Machine Translation
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

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

Crossed units have been dropped.

Model combination nearly always improves the performance of machine learning methods. With large neural networks, however, the obvious idea of averaging the outputs of many separately trained nets is prohibitively expensive. Combining several models is most helpful when the individual models are different from each other and in order to make neural net models different, they should either have different architectures or be trained on different data. Training many different architectures is hard because finding optimal hyperparameters for each architecture is a daunting task and training each large network requires a lot of computation. Moreover, large networks normally require large amounts of training data and there may not be enough data available to train different networks on different subsets of the data. Even if one was able to train many different large networks, using them all at test time is infeasible in applications where it is important to respond quickly.

Dropout is a technique that addresses both these issues. It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 1. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability $p$ independent of other units, where $p$ can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks. For the input units, however, the optimal probability of retention is usually closer to 1 than to 0.5.

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)
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.
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

\[
\mu = \frac{1}{H} \sum_{i=1}^{H} a_i \\
\sigma = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i - \mu)^2} \\
h = \left[ \frac{g}{\sigma} \odot (a - \mu) + b \right]
\]
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:

\[ h_{i,1} = g(U_1 h_{i-1,1} + W_1 x_i) \]
\[ h_{i,2} = g(U_2 h_{i-1,2} + W_2 h_{i,1}) + h_{i,1} \]
\[ h_{i,3} = g(U_3 h_{i-1,3} + W_3 h_{i,2}) + h_{i,2} \]
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

When working with sequences, we often use 1D convolutions.

1D convolution with width-3 kernel:

- The length of the convolution is well known to be: \( \text{seq length} - \text{kernel width} + 1 = 8 - 3 + 1 = 6 \)
- So the output matrix will be \((6, 100)\) because there was no padding.
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
  → 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

architecture of [Gehring et al., 2017], as illustrated in P. Koehn, Neural Machine Translation
Convolutional Neural Machine Translation with Attention

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.

3.1 Encoder and Decoder Stacks

Encoder: The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

where $\text{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{\text{model}} = 512$.

Decoder: The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, ensures that the predictions for position $i$ can depend only on the known outputs at positions less than $i$.

3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.
Multi-Head Attention

- basic attention mechanisms: 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 \(Q, K, \text{and } V\)

![Diagram of Scaled Dot-Product Attention and Multi-Head Attention](image)
motivation for multi-head attention:
different heads can attend to different states
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>
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
A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.
ArXiv e-prints.

Convolutional Sequence to Sequence Learning.
CoRR, abs/1705.03122.

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.
Decoding with Large-Scale Neural Language Models Improves Translation.

Recurrent Neural Network Regularization.
CoRR, abs/1409.2329.