# Neural IR

## [DAT640] Information Retrieval and Text Mining

Trond Linjordet
University of Stavanger

November 4, 2019

# Outline (course)

- ~~Search engine architecture, indexing~~
- ~~Evaluation~~
- ~~Retrieval models~~
- ~~Query modeling~~
- ~~Learning-to-rank~~, **Neural IR** $\Leftarrow$ today
- Semantic search $\sim 80\%$

# Outline (Neural IR)

- Neural Networks
- Word Embeddings - Word2Vec
- Neural IR Models

# Neural Networks

# Neural networks: The Perceptron

- The perceptron illustrates the idea of an artificial neuron, or activation unit.

$$z = b + \sum_j w_j \times x_j$$

$$y = f_{\text{activation}}(z) \doteq \begin{cases} 0 & \text{if } z > 0 \\ 1 & \text{if } z \leq 0 \end{cases}$$

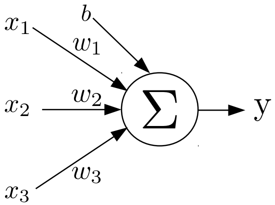


Figure: The perceptron.

# Neural networks: Multilayer perceptron

- Continuous non-linear functions with defined derivatives, e.g. the sigmoid logistic function:

$$f_{\text{activation}}(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

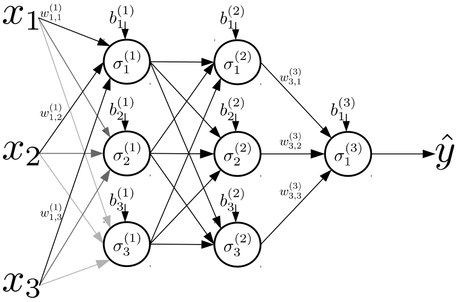


Figure: One example of a multilayer perceptron.

# Neural networks: Multilayer perceptron

- Example MLP from previous slide, feedforward as equation:

$$\begin{aligned} y &= \sigma(\mathbf{W}^{(3)\intercal}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}) \\ &= \sigma(\mathbf{W}^{(3)\intercal}\sigma(\mathbf{W}^{(2)\intercal}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) \\ &= \sigma(\mathbf{W}^{(3)\intercal}\sigma(\mathbf{W}^{(2)\intercal}\sigma(\mathbf{W}^{(1)\intercal}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) \end{aligned}$$

- Loss function: $J(\theta) \propto ||y - f(\mathbf{x};\theta)||$
- Gradient descent to minimize loss: $\theta_{\text{new}} \leftarrow \theta_{\text{old}} - \alpha \nabla_{\theta_{\text{old}}} J(\theta_{\text{old}})$
- Backpropagation, the chain rule, and vanishing gradients:

$$\frac{\partial}{\partial w_j^{(L)}} J(\theta) = \frac{\partial z^{(L)}}{\partial w_j^{(L)}} \frac{\partial h^{(L)}}{\partial z^{(L)}} \frac{\partial J(\theta)}{\partial h^{(L)}}$$

# Word Embeddings - Word2Vec

# Word embeddings - background

- Vector space models (e.g. TF-IDF)

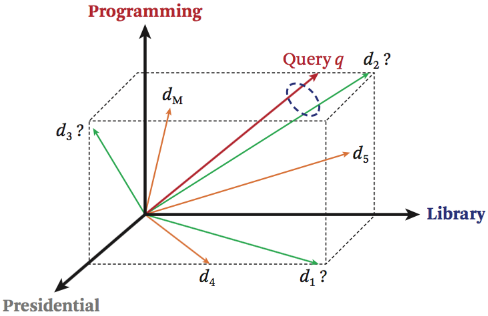


Figure: Vector space model. Illustration is taken from (Zhai&Massung, 2016)[Fig. 6.2]

# Word Embeddings - Background

- Terms represented as atomic symbols by discrete, local vectors:
- *one-hot encodings*, bit vectors with one $1$ element and the rest $0$.

$$\mathbf{w}_{\text{hotel}} = (0\,0\,1\,0\,0\,0\ldots0\,0)^\mathsf{T}$$
$$\mathbf{w}_{\text{motel}} = (0\,0\,0\,1\,0\,0\ldots0\,0)^\mathsf{T}$$

- Can count term frequencies, but do not capture relationships (similarity) of meaning between different words.
- Every vector has the same dimensionality as the entire vocabulary.

# Word Embeddings - Objective

- Can words be represented vector space so that the similarity of meanings can be quantified directly from the words' vector representation?
- Then we want dense, continuous vectors of lesser dimensionality:

$$\mathbf{v}_{\text{hotel}} = \begin{pmatrix} 0.19 & 0.2 & -0.9 & 0.4 \end{pmatrix}^\intercal$$
$$\mathbf{v}_{\text{motel}} = \begin{pmatrix} 0.27 & 0.01 & -0.7 & 0.3 \end{pmatrix}^\intercal$$

- This lets us quantify a measure of similarity: $\mathbf{v}_{\text{hotel}}^\intercal \mathbf{v}_{\text{motel}}$

# Word Embeddings - Word2Vec

- Distributional hypothesis: "You shall know a word by the company it keeps." (Firth, J. R., 1957)
- Word2Vec (Mikolov, 2013)
- Represent words based on the contexts in which they occur.
- CBOW: Predict target word $w_t$ based on context words $w_{t-j}$, $w_{t+j}$ within some context window $C$ around $w_t$ .
- Skip-gram: Predict context words $w_{t-j}$, $w_{t+j}$ within some context window $C$ with radius $m$ around $w_t$, based on $w_t$.
  - We will focus on this algorithm.

# Word Embeddings - Word2Vec

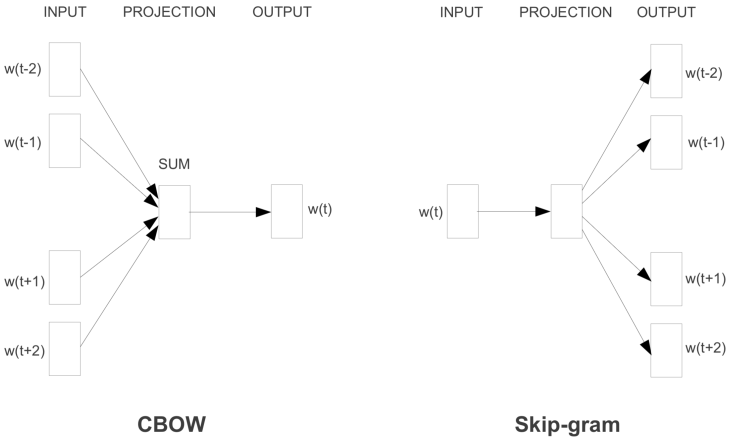


Figure: The continuous-bag-of-words (CBOW) and Skip-gram algorithms. Illustration is taken from (Mikolov, et al., 2013).

# Word Embeddings - Word2Vec - Skip-gram

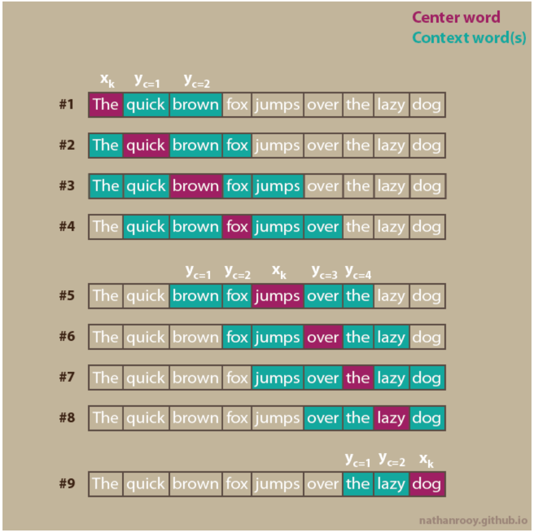


Figure: A sliding word window example. Illustration is taken from (Rooy, 2018).

# Word Embeddings - Word2Vec - Skip-gram

- Maximize the probability of true context words $w_{t-m}$, $w_{t-m+1}$, ..., $w_{t-1}$, $w_{t+1}$, ..., $w_{t+m-1}$, $w_{t+m}$ for each target word $w_t$:

$$J'(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

- Negative Log Likelihood:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

# Word Embeddings - Word2Vec - Skip-gram - Embedding

- Take $\mathbf{w}_j$ as the one-hot encoding vector for the word $w_j$.
- Target words $\mathbf{w}_t$ are embedded with matrix $\mathbf{W}$ as follows:

$$\mathbf{v}_t = \mathbf{W}\mathbf{w}_t$$

- This picks the $n$'th row of $\mathbf{W}$, given that $\mathbf{w}_t$ is the $n$'th word in the vocabulary.
- With a different embedding matrix $\mathbf{W}'$ for context words $w_c$, similarly we get

$$\mathbf{u}_c = \mathbf{W}'\mathbf{w}_c$$

# Word Embeddings - Word2Vec - Skip-gram - Visualization

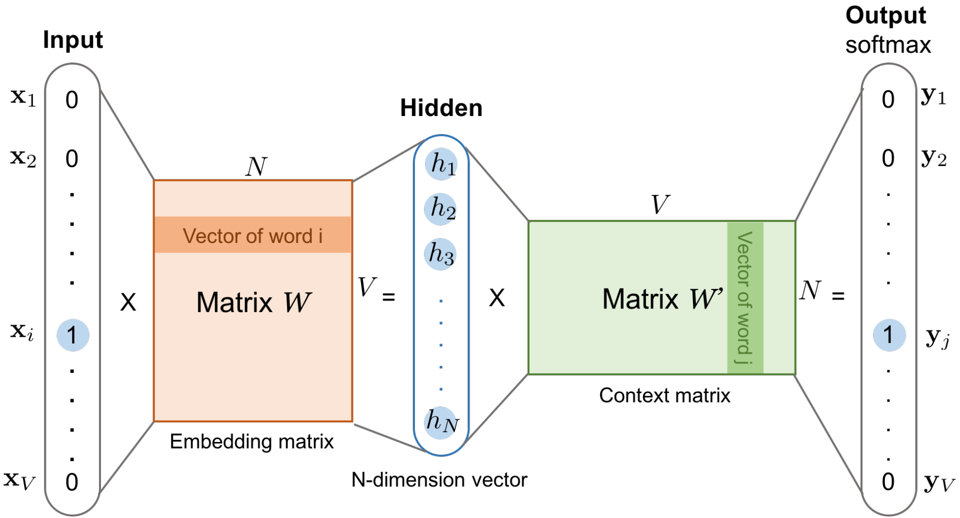


Figure: Forward pass of Skip-gram. Illustration is taken from (L. Weng, 2017).

# Word Embeddings - Word2Vec - Skip-gram - Forward pass

- What should a prediction then look like?
- For each target word, one could take any row $\mathbf{u}_j$ in $\mathbf{W}'$ to evaluate the probability that $w_j$ is in the context of $w_t$:

$$P(w_j \in C|w_t) = \frac{e^{\mathbf{u}_j^\intercal \mathbf{v}_t}}{\sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t}}$$

- This form is a Softmax function, which is here used to express a discrete probability distribution over the vocabulary.
- For generative modeling, take the $w_j$ with the highest value of $P(w_j \in C|w_t)$ as the predicted word.

# Word Embeddings - Word2Vec - Skip-gram - Training

- For training, compare the dense probability vector (elementwise on rows of $\mathbf{W}'$)

$$\hat{\mathbf{y}} = \frac{e^{\mathbf{W}'\mathbf{v}_t}}{\sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t}}$$

- with each of the ground truth context words' one-hot encoding vector $\mathbf{y}_c = \mathbf{w}_c$.
- For example:

$$\hat{\mathbf{y}} = \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{pmatrix}^\intercal$$
$$\mathbf{y}_c = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}^\intercal$$

- All the elementwise differences between these two vectors contribute to the loss function's value, and hence the updates to the parameter values in $\mathbf{W}'$ and $\mathbf{W}$.

# Word Embeddings - Word2Vec - Skip-gram - Training

- For training, compare the dense probability vector (elementwise on rows of $\mathbf{W}'$)

$$\hat{\mathbf{y}} = \frac{e^{\mathbf{W}'\mathbf{v}_t}}{\sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t}}$$

  with each of the ground truth context words' one-hot encoding vector $\mathbf{y}_c = \mathbf{w}_c$.
- For example:

$$\hat{\mathbf{y}} = \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 \end{pmatrix}^\intercal$$
$$\mathbf{y}_c = \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix}^\intercal$$

- All the elementwise differences between these two vectors contribute to the loss function's value, and hence the updates to the parameter values in $\mathbf{W}'$ and $\mathbf{W}$.

# Word Embeddings - Word2Vec - Skip-gram - Loss Function

- We can express the loss function in a bit more detail:

$$
\begin{aligned}
J(\theta) &= -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t) \\
&= -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log \frac{e^{\mathbf{u}_{t+j}^{\intercal} \mathbf{v}_t}}{\sum_{i=1}^{V} e^{\mathbf{u}_i^{\intercal} \mathbf{v}_t}}
\end{aligned}
$$

- We then need to take the partial derivative of the loss function with respect to the model parameters to be able to update the model during training.

# Word Embeddings - Word2Vec - Skip-gram - $\nabla_\theta$ Loss Function

- We want to find the gradient to be able to update the model.
- For example, if we want to know how to update the target word embeddings $\mathbf{W}$:

$$
\begin{aligned}
\frac{\partial}{\partial \mathbf{v}_t} \log \frac{e^{\mathbf{u}_j^\intercal \mathbf{v}_t}}{\sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t}} &= \frac{\partial}{\partial \mathbf{v}_t} \log e^{\mathbf{u}_j^\intercal \mathbf{v}_t} - \frac{\partial}{\partial \mathbf{v}_t} \log \sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t} \\
&= \mathbf{u}_j - \frac{1}{\sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t}} \left( \frac{\partial}{\partial \mathbf{v}_t} \sum_{k=1}^{V} e^{\mathbf{u}_k^\intercal \mathbf{v}_t} \right) \\
&= \mathbf{u}_j - \sum_{k=1}^{V} \frac{e^{\mathbf{u}_k^\intercal \mathbf{v}_t}}{\sum_{i=1}^{V} e^{\mathbf{u}_i^\intercal \mathbf{v}_t}} \mathbf{u}_k
\end{aligned}
$$

- This can be read as the difference between observed and expected context words.
- Gradient descent is aimed at reducing this difference.

# Word Embeddings - Word2Vec - Summary

- Learn to predict context words given target word. (Or vice versa.)
- These word embeddings can capture relationships between words, e.g.:

$$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx \mathbf{v}_{\text{queen}}$$

- Initialize parameters with small random values.
- *Stochastic* gradient descent
- Negative sampling, with modified unigram probability distribution.
- Alternative word embedding algorithms: GloVe
- Alternative objects to embed: graph, track, sentence, paragraph...

# Exercise #1

- Train Word2Vec word embeddings using the Gensim library for Python.
- Train with different corpora and see how the relationships between words differ based on the training data.
- Code skeleton on GitHub: `exercises/lecture_18/exercise_1.ipynb` (make a local copy)

# Neural IR Models

# Discussion

- How could word embeddings trained using Word2Vec (or a similar) be used for determining the relevance of documents to queries?

# Information Retrieval using Word Embeddings

- Scoring with embeddings
  - Relevance $\sim$ Similarity?
  - Relevance $\sim$ Distance$^{-1}$ ? How do these quantities relate?
  - Use one or two embedding matrices?
- Decisions: (Regression, classification) $\times$(Scoring, ranking).
- Projecting multiword texts into embedding space:
  - Centroid?
  - Pairwise comparison of query and candidate document words? $f(w_q, w_d)$
- We will look at some early models of neural IR.

# More neural networks terminology

- Convolution: A smaller matrix (Filter, Kernel) as a sliding window over input, and take the sum of the elementwise products.
  - Useful for weight-sharing, finding local features, e.g., edges.
- Pooling: Aggregate function (e.g., Max. or Avg.) over a window of the output.
- Dropout: For each minibatch, randomly drop some of the non-output units.

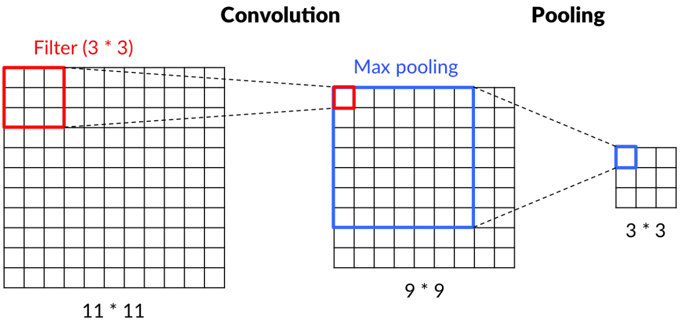


Figure: Forward pass of Skip-gram. Illustration is taken from (L. Weng, 2017).

# Neural IR Model: DSSM

- **DSSM** - Deep Semantic Similarity Model (Huang, et al., 2013).
- Projects query and relevant and non-relevant documents into concept embedding space, then calculates SoftMax over smoothed cosine similarity $\gamma R(Q, D)$ of query and document concept vectors.

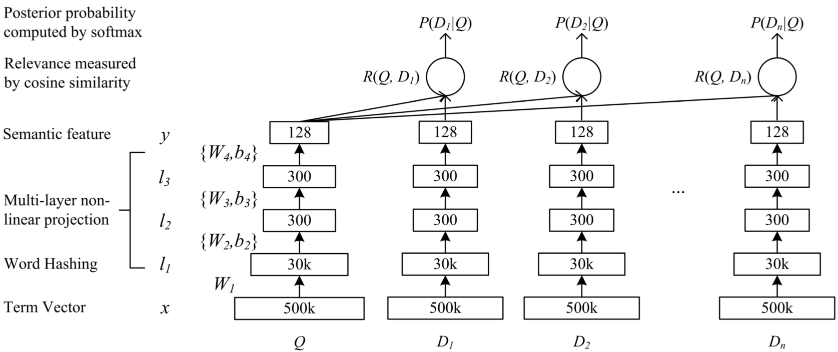


Figure: The architecture of DSSM. Illustration is taken from (Huang, et al., 2013).

# Neural IR Model: DSSM

- The SoftMax can be expressed as

$$P(D|Q) = \frac{e^{\gamma R(Q,D)}}{\sum_{D' \in \mathbf{D}} e^{\gamma R(Q,D')}}, \text{ with } \mathbf{D} \approx \{D^+\} \cup \{D^-\}_{\text{sampled}}.$$

- Loss function can then be expressed as

$$J(\theta) = -\log \prod_{(Q,D^+)} P(D^+|Q).$$

- The DSSM architecture can also be trained for other tasks, given appropriately structured training data pairs:
  - query, document titles $\rightarrow$ document ranking
  - query prefix, query suffix $\rightarrow$ query auto-completion
  - prior query, subsequent query $\rightarrow$ next query suggestion
- In general, are the right latent semantic dimensions being learned for a given task?

# Neural IR Model: Duet

- One strength of local representations over distributed representation is for very rare words in the vocabulary!
- "Aardvark" may not occur often enough to get a very useful word embedding, but its one-hot encoding can still give an exact match.
- **Duet** - Learning to Match Using Local and Distributed Representations of Text for Web Search (Bhaskar, et al., 2017).
- This architecture trains two separate deep neural network submodels jointly, one on local representations and the other on distributed representations.
- Both have submodels include convolution.

$$f(\mathbf{Q}, \mathbf{D}) = f_l(\mathbf{Q}, \mathbf{D}) + f_d(\mathbf{Q}, \mathbf{D})$$

# Neural IR Model: Duet

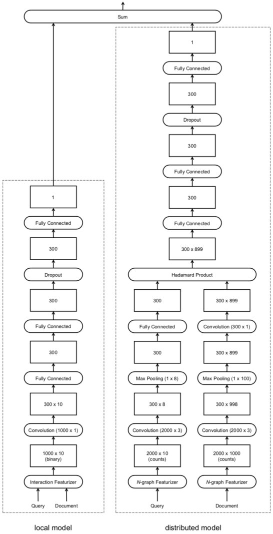


Figure: The architecture of Duet. Illustration is taken from (Bhaskar, et al., 2017).

# Neural IR Model: NRM-F

- **NRM-F** - Neural Ranking Models with Multiple Document Fields (Zamani, et al., 2017). Illustrations are taken from (Zamani, et al., 2017).

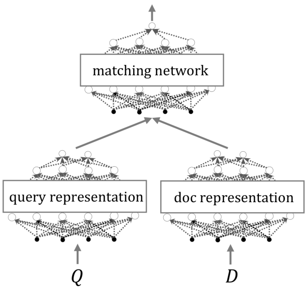


Figure: High-level NRM-F architecture.

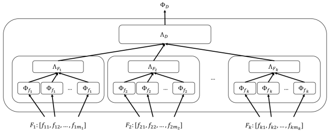


Figure: Multi-field representation embedding.

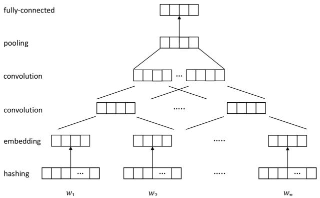


Figure: Instance-level representation learning.

# Neural IR Model: NRM-F

- A specific query embedding is learned for each field in the documents, and a specific document embedding is learned for each field in the documents.
- As these field-specific representations have the same dimensions, a Hadamard product for each field, $\mathbf{q}_{i,f} \circ \mathbf{d}_{j,f}$ is concatenated, with field-level dropout, and passed to the fully-connected matching network.

# Exercise #2

- Use a pre-trained Word2Vec model to implement relevance ranking documents with respect to query.
- Use gensim similarity between centroids of query and document. Need: pre-trained word-embeddings a set of queries+documents, 3000 abstracts in earlier lecture (lecture 7). Open-ended final task: Weight the vectors according to TF-IDF when calculating centroid of query or document.
- Code skeleton on GitHub: `exercises/lecture_18/exercise_2.ipynb` (make a local copy)

# Conclusion

- Neural methods can complement traditional IR methods.
- A variety of patterns can be combined in different configurations.

# References


- Text Data Management and Analysis (Zhai&Massung), Chapters 6.
- T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. In *Proc. of ICLR*, 2013.
- YouTube: Chris Manning, Lecture 2: Word2Vec - Deep Learning for NLP, Stanford, 2017.
- YouTube: Richard Socher, Lecture 3: GloVe - Deep Learning for NLP, Stanford, 2017.
- Word2vec from Scratch with Python and NumPy, Nathan Rooy, March 22, 2018.
  - `https://nathanrooy.github.io/posts/2018-03-22/word2vec-from-scratch-with-python-and-numpy`
- Learning Word Embedding, Lilian Weng, Oct 15, 2017.
  - `https://lilianweng.github.io/lil-log/2017/10/15/learning-word-embedding.html`

# References (continued)


- YouTube: Bhaskar Mitra, Neural Models for Information Retrieval, Microsoft Research, 2018.
- P.S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning Deep Structured Semantic Models for Web Search using Clickthrough Data . In *Proc. of CIKM*, 2013.
- B. Mitra, F. Diaz, and N. Craswell. Learning to Match Using Local and Distributed Representations of Text for Web Search. In *Proc. of WWW*, 2017.
- H. Zamani, B. Mitra, X. Song, N. Craswell, and S. Tiwary. Neural Ranking Models with Multiple Document Fields. In *Proc. of WSDM*, 2017.