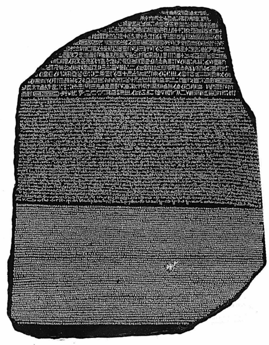
COMS 4705.H:
Neural Machine Translation

Karl Stratos

April 7, 2017

Machine Translation (MT)

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



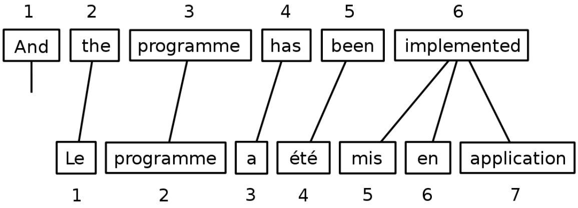
Some History

- 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)



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. . . .

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 propogation

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

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

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

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

  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*.

$$h_i = \phi\left(x_i, \phi\left(x_{i-1}, \phi\left(x_{i-2}, \cdots \phi\left(x_1, h_0\right) \cdots\right)\right)\right)$$

Variety 1: "Simple" RNN

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

$$h_i = \tanh\left(Ux_i + Vh_{i-1}\right)$$

Stacked Simple RNN

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

$$
\begin{aligned}
h_i^{(1)} &= \tanh\left(U^{(1)} x_i + V^{(1)} h_{i-1}^{(1)}\right) \\
h_i^{(2)} &= \tanh\left(U^{(2)} h_i^{(1)} + V^{(2)} h_{i-1}^{(2)}\right) \\
&\vdots \\
h_i^{(L)} &= \tanh\left(U^{(L)} h_i^{(L-1)} + V^{(L)} h_{i-1}^{(L)}\right)
\end{aligned}
$$

Stacked Simple RNN

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

$$
\begin{aligned}
h_i^{(1)} &= \tanh\left(U^{(1)} x_i + V^{(1)} h_{i-1}^{(1)}\right) \\
h_i^{(2)} &= \tanh\left(U^{(2)} h_i^{(1)} + V^{(2)} h_{i-1}^{(2)}\right) \\
&\vdots \\
h_i^{(L)} &= \tanh\left(U^{(L)} h_i^{(L-1)} + V^{(L)} h_{i-1}^{(L)}\right)
\end{aligned}
$$

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

$$
x_i \quad \begin{bmatrix} h_{i-1}^{(1)} \\ \vdots \\ h_{i-1}^{(L)} \end{bmatrix} \quad \mapsto \quad \begin{bmatrix} h_i^{(1)} \\ \vdots \\ h_i^{(L)} \end{bmatrix}
$$

Variety 2: Long Short-Term Memory (LSTM)

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

$$
\begin{aligned}
q_i &= \sigma\left(U^q x_i + V^q h_{i-1} + W^q c_{i-1}\right) \\
c_i &= (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x_i + V^c h_{i-1}\right) \\
o_i &= \sigma\left(U^o x_i + V^o h_{i-1} + W^o c_i\right) \\
h_i &= o_i \odot \tanh(c_i)
\end{aligned}
$$

Variety 2: Long Short-Term Memory (LSTM)

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

$$
\begin{aligned}
q_i &= \sigma\left(U^q x_i + V^q h_{i-1} + W^q c_{i-1}\right) \\
c_i &= (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x_i + V^c h_{i-1}\right) \\
o_i &= \sigma\left(U^o x_i + V^o h_{i-1} + W^o c_i\right) \\
h_i &= o_i \odot \tanh(c_i)
\end{aligned}
$$

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

$$
\begin{aligned}
q_i &= \sigma\left(U^q x_i + V^q h_{i-1} + W^q c_{i-1}\right) \\
c_i &= (1 - q_i) \odot c_{i-1} + q_i \odot \tanh\left(U^c x_i + V^c h_{i-1}\right) \\
o_i &= \sigma\left(U^o x_i + V^o h_{i-1} + W^o c_i\right) \\
h_i &= o_i \odot \tanh(c_i)
\end{aligned}
$$

- 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

Variety 3: Gated Recurrent Unit (GRU)

- Think: Simplified LSTM

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

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

Variety 3: Gated Recurrent Unit (GRU)

- Think: Simplified LSTM

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

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

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

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

- 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

```
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 $p(x_1 \dots x_m)$ over all sentences $x_1 \dots x_m$ in vocabulary V.

Goal: Design a *good* language model

$$
\begin{aligned}
p(\texttt{the dog barked}) &> p(\texttt{the cat barked}) \\
&> p(\texttt{dog the barked}) \\
&> p(\texttt{oqc shgwqw\#w 1g0})
\end{aligned}
$$

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

Basic RNN Language Model

Denote all model parameters by Θ

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

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

$$* \; x_1 \ldots x_m \; \texttt{STOP}$$

Distribution over Words and Sentences

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

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

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

Distribution over Words and Sentences

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

$$h_i = \phi(e_{x_{i-1}}, h_{i-1})$$
$$p_\Theta(x|x_0 \ldots x_{i-1}) = \mathsf{softmax}_x(f(h_i))$$

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

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

No Markov assumption.

Training

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

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

Training

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

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

Leave this to DyNet...

```
loss.backward()
trainer.update()
```

Aside: Sentence Generation

We can sample a random sentence S from the model.

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

$$
\begin{aligned}
h &\leftarrow \phi(e_{S[-1]}, h) \\
x &\sim \mathsf{softmax}\left(f(h)\right) \\
S &\leftarrow S + [x]
\end{aligned}
$$

   until $x = \texttt{STOP}$.

Overview

Review of RNN

Review of RNN Language Model

Neural MT: Conditional RNN Language Model

Translation Problem

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

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

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

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

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

$$\text{개가 짖었다} \quad \Longrightarrow \quad \text{the dog barked}$$

Evaluating Machine Translation

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

Evaluating Machine Translation

- T: human-translated sentences
- $\widehat{T}$: machine-translated sentences
- p_n: precision of n-grams in $\widehat{T}$ against n-grams in T (sentence-wise)

Evaluating Machine Translation

- T: human-translated sentences
- $\widehat{T}$: machine-translated sentences
- p_n: precision of n-grams in $\widehat{T}$ against n-grams in T (sentence-wise)
- BLEU: Controversial but popular scheme to automatically evaluate translation quality

$$\text{BLEU} = \min\left(1, \frac{\left|\widehat{T}\right|}{|T|}\right) \times \left(\prod_{n=1}^{4} p_n\right)^{\frac{1}{4}}$$

Translation Model: Conditional Language Model

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 $x_1 \ldots x_m \in V^{\text{src}}$.

Translation Model: Conditional Language Model

A **translation model** defines a probability distribution $p(y_1 \ldots y_n | x_1 \ldots x_m)$ over all sentences $y_1 \ldots y_n \in V^{\textsf{trg}}$ conditioning on any sentence $x_1 \ldots x_m \in V^{\textsf{src}}$.

Goal: Design a *good* translation model

$$
\begin{aligned}
p(\texttt{the dog barked}|\text{개가 짖었다}) &> p(\texttt{the cat barked}|\text{개가 짖었다}) \\
&> p(\texttt{dog the barked}|\text{개가 짖었다}) \\
&> p(\texttt{oqc shgwqw\#w 1g0}|\text{개가 짖었다})
\end{aligned}
$$

Translation Model: Conditional Language Model

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

$$
\begin{aligned}
p(\texttt{the dog barked}|\text{개가 짖었다}) &> p(\texttt{the cat barked}|\text{개가 짖었다}) \\
&> p(\texttt{dog the barked}|\text{개가 짖었다}) \\
&> p(\texttt{oqc shgwqw\#w 1g0}|\text{개가 짖었다})
\end{aligned}
$$

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

Basic Encoder-Decoder Framework

Model parameters

- Vector $e_x \in \mathbb{R}^d$ for every $x \in V^{\text{src}}$
- Vector $e_y \in \mathbb{R}^d$ for every $y \in V^{\text{trg}} \cup \{*\}$
- 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 $f : \mathbb{R}^{d'} \rightarrow \mathbb{R}^{|V^{\text{trg}}|+1}$

Basic idea

1. Transform $x_1 \ldots x_m \in V^{\text{src}}$ with ψ into some representation ξ.
2. Build a language model ϕ over V^{trg} conditioning on ξ.

Encoder

For $i = 1 \ldots m$,

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

Encoder

For $i = 1 \dots m$,

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

$$h_m^{\psi} = \psi\left(e_{x_m}, \psi\left(e_{x_{m-1}}, \psi\left(e_{x_{m-2}}, \cdots \psi\left(e_{x_1}, h_0^{\psi}\right) \cdots \right)\right)\right)$$

Decoder

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

Decoder

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

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

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

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

Decoder

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

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

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

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

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

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

Training

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

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

Training

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

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

Leave this to DyNet...

```
loss.backward()
trainer.update()
```

Greedy Translation

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

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

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

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 $S \leftarrow [*]$.

Greedy Translation

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

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

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

2. Initialize $h^{\phi} \leftarrow h_m^{\psi}$ and $S \leftarrow [*]$.
3. Keep repeating

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

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

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

   until $y = \texttt{STOP}$.

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 $S \leftarrow [*]$.
3. Keep repeating

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

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

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

until $y = \texttt{STOP}$.

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 $\psi^f : \mathbb{R}^d \times \mathbb{R}^{d'/2} \to \mathbb{R}^{d'/2}$
  2. Backward RNN $\psi^b : \mathbb{R}^d \times \mathbb{R}^{d'/2} \to \mathbb{R}^{d'/2}$

- For $i = 1 \dots m$, 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:

$$h_i^{\psi} = \begin{bmatrix} f_i \\ b_i \end{bmatrix}$$

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 $x_1 \dots x_m$, **decoder decides which words to pay attention to**.

- For $i = 1, 2, \dots$,

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

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

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

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

$$\beta_{i,j} = \left(h_{i-1}^{\phi}\right)^{\top} h_j^{\psi}$$

$$\beta_{i,j} = \left(h_{i-1}^{\phi}\right)^{\top} B h_j^{\psi}$$

Typically take softmax to make them probabilities:

$$(\alpha_{i,1} \ldots \alpha_{i,m}) = \text{softmax}\left(\beta_{i,1} \ldots \beta_{i,m}\right)$$

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