10. Neural Learning to Rank

Prof. Dr. Goran Glavaš
(Slides by Robert Litschko)

Center for AI and Data Science (CAIDAS)
Fakultät für Mathematik und Informatik
Universität Würzburg

After this lecture, you'll

- Have an overview of a range of Neural Rankers
- Understand Convolutional Neural Networks in the context of IR
- Understand how BERT is used in IR models

Outline

- Recap of lecture #9: Learning to rank principles
- Position-Aware Convolutional Relevance Matching Model (PACRR)
- Multi-stage ranking with BERT

Recap of the previous lecture

- Classification
  - **Q:** Why is text classification relevant for IR?
  - **Q:** What text representations can we use in text classification?
  - **Q**: Common classifiers to use with sparse/dense text representations?
- Clustering
  - **Q:** What are the use-cases for text/document clustering in IR?
  - **Q:** How do we represent documents for IR clustering?
  - **Q:** What are the components of (any) clustering algorithm?
- Learning to Rank
  - **Q:** What is learning to rank and how does it relate to multi-criteria ranking?
  - **Q:** What are the differences between pointwise, pairwise, and list-wise L2R?
  - **Q:** Advantages and shortcomings of different L2R strategies?

Learning to Rank

- So far, each IR model was ranking the documents according to a **single** similarity function between the document and the query
  - VSM: cosine between the (sparse) TF-IDF vectors of the document and query
  - Latent/semantic IR: cosine between dense semantic vectors
  - Probabilistic IR: P(d, q | relevance)
  - Language modelling for IR: P(q | d)

- **Idea**: Combine different similarity scores as features of a supervised model (traditional), or learn to match documents based on latent features (neural)

$$\vec{f}(d,q) = \begin{pmatrix} VSM_q(d) \\ P(q|d) \\ \text{Jaccard(qterms, dterms)} \end{pmatrix}$$

Learning to Rank

- Learning to rank is a supervised information retrieval paradigm that
  - Describes instances of document-query pairs (d, q) with a range of features
  - Learns (with some ML algorithm) the mapping between these features and relevance

- Three different learning-to-rank approaches:
  1. **Point-wise approach**
     - Classify a single document-query (d, q) pair for relevance
  2. **Pair-wise approach**
     - Classify, for a pair of documents, which one is more relevant for the query, i.e., whether $r(d_1, q) > r(d_2, q)$ or $r(d_1, q) < r(d_2, q)$
  3. **List-wise approach**
     - Classify the whole ranking as either correct or wrong

Learning to Rank

- **Point-wise** learning to rank
  - Train a supervised classifier that for a given query q classifies each document as relevant or non-relevant
  - Binary classification task: document is either relevant or non-relevant
  - Training instances:
    - Query-document pairs (q, d) with relevance annotations

- Issues with point-wise learning to rank
  - Do not care about absolute relevance, but relative order of documents by relevance
  - If pairs (q, d_1) and (q, d_2) are classified as relevant, which document to rank higher?
    - Supervised classifiers usually have confidence/probability scores assigned to predictions
    - Rank d_1 higher than d_2 if the classifier is more confident about relevance of pair (q, d_1)

Learning to Rank

- **Pair-wise** learning to rank
  - Train a supervised classifier that for a given query q and two documents d_1 and d_2 predicts which document is more relevant for the query
  - Binary classification task:
    - Class 1: „d_1 more relevant than d_2"
    - Class 2: „d_1 less relevant than d_2"
  - Training instances:
    - Triples (q, d_1, d_2) consisting of queries and document pairs
    - We may need comparison features – compare d_1 and d_2 with respect to q
      - E.g., binary feature: $VSM(q, d_1) > VSM(q, d_2)$
    - Generating gold labels from relevance annotations:
      - For query q we have: d_1(r), d_2(nr), d_3(r), d_4(nr)
      - We create the following training instances:
        - $\{(q, d_1, d_2), 1\}$, $\{(q, d_1, d_4), 1\}$, $\{(q, d_2, d_3), 2\}$, $\{(q, d_3, d_4), 1\}$

Learning to Rank

- Issues with pair-wise learning to rank
  - If we don't use comparison features (but direct similarities of d1 and d2 with q as features), the model may not generalize well for new queries!

  - We only obtain independent pair-wise decisions
  - **Q:** What if pair-wise decisions are mutually inconsistent?
    - E.g., (q, d1, d2) -> 1, (q, d2, d3) -> 1, (q, d1, d3) -> 2

  - We need an additional postprocessing step
    - To turn the sorted pairs into a ranking, i.e., partial ordering into global ordering
    - Inconsistencies need to be resolved
      - E.g., In a set of conflicting decisions, the one with the lowest classifier confidence is discarded

  - Another issue: we effectively treat pairs from the bottom of ranking same as those from the top of the ranking (and eval. metrics don't treat them equally!)

Learning to Rank

- List-wise ranking approach
  - Instead of learning decisions for individual documents or pairs of documents, learn to classify entire rankings as correct or wrong
  - Training instances: query and an entire ranking of documents $(q, d_1, \ldots, d_n)$
  - Binary classification task:
    - Class 1: the ranking $(q, d_1, \ldots, d_n)$ is correct
    - Class 2: the ranking $(q, d_1, \ldots, d_n)$ is incorrect
  - Advantage: optimization criteria for the machine learning algorithm can be the concrete IR evaluation metric we're looking to optimize
- Issues with list-wise approach
  - Entire ranking just one training instance
    - Difficult to collect many positive training instances
  - Informative features for the whole ranking are difficult to design

Outline

- Recap of lecture #9: Learning to rank principles
- Position-Aware Convolutional Relevance Matching Model (PACRR)
- Multi-stage ranking with BERT

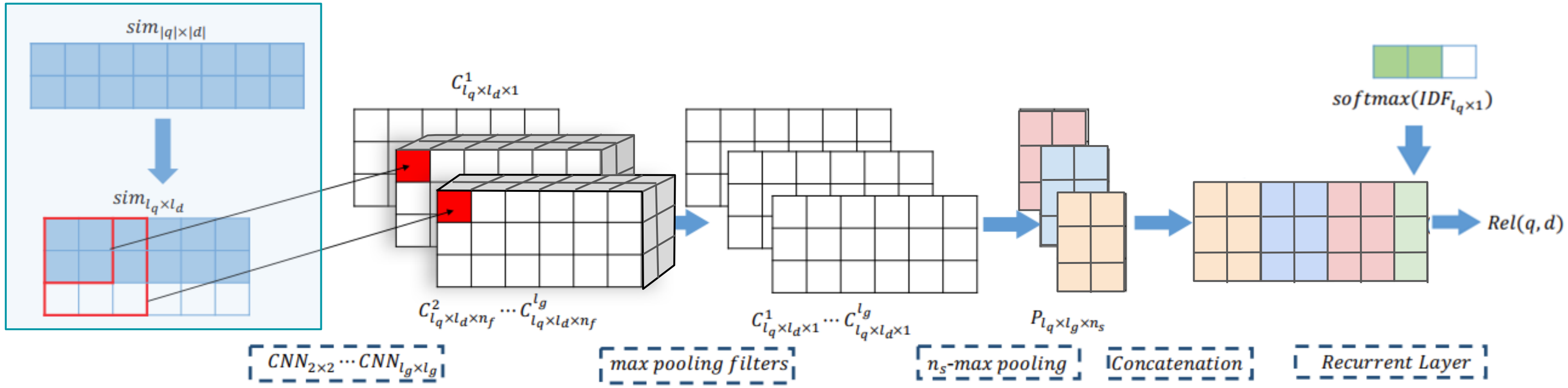
Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)

Position-Aware Convolutional Recurrent Relevance (PACRR):

- Position-Aware: Model learns to match n-gram patterns
- Convolutional: Architecture uses a CNN to learn features
- Recurrent: Long Short-Term Memory Network (LSTM) to summarize features into matching score

Uses Inverse Document Frequency (IDF) as feature, rather than scaling weights.

Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)



Input representation

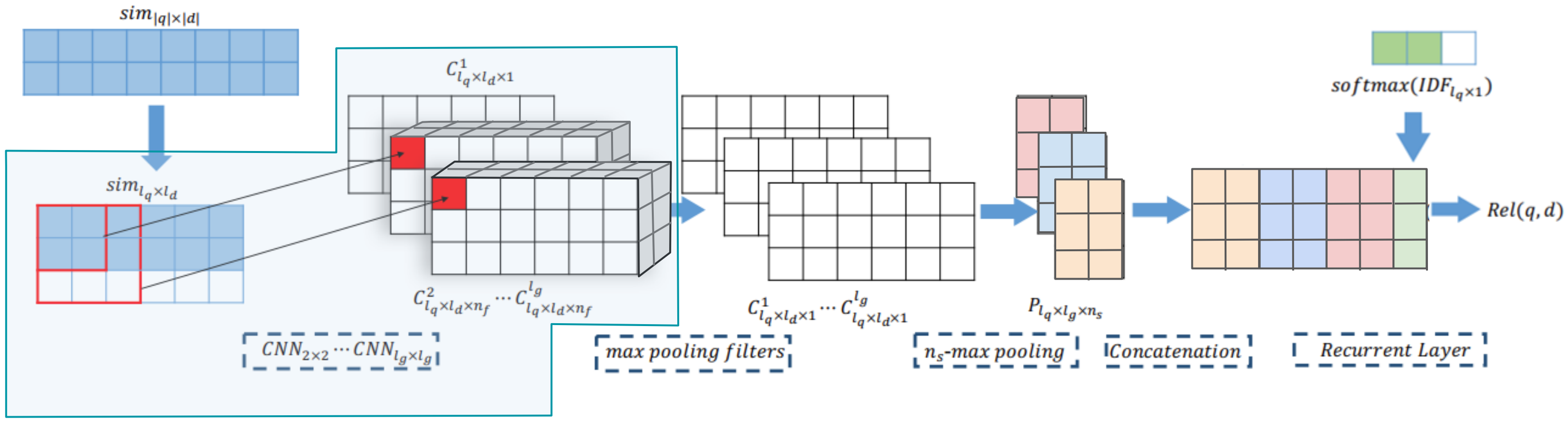
Term similarity matrix $sim_{|q|\times|d|}$ (cutting / zero-padding to max. seq. len. $sim_{l_q \times l_d}$)

Each element $sim_{i,j}$ describes semantic similarity (cosine) between embeddings of word i and word j

Captures unigram matching signals

Image based on Hui et al. 2017

Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)



N-Gram Matching Signals: Modelling Positional Information

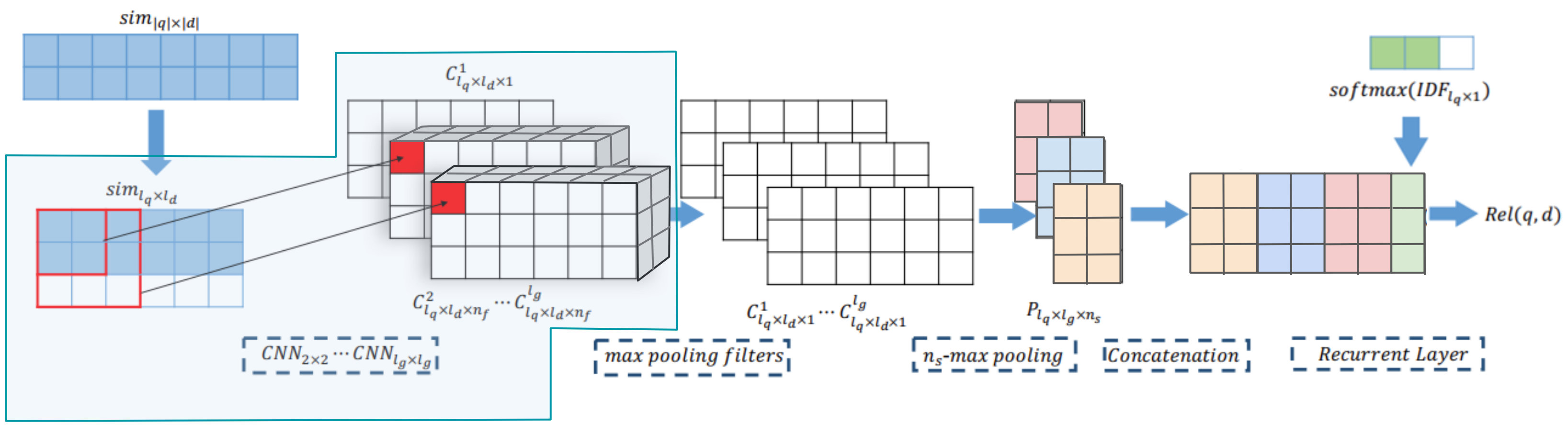
Apply multiple CNN layers $CNN_{2\times 2} \ldots CNN_{l_g\times l_g}$ to learn to match different n-gram sizes.

Each layer applies n_f different filters to learn different matching patterns (cf. next slide).

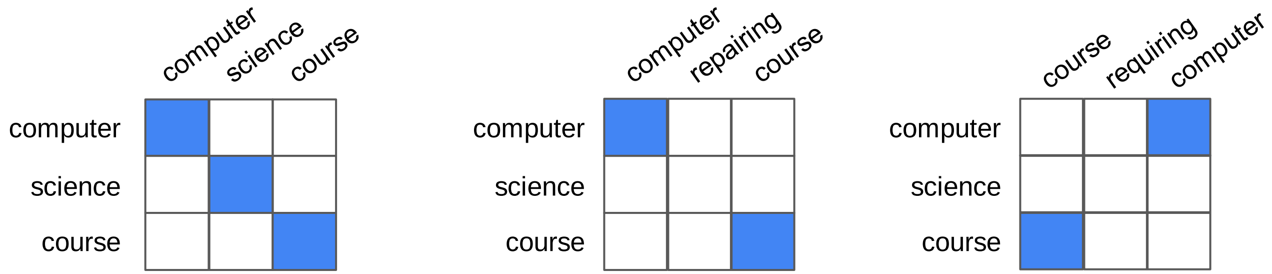
Sliding each convolutional filter along the similarity matrix leads to $l_g - 1$ feature tensors $C_{l_q\times l_d\times n_f}$.

Image based on Hui et al. 2017

Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)

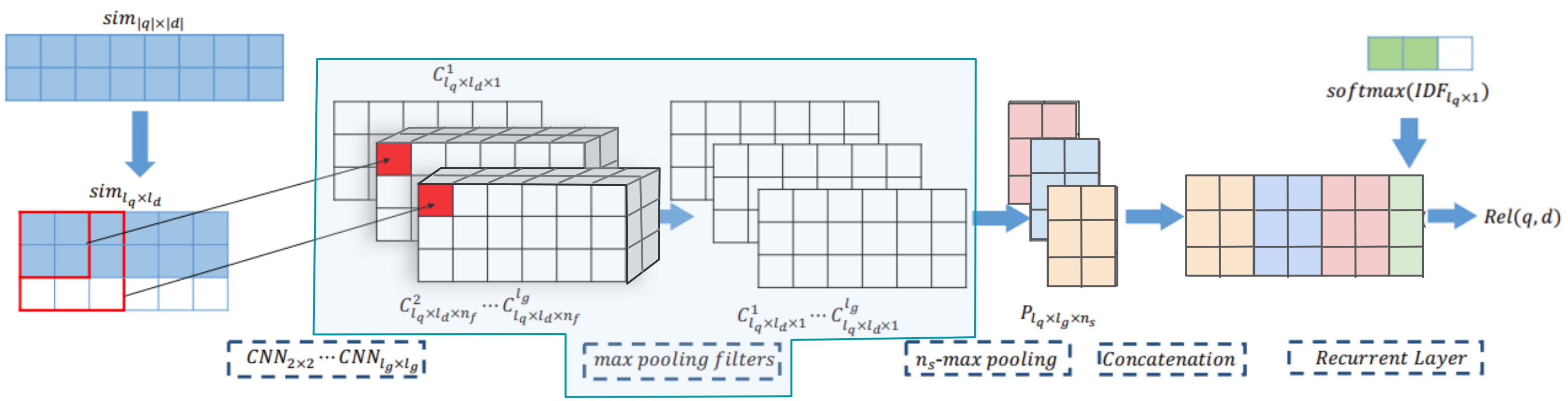


Examples of n-Gram Matching Signals (Modelling Positional Information) $n_f = 3$



Example taken from https://vimeo.com/238235171

Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)



1-max pooling among filters

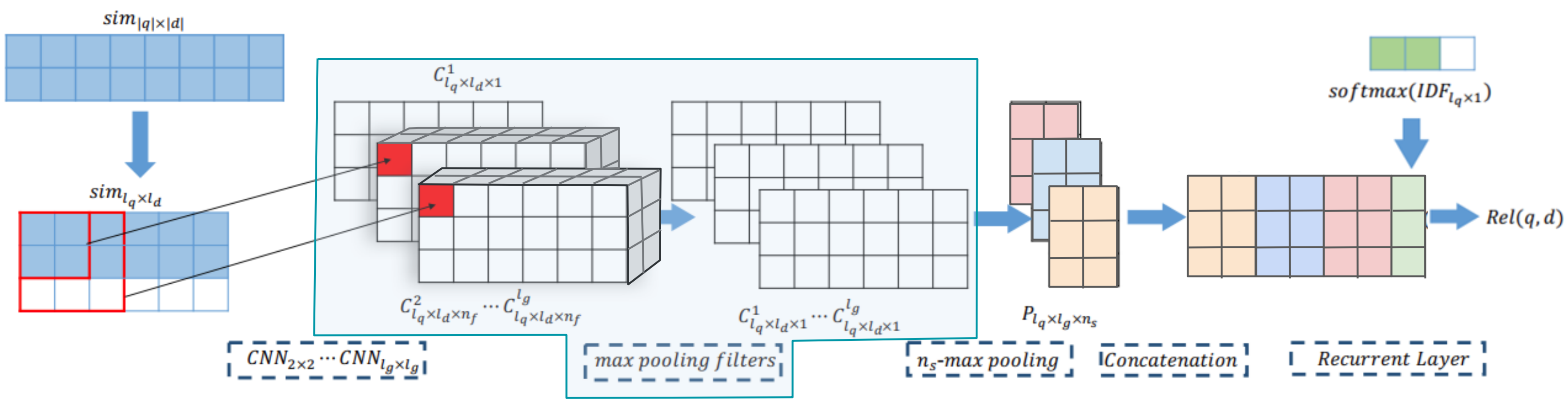
Accross all filters, keep for each n-gram kernel only the strongest signals.

Reduces feature tensors from $C^{l_g}_{l_q \times l_d \times n_f}$ to $C^{l_g}_{l_q \times l_d \times 1}$ (one "feature image" for each n-gram).

Assumes there is only one true matching pattern in a given $n \times n$ window.

Image based on Hui et al. 2017

Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)



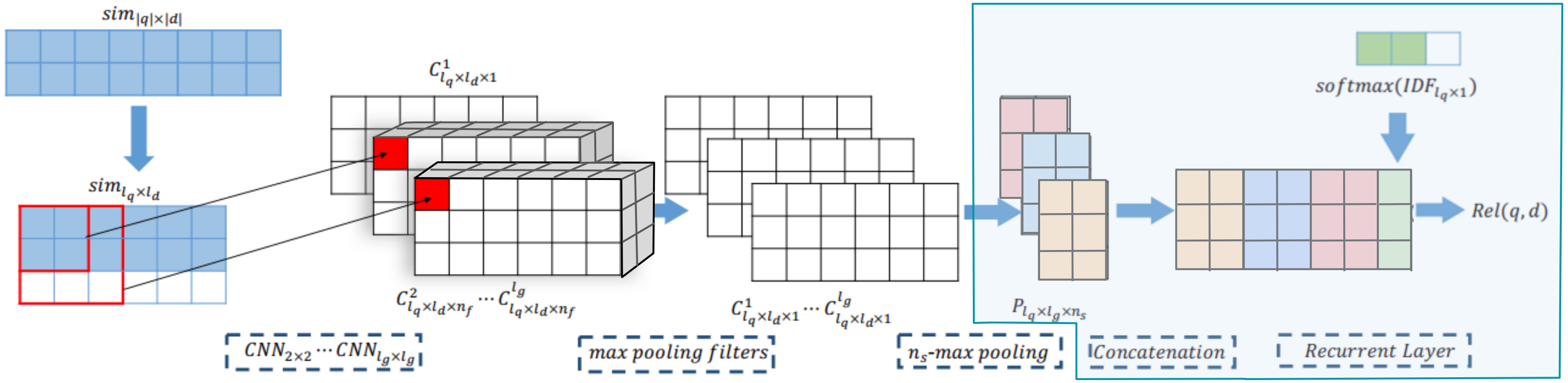
n_s -max pooling along query dimension

Keep n_s strongest similarity signals (example above: $n_s = 2$).

Resulting tensor $P_{l_q \times n_s \times l_g}$ contains n_s strongest signals for each query term and n-gram size across all filters.

Image based on Hui et al. 2017

Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)



Final feature matrix and relevance score

Feature tensor $P_{l_q \times n_s \times l_g}$ is reshaped (flattened) into matrix $P_{l_q \times (n_s \cdot l_g)}$ (cf. example above).

Query terms' normalized IDF-values are appended (concatenated).

Final representation is processed by a recurrent model (LSTM) to produce final relevance score.

Image based on Hui et al. 2017

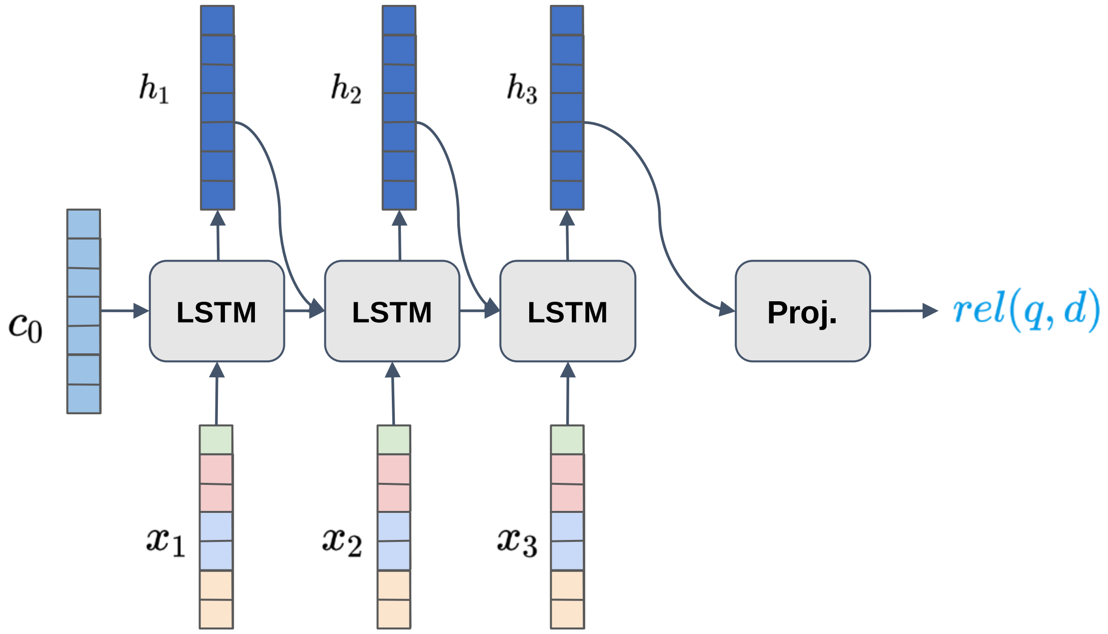
Position-Aware Convolutional Recurrent Relevance Matching (Hui et al. 2017)

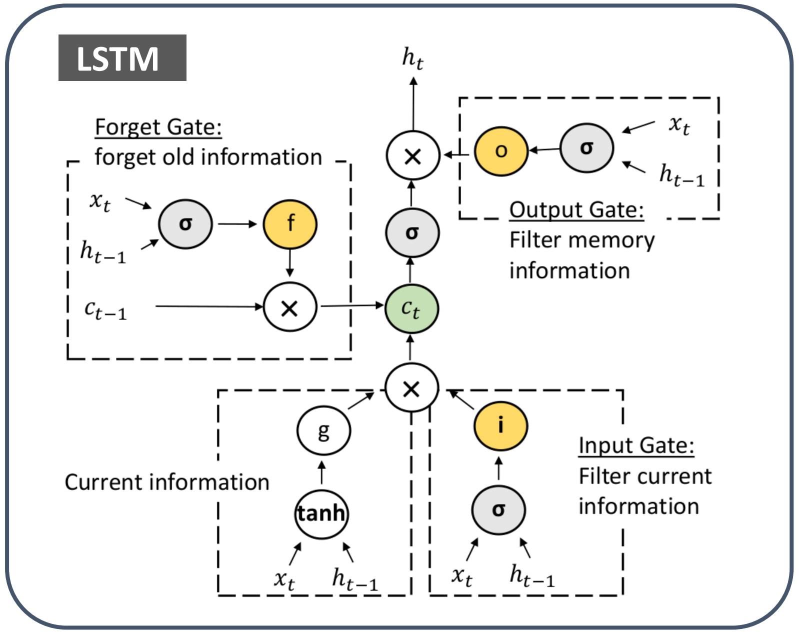
Recurrent Layer summarizes sequence of query token-wise matching information into final feature vector

Uses Long Shot-Term Memory Network (LSTM): RNN that maintains internal memory and summarizes sequential information

Relevance score is computed by (linearly) projecting feature vector to 1-d space.

Ranking loss same as in DRMM (pair-wise ranking): $\mathcal{L}(q, d^+, d^-; \Theta) = max(0, 1 - rel(q, d^+) + rel(q, d^-))$





Outline

- Recap of lecture #9: learning to rank principles
- Position-Aware Convolutional Relevance Matching Model (PACRR)
- Multi-stage ranking with BERT

- We have access to **enormous amounts of raw unannotated texts** (at least for major languages)
- **Can we somehow pre-train the encoder using raw text?**
  - Yes, via **language modeling**! Task is to predict the word from the text based on the encoding of the surrounding context
- **LM-pretraining**
  - Causal (unidirectional) language modeling: **GPT (1, 2, 3, ...)**
  - Bidirectional language modeling: **ELMo**
  - Masked language modeling: **BERT**

Bidirectional Transformer (BERT)

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, January). **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. *NAACL 2019*.

- **Pretraining:** Masked language modeling, MLM (and next sentence prediction, NSP)
- **Encoder architecture**: deep Transformer (attention-based) network
- Encoder's parameters (learned in pre-training) further updated in task-specific training (aka **fine-tuning**)
- After task-specific training (aka **fine-tuning**), we have a **task-specific encoder**

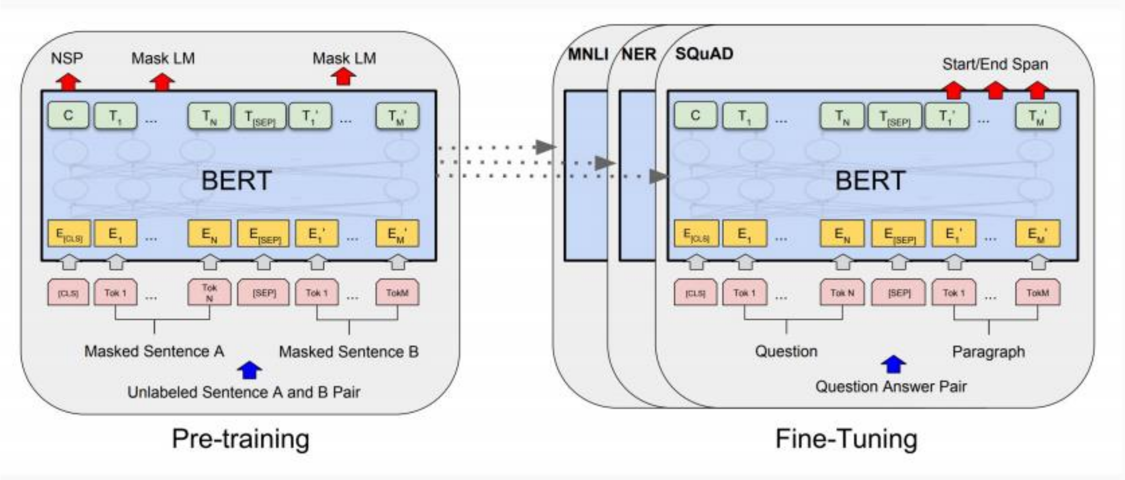


Image from [Devlin et al., NAACL 19]

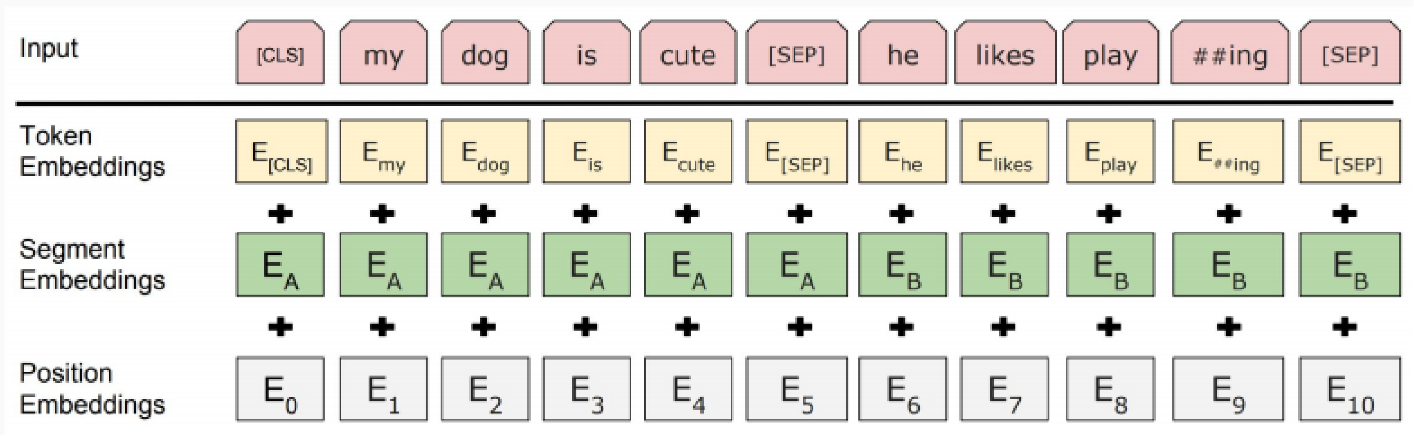
Bidirectional Transformer (BERT)

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. *NAACL 2019*.

- **Training instances**: sentence pairs, with special tokens inserted
  - Ca. 15% of tokens masked out (replaced with [MASK] token)
  - Sequence start token [CLS] and sentence separation token [SEP]

- **Pretraining:** two **self-supervised** objectives
  - Masked language modeling, MLM (predict the masked token from the context)
  - Next sentence prediction, NSP (if sentences adjacent or not)

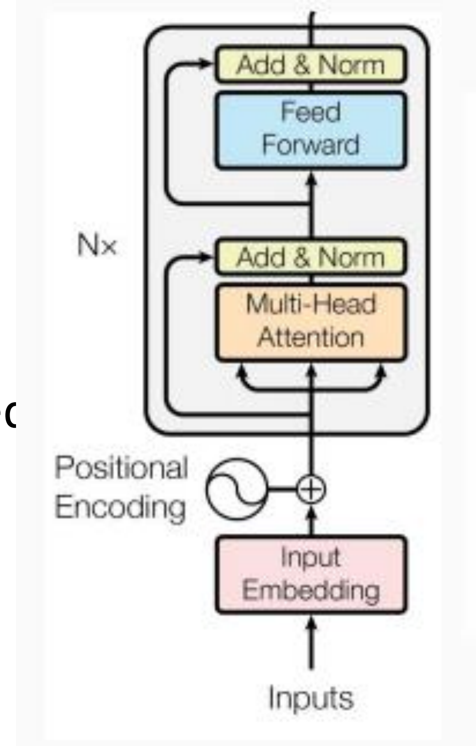
Image from [Devlin et al., NAACL 19]



Bidirectional Transformers for LU (BERT)

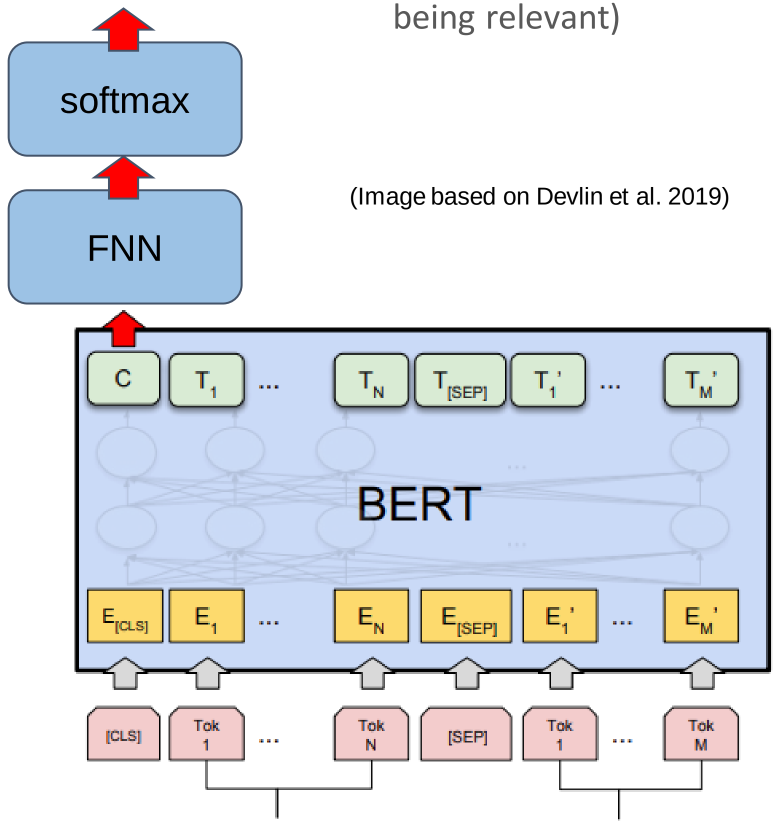
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, January). **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. *NAACL 2019*.

- **Encoder architecture**: deep Transformer (attention-based) network
  - Deep architecture consisting of N transformer layers
  - Each transformer layer:
    - Multi-head attention layer
    - Feed-forward layers
    - Residual connection (representation before the layer added to the result of the layer)
    - Layer normalization
  - All parameters of the Transformer: θ_{TRANS}



Multi-stage ranking with BERT (Nogueira et al. 2019)

$s_i = rel(q, d)$ (probability of document d_i being relevant)



(Image based on Devlin et al. 2019)

[CLS] Query q [SEP] Document d

BERT as a point-wise ranker (monoBERT): binary relevance classifier

- Feeds concatenation of query and document to BERT
  - Truncate query to at most 64 tokens
  - Concatenate query with document ([SEP]-token)
  - Truncate whole sequence to 512 tokens (max. seq. length)
- Obtain representation representation of [CLS]-token in last layer
- Feed [CLS] vector to single layered Feedforward Neural Network (FNN, binary classification model) to obtain relevance score

Optimize the following loss:

$$\mathcal{L}_{mono} = -\sum_{j \in J_{pos}} \log(s_j) - \sum_{j \in J_{neg}} \log(1 - s_j)$$

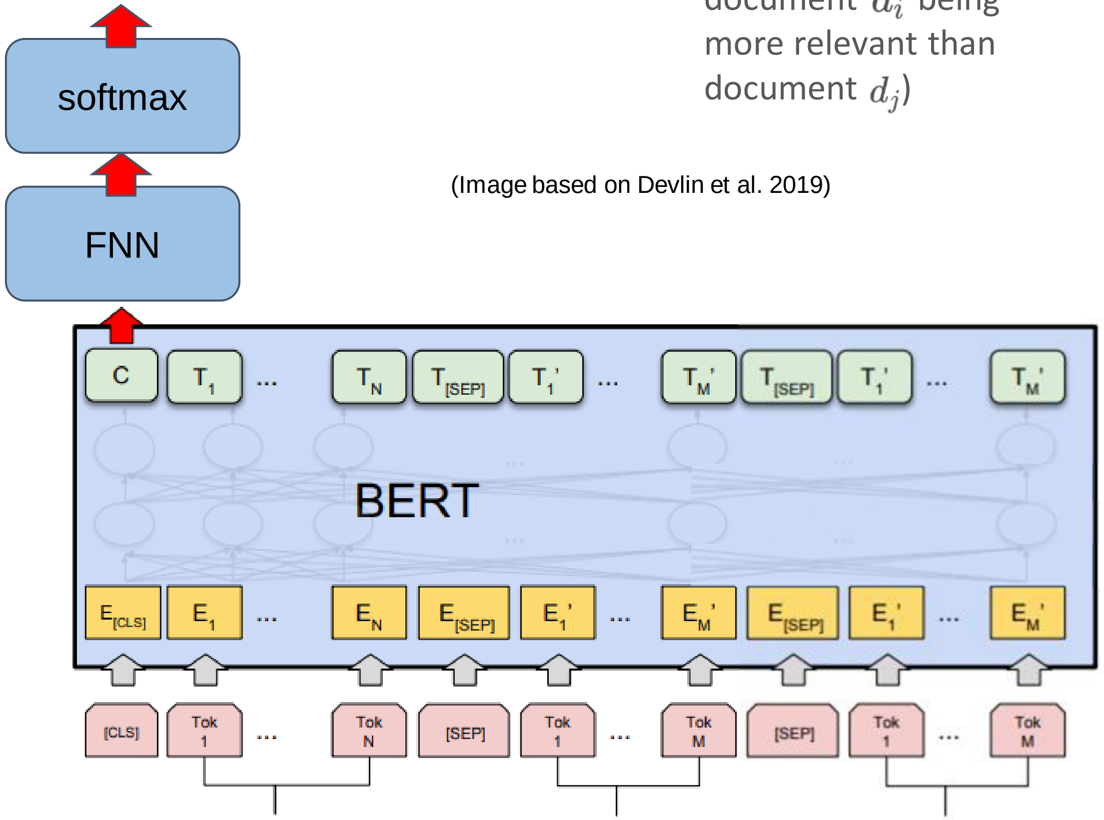
J_pos/neg = set of indexes of relevant/non-relevant documents

Retrieval: Rank documents by their probability of being relevant s_j

Multi-stage ranking with BERT (Nogieura et al. 2019)

$p_{i,j} = rel(q, d_i) > rel(q, d_j)$

(probability of document d_i being more relevant than document d_j)



(Image based on Devlin et al. 2019)

[CLS] Query q [SEP] Document d_i [SEP] Document d_j

BERT as a pair-wise ranker (duoBERT):

- Truncate the query, candididate document d_i and d_j to 62, 223 and 223 tokens respectively
- Concatenate query and document pair into single sequence
- For a candidate list of k_1 documents, compute $k_1(k_1-1)$ probabilities

Optimize the following loss:

$$\mathcal{L}_{duo} = -\sum_{i \in J_{pos}, j \in J_{neg}} \log(p_{i,j}) - \sum_{i \in J_{neg}, j \in J_{pos}} \log(1 - p_{i,j})$$

Retrieval:

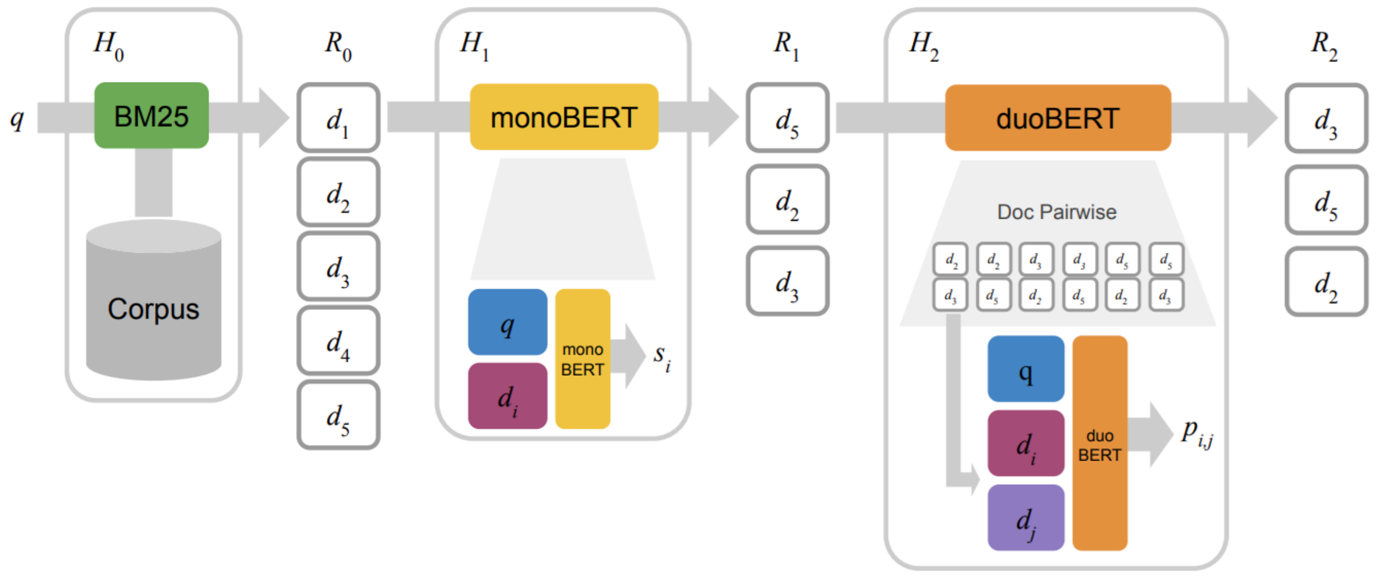
Aggregate pairwise scores $p_{i,j}$ into single score s_i

Set of all (other) document indexes in ranking R1: $J_i = \{0 \leq j \leq |R_1|, j \neq i\}$

Relevance score as **pair-wise agreement** that d_i is more relevant than the rest of the candidates (other aggregation methods possible too, cf. paper):

$$s_i = \sum_{j \in J_i} p_{i,j}$$

Multi-stage ranking with BERT (Nogueira et al. 2019)



Combining monoBERT and duoBERT into a multi-stage ranking architecture

Stage 1: Retrieve top-$k_0 = 1000$ documents using BM25 ($k_0 = 5$ in example above) → input to monoBERT

Stage 2: Re-rank top-$k_1 = 50$ documents with monoBERT ($k_1 = 3$ in example above) → input to duoBERT

Stage 3: Re-rank subset with duoBERT

Image source: Nogueira et al. 2019

Multi-stage ranking with BERT (Nogueira et al. 2019)

Summary

It’s common practice to use neural rankers for re-ranking, ranking the full collection would be too slow for practical purpose

Arranging retrieval in a multi-stage pipeline allows for trading off quality against latency by controlling admission of candidates at each stage

Target Corpus Pre-training (Masked Language Modelling on document collection) before training monoBERT/duoBERT improves results

Challenges for pair-wise ranking revisited:

1. We only obtain independent pair-wise decisions (inconsistent ranking): Aggregate (all) possible pair-wise agreements into relevance scores
2. We effectively treat pairs from the bottom of ranking same as those from the top of the ranking (and eval. metrics don’t treat them equally!): Neural model only re-ranks top k documents (ignore bottom of ranking)

Now you...

- Have an overview of a range of Neural Rankers
- Understand Convolutional Neural Networks in the context of IR
- Understand how BERT is used in recent IR models