Refresher: The simple word2vec model

- Main cost function $J$: 

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

- With probabilities defined as: 

$$p(w_O | w_I) = \frac{\exp \left( v'_{w_O}^\top v_{w_I} \right)}{\sum_{w=1}^{W} \exp \left( v'_{w}^\top v_{w_I} \right)}$$

- We derived the gradient for the internal vectors $v_{w_I}$ (v_c on the board)
Calculating all gradients!

- We went through gradients for each center vector $v$ in a window
- We also need gradients for external vectors $v'$ (u on the board)
- Derive!

- Generally in each window we will compute updates for all parameters that are being used in that window.
- For example window size $c = 1$, sentence: “I like learning .”
- First window computes gradients for:
  - internal vector $v_{\text{like}}$ and external vectors $v'_{\text{I}}$ and $v'_{\text{learning}}$
- Next window in that sentence?
Compute all vector gradients!

• We often define the set of ALL parameters in a model in terms of one long vector $\theta$

• In our case with d-dimensional vectors and $V$ many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ v'_{aardvark} \\ v'_a \\ \vdots \\ v'_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$
Gradient Descent

• To minimize $J(\theta)$ over the full batch (the entire training data) would require us to compute gradients for all windows.

• Updates would be for each element of $\theta$:

$$
\theta^\text{new}_j = \theta^\text{old}_j - \alpha \frac{\partial}{\partial \theta^\text{old}_j} J(\theta)
$$

• With step size $\alpha$

• In matrix notation for all parameters:

$$
\theta^\text{new} = \theta^\text{old} - \alpha \nabla_{\theta} J(\theta)
$$
Vanilla Gradient Descent Code

\[ \theta_{new} = \theta_{old} - \alpha \nabla_{\theta} J(\theta) \]

```
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```
Intuition

- For a simple convex function over two parameters.

- Contour lines show levels of objective function

- See Whiteboard
Stochastic Gradient Descent

- But Corpus may have 40B tokens and windows
- You would wait a very long time before making a single update!
- Very bad idea for pretty much all neural nets!
- Instead: We will update parameters after each window $t$ → Stochastic gradient descent (SGD)

$$\theta_{new} = \theta_{old} - \alpha \nabla_{\theta} J_t(\theta)$$

```python
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J, window, theta)
    theta = theta - alpha * theta_grad
```
Stochastic gradients with word vectors!

- But in each window, we only have at most $2c - 1$ words, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \mathbf{v}_{\text{like}} \\ \vdots \\ 0 \\ \mathbf{v}'_I \\ \vdots \\ \mathbf{v}'_{\text{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$
Stochastic gradients with word vectors!

• We may as well only update the word vectors that actually appear!

• Solution: either keep around hash for word vectors or only update certain columns of full embedding matrix $L$ and $L'$

$$
\begin{bmatrix}
\vdots & \cdots & V \\
\vdots & \vdots & \vdots \\
\end{bmatrix}
$$

• Important if you have millions of word vectors and do distributed computing to not have to send gigantic updates around.
Approximations: PSet 1

• The normalization factor is too computationally expensive

\[ p(w_O|w_I) = \frac{\exp \left( v'_{w_O} \top v_{w_I} \right)}{\sum_{w=1}^{W} \exp \left( v'_w \top v_{w_I} \right)} \]

• Hence, in PSet1 you will implement the skip-gram model

• Main idea: train binary logistic regressions for a true pair (center word and word in its context window) and a couple of random pairs (the center word with a random word)
PSet 1: The skip-gram model and negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)

\[
\log \sigma(v_w^\top v) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \left[ \log \sigma(-v_w^\top v) \right]
\]

- Where \( k \) is the number of negative samples and we use,

- The sigmoid function! \( \sigma(x) = \frac{1}{1+e^{-x}} \)
  (we’ll become good friends soon)

- So we maximize the probability of two words co-occurring in first log
PSet 1: The skip-gram model and negative sampling

- Slightly clearer notation:

\[ \log \sigma \left( v_{wI}^T v_{wO}' \right) + \sum_{i \sim P_n(w)} \log \sigma \left( -v_{wI}^T v_{wi}' \right) \]

- Max. probability that real outside word appears, minimize prob. that random words appear around center word

- \( P_n = U(w)^{3/4}/Z \), the unigram distribution \( U(w) \) raised to the 3/4rd power (We provide this function in the starter code).

- The power makes less frequent words be sampled more often
PSet 1: The continuous bag of words model

• Main idea for continuous bag of words (CBOW): Predict center word from sum of surrounding word vectors instead of predicting surrounding single words from center word as in skip-gram model

• To make PSet slightly easier:

  The implementation for the CBOW model is not required and for bonus points!
What to do with the two sets of vectors?

• We end up with $L$ and $L'$ from all the vectors $v$ and $v'$

• Both capture similar co-occurrence information. It turns out, the best solution is to simply sum them up:

$$L_{\text{final}} = L + L'$$

• One of many hyperparameters explored in *GloVe: Global Vectors for Word Representation* (Pennington et al. (2014))
How to evaluate word vectors?

• Related to general evaluation in NLP: Intrinsic vs extrinsic
  • Intrinsic:
    • Evaluation on a specific/intermediate subtask
    • Fast to compute
    • Helps to understand that system
    • Not clear if really helpful unless correlation to real task is established
  • Extrinsic:
    • Evaluation on a real task
    • Can take a long time to compute accuracy
    • Unclear if the subsystem is the problem or its interaction or other subsystems
    • If replacing one subsystem with another improves accuracy → Winning!
Intrinsic word vector evaluation

• Word Vector Analogies: Syntactic and Semantic

\[
\begin{align*}
\text{a:b :: c:?} & \\
\text{man:woman :: king:?}
\end{align*}
\]

\[
d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}
\]

• Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions

• Discarding the input words from the search!

• Problem: What if the information is there but not linear?
Intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and Semantic examples from http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt

: city-in-state

Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

problem: different cities may have same name
Intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from

: capital-world
Abuja Nigeria Accra Ghana
Abuja Nigeria Algiers Algeria
Abuja Nigeria Amman Jordan
Abuja Nigeria Ankara Turkey
Abuja Nigeria Antananarivo Madagascar
Abuja Nigeria Apia Samoa
Abuja Nigeria Ashgabat Turkmenistan
Abuja Nigeria Asmara Eritrea
Abuja Nigeria Astana Kazakhstan

problem: can change
Intrinsic word vector evaluation

• Word Vector Analogies: **Syntactic** and Semantic examples from:

  : gram4-superlative
  bad worst big biggest
  bad worst bright brightest
  bad worst cold coldest
  bad worst cool coolest
  bad worst dark darkest
  bad worst easy easiest
  bad worst fast fastest
  bad worst good best
  bad worst great greatest
Intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from gram7-past-tense:
  
dancing danced decreasing decreased
dancing danced describing described
dancing danced enhancing enhanced
dancing danced falling fell
dancing danced feeding fed
dancing danced flying flew
dancing danced generating generated
dancing danced going went
dancing danced hiding hid
dancing danced hitting hit
### Analogy evaluation and hyperparameters

- Most careful analysis so far: Glove word vectors (which also capture cooccurrence counts but more directly so than skip-gram)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ivLBL</td>
<td>100</td>
<td>1.5B</td>
<td>55.9</td>
<td>50.1</td>
<td>53.2</td>
</tr>
<tr>
<td>HPCA</td>
<td>100</td>
<td>1.6B</td>
<td>4.2</td>
<td>16.4</td>
<td>10.8</td>
</tr>
<tr>
<td>GloVe</td>
<td>100</td>
<td>1.6B</td>
<td>67.5</td>
<td>54.3</td>
<td>60.3</td>
</tr>
<tr>
<td>SG</td>
<td>300</td>
<td>1B</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>CBOV</td>
<td>300</td>
<td>1.6B</td>
<td>16.1</td>
<td>52.6</td>
<td>36.1</td>
</tr>
<tr>
<td>vLBL</td>
<td>300</td>
<td>1.5B</td>
<td>54.2</td>
<td>64.8</td>
<td>60.0</td>
</tr>
<tr>
<td>ivLBL</td>
<td>300</td>
<td>1.5B</td>
<td>65.2</td>
<td>63.0</td>
<td>64.0</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>1.6B</td>
<td>80.8</td>
<td>61.5</td>
<td>70.3</td>
</tr>
<tr>
<td>SVD</td>
<td>300</td>
<td>6B</td>
<td>6.3</td>
<td>8.1</td>
<td>7.3</td>
</tr>
<tr>
<td>SVD-S</td>
<td>300</td>
<td>6B</td>
<td>36.7</td>
<td>46.6</td>
<td>42.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>300</td>
<td>6B</td>
<td>56.6</td>
<td>63.0</td>
<td>60.1</td>
</tr>
<tr>
<td>CBOV†</td>
<td>300</td>
<td>6B</td>
<td>63.6</td>
<td>67.4</td>
<td>65.7</td>
</tr>
<tr>
<td>SG†</td>
<td>300</td>
<td>6B</td>
<td>73.0</td>
<td>66.0</td>
<td>69.1</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>6B</td>
<td>77.4</td>
<td>67.0</td>
<td>71.7</td>
</tr>
<tr>
<td>CBOV</td>
<td>1000</td>
<td>6B</td>
<td>57.3</td>
<td>68.9</td>
<td>63.7</td>
</tr>
<tr>
<td>SG</td>
<td>1000</td>
<td>6B</td>
<td>66.1</td>
<td>65.1</td>
<td>65.6</td>
</tr>
<tr>
<td>SVD-L</td>
<td>300</td>
<td>42B</td>
<td>38.4</td>
<td>58.2</td>
<td>49.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>300</td>
<td>42B</td>
<td>81.9</td>
<td>69.3</td>
<td>75.0</td>
</tr>
</tbody>
</table>
Analogy evaluation and hyperparameters

- Asymmetric context (only words to the left) are not as good

![Graphs showing accuracy vs. vector dimension and window size]

- Best dimensions ~300, slight drop-off afterwards
- But this might be different for downstream tasks!

- Window size of 8 around each center word is good for Glove vectors
Analogies in evaluation and hyperparameters

- More training time helps

Figure 4: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus (Wikipedia 2014 + Gigaword 5) with the same 400,000 word vocabulary, and use a symmetric context window of size 10.

In Fig. 4, we plot the overall performance on the analogy task as a function of training time. The two x-axes at the bottom indicate the corresponding number of training iterations for GloVe and negative samples for word2vec. We note that word2vec’s performance actually decreases if the number of negative samples increases beyond about 10. Presumably this is because the negative sampling method does not approximate the target probability distribution well.

For the same corpus, vocabulary, window size, and training time, GloVe consistently outperforms word2vec. It achieves better results faster, and also obtains the best results irrespective of speed.

Conclusion

Recently, considerable attention has been focused on the question of whether distributional word representations are best learned from count-based methods or from prediction-based methods. Currently, prediction-based models garner substantial support; for example, Baroni et al. (2014) argue that these models perform better across a range of tasks. In this work we argue that the two classes of methods are not dramatically different at a fundamental level since they both probe the underlying co-occurrence statistics of the corpus, but the efficiency with which the count-based methods capture global statistics can be advantageous.

We construct a model that utilizes this main benefit of count data while simultaneously capturing the meaningful linear substructures prevalent in recent log-bilinear prediction-based methods like word2vec. The result, GloVe, is a new global log-bilinear regression model for the unsupervised learning of word representations that outperforms other models on word analogy, word similarity, and named entity recognition tasks.

Acknowledgments

We thank the anonymous reviewers for their valuable comments. Stanford University gratefully acknowledges the support of the Defense Threat Reduction Agency (DTRA) under Air Force Research Laboratory (AFRL) contract no. FA8650-10-C-7020 and the Defense Advanced Research Projects Agency (DARPA) Deep Exploration and Filtering of Text (DEFT) Program under AFRL contract no. FA8750-13-2-0040. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the DTRA, AFRL, DEFT, or the US government.
Analogy evaluation and hyperparameters

- More data helps, Wikipedia is better than news text!
Intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353

<table>
<thead>
<tr>
<th>Word 1</th>
<th>Word 2</th>
<th>Human (mean)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tiger</td>
<td>cat</td>
<td>7.35</td>
</tr>
<tr>
<td>tiger</td>
<td>tiger</td>
<td>10.00</td>
</tr>
<tr>
<td>book</td>
<td>paper</td>
<td>7.46</td>
</tr>
<tr>
<td>computer</td>
<td>internet</td>
<td>7.58</td>
</tr>
<tr>
<td>plane</td>
<td>car</td>
<td>5.77</td>
</tr>
<tr>
<td>professor</td>
<td>doctor</td>
<td>6.62</td>
</tr>
<tr>
<td>stock</td>
<td>phone</td>
<td>1.62</td>
</tr>
<tr>
<td>stock</td>
<td>CD</td>
<td>1.31</td>
</tr>
<tr>
<td>stock</td>
<td>jaguar</td>
<td>0.92</td>
</tr>
</tbody>
</table>
Correlation evaluation

- Word vector distances and their correlation with human judgments

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>WS353</th>
<th>MC</th>
<th>RG</th>
<th>SCWS</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVD</td>
<td>6B</td>
<td>35.3</td>
<td>35.1</td>
<td>42.5</td>
<td>38.3</td>
<td>25.6</td>
</tr>
<tr>
<td>SVD-S</td>
<td>6B</td>
<td>56.5</td>
<td>71.5</td>
<td>71.0</td>
<td>53.6</td>
<td>34.7</td>
</tr>
<tr>
<td>SVD-L</td>
<td>6B</td>
<td>65.7</td>
<td>72.7</td>
<td>75.1</td>
<td>56.5</td>
<td>37.0</td>
</tr>
<tr>
<td>CBOW†</td>
<td>6B</td>
<td>57.2</td>
<td>65.6</td>
<td>68.2</td>
<td>57.0</td>
<td>32.5</td>
</tr>
<tr>
<td>SG†</td>
<td>6B</td>
<td>62.8</td>
<td>65.2</td>
<td>69.7</td>
<td>58.1</td>
<td>37.2</td>
</tr>
<tr>
<td>GloVe</td>
<td>6B</td>
<td>65.8</td>
<td>72.7</td>
<td>77.8</td>
<td>53.9</td>
<td>38.1</td>
</tr>
<tr>
<td>SVD-L</td>
<td>42B</td>
<td>74.0</td>
<td>76.4</td>
<td>74.1</td>
<td>58.3</td>
<td>39.9</td>
</tr>
<tr>
<td>GloVe</td>
<td>42B</td>
<td>75.9</td>
<td>83.6</td>
<td>82.9</td>
<td>59.6</td>
<td>47.8</td>
</tr>
<tr>
<td>CBOW*</td>
<td>100B</td>
<td>68.4</td>
<td>79.6</td>
<td>75.4</td>
<td>59.4</td>
<td>45.5</td>
</tr>
</tbody>
</table>

- Some ideas from Glove paper have been shows to improve skip-gram (SG) model also (e.g. sum both vectors)
But what about ambiguity?

• You may hope that one vector captures both kinds of information (run = verb and noun) but then vector is pulled in different directions

• Alternative described in: *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)

• Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters $\text{bank}_1$, $\text{bank}_2$, etc
But what about ambiguity?

- *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)
Extrinsic word vector evaluation

• Extrinsic evaluation of word vectors: All subsequent tasks in this class

• One example where good word vectors should help directly: named entity recognition: finding a person, organization or location

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
<th>ACE</th>
<th>MUC7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete</td>
<td>91.0</td>
<td>85.4</td>
<td>77.4</td>
<td>73.4</td>
</tr>
<tr>
<td>SVD</td>
<td>90.8</td>
<td>85.7</td>
<td>77.3</td>
<td>73.7</td>
</tr>
<tr>
<td>SVD-S</td>
<td>91.0</td>
<td>85.5</td>
<td>77.6</td>
<td>74.3</td>
</tr>
<tr>
<td>SVD-L</td>
<td>90.5</td>
<td>84.8</td>
<td>73.6</td>
<td>71.5</td>
</tr>
<tr>
<td>HPCA</td>
<td>92.6</td>
<td><strong>88.7</strong></td>
<td>81.7</td>
<td>80.7</td>
</tr>
<tr>
<td>HSMN</td>
<td>90.5</td>
<td>85.7</td>
<td>78.7</td>
<td>74.7</td>
</tr>
<tr>
<td>CW</td>
<td>92.2</td>
<td>87.4</td>
<td>81.7</td>
<td>80.2</td>
</tr>
<tr>
<td>CBOW</td>
<td>93.1</td>
<td>88.2</td>
<td>82.2</td>
<td>81.1</td>
</tr>
<tr>
<td>GloVe</td>
<td><strong>93.2</strong></td>
<td>88.3</td>
<td><strong>82.9</strong></td>
<td><strong>82.2</strong></td>
</tr>
</tbody>
</table>

• Next: How to use word vectors in neural net models!
Simple single word classification

• What is the major benefit of deep learned word vectors?
  • Ability to also classify words accurately
  • Countries cluster together → classifying location words should be possible with word vectors
  • Incorporate any information into them other tasks
  • Project sentiment into words to find most positive/negative words in corpus
The softmax

Logistic regression = Softmax classification on word vector $x$ to obtain probability for class $y$:

$$p(y|x) = \frac{\exp(W_{y}.x)}{\sum_{c=1}^{C} \exp(W_c.x)}$$

where: $W \in \mathbb{R}^{C \times d}$

Generalizes >2 classes
(for just binary sigmoid unit would suffice as in skip-gram)
The softmax - details

- Terminology: Loss function = cost function = objective function
- Loss for softmax: Cross entropy

- To compute \( p(y|x) \): first take the \( y \)'th row of \( W \) and multiply that with row with \( x \):

\[
W_y . x = \sum_{i=1}^{d} W_{yi} x_i = f_y
\]

- Compute all \( f_c \) for \( c=1,\ldots,C \)
- Normalize to obtain probability with softmax function:

\[
p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)}
\]
The softmax and cross-entropy error

- The loss wants to maximize the probability of the correct class $y$

- Hence, we minimize the negative log probability of that class:

$$ - \log p(y|x) = - \log \left( \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} \right) $$

- As before: we sum up multiple cross entropy errors if we have multiple classifications in our total error function over the corpus (more next lecture)
Background: The Cross entropy error

• Assuming a ground truth (or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: $p = [0,\ldots,0,1,0,\ldots,0]$ and our computed probability is $q$, then the cross entropy is:

$$H(p, q) = -\sum_{c=1}^{C} p(c) \log q(c)$$

• Because of one-hot $p$, the only term left is the negative probability of the true class

• Cross-entropy can be re-written in terms of the entropy and Kullback-Leibler divergence between the two distributions:

$$H(p, q) = H(p) + D_{KL}(p||q)$$
The KL divergence

- **Cross entropy:** \( H(p, q) = H(p) + D_{KL}(p\|q) \)
- Because \( p \) is zero in our case (and even if it wasn’t it would be fixed and have no contribution to gradient), to minimize this is equal to minimizing the KL divergence

- The KL divergence is **not a distance** but a non-symmetric measure of the difference between two probability distributions \( p \) and \( q \)

\[
D_{KL}(p\|q) = \sum_{c=1}^{C} p(c) \log \frac{p(c)}{q(c)}
\]
PSet 1

- Derive the gradient of the cross entropy error with respect to the input word vector $x$ and the matrix $W$
Simple single word classification

- Example: Sentiment

- Two options: train only *softmax* weights $W$ and fix word vectors or also train word vectors

- Question: What are the advantages and disadvantages of training the word vectors?

  - Pro: better fit on training data
  - Con: Worse generalization because the words move in the vector space
Visualization of sentiment trained word vectors
Next level up: Window classification

• Single word classification has no context!

• Let’s add context by taking in windows and classifying the center word of that window!

• Possible: Softmax and cross entropy error or max-margin loss

• Next class!