Neural Language Modeling
Cont. and Pretrained LMs

CSCI 1460: Computational Linguistics
Lecture 11

Ellie Pavlick
Fall 2022
The below question concerns scaled dot-product attention in the context of the Transformer architecture. Below are the key, query, and value vectors for each word in a sentence at layer k. Consider how much each word will influence the representation of the word "NLP" at layer k+1. Rank the words in the sentence in decreasing order of influence (i.e., the word with the most influence/highest weight ranked first, the word with the lowest weight ranked last).

\[
x_0 = 1 \quad x_1 = \text{effing} \quad x_2 = \text{love} \quad x_3 = \text{NLP}
\]

\[
q_0 = [1 \ 5 \ 7] \quad q_1 = [2 \ 5 \ 6] \quad q_2 = [2 \ 7 \ 5] \quad q_3 = [3 \ 8 \ 1]
\]

\[
k_0 = [3 \ 4 \ 9] \quad k_1 = [3 \ 5 \ 3] \quad k_2 = [5 \ 4 \ 9] \quad k_3 = [1 \ 5 \ 4]
\]

\[
v_0 = [0 \ 2 \ 6] \quad v_1 = [1 \ 9 \ 1] \quad v_2 = [7 \ 7 \ 2] \quad v_3 = [4 \ 6 \ 5]
\]

- i, effing, love, NLP
- NLP, love, effing, i
- effing, love, NLP, i
- love, effing, i, NLP
Step 1: dot($q$, $k$) to determine the scores

\[
\begin{align*}
  x_0 &= \text{l} & x_1 &= \text{effing} & x_2 &= \text{love} & x_3 &= \text{NLP} \\
  q_0 &= [1 \ 5 \ 7] & q_1 &= [2 \ 5 \ 6] & q_2 &= [2 \ 7 \ 5] & q_3 &= [3 \ 8 \ 1] \\
  k_0 &= [3 \ 4 \ 9] & k_1 &= [3 \ 5 \ 3] & k_2 &= [5 \ 4 \ 9] & k_3 &= [1 \ 5 \ 4] \\
  v_0 &= [0 \ 2 \ 6] & v_1 &= [1 \ 9 \ 1] & v_2 &= [7 \ 7 \ 2] & v_3 &= [4 \ 6 \ 5]
\end{align*}
\]
Step 1: dot(q, k) to determine the scores

<table>
<thead>
<tr>
<th>(x_0 = I)</th>
<th>(x_1 = \text{effing})</th>
<th>(x_2 = \text{love})</th>
<th>(x_3 = \text{NLP})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0 = [1 \ 5 \ 7])</td>
<td>(q_1 = [2 \ 5 \ 6])</td>
<td>(q_2 = [2 \ 7 \ 5])</td>
<td>(q_3 = [3 \ 8 \ 1])</td>
</tr>
<tr>
<td>(k_0 = [3 \ 4 \ 9])</td>
<td>(k_1 = [3 \ 5 \ 3])</td>
<td>(k_2 = [5 \ 4 \ 9])</td>
<td>(k_3 = [1 \ 5 \ 4])</td>
</tr>
<tr>
<td>(v_0 = [0 \ 2 \ 6])</td>
<td>(v_1 = [1 \ 9 \ 1])</td>
<td>(v_2 = [7 \ 7 \ 2])</td>
<td>(v_3 = [4 \ 6 \ 5])</td>
</tr>
</tbody>
</table>
Step 1: \( \text{dot}(q, k) \) to determine the scores

\[
\begin{align*}
\mathbf{x}_0 &= \text{l} \\
\mathbf{q}_0 &= [1 \ 5 \ 7] \\
\mathbf{k}_0 &= [3 \ 4 \ 9] \\
\mathbf{v}_0 &= [0 \ 2 \ 6] \\
\mathbf{x}_1 &= \text{effing} \\
\mathbf{q}_1 &= [2 \ 5 \ 6] \\
\mathbf{k}_1 &= [3 \ 5 \ 3] \\
\mathbf{v}_1 &= [1 \ 9 \ 1] \\
\mathbf{x}_2 &= \text{love} \\
\mathbf{q}_2 &= [2 \ 7 \ 5] \\
\mathbf{k}_2 &= [5 \ 4 \ 9] \\
\mathbf{v}_2 &= [7 \ 7 \ 2] \\
\mathbf{x}_3 &= \text{NLP} \\
\mathbf{q}_3 &= [3 \ 8 \ 1] \\
\mathbf{k}_3 &= [1 \ 5 \ 4] \\
\mathbf{v}_3 &= [4 \ 6 \ 5]
\end{align*}
\]
Step 1: dot(q, k) to determine the scores

\[ x_0 = \text{l}, \quad x_1 = \text{effing}, \quad x_2 = \text{love}, \quad x_3 = \text{NLP} \]

\[ q_0 = [1 \ 5 \ 7], \quad q_1 = [2 \ 5 \ 6], \quad q_2 = [2 \ 7 \ 5], \quad q_3 = [3 \ 8 \ 1] \]

\[ k_0 = [3 \ 4 \ 9], \quad k_1 = [3 \ 5 \ 3], \quad k_2 = [5 \ 4 \ 9], \quad k_3 = [1 \ 5 \ 4] \]

\[ v_0 = [0 \ 2 \ 6], \quad v_1 = [1 \ 9 \ 1], \quad v_2 = [7 \ 7 \ 2], \quad v_3 = [4 \ 6 \ 5] \]

\[ 50 \quad 52 \]
Step 1: dot($q, k$) to determine the scores

<table>
<thead>
<tr>
<th>$x_0 = l$</th>
<th>$x_1 = effing$</th>
<th>$x_2 = love$</th>
<th>$x_3 = NLP$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0 = [1 \ 5 \ 7]$</td>
<td>$q_1 = [2 \ 5 \ 6]$</td>
<td>$q_2 = [2 \ 7 \ 5]$</td>
<td>$q_3 = [3 \ 8 \ 1]$</td>
</tr>
<tr>
<td>$k_0 = [3 \ 4 \ 9]$</td>
<td>$k_1 = [3 \ 5 \ 3]$</td>
<td>$k_2 = [5 \ 4 \ 9]$</td>
<td>$k_3 = [1 \ 5 \ 4]$</td>
</tr>
<tr>
<td>$v_0 = [0 \ 2 \ 6]$</td>
<td>$v_1 = [1 \ 9 \ 1]$</td>
<td>$v_2 = [7 \ 7 \ 2]$</td>
<td>$v_3 = [4 \ 6 \ 5]$</td>
</tr>
</tbody>
</table>

50 \hspace{2cm} 52 \hspace{2cm} 56 \hspace{2cm} 47
The below question concerns scaled dot-product attention in the context of the Transformer architecture. Below are the key, query, and value vectors for each word in a sentence at layer k. Consider how much each word will influence the representation of the word "NLP" at layer k+1. Rank the words in the sentence in decreasing order of influence (i.e., the word with the most influence/highest weight ranked first, the word with the lowest weight ranked last).

\[ x_0 = [3, 8, 1] \]
\[ q = [3, 8, 1] \]
\[ k_0 = [3, 5, 3] \]
\[ v_0 = [1, 5, 4] \]
\[ \text{lst} = [ ('i', [3, 4, 9]), ('effing', [3, 5, 3]), ('love', [5, 4, 9]), ('NLP', [1, 5, 4]) ] \]

scores = []
for word, vec in lst:
    score = np.dot(q, vec)
scores.append((word, score))
for w, s in sorted(scores, key=lambda e:e[1], reverse=True):
    print(f'{w} {s}')

love 56
effing 52
i 50
NLP 47
Step 2: normalize (with softmax)

\[ x_0 = \text{l} \quad x_1 = \text{effing} \quad x_2 = \text{love} \quad x_3 = \text{NLP} \]

\[
\begin{align*}
q_0 &= [1 \ 5 \ 7] \\
k_0 &= [3 \ 4 \ 9] \\
v_0 &= [0 \ 2 \ 6]
\end{align*}
\]
\[
\begin{align*}
q_1 &= [2 \ 5 \ 6] \\
k_1 &= [3 \ 5 \ 3] \\
v_1 &= [1 \ 9 \ 1]
\end{align*}
\]
\[
\begin{align*}
q_2 &= [2 \ 7 \ 5] \\
k_2 &= [5 \ 4 \ 9] \\
v_2 &= [7 \ 7 \ 2]
\end{align*}
\]
\[
\begin{align*}
q_3 &= [3 \ 8 \ 1] \\
k_3 &= [1 \ 5 \ 4] \\
v_3 &= [4 \ 6 \ 5]
\end{align*}
\]

\[
\begin{align*}
0.002 & \quad 0.018 \\
0.980 & \quad 0.000
\end{align*}
\]

```python
# softmax
scores = exp(scores) / (exp(scores).sum())
print("scores after softmax", scores)
```
Step 3: multiply score by \( v \)

<table>
<thead>
<tr>
<th>( x_0 = l )</th>
<th>( x_1 = \text{effing} )</th>
<th>( x_2 = \text{love} )</th>
<th>( x_3 = \text{NLP} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 = [1 \ 5 \ 7] )</td>
<td>( q_1 = [2 \ 5 \ 6] )</td>
<td>( q_2 = [2 \ 7 \ 5] )</td>
<td>( q_3 = [3 \ 8 \ 1] )</td>
</tr>
<tr>
<td>( k_0 = [3 \ 4 \ 9] )</td>
<td>( k_1 = [3 \ 5 \ 3] )</td>
<td>( k_2 = [5 \ 4 \ 9] )</td>
<td>( k_3 = [1 \ 5 \ 4] )</td>
</tr>
<tr>
<td>( v_0 = [0 \ 2 \ 6] )</td>
<td>( v_1 = [1 \ 9 \ 1] )</td>
<td>( v_2 = [7 \ 7 \ 2] )</td>
<td>( v_3 = [4 \ 6 \ 5] )</td>
</tr>
</tbody>
</table>

| \( 0.002 \) | \( 0.018 \) | \( 0.980 \) | \( 0.000 \) |

\( v_0 = [0.00 \ 0.00 \ 0.01] \quad v_1 = [0.02 \ 0.16 \ 0.02] \quad v_2 = [6.86 \ 6.86 \ 1.96] \quad v_3 = [0.00 \ 0.00 \ 0.00] \)

```python
output_vector = [0., 0., 0.]
for i, (word, v) in enumerate(vs):
    output_vector += np.multiply(scores[i], v)
print("output vector: ", output_vector)
```
Step 4: sum up the weighted vs

<table>
<thead>
<tr>
<th>(x_0 = I)</th>
<th>(x_1 = \text{effing})</th>
<th>(x_2 = \text{love})</th>
<th>(x_3 = \text{NLP})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0 = [1 \ 5 \ 7])</td>
<td>(q_1 = [2 \ 5 \ 6])</td>
<td>(q_2 = [2 \ 7 \ 5])</td>
<td>(q_3 = [3 \ 8 \ 1])</td>
</tr>
<tr>
<td>(k_0 = [3 \ 4 \ 9])</td>
<td>(k_1 = [3 \ 5 \ 3])</td>
<td>(k_2 = [5 \ 4 \ 9])</td>
<td>(k_3 = [1 \ 5 \ 4])</td>
</tr>
<tr>
<td>(v_0 = [0 \ 2 \ 6])</td>
<td>(v_1 = [1 \ 9 \ 1])</td>
<td>(v_2 = [7 \ 7 \ 2])</td>
<td>(v_3 = [4 \ 6 \ 5])</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
0.002 & \quad 0.018 & \quad 0.980 & \quad 0.000 \\
\end{align*}
\]

\[
\begin{align*}
v_0 &= [0.00 \ 0.00 \ 0.01] & v_1 &= [0.02 \ 0.16 \ 0.02] & v_2 &= [6.86 \ 6.86 \ 1.96] & v_3 &= [0.00 \ 0.00 \ 0.00] \\
\end{align*}
\]

\[
[6.87 \ 7.02 \ 1.99]
\]

```python
output_vector = [0., 0., 0.]
for i, (word, v) in enumerate(vs):
    output_vector += np.multiply(scores[i], v)
print(f"output vector: [{output_vector}]")
```
Topics

• Transformers Cont. (+ any questions you have!)
• What is pretraining?
• Contextualized Word Representations:
  • ELMo
  • BERT
  • GPT
• Finetuning and Other Transfer Methods
Topics

- **Transformers Cont. (+ any questions you have!)**
- What is pretraining?
- Contextualized Word Representations:
  - ELMo
  - BERT
  - GPT
- Finetuning and Other Transfer Methods
Topics

• Transformers
  • Architecture
• Self Attention
• Blocks
• Positional Encodings
• So What?
Transformers

Attention

“the”

\( o_0 \)

\( h_{00} \)

\( e_0 \)

\( t_0 = \text{the} \)

“cat”

\( o_1 \)

\( h_{11} \)

\( e_1 \)

\( t_1 = \text{cat} \)

“sat”

\( o_2 \)

\( h_{02} \)

\( e_2 \)

\( t_2 = \text{sat} \)
 Transformers

Attention

“Encoder-Decoder” Architecture

h_{00} \quad h_{11} \quad h_{02}

h_{00} \quad h_{01} \quad h_{02}

e_0 \quad e_1 \quad e_2

*Note: This figure is a cartoon to communicate the idea of attention. It doesn’t necessarily align with the actual implementation of a transformer.
Transformers

Attention

“Encoder-Decoder” Architecture

\[ t_0 = \text{the} \quad t_1 = \text{cat} \quad t_2 = \text{sat} \]

\[ h_{00} \quad h_{11} \quad h_{02} \]

\[ e_0 \quad e_1 \quad e_2 \]

“el”

“gato”

“sentó”

*Note: This figure is a cartoon to communicate the idea of attention. It doesn’t necessarily align with the actual implementation of a transformer.
**Transformers**

**Attention**

"Encoder-Decoder" Architecture

```
\begin{align*}
    t_0 &= \text{the} \\
    t_1 &= \text{cat} \\
    t_2 &= \text{sat}
\end{align*}
```

```
\begin{align*}
    h_{00} &\rightarrow h_{11} \\
    h_{00} &\rightarrow h_{01} \\
    h_{00} &\rightarrow h_{02} \\
    e_0 &\rightarrow h_{00} \\
    e_1 &\rightarrow h_{01} \\
    e_2 &\rightarrow h_{02}
\end{align*}
```

`O_0`'s attention is directed towards `"el"`

`O_1`'s attention is directed towards `"gato"`

`O_2`'s attention is directed towards `"sentó"`

*Note: This figure is a cartoon to communicate the idea of attention. It doesn't necessarily align with the actual implementation of a transformer.*
Transformers

Attention

https://www.tensorflow.org/text/tutorials/transformer
Transformers
(Self) Attention

learned key, query, and value vectors for each position
Transformers
(Self) Attention

key * query produces a weight
Transformers  
(Self) Attention

$\text{the}$  
$\text{cat}$  
$\text{sat}$

The inputed to next layer is weighted combination of values

$v_{00}$  
v$_{01}$  
v$_{02}$

$h_{00}$  
h$_{01}$  
h$_{02}$

t$_0 = \text{the}$  
t$_1 = \text{cat}$  
t$_2 = \text{sat}$
Transformers
(Self) Attention

• Each word has three roles:
  • Query: The word as the current focus. I.e., attention is trying to determine how to process this word.
  • Key: The word as a context word. I.e., attention is determining how to use this word to inform the query.
  • Value: The word as part of the output. I.e., attention is determining how to use this word, based on the key-query, to produce an output.

• Every word acts in all three roles at each timestep.
• We learn three weight matrices (Q,K,V) to cast each word into each role
Transformers
(Self) Attention

- Each word has three roles:
  - Query: The word as the current focus. I.e., attention is trying to determine how to process this word.
  - Key: The word as a context word. I.e., attention is determining how to use this word to inform the query.
  - Value: The word as part of the output. I.e., attention is determining how to use this word, based on the key-query, to produce an output.
- Every word acts in all three roles at each timestep.
- We learn three weight matrices \((Q,K,V)\) to cast each word into each role

\[
q_i = W^Q x_i; \quad k_i = W^K x_i; \quad v_i = W^V x_i
\]
Transformers
(Self) Attention

• To actually compute the attention:
  • score = dot(k,q)
  • score is just a scaler number
  • y = weighted_sum of values = sum (score*v)
Transformers
(Self) Attention

Figure 9.16 Calculating the value of $y_3$, the third element of a sequence using causal (left-to-right) self-attention.
Transformers
(Self) Attention

\[ \text{dot}(q, k) \]

**Figure 9.16** Calculating the value of \( y_3 \), the third element of a sequence using causal (left-to-right) self-attention.
Transformers
(Self) Attention

score*value

Figure 9.16 Calculating the value of $y_3$, the third element of a sequence using causal (left-to-right) self-attention.
Repeat the attention process multiple times. Each KQV set can focus on a slightly different aspect of the representation.
Topics

- Transformers
  - Architecture
  - Self Attention
- Blocks
  - Positional Encodings
  - So What?
Transformers
Transformer Blocks

\[ t_0 = \text{the} \]
\[ t_1 = \text{cat} \]
\[ t_2 = \text{sat} \]
Transformers
Transformer Blocks

“the”

“cat”

“sat”

y_0

y_1

y_2

X_0

t_0 = the

X_1

t_1 = cat

X_2

t_2 = sat
Figure 9.18 A transformer block showing all the layers.
Transformers
Transformer Blocks

Figure 9.18  A transformer block showing all the layers.
Transformers
Transformer Blocks

same idea z-score normalization: subtract mean, divide by standard deviation (with some learnable parameters, of course)

Figure 9.18 A transformer block showing all the layers.
Transformers
Transformer Blocks

A transformer block showing all the layers.

Figure 9.18
Topics

- Transformers
  - Architecture
  - Self Attention
  - Blocks
- Positional Encodings
- So What?
Transformers
Positional Encodings

• Unlike RNNs/LSTMs—Transformers aren’t actually aware of the order in which words occur!
• Essentially, they are a (very fancy) bag of words
• Solution: Positional encodings
  • Idea: Just include an input with each word saying what position it is (e.g., “cat in the 3rd position”, “sat in the 4th position”)
A simple way to model position: simply adding an embedding representation of the absolute position to the input word embedding.
Transformers
Positional Encodings

- Problems
  - Not the same as relative/order information (which we want in language). (Later work introduces relative positional encodings instead, and they seem to work better)
  - Less supervision for later positions
  - What about language being infinitely recursive?
Topics

- Transformers
  - Architecture
  - Self Attention
  - Blocks
  - Positional Encodings
- So What?
Transformers
What’s the big deal?

- Attention
  - Very **minimal inductive bias**
  - Any **arbitrary graph structure** over the inputs can be learned
- Multiheadedness
  - Each “head” focuses on a **different subspace** of the input
  - E.g., one head can highlight syntactically connected words, while another finds pragmatically relevant information unconstrained by syntax
  
  - Ty is my cat. Yesterday while my husband and I weren’t looking **he** killed a bird
Transformers
What’s the big deal?

• Attention
  • Very **minimal inductive bias**
  • Any **arbitrary graph structure** over the inputs can be learned
• Multiheadedness
  • Each “head” focuses on a **different subspace** of the input
  • E.g., one head can highlight syntactically connected words, while another finds pragmatically relevant information unconstrained by syntax
  • Ty is my cat. Yesterday while my husband and I weren’t looking he killed a **bird**
Transformers
What’s the big deal?

• Also: very scalable!
  • At layer N, no dependency between timesteps, so can be trained completely in parallel (unlike RNNs)
  • Faster training = bigger models + more data
  • Allows for massive pretraining
Topics

- Transformers Cont. (+ any questions you have!)
- **What is pretraining?**
- Contextualized Word Representations:
  - ELMo
  - BERT
  - GPT
- Finetuning and Other Transfer Methods
NLP: A brief history

1910: Church-Turing Thesis

1930: Logic and Computation: Tarski, Church, Turing

1950: Formal Linguistics: Montague, Chomsky

1956: Dartmouth Workshop

1962: ACL Founded

1965: Chomsky “Aspects of a Theory of Syntax”

1970: Early NLP: Mix of rule-based and info-theory methods

1988: IBM Model 1

1990: Statistical NLP: “Traditional” ML, standardized evals

2012: AlexNet

2018: ELMo

Pretraining and Transfer Learning

End-to-end deep learning
Recap: Basic Perceptron

Trainable Embeddings

Task: Predict the next word
Input: cat
Expected: sat

Train embeddings via backprop, just like other parameters
Recap: Pretrained Word Embeddings
Evaluations of word2vec embeddings
**Pretraining**

What makes “pretraining” different than just a trained embedding layer?

- No need to train embeddings anew for every task.
- Instead, train on some “general” task that encourages good representations, and then transfer these representations with little (or no) updating to other tasks.
- So: word embeddings are one type of “pretraining”
- But: no need to pretrain just the embeddings. Instead, pretrain the entire network. When it’s time to transfer, update only a small number of parameters.
Pretraining

Model Size (number of parameters)

- Spring 2018: ELMo
- Fall 2018: BERT
- Spring 2019: GPT2
- Winter 2020: TNLG
- Spring 2020: OpenAI
- Spring 2022: PALM

figure adapted from https://bmk.sh/2020/05/29/GPT-3-A-Brief-Summary/
Topics

- Transformers Cont. (+ any questions you have!)
- What is pretraining?
- **Contextualized Word Representations:**
  - ELMo
  - BERT
  - GPT
- Finetuning and Other Transfer Methods
Basic Idea

Contextualized Word Representations
Contextualized Word Representations

Basic Idea

word embedding
Contextualized Word Representations

Basic Idea

Static

that car > hi

that car

Contextualized

that car is my car</s>

<s> that car is my car</s>
Contextualized Word Representations

Basic Idea

Static

my  car  you

Car > you

Contextualized

that  car  is  my  car  </s>

<s>  that  car  is  my  car  </s>
Contextualized Word Representations

Basic Idea

Static

that car is my car

Contextualized

that car is my car

different vector for each instance

same vector for every instance of car
Contextualized Word Representations
Static vs. Contextual Embeddings

- Recall: type vs. token
  - type = the “dictionary” instance of a word
  - toke = an individual use
  - e.g., “a brown cat and a white cat” has 7 tokens but only 5 types
- Traditional word embedding methods (e.g., word2vec) produce type-level representations
- Contextualized word embeddings (e.g., BERT, ELMo) produce token-level representations
Contextualized Word Representations

Advantages over static WEs

• Capture word sense
  • In traditional WEs, “a run along the bank of the Seine” and “after the market crash, there was bank run” would use the same vectors for “bank” and “run”, despite the obvious differences in meaning

• Can capture syntactic and semantic context
  • What should be in a static vector for “it” or “my” or “here”?
Contextualized Word Representations
Disadvantages compared to static WEs

• Variable-length sentence representations. Need more space to represent a long sentence than a short one.
• Should “my house” and “the house that is mine” really require different space to represent?
• No clear “lexicon”, which (traditionally) linguistics likes to have
  • What is the context-independent meaning of e.g., “cat”? Does it exist?
  • (Debatable how much of a disadvantage this is)
• Need more data to train
Topics

• Transformers Cont. (+ any questions you have!)
• What is pretraining?
• **Contextualized Word Representations:**
  • ELMo
  • BERT
  • GPT
• Finetuning and Other Transfer Methods
Deep contextualized word representations

Matthew E. Peters†, Mark Neumann†, Mohit Iyyer†, Matt Gardner†,
{mattewp,markn,mohiti,mattg}@allenai.org

Christopher Clark*, Kenton Lee*, Luke Zettlemoyer†*
{csquared,kentonl,lsz}@cs.washington.edu

†Allen Institute for Artificial Intelligence
*Paul G. Allen School of Computer Science & Engineering, University of Washington

https://allenai.org/allennlp/software/elmo
ELMo

- Architecture: Two-layer BiLSTM
  - "bi" = bidirectional, i.e., trained left-to-right and also right-to-left
- Layer 0 = pretrained static embeddings (Glove)
- Trained on vanilla language modeling task
- "Finetuned" by learning a simple linear combination of the learned embeddings
ELMo
Architecture

“cat”

“sat”

“on”

\[ e_0 \rightarrow h_{00} \rightarrow o_0 \]

\[ e_1 \rightarrow h_{01} \rightarrow o_1 \]

\[ e_2 \rightarrow h_{02} \rightarrow o_2 \]

\[ t_0 = \text{the} \]

\[ t_1 = \text{cat} \]

\[ t_2 = \text{sat} \]
ELMo
Architecture
ELMo
Transfer

Task-Specific Classifier
Linear Combination

“cat”
“sat”
“on”

0
1
2

0
1
2

0
1
2

0 = the
1 = cat
2 = sat
## ELMo Evaluation

<table>
<thead>
<tr>
<th>Task</th>
<th>Previous SOTA</th>
<th>Our Baseline</th>
<th>ELMo + Baseline</th>
<th>Increase (Absolute/Relative)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Liu et al. (2017)</td>
<td>84.4</td>
<td>81.1</td>
<td>85.8</td>
</tr>
<tr>
<td>SNLI</td>
<td>Chen et al. (2017)</td>
<td>88.6</td>
<td>88.0</td>
<td>88.7 ± 0.17</td>
</tr>
<tr>
<td>SRL</td>
<td>He et al. (2017)</td>
<td>81.7</td>
<td>81.4</td>
<td>84.6</td>
</tr>
<tr>
<td>Coref</td>
<td>Lee et al. (2017)</td>
<td>67.2</td>
<td>67.2</td>
<td>70.4</td>
</tr>
<tr>
<td>NER</td>
<td>Peters et al. (2017)</td>
<td>91.93 ± 0.19</td>
<td>90.15</td>
<td>92.22 ± 0.10</td>
</tr>
<tr>
<td>SST-5</td>
<td>McCann et al. (2017)</td>
<td>53.7</td>
<td>51.4</td>
<td>54.7 ± 0.5</td>
</tr>
<tr>
<td>Source</td>
<td>Nearest Neighbors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GloVe</td>
<td>playing, game, games, played, players, plays, player, Play, football, multiplayer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>biLM</td>
<td>Chico Ruiz made a spectacular play on Alusik's grounder {...}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kieffer, the only junior in the group, was commended for his ability to hit in the clutch, as well as his all-round excellent play.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Olivia De Havilland signed to do a Broadway play for Garson {...}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>{...} they were actors who had been handed fat roles in a successful play, and had talent enough to fill the roles competently, with nice understatement.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Nearest neighbors to “play” using GloVe and the context embeddings from a biLM.
Topics

- Transformers Cont. (+ any questions you have!)
- What is pretraining?
- **Contextualized Word Representations:**
  - ELMo
  - **BERT**
  - GPT
- Finetuning and Other Transfer Methods
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin  Ming-Wei Chang  Kenton Lee  Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com
BERT

- Architecture: Deep Transformer (small = 12, large = 24)
- Layer 0 = wordpiece embeddings
- Trained on masked language modeling + next-sentence prediction
- Typically fine-tuned by updating all parameters (though there are other strategies)
BERT

Big (12 or 24) stack of transformer blocks, just like we discussed last lecture.
BERT

- Architecture: Deep Transformer (small = 12, large = 24)
- Layer 0 = wordpiece embeddings
- Trained on masked language modeling + next-sentence prediction
- Typically finetuned by updating all parameters (though there are other strategies)
BERT

Masked Language Modeling

• Traditional language modeling:
  • The cat sat on ???
  • The cat sat on the ???
  • etc.
• Masked Language Modeling (MLM)
  • The cat sat on the [MASK]
  • The [MASK] sat on the mat
  • etc.
The cat sat on the mat.
He stretched and yawned and went to sleep.

BERT
Masked Language Modeling

The cat sat on the mat.
He stretched and yawned and went to sleep.


Special tokens
The cat sat on the mat.
He stretched and yawned and went to sleep.

[MASK] hides words so that they have to be predicted based on context
[SEP] = tells the model where the sentence boundary is
[CLS] = Used for the “next sentence prediction” training objective (more soon)
BERT
Masked Language Modeling

cat
the
sat
he
was
tired

[CLS] [MASK] [SEP]
BERT
Masked Language Modeling

- Choose 15% of words to me “masked”. If a word is “masked”
  - 80% of the time, replace with the token [MASK]
  - 10% of the time, replace with a random token
  - 10% of the time, do nothing
- At training time, model has to predict the masked words
- The above weirdness is so that the model learns to encode all words, since [MASK] won’t occur at test time
BERT
Next Sentence Prediction

• In addition to MLM, model has to predict whether or not the two sentences are consecutive
• Like SkipGram but for sentences
• Intended to encourage discourse structure
• 50% of time, sentence B actually follows sentence A in the corpus, 50% of the time, sentence B is a random sentence
BERT
Masked Language Modeling

True
[CLS] the [MASK] sat [SEP] he was [MASK]
tired
### BERT

#### Next Sentence Prediction

<table>
<thead>
<tr>
<th>Input</th>
<th>[CLS]</th>
<th>my</th>
<th>dog</th>
<th>is</th>
<th>cute</th>
<th>[SEP]</th>
<th>he</th>
<th>likes</th>
<th>play</th>
<th>#ing</th>
<th>[SEP]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token Embeddings</td>
<td>$E_{[CLS]}$</td>
<td>$E_{my}$</td>
<td>$E_{dog}$</td>
<td>$E_{is}$</td>
<td>$E_{cute}$</td>
<td>$E_{[SEP]}$</td>
<td>$E_{he}$</td>
<td>$E_{likes}$</td>
<td>$E_{play}$</td>
<td>$E_{#ing}$</td>
<td>$E_{[SEP]}$</td>
</tr>
<tr>
<td>Segment Embeddings</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_A$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
<td>$E_B$</td>
</tr>
<tr>
<td>Position Embeddings</td>
<td>$E_0$</td>
<td>$E_1$</td>
<td>$E_2$</td>
<td>$E_3$</td>
<td>$E_4$</td>
<td>$E_5$</td>
<td>$E_6$</td>
<td>$E_7$</td>
<td>$E_8$</td>
<td>$E_9$</td>
<td>$E_{10}$</td>
</tr>
</tbody>
</table>
BERT
Next Sentence Prediction

CLS token used to in binary NSP task.
BERT

- Architecture: Deep Transformer (small = 12, large = 24)
- Layer 0 = wordpiece embeddings
- Trained on masked language modeling + next-sentence prediction
- Typically finetuned by updating all parameters (though there are other strategies)
BERT
Finetuning

Pre-training

Fine-Tuning
# BERT Evaluation

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k/363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT$_{BASE}$</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT$_{LARGE}$</td>
<td><strong>86.7/85.9</strong></td>
<td><strong>72.1</strong></td>
<td><strong>92.7</strong></td>
<td><strong>94.9</strong></td>
<td><strong>60.5</strong></td>
<td><strong>86.5</strong></td>
<td><strong>89.3</strong></td>
<td><strong>70.1</strong></td>
<td><strong>82.1</strong></td>
</tr>
</tbody>
</table>
Topics

- Transformers Cont. (+ any questions you have!)
- What is pretraining?
- **Contextualized Word Representations:**
  - ELMo
  - BERT
  - GPT
- Finetuning and Other Transfer Methods
Improving Language Understanding by Generative Pre-Training

Alec Radford
Karthik Narasimhan
Tim Salimans
Ilya Sutskever
OpenAI
alec@openai.com

Language Models are Unsupervised Multitask Learners

Alec Radford * 1  Jeffrey Wu * 1  Rewon Child 1  David Luan 1  Dario Amodei ++ 1  Ilya Sutskever ++ 1

Language Models are Few-Shot Learners

Tom B. Brown*  Benjamin Mann*  Nick Ryder*  Melanie Subbiah*
Jared Kaplan*  Prafulla Dhariwal  Arvind Neelakantan  Pranav Shyam  Girish Sastry
Amanda Askell  Sandhini Agarwal  Ariel Herbert-Voss  Gretchen Krueger  Tom Henighan
GPT

- Architecture: Deep Transformer
- Layer 0 = BPE
- Trained on vanilla language modeling
- Notable because the recent versions (GPT-3) are HUGE, and very impressive
- Typically finetuned by updating all parameters (for the small models) or (for the large models) “prompting” (more soon)
Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.
Topics

- Transformers Cont. (+ any questions you have!)
- What is pretraining?
- **Contextualized Word Representations:**
  - ELMo
  - BERT
  - GPT
- **Finetuning and Other Transfer Methods**
Pretraining
Transferring pretrained representations

- Frozen: Just pool representations and train a new classifier on top
- Finetuning: Just treat pretraining as a good initialization. On new task, continue to update all parameters
- Parameter-Efficient or “Prompt” Tuning: Just update a small number of parameters at the bottom of the network
- Adapters: Just update a small number of parameters throughout the network
- Zero-Shot or “In-Context” Learning: Cast all tasks as an instance of the task the model was trained on (e.g., language modeling)
Pretraining

Transferring pretrained representations

- Frozen: Just pool representations and train a new classifier on top
Pretraining

Transferring frozen pretrained representations

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Transferring frozen pretrained representations

Freeze the parameters (no more updating).

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining

Transferring frozen pretrained representations

Pool some representations (e.g., final layer or CLS)

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining

Transferring frozen pretrained representations

Train a new network on top of the pooled representation (optionally, backprop through the pooled representation).

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Transferring pretrained representations

• Frozen: Just pool representations and train a new classifier on top
• Finetuning: Just treat pretraining as a good initialization. On new task, continue to update all parameters

• Parameter-Efficient or “Prompt” Tuning: Just update a small number of parameters at the bottom of the network
• Adapters: Just update a small number of parameters throughout the network

• Zero-Shot or “In-Context” Learning: Cast all tasks as an instance of the task the model was trained on (e.g., language modeling)
Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Finetuning

Adjust output layers to reflect target task

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining

Finetuning

Keep training network, backproping through everything as needed.

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Transferring pretrained representations

• Frozen: Just pool representations and train a new classifier on top
• Finetuning: Just treat pretraining as a good initialization. On new task, continue to update all parameters
• Parameter-Efficient or “Prompt” Tuning: Just update a small number of parameters at the bottom of the network
• Zero-Shot or “In-Context” Learning: Cast all tasks as an instance of the task the model was trained on (e.g., language modeling)
Pretraining
Parameter-Efficient ("Prompt") Tuning

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Parameter-Efficient ("Prompt") Tuning

Adjust output layers to reflect target task

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Parameter-Efficient ("Prompt") Tuning

Freeze everything except a small number of parameters (e.g., the embeddings)

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining

Transferring pretrained representations

- Frozen: Just pool representations and train a new classifier on top
- Finetuning: Just treat pretraining as a good initialization. On new task, continue to update all parameters
- Parameter-Efficient or “Prompt” Tuning: Just update a small number of parameters at the bottom of the network
- Adapters: Just update a small number of parameters throughout the network
- Zero-Shot or “In-Context” Learning: Cast all tasks as an instance of the task the model was trained on (e.g., language modeling)
Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining

Parameter-Efficient ("Prompt") Tuning

Add a few parameters at each layer to tune, freeze the rest.

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining

Transferring pretrained representations

• Frozen: Just pool representations and train a new classifier on top
• Finetuning: Just treat pretraining as a good initialization. On new task, continue to update all parameters
• Parameter-Efficient or “Prompt” Tuning: Just update a small number of parameters at the bottom of the network
• Adapters: Just update a small number of parameters throughout the network
• Zero-Shot or “In-Context” Learning: Cast all tasks as an instance of the task the model was trained on (e.g., language modeling)
Pretraining
Zero-Shot Transfer

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative
Pretraining
Zero-Shot Transfer

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative

Recast target task to look like language modeling...
Pretraining

Zero-Shot Transfer

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative

P(“positive”) vs. P(“negative”)

Frozen, Pretrained Model

Is the following review positive or negative?
This food sucks.

Recast target task to look like language modeling...
Pretraining
In-Context Learning

Task: Sentiment Analysis
Input: “the food sucks”
Expected: Negative

Frozen, Pretrained Model

P(“positive”) vs. P(“negative”)
Pretraining
How well do these methods work?

• In general, very well.
• Most evaluations focus on task performance, models do very well on that front
• Finetuning: Models perform better and require less data to learn the target task. Still require 100s or 1000s of examples, typically. And can result in “catastrophic forgetting”
• Parameter-Efficient/Adapters: Newer, so still being explored. But current results suggest very large models can be adapted in ~15 minutes with O(100) examples.
• Zero-Shot/In-Context Learning: More controversial. Results are high variance, depend on exact wording of the prompts. Unclear what was seen during training so hard to know how much they are truly learning and generalizing vs. parroting.