Finally, The outputted `[CLS]` token is fed to a fully connected FFNN, that outputs the relevance score s_i. Formally, this corresponds to:

$$P(\text{Relevant} = 1 | d_i, q) = s_i \triangleq \text{softmax}\left(T_{\texttt{[CLS]}}\mathbf{W} + \mathbf{b}\right)_1, \tag{3.3}$$

with $T_{\texttt{[CLS]}} \in \mathbb{R}^D$ as the output embedding of the `[CLS]` token, $\mathbf{W} \in \mathbb{R}^{D\times 2}$ as the FFN's weight matrix, $\mathbf{b} \in \mathbb{R}^2$ as the FFNN's bias and $\text{softmax}(\cdot)_i$ as the i-th element of the softmax output, for D as the model embedding dimension. Note that the two dimensions of the output, correspond to the classes "relevant" and "non-relevant", with the score of the first, corresponding with the actual document relevance score. Although unnamed at first, the authors adopted the name monoBERT for this approach, in a later publication (Nogueira et al., 2019b), with *mono* alleging to the inference of candidate documents with BERT, one at a time. Humeau et al. (2019) generalized this kind of approach as a "cross-encoder", alleging to the cross-attention between query and document (see Section 3.3.1.A), being the common nomenclature, in most literature. The model's architecture can be viewed on Figure 3.1, for an input sequence of tokens composed of a query $(\text{In}_q\ 1, \ldots, \text{In}_q\ N_q)$, and a document $(\text{In}_d\ 1, \ldots, \text{In}_d\ N_d)$, with N_q and N_d tokens, respectively. This sequence is fed to a transformer model, whose outputs pass trough an agreggator function to extract a representative embedding, usually correspondent with the `[CLS]` token, in the case of BERT. The representative embedding then passes through a dimensionality reduction function, such as a simple linear layer, outputting the ranking score.

In the case of MonoBERT, it is fine-tuned for the task of relevance classification, using the standart cross-entropy loss function:

$$\mathcal{L} = -\sum_{j\in J_{pos}} \log(s_j) - \sum_{j\in J_{neg}} \log(1 - s_j), \tag{3.4}$$

with J_{pos} and J_{neg} correspondent to the sets of relevant and irrelevant indexes, respectively, retrieved in the initial stage. The reasoning behind picking the training examples from the early retrieval, instead of random picking, for instance, is that the first stage has subtle bias. This affects the distribution of the retrieved documents, differing from a normal distribution, and this way, BERT is introduced to it right from the training. The loss further proves the slight inadequacy of this approach for ranking, as each document is considered independently. This means that improvements of the model over this metric,

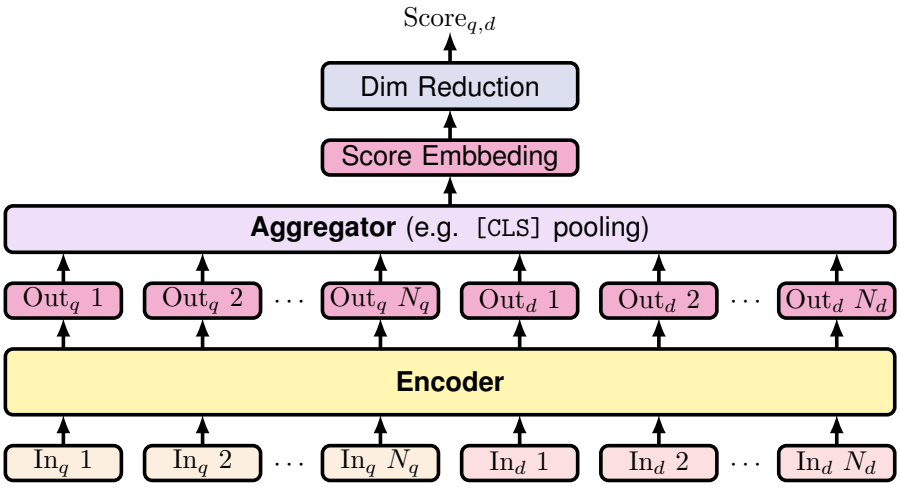


Figure 3.1: Graphical representation of a cross-encoder.

don't necessarily translate into improvements over ranking metrics, a phenomenon first reported by Morgan et al. (2004), under the name "metric divergence". Despite this, monoBERT was not only able to effectively rank passages, it did so by beating the previous SOTA by a considerable margin.

3.2.2 MonoBERT Extensions

MonoBERT is arguably one of the simplest approaches for ranking with BERT, prompting research on possible extensions. Most of them, focus on improving the training, as the divergences between the train and testing cases, seem to limit ranking performance. One of the keys for improving it, lies precisely on the selection of the negative training examples, having a direct impact in monoBERT's ability to rank. As per the formulation, this is in fact the score s_i, that ranks the documents, and as such, has a direct impact in the ranking performance, hence the interest. An obvious method for selecting these pairs, is to consider the choice itself as a learning problem. In Zhang et al. (2020), this is achieved trough Reinforcement Learning[1], guided by a weaker model, directly trained for ranking. With a similar idea, MacAvaney et al. (2020) proposed Curriculum learning (Bengio et al., 2009) for BERT, where training is guided with progressively more difficult examples, similarly to how humans learn. The difficulty of the examples is assessed based on weaker model's ability to properly rank the example, specifically BM25, in this case. Both prior models achieved significant improvements over the original proposition by Nogueira and Cho (2019), proving proper training guidance is crucial for this approach.

Recently, the introduction of TF-Ranking, by Pasumarthi et al. (2019), allowed the training of most ML models for ranking, with ranking-specific loss functions, that consider inferences on multiple documents, unlike Equation 3.4. This is precisely one of the drawbacks in the original monoBERT approach (metric

[1] Reinforcement learning extends past the reach of this thesis, although I can refer interested readers to a survey on the matter by Kaelbling et al. (1996).

divergence), motivating Han et al. (2020) to propose TFR-BERT, a monoBERT implementation that uses listwise softmax loss, over a list of 12 documents, with a single relevant one. In their work, they also train other 2 models, from the same training examples, one under the original loss (Equation 3.4), and a second with pairwise logistic loss (a loss that takes into consideration inferences on 2 different documents, simultaneously). This allows them to conclude that any performance gains are directly attributed to the listwise loss, proving that metric divergence does play a rule in monoBERT's performance.

3.2.3 Document Ranking

So far we've been purposefully avoiding an obvious limitation in the cross-encoder architecture: transformers are bag-of-words models, having clear limits on the number of tokens an input sequence has. More importantly, BERT's all-to-all attention introduces spacial and temporal quadratic complexity, both in space and time (Kitaev et al., 2020), severely harming attempts to extend the standard limit of 512 tokens. Although manageable in most NLP tasks, for ranking this can become a problem, since there are several applications that require ranking larger documents, far beyond BERT's limit. In the cases reported in the last section this wasn't a problem, as the use cases were "handpicked" around this limitation (e.g. passage ranking)

Several approaches have surfaced attempting to address this problem, not only ranking-specific but also on a architectural level, with noticeable success. For the purposes of this thesis, however, passage ranking is sufficient and as such, document ranking approaches fall outside of the scope. If the reader has interest in this theme, I suggest Yates et al. (2021)' survey, offering an insightful overview on the range of techniques used to tackle this problem. Note that in the next sections, unless specifically mentioned, we will address passages as documents, with respect to the common IR nomenclature, even thought all models addressed suffer from this limitation.

3.2.4 Multi-Stage Rerankers

In Section 3.2.1 we have looked at the simplest approach for BERT-based multistage-ranking, a retrieve-and-rerank setup. However, this can easily be extended to accommodate other stages, with different approaches. Specifically, we can add a model more powerful than monoBERT, that reranks a top-k', out of the top-k returned by monoBERT, with $k' < k$, further improving ranking. This can be repeated for a new model, indefinitely, a design also know as reranking pipelines, ranking cascades or *telescoping*.

3.2.5 Beyond BERT

So far we have only taken a look at BERT-based approaches, and for good reason, since they're by far the most explored. This is mostly due to BERT's rapid growth, stapling it as "the" transformer for

classification, and, by extension, ranking. There are, however, transformer-based alternatives that have been seeing some attention, which could be key in addressing some of BERT's limitations (see Section 2.1.6). Here, we'll briefly take a look at some of these, and I once again recommend Yates et al. (2021) for readers interested in further reading.

3.2.5.A Pretrained BERT Variants

Unsurprisingly, an obvious approach is to use the same discussed approaches, but with BERT variants, as they're purposefully designed as BERT replacements. Although this is underwhelming, as far going beyond BERT, there is merit in these approaches, proving how diversified and inclusive the transformers' ecosystem has become. This empowers anyone to easily extend existing approaches, in pursue of better efficiency, performance or even both. And this improvements are nearly free, as they recycle most of the work put not only on the ranking approach, but on the BERT variant as well (this is also extendable to any NLP problem, not ranking exclusively). Not only that, as new and improved BERT variants surface, they can easily be integrated into prior approaches, particularly important, given the current interest in further developing and improving transformer architectures.

Despite this, ranking with BERT variants is still somewhat unexplored, mostly seeing use in ensembles, for high-end performance benchmarking (e.g. the top submissions in MS MARCO (Nguyen et al., 2016)). Note that a good performance on relevance classification does not translate into good ranking performance. By extension, better performance in other common NLP tasks, doesn't necessarily entail better ranking performance. Specifically, in Section 2.1.6, we've seen how most BERT variants were developed around improving performance in a plethora of NLP tasks (e.g GLUE (Wang et al., 2018)), but crucially, not ranking. This is for good reason, as not only is ranking an almost isolated NLP task, but most importantly, it's extremely expensive to test, making it unpractical for large scale hyper-parametrizations. What this means for ranking, is that most assumptions that could be made, regarding the effectiveness of BERT variants, prior to testing, are probable at best. A great example of this, is provided in (Yates et al., 2021), where RoBERTa is tested alongside monoBERT, in the exact same conditions, finding that RoBERTa actually underperformed BERT, despite more pretraining and better results in other NLP tasks.

3.2.5.B Distillation

A number of approaches have surfaced, that explore knowledge distillation for multi-stage ranking. Knowledge distillation (Ba and Caruana, 2013; Hinton et al., 2015) refers to a set of techniques, aimed at distilling the knowledge of a larger model, the *teacher*, by having a smaller model, the *student*, learning directly from it, allowing the *student* to retain most of the performance, but at higher efficiency. This idea can be applied to any model, but most importantly, to BERT. An obvious approach, is to distill a

pre-trained BERT, to a smaller model, with a simpler architecture (Liu et al., 2019a; Tang et al., 2019) or even a smaller BERT model (Sanh et al., 2019; Sun et al., 2019a). These can be thought of as BERT variants, and be adapted for ranking, just like the previously discussed models (Section 3.2.5.A).

An arguably more interesting approach surfaces, however, when we consider an already fine-tuned model, for relevance classification. Here, rather than fine-tuning a distilled model, the *student* could learn directly from monoBERT, learning relevance classification directly, which could help a smaller model learn task specific knowledge that otherwise would not have been able. Gao et al. (2020a) goes even further, proposing a third approach, that mixes the previous two: first, just like the first one, distil a pre-trained BERT model, to a smaller model, of the same type of architecture; secondly, rather than fine-tuning the distilled model, for relevance classification, train it, with an already fine-tuned, larger model (just like normal knowledge distillation, but with a better initialization of the *student*'s weights). In their study, Gao et al. (2020a) test the 3 approaches, for reranking in MS MARCO and TREC 2019 DL (Craswell et al., 2020), discovering that whilst direct distillation (random initialization) from monoBERT does lead to a noticeable performance drop, both the other considered approaches perform on par with the *teacher* model, even outperforming it in some cases, while having half the number of layers (6) and half the inference time.

Li et al. (2020) also investigated the previous two best approaches, but for document ranking, finding in distillation, an effective tool to improve efficiency, with limited performance degradation. Both studies show positive results for ranker distillation, providing a competitive trade-off of efficiency and performance, definitely deserving of further research.

3.2.5.C Ranking with Sequence-to-Sequence Models with monoT5

Inspired by T5's versatility, successfully adapting to a plethora of NLP tasks, Nogueira et al. (2020) explore T5's ability to rank documents. Even thought other possible formulations exist, the authors decided to treat the problem as a straightforward relevance classification task, such as monoBERT, since T5 has already shown its potential for other classification tasks. Specifically, given a query `[q]` and a document `[d]`, to classify, T5 receives the following input template:

$$\text{Query: } \texttt{[q]} \text{ Document: } \texttt{[d]} \text{ Relevant:}, \tag{3.5}$$

with "Query:", "Document:" and "Query:" as literal strings. The model is then fine-tuned to produce the tokens "true" or "false" (literally), corresponding to a prediction of relevance or irrelevance, respectively.

Note that unlike monoBERT, that directly scores the document at the classification task (prediction probability), in T5, the model simply predicts a token correspondent with the predicted class, leaving many possible formulations for retrieving a score. One obvious approach is to leverage T5's token prediction scores (like the original transformer, a each decoding step, the model outputs a probability

distribution, over the entire vocabulary), since *true* and *false* are single tokens, requiring a single decoding step to be computed. After some trial and error, the authors find that computing the softmax scores over the *true* and *false* tokens' probabilities, and using the outputted score correspondent with the *true* token, as a ranking score, yielded the best results. The model is trained just like any other task in T5, maximum likelihood learning (Equation 2.14). Given the resemblance with monoBERT's architecture, the authors coined the name monoT5.

Tested in MS MARCO passage ranking, monoT5 achieved considerable improvements over the scores from its BERT counterparts, despite sharing similar architectures. Perhaps even more impressive, is that by using the same models, fine-tuned in MS MARCO, and testing them on document ranking datasets, without further fine-tuning, monoT5 was able to beat most models, designed around document retrieval. It is important to note, however, that the T5 model that achieved the best scores, had around 3 billion parameters, being considerably bigger than the other models. Despite that, achieving such effectiveness without training on any of the test collections, an approach known as zero-shot learning, is a testament of T5's ability to leverage general purpose knowledge, for unseen tasks.

3.2.6 Document Preprocessing Techniques

In most ranking scenarios, the text corpus is mostly static and any updates are usually planed. Such a setting allows the consideration of techniques that leverage some offline preprocessing of the document collection, prior to the online setting. Here we'll tackle approaches that aim to improve document representations, either directly, improving the indices of first retrieval methods, or indirectly, through target corpus pretraining, improving the representations learned by the transformer models.

3.2.6.A Document Expansion

In Section 3.1.1 we have discussed how document expansion can be an effective tool at combating BM25's problems with exact matching. With the rise of transformer models, new opportunities to tackle this approach have been considered.

The first successful application of this idea was proposed by Nogueira et al. (2019c), introducing a new technique, doc2query. In essence, the idea is to train a seq2seq model, so that given a document, it is able to produce synthetic queries for which the document might be relevant. To train it, the model simply leverages standard relevance judgements, pairs of the form (query, relevant document), already used to train ranking models. The trained model is then used in each document of the corpus, in order to produce multiple synthetic queries related to it. These are then appended to the original document, forming an "expanded document". The new expanded corpus, can now be leveraged by common bag-of-words models, namely BM25, producing richer indices.

The idea behind the expanded documents, is that synthetic queries not only introduce new related terms, previously uncaptured by the original documents and directly tackling the problems of exact matching, but also reweights the importance of existing terms, by copying them in the produced queries. To evaluate both contributions, the authors experiment expansions with only the new terms or only the existing ones, finding that neither match the original expansion and suggesting that both types of terms are complementary.

Besides doc2query, that used the original transformer, Nogueira et al. (2019a) revisited the idea of document expansion through query generation, but using T5 as the expansion model, motivated by its pretraining strategy, and positive results on a wide range of tasks. This new technique, docTTTTTquery, led to a considerable improvement in performance, once again reiterating the importance of pretraining. Both models where trained with maximum likelihood learning (Equation 2.14), for predicting related queries, given the corresponding documents.

The findings in the previous work suggest that a good portion of BM25's ineffectiveness lies in incorrect estimation of terms importance, as frequency based metrics fail to estimate the actual relevance. Motivated by this idea, Dai and Callan (2019a) introduces Deep Contextualized Term Weighting (DeepCT), to tackle the problem directly. The premise is simple, given that Term Frequency (TF) (see Section 2.2.1.A) fails to capture actual relevance, as terms may appear repeatedly without actually mattering in the context of the document, they propose a model that is able to predict a better estimation, that can then be used by BM25, in place of TF.

The first question they tackle, is how to find better relevance estimations scores for the model to train on. Although the notion of the relevance of each term is intuitive, it is not clear how to quantify it, let alone finding the human force required to label the entire text corpus. Instead, the authors suggest a simple metric, Query Term Recall (QTR), that leverages the known relevance judgements:

$$\mathrm{QTR}(t, d) = \frac{|Q_{d,t}|}{|Q_d|}, \tag{3.6}$$

with $|Q_d|$ as the set of queries that are relevant to document d, and $|Q_{d,t}|$ as its subset, containing the term t. QTR is based on the notion that a document and its set of relevant queries, tend to share relevant terms, giving a good estimation of the relevance of each term. Given this, DeepCT treats the predictions of these weights as a regression problem, leveraging BERT for contextualizing the tokens of the input text, followed by a simple regression layer, that estimates QTR for each embedding. Finally, DeepCT is used to predict the term weights of every document in the corpus, that after rescaling to match TF's scale (QTR, and therefore DeepCT, output values in the range $[0..1]$), can be used as drop-in replacements for TF, in the BM25 indexing function. Like the previous methods, DeepCT achieves considerable performance gains, over the "vanilla" BM25, without beating docTTTTTquery however, suggesting once again that the inclusion on new terms is crucial for its effectiveness.

The discussed models accomplish two great achievements: First, they are able to leverage transformers' powerful expressivity, without the added costs they entail at retrieval , by pushing associated computations to a prior step, during indexing. Second, they are able to do so, whilst mostly maintaining an inference time consistent with simpler indexing methods, namely BM25. These properties make them perfect candidates for first-stage retrieval, providing downstream rankers with a richer set of candidate texts to analyze, whilst maintaining inference times. Also note that for simpler scenarios, where the effectiveness of BM25 with document expansion alone is sufficient, queries can be executed on CPUs with low latency and high throughput (Nogueira et al., 2019c).

3.2.6.B Target Corpus Pretraining and Relevance Transfer

In the previous section, we have seen how to leverage document preprocessing, for simpler first-stage retrievers, allowing multi-stage rankers to achieve better performance, without added inference costs. Similar ideas can be applied to the transformer-based models, with the same goal in mind, specifically tackling their training. So far we've been assuming the common use case, where an already pretrained model (e.g. BERT's original checkpoint, as provided by Google) is fine-tuned for the ranking task, on the corpus data. Although there is definitely some overlap, between the pretraining and the target corpus, they may differ in vocabulary, style, structure, and other similar factors, particularly for specialized target domains (e.g. scientific or legal). As such, the model may benefit from additional pretraining in the target corpus, to mitigate this mismatch. Note that this additional pretraining, like the original, is merely based on weak supervision tasks, such as MLM or NSP, and does not require labeled data, allowing the model to manipulate data that otherwise would not. This approach has been shown to improve performance, even beyond ranking tasks, as reported by Raffel et al. (2020), Gururangan et al. (2020), Beltagy et al. (2019) and Nogueira et al. (2019b)

Similarly to how a model may benefit from contacting the training data, prior to finetuning, it may also benefit from contacting the task at hand in a out-of-domain setting (in a different labeled collection), prior to fine-tuning on the actual dataset. This idea of "stage-wise" or "multi-phase" finetuning, is particularly useful in cases where the in-domain data is insufficiently labeled, a situation prone to overfitting (failing to generalize). By finetunning in a properly labeled, out-of-domain dataset, where the model is able to learn the task effectively, it is then able to transfer it to the target corpus, during fine-tuning, an approach known as few-shot learning. In extreme cases, the model can even skip fine-tuning on the target corpus entirely, an approach known as zero-shot learning. These have also been shown to improve performance, as reported by Dai and Callan (2019b) and Xie et al. (2020).

The two described approaches can be thought of as safe techniques to improve performance, being worth a try in any experiment.

3.3 Representation-Based Transformer Architectures

Similarly to pre-BERT approaches, a good portion of neural ranking models with BERT, has followed representation-based architectures, a natural fit, given how BERT handles representations. The goal has been to leverage BERT, to learn query and document representations, later to be ranked according to a similarity function. Formally, for a query q, and a document d_i:

$$P(\text{Relevant} = 1|d_i, q) \triangleq \phi(\eta_q(q), \eta_d(d_i)), \tag{3.7}$$

where η_q and η_d are BERT-based models that generate representations for queries and documents respectively, and ϕ, a function that judges the similarity of both representations.

The interest behind this formulation lies in its ability to tackle two major limitations with the interaction-based approaches described previously. Firstly, it tackles BERT's slow inference, by pushing most computations to a prior step. This is possible since documents and queries are computed separately, as evidenced in Equation 3.7, allowing the precomputation of the representations of all documents (assuming a mostly static collection). At inference, only the query is computed and the similarity is calculated with ϕ, over all precomputed document representations, making ϕ also crucial for the speedup. Second, it doesn't require a multi-stage architecture , as long as ϕ enables fast similarity search algorithms. Multi-stage architectures can't be trained end-to-end, being optimized separately. For instance, first-stage retrieval is optimized for recall (see Section 5.1.1), in order to maximize the number of relevant candidates for the next stage. This, however, has no performance guarantees for the full model, as the distributions of the reranker's training data (sampled from the collection), and candidate set differs. Although this mismatch can be somewhat mitigated by careful sampling of training data, as seen on some extensions in Section 3.2.2, an end-to-end approach is a far more elegant solution.

So far we haven't specified ϕ or the format of η's output, a careful consideration to include all representation-based transformer architectures under the formula in Equation 3.7. This specification is precisely what separates most approaches, namely in ones that use simple comparison functions, such as cosine similarity, and consequently represent q, and d as vectors, or approaches that use complex comparison functions, and generally represent q and/or d, as sets of vectors. In the subsequent sections we'll discuss in detail the two cases.

3.3.1 Simple Comparison Functions for Ranking

The interest behind ranking with simple comparison functions (ϕ) (e.g cosine similarity) lies in the possibility of breaking the ranking down into a simple Nearest Neighbours Search (NNS), of a query vector, over the space of document representations. NNS is the optimization problem of finding the point in a given set that is closest (or most similar) to a given point. Although some efficient solutions to this prob-

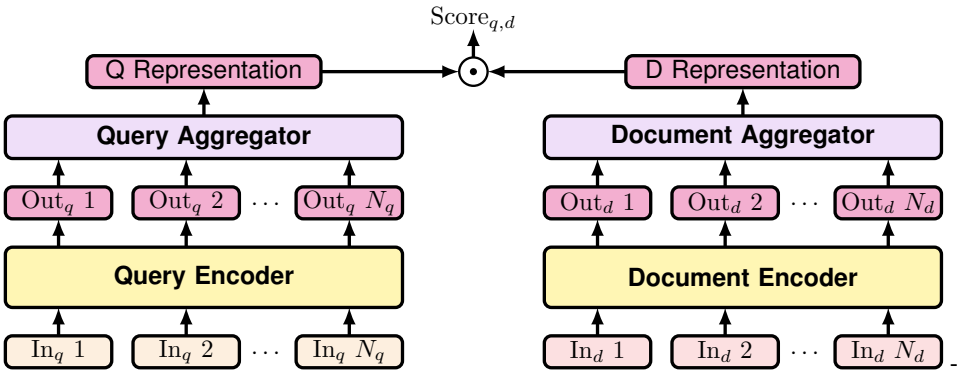


Figure 3.2: Graphical representation of a bi-encoder.

lem exist, they scale poorly beyond the dimension size of the considered representations. If, however, we allow slight inaccuracies in the comparison function, a version of the problem known as Approximate Nearest Neighbour (ANN), some highly scalable and efficient solutions do exist (Johnson et al., 2019; Malkov and Yashunin, 2018) under accessible libraries. Note that for the purposes of ranking, since ϕ isn't perfect at modeling the relevance task to begin with, the slight inaccuracies are acceptable. Also, the solutions for ANN problems go beyond the purposes of this thesis, however, assume their availability as efficient search libraries for the considered models.

ANN search libraries offer the speedup necessary to retrieve over the entire collection, excusing the need of a first-stage retrieval. The possibility of end-to-end retrieval explains why most representation-based architectures, that have surfaced, fall under this category.

3.3.1.A Bi-Encoder

To my knowledge, the first mention of the use of a representation-based transformer architecture for the purposes of ranking, is attributed to Humeau et al. (2019), naming the model "bi-encoder", alleging to the encoding of queries and documents separately, opposing the previously mentioned cross-encoder architectures. The authors pretrain a BERT-like model from scratch, with a special token `[S]`, but for simplicity, and in accordance with other variations in the literature (see Section 3.3.1.B), let us consider the use of BERT, and it's corresponding `[CLS]` token.

The architecture of the proposed model can be seen on Figure 3.2 and it follows the same terminology used in Section 3.2.1, for the cross-encoder. In general, the query and document sequences are encoded separately by a BERT encoder and an aggregator function (e.g., average pooling) is used to extract a representative vector representation of each, which are then compared trough a comparison function, outputting a relevance score.

Maintaining the nomenclature introduced in Equation 3.7, the authors consider the dot-product as the comparison function ϕ (same as cosine similarity for normalized vectors). The encoders η_q and η_d start

as the same BERT model (same initial weights), but are allowed to update separately during finetuning, and both are of the form:

$$\text{reduce}\left(\text{BERT}\left(t\right)\right), \tag{3.8}$$

where t is an input text, namely a query or document, BERT, preprocesses t as usual, by appending the `[CLS]` token to the beginning of the sequence, and $\text{reduce}(\cdot)$ is a function that reduces BERT's output sequence of vectors into a single vector, representative of t.

For $\text{reduce}(\cdot)$, the authors consider 3 possibilities: 1) to extract the first output embedding, correspondent to the `[CLS]` token, an obvious approach, considering how BERT leverages that special token for most tasks; 2) mean pooling over the entire sequence, computing the average embedding, an approach consistent with pre-BERT architectures that attempted document representations, as the average of its word embeddings; 3) mean pooling over the first m output embeddings, with m as an hyper-parameter $\leq N$ outputs, also consistent with pre-BERT architectures.

For a vector $\mathbf{v_q}$, the computed representation of a query q, the relevance score of a candidate document d, and its computed representation $\mathbf{v_d}$, is given by $\phi(\mathbf{v_q}, \mathbf{v_d})$. During fine-tuning, the network is trained to minimize the negative log likelihood (NLL) of the relevant document:

$$\mathcal{L}_{\text{nll}}(\mathbf{v_q}, \mathbf{v_{d^+}}, \mathbf{v_{d_1^-}}, \ldots, \mathbf{v_{d_N^-}}) = \log\left(\frac{e^{\phi(\mathbf{v_q}, \mathbf{v_{d^+}})}}{e^{\phi(\mathbf{v_q}, \mathbf{v_{d^+}})} + \sum_{i=1}^{N} e^{\phi(\mathbf{v_q} \cdot \mathbf{v_{d_i^-}})}}\right), \tag{3.9}$$

where d^+ is a relevant document and the others $\{d_1^-, \ldots, d_N^-\}$ are N documents chosen from the training set. The authors make use of a technique known as in-batch negatives (Yih et al., 2011), in which the relevant documents of other queries in the mini-batch, are used as negative examples for a given query, allowing efficient matrix multiplications that reuse the computed embeddings. This technique will be analyzed in detail on the next Chapter (see Chapter 4)

Trough experimentation, the authors found that using the representation extracted from the `[CLS]` token yielded slightly better results over the other approaches, for the finetuned model. The size of the mini-batch, namely the number of negatives, was also tested, resulting in performance gains, as the size increases. The authors also tested target corpus pretraining, which unsurprisingly also lead to a performance boost (see Section 3.2.6.B).

3.3.1.B Bi-Encoder Variations

Similarly to monoBERT, several approaches have surfaced following the original Bi-encoder, proposing several variations on the original architecture, namely on the sampling of training examples, loss functions, and integration with simple keyword search, specifically BM25.

Dense Passage Retriever (DPR) (Karpukhin et al., 2020) adapts the bi-encoder architecture for open-domain question answering. Specifically, the authors train a bi-encoder to retrieve the passages most likely to contain the answer to a query, the "retriever", followed by a BERT-based model, the "reader", trained to extract the answer span, achieving SOTA performance in various tested datasets. In their study, the authors test negative sampling techniques, namely random sampling, hard negatives picked with BM25 (documents with high BM25 scores but that are nonetheless non relevant) and in-batch negatives. From this study they found that in-batch negatives along with a single BM25 hard negative per query, resulted in a significant performance gain. A linear combination of DPR and BM25 scores further boosts retrieval effectiveness, suggesting that term-matching methods such as BM25 are sensitive to highly selective keywords and phrases, whilst bi-encoders capture lexical variations or semantic relationships better, complementing each other.

Complementing Lexical Retrieval with Semantic Residual Embedding (CLEAR) (Gao et al., 2020b) further investigates the interaction between keyword search and bi-encoder models. Contrary to the original bi-encoder, CLEAR uses the same BERT model for both encoders ($\eta_q = \eta_d$), but queries and documents are prepended with special tokens, `[QRY]` and `[DOC]`, respectively, in replacement of the `[CLS]` token. Another key difference lies in the final vector representation, which is produced by applying average pooling to BERT's output contextual representations. Like with DPR, the authors find that linearly combining the bi-encoder's scores with BM25's results in a performance gain. The authors also pick hard negatives with BM25 but use a triplet hinge loss (Weston et al., 1999), maximizing the similarity between the representations of a query, $\mathbf{v_q}$, and a relevant document, $\mathbf{v_{d^+}}$, whilst simultaneously minimizing the similarity between $\mathbf{v_q}$ and a non-relevant document $\mathbf{v_{d^-}}$, under a given margin m. This loss function corresponds to:

$$\mathcal{L}_{\text{hinge}}\left(\mathbf{v_q}, \mathbf{v_{d^+}}, \mathbf{v_{d^-}}\right) = \max\left(0, m - \phi\left(\mathbf{v_q}, \mathbf{v_{d^+}}\right) + \phi\left(\mathbf{v_q}.\mathbf{v_{d^-}}\right)\right), \tag{3.10}$$

However, rather than fixing the margin as is common (e.g m = 1), m is dynamically computed in accordance with BM25's retrieval scores for the same query and documents:

$$m = c - \lambda\left(\text{BM25}\left(\mathbf{v_q}, \mathbf{v_{d^+}}\right) - \text{BM25}\left(\mathbf{v_q}, \mathbf{v_{d^-}}\right)\right), \tag{3.11}$$

with c and λ as tunable hyper-parameters. The intuition here is that by discounting the loss in cases where BM25 is able to correctly rank the documents, since the value of m is lower, incurring in little or even no loss, the model specializes in harder, semantic cases, that BM25 fails to capture.

Representation-focused BERT (RepBERT) (Zhan et al., 2020b) shares CLEAR's similarity function, along with its aggregator and loss (Equation 3.11, with $m = 1$) functions, but drops the interaction with BoW models, opting instead for randomly pooled negatives.

Approximate Nearest Neighbor Negative Contrastive Estimation (ANCE) (Xiong et al., 2021) aims to improve the sampling of hard negatives during training. The authors argue that random negative sampling fails to provide hard negatives and sampling based on other models provides static hard negatives, which fail to properly prepare the model for testing, since the negatives differ considerably from those encountered in training. Optimally, the model should learn to distinguish negatives that are the closest to the positives, in its representation space. This is exactly what the authors promise, by having the model guide itself during training. Specifically, the model is used to dynamically sample the hard negatives, irrelevant documents ranked high, to be trained on. This is achieved by maintaining an ANN index, that is updated asynchronously as the training progresses. Specifically, every m training batches, a new ANN index is computed with the current checkpoint, as the training progresses parallelly with the current ANN index. Although maintaining the ANN index is computationally expensive, the expense can be mitigated by tuning the index refresh rate (m), for an effectiveness trade-off.

Despite the improvements brought on by ANCE, Zhan et al. (2020a) note the instability of the technique on the later training stages. The authors demonstrated that as training progresses, and the encoder refines the document and query representations, the asynchronous representations available at the ANN index become dated and provide inaccurate relevance estimations, resulting in weaker pooled hard-negatives. Although the problem can be somewhat mitigated by increasing the index refresh rate (m), this becomes unfeasible for large training collections. Instead, the authors propose Learning To Retrieve (LTRe), a final training stage that freezes the weights on the document encoder, guaranteeing the validity of representations within the ANN index, without requiring computationally expensive refreshes. On top of freezing the document encoder, the authors consider a pairwise loss function that is only applied on the cases where the model fails to retrieve the relevant document (i.e., the cases where the negative document produces higher relevance score than the positive), from the documents on the batch. Formally, this loss corresponds to:

$$\mathcal{L}_{\text{LTRe}}\left(\mathbf{v_q}, \mathbf{v_{d^+}}, \mathbf{v_{d^-}}\right) = \begin{cases} \mathcal{L}_{\text{RankNet}}\left(\mathbf{v_q}, \mathbf{v_{d^+}}, \mathbf{v_{d^-}}\right) & \phi\left(\mathbf{v_q}, \mathbf{v_{d^+}}\right) < \phi\left(\mathbf{v_q}, \mathbf{v_{d^-}}\right) \\ 0 & \phi\left(\mathbf{v_q}, \mathbf{v_{d^+}}\right) \geq \phi\left(\mathbf{v_q}, \mathbf{v_{d^-}}\right) \end{cases}, \tag{3.12}$$

where for the pairwise loss function, the authors consider the loss used in RankNet[2] (Burges, 2010):

$$\mathcal{L}_{\text{RankNet}}\left(\mathbf{v_q}, \mathbf{v_{d^+}}, \mathbf{v_{d^-}}\right) = \log\left(1 + e^{\phi\left(\mathbf{v_q}, \mathbf{v_{d^+}}\right) - \phi\left(\mathbf{v_q} \cdot \mathbf{v_{d^-}}\right)}\right). \tag{3.13}$$

Beyond the performance improvements brought on by the technique, freezing the document encoder may be key in pooling stronger hard-negatives akin to ANCE, without the computational expense of refreshing the ANN index. Despite this, at the time, the technique relied on a ANCE checkpoint in order to achieve better performance. On their follow-up work, Zhan et al. (2021) tackled this issue,

[2]The authors also considered LambdaRank (Burges, 2010), although noticing only small improvements on a per-case basis.

suggesting a new technique, Stable Training Algorithm for dense Retrieval (STAR), that achieves higher performance than ANCE's. More importantly, however, when used in conjunction with LTRe, produces even stronger results. Note that Zhan et al. (2021) revisits and renames LTRe, proposing an Algorithm for Directly Optimizing Ranking pErformance (ADORE).

STAR joins random and hard negative sampling, by pooling static hard negatives for each query, but sharing them within the batches. Specifically, the technique starts by using the base model to retrieve the top documents for all training queries, which serve as the static hard negatives that will not be updated. The batches are formed by sampling queries and their respective positive document, along with the hard negatives. However, unlike previous methods, the loss function considers for each query, not only the pooled hard negatives, but also all other documents present in batch (i.e., positives and negatives from other queries). Although not explicitly random, the shared negatives prevent the bias on the static hard negatives, stabilizing training.

So far, every bi-encoder discussed above achieves the performance of a simple monoBERT, at best, which reinforces the idea that interaction-based models have performed better than representation-based. Despite this, both architectures can have an almost symbiotic relation. We can leverage cross-encoders expressivity, to better guide a bi-encoder, for example by distilling a cross-encoder, to a much cheaper and faster bi-encoder. This is what Barkan et al. (2019) proposed , specifically with BERT-based cross-encoders and bi-encoders, for the task of sentence similarity. Remembering the equations 3.3 and 3.7, we can break down this transfer as:

$$\operatorname{softmax}\left(T_{\texttt{[CLS]}}\mathbf{W}+\mathbf{b}\right)_1 \triangleq \phi(\eta_q(q), \eta_d(d_i)). \tag{3.14}$$

After the distillation, the bi-encoder student model performs competitively with other bi-encoders, but remains consistently less effective than the cross-encoder, albeit much more efficient.

Qu et al. (2021), in RocketQA, leverages the higher expressivity of a cross-encoder to guide the training of the bi-encoder. The authors build upon DPR's architecture, but drop the interaction with the BoW model and devise a training pipeline, in which the bi-encoder and cross-encoder improve iteratively. In the first stage, the bi-encoder $M_D^{(0)}$ is trained without any particularity, but the use of a technique the authors named cross-batch negatives, similar to in-batch but also shares negatives between batches, when training on multiple GPUs. In the second stage, the trained bi-encoder $M_D^{(0)}$ is used to guide the training of the cross-encoder M_C. Specifically, the bi-encoder is used to select hard negatives, by random sampling from the retrieved top-k. This way, the M_C is adjusted to the distribution of the results retrieved by the $M_D^{(0)}$. In the third step, a bi-encoder $M_D^{(1)}$ is trained with hard negatives pooled from $M_D^{(0)}$, with the exception that negatives confidently classified as positives by the cross-encoder, aren't considered. This exception is meant to "denoise" the data, since the authors suggest that most big collections suffer heavily from false negatives, as only a small portion of the data is labeled. Finally,

in the last step, the collection is augmented, using M_C to label the top-k passages retried by $M_D^{(1)}$, for each query, and the final bi-encoder, $M_D^{(2)}$ is trained on it. With this training pipeline, RocketQA is able to surpass the performance of monoBERT on MS MARCO (Nguyen et al., 2016). In their ablation studies, the authors found the cross-batch negatives technique to be decisive to achieve performance gains and to provide efficacy of random negatives, when pooled in large numbers.

RocketQA also uses the bi-encoder as the first-stage retriever leveraging the fact that bi-encoders have been able to surpass the performance of BoW models, and ANN libraries allow relatively low inference times. Case in point, a multi-stage architecture joining RocketQA, and ERNIE (Sun et al., 2019b), a powerful BERT variation used as a cross-encoder, has near state of the art performance in the MS MARCO Passage Ranking benchmark, at the time of writing.

Although the previously discussed methods have tested different loss functions, the idea behind those losses is generally the same: maximizing the similarity between related document and query' representations, and minimizing the similarity for unrelated pairs, which as Ren et al. (2021) remarked, are exclusively "query-centric" similarity relations. Even though such relations are the only ones explicitly related to the ranking task, Ren et al. (2021) argues that bi-encoders may improve ranking performance from exploring other relations available within the training data. Specifically, the authors point out how query-centric relations fail to properly separate document representations, which can harm generalization for new cases. With PAssage-centric sImilarity Relations (PAIR) (Ren et al., 2021), the authors propose a document-centric (or passage-centric, as in accordance with the authors terminology) loss, that promotes the separation of unrelated document representations and complements the query-centric loss of Equation 3.9:

$$\mathcal{L}_{\text{d-centric}}(\mathbf{v_q}, \mathbf{v_{d^+}}, \mathbf{v_{d_1^-}}, \ldots, \mathbf{v_{d_N^-}}) = \log\left(\frac{e^{\phi(\mathbf{v_q}, \mathbf{v_{d^+}})}}{e^{\phi(\mathbf{v_q}, \mathbf{v_{d^+}})} + \sum_{i=1}^{N} e^{\phi(\mathbf{v_d^+}, \mathbf{v_{d_i^-}})}}\right). \tag{3.15}$$

Given how the document-centric loss does not relate directly with the ranking task, the authors propose a two-stage training procedure: (1) train the model with a combined loss, that considers both the document-centric and query-centric losses, enriching the learned representations; (2) train the model with the query-centric loss exclusively, optimizing the learned representations for the ranking task. Like Qu et al. (2021), the authors consider the powerful cross-batch negatives technique and also augment the training data with pseudo-labels produced by a well-trained cross-encoder. PAIR is able to outperform RocketQA, validating the impact of considering similarity relations beyond the common query-centric technique while training bi-encoders.

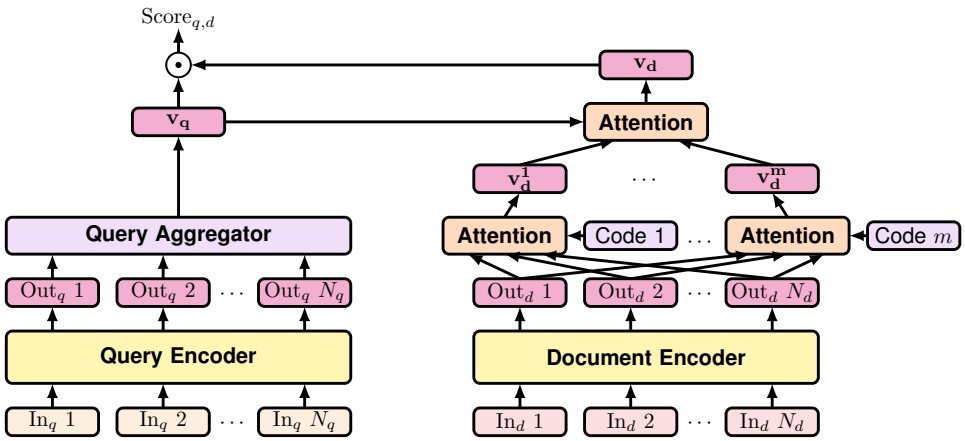


Figure 3.3: Graphical representation of the poly-encoder.

3.3.2 Complex Comparison Functions for Ranking

So far, the bi-encoder architectures we've addressed leverage simple comparison functions, enabling the use of fast ANN libraries which considerably speed up inference, enabling full-retrieval. Despite undeniably fast, these architectures have consistently performed worse than cross-encoders, suggesting that attention between query and document is able to capture relevance signals otherwise missed. A possible way of improving bi-encoder's performance is to use comparison functions of higher complexity, posing as a middle ground between simple bi-encoders and cross-encoders.

3.3.2.A Poly-Encoder

Humeau et al. (2019) proposed the poly-encoder architecture (see Figure 3.3), as a better performant alternative of their bi-encoder, which utilizes an attention-based comparison function that leverages multiple representations of the candidate text. The intuition behind the idea of using multiple representations comes from the fact that documents are commonly much longer than queries and, therefore, a single vector is insufficient to model their interaction. To address this, the authors propose an architecture on top of the usual encoder, that learns m context[3] codes $(\mathbf{c_1}, \ldots, \mathbf{c_m})$, in order to obtain m representative vectors $(\mathbf{v_d^1}, \ldots, \mathbf{v_d^m})$, where $\mathbf{c_1}$ extracts the representation $\mathbf{v_d^1}$ by attending over all the embeddings of the output layer $(\mathbf{h_1}, \ldots, \mathbf{h_N})$. Formally:

$$\mathbf{v_d^i} = \sum_j w_j^{\mathbf{c_i}} \mathbf{h_j} \quad \text{where} \quad (w_1^{\mathbf{c_i}}, \ldots, w_N^{\mathbf{c_i}}) = \text{softmax}\left(\mathbf{c_i} \cdot \mathbf{h_1}, \ldots, \mathbf{c_i} \cdot \mathbf{h_N}\right), \tag{3.16}$$

where N is the sequence length and m is an hyperparameter correspondent with the number of vector representations per document, controlling the trade-off between inference speed and performance.

[3] the word context refers to Humeau et al. (2019)'s terminology, where documents are portrayed as context and queries as candidates. For simplicity, we have maintained the terminology used throughout this thesis.

The query representation $\mathbf{v_q}$ is obtained similarly to other bi-encoder architectures, by extracting the `[CLS]` token's embedding. With both prior representations, the document vector $\mathbf{v_d}$ is computed as:

$$\mathbf{v_d} = \sum_i w_i \mathbf{v_d^i} \quad \text{where} \quad (w_1, \ldots, w_m) = \text{softmax}\left(\mathbf{v_q} \cdot \mathbf{v_d^1}, \ldots, \mathbf{v_q} \cdot \mathbf{v_d^m}\right). \tag{3.17}$$

The final score for that candidate label is then $\mathbf{v_d} \cdot \mathbf{v_q}$, as in a simple bi-encoder and the model is trained end-to-end, with cross-entropy loss (Equation 3.4). Since m ¡ N, and the query-document attention is only performed at the top layer, poly-encoder's attention is much faster than the cross-encoder's full self-attention, whilst achieving better performance than a comparable bi-encoder. However, unlike a bi-encoder, the poly-encoder architecture can't leverage ANN libraries for fast search, since despite computing the final ranking score as simple dot product, the computation of the final document representation requires query-document attention, making full-retrieval unpractical.

Poly-encoders introduced the idea of using multiple representations to represent each document, which has been continuously explored by others. One interesting approach is that of Luan et al. (2020), which proposed a simpler alternative to the poly-encoder's query-document attention, solely based on inner products. Instead of attention, the query–document score is simply the largest inner product between the query and each of the m document representations, which enables the use of ANN to speedup retrieval.

3.3.2.B ColBERT

Recently, Khattab and Zaharia (2020) proposed ColBERT, extending the idea of multiple representations to queries as well, computing a dense vector for each query or document token based on BERT's contextual representations. Formally, for a given sequence of tokens $s = [t_1, \ldots, t_n]$, ColBERT computes a matrix $\eta(s) \in \mathbb{R}^{n \times D}$, where n is the length of the sequence s and D is the dimension of each token representation, with each row corresponding to a vector representation. Similarly to the previously discussed CLEAR, ColBERT uses the same BERT model for encoding queries and documents, by prepending a special token, `[Q]` and `[D]`, respectively. After encoding with BERT, each output embedding passes through a linear layer without activation, responsible for translating each embedding into a vector of smaller dimension, D, followed by normalization (L2 norm), so that cosine similarity corresponds to a simple inner product. For the comparison function, ColBERT uses late interaction, as the authors name it, formally portrayed as:

$$s_{q,d} = \text{LateInteraction}(q, d) = \sum_{i \in \eta(q)} \max_{j \in \eta(d)} \eta(q)_i \cdot \eta(d)_j^{\mathrm{T}}, \tag{3.18}$$

where $s_{q,d}$ is the ranking score for a query q and document d, and the $\max(\cdot)$ function returns the maximum inner product between the query vector representation $\eta(q)_i$ (row i of the matrix), and each of the

document's representations (columns of the matrix $\eta(d)^{\mathrm{T}}$), an operation the author refer to as "MaxSim". Despite the reliance on simple inner products, late interaction fails to directly leverage ANN libraries, since computing the ranking scores relies on summing inner products for each query representation. To address this, the authors propose a two-stage retrieval approach: for the first-stage the authors index all document representations with an ANN library, annotated with their respective document. At inference, each query term embedding $\eta(q)_i$ is issued concurrently as a query and the top k' document representations are retrieved, where k' is an hyperparameter, with k' ¡ k and k is the number of documents to retrieve (e.g. k = 1000). The total number of representations is then $m \times k'$ and the actual documents are retrieved from these, through the annotations, resulting in K unique documents, with $K \leq m \times k'$, where m corresponds to the number of query representations. In the second-stage, the late interaction is computed for the entire K set, and the documents are ranked accordingly.

ColBERT is trained end-to-end, being optimized via pairwise softmax cross-entropy loss over the computed scores s_{q,d^+} and s_{q,d^-}, for triplets $\langle q, d^+, d^- \rangle$, corresponding to a query, a relevant document and an irrelevant document respectively. Formally:

$$\mathcal{L}_{\text{pair-nll}}(q, d^+, d^-) = -\log \frac{e^{s_{q,d^+}}}{e^{s_{q,d^+}} + e^{s_{q,d^-}}}. \tag{3.19}$$

ColBERT is able to achieve the effectiveness of simpler cross-encoders, such as monoBERT, despite having much faster inference. One of the secrets for this result, as tested by the authors, lies on a "query augmentation" step, during encoding, where query sequences are padded with BERT's `[MASK]` tokens, up to a desired sequence lenght. Given BERT's MLM pretraining task, the model is able to predict relevant embeddings for those positions, which can then be effectively leveraged by late interaction.

Despite the quick inference, ColBERT has poor space efficiency, since the number of document representations is tied to the sequence length, which can potentially reach 512 tokens. Although the authors managed to severely improve the space occupied by representations, trough techniques of lossless compression, the full index still remains almost 2 orders of magnitude larger than a BoW alternative.

To conclude the literature review on dense template retrieval approaches, table 3.1 aggregates all of the discussed methods, according with their mains characteristics such as loss functions, pooling techniques and sampling of training examples.

Method	Reference	Base Model	$\eta_q = \eta_t$	Pooling	Shared Neg.	Hard Neg.	Loss Function
DPR	(Karpukhin et al., 2020)	$\text{BERT}_{\text{base}}$	✗	[CLS]	in-batch	BM25	$\mathcal{L}_{\text{nll}}$ (eq. 3.9)
CLEAR	(Gao et al., 2020b)	$\text{BERT}_{\text{base}}$	✓	mean	-	BM25	$\mathcal{L}_{\text{hinge}}$ (eq. 3.10)
RepBERT	(Zhan et al., 2020b)	$\text{BERT}_{\text{base}}$	✓	mean	in-batch	-	$\mathcal{L}_{\text{hinge}}$ (eqs. 3.10, 3.11)
ANCE	(Xiong et al., 2021)	DPR	✓	[CLS]	-	prior checkpoint	$\mathcal{L}_{\text{nll}}$ (eq. 3.9)
LTRe	(Zhan et al., 2020a)	ANCE	✗	[CLS]	-	current checkpoint	$\mathcal{L}_{\text{RankNet}}$ (eq. 3.13)
STAR ADORE	(Zhan et al., 2021)	DPR STAR	✗	[CLS]	all documents -	base model current checkpoint	$\mathcal{L}_{\text{RankNet}}$ (eq. 3.13)
RocketQA	(Qu et al., 2021)	$\text{ERNIE}_{\text{base}}$	✗	[CLS]	cross-batch	cross-encoder	$\mathcal{L}_{\text{nll}}$ (eq. 3.9)
PAIR	(Ren et al., 2021)	$\text{ERNIE}_{\text{base}}$	✗	[CLS]	cross-batch	cross-encoder	$\mathcal{L}_{\text{nll}} + \mathcal{L}_{\text{d-centric}}$ (eqs. 3.9, 3.15)
Poly-Encoder	(Humeau et al., 2019)	$\text{BERT}_{\text{base}}$	✗	Attention	in-batch	-	$\mathcal{L}_{\text{nll}}$ (eq. 3.9)
ColBERT	(Khattab and Zaharia, 2020)	$\text{BERT}_{\text{base}}$	✓	-	-	BM25	$\mathcal{L}_{\text{pair-nll}}$ (eq. 3.19)

Table 3.1: Comparison of the dense retrieval methods presented in this section, according to their main characteristics: **Base Model** refers to model used to provide the initial parameters of the encoders, $\eta_{\mathbf{q}} = \eta_{\mathbf{t}}$ indicates whether the query and document encoders share parameters, **Pooling** indicates the technique used to pool the vectorial representations from the encoders, **Shared Neg.** indicates if and how negatives were shared within the batches, **Hard Neg.** indicates if hard negatives were used, and which model was used for sampling, **Loss Function** indicates the loss function that was used during training. Note that the information presented in this table is purposefully over-simplified, meant only to provide a brief overview over the discussed methods and facilitate comparisons. For clarifications and further details, the reader should consult the original papers.

4

Methodology

Contents

4.1 Simple Dense Template Retrieval

Following the customer support application introduced in Chapter 1, the formal definition for the problem of template retrieval is as follows: given a query q, the model must retrieve the single template t, from a relatively small collection of N_t templates, that better answers the query.

4.1.1 Architecture

Let us consider the commonly used dual-encoder architecture, as presented in DPR (Karpukhin et al., 2020) (see Chapter 3) , in which 2 independent encoders $E_Q(\cdot)$ and $E_T(\cdot)$ encode a query q and a template t into d-dimensional vectors, with different representation spaces. For ranking the templates, the cosine similarity between a query q and a template t is computed from the respective representations:

$$\mathrm{s}(q,t) = \text{cosine-sim}\left(E_Q(q), E_T(t)\right). \tag{4.1}$$

4.1.2 Loss Function

The loss function for training the encoders should maximize the similarity between positive query-template pairs $s(q,t^+)$ and minimize the similarity between negative query-template pairs $s(q,t^-)$. A commonly used loss term for this retrieval task is the negative log likelihood comparing the positive template t^+ against a set of negative templates $\mathcal{T}^-$:

$$\mathcal{L}_q(q,t^+,\mathcal{T}^-) = -\log\left(\frac{e^{\mathrm{s}(q,t^+)}}{e^{\mathrm{s}(q,t^+)} + \sum_{t^-\in\mathcal{T}^-} e^{\mathrm{s}(q,t^-)}}\right). \tag{4.2}$$

The final loss is then obtained by averaging the per-query loss from Equation 4.2 over all queries (and respective negative templates) considered in a batch from the dataset.

4.1.3 In-Batch Negatives

Selecting negative examples $\mathcal{T}^-$ for training dense retrievers is still an open problem, as seen in Chapter 3. Given the dynamic nature of customer support, the focus of this work is on methods that maximize training efficiency. Simple in-batch negatives, as described in DPR (Karpukhin et al., 2020), make optimal use of the batch space, by sampling query-document positive pairs and considering, for each query, all other documents within the batch as negatives. However, hidden in its simplicity lie 2 important assumptions: (1) the positive document of each query is in fact unrelated with all other queries within the batch; (2) the shared negatives provide a good estimation of instances within the full dataset. The weight of both assumptions is small for large corpora, where each document has a limited amount of

related queries and vice-versa, making false in-batch negatives (i.e., related documents that appear in other pairs within the batch) unlikely. Still, for smaller corpora such as those from customer support with templates, the assumptions can be problematic, requiring careful selection of the pairs, such that in all pairs within a batch, a document relates to their correspondent query and none other in the batch.

4.2 Improved Dense Template Retrieval

To improve on the method outlined in the previous section, two orthogonal innovative contributions are proposed and described next. The first one, described in Section 4.2.1, relates to a sampling strategy that correctly models the many-to-one relation between queries and templates, whilst the second one, described in Section 4.2.2, refers to an expanded loss function that provides better exploration of the training data, exploiting different similarity relations for queries and templates and considering hard sampling techniques to filter negatives within the batch.

4.2.1 Batch Generation

Template retrieval relates queries and templates in a strictly many-to-one correspondence, at the same time involving a small template collection. Moreover, since templates see different use, the number of queries per template varies considerably. These characteristics actively challenge the assumptions of vanilla in-batch negatives. In order to guarantee that the in-batch negatives are in fact negative, the sampled pairs must have different templates. This condition influences the distribution of training examples, penalizing frequently used templates and resulting in a distribution of negatives within the batch that follows the distribution of the templates, and not the real one.

4.2.1.A Labeled In-Batch Negatives

Given that, by definition, each query has a single related template, labelling each text (i.e., query or template) in a training batch with the corresponding template identifier provides sufficient information to create all valid positive and negative pairs. More specifically, let t_i correspond to the i-th template and $q_{i,j}$ to the j-th query, from the sub-collection of queries that is answered by t_i. Given a batch of N_q queries and N_t templates, with each text labeled with the corresponding template index i, for each query $q_{i,j}$ the template t_i is considered as positive, and all other templates t_n within the batch, with $n \neq i$, as negatives. This technique, referred to as labeled in-batch negatives, not only prevents in-batch false negatives, but also eliminates the paired sampling restrictions (i.e., the training examples do not have to be explicit query-template pairs) imposed by vanilla in-batch negatives. The rightmost part of Figure 4.1 shows a working example, for a simple batch of 5 queries and templates.

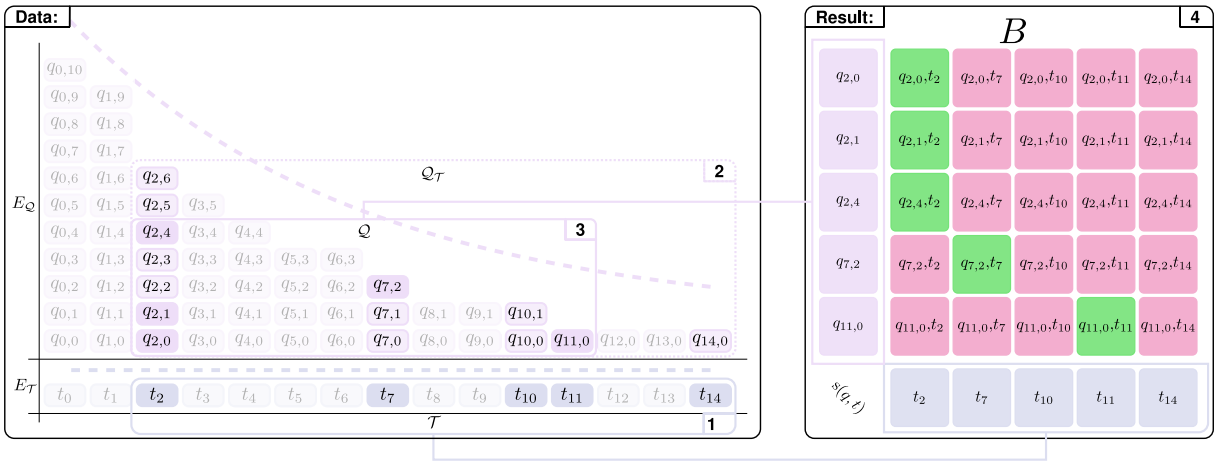


Figure 4.1: An illustrative case of applying semi-independent query-template sampling (Algorithm 4.1) on an example training dataset. The illustrated collection contains 15 templates $E_{\mathcal{T}}$ and 60 queries $E_{\mathcal{Q}}$, where each query is represented above the template that answers it. Each line in the Algorithm 4.1 corresponds to a rectangle in the figure, denoting the resultant subset of queries or templates, represented in higher opacity. The resultant batch B is followed by the application of the labelled in-batch negatives technique, yielding the represented matrix of query-template pairs, where the positive pairs are represented in green, and the negative ones in red.

4.2.1.B Semi-independent Query-Template Sampling

As a general rule, training instances should follow 2 principles: (1) uniform sampling of positive pairs, since these offer explicitly labeled relevance information that should be uniformly explored; (2) sampling negatives according to a distribution that is consistent with the corpus. Vanilla in-batch negatives fails to follow both principles, as the distribution of negatives within the batch follows the distribution of templates available in the positive pairs, and not the real one. With labeled in-batch negatives, on the other hand, positives and negatives are not directly tied, enabling the consideration of both principles.

To respect both principles, whilst maximizing the utility of the instances within the batch, a semi-independent query-template sampling strategy is proposed, as described in algorithm 4.1, and illustrated in Figure 4.1.

Algorithm 4.1: Semi-independent query-template sampling

Data: collection of training queries $E_{\mathcal{Q}}$ and the collection of their correspondent templates $E_{\mathcal{T}}$.
Result: batch B, of b labeled queries and b labeled templates.

1 Fetch a set $\mathcal{T}$ of b templates uniformly selected from $E_{\mathcal{T}}$, each labelled with its identifier.
2 Extract from $E_{\mathcal{Q}}$, the set $\mathcal{Q}_{\mathcal{T}}$ of all queries that are answered by the templates in $\mathcal{T}$.
3 Produce the set $\mathcal{Q}$, by randomly sampling b queries from $\mathcal{Q}_{\mathcal{T}}$, labelling them accordingly.
4 Compose batch B, from the queries in $\mathcal{Q}$ and the templates in $\mathcal{T}$.

Line 1 ensures that the sampled templates follow the distribution of the template collection, as in accordance with the second principle. Line 3 ensures that the sampled queries roughly follow the distribution of the available examples, as in accordance with the first principle.

4.2.2 Batch Exploration

The previous section focused on the proposed sampling technique that populates the training batches with queries and templates. This section follows the exploration of said batches, focusing on the extraction of useful information from other similarity relations and filtration of unnecessary training examples.

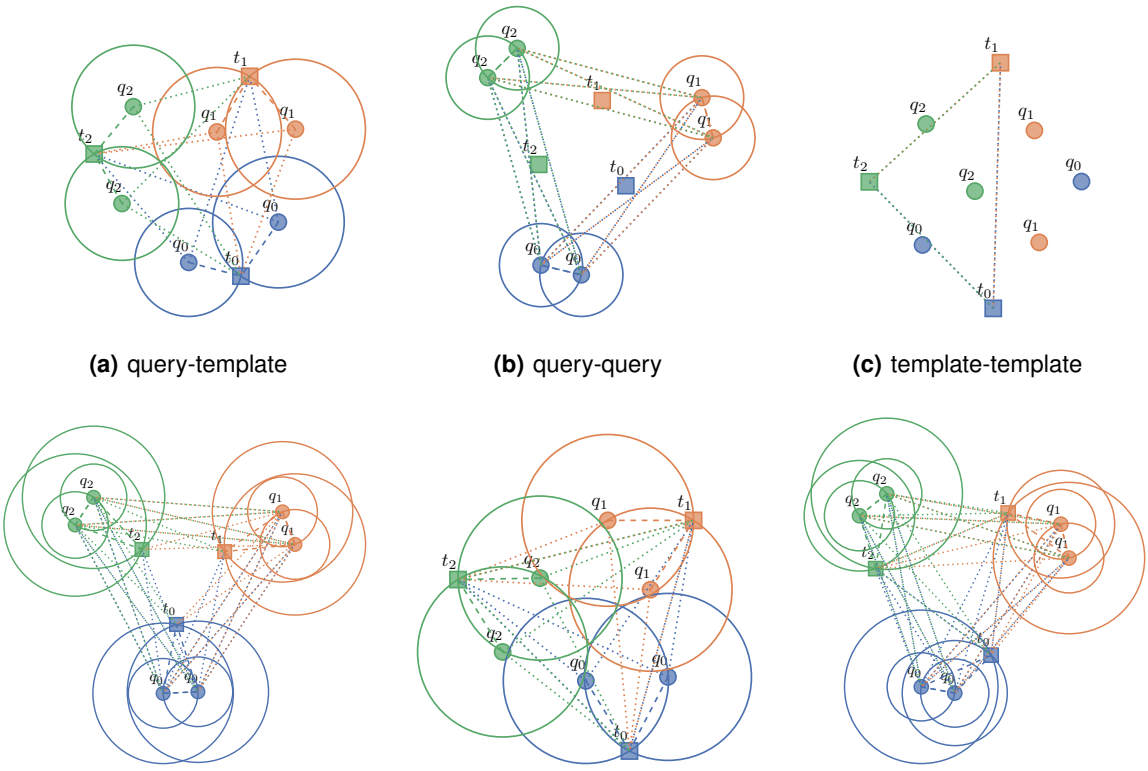
4.2.2.A Expanded Loss Function

In most information retrieval scenarios, the explicit relevance information available follows the retrieval task, relating queries with documents exclusively, that is, the relevance between each query and document is generally known, but not how queries and documents relate between themselves. As such, the query-document relations guide the majority of the proposed loss functions to model the retrieval task, namely Equation 4.2, promoting the similarity of related query-document pairs and the dissimilarity of unrelated query-document pairs. PAIR (Ren et al., 2021), as discussed on Chapter 3, extends this idea by also considering the relations between unrelated documents. To surpass the unavailability of explicit relevance information, the authors guide the selection of document pairs with powerful ranking models, producing pseudo-relevance information. Their work showed that considering similarity relations beyond the typical query-centric ones improves performance, despite not enforcing the ranking task directly.

In our information retrieval scenario with templates, like the ones discussed above, there is only explicit relevance information relating queries and templates. Unlike them, however, one can infer further valid similarity relations by considering the strict structure of the template collection, and its relation to the available queries. Since each query relates to a single template, being unrelated with any other, there is no overlap between the sets of related queries of each template. Since any two templates cover different queries, all templates cover different topics and are unrelated. Similarly, a query is more related with other queries covered by the same template, than queries covered by other templates. Considering this, one can assess whether any two texts (i.e., queries or templates) are related or unrelated. In order to leverage the additional relevance information, and build upon the works of PAIR, a novel loss function is proposed, that is expanded at the batch level, considering interactions not only between query-template pairs, but also query-query, template-template and template-query. To achieve this, the loss of a batch is given by a weighted sum of four terms, each focused on a type of pair:

$$\mathcal{L}_{\text{batch}} = \alpha\,\mathcal{L}(\mathcal{Q},\mathcal{T}) + \beta\,\mathcal{L}(\mathcal{Q},\mathcal{Q}) + \gamma\,\mathcal{L}(\mathcal{T},\mathcal{T}) + \theta\,\mathcal{L}(\mathcal{T},\mathcal{Q}), \tag{4.3}$$

where α, β, γ and θ are adjustable hyper-parameters, $\mathcal{Q}$ is the set of query representations within the batch and $\mathcal{T}$ is the set of template representations within the batch. The loss $\mathcal{L}(\mathcal{A},\mathcal{B})$ over two sets of representations $\mathcal{A}$ and $\mathcal{B}$, can be any loss function that for all pairs of the form $\{a,b\}$, with $a \in \mathcal{A}$ and $b \in \mathcal{B}$, promotes the similarity of the pairs where a and b are related, and dissimilarity for the pairs where



(a) query-template **(b)** query-query **(c)** template-template

(d) query-template + query-query **(e)** query-template + template-template **(f)** query-template + query-query + template-template

Figure 4.2: An illustrative case of 6 query representations $\{q_0, q_0, q_1, q_1, q_2, q_2\}$, and their respective positive template representations $\{t_0, t_1, t_2\}$ with t_i answering q_i, after enforcing different similarity relations. The distances between each point represent similarity (i.e., the more distant, the less similar), the dashed lines along with the circumferences represent similarity relations, whilst the dotted lines represent dissimilarity relations

a and b are unrelated. Although most loss functions discussed in Chapter 3 fit this criteria, the negative log likelihood (Equation 4.2) is widely used, namely in PAIR, being the one proposed to adapt.

For that, let us first notice that each text in a batch (which can be either from a query or a template) is given a label corresponding to the correct template (or itself). The negative log likelihood loss function can be adapted as the following generic loss term that takes two sets ($\mathcal{A}$ and $\mathcal{B}$) of labeled text representations (that can be from queries or templates) from the batch:

$$\mathcal{L}(\mathcal{A}, \mathcal{B}) = -\frac{1}{|\mathcal{A}|} \sum_{i \in \mathcal{I}} \sum_{a_i \in \mathcal{A}_i} \frac{1}{|\mathcal{B}_i|} \sum_{b_i \in \mathcal{B}_i} \log \left(\frac{e^{\mathrm{s}(a_i, b_i)}}{e^{\mathrm{s}(a_i, b_i)} + \sum\limits_{j \neq i} \sum\limits_{b_j \in \mathcal{B}_j} e^{\mathrm{s}(a_i, b_j)}} \right), \tag{4.4}$$

where $\mathcal{I}$ is the set of all labels in the batch while $\mathcal{A}_i$ is the set of texts in $\mathcal{A}$ that have label i (i.e., those that correspond to template i) and $\mathcal{B}_i$ is the set of texts in $\mathcal{B}$ that have label i.

Reconsidering Equation 4.3, now with the loss function defined in Equation 4.4, the different loss terms are as follows:

1. $\mathcal{L}(\mathcal{Q}, \mathcal{T})$; $\mathcal{A} = \mathcal{Q}$ is the set of all queries in a batch and $\mathcal{B} = \mathcal{T}$ is the set of all templates. This term corresponds to averaging the loss of each query $q \in \mathcal{Q}$, using the negative log likelihood of the positive template (Equation 4.2) combined with each possible negative template in the batch;

2. $\mathcal{L}(\mathcal{T}, \mathcal{T})$; $\mathcal{A} = \mathcal{T}$ and $\mathcal{B} = \mathcal{T}$ both correspond to the set of all templates in the batch. This term enforces the dissimilarity between templates;

3. $\mathcal{L}(\mathcal{Q}, \mathcal{Q})$; $\mathcal{A} = \mathcal{Q}$ and $\mathcal{B} = \mathcal{Q}$ both correspond to the set of all queries in the batch. This term enforces the dissimilarity between query representations from different templates, and promotes the similarity of representations for queries from the same template;

4. $\mathcal{L}(\mathcal{T}, \mathcal{Q})$; $\mathcal{A} = \mathcal{T}$ is the set of all templates in a batch and $\mathcal{B} = \mathcal{Q}$ is the set of all queries in the batch. This term is the transpose of the first one, having a similar effect but acting on each template instead of each query;

Figure 4.2 provides a visual representation of the similarity relations enforced by each of the loss terms, along with their combination. Query-centric similarity relations, Figure 4.2(a) enforce the similarity of query-template positive pairs and dissimilarity for negative ones. Figures 4.2(b) and 4.2(c) simply enforce query-query and template-template relations respectively, not being aligned with the retrieval task. Figures 4.2(d) and 4.2(e) combine query-template relations with the previous two relations, respectively; combining all, Figure 4.2(f) further enforces the similarity between valid query-query pairs and the dissimilarity between invalid query-query pairs and template-template pairs.

4.2.2.B In-Batch Top-k Negatives

Chapter 3 discussed recent methods that use the model's own representations to guide the selection of hard-negatives (Xiong et al., 2021; Zhan et al., 2021, 2020a). Although potentially effective, these techniques are computationally more demanding than the ones proposed, missing the established efficiency goals. Inspired by these approaches, a cheaper alternative is considered, consisting of using in-batch top-k negatives that instead of retrieving the top-k negatives over the entire corpus, retrieves them from within the batch. The process, as seen in Figure 4.3, consists of selecting the first $k + 1$ columns, from the sorted matrix of similarity scores, for computing the loss function. By reusing the representations within a batch, the approach is much cheaper while also guaranteeing that the representations are synchronous. Unlike ANCE (Xiong et al., 2021) and the other methods, the value of selecting the in-batch top-k negatives is not on the selected hard negatives, since they are already present in the batch, but

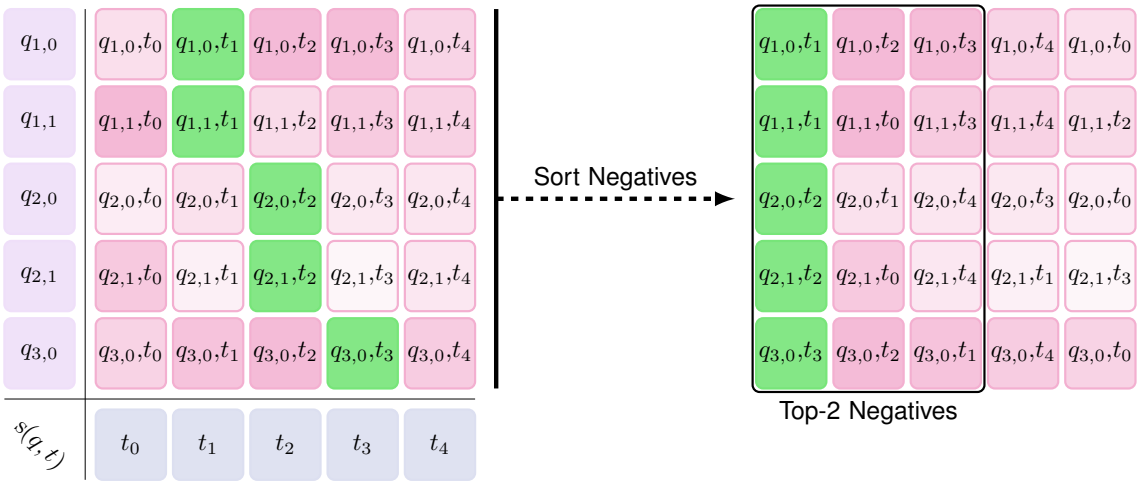


Figure 4.3: Diagram of labeled in-batch negatives followed by in-batch top-k negatives sampling. The batch-size (b) is 5 and the k in in-batch top-k negatives technique is 2. The positive query-template pairs are represented in green, and the negative pairs in red, with their opacity corresponding to the similarity score (i.e., the higher the opacity, the higher the score).

in discarding all others. This delays over-fitting on simpler negatives, allowing the model to learn the harder ones.

4.3 Summary of Techniques

This Chapter described all techniques that support the proposed model for semi-automatic selection of answer templates. Section 4.1 reviewed related work' techniques in the context of template retrieval, namely the in-batch negatives techniques and the negative log likelihood loss function. Section 4.2 exposed the limitations of the previous techniques on the template retrieval setting and proposed improvements. Specifically, Section 4.2.1 proposed an alternative to in-batch negatives, that considers labels for generating positive and negative pairs, detailed in Section 4.2.1.A, along with a sampling technique that samples queries and templates in a semi-independent manner, detailed in Section 4.2.1.B, Section 4.2.2 proposed a loss function that considers multiple negative log likelihoods enforcing different similarity relations, detailed in Section 4.2.2.A, along with a technique that filters easy negatives within the batch, detailed in 4.2.2.B.

5

Experiments

Contents

This section presents the experimental validation of our contributions in two datasets of costumer support interactions.

5.1 Datasets and Metrics

The experiments were mainly conducted on a private anonymized dataset (CS-P1) consisting of real customer support interactions in English, made over email, where a human agent handpicked templates for answering customer requests. We also tested the models on three smaller private anonymized datasets in Portuguese (CS-P2, CS-P3, CS-P4). Built from real interactions, these datasets are the most representative of the task at hand, although they cannot be made available. due to company/client restrictions

In the interest of reproducibility and to further stimulate research on the topic, we also crafted a dataset with public content[1] that corresponds to customer support interactions and approximates the task at hand. To create this public dataset (CS-Twitter), we built on previous work (Hardalov et al., 2018), in which the authors handcrafted a customer support dataset for chatbots, from the public *Customer Support on Twitter* data that is available on Kaggle[2]. We followed the same preprocessing, isolating the tweets related to Apple support and filtering out the noisy ones. We nonetheless processed the data further, to attend the needs of our problem. Given the apparent use of templated responses, we started by filtering out all tweets beyond the first interaction, as these were context specific and less likely to include templated answers. For the remaining tweets, we clustered similar responses, hopefully sharing a template, and assumed the clusters to be the golden template identifiers for the tweets in the clusters.

After analysing various text distances, we decided to capture the similarities between response tweets leveraging a pre-trained model from the Sentence-Transformers library (Reimers et al., 2019) to produce sentence embeddings. In order to aggregate the similar sentences, we used HDBSCAN (Campello et al., 2013) to produce clusters over the response embeddings. Finally, we removed all clusters containing less than 5 elements and selected, for each cluster, the response that is closest to the centroid, as the template. From inspection of the produced data, HDBSCAN could provide better clusters than alternative methods, such as DBSCAN or k-means.

We split all datasets into 3 partitions, namely *train*, *val*, and *test*. The *test* split is composed of the most recent customer interactions , simulating the real scenario, whilst the *train* and *val* splits are composed of the remaining examples on a 85/15 stratified split (see Table 5.1 for a characterization of the datasets).

[1] https://github.com/t-mesq/twitter-apple-cs-hdbscan
[2] https://www.kaggle.com/thoughtvector/customer-support-on-twitter

5.1.1 Evaluation Metrics

The used evaluations metrics are meant to capture the models' performance in the real world setting. As stated, the objective for the models is to retrieve a subset with k templates that is likely to contain the single, correct one, for an incoming query. Given how the model ranks all templates, based on the likelihood of corresponding to the correct answer, the subset simply corresponds to the top-k.

This idea is perfectly captured by the metric *recall@k*, that reports on average, the percentage of known correct answers, present in the retrieved top-k candidates. Given our specific case, where only one correct answer exists, we can formally define *recall@k* as:

$$\text{recall@k} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} I_{A_{[k]}} \left(A^* \right), \tag{5.1}$$

where Q corresponds to the set of queries for evaluation, $I(\cdot)$ corresponds to the indicator function, $A_{[k]}$ corresponds to the subset with the top-k candidates, of the full set of candidates A, and A^* corresponds to the correct answer. The preferred value for k is 3, as it corresponds to the accorded goal of retrieving the top-3 candidates for posterior selection by the human operator, but other values will also be recorded in order to better assess the progression and performance of the model.

Besides *recall@k*, that tracks the retrieval performance, since the actual rank of the correct answer does not impact the metric, as long as it is included in the top-k, Mean Reciprocal Rank (MRR) is used to track the ranking effectiveness. MRR reflects the inverse of the average rank of the first correct candidate, being a good metric to track the models' progression. Once again, given our specification with a single correct answer, MRR is formally defined as:

$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}\left(A^* \right)}, \tag{5.2}$$

with the same definitions as the prior Equation (5.1) and where $\text{rank}(\cdot)$ refers to the position of a candidate, in the ranked set.

Dataset	#query-response pairs			#templates		$P_{80\%}$ token-length		Language
	train	val	test	#total	%unseen	queries	templates	
CS-Twitter	7969	1425	2884	480	41.2	34	35	English
CS-P1	17858	3127	3918	445	2.8	113	396	English
CS-P2	1092	187	650	82	1.2	111	164	Portuguese
CS-P3	973	170	352	36	2.0	103	221	Portuguese
CS-P4	275	44	150	50	18.7	216	215	Portuguese

Table 5.1: Statistics for CS-Twitter and the private datasets. %Unseen refers to the percentage of test examples that include templates not included in the train split and Token-length indicates the number of tokens of texts, in terms of DistilmBERT$_{\text{base}}$ tokens.

5.2 Experimental Setup

This section details the baselines considered for the experimental evaluation, along with the pre-trained language models and hyper-parameters used to train the proposed approach.

5.2.1 Baselines

As a sparse retrieval baseline we consider a traditional BM25 (Lin et al., 2021) approach. For a dense retrieval baseline, we tested all multilingual models in Sentence-Transformers (Reimers and Gurevych, 2020), in a 0-shot manner, and report results for the best: `distiluse-base-multilingual-cased-v1`. Finally, as an alternative to retrieval, and given the small size of the template space, we considered a simple multi-class classifier baseline, where each class corresponds to a template and we use the predicted probabilities as the ranking scores. Given that the model's output space is directly tied with the train template space, the model is unable to retrieve templates not found in training.

5.2.2 Pre-Trained Language Models

Although our datasets were mostly in english, we only considered multilingual models in accordance with the real world scenario where we envision the models will be applied. Both the query and template encoders, on the trained dense retrievers, were initialized with the parameters of the `distiluse-base-multilingual-cased-v1` model from the Sentence-Transformers library (Reimers and Gurevych, 2020), as this was the best model in a zero-shot retrieval setting. For the classifier baseline, we used DistilmBERT$_{\text{base}}$ model.

5.2.3 Hyper-Parameters

The batch-size (b) used for training was 32 for all private datasets, and 192 for CS-Twitter, as its templates and queries are considerably smaller in length, in all experiments with in-batch negatives. For experiments with random negatives, we used b=8 in for all private datasets and B=64, for CS-Twitter, sampling $N = 4$ negatives in both cases. We used larger batch-sizes for CS-twitter because the texts are much smaller then the emails in the private datasets. We also set the maximum number of training epochs to 30, for all private datasets, and 50 for CS-Twitter. Finally, we used linear learning-rate scheduling with 500 warmup steps, and the ADAM optimizer (Kingma and Ba, 2015) with a learning-rate of 3e-5.

Methods	CS-P1				CS-Twitter			
	MRR@10	R@3	R@10	Epochs	MRR@10	R@3	R@10	Epochs
Classifier	41.2	46.9	65.4	17	5.6	7.3	12.9	35
BM25	8.5	10.1	19.0	-	2.6	3.2	6.7	-
SBERT 0-shot	10.2	12.0	25.3	-	3.2	4.0	8.6	-
Random negatives	40.2	45.9	65.5	16	7.6	8.9	18.2	16
In-batch neg^{t}	38.2	44.8	63.8	24	6.6	7.7	14.7	44
In-batch neg^{q}	32.9	37.7	51.1	26	6.9	7.8	15.4	48
Labeled in-batch neg^{q}	39.2	45.6	63.7	4	7.2	8.4	16.7	6
Labeled in-batch neg^{t,q}	41.8	47.0	67.5	6	7.8	9.6	**19.1**	6
Proposed approach	**42.7**	**48.4**	**68.3**	3	**8.6**	**10.6**	18.9	8

Table 5.2: Experimental results on CS-Twitter and CS-P1.

5.3 Experimental Results

This section presents the main experiments, studying different sampling techniques and the impact of the each term in the proposed loss.

5.3.1 Main Results

Besides the baselines and proposed approach, that integrates both the proposed loss and sampling techniques, we also considered 5 other settings corresponding to the use of the vanilla loss ($\mathcal{L}(\mathcal{Q}, \mathcal{T})$), but with different mechanisms to construct the negative instances:

- **Random negatives**: randomly samples N negative templates for each query-template pair;
- **In-batch neg^{t}**: samples B templates, uniformly and without repetition, along with a positive query for each template;
- **In-batch neg^{q}**: samples B templates, weighed by frequency of positive queries and without repetition, and a positive query for each template;
- **Labeled in-batch neg^{q}**: samples B queries, uniformly and without repetition, along with each positive template. If this produces repeated templates, we swap them with uniformly sampled templates not present in the batch;
- **Labeled in-batch neg^{t,q}**: corresponds to the proposed sampling technique, as described in Chapter 4;

Table 5.2 presents the obtained results, from which we can infer the following main conclusions:

1. The proposed sampling technique not only outperforms all the alternative methods in both datasets, but it does so with considerably fewer training steps . As expected, vanilla in-batch negatives is sub-optimal for template retrieval. The labeled in-batch negatives were key in overcoming the sampling limitations, and the proposed technique makes good use of its capabilities.

2. The proposed loss, that also considers template-template and query-query similarity relations, yields a significant performance boost. This result suggests that exploring semantic relations beyond the ranking task is beneficial, likely being a result of learning more robust representations with better generalization capabilities.

3. The overall poor performance on CS-Twitter exposes potential problems with the dataset, probably because it was created semi-automatically with reduced human supervision. Despite this, most results seem to be in agreement with those from CS-P1, enforcing the validity of the conclusions. In fact, the only noticeable discrepancy occurs in the experiment involving the **in-batch neg**^{q} strategy, with CS-Twitter exhibiting better relative performance. The discrepancy can be explained by the large batch-size (192), corresponding to almost half of the total number of templates. Since each batch will include a large number of sampled templates, the model is able to better explore the full corpus, independently of the sampling strategy.

4. The poor performance of BM25 exposes the difficulty of the template retrieval task. Since each template covers a range of queries, the text is generally unspecific, resulting in reduced term overlap between templates and queries. Trained dense retrievers, or the classification model, on the other hand, were able to achieve good performance, showing that semantic relations are effectively superior to simple term matching.

5. The classification baseline is in fact quite strong, outperforming several of the retrieval methods. This can be attributed to the small number of templates, although it should be emphasised that template collections can be highly dynamic in real settings, motivating the use of retrieval methods that can adapt to the collection without model re-training.

5.3.2 Analysis on the Sampling Techniques

The experiments in Table 5.2 already compare the different sampling techniques. To provide better insights over the practical differences of each method, we plot the distributions of templates and queries, throughout training, for each technique. To collect the data points, we record the template identifiers of the sampled queries and templates, at each step, for a total of 10 epochs in CS-P1. The result of this study is presented in Figure 5.1, which confirms the intuitions behind the design of the proposed sampling technique. As expected, vanilla in-batch negatives mirrors the distributions of queries and

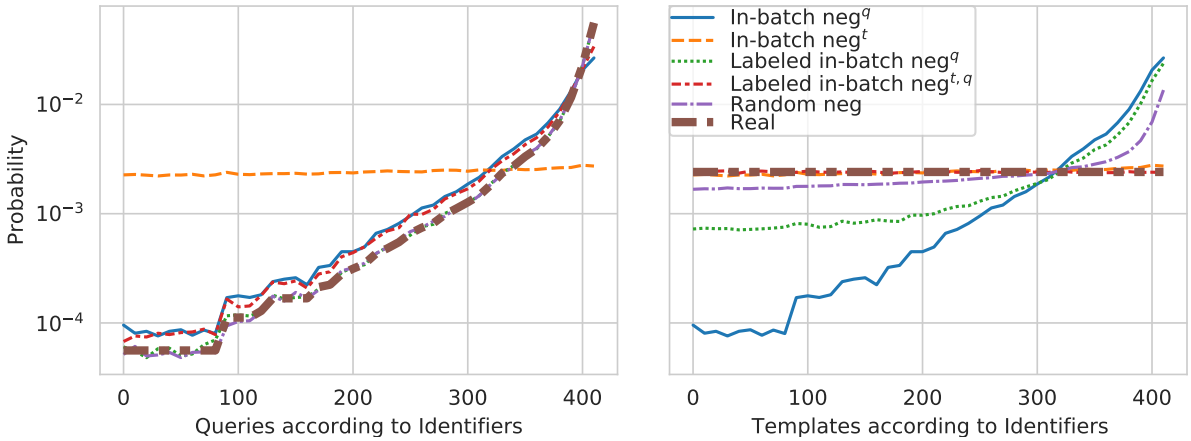


Figure 5.1: Comparison between the real and observed distributions obtained with different sampling techniques, during training, for queries (LEFT) and templates (RIGHT). The template identifiers are ordered by the real distribution and we plot the mean over bins of 10 templates, to reduce the number of data points and generate smoother lines that are easier to interpret.

templates, as they are sampled in pairs. This results in techniques that are only capable of optimizing the distribution of templates (i.e., **in-batch neg**^{t}) or queries (**in-batch neg**^{q}), but not both, resulting in sub-optimal performance. Labeled in-batch negatives are effectively able to decouple both distributions, being key in providing good estimations. **Labeled in-batch neg**^{q} provides a good estimation over the distribution of queries, although, the observed distribution of templates is slightly biased towards the most frequent. This results from the query-guided sampling technique, which explains the slightly worse performance. **Labeled in-batch neg**^{t,q}, on the other hand, is able to provide a uniform distribution of templates, whilst maintaining a distribution of queries very close to the real one, providing by far, the best balance and accompanying best performance. **Random negatives**, despite selecting templates on a per-query basis, is still slightly biased towards the most frequent templates, a result of the positive examples still following the query distribution. This, coupled with the reduced number of negatives, are likely the main factors for the lower performance.

Overall, the results seem to imply a strong correlation between the quality of the sampling techniques, as an estimator of the involved distributions, and the observed retrieval performance.

5.3.3 Analysis on the Loss Terms

The proposed loss function combines different negative log-likelihoods, each enforcing a different similarity relation. In order to assess the contribution of each component, along with their interaction, we tested 5 different combinations:

- $\mathcal{L}(\mathcal{Q}, \mathcal{T})$: corresponds to the control experiment and simply considers the common negative log likelihood over the positive template;

Loss	in-batch top-k neg	CS-P1			CS-Twitter		
		MRR@10	R@3	R@10	MRR@10	R@3	R@10
$\mathcal{L}(\mathcal{Q},\mathcal{T})$	✗	41.8	47.0	67.5	7.8	9.6	**19.1**
	✓	41.9	48.3	67.9	8.1	9.6	18.4
$\mathcal{L}(\mathcal{Q},\mathcal{T})+\mathcal{L}(\mathcal{T},\mathcal{T})$	✗	38.7	45.3	65.4	7.2	8.7	18.0
	✓	39.7	45.9	65.6	7.6	8.8	18.0
$\mathcal{L}(\mathcal{Q},\mathcal{T})+\mathcal{L}(\mathcal{Q},\mathcal{Q})$	✗	41.5	46.7	67.8	8.0	9.6	17.8
	✓	41.6	**48.4**	67.1	7.7	9.2	18.6
$\mathcal{L}(\mathcal{Q},\mathcal{T})+0.5(\mathcal{L}(\mathcal{Q},\mathcal{Q})+\mathcal{L}(\mathcal{T},\mathcal{T}))$	✗	41.8	47.0	67.7	**8.6**	**10.6**	18.9
	✓	**42.7**	**48.4**	**68.3**	8.3	10.3	18.3
$\mathcal{L}(\mathcal{Q},\mathcal{T})+\mathcal{L}(\mathcal{Q},\mathcal{Q})+\mathcal{L}(\mathcal{T},\mathcal{T})+\mathcal{L}(\mathcal{T},\mathcal{Q})$	✗	41.3	47.0	65.9	8.4	10.2	19.0
	✓	41.4	47.7	67.4	7.5	8.7	16.8

Table 5.3: Ablation study on the components of the loss and in-batch top-k sampling, on CS-Twitter and CS-P1.

- $\mathcal{L}(\mathcal{Q},\mathcal{T}) + \mathcal{L}(\mathcal{T},\mathcal{T})$: considers equal contribution of and template-template and query-template relations,
- $\mathcal{L}(\mathcal{Q},\mathcal{T}) + \mathcal{L}(\mathcal{Q},\mathcal{Q})$: considers equal contribution of query-template and query-query relations;
- $\mathcal{L}(\mathcal{Q},\mathcal{T}) + 0.5(\mathcal{L}(\mathcal{Q},\mathcal{Q}) + \mathcal{L}(\mathcal{T},\mathcal{T}))$: combines the query-template relations with equally contributing template-template and query-query relations;
- $\mathcal{L}(\mathcal{Q},\mathcal{T}) + \mathcal{L}(\mathcal{Q},\mathcal{Q}) + \mathcal{L}(\mathcal{T},\mathcal{T}) + \mathcal{L}(\mathcal{T},\mathcal{Q})$: similar to the previous loss, but additionally considering template-query relation.

For each of the considered losses, we also test the impact of using in-batch top-k negatives. We selected values for k experimentally, after testing different powers of 2, resulting in $k = 4$ for CS-P1 and $k = 24$ for CS-Twitter. The results of these experiments are presented in Table 5.3.

Just as reported in PAIR (Ren et al., 2021), the loss that combines query-template and template-template relations under-performs, suggesting some misalignment with the retrieval task. Similarly, combining query-template and query-query relations also appears ineffective. The combination of both, however, produces a considerable gains, suggesting complementarity. Moreover, in CS-P1, in-batch top-k sampling improved performance consistently, regardless of the considered loss. The same is however not true in the case of CS-Twitter. We believe this discrepancy relates to potential problems in the dataset. Specifically, during the construction of the dataset, the cases where the clustering algorithm failed to group responses sharing an underlying template produce incorrect hard negatives that are harmful for training, often being picked up in the top-k lists.

5.3.4 Analysis on Harder Datasets

The previous studies focused on CS-Twitter and CS-P1, as the sizes of these datasets provide better training conditions and more representative results. However, in a real world scenario, training data is

Methods	CS-P2			CS-P3			CS-P4		
	MRR	R@3	R@10	MRR	R@3	R@10	MRR	R@3	R@10
Classifier	55.1	63.5	84.5	62.2	71.9	**91.5**	20.9	28.0	48.0
BM25	13.7	16.6	25.1	14.2	15.3	44.0	14.2	14.7	36.0
SBERT 0-shot	16.2	20.2	35.5	24.2	28.1	57.1	18.6	24.0	44.7
Random negatives	65.7	76.9	87.1	65.4	73.3	87.5	30.3	35.3	70.0
In-batch neg^{t}	64.6	74.0	89.1	65.2	72.7	86.9	33.9	40.7	60.0
In-batch neg^{q}	63.8	73.8	87.7	64.9	73.3	88.1	36.4	**44.0**	**72.0**
Labeled in-batch neg^{q}	64.1	72.6	85.2	64.8	72.7	87.2	35.4	40.0	69.3
Labeled in-batch neg^{t,q}	64.3	76.6	**91.1**	66.4	71.9	89.8	34.3	40.0	68.7
Proposed Approach	**68.4**	**77.5**	89.8	**67.4**	**74.7**	89.8	**36.8**	42.7	64.7

Table 5.4: Experimental results on other private datasets

not always abundant (e.g., brand new services) and, as such, achieving good performance in these conditions is imperative. To test these scenarios, we evaluate the previous models on 3 smaller portuguese datasets and present the results in Table 5.4:

1. Generically, CS-P2 and CS-P3 register a considerable higher performance when compared to all other tested datasets, despite having relatively few training examples. This is likely a result of the smaller number of templates, facilitating the retrieval task. Still, CS-P2 and CS-P3' examples-templates ratio is considerably lower than CS-P1, suggesting that as the template collection grows, effective training requires considerably more training data.

2. The classifier's performance was noticeably lower than that of the dense retriver models, especially in CS-P4. This is expected as unlike the bi-encoders that can extract relevance judgements from the template text, the classifier depends exclusively on the available training queries. For smaller datasets, these are often insufficient to construct a general understanding of the query space and the model fails to generalize for newer queries. This is especially noticeable when comparing the classifier's performance on CS-P2 and CS-P3. Unlike the bi-encoders, which achieve similar performance in both datasets , the classifier clearly performed worse on CS-P2, the dataset with a less queries per template. In the case of CS-P4, the larger difference is also partially explained by the higher percentage of text examples involving templates that weren't seen during training, which the classifier is functionally unable to retrieve. Both factors cement the dense retriever architecture as the stronger model in harsher training conditions.

3. The proposed approach mostly retained the best performance, further validating its ideas and proving the robustness of the new methods in harsher training conditions.

4. The **in-batch neg**^{q} strategy was, in general, moderately effective , just like in CS-twitter. This, again, is likely a result of the relatively small number of templates, in comparison to the considered

batch-size (32). In particular, the method achieves better performance than **in-batch neg**^{t}, in CS-P3, but not on CS-P2, which is likely the consequence of CS-P2 having a considerably higher number of templates than CS-P3 (82 $>>$ 36).

5. The overall good results on multilingual data validate the applicability of the proposed approach in other languages. This is expected, as the approach simply refines the representation space of the base model. Given the multilingual pretraining of the base model, the approach is effectively able to transfer multilingual knowledge.

5.4 Summary of Experiments

This chapter described the conducted experiments, providing insight over the challenges of template retrieval and the proposed bi-encoder architecture. Section 5.1 detailed the datasets where the experiments were performed, along with the metrics used to track the models' performance. Section 5.2 described the experimental methodology. Finally, Section 5.3 presented all the reported experiments. Specifically, Section 5.3.1 compared the performance of the tested approaches on the main datasets, CS-P1 and CS-Twitter, showing the proposed approach to noticeably outperform all others, Section 5.3.2 tested the different sampling techniques and related their performance with the observed training distributions, demonstrating their clear correlation and explaining the success of the proposed sampling technique. Section 5.3.3 tested the contribution of each loss term in the proposed loss function, along with the in-batch top-k negative technique, evidencing the complementary nature of the loss terms, and finally, Section 5.3.4 compared the performance of the tested approaches on smaller datasets, exposing the classifier's poor performance when training with fewer training examples, in comparison with the ranking models.

6

Conclusions and Future Work

Contents

This chapter recaps the main contributions, along with the conclusions they bring, and suggests future works focused on improving and adapting the proposed techniques.

6.1 Conclusions

This dissertation proposed a model for semi-automatic selection of answer templates, adapting SOTA dense retrieval models to retrieve over the template collection. Our work explored the main challenging factors that oppose this adaptation, developing novel techniques to mitigate them, namely a semi-independent query-template sampling technique and an expanded loss function that considers multiple similarity relations. Experiments on several private datasets, and a purposely built dataset with public Twitter customer support data, attest to the efficacy of the proposed techniques and the complete model.

Tests showed that the proposed sampling technique is not only able to model the skewed distribution of training examples resultant from the different use frequencies of each template, but also does so efficiently. It achieves this by sampling queries and templates in separate stages, each focused on the respective distribution. The technique, semi-independent query-template sampling, is likely the most insightful contribution of our work. Its importance lies not only on the performance/efficiency gains it provided, but also on the insight regarding the importance of separating the distribution of queries and documents (i.e., templates) in training examples. We showed empirical evidence that correctly approximating these distributions is key in achieving good performance. Furthermore, we highlight the common in-batch negatives technique's inability to correctly model both distributions, as pair sampling imposes the same distribution of queries and templates. This problem is, for the most part, overlooked in the recent works of information retrieval, being often undermined by the lack of hard negatives, or abundance of easy negatives, also characteristic with in-batch negatives (Yates et al., 2021). The constant undermining of this problem is likely a result of 2 main factors: (1) The success of hard negative sampling techniques, proving by opposition the importance of explicitly including hard negatives; (2) The public datasets that see most work (Nguyen et al., 2016) are often sparse, with most queries having a single positive document and vice-versa, resulting in similar distributions, which is the optimal case for in-bath negatives. These conclusions highlight the importance of testing different IR scenarios that challenge the norm.

In regards to the proposed loss function, Ren et al. (2021), with PAIR, already introduced the potential performance gains associated with considering similarity relations beyond the typical query-centric, but only explored the dissimilarity between documents. The tests regarding the proposed loss show that the query-query similarity relations are not only useful, but also appear to complement the former.

Although less impactful, the in-batch top-k negatives technique also produced modest improvements. The technique is highly integrable and analogue to ANCE's top-k negative sampling (Xiong et al., 2021),

but performed on a batch level. Surprisingly, it seems mostly unexplored, despite its simplicity. This is likely the result of the somewhat counter-intuitive notion that a model may benefit from constricting the available information within the batch. The idea works since the process only filters the easy negatives, which not only fail to provide useful information, but may even be harmful.

Overall, the experiments on different datasets of customer support interactions attest to improvements brought forward by the proposed ideas and lead us to conclude that a dense retriever framework for retrieval of customer support templates is not only feasible, but competent.

6.2 Future Work

Despite positive, the results achieved, along with the techniques proposed, motivate their further exploration. For organization, this section is subdivided into two: Section 6.2.1 pertaining to future work on the topic of customer support with templates, improving the proposed techniques and proposing others, whilst Section 6.2.2 focuses on extending and adapting the proposed methods to other areas of information retrieval.

6.2.1 Customer Support

Continuing the work on the customer support with templates is imperative, in order to bring more attention to an area of interest that is mostly unexplored, despite offering new and important challenges not found on the currently considered information retrieval (IR) scenarios, and with any improvements having positive real-world implications.

First of all, despite the noticeable improvements, some of the techniques proposed remain mostly unexplored. In the case of the proposed loss function, despite containing four independently adjusted terms, exploration mostly focused on testing different combinations but failed to consider different weighting schemes, given the high costs associated with tuning 3 independent hyper-parameters (i.e., assuming the weight of the forth term as the complement to the others). Even so, the consistent performance gains, suggest that tuning may be key in achieving higher performances. Still on the topic, Ren et al. (2021) suggested a training scheme with 2 phases, the first considering extended similarity relations and the second simply query-centric, showing the latter to be crucial for optimizing the model to the retrieval task. A similar training scheme could be applied here, by a first-stage that enforces all relations, possibly even considering an extended dataset (Ren et al., 2021), and a second stage that focuses exclusively on the retrieval task. Alternatively, instead of two separate training stages, one could consider dynamic loss term weights, that vary throughout training (e.g., gradually reducing the weights of the loss terms misaligned with the retrieval task). In-batch top-k sampling is another technique that could be explored further, specifically by considering different values of k for each of the loss terms, as they're likely inde-

pendently optimized. Similarly to scheduling the loss weights, dynamically varying k may also benefit the model, since as training progresses, and the model improves retrieval, the number of easy negatives within the batch increases, suggesting a smaller k, to hopefully filter them. This idea is inline with the Learning To Retrieve Zhan et al. (2021, 2020a) technique, albeit less aggressive.

As discussed in Section 3, one of the main benefits of considering the transformer framework lies in the ease of swapping the base model. Given the continuous development and improvement of pre-trained models, a fruitful line of future work lies on testing newer base models, likely to improve performance/efficiency at virtually no cost (i.e., other than training). Furthermore, preliminary tests with the recently developed `paraphrase-multilingual-MiniLM-L12-v2` model from the Sentence-Transformers library (Reimers and Gurevych, 2020), show modest performance improvements, despite having close to twice as fast inference speed and smaller vector representations (25% smaller).

Finally, although this thesis focused on the bi-encoder architecture, the strong performance behind the classifier baseline is notable, despite its relative simplicity. These characteristics make it not only a solid baseline, but a full-fledged model for the real-world scenario, under the right conditions (e.g., well established template collections, mostly static and with abundant training examples). As such, improving the classifier model should also be considered as a direction for future work. Here, just like with the bi-encoder architecture, changing the base model with newer developments is an obvious follow-up, very likely to produce results. Another possible line of work, more specific with the classifier architecture, lies in considering newer developments on loss functions for single-label classification, namely the asymmetric loss (Ben-Baruch et al., 2020). This loss tackles the common positive-negative imbalance, characteristic with classification problems with a large number of classes, as is the case of template classification, making it a great fit.

6.2.2 Beyond Customer Support

Although the proposed techniques were motivated by the challenges associated with customer support with templates, they may provide a useful contribution to works beyond this scenario.

Given the simplicity of in-batch top-k negatives, the technique can virtually be used in any retrieval scenario, without any adaptation (i.e., apart from tuning), although it's likely more useful when using shared negatives, such as the common in-batch negatives. A particularly interesting case to test is with cross-batch negatives Qu et al. (2021), where the massive batches are very likely to include an abundance of easy negatives, and may benefit heavily from the technique.

Extending PAIR with the consideration of other loss terms, may improve performance. Furthermore, just like in pair, a powerful cross-encoder model may be used to provide pseudo-labels, but for query pairs, allowing the formation o groups of queries that should have similar representations, akin to the assumption that queries from the same template should be similar.

The labelled in-batch negatives technique not only counters the challenges associated with the many-to-one relation between queries and templates, but it thrives on it, leveraging the fact the template label is informative enough to identify the positive and negative pairs. This makes it a great fit for IR scenarios such as FAQ retrieval or QA with query paraphrases, where many-to-one relations are commonly found. However, the idea fails when considering many-to-many relations, which are typical in common retrieval scenarios. Here, one could generate pseudo-labels that follow a many-to-one relation, by grouping clusters of similar queries or documents. Given this structure, training could follow two stages: (1) A coarse-grain stage that leverages the many-to-one structure to efficiently train to model to correctly group query and document representations, within the same cluster; (2) A fine-grained stage that learns to retrieve the correct documents, within each cluster (e.g., with ADORE (Zhan et al., 2021)).

Bibliography

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Ba, L. J. and Caruana, R. (2013). Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*.

Barkan, O., Razin, N., Malkiel, I., Katz, O., Caciularu, A., and Koenigstein, N. (2019). Scalable attentive sentence-pair modeling via distilled sentence embedding. *arXiv preprint arXiv:1908.05161*.

Beltagy, I., Lo, K., and Cohan, A. (2019). SciBERT: A pretrained language model for scientific text. *arXiv preprint arXiv:1903.10676*.

Ben-Baruch, E., Ridnik, T., Zamir, N., Noy, A., Friedman, I., Protter, M., and Zelnik-Manor, L. (2020). Asymmetric loss for multi-label classification. *arXiv preprint arXiv:2009.14119*.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48. Association for Computing Machinery.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Burges, C. J. (2010). From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581).

Campello, R. J. G. B., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., editors, *Advances in Knowledge Discovery and Data Mining*.

Caruana, R. (1997). Multitask learning. *Mach. Learn.*, 28(1):41–75.

Clark, K., Luong, M.-T., Le, Q. V., and Manning, C. D. (2020). ELECTRA: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167. Association for Computing Machinery.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020). Unsupervised cross-lingual representation learning at scale.

Craswell, N., Mitra, B., Yilmaz, E., Campos, D., and Voorhees, E. M. (2020). Overview of the TREC 2019 deep learning track. *arXiv preprint arXiv:2003.07820*.

Crestani, F., Lalmas, M., Van Rijsbergen, C. J., and Campbell, I. (1998). "is this document relevant?. . . probably": A survey of probabilistic models in information retrieval. *ACM Comput. Surv.*, 30:528–552.

Dai, Z. and Callan, J. (2019a). Context-aware sentence/passage term importance estimation for first stage retrieval. *arXiv preprint arXiv:1910.10687*.

Dai, Z. and Callan, J. (2019b). Context-aware sentence/passage term importance estimation for first stage retrieval.

Devlin, J. (2018). Multilingual BERT README.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North*.

Furnas, G. W., Landauer, T. K., Gomez, L. M., and Dumais, S. T. (1987). The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971.

Gao, L., Dai, Z., and Callan, J. (2020a). Understanding BERT rankers under distillation. *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval*.

Gao, L., Dai, Z., Chen, T., Fan, Z., Van Durme, B., and Callan, J. (2020b). Complementing lexical retrieval with semantic residual embedding. *arXiv preprint arXiv:2004.13969*.

Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., and Smith, N. A. (2020). Don't stop pretraining: adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*.

Han, S., Wang, X., Bendersky, M., and Najork, M. (2020). Learning-to-rank with BERT in TF-ranking. *arXiv preprint arXiv:2004.08476*.

Hardalov, M., Koychev, I., and Nakov, P. (2018). Towards automated customer support. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*. Springer.

Harman, D. (2019). Information retrieval: The early years. *Foundations and Trends® in Information Retrieval*, 13(5):425–577.

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

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257.

Humeau, S., Shuster, K., Lachaux, M.-A., and Weston, J. (2019). Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. *arXiv preprint arXiv:1905.01969*.

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

Kaelbling, L., Littman, M., and Moore, A. (1996). Reinforcement learning: A survey. *J. Artif. Intell. Res.*, 4:237–285.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and tau Yih, W. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kitaev, N., Kaiser, Ł., and Levskaya, A. (2020). Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Lai, G., Xie, Q., Liu, H., Yang, Y., and Hovy, E. (2017). RACE: Large-scale reading comprehension dataset from examinations. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 785–794.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2019). AlBERT: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Li, C., Yates, A., MacAvaney, S., He, B., and Sun, Y. (2020). PARADE: Passage representation aggregation for document reranking. *arXiv preprint arXiv:2008.09093*.

Lin, J., Ma, X., Lin, S.-C., Yang, J.-H., Pradeep, R., and Nogueira, R. (2021). Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Liu, L., Wang, H., Lin, J., Socher, R., and Xiong, C. (2019a). MKD: a multi-task knowledge distillation approach for pretrained language models. *arXiv preprint arXiv:1911.03588*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Luan, Y., Eisenstein, J., Toutanova, K., and Collins, M. (2020). Sparse, dense, and attentional representations for text retrieval. *arXiv preprint arXiv:2005.00181*.

MacAvaney, S., Nardini, F. M., Perego, R., Tonellotto, N., Goharian, N., and Frieder, O. (2020). Training curricula for open domain answer re-ranking. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mitra, B. and Craswell, N. (2018). An introduction to neural information retrieval. *Foundations and Trends® in Information Retrieval*, 13(1):1–126.

Morgan, W., Greiff, W., and Henderson, J. (2004). Direct maximization of average precision by hill-climbing, with a comparison to a maximum entropy approach. In *Proceedings of HLT-NAACL 2004: Short Papers*, pages 93–96. Association for Computational Linguistics.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*.

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

Nogueira, R., Jiang, Z., and Lin, J. (2020). Document ranking with a pretrained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713*.

Nogueira, R., Lin, J., and Epistemic, A. (2019a). From doc2query to docTTTTTquery. *Online preprint*.

Nogueira, R., Yang, W., Cho, K., and Lin, J. (2019b). Multi-stage document ranking with BERT. *arXiv preprint arXiv:1910.14424*.

Nogueira, R., Yang, W., Lin, J., and Cho, K. (2019c). Document expansion by query prediction. *arXiv preprint arXiv:1904.08375*.

Onal, K. D., Zhang, Y., Altingovde, I. S., Rahman, M. M., Karagoz, P., Braylan, A., Dang, B., Chang, H.-L., Kim, H., McNamara, Q., Angert, A., Banner, E., Khetan, V., McDonnell, T., Nguyen, A. T., Xu, D., Wallace, B. C., de Rijke, M., and Lease, M. (2018). Neural information retrieval: at the end of the early years. *Information Retrieval Journal*, 21(2):111–182.

Pasumarthi, R. K., Bruch, S., Wang, X., Li, C., Bendersky, M., Najork, M., Pfeifer, J., Golbandi, N., Anil, R., and Wolf, S. (2019). Tf-ranking. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

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*.

Qu, Y., Ding, Y., Liu, J., Liu, K., Ren, R., Zhao, W. X., Dong, D., Wu, H., and Wang, H. (2021). RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392.

Reimers, N. and Gurevych, I. (2020). Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Reimers, N., Gurevych, I., Reimers, N., Gurevych, I., Thakur, N., Reimers, N., Daxenberger, J., Gurevych, I., Reimers, N., Gurevych, I., et al. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Ren, R., Lv, S., Qu, Y., Liu, J., Zhao, W. X., She, Q., Wu, H., Wang, H., and Wen, J.-R. (2021). PAIR: Leveraging passage-centric similarity relation for improving dense passage retrieval. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*.

Robertson, S. (1977). The probability ranking principle in ir. *Journal of Documentation*, 33:294–304.

Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M. M., and Gatford, M. (1995). Okapi at trec-3. In *Overview of the Third Text REtrieval Conference (TREC-3)*, pages 109–126. Gaithersburg, MD: NIST.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4).

Rumelhart, D., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Salton, G. and Buckley, C. (1988). On the use of spreading activation methods in automatic information. In *Proceedings of the 11th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '88, page 147–160. Association for Computing Machinery.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

Simmons, R. F. (1965). Answering english questions by computer: A survey. *Commun. ACM*, 8:53–70.

Sun, S., Cheng, Y., Gan, Z., and Liu, J. (2019a). Patient knowledge distillation for BERT model compression. *arXiv preprint arXiv:1908.09355*.

Sun, Y., Wang, S., Li, Y., Feng, S., Chen, X., Zhang, H., Tian, X., Zhu, D., Tian, H., and Wu, H. (2019b). Ernie: Enhanced representation through knowledge integration. *arXiv preprint arXiv:1904.09223*.

Tang, R., Lu, Y., Liu, L., Mou, L., Vechtomova, O., and Lin, J. (2019). Distilling task-specific knowledge from BERT into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Taylor, W. L. (1953). "cloze procedure": A new tool for measuring readability. *Journalism Quarterly*, 30(4):415–433.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*.

Voorhees, E. M. (1994). Query expansion using lexical-semantic relations. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, page 61–69, Berlin, Heidelberg. Springer-Verlag.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018). GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Weston, J., Watkins, C., et al. (1999). Support vector machines for multi-class pattern recognition. In *Esann*, volume 99, pages 219–224.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xie, Y., Yang, W., Tan, L., Xiong, K., Yuan, N. J., Huai, B., Li, M., and Lin, J. (2020). Distant supervision for multi-stage fine-tuning in retrieval-based question answering. In *Proceedings of The Web Conference 2020*, WWW '20, page 2934–2940, New York, NY, USA. Association for Computing Machinery.

Xiong, L., Xiong, C., Li, Y., Tang, K.-F., Liu, J., Bennett, P. N., Ahmed, J., and Overwijk, A. (2021). Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *International Conference on Learning Representations*.

Xu, J. and Croft, W. B. (2000). Improving the effectiveness of information retrieval with local context analysis. *ACM Trans. Inf. Syst.*, 18(1):79–112.

Xu, J., He, X., and Li, H. (2020). Deep learning for matching in search and recommendation. *Foundations and Trends® in Information Retrieval*, 14(2–3):102–288.

Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2020). mt5: A massively multilingual pre-trained text-to-text transformer.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., and Le, Q. V. (2020). Xlnet: Generalized autoregressive pretraining for language understanding.

Yates, A., Nogueira, R., and Lin, J. (2021). Pretrained transformers for text ranking: Bert and beyond. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*.

Yih, S. W.-t., Toutanova, K., Platt, J., and Meek, C. (2011). Learning discriminative projections for text similarity measures. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics.

Zhan, J., Mao, J., Liu, Y., Guo, J., Zhang, M., and Ma, S. (2021). Optimizing dense retrieval model training with hard negatives. *arXiv preprint arXiv:2104.08051*.

Zhan, J., Mao, J., Liu, Y., Zhang, M., and Ma, S. (2020a). Learning to retrieve: How to train a dense retrieval model effectively and efficiently. *arXiv preprint arXiv:2010.10469*.

Zhan, J., Mao, J., Liu, Y., Zhang, M., and Ma, S. (2020b). RepBERT: Contextualized text embeddings for first-stage retrieval. *arXiv preprint arXiv:2006.15498*.

Zhang, K., Xiong, C., Liu, Z., and Liu, Z. (2020). Selective weak supervision for neural information retrieval. *Proceedings of The Web Conference 2020*.