Attention Mechanisms

Jacob Andreas / MIT 6.804–6.864 / Spring 2020
Admin

HW1 is done! Look out for survey.

HW2b will be released **tonight**.
**6.864 students only!**

Peer reviews will be assigned on OpenReview **tomorrow**. Each student will get 2 papers to review. Plan to spend ~15min / paper.
Recap: recurrent neural networks
Neural networks

\[ S = W_2^T f(W_1^T x) \]

<table>
<thead>
<tr>
<th>cheap</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>nasty</td>
<td>0</td>
</tr>
<tr>
<td>tasty</td>
<td>0</td>
</tr>
<tr>
<td>bad</td>
<td>0</td>
</tr>
<tr>
<td>good</td>
<td>1</td>
</tr>
</tbody>
</table>

Input

\[ f(W_1^T x) = h_1 \]

"Hidden layer"

\[ W_2^T h_1 = S \]

Output
Variable-sized inputs

No ordering information!

\[ \begin{bmatrix} x \\ \text{cheap} \\ \text{nasty} \\ \text{good} \\ \text{bad} \\ \text{tasty} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \text{ input} \]
Variable-sized inputs

No ordering information!

\[ x \]

\[
\begin{bmatrix}
cheap \\ nasty \\ good \\ bad \\ tasty
\end{bmatrix}
= 
\begin{bmatrix}
1 \\ 0 \\ 0 \\ 0 \\ 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
cheap & and & very & tasty
\end{bmatrix}
= 
\begin{bmatrix}
1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots 
\end{bmatrix}
\]

No fixed input dimension!
Recurrent neural networks

cheap and very tasty
Recurrent neural networks

Hidden states depend on an earlier state and an input

cheap and very tasty
Recurrent neural networks

Hidden states depend on an earlier state and an input

Same weights at every state!

cheap and very tasty
“Vanilla” RNNs

\[ h_t = f(W_{hh}h_{t-1} + W_{xh}x_i + b) \]
Gated Recurrent Units

\[ z_t = \sigma (W_z \cdot [h_{t-1}, x_t]) \]
\[ r_t = \sigma (W_r \cdot [h_{t-1}, x_t]) \]
\[ \tilde{h}_t = \tanh (W \cdot [r_t \ast h_{t-1}, x_t]) \]
\[ h_t = (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t \]
Gated Recurrent Units

\[ z_t = \sigma \left( W_z \cdot [h_{t-1}, x_t] \right) \]

\[ r_t = \sigma \left( W_r \cdot [h_{t-1}, x_t] \right) \]

\[ \tilde{h}_t = \tanh \left( W \cdot [r_t \ast h_{t-1}, x_t] \right) \]

\[ h_t = (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t \]
Long Short-Term Memory Units

[Image: Cristopher Olah]
Bidirectional RNNs
RNNs and word embeddings

cheap and very tasty
RNNs and word embeddings

one-hot vectors

cheap and very tasty

word embedding matrix
RNNs and word embeddings

learned word embeddings

one-hot vectors

cheap and very tasty

word embedding matrix
RNNs and word embeddings

pre-trained word embeddings

cheap and very tasty
Text classification

Positive
Loss $L$
class scores $s$

cheap and very tasty
Sequence labeling

Adj → $L_1$  Conj → $L_2$  Adv → $L_3$  Adj → $L_4$

$\text{cheap}$  $\text{and}$  $\text{very}$  $\text{tasty}$

$L = \sum_t L_t$

e.g.

$= - \sum_t \log p(y_t | x_t)$
Sequence labeling

\[ L = \sum_{t} L_t \]

e.g.

\[ = - \sum_{t} \log p(y_t | x_t) \]

product of indep. conditionals!
A (unidirectional) can compute $p(y_t \mid x_{:t})$.

Suppose for a sequence $x$ we set $y_t = x_{t+1}$.

Then

$$\sum_t \log p(x_{t+1} \mid x_{:t}) = p(x)$$
Language modeling: sampling

How do we sample from $p(x)$?

$x_1 \sim p(x_1 | \text{start})$

$sphinx$

\[ \text{start} \]
Language modeling: sampling

How do we sample from $p(x)$?

$x_1 \sim p(x_1 | \text{start})$

\[\text{sphinx} \rightarrow \text{start} \rightarrow \text{sphinx}\]
Language modeling: sampling

How do we sample from $p(x)$?

$x_1 \sim p(x_1 | \text{start})$  
$x_2 \sim p(x_2 | \text{sphinx})$
Language modeling: sampling

How do we sample from $p(x)$?

$x_1 \sim p(x_1 | \text{start})$  \hspace{1cm}  $x_2 \sim p(x_2 | \text{sphinx})$  \hspace{1cm}  $x_2 \sim p(x_2 | \text{sphinx of})$
SPHINX OF BLACK QUARTZ, JUDGE MY VOW
Language modeling as representation learning

Learned word embeddings

One-hot vectors

cheap and very tasty

Word embedding matrix
RNNs as Markov chains

I can train this network to predict:

$$\log p(y_t | x_{.t})$$
RNNs as Markov chains

I can train this network to predict:

$$\log p(y_t | x_{:t}) = p(q_t | O_{:t})$$

same as forward algorithm!
RNNs as Markov chains

I can train this network to predict:

\[ \log p(y_t | x_{:t}) = p(q_t | O_{:t}) \]

I can train this network to predict:

\[ \log p(y_t | x) = p(q_t | O) \]

forward-backward algo!
Sequence-to-sequence models
A dataset of math problems

One plus one equals two.

Two times two equals four.

Seven is prime.

One plus two times three equals seven.
A dataset of math problems

One plus one equals two.

Two times two equals four.

Seven is prime.

One plus two times three equals seven.

Two times three times three equals ???
Answering math problems with LMs

\[ x_1 \sim p(x_1 \mid \ldots \text{ times three equals}) \]
Answering math problems with LMs

\[ x_1 \sim p(x_1 | \ldots \text{ times three equals}) \]

twenty

equals
Answering math problems with LMs

\[ x_1 \sim p(x_1 \mid \text{... times three equals}) \]

\[ x_2 \sim p(x_2 \mid \text{... equals twenty}) \]
(don't try this at home)
A dataset of translated sentences

Caecilius est in horto. [SEP] Caecilius is in the garden.

Caecilius in horto sedet. [SEP] Caecilius sits in the garden.

Grumio est in atrio. [SEP] Grumio is in the atrium.

Grumio in atrio laborat. [SEP] ???
(try this at home!)
Idea 1: only these losses:

Sequence-to-sequence models

<table>
<thead>
<tr>
<th>in</th>
<th>horto</th>
<th>[SEP]</th>
<th>Caecilius</th>
<th>is</th>
<th>in</th>
</tr>
</thead>
</table>

Diagram showing the sequence-to-sequence model with losses highlighted in Caecilius.
Sequence-to-sequence models

Idea 2: separate encoder/decoder params

in horto [SEP] Caecilius is in
Sequence-to-sequence models

ENCODER

DECODER

in  horto  [SEP]  Caecilius  is  in
Sequence-to-sequence models

Idea 3: bidirectional encoder
Revenge of the vanishing gradients

... (many words) ...

Primo

militibus

silvanus

First

[SEP]
First

\[
\frac{\partial h_t}{\partial W_{hh}} = \sum_{i=1}^{t} (W_{hh}^T)^{t-i} h_i \\
\frac{\partial h_t}{\partial W_{xh}} = \sum_{i=1}^{t} (W_{hh}^T)^{t-i} v_i
\]

\(\hspace{1cm}\)

... (many words) ...

Primo

militibus silvanus [SEP]
Attention mechanisms
Gated Recurrent Units

\[
\begin{align*}
    z_t &= \sigma \left( W_z \cdot [h_{t-1}, x_t] \right) \\
    r_t &= \sigma \left( W_r \cdot [h_{t-1}, x_t] \right) \\
    \tilde{h}_t &= \tanh \left( W \cdot [r_t \ast h_{t-1}, x_t] \right) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]
Shortcuts
Shortcuts

Direct "copying" between hidden states makes it easy to propagate information.
Can we go farther?

Primo

First

Primo
Can we go farther?

Primo

First
Can we go farther?

Primo

First

Not super useful: no selectivity for the relevant word (since we don’t know which word is relevant when we add connections)
Can we hard-code connections?

Porta  aquam  ad  casa

Carry  water  to
Can we hard-code connections?

Carry water to

Aquam porta ad casa

Words aren't one-to-one (and order can change!)
Can we **learn** connections?
Sentence representations

This vector represents the whole sentence!
You can’t cram the meaning of a whole %&!$# sentence into a single $&!#$* vector!

[Ray Mooney, ca. 2014]
You can’t cram the meaning of a whole sentence into a single *vector*

Actually you can! (But you usually shouldn’t.)
Attention mechanisms

1. When predicting output $i$, assign a weight $\alpha_{ij}$ to each encoder state $h_j$ (weights sum to 1)
1. When predicting output \( i \), assign a weight \( \alpha_{ij} \) to each encoder state \( h_j \).

2. Compute a pooled input \( c_i = \sum_j \alpha_{ij} h_j \).
1. When predicting output $i$, assign a weight $\alpha_{ij}$ to each encoder state $h_j$

2. Compute a pooled input $c_i = \sum_j \alpha_{ij} h_j$

3. Use $c_i$ to update the decoder

Aquam | porta | ad | casa
Design decision: how to compute $\alpha_{ij}$?

1. When predicting output $i$, assign a weight $\alpha_{ij}$ to each encoder state $h_j$

$$e_{ij} = \tanh(W[h_i, h_j])$$  
[Bahdanau 2014]

$$e_{ij} = h_i^\top Wh_j$$  
[Luong 2015]

$$\alpha_{i:} = \text{softmax}(e_{i:})$$
Design decision: how to use $c_i$?

3. Use $c_i$ to update the decoder
Why does this work?

Aquam  porta  ad  casa
Why does this work?
Why does this work?

Aquam porta ad casa

MAIN VERB
INDEX 2
IMPERATIVE

SUBJECT?
IMP. VERB?

Aquam porta ad casa
Decoder hidden states are now mostly responsible for selecting what to attend to.

 Doesn't take a complex hidden state to walk monotonically through a sentence and spit out word-by-word translations.

Encoder hidden states capture contextual source word identity.

[Example from Greg Durrett]
Multi-headed attention

Look two places at once!

\[ e_{ij}^a = h_i^\top W_a h_j \]

\[ e_{ij}^b = h_i^\top W_b h_j \]

etc.
Self-attention

Attention to lower RNN layers (instead of decoder $\rightarrow$ encoder)
Non-textual attention

a desk behind
Caecilius in
Caecilius probably isn’t in the training set.
Caecilius probably isn’t in the training set.

We want the ability to generate *in* via copying and direct prediction.
Caecilius in horto

Copying

Caecilius in

$s_1$

$s_{23}$

$e_{i1}$

$e_{i2}$

$e_{i3}$

a

in

Caecilius

horto
In is double-counted: just add scores together

Caecilius  in

\[
\begin{bmatrix}
  s_1 \\
  s_{23} \\
  e_{i1} \\
  e_{i2} \\
  e_{i3}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
  s_{23} + e_{i2}
\end{bmatrix}
\]
Hard attention

1. When predicting output $i$, assign a weight $\alpha_{ij}$ to each encoder state $h_j$

$$e_{ij} = \tanh(W[h_i, h_j])$$  
[Bahdanau 2014]  

$$e_{ij} = h_i^\top Wh_j$$  
[Luong 2015]

attention  
$$\alpha_i = \text{argmax}(e_i:)$$

context repr  
$$c_i = h_{\alpha_i}$$

nondifferentiable!  
but sometimes better generalization
(now you know how to build anything)
Self-attention revisited

\[ h_{i-1}^{(2)} \quad h_{i}^{(2)} \quad h_{i+1}^{(2)} \]

\[ h_{i-1}^{(1)} \quad h_{i}^{(1)} \quad h_{i+1}^{(1)} \]
Next class: transformers