Natural Language Processing with Deep Learning
CS224N/Ling284

Christopher Manning
Lecture 2: Word Vectors, Word Senses, and Classifier Review
Lecture Plan

Lecture 2: Word Vectors and Word Senses

1. Finish looking at word vectors and word2vec (10 mins)
2. Optimization basics (8 mins)
3. Can we capture this essence more effectively by counting? (12 mins)
4. The GloVe model of word vectors (10 min)
5. Evaluating word vectors (12 mins)
6. Word senses (6 mins)
7. Review of classification and how neural nets differ (10 mins)
8. Course advice (2 mins)

Goal: be able to read word embeddings papers by the end of class
1. Review: Main idea of word2vec

- Start with random word vectors
- Iterate through each word in the whole corpus
- Try to predict surrounding words using word vectors

\[
P(o | c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}
\]

- Update vectors so you can predict better

- This algorithm learns word vectors that capture word similarity and meaningful directions in a wordspace
**Word2vec parameters and computations**

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (fairly often)
Word2vec maximizes objective function by putting similar words nearby in space.
2. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- **Idea**: for current value of $\theta$, calculate gradient of $J(\theta)$, then take small step in the direction of negative gradient. Repeat.

Note: Our objectives may not be convex like this.
Gradient Descent

- Update equation (in matrix notation):

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

\(\alpha = \text{step size or learning rate}\)

- Update equation (for a single parameter):

\[
\theta_{j}^{\text{new}} = \theta_{j}^{\text{old}} - \alpha \frac{\partial}{\partial \theta_{j}^{\text{old}}} J(\theta)
\]

- Algorithm:

```python
while True:
    theta_grad = evaluate_gradient(J, corpus, theta)
    theta = theta - alpha * theta_grad
```
Stochastic Gradient Descent

- **Problem**: $J(\theta)$ is a function of all windows in the corpus (potentially billions!)
  - So $\nabla_{\theta} J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!

- **Very** bad idea for pretty much all neural nets!
- **Solution**: Stochastic gradient descent (SGD)
  - Repeatedly sample windows, and update after each one
- Algorithm:

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

- Iteratively take gradients at each such window for SGD
- But in each window, we only have at most \(2m + 1\) words, so \(\nabla_{\theta} J_t(\theta)\) is very sparse!

\[
\nabla_{\theta} J_t(\theta) = \begin{bmatrix}
0 \\
\vdots \\
\nabla_{v_{like}} \\
\vdots \\
0 \\
\nabla_{u_I} \\
\vdots \\
\nabla_{u_{learning}} \\
\vdots 
\end{bmatrix} \in \mathbb{R}^{2dV}
\]
Stochastic gradients with word vectors!

• We might only update the word vectors that actually appear!

• Solution: either you need sparse matrix update operations to only update certain rows of full embedding matrices $U$ and $V$, or you need to keep around a hash for word vectors

• If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!
1b. Word2vec: More details

Why two vectors? → Easier optimization. Average both at the end
  • But can do algorithm with just one vector per word

Two model variants:

1. Skip-grams (SG)
   Predict context (“outside”) words (position independent) given center word

2. Continuous Bag of Words (CBOW)
   Predict center word from (bag of) context words

We presented: Skip-gram model

Additional efficiency in training:

1. Negative sampling
   So far: Focus on naïve softmax (simpler, but expensive, training method)
The skip-gram model with negative sampling (HW2)

• The normalization factor is too computationally expensive.

\[ P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \]

• Hence, in standard word2vec and HW2 you implement the skip-gram model with negative sampling

• Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)
The skip-gram model with negative sampling (HW2)

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function (they maximize): \( J(\theta) = \frac{1}{T} \sum_{t=1}^{T} J_t(\theta) \)

\[
J_t(\theta) = \log \sigma \left( u_o^T v_c \right) + \sum_{i=1}^{k} \mathbb{E}_{j \sim P(w)} \left[ \log \sigma \left( -u_j^T v_c \right) \right]
\]

- 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 →
The skip-gram model with negative sampling (HW2)

• Notation more similar to class and HW2:

\[
J_{\text{neg-sample}}(o, v_c, U) = -\log(\sigma(u_o^T v_c)) - \sum_{k=1}^{K} \log(\sigma(-u_k^T v_c))
\]

• We take \( k \) negative samples (using word probabilities)
• Maximize probability that real outside word appears, minimize prob. that random words appear around center word

• \( P(w) = U(w)^{3/4}/Z \), the unigram distribution \( U(w) \) raised to the 3/4 power (We provide this function in the starter code).
• The power makes less frequent words be sampled more often
3. Why not capture co-occurrence counts directly?

With a co-occurrence matrix $X$

- 2 options: windows vs. full document
- Window: Similar to word2vec, use window around each word \( \rightarrow \) captures both syntactic (POS) and semantic information
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.
Window based co-occurrence matrix

- **Example corpus:**
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

<table>
<thead>
<tr>
<th>counts</th>
<th>I</th>
<th>like</th>
<th>enjoy</th>
<th>deep</th>
<th>learning</th>
<th>NLP</th>
<th>flying</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>like</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>enjoy</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>deep</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>learning</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NLP</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flying</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: requires a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust
Solution: Low dimensional vectors

• Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector

• Usually 25–1000 dimensions, similar to word2vec

• How to reduce the dimensionality?
Method: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix $X$

Factorizes $X$ into $U\Sigma V^T$, where $U$ and $V$ are orthonormal.

$X^k = U \Sigma V^T$

Retain only $k$ singular values, in order to generalize.

$\hat{X}$ is the best rank $k$ approximation to $X$, in terms of least squares.

Classic linear algebra result. Expensive to compute for large matrices.
Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

```python
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy", "deep", "learning", "NLP", "flying", "."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])
U, s, Vh = la.svd(X, full_matrices=False)
```
Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.
Printing first two columns of U corresponding to the 2 biggest singular values

```python
for i in range(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```
Hacks to X (several used in Rohde et al. 2005)

Scaling the counts in the cells can help *a lot*

- Problem: function words (*the, he, has*) are too frequent $\rightarrow$ syntax has too much impact. Some fixes:
  - $\min(X,t)$, with $t \approx 100$
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.
Interesting syntactic patterns emerge in the vectors

COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005
Interesting semantic patterns emerge in the vectors

Figure 13: Multidimensional scaling for nouns and their associated verbs.

Table 10
The 10 nearest neighbors and their percent correlation similarities for a set of nouns, under the COALS-14K model.

Table 11
The 10 nearest neighbors for a set of verbs, according to the COALS-14K model.

Table 12
The 10 nearest neighbors for a set of adjectives, according to the COALS-14K model.

COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005
4. Towards GloVe: Count based vs. direct prediction

- **LSA, HAL (Lund & Burgess),**
- **COALS, Hellinger-PCA** (Rohde et al, Lebret & Collobert)

- Fast training
- Efficient usage of statistics
  - Primarily used to capture word similarity
  - Disproportionate importance given to large counts

- **Skip-gram/CBOW** (Mikolov et al)
- **NNLM, HLBL, RNN** (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton)

- Scales with corpus size
- Inefficient usage of statistics
  - Generate improved performance on other tasks
  - Can capture complex patterns beyond word similarity
**Ratios of co-occurrence probabilities can encode meaning components**

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components.

<table>
<thead>
<tr>
<th></th>
<th>$x = \text{solid}$</th>
<th>$x = \text{gas}$</th>
<th>$x = \text{water}$</th>
<th>$x = \text{random}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(x</td>
<td>\text{ice})$</td>
<td>large</td>
<td>small</td>
<td>large</td>
</tr>
<tr>
<td>$P(x</td>
<td>\text{steam})$</td>
<td>small</td>
<td>large</td>
<td>large</td>
</tr>
<tr>
<td>$\frac{P(x</td>
<td>\text{ice})}{P(x</td>
<td>\text{steam})}$</td>
<td>large</td>
<td>small</td>
</tr>
</tbody>
</table>
Ratios of co-occurrence probabilities can encode meaning components

\[
\begin{array}{cccc}
\multicolumn{4}{c}{\text{Crucial insight:}} \\
\multicolumn{4}{c}{\text{Ratios of co-occurrence probabilities can encode meaning components}} \\
\hline
& x = \text{solid} & x = \text{gas} & x = \text{water} & x = \text{fashion} \\
\hline
P(x|\text{ice}) & 1.9 \times 10^{-4} & 6.6 \times 10^{-5} & 3.0 \times 10^{-3} & 1.7 \times 10^{-5} \\
\hline
P(x|\text{steam}) & 2.2 \times 10^{-5} & 7.8 \times 10^{-4} & 2.2 \times 10^{-3} & 1.8 \times 10^{-5} \\
\hline
\frac{P(x|\text{ice})}{P(x|\text{steam})} & 8.9 & 8.5 \times 10^{-2} & 1.36 & 0.96 \\
\end{array}
\]
Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: 
\[ w_i \cdot w_j = \log P(i|j) \]

with vector differences 
\[ w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)} \]
Combining the best of both worlds

GloVe  [Pennington et al., EMNLP 2014]

\[ w_i \cdot w_j = \log P(i|j) \]

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

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors
GloVe results

Nearest words to frog:
1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus
5. 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 exactly one subsystem with another improves accuracy → Winning!
Intrinsic word vector evaluation

- Word Vector Analogies

\[
\text{man:woman :: king:?}
\]

- 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?
Glove Visualizations
Glove Visualizations: Company - CEO
## Analogy evaluation and hyperparameters

### Glove word vectors evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<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>CBOW†</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><strong>77.4</strong></td>
<td><strong>67.0</strong></td>
<td><strong>71.7</strong></td>
</tr>
</tbody>
</table>
Analogy evaluation and hyperparameters

- More data helps
- Wikipedia is better than news text!

- Dimensionality
- Good dimension is ~300
Another 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</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 shown to improve skip-gram (SG) model also (e.g. sum both vectors)
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>88.7</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>93.2</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!
6. Word senses and word sense ambiguity

• Most words have lots of meanings!
  • Especially common words
  • Especially words that have existed for a long time

• Example: pike

• Does one vector capture all these meanings or do we have a mess?
pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one’s way (pike along)
- In Australian English, pike means to pull out from doing something: *I reckon he could have climbed that cliff, but he piked!*
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 bank$_1$, bank$_2$, etc
Linear Algebraic Structure of Word Senses, with Applications to Polysemy  
(Arora, …, Ma, …, TACL 2018)

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec

\[ v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3} \]

- Where \( \alpha_1 = \frac{f_1}{f_1+f_2+f_3} \), etc., for frequency \( f \)

- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)

<table>
<thead>
<tr>
<th>tie</th>
<th>trousers</th>
<th>blouse</th>
<th>waistcoat</th>
<th>skirt</th>
<th>sleeved</th>
<th>pants</th>
<th>season</th>
<th>teams</th>
<th>winning</th>
<th>league</th>
<th>finished</th>
<th>championship</th>
<th>scoreline</th>
<th>goalless</th>
<th>equaliser</th>
<th>clinching</th>
<th>scoreless</th>
<th>replay</th>
<th>wires</th>
<th>cables</th>
<th>wiring</th>
<th>electrical</th>
<th>wire</th>
<th>cable</th>
<th>operatic</th>
<th>soprano</th>
<th>mezzo</th>
<th>contralto</th>
<th>baritone</th>
<th>coloratura</th>
</tr>
</thead>
</table>
7. Classification review and notation

• Generally we have a training dataset consisting of samples

\[ \{x_i, y_i\}_{i=1}^{N} \]

• \( x_i \) are inputs, e.g. words (indices or vectors!), sentences, documents, etc.
  • Dimension \( d \)

• \( y_i \) are labels (one of \( C \) classes) we try to predict, for example:
  • classes: sentiment, named entities, buy/sell decision
  • other words
  • later: multi-word sequences
Classification intuition

- Training data: \( \{x_i, y_i\}_{i=1}^N \)

- Simple illustration case:
  - Fixed 2D word vectors to classify
  - Using softmax/logistic regression
  - Linear decision boundary

- **Traditional ML/Stats approach:** assume \( x_i \) are fixed, train (i.e., set) softmax/logistic regression weights \( W \in \mathbb{R}^{C \times d} \) to determine a decision boundary (hyperplane) as in the picture

- **Method:** For each fixed \( x \), predict:

\[
p(y|x) = \frac{\exp(W_{y.x})}{\sum_{c=1}^C \exp(W_{c.x})}
\]
Details of the softmax classifier

\[ p(y|x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^{C} \exp(W_c \cdot x)} \]

We can tease apart the prediction function into three steps:

1. Take the \( y^{th} \) row of \( W \) and multiply that row with \( x \):

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

Compute all \( f_c \) for \( c = 1, \ldots, C \)

2. Apply softmax function to get normalized probability:

\[ p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^{C} \exp(f_c)} = \text{softmax}(f_y) \]

3. Choose the \( y \) with maximum probability
Training with softmax and cross-entropy loss

• For each training example \((x, y)\), our objective is to maximize the probability of the correct class \(y\).

• Or we can minimize the negative log probability of that class:

\[
-\log p(y|x) = -\log \left( \frac{\exp(y)}{\sum_{c=1}^{C} \exp(f_c)} \right)
\]
Background: What is “cross entropy” loss/error?

• Concept of “cross entropy” is from information theory
• Let the true probability distribution be $p$
• Let our computed model probability be $q$
• The cross entropy is:

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

• Assuming a ground truth (or true or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: $p = [0,\ldots,0,1,0,\ldots,0]$ then:
• Because of one-hot $p$, the only term left is the negative log probability of the true class
Classification over a full dataset

• Cross entropy loss function over full dataset \( \{x_i, y_i\}_{i=1}^{N} \)

\[
J(\theta) = \frac{1}{N} \sum_{i=1}^{N} - \log \left( \frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_{c}}} \right)
\]

• Instead of

\[
f_y = f_y(x) = W_y \cdot x = \sum_{j=1}^{d} W_{y_j} x_j
\]

We will write \( f \) in matrix notation:

\[
f = Wx
\]
Traditional ML optimization

- For general machine learning $\theta$ usually only consists of columns of $W$:

$$\theta = \begin{bmatrix} W_1 \\ \vdots \\ W_d \end{bmatrix} = W(\cdot) \in \mathbb{R}^{C \times d}$$

- So we only update the decision boundary via

$$\nabla_{\theta} J(\theta) = \begin{bmatrix} \nabla W_1 \\ \vdots \\ \nabla W_d \end{bmatrix} \in \mathbb{R}^{C \times d}$$
Neural Network Classifiers

- Softmax ($\approx$ logistic regression) alone not very powerful
- Softmax gives only linear decision boundaries

This can be quite limiting

→ Unhelpful when a problem is complex

Wouldn’t it be cool to get these correct?
Neural Nets for the Win!

• Neural networks can learn much more complex functions and nonlinear decision boundaries!
  • In the original space
Classification difference with word vectors

- Commonly in NLP deep learning:
  - We learn **both** $W$ and word vectors $x$
  - We learn **both** conventional parameters and representations
  - The word vectors re-represent one-hot vectors—move them around in an intermediate layer vector space—for easy classification with a (linear) softmax classifier via layer $x = Le$

\[
\nabla \theta J(\theta) = \begin{bmatrix}
\nabla W_1 \\
\vdots \\
\nabla W_d \\
\n\nabla x_{aardvark} \\
\vdots \\
\n\nabla x_{zebra}
\end{bmatrix} \in \mathbb{R}^{Cd+Vd}
\]
8. The course
A note on your experience 😊

• This is a hard, advanced, graduate level class
• I and all the TAs really care about your success in this class
• Give Feedback. Take responsibility for holes in your knowledge
• **Come to office hours/help sessions**

“Terrible class”
“Don’t take it”
“Too much work”

“Best class at Stanford”
“Changed my life”
“Obvious that instructors care”
“Learned a ton”
“Hard but worth it”
Office Hours / Help sessions

- **Come to office hours/help sessions!**
  - Come to discuss final project ideas as well as the assignments
  - Try to come early, often and off-cycle
- **Help sessions:** daily, at various times, see calendar
  - Attending in person: Just show up! Our friendly course staff will be on hand to assist you
  - SCPD/remote access: Use queuestatus
- **Chris’s office hours:**
  - Mon 4–6pm. Come along next Monday?