ψ and η, and the aggregation function f through (deep) neural networks. Depending on the assumptions over the representation functions, the neural ranking models can be classified in interaction-focused models and representation-focused models. In interaction-focused models, the query-document representation function $\eta(q, d)$, taking into account the interaction between the query and document contents, is explicitly constructed and used as input to a deep neural network, or it is implicitly generated and directly used by a deep neural network. In representation-focused models, the query-document representation function $\eta(q, d)$ is not present; query and document representations $\phi(q)$ and $\psi(d)$ are computed independently by deep neural networks In the following, we discuss the main interaction-focused models for ad-hoc ranking (Section 2) and representation-focused models for queries and documents (Section 3).

2 Interaction-focused Systems

The interaction-focused systems used in neural IR model the word and n-gram relationships across a query and a document using deep neural networks. These systems receive as input both a query q and a document d, and output a query-document representation $\eta(q, d)$. Among others, two neural network architectures have been investigated to build a representation of these relationships: convolutional neural networks and transformers. Convolutional neural networks represent one of the first approaches in building joint representations of queries and documents, as discussed in Section 2.1. Transformers represent the major turning point in neural IR, as their application to textual inputs gave birth to pre-trained language models, presented in Section 2.2. In neural IR, pre-trained language models are used to compute query-document representations, and the two main transformer models used for this task are BERT and T5 illustrated in Sections 2.3 and 2.4, respectively. Section 2.5 describes how pre-trained language models are fine-tuned to compute effective query-document representations, and Section 2.6 briefly discusses how pre-trained language models can deal with long documents.

2.1 Convolutional Neural Networks

A convolutional neural network is a family of neural networks designed to capture local patterns in structured inputs, such as images and texts [LeCun and Bengio 1998]. The core component of a convolutional neural network is the *convolution* layer, used in conjunction with feed forward and pooling layers. A convolutional layer can be seen as a small linear filter, sliding over the input and looking for proximity patterns.

Several neural models employ convolutional neural networks over the interactions between queries and documents to produce relevance scores. Typically, in these models, the word embeddings of the query and document tokens are aggregated into an *interaction matrix*, on top of which convolutional neural networks are used to learn hierarchical *proximity patterns* such as unigrams, bigrams and so on. Then, the final top-level proximity patterns are fed into a feed forward neural network to produce the relevance score $s(q, d)$ between the query q and the document d, as illustrated in Figure 2.

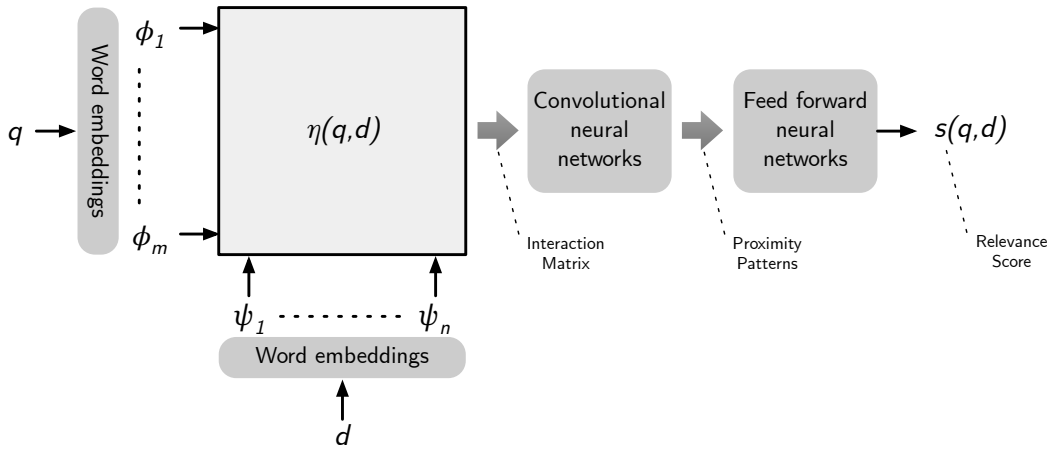

Figure 2: Scheme of an interaction-focused model based on convolutional neural networks.

The query q and the document d are tokenised into m and n tokens, respectively, and each token is mapped to a corresponding static word embedding. The interaction matrix $\eta(q, d) \in \mathbb{R}^{m \times n}$ is composed of the cosine similarities between a query token embedding and a document token embedding.

One of the first neural models leveraging the interaction matrix is the Deep Relevance Matching Model (DRMM) [Guo et al. 2016]). In DRMM, the cosine similarities of every query token w.r.t. the document tokens are converted into a discrete distribution using *hard bucketing*, i.e., into a query token histogram. Then the histogram of each query token is provided as input to a feed forward neural network to compute the final query token-document relevance score. These relevance scores are then aggregated through an IDF-based weighted sum across the different query terms. Instead of using hard bucketing, the Kernel-based Neural Ranking Model (KNRM) [Xiong et al. 2017] proposes to use Gaussian kernels to smoothly distribute the contribution of each cosine similarity across different buckets before providing the histograms with *soft bucketing* to the feed forward neural networks.

Both DRMM and KNRM exploit the interaction matrix, but they do not incorporate

any convolutional layer. In the Convolutional KNRM model (ConvKNRM) [Dai et al. 2018], the query and document embeddings are first independently processed through k convolutional neural networks, to build unigam, bigram, up to k-gram embeddings. These convolutions allow to build word embeddings taking into account multiple close words at the same time. Then, k^2 cosine similarity matrices are built, between each combination of query and document n-gram embeddings, and these matrices are processed with KNRM. In the Position-Aware Convolutional Recurrent Relevant model (PACRR) [Hui et al. 2017], the interaction matrix is processed through several convolutional and pooling layers to take into account words proximity. Convolutional layers are used also in other similar neural models [Fan et al. 2018b, Hu et al. 2014, Hui et al. 2018, Pang et al. 2016, 2017].

2.2 Pre-trained Language Models

Static word embeddings map words with multiple senses into an average or most common-sense representation based on the training data used to compute the vectors. The vector of a word does not change with the other words used in a sentence around it. The *transformer* is a neural network designed to explicitly take into account the context of arbitrary long sequences of text, thanks to a special neural layer called *self-attention*, used in conjunction with feed forward and linear layers. The self-attention layer maps input sequences to output sequences of the same length. When computing the i-th output element, the layer can access all the n input elements (bidirectional self-attention) or only the first i input elements (causal self-attention). A self-attention layer allows the network to take into account the relationships among different elements in the same input. When the input elements are tokens of a given text, a self-attention layer computes token representations that take into account their context, i.e., the surrounding words. In doing so, the transformer computes *contextualised word embeddings*, where the representation of each input token is conditioned by the whole input text.

Transformers have been successfully applied to different natual language processing tasks, such as machine translation, summarisation, question answering and so on. All these tasks are special instances of a more general task, i.e., transforming an input text sequence to some output text sequence. The *sequence-to-sequence* model has been designed to address this general task. The sequence-to-sequence neural network is composed of two parts: an *encoder* model, which receives an input sequence and builds a contextualised representation of each input element, and a *decoder* model, which uses these contextualised representations to generate a task-specific output sequence.

Both models are composed of several stacked transformers. The transformers in the encoder employ bidirectional self-attention layers on the input sequence or the output sequence of the previous transformer. The transformers in the decoder employ causal self-attention on the previous decoder transformer's output and bidirectional cross-attention of the output of the final encoder transformer's output.

In neural IR, two specific instances of the sequence-to-sequence models have been studied: encoder-only models and encoder-decoder models. *Encoder-only models* receive as input all the tokens of a given input sentence, and they compute an output contextualised word embedding for each token in the input sentence. Representatives of this family of models include BERT [Devlin et al. 2019], RoBERTa [Liu et al. 2019], and DistilBERT [Sanh et al. 2019]. *Encoder-decoder models* generate new output sentences depending on the given input sentence. The encoder model receives as input all the tokens of a given sequence and builds a contextualised representation, and the decoder model sequentially accesses these embeddings to generate new output tokens, one token at a time. Representatives of this family of models include BART [Lewis et al. 2020] and T5 [Raffel et al. 2020].

Sequence-to-sequence models can be trained as language models, by projecting with a linear layer every output embedding to a given vocabulary and computing the tokens probabilities with a softmax operation. The softmax operation is a function $\sigma : \mathbb{R}^k \rightarrow [0,1]^k$ that takes as input $k > 1$ real values $z_1, z_2, \ldots, z_k$ and transforms each input z_i as follows:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \tag{2}$$

The softmax operation normalises the input values into a probability distribution. In the context of deep learning, the inputs of a softmax operation are usually called *logits*, and they represent the raw predictions generated by a multi-class classification model, turned into a probability distribution over the classes by the softmax operation.

Depending on the training objective, a sequence-to-sequence model can be trained as a *masked language model* (MLM), as for BERT, or a *casual language model* (CLM), as for T5. MLM training focuses on learning to predict missing tokens in a sequence given the surrounding tokens; CLM training focuses on predicting the next token in an output sequence given the preceding tokens in the input sequence. In both cases, it is commonplace to train these models using massive text data to obtain *pre-trained language models*. In doing so, we allow the model to learn general-purpose knowledge about a language that can be adapted afterwards to a more specific *downstream task*. In this *transfer learning* approach, a pre-trained language model is used as initial model to fine-tune it on a domain-specific, smaller training dataset for the downstream target

task. In other words, *fine-tuning* is the procedure to update the parameters of a pre-trained language model for the domain data and target task.
As illustrated in Figure 3, pre-training typically requires a huge general-purpose training corpus, such as Wikipedia or Common Crawl web pages, expensive computation resources and long training times, spanning several days or weeks. On the other side, fine-tuning requires a small domain-specific corpus focused on the downstream task, affordable computational resources and few hours or days of additional training. Special cases of fine-tuning are *few-shot learning*, where the domain-specific corpus is composed of a very limited number of training data, and *zero-shot learning*, where a pre-trained language model is used on a downstream task that it was not fine-tuned on.

Figure 3: Transfer learning of a pre-trained language model to a fine-tuned language model.

In neural IR, the interaction-focused systems that use pre-trained language models are called *cross-encoder* models, as they receive as input a pair (q, d) of query and document texts. Depending on the type of sequence-to-sequence model, different cross-encoders are fine-tuned in different ways, but, in general, they aim at computing a relevance score $s(q, d)$ to rank documents w.r.t. a given query. In the following, we illustrate the most common cross-encoders leveraging both encoder-only models (Sec. 2.3) and encoder-decoder models (Sec. 2.4).

2.3 Ranking with Encoder-only Models

The most widely adopted transformer architecture in neural IR is BERT, an encoder-only model. Its input text is tokenised using the WordPiece sub-word tokeniser [Wu et al. 2016]. The vocabulary $\mathcal{V}$ of this tokeniser is composed of 30,522 terms, where the uncommon/rare words, e.g., `goldfish`, are splitted up in sub-words, e.g., `gold##` and `##fish`. The first input token of BERT is always the special `[CLS]` token, that stands for "classification". BERT accepts as input other special tokens, such as `[SEP]`, that denotes the end of a text provided as input or to separate two different texts provided as a single input. BERT accepts as input at most 512 tokens, and produces an output embedding in $\mathbb{R}^\ell$ for each input token. The most commonly adopted BERT version

is BERT base, which stacks 12 transformer layers, and whose output representation space has $\ell = 768$ dimensions.

Nogueira and Cho [2019] and MacAvaney et al. [2019] illustrated how to fine-tune BERT as a cross-encoder[1], in two slightly different ways. Given a query-document pair, both texts are tokenised into token sequences $q_1, \ldots, q_m$ and $d_1, \ldots, d_n$. Then, the tokens are concatenated with BERT special tokens to form the following input configuration:

$$\texttt{[CLS]}\ q_1 \cdots q_m\ \texttt{[SEP]}\ d_1 \cdots d_n\ \texttt{[SEP]}$$

that will be used as BERT input. In doing so, the self-attention layers in the BERT encoders are able to take into account the semantic interactions among the query tokens and the document tokens. The output embedding $\eta_{\texttt{[CLS]}} \in \mathbb{R}^\ell$, corresponding to the input [CLS] token, serves as a contextual representation of the query-document pair as a whole.

Nogueira et al. [2019a] fine-tune BERT on a binary classification task to compute the query-document relevance score, as illustrated in Figure 4. To produce the relevance score $s(q, d)$, the query and the document are processed by BERT to generate the output embedding $\eta_{\texttt{[CLS]}} \in \mathbb{R}^\ell$, that is multiplied by a learned set of classification weights $W_2 \in \mathbb{R}^{2\times\ell}$ to produce two real scores z_0 and z_1, and then through a softmax operation to transform the scores into a probability distribution p_0 and p_1 over the non-relevant and relevant classes. The probability corresponding to the relevant class, conventionally assigned to label 1, i.e., p_1, is the final relevance score.

MacAvaney et al. [2019] fine-tune BERT by projecting the output embedding $\eta_{\texttt{[CLS]}} \in \mathbb{R}^\ell$ through the learned matrix $W_1 \in \mathbb{R}^{1\times\ell}$ into a single real value z, that represents the final relevance score.

$$\begin{aligned} \eta_{\texttt{[CLS]}} &= \mathsf{BERT}(q, d) \\ [z_0, z_1] &= W_2 \eta_{\texttt{[CLS]}} \quad \text{or} \quad z = W_1 \eta_{\texttt{[CLS]}} \\ [p_0, p_1] &= \mathsf{softmax}([z_0, z_1]) \\ s(q, d) &= p_1 \quad \text{or} \quad s(q, d) = z \end{aligned} \tag{3}$$

In Section 2.5 we illustrate how a BERT-based cross-encoder is typically fine-tuned for ad-hoc ranking.

[1] Nogueira and Cho [2019] call it *monoBERT*, and MacAvaney et al. [2019] call it *vanilla BERT*.

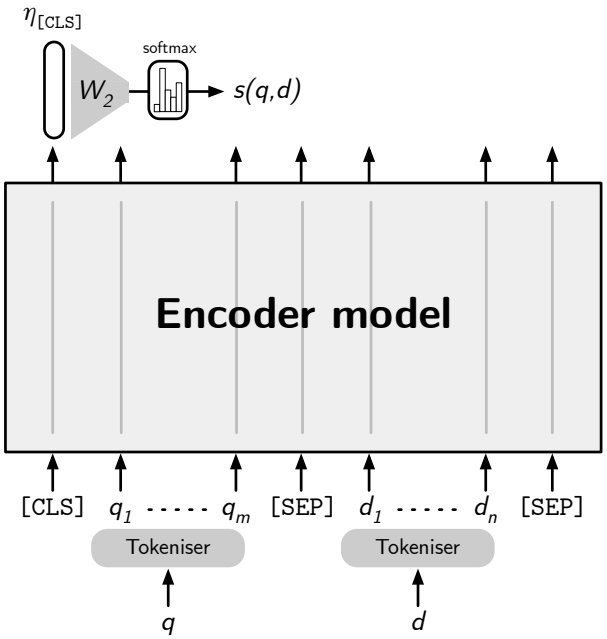

Figure 4: BERT classification model for ad-hoc ranking.

2.4 Ranking with Encoder-decoder Models

Instead of using an encoder-only transformer model to compute the latent representation of a query-document pair and to convert it into a relevance score, it is possible also to use an encoder-decoder model [Raffel et al. 2020] with *prompt learning*, by converting the relevance score computation task into a cloze test, i.e., a fill-in-the-blank problem. Prompting has been successfully adopted in article summarisation tasks [Radford et al. 2019] and knowledge base completion tasks [Petroni et al. 2019].

In prompt learning, the input texts are reshaped as a natural language template, and the downstream task is reshaped as a cloze-like task. For example, in topic classification, assuming we need to classify the sentence *text* into two classes c_0 and c_1, the input template can be:

$$\texttt{Input: } \textit{text} \quad \texttt{Class: [OUT]}$$

Among the vocabulary terms, two *label terms* w_0 and w_1 are selected to correspond to the classes c_0 and c_1, respectively. The probability to assign the input *text* to a class can be transferred into the probability that the input token `[OUT]` is assigned to the corresponding label token:

$$p(c_0|\textit{text}) = p(\texttt{[OUT]} = w_0|\texttt{Input: } \textit{text} \quad \texttt{Class: [OUT]})$$
$$p(c_1|\textit{text}) = p(\texttt{[OUT]} = w_1|\texttt{Input: } \textit{text} \quad \texttt{Class: [OUT]})$$

Nogueira et al. [2020] proposed a prompt learning approach for relevance ranking using a T5 model[2], as illustrated in Figure 5. The query and the document texts q and d are concatenated to form the following input template:

$$\texttt{Query: } q \texttt{ Document: } d \texttt{ Relevant: [OUT]}$$

An encoder-decoder model is fine-tuned with a downstream task taking as input this input configuration, and generating an output sequence whose last token is equal to `True` or `False`, depending on whether the document d is relevant or non-relevant to the query q.
The query-document relevance score is computed by normalising only the `False` and `True` output probabilities, computed over the whole vocabulary, with a softmax operation.

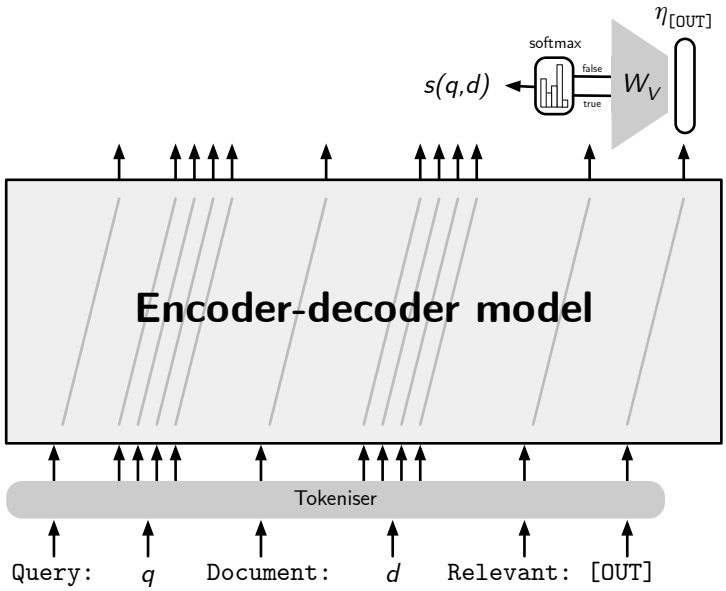

Figure 5: T5 model for ad-hoc ranking.

To produce the relevance score $s(q,d)$, the query and the document are processed by T5 to generate the output embedding $\eta_{\texttt{[OUT]}} \in \mathbb{R}^\ell$, that is projected by a learned set of classification weights $W_V \in \mathbb{R}^{|\mathcal{V}|\times\ell}$ over the vocabulary $\mathcal{V}$. The outputs $z_{\texttt{False}}$ and $z_{\texttt{True}}$ corresponding to the `False` and `True` terms, respectively, are transformed into a probability distribution with a softmax operation, to yield the required predictions $p_{\texttt{False}}$ and $p_{\texttt{True}}$ over the "non-relevant" and "relevant" classes. The prediction corresponding to

[2]Nogueira et al. [2020] call it *monoT5*.

the relevant class, i.e., $p_{\texttt{True}}$, is the final relevance score.

$$\begin{aligned} \eta_{\texttt{[OUT]}} &= \texttt{T5}(q,d) \\ [\ldots, z_{\texttt{False}}, \ldots, z_{\texttt{True}}, \ldots]^T &= W_V \eta_{\texttt{[OUT]}} \\ [p_{\texttt{False}}, p_{\texttt{True}}] &= \texttt{softmax}([z_{\texttt{False}}, z_{\texttt{True}}]) \\ s(q,d) &= p_{\texttt{True}} \end{aligned} \tag{4}$$

In the next section we discuss how a T5-based cross-encoder is typically fine-tuned for ad-hoc ranking.

2.5 Fine-tuning Interaction-focused Systems

As discussed in Section 2.2, the pre-trained language models adopted in IR require a fine-tuning of the model on a specific downstream task. Given an input query-document pair (q,d) a neural IR model $\mathcal{M}(\theta)$, parametrised by θ, computes $s_\theta(q,d) \in \mathbb{R}$, the score of the document d w.r.t. the query q. We are supposed to predict $y \in \{+,-\}$ from $(q,d) \in \mathcal{Q} \times \mathcal{D}$, where $-$ stands for non-relevant and $+$ for relevant.
This problem can be framed as a binary classification problem. We perform the classification by assuming a joint distribution p over $\{+,-\} \times \mathcal{Q} \times \mathcal{D}$ from which we can sample correct pairs $(+,q,d) \equiv (q,d^+)$ and $(-,q,d) \equiv (q,d^-)$. Using sampled correct pairs we learn a score function $s_\theta(q,d)$ as an instance of a metric learning problem [Xing et al. 2002]: the score function must assign a high score to a relevant document and a low score to a non-relevant document, as in Eq. (3) and Eq. (4). Then we find θ^* that minimises the (binary) cross entropy l_{CE} between the conditional probability $p(y|q,d)$ and the model probability $p_\theta(y|q,d)$:

$$\theta^* = \arg\min_\theta \mathbb{E}\Big[l_{CE}(y,q,d)\Big] \tag{5}$$

where the expectation is computed over $(y,q,d) \sim p$, and the cross entropy is computed as:

$$l_{CE}\big(y,(q,d)\big) = \begin{cases} -\log\big(s_\theta(q,d)\big) & \text{if } y = + \\ -\log\big(1 - s_\theta(q,d)\big) & \text{if } y = - \end{cases} \tag{6}$$

Typically, a dataset $\mathcal{T}$ available for fine-tuning pre-trained language models for relevance scoring is composed of a list of triples (q,d^+,d^-), where q is a query, d^+ is a relevant document for the query, and d^- is a non-relevant document for the query. In this case, the expected cross entropy is approximated by the sum of the cross entropies

computed for each triple:

$$\mathbb{E}\Big[l_{CE}\big(y,(q,d)\big)\Big] \approx \frac{1}{2|\mathcal{T}|} \sum_{(q,d^+,d^-)\in\mathcal{T}} \Big(-\log\big(s_\theta(q,d^+)\big) - \log\big(1-s_\theta(q,d^-)\big)\Big) \quad (7)$$

In doing so, we do not take into account documents in the collection that are not explicitly labeled as relevant or non-relevant. This approach is limited to take into account positive and negative triples in a pairwise independent fashion. In Section 3.3 we will discuss a different fine-tuning approach commonly used for representation-focused systems, taking into account multiple non-relevant documents per relevant document.

2.6 Dealing with long texts

BERT and T5 models have an input size limited to 512 tokens, including the special ones. When dealing with long documents, that cannot be fed completely into a transformer model, we need to split them into smaller texts in a procedure referred to as *passaging*. Dai and Callan [2019a] propose to split a long document into overlapping shorter passages, to be processed independently together with the same query by a cross-encoder. During training, if a long document is relevant, all its passages are relevant, and vice-versa. Then, the relevance scores for each composing passage are aggregated back to a single score for the long document. In this scenario, the common aggregation functions are FirstP, i.e., the document score is the score of the first passage, MaxP, i.e., the document score is the highest score across all passages, and SumP, i.e., the document score is the sum of the scores of its passages. Alternatively, Li et al. [2020] generate the [CLS] output embedding for each passage to compute a query-passage representation for each passage. Then, the different passage embeddings are aggregated together to compute a final relevance score for the whole document using feed forward neural networks, convolutional neural networks or simple transformer architectures.

3 Representation-focused Systems

The representation-focused systems build up independent query and document representations, in such a way that document representations can be pre-computed and stored in advance. During query processing, only the query representation is computed, and the top documents are searched through the stored document representations. In doing so, representation-based systems are able to identify the relevant docu-