CS388: Natural Language Processing
Lecture 6: Language Modeling, Self Attention

Greg Durrett
Adminstrivia

• Project 2 due on Feb 13

• Greg’s Wednesday OHs pushed back to 1:15pm-2:15pm (by 15 minutes)
Recap: Skip-Gram

• Predict one word of context from word

\[
\text{the dog bit the man}
\]

• Parameters: \( d \times |V| \) vectors, \(|V| \times d\) output parameters (W) (also usable as vectors!)

• Predicting the next word from a word will be similar to language modeling (focus of this lecture!)

\[
P(w' | w) = \text{softmax}(W e(w))
\]

Mikolov et al. (2013)
Recap: GloVe

- Objective: \[ \sum_{i,j} f(\text{count}(w_i, c_j)) \left( w_i^\top c_j + a_i + b_j - \log \text{count}(w_i, c_j) \right)^2 \]

<table>
<thead>
<tr>
<th></th>
<th>the</th>
<th>dog</th>
<th>cat</th>
<th>ran</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0</td>
<td>200</td>
<td>200</td>
<td>0</td>
</tr>
<tr>
<td>dog</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>cat</td>
<td>200</td>
<td>0</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>ran</td>
<td>0</td>
<td>15</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

Linear regression with 12 pairs: each element is plugged into the above equation

+ constant = log count of pair

(made up values — matrix will generally be symmetric, though)

Pennington et al. (2014)
Recap: Using Embeddings

- Approach 1: learn embeddings as parameters from your data
- Approach 2: initialize using GloVe, keep fixed
- Approach 3: initialize using GloVe, fine-tune

- Nearly all modern transfer learning uses Approach 3 (e.g., fine-tuning BERT). And you don’t just fine-tune embeddings, but instead use an entire language model
Today

- Language modeling intro
- Neural language modeling
- Self-attention
- Multi-head self-attention
- Positional encodings (if time)
Language Modeling
Language Modeling

- Fundamental task in both linguistics and NLP: can we determine of a sentence is *acceptable* or not?
- Related problem: can we evaluate if a sentence is grammatical? Plausible? Likely to be uttered?
- Language models: place a distribution $P(w)$ over strings $w$ in a language. This is related to all of these tasks but doesn’t exactly map onto them.
- Today: autoregressive models $P(w) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots$
- Turns out this is also useful as a pre-training task like skip-gram!
N-gram Language Models

\[ P(w) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots \]

- n-gram models: distribution of next word is a categorical conditioned on previous n-1 words
  \[ P(w_i|w_1, \ldots, w_{i-1}) = P(w_i|w_{i-n+1}, \ldots, w_{i-1}) \]

- Markov property: don’t remember all the context but only consider a few previous words

I visited San _____ put a distribution over the next word
2-gram: \( P(w | \text{San}) \)
3-gram: \( P(w | \text{visited San}) \)
4-gram: \( P(w | \text{I visited San}) \)
N-gram Language Models

\[ P(w) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \ldots \]

- n-gram models: distribution of next word is a categorical conditioned on previous n-1 words
  \[ P(w_i|w_1, \ldots, w_{i-1}) = P(w_i|w_{i-n+1}, \ldots, w_{i-1}) \]

\[ P(w|\text{visited San}) = \frac{\text{count(visited San, } w)}{\text{count(visited San)}} \]

- 3-gram probability, maximum likelihood estimate from a corpus (remember: count and normalize for MLE)

- Just relies on counts, even in 2008 could scale up to 1.3M word types, 4B n-grams (all 5-grams occurring >40 times on the Web)
Smoothing N-gram Language Models

- What happens when we scale to longer contexts?

\[ P(w|\text{to}) \] \( \text{to occurs 1M times in corpus} \)

\[ P(w|\text{go to}) \] \( \text{go to occurs 50,000 times in corpus} \)

\[ P(w|\text{to go to}) \] \( \text{to go to occurs 1500 times in corpus} \)

\[ P(w|\text{want to go to}) \] \( \text{want to go to: only 100 occurrences} \)

- Probability counts get very sparse, and we often want information from 5+ words away

- What can we do?
Smoothing N-gram Language Models

I visited San _____ put a distribution over the next word

- Smoothing is very important, particularly when using 4+ gram models

\[
P(w|\text{visited San}) = (1 - \lambda) \frac{\text{count}(\text{visited San}, w)}{\text{count}(\text{visited San})} + \lambda \frac{\text{count}(\text{San}, w)}{\text{count}(\text{San})}
\]

- One technique is “absolute discounting:” subtract off constant \( k \) from numerator, set lambda to make this normalize \((k=1 \text{ is like leave-one-out})\)

\[
P(w|\text{visited San}) = \frac{\text{count}(\text{visited San}, w) - k}{\text{count}(\text{visited San})} + \lambda \frac{\text{count}(\text{San}, w)}{\text{count}(\text{San})}
\]

- Smoothing schemes get very complex!
The Power of Language Modeling

My name _____
- One good option (is)?

My name is _____
- Flat distribution over many alternatives. But hard to get a good distribution?

I visited San _____
- Requires some knowledge but not one right answer

The capital of Texas is _____
- Requires more knowledge (one answer...or is there?)

The casting and direction were top notch. Overall I thought the movie was ___
- Requires basically doing sentiment analysis!
Neural Language Modeling
Early work: feedforward neural networks looking at context

\[ P(w_i|w_{i-n}, \ldots, w_{i-1}) \]

I visited New _____

Slow to train over lots of data! But otherwise this seems okay?

Bengio et al. (2003)
Problems with FFNNs

\[ x = I \text{ visited New York. I had a really fun time going up the ___} \]

- What are some words that can show up here? How do we know?

- What do we learn from this example?
Challenges of Neural Language Modeling

I visited New _____

‣ Advantages and disadvantages of these?
Contextualized Embeddings

- Both RNNs and Transformers (and other models) can produce *contextualized embeddings*

\[
e = (e_1, e_2, \ldots, e_n) \quad e_i = f(x_1, x_2, \ldots, x_i)
\]

\[
x = (x_1, x_2, \ldots, x_n)
\]

\[
x = I \text{ visited New York. I had a really fun time going up the __} 
\]

- Can also have bidirectional embedding representations, but learning these needs *masked language models* (later in the course)

- One solution: \( e(x) = f(x_{-1}, the) \)
RNNs: Why not?

- Slow. They do not parallelize and there are $O(n)$ non-parallel operations to encode $n$ items.
- Even modifications like LSTMs still don’t enable learning over very long sequences. Transformers can scale to thousands of words!
(Self-)Attention
Fixed-length sequence of As and Bs

- All As = last letter is A; any B = last letter is B

- **Attention**: method to access arbitrarily far back in context from this point

RNNs generally struggle with this; remembering context for many positions is hard (though of course they can do this simplified example — you can even hand-write weights to do it!)
Keys and Query

- Keys: embedded versions of the sentence; query: what we want to find

Assume $A = [1, 0]$; $B = [0, 1]$ (one-hot encodings of the tokens); call these $e_i$

Step 1: Compute scores for each key

<table>
<thead>
<tr>
<th>keys $k_i$</th>
<th>query: $q = [0, 1]$ (we want to find Bs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[1, 0]$</td>
<td>$[1, 0]$</td>
</tr>
<tr>
<td>$[0, 1]$</td>
<td>$[1, 0]$</td>
</tr>
</tbody>
</table>

$A \quad A \quad B \quad A$

$s_i = k_i^T q$

| 0 | 0 | 1 | 0 |
Step 1: Compute scores for each key

keys $k_i$

$[1, 0] [1, 0] [0, 1] [1, 0]$

query: $q = [0, 1]$ (we want to find Bs)

$A A B A$

$$s_i = k_i^T q$$

$$0 0 1 0$$

Step 2: Softmax the scores to get probabilities $\alpha$

$$0 0 1 0 \Rightarrow (1/6, 1/6, 1/2, 1/6) \text{ if we assume } e=3$$

Step 3: Compute output values by multiplying embs. by alpha + summing

result = $\sum (\alpha_i e_i) = 1/6 [1, 0] + 1/6 [1, 0] + 1/2 [0, 1] + 1/6 [1, 0] = [1/2, 1/2]$
Attention

keys $k_i$

[1, 0] [1, 0] [0, 1] [1, 0]

(1/6, 1/6, 1/2, 1/6) if we assume $e=3$

query: $q = [0, 1]$ (we want to find Bs)

A   A   B   A

result = $\sum(\alpha_i e_i) = 1/6 [1, 0] + 1/6 [1, 0] + 1/2 [0, 1] + 1/6 [1, 0] = [1/2, 1/2]$

How does this differ from just averaging the vectors (DAN)?

What if we have a very very long sequence?
New Keys

keys $k_i$

$[1, 0] [1, 0] [0, 1] [1, 0]$

query: $q = [0, 1]$ (we want to find Bs)

A  A  B  A

We can make attention more peaked by not setting keys equal to embeddings.

$k_i = W^K e_i$

$W^K = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$

$[10, 0][10, 0][0, 10][10, 0]$

What will new attention values be with these keys?
Attention, Formally

- Original “dot product” attention: \( s_i = k_i^T q \)

- Scaled dot product attention: \( s_i = k_i^T W q \)

- Equivalent to having two weight matrices: \( s_i = (W^K k_i)^T (W^Q q) \)

- Other forms exist: Luong et al. (2015), Bahdanau et al. (2014) present some variants (originally for machine translation)
Self-Attention

- Self-attention: every word is both a key and a query simultaneously

Q: seq len x d matrix (d = embedding dimension = 2 for these slides)

K: seq len x d matrix

\[ W^Q = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \]

no matter what the value is, we’re going to look for Bs

\[ W^K = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \]

“booster” as before

Note: there are many ways to set up these weights that will be equivalent to this
Self-Attention

\[ E = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \]

\[ W^Q = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ W^K = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix} \]

\[ Q = E \left( W^Q \right) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ K = E \left( W^K \right) = \begin{pmatrix} 10 & 0 \\ 10 & 0 \\ 0 & 10 \\ 10 & 0 \end{pmatrix} \]

Scores \( S = QK^T \quad S_{ij} = q_i \cdot k_j \)

len x len = (len x d) x (d x len)

Let’s compute these now!
Self-Attention

\[ E = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \]

\[ W^Q = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ W^K = \begin{pmatrix} 10 & 0 \\ 0 & 10 \end{pmatrix} \]

\[ Q = E (W^Q) = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \]

\[ K = E (W^K) = \begin{pmatrix} 10 & 0 \\ 10 & 0 \\ 0 & 10 \\ 10 & 0 \end{pmatrix} \]

Scores \( S = QK^T \)

[\( S_{ij} = q_i \cdot k_j \)]

[\( \text{len x len} = (\text{len x d}) \times (\text{d x len}) \)]

Final step: softmax to get attentions \( A \), then output is \( AE \)

*technically it’s \( A (E W^V) \), using a values matrix \( V = EW^V \)
Self-Attention (Vaswani et al.)

\[
\text{Attention}(Q, K, V) = \text{softmax}\left( \frac{QQ^T}{\sqrt{d_k}} \right) V
\]

\[
Q = EW^Q, \quad K = EW^K, \quad V = EW^V
\]

- Normalizing by \( \sqrt{d_k} \) helps control the scale of the softmax, makes it less peaked
- This is just one head of self-attention — produce multiple heads via randomly initialize parameter matrices (more in a bit)

Vaswani et al. (2017)
Self-Attention

Alammar, *The Illustrated Transformer*
Self-Attention

Alammar, *The Illustrated Transformer*

sent len x sent len (attn for each word to each other)

\[
\begin{align*}
X \times W^Q &= Q \\
X \times W^K &= K \\
X \times W^V &= V \\
\text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) = Z \\
Z &= \text{sent len x hidden dim}
\end{align*}
\]

Z is a weighted combination of V rows
## Properties of Self-Attention

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Complexity per Layer</th>
<th>Sequential Operations</th>
<th>Maximum Path Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Attention</td>
<td>$O(n^2 \cdot d)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Recurrent</td>
<td>$O(n \cdot d^2)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Convolutional</td>
<td>$O(k \cdot n \cdot d^2)$</td>
<td>$O(1)$</td>
<td>$O(\log_k(n))$</td>
</tr>
<tr>
<td>Self-Attention (restricted)</td>
<td>$O(r \cdot n \cdot d)$</td>
<td>$O(1)$</td>
<td>$O(n/r)$</td>
</tr>
</tbody>
</table>

- $n = \text{sentence length}$, $d = \text{hidden dim}$, $k = \text{kernel size}$, $r = \text{restricted neighborhood size}$

- **Quadratic complexity**, but $O(1)$ sequential operations (not linear like in RNNs) and $O(1)$ “path” for words to inform each other

Vaswani et al. (2017)
Multi-Head Self-Attention
Multi-head Self-Attention

Just duplicate the whole computation with different weights:

Alammar, *The Illustrated Transformer*
Multi-head Self-Attention

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads.
   We multiply $X$ or $R$ with weight matrices

* In all encoders other than #0, we don't need embedding.
   We start directly with the output of the encoder right below this one.
Multi-head Self-Attention

1) This is our input sentence*
2) We embed each word*
3) Split into 8 heads. We multiply $X$ or $R$ with weight matrices
4) Calculate attention using the resulting $Q/K/V$ matrices
5) Concatenate the resulting $Z$ matrices, then multiply with weight matrix $W^o$ to produce the output of the layer

* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one.
Challenges of Neural Language Modeling

FFNN

Self-attention:

DAN

Still missing one component:
position sensitivity

I visited New _____

I visited New _____
Positional Encodings
Transformers: Position Sensitivity

- Encode each sequence position as an integer, add it to the word embedding vector

- Why does this work?
Transformers

Alammar, *The Illustrated Transformer*

- Alternative from Vaswani et al.: sines/cosines of different frequencies (closer words get higher dot products by default)
Takeaways

‣ Language modeling is a fundamental task

‣ n-gram models are a basic, scalable solution but have limited context

‣ Self-attention is a solution to the question of: how do we look at a lot of context, efficiently, without blowing up parameter counts, and without forgetting far-back things?

‣ Next time: see the whole Transformer architecture and extensions of it