Today's Lecture

so far

- we discussed RNNs as encoder and decoder
- we discussed some architecture variants:
  - RNN vs. GRU vs. LSTM
  - attention mechanisms

today

- some important components of neural MT architectures:
  - dropout
  - layer normalization
  - deep networks
  - non-recurrent architectures:
    - convolutional networks
    - self-attentional networks

1. General Architecture Variants
2. NMT with Convolutional Neural Networks
3. NMT with Self-Attention

Dropout

- wacky idea: randomly set hidden states to 0 during training
- motivation: prevent "co-adaptation" of hidden units
  → better generalization, less overfitting

(a) Standard Neural Net
(b) After applying Dropout

[Srivastava et al., 2014]
Dropout

- implementation:
  - for training, multiply layer with "dropout mask"
  - randomly sample new mask for each layer and training example
  - hyperparameter $p$: probability that state is retained
    (some tools use $p$ as probability that state is dropped)
  - at test time, don’t apply dropout, but re-scale layer with $p$ to ensure expected output is the same
  - (you can also re-scale by $1/p$ at training time instead)

![Standard Neural Net](image1)

![After applying dropout](image2)

Dropout and RNNs

for recurrent connections, applying dropout at every time step blocks information flow

solution 1: only apply dropout to feedforward connections

![Regularized multilayer RNN](image3)

solution 2: variational dropout: use same dropout mask at each time step

Layer Normalization

- if input distribution to NN layer changes, parameters need to adapt to this covariate shift

- especially bad: RNN state grows/shrinks as we go through sequence

- normalization of layers reduces shift, and improves training stability

- re-center and re-scale each layer $a$ (with $H$ units)

- two bias parameters, $g$ and $b$, restore original representation power

\[
\begin{align*}
\mu &= \frac{1}{H} \sum_{i=1}^{H} a_i \\
\sigma &= \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i - \mu)^2} \\
h &= \frac{g}{\sigma} \odot (a - \mu) + b
\end{align*}
\]
Deep Networks

- Increasing model depth often increases model performance
- Example: stack RNN:
  \[ h_{i,1} = g(U_1 h_{i-1,1} + W_1 x_i) \]
  \[ h_{i,2} = g(U_2 h_{i-1,2} + W_2 h_{i,1}) \]
  \[ h_{i,3} = g(U_3 h_{i-1,3} + W_3 h_{i,2}) \]

Layer Normalization and Deep Models: Results from UEDIN@WMT17

- Layer normalization and deep models generally improve quality
- Layer normalization also speeds up convergence when training (fewer updates needed)
- Dropout used for low-resource system (TR→EN)

<table>
<thead>
<tr>
<th>system</th>
<th>CS→EN 2017</th>
<th>DE→EN 2017</th>
<th>LV→EN 2017</th>
<th>RU→EN 2017</th>
<th>TR→EN 2017</th>
<th>ZH→EN 2017</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>27.5</td>
<td>32.0</td>
<td>16.4</td>
<td>31.3</td>
<td>19.7</td>
<td>21.7</td>
</tr>
<tr>
<td>+layer normalization</td>
<td>28.2</td>
<td>32.1</td>
<td>17.0</td>
<td>32.3</td>
<td>18.8</td>
<td>22.5</td>
</tr>
<tr>
<td>+deep model</td>
<td>28.9</td>
<td>33.5</td>
<td>16.6</td>
<td>32.7</td>
<td>20.6</td>
<td>22.9</td>
</tr>
</tbody>
</table>

MT – 2018 – 10

1. General Architecture Variants
2. NMT with Convolutional Neural Networks
3. NMT with Self-Attention
Convolutional Networks

**core idea:** rather than using fully connected matrix between two layers, repeatedly compute dot product with small *filter* (or *kernel*)

2d convolution with 3x3 kernel

- `0 1 1 0 0 0`  
- `0 0 1 1 0 0`  
- `0 0 0 1 1 0`  
- `0 1 0 0 1 0`  
- `0 1 1 0 0 1`  
- `1 1 0 0 0 0`

I  
K  
I + K

---

**when working with sequences, we often use 1d convolutions**

1d convolution with width-3 kernel

- Keep in mind we are working with 100 dimensions although here we depicted just one for simplicity
- The length result of the convolution is well known to be:  `seq_length - width + 1 = 6 - 3 + 1 = 6`
- So the output matrix will be `6 x 100`
- because there was no padding

---

Convolutional Neural Machine Translation

**convolutional encoder actually predates RNN encoder**

While the just-described convolutional neural machine translation model helped to set the scene for neural network approaches for machine translation, it could not be demonstrated to achieve competitive results compared to traditional approaches. The compression of the sentence representation into a single vector is especially a problem for long sentences. However, the model was used successfully in reranking candidate translations generated by traditional statistical machine translation systems.

---

**Convolutional Networks**

- **2d convolution with 3x3 kernel**
- ![2d convolution with 3x3 kernel](image)

**Convolutional Neural Machine Translation**

- **convolutional encoder actually predates RNN encoder**
- ![Convolutional Neural Machine Translation](image)
• to keep representation size constant, use padding
  → similar variable-size representation as RNN encoder
• kernel can be applied to all windows in parallel

- use your favourite attention mechanism to obtain input context
- in decoder, information from future tokens is masked during training
- effective context window depends on network depth and kernel size

Convolutional Neural Machine Translation (ByteNet)

1 General Architecture Variants
2 NMT with Convolutional Neural Networks
3 NMT with Self-Attention
3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of identical layers. In addition to the two sub-layers, there are extremely small gradients in the sub-layers. To counteract this effect, we scale the dot products by \( \frac{1}{\sqrt{d_k}} \).

Multi-head attention: use \( K \) key here: attend over previous layer of deep network.

3.2 Attention

Multi-Head Attention

- basic attention mechanism in AIAYN: Scaled Dot-Product Attention
  \[
  \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V
  \]
- query \( Q \) is decoder/encoder state (for attention/self-attention)
- key \( K \) and value \( V \) are encoder hidden states
- multi-head attention: use \( h \) parallel attention mechanisms with low-dimensional, learned projections of \( Q, K, \) and \( V \)

Transformer architecture

- stack of \( N \) self-attention layers
- self-attention in decoder is masked
- decoder also attends to encoder states
- \textbf{Add & Norm}: residual connection and layer normalization
Comparison

**Empirical comparison difficult**
- some components could be mix-and-matched
  - choice of attention mechanism
  - choice of positional encoding
  - hyperparameters and training tricks
- different test sets and/or evaluation scripts

**SOCKEYE [Hieber et al., 2017] (EN-DE; newstest2017)**

<table>
<thead>
<tr>
<th>system</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>deep LSTM</td>
<td>25.6</td>
</tr>
<tr>
<td>Convolutional</td>
<td>24.6</td>
</tr>
<tr>
<td>Transformer</td>
<td>27.5</td>
</tr>
</tbody>
</table>

**Marian (EN-DE; newstest2016)**

<table>
<thead>
<tr>
<th>system</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>deep LSTM</td>
<td>32.6</td>
</tr>
<tr>
<td>Transformer</td>
<td>33.4</td>
</tr>
</tbody>
</table>

Empiricism vs. Theory

- our theoretical understanding of neural networks lags behind empirical progress
- there are some theoretical arguments why architectures work well...
  (e.g. self-attention reduces distance in network between words)
- …but these are very speculative

Further Reading

- required reading: Koehn, 13.7
- consider original literature cited on relevant slides
<table>
<thead>
<tr>
<th>Bibliography I</th>
<th>Bibliography II</th>
</tr>
</thead>
</table>
A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.  
Decoding with Large Scale Neural Language Models Improves Translation.  
Convolutional Sequence to Sequence Learning.  
Recurrent Neural Network Regularization.  
CoRR, abs/1409.2329.  |
ArXiv e-prints.  |  |
Recurrent Continuous Translation Models.  
Neural Machine Translation in Linear Time.  
ArXiv e-prints.  |  |
Dropout: A Simple Way to Prevent Neural Networks from Overfitting.  
Attention Is All You Need.  
CoRR, abs/1706.03762.  |  |