Multi-Resolution Models for Learning Multilevel Abstract Representation of Text

Xiaowei Xu
University of Arkansas
Little Rock

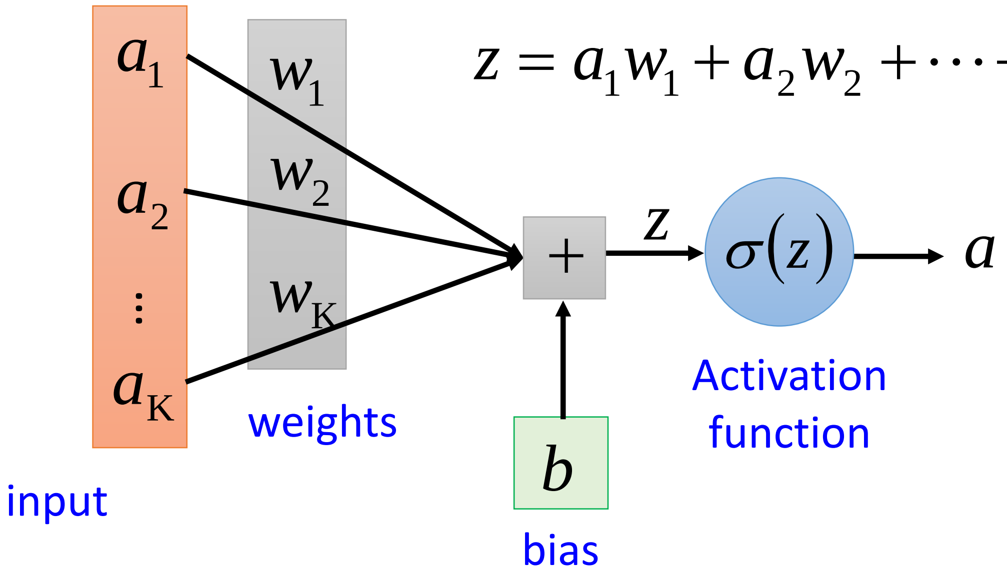
Data representation

- Traditional machine learning (before deep learning) use handcrafted features to represent data.
- Problems:
  - Manual and tedious.
  - Dependent on domain knowledge.
  - Unhinged from the machine learning task.

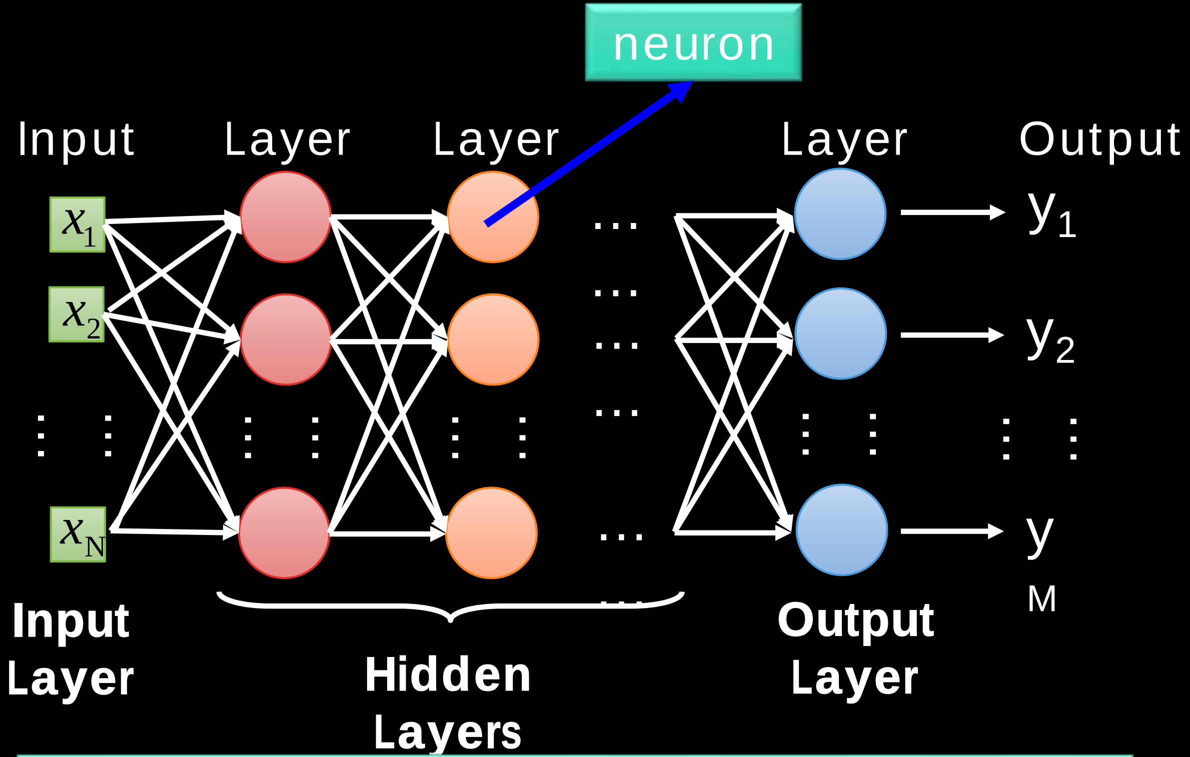
Element of Neural Network

Neuron $f: R^K \rightarrow R$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$



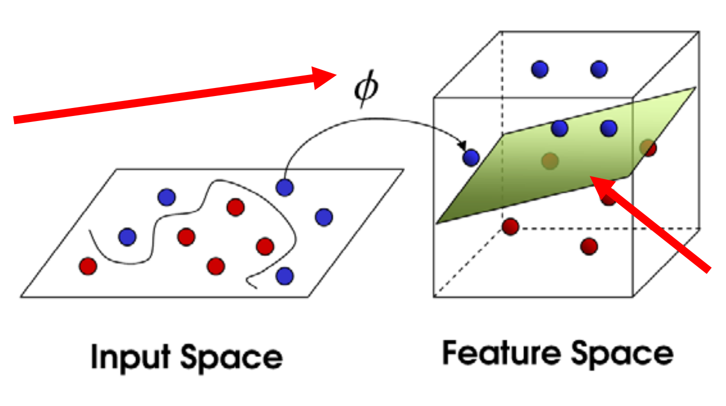
NEURAL NETWORK



Deep means many hidden layers

SVM

Hand-crafted kernel function



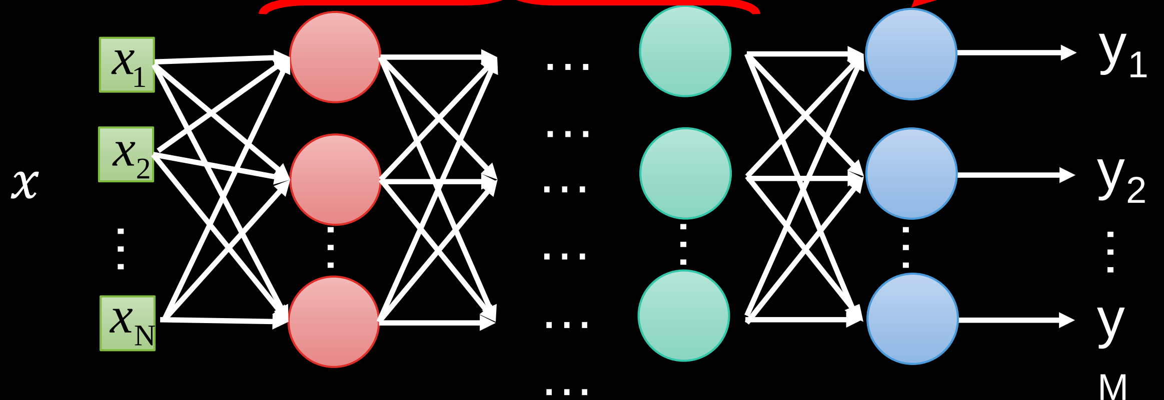
Apply simple classifier

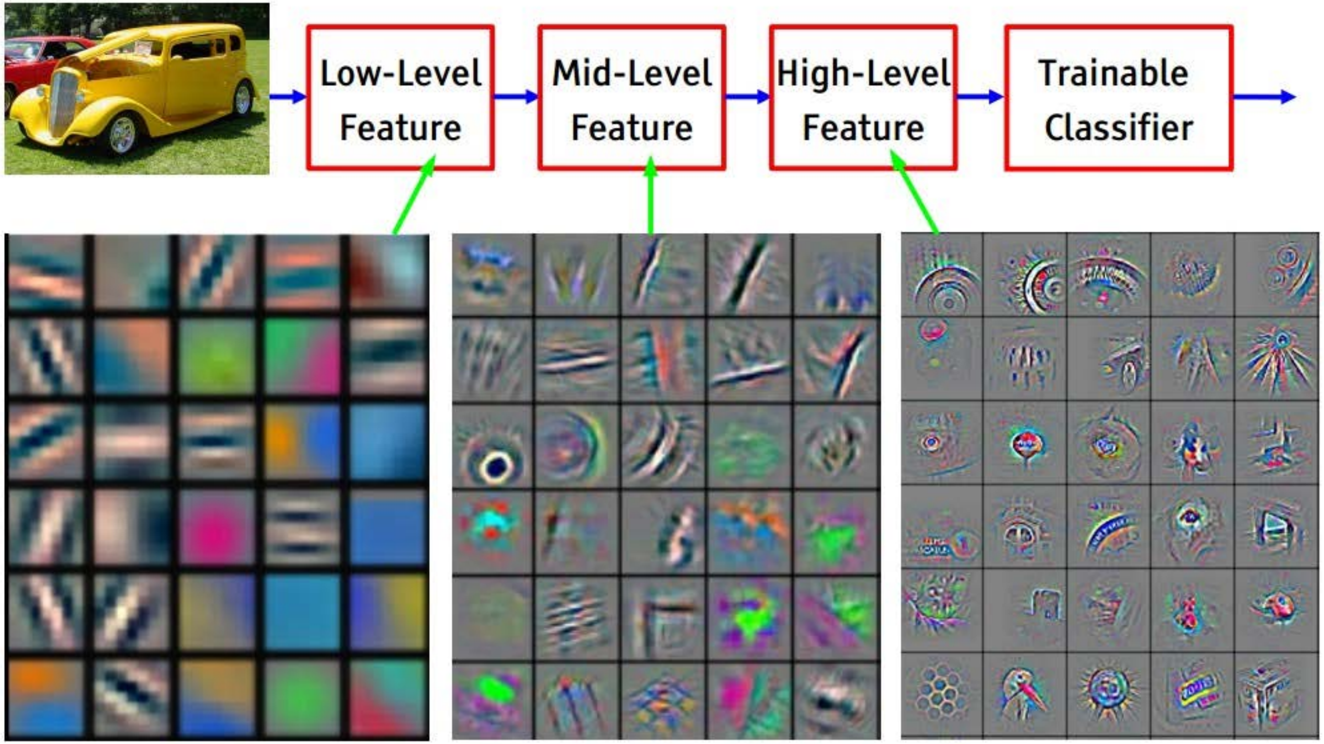
Deep Learning

Learnable kernel

$\phi(x)$

simple classifier



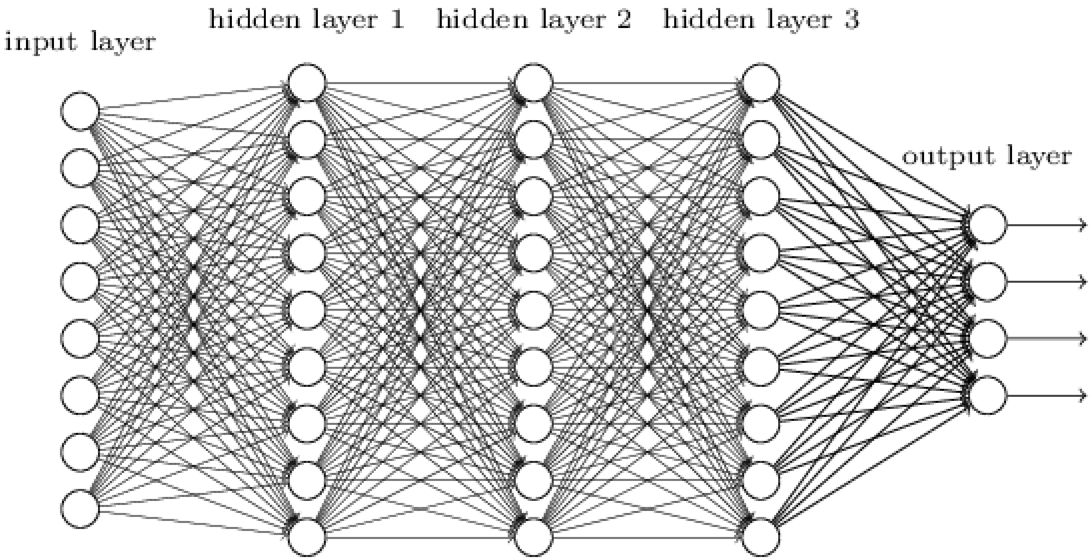


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

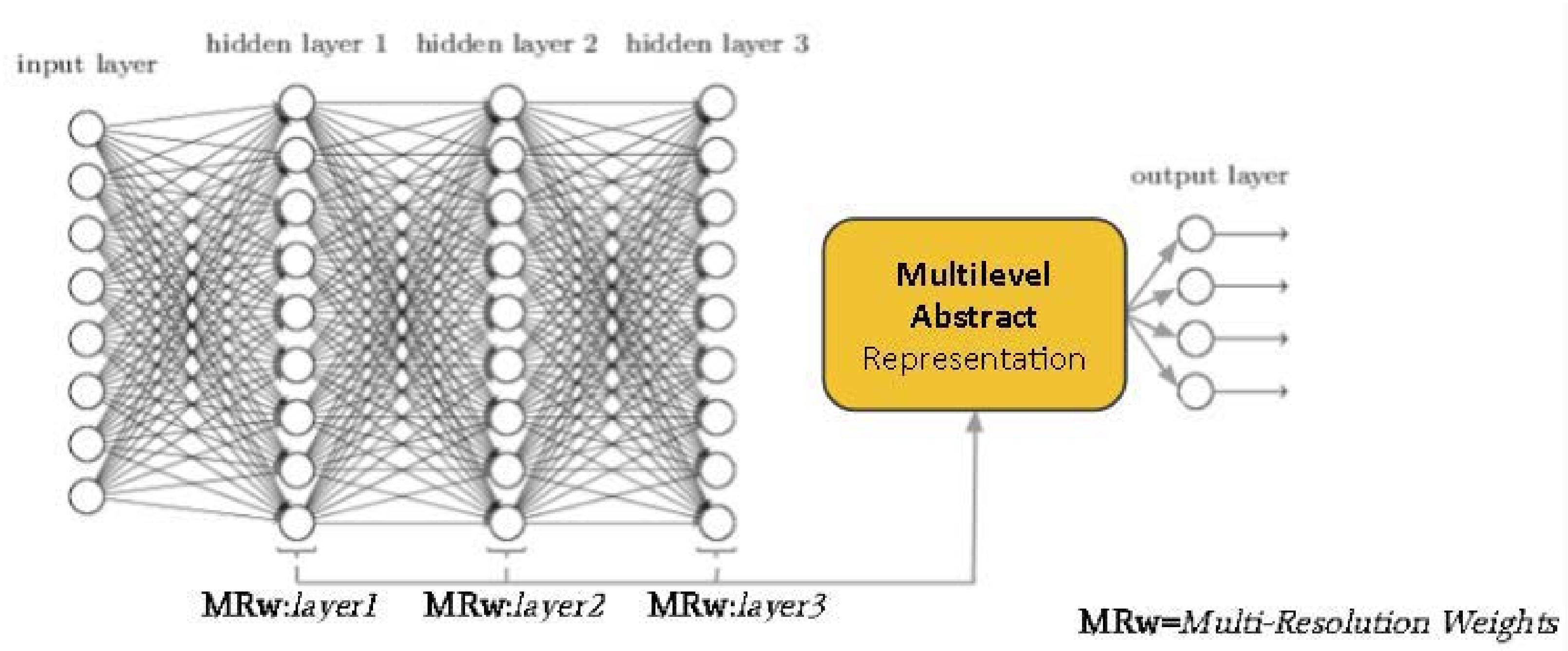
Deep learning representations

- A multilayer neural network is a composite function.
- Each layer is a function that transform the output of previous layer.
- As result each layer is a learned representation.
- The layer before output layer produces the final representation that is used for the machine learning task.

Traditional Deep Learning Approach



Proposed Multi-Resolution Approach



Multi-Resolution Models for Learning Multilevel Abstract Representation of Text

- Word embedding
- Document retrieval for question answering

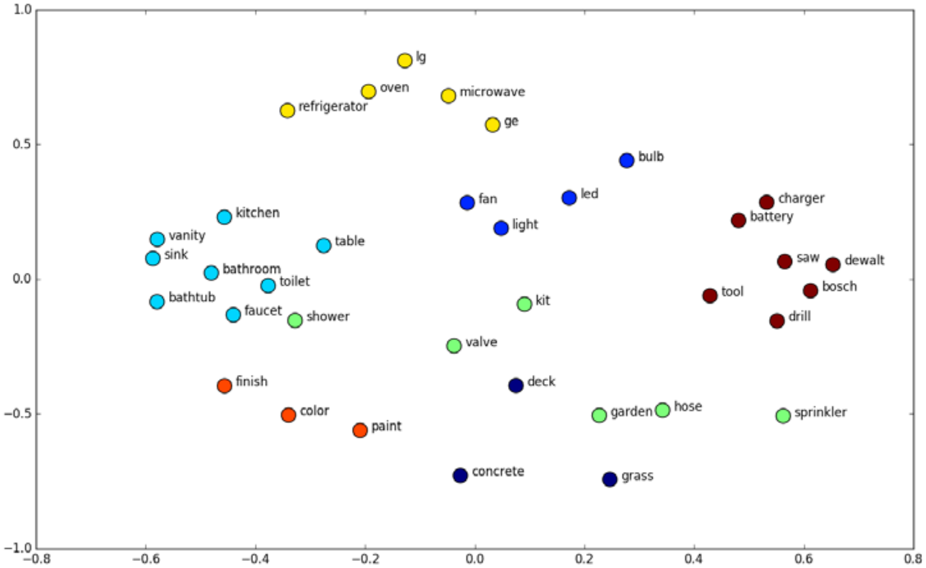
What is word embedding

Word embedding transforms words into vectors of embedded space that keep the syntax and semantics.

It is the first step for NLP, IR, and deep text mining.

It typically use a pretrained deep neural network language model such as Word2Vec, GloVe, ELMo, BERT and fastText.

Words are clustered



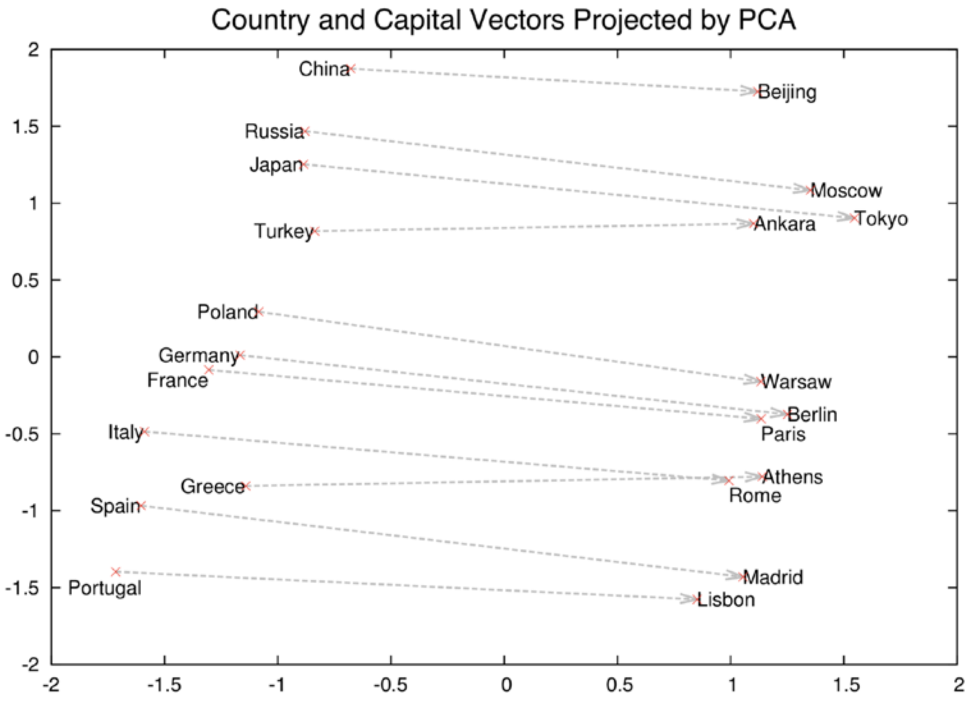
Work with unseen/misspellings/slang words

```
custom_word_vectors.most_similar('thx')
```

```
[('thanks', 0.4921847879886627),
 ('thankyou', 0.4289834201335907),
 ('thansk', 0.3909286856651306),
 ('tks', 0.3625342845916748),
 ('thanx', 0.36105877161026),
 ('thnaks', 0.3544262647628784),
 ('plz', 0.3251364529132843),
 ('thnx', 0.3166268169879913),
 ('cheers', 0.3164141476154327),
 ('thnks', 0.313978642225265 5)]
```

Semantic relationship

- (Paris) – (France) + (Germany) = (Berlin)
- (king) – (man) + (woman) = (queen)



Vector algebra

```
In [25]: model.similar_by_vector(model['Obama'] - model['USA'] + model['France'])
Out[25]: [('Sarkozy', 0.704483687877655),
          ('Obama', 0.701500236988067 6),
          ('President_Nicolas_Sarkozy', 0.642586350440979),
```

```
In [15]: model.similar_by_vector(model['Dublin'] - model['Ireland'] + model['France'])
Out[15]: [('Paris', 0.740702748298645),
          ('France', 0.6797398924827576),
          ('Issy_les_Moulineaux', 0.6246635317802429),
```

One hot encoding

- The vector is n-dimensional, n is the size of the vocabulary.
- Each word is a dimension, the encoding has only a single 1 in the dimension of the corresponding word.

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

Issues with one hot embedding

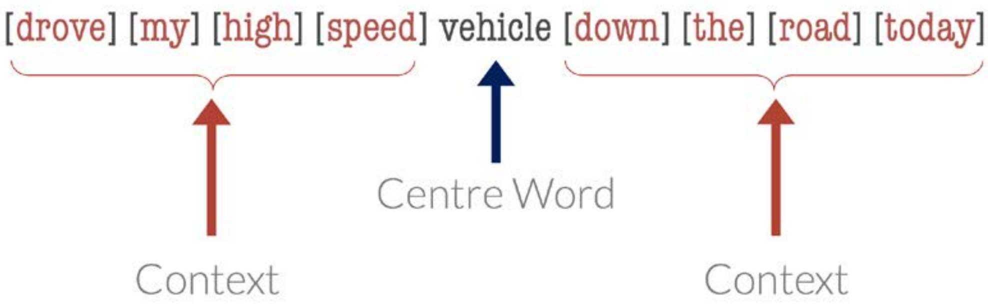
Very high dimensional

Very sparse

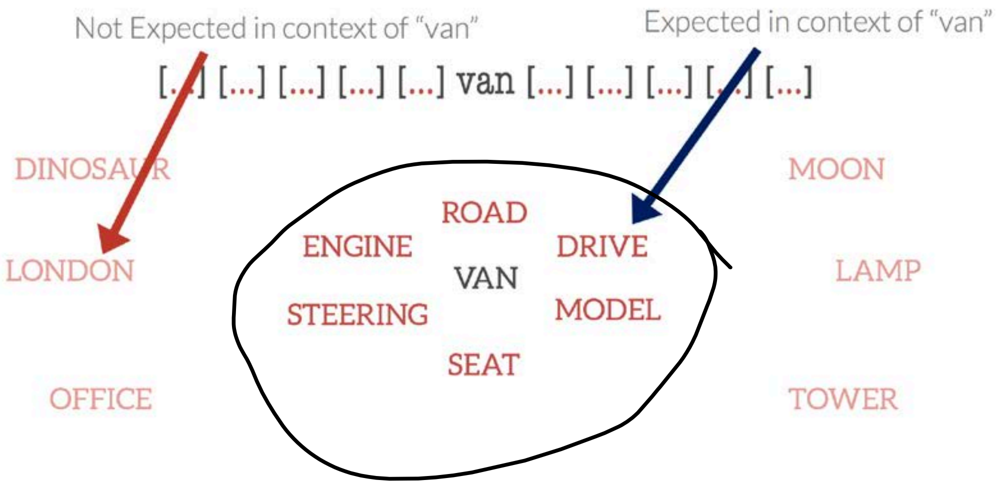
No syntax and semantic relationships of words

Word embedding algorithms

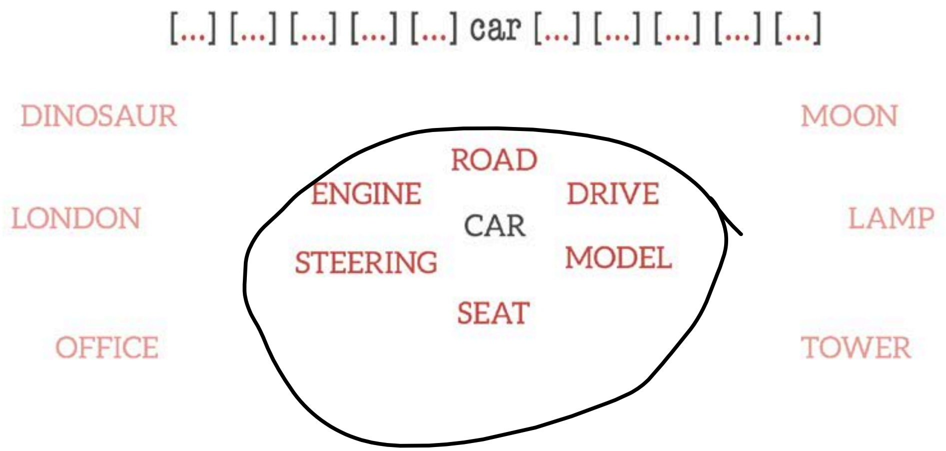
- The goal is to optimize the embeddings such that the meanings and the relationships between words is maintained.
- The main idea is that the surrounding words (context) for any word are useful to capture the meaning of that word, called center word.



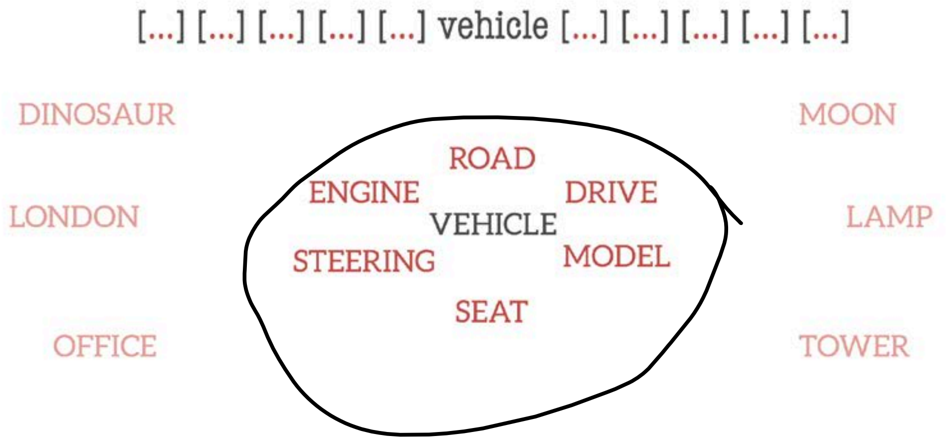
So similar words are typically surrounded by the same "context" words



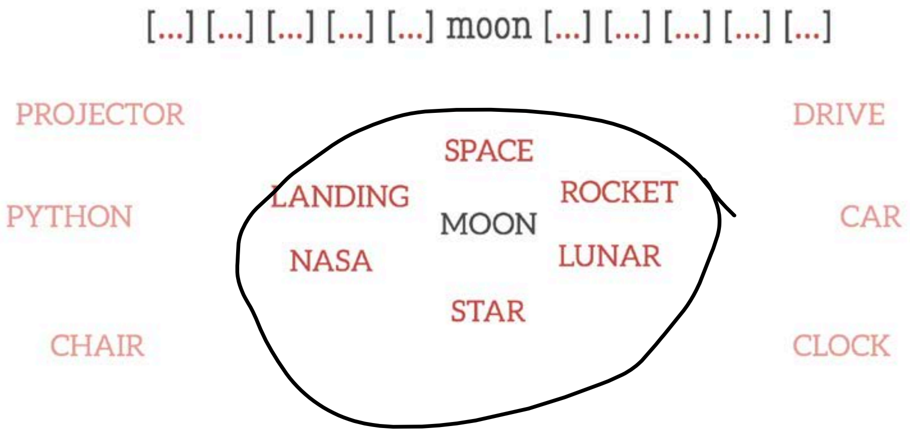
So similar words are typically surrounded by the same "context" words



So similar words are typically surrounded by the same "context" words



So similar words are typically surrounded by the same "context" words



Word2Vec

Developed by Google with two models to predict word and context

Skip-Gram

- One layer neural network model to predict the context of a center word

CBOW

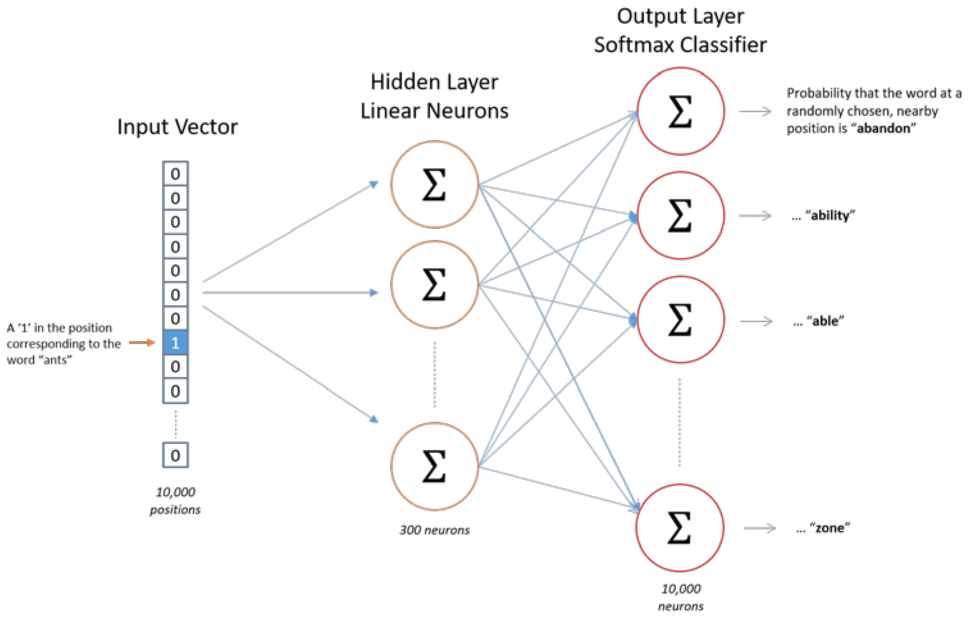
- One layer neural network model to predict the center word of a context

Training data

Window size = 2

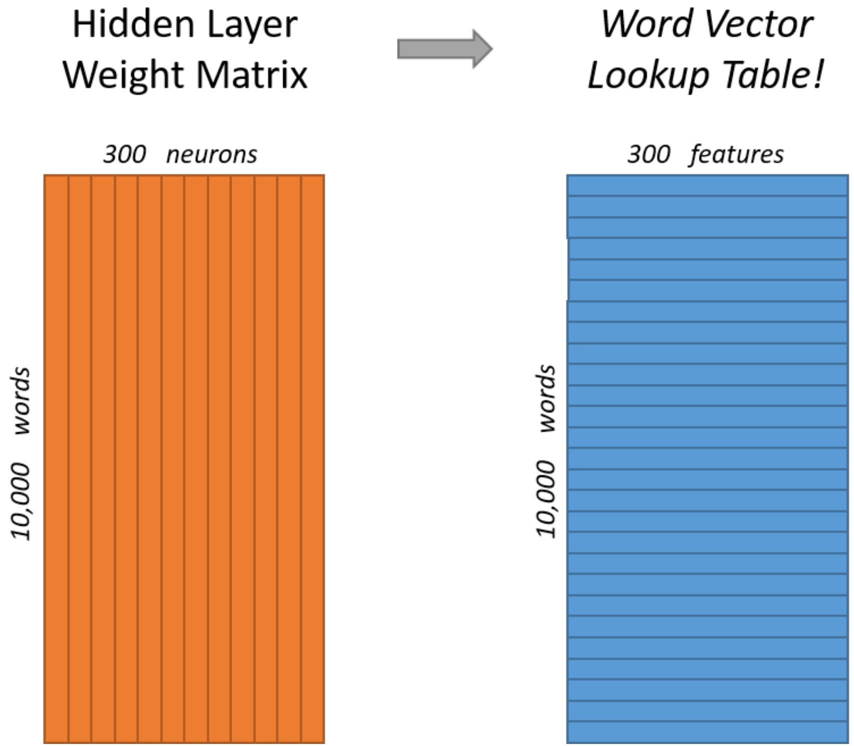
Source Text	Training Samples
The quick brown fox jumps over the lazy dog. ⟹	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog. ⟹	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog. ⟹	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over the lazy dog. ⟹	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Skip-Gram


Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"

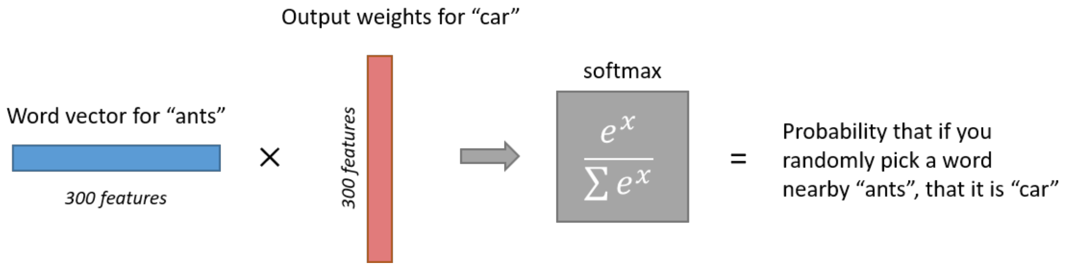
The hidden layer

- There is no activation function.
- The number of neurons is a hyper parameter, determines the dimension of the learned word vector.
- The learned weigh matrix gives the word vectors.



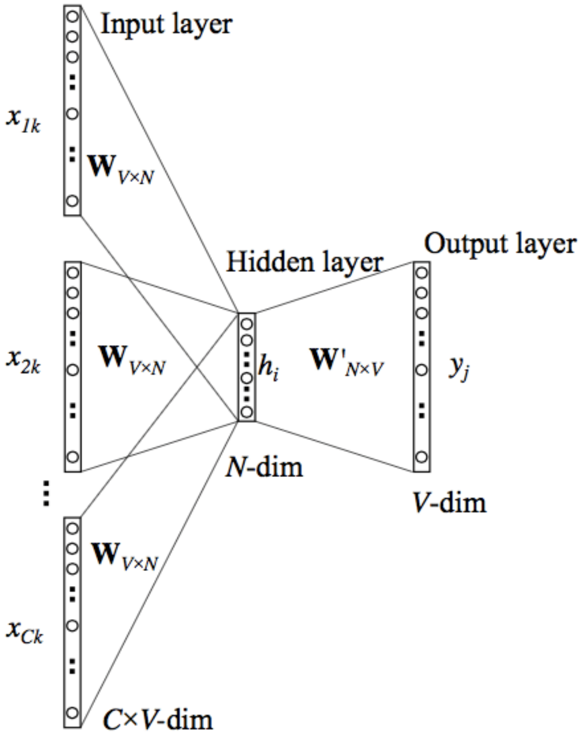
Output layer

- The output layer is a vector of probabilities of word in the context.
- The probability is estimated using SoftMax.
- Each word of the output layer has a weight vector, consisting one weight for each neuron.
- The output layer weight vector multiplies again the vector from the hidden layer.
- The result applies to the SoftMax function.



Continuous bag of words (CBOW)

- Use context words to predict center work.
- Reverse the input and target of training data.



Skip-gram or CBOW?

- Skip Gram works well with small amount of data and is found to represent rare words well.
- On the other hand, CBOW is faster and has better representations for more frequent words.

Training

- Subsampling frequent words like “the” in the training data.
- Negative sampling.
- Hierarchical SoftMax.
- Cross entropy loss function.
- Dimensionality typically around 100 to 1000.
- Context window typically 10 for skip-gram and 5 for CBOW.

Subsampling

- There are two "problems" with common words like "the":
  - When looking at word pairs, ("fox", "the") doesn't tell us much about the meaning of "fox". "the" appears in the context of pretty much every word.
  - We will have many more samples of ("the", …) than we need to learn a good vector for "the".
- Subsampling allows delete a word proportional to the frequency.
- The probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

- $z(w)$ is fraction of word count w.

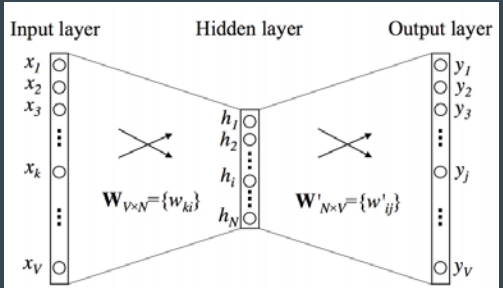
Negative sampling

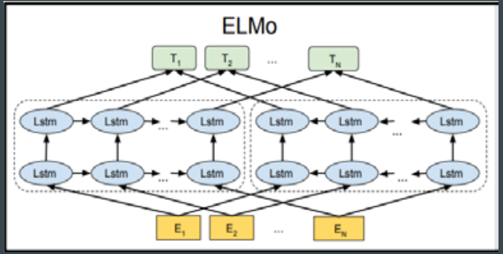
- The target is one-hot vector, only one out neuron outputs 1 (positive word) , the rest output 0 (negative word).
- Negative sampling randomly selects just a small number of “negative” words (let’s say 5) to update the weights for.
- If the size of vocabulary is 10,000 and the number of hidden neurons is 300, then the number of updating weights is 300x(1+number of sampled negative words) instead of 3 millions. In case # sampled negative words = 5, the saving is 94%!
- Frequent words are more likely to be selected as negative words.

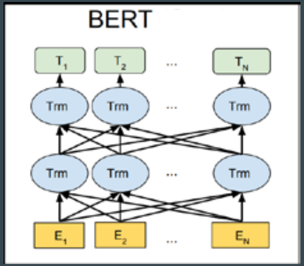
$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n}\left(f(w_j)^{3/4}\right)}$$

Embedding Models

- **Global Vectors (GloVe):**
  - Stanford - Pennington et al. (2014)
  - Context-Free (CF)
  - Dataset: Common Crawl
  - Embedding: (1 x 300) dimensional word vectors
- **fastText:**
  - Facebook - Mikolov et al. (2018)
  - Context-Free (CF)
  - Dataset: Common Crawl
  - Embedding: (1 x 300) dimensional word vectors
- **Embedding from Language Models (ELMo):**
  - Washington Uni./Microsoft/Allen Inst. - Peters et al. (2018)
  - Contextual (C), BiDirectional
  - Dataset: 1 Billion Word Benchmark.
  - Embedding: (3 x 1024) dimensional word matrices.
- **Bidirectional Encoder Representations from Transformers (BERT):**
  - Google - Devlin et al. (2018)
  - Contextual (C), BiDirectional
  - Dataset: Wikipedia + BookCorpus
  - Embedding: (24 x 768) dimensional word matrices.

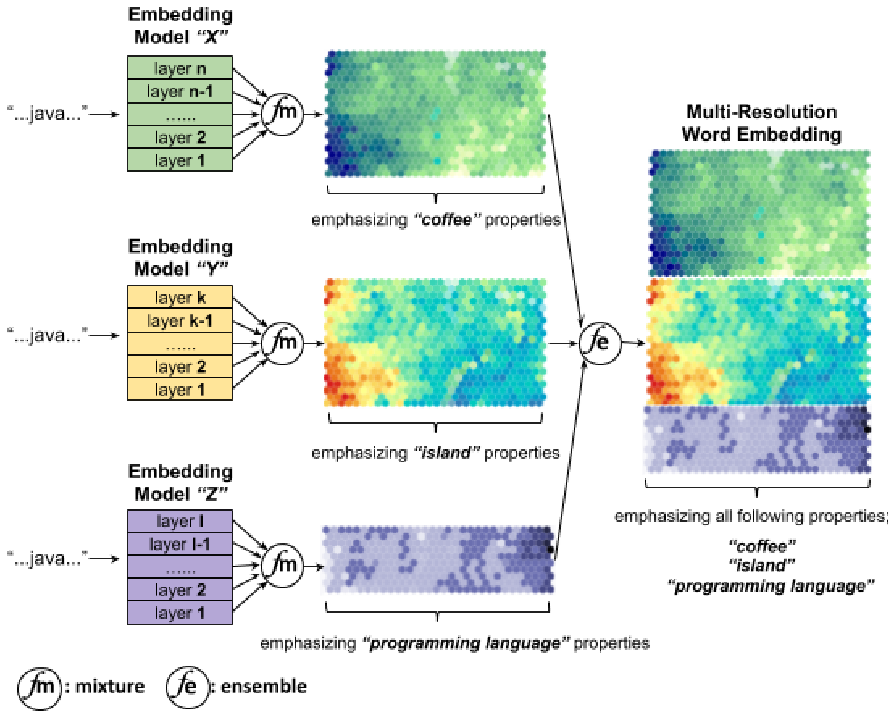






How actually Multi-Resolution Word Embedding works?

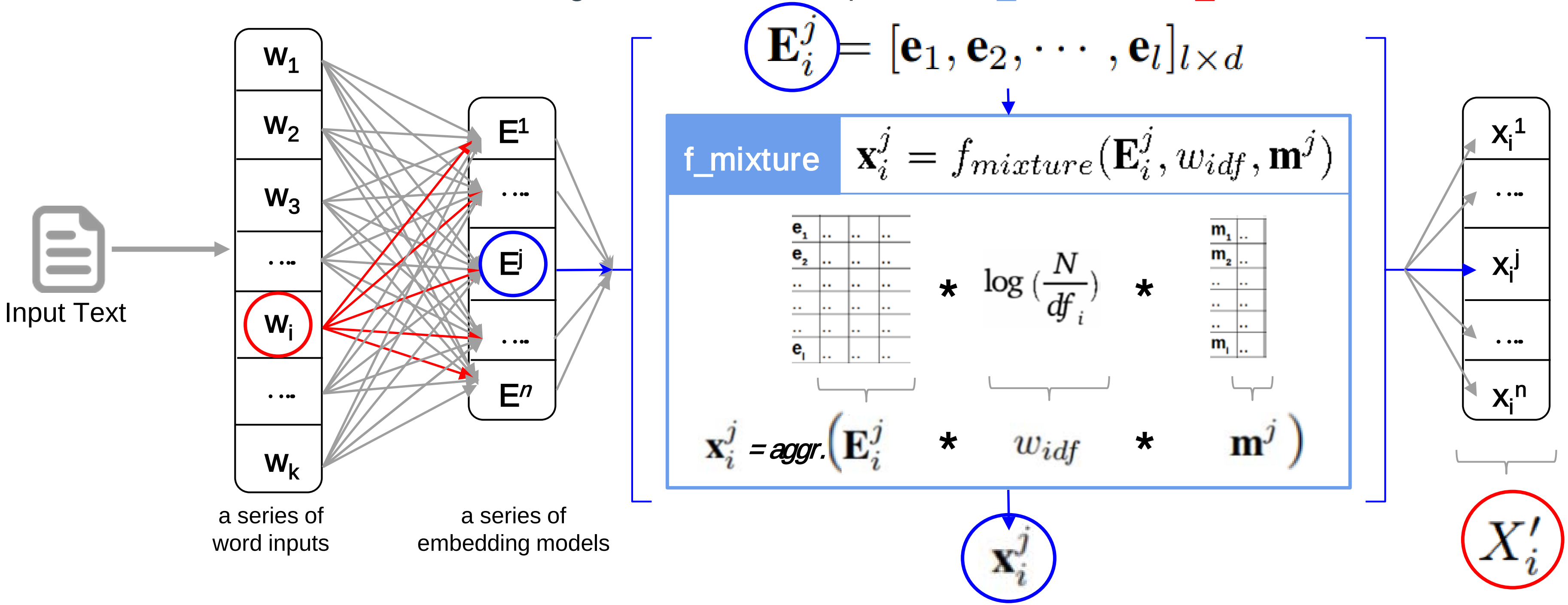
The illustration of multi-resolution word embedding method using an example of ”· · · java · · · ”



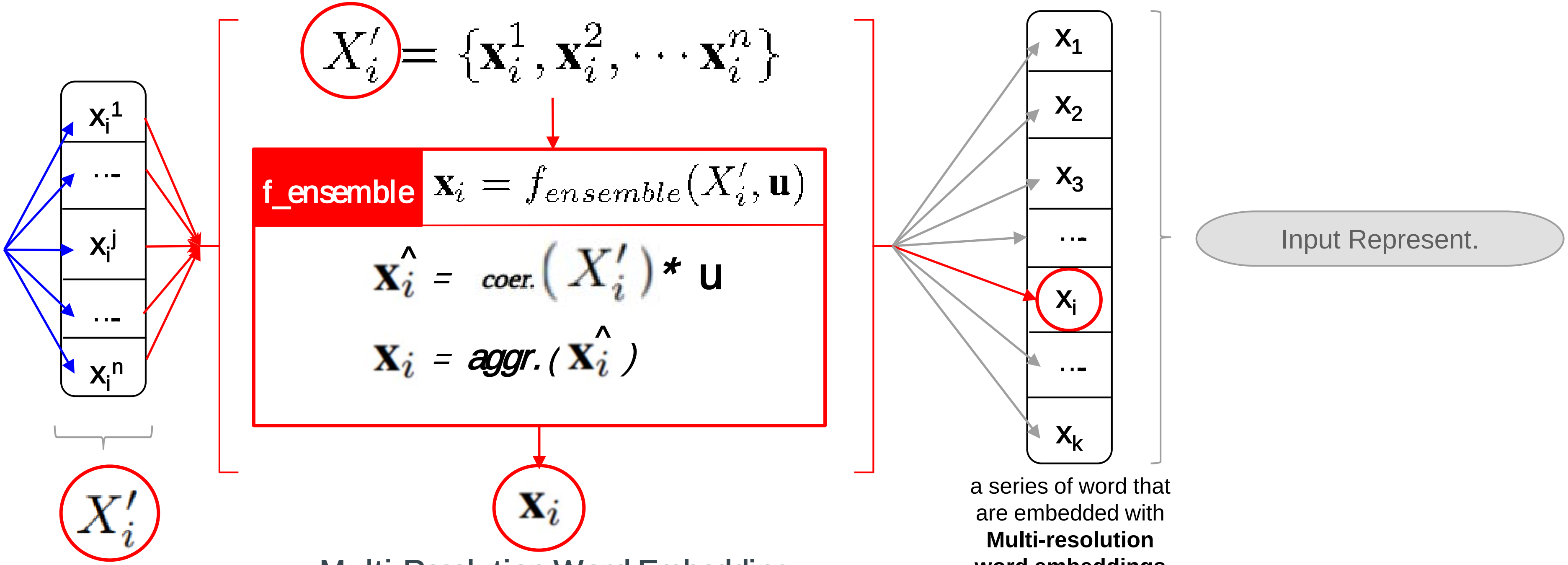
How actually Multi-Resolution Word Embedding works? *(cont.)*

What is happening in **Generate Input Representation with Multi-Resolution Word Embedding**

The **multi-resolution word embedding** has two cascaded operations: f_mixture and f_ensemble.



How actually Multi-Resolution Word Embedding works? *(cont.)*

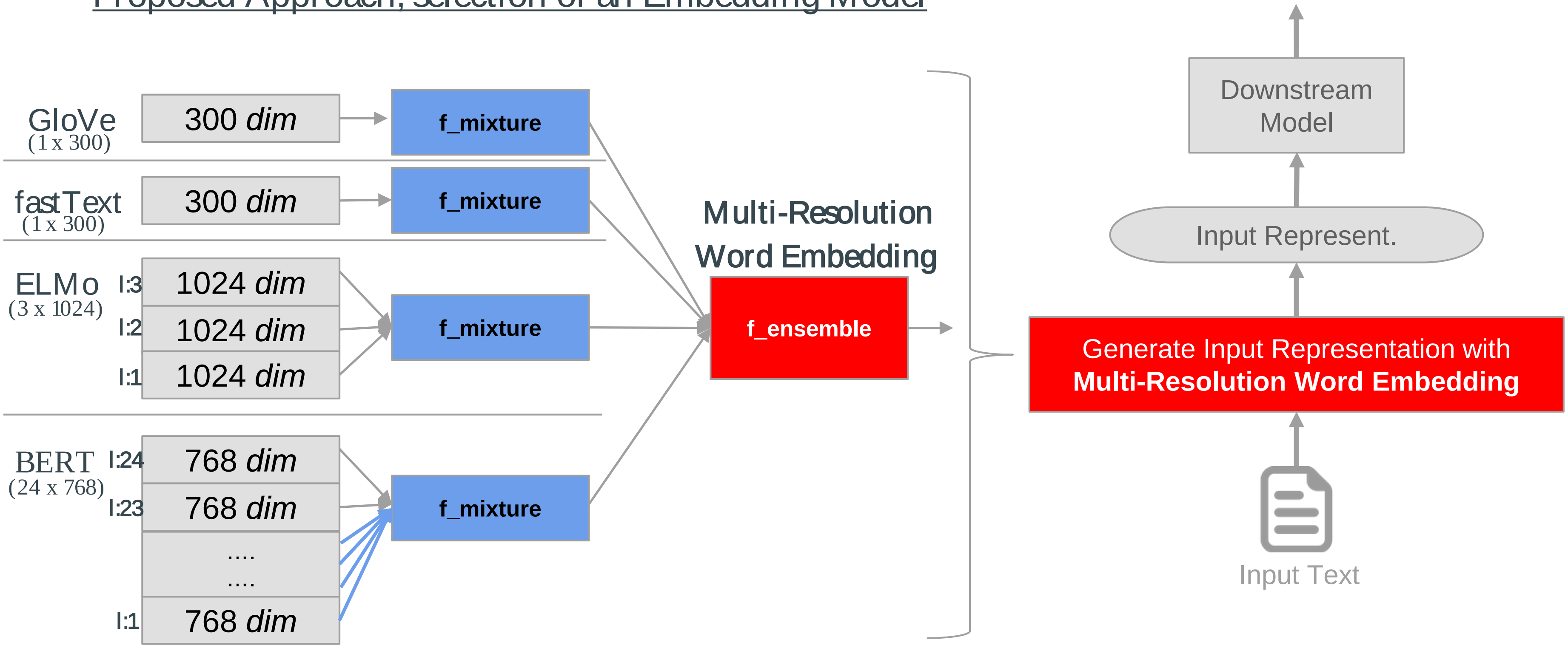


Multi-Resolution Word Embedding for word *i*-th

a series of word that are embedded with **Multi-resolution word embeddings**

Multi-Resolution Word Embedding for QA Document Retrieval

- Proposed Approach, selection of an Embedding Model



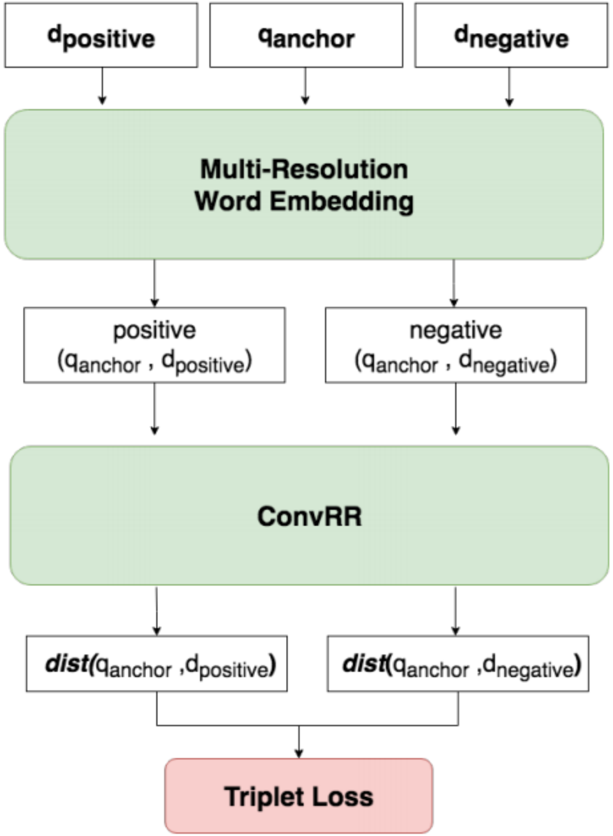
Datasets

Four large and popular question-answering datasets:

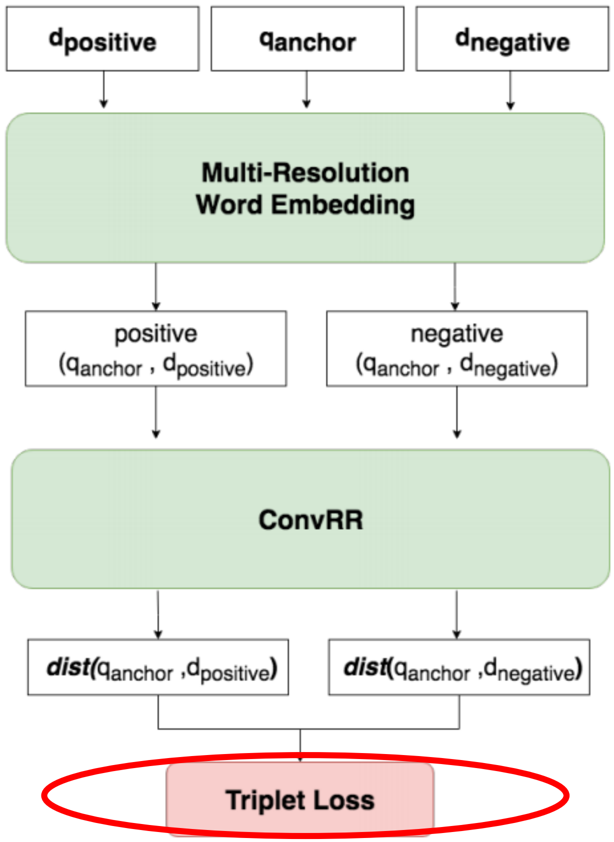
- **SQuAD:** The Stanford Question Answering Dataset (SQuAD)
  - 100,000+ questions derived from a set of Wikipedia documents.
- **WikiQA:** The Wikipedia open-domain Question Answering (WikiQA)
  - 1,000+ questions derived from Wikipedia pages.
- **QUASAR:** The Question Answering by Search And Reading
  - 43,012 open-domain questions collected from the ClueWeb09 dataset (Callan et al. (2009)).
- **TrecQA:** Text Retrieval Conference Question (TrecQA)
  - 1,000+ factoid questions from crowdworkers.

DATASET	TRAIN	VALID.	TEST	TOTAL
SQUAD	87,599	10,570	HIDDEN	98,169+
WIKIQA	873	126	243	1,242
QUASAR-T	37,012	3,000	3,000	43,012
TRECQA	1,162	65	68	1,295

Multi-Resolution Word Embedding for QA Document Retrieval



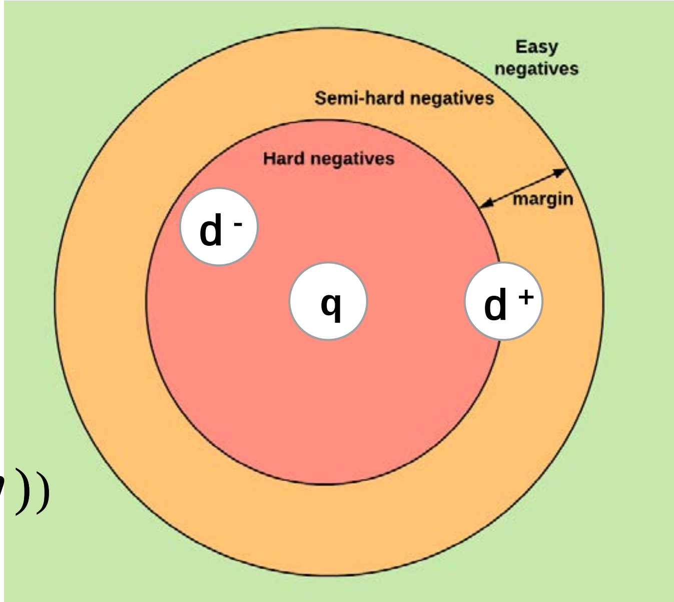
Multi-Resolution Word Embedding for QA Document Retrieval



Multi-Resolution Word Embedding for QA Document Retrieval

- **Triplet Loss:** *Distance Metric Learning Loss Function*
- *Hard Triplets Mining Strategy* to select triplets.

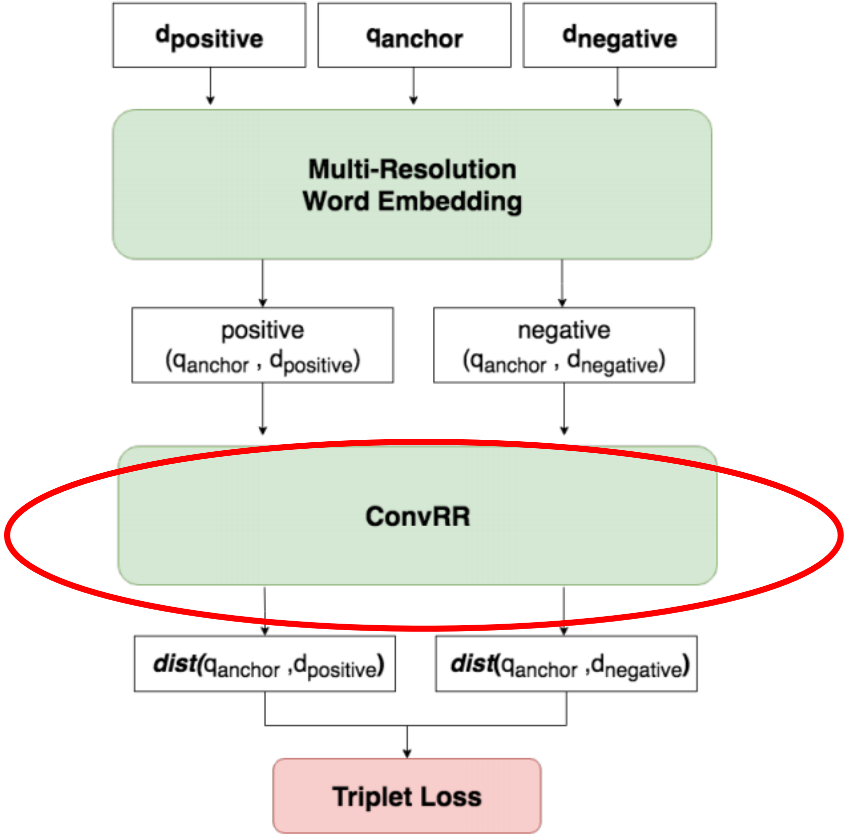
$$T = (\boldsymbol{d}_{\text{positive}}, \boldsymbol{q}_{\text{anchor}}, \boldsymbol{d}_{\text{negative}})$$



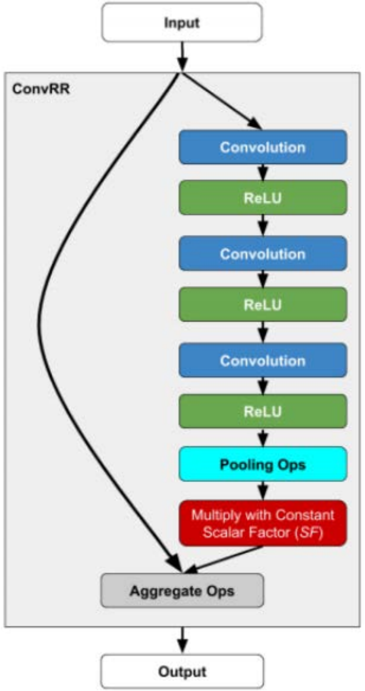
$$L_{triplet} = \max(0, (||q_{anchor}, d_{positive}|| - ||q_{anchor}, d_{negative}|| + m))$$

m is a scalar value, namely margin; $m > 0$

Multi-Resolution Word Embedding for QA Document Retrieval



Convolutional Residual Retrieval Network (ConvRR)



$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \cdots, \mathbf{x}_k]_{k \times d''}$$

$$\mathbf{X}'' = f(\mathbf{W}, \mathbf{X}, sf)$$

$$\mathbf{o} = \mathbf{X}'' + \frac{1}{k}\sum_{i=1}^{k} \mathbf{x}_i$$

$$L_2norm\,(\,\mathbf{o}\,)$$

Training Configuration

- ADAM optimizer, by Kingma & Ba (2014), with a learning rate of 10^{-3}.
- Randomization is fixed.
- Weight Decay is set to 10^{-3}
- For Convolutions:
    - Windows-size **ws** = 5
    - Number of kernel **d''** = 4, 372
- Scaling factor sf = 0.05
- 400 iterations with a batch size of 2, 000 using a triplet loss (*with a margin m = 1*).
- Implemented with Tensorflow 1.8+ on 2 × NVIDIA Tesla K80 GPUs.

Results

recall@k

- The number of correct documents listed within top-k order out all possible candidates.

SQuAD

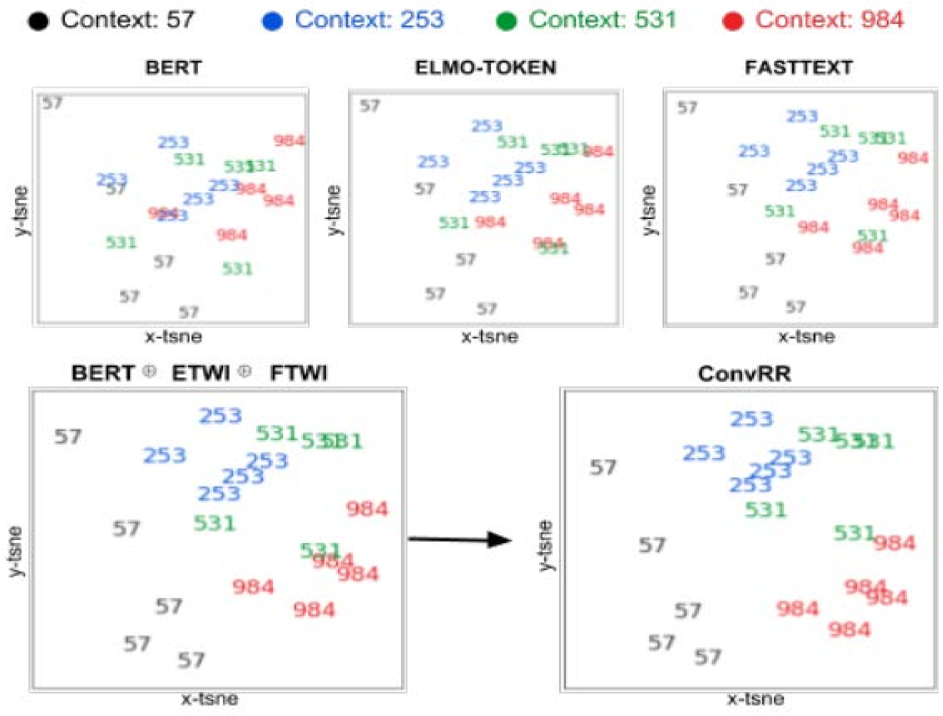
Embedding/Model	@1	@3	@5
Base Embeddings			
TF-IDF	8.77	15.46	19.47
BERT	18.89	32.31	39.52
ELMO-AVG	21.24	36.24	43.88
GLOVE	30.84	47.14	54.01
FASTTEXT	42.23	59.86	67.12
Multi-Resolution Emb. (w/o Ensemble)			
ELMO-LSTM1	19.65	34.34	42.52
BERT w/ IDF	21.81	36.35	43.56
ELMO-LSTM2	23.68	39.39	47.23
ELMO-TOKEN	41.62	57.79	64.36
ELMO-TOKEN w/ IDF	44.85	61.55	68.07
FASTTEXT w/ IDF	45.13	62.80	69.85
Multi-Resolution Emb. (w/ Ensemble)			
ETwI ⊕ FTwI	46.33	63.13	69.70
BERT ⊕ ETwI ⊕ FTwI	48.49	64.96	71.05
Base Embedding + Downstream Models			
FASTTEXT + FCRR	45.7	63.15	70.02
FASTTEXT + ConvRR	47.14	64.16	70.87
Multi-Resolution Emb. + Downs. Models			
BERT ⊕ ETwI ⊕ FTwI + FCRR	50.64	66.16	73.44
BERT ⊕ ETwI ⊕ FTwI + ConvRR	**52.32**	**68.26**	**75.68**

QUASAR-T

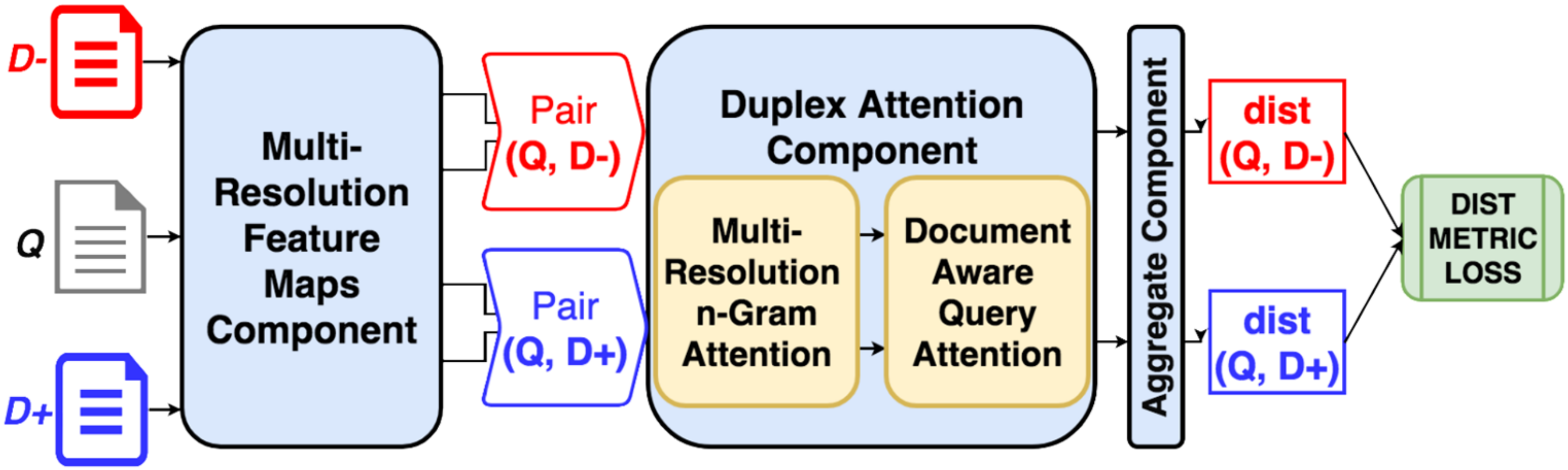
Embedding/Model	@1	@3	@5
Base Embeddings			
TF-IDF	13.86	20.2	23.13
BERT	25.5	34.2	37.86
ELMO-AVG	27.93	37.86	42.33
GLOVE	32.63	40.73	44.03
FASTTEXT	46.13	56.00	59.46
Multi-Resolution Emb. (w/o Ensemble)			
ELMO-LSTM1	24.6	33.01	36.9
ELMO-LSTM2	27.03	36.33	40.56
BERT w/ IDF	27.33	38.43	40.11
ELMO-TOKEN	44.46	54.86	59.36
ELMO-TOKEN w/ IDF	48.86	60.56	65.03
FASTTEXT w/ IDF	49.66	58.70	61.96
Multi-Resolution Emb. (w/ Ensemble)			
ETwI ⊕ FTwI	48.78	60.05	64.10
BERT ⊕ ETwI ⊕ FTwI	49.46	60.93	65.66
Base Embedding + Downstream Models			
FASTTEXT + FCRR	47.11	58.25	62.12
FASTTEXT + ConvRR	48.17	59.06	63.07
Multi-Resolution Emb. + Downs. Models			
BERT ⊕ ETwI ⊕ FTwI + FCRR	49.55	61.58	64.53
BERT ⊕ ETwI ⊕ FTwI + ConvRR	**50.67**	**63.09**	**67.38**

Visualization

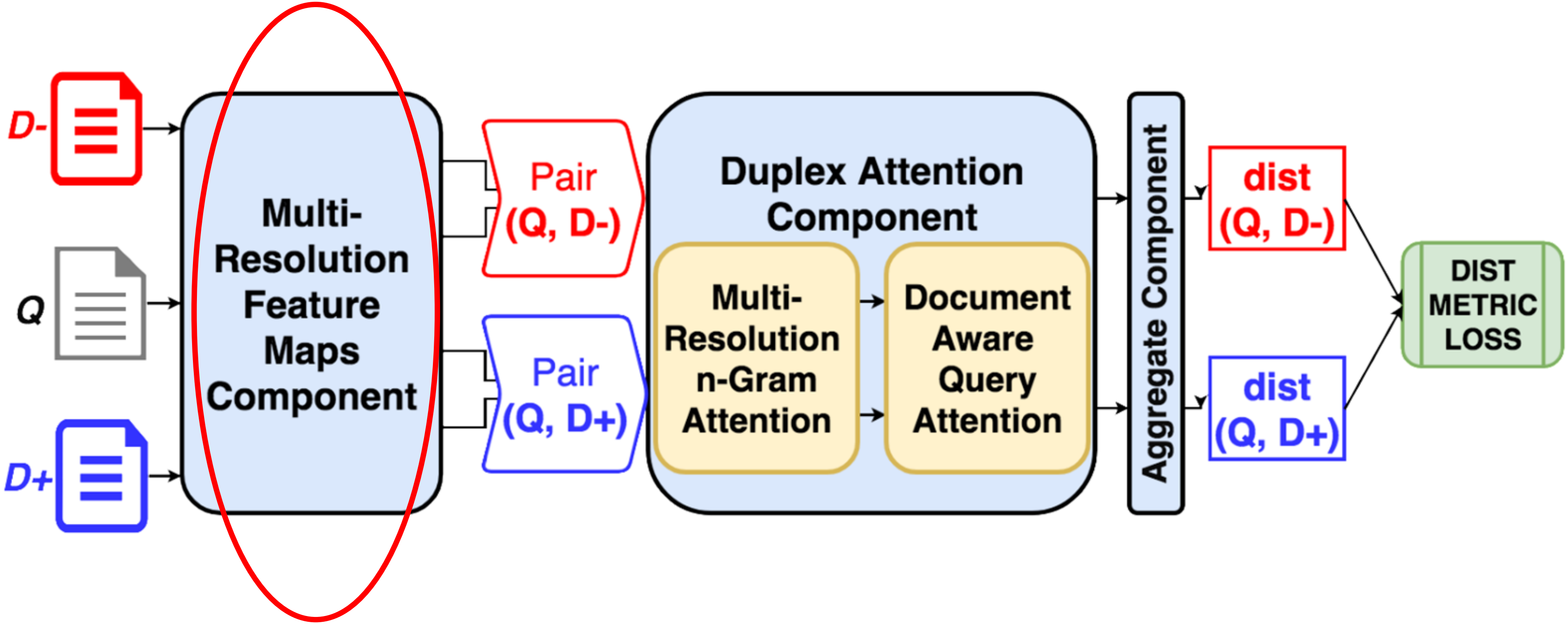
The t-Distributed Stochastic Neighbor Embedding (t-SNE) visualization of question embeddings that are derived using different embedding models.



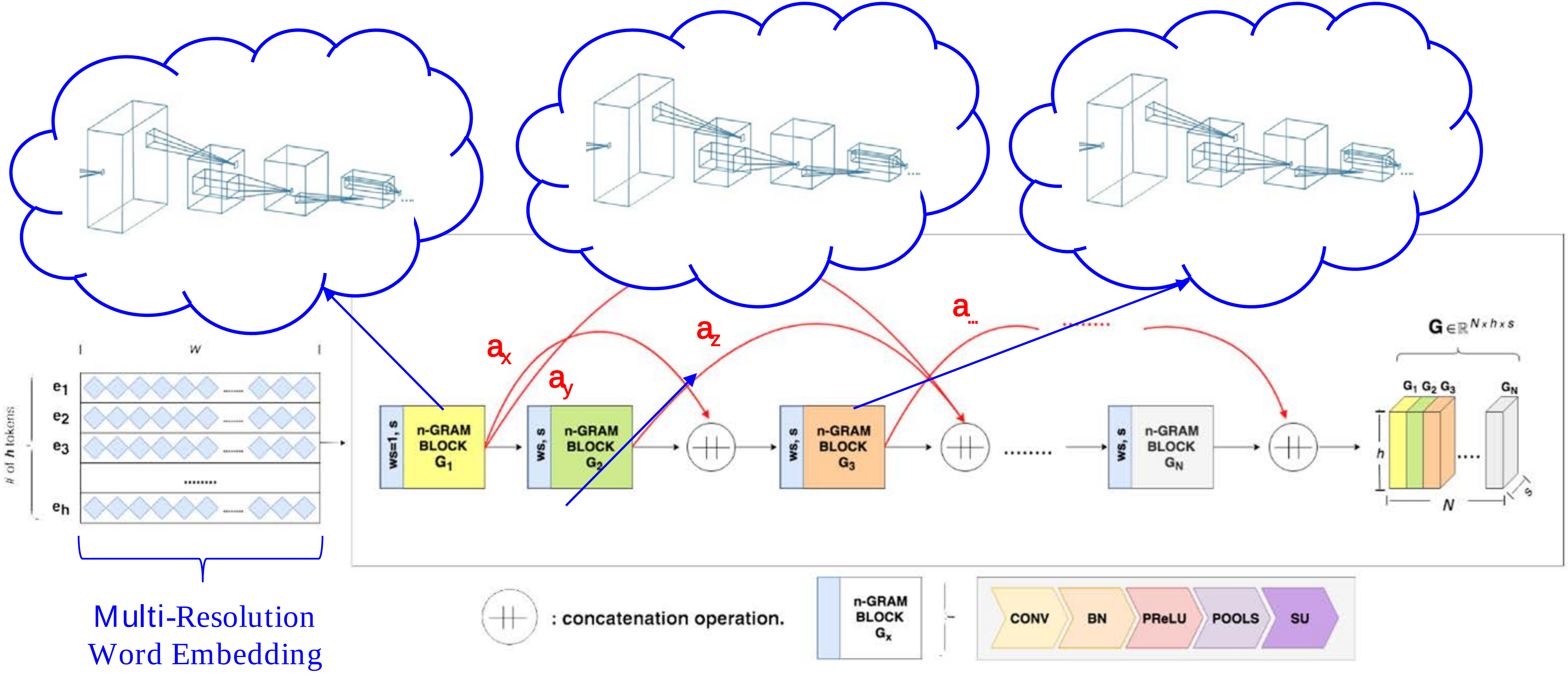
MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval



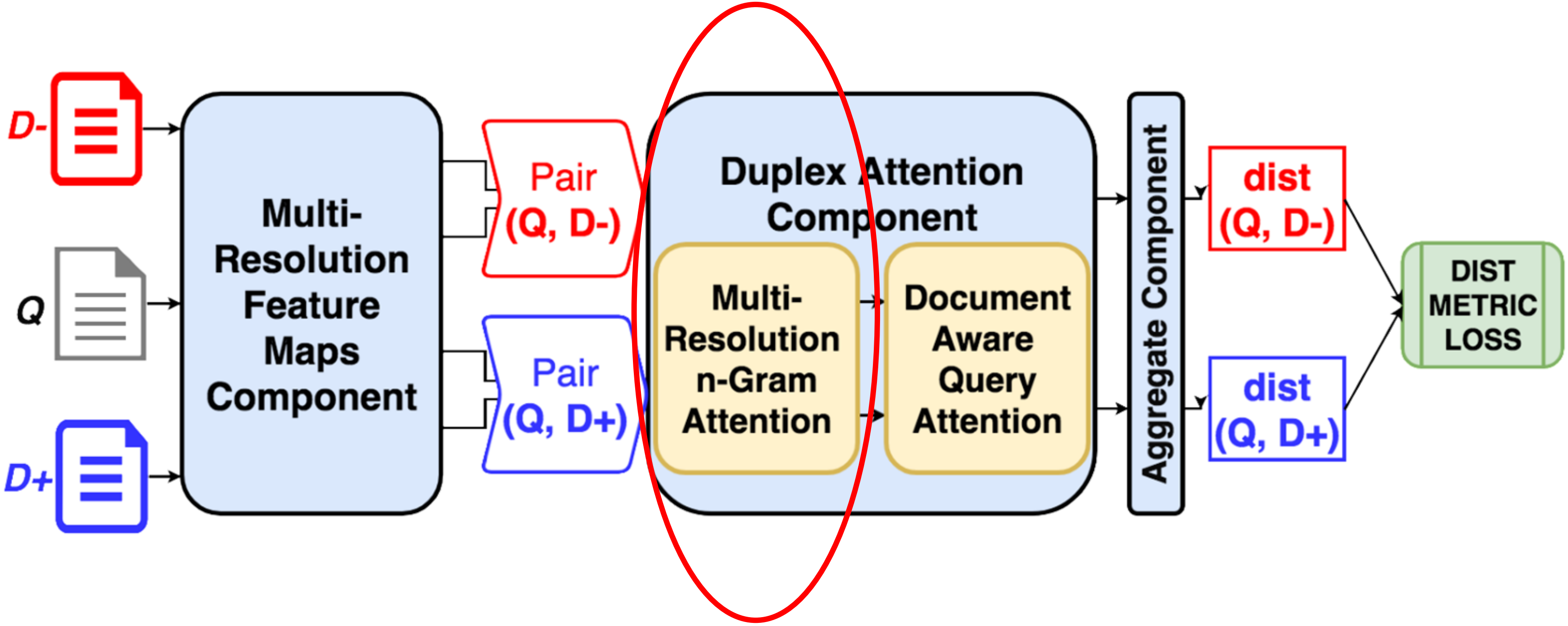
MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*



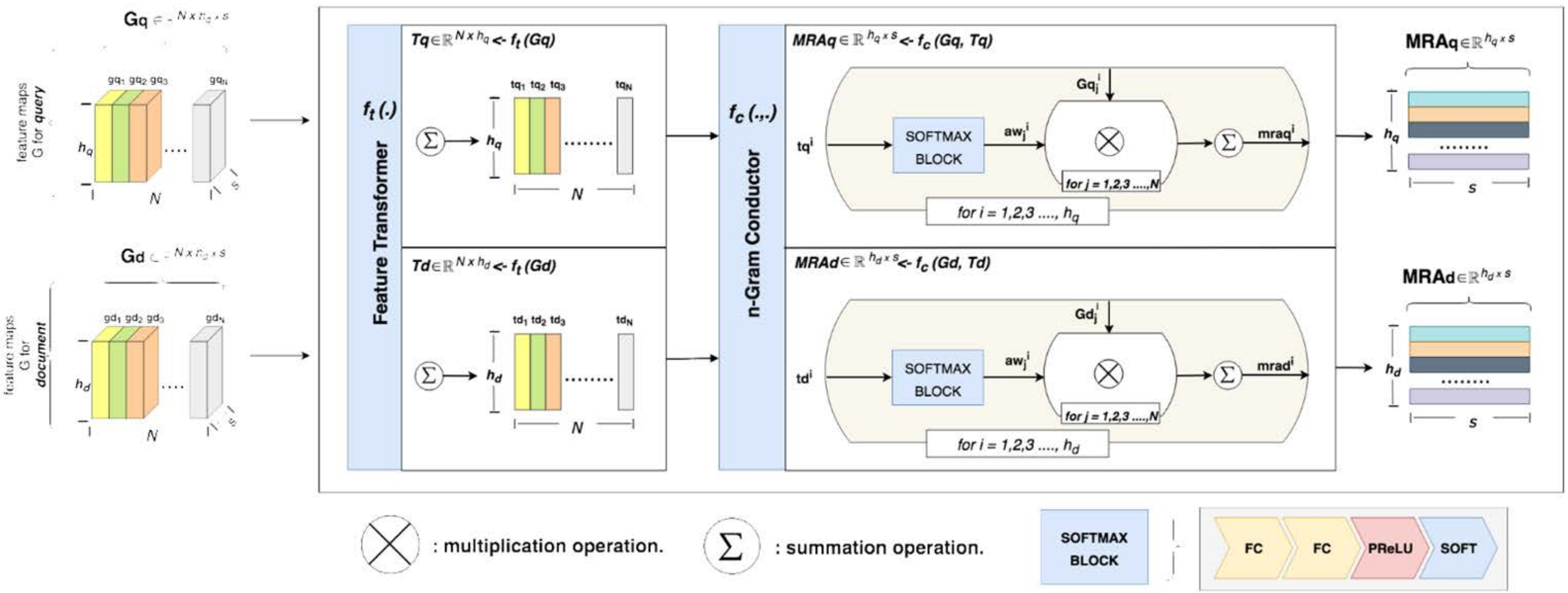
Multi-Resolution Feature Maps Component



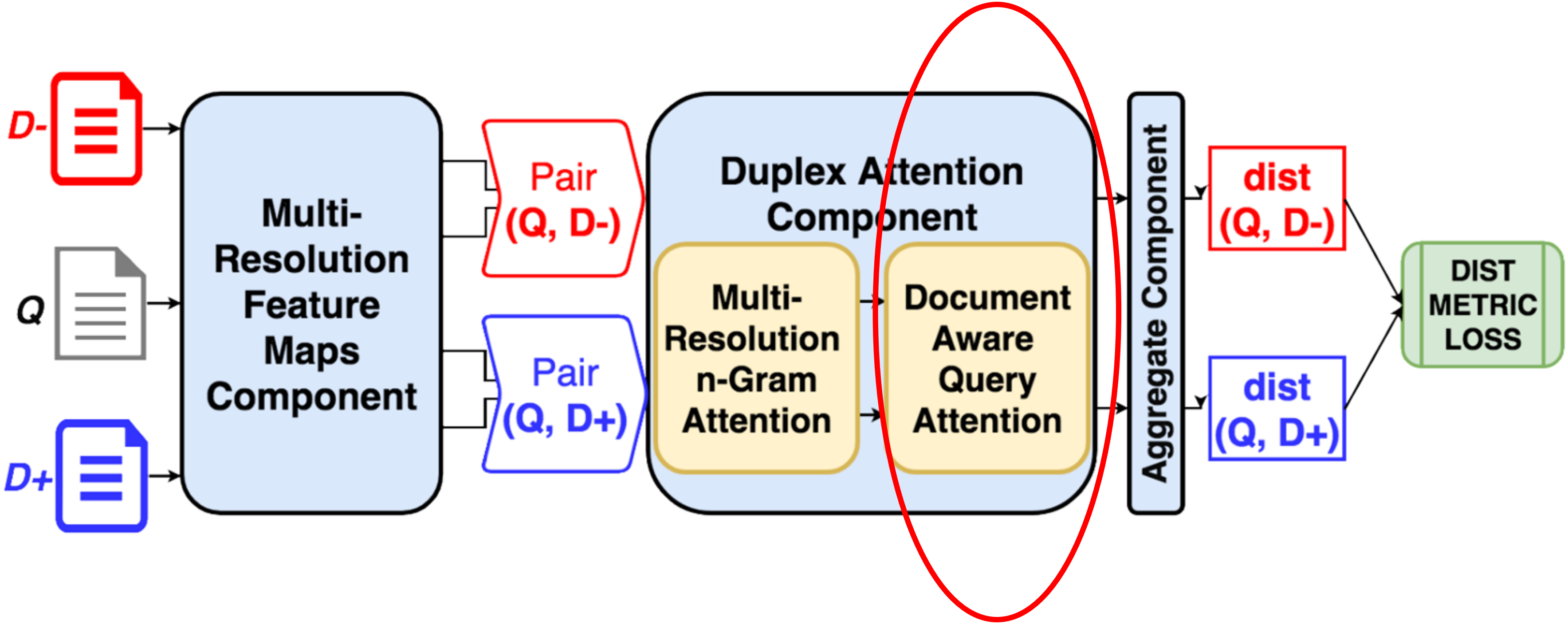
MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*



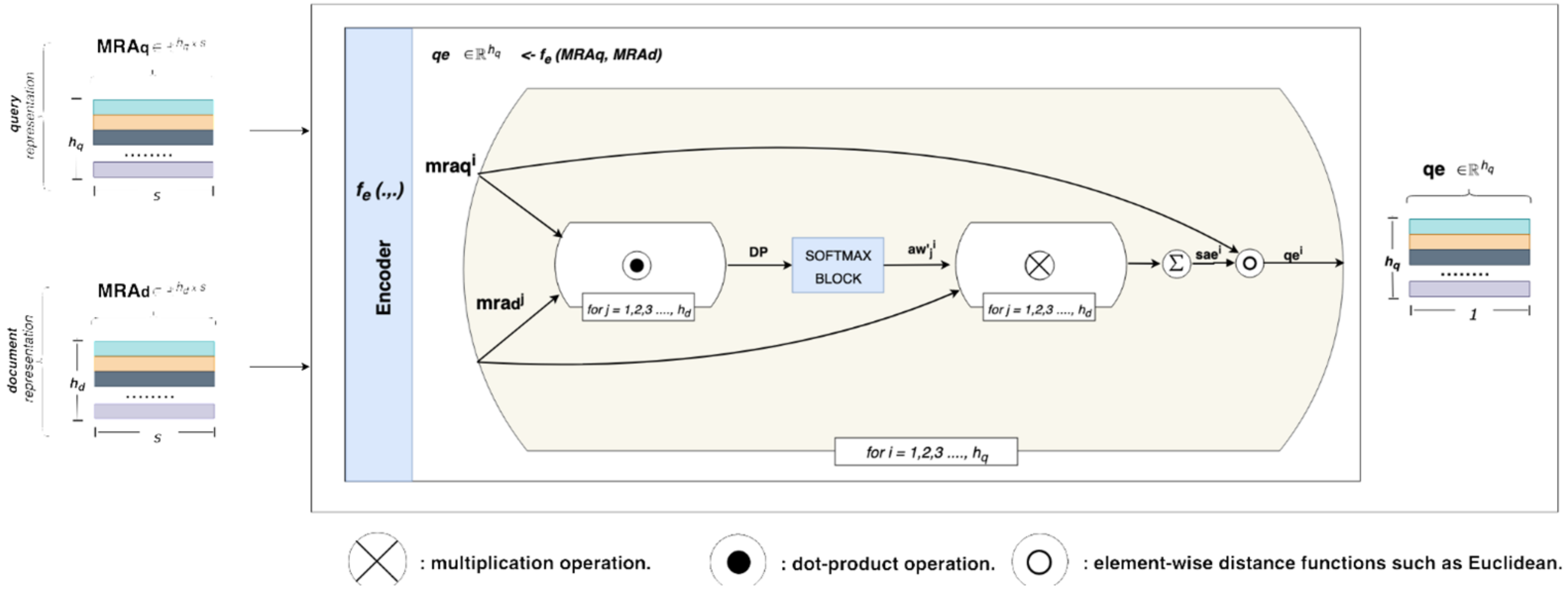
Duplex Attention Component - Multi-Resolution n-Gram Attention



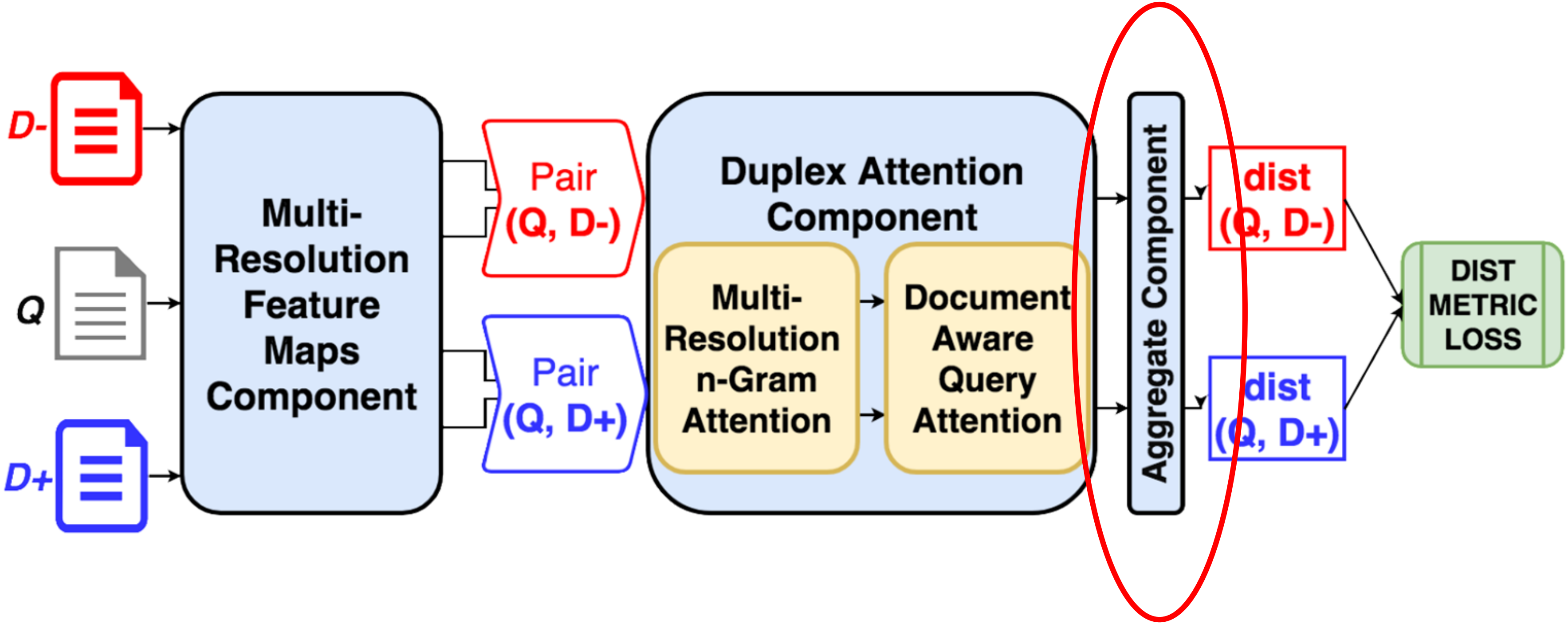
MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*



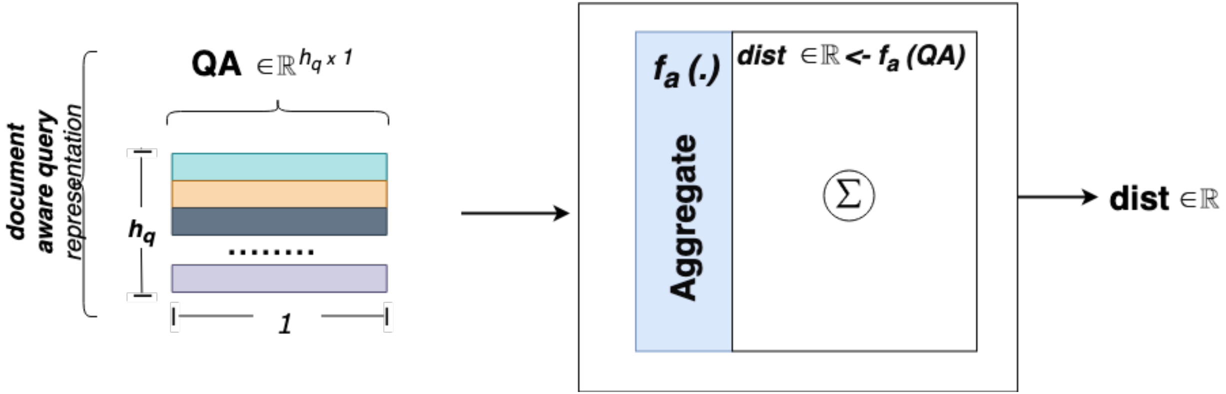
Duplex Attention Component - Document Aware Query Attention



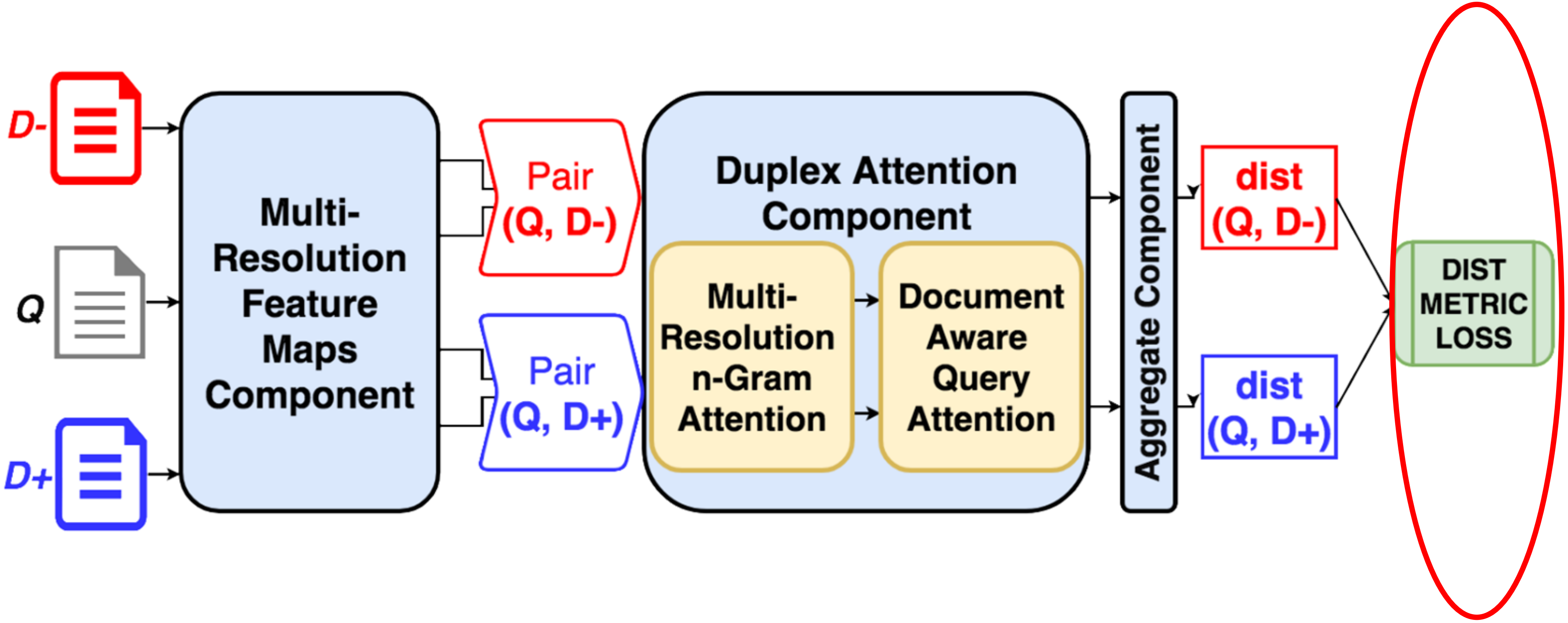
MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*



Aggregate Component



MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*



MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*

Training Configuration:

- ADAM optimizer, by Kingma & Ba (2014), with a learning rate of 10^{-4}.
- Randomization is fixed.
- Weight Decay is set to 10^{-3}
- N-gram blocks :
    - N = 6 (6 x n-gram blocks): SQuAD, QUASAR-T
    - N = 4 (4 x n-gram blocks): WikiQA, TrecQA
    - Windows-size **ws** = 3
    - Number of kernel **s** = 1,024
- A batch size of 512 using a triplet loss:
    - m = 1 : SQuAD
    - m = 0.8 : QUASAR-T
    - m = 0.5 : WikiQA, TrecQA
- Implemented with Tensorflow 1.8+ on 2 × NVIDIA Tesla K40c GPUs.

MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*

Results: *recall@k*

SQuAD

Model	@5
ConvRR	75.6
DrQA document-retrieval	77.8
MRNN	**80.4**

QUASAR-T

Model	@1	@3	@5
BM25	19.7	36.3	44.3
SR^2: Simple Ranker-Reader	28.8	46.4	54.9
InferSent Ranker	36.1	52.8	56.7
SR^3: Reinforced Ranker-Reader	40.3	51.3	54.5
ConvRR	50.67	63.1	67.4
Relation-Networks Ranker	51.4	67.7	69.9
MRNN	**52.8**	**68.2**	**70.3**

MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*

Results: *Mean Average Precision (MAP), Mean Reciprocal Rank (MRR)*

WikiQA

Model	MAP	MRR
WordCount	0.4891	0.4924
WgtWordCnt	0.5099	0.5132
PV	0.511	0.516
PV + Cnt	0.599	0.609
CNN + Cnt	0.652	0.6652
QA-LSTM	0.654	0.665
AP-LSTM	0.670	0.684
AP-CNN	0.689	0.696
Self-LSTM	0.693	0.704
ABCNN	0.692	0.71
Rank MP-CNN	0.701	0.718
RNN-POA	0.721	0.731
Multihop-Sequential-LSTM	0.722	0.738
MRNN	**0.731**	**0.745**

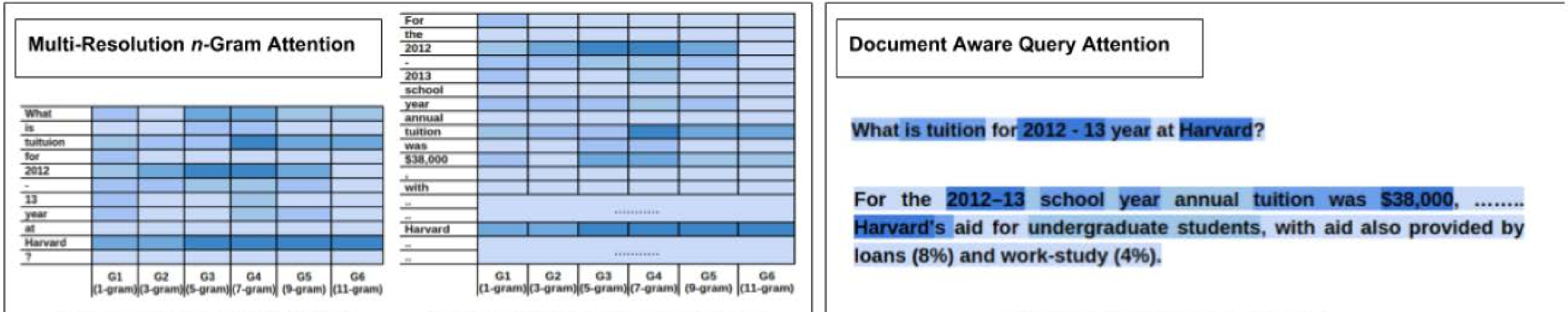
TrecQA

Model	MAP	MRR
Tree Edit Model	0.609	0.692
LSTM	0.713	0.791
CNN	0.746	0.808
AP-LSTM	0.753	0.830
AP-CNN	0.753	0.851
Self-LSTM	0.759	0.830
RNN-POA	0.781	0.851
Multihop-Sequential-LSTM	0.813	0.893
MRNN	**0.822**	**0.898**

MRNN: A Multi-Resolution Neural Network to Improve Ad-Hoc Retrieval *(cont.)*

Visualization:

The attention visualizations of the duplex attention component for the sample query and corresponding document extracted from the SQuAD dataset



a) *n*-Gram attentions for the query

b) *n*-Gram attentions for the document

c) Document aware query attentions

Conclusion

- The proposed multi-resolution word embedding using large datasets, including SQuAD and QUASAR benchmark datasets, shows a significant performance gain in terms of the recall.

- The multi-resolution neural network, which is the first model that leverages the strength of representations of different abstract levels in the learned hierarchical representation incorporated with a new duplex attention mechanism, can significantly improve the performance of ad-hoc retrieval.

Future Works

- Develop new loss functions to replace it with the current distance metric losses.
- Implement multilevel abstract representations to other fields, such as pattern recognition and computer vision.
- Apply our multi-resolution models to other AI tasks.

Acknowledgement

- Joint work with Dr. Tolgahan Cakaloglu
- Funding from Google

Questions ?

Thank you. ...

References

- Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems 26, pp. 3111?3119. 2013.
- Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In Empirical Methods in Natural Language Processing (EMNLP), pp. 1532?1543, 2014.
- Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. Advances in pre-training distributed word representations. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.
- Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. Deep contextualized word representations. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 2227?2237, 2018.
- Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805, 2018.
- Cakaloglu, T., and Xu, X. A Multi-Resolution Word Embedding for Document Retrieval from Large Unstructured Knowledge Bases. arXiv preprint arXiv:1902.00663