CSEP 517
Natural Language Processing

Neural Machine Translation

Luke Zettlemoyer

(Slides adapted from Danqi Chen, Greg Durrett, Chris Manning, Dan Jurafsky)
Last time

- Statistical MT
- Word-based
- Phrase-based
- Syntactic

NP → [DT₁ JJ₂ NN₃; DT₁ NN₃ JJ₂]
DT → [the, la]
DT → [the, le]
NN → [car, voiture]
JJ → [yellow, jaune]
NMT: the biggest success story of NLP Deep Learning

Neural Machine Translation went from a fringe research activity in 2014 to the leading standard method in 2016

• 2014: First seq2seq paper published

• 2016: Google Translate switches from SMT to NMT

• This is amazing!
  • SMT systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a handful of engineers in a few months
Neural Machine Translation

- A **single neural network** is used to translate from source to target

- Architecture: Encoder-Decoder

  - Two main components:
    - **Encoder**: Convert source sentence (input) into a vector/matrix
    - **Decoder**: Convert encoding into a sentence in target language (output)
Recall: RNNs

\[
h_t = g(W h_{t-1} + U x_t + b) \in \mathbb{R}^d
\]
Sequence to Sequence learning (Seq2seq)

- Encode entire input sequence into a single vector (using an RNN)
- Decode one word at a time (again, using an RNN!)
- Beam search for better inference
- Learning is not trivial! (vanishing/exploding gradients)

(Sutskever et al., 2014)
Neural Machine Translation (NMT)

Encoder RNN

Source sentence (input)

les pauvres sont démunis

Encoding of the source sentence. Provides initial hidden state for Decoder RNN.

Target sentence (output)

<START> the poor don’t have any money <END>

Decoder RNN

Decoder RNN is a Language Model that generates target sentence conditioned on encoding.

Note: This diagram shows test time behavior: decoder output is fed in --> as next step’s input
Seq2seq training

- Similar to training a language model!
- Minimize cross-entropy loss:

\[
\sum_{t=1}^{T} - \log P(y_t | y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)
\]

- Back-propagate gradients through both decoder and encoder
- Need a really big corpus

36M sentence pairs

Russian: Машинный перевод - это круто!

English: Machine translation is cool!
Training a Neural Machine Translation system

\[ J = \frac{1}{T} \sum_{t=1}^{T} J_t = J_1 + J_2 + J_3 + J_4 + J_5 + J_6 + J_7 \]

Seq2seq is optimized as a single system. Backpropagation operates “end to end”.

Source sentence (from corpus)

<START> the poor don’t have any money

Target sentence (from corpus)

les pauvres sont démunis

Encoder RNN

Decoder RNN
Greedy decoding

- Compute argmax at every step of decoder to generate word

- What’s wrong?
Exhaustive search?

- Find arg max \( P(y_1, \ldots, y_T | x_1, \ldots, x_n) \)

- Requires computing all possible sequences

  - \( O(V^T) \) complexity!

- Too expensive
A middle ground: Beam search

- **Key idea:** At every step, keep track of the $k$ most probable partial translations (hypotheses)

- Score of each hypothesis = log probability

\[
\sum_{t=1}^{j} \log P(y_t | y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)
\]

- Not guaranteed to be optimal

- More efficient than exhaustive search
Beam decoding

Beam size = $k = 2$. Blue numbers = $\text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$
Beam decoding

Beam size = k = 2. Blue numbers = \( \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x) \)

(slide credit: Abigail See)
Beam decoding

Beam size = $k = 2$. Blue numbers $= \text{score}(y_1, \ldots, y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i|y_1, \ldots, y_{i-1}, x)$

(slide credit: Abigail See)
Beam decoding

- Different hypotheses may produce \( \langle e \rangle \) (end) token at different time steps
  - When a hypothesis produces \( \langle e \rangle \), stop expanding it and place it aside
- Continue beam search until:
  - All \( k \) hypotheses produce \( \langle e \rangle \) OR
  - Hit max decoding limit \( T \)
- Select top hypotheses using the *normalized* likelihood score
  \[
  \frac{1}{T} \sum_{t=1}^{T} \log P(y_t | y_1, \ldots, y_{t-1}, x_1, \ldots, x_n)
  \]
  - Otherwise shorter hypotheses have higher scores
NMT vs SMT

Pros

‣ Better performance
  ▶ Fluency
  ▶ Longer context
‣ Single NN optimized end-to-end
‣ Less engineering
‣ Works out of the box for many language pairs

Cons

‣ Requires more data and compute
‣ Less interpretable
  ▶ Hard to debug
‣ Uncontrollable
  ▶ Heavily dependent on data - could lead to unwanted biases
‣ More parameters
How seq2seq changed the MT landscape
MT Progress

(source: Rico Sennrich)
Versatile seq2seq

- Seq2seq finds applications in many other tasks!
- Any task where inputs and outputs are sequences of words/characters
  - Summarization (input text → summary)
  - Dialogue (previous utterance → reply)
  - Parsing (sentence → parse tree in sequence form)
  - Question answering (context+question → answer)
Issues with vanilla seq2seq

- A single encoding vector, $h^{enc}$, needs to capture all the information about source sentence.
- Longer sequences can lead to vanishing gradients.
- Overfitting.

Diagram showing the bottleneck in a seq2seq model.
Remember alignments?

\[
a = (3, 4, 2, 1)^\top
\]

\[
a = (1, 2, 3, 0, 4)^\top
\]
Attention

- The neural MT equivalent of alignment models

- **Key idea:** At each time step during decoding, focus on a particular part of source sentence

  - This depends on the decoder’s current hidden state (i.e. notion of what you are trying to decode)

  - Usually implemented as a probability distribution over the hidden states of the encoder ($h_{i}^{enc}$)
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

dot product

les pauvres sont démunis

<START>

Source sentence (input)
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

dot product

les pauvres sont démunis

<START>
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Decoder RNN

Source sentence (input)

les pauvres sont démunis

<START>
Sequence-to-sequence with attention

- Encoder RNN
- Attention scores
- Decoder RNN

Source sentence (input):
les pauvres sont démunis

Warm up code:
```python
sequences = [5, 2, 3, 1, 4] # ['hello', 'world', 'this', 'is', 'a', 'sequence']
```

Output:
```
5 2 3 1 4
```

```python
from __future__ import print_function
import tensorflow as tf

# Encoder RNN
encoder_inputs = tf.placeholder(tf.int32, [None, None], name='encoder_inputs')
encoder_outputs = tf.nn.rnn(encoder_inputs, ...)  # Pooling, etc.

# Attention scores
attention_scores = tf.nn.softmax(...)  # dot product

# Decoder RNN
decoder_inputs = tf.placeholder(tf.int32, [None, None], name='decoder_inputs')
decoder_outputs = tf.nn.rnn(...)  # Pooling, etc.
```

Output:
```
5 2 3 1 4
```
On this decoder timestep, we’re mostly focusing on the first encoder hidden state (”les”)

Take softmax to turn the scores into a probability distribution

Source sentence (input)

les pauvres sont démunis <START>
Sequence-to-sequence with attention

Use the attention distribution to take a weighted sum of the encoder hidden states.

The attention output mostly contains information the hidden states that received high attention.

les pauvres sont démunis
Sequence-to-sequence with attention

Source sentence (input)

Encoder RNN

Attention distribution

Attention scores

Decoder RNN

les pauvres sont démunis

<START>

concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before
Sequence-to-sequence with attention

Encoder RNN

Source sentence (input)

les pauvres sont démunis

<START> the

Decoder RNN

Attention distribution

Attention scores

Attention output

poor

\( \hat{y}_2 \)
Sequence-to-sequence with attention

Encoder RNN

Attention distribution

Attention scores

Decoder RNN

Source sentence (input)

les pauvres sont démunis

<START> the poor

don’t
Sequence-to-sequence with attention

Encoder RNN

Attention scores

Attention distribution

Attention output

have

\( \hat{y}_4 \)

Source sentence (input)

les pauvres sont démunis

<START> the poor don’t
Sequence-to-sequence with attention

Source sentence (input)

les pauvres sont démunis

<START> the poor don't have

any

\hat{y}_5
Sequence-to-sequence with attention

Encoder RNN

Attention distribution

Attention scores

les pauvres sont démunis

<START> the poor don’t have any

Decoder RNN

money

Attention output

\[ y_6 \]
Computing attention

- Encoder hidden states: $h^\text{enc}_1, \ldots, h^\text{enc}_n$

- Decoder hidden state at time $t$: $h^\text{dec}_t$

- First, get attention scores for this time step (we will see what $g$ is soon!):
  
  $$e^t = [g(h^\text{enc}_1, h^\text{dec}_t), \ldots, g(h^\text{enc}_n, h^\text{dec}_t)]$$

- Obtain the attention distribution using softmax:
  
  $$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^n$$

- Compute weighted sum of encoder hidden states:
  
  $$a_t = \sum_{i=1}^{n} \alpha_i^t h^\text{enc}_i \in \mathbb{R}^h$$

- Finally, concatenate with decoder state and pass on to output layer: $[a_t; h^\text{dec}_t] \in \mathbb{R}^{2h}$
Types of attention

- Assume encoder hidden states $h_1, h_2, \ldots, h_n$ and decoder hidden state $z$.

1. **Dot-product attention** (assumes equal dimensions for $a$ and $b$:
   
   $$ e_i = g(h_i, z) = z^T h_i \in \mathbb{R} $$

2. **Multiplicative attention:**
   
   $$ g(h_i, z) = z^T W h_i \in \mathbb{R}, \text{ where } W \text{ is a weight matrix} $$

3. **Additive attention:**
   
   $$ g(h_i, z) = v^T \tanh (W_1 h_i + W_2 z) \in \mathbb{R} $$
   
   where $W_1, W_2$ are weight matrices and $v$ is a weight vector.
Rare Words and Monolingual Text
Handling Rare Words

- Words are a difficult unit to work with, e.g. vocabularies get very large, how to handle OOV?
- Character-level models are possible, but expensive
- Compromise solution: use thousands of “word pieces” (which may be full words but may also be parts of words)

Input: _the __eco tax _port i co _in _Po nt - de - Bu is ...

Output: _le _port ique _éco taxe _de _Pont - de - Bui s

- Can do transliteration, model sub-word regularities, etc.

Sennrich et al. (2016)
Byte Pair Encoding (BPE)

- Start with every individual byte (basically character) as its own symbol

```python
for i in range(num_merges):
    pairs = get_stats(vocab)
    best = max(pairs, key=pairs.get)
    vocab = merge_vocab(best, vocab)
```

- Count bigram character cooccurrences
- Merge the most frequent pair of adjacent characters

- Use large corpus of text for counting
- 8k merges => vocabulary of around 8000 word pieces. Includes many whole words
- Most SOTA NMT systems use this on both source + target

Sennrich et al. (2016)
Backtranslation

- Classical MT methods used a bilingual corpus of sentences $B = (S, T)$ and a large monolingual corpus $T'$ to train a language model. Can neural MT do the same?

- Approach 1: force the system to generate $T'$ as targets from null inputs

  $s_1, t_1$
  $s_2, t_2$
  ...
  [null], $t'_1$
  [null], $t'_2$
  ...

- Approach 2: generate synthetic sources with a $T$->$S$ machine translation system (backtranslation)

  $s_1, t_1$
  $s_2, t_2$
  ...
  $MT(t'_1), t'_1$
  $MT(t'_2), t'_2$
  ...

Sennrich et al. (2015)
Backtranslation

<table>
<thead>
<tr>
<th>name</th>
<th>training data</th>
<th>instances</th>
<th>Bleu tst2011</th>
<th>Bleu tst2012</th>
<th>Bleu tst2013</th>
<th>Bleu tst2014</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline (Gülçehre et al., 2015)</td>
<td>parallel</td>
<td>7.2m</td>
<td>18.4</td>
<td>18.8</td>
<td>19.9</td>
<td>18.7</td>
</tr>
<tr>
<td>deep fusion (Gülçehre et al., 2015)</td>
<td>parallel/parallel synth</td>
<td>6m/6m</td>
<td>20.2</td>
<td>20.2</td>
<td>21.3</td>
<td>20.6</td>
</tr>
<tr>
<td>baseline</td>
<td>parallel</td>
<td>7.2m</td>
<td>18.6</td>
<td>18.2</td>
<td>18.4</td>
<td>18.3</td>
</tr>
<tr>
<td>parallel synth</td>
<td>parallel/parallel synth</td>
<td>6m/6m</td>
<td>19.9</td>
<td>20.4</td>
<td>20.1</td>
<td>20.0</td>
</tr>
<tr>
<td>Gigaword mono</td>
<td>parallel/Gigaword mono</td>
<td>7.6m/7.6m</td>
<td>18.8</td>
<td>19.6</td>
<td>19.4</td>
<td>18.2</td>
</tr>
<tr>
<td>Gigaword synth</td>
<td>parallel/Gigaword synth</td>
<td>8.4m/8.4m</td>
<td><strong>21.2</strong></td>
<td><strong>21.1</strong></td>
<td><strong>21.8</strong></td>
<td>20.4</td>
</tr>
</tbody>
</table>

- Gigaword: large monolingual English corpus
- parallel synth: backtranslate training data; makes additional noisy source sentences which could be useful

Sennrich et al. (2015)
Google’s NMT System

- 8-layer LSTM encoder-decoder with attention, word piece vocabulary of 8k-32k

Wu et al. (2016)
Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation

Table 10: Mean of side-by-side scores on production data

| English → Spanish | 4.885 | 5.428 | 5.504 | 87% |
| English → French | 4.932 | 5.295 | 5.496 | 64% |
| English → Chinese| 4.035 | 4.594 | 4.987 | 58% |
| Spanish → English | 4.872 | 5.187 | 5.372 | 63% |
| French → English | 5.046 | 5.343 | 5.404 | 83% |
| Chinese → English | 3.694 | 4.263 | 4.636 | 60% |

(Wu et al., 2016)
Google’s NMT System

<table>
<thead>
<tr>
<th>Source</th>
<th>Translation</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>She was spotted three days later by a dog walker trapped in the quarry</td>
<td></td>
</tr>
<tr>
<td>PBMT</td>
<td>Elle a été repéré trois jours plus tard par un promeneur de chien piégé dans la carrière</td>
<td>6.0</td>
</tr>
<tr>
<td>GNMT</td>
<td>Elle a été repérée trois jours plus tard par un traîneau à chiens piégé dans la carrière</td>
<td>2.0</td>
</tr>
<tr>
<td>Human</td>
<td>Elle a été repérée trois jours plus tard par une personne qui promenait son chien coincée dans la carrière</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Gender is correct in “sled” GNMT but not in PBMT

Wu et al. (2016)
Transformers for MT
New Twist: Self-Attention

- Each word computes attention over every other word

\[ \alpha_{i,j} = \text{softmax}(x_i^\top x_j) \quad \text{scalar} \]

\[ x'_i = \sum_{j=1}^{n} \alpha_{i,j} x_j \quad \text{vector = sum of scalar} \times \text{vector} \]

- Multiple “heads”: Use parameters \( W_k \) and \( V_k \) to get different attention values + transform vectors

\[ \alpha_{k,i,j} = \text{softmax}(x_i^\top W_k x_j) \quad x'_{k,i} = \sum_{j=1}^{n} \alpha_{k,i,j} V_k x_j \]

Vaswani et al. (2017)
Transformers

- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

Vaswani et al. (2017)
Transformers

- Encoder and decoder are both transformers
- Decoder consumes the previous generated token (and attends to input), but has no recurrent state

Vaswani et al. (2017)
Transformers

- Big = 6 layers, 1000 dim for each token, 16 heads, base = 6 layers + other params halved

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EN-DE</td>
</tr>
<tr>
<td>ByteNet [18]</td>
<td>23.75</td>
</tr>
<tr>
<td>GNMT + RL [38]</td>
<td>24.6</td>
</tr>
<tr>
<td>ConvS2S [9]</td>
<td>25.16</td>
</tr>
<tr>
<td>MoE [32]</td>
<td>26.03</td>
</tr>
<tr>
<td>Deep-Att + PosUnk Ensemble [39]</td>
<td></td>
</tr>
<tr>
<td>GNMT + RL Ensemble [38]</td>
<td>26.30</td>
</tr>
<tr>
<td>ConvS2S Ensemble [9]</td>
<td>26.36</td>
</tr>
<tr>
<td>Transformer (base model)</td>
<td>27.3</td>
</tr>
<tr>
<td>Transformer (big)</td>
<td><strong>28.4</strong></td>
</tr>
</tbody>
</table>

Vaswani et al. (2017)
Visualization

Vaswani et al. (2017)
Visualization

Vaswani et al. (2017)
So is Machine Translation solved?

- **Nope!**
- **Many difficulties remain:**
  - Out-of-vocabulary words
  - Domain mismatch between train and test data
  - Maintaining context over longer text
  - Low-resource language pairs
So is Machine Translation solved?

• Nope!
• Using *common sense* is still hard
So is Machine Translation solved?

• Nope!
• NMT picks up biases in training data

Source: https://hackernoon.com/bias-sexist-or-this-is-the-way-it-should-be-ce1f7c8c683c
So is Machine Translation solved?

- Nope!
- Uninterpretable systems do strange things

Source: http://languagelog.ldc.upenn.edu/nll/?p=35120#more-35120
Massively multilingual MT

- Train a *single* neural network on 103 languages paired with English (remember Interlingua?)

- Massive improvements on low-resource languages

(Arivazhagan et al., 2019)