Basic Sequence-to-Sequence (with Attention)

Ondřej Bojar

March 12, 2020
Outline

• Basic NN building blocks for NMT.
• Processing Text.
• Neural Language Model.
• Vanilla Sequence-to-Sequence.
• Attention.

Many of the slides based on RANLP 2017 tutorial (Helcl and Bojar, 2017).
Encoder-Decoder Architecture

\[ f = (La, \text{ croissance, économique, s'est, ralentie, ces, dernières, années, .}) \]

\[ e = (\text{Economic, growth, has, slowed, down, in, recent, years, .}) \]

Basic NN Building Blocks
One Fully Connected Layer

- One fully-connected layer converts an input (column) vector $x$ to an output (column) vector $h$:

$$h = f(Wx + b), \tag{1}$$

- $W$ is a weight matrix of input columns and output rows,
- $b$ a bias vector of length of output,
- $f(\cdot)$ is a non-linearity applied usually elementwise.
One Layer $\tanh(Wx + b)$, 2D→2D

Skew: $W$

Transpose: $b$

Non-lin.: $\tanh$

Feed-Forward Neural Network

\[
\begin{align*}
    x & \downarrow \quad h_1 = f(W_1 x + b_1) \\
    h_1 & \uparrow \quad h_2 = f(W_2 h_1 + b_2) \\
    \vdots & \uparrow \quad h_n = f(W_n h_{n-1} + b_n) \\
    h_n & \uparrow \quad o = g(W_o h_n + b_o) \\
    E & = e(o, t) \quad \rightarrow \quad \frac{\partial E}{\partial W_o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial W_o}
\end{align*}
\]

blue: Training item (input \( x \), output \( t \)), red: Trainable parameters (\( W_1, b_1, \ldots \)).
Four Layers, Disentangling Spirals

Variable-Length Inputs and Outputs

Variable-length input can be handled by recurrent NNs:

- Processing one input symbol at a time.
  - Initial state $h_0 = (0)$ (or some sentence representation).
  - The same (trained) transformation $A$ used every time.

$$h_t = A(h_{t-1}, x_t) \quad (2)$$

- Unroll in time (up to a fixed length limit).

\[ \begin{array}{c}
\text{h}_0 \\
\downarrow \quad \downarrow \quad \downarrow \\
A \\
\downarrow \\
x_0
\end{array} \quad = \quad \begin{array}{c}
\text{h}_0 \\
\downarrow \quad \downarrow \\
A \\
\downarrow \\
x_0
\end{array} \quad \begin{array}{c}
\text{h}_1 \\
\downarrow \quad \downarrow \\
A \\
\downarrow \\
x_1
\end{array} \quad \begin{array}{c}
\text{h}_2 \\
\downarrow \\
A \\
\downarrow \\
x_2
\end{array} \quad \cdots \quad \begin{array}{c}
\text{h}_t \\
\downarrow \\
A \\
\downarrow \\
x_t
\end{array} \]
Vanilla RNN

\[ h_t = \tanh(W[h_{t-1}; x_t] + b) \]

- Vanishing gradient problem.
- Non-linear transformation always applied.
  \[ [h_{t-1}; x_t] \] is concatenation of \( h_{t-1} \) and \( x_t \)

\[ \Rightarrow \] Type theory: \( h_t \) and \( h_{t-1} \) live in different vector spaces.
LSTM and GRU Cells for RNN

- LSTM, Long Short-Term Memory Cells (Hochreiter and Schmidhuber, 1997).
- GRU, Gated Recurrent Unit Cells (Chung et al., 2014):

\[
\begin{align*}
    z_t &= \sigma (W_z[h_{t-1}; x_t] + b_z) \\
    r_t &= \sigma (W_r[h_{t-1}; x_t] + b_r) \\
    \tilde{h}_t &= \tanh (W[r_t \odot h_{t-1}; x_t]) \\
    h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{align*}
\]

- Gates control:
  - what to use from input \(x_t\) (GRU: everything),
  - what to use from hidden state \(h_{t-1}\) (reset gate \(r_t\)),
  - what to put into output (update gate \(z_t\))
- Linear “information highway” preserved.
  \(\Rightarrow\) All states \(h_t\) belong to the same vector space.
Processing Text
From Categorical Words to Numbers

• Map each word to a vector of 0s and 1s (“1-hot repr.”):
  \[ \text{cat} \mapsto (0, 0, \ldots, 0, 1, 0, \ldots, 0) \]

• Sentence is then a matrix:

\[
\begin{array}{cccccccc}
& \text{the} & \text{cat} & \text{is} & \text{on} & \text{the} & \text{mat} \\
\uparrow & a & 0 & 0 & 0 & 0 & 0 & 0 \\
& \text{about} & 0 & 0 & 0 & 0 & 0 & 0 \\
& \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
& \text{cat} & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{Vocabulary size:} & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\text{1.3M English} & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\text{2.2M Czech} & \text{is} & 0 & 0 & 1 & 0 & 0 & 0 \\
& \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
& \text{on} & 0 & 0 & 0 & 1 & 0 & 0 \\
& \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
& \text{the} & 1 & 0 & 0 & 0 & 1 & 0 \\
& \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\downarrow & \text{zebra} & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Sub-Words to Reduce Vocabulary Size

- SMT struggles with productive morphology (>1M wordforms).
  nejneobhodpodárovávatelnějšími, Donaudampfschifffahrtsgesellschaftskapitän
- NMT can handle only 30–80k dictionaries.

⇒ Resort to sub-word units.

<table>
<thead>
<tr>
<th></th>
<th>český politik svezly migranty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syllables</td>
<td>český po li tik sve zl mig ran ty</td>
</tr>
<tr>
<td>Morphemes</td>
<td>český politik s vez l migrant y</td>
</tr>
<tr>
<td>Char Pairs</td>
<td>český po li ti k sve zl mi gr an ty</td>
</tr>
<tr>
<td>Chars</td>
<td>český politik sve zl mig ran ty</td>
</tr>
<tr>
<td>BPE 30k</td>
<td>český politik vez migrant ty</td>
</tr>
</tbody>
</table>

BPE (Byte-Pair Encoding, (Sennrich et al., 2016)) or Google’s wordpieces (Wu et al., 2016) and Tensor2Tensor’s SubwordTextEncoder use $n$ most common substrings (incl. frequent words).
Word (Actually Token) Embeddings

- Idea: Map each token to a dense vector in continuous space.
- Result: 300–2000 dimensions instead of 1–2M.
  - The dimensions have no clear interpretation.
- The “embedding” is the mapping.
  - Technically, the first layer of NNs for NLP is the matrix that maps 1-hot input to the first layer.

- Embeddings are trained for each particular task.
  - Sentence classification (sentiment analysis, etc.)
  - Neural language modelling.
  - The famous word2vec (Mikolov et al., 2013):
    - CBOW: Predict the word from its four neighbours.
    - Skip-gram: Predict likely neighbours given the word.
  - End-to-end neural MT.
Output: Softmax over Vocabulary

Outputs of the RNN are:

1. Projected (scaled up) to the size of the vocabulary $V$,
2. Normalized with softmax.

⇒ Distribution over all possible target tokens.

$$l(w)_t = W_l h_t + b_l$$

- $l(w)_t =$ logits/energies for word $w$ in time $t$
- $W_l$: weight matrix (hidden state $\times$ voc. size)
  ... this is big.

$$p(w)_t = \frac{\exp l(w)_t}{\sum_{w' \in V} \exp l(w')_t}$$

- Softmax normalization: $\frac{\exp \cdot}{\sum \exp \cdot}$
  ... this is costly.

- Tricks what to do with it
  (negative sampling, hierarchical softmax)
  – not frequently used
Neural Language Modeling
RNN Language Model

• Train RNN as a **classifier for next words** (unlimited history):

```
<s> w1 w2 w3 w4
p(w1) p(w2) p(w3) p(w4)
```

• Can be used:
  • To estimate sentence probability / perplexity.
  • To sample from the distribution:

```
<s> ~w1 ~w2 ~w3 ~w4 ~w5
p(w1) p(w2) p(w3) p(w4) p(w5)
```
Two Views on RNN LM

- RNN is a for loop / functional map over sequential data
- all outputs are conditional distributions
  \[ \rightarrow \text{probabilistic distribution over sequences of words} \]

\[
P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_{i-1}, \ldots, w_1)
\]
Bidirectional RNN for Input

- Read the input sentence from both sides.
- Concatenate hidden states from each direction.
- Every $h_i$ stores information about the whole sentence.
Bidirectional RNN for Input

- read the input sentence from both sides
Bidirectional RNN for Input

- read the input sentence from both sides
- concatenate hidden states from each direction
- every $h_i$ stores information about the whole sentence
Encoder-Decoder Architecture
Encoder-Decoder Architecture

- exploits the conditional LM scheme

Encoder-Decoder Architecture

• exploits the conditional LM scheme
• two networks

Encoder-Decoder Architecture

- exploits the conditional LM scheme
- two networks
  1. a network processing the input sentence into a single vector representation (encoder)

Encoder-Decoder Architecture

- exploits the conditional LM scheme
- two networks
  1. a network processing the input sentence into a single vector representation (encoder)
  2. a neural language model initialized with the output of the encoder (decoder)

Encoder-Decoder Model – Image

<x1> x2 x3 x4
~y1 ~y2 ~y3 ~y4 ~y5

source language input + target language LM
Encoder-Decoder Model – Image

source language input + target language LM
state = np.zeros(sent_repr_size)
for w in input_words:
    input_embedding = source_embeddings[w]
    state, _ = enc_cell(state, input_embedding)

last_w = "<s>
while last_w != "</s>":
    last_w_embedding = target_embeddings[last_w]
    state, dec_output = dec_cell(state, last_w_embedding)
    logits = output_projection(dec_output)
    last_w = np.argmax(logits)
    yield last_w
Encoder-Decoder Model – Formal Notation

Data
input tokens (source language) \( x = (x_1, \ldots, x_{T_x}) \)
output tokens (target language) \( y = (y_1, \ldots, y_{T_y}) \)
Encoder-Decoder Model – Formal Notation

Data
input tokens (source language) \( \mathbf{x} = (x_1, \ldots, x_{T_x}) \)
output tokens (target language) \( \mathbf{y} = (y_1, \ldots, y_{T_y}) \)

Encoder
initial state \( h_0 \equiv \mathbf{0} \)
j-th state \( h_j = \text{RNN}_{\text{enc}}(h_{j-1}, x_j) = \tanh(U_e h_{j-1} + W_e E_e x_j + b_e) \)
final state \( h_{T_x} \)
Data
input tokens (source language) \( \mathbf{x} = (x_1, \ldots, x_{T_x}) \)
output tokens (target language) \( \mathbf{y} = (y_1, \ldots, y_{T_y}) \)

Encoder
initial state \( h_0 \equiv \mathbf{0} \)
\( j \)-th state \( h_j = \text{RNN}_{\text{enc}}(h_{j-1}, x_j) = \tanh(U_e h_{j-1} + W_e E_e x_j + b_e) \)
final state \( h_{T_x} \)

Decoder
initial state \( s_0 = h_{T_x} \)
\( i \)-th decoder state \( s_i = \text{RNN}_{\text{dec}}(s_{i-1}, \hat{y}_{i-1}) = \tanh(U_d s_{i-1} + W_d E_d \hat{y}_{i-1} + b_d) \)
\( i \)-th word score \( t_i = \tanh(U_o s_i + W_o E_d \hat{y}_{i-1} + b_o) \) (“output projection”)
output \( \hat{y}_i = \arg\max V_o t_i \)
For output word $y_i$ we have:

- estimated conditional distribution $\hat{p}_i = \frac{\exp t_i}{\sum \exp t_j}$ (softmax function)
Encoder-Decoder: Training Objective

For output word $y_i$ we have:

- estimated conditional distribution $\hat{p}_i = \frac{\exp t_i}{\sum \exp t_j}$ (softmax function)
- unknown true distribution $p_i$, we lay $p_i \equiv 1 [y_i]$
Encoder-Decoder: Training Objective

For output word $y_i$ we have:

- estimated conditional distribution $\hat{p}_i = \frac{\exp t_i}{\sum \exp t_j}$ (softmax function)
- unknown true distribution $p_i$, we lay $p_i \equiv 1[y_i]

Cross entropy $\approx$ distance of $\hat{p}$ and $p$:

$$\mathcal{L} = H(\hat{p}, p) = \mathbb{E}_p (-\log \hat{p})$$
Encoder-Decoder: Training Objective

For output word $y_i$ we have:

- estimated conditional distribution $\hat{p}_i = \frac{\exp t_i}{\sum \exp t_j}$ (softmax function)
- unknown true distribution $p_i$, we lay $p_i \equiv 1[y_i]$ 

Cross entropy $\approx$ distance of $\hat{p}$ and $p$:

$$\mathcal{L} = H(\hat{p}, p) = E_p (-\log \hat{p}) = - \sum_{v \in V} p(v) \log \hat{p}(v) = - \log \hat{p}(y_i)$$

See https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
Encoder-Decoder: Training Objective

For output word $y_i$ we have:

- estimated conditional distribution $\hat{p}_i = \frac{\exp t_i}{\sum \exp t_j}$ (softmax function)
- unknown true distribution $p_i$, we lay $p_i \equiv 1 [y_i]$

Cross entropy $\approx$ distance of $\hat{p}$ and $p$:

$$
\mathcal{L} = H(\hat{p}, p) = \mathbb{E}_p (-\log \hat{p}) = - \sum_{v \in V} p(v) \log \hat{p}(v) = - \log \hat{p}(y_i)
$$

...computing $\frac{\partial \mathcal{L}}{\partial t_i}$ is quite simple

See [https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/](https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/)
For output word $y_i$ we have:

- estimated conditional distribution $\hat{p}_i = \frac{\exp t_i}{\sum \exp t_j}$ (softmax function)
- unknown true distribution $p_i$, we lay $p_i \equiv 1 [y_i]$

Cross entropy $\approx$ distance of $\hat{p}$ and $p$:

$$\mathcal{L} = H(\hat{p}, p) = \mathbf{E}_p (-\log \hat{p}) = - \sum_{v \in V} p(v) \log \hat{p}(v) = - \log \hat{p}(y_i)$$

...computing $\frac{\partial \mathcal{L}}{\partial t_i}$ is quite simple

See https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/

...but we expect the model to produce the exact word at the exact position!
Implementation: Runtime vs. Training

runtime: \( \hat{y}_j \) (decoded) \( \times \) training: \( y_j \) (ground truth)

\[
\begin{align*}
\text{x1} \quad \text{x2} \quad \text{x3} \quad \text{x4} \\
\text{~y1} \quad \text{~y2} \quad \text{~y3} \quad \text{~y4} \quad \text{~y5} \\
\text{y1} \quad \text{y2} \quad \text{y3} \quad \text{y4} \\
\text{loss}
\end{align*}
\]
Encoder-Decoder Architecture

Decoding
Greedy Decoding

- In each step, the model computes a distribution over the vocabulary \( V \) (given source \( x \), the previous outputs \( h \), and the model parameters \( \theta \)).

\[
p(y|h) = g(x, h, \theta)
\]
In each step, the model computes a distribution over the vocabulary $V$ (given source $x$, the previous outputs $h$, and the model parameters $\theta$):

$$p(y|h) = g(x, h, \theta)$$

In greedy decoding:

$$y^* = \arg\max_{y \in V} p(y|h)$$
Greedy Decoding

• In each step, the model computes a distribution over the vocabulary $V$ (given source $x$, the previous outputs $h$, and the model parameters $\theta$).

$$p(y|h) = g(x, h, \theta)$$

• In greedy decoding:

$$y^* = \arg\max_{y \in V} p(y|h)$$

• Repeat, until an end-of-sentence symbol ($\langle /s \rangle$) is decoded.
• **Pros:**
  • Fast and memory-efficient
  • Gives reasonable results

• **Cons:**
  • We are interested in the most probable sentence:

\[ (y^*)_{i=0}^N = \underset{(y)_{i=0}^N}{\text{argmax}} \, p(y_0, \ldots, y_N|h) \]

• Other methods: better results for the cost of a slow-down.
Beam Search

Instead of taking the \texttt{argmax} in every step, keep a list (or \texttt{beam}) of $k$-best scoring hypotheses.
Beam Search

- Instead of taking the arg max in every step, keep a list (or beam) of $k$-best scoring hypotheses.
- Hypothesis = partially decoded sentence $\rightarrow$ score
Beam Search

• Instead of taking the \texttt{arg max} in every step, keep a list (or \texttt{beam}) of \(k\)-best scoring hypotheses.

• Hypothesis = partially decoded sentence \(\rightarrow\) score

• Hypothesis score \(\psi_t = (y_1, y_2 \ldots, y_t)\) is the probability of the decoded sentence prefix up to \(t\)-th word.

\[
p(y_1, \ldots, y_t | h) = p(y_1 | h) \cdot \ldots \cdot p(y_t | y_1, \ldots, y_{t-1} | h)
\]
Beam Search

• Instead of taking the argmax in every step, keep a list (or beam) of $k$-best scoring hypotheses.
• Hypothesis = partially decoded sentence $\rightarrow$ score
• Hypothesis score $\psi_t = (y_1, y_2, \ldots, y_t)$ is the probability of the decoded sentence prefix up to $t$-th word.

$$p(y_1, \ldots, y_t | h) = p(y_1 | h) \cdot \cdots \cdot p(y_t | y_1, \ldots, y_{t-1} | h)$$

• Rule to compute the score of an extended hypothesis $\psi_t$:

$$p(\psi_t, y_{t+1} | h) = p(\psi_t | h) \cdot p(y_{t+1} | h)$$
Beam Search

- Instead of taking the $\text{argmax}$ in every step, keep a list (or beam) of $k$-best scoring hypotheses.
- Hypothesis = partially decoded sentence $\rightarrow$ score
- Hypothesis score $\psi_t = (y_1, y_2 \ldots, y_t)$ is the probability of the decoded sentence prefix up to $t$-th word.

$$p(y_1, \ldots, y_t|h) = p(y_1|h) \cdot \ldots \cdot p(y_t|y_1, \ldots, y_{t-1}|h)$$

- Rule to compute the score of an extended hypothesis $\psi_t$:

$$p(\psi_t, y_{t+1}|h) = p(\psi_t|h) \cdot p(y_{t+1}|h)$$

- Prefers shorter hypotheses $\rightarrow$ normalization necessary.
1. Begin with a single empty hypothesis in the beam.
Beam Search — Algorithm

1. Begin with a single empty hypothesis in the beam.
2. In each time step:
   2.1 Extend all hypotheses in the beam by $k$ most probable words (we call these candidate hypotheses).
   2.2 Sort the candidate hypotheses by their score.
   2.3 Put the best $k$ hypotheses in the new beam.
   2.4 If a hypothesis from the beam reaches the end-of-sentence symbol, we move it to the list of finished hypotheses.
3. Finish (1) at the final time step or (2) all $k$-best hypotheses end with $\text{}</s>$.
4. Sort the hypotheses by their score and output the best one.
Beam Search — Algorithm

1. Begin with a single empty hypothesis in the beam.
2. In each time step:
   2.1 Extend all hypotheses in the beam by $k$ most probable words (we call these candidate hypotheses).
3. Finish (1) at the final time step or (2) all $k$-best hypotheses end.
4. Sort the hypotheses by their score and output the best one.
Beam Search — Algorithm

1. Begin with a single empty hypothesis in the beam.

2. In each time step:
   
   2.1 Extend all hypotheses in the beam by $k$ most probable words (we call these candidate hypotheses).
   
   2.2 Sort the candidate hypotheses by their score.

3. Finish (1) at the final time step or (2) all $k$-best hypotheses end with <s>.

4. Sort the hypotheses by their score and output the best one.
Beam Search — Algorithm

1. Begin with a single empty hypothesis in the beam.
2. In each time step:
   2.1 Extend all hypotheses in the beam by $k$ most probable words (we call these candidate hypotheses).
   2.2 Sort the candidate hypotheses by their score.
   2.3 Put the best $k$ hypotheses in the new beam.
3. Finish (1) at the final time step or (2) all $k$-best hypotheses end with $\text{end}$.
Beam Search — Algorithm

1. Begin with a single empty hypothesis in the beam.

2. In each time step:
   
   2.1 Extend all hypotheses in the beam by $k$ most probable words (we call these candidate hypotheses).
   
   2.2 Sort the candidate hypotheses by their score.
   
   2.3 Put the best $k$ hypotheses in the new beam.
   
   2.4 If a hypothesis from the beam reaches the end-of-sentence symbol, we move it to the list of finished hypotheses.

3. Finish (1) at the final time step or (2) all $k$-best hypotheses end. 

4. Sort the hypotheses by their score and output the best one.
Beam Search — Algorithm

1. Begin with a single empty hypothesis in the beam.

2. In each time step:
   
   2.1 Extend all hypotheses in the beam by $k$ most probable words (we call these candidate hypotheses).
   
   2.2 Sort the candidate hypotheses by their score.
   
   2.3 Put the best $k$ hypotheses in the new beam.
   
   2.4 If a hypothesis from the beam reaches the end-of-sentence symbol, we move it to the list of finished hypotheses.

3. Finish (1) at the final time step or (2) all $k$-best hypotheses end with $</s>$.
1. Begin with a single empty hypothesis in the beam.

2. In each time step:
   
   2.1 Extend all hypotheses in the beam by \( k \) most probable words (we call these candidate hypotheses).
   
   2.2 Sort the candidate hypotheses by their score.
   
   2.3 Put the best \( k \) hypotheses in the new beam.
   
   2.4 If a hypothesis from the beam reaches the end-of-sentence symbol, we move it to the list of finished hypotheses.

3. Finish (1) at the final time step or (2) all \( k \)-best hypotheses end with \(</s>\).

4. Sort the hypotheses by their score and output the best one.
Attentive Sequence-to-Sequence Learning
Main Idea

Vanilla sequence-to-sequence was degrading with sentence length.

Goal of attention:
- Do not force the network to catch long-distance dependencies.
- Use decoder state only for:
  - target sentence dependencies (≈LM) and
  - a as query for the source word sentence
**Inspiration: Neural Turing Machine**

- general architecture for learning algorithmic tasks, finite imitation of Turing Machine

![Diagram of Neural Turing Machine](image)
Inspiration: Neural Turing Machine

- general architecture for learning algorithmic tasks, finite imitation of Turing Machine
- needs to address memory somehow – either by position or by content
Inspiration: Neural Turing Machine

- general architecture for learning algorithmic tasks, finite imitation of Turing Machine
- needs to address memory somehow – either by position or by content

- in fact does not work well
  - it hardly manages simple algorithmic tasks
Inspiration: Neural Turing Machine

• general architecture for learning algorithmic tasks, finite imitation of Turing Machine
• needs to address memory somehow – either by position or by content

• in fact does not work well
  – it hardly manages simple algorithmic tasks
• content-based addressing $\rightarrow$ attention
Attentive Sequence-to-Sequence Learning

Attention Mechanism
Attention Mechanism

\[
\begin{align*}
&\langle s \rangle \\
&\rightarrow h_0 \\
&\rightarrow h_1 \\
&\rightarrow h_2 \\
&\rightarrow h_3 \\
&\rightarrow h_4 \\
&\quad \vdots \\
&\rightarrow s_{i-1} \\
&\rightarrow s_i \\
&\rightarrow s_{i+1} \\
&\rightarrow \sim y_i \\
&\rightarrow \sim y_{i+1}
\end{align*}
\]
Attention Mechanism in Equations (1)

**Inputs:**
- decoder state $s_i$
- encoder states $h_j = [\overrightarrow{h}_j; \overleftarrow{h}_j] \quad \forall i = 1 \ldots T_x$

**Attention energies:**

$$e_{ij} = v_a^\top \tanh (W_a s_{i-1} + U_a h_j + b_a)$$

**Attention distribution:**

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

**Context vector:**

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$
Attention Mechanism in Equations (2)

Decoder state:

\[ s_i = \tanh(U_d s_{i-1} + W_d E_d \hat{y}_{i-1} + C c_i + b_d) \]

Output projection:

\[ t_i = \tanh(U_o s_i + W_o E_d \hat{y}_{i-1} + C_o c_i + b_o) \]

...context vector is mixed with the hidden state

Output distribution:

\[ p(y_i = k \mid s_i, y_{i-1}, c_i) \propto \exp(W_o t_i)_k + b_k \]
Attention Visualization

The agreement on the European Economic Area was signed in August 1992.

It should be noted that the marine environment is the least known of environments.
Attentive Sequence-to-Sequence Learning

Attention vs. Alignment
Attention vs. Alignment

Differences between attention model and word alignment used for phrase table generation:
Differences between attention model and word alignment used for phrase table generation:

Attention (NMT)    Alignment (SMT)
Attention vs. Alignment

Differences between attention model and word alignment used for phrase table generation:

Attention (NMT)  
Probabilistic

Alignment (SMT)  
Discrete
Attention vs. Alignment

Differences between attention model and word alignment used for phrase table generation:

**Attention (NMT)**  
Probabilistic  
Declarative

**Alignment (SMT)**  
Discrete  
Imperative
## Attention vs. Alignment

Differences between attention model and word alignment used for phrase table generation:

<table>
<thead>
<tr>
<th>Attention (NMT)</th>
<th>Alignment (SMT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilistic</td>
<td>Discrete</td>
</tr>
<tr>
<td>Declarative</td>
<td>Imperative</td>
</tr>
<tr>
<td>LM generates</td>
<td>LM discriminates</td>
</tr>
</tbody>
</table>
## Attention vs. Alignment

Differences between attention model and word alignment used for phrase table generation:

<table>
<thead>
<tr>
<th>Attention (NMT)</th>
<th>Alignment (SMT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probabilistic</td>
<td>Discrete</td>
</tr>
<tr>
<td>Declarative</td>
<td>Imperative</td>
</tr>
<tr>
<td>LM generates</td>
<td>LM discriminates</td>
</tr>
<tr>
<td>Learnt with translation</td>
<td>Prerequisite</td>
</tr>
</tbody>
</table>
Attention Off by One

The relationship between Obama and Netanyahu has been stretched for years.

- Attention can appear on the neighbouring token.

• To benefit from PBMT, append its output to NMT input.
• Standard attentional model will learns to follow both.

Attention over CNN for image classification:

- A woman is throwing a **frisbee** in a park.
- A **dog** is standing on a hardwood floor.
- A **stop** sign is on a road with a mountain in the background.
- A little **girl** sitting on a bed with a teddy bear.
- A group of **people** sitting on a boat in the water.
- A giraffe standing in a forest with **trees** in the background.

Encoder-Decoder vs. Attentive Models
Sutskever+ (2014) × Bahdanau+ (2014)

Two key papers on NMT in 2014:
- Bahdanau et al. (2015) Attention model,
- Sutskever et al. (2014) Seq2seq impressive empirical results:
  - Made researchers believe NMT is the way to go.
  - (Used reversed input.)

Evaluation on WMT14 EN → FR test set:

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU score</th>
</tr>
</thead>
<tbody>
<tr>
<td>vanilla SMT</td>
<td>33.0</td>
</tr>
<tr>
<td>tuned SMT</td>
<td>37.0</td>
</tr>
<tr>
<td>Sutskever et al.: reversed</td>
<td>30.6</td>
</tr>
<tr>
<td>&quot;&quot;-&quot;: ensemble + beam search</td>
<td>34.8</td>
</tr>
<tr>
<td>&quot;&quot;-&quot;: vanilla SMT rescoring</td>
<td>36.5</td>
</tr>
<tr>
<td>Bahdanau’s attention</td>
<td>28.5</td>
</tr>
</tbody>
</table>
Two key papers on NMT in 2014:

- Bahdanau et al. (2015) Attention model,
- Sutskever et al. (2014) Seq2seq impressive empirical results:
  - Made researchers believe NMT is the way to go.
  - (Used reversed input.)

Evaluation on WMT14 EN → FR test set:

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU score</th>
</tr>
</thead>
<tbody>
<tr>
<td>vanilla SMT</td>
<td>33.0</td>
</tr>
<tr>
<td>tuned SMT</td>
<td>37.0</td>
</tr>
<tr>
<td>Sutskever et al.: reversed</td>
<td>30.6</td>
</tr>
<tr>
<td>−”−: ensemble + beam search</td>
<td>34.8</td>
</tr>
<tr>
<td>−”−: vanilla SMT rescoring</td>
<td>36.5</td>
</tr>
<tr>
<td>Bahdanau’s attention</td>
<td>28.5</td>
</tr>
</tbody>
</table>

Why worse?
Sutskever et al. (2014) × Bahdanau et al. (2014)

<table>
<thead>
<tr>
<th>Vocabulary</th>
<th>Sutskever et al.</th>
<th>160k enc, 80k dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder</td>
<td>4× LSTM, 1,000 units</td>
<td>4× LSTM, 1,000 units</td>
</tr>
<tr>
<td>Decoder</td>
<td>1,000 dimensions</td>
<td>1,000 dimensions</td>
</tr>
<tr>
<td>Word Embeddings</td>
<td>7.5 epochs</td>
<td>620 dimensions</td>
</tr>
<tr>
<td>Training Time</td>
<td>5 epochs</td>
<td>7.5 epochs</td>
</tr>
</tbody>
</table>

Comparison with Bahdanau's model size:

<table>
<thead>
<tr>
<th>Method</th>
<th>BLEU Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoder-Decoder</td>
<td>13.9</td>
</tr>
<tr>
<td>Attention Model</td>
<td>28.5</td>
</tr>
</tbody>
</table>

**Method**

- *Sutskever et al.*
  - Vocabulary: 160k enc, 80k dec
  - Encoder: 4× LSTM, 1,000 units
  - Decoder: 4× LSTM, 1,000 units
  - Word Embeddings: 1,000 dimensions
  - Training Time: 7.5 epochs

- *Bahdanau et al.*
  - Vocabulary: 30k both
  - Encoder: bidi GRU, 2,000 units
  - Decoder: GRU, 1,000 units
  - Word Embeddings: 620 dimensions
  - Training Time: 5 epochs
Sutskever et al. Bahdanau et al.

<table>
<thead>
<tr>
<th>vocabulary</th>
<th>160k enc, 80k dec</th>
<th>30k both</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoder</td>
<td>$4 \times$ LSTM, 1,000 units</td>
<td>bidi GRU, 2,000</td>
</tr>
<tr>
<td>decoder</td>
<td>$4 \times$ LSTM, 1,000 units</td>
<td>GRU, 1,000 units</td>
</tr>
<tr>
<td>word embeddings</td>
<td>1,000 dimensions</td>
<td>620 dimensions</td>
</tr>
<tr>
<td>training time</td>
<td>7.5 epochs</td>
<td>5 epochs</td>
</tr>
</tbody>
</table>

Comparison with Bahdanau’s model size:

<table>
<thead>
<tr>
<th>method</th>
<th>BLEU score</th>
</tr>
</thead>
<tbody>
<tr>
<td>encoder-decoder</td>
<td>13.9</td>
</tr>
<tr>
<td>attention model</td>
<td>28.5</td>
</tr>
</tbody>
</table>
Summary

We discussed:

• Basic building blocks of NN for NMT.
  • Fully-connected, RNN, LSTM and GRU.
  • Output softmax.

• Neural LM.

• Sequence-to-sequence (two RNNs attached).
  • Architecture.
  • Training.
  • Decoding (Greedy vs. Beam)

• Attention (decoder attends to a mix on encoder states).


