COS 484: Natural Language Processing

Midterm Review

Fall 2019
Anouncements

- **Midterm exam**: this Thursday, Oct 24th 1:30-2:45 (75 minutes)
  - Everyone is seated in COS 104 (alternating seats). Please arrive 10 minutes early!
  - One single-sided cheatsheet is allowed
  - No phone/laptop, no calculator or any internet access
  - If you have an exam (e.g. COS 429) at 3pm, you have an option to start at 1pm.
Announcements

• **Assignment 2** grades were out
  • No separate grades for the code submission

• **Assignment 3** due on Friday 11:59pm
• **Assignment 4** will be out on Friday too
Today’s Plan

• Dependency parsing (10 mins)
• Midterm review (65 minutes)
“Book me the morning flight”

Dependency parsing

<table>
<thead>
<tr>
<th>stack</th>
<th>buffer</th>
<th>action</th>
<th>added arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[ROOT]</td>
<td>[Book, me, the, morning, flight]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>1</td>
<td>[ROOT, Book]</td>
<td>[me, the, morning, flight]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>2</td>
<td>[ROOT, Book, me]</td>
<td>[the, morning, flight]</td>
<td>RIGHT-ARC(iobj)</td>
</tr>
<tr>
<td>3</td>
<td>[ROOT, Book]</td>
<td>[the, morning, flight]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>4</td>
<td>[ROOT, Book, the]</td>
<td>[morning, flight]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>5</td>
<td>[ROOT, Book, the, morning]</td>
<td>[flight]</td>
<td>SHIFT</td>
</tr>
<tr>
<td>6</td>
<td>[ROOT, Book, the, morning, flight]</td>
<td>[]</td>
<td>LEFT-ARC(nmod)</td>
</tr>
<tr>
<td>7</td>
<td>[ROOT, Book, the, flight]</td>
<td>[]</td>
<td>LEFT-ARC(det)</td>
</tr>
<tr>
<td>8</td>
<td>[ROOT, Book, flight]</td>
<td>[]</td>
<td>RIGHT-ARC(dobj)</td>
</tr>
<tr>
<td>9</td>
<td>[ROOT, Book]</td>
<td>[]</td>
<td>RIGHT-ARC(root)</td>
</tr>
<tr>
<td>10</td>
<td>[ROOT]</td>
<td>[]</td>
<td></td>
</tr>
</tbody>
</table>
• Extract features from the configuration
• Use your favorite classifier: logistic regression, SVM...

<table>
<thead>
<tr>
<th>Source</th>
<th>Feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One word</strong></td>
<td>$s_1.w$</td>
</tr>
<tr>
<td></td>
<td>$s_2.w$</td>
</tr>
<tr>
<td></td>
<td>$b_1.w$</td>
</tr>
<tr>
<td><strong>Two word</strong></td>
<td>$s_1.w \circ s_2.w$</td>
</tr>
<tr>
<td></td>
<td>$s_1.t \circ s_2.wt$</td>
</tr>
<tr>
<td></td>
<td>$s_1.w \circ s_1.t \circ s_2.t$</td>
</tr>
</tbody>
</table>

w: word, t: part-of-speech tag

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing
**Feature templates**

\[ s_2 \cdot w \circ s_2 \cdot t \]
\[ s_1 \cdot w \circ s_1 \cdot t \circ b_1 \cdot w \]
\[ lc(s_2) \cdot t \circ s_2 \cdot t \circ s_1 \cdot t \]
\[ lc(s_2) \cdot w \circ lc(s_2) \cdot l \circ s_2 \cdot w \]

**Features**

\[ s_2 \cdot w = \text{has} \circ s_2 \cdot t = \text{VBZ} \]
\[ s_1 \cdot w = \text{good} \circ s_1 \cdot t = \text{JJ} \circ b_1 \cdot w = \text{control} \]
\[ lc(s_2) \cdot t = \text{PRP} \circ s_2 \cdot t = \text{VBZ} \circ s_1 \cdot t = \text{JJ} \]
\[ lc(s_2) \cdot w = \text{He} \circ lc(s_2) \cdot l = \text{nsubj} \circ s_2 \cdot w = \text{has} \]

Usually a combination of 1-3 elements from the configuration

Binary, sparse, millions of features

(Nivre 2008): Algorithms for Deterministic Incremental Dependency Parsing
More feature templates

# From Single Words
pair { stack.tag stack.word }
stack { word tag }
pair { input.tag input.word }
input { word tag }
pair { input(1).tag input(1).word }
input(1) { word tag }
pair { input(2).tag input(2).word }
input(2) { word tag }

# From word pairs
quad { stack.tag stack.word input.tag input.word }
triple { stack.tag stack.word input.tag input.word }
triple { stack.word input.tag input.word }
triple { stack.tag stack.word input.tag }
pair { stack.word input.word }
pair { stack.tag input.tag }
pair { input.tag input(1).tag }

# From word triples
triple { input.tag input(1).tag input(2).tag }
triple { stack.tag input.tag input(1).tag }
triple { stack.head(1).tag stack.tag input.tag }
triple { stack.tag stack.child(-1).tag input.tag }
triple { stack.tag stack.child(1).tag input.tag }
triple { stack.tag input.tag input.child(-1).tag }

# Distance
pair { stack.distance stack.word }
pair { stack.distance stack.tag }
pair { stack.distance input.word }
pair { stack.distance input.tag }
triple { stack.distance stack.word input.word }
triple { stack.distance stack.tag input.tag }

# valency
pair { stack.word stack.valence(-1) }
pair { stack.word stack.valence(1) }
pair { stack.tag stack.valence(-1) }
pair { stack.tag stack.valence(1) }
pair { input.word input.valence(-1) }
pair { input.tag input.valence(-1) }

# unigrams
stack.head(1) {word tag}
stack.label
stack.child(-1) {word tag label}
stack.child(1) {word tag label}
input.child(-1) {word tag label}

# third order
stack.head(1).head(1) {word tag}
stack.head(1).label
stack.child(-1).sibling(1) {word tag label}
stack.child(1).sibling(-1) {word tag label}
input.child(-1).sibling(1) {word tag label}
triple { stack.tag stack.child(-1).tag stack.child(-1).sibling(1) }
triple { stack.tag stack.child(1).tag stack.child(1).sibling(-1) }
triple { stack.tag stack.head(1).tag stack.head(1).head(1).tag }
triple { input.tag input.child(-1).tag input.child(-1).sibling(1) }

# label set
pair { stack.tag stack.child(-1).label }
triple { stack.tag stack.child(-1).label stack.child(-1).sibling(1) }
quad { stack.tag stack.child(-1).label stack.child(-1).sibling(1)
pair { stack.tag stack.child(1).label }
triple { stack.tag stack.child(1).label stack.child(1).sibling(-1) }
quad { stack.tag stack.child(1).label stack.child(1).sibling(-1)
pair { input.tag input.child(-1).label }
triple { input.tag input.child(-1).label input.child(-1).sibling(1) }
quad { input.tag input.child(-1).label input.child(-1).sibling(1) }
}
Parsing with neural networks

(Chen and Manning, 2014): A Fast and Accurate Dependency Parser using Neural Networks
Parsing with neural networks

- Used pre-trained word embeddings
- Part-of-speech tags and dependency labels are also represented as vectors
- Eliminated feature templates!

A simple feedforward NN — what is left is backpropagation!

(Chen and Manning, 2014): A Fast and Accurate Dependency Parser using Neural Networks
Further improvements

- Bigger, deeper networks with better tuned hyperparameters
- Beam search
- Global normalization

<table>
<thead>
<tr>
<th>Method</th>
<th>UAS</th>
<th>LAS (PTB WSJ SD 3.3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen &amp; Manning 2014</td>
<td>92.0</td>
<td>89.7</td>
</tr>
<tr>
<td>Weiss et al. 2015</td>
<td>93.99</td>
<td>92.05</td>
</tr>
<tr>
<td>Andor et al. 2016</td>
<td>94.61</td>
<td>92.79</td>
</tr>
</tbody>
</table>

Google’s SyntaxNet and the Parsey McParseFace (English) model

Announcing SyntaxNet: The World’s Most Accurate Parser
Goes Open Source

Thursday, May 12, 2016
Handling non-projectivity

- The arc-standard algorithm we presented only builds **projective** dependency trees

- Possible directions:
  - Give up!
  - Post-processing
  - Add new transition types (e.g., SWAP)
  - Switch to a different algorithm (e.g., graph-based parsers such as MSTParser)
Language Models

Review
For a sequence of words/tokens $w_1, w_2, \ldots, w_N$, a LM outputs the probability of the sequence $P(w_1, w_2, \ldots, w_N)$

- $P(w_1, w_2, \ldots, w_N) = p(w_1)\ p(w_2|w_1)\ p(w_3|w_1, w_2)\ \times\ \ldots\ \times\ p(w_N|w_1, w_2, \ldots, w_{N-1})$

- Each $P(\ldots)$ is determined by counting:

\[
P(\text{sat} | \text{the cat}) = \frac{\text{count(\text{the cat sat})}}{\text{count(\text{the cat})}}
\]

\[
P(\text{on} | \text{the cat sat}) = \frac{\text{count(\text{the cat sat on})}}{\text{count(\text{the cat sat})}}
\]

This process of computing P’s is called *maximum likelihood estimation* (MLE).

*Estimate* the parameters of the model such that, according to the model, the likelihood of the observed data is *maximized*. 
• To reduce the number of parameters, use only the recent past to predict the next token/word.

• 1st order

\[ P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the}) \]

• 2nd order

\[ P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the}) \]

• Consider only the last \( k \) words for context

\[ P(\text{w}_i \mid \text{w}_1\text{w}_2\ldots\text{w}_{i-1}) \approx P(\text{w}_i \mid \text{w}_{i-k}\ldots\text{w}_{i-1}) \]

• If we had infinite corpus, larger \( n \) \( \rightarrow \) more accurate model
Once you train a LM on a corpus, you need to test it on a different, unseen corpus.

- Note: a separate “dev set” is useful for tuning hyperparameters, such as $\alpha$ for Laplace smoothing.

- **Extrinsic evaluation**: Evaluate LM on a downstream task
  - Machine translation
  - Text classification
  - Sentence similarity

- **Intrinsic evaluation**: Just use a held-out test corpus
• Perplexity is how well a probability distribution or model predicts a sample sequence (lower is better).

• Perplexity is $2^{CE}$. Fundamentally, cross-entropy (CE) measures how deficient/difficult the model is at predicting the corpus.

\[
CE = -\frac{1}{W} \sum_{i=1}^{n} \log_2 P(S_i) \quad \text{where } W \text{ is the num. of words in corpus}
\]

• Remember: perplexity is the inverse probability of the corpus according to the LM, normalized by the number of words.

\[
ppl = 2^{-\frac{1}{W}W \cdot \log(1/|V|)} = |V|
\]

Size of vocab
• Think of perplexity as a weighted average branching factor: Lower perplexity means it’s easier to predict the next word, in the corpus that’s being evaluated.

<table>
<thead>
<tr>
<th>Example: Rolling a 6-sided dice (die)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Train &amp; test with perfectly-fair die</td>
</tr>
<tr>
<td>Train &amp; test with same loaded die</td>
</tr>
<tr>
<td>Train &amp; test with differently-loaded die</td>
</tr>
</tbody>
</table>

• Low perplexity does not guarantee good extrinsic results!
• Many n-grams can appear in the test corpus but not training

• A few very frequent words; a long tail of infrequent words

• Need some way to remove zero probabilities
**Add-alpha (Laplace) smoothing**  
Add a small amount to all probabilities

\[
P(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + \alpha}{C(w_{i-1}) + \alpha |V|}
\]

**Linear interpolation**  
Use a combination of different granularities of n-grams

\[
\hat{P}(w_i | w_{i-1}, w_{i-2}) = \lambda_1 P(w_i | w_{i-1}, w_{i-2}) + \lambda_2 P(w_i | w_{i-1}) + \lambda_3 P(w_i)
\]

where \( \sum_i \lambda_i = 1 \)

**Average count**  
Like simple interpolation, but with more specific lambdas

\[
P_{\text{interp}}(w_i | w_{i-n+1}) = \lambda_{w_{i-n+1}} P_{\text{ML}}(w_i | w_{i-n+1}) + (1 - \lambda_{w_{i-n+1}}) P_{\text{interp}}(w_i | w_{i-n+2})
\]

where \( \lambda_{w_{i-n+1}} \) is based on \( \frac{c(w_{i-n+1})}{|w_i : c(w_{i-n+1}) > 0|} \)

"The less sparse the data the larger lambda should be. The more accurate counts we have, the more trustworthy the n-gram is, and the higher we can make lambda."

**Absolute discounting**  
Redistribute probability mass from observed n-grams to unobserved ones

\[
P_{\text{abs.discount}}(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \alpha(w_{i-1}) P(w)
\]

where \( \alpha(w_{i-1}) = 1 - \sum_w \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})} \)

**Back-off**  
Use lower order n-grams if higher order ones are too sparse

\[
P_{\text{bo}}(w_i | w_{i-n+1} \cdots w_{i-1}) = \begin{cases} 
\frac{C(w_{i-n+1} \cdots w_i)}{C(w_{i-n+1} \cdots w_{i-1})} & \text{if } C(w_{i-n+1} \cdots w_i) > k \\
\alpha_{w_{i-n+1} \cdots w_{i-1}} P_{\text{bo}}(w_i | w_{i-n+2} \cdots w_{i-1}) & \text{otherwise}
\end{cases}
\]

Katz
Focus(1):

• Naive Bayes
• Logistic Regression
  • Training
• Difference between discriminative and generative models
• Evaluation metrics
  • Precision
  • Recall
  • F Score
Naive Bayes

• Example of email classification:
  • Given documents and classes, how to classify another document?

• Formula snippet:

  - Bayes Rule:
    \[ P(c \mid d) = \frac{P(c) \cdot P(d \mid c)}{P(d)} \]

  - Option 1: represent the entire sequence of words
    • \( P(w_1, w_2, w_3, \ldots, w_k \mid c) \) \hspace{1cm} (too many sequences!)

  - Option 2: Bag of words
    • Assume position of each word is irrelevant
      (both absolute and relative)
    • \( P(w_1, w_2, w_3, \ldots, w_k \mid c) = P(w_1 \mid c) \cdot P(w_2 | c) \ldots P(w_k \mid c) \)

• Makes strong (naive) independence assumptions

• Probability of each word is *conditionally independent* given class \( c \)
Data Sparsity

Maximum likelihood estimates:

\[ \hat{P}(c_j) = \frac{\text{count}(\text{class} = c_j)}{\sum_c \text{count}(\text{class} = c)} \]

\[ \hat{P}(w_i|c_j) = \frac{\text{count}(w_i, c_j)}{\sum_w \text{count}(w, c_j)} \]

- Laplace smoothing:

\[ \hat{P}(w_i|c) = \frac{\text{count}(w_i, c) + \alpha}{\left[ \sum_w \text{count}(w, c) \right] + \alpha|V|} \]
Input: Set of annotated documents \( \{(d_i, c_i)\}_{i=1}^{n} \)

A. Compute vocabulary \( V \) of all words

B. Calculate
\[
\hat{P}(c_j) = \frac{\text{Count}(c_j)}{\text{Total \# docs}}
\]

C. Calculate
\[
\hat{P}(w_i | c_j) = \frac{\text{Count}(w_i, c_j) + \alpha}{\sum_{w \in V} [\text{Count}(w, c_j) + \alpha]}
\]

D. (Prediction) Given document \( d = (w_1, w_2, \ldots, w_k) \)
\[
C_{\text{MAP}} = \arg\max_{c} \hat{P}(c) \cdot \prod_{i=1}^{K} \hat{P}(w_i | c)
\]
## Evaluation Metrics

<table>
<thead>
<tr>
<th>Truth</th>
<th>Predicted</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>100</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>45</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>

- True positive: Predicted + and actual +
- True negative: Predicted - and actual -
- False positive: Predicted + and actual -
- False negative: Predicted - and actual +

**Precision** (++) = \( \frac{TP}{TP + FP} \)

**Recall** (++) = \( \frac{TP}{TP + FN} \)

\[
F_\beta = \frac{(1 + \beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}
\]
Logistic Regression

Using Logistic Regression

- Inputs:
  1. Classification instance in a [feature representation] \([x_1, x_2, \ldots, x_d]\)
  2. Classification function to compute \(\hat{y}\) using \(P(\hat{y} | x)\)
  3. Loss function (for learning)
  4. Optimization algorithm

- Train phase: Learn the parameters of the model to minimize loss function
- Test phase: Apply parameters to predict class given a new input \(x\)
Feature Representation

**Sample feature vector**

It's a(n) [key] time there are virtually no surprises, and the writing is [second-rate]. So why was it so [enjoyable]? For one thing, the cast is [great]. Another nice touch is the music I was overcome with the urge to get off the couch and start dancing. It sucked me in, and I'll do the same to you.

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon) $\in$ doc</td>
<td>3</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon) $\in$ doc</td>
<td>2</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\begin{cases} 1 &amp; \text{if } \text{“no” } \in \text{doc} \ 0 &amp; \text{otherwise}\end{cases}$</td>
<td>1</td>
</tr>
<tr>
<td>$x_4$</td>
<td>count(1st and 2nd pronouns) $\in$ doc</td>
<td>3</td>
</tr>
<tr>
<td>$x_5$</td>
<td>$\begin{cases} 1 &amp; \text{if } \text{“!” } \in \text{doc} \ 0 &amp; \text{otherwise}\end{cases}$</td>
<td>0</td>
</tr>
<tr>
<td>$x_6$</td>
<td>log(word count of doc)</td>
<td>$\ln(64) = 4.15$</td>
</tr>
</tbody>
</table>
Classification Function

Given \( x \), compute \( z = w \cdot x + b \)

Compute probabilities: \( P(y = 1 \mid x) = \frac{1}{1 + e^{-z}} \)

\[
P(y = 1) = \sigma(w \cdot x + b) = \frac{1}{1 + e^{-(w \cdot x + b)}}
\]

\[
P(y = 0) = 1 - \sigma(w \cdot x + b) = 1 - \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{e^{-(w \cdot x + b)}}{1 + e^{-(w \cdot x + b)}}
\]

Decision boundary: \( \hat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid x) > 0.5 \\ 0 & \text{otherwise} \end{cases} \)
Loss Function – Assigning ‘w’ and ‘b’?

**Goal:** predicted label $\hat{y}$ as close as possible to actual label $y$

**Properties of CE Loss**

$$L_{CE} = - \sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)})]$$

- Ranges from 0 (perfect predictions) to $\infty$
- Lower the value, better the classifier
- Cross-entropy between the true distribution $P(y | x)$ and predicted distribution $P(\hat{y} | x)$
Optimisation

Gradient for logistic regression

\[ L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log(\sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)})\log(1 - \sigma(w \cdot x^{(i)} + b)))] \]

Gradient, \( \frac{dL_{CE}(w, b)}{dw_j} = \sum_{i=1}^{n} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}]x_j^{(i)} \)

\( \frac{dL_{CE}(w, b)}{db} = \sum_{i=1}^{n} [\sigma(w \cdot x^{(i)} + b) - y^{(i)}] \)

\[ \theta = [w; b] \]

\[ \hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L_{CE}(y^{(i)}, x^{(i)}; \theta) \]

\[ \theta^{i+1} = \theta^i - \eta \frac{d}{d\theta} f(x; \theta) \]
function STOCHASTIC GRADIENT DESCENT($L(), f(), x, y$) returns $\theta$

# where: $L$ is the loss function
# $f$ is a function parameterized by $\theta$
# $x$ is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
# $y$ is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$

repeat til done  # see caption
For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

1. Optional (for reporting):  # How are we doing on this tuple?
   Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$  # What is our estimated output $\hat{y}$?

2. $g \leftarrow \nabla_\theta L(f(x^{(i)}; \theta), y^{(i)})$  # How far off is $\hat{y}^{(i)}$ from the true output $y^{(i)}$?
3. $\theta \leftarrow \theta - \eta g$  # How should we move $\theta$ to maximize loss?

return $\theta$
Regularisation

• Prevents Overfitting:

Training objective: $\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)})$

Regularization helps prevent overfitting

$\hat{\theta} = \arg \max_{\theta} \sum_{i=1}^{n} \log P(y^{(i)} | x^{(i)}) - \alpha R(\theta)$
Word embeddings

COS484, Midterm Review
• Distributional hypothesis:
  • “words that occur in similar contexts tend to have similar meanings”

• Word embedding:
  • A vector that captures the meaning of a word.
  • Can be sparse (word-word occurrence) or dense.

  Goal: represent words as short (50-300 dimensional) & dense (real-valued) vectors.

employees =

\[
\begin{pmatrix}
0.286 \\
0.792 \\
-0.177 \\
-0.107 \\
10.109 \\
-0.542 \\
0.349 \\
0.271 \\
0.487
\end{pmatrix}
\]
Word embeddings

- **Word2Vec**
  - **Skip-gram**: given a *target word*, predict the *context words* in a fixed window of size $m$.

I understand the word embedding now.

**context words** $\quad$ **target word** $\quad$ **context words**

$(m = 2)$ $\quad$ $m = 2$
Word embeddings

- **Word2Vec**
  - **Objective function:** Average Negative Log Likelihood (NLL)

\[
J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m \atop j \neq t} \log \mathbb{P}(w_{t+j} | w_t; \theta)
\]

given a sentence of length \( T \)

\( w_1, \ldots, w_{t-2}, w_{t-1}, w_t, w_{t+1}, w_{t+2}, \ldots, w_T \)
Word embeddings

• **Word2Vec**

• **Objective function:** Average Negative Log Likelihood (NLL)

\[
J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq t} \log P(w_{t+j} | w_t; \theta)
\]

• **Similarity based softmax** ($V$ is vocabulary):

\[
\theta = \{ \{ u_w \in \mathbb{R}^d \}_{w \in V}, \{ v_w \in \mathbb{R}^d \}_{w \in V} \}
\]

embeddings of target words

embeddings of context words

\[
P(w_{t+j} | w_t; \theta) = \frac{\exp(u_{w_t} \cdot v_{w_{t+j}})}{\sum_{w \in V} \exp(u_{w_t} \cdot v_w)}
\]
Word embeddings

- Word2Vec

- Gradient of objective function:

$$
\nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m \leq k \leq m} \nabla_\theta \left( -\log(P(w_{t+j}|w_t; \theta)) \right)
$$

$$
\Rightarrow \begin{cases} 
\frac{\partial y}{\partial u_{w_t}} &= -v_{w_c} + \sum_{w \in V} P(w|w_t)v_w \\
\frac{\partial y}{\partial v_w} &= -1(w = w_c)u_{w_t} + P(w|w_t)u_{w_t}
\end{cases}
$$
Word embeddings

\[ y = - \log \left( \frac{\exp(u_{w_t} \cdot v_{w_c})}{\sum_{w \in V} \exp(u_{w_t} \cdot v_w)} \right) \]

- Word2Vec

- Gradient of objective function:

\[ \nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m \leq k \leq m} \nabla_\theta - \log(\mathbb{P}(w_{t+j} | w_t; \theta)) \]

\[ \Rightarrow \begin{cases} \frac{\partial y}{\partial u_{w_t}} = -v_{w_c} + \sum_{w \in V} \mathbb{P}(w | w_t) v_w \\ \frac{\partial y}{\partial v_w} = -1(w = w_c) u_{w_t} + \mathbb{P}(w | w_t) u_{w_t} \end{cases} \]

- Full softmax is computationally intractable
Word embeddings

• **Word2Vec**

• **Gradient of objective function:**

\[
\nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m \leq k \leq m} \nabla_\theta - \log(\mathbb{P}_{\text{NS}}(w_{t+j}|wt; \theta))
\]

\[
\begin{aligned}
\frac{\partial y}{\partial u_{wt}} &= -\sigma(-u_{wt} \cdot v_{wc})v_{wc} + \sum_{k=1}^{K} \sigma(u_{wt} \cdot v_{wk})v_{wk} \\
\frac{\partial y}{\partial v_{w}} &= -1_{w=w_c} \sigma(-u_{wt} \cdot v_{wc})u_{wt} + \sum_{k=1}^{K} \frac{\partial v_{wk}}{\partial v_{w}} \sigma(u_{wt} \cdot v_{w})u_{wt}
\end{aligned}
\]

• **Negative Sampling:** Randomly sample \( K \) (5-20) negative examples.

\[
y = - \log \left( \frac{\sigma(u_{wt} \cdot v_{wc})}{\prod_{k=1}^{K} \sigma(-u_{wt} \cdot v_{wk})} \right)
\]
Word embeddings

\[ y = -\log \left( \frac{\sigma(u_{w_t} \cdot v_{w_c})}{\prod_{k=1}^{K} \sigma(-u_{w_t} \cdot v_{w_k})} \right) \]

- **Word2Vec**

- **Gradient of objective function:**

  \[
  \nabla_\theta J(\theta) = \sum_{t=1}^{T} \sum_{-m \leq k \leq m} \nabla_\theta - \log(\mathbb{P}_{NS}(w_{t+j} | w_t; \theta))
  \]

  \[
  \begin{cases}
  \frac{\partial y}{\partial u_{w_t}} = -\sigma(-u_{w_t} \cdot v_{w_c})v_{w_c} + \sum_{k=1}^{K} \sigma(u_{w_t} \cdot v_{w_k})v_{w_k} \\
  \frac{\partial y}{\partial v_{w}} = -1_{w=w_c} \sigma(-u_{w_t} \cdot v_{w_c})u_{w_t} + \sum_{k=1}^{K} \frac{\partial v_{w_k}}{\partial v_{w}} \sigma(u_{w_t} \cdot v_{w})u_{w_t}
  \end{cases}
  \]

- **Update** for sampled target-context word pairs \((w_t, w_c)\):

  \[
  \begin{cases}
  u_{w_t} \leftarrow u_{w_t} - \eta \frac{\partial y}{\partial u_{w_t}} \\
  v_{w_c} \leftarrow v_{w_c} - \eta \frac{\partial y}{\partial v_{w_c}}, \quad v_{w_k} \leftarrow v_{w_k} - \eta \frac{\partial y}{\partial v_{w_k}}
  \end{cases}
  \]
Word embeddings

• Evaluation
  • Intrinsic
    • Evaluate on an intermediate subtask (e.g. word similarity)
    • Fast to compute
    • Not clear if it really helps the downstream task
  • Extrinsic
    • Use word embeddings in downstream tasks and measure the performance improvement
    • Time-costly but still the most important evaluation metric
Neural Networks

COS484, Midterm Review
Neural Networks

- The activity of neuron $i$:

\[ x_i \leftarrow f \left( \sum_j W_{ij} x_j + b_i \right) \]

- activity of presynaptic neuron $j$
- synaptic weight from neuron $j$ to neuron $i$
- bias of neuron $i$
- activation function
- activity of postsynaptic neuron $i$

- repeat for other neurons **in some order** to be specified
Neural Networks

- Useful activation functions and their derivatives

**sigmoid** $(0,1)$

$$f(z) = \frac{1}{1 + \exp(-z)}$$

**tanh** $(-1,1)$

$$f(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1}$$

**ReLU** $[0, +\infty)$

$$f(z) = \max(0, z)$$

$$f'(z) = f(z)(1 - f(z))$$

$$f'(z) = 1 - f^2(z)$$

$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$
Neural Networks

- L-Multilayer Formulation (using matrix-vector notation)

For layer $\ell = 0$ to $L-1$:

$$ x^{\ell} \leftarrow f \left( W^{\ell} x^{\ell-1} + b^{\ell} \right) $$

neural activity of previous layer

weight matrix

bias vector

Forward Pass

$$ x^{0} \xrightarrow{W^1, b^1} x^{1} \xrightarrow{W^2, b^2} \ldots \xrightarrow{W^L, b^L} x^{L} $$
Neural Networks

- **Loss function** (error between output and ground truth)
  - Logistic Regression / Classification - Cross Entropy (CE)
  - Linear Regression - Mean Squared Error (MSE)
  - ....

\[
\begin{align*}
\mathbf{x}_0 & \xrightarrow{W^1, b^1} \mathbf{x}_1 & \xrightarrow{W^2, b^2} & \cdots & \xrightarrow{W^L, b^L} & \mathbf{x}^L \\
\end{align*}
\]

Forward Pass

- Loss Function

\[
\mathcal{L}(\mathbf{x}^L, y^*)
\]

- Update weights and biases using gradient descent (Backprop)

\[
\theta := \{ \{ W^\ell \}, \{ b^\ell \} \} \quad \theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)
\]
Neural Networks

- **Backpropagation** (using chain rule of derivative)

\[
x^0 \xrightarrow{W^1, b^1} x^1 \xrightarrow{W^2, b^2} \ldots \xrightarrow{W^L, b^L} x^L \rightarrow \mathcal{L}(x^L, y^*)
\]

**Forward Pass**
\[
x^\ell \leftarrow f(W^\ell x^{\ell-1} + b^\ell)
\]

Define dual variables to maintain negative gradients
\[
\rho^L \leftarrow -\frac{\partial \mathcal{L}}{\partial x^L} \circ f'(f^{-1}(x^L))
\]

\[
\rho^0 \xleftarrow{W^1, b^1} \rho^1 \xleftarrow{W^2, b^2} \ldots \xleftarrow{W^L, b^L} \rho^L \xleftarrow{\mathcal{L}(x^L, y^*)}
\]

**Backward Pass**
\[
\rho^{\ell-1} \leftarrow f'(f^{-1}(x^{\ell-1})) \circ (W^\ell)^\top \rho^\ell
\]
Neural Networks

- **Backpropagation** (using chain rule of derivative)

  Define dual variables to maintain negative gradients
  \[
  \rho^L \leftarrow - \frac{\partial \mathcal{L}}{\partial \mathbf{x}^L} \circ f'(f^{-1}(\mathbf{x}^L))
  \]

  \[
  \rho^0 \leftrightarrow W^1, b^1 \quad \rho^1 \leftrightarrow W^2, b^2 \quad \ldots \quad \rho^L \leftrightarrow W^L, b^L \quad \mathcal{L}(\mathbf{x}^L, y^*)
  \]

  **Backward Pass**
  \[
  \rho^{\ell-1} \leftarrow f'(f^{-1}(\mathbf{x}^{\ell-1})) \circ (W^\ell)^\top \rho^\ell
  \]

- **Update weights and biases**

  \[
  \Delta W^\ell \propto \rho^\ell (\mathbf{x}^{\ell-1})^\top \\
  \Delta b^\ell \propto \rho^\ell
  \]
Neural Networks

- **Feedforward Language Model**
  - **N-gram models:** $P(\text{mat} \mid \text{the cat sat on the})$
  - **Input:** concatenation of previous words (with fixed context size)
  - **Hidden Layer:** fully connected, use tanh activation
  - **Output:** softmax over vocabulary
Neural Networks

• Recurrent neural networks
  • Simple RNN (Key: weight sharing)

\[ h_t = f(W h_{t-1} + U x_t + b) \]

previous hidden state \quad current input

• Recurrent Neural Language Models
Neural Networks

- Recurrent neural networks

  - Simple RNN (Key: weight sharing)

- Recurrent Neural Language Models
Neural Networks

- Recurrent neural networks
- Backprop through time (expensive for long sequences)

\[ h_1 = g(Wh_0 + Ux_1 + b) \]
\[ h_2 = g(Wh_1 + Ux_2 + b) \]
\[ h_3 = g(Wh_2 + Ux_3 + b) \]
\[ L_3 = -\log \hat{y}_3(w_4) \]

You should know how to compute: \( \frac{\partial L_3}{\partial h_3} \)

\[
\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial h_3} \frac{\partial h_3}{\partial W} + \frac{\partial L_3}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial W} + \frac{\partial L_3}{\partial h_3} \frac{\partial h_3}{\partial h_1} \frac{\partial h_1}{\partial W}
\]

\[
\frac{\partial L}{\partial W} = -\frac{1}{n} \sum_{t=1}^{n} \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \left( \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial W}
\]
Neural Networks

- Recurrent neural networks
  - Backprop through time (expensive for long sequences)
  - Run forward and backward through chunks of the sequence
  - Only backpropagate for some smaller number of steps
Neural Networks

- **Long Short-term Memory (LSTM)**

  cell state vector
  \[
  c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_c x_t + U_c h_{t-1} + b_c)
  \]

  forget gate
  \[
  f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)
  \]

  input gate
  \[
  i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)
  \]

  output gate
  \[
  o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)
  \]

  hidden state vector/output vector
  \[
  h_t = o_t \circ \tanh(c_t)
  \]
Focus(2):

- HMM
- MEMM
- 3 Problems in HMM
  - Decoding
  - Observation sequence
  - Training
A Markov chain is useful when we need to compute a probability for a sequence of observable events. In many cases, however, the events we are interested in are hidden: we don’t observe them directly. For example we don’t normally observe part-of-speech tags in a text. Rather, we see words, and must infer the tags from the word sequence. We call the tags hidden because they are not observed.

**Problem 1 (Likelihood):** Given an HMM $\lambda = (A, B)$ and an observation sequence $O$, determine the likelihood $P(O|\lambda)$.

**Problem 2 (Decoding):** Given an observation sequence $O$ and an HMM $\lambda = (A, B)$, discover the best hidden state sequence $Q$.

**Problem 3 (Learning):** Given an observation sequence $O$ and the set of states in the HMM, learn the HMM parameters $A$ and $B$.

1. Markov assumption:

   \[ P(s_{t+1} | s_1, \ldots, s_t) = P(s_{t+1} | s_t) \]

2. Output independence:

   \[ P(o_t | s_1, \ldots, s_t) = P(o_t | s_t) \]
Problem 2 – Decoding

\[ \hat{S} = \operatorname{arg\,max}_S p(s|o) = \operatorname{arg\,max}_S \frac{p(s) p(o|s)}{p(o)} \quad \text{[Bayes]} \]

\[ = \operatorname{arg\,max}_S p(s) p(o|s) \]

Greedy Decoding:

\[ \forall t, \hat{S}_{t+1} = \operatorname{arg\,max}_S p(s_t \mid \hat{S}_t) p(o_{t+1} \mid s_t) \]
Viterbi:

$M[i, j]$: Most probable sequence of states ending with state $j$ at time $i$

$M[i, j] = \max_k M[i-1, k] P(s_j | s_k) P(o_i | s_j)$  \hspace{1cm} 1 \leq k \leq K  \hspace{1cm} 1 \leq i \leq n$

**Backward**: Pick $\max_k M[n, k]$ and backtrack
MEMM

In general, we can use all observations and all previous states:

\[
\hat{S} = \arg \max_S P(S \mid O) = \arg \max_S \prod_i P(s_i \mid o_n, o_{i-1}, \ldots, o_1, s_{i-1}, \ldots, s_1)
\]
Problem 1: Prob. of Observation sequence

• Forward and backward probabilities:

Define:

\[ \alpha_s(j) = P(x_1, \ldots, x_{j-1}, y_j = s \mid \theta, \phi) \] (forward probability)

\[ \beta_s(j) = P(x_j, \ldots, x_m \mid y_j = s, \theta, \phi) \] (backward probability)

Observation likelihood,

\[ Z = P(x_1, x_2, \ldots, x_m \mid \theta, \phi) = \sum_s \alpha_s(j) \beta_s(j) \text{ for any } j \in 1, \ldots, m \]
Problem 3: HMM Training

• Intuitive Idea of EM:
  
  - $\theta^t$ is the parameter vector at the $t^{th}$ iteration
  - Choose $\theta^0$ at random (or using smart heuristics)
  - (E step): Compute expected counts
    \[
    \overline{\text{Count}}(r) = \sum_{i=1}^{n} \sum_{y} P(y|x_i, \theta^{t-1}) \text{ Count}(x_i, y, r)
    \]
    for every parameter $\theta_r$
  - e.g.
    \[
    \overline{\text{Count}}(DT \rightarrow NN) = \sum_{i} \sum_{y} P(S|O_i, \theta^{t-1}) \text{ Count}(O_i, S, \theta_{DT\rightarrow NN})
    \]
  - (M step): Re-estimate parameters using expected counts to maximize likelihood
    e.g. $\theta_{DT\rightarrow NN} = \frac{\overline{\text{Count}}(DT \rightarrow NN)}{\sum_{\beta} \overline{\text{Count}}(DT \rightarrow \beta)}$
Continued: HMM Training

• Forward – backward algorithm:

\[
\overline{\text{Count}}(\theta_k) = \sum_{i=1}^{n} \sum_{Y} P(Y|X_i, \theta^{i-1}, \phi^{i-1}) \text{Count}(X_i, Y, \theta_k)
\]

\[
= \sum_{i=1}^{n} \sum_{Y} P(Y|X_i, \theta^{i-1}, \phi^{i-1}) \text{Count}(Y, \theta_k)
\]

(E-Step)

\[
\overline{\text{Count}}(\phi_k) = \sum_{i=1}^{n} \sum_{Y} P(Y|X_i, \theta^{i-1}, \phi^{i-1}) \text{Count}(X_i, Y, \phi_k)
\]

(M-Step)

\[
\theta_k = \frac{\overline{\text{Count}}(\theta_k)}{\sum_{\theta \in M(\theta_k)} \overline{\text{Count}}(\theta')} \quad \text{where } M(\theta_k) \text{ is the set of all transitions}
\]

\[
(a \to b, \text{ all } b) \text{ that share the same previous state as the } k^{th} \text{ transition}
\]

\[
(a \to c \text{ for some } c).
\]

\[
\phi_k' = \frac{\overline{\text{Count}}(\phi_k)}{\sum_{\phi \in M'(\phi_k)} \overline{\text{Count}}(\phi')} \quad \text{where } M'(\phi_k) \text{ is the set of all}
\]

\[
\text{emissions } (a \to x, \text{ all } x) \text{ that share the same hidden state as the } k^{th}
\]

\[
\text{emission } (a \to x', \text{ for some } x').
\]
Continued

\[ P(y_j = s \mid X, \theta, \phi) = \frac{\alpha_s(j)\beta_s(j)}{Z} \]

\[ P(y_j = s, y_{j+1} = s' \mid X, \theta, \phi) = \frac{\alpha_s(j) \theta_{s \rightarrow s'} \phi_{s \rightarrow x_j} \beta_{s'}(j + 1)}{Z} \]

Given these, we can now estimate:

\[ \overline{\text{Count}}(\theta_{s \rightarrow s'}) = \sum_{i} \sum_{j=1}^{m} P(y_j = s, y_{j+1} = s' \mid X_i, \theta, \phi) \]

\[ \overline{\text{Count}}(\phi_{s \rightarrow o}) = \sum_{i} \sum_{j: X_{ij} = o} P(y_j = s \mid X_i, \theta, \phi) \]