Part I: Overview of Text Embedding Methods

KDD 2020 Tutorial
Embedding-Driven Multi-Dimensional Topic Mining and Text Analysis
Yu Meng, Jiaxin Huang, Jiawei Han
Computer Science, University of Illinois at Urbana-Champaign
August 23, 2020
Outline

- Introduction to text embeddings
- Local context-based word embeddings
- Joint local and global context-based text embeddings
- Deep contextualized embeddings via neural language models
- Extend unsupervised embeddings to incorporate weak supervision
Introduction to Text Embeddings

- A milestone in NLP and ML:
  - Unsupervised learning of text representations—No supervision needed
  - Embed one-hot vectors into lower-dimensional space—Address “curse of dimensionality”
  - Word embedding captures useful properties of word semantics
    - Word similarity: Words with similar meanings are embedded closer
    - Word analogy: Linear relationships between words (e.g. king - queen = man - woman)
Introduction to Text Embeddings

- Text embeddings can be used in a lot of downstream applications
  - Word/token/entity-level tasks
    - Keyword extraction/clustering
    - Taxonomy construction
  - Document/paragraph-level tasks
    - Document classification/clustering/retrieval
    - Question answering/text summarization

Taxonomy Construction

Document Classification
Introduction to text embeddings

Local context-based word embeddings
- Euclidean space: Word2Vec, GloVe, fastText
- Hyperbolic space: Poincaré embeddings

Joint local and global context-based text embeddings

Deep contextualized embeddings via neural language models

Extend unsupervised embeddings to incorporate weak supervision
Word2Vec

- Local context-based word embedding learning pushes together terms that share same or similar local contexts.
- For example, Word2Vec maximizes the probability of observing a word based on its contexts.
- As a result, semantically coherent terms are more likely to have close embeddings.

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

$$p(w_0 | w_I) = \frac{\exp \left( v'_{w_0}^T v_{w_I} \right)}{\sum_{w=1}^{W} \exp \left( v'_{w}^T v_{w_I} \right)}$$

Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS.
GloVe factorizes a global co-occurrence matrix derived from the entire corpus.

Low-dimensional representations are obtained by solving a least-squares problem to “recover” the co-occurrence matrix.

\[
J = \sum_{i,j=1}^{V} f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2
\]

fastText

- fastText improves upon Word2Vec by incorporating subword information into word embedding
- fastText allows sharing subword representations across words, since words are represented by the aggregation of their n-grams

Word2Vec probability expression

\[ p(w_O|w_I) = \frac{\exp(v'_w w_Q \top v_w I)}{\sum_{w=1}^W \exp(v'_w w_I \top v_w I)} \]

Represent a word by the sum of the vector representations of its n-grams

Introduction to text embeddings

Local context-based word embeddings

- Euclidean space: Word2Vec, GloVe, fastText
- Hyperbolic space: Poincaré embeddings

Joint local and global context-based text embeddings

Deep contextualized embeddings via neural language models

Extend unsupervised embeddings to incorporate weak supervision
Hyperbolic Embedding: Poincaré embedding

- Why non-Euclidean embedding space?
  - Data can have specific structures that Euclidean-space models struggle to capture

- The hyperbolic space
  - Continuous version of trees
  - Naturally equipped to model hierarchical structures

- Poincaré embedding
  - Learn hierarchical representations by pushing general terms to the origin of the Poincaré ball, and specific terms to the boundary

\[
d(u, v) = \text{arcosh} \left( 1 + \frac{\|u - v\|^2}{(1 - \|u\|^2)(1 - \|v\|^2)} \right)
\]

Texts in Hyperbolic Space: Poincaré GloVe

- GloVe in hyperbolic space
- Motivation: latent hierarchical structure of words exists among text
  - Hypernym-hyponym
  - Textual entailment
- Approach: use hyperbolic kernels!
- Effectively model generality/specificity

\[ J = \sum_{i,j=1}^{V} f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2 \]

Hyperbolic metric

Outline

- Introduction to text embeddings
- Local context-based word embeddings
- Joint local and global context-based text embeddings
  - Spherical space: JoSE
- Deep contextualized embeddings via neural language models
- Extend unsupervised embeddings to incorporate weak supervision
Directional Analysis for Text Embeddings

- How to use text embeddings? Mostly directional similarity (i.e., cosine similarity)
  - Word similarity is derived using cosine similarity
  
  ![Diagram showing directional similarity between words](image)

  - Vector direction is what actually matters!

- Better clustering performances when embeddings are normalized and spherical clustering algorithms are used (Spherical K-means)
Issues with previous word embedding frameworks:

- Although directional similarity has shown effective for various applications, previous embeddings (e.g., Word2Vec, GloVe, fastText) are trained in the Euclidean space
- A gap between training space and usage space: Trained in Euclidean space but used on sphere

Embedding Training in Euclidean Space

Post-processing (Normalization)

Embedding Usage on the Sphere (Similarity, Clustering, etc.)
Motivation

- What is the consequence of the inconsistency between word embedding training and usage space?
- The objective we optimize during training is not really the one we use.
- Regardless of the different training objective, Word2Vec, GloVe and fastText all optimize the embedding dot product during training, but cosine similarity is what actually used in applications.

Embedding dot product is optimized during training.

\[
p(w_O | w_I) = \frac{\exp(v_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} \exp(v_{w}^T v_{w_I})}
\]

\[
J = \sum_{i,j=1}^{V} f(X_{ij}) (\tilde{w}_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2
\]

\[
s(w, c) = \sum_{g \in G_w} z_g^T v_c
\]
What is the consequence of the inconsistency between word embedding training and usage space?

- The objective we optimize during training is not really the one we use.
- E.g. Consider two pairs of words, A: lover-quarrel; B: rock-jazz. Pair B has higher ground truth similarity than pair A in WordSim353 (a benchmark testset).
- Word2Vec assigns higher dot product to pair B, but its cosine similarity is still smaller than pair A.

<table>
<thead>
<tr>
<th>Metrics</th>
<th>A: lover-quarrel</th>
<th>B: rock-jazz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dot Product</td>
<td>5.284</td>
<td>6.287</td>
</tr>
<tr>
<td>Cosine Similarity</td>
<td>0.637</td>
<td>0.628</td>
</tr>
</tbody>
</table>

Inconsistency
Motivation

- Apart from the training/usage space inconsistency issue, previous embedding frameworks only leverage local contexts to learn word representations.
- Local contexts can only partly define word semantics in unsupervised word embedding learning.

If I hear someone screwing with my car (ie, setting off the alarm) and taunting me to come out, you can be very sure that my Colt Delta Elite will also be coming with me. It is not the screwing with the car that would get them shot, it is the potential physical danger. If they are taunting like that, it’s very possible that they also intend to rob me and or do other physically harmful things. Here in Houston last year a woman heard the sound of someone …
We design a generative model on the sphere that follows how humans write articles:

- We first have a general idea of the paragraph/document, and then start to write down each word in consistent with not only the paragraph/document, but also the surrounding words.
- Assume a two-step generation process:

\[ p(u \mid d) \propto \exp(\cos(u, d)) \quad p(v \mid u) \propto \exp(\cos(v, u)) \]

How to define the generative model in the spherical space?

We prove a theorem connecting the above generative model with a spherical probability distribution:

**Theorem 1.** When the corpus has infinite vocabulary, i.e., $|V| \to \infty$, the analytic forms of $p(u \mid d) \propto \exp(\cos(u, d))$ and $p(v \mid u) \propto \exp(\cos(v, u))$ are given by the von Mises-Fisher (vMF) distribution with the prior embedding as the mean direction and constant 1 as the concentration parameter, i.e.,

$$
\lim_{|V| \to \infty} p(v \mid u) = \text{vMF}_p(v; u, 1), \quad \lim_{|V| \to \infty} p(u \mid d) = \text{vMF}_p(u; d, 1).
$$
Understanding the spherical generative model
Training objective:

The final generation probability:

\[ p(v, u \mid d) = p(v \mid u) \cdot p(u \mid d) = \text{vMF}_p(v; u, 1) \cdot \text{vMF}_p(u; d, 1) \]

Maximize the log-probability of a real co-occurred tuple \((v, u, d)\), while minimize that of a negative sample \((v, u', d)\), with a max-margin loss:

\[
\mathcal{L}_{\text{joint}}(u, v, d) = \max \left( 0, m - \log (c_p(1) \exp(\cos(v, u)) \cdot c_p(1) \exp(\cos(u, d))) \right) \\
+ \log (c_p(1) \exp(\cos(v, u')) \cdot c_p(1) \exp(\cos(u', d))) \\
= \max \left( 0, m - \cos(v, u) - \cos(u, d) + \cos(v, u') + \cos(u', d) \right),
\]
Riemannian optimization with Riemannian SGD:

Riemannian gradient:

$$\text{grad } f(x) := (I - xx^T) \nabla f(x)$$

Exponential mapping (maps from the tangent plane to the sphere):

$$\exp_x(z) := \begin{cases} 
\cos(\|z\|)x + \sin(\|z\|) \frac{z}{\|z\|}, & z \in T_xS^{p-1}\setminus\{0\}, \\
x, & z = 0.
\end{cases}$$

Riemannian SGD:

$$x_{t+1} = \exp_{x_t}(-\eta_t \text{grad } f(x_t))$$

Retraction (first-order approximation of the exponential mapping):

$$R_x(z) := \frac{x + z}{\|x + z\|}$$
Training details:

- Incorporate angular distances into Riemannian optimization

$$z = \text{grad} f(x_t)$$
$$d_{\text{cos}} = 1 - \cos (x_t, -\nabla f(x_t)) = 1 + \frac{x_t^T \nabla f(x_t)}{\|\nabla f(x_t)\|}$$

- Multiply the Euclidean gradient with its angular distance from the current point

$$x_{t+1} = R_{x_t} \left( -\eta_t \left( 1 + \frac{x_t^T \nabla f(x_t)}{\|\nabla f(x_t)\|} \right) \left( I - x_t x_t^T \right) \nabla f(x_t) \right).$$
Word similarity results:

<table>
<thead>
<tr>
<th>Embedding Space</th>
<th>Model</th>
<th>WordSim353</th>
<th>MEN</th>
<th>SimLex999</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>Word2Vec</td>
<td>0.711</td>
<td>0.726</td>
<td>0.311</td>
</tr>
<tr>
<td></td>
<td>GloVe</td>
<td>0.598</td>
<td>0.690</td>
<td>0.321</td>
</tr>
<tr>
<td></td>
<td>fastText</td>
<td>0.697</td>
<td>0.722</td>
<td>0.303</td>
</tr>
<tr>
<td></td>
<td>BERT</td>
<td>0.477</td>
<td>0.594</td>
<td>0.287</td>
</tr>
<tr>
<td>Poincaré</td>
<td>Poincaré GloVe</td>
<td>0.623</td>
<td>0.652</td>
<td>0.321</td>
</tr>
<tr>
<td>Spherical</td>
<td>JoSE</td>
<td><strong>0.739</strong></td>
<td><strong>0.748</strong></td>
<td><strong>0.339</strong></td>
</tr>
</tbody>
</table>

Why does BERT fall behind on this task?

- BERT learns contextualized representations, but word similarity is conducted in a context-free manner.
- BERT is optimized on specific pre-training tasks like predicting masked words and sentence relationships, which have no direct relation to word similarity.
Document clustering results:

Embedding quality is generally more important than clustering algorithms:

- Using spherical K-Means only gives marginal performance boost over K-Means
- JoSE embedding remains optimal regardless of clustering algorithms

### Table 2: Document clustering evaluation on the 20 Newsgroup dataset.

<table>
<thead>
<tr>
<th>Embedding</th>
<th>Clus. Alg.</th>
<th>MI</th>
<th>NMI</th>
<th>ARI</th>
<th>Purity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. W2V</td>
<td>K-Means</td>
<td>1.328 ± 0.024</td>
<td>0.453 ± 0.009</td>
<td>0.250 ± 0.008</td>
<td>0.419 ± 0.012</td>
</tr>
<tr>
<td></td>
<td>SK-Means</td>
<td>1.299 ± 0.031</td>
<td>0.445 ± 0.009</td>
<td>0.247 ± 0.008</td>
<td>0.408 ± 0.014</td>
</tr>
<tr>
<td>SIF</td>
<td>K-Means</td>
<td>0.958 ± 0.012</td>
<td>0.322 ± 0.004</td>
<td>0.164 ± 0.004</td>
<td>0.331 ± 0.005</td>
</tr>
<tr>
<td></td>
<td>SK-Means</td>
<td>0.893 ± 0.028</td>
<td>0.308 ± 0.009</td>
<td>0.137 ± 0.006</td>
<td>0.285 ± 0.011</td>
</tr>
<tr>
<td>BERT</td>
<td>K-Means</td>
<td>0.854 ± 0.022</td>
<td>0.289 ± 0.008</td>
<td>0.127 ± 0.003</td>
<td>0.281 ± 0.010</td>
</tr>
<tr>
<td></td>
<td>SK-Means</td>
<td>0.719 ± 0.013</td>
<td>0.248 ± 0.004</td>
<td>0.100 ± 0.003</td>
<td>0.233 ± 0.005</td>
</tr>
<tr>
<td>Doc2Vec</td>
<td>K-Means</td>
<td>1.876 ± 0.020</td>
<td>0.630 ± 0.007</td>
<td>0.494 ± 0.012</td>
<td>0.648 ± 0.017</td>
</tr>
<tr>
<td></td>
<td>SK-Means</td>
<td>1.856 ± 0.020</td>
<td>0.626 ± 0.006</td>
<td>0.469 ± 0.015</td>
<td>0.640 ± 0.016</td>
</tr>
<tr>
<td>JoSE</td>
<td>K-Means</td>
<td>1.975 ± 0.026</td>
<td>0.663 ± 0.008</td>
<td>0.556 ± 0.018</td>
<td>0.711 ± 0.020</td>
</tr>
<tr>
<td></td>
<td>SK-Means</td>
<td>1.982 ± 0.034</td>
<td>0.664 ± 0.010</td>
<td>0.568 ± 0.020</td>
<td>0.721 ± 0.029</td>
</tr>
</tbody>
</table>
Experiments

- Training efficiency:

<table>
<thead>
<tr>
<th>Model</th>
<th>Training time (per iteration) on the latest Wikipedia dump.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec</td>
<td>0.81 hrs</td>
</tr>
<tr>
<td>GloVe</td>
<td>0.85 hrs</td>
</tr>
<tr>
<td>fastText</td>
<td>2.11 hrs</td>
</tr>
<tr>
<td>BERT</td>
<td>&gt; 5 days</td>
</tr>
<tr>
<td>Poincaré GloVe</td>
<td>1.25 hrs</td>
</tr>
<tr>
<td>JoSE</td>
<td>0.73 hrs</td>
</tr>
</tbody>
</table>

- Why is JoSE training efficient?

  - Other models’ objectives contain many non-linear operations (Word2Vec & fastText's objectives involve exponential functions; GloVe's objective involves logarithm functions), while JoSE only has linear terms in the objective.
Outline

- Introduction to text embeddings
- Local context-based word embeddings
- Joint local and global context-based text embeddings
- Deep contextualized embeddings via neural language models
- Extend unsupervised embeddings to incorporate weak supervision
From Context-Free Embedding to Contextualized Embedding

- Previous unsupervised word embeddings like Word2Vec and GloVe learn context-free word embedding
  - Each word has one representation regardless of specific contexts it appears in
  - E.g. “bank” is a polysemy, but only has one representation

- Deep neural language models overcome this problem by learning contextualized word semantics
Word representations are learned functions of the internal states of a deep bi-directional LSTMs

Results in a pre-trained network that benefits several downstream tasks (e.g. Sentiment analysis, Named entity extraction, Question answering)

However, left-to-right and right-to-left LSTMs are independently trained and concatenated

- **Bidirectional**: BERT leverages masked language modeling learning to introduce real bidirectionality training.
- **Masked LM**: With 15% words randomly masked, the model learns bidirectional contextual information to predict the masked words.

BERT: Deep Bidirectional Transformers

- Transformer Encoder: Reads the entire sequence of words at once; learns the context of a word based on every token in the sequence.
- The Transformer employs a self-attention mechanism that learns contextual relations between words (and sub-words) in a text sequence.
Next Sentence Prediction: learn to predict if the second sentence in the pair is the subsequent sentence in the original document
Several simple modifications that make BERT more effective:

- train the model longer, with bigger batches over more data
- remove the next sentence prediction objective
- train on longer sequences
- dynamically change the masking pattern applied to the training data

<table>
<thead>
<tr>
<th>Model</th>
<th>data</th>
<th>bsz</th>
<th>steps</th>
<th>SQuAD (v1.1/2.0)</th>
<th>MNLI-m</th>
<th>SST-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RoBERTa</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with BOOKS + WIKI</td>
<td>16GB</td>
<td>8K</td>
<td>100K</td>
<td>93.6/87.3</td>
<td>89.0</td>
<td>95.3</td>
</tr>
<tr>
<td>+ additional data (§3.2)</td>
<td>160GB</td>
<td>8K</td>
<td>100K</td>
<td>94.0/87.7</td>
<td>89.3</td>
<td>95.6</td>
</tr>
<tr>
<td>+ pretrain longer</td>
<td>160GB</td>
<td>8K</td>
<td>300K</td>
<td>94.4/88.7</td>
<td>90.0</td>
<td>96.1</td>
</tr>
<tr>
<td>+ pretrain even longer</td>
<td>160GB</td>
<td>8K</td>
<td>500K</td>
<td><strong>94.6/89.4</strong></td>
<td><strong>90.2</strong></td>
<td><strong>96.4</strong></td>
</tr>
<tr>
<td>BERT_LARGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with BOOKS + WIKI</td>
<td>13GB</td>
<td>256</td>
<td>1M</td>
<td>90.9/81.8</td>
<td>86.6</td>
<td>93.7</td>
</tr>
<tr>
<td>XLNet_LARGE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with BOOKS + WIKI</td>
<td>13GB</td>
<td>256</td>
<td>1M</td>
<td>94.0/87.8</td>
<td>88.4</td>
<td>94.4</td>
</tr>
<tr>
<td>+ additional data</td>
<td>126GB</td>
<td>2K</td>
<td>500K</td>
<td>94.5/88.8</td>
<td>89.8</td>
<td>95.6</td>
</tr>
</tbody>
</table>

Simple modifications that make BERT more **efficient**:

- Factorized embedding parameterization: use lower-dimensional token embeddings; project token embeddings to hidden layer dimension
- Cross-layer parameter sharing: share feed-forward network parameters/attention parameters across layers
- Inter-sentence coherence loss: change the next sentence prediction task to sentence order prediction

<table>
<thead>
<tr>
<th>Model</th>
<th>Parameters</th>
<th>SQuAD1.1</th>
<th>SQuAD2.0</th>
<th>MNLI</th>
<th>SST-2</th>
<th>RACE</th>
<th>Avg</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>base</td>
<td>108M</td>
<td>90.4/83.2</td>
<td>80.4/77.6</td>
<td>84.5</td>
<td>92.8</td>
<td>68.2</td>
<td>82.3</td>
</tr>
<tr>
<td></td>
<td>large</td>
<td>334M</td>
<td>92.2/85.5</td>
<td>85.0/82.2</td>
<td>86.6</td>
<td>93.0</td>
<td>73.9</td>
<td>85.2</td>
</tr>
<tr>
<td>ALBERT</td>
<td>base</td>
<td>12M</td>
<td>89.3/82.3</td>
<td>80.0/77.1</td>
<td>81.6</td>
<td>90.3</td>
<td>64.0</td>
<td>80.1</td>
</tr>
<tr>
<td></td>
<td>large</td>
<td>18M</td>
<td>90.6/83.9</td>
<td>82.3/79.4</td>
<td>83.5</td>
<td>91.7</td>
<td>68.5</td>
<td>82.4</td>
</tr>
<tr>
<td></td>
<td>xlarge</td>
<td>60M</td>
<td>92.5/86.1</td>
<td>86.1/83.1</td>
<td>86.4</td>
<td>92.4</td>
<td>74.8</td>
<td>85.5</td>
</tr>
<tr>
<td></td>
<td>xxlarge</td>
<td>235M</td>
<td><strong>94.1/88.3</strong></td>
<td><strong>88.1/85.1</strong></td>
<td><strong>88.0</strong></td>
<td><strong>95.2</strong></td>
<td><strong>82.3</strong></td>
<td><strong>88.7</strong></td>
</tr>
</tbody>
</table>

XLNet: Autoregressive Language Modeling

- Issues with BERT: Masked tokens are predicted independently, and [MASK] token brings discrepancy between pre-training and fine-tuning
- XLNet uses Permutation Language Modeling
  - Permutes the text sequence and predicts the target word using the remaining words in the sequence
  - Since words in the original sequence are permuted, both forward direction information and backward direction information are leveraged

**XLNet: Two-Stream Self-Attention**

- Content representation: Encodes both token position as well as content
- Query representation: Encodes only token position
- Change masked language modeling to a more sample-efficient pre-training task, replaced token detection

- Why more efficient:
  - Replaced token detection trains on all tokens, instead of just on those that are masked (15%)
  - The generator trained with MLM is small (parameter size is ~1/10 of discriminator)
  - The discriminator is trained with a binary classification task, instead of MLM (classification over the entire vocabulary)

Clark, K., Luong, M. T., Le, Q. V., & Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. ICLR.
ELECTRA

- State-of-the-art GLUE (General Language Understanding Evaluation) test performance with the same compute (measured by Floating Point Operations)

<table>
<thead>
<tr>
<th>Model</th>
<th>Train FLOPs</th>
<th>CoLA</th>
<th>SST</th>
<th>MRPC</th>
<th>STS</th>
<th>QQP</th>
<th>MNLI</th>
<th>QNLI</th>
<th>RTE</th>
<th>WNLI</th>
<th>Avg.*</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>1.9e20 (0.06x)</td>
<td>60.5</td>
<td>94.9</td>
<td>85.4</td>
<td>86.5</td>
<td>89.3</td>
<td>86.7</td>
<td>92.7</td>
<td>70.1</td>
<td>65.1</td>
<td>79.8</td>
<td>80.5</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>3.2e21 (1.02x)</td>
<td>67.8</td>
<td>96.7</td>
<td>89.8</td>
<td>91.9</td>
<td>90.2</td>
<td>90.8</td>
<td>95.4</td>
<td>88.2</td>
<td>89.0</td>
<td>88.1</td>
<td>88.1</td>
</tr>
<tr>
<td>ALBERT</td>
<td>3.1e22 (10x)</td>
<td>69.1</td>
<td>97.1</td>
<td>91.2</td>
<td>92.0</td>
<td>90.5</td>
<td>91.3</td>
<td>–</td>
<td>89.2</td>
<td>91.8</td>
<td>89.0</td>
<td>–</td>
</tr>
<tr>
<td>XLNet</td>
<td>3.9e21 (1.26x)</td>
<td>70.2</td>
<td>97.1</td>
<td>90.5</td>
<td>92.6</td>
<td>90.4</td>
<td>90.9</td>
<td>–</td>
<td>88.5</td>
<td>92.5</td>
<td>89.1</td>
<td>–</td>
</tr>
<tr>
<td>ELECTRA</td>
<td>3.1e21 (1x)</td>
<td>71.7</td>
<td>97.1</td>
<td>90.7</td>
<td>92.5</td>
<td>90.8</td>
<td>91.3</td>
<td>95.8</td>
<td>89.8</td>
<td>92.5</td>
<td>89.5</td>
<td>89.4</td>
</tr>
</tbody>
</table>
Outline

- Introduction to text embeddings
- Local context-based word embeddings
- Joint local and global context-based text embeddings
- Deep contextualized embeddings via neural language models
- Extend unsupervised embeddings to incorporate weak supervision
Unsupervised word embedding can be used as word representations/features in a wide spectrum of text mining tasks

However, unsupervised word embeddings are generic word representations

- Not yielding the best performance on downstream tasks (e.g., taxonomy construction, document classification)
- Reason: Not incorporating task-specific information

We will introduce a weakly-supervised text embedding method in Part 3

Unsupervised word embedding (Word2Vec) fails to discriminate opposite meaning words


Clark, K., Luong, M. T., Le, Q. V., & Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. ICLR.


Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. NIPS.


