10: Advanced Neural Machine Translation Architectures

Rico Sennrich

University of Edinburgh
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

[Srivastava et al., 2014]
Dropout

(a) Standard Neural Net
(b) After applying 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 $\frac{1}{p}$ at training time instead)

[Srivastava et al., 2014]
for recurrent connections, applying dropout at every time step blocks information flow

solution 1: only apply dropout to feedforward connections

Figure 2: Regularized multilayer RNN. The dashed arrows indicate connections where dropout is applied, and the solid lines indicate connections where dropout is not applied.

[Zaremba et al., 2014]
for recurrent connections, applying dropout at every time step blocks information flow

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

[Gal, 2015]
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

\[
\begin{align*}
\mu &= \frac{1}{H} \sum_{i=1}^{H} a_i \\
\sigma &= \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i - \mu)^2} \\
h &= \left[ \frac{g}{\sigma} \otimes (a - \mu) + b \right]
\end{align*}
\]
- increasing model depth often increases model performance
- example: stack RNN:

\[
\begin{align*}
    h_{i,1} &= g(U_1 h_{i-1,1} + W_1 x_i) \\
    h_{i,2} &= g(U_2 h_{i-1,2} + W_2 h_{i,1}) \\
    h_{i,3} &= g(U_3 h_{i-1,3} + W_3 h_{i,2})
\end{align*}
\]
Deep Networks

often necessary to combat vanishing gradient: residual connections between layers:

\[
\begin{align*}
    h_{i,1} &= g(U_1 h_{i-1,1} + W_1 x_i) \\
    h_{i,2} &= g(U_2 h_{i-1,2} + W_2 h_{i,1}) + h_{i,1} \\
    h_{i,3} &= g(U_3 h_{i-1,3} + W_3 h_{i,2}) + h_{i,2}
\end{align*}
\]
Layer Normalization and Deep Models: Results from UEDIN@WMT17

<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>

- 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).
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 \textit{filter} (or \textit{kernels})

\begin{itemize}
\item 2d convolution with 3x3 kernel
\end{itemize}

Convolutional Networks

when working with sequences, we often use 1d convolutions

1d convolution with width-3 kernel
Convolutional Networks

(this is similar to how we obtained hidden state for n-gram LM)

[Vaswani et al., 2013]
convolutional encoder actually predates RNN encoder

Figure 13.42: Refinement of the convolutional neural network model. Convolutions do not result in a single sentence embedding but a sequence. The encoder is also informed by a recurrent neural network (connections from output word embeddings to final decoding layer).

Generating the output sentence translation reverses the bottom-up process. One problem for the decoder is to decide the length of the output sentence. One option to address this problem is to add a model that predicts output length from input length. This then leads to the selection of the size of the reverse convolution matrices. See Figure 13.42 for an illustration of a variation of this idea. The shown architecture always uses a $K_2$ and a $K_3$ convolutional layer, resulting in a sequence of phrasal representations, not a single sentence embedding. There is an explicit mapping step from phrasal representations of input words to phrasal representations of output words, called transfer layer.

The decoder of the model includes a recurrent neural network on the output side. Sneaking in a recurrent neural network here does undermine a bit the argument about better parallelization. However, the claim still holds true for encoding the input, and a sequential language model is just a too powerful tool to disregard.

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.
to keep representation size constant, use padding
  $\rightarrow$ similar variable-size representation as RNN encoder

kernel can be applied to all windows in parallel
Convolutional Neural Machine Translation with Attention

- 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)

architecture of [Kalchbrenner et al., 2016]
1. General Architecture Variants

2. NMT with Convolutional Neural Networks

3. NMT with Self-Attention
same criticisms of recurrent architecture: recurrent computations cannot be parallelized
core idea: instead of fixed-width convolutional filter, use attention
there are different flavours of self-attention here: attend over previous layer of deep network
Transformer architecture

- stack of $N$ self-attention layers
- self-attention in decoder is *masked*
- decoder also attends to encoder states
- *Add & Norm*: residual connection and layer normalization

Figure 1: The Transformer - model architecture.
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 \(V\)
motivation for multi-head attention: different heads can attend to different states

Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.
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
## Comparison

### 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><strong>Transformer</strong></td>
<td><strong>27.5</strong></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><strong>Transformer</strong></td>
<td><strong>33.4</strong></td>
</tr>
</tbody>
</table>

https://github.com/marian-nmt/marian-dev/issues/116#issuecomment-340212787
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


