Embedding Methods for NLP
Part 1: Unsupervised and Supervised Embeddings

Jason Weston & Antoine Bordes

Facebook, NY, USA
What is a word embedding?

Suppose you have a dictionary of words.

The $i^{th}$ word in the dictionary is represented by an embedding:

$$w_i \in \mathbb{R}^d$$

i.e. a $d$-dimensional vector, which is learnt!

- $d$ typically in the range 50 to 1000.
- Similar words should have similar embeddings (share latent features).
- Embeddings can also be applied to symbols as well as words (e.g. Freebase nodes and edges).
- Discuss later: can also have embeddings of phrases, sentences, documents, or even other modalities such as images.
Learning an Embedding Space

Example of Embedding of 115 Countries (Bordes et al., ’11)
Main methods we highlight, ordered by date.

- Latent Semantic Indexing (Deerwester et al., ’88).
- Neural Net Language Models (NN-LMs) (Bengio et al., ’06).
- Convolutional Nets for tagging (SENNNA) (Collobert & Weston, ’08).
- Supervised Semantic Indexing (Bai et al., ’09).
- Wsabie (Weston et al., ’10).
- Recurrent NN-LMs (Mikolov et al., ’10).
- Recursive NNs (Socher et al., ’11).
- Word2Vec (Mikolov et al., ’13).
- Paragraph Vector (Le & Mikolov, ’14).
- Overview of recent applications.
Main methods we highlight, ordered by topic.

Embeddings for Ranking and Retrieval:
- Latent Semantic Indexing (Deerwester et al., '88).
- Supervised Semantic Indexing (Bai et al, '09).
- Wsabie (Weston et al., '10).

Embeddings for Language Modeling (useful for speech, translation, ...):
- Neural Net Language Models (NN-LMs) (Bengio et al., '06)
- Recurrent NN-LMs (Mikolov et al., '10).
- Word2Vec (Mikolov et al., '13).

Embeddings for Supervised Prediction Tasks (POS, chunk, NER, SRL, sentiment, etc.):
- Convolutional Nets for tagging (SENNNA) (Collobert & Weston, '08).
- Recursive NNs (Socher et al., '11).
- Paragraph Vector (Le & Mikolov, '14).
Main methods we highlight, ordered by topic.

**Embeddings for Ranking and Retrieval:**
- Latent Semantic Indexing (Deerwester et al., ’88).
- Supervised Semantic Indexing (Bai et al, ’09).
- Wsabie (Weston et al., ’10).

**Embeddings for Language Modeling (useful for speech, translation, ...):**
- Neural Net Language Models (NN-LMs) (Bengio et al., ’06)
- Recurrent NN-LMs (Mikolov et al., ’10).
- Word2Vec (Mikolov et al., ’13).

**Embeddings for Supervised Prediction Tasks (POS, chunk, NER, SRL, sentiment, etc.):**
- Convolutional Nets for tagging (SENNA) (Collobert & Weston, ’08).
- Recursive NNs (Socher et al., ’11).
- Paragraph Vector (Le & Mikolov, ’14).
Ranking and Retrieval: The Goal

We want to learn to match a query (text) to a target (text).

Many classical supervised ranking methods use hand-coded features.

Methods like LSI that learn from words are unsupervised.

Supervised Semantic Indexing (SSI) uses supervised learning from text only:


Outperforms existing methods (on words) like TFIDF, LSI or a (supervised) margin ranking perceptron baseline.
Basic Bag-O’-Words

Bag-of-words + cosine similarity:

- Each doc. \( \{d_t\}_{t=1}^{N} \subset \mathbb{R}^D \) is a normalized bag-of-words.
- Similarity with query \( q \) is: \( f(q, d) = q^\top d \)

- Doesn’t deal with synonyms: bag vectors can be orthogonal

- No machine learning at all
Latent semantic indexing (LSI)

Learn a linear embedding $\phi(d_i) = Ud_i$ via a reconstruction objective.

- Rank with: $f(q, d) = q^\top U^\top Ud = \phi(q)^\top \phi(d_i)$ \footnote{\(f(q, d) = q^\top (U^\top U + \alpha I)\) gives better results. Also, usually normalize this $\rightarrow$ cosine similarity.}.

Uses "synonyms": low-dimensional latent "concepts".

Unsupervised machine learning: useful for goal?
Supervised Semantic Indexing (SSI)

- Basic model: rank with
  \[ f(q, d) = q^\top Wd = \sum_{i,j=1}^{D} q_i W_{ij} d_j \]
i.e. learn weights of polynomial terms between documents.
- Learn \( W \in \mathbb{R}^{D \times D} \) (huge!) with click-through data or other labels.

Uses “synonyms”

Supervised machine learning: targeted for goal

Too Big/Slow?! Solution = Constrain \( W \):
  low rank \( \rightarrow \) embedding model!
SSI: why is this a good model?

Classical bag-of-words doesn't work when there are few matching terms:

$q = (\text{kitten, vet, nyc})$

$d = (\text{cat, veterinarian, new, york})$

Method $q^T W d$ learns that e.g. kitten and cat are highly related.

E.g. if $i$ is the index of kitten and $j$ is the index of cat, then $W_{ij} > 0$ after training.
SSI: Why the Basic Model Sucks

$W$ is big: 3.4Gb if $D = 30000$, 14.5Tb if $D = 2.5M$.

Slow: $q^T W d$ computation has $mn$ computations $q_j W_{ij} d_i$, where $q$ and $d$ have $m$ and $n$ nonzero terms.

Or one computes $v = q^T W$ once, and then $vd$ for each document. Classical speed where query has $D$ terms, assuming $W$ is dense → still slow.

One could minimize $||W||_1$ and attempt to make $W$ sparse. Then at most $mp$ times slower than classical model (with $p$ nonzeros in a column.)
SSI Improved model: Low Rank $W$

**Constrain $W$:**

$$W = U^\top V + I.$$  

$U$ and $V$ are $N \times D$ matrices $\rightarrow$ smaller

Low dimensional “latent concept” space like LSI (same speed).

Differences: supervised, asymmetric, learns with $I$.

**Variants:**

- $W = I$: bag-of-words again.
- $W = D$, reweighted bag-of-words related to [Grangier and Bengio, 2005].
- $W = U^\top U + I$: symmetric.
SSI: Training via maximizing AUC

- Given a set of tuples $\mathcal{R}$ with a query $q$, a related document $d^+$ and an unrelated (or lower ranked) document $d^-$.
- We would like $f(q, d^+) > f(q, d^-)$.
- Minimize margin ranking loss [Herbrich et al., 2000]:

$$\sum_{(q, d^+, d^-) \in \mathcal{R}} \max(0, 1 - f(q, d^+) + f(q, d^-)).$$

**Learning Algorithm Stochastic Gradient Descent: Fast & scalable.**

| Iterate | Sample a triplet $(q, d^+, d^-)$, Update $W \leftarrow W - \lambda \frac{\partial}{\partial W} \max(0, 1 - f(q, d^+) + f(q, d^-))$. |

Other options: batch gradient, parallel SGD (hogwild), Adagrad \ldots
Prior Work: Summary of learning to Rank

- [Grangier & Bengio, ’06] used similar methods to basic SSI for retrieving images.
- [Goel, Langord & Strehl, ’08] used Hash Kernels (Vowpal Wabbit) for advert placement.
- Main difference: we use low rank+CFH on W.
- SVM [Joachims, 2002] and NN ranking methods [Burges, 2005]. Use hand-coded features: title, body, URL, search rankings,… (don’t use words)
  (e.g. Burges uses 569 features in all).
- In contrast we use only the words and train on huge feature sets.
- Several works on optimizing different loss functions (MAP, ROC, NDCG): [Cao, 2008], [Yu, 2007], [Qin, 2006],…
- Lots of stuff for “metric learning” problem as well..

One could also add features + new loss to this method ..
Experimental Comparison

- **Wikipedia**
  - 1,828,645 documents. 24,667,286 links.
  - Split into 70% train, 30% test.
- Pick random doc. as query, then rank other docs.
- Docs that are linked to it should be highly ranked.
- **Two setups:**
  - (i) whole document is used as query;
  - (ii) 5,10 or 20 words are picked to mimic keyword search.
Wikipedia Experiments: Document Retrieval Performance

Experiments on Wikipedia, which still contains 2 million documents: retrieval task using the link structure and separated the data into 70% for training and 30% for test.

**Document based retrieval:**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Rank-Loss</th>
<th>MAP</th>
<th>P10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFIDF</td>
<td>0.842%</td>
<td>0.432±0.012</td>
<td>0.193</td>
</tr>
<tr>
<td>αLSI + (1 − α)TFIDF</td>
<td>0.721%</td>
<td>0.433</td>
<td>0.193</td>
</tr>
<tr>
<td>Linear SVM Ranker</td>
<td>0.410%</td>
<td>0.477</td>
<td>0.212</td>
</tr>
<tr>
<td>Hash Kernels + αI</td>
<td>0.322%</td>
<td>0.492</td>
<td>0.215</td>
</tr>
<tr>
<td>Wsabie + αI</td>
<td>0.158%</td>
<td>0.547±0.012</td>
<td>0.239±0.008</td>
</tr>
</tbody>
</table>

**k-keywords based retrieval:**

<table>
<thead>
<tr>
<th>k = 5: Algorithm</th>
<th>Params</th>
<th>Rank</th>
<th>MAP</th>
<th>P@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFIDF</td>
<td>0</td>
<td>21.6%</td>
<td>0.047</td>
<td>0.023</td>
</tr>
<tr>
<td>αLSI + (1 − α)TFIDF</td>
<td>200D+1</td>
<td>14.2%</td>
<td>0.049</td>
<td>0.023</td>
</tr>
<tr>
<td>Wsabie + αI</td>
<td>400D</td>
<td>4.37%</td>
<td>0.166</td>
<td>0.083</td>
</tr>
</tbody>
</table>
Scatter Plots: SSI vs. TFIDF and LSI

Figure: Scatter plots of Average Precision for 500 documents: (a) SSI vs. TFIDF, (b) SSI vs. $\alpha$LSI + $(1 - \alpha)$ TFIDF.
Experiments: Cross-Language Retrieval
Retrieval experiments using a query document in japanese, where the task is to retrieve documents in English (using link structure as ground truth).

SSI can do this without doing a translation step first as it learns to map the two languages together in the embedding space.

$D = 30,000$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Rank-Loss</th>
<th>MAP</th>
<th>P10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TFIDF$_{EngEng}$ (Google translated queries)</td>
<td>4.78%</td>
<td>0.319</td>
<td>0.259</td>
</tr>
<tr>
<td>$\alpha$LSI$<em>{EngEng} + (1 - \alpha)$TFIDF$</em>{EngEng}$</td>
<td>3.71%</td>
<td>0.300</td>
<td>0.253</td>
</tr>
<tr>
<td>$\alpha$CL-LSI$<em>{JapEng} + (1 - \alpha)$TFIDF$</em>{EngEng}$</td>
<td>3.31%</td>
<td>0.275</td>
<td>0.212</td>
</tr>
<tr>
<td>SSI$_{EngEng}$</td>
<td>1.72%</td>
<td>0.399</td>
<td>0.325</td>
</tr>
<tr>
<td>SSI$_{JapEng}$</td>
<td>0.96%</td>
<td>0.438</td>
<td>0.351</td>
</tr>
<tr>
<td>$\alpha$SSI$<em>{JapEng} + (1 - \alpha)$TFIDF$</em>{EngEng}$</td>
<td>0.75%</td>
<td>0.493</td>
<td>0.377</td>
</tr>
<tr>
<td>$\alpha$SSI$<em>{JapEng} + (1 - \alpha)$SSI$</em>{EngEng}$</td>
<td><strong>0.63%</strong></td>
<td><strong>0.524</strong></td>
<td><strong>0.386</strong></td>
</tr>
</tbody>
</table>

Some recent related translation-based embeddings: (Hermann & Blunsom, ICLR ’14) and (Mikolov et al., ’13).
**Wsabie** (Weston, Usunier, Bengio, ’10)

- Extension to SSI, also embeds objects other than text, e.g. images.
- WARP loss function that optimizes precision@k.

*Learn $\Phi_1(\cdot)$ and $\Phi_w(\cdot)$ to optimize precision@k.*
Joint Item-Item Embedding Model

L.H.S: Image, query string or user profile (depending on the task)

\[ \Phi_{LHS}(x) = U\Phi_x(x) : \mathbb{R}^{d_x} \to \mathbb{R}^{100}. \]

R.H.S: document, image, video or annotation (depending on the task)

\[ \Phi_{RHS}(y) = V\Phi_y(y) : \mathbb{R}^{d_y} \to \mathbb{R}^{100}. \]

This model again compares the degree of match between the L.H.S and R.H.S in the embedding space:

\[ f_y(x) = \text{sim}(\Phi_{LHS}(x), \Phi_{RHS}(y)) = \Phi_x(x)^\top U^\top V\Phi_y(y) \]

Also constrain the weights (regularize):

\[ \|U_i\|_2 \leq C, \quad i = 1, \ldots, d_x, \quad \|V_i\|_2 \leq C, \quad i = 1, \ldots, d_y. \]
Ranking Annotations: AUC is Suboptimal

Classical approach to learning to rank is maximize AUC by minimizing:

$$\sum \sum \max(0, 1 + f_{\tilde{y}}(x) - f_y(x))$$

**Problem:** All pairwise errors are considered the same, it counts the number of ranking violations.

**Example:**

- Function 1: true annotations ranked 1st and 101st.
- Function 2: true annotations ranked 50th and 52nd.

AUC prefers these equally as both have 100 “violations”.

We want to optimize the top of the ranked list!
Rank Weighted Loss [Usunier et al. ’09]

Replace classical AUC optimization:

\[ \sum_{x} \sum_{y} \sum_{\bar{y} \neq y} \max(0, 1 + f_{\bar{y}}(x) - f_{y}(x)) \]

With weighted version:

\[ \sum_{x} \sum_{y} \sum_{\bar{y} \neq y} L(rank_{y}(x)) \max(0, 1 + f_{\bar{y}}(x) - f_{y}(x)) \]

where \( rank_{y}(f(x)) \) is the rank of the true label:

\[ rank_{y}(f(x)) = \sum_{\bar{y} \neq y} I(f_{\bar{y}}(x) \geq f_{y}(x)) \]

and \( L(\eta) \) converts the rank to a weight, e.g. \( L(\eta) = \sum_{i=1}^{\eta} 1/\eta \).
Weighted Approximate-Rank Pairwise (WARP) Loss

**Problem:** we would like to apply SGD:

\[
\text{Weighting } L(\text{rank}_y(f(x))), \quad \text{rank}_y(f(x)) = \sum_{\bar{y} \neq y} I(f_{\bar{y}}(x) + 1 \geq f_y(x))
\]

\[\ldots\text{too expensive to compute per } (x, y) \text{ sample as } y \in \mathcal{Y} \text{ is large.}\]

**Solution:** approximate by sampling \( f_i(x) \) until we find a violating label \( \bar{y} \)

\[
\text{rank}_y(f(x)) \approx \left\lceil \frac{|\mathcal{Y}| - 1}{N} \right\rceil
\]

where \( N \) is the number of trials in the sampling step.
Online WARP Loss

**Input:** labeled data \((x_i, y_i), y_i \in \{1, \ldots, Y\}\).

**repeat**
- Pick a random labeled example \((x_i, y_i)\)
- Set \(N = 0\).

**repeat**
- Pick a random annotation \(\bar{y} \in \{1, \ldots, Y\} \setminus y_i\).
  - \(N = N + 1\).

**until** \(f_{\bar{y}}(x) > f_{y_i}(x) - 1\) or \(N > Y - 1\)

**if** \(f_{\bar{y}}(x) > f_{y_i}(x) - 1\) **then**
  - Make a gradient step to minimize:
    \[
    L\left(\left\lfloor \frac{Y-1}{N} \right\rfloor\right) |1 - f_{\bar{y}}(x) + f_{y_i}(x)|_+
    \]
**end if**

**until** validation error does not improve.
## Image Annotation Performance

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>16k ImageNet</th>
<th>22k ImageNet</th>
<th>97k Web Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nearest Means</td>
<td>4.4%</td>
<td>2.7%</td>
<td>2.3%</td>
</tr>
<tr>
<td>One-vs-all SVMs 1+:1-</td>
<td>4.1%</td>
<td>3.5%</td>
<td>1.6%</td>
</tr>
<tr>
<td>One-vs-all SVMs</td>
<td>9.4%</td>
<td>8.2%</td>
<td>6.8%</td>
</tr>
<tr>
<td>AUC SVM Ranker</td>
<td>4.7%</td>
<td>5.1%</td>
<td>3.1%</td>
</tr>
<tr>
<td>Wsabie</td>
<td>11.9%</td>
<td>10.5%</td>
<td>8.3%</td>
</tr>
</tbody>
</table>

## Training time: WARP vs. OWPC-SGD & AUC

![Training time graph](graph.png)

The graph above illustrates the training time comparison between WARP, AUC, and OWPC-SGD on ImageNet. The y-axis represents the Test Precision at Top10, while the x-axis represents the training time in hours.
Learned Annotation Embedding (on Web Data)

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Neighboring Annotations</th>
</tr>
</thead>
<tbody>
<tr>
<td>barack obama</td>
<td>barak obama, obama, barack, barrack obama, bow wow beckham, david beckam, alessandro del piero, del piero santa claus, papa noel, pere noel, santa clause, joyeux noel</td>
</tr>
<tr>
<td>david beckham</td>
<td></td>
</tr>
<tr>
<td>santa</td>
<td></td>
</tr>
<tr>
<td>dolphin</td>
<td>delphin, dauphin, whale, delfin, delfini, baleine, blue whale cattle, shire, dairy cows, kuh, horse, cow, shire horse, kone</td>
</tr>
<tr>
<td>cows</td>
<td></td>
</tr>
<tr>
<td>rose</td>
<td>rosen, hibiscus, rose flower, rosa, roze, pink rose, red rose abies alba, abies, araucaria, pine, neem tree, oak tree</td>
</tr>
<tr>
<td>pine tree</td>
<td></td>
</tr>
<tr>
<td>mount fuji</td>
<td>mt fuji, fuji, fujisan, fujiyama, mountain, zugspitze eiffel, tour eiffel, la tour eiffel, big ben, paris, blue mosque</td>
</tr>
<tr>
<td>eiffel tower</td>
<td></td>
</tr>
<tr>
<td>ipod</td>
<td>i pod, ipod nano, apple ipod, ipod apple, new ipod f 18, eurofighter, f14, fighter jet, tomcat, mig 21, f 16</td>
</tr>
<tr>
<td>f18</td>
<td></td>
</tr>
</tbody>
</table>
Summary

Conclusion

Powerful: supervised methods for ranking.

- Outperform classical methods

Efficient low-rank models → learn hidden representations.

Embeddings good for generalization, but can “blur” too much e.g. for exact word matches.

Extensions

- Nonlinear extensions – e.g. convolutional net instead.
Main methods we highlight, ordered by topic.

Embeddings for Ranking and Retrieval:
- Latent Semantic Indexing (Deerwester et al., ’88).
- Supervised Semantic Indexing (Bai et al, ’09).
- Wsabie (Weston et al., ’10).

Embeddings for Language Modeling (useful for speech, translation)
- Neural Net Language Models (NN-LMs) (Bengio et al., ’06)
- Recurrent NN-LMs (Mikolov et al., ’10).
- Word2Vec (Mikolov et al., ’13).

Embeddings for Supervised Prediction Tasks (POS, chunk, NER, SRL, sentiment, etc.):  
- Convolutional Nets for tagging (SENNA) (Collobert & Weston, ’08).
- Recursive NNs (Socher et al., ’11).
- Paragraph Vector (Le & Mikolov, ’14).
Language Modeling

Task: given a sequence of words, predict the next word.

- $n$-gram models are a strong baseline on this task.
- A variety of embedding models have been tried, they can improve results.
- The embeddings learnt from this unsupervised task can also be used to transfer to and improve a supervised task.
Neural Network Language Models

Neural Network Language Models: Hierarchical Soft Max Trick (Morin & Bengio ’05)

Predicting the probability of each next word is slow in NNLMs because the output layer of the network is the size of the dictionary.

Can predict via a tree instead:

1. Cluster the dictionary either according to semantics (similar words in the same cluster) or frequency (common words in the same cluster). *This gives a two-layer tree, but a binary tree is another possibility.*

2. The internal nodes explicitly model the probability of its child nodes.

3. The cost of predicting the probability of the true word is now: traversal to the child, plus normalization via the internal nodes and children in the same node.

This idea is used in Word2Vec and RNN models as well.
Recurrent Neural Network Language Models

**Key idea:** input to predict next word is current word plus context fed-back from previous word (i.e. remembers the past with recurrent connection).

**Figure:** Recurrent neural network based LM

Recurrent neural network based language model. Mikolov et al., Interspeech, ’10.
Results

NNLMS vs. RNNS: Penn Treebank Results (Mikolov)

<table>
<thead>
<tr>
<th>Model</th>
<th>Weight</th>
<th>PPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-gram with Good-Turing smoothing (GT3)</td>
<td>0</td>
<td>165.2</td>
</tr>
<tr>
<td>5-gram with Kneser-Ney smoothing (KN5)</td>
<td>0</td>
<td>141.2</td>
</tr>
<tr>
<td>5-gram with Kneser-Ney smoothing + cache</td>
<td>0.0792</td>
<td>125.7</td>
</tr>
<tr>
<td>Maximum entropy model</td>
<td>0</td>
<td>142.1</td>
</tr>
<tr>
<td>Random clusterings LM</td>
<td>0</td>
<td>170.1</td>
</tr>
<tr>
<td>Random forest LM</td>
<td>0.1057</td>
<td>131.9</td>
</tr>
<tr>
<td>Structured LM</td>
<td>0.0196</td>
<td>146.1</td>
</tr>
<tr>
<td>Within and across sentence boundary LM</td>
<td>0.0838</td>
<td>116.6</td>
</tr>
<tr>
<td>Log-bilinear LM</td>
<td>0</td>
<td>144.5</td>
</tr>
<tr>
<td>Feedforward NNLM</td>
<td>0</td>
<td>140.2</td>
</tr>
<tr>
<td>Syntactical NNLM</td>
<td>0.0828</td>
<td>131.3</td>
</tr>
<tr>
<td>Combination of static RNNLMs</td>
<td>0.3231</td>
<td>102.1</td>
</tr>
<tr>
<td>Combination of adaptive RNNLMs</td>
<td>0.3058</td>
<td>101.0</td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td><strong>1</strong></td>
<td><strong>83.5</strong></td>
</tr>
</tbody>
</table>

Recent uses of NNLMs and RNNs to improve machine translation:
Also Kalchbrenner ’13, Sutskever et al., ’14., Cho et al., ’14.
Word2Vec: very simple LM, works well

Word2Vec: compositionality

Table 7: Comparison and combination of models on the Microsoft Sentence Completion Challenge.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-gram [32]</td>
<td>39</td>
</tr>
<tr>
<td>Average LSA similarity [32]</td>
<td>49</td>
</tr>
<tr>
<td>Log-bilinear model [24]</td>
<td>54.8</td>
</tr>
<tr>
<td>RNNLMs [19]</td>
<td>55.4</td>
</tr>
<tr>
<td>Skip-gram</td>
<td>48.0</td>
</tr>
<tr>
<td>Skip-gram + RNNLMs</td>
<td>58.9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Czech + currency</th>
<th>Vietnam + capital</th>
<th>German + airlines</th>
<th>Russian + river</th>
<th>French + actress</th>
</tr>
</thead>
<tbody>
<tr>
<td>koruna</td>
<td>Hanoi</td>
<td>airline Lufthansa</td>
<td>Moscow</td>
<td>Juliette Binoche</td>
</tr>
<tr>
<td>Check crown</td>
<td>Ho Chi Minh City</td>
<td>carrier Lufthansa</td>
<td>Volga River</td>
<td>Vanessa Paradis</td>
</tr>
<tr>
<td>Polish zolty</td>
<td>Viet Nam</td>
<td>flag carrier Lufthansa</td>
<td>upriver</td>
<td>Charlotte Gainsbourg</td>
</tr>
<tr>
<td>CTK</td>
<td>Vietnamese</td>
<td>Lufthansa</td>
<td>Russia</td>
<td>Cecile De</td>
</tr>
</tbody>
</table>

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

Code: https://code.google.com/p/word2vec/
Main methods we highlight, ordered by topic.

Embeddings for Ranking and Retrieval:
- Latent Semantic Indexing (Deerwester et al., ’88).
- Supervised Semantic Indexing (Bai et al., ’09).
- Wsabie (Weston et al., ’10).

Embeddings for Language Modeling (useful for speech, translation, . . . ):
- Neural Net Language Models (NN-LMs) (Bengio et al., ’06)
- Recurrent NN-LMs (Mikolov et al., ’10).
- Word2Vec (Mikolov et al., ’13).

Embeddings for Supervised Prediction Tasks (POS, chunk, NER, SRL, sentiment, etc.):
- Convolutional Nets for tagging (SENNNA) (Collobert & Weston, ’08).
- Recursive NNs (Socher et al., ’11).
- Paragraph Vector (Le & Mikolov, ’14).
NLP Tasks

- Part-Of-Speech Tagging (POS): syntactic roles (noun, adverb...)
- Chunking: syntactic constituents (noun phrase, verb phrase...)
- Name Entity Recognition (NER): person/company/location...
- Semantic Role Labeling (SRL):
  
  [John]_{ARG0} [ate]_{REL} [the apple]_{ARG1} [in the garden]_{ARGM−LOC}

Complex Systems

- Two extreme choices to get a complex system
  - Large Scale Engineering: design a lot of complex features, use a fast existing linear machine learning algorithm
  - Large Scale Machine Learning: use simple features, design a complex model which will implicitly learn the right features
The Large Scale Feature Engineering Way

- Extract **hand-made features** e.g. from the parse tree
- **Disjoint**: all tasks trained separately, **Cascade** features
- Feed these features to a **shallow** classifier like **SVM**
### ASSERT: many hand built features for SRL (Pradhan et al, ’04)

Problems:
1) Features rely on other solutions *(parsing, named entity, word-sense)*
2) Technology task-transfer is difficult
   - Choose some **good hand-crafted features**

<table>
<thead>
<tr>
<th>Predicate and POS tag of predicate</th>
<th>Voice: active or passive (hand-built rules)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phrase type: adverbial phrase, prepositional phrase</td>
<td>Governing category: Parent node’s phrase type(s)</td>
</tr>
<tr>
<td>Head word and POS tag of the head word</td>
<td>Position: left or right of verb</td>
</tr>
<tr>
<td>Path: traversal from predicate to constituent</td>
<td>Predicted named entity class</td>
</tr>
<tr>
<td>Word-sense disambiguation of the verb</td>
<td>Verb clustering</td>
</tr>
<tr>
<td>Length of the target constituent (number of words)</td>
<td>NEG feature: whether the verb chunk has a &quot;not&quot;</td>
</tr>
<tr>
<td>Partial Path: lowest common ancestor in path</td>
<td>Head word replacement in prepositional phrases</td>
</tr>
<tr>
<td>First and last words and POS in constituents</td>
<td>Ordinal position from predicate + constituent type</td>
</tr>
<tr>
<td>Constituent tree distance</td>
<td>Temporal cue words (hand-built rules)</td>
</tr>
<tr>
<td>Dynamic class context: previous node labels</td>
<td>Constituent relative features: phrase type</td>
</tr>
<tr>
<td>Constituent relative features: head word</td>
<td>Constituent relative features: head word POS</td>
</tr>
<tr>
<td>Constituent relative features: siblings</td>
<td>Number of pirates existing in the world...</td>
</tr>
</tbody>
</table>

- Feed them to a **shallow classifier** like SVM
The Suboptimal (?) Cascade

- POS tagger
- NER tagger
- Parser
- SRL labeler

Hand coded outputs

Hand coded input features

1-800-222-1222

POISON HELP!
NLP: Large Scale Machine Learning

Goals
- Task-specific engineering limits NLP scope
- Can we find unified hidden representations?
- Can we build unified NLP architecture?

Means
- Start from scratch: forget (most of) NLP knowledge
- Compare against classical NLP benchmarks
- Our dogma: avoid task-specific engineering
NLP Benchmarks

- Datasets:
  - POS, CHUNK, SRL: WSJ (≈ up to 1M labeled words)
  - NER: Reuters (≈ 200K labeled words)

<table>
<thead>
<tr>
<th>System</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shen, 2007</td>
<td>97.33%</td>
</tr>
<tr>
<td><strong>Toutanova, 2003</strong></td>
<td><strong>97.24%</strong></td>
</tr>
<tr>
<td>Gimenez, 2004</td>
<td>97.16%</td>
</tr>
</tbody>
</table>

(a) POS: As in (Toutanova, 2003)

<table>
<thead>
<tr>
<th>System</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ando, 2005</strong></td>
<td><strong>89.31%</strong></td>
</tr>
<tr>
<td>Florian, 2003</td>
<td>88.76%</td>
</tr>
<tr>
<td>Kudoh, 2001</td>
<td>88.31%</td>
</tr>
</tbody>
</table>

(c) NER: CoNLL 2003

<table>
<thead>
<tr>
<th>System</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shen, 2005</td>
<td>95.23%</td>
</tr>
<tr>
<td><strong>Sha, 2003</strong></td>
<td><strong>94.29%</strong></td>
</tr>
<tr>
<td>Kudoh, 2001</td>
<td>93.91%</td>
</tr>
</tbody>
</table>

(b) CHUNK: CoNLL 2000

<table>
<thead>
<tr>
<th>System</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Koomen, 2005</td>
<td>77.92%</td>
</tr>
<tr>
<td>Pradhan, 2005</td>
<td>77.30%</td>
</tr>
<tr>
<td>Haghighi, 2005</td>
<td>77.04%</td>
</tr>
</tbody>
</table>

(d) SRL: CoNLL 2005

- We chose as benchmark systems:
  - Well-established systems
  - Systems avoiding external labeled data

- Notes:
  - **Ando, 2005** uses external unlabeled data
  - **Koomen, 2005** uses 4 parse trees not provided by the challenge
The “Deep Learning” Way
Deep approach attempts to propose a radically different end-to-end approach:

- Avoid building a parse tree. Humans don’t need this to talk.
- Try to avoid all hand-built features → monolithic systems.
- Humans implicitly learn these features. Neural networks can too…?
Neural Networks

- Stack several layers together

![](image)

- Increasing level of abstraction at each layer
- Requires simpler features than "shallow" classifiers
- The "weights" $W_i$ are trained by gradient descent
- How can we feed words?
The Big Picture

A unified architecture for all NLP (labeling) tasks:

| Sentence: | Felix sat on the mat . |
| POS:       | NNP VBD IN DT NN .    |
| CHUNK:     | NP VP PP NP NP-I .    |
| NER:       | PER - - - - - .       |
| SRL:       | ARG1 REL ARG2 ARG2-I ARG2-I - |
**Words into Vectors**

**Idea**
- Words are embed in a vector space

![Diagram showing word embeddings]

- Embeddings are trained

**Implementation**
- A word $w$ is an index in a dictionary $D \in \mathbb{N}$
- Use a lookup-table ($W \sim \text{feature size} \times \text{dictionary size}$)

\[ LT_W(w) = W_w \]

**Remarks**
- Applicable to any discrete feature (words, caps, stems...)
- See (Bengio et al, 2001)
The Lookup Tables

Each word/element in dictionary maps to a vector in \( \mathbb{R}^d \).

- **We learn these vectors.**
- **LookupTable**: input of \( i^{th} \) word is

\[
x = (0, 0, \ldots, 1, 0, \ldots, 0) \quad 1 \text{ at position } i
\]

In the original space words are orthogonal.

\[
cat = (0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0, \ldots)
kitten = (0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0, \ldots)
\]

To get the \( \mathbb{R}^d \) embedding vector for the word we multiply \( Wx \) where \( W \) is a \( d \times N \) vector with \( N \) words in the dictionary.
**Window Approach**

- Tags **one word** at the time
- Feed a **fixed-size window** of text around each word to tag
- **Works** fine for most tasks
- **How do deal with long-range dependencies?**

*E.g. in SRL, the verb of interest might be outside the window!***
**Sentence Approach**

1. Feed the *whole sentence* to the network
2. Tag *one word* at the time: add extra *position* features
3. **Convolutions** to handle variable-length inputs

See (Bottou, 1989) or (LeCun, 1989).

- Produces *local* features with higher level of abstraction
- **Max over time** to capture most relevant features
  - Outputs a *fixed-sized* feature vector
Sentence Approach

Input Sentence
- Text: The cat sat on the mat
- Feature 1: \( w_1^1 \), \( w_1^2 \), ..., \( w_K^1 \)
- Feature K: \( w_1^K \), \( w_2^K \), ..., \( w_K^K \)

Lookup Table
- \( LT_{W1} \)
- \( LT_{W2} \)
- \( LT_{WK} \)

Convolution

Max Over Time
- \( \text{max}(\cdot) \)

Linear
- \( M^2 \)
- \( M^3 \)

HardTanh

Linear
- \( n_{ku} = \phi(\text{tanh}) \)
Deep SRL

This is the network for a single window. We train/test predicting the entire sentence of tags ("structured outputs") using viterbi approach, similar to other NLP methods.
yesterday, after Microsoft bought Google, the dollar went down under half a euro and the fish market exploded.
yesterday, after Microsoft bought Google, the dollar went down under half a euro and the fish market exploded.
**Word Tag Likelihood (WTL)**

- The network has one output $f(x, i, \theta)$ per tag $i$
- Interpreted as a probability with a softmax over all tags

$$p(i | x, \theta) = \frac{e^{f(x, i, \theta)}}{\sum_j e^{f(x, j, \theta)}}$$

.. we can train directly for that (word tag likelihood) or we could train in a structured way by predicting the entire sentence’s tags.

That should be useful because tags are not independent.
Sentence Tag Likelihood (STL)

- The network score for tag \( k \) at the \( t^{th} \) word is \( f([x]_1^T, k, t, \theta) \)
- \( A_{kl} \) transition score to jump from tag \( k \) to tag \( l \)

Sentence score for a tag path \([i]_1^T\)

\[
s([x]_1^T, [i]_1^T, \tilde{\theta}) = \sum_{t=1}^{T} \left( A_{[i]_{t-1}[i]_t} + f([x]_1^T, [i]_t, t, \theta) \right)
\]
Supervised Benchmark Results

- **Network architectures:**
  - Window (5) approach for POS, CHUNK & NER (300HU)
  - Convolutional (3) for SRL (300+500HU)
  - Word Tag Likelihood (WTL) and Sentence Tag Likelihood (STL)

- **Network features:** lower case words (size 50), capital letters (size 5) dictionary size 100,000 words

<table>
<thead>
<tr>
<th>Approach</th>
<th>POS (PWA)</th>
<th>Chunking (F1)</th>
<th>NER (F1)</th>
<th>SRL (F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark Systems</td>
<td>97.24</td>
<td>94.29</td>
<td>89.31</td>
<td>77.92</td>
</tr>
<tr>
<td>NN+WTL</td>
<td>96.31</td>
<td>89.13</td>
<td>79.53</td>
<td>55.40</td>
</tr>
<tr>
<td>NN+STL</td>
<td>96.37</td>
<td>90.33</td>
<td>81.47</td>
<td>70.99</td>
</tr>
</tbody>
</table>

- **STL helps, but... fair performance.**
- **Capacity mainly in words features... are we training it right?**
Supervised Word Embeddings

- Sentences with similar words should be tagged in the same way:
  - The cat sat on the mat
  - The feline sat on the mat

<table>
<thead>
<tr>
<th>france</th>
<th>jesus</th>
<th>xbox</th>
<th>reddish</th>
<th>scratched</th>
<th>megabits</th>
</tr>
</thead>
<tbody>
<tr>
<td>454</td>
<td>1973</td>
<td>6909</td>
<td>11724</td>
<td>29869</td>
<td>87025</td>
</tr>
</tbody>
</table>

- About **1M** of words in WSJ
- **15%** of most frequent words in the dictionary are seen **90%** of the time
- Cannot expect words to be trained properly!
Improving Word Embedding

- Rare words are not trained properly
- Sentences with similar words should be tagged in the same way:
  - The cat sat on the mat
  - The feline sat on the mat

Only 1M WSJ not enough – let’s use lots of unsupervised data!
Semi-supervised: MTL with Unlabeled Text

- **Language Model:** “is a sentence actually english or not?”
  Implicitly captures: * syntax * semantics
- **Bengio & Ducharme (2001)** Probability of next word given previous words. Overcomplicated – we do not need probabilities here
- **English sentence windows:** Wikipedia (∼ 631M words)
- **Non-english sentence windows:** middle word randomly replaced

  the champion federer wins wimbledon
  vs. the champion saucepan wins wimbledon

- **Multi-class margin cost:**

\[
\sum_{s \in S} \sum_{w \in D} \max(0, 1 - f(s, w^*_s) + f(s, w))
\]

* \(S\): sentence windows  
* \(D\): dictionary  
* \(w^*_s\): true middle word in \(s\)  
* \(f(s, w)\): network score for sentence \(s\) and middle word \(w\)
## Language Model: Embedding

Nearest neighbors in 100-dim. embedding space:

<table>
<thead>
<tr>
<th>Language</th>
<th>Nearest Neighbor</th>
<th>Embedding Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>France</td>
<td>Jesus</td>
<td>454</td>
</tr>
<tr>
<td></td>
<td>Jesus</td>
<td>1973</td>
</tr>
<tr>
<td></td>
<td>Xbox</td>
<td>6909</td>
</tr>
<tr>
<td></td>
<td>Reddish</td>
<td>11724</td>
</tr>
<tr>
<td></td>
<td>Scratched</td>
<td>29869</td>
</tr>
<tr>
<td>Spain</td>
<td>Christ</td>
<td>454</td>
</tr>
<tr>
<td>Italy</td>
<td>God</td>
<td>1973</td>
</tr>
<tr>
<td>Russia</td>
<td>Resurrection</td>
<td>6909</td>
</tr>
<tr>
<td>Poland</td>
<td>Prayer</td>
<td>11724</td>
</tr>
<tr>
<td>England</td>
<td>Yahweh</td>
<td>29869</td>
</tr>
<tr>
<td>Denmark</td>
<td>Josephus</td>
<td>454</td>
</tr>
<tr>
<td>Germany</td>
<td>Moses</td>
<td>1973</td>
</tr>
<tr>
<td>Portugal</td>
<td>Sin</td>
<td>6909</td>
</tr>
<tr>
<td>Sweden</td>
<td>Heaven</td>
<td>11724</td>
</tr>
<tr>
<td>Austria</td>
<td>Salvation</td>
<td>29869</td>
</tr>
<tr>
<td></td>
<td>PlayStation</td>
<td>Yellowish</td>
</tr>
<tr>
<td></td>
<td>Dreamcast</td>
<td>Greenish</td>
</tr>
<tr>
<td></td>
<td>Psnumber</td>
<td>Brownish</td>
</tr>
<tr>
<td></td>
<td>Psnumber</td>
<td>Bluish</td>
</tr>
<tr>
<td></td>
<td>Nintendo</td>
<td>Creamy</td>
</tr>
<tr>
<td></td>
<td>Gamecube</td>
<td>Whitish</td>
</tr>
<tr>
<td></td>
<td>Psp</td>
<td>Blackish</td>
</tr>
<tr>
<td></td>
<td>Amiga</td>
<td>Silvery</td>
</tr>
<tr>
<td></td>
<td>Paler</td>
<td>Blasted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Tanged</td>
</tr>
</tbody>
</table>

(Even fairly rare words are embedded well.)
### Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>POS (PWA)</th>
<th>CHUNK (F1)</th>
<th>NER (F1)</th>
<th>SRL (F1)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baselines</strong></td>
<td>97.24</td>
<td>94.29</td>
<td>89.31</td>
<td>77.92</td>
</tr>
<tr>
<td>[Toutanova ’03] + [Sha ’03] + [Ando ’05] + [Koomen ’05]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NN + WTL</strong></td>
<td>96.31</td>
<td>89.13</td>
<td>79.53</td>
<td>55.40</td>
</tr>
<tr>
<td><strong>NN + STL</strong></td>
<td>96.37</td>
<td>90.33</td>
<td>81.47</td>
<td>70.99</td>
</tr>
<tr>
<td><strong>NN + LM + STL</strong></td>
<td>97.22</td>
<td>94.10</td>
<td>88.67</td>
<td>74.15</td>
</tr>
<tr>
<td><strong>NN + ... + tricks</strong></td>
<td>97.29</td>
<td>94.32</td>
<td>89.95</td>
<td>76.03</td>
</tr>
<tr>
<td>[+suffix]</td>
<td></td>
<td></td>
<td>[+POS]</td>
<td>[+gazetteer] [+Parse Trees]</td>
</tr>
</tbody>
</table>

### NOTES:
- Didn’t compare to benchmarks that used external labeled data.
- [Ando ’05] uses external unlabeled data.
- [Koomen ’05] uses 4 parse trees not provided by the challenge. Using only 1 tree it gets 74.76.
### Software

Code for tagging with POS, NER, CHUNK, SRL + parse trees:

http://ml.nec-labs.com/senna/

<table>
<thead>
<tr>
<th>System</th>
<th>RAM (Mb)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toutanova, 2003</td>
<td>1100</td>
<td>1065</td>
</tr>
<tr>
<td>Shen, 2007</td>
<td>2200</td>
<td>833</td>
</tr>
<tr>
<td>SENNA</td>
<td>32</td>
<td>4</td>
</tr>
</tbody>
</table>

(a) POS

<table>
<thead>
<tr>
<th>System</th>
<th>RAM (Mb)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Koomen, 2005</td>
<td>3400</td>
<td>6253</td>
</tr>
<tr>
<td>SENNA</td>
<td>124</td>
<td>52</td>
</tr>
</tbody>
</table>

(b) SRL

See also Torch: [http://www.torch.ch](http://www.torch.ch)
Recursive NNs for Parsing, Sentiment, ... and more!
(Socher et al., ICML ’13), (Socher et al., EMNLP, ’13))

Build sentence representations using the parse tree to compose embeddings via a nonlinear function taking pairs \((c_1, c_2)\) and output \(p\).

\[
s = W^{\text{score}} p \quad (9)
\]

\[
p = f(W[c_1; c_2] + b)
\]
Paragraph Vector

(Le & Mikolov, ’14)

A Paragraph Vector (a vector that represents a paragraph/doc) learned by:

1) Predicting the words in a doc;
2) predict \( n \)-grams in the doc:

At test time, for a new document, one needs to learn its vector, this can encode word order via the \( n \)-gram prediction approach.
## Comparison of CNN, RNN & PV (Kim ’14)

<table>
<thead>
<tr>
<th>Model</th>
<th>MR</th>
<th>SST-1</th>
<th>SST-2</th>
<th>Subj</th>
<th>TREC</th>
<th>CR</th>
<th>MPQA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN-rand</td>
<td>76.1</td>
<td>45.0</td>
<td>82.7</td>
<td>89.6</td>
<td>91.2</td>
<td>79.8</td>
<td>83.4</td>
</tr>
<tr>
<td>CNN-static</td>
<td>81.0</td>
<td>45.5</td>
<td>86.8</td>
<td>93.0</td>
<td>92.8</td>
<td>84.7</td>
<td>89.6</td>
</tr>
<tr>
<td>CNN-non-static</td>
<td><strong>81.5</strong></td>
<td>48.0</td>
<td>87.2</td>
<td>93.4</td>
<td>93.6</td>
<td>84.3</td>
<td>89.5</td>
</tr>
<tr>
<td>CNN-multichannel</td>
<td>81.1</td>
<td>47.4</td>
<td><strong>88.1</strong></td>
<td>93.2</td>
<td>92.2</td>
<td><strong>85.0</strong></td>
<td>89.4</td>
</tr>
<tr>
<td>RAE (Socher et al., 2011)</td>
<td>77.7</td>
<td>43.2</td>
<td>82.4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>86.4</td>
</tr>
<tr>
<td>MV-RNN (Socher et al., 2012)</td>
<td>79.0</td>
<td>44.4</td>
<td>82.9</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>RNTN (Socher et al., 2013)</td>
<td>–</td>
<td>45.7</td>
<td>85.4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>DCNN (Kalchbrenner et al., 2014)</td>
<td>–</td>
<td>48.5</td>
<td>86.8</td>
<td>–</td>
<td>93.0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Paragraph-Vec (Le and Mikolov, 2014)</td>
<td>–</td>
<td><strong>48.7</strong></td>
<td>87.8</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>CCAE (Hermann and Blunsom, 2013)</td>
<td>77.8</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>87.2</td>
</tr>
<tr>
<td>Sent-Parser (Dong et al., 2014)</td>
<td>79.5</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>86.3</td>
</tr>
<tr>
<td>NBSVM (Wang and Manning, 2012)</td>
<td>79.4</td>
<td>–</td>
<td>–</td>
<td>93.2</td>
<td>–</td>
<td>81.8</td>
<td>86.3</td>
</tr>
<tr>
<td>MNB (Wang and Manning, 2012)</td>
<td>79.0</td>
<td>–</td>
<td>–</td>
<td><strong>93.6</strong></td>
<td>–</td>
<td>80.0</td>
<td>86.3</td>
</tr>
<tr>
<td>G-Dropout (Wang and Manning, 2013)</td>
<td>79.0</td>
<td>–</td>
<td>–</td>
<td>93.4</td>
<td>–</td>
<td>82.1</td>
<td>86.1</td>
</tr>
<tr>
<td>F-Dropout (Wang and Manning, 2013)</td>
<td>79.1</td>
<td>–</td>
<td>–</td>
<td><strong>93.6</strong></td>
<td>–</td>
<td>81.9</td>
<td>86.3</td>
</tr>
<tr>
<td>Tree-CRF (Nakagawa et al., 2010)</td>
<td>77.3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>81.4</td>
<td>86.1</td>
</tr>
<tr>
<td>CRF-PR (Yang and Cardie, 2014)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>82.7</td>
<td>–</td>
</tr>
<tr>
<td>SVM$_S$ (Silva et al., 2011)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td><strong>95.0</strong></td>
<td>–</td>
</tr>
</tbody>
</table>

Table 2: Results of our CNN models against other methods. **RAE**: Recursive Autoencoders with pre-trained word vectors from Wikipedia (Socher et al., 2011). **MV-RNN**: Matrix-Vector Recursive Neural Network with parse trees (Socher et al., 2012). **RNTN**: Recursive Neural Tensor Network with tensor-based feature function and parse trees (Socher et al., 2013). **DCNN**: Dynamic Convolutional Neural Network with k-max pooling (Kalchbrenner et al., 2014). **Paragraph-Vec**: Logistic regression on top of paragraph vectors (Le and Mikolov, 2014). **CCAE**: Combinatorial Category Autoencoders with combinatorial category grammar operators (Hermann and Blunsom, 2013). **Sent-Parser**: Sentiment analysis-specific parser (Dong et al., 2014). **NBSVM, MNB**: Naive Bayes SVM and Multinomial Naive Bayes with uni-bigrams from Wang and Manning (2012). **G-Dropout, F-Dropout**: Gaussian Dropout and Fast Dropout from Wang and Manning (2013). **Tree-CRF**: Dependency tree
Some More Recent Work

- Compositionality approaches by Marco Baroni’s group:
  Words are combined with linear matrices dependent on the P.O.S.:
  G. Dinu and M. Baroni. How to make words with vectors: Phrase generation in distributional semantics. ACL ’14.

- Document representation by Phil Blunson’s group:
  Variants of convolutional networks for text:
  Kalchbrenner et al. A Convolutional Neural Network for Modelling Sentences. ACL ’14

Good tutorial slides from these teams covering multiple topics:
New Directions in Vector Space Models of Meaning
http://www.cs.ox.ac.uk/files/6605/aclVectorTutorial.pdf
Summary

- Generic end-to-end deep learning system for NLP tasks

- Word embeddings combined to form sentence or document embeddings can perform well on supervised tasks.

- Previous common belief in NLP: engineering syntactic features necessary for semantic tasks.
  One can do well by engineering a model/algorithm rather than features.

Attitude is changing in recent years... let’s see what happens!