CS 371 N Lecture 10

LM2: Self-attention,
Transformers

Announcements
- A2 due
- A3 out, due in 2 weeks
- Bias in embs response due

Recap Language models:
\[ P(\overline{w}) = \prod_{i=1}^{l} P(w_i|w_1,...,w_{i-1}) \]

n-gram LMs: \[ P(\overline{w}) = \prod_{i=1}^{l} P(w_i|w_{i-n+1},...w_{i-1}) \]

Store probabilities explicitly (model as categorical distributions)

Estimation: count + normalize
Neural LMs:

predict \( w_i \mid w_1, \ldots, w_{i-1} \)

DANs? FFNNs?

Today

- RNNs and their shortcomings
- (Self)-attention

RNNs encode a sequence of vectors sequentially by tracking a "hidden state" "summary" of the sequence:

\[
P(w_i \mid w_1, \ldots, w_{i-1}) = \text{softmax}(U \tilde{h}_3)
\]
Elman network

RNN cell

\[ h_i = \tanh(W\bar{w}_i + V\bar{h}_{i-1}) \]

\( W \): d x (word emb) matrix

\( V \): d x d

\( \bar{w}_i \)

\( d \)-dim.
Key property: params don’t depend on sequence length

→ can scale to arbitrarily long inputs

However, it’s position-sensitive!

Training

\[
P(w_4 | w_1, w_2, w_3) \quad \downarrow \quad \text{loss (NLL)}
\]

backprop

updates to \( W, V \) accumulate over whole sequence

Hard to learn! ⇒ long short-term memory net (LSTM)
Shortcomings of RNNs

"Forgetfulness": hard to track information over many steps

\[ \mathbf{h}_i = \text{tanh}(\mathbf{\tilde{h}} + \mathbf{V}\mathbf{h}_{i-1}) \]

\[ \mathbf{h}_i = \mathbf{V}\mathbf{h}_{i-1} \quad h_i \approx \sum_{j=1}^{i} \mathbf{V}^{(i-j)} \mathbf{W}_j \]

LSTM: "gates" to control what parts of the vector change

O(n) sequential dependence
RNN "API"

$\overrightarrow{h_1} \overrightarrow{h_2} \overrightarrow{h_3}$

$w_1 \rightarrow w_2 \rightarrow w_3$

$\overrightarrow{h_3}$ "context-aware" word embedding for word $w_3$

$\overrightarrow{h_3}$ blends $\overrightarrow{h_2}$ (context) w/ $w_3$ (word 3)

RNN(seq of vectors) $\Rightarrow$ seq of vectors aware of context

Stack these layers
Can increase depth

Transformer: layer that contextualizes words based on other words in the sequence

\((e_1', e_2', e_3') = \text{Transformer}(e_1, e_2, e_3)\)
Running example:
Suppose we have seqs of As and Bs of length 4
if all As \rightarrow \text{next is } A
if any B \rightarrow \text{next is } B

AAAAA A predict next char
AB A A B using this sequence
B A A B B that came before

BAAAA ... A B hard for RNNs to predict
Attention allows us to do "random access" on the context to retrieve info we need.

"Souped up" DAN will add order information next time.

$K_i, C_i$

Keys: embedding of the sequence

Query: vector representing what we want to find

Assume key $A = [0]^{e_A}$, $B = [0]^{e_B}$ (word embeddings)

A A B A
Query: what we want to find

find Bs!

$\mathbf{q} = [0, 1] \ "B"$

Attention will compute a distribution over the tokens so far with higher weight for things that match $\mathbf{q}$

Steps ① Compute score for each key based on query

$$S_i = k_i \mathbf{q}$$

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0
\end{bmatrix}
\mathbf{q} = [0, 1]
\]

\[
\begin{bmatrix}
A & A & B & A
\end{bmatrix}
\]
2) Softmax scores to get probs.

\[ \bar{x} = \text{softmax}(\vec{S}) \]

\[
\begin{bmatrix}
0 & 0 & 1 & 0
\end{bmatrix}
\]

Assume \( e = 3 \)

\[
\frac{e_i}{e_0 + e_1 + e_2 + e_3} \approx \frac{1}{2}
\]

\[
\begin{bmatrix}
\frac{1}{6} & \frac{1}{6} & \frac{1}{2} & \frac{1}{6}
\end{bmatrix}
\]

3) Compute the output

output = \[ \sum x_i e_i \]

weighted sum of \( e_i \)

\[
= \frac{1}{6} \begin{bmatrix} 1 \end{bmatrix} + \frac{1}{6} \begin{bmatrix} 1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 0 \end{bmatrix} + \frac{1}{6} \begin{bmatrix} 0 \end{bmatrix}
\]

\[
= \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix}
\]
Compare to DAN:

\[
\frac{1}{q} \left[ 0 \right] + \frac{1}{q} \left[ 0 \right] + \frac{1}{q} \left[ 1 \right] + \frac{1}{q} \left[ 0 \right] = \left[ \frac{3}{4} \right]
\]

\[
\left[ \frac{1}{2} \right]
\]
weights the B more highly.

Ideally want:

\[
\left[ 0 \right]
\]
if all As √

\[
\left[ 0 \right]
\]
if any B \[ \left[ \frac{1}{2} \right] \] x

Let \( q = [0, 10] \)

\[
\text{Softmax } [0, 0, 10, 0]
\]

\[
0, 0, 1, 0 \text{ probs.}
\]
Decouple keys + queries from embeddings

Embedding matrix \( E = \begin{bmatrix}
A & 10 \\
10 & 10 \\
01 & 10
\end{bmatrix} \)

\( \text{target } e \)

\[
\overbrace{(E^T W^K)}^{K} (W^Q e) \quad B \quad e = [i]
\]

\( W^K = I \quad W^Q = 10 I \quad e = [i] \quad q = [10] \)

Parameters \( W^K \) and \( W^Q \) will let us learn how to query
Self-attention

Every word is a key and query simultaneously
do one attention computation per word ⇒ contextualized embeddings for each word

\( E: \) same embs, seq len \( \times d \)

\( K: \) same, seq len \( \times d \)

\( Q: \) seq len \( \times d \) (rather than 1 \( \times d \))
Scores $S = Q K^T$

$S_{ij} = q_i \cdot k_j$ (ith row of $Q$ and jth row of $K$)

For now: $K = E$

$Q = E$

$S$