Last time: IBM Model 1

- Assume $p(a_m | m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}}$

- We then have:

$$p(w^{(s)}, w^{(t)}) = p(w^{(t)}) \sum_A \left( \frac{1}{M^{(t)}} \right)^{M^{(s)}} p(w^{(s)} | w^{(t)})$$

- How do we estimate $p(w^{(s)} = v | w^{(t)} = u)$?
If we have word-to-word alignments, we can compute the probabilities using the MLE:

\[ p(v | u) = \frac{\text{count}(u, v)}{\text{count}(u)} \]

where \( \text{count}(u, v) \) = #instances where target word \( u \) was aligned to source word \( v \) in the training set

However, word-to-word alignments are often hard to come by
EM for Model 1

- **(E-Step)** If we had an accurate translation model, we can estimate likelihood of each alignment as:

\[ q_m(a_m | \mathbf{w}^{(s)}, \mathbf{w}^{(t)}) = \frac{p(a_m | m, M^{(s)}, M^{(t)}) \times p(w_m^{(s)} | w_{a_m}^{(t)})}{\text{count}(u)} \]

- **(M Step)** Use expected count to re-estimate translation parameters:

How would you compute the new probabilities \( p(v | u) \)?

A) \( p(v | u) = \frac{E_q[\text{count}(u, v)]}{\text{count}(u)} \)

B) \( p(v | u) = \frac{E_q[\text{count}(u, v)]}{\text{count}(v)} \)

C) \( p(v | u) = E_q[\text{count}(u, v)] \)

Remember these are fixed
EM for Model 1

• **(E-Step)** If we had an accurate translation model, we can estimate likelihood of each alignment as:

\[ q_m(a_m \mid w^{(s)}, w^{(t)}) \propto p(a_m \mid m, M^{(s)}, M^{(t)}) \times p(w^{(s)}_m \mid w^{(t)}_{a_m}) \]

Remember these are fixed

• **(M Step)** Use expected count to re-estimate translation parameters:

\[
E_q [\text{count}(u, v)] = \sum_m q_m(a_m \mid w^{(s)}, w^{(t)}) \times \delta(w^{(s)}_m = v) \times \delta(w^{(t)}_{a_m} = u).
\]

\[
p(v \mid u) = \frac{E_q[\text{count}(u, v)]}{\text{count}(u)}
\]
Decoding: How do we translate?

- We want: \( \arg \max_{w^{(t)}} p(w^{(t)} \mid w^{(s)}) = \arg \max_{w^{(t)}} \frac{p(w^{(s)}, w^{(t)})}{p(w^{(s)})} \)

- Sum over all possible alignments:

\[
p(w^{(s)}, w^{(t)}) = \sum_{A} p(w^{(s)}, w^{(t)}, A) = p(w^{(t)}) \sum_{A} p(A) \times p(w^{(s)} \mid w^{(t)}, A)
\]

- Alternatively, take the max over alignments

- Decoding: Greedy/beam search
Model 1: Decoding

At every step $m$, pick target word $w_m^{(t)}$ to maximize product of:

1. Language model: $p_{LM}(w_m^{(t)} | w_1^{(t)}, \ldots, w_{m-1}^{(t)})$

2. Translation model: $p(w_{b_m}^{(s)} | w_m^{(t)})$

where $b_m$ is the inverse alignment from target to source
IBM Model I

- Assume \( p(a_m \mid m, M^{(s)}, M^{(t)}) = \frac{1}{M^{(t)}} \)

- Each source word is aligned to at most one target word

- We then have:

\[
p(w^{(s)}, w^{(t)}) = p(w^{(t)}) \sum_A \left( \frac{1}{M^{(t)}} \right)^{M^{(s)}} \ p(w^{(s)} \mid w^{(t)})
\]
IBM Model 2

- Slightly relaxed assumption:
  - \( p(a_m \mid m, M^{(s)}, M^{(t)}) \) is also estimated/learned

- Some independence assumptions from Model 1 still required:
  - Alignment probability factors across tokens:
    \[
    p(A \mid w^{(s)}, w^{(t)}) = \prod_{m=1}^{M^{(s)}} p(a_m \mid m, M^{(s)}, M^{(t)}).
    \]
  - Translation probability factors across tokens:
    \[
    p(w^{(s)} \mid w^{(t)}, A) = \prod_{m=1}^{M^{(s)}} p(w^{(s)}_m \mid w^{(t)}_{a_m}),
    \]
Other IBM models

- Models 3 - 6 make successively weaker assumptions
- But get progressively harder to optimize
- Simpler models are often used to ‘initialize’ complex ones
- e.g train Model 1 and use it to initialize Model 2 translation parameters

| Model 1: lexical translation |
| Model 2: additional absolute alignment model |
| Model 3: extra fertility model |
| Model 4: added relative alignment model |
| Model 5: fixed deficiency problem. |
| Model 6: Model 4 combined with a HMM alignment model in a log linear way |
Phrase-based MT

Nous allons prendre un verre
We will take a glass
We’ll have a drink

- Word-by-word translation is not sufficient in many cases
- Solution: build alignments and translation tables between multiword spans or “phrases”
Phrase-based MT

- Solution: build alignments and translation tables between multiword spans or “phrases”

- **Translations** condition on multi-word units and assign probabilities to multi-word units

- **Alignments** map from spans to spans

\[ p(w^{(s)} | w^{(t)}, A) = \prod_{((i,j),(k,\ell)) \in A} p_{w^{(s)}|w^{(t)}}(\{w_{i+1}^{(s)}, w_{i+2}^{(s)}, \ldots, w_{j}^{(s)} \} | \{w_{k+1}^{(t)}, w_{k+2}^{(t)}, \ldots, w_{\ell}^{(t)} \}) \]
Vauquois Pyramid

- Hierarchy of concepts and distances between them in different languages
- Lowest level: individual words/characters
- Higher levels: syntax, semantics
- Interlingua: Generic language-agnostic representation of meaning
Syntactic MT

- Rather than use phrases, use a *synchronous context-free grammar*: constructs “parallel” trees in two languages simultaneously

NP → [DT₁ JJ₂ NN₃; DT₁ NN₃ JJ₂]
DT → [the, la]
DT → [the, le]
NN → [car, voiture]
JJ → [yellow, jaune]

- Assumes parallel syntax up to reordering
- Translation = parse the input with “half” the grammar, read off other half

(Slide credit: Greg Durrett)
Syntactic MT

Input

Output

Grammar

- Relax this by using lexicalized rules, like “syntactic phrases”
- Leads to HUGE grammars, parsing is slow
Neural Machine Translation
Neural Machine Translation

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

- 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

\[ h_t = g(Wh_{t-1} + Ux_t + b) \in \mathbb{R}^d \]
Recall: RNNs

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

What is the maximum sequence length an RNN could theoretically take as input?

A) 10
B) 128
C) \( \infty \)
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)*
Encoder

Sentence: This cat is cute

word embedding

This cat is cute
Sentence: This cat is cute

Encoder

$h_0 \rightarrow h_1$

$x_1 \rightarrow \text{word embedding}$

This          cat          is          cute
Sentence: *This cat is cute*

Encoder
**Encoder**

Sentence: *This cat is cute*

\[
\begin{align*}
\text{This} & \quad x_1 & \quad \rightarrow & \quad h_0 \\
\text{cat} & \quad x_2 & \quad \rightarrow & \quad h_1 \\
\text{is} & \quad x_3 & \quad \rightarrow & \quad h_2 \\
\text{cute} & \quad x_4 & \quad \rightarrow & \quad h_3 \\
\end{align*}
\]

\[h_4 \rightarrow h^{enc}\]
Decoder

\[ h^{enc} \]

word embedding

\[ \langle s \rangle \]
Decoder

\[ \begin{align*}
  h^{enc} & \rightarrow z_1 \\
  z_1 & \rightarrow y_1 \\
  y_1 & \rightarrow o \\
  o & \rightarrow ce
\end{align*} \]

word embedding

\[<s>\]
Decoder

\[ h^{enc} \rightarrow z_1 \rightarrow z_2 \]

\[ \begin{align*}
  & y_1 \\
  & y_2
\end{align*} \]

\[ \begin{align*}
  & o \\
  & o
\end{align*} \]

\[ \begin{align*}
  \text{word embedding} \\
  & <s> \\
  & ce
\end{align*} \]

ce

chat
Decoder

- A conditioned language model

$$h_{enc} \rightarrow z_1 \rightarrow z_2 \rightarrow z_3 \rightarrow z_4 \rightarrow z_5$$

Word embedding

$$<s> \quad \text{ce} \quad \text{chat} \quad \text{est} \quad \text{mignon} \quad <e>$$
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

English: Machine translation is cool!

Russian: Машиный перевод - это круто!
**Seq2seq training**

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

= negative log prob of “he”
= negative log prob of “with”
= negative log prob of <END>

Encoder RNN

Source sentence (from corpus)

Target sentence (from corpus)

Decoder RNN

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

(slide credit: Abigail See)
Greedy decoding

- Compute argmax at every step of decoder to generate word
- What’s wrong?
Exhaustive search?

- Find $\arg\max_{y_1,\ldots,y_T} P(y_1,\ldots,y_T|x_1,\ldots,x_n)$
- Requires computing all possible sequences

What is the complexity of doing this search?

A) $O(VT)$
B) $O(V^T)$
C) $O(T^V)$
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 of sequence so far

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

- Not guaranteed to be optimal

- More efficient than exhaustive search
Beam decoding

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

(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

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)
Backtrack

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

- 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

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>