

# Neural Networks for Information Retrieval

Sebastian Hofstätter

[sebastian.hofstaetter@tuwien.ac.at](mailto:sebastian.hofstaetter@tuwien.ac.at)

# Today

## Neural Networks for IR

1

### Workflow

- Connection to first stage ranker
- Training & Evaluation

2

### Models

- Differences to other NLP tasks
- MatchPyramid (inspired by image classification)
- KNRM, CONV-KNRM (Kernel-based)

# Disclaimer: Content based Neural IR models

---

- Learning-to-rank systems in production use external features\*
  - Click count, recency, author, personalization, etc ...
  - Use classical ML algorithms (SVMs, Trees, linear regression...)
- Deploying a re-ranker in production is a huge effort
  - Performance, Quality control, version management, changing indices, ...
- This lecture: Neural IR models = content based ad-hoc retrieval
  - We only use the content of the query and document
  - Can be thought of as another signal for a learning-to-rank system

\* As far as I know as of April 2019

# Desired properties of Neural IR models\*

---

- **Effective**

- They should work – duh! Otherwise what's the point of using them

- **Fast & Scalable**

- 10 to 100 ms time budget for the full re-ranking
- Potentially TB+ indices (this can vary a lot depending on the use case)

- **Interpretable**

- Search engines filter a lot of information – they act as a gate  
The reasons for including/excluding certain results needs to be explainable

\* Pretty much in line with other ML tasks

# Versatility of Neural Networks

---

- Neural network components are truly like LEGO bricks
- We can combine them however we want
  - As long as the tensor shapes match
  - And the network stays differentiable (when using gradient descent)
- They adapt quite easily to new tasks
  - In comparison to handcrafted feature engineering

# Beware! Neural Networks fail silently ...

---

- Training neural networks is hard
  - And far from the 50-line example on getting started pages of libraries
- Neural IR model training has even more special things to look out for
  - Wild west of a research field in comparison to computer vision or translation
- More information: <http://karpathy.github.io/2019/04/25/recipe/>
  - Might come in handy for the exercise :)

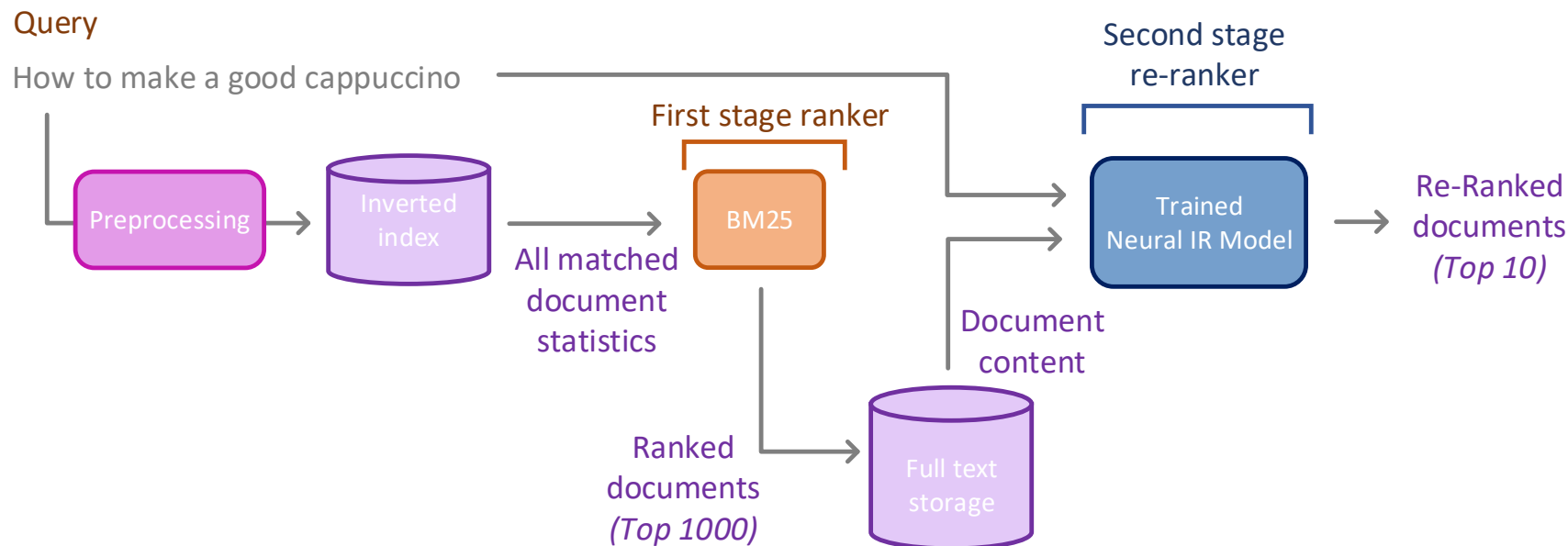
# Workflow

What's different for neural IR models?

# Neural IR Models

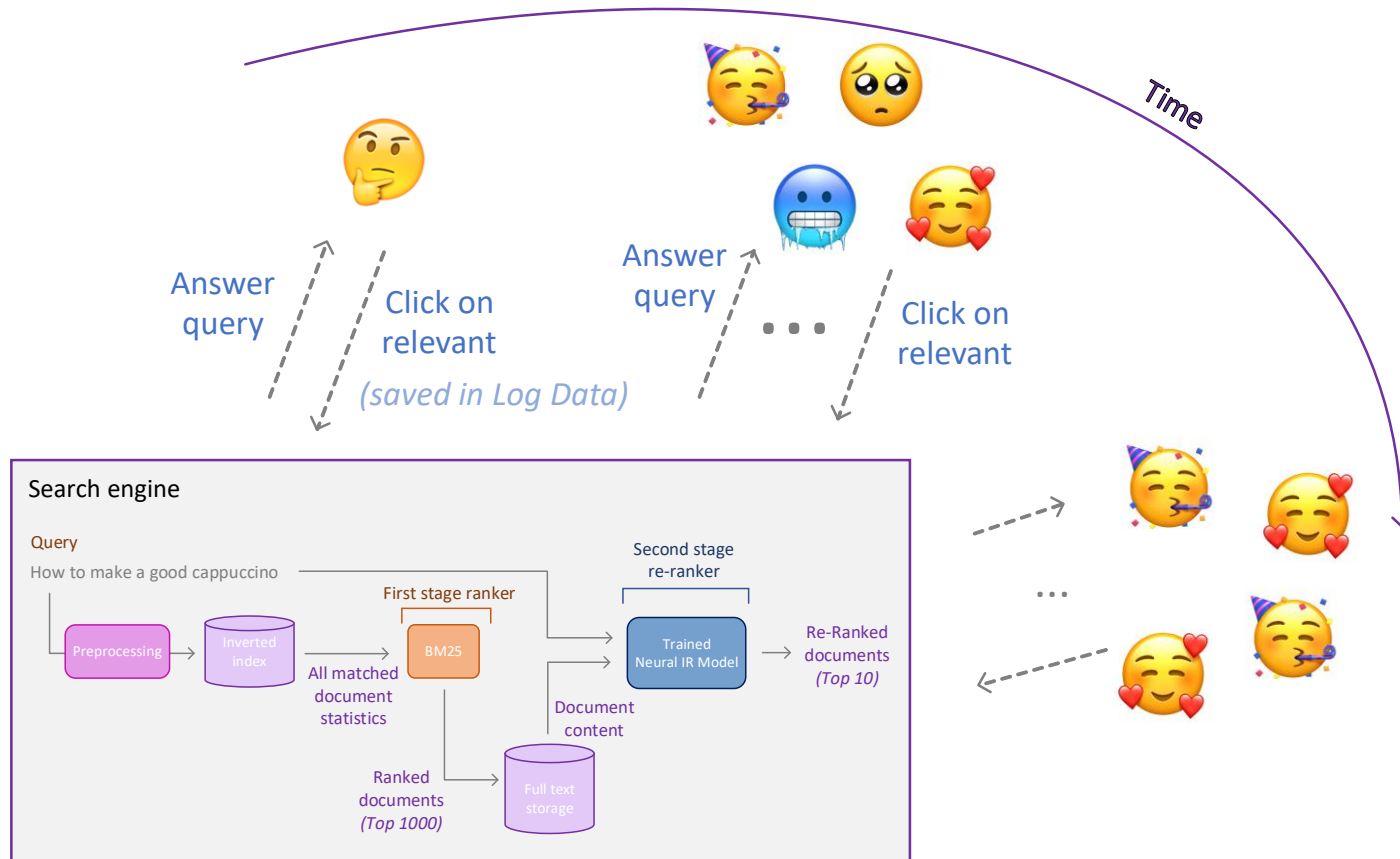
---

- Are re-rankers: They change the ranking of a pre-selected list of results
  - Same interface as classical ranking methods:  $score(q, d)$
- Query Workflow:





# Context

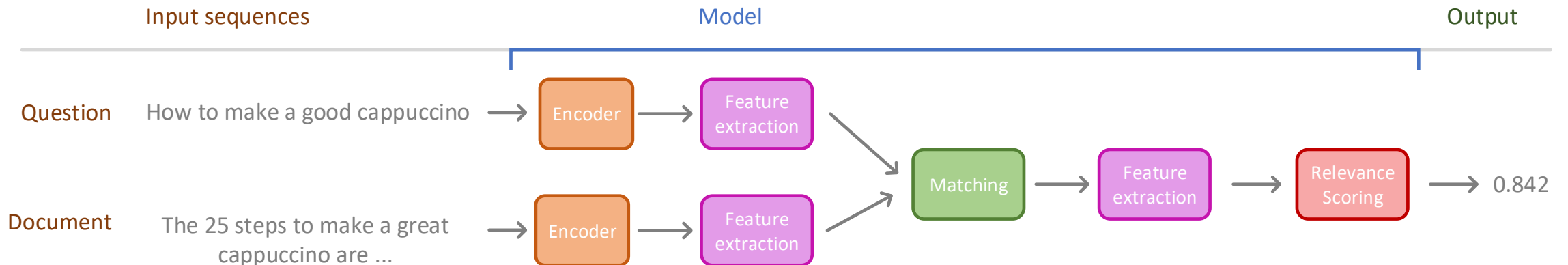


- Search engines have a lot of users
- Interaction is in two-ways:
  - Query results (search engine)
  - Activity data (users)
- Over time logs can be used to improve the search
- Neural IR model trained “offline” and is then swapped

# Inside Neural Re-ranking Models

---

- Core part of re-ranking models is a matching module
  - Operating on a word interaction level



# Training

---

- Training is independent from the rest of the search engine operations
- Neural IR models are typically trained with triples (pairwise +,-)
  - Triple: 1 query, 1 relevant + 1 non-relevant document
  - 2 forward passes: 1 relevant doc, 1 non-relevant doc
  - Loss function: Maximize margin between rel/non-rel document
  - Pytorch: `torch.nn.MarginRankingLoss(margin=1, reduction='elementwise_mean')`
- All model components are trained end-to-end

# Evaluation

---

- Scoring per tuple (1 query, 1 document)
- List of tuples is then sorted & evaluated with ranking metric per query (for example: MRR@10)
  - MRR@10 = Mean Reciprocal Rank, stop to look at position 10 or first relevant
- Mismatch: You can't really compare training loss and IR evaluation metric
  - Training loss is only good for checking at the beginning if your network is not completely broken :) – it should go down very quick and then not change

# MS MARCO Microsoft MACHine Reading COMprehension Dataset

---

- First re-ranking dataset with too much training data
  - “Too much” = models don’t benefit from more training data after some point
  - Scale of training & evaluation data now is an issue (luxury problem!)
- Real-world web search queries and passage-level answers from Bing
  - Released by Microsoft Research
- High-quality judgement labels
  - Human annotated
  - Only ~1 relevant judged document per query

# MS MARCO

---

- Training triples
  - **Query:** *what fruit is native to Australia*
  - **Relevant:** *Passiflora herbertiana. A rare passion fruit native to Australia. Fruits are green-skinned, ...*
  - **Non-Relevant:** *The kola nut is the fruit of the kola tree, a genus (Cola) of trees that are native to the tropical rainforests of Africa.*
- Evaluation tuples
  - **Ids:** 837202 1000252
  - **Query:** *what is the nutritional value of oatmeal*
  - **Document:** *Oats make an easy, balanced breakfast. One cup of cooked oatmeal contains about 150 calories, four grams of fiber (about half soluble and half insoluble), and six grams of protein. ...*

# Actual input & output

---

- 2 Tensors of word ids containing a batch of samples
  - Tensor = multidimensional array abstraction
  - Batch = For efficiency multiple samples are computed in parallel (on the GPU)
- Same dimension for all entries
  - Shorter sequences are padded with 0
- Query tokens:
  - Shape: [batch, query\_seq\_len]
- Document tokens:
  - [batch, doc\_seq\_len]

```
[[ 58,  83, 5401, 821,  18,   9,   3, 399],  
[ 48,  10,   3, 1310,   5, 2654, 16729,  0],  
[ 48,  10, 2654, 7545, 3700,   0,   0,  0],  
[ 128,  10, 7397, 170, 2652,   0,   0,  0]  
  
...
```

# Models

How are they actually implemented?



# The encoding layer

---

- Starting point for text processing neural network models
- Word token (id/piece/char based) to dense representation
  - Having word boundaries is important in IR
- Actual implementation or pre-trained-data easily swappable
  - Latest research papers: Evaluating 1 IR model with different encoding layers
- Usually shared between query & document
  - If the same word appears in the query and the document
    - it gets assigned the same vector\*

\* Does not hold for contextualized representation, i.e. ELMo/BERT (but keep it simple for a minute)

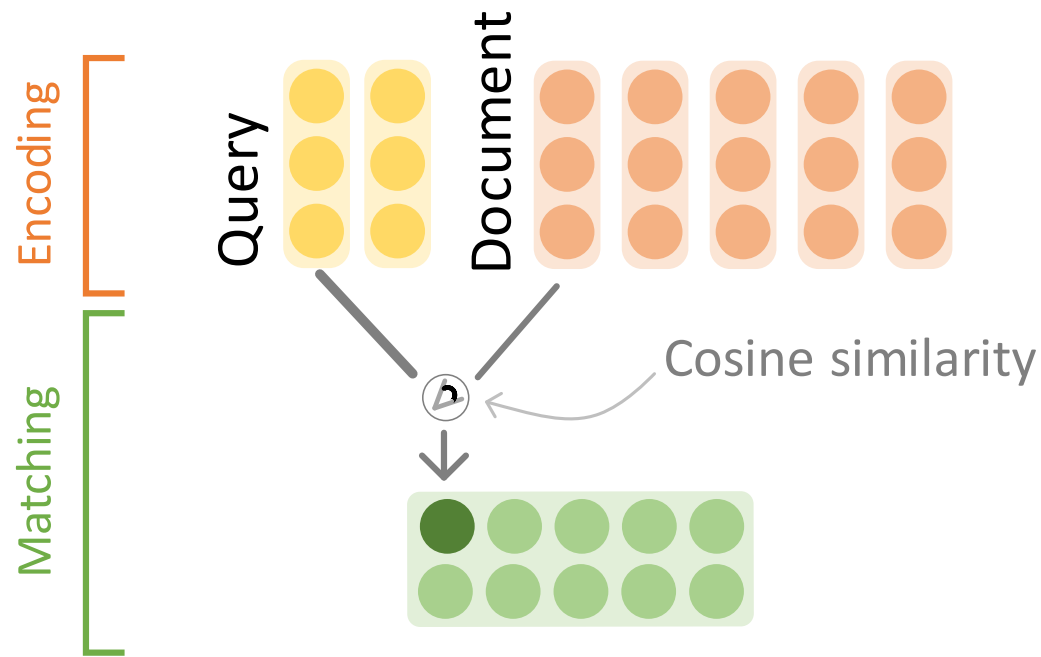
# The encoding layer

---

- Typically (<2019 🤖) a word embedding
  - Pre-trained Word2vec/Glove
  - Fine-tuned (= trained with the rest of the model)
- 2019: BERT (huge transformer based model, only pre-trained)
  - Shows very strong results
- A simple word embedding has still a strong benefits
  - #1 = Speed (after all it is just an memory address lookup on inference)
  - Somewhat interpretable / easily extracted + analyzed & reused in another task

# The match matrix

---

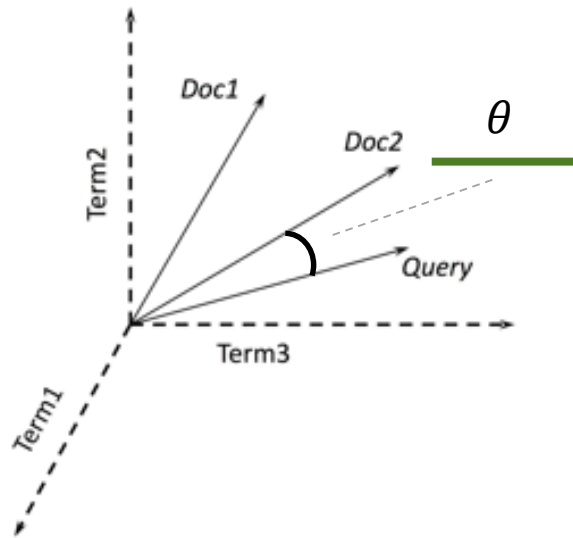


- The core of many successful neural IR models\*
- Matrix of similarities of individual word combinations
- Only a transformation – not parameterized by itself

\* Of course there are other approaches, but we focus on models based on the match matrix

# Cosine similarity

---



$$\begin{aligned}\text{sim}(d, q) &= \cos(\theta) \\ &= \frac{d \cdot q}{|d||q|}\end{aligned}$$

- Cosine similarity measures direction of vectors, but not the magnitude
- Not a distance – but equivalent to Euclidean distance of unit (length=1) vectors

$\theta$  Angle between two vectors

$q$  Vector of query

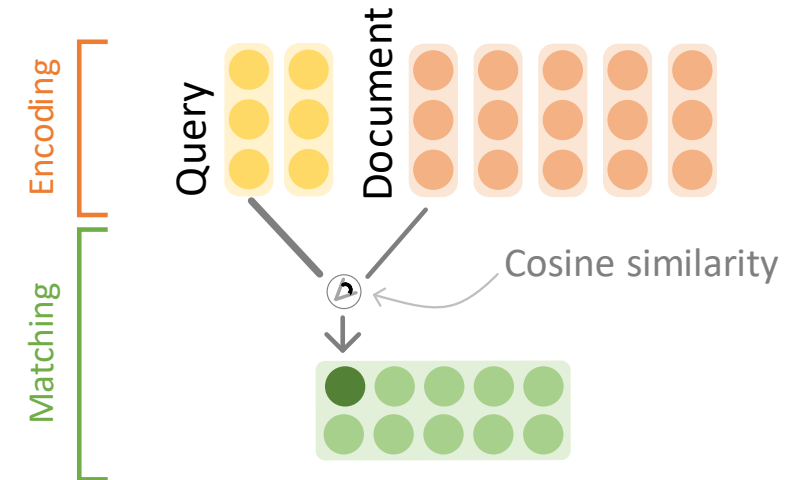
$d$  Vector of document

$d \cdot q$  Dot product

$$= \sum_{i=1}^{dim} d_i * q_i$$

# Cosine similarity in PyTorch

- Using efficient batched-matrix multiplication
- Input shape:
  - Query: [batch, query\_seq\_len, emb\_dim]
  - Document: [batch, doc\_seq\_len, emb\_dim]
- Output shape: [batch, query\_seq\_len, doc\_seq\_len]



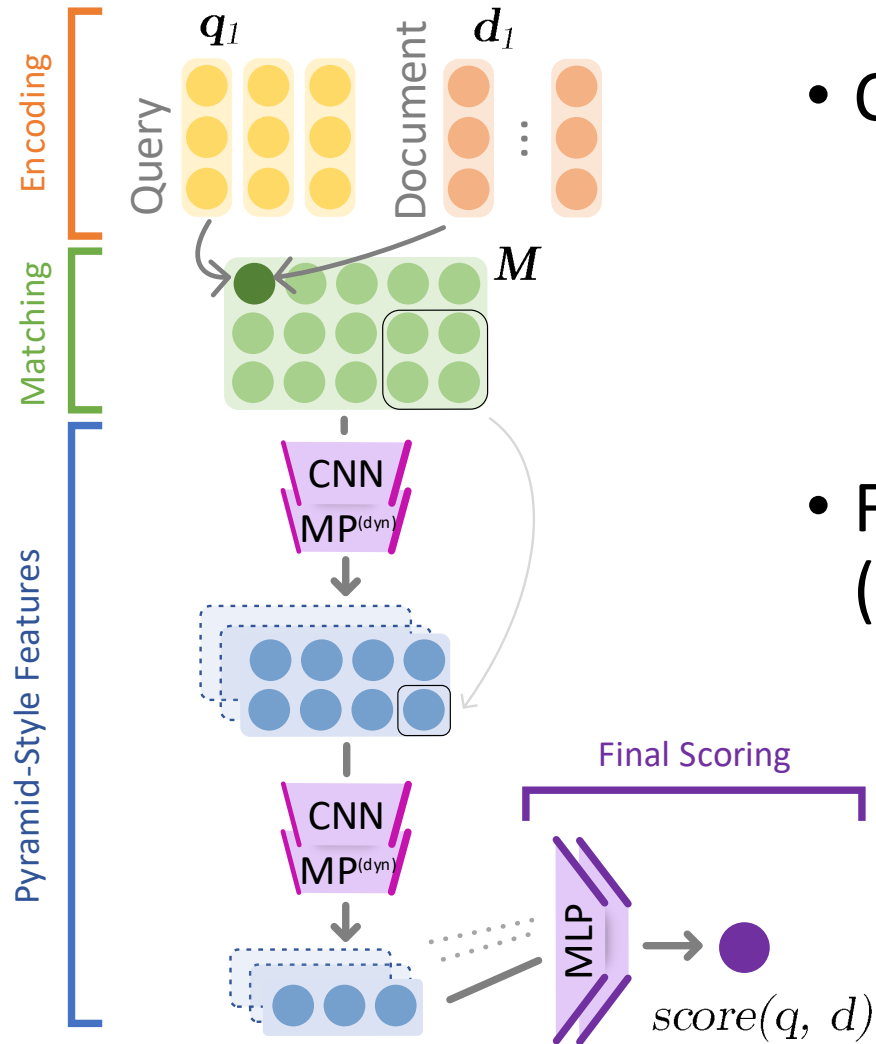
```
a_norm = matrix_1 / (matrix_1.norm(p=2, dim=-1, keepdim=True) + 1e-13)
b_norm = matrix_2 / (matrix_2.norm(p=2, dim=-1, keepdim=True) + 1e-13)
result = torch.bmm(a_norm, b_norm.transpose(-1, -2))
```

# MatchPyramid

---

- Apply a set of 2D convolutional layers on top of the match matrix
  - Inspired by computer vision: match matrix = image
- Each Conv Layer: 2D-CNN & 2D-Dynamic-Pooling
  - Dynamic Pooling takes care of variable length input to fixed output
- Architecture & effectiveness strongly depends on configuration
  - How many layers
  - Which CNN & pooling kernel sizes
  - Generally: Pooling output becomes gradually smaller (like the Pyramid)

# MatchPyramid



- Conv-layers extract local interaction features
  - Max pooling only keeps strong interaction signals (better matches)
  - Different channels can learn different interaction patterns
- Finally a multi-layered feed forward module (MLP) scores the extracted feature vectors

**Simplified architecture illustration:** Omitted zero-padding (at the start of the pyramid), more layers & channels in practice, last layer can be >1D (nD can be flattened to 1D)

# MatchPyramid

Matching

$$M_{ij} = \mathbf{cos}(q_i, d_j) = \frac{d_j \cdot q_i}{|d_j| |q_i|}$$

Pyramid-Style Features

$$z_{ij}^{(1,c)} = \mathbf{2D\_Conv}(M_{ij})$$

$$= \mathbf{ReLU}\left(\sum_{s=0}^{r_c-1} \sum_{t=0}^{r_c-1} w_{s,t}^{(1,c)} * M_{i+s,j+t} + b^{(1,c)}\right)$$

$$z_{ij}^{(2,c)} = \mathbf{dyn\_max\_pool}\left(z_{ij}^{(1,c)}\right) = \max_{0 \leq s < d_c} \max_{0 \leq t < d_c} z_{i*d_c+s,j*d_c+t}^{(1,c)}$$

...

$$z_{ij}^{(l,c)} = \mathbf{max\_pool}\left(\mathbf{2D\_Conv}(z_{ij}^{(l-1)})\right)$$

Scoring

$$s = \mathbf{MLP}(z^l) = W_2 * \mathbf{ReLU}(W_1 * z^l + b_1) + b_2$$

$q_i$	Vector of $i$ -th query token
$d_j$	Vector of $j$ -th document token
$M$	Match-matrix
$z^{(n,-)}$	Sequential variable (n layers)
$c$	Channels
$r_c$	Size of channel $c$
$d_c$	Dynamic pooling kernel size
$W_*, b_*$	Weights & biases
$s$	Output score

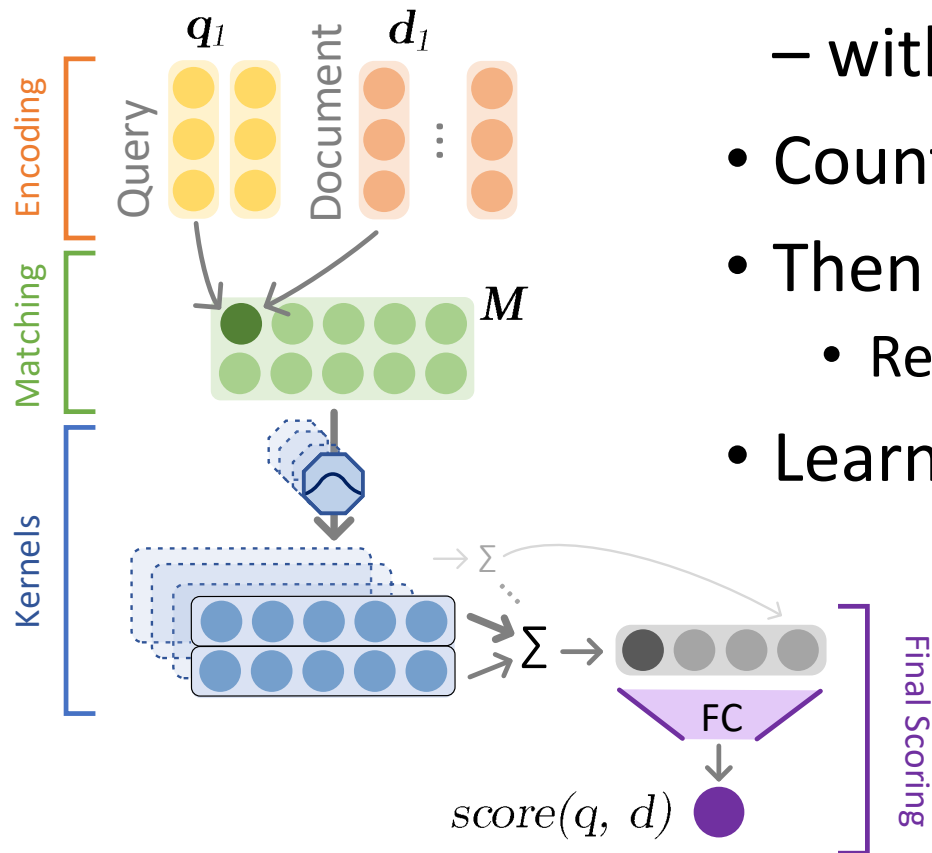


# KNRM

---

- KNR**M**: **K**ernel based **N**eural **R**anking **M**odel
- Counts the amount of different similarities between the query and document
- Very few learnable parameters (other than the embedding)
- Very fast – there is no complicated architecture increasing the runtime
- Roughly the same effectiveness as MatchPyramid

# KNRM



- Transform the similarity values of the match-matrix – with a “RBF-kernel” function
- Count the matches in different similarity bins
- Then sum up on doc and query dimensions
  - Result: 1 value per bin
- Learn bin value combination weights

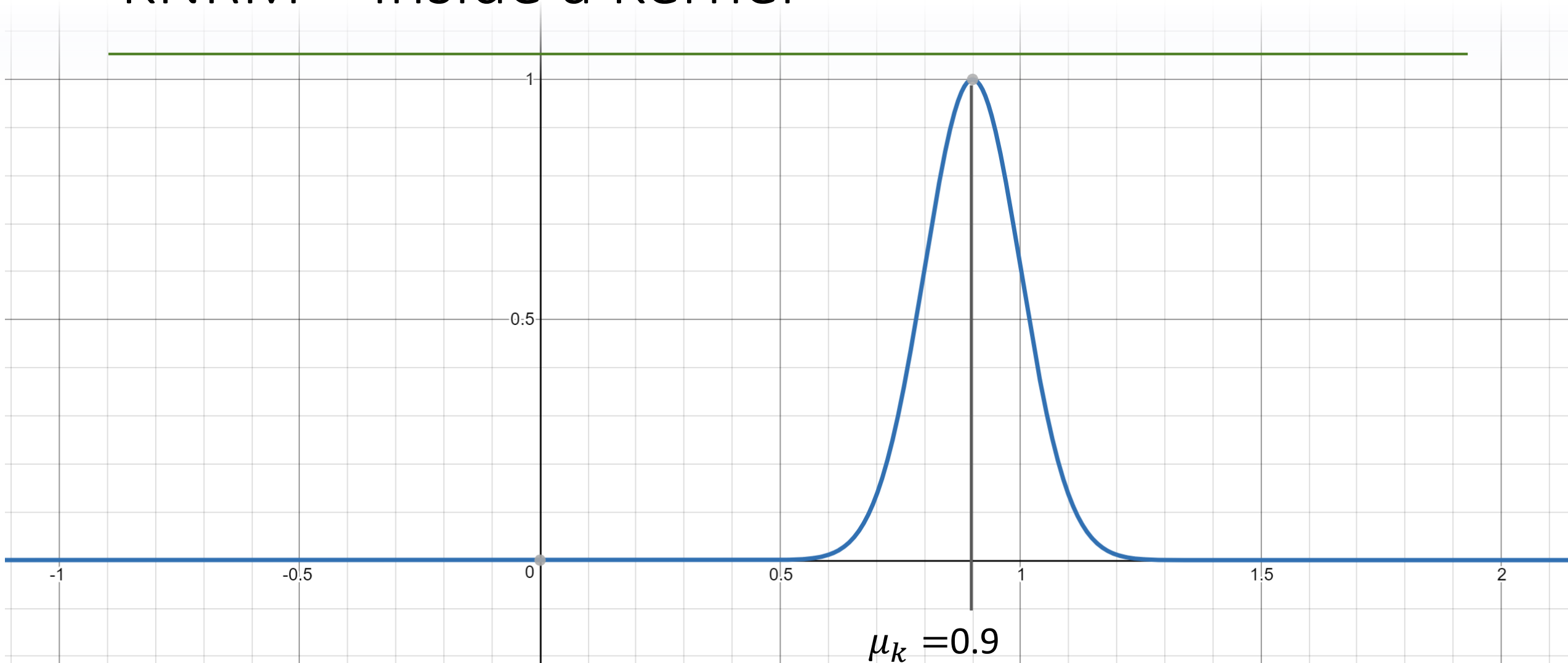
**Simplified architecture illustration:**  $\Sigma$  represents 2x summation (1x on the query dim & 1x on the doc dim), more kernels in practice

# KNRM

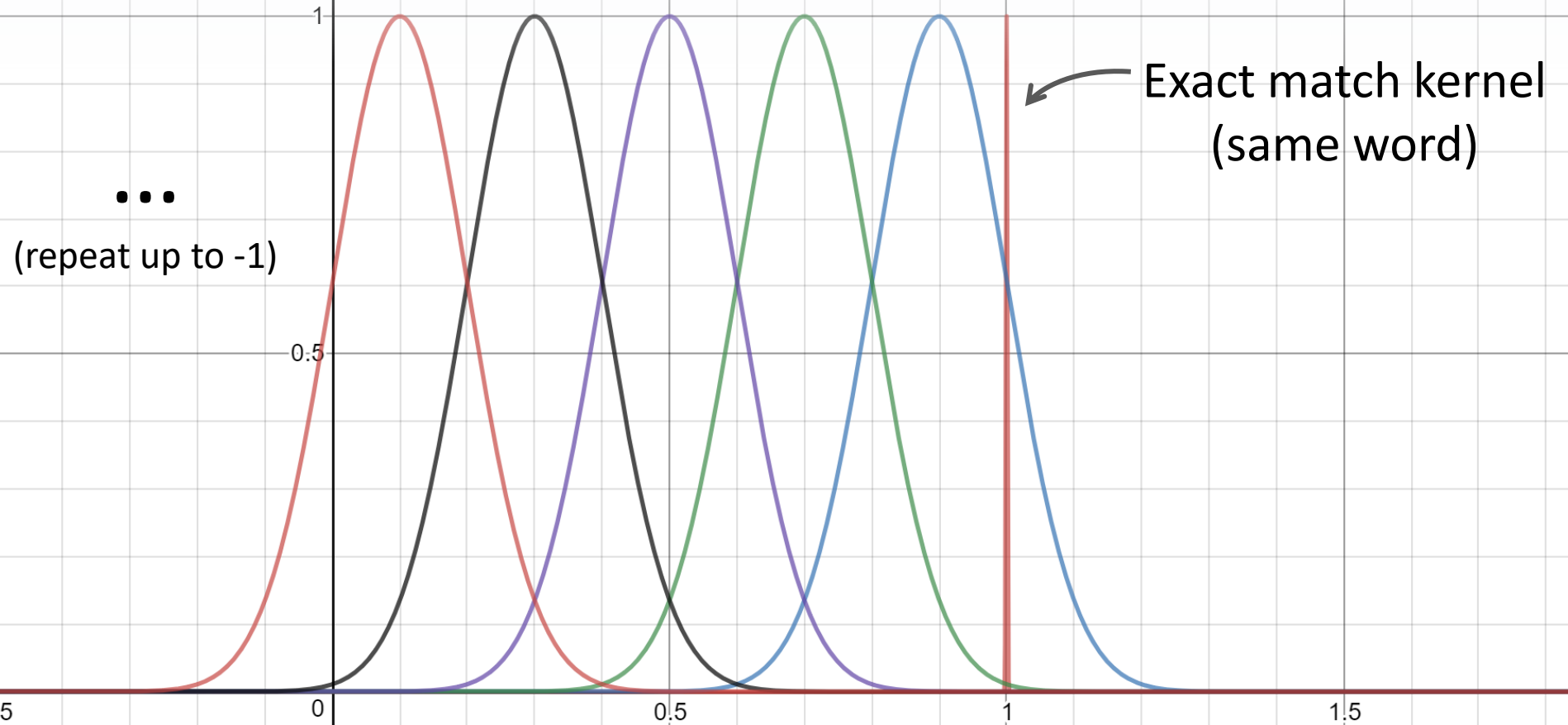
Matching	$M_{ij} = \mathbf{cos}(q_i, d_j) = \frac{d_j \cdot q_i}{ d_j  q_i }$	
Kernels	$K_k(M_i) = \sum_j \exp\left(-\frac{(M_{ij} - \mu_k)^2}{2\sigma_k^2}\right)$	"RBF-Kernel" (applied on a single match)
	$K_k(M) = \sum_{i=1}^n \log(K_k(M_i))$	Sum alongside document dimension
Scoring	$s = \mathbf{FC}(K) = W * K + b$	Sum alongside query dimension

$q_i$	Vector of $i$ -th query token
$d_j$	Vector of $j$ -th document token
$M$	Match-matrix
$K$	All Kernels
$K_k$	$k$ -th kernel
$\mu_k$	Similarity level
$\sigma_k$	Kernel-width/range
$W, b$	Weights & biases
$s$	Output score

# KNRM – Inside a Kernel



# KNRM – Inside all Kernels



# And now ... A short tale about reproducibility

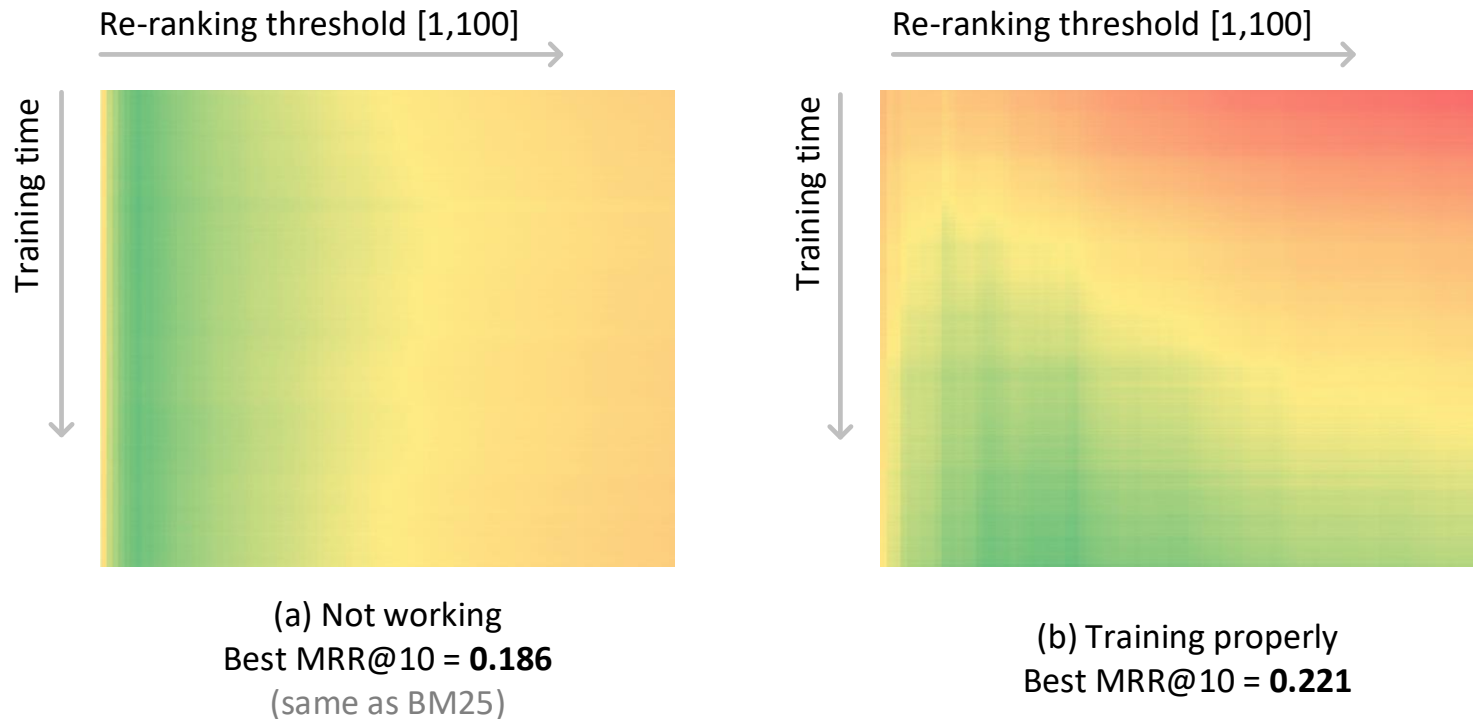
---



# KNRM – Implementation Details Matter

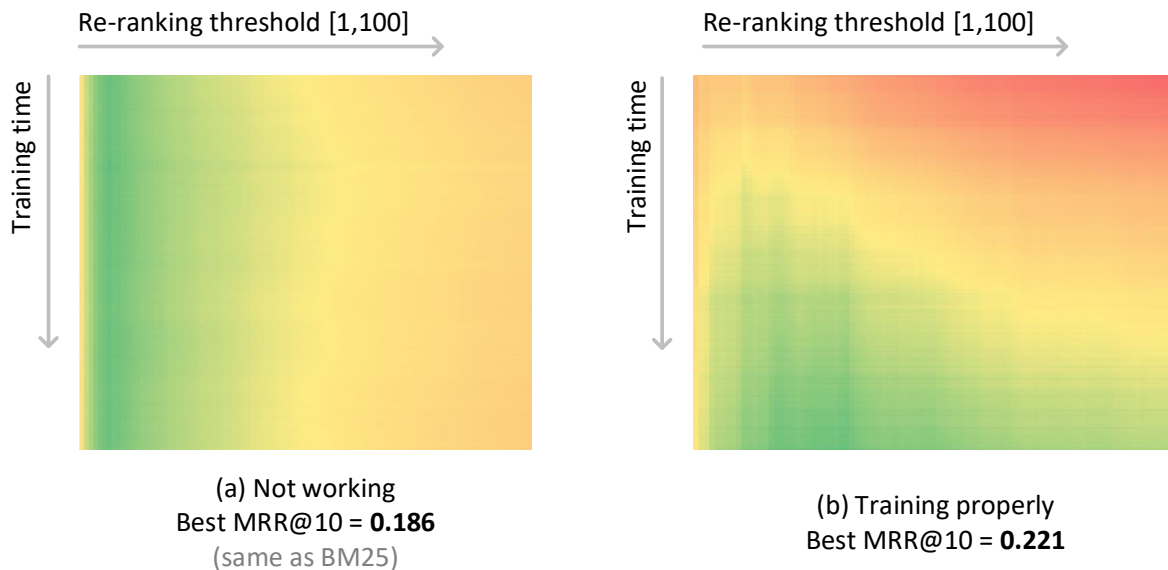
---

- What is different between these two output heatmaps?



*\* Note: the color scales are min (red, yellow) to max (green) individual per run*

# KNRM – Implementation Details Matter



\* Note: the color scales are min (red, yellow) to max (green) individual per run

- Difference: (a) =  $\log(1 + \text{soft\_tf})$   
(b) =  $\log(\text{soft\_tf})$
- There is open-source code for both
- At first it seems counterintuitive, our best educated guess:
  - $\log(\text{soft\_tf})$  acts as regularization, meaning it basically ignores single occurrences as noise

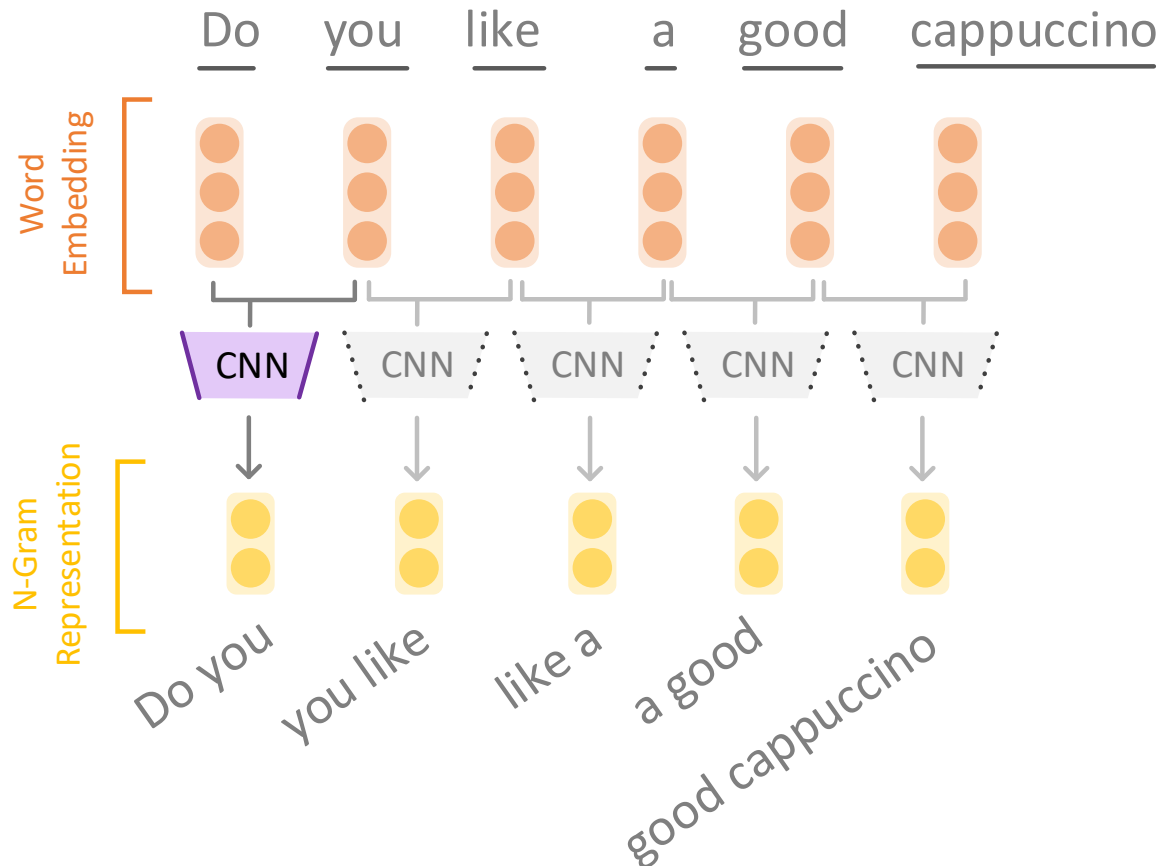


# Conv-KNRM

---

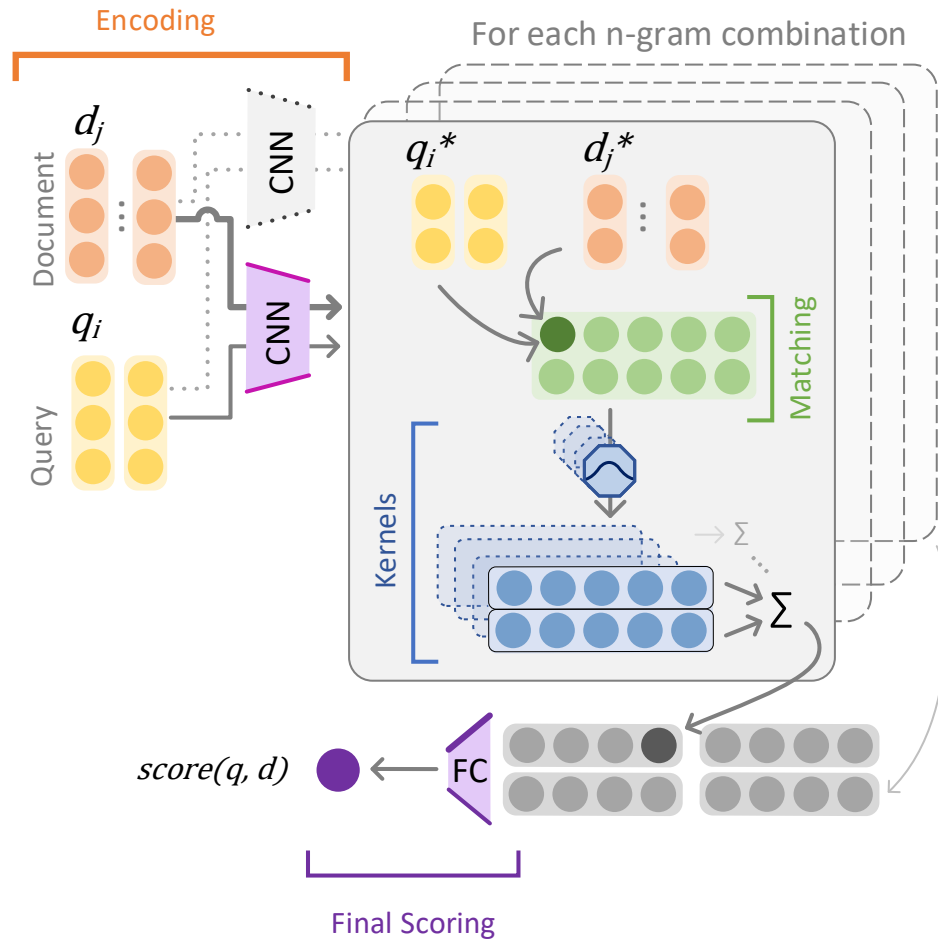
- Extends KNRM before the match-matrix
- Crossmatches n-gram representations, and then applies KNRM
  - Allows to match: “convolutional neural networks” with “deep learning”
- N-grams (and term proximity) are very important in retrieval
  - Not feasible to create a vocabulary with all possible n-grams
- Most effective model highlighted today

# Recall: Word N-Grams with 1D CNNs (Lecture 4)



- Apply a 1D CNN on a sequence of word vectors
- N of N-grams = filter size
  - In this example N=2
- Output is a sequence of N-gram representations
  - Further used in other network components
- WE & CNN can be trained end-to-end

# Conv-KNRM



- Creates n-gram representations for query and doc sequences
- Matches the n-gram vectors
  - 1 match matrix for every  $n \times n$ : 1,1 – 1,2 – 1,3 – 2,2, ...
- Then KNRM and concatenation of result vectors

# Conv-KNRM

Encoding

$$q_{1..n}^h = 1D\_CNN^h(q_{1..n}) \quad d_{1..m}^h = 1D\_CNN^h(d_{1..m})$$

Matching

$$M_{ij}^{h_q h_d} = \mathbf{cos}(q_i^{h_q}, d_j^{h_d})$$

For each  $h_q h_d$  combination

Kernels

$$K_k^{h_q h_d}(M) = \sum_{i=1}^n \log\left(\sum_j \exp\left(-\frac{\left(M_{ij}^{h_q h_d} - \mu_k\right)^2}{2\sigma_k^2}\right)\right)$$

Scoring

$$s = \mathbf{FC}(K) = W * K + b$$

Same as KNRM

$q_{1..n}$  Query token vecs

$d_{1..m}$  Doc. token vecs

$M$  Match-matrix

$h, h_q, h_d$  n-gram size

$K$  All Kernels

$K_k$   $k$ -th kernel

$\mu_k$  Similarity level

$\sigma_k$  Kernel-width/range

$W, b$  Weights & biases

$s$  Output score

# Other models

---

- PACCR, Hui et al. 2017
  - Applies multiple 2D-Conv layers on top of the match matrix
  - N-gram matching after the single match matrix
- DUET, Mitra et al. 2017
  - Individual word matches + single vector per doc & query matching
- DRMM, Guo et al. 2016
  - Does not use a match matrix, rather histogram of similarities
  - Embedding is not updated

# Bonus paper 100

On the Effect of Low-Frequency Terms on Neural IR models

# On the Effect of Low-Frequency Terms

---

- Infrequent terms carry a lot of relevance information in IR
- But: removed with a fixed vocabulary
  - Mainly as a concession to efficiency demands
  - Neural IR model doesn't "see" removed terms
- Fixed vocabulary for all terms doesn't scale
  - Missing or very little training data – useless representations
  - Unseen query terms are again OOV

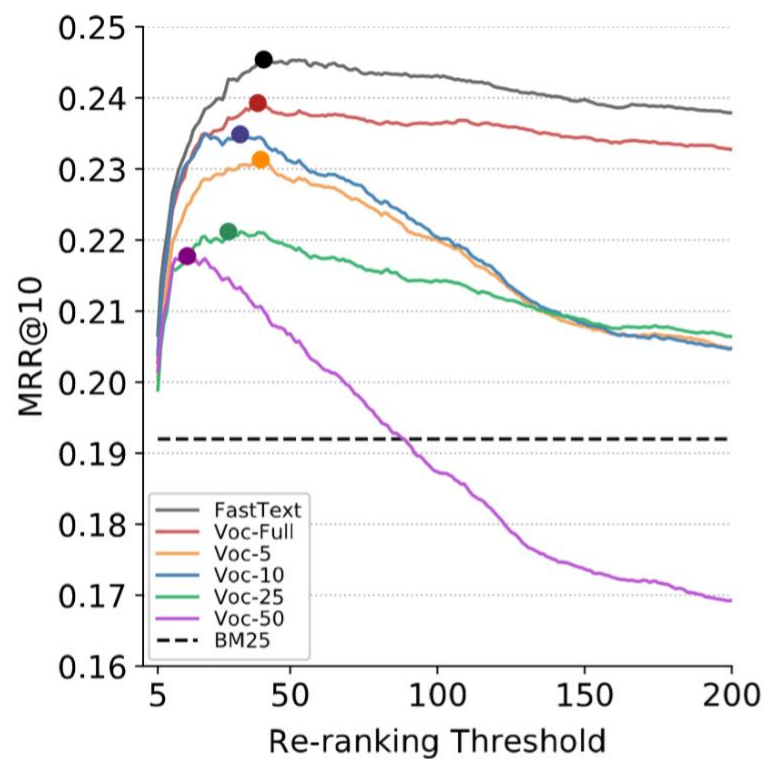
# On the Effect of Low-Frequency Terms

---

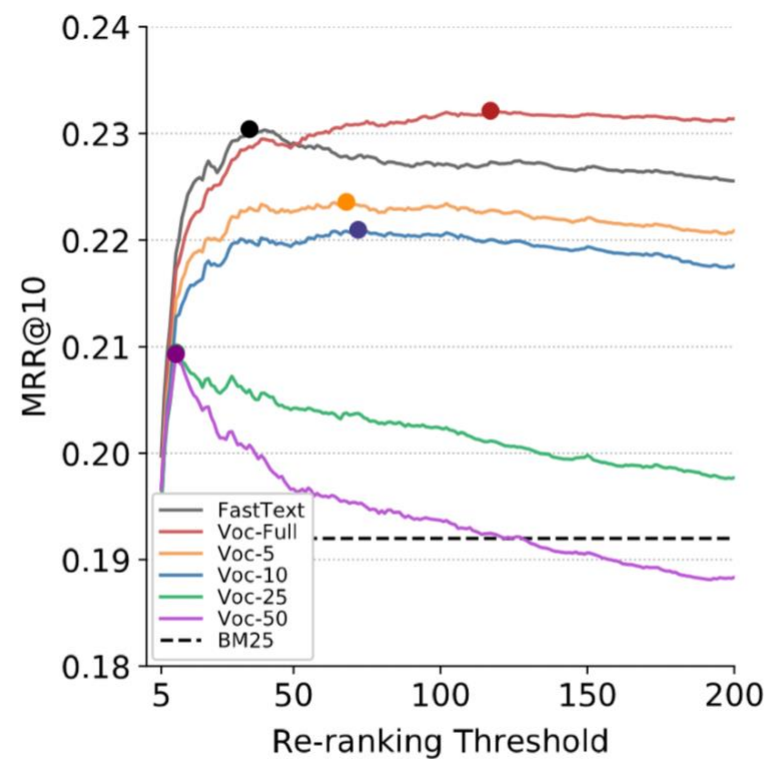
- ① We show the importance of covering all terms of the collection
  - First to analyze the re-ranking threshold – great tool for diagnostics
  - If the model doesn't perform well: more re-ranking docs decrease effectiveness
- ② Using the FastText strongly increases the effectiveness on queries with low-frequency terms
  - FastText: Sub-word embedding
  - Character n-gram composition for low frequency terms
  - Keeps good inference performance



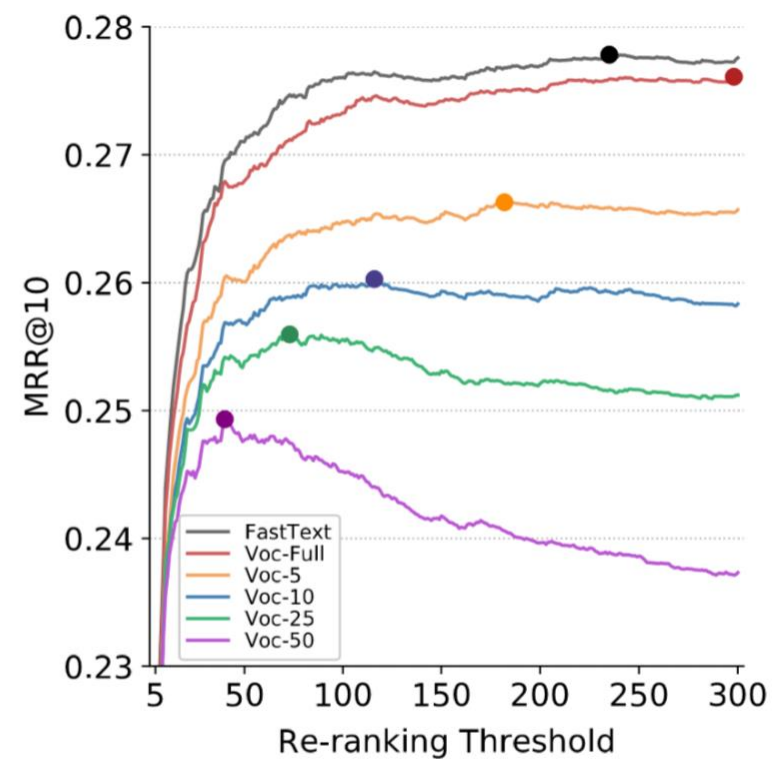
# Effect of the Fixed Vocabulary Size



(a) *MatchPyramid*

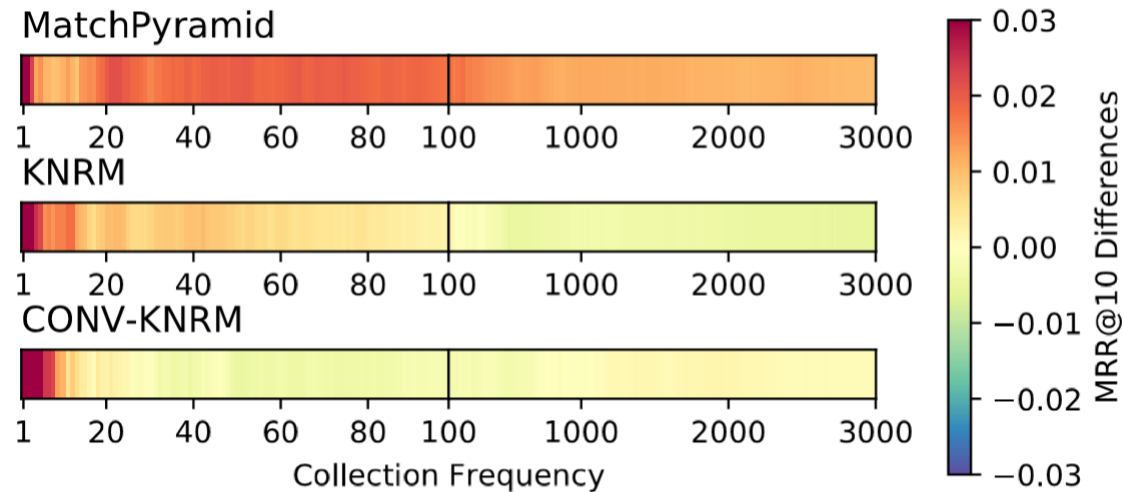


(b) *KNRM*



(c) *CONV-KNRM*

# Handling of Low-Frequency Terms



**Figure 2: MRR differences of the models, trained on the FastText embeddings and the embeddings with full vocabularies, over the queries with minimum collection frequency of their terms smaller or equal to the X-axis**  
(red = FastText is better, blue=Vocab-Full is better)

- Focus on difference between full vocab  $\leftrightarrow$  FastText
- FastText is better on queries with  $<10$  occurrence terms
- Differences become less with higher frequency terms
  - Fundamental the same concept of encoding (not contextualized etc..)

# Handling of Low-Frequency Terms

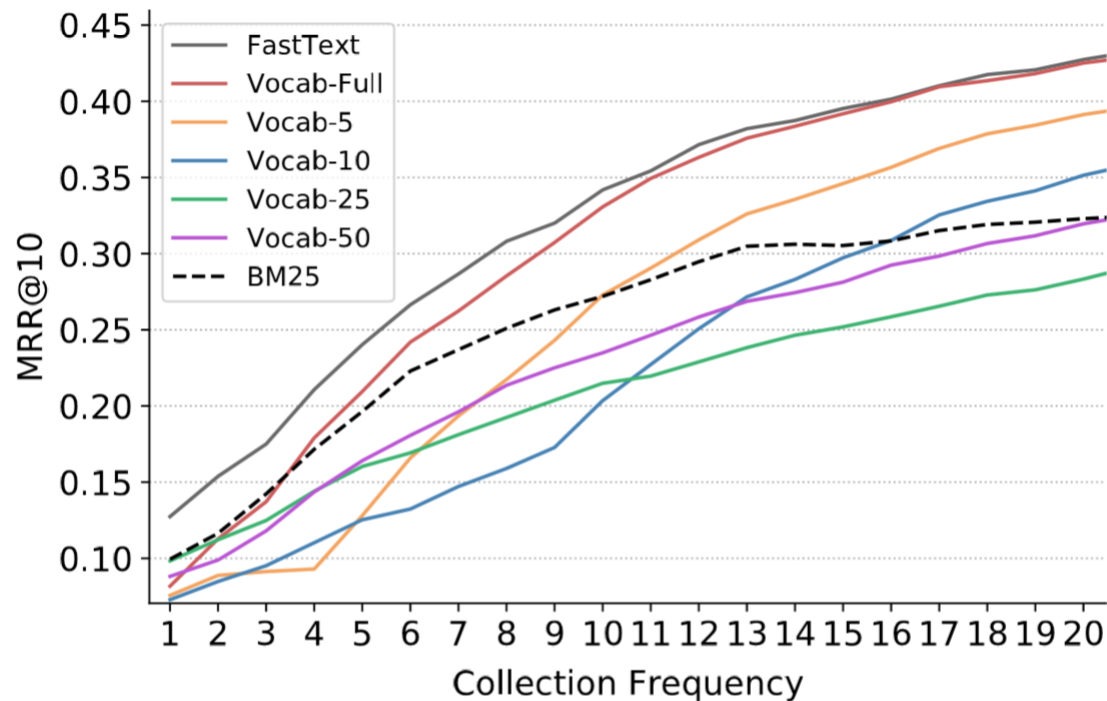


Figure 3: MRR results of *CONV-KNRM* over the queries with at least one term with collection frequency smaller than or equal to the values on the X-axis

- Focus on difference between all vocabs  $\leftrightarrow$  FastText
- Keep in mind the huge scale of the y-axis
- Secondary insight: Neural IR models strongly bounded by BM25 first stage ranking
  - Future Work!

# Interested? Here is more...

---

- Neural Models for Information Retrieval Bhaskar Mitra, Nick Craswell  
<https://arxiv.org/abs/1705.01509>
  - Very good survey article
- Conference tutorials on neural networks for IR <http://nn4ir.com/>

# Summary: Neural Networks for IR

---

- 1 Pair (1 query, 1 doc) scoring – triples training – listwise evaluation
- 2 Word level match matrix: core building block of Neural IR models
- 3 Environment (vocabulary, re-ranking depth) matters a lot

- 1 Pair (1 query, 1 doc) scoring – triples training – listwise evaluation
- 2 Word level match matrix: core building block of Neural IR models
- 3 Environment (vocabulary, re-ranking depth) matters a lot

Thank You