COMS 4705.H: Neural Machine Translation

Karl Stratos

April 7, 2017
Machine Translation (MT)

- Goal: Translate text from one language to another.
- One of the oldest problems in artificial intelligence.
Some History

- Early ’90s: Rise of **statistical** MT (SMT)

- **Exploit** parallel text.

  And the programme has been implemented

  Le programme a été mis en application

- **Infer word alignment** (“IBM” models, Brown et al., 1993)

```
And  the  programme  has  been  implemented
     |        |
Le  programme  a  été  mis  en  application
   |         |
1 2 3 4 5 6 7
```
SMT: Huge Pipeline

1. Use IBM models to extract word alignment, phrase alignment (Koehn et al., 2003).
2. Use syntactic analyzers (e.g., parser) to extract features and manipulate text (e.g., phrase re-ordering).
3. Use a separate language model to enforce fluency.
4. ...

Multiple independently trained models patched together
▶ Really complicated, prone to error propagation
SMT: Huge Pipeline

1. Use IBM models to extract word alignment, phrase alignment (Koehn et al., 2003).
2. Use syntactic analyzers (e.g., parser) to extract features and manipulate text (e.g., phrase re-ordering).
3. Use a separate language model to enforce fluency.
4. ...

Multiple independently trained models patched together
- Really complicated, prone to error propagation
Rise of Neural MT

Started taking off around 2014

▶ Replaced the entire pipeline with a single model

▶ Called “end-to-end” training/prediction

   **Input:** Le programme a été mis en application
   **Output:** And the programme has been implemented

▶ **Revolution** in MT

   ▶ Better performance, way simpler system
   ▶ A hallmark of the recent neural domination in NLP
Overview

Review of RNN

Review of RNN Language Model

Neural MT: Conditional RNN Language Model
Recap: Recurrent Neural Network (RNN)

- Always think of an RNN as a mapping \( \phi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'} \)

  **Input:** an input vector \( x \in \mathbb{R}^d \), a state vector \( h \in \mathbb{R}^{d'} \)

  **Output:** a new state vector \( h' \in \mathbb{R}^{d'} \)
Recap: Recurrent Neural Network (RNN)

- Always think of an RNN as a mapping $\phi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$
  
  **Input:** an input vector $x \in \mathbb{R}^d$, a state vector $h \in \mathbb{R}^{d'}$
  
  **Output:** a new state vector $h' \in \mathbb{R}^{d'}$

- Left-to-right RNN processes input sequence $x_1 \ldots x_m \in \mathbb{R}^d$ as

$$h_i = \phi (x_i, h_{i-1})$$

where $h_0$ is an initial state vector.
Recap: Recurrent Neural Network (RNN)

- Always think of an RNN as a mapping $\phi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$

  **Input:** an input vector $x \in \mathbb{R}^d$, a state vector $h \in \mathbb{R}^{d'}$
  **Output:** a new state vector $h' \in \mathbb{R}^{d'}$

- Left-to-right RNN processes input sequence $x_1 \ldots x_m \in \mathbb{R}^d$ as

  $$h_i = \phi (x_i, h_{i-1})$$

  where $h_0$ is an initial state vector.

- Idea: $h_i$ is a *representation* of $x_i$ that has incorporated all inputs to the left.

  $$h_i = \phi (x_i, \phi (x_{i-1}, \phi (x_{i-2}, \cdots \phi (x_1, h_0) \cdots )))$$
Variety 1: “Simple” RNN

Parameters $U \in \mathbb{R}^{d' \times d}$ and $V \in \mathbb{R}^{d' \times d'}$

$$h_i = \tanh (Ux_i + Vh_{i-1})$$
Stacked Simple RNN

- Parameters $U^{(1)} \ldots U^{(L)} \in \mathbb{R}^{d' \times d}$ and $V^{(1)} \ldots V^{(L)} \in \mathbb{R}^{d' \times d'}$

\[
\begin{align*}
    h^{(1)}_i &= \tanh \left( U^{(1)} x_i + V^{(1)} h^{(1)}_{i-1} \right) \\
    h^{(2)}_i &= \tanh \left( U^{(2)} h^{(1)}_i + V^{(2)} h^{(2)}_{i-1} \right) \\
    &\vdots \\
    h^{(L)}_i &= \tanh \left( U^{(L)} h^{(L-1)}_i + V^{(L)} h^{(L)}_{i-1} \right)
\end{align*}
\]
Stacked Simple RNN

- Parameters $U^{(1)} \ldots U^{(L)} \in \mathbb{R}^{d' \times d}$ and $V^{(1)} \ldots V^{(L)} \in \mathbb{R}^{d' \times d}$

$$
  h_{i}^{(1)} = \tanh \left( U^{(1)} x_i + V^{(1)} h_{i-1}^{(1)} \right)
$$

$$
  h_{i}^{(2)} = \tanh \left( U^{(2)} h_{i}^{(1)} + V^{(2)} h_{i-1}^{(2)} \right)
$$

$$
  \vdots
$$

$$
  h_{i}^{(L)} = \tanh \left( U^{(L)} h_{i}^{(L-1)} + V^{(L)} h_{i-1}^{(L)} \right)
$$

- Think of it as mapping $\phi : \mathbb{R}^d \times \mathbb{R}^{Ld'} \rightarrow \mathbb{R}^{Ld'}$.

$$
  x_i \begin{bmatrix}
    h_{i-1}^{(1)} \\
    \vdots \\
    h_{i-1}^{(L)}
  \end{bmatrix} \mapsto \begin{bmatrix}
    h_{i}^{(1)} \\
    \vdots \\
    h_{i}^{(L)}
  \end{bmatrix}
$$
Variety 2: Long Short-Term Memory (LSTM)

- Parameters $U^q, U^c, U^o \in \mathbb{R}^{d' \times d}$, $V^q, V^c, V^o, W^q, W^o \in \mathbb{R}^{d' \times d'}$

  \[
  q_i = \sigma(U^q x_i + V^q h_{i-1} + W^q c_{i-1}) \\
  c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh(U^c x_i + V^c h_{i-1}) \\
  o_i = \sigma(U^o x_i + V^o h_{i-1} + W^o c_{i}) \\
  h_i = o_i \odot \tanh(c_i)
  \]
Variety 2: Long Short-Term Memory (LSTM)

- Parameters $U^q, U^c, U^o, V^q, V^c, V^o, W^q, W^o \in \mathbb{R}^{d' \times d}$

  \[
  q_i = \sigma (U^q x_i + V^q h_{i-1} + W^q c_{i-1})
  \]
  \[
  c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh (U^c x_i + V^c h_{i-1})
  \]
  \[
  o_i = \sigma (U^o x_i + V^o h_{i-1} + W^o c_i)
  \]
  \[
  h_i = o_i \odot \tanh(c_i)
  \]

- Idea: “Memory cells” $c_i$ can carry long-range information.
  - What happens if $q_i$ is close to zero?
Variety 2: Long Short-Term Memory (LSTM)

- Parameters $U^q, U^c, U^o \in \mathbb{R}^{d' \times d}, V^q, V^c, V^o, W^q, W^o \in \mathbb{R}^{d' \times d'}$

  \[
  q_i = \sigma (U^q x_i + V^q h_{i-1} + W^q c_{i-1}) \\
  c_i = (1 - q_i) \odot c_{i-1} + q_i \odot \tanh (U^c x_i + V^c h_{i-1}) \\
  o_i = \sigma (U^o x_i + V^o h_{i-1} + W^o c_i) \\
  h_i = o_i \odot \tanh(c_i)
  \]

- Idea: “Memory cells” $c_i$ can carry long-range information.
  - What happens if $q_i$ is close to zero?

- Can be stacked as in simple RNN.
Variety 3: Gated Recurrent Unit (GRU)

- Think: Simplified LSTM

\[
\begin{align*}
\hat{r}_i &= \sigma(U_r x_i + V_r h_{i-1}) \\
\bar{h}_i &= \tanh(U_o i + \sum_i V(z_i \odot h_{i-1})) \\
z_i &= \sigma(U_z x_i + V_z h_{i-1}) \\
\hat{h}_i &= (1 - z_i) h_{i-1} + z_i \bar{h}_i
\end{align*}
\]

- No explicit memory cells: \( r_i \) “resetting” and \( z_i \) updating state

- What happens if \( z_i \) is close to zero?

- Can be stacked as in simple RNN.
Variety 3: Gated Recurrent Unit (GRU)

- Think: Simplified LSTM

- Parameters $U^r, U^z, U \in \mathbb{R}^{d' \times d}, V^r, V^z, V \in \mathbb{R}^{d' \times d'}$

$$
\begin{align*}
    r_i &= \sigma (U^r x_i + V^r h_{i-1}) \\
    \bar{h}_i &= \tanh(U o_i + V (r_i \odot h_{i-1})) \\
    z_i &= \sigma (U^z x_i + V^z h_{i-1}) \\
    h_i &= (1 - z_i) h_{i-1} + z_i \bar{h}_i
\end{align*}
$$

- No explicit memory cells: $r_i$ "resetting" and $z_i$ updating state

- What happens if $z_i$ is close to zero?

- Can be stacked as in simple RNN.
Variety 3: Gated Recurrent Unit (GRU)

- Think: Simplified LSTM

- Parameters $U^r, U^z, U \in \mathbb{R}^{d' \times d}, V^r, V^z, V \in \mathbb{R}^{d' \times d'}$

  \[ r_i = \sigma (U^r x_i + V^r h_{i-1}) \]
  \[ \bar{h}_i = \tanh(U o_i + V (r_i \odot h_{i-1})) \]
  \[ z_i = \sigma (U^z x_i + V^z h_{i-1}) \]
  \[ h_i = (1 - z_i) h_{i-1} + z_i \bar{h}_i \]

- No explicit memory cells: $r_i$ “resetting” and $z_i$ updating state
  - What happens if $z_i$ is close to zero?
Variety 3: Gated Recurrent Unit (GRU)

- Think: Simplified LSTM

- Parameters $U^r, U^z, U \in \mathbb{R}^{d' \times d}$, $V^r, V^z, V \in \mathbb{R}^{d' \times d'}$

  \[
  r_i = \sigma(U^r x_i + V^r h_{i-1})
  \]

  \[
  \bar{h}_i = \tanh(U o_i + V (r_i \odot h_{i-1}))
  \]

  \[
  z_i = \sigma(U^z x_i + V^z h_{i-1})
  \]

  \[
  h_i = (1 - z_i) h_{i-1} + z_i \bar{h}_i
  \]

- No explicit memory cells: $r_i$ “resetting” and $z_i$ updating state
  - What happens if $z_i$ is close to zero?

- Can be stacked as in simple RNN.
Using RNN in DyNet

```python
model = Model()
lstm = LSTMBuilder(LAYERS, INPUT_DIM, STATE_DIM, model)

s = lstm.initial_state()

states = []
for x in inputs:
    s = s.add_input(x)
    h = s.output()
    states.append(h)
```
Overview

Review of RNN

Review of RNN Language Model

Neural MT: Conditional RNN Language Model
Recall Language Model

A **language model** defines a probability distribution \( p(x_1 \ldots x_m) \) over all sentences \( x_1 \ldots x_m \) in vocabulary \( V \).

**Goal:** Design a *good* language model

\[
p(\text{the dog barked}) > p(\text{the cat barked}) > p(\text{dog the barked}) > p(\text{oqc shgwqw#w 1g0})
\]

How can we use an RNN to build a language model?
Basic RNN Language Model

Denote all model parameters by $\Theta$

- Vector $e_x \in \mathbb{R}^d$ for every $x \in V \cup \{\star\}$
- Left-to-right RNN $\phi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$
- Feedforward $f : \mathbb{R}^{d'} \rightarrow \mathbb{R}^{|V|+1}$

Every sentence $x_1 \ldots x_m$ is padded with $x_0 = \star$ and $x_{m+1} = \text{STOP}$

\[ \star \, x_1 \ldots x_m \, \text{STOP} \]
The network computes a probability distribution over $V \cup \{\text{STOP}\}$ at position $i = 1 \ldots m$ by

$$h_i = \phi(e_{x_{i-1}}, h_{i-1})$$

$$p_\Theta(x|x_0 \ldots x_{i-1}) = \text{softmax}_x(f(h_i))$$
Distribution over Words and Sentences

The network computes a probability distribution over $V \cup \{\text{STOP}\}$ at position $i = 1 \ldots m$ by

$$h_i = \phi(e_{x_{i-1}}, h_{i-1})$$

$$p_{\Theta}(x|x_0 \ldots x_{i-1}) = \text{softmax}_x(f(h_i))$$

Probability of sentence $x_1 \ldots x_m$ under the network:

$$p_{\Theta}(x_1 \ldots x_m) = \prod_{i=1}^{m} p_{\Theta}(x_i|x_0 \ldots x_{i-1}) \times p_{\Theta}(\text{STOP}|x_0 \ldots x_m)$$

No Markov assumption.
Training

Given a corpus of $N$ training sentences $x^{(1)} \ldots x^{(N)}$, find parameters $\Theta^*$ that maximize the log likelihood of the corpus:

$$
\Theta^* \approx \arg \min_{\Theta} - \sum_{i=1}^{N} \log p_{\Theta}(x^{(i)})
$$

loss
Training

Given a corpus of \( N \) training sentences \( x^{(1)} \ldots x^{(N)} \), find parameters \( \Theta^* \) that maximize the log likelihood of the corpus:

\[
\Theta^* \approx \arg \min_{\Theta} - \sum_{i=1}^{N} \log p_{\Theta}(x^{(i)})
\]

Leave this to DyNet...

\[
\text{loss.backward()}
\]
\[
\text{trainer.update()}
\]
Aside: Sentence Generation

We can sample a random sentence $S$ from the model.

1. Initialize $S \leftarrow [\ast]$ and $h \leftarrow h_0$.
2. Keep repeating

$$h \leftarrow \phi (e_{S[-1]}, h)$$
$$x \sim \text{softmax} (f(h))$$
$$S \leftarrow S + [x]$$

until $x = \text{STOP}$. 
Overview

Review of RNN

Review of RNN Language Model

Neural MT: Conditional RNN Language Model
Translation Problem

- Vocabulary of the source language $V^{src}$

$$V^{src} = \left\{ \text{그, 개가, 보았다, 소식, 2017, 5월...} \right\}$$

- Vocabulary of the target language $V^{trg}$

$$V^{trg} = \left\{ \text{the, dog, cat, 2021, May, ...} \right\}$$

- Task. Given any sentence $x_1 \ldots x_m \in V^{src}$, produce a corresponding translation $y_1 \ldots y_n \in V^{trg}$.

개가 짖었다 $\implies$ the dog barked
Evaluating Machine Translation

- $T$: human-translated sentences
- $\hat{T}$: machine-translated sentences

$\text{BLEU} = \min \left( \frac{1}{|\hat{T}|} \sum_{n=1}^{\infty} p_n \right)^{\frac{1}{4}}$
Evaluating Machine Translation

- $T$: human-translated sentences

- $\hat{T}$: machine-translated sentences

- $p_n$: precision of $n$-grams in $\hat{T}$ against $n$-grams in $T$ (sentence-wise)
Evaluating Machine Translation

- \( T \): human-translated sentences

- \( \hat{T} \): machine-translated sentences

- \( p_n \): precision of \( n \)-grams in \( \hat{T} \) against \( n \)-grams in \( T \) (sentence-wise)

- **BLEU**: Controversial but popular scheme to automatically evaluate translation quality

\[
BLEU = \min \left( 1, \frac{|\hat{T}|}{|T|} \right) \times \left( \prod_{n=1}^{4} p_n \right)^{\frac{1}{4}}
\]
A **translation model** defines a probability distribution $p(y_1 \ldots y_n | x_1 \ldots x_m)$ over all sentences $y_1 \ldots y_n \in V^{trg}$ conditioning on any sentence $x_1 \ldots x_m \in V^{src}$.
A **translation model** defines a probability distribution
\[ p(y_1 \ldots y_n | x_1 \ldots x_m) \]
over all sentences \( y_1 \ldots y_n \in V^{trg} \)
conditioning on any sentence \( x_1 \ldots x_m \in V^{src} \).

**Goal:** Design a *good* translation model

\[ p(\text{the dog barked} | \개가 짖었다}) > p(\text{the cat barked} | \개가 짖었다}) \]
\[ > p(\text{dog the barked} | \개가 짖었다}) \]
\[ > p(\text{oqc shgwqw#w 1g0} | \개가 짖었다}) \]
A **translation model** defines a probability distribution

\[ p(y_1 \ldots y_n|x_1 \ldots x_m) \] over all sentences \( y_1 \ldots y_n \in V^{\text{trg}} \)

conditioning on any sentence \( x_1 \ldots x_m \in V^{\text{src}} \).

**Goal:** Design a *good* translation model

\[
p(\text{the dog barked}|개가 짖었다}) > p(\text{the cat barked}|개가 짖었다}) > p(\text{dog the barked}|개가 짖었다}) > p(\text{oqc shgwqw#w 1g0}|개가 짖었다})
\]

How can we use an RNN to build a translation model?
Basic Encoder-Decoder Framework

Model parameters

- Vector $e_x \in \mathbb{R}^d$ for every $x \in V^{src}$
- Vector $e_y \in \mathbb{R}^d$ for every $y \in V^{trg} \cup \{\ast\}$
- Encoder RNN $\psi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$ for $V^{src}$
- Decoder RNN $\phi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$ for $V^{trg}$
- Feedforward $f : \mathbb{R}^{d'} \rightarrow \mathbb{R}^{|V^{trg}|+1}$
Basic Encoder-Decoder Framework

Model parameters

- Vector $e_x \in \mathbb{R}^d$ for every $x \in V^{src}$
- Vector $e_y \in \mathbb{R}^d$ for every $y \in V^{trg} \cup \{\ast\}$
- Encoder RNN $\psi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$ for $V^{src}$
- Decoder RNN $\phi : \mathbb{R}^d \times \mathbb{R}^{d'} \rightarrow \mathbb{R}^{d'}$ for $V^{trg}$
- Feedforward $f : \mathbb{R}^{d'} \rightarrow \mathbb{R}^{|V^{trg}|+1}$

Basic idea

1. Transform $x_1 \ldots x_m \in V^{src}$ with $\psi$ into some representation $\xi$.
2. Build a language model $\phi$ over $V^{trg}$ conditioning on $\xi$. 
Encoder

For $i = 1 \ldots m$,

$$h_i^{\psi} = \psi \left( e^{x_i}, h_{i-1}^{\psi} \right)$$
For \( i = 1 \ldots m \),

\[
\begin{align*}
    h^\psi_i &= \psi (e x_i, h^\psi_{i-1}) \\
    h^\psi_m &= \psi (e x_m, \psi (e x_{m-1}, \psi (e x_{m-2}, \cdots \psi (e x_1, h^\psi_0) \cdots )))
\end{align*}
\]
Decoder

Initialize $h_0^\phi = h_m^\psi$ and $y_0 = *$. 
Decoder

Initialize $h_0^\phi = h_m^\psi$ and $y_0 = \ast$.

For $i = 1, 2, \ldots$, the decoder defines a probability distribution over $V^\text{trg} \cup \{\text{STOP}\}$ as ($\oplus$ denotes vector concatenation)

$$h_i^\phi = \phi \left( e_y y_{i-1} \oplus h_m^\psi, h_{i-1}^\phi \right)$$

$$p_{\Theta}(y|x_1 \ldots x_m, y_0 \ldots y_{i-1}) = \text{softmax}_y(f(h_i^\phi))$$
Decoder

Initialize $h_0^\phi = h_m^\psi$ and $y_0 = \ast$.

For $i = 1, 2, \ldots$, the decoder defines a probability distribution over $V_{\text{trg}} \cup \{\text{STOP}\}$ as ($\oplus$ denotes vector concatenation)

$$h_i^\phi = \phi \left( e_{y_{i-1}} \oplus h_m^\psi, h_{i-1}^\phi \right)$$

$$p_\Theta(y|x_1 \ldots x_m, y_0 \ldots y_{i-1}) = \text{softmax}_y(f(h_i^\phi))$$

Probability of translation $y_1 \ldots y_n$ given $x_1 \ldots x_m$:

$$p_\Theta(y_1 \ldots y_n|x_1 \ldots x_m) = \prod_{i=1}^n p_\Theta(y_i|x_1 \ldots x_m, y_0 \ldots y_{i-1}) \times$$

$$p_\Theta(\text{STOP}|x_1 \ldots x_m, y_0 \ldots y_n)$$
Training

Given parallel text of $N$ sentence-translation pairs $(x^{(1)}, y^{(1)}) \ldots (x^{(N)}, y^{(N)})$, find parameters $\Theta^*$ that maximize the log likelihood of the data:

$$\Theta^* \approx \arg \min_{\Theta} - \sum_{i=1}^{N} \log p_{\Theta}(y^{(i)} | x^{(i)})$$

loss
Given parallel text of $N$ sentence-translation pairs $(x^{(1)}, y^{(1)}) \ldots (x^{(N)}, y^{(N)})$, find parameters $\Theta^*$ that maximize the log likelihood of the data:

$$\Theta^* \approx \arg \min_{\Theta} - \sum_{i=1}^{N} \log p_{\Theta}(y^{(i)} | x^{(i)})$$

Leave this to DyNet…

loss.backward()

trainer.update()
Greedy Translation

Given sentence $x_1 \ldots x_m \in V^{src}$,

1. Encode the sentence: for $i = 1 \ldots m$,

$$h_i^\psi = \psi(e_{x_i}, h_{i-1}^\psi)$$
Greedy Translation

Given sentence $x_1 \ldots x_m \in V^{\text{src}}$,

1. Encode the sentence: for $i = 1 \ldots m$,

$$h^\psi_i = \psi \left( e_{x_i}, h^\psi_{i-1} \right)$$

2. Initialize $h^\phi \leftarrow h^\psi_m$ and $S \leftarrow [*]$. 

Extra credit: Do beam search instead of greedy prediction.
Greedy Translation

Given sentence $x_1 \ldots x_m \in V^{\text{src}}$,

1. Encode the sentence: for $i = 1 \ldots m$,

   $$h_{i}^{\psi} = \psi(e_{x_i}, h_{i-1}^{\psi})$$

2. Initialize $h^{\phi} \leftarrow h_{m}^{\psi}$ and $S \leftarrow [*]$.

3. Keep repeating

   $$h^{\phi} \leftarrow \phi(e_{S[-1]} \oplus h_{m}^{\psi}, h^{\phi})$$

   $$y \leftarrow \arg \max_{y \in V^{\text{trg}} \cup \{\text{STOP}\}} \text{softmax}_{y} \left( f(h^{\phi}) \right)$$

   $$S \leftarrow S + [y]$$

   until $y = \text{STOP}$.

Extra credit: Do beam search instead of greedy prediction.
Greedy Translation

Given sentence $x_1 \ldots x_m \in V^{\text{src}}$,

1. Encode the sentence: for $i = 1 \ldots m$,

   $$ h_i^\psi = \psi \left( e_{x_i}, h_{i-1}^\psi \right) $$

2. Initialize $h^\phi \leftarrow h_m^\psi$ and $S \leftarrow [\ast]$.

3. Keep repeating

   $$ h^\phi \leftarrow \phi(e_{S[-1]} \oplus h_m^\psi, h^\phi) $$

   $$ y \leftarrow \arg \max_{y \in V^{\text{trg}} \cup \{\text{STOP}\}} \text{softmax}_y \left( f(h^\phi) \right) $$

   $$ S \leftarrow S + [y] $$

   until $y = \text{STOP}$.

Extra credit: Do beam search instead of greedy prediction.
Bidirectional Encoder

- The encoder now consists of 2 RNNs.
  1. Forward RNN $\psi^f : \mathbb{R}^d \times \mathbb{R}^{d'/2} \rightarrow \mathbb{R}^{d'/2}$
  2. Backward RNN $\psi^b : \mathbb{R}^d \times \mathbb{R}^{d'/2} \rightarrow \mathbb{R}^{d'/2}$
Bidirectional Encoder

- The encoder now consists of 2 RNNs.
  1. Forward RNN $\psi^f : \mathbb{R}^d \times \mathbb{R}^{d'/2} \rightarrow \mathbb{R}^{d'/2}$
  2. Backward RNN $\psi^b : \mathbb{R}^d \times \mathbb{R}^{d'/2} \rightarrow \mathbb{R}^{d'/2}$

- For $i = 1 \ldots m$, compute

$$f_i = \psi^f (x_i, f_{i-1})$$
Bidirectional Encoder

- The encoder now consists of 2 RNNs.
  1. Forward RNN $\psi^f : \mathbb{R}^d \times \mathbb{R}^{d'/2} \rightarrow \mathbb{R}^{d'/2}$
  2. Backward RNN $\psi^b : \mathbb{R}^d \times \mathbb{R}^{d'/2} \rightarrow \mathbb{R}^{d'/2}$

- For $i = 1 \ldots m$, compute
  \[ f_i = \psi^f(x_i, f_{i-1}) \]

- For $i = m \ldots 1$, compute
  \[ b_i = \psi^b(x_i, b_{i+1}) \]
Bidirectional Encoding of Each Word

- Bidirectional representation of $x_1 \ldots x_m$:

$$h_\psi = \begin{bmatrix} f_m \\ b_1 \end{bmatrix}$$

Use as before.
Bidirectional Encoding of Each Word

- Bidirectional representation of $x_1 \ldots x_m$:

$$h^\psi = \begin{bmatrix} f_m \\ b_1 \end{bmatrix}$$

Use as before.

- Bidirectional representation of each $x_i$:

$$h^\psi_i = \begin{bmatrix} f_i \\ b_i \end{bmatrix}$$

Use for decoder with attention (next slide).
Decoder with Attention

- Instead of using 1 fixed vector to encode all $x_1 \ldots x_m$, decoder decides which words to pay attention to.
Decoder with Attention

▶ Instead of using 1 fixed vector to encode all $x_1 \ldots x_m$, decoder decides which words to pay attention to.

▶ For $i = 1, 2, \ldots$,

$$h_{i}^{\phi} = \phi \left( e_{y_{i-1}} \oplus \left( \sum_{j=1}^{m} \alpha_{i,j} h_{j}^{\psi} \right), h_{i-1}^{\phi} \right)$$

$$p_{\Theta}(y|x_1 \ldots x_m, y_0 \ldots y_{i-1}) = \text{softmax}_{y}(f(h_{i}^{\phi}))$$
Attention Weights

\[
\sum_{j=1}^{m} \alpha_{i,j} h_j^\psi
\]

- \( \alpha_{i,j} \): Importance of \( x_j \) for predicting \( i \)-th translation
Attention Weights

\[
\sum_{j=1}^{m} \alpha_{i,j} h_j^\psi
\]

- \( \alpha_{i,j} \): Importance of \( x_j \) for predicting \( i \)-th translation

- Various options

\[
\beta_{i,j} = u^\top \tanh \left( Wh_i^\phi + V h_j^\psi \right)
\]
\[
\beta_{i,j} = \left( h_{i-1}^\phi \right)^\top h_j^\psi
\]
\[
\beta_{i,j} = \left( h_{i-1}^\phi \right)^\top Bh_j^\psi
\]

Typically take softmax to make them probabilities:

\[
(\alpha_{i,1} \ldots \alpha_{i,m}) = \text{softmax} (\beta_{i,1} \ldots \beta_{i,m})
\]
Assignment 4

- Part 1. Basic encoder-decoder with bidirectional encoder
  Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (Cho et al., 2014)

- Part 2. Decoder with attention
  Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et al., 2016)

- Extra credit
  - Initializing neural parameters with pre-trained word embeddings
  - Beam search instead of greedy decoding
  - Top 20 BLEU scores