Language Modeling
+
Feed-Forward Networks 3

CS 287
Review: LM ML Setup

Multi-class prediction problem,

\[(x_1, y_1), \ldots, (x_n, y_n)\]

- \(y_i\); the one-hot next word
- \(x_i\); representation of the prefix \((w_1, \ldots, w_{t-1})\)

Challenges:

- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)
Review: Perplexity

Previously, used \textit{accuracy} as a metric.

Language modeling uses of version average negative log-likelihood

\begin{itemize}
\item For test data $\tilde{w}_1, \ldots, \tilde{w}_n$
\item
\end{itemize}

\[
NLL = -\frac{1}{n} \sum_{i=1}^{n} \log p(w_i|w_1, \ldots, w_{i-1})
\]

Actually report \textit{perplexity},

\[
perp = \exp\left(-\frac{1}{n} \sum_{i=1}^{n} \log p(w_i|w_1, \ldots, w_{i-1})\right)
\]

Requires modeling full distribution as opposed to argmax (hinge-loss)
Review: Interpolation (Jelinek-M Mercer Smoothing)

Can write recursively,

\[ p_{\text{interp}}(w|c) = \lambda p_{\text{ML}}(w|c) + (1 - \lambda)p_{\text{interp}}(w|c') \]

Ensure that \( \lambda \) form convex combination

\[ 0 \leq \lambda \leq 1 \]

How do you learn conjunction combinations?
Quiz

Assume we have seen the following training sentences,

- a tractor drove slow
- the red tractor drove fast
- the parrot flew fast
- the parrot flew slow
- the tractor slowed down

Compute $p_{ML}$ for bigrams and use them to estimate whether *parrot* or *tractor* fit better in the following contexts.

1. the red ___ ?
2. the ___ ?
3. the ___ drove?
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>tractor</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>red</td>
<td>(\frac{1}{4})</td>
</tr>
<tr>
<td>the</td>
<td>parrot</td>
<td>(\frac{1}{2})</td>
</tr>
<tr>
<td>the</td>
<td>tractor</td>
<td>(\frac{1}{4})</td>
</tr>
<tr>
<td>red</td>
<td>tractor</td>
<td>1</td>
</tr>
<tr>
<td>tractor</td>
<td>drove</td>
<td>(\frac{2}{3})</td>
</tr>
<tr>
<td>tractor</td>
<td>slowed</td>
<td>(\frac{1}{3})</td>
</tr>
<tr>
<td>parrot</td>
<td>flew</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Answer II

- the red tractor
- the parrot
- the tractor drove
Today’s Class

\[ p(w_i|w_{i-n+1}, \ldots w_{i-1}; \theta) \]

- Estimate this directly as a neural network.
- Two types of models, neural network and log-bilinear.
- Efficient methods for approximated estimation.
Intuition: NGram Issues

In training we might see,

the arizona corporations commission authorized

But at test we see,

the colorado businesses organization ___

- Does this training example help here?
  - Not really. No count overlap.

- Does backoff help here?
  - Maybe, if we have seen organization.
  - Mostly get nothing from the earlier words.
Intuition: NGram Issues

In training we might see,

the arizona corporations commission authorized

But at test we see,

the colorado businesses organization ...

- Does this training example help here?
  - Not really. No count overlap.

- Does backoff help here?
  - Maybe, if we have seen organization.
  - Mostly get nothing from the earlier words.
Intuition: NGram Issues

In training we might see,

the arizona corporations commission authorized

But at test we see,

the colorado businesses organization ___

- Does this training example help here?
  - Not really. No count overlap.

- Does backoff help here?
  - Maybe, if we have seen organization.
  - Mostly get nothing from the earlier words.
Goal

- Learn representations that share properties between similar words.
- Particularly helpful for unseen contexts.
- Not a silver bullet, e.g. proper nouns

  the eagles play the arizona **diamondbacks**

  Whereas at test we might see,

  the eagles play the colorado ___

(We will discuss this issue more for in MT)
Baseline: Class-Based Language Models

- Groups words into classes based on word-context.

```
...  3  ...  5

  dog  cat  horse  ...  car  truck  motorcycle  ...
```

- Various factorization methods for estimating with count-based approaches.

- However, assumes a hard-clustering, often estimated separately.
Contents

Neural Language Models

Noise Contrastive Estimation
Recall: Word Embeddings

- Embeddings give multi-dimensional representation of words.

- Ex: Closest by cosine similarity

<table>
<thead>
<tr>
<th>State</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>texas</td>
<td>0.932968706025</td>
</tr>
<tr>
<td>florida</td>
<td>0.932696958878</td>
</tr>
<tr>
<td>kansas</td>
<td>0.914805968271</td>
</tr>
<tr>
<td>colorado</td>
<td>0.904197441085</td>
</tr>
<tr>
<td>arizona</td>
<td></td>
</tr>
<tr>
<td>minnesota</td>
<td>0.863925347525</td>
</tr>
<tr>
<td>carolina</td>
<td>0.862697751337</td>
</tr>
<tr>
<td>utah</td>
<td>0.861915722889</td>
</tr>
<tr>
<td>miami</td>
<td>0.842350326527</td>
</tr>
<tr>
<td>oregon</td>
<td>0.842065064748</td>
</tr>
</tbody>
</table>

- Gives a multi-clustering over words.
Feed-Forward Neural NNLM (Bengio, 2003)

- $w_{i-n+1}, \ldots w_{i-1}$ are input embedding representations
- $w_i$ is an output embedded representation
- Model simultaneously learns,
  - input word representations
  - output word representations
  - conjunctions of input words (through NLM, no n-gram features)
Feed-Forward Neural Representation

- \( p(w_i|w_{i-n+1}, \ldots w_{i-1}; \theta) \)
- \( f_1, \ldots, f_{d_{\text{win}}} \) are words in window
- Input representation is the concatenation of embeddings

\[
x = [v(f_1) \ v(f_2) \ \ldots \ v(f_{d_{\text{win}}})]
\]

Example: NNLM \( (d_{\text{win}} = 5) \)

\[
\begin{bmatrix}
w_3 & w_4 & w_5 & w_6 & w_7
\end{bmatrix} \ w_8
\]

\[
x = [v(w_3) \ v(w_4) \ v(w_5) \ v(w_6) \ v(w_7)]
\]
A Neural Probabilistic Language Model (Bengio, 2003)

One hidden layer multi-layer perceptron architecture,

\[ NN_{MLP1}(x) = \tanh(xW^1 + b^1)W^2 + b^2 \]

Neural network architecture on top of concat.

\[ \hat{y} = \text{softmax}(NN_{MLP1}(x)) \]

Best model uses \( d_{in} = 30 \times d_{win}, d_{hid} = 100. \)
A Neural Probabilistic Language Model

Optional, direct connection layers,

\[ \text{NN}_{DMLP1} (x) = [\tanh(xW^1 + b^1), x]W^2 + b^2 \]

- \( W^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}} \), \( b^1 \in \mathbb{R}^{1 \times d_{\text{hid}}} \); first affine transformation
- \( W^2 \in \mathbb{R}^{(d_{\text{hid}} + d_{\text{in}}) \times d_{\text{out}}} \), \( b^2 \in \mathbb{R}^{1 \times d_{\text{out}}} \); second affine transformation
A Neural Probabilistic Language Model (Bengio, 2003)

\[ i\text{-th output} = P(w_i = i \mid \text{context}) \]

Dashed-lines show the optional direct connections, \( C = \nu \).
## A Neural Probabilistic Language Model

<table>
<thead>
<tr>
<th>Model</th>
<th>n</th>
<th>c</th>
<th>h</th>
<th>m</th>
<th>Direct</th>
<th>Mix</th>
<th>Train</th>
<th>Valid</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP1</td>
<td>5</td>
<td></td>
<td>50</td>
<td>60</td>
<td>yes</td>
<td>no</td>
<td>182</td>
<td>284</td>
<td>268</td>
</tr>
<tr>
<td>MLP2</td>
<td>5</td>
<td></td>
<td>50</td>
<td>60</td>
<td>yes</td>
<td>yes</td>
<td>275</td>
<td>257</td>
<td></td>
</tr>
<tr>
<td>MLP3</td>
<td>5</td>
<td></td>
<td>0</td>
<td>60</td>
<td>yes</td>
<td>no</td>
<td>201</td>
<td>327</td>
<td>310</td>
</tr>
<tr>
<td>MLP4</td>
<td>5</td>
<td></td>
<td>0</td>
<td>60</td>
<td>yes</td>
<td>yes</td>
<td>286</td>
<td>272</td>
<td></td>
</tr>
<tr>
<td>MLP5</td>
<td>5</td>
<td></td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td>209</td>
<td>296</td>
<td>279</td>
</tr>
<tr>
<td>MLP6</td>
<td>5</td>
<td></td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>yes</td>
<td>273</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>MLP7</td>
<td>3</td>
<td></td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>no</td>
<td>210</td>
<td>309</td>
<td>293</td>
</tr>
<tr>
<td>MLP8</td>
<td>3</td>
<td></td>
<td>50</td>
<td>30</td>
<td>yes</td>
<td>yes</td>
<td>284</td>
<td>270</td>
<td></td>
</tr>
<tr>
<td>MLP9</td>
<td>5</td>
<td></td>
<td>100</td>
<td>30</td>
<td>no</td>
<td>no</td>
<td>175</td>
<td>280</td>
<td>276</td>
</tr>
<tr>
<td>MLP10</td>
<td>5</td>
<td></td>
<td>100</td>
<td>30</td>
<td>no</td>
<td>yes</td>
<td>265</td>
<td>252</td>
<td></td>
</tr>
<tr>
<td>Del. Int.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>352</td>
<td>336</td>
</tr>
<tr>
<td>Kneser-Ney</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>334</td>
<td>323</td>
<td></td>
</tr>
<tr>
<td>Kneser-Ney</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>Kneser-Ney</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>332</td>
<td>321</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>348</td>
<td>334</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>3</td>
<td>150</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>354</td>
<td>340</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>3</td>
<td>200</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>326</td>
<td>312</td>
<td>312</td>
</tr>
<tr>
<td>Class-based</td>
<td>3</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>335</td>
<td>319</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>3</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>343</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>4</td>
<td>2000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>4</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
<tr>
<td>Class-based</td>
<td>5</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>327</td>
<td>312</td>
<td></td>
</tr>
</tbody>
</table>
Parameters

- Bengio NNLM has $d_{\text{hid}} = 100$, $d_{\text{win}} = 5$, $d_{\text{in}} = 5 \times 50$

- In-Class: How many parameters does it have? How does this compare to Kneser-Ney smoothing?
Historical Note

- Bengio et al. notes that many of these aspects predate the work.
- Furthermore proposes many of the ideas that Collobert et al. and word2vec implement and scale.
- Around this time, very few NLP papers on NN, most-cited papers are about conditional random fields (CRFs).
Log-Bilinear Language Model (Mnih & Hinton, 2007)

Slightly different input representation. Now let:

\[ x = \sum_{i=1}^{d_{\text{win}}} v(f_i) C_i \]

- Instead of concatenating, weight each \( v(f_i) \) by position-specific weight matrix \( C_i \).

Then use:

\[ \hat{y} = \text{softmax}(xW^1 + b) \]

- Note no tanh layer.
- \( W^1 \) can use input embeddings too, or not (Mnih and Teh, 2012)
- Can be faster to use, and in some cases simpler.
Comparison

Both count-based models and feed-forward NNLMs are Markovian language models,

Comparison:

- Training Speed: ngrams are much faster (more coming)
- Usage Speed: ngrams very fast, NN can be fast with some tricks.
- Memory: NN models can be much smaller (but there are big ones)
- Accuracy: Comparable for small data, NN does better with more.

Advantages of NN model

- Can be trained end-to-end.
- Does not require smoothing methods.
Translation Performance (and Blunsom, 2015)
Contents

Neural Language Models

Noise Contrastive Estimation
Review: Softmax Issues

Use a softmax to force a distribution,

\[
\text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{\mathbf{w} \in \mathcal{C}} \exp(\mathbf{z}_w)}
\]

\[
\log \text{softmax}(\mathbf{z}) = \mathbf{z} - \log \sum_{\mathbf{w} \in \mathcal{C}} \exp(\mathbf{z}_w)
\]

- **Issue**: class \( \mathcal{C} \) is huge.
- For C&W, 100,000, for word2vec 1,000,000 types
- Note largest dataset is 6 billion words
Unnormalized Scores

Recall the score defined as (dropping bias)

\[ z = \tanh(xW^1)W^2 \]

Unnormalized score of each word before soft-max,

\[ z_j = \tanh(xW^1)W^2_{*,j} \]

for any \( j \in \{1, \ldots, d_{\text{out}}\} \)

Note: can be computed efficiently \( O(1) \) versus \( O(d_{\text{out}}) \).
Coherence

- Saw similar idea earlier for ranking embedding.

- **Idea:** Learn to distinguish coherent n-grams from corruption.

- Want to discriminate correct next words from other choices.

  - [ the dog walks ]
  - [ the dog house ]
  - [ the dog cats ]
  - [ the dog skips ]
Imagine we have a new dataset,

\[((x_1, y_1), d_1), \ldots, ((x_n, y_n), d_n)\],

- \(x\); representation of context \(w_{i-n+1}, \ldots, w_{i-1}\)
- \(y\); a possible \(w_i\)
- \(d\); 1 if \(y\) is correct, 0 otherwise

Objective is based on predicted \(\hat{d}\):

\[
\mathcal{L}(\theta) = \sum_i L_{\text{crossentropy}}(d_i, \hat{d}_i)
\]
Warm-Up: Binary Classification

How do we score \((x_i, y_i = \delta(w))\)?

Could use unnormalized score,

\[ z_w = \tanh(xW^1)W^2 \]

Becomes softmax regression/non-linear logistic regression,

\[ \hat{d} = \sigma(z_w) \]

- Much faster
- But does not help us train LM.
Implementation

Standard MLP language model, (only takes in $x$)

$$x \Rightarrow W^1 \Rightarrow \tanh \Rightarrow W^2 \Rightarrow \text{softmax}$$

Computing binary (takes in $x$ and $y$)

$$\hat{d} = \sigma(z_w)$$

$$x \Rightarrow W^1 \Rightarrow \tanh \Rightarrow \cdot \Rightarrow W^2_{*,w}(\text{Lookup}) \Rightarrow \sigma$$
Noise Contrastive Estimation 1

Probabilistic model,

- Introduce random variable $D$
- If $D = 1$ produce true sample
- If $D = 0$ produce sample from a noise distribution.
- Hyperparameter $K$ is ratio of noise

\[
p(D = 1) = \frac{1}{K + 1}
\]

\[
p(D = 0) = \frac{K}{K + 1}
\]
Noise Contrastive Estimation 2

For a given $x, y$,

$$p(D = 1 | x, y) = \frac{p(y | D = 1, x)p(D = 1 | x)}{\sum_d p(y | D = d, x)p(D = d | x)}$$

$$= \frac{p(y | D = 1, x)p(D = 1 | x)}{p(x | D = 0)p(D = 0 | x) + p(y | D = 1, x)p(D = 1 | x)}$$

Plug-in the noise distribution and hyperparameters,

$$p(D = 1 | x, y) = \frac{\frac{1}{K+1} p(y | D = 1, x)}{\frac{1}{K+1} p(y | D = 1, x) + \frac{K}{K+1} p(y | D = 0, x)}$$

$$= \frac{p(y | D = 1, x)}{p(y | D = 1, x) + Kp(y | D = 0, x)}$$

$$= \sigma(\log p(y | D = 1, x) - \log(Kp(y | D = 0, x)))$$
Noise Contrastive Estimation 3

With

\[ p(D = 1|x, y) = \sigma(\log p(y|D = 1, x) - \log(Kp(y|D = 0, x))) \]

we the training objective for a corpus that has \( K \) noise samples \( s_{i,k} \) per example is:

\[
\mathcal{L}(\theta) = \sum_i \log p(D = 1|x_i, y_i) + \sum_{k=1}^K \log p(D = 0|x_i, Y = s_{i,k}) \\
= \sum_i \log \sigma (\log p(y_i|D = 1, x_i) - \log(Kp(y_i|D = 0, x_i))) \\
+ \sum_{k=1}^K \log (1 - \sigma (\log p(s_{i,k}|D = 1, x_i) - \log(Kp(s_{i,k}|D = 0, x_i))))
\]

- In practice, sample \( s_{i,k} \) from unigram distribution
But we still have a problem: $\mathcal{L}$ defined in terms of normalized distributions $\log p(\mathbf{y}|D = 1, \mathbf{x})$

**Solution:**

- instead of explicitly normalizing, estimate $Z(\mathbf{x})$, normalizing constant of each context $\mathbf{x}$, *as a parameter* (Gutmann & Hyvärinen, 2010)
- Mnih and Teh (2012) show that fixing $Z(\mathbf{x}) = 1$ for all contexts works just as well
- So we can replace $\log p(\mathbf{y} = \delta(w)|D = 1, \mathbf{x})$ with $z_w$, as computed by our network
Noise Contrastive Estimation 5

So we now have

\[
\mathcal{L}(\theta) = \sum_i \log \sigma(z_{w_i} - \log(Kp_{ML}(w_i))) \\
+ \sum_{k=1}^{K} \log(1 - \sigma(z_{s_{i,k}} - \log(Kp_{ML}(s_{i,k}))))
\]

- Mnih and Teh (2012) show that gradient of \( \mathcal{L} \) approaches gradient of true language model’s log-likelihood objective as \( k \to \infty \).
Implementation

- How do you efficiently compute $z_w$?
  Need a lookup table (and dot-product) for output embeddings!
  (Not full matrix-vector product).

- How do you efficiently handle $\log p_{ML}(w)$?
  Can be precomputed or placed in a lookup table.

- How do you handle sampling?
  Can precompute large number of samples (not example specific).

- How do you handle loss?
  Simply BinaryNLL Objective.
Implementation

Standard MLP language model,

\[ \mathbf{x} \Rightarrow \mathbf{W}^1 \Rightarrow \tanh \Rightarrow \mathbf{W}^2 \Rightarrow \text{softmax} \]

Computing \( \sigma(z_w - \log(Kp_{ML}(w))) \),

\[ \mathbf{x} \Rightarrow \mathbf{W}^1 \Rightarrow \tanh \Rightarrow \mathbf{W}^2_{*,w}(\text{Lookup}) \Rightarrow - \log Kp_{ML}(w)(\text{input}) \Rightarrow \sigma \]

(Efficiency, compute first three layers only once for \( K + 1 \))
Several options for test time,

- Use full softmax with learned parameters.
- Compute subset of scores and renormalize (homework).
- Can sometimes just use treat unnormalized params as being normalized (self-normalization).