Sequence Modeling

1. Language Modeling with RNNs
2. Sequence-to-Sequence with Attention

MIT 6.S898 / Fall 2021

(many slides/images from Jacob Andreas)
The cat ___
The cat sat on the ___
Language Models

- Language Model: a probability distribution over strings in a language.

\[ p(x) \]

\[ x = x_1, x_2, \ldots, x_T \]
Language Models

- Language Model: a probability distribution over strings in a language.

\[
p(I’m \ not \ a \ cat) = 0.0000004
\]
\[
p(He \ is \ hungry) = 0.000025
\]
\[
p(Dog \ the \ asd@sdf \ 1124 \ !?) \approx 0
\]
Language Models

- Language Model: a probability distribution over strings in a language.
Language Modeling

- Language Modeling: the task of estimating this distribution from data
  - Define a statistical model $p_\theta(x)$ with parameters $\theta$
  - Given a corpus $\{x^{(1)}, \ldots, x^{(N)}\}$, maximize log likelihood

$$\theta = \arg \max_\theta \sum_{n=1}^{N} \log p_\theta \left(x^{(n)}\right)$$
Language Modeling: Motivation

You use language models everyday!
Language Modeling: History

• The Shannon game [Shannon 1951]:

How well can you predict the next character?

(1) THE ROOM WAS NOT VERY LIGHT A SMALL OBLONG
(2) ___ROO_____NOT-V____I_____SM____OBL_____ 
(1) READING LAMP ON THE DESK SHED GLOW ON
(2) REA----------D_____SHED-GLO--O--
(1) POLISHED WOOD BUT LESS ON THE SHABBY RED CARPET
(2) P-L_S-----O--BU--L_S--O-----SH-----RE--C------
Language Modeling

- Need to parameterize

\[ p_\theta(x_1, \ldots, x_T) \]

- Common strategy: left-to-right factorization via the chain rule

\[ p_\theta(x_1, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \]
Count-based Language Model

\[ p_\theta(x_1, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \]

- Idea 1: Make an n-th order Markov assumption

\[ p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \approx p_\theta(x_t \mid x_{t-2}, x_{t-1}) \]

\[ p(w \mid \text{For lunch I ate an}) \approx p(w \mid \text{ate an}) \]
Count-based Language Model

\[ p_\theta(x_1, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \]

- Idea 1: Make an n-th order Markov assumption
  \[ p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \approx p_\theta(x_t \mid x_{t-2}, x_{t-1}) \]

- Maximum likelihood:
  \[ \arg \max_{\theta} p_\theta(x_1, \ldots, x_T) \]
Count-based Language Model

\[ p_\theta(x_1, \ldots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \]

- **Idea 1**: Make an n-th order Markov assumption

\[ p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \approx p_\theta(x_t \mid x_{t-2}, x_{t-1}) \]

- **Maximum likelihood**:

\[ p_\theta(x_t \mid x_{t-2}, x_{t-1}) = \frac{\#(x_{t-2}, x_{t-1}, x_t)}{\#(x_{t-2}, x_{t-1})} \]
Neural Language Model

- Idea 2: Use a neural network over of word embeddings

\[
\text{Number of dimensions } = |V| \\
\begin{bmatrix}
0 \\
1 \\
0 \\
0 \\
\end{bmatrix}
\]

1 if \( x_{t-1} = \text{“the”} \)
Neural Language Model

- Idea 2: Use a neural network over of word embeddings

\[ W \begin{bmatrix} 0 \\ 1 \\ 0 \\ . \\ 0 \end{bmatrix} = \text{word embedding of “the”} \]

\[ d \times |V| \text{ input embedding matrix} \]
Neural Language Model

- Idea 2: Use a neural network over word embeddings

\[
\begin{align*}
    h &= [W x_{t-1}, W x_{t-2}] \\
    s &= U f(h) \\
    p_\theta(x_t | x_{t-2}, x_{t-1}) &= \text{softmax}(s) x_t = \frac{\exp(s_{x_t})}{\sum_{v \in V} \exp(s_v)}
\end{align*}
\]
Neural Language Model

- Idea 2: Use a neural network over of word embeddings

- Exact MLE intractable $\Rightarrow$ gradient ascent on log likelihood

\[
\theta = \{U, W\}
\]

\[
\theta \leftarrow \theta + \alpha \nabla_{\theta} \log p_{\theta}(x)
\]
Neural Language Model

“A Neural Probabilistic Language Model” [Bengio et al. 2003]
“The CEO who testified before the senators last Monday was from Bulgaria”

“The CEOs who testified before the senators last Monday were from Bulgaria”
Feed-forward neural language models cannot model long-range dependencies.

Problem: How can we encode variable-sized input $x_1, \ldots, x_t$ into fixed dimensional vector so we can apply $s = U f(h)$?

Summing, max-pooling...? Recurrence?

“**The CEO who testified before the senators last Monday was** from Bulgaria”

“The **CEOs who testified before the senators last Monday were** from Bulgaria”
Recurrent Neural Networks

- Hidden state is a function of previous hidden state and current input.
Recurrent Neural Networks

- Hidden state is a function of previous hidden state and current input.

- Same weights at each state!

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

`cheap and very tasty`
Recurrent Neural Networks

\[ h_t = f(W_{hh}h_{t-1} + W_{xh}x_t) \]
Recurrent Neural Networks

\[ h_t = f(W_{hh}h_{t-1} + W_{xh}x_t) \]
\[ s = U h_t \]
\[ p_\theta(x_{t+1} | x_1, \ldots, x_t) = \text{softmax}(s)x_{t+1} \]
RNN Language Model

No Markov assumption!

cheap and very tasty
Recurrent Neural Networks

\[ W_{hh} = \begin{pmatrix} 0 & I_{d \times d} \\ 0 & 0 \end{pmatrix} \]

2d × 2d block matrix

\[ W_{xh} = \begin{pmatrix} W \\ 0 \end{pmatrix} \]

2d × V matrix

\[ f = \text{identity} \]

What is \[ h_t = f(W_{hh} h_{t-1} + W_{xh} x_t) \]?
Recurrent Neural Networks

\[ W_{hh} = \begin{pmatrix} 0 & I_{d \times d} \\ 0 & 0 \end{pmatrix} \quad \text{block matrix} \]

\[ W_{xh} = \begin{pmatrix} W \\ 0 \end{pmatrix} \quad \text{matrix} \]

\[ f = \text{identity} \]

\[ h_t = f(W_{hh} h_{t-1} + W_{xh} x_t) = [W x_t, W x_{t-1}] \]
Recurrent Neural Networks

\[
W_{hh} = \begin{pmatrix}
0 & I_{d \times d} \\
0 & 0
\end{pmatrix} \quad 2d \times 2d \text{ block matrix}
\]

\[
W_{xh} = \begin{pmatrix}
W \\
0
\end{pmatrix} \quad 2d \times V \text{ matrix}
\]

RNN LMs (unsurprisingly) generalize feedforward LMs

\[
h_t = f(W_{hh} h_{t-1} + W_{xh} x_t) = \left[ W x_t, W x_{t-1} \right]
\]

\( f = \text{identity} \)
RNN Language Models: Training

cheap and very
RNN Language Models: Training

\[ L_t = -\log p_\theta(x_t | x_1, \ldots, x_{t-1}) \]
RNN Language Models: Training

\[ L_t = -\log p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \]

\[ L = -\log p_\theta(x) = \sum_{t=1}^{T} L_t \]
RNN Language Models: Training

\[ L_t = -\log p_\theta(x_t | x_1, \ldots, x_{t-1}) \]

\[ L = -\log p_\theta(x) = \sum_{t=1}^{T} L_t \]

Obtain \( \nabla_\theta \) via backpropagation as usual (“backpropagation through time”)

\[ \theta = \{ U, W_{hh}, W_{xh} \} \]

\[ \theta \leftarrow \theta + \alpha \nabla_\theta \log p_\theta(x) \]
RNN Language Models

- RNNs can *theoretically* model infinite history... what about in practice?
RNN Language Models: Training

\[ \frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_{hh}} \]

\[ L_t = -\log p_\theta(x_t \mid x_1, \ldots, x_{t-1}) \]

\[ L = -\log p_\theta(x) = \sum_{t=1}^{T} L_t \]
RNN Language Models: Training

\[
\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_{hh}}
\]

\[
\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}}
\]

\[
L_t = - \log p_\theta(x_t | x_1, \ldots, x_{t-1})
\]

\[
L = - \log p_\theta(x) = \sum_{t=1}^{T} L_t
\]
RNN Language Models: Training

\[
\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_{hh}}
\]

\[
\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}} + \frac{\partial L_t}{\partial h_t} \frac{\partial h_{t-1}}{\partial W_{hh}} + \frac{\partial L_t}{\partial h_{t-1}} \frac{\partial h_{t-2}}{\partial W_{hh}} + \ldots
\]

\[
= \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}
\]
RNN Language Models: Training

\[
\frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W_{hh}}
\]

\[
\frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial W_{hh}} + \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_{hh}} + \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_{hh}} + \ldots
\]

\[
= \sum_{k=1}^{t} \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_{hh}}
\]

\[
\frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \ldots \frac{\partial h_{k+1}}{\partial h_k}
\]
RNN Language Models: Training

\[
\frac{\partial h_t}{\partial h_{t-1}} = \text{diag}(f'(W_{hh}h_{t-1} + W_{xh}x_t))W_{hh}
\]
\[
\frac{\partial h_t}{\partial h_{t-1}} = \text{diag}(f'(W_{hh}h_{t-1} + W_{xh}x_t))W_{hh}
\]

I if \( f = \text{identity} \)
RNN Language Models: Training

\[ \frac{\partial h_t}{\partial h_{t-1}} = W_{hh} \]
RNN Language Models: Training

\[ \frac{\partial h_t}{\partial h_{t-1}} = W_{hh} \]

\[ \frac{\partial h_t}{\partial h_k} = \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial h_{t-3}} \ldots \frac{\partial h_{k+1}}{\partial h_k} \]

\[ \frac{\partial L_t}{\partial h_k} = \frac{\partial L_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \]

\[ = \frac{\partial L_t}{\partial h_t} W_{hh}^{t-k} \]
RNN Language Models: Training

\[
\frac{\partial h_t}{\partial h_{t-1}} = W_{hh} \quad \frac{\partial h_t}{\partial h_k} = W_{hh}^{t-k} \quad \frac{\partial L_t}{\partial h_k} = \frac{\partial L_t}{\partial h_t} W_{hh}^{t-k}
\]

- If largest eigenvalue > 1, gradients explode
  < 1, gradients vanish

- Many simplifying assumptions (diagonalizability, nonzero eigenvalues, linear transition, etc.), but results hold even if assumptions are relaxed [Pascanu '2013]
RNN Language Models: Training

- Exploding gradients

\[ \theta \leftarrow \theta + \alpha \nabla_{\theta} \log p_{\theta}(x) \]

- Solution: gradient norm clipping

```
Algorithm 1 Pseudo-code for norm clipping

\[ \hat{g} \leftarrow \frac{\partial \mathcal{L}}{\partial \theta} \]

if \( \|\hat{g}\| \geq \text{threshold} \) then

\[ \hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g} \]

end if
```

RNN Language Models: Training

- Vanishing gradients
RNN Language Models: Training

- Vanishing gradients

\[
\left| \frac{\partial L_{\text{was}}}{\partial h_{\text{CEO}}} \right| << \left| \frac{\partial L_{\text{testified}}}{\partial h_{\text{CEO}}} \right|
\]

\[\implies \text{most of the gradient information is from short range dependencies}\]
• Vanilla RNN: only way for hidden state at time $t$ to affect later layers is through the “bottleneck” transition layer

$$h_t = f(W_{hh} h_{t-1} + W_{xh} x_t)$$
RNN Variants

- Vanilla RNN: only way for hidden state at time t to affect later layers is through the “bottleneck” transition layer

\[ h_t = f(W_{hh} h_{t-1} + W_{xh} x_t) \]

- Variant: What if we had more “direct” connections?

\[ \tilde{h}_t = f(W_{hh} h_{t-1} + W_{xh} x_t) \]

\[ h_t = (1 - \alpha) h_{t-1} + \alpha \tilde{h}_t \quad \alpha \in (0, 1) \]
RNN Variants

• What if we allowed to $\alpha$ vary for each time step, and learned this?
RNN Variants

- What if we allowed $\alpha$ to vary for each time step, and learned this?

Update gate: decides how much each dimension of the hidden state gets preserved

$u_t = \sigma(U_{hh}h_{t-1} + U_{xh}x_t)$  \hspace{1cm} u_t \in (0, 1)^d$

$\tilde{h}_t = f(W_{hh}h_{t-1} + W_{xh}x_t)$

$h_t = (1 - u_t) \circ h_{t-1} + u_t \circ \tilde{h}_t$
RNN Variants

- What if we did only used a part of the previous hidden state to get $\tilde{h}_t$?

Update gate: decides how much each dimension of the hidden state gets preserved

$u_t = \sigma(U_{hh}h_{t-1} + U_{xh}x_t)$  \hspace{1cm} u_t \in (0, 1)^d$

Reset gate: how much of the previous hidden state is used to get the new content state

$r_t = \sigma(R_{hh}h_{t-1} + R_{xh}x_t)$  \hspace{1cm} r_t \in (0, 1)^d$

$\tilde{h}_t = f(W_{hh}(r_t \circ h_{t-1}) + W_{xh}x_t)$

$h_t = (1 - u_t) \circ h_{t-1} + u_t \circ \tilde{h}_t$
RNN Variants

Gated Recurrent Unit (GRU) [Chung et al. 2014, Cho et al. 2014]

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

[Image: Cristopher Olah]
Another Variant

Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber 1997]
Another Variant

**Forget gate**: controls what is kept vs forgotten, from previous cell state

**Input gate**: controls what parts of the new cell content are written to cell

**Output gate**: controls what parts of cell are output to hidden state

**New cell content**: this is the new content to be written to the cell

**Cell state**: erase ("forget") some content from last cell state, and write ("input") some new cell content

**Hidden state**: read ("output") some content from the cell

**Sigmoid function**: all gate values are between 0 and 1

\[
\begin{align*}
    f(t) &= \sigma \left( W_f h^{(t-1)} + U_f x^{(t)} + b_f \right) \\
    i(t) &= \sigma \left( W_i h^{(t-1)} + U_i x^{(t)} + b_i \right) \\
    o(t) &= \sigma \left( W_o h^{(t-1)} + U_o x^{(t)} + b_o \right) \\
    c(t) &= f(t) \odot c^{(t-1)} + i(t) \odot \tilde{c}(t) \\
    h(t) &= o(t) \odot \tanh(c(t))
\end{align*}
\]

All these are vectors of same length \( n \)

Gates are applied using element-wise product

[Image: Stanford CS224]
“Skip” Connections

● Vanishing gradients is not just a problem with RNNs!

● Skip connections can allow for models with 100s of layers.

● Residual Networks (ResNets) [He et al. 2015]: Ubiquitous in computer vision systems.

Figure 2. Residual learning: a building block.
[Image: He et al. 2015]
RNN Language Models: Sampling

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

$x_1 \sim p(x_1 | \text{start})$  
$x_2 \sim p(x_2 | \text{sphinx})$  
$x_2 \sim p(x_2 | \text{sphinx of})$
What can language models do?

One plus one equals two

Two times two equals four

Ten plus thirteen equals twenty three
What can language models do?

One plus one equals two

Two times two equals four

Ten plus thirteen equals twenty three

Three times three times three equals ???
A language model for math

\[ x_1 \sim p(x_1 | \ldots \text{ times three equals}) \]
A language model for math

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

twenty

equals
A language model for math

\[ x_1 \sim p(x_1 | \text{... times three equals}) \quad x_2 \sim p(x_2 | \text{... equals twenty}) \]
I’m going to build a system to solve arithmetic problems. Not with a language model though, right? Right?
One plus one equals two
Two times two equals four
Ten plus thirteen equals twenty three
Three times three times three equals
One plus one equals two
Two times two equals four
Ten plus thirteen equals twenty three
Three times three times three equals 27
Four times four equals sixteen
Twelve plus twenty six equals thirty eight
One plus one equals two
Two times two equals four
Ten plus thirteen equals twenty three
Three times three times three equals 27
Four times four equals sixteen
Twelve plus twenty six equals thirty eight

Please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please, please
1 + 1 = 2
2 * 2 = 4
10 + 13 = 23
3 * 3 * 3 = 27
2 + 3 = 5
6 * 3 = 18
5 + 4 = 9
1 + 2 = 3
3 * 5 = 15
4 + 5 = 9
1 + 9 = 10
Language models as “few-shot” learners

- Try it out for yourself!
  

- Companies starting to provide language model as a service.
A dataset of math problems

One plus one equals two

Two times two equals four

Ten plus thirteen equals twenty three

Three times three times three times three equals ???
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] ???
Sequence-to-sequence models

Encoder

Decoder

Skip connections

Convolutions

Deconvolutions
Sequence-to-sequence models
Sequence-to-sequence models

\[
\text{in} \rightarrow \text{horto} \rightarrow [\text{SEP}] \rightarrow \text{Caecilius} \rightarrow \text{is} \rightarrow \text{in}
\]
Vanishing gradients

... (many words) ...

Primo

militibus

silvanus

[SEP]

First
Vanishing gradients

Gated RNNs help with vanishing gradients, but not a panacea.
Communication bottleneck

This vector has to represent the entire sentence!
Skip connections?

Primo

[SEP]
Skip connections?

$$c = \frac{1}{S} \sum_{j=1}^{S} q_j$$

$$p_\theta(\cdot | x, y_{\leq i}) = \text{softmax}(W[h_i, c])$$
Skip connections?

\[ c = \frac{1}{S} \sum_{j=1}^{S} q_j \]

\[ p_{\theta}(\cdot | x, y_{\leq i}) = \text{softmax}(W[h_i, c]) \]

Not too useful. No selectivity for relevant words.
Hard connections?

\[ c_i = q_i \]

\[ p_\theta(\cdot \mid x, y_{\leq i}) = \text{softmax} \left( W[h_i, c_i] \right) \]
Hard connections?

$c_i = q_i$

$p_{\theta}(\cdot | x, y_{\leq i}) = \text{softmax}(W[h_i, c_i])$
Hard connections?

\[
c_i = \sum_{j=1}^{J} \alpha_{ij} q_j \quad \alpha_{ij} = 1, \quad i = j
\]
\[
= 0, \quad i \neq j
\]
\[p_\theta(\cdot | x, y_{\leq i}) = \text{softmax} \left( W [h_i, c_i] \right)\]
Hard connections?

Words aren’t one-to-one and order can change.

Aquam -> porta -> ad -> casa -> [SEP]
Can we *learn* these connections?
Can we learn these connections?

\[ e_{ij} = h_i^\top q_j, \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{S} \exp(e_{ik})} \]

“Attention distribution”

\( \alpha_{i1} \quad 0.1 \quad \alpha_{i2} \quad 0.7 \quad \alpha_{i3} \quad 0.1 \quad \alpha_{i4} \quad 0.1 \)

Aquam \quad porta \quad ad \quad casa \quad [SEP]
Can we *learn* these connections?

\[ e_{ij} = h_i^T q_j, \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{S} \exp(e_{ik})} \]

\[ c_i = \sum_{j=1}^{S} \alpha_{ij} q_j \]

“Attention distribution”

“Context vector”

Aquam  porta  ad  casa

\[ i + 1 \]
Can we learn these connections?

\[ e_{ij} = h_i^\top q_j, \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{S} \exp(e_{ik})} \]

\[ c_i = \sum_{j=1}^{S} \alpha_{ij} q_j \quad \text{“Context vector”} \]

“Attention distribution”

\[ p_\theta(\cdot \mid x, y \leq i) = \text{softmax} \left( W[h_i, c_i] \right) \]

Aquam \quad porta \quad ad \quad casa
Different similarity functions

\[
\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{S} \exp(e_{ik})}
\]

\[
e_{ij} = h_i^T q_j \quad \text{Dot product}
\]

\[
e_{ij} = h_i^T U q_j \quad \text{Bilinear [Luong et al. 2015]}
\]

\[
e_{ij} = u^T \tanh(U_1 h_i + U_2 q_j) \quad \text{MLP [Bahdanau et al. 2015]}
\]
Attention recap
Attention recap
Attention recap
Attention recap

Over the line!

<s>
Attention recap
Attention recap
Attention recap

Over the line!

<s> Çizgiyi geçtin! </s>
Why should this work?

Communication bottleneck

Aquam porta
Aquam porta
Aquam porta
Aquam porta

ad
ad
casa
Why should this work?
Why should this work?
Learned attention is meaningful!

[Bahdanau et al. 2015]
Learned attention is meaningful?

Attention is not Explanation

Sarthak Jain  
Northeastern University

Byron C. Wallace  
Northeastern University

SST

Original: reggio falls victim to relying on the very digital technology that he fervently scorns creating a meandering inarticulate and ultimately disappointing film

Adversarial: reggio falls victim to relying on the very digital technology that he fervently scorns creating a meandering inarticulate and ultimately disappointing film $\Delta \hat{j} : 0.005$
Learned attention is meaningful?

**Attention is not Explanation**

Sarthak Jain  
Northeastern University

Byron C. Wallace  
Northeastern University

**Attention is not not Explanation**

Sarah Wiegrefe*  
School of Interactive Computing  
Georgia Institute of Technology

Yuval Pinter*  
School of Interactive Computing  
Georgia Institute of Technology
Decoding

- During training \((x, y)\) is given so we can train as a conditional language model ("teacher forcing"): 
  \[
  \log p_\theta(y \mid x) = \sum_{t=1}^{T} \log p_\theta(y_t \mid x, y_{<t})
  \]
- During testing, only \(x\) is given.
- How to decode from the learned model?
Decoding

- Sampling: not so good for accuracy

\[ y \sim p_\theta(y \mid x) \]

\[ y_t \sim p_\theta(y_t \mid x, y_{<t}) \]

- Argmax decoding:

\[ \arg \max_y p_\theta(y \mid x) \]
Argmax decoding

\[ \arg \max_{y \in \mathcal{V}^T} p_\theta(y \mid x) \]

- All possible sentences of length up to $T$!

(Completely intractable)
Approximate argmax decoding

\[
\arg \max_{y \in \mathcal{V}^T} p_\theta(y \mid x)
\]

- Greedy decoding:

\[
\hat{y}_t = \arg \max_{w \in \mathcal{V}} p_\theta(w \mid x, y_{<t})
\]
Greedy decoding issue

\[ \hat{y}_t = \arg \max_{w \in V} p_{\theta}(w \mid x, y_{<t}) \]

I’m not a cat          Je ne suis pas ???
Greedy decoding issue

\[ \hat{y}_t = \arg \max_{w \in V} p_\theta(w \mid x, y_{<t}) \]

I’m not a cat Je ne suis pas une
Greedy decoding issue

\[ \hat{y}_t = \arg \max_{w \in V} p_\theta(w \mid x, y_{<t}) \]

I’m not a cat       Je ne suis pas une chat
Greedy decoding issue

\[ \hat{y}_t = \arg \max_{w \in V} p_\theta(w \mid x, y_{<t}) \]

I'm not a cat  Je ne suis pas une chat

- Unlike English, French has gendered nouns.

- If during decoding the model incorrectly translated \textit{a} to a feminine indefinite article \textit{une}, then we are stuck with this decision with greedy decoding.
Beam search

- Idea: maintain a *beam* of K partial hypotheses during decoding.

- At each step, expand all hypotheses to the next step, then take the top K to maintain a beam size K.
Beam search

\[ <s> \]

I'm not a cat
Beam search

I’m not a cat
Beam search

Get top K elements from $\log p_\theta(y_1 = \text{Je} | x = \text{I'm not a cat})$

$\log p_\theta(y_1 = \text{Tu} | x = \text{I'm not a cat})$
Beam search

<s>

-0.7
Je

-1.2
Tu
Beam search

\[
\begin{align*}
\langle s \rangle & \rightarrow \text{Je} \quad \text{(score: -0.7)} \\
\langle s \rangle & \rightarrow \text{Tu} \quad \text{(score: -1.2)}
\end{align*}
\]
Get top K elements from each hypothesis on the beam
Beam search

Get top K elements from each hypothesis on the beam
Add local score to get a score for each hypothesis
Beam search

Get top-K scores from each of the $K^2$ hypotheses
Je ne suis pas un pas

<s> Expand again from the beam
At each time step, we only consider $K^2$ hypotheses.
Beam search

Each path represents a partial hypothesis
Je ne suis pas un chat -7.6

Tu n'as pas une -7.5

Tu n'es pas un chien -8.0

Tu n’es pas un chat -8.1

Tu n’es pas un chien -8.3

Tu n’es pas un chat -8.7
Je ne suis pas un chat -7.6
-7.5 une

Tu n'es pas un chien -8.0
-8.1 chien
-8.3 chat

Beam search
Je ne suis pas chat ou chien.

Tu n’es pas un chat ou chien.
Je ne suis pas un chat.

Tu n’es pas un chien.
Beam search

• Complexity

Greedy: \( O(TV) \)

Beam: \( O(TK(V + K \log K)) = O(TKV) \)
Beam search

- Complexity

  Greedy: $O(TV)$

  Beam: $O(TK(V + K \log K)) = O(TKV)$

- Generally applicable strategy for search:

  $$\text{score}(y_1, \ldots, y_t) = \log p_\theta(y_1, \ldots, y_t | x)$$
History: Seq2Seq for Machine Translation

Now ~30

MT Quality (BLEU)

[Image: Christopher Manning]
Application: Summarization

[Image of a diagram with phrases and their corresponding attention scores]

[Rush et al. 2015]
Application: Image Captioning

A dog is standing on a hardwood floor.

A woman holding a clock in her hand.

A group of people sitting on a boat in the water.

A woman is sitting at a table with a large pizza.

[Xu et al. 2015]
Application: Speech Recognition

FDHC0_SX209: Michael colored the bedroom wall with crayons.

[Chorowski et al. 2015]
Application: Gameplaying
Application: Pretty much everything

- Prior to attention, everything had to be forced into a fixed dimensional bottleneck layer.

- Representing the input with a set with vectors and attending over them is an incredibly powerful idea!
Every step of this computation depends on the previous one. We can’t parallelize it!
Transformers

Main idea: route all information via attention mechanisms

via attention

via the RNN state
Transformers

Before:

\[ h_t = f(x_t, h_{t-1}, \text{att}(H \mid h_{t-1})) \]
Transformers

Before:
\[ h_t = f(x_t, h_{t-1}, \text{att}(H \mid h_{t-1})) \]

Now:
\[ h_t = f(x_t, \text{att}(X \mid x_t)) \]
Transformers

Before:

\[ h_t = f(x_t, h_{t-1}, \text{att}(h \mid h_{t-1})) \]

Now:

\[ h_t = f(x_t, \text{att}(X \mid x_t)) \]

\[ h^i_t = f(h^{i-1}_t, \text{att}(H^{i-1} \mid h^{i-1}_t)) \]
Transformers in detail

\[ Q^i = W_Q H^i \]
\[ k_t^i = W_K h_t^i \]
\[ a_t^i = \frac{Q^i k_t^i T}{\sqrt{d}} \]
\[ \alpha_t^i = \text{softmax}(a_t^i) \]
\[ V^i = W_V H^i \]
\[ o_t^i = \alpha_t^i V^i \]
Transformer layer in Matrix Notation

\[ Q^{i-1} = W_Q H^{i-1} \]
\[ k_t^{i-1} = W_K h_t^{i-1} \]
\[ a_t^{i-1} = \frac{Q_t^{i} k_t^{i \top}}{\sqrt{d}} \]
\[ \alpha_t^i = \text{softmax}(a_t^i) \]
\[ V^{i-1} = W_V H^{i-1} \]
\[ h_t^{i+1} = \alpha^i V^i \]

**self-attention**

\[ Q^i = W_Q H^i \]
\[ K_t^i = W_K H_t^i \]
\[ V_t^i = W_V H_t^i \]
\[ H_t^{i+1} = \left( \frac{\text{softmax}(Q^i K_t^{i \top})}{\sqrt{d}} \right) V^i \]
Transformer Parallelism

RNN

dependency length: # words

Transformer

dependency length: # layers
Multi-layer Transformers

\[ Y \]

\[ \text{self-attention} \]

\[ \text{self-attention} \]

\[ \text{self-attention} \]

\[ X \]
Multi-head Attention

\[ h_i^j = f(h_t^{i-1}, \text{att}(h_t^{i-1} | h_t^{i-1}), \text{att}'(h_t^{i-1} | h_t^{i-1})) \]
Multi-head Attention

Y

self-attention-2

self-attention-1

self-attention-2

self-attention-1

X
Multi-head Attention

\[ H^{i+1} = \text{self-attention}(H^i) \]

\[ H^{i+1} = [\text{self-att-1}(H^i), \ldots, \text{self-att-n}(H^i)]W_O + H^i \]
How do we know what to attend to first?
Inject “positional embeddings” that encode loc. in input.

index(0) + $h_1^1$
Pat

index(1) + $h_2^1$
loves

index(2) + $h_3^1$
Sal

index(3) + $h_4^1$
???
Transformer Encoder

cheap
and
very
tasty
Transformer Decoder

$p(\text{and})$  $p(\text{very})$  $p(\text{tasty})$

cheap  and  very
Transformer Attention Masking

Idea: when training a decoder, only let tokens attend to the left. Compute loss at all positions in parallel.

\[
H^{i+1} = \left( \frac{\text{softmax}(Q^i K^{i\top} + M)}{\sqrt{d}} \right) V^i
\]
But in sequence-to-sequence problems, we want to let all source tokens look at all other source tokens (but target tokens only look backward).
Transformers for Language Modeling

[https://paperswithcode.com/sota/language-modelling-on-wikitext-103]
Transformer for Machine Translation

Analyzing attention heads

Head 8-10
- Direct objects attend to their verbs
- 86.8% accuracy at the dobj relation

Head 8-11
- Noun modifiers (e.g., determiners) attend to their noun
- 94.3% accuracy at the det relation

Head 7-8
- Possessive pronouns and apostrophes attend to the head of the corresponding NP
- 80.5% accuracy at the poss relation

[Clark et al. 2019. What does BERT look at?]
Summary

- **Language modeling:**
  - Count-based LMs
  - Feedforward LMs
  - RNN
  - GRU/LSTM

- **Data sparsity**
- **Variable length input**
- **Vanishing gradients**

- **Sequence-to-Sequence learning:**
  - Language Model
  - Encoder-Decoder
  - Attention
  - Transformers

- **Conditional language model**
- **Fixed dimensional bottleneck**
- **Parallelizability**