Sequence-to-Sequence Learning using Recurrent Neural Networks

Jindřich Helcl, Jindřich Libovický

March 4, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

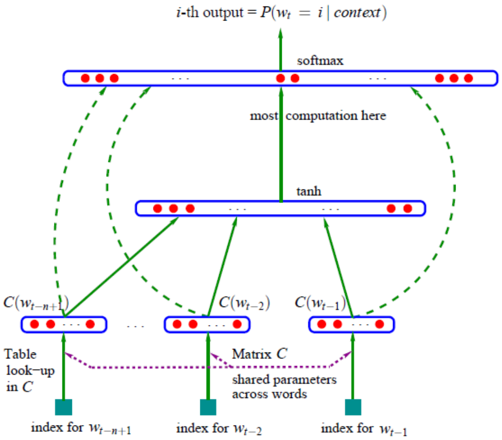
Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Outline

Symbol Embeddings

Discrete symbol vs. continuous representation

Simple task: predict next word given three previous:



Source: Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.
`http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf`

Embeddings

- Natural solution: one-hot vector (vector of vocabulary length with exactly one 1)
- It would mean a huge matrix every time a symbol is on the input
- Rather factorize this matrix and share the first part $\Rightarrow$ embeddings
- "Embeddings" because they embed discrete symbols into a continuous space

Embeddings

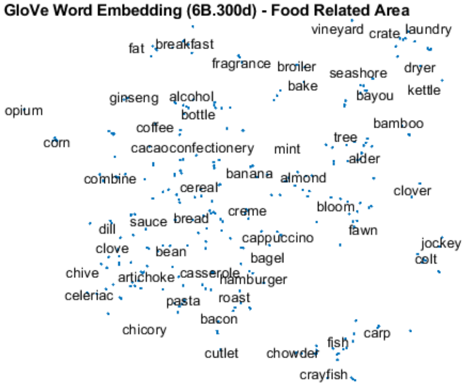
- Natural solution: one-hot vector (vector of vocabulary length with exactly one 1)
- It would mean a huge matrix every time a symbol is on the input
- Rather factorize this matrix and share the first part $\Rightarrow$ embeddings
- "Embeddings" because they embed discrete symbols into a continuous space

Think of training-related problems when using word embeddings...

Embeddings

- Natural solution: one-hot vector (vector of vocabulary length with exactly one 1)
- It would mean a huge matrix every time a symbol is on the input
- Rather factorize this matrix and share the first part $\Rightarrow$ embeddings
- "Embeddings" because they embed discrete symbols into a continuous space

Think of training-related problems when using word embeddings...
Embeddings get updated only rarely – only when a symbol appears.



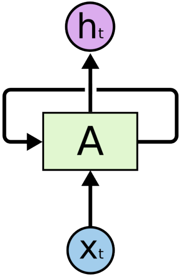
Source:
https://blogs.mathworks.com/loren/2017/09/21/math-with-words-word-embeddings-with-matlab-and-text-analytics-toolbox/

Recurrent Networks

Why RNNs

- `for` loops over sequential data
- the most frequently used type of network in NLP

General Formulation

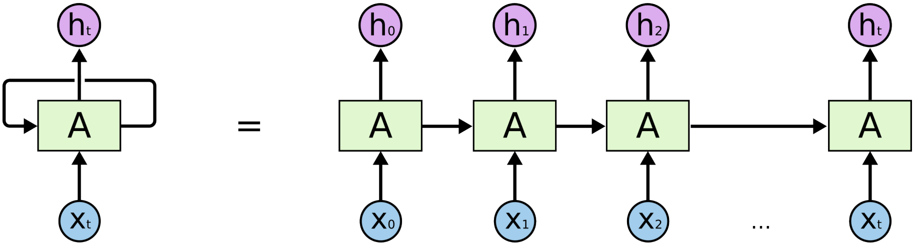


- inputs: $x_1, \dots, x_T$
- initial state h_0:
  - $\mathbf{0}$
  - result of previous computation
  - trainable parameter
- recurrent computation: $h_t = A(h_{t-1}, x_t)$

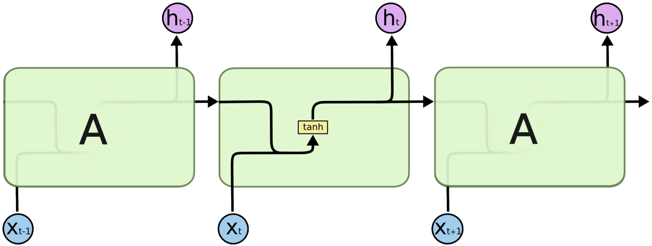
RNN as Imperative Code

```
def rnn(initial_state, inputs):
  prev_state = initial_state
  for x in inputs:
    new_state, output = rnn_cell(x, prev_state)
    prev_state = new_state
    yield output
```

RNN as a Fancy Image



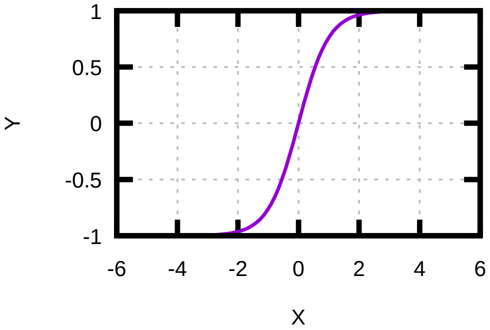
Vanilla RNN



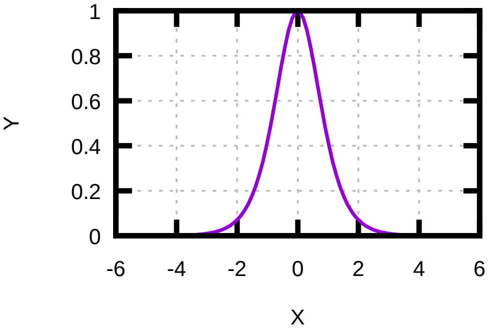
$$h_t = \tanh\left(W[h_{t-1}; x_t] + b\right)$$

- cannot propagate long-distance relations
- vanishing gradient problem

Vanishing Gradient Problem (1)

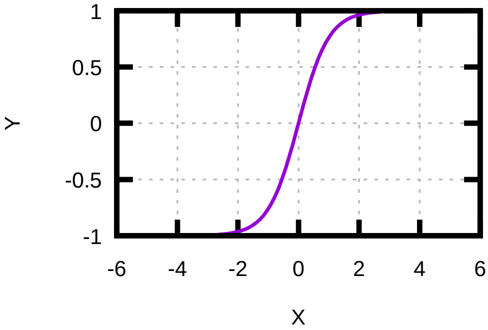


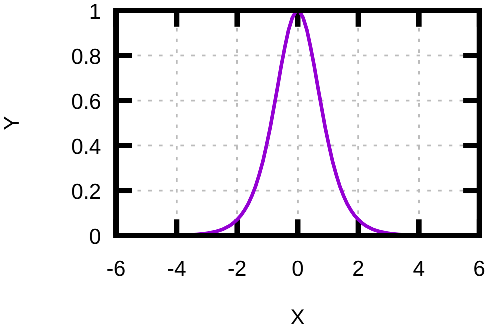
$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



$$\frac{\mathrm{d} \tanh x}{\mathrm{d} x} = 1 - \tanh^2 x \in (0, 1]$$

Vanishing Gradient Problem (1)



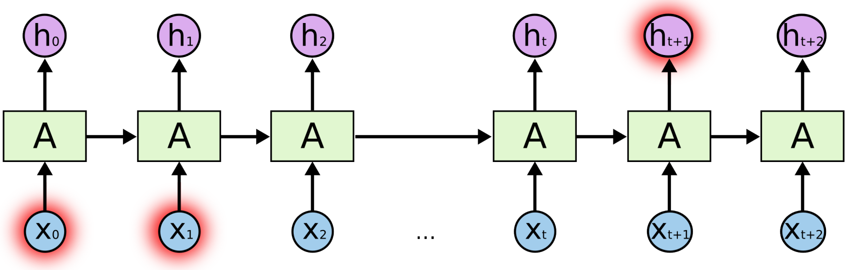


$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

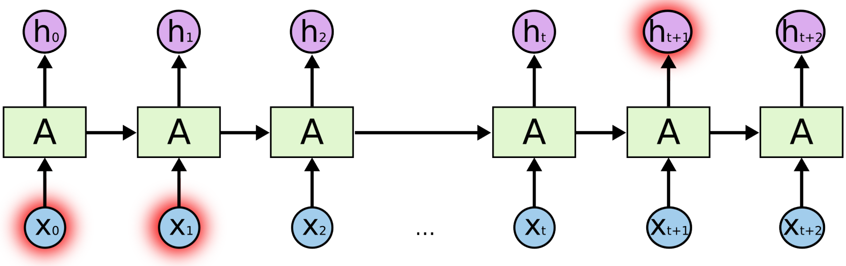
$$\frac{\mathrm{d} \tanh x}{\mathrm{d} x} = 1 - \tanh^2 x \in (0, 1]$$

Weights initialized $\sim \mathcal{N}(0, 1)$ to have gradients further from zero.

Vanishing Gradient Problem (2)

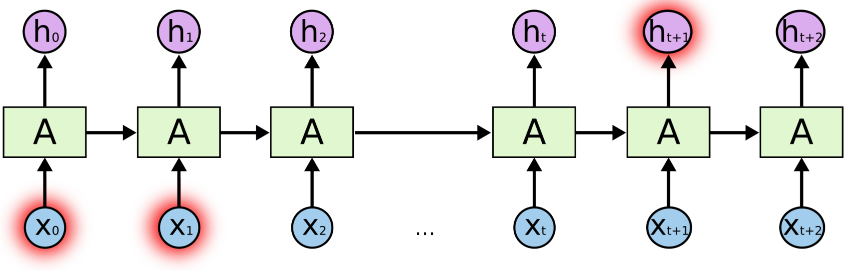


Vanishing Gradient Problem (2)



$$\frac{\partial E_{t+1}}{\partial b} =$$

Vanishing Gradient Problem (2)



$$\frac{\partial E_{t+1}}{\partial b} = \frac{\partial E_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial b}$$ (chain rule)

Vanishing Gradient Problem (3)

$$\frac{\partial h_t}{\partial b} =$$

Vanishing Gradient Problem (3)

$$\frac{\partial h_t}{\partial b} = \frac{\partial \tanh \overbrace{(W_h h_{t-1} + W_x x_t + b)}^{=z_t \text{ (activation)}}}{\partial b} \quad (\tanh' \text{ is derivative of } \tanh)$$

Vanishing Gradient Problem (3)

$$
\begin{aligned}
\frac{\partial h_t}{\partial b} &= \frac{\partial \tanh \overbrace{(W_h h_{t-1} + W_x x_t + b)}^{=z_t \text{ (activation)}}}{\partial b} \qquad (\tanh' \text{ is derivative of } \tanh) \\
&= \tanh'(z_t) \cdot \left(\frac{\partial W_h h_{t-1}}{\partial b} + \underbrace{\frac{\partial W_x x_t}{\partial b}}_{=0} + \underbrace{\frac{\partial b}{\partial b}}_{=1} \right)
\end{aligned}
$$

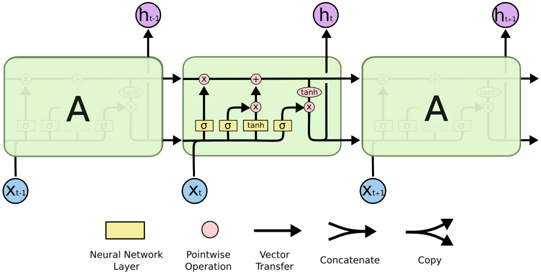
Vanishing Gradient Problem (3)

$$\begin{aligned}
\frac{\partial h_t}{\partial b} &= \frac{\partial \tanh \overbrace{(W_h h_{t-1} + W_x x_t + b)}^{= z_t \text{ (activation)}}}{\partial b} \qquad (\tanh' \text{ is derivative of } \tanh) \\
&= \tanh'(z_t) \cdot \left(\frac{\partial W_h h_{t-1}}{\partial b} + \underbrace{\frac{\partial W_x x_t}{\partial b}}_{=0} + \underbrace{\frac{\partial b}{\partial b}}_{=1} \right) \\
&= \underbrace{W}_{\sim \mathcal{N}(0,1)} \underbrace{\tanh'(z_t)}_{\in (0;1]} \frac{\partial h_{t-1}}{\partial b} + \tanh'(z_t)
\end{aligned}$$

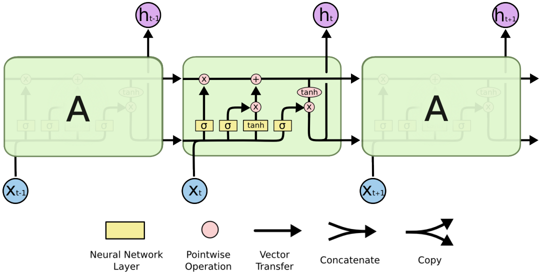
LSTMs

LSTM = Long short-term memory

LSTM = Long short-term memory



LSTM = Long short-term memory

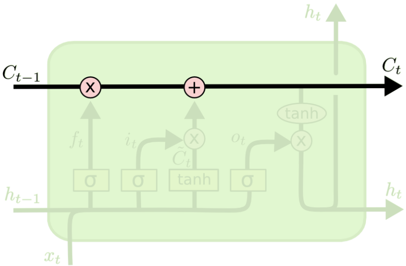


Control the gradient flow by explicitly gating:

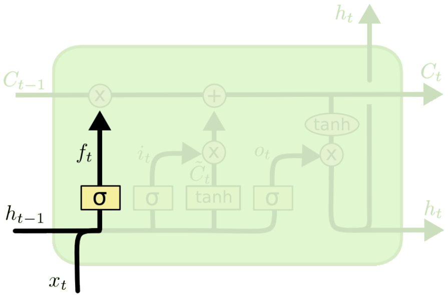
- what to use from input,
- what to use from hidden state,
- what to put on output

Hidden State

- two types of hidden states
- h_t — "public" hidden state, used an output
- c_t — "private" memory, no non-linearities on the way
  - direct flow of gradients (without multiplying by $\leq$ derivatives)
  - only vectors guaranteed to live in the same space are manipulated
- information highway metaphor



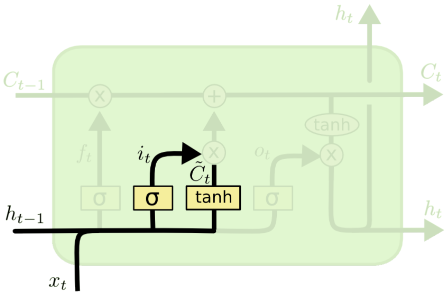
Forget Gate



$$f_t = \sigma \left(W_f [h_{t-1}; x_t] + b_f \right)$$

- based on input and previous state, decide what to forget from the memory

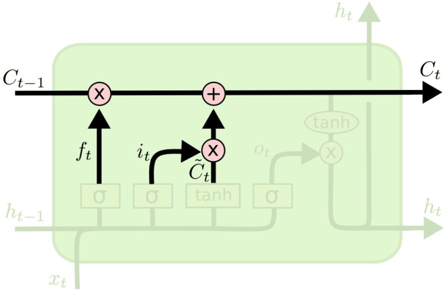
Input Gate



$$i_t = \sigma\left(W_i \cdot [h_{t-1}; x_t] + b_i\right)$$
$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}; x_t] + b_C\right)$$

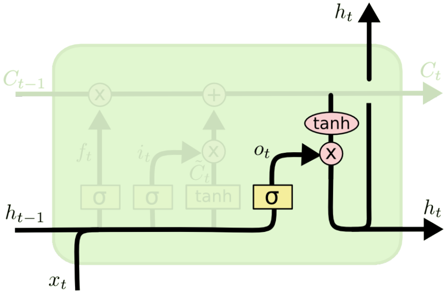
- $\tilde{C}$ — candidate what may want to add to the memory
- i_t — decide how much of the information we want to store

Cell State Update



$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

Output Gate



$$o_t = \sigma\left(W_o \cdot [h_{t-1}; x_t] + b_o\right)$$

$$h_t = o_t \odot \tanh C_t$$

Here we are!

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}; x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}; x_t] + b_i\right) \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}; x_t] + b_o\right) \\
\tilde{C}_t &= \tanh\left(W_c \cdot [h_{t-1}; x_t] + b_C\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh C_t
\end{aligned}
$$

Here we are!

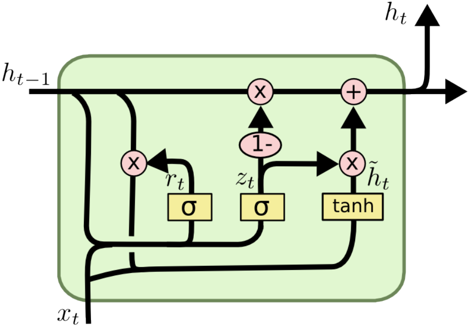
$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}; x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}; x_t] + b_i\right) \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}; x_t] + b_o\right) \\
\tilde{C}_t &= \tanh\left(W_c \cdot [h_{t-1}; x_t] + b_C\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh C_t
\end{aligned}
$$

How would you implement it efficiently?

Here we are!

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}; x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}; x_t] + b_i\right) \\
o_t &= \sigma\left(W_o \cdot [h_{t-1}; x_t] + b_o\right) \\
\tilde{C}_t &= \tanh\left(W_c \cdot [h_{t-1}; x_t] + b_C\right) \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \\
h_t &= o_t \odot \tanh C_t
\end{aligned}
$$

How would you implement it efficiently?
Compute all gates in a single matrix multiplication.



$$z_t = \sigma\left(W_z[h_{t-1}; x_t] + b_z\right)$$
$$r_t = \sigma\left(W_r[h_{t-1}; x_t] + b_r\right)$$
$$\tilde{h}_t = \tanh\left(W[r_t \odot h_{t-1}; x_t]\right)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

GRU and LSTM

LSTM

$$
\begin{aligned}
f_t &= \sigma\left(W_f[h_{t-1};x_t] + b_f\right)\\
i_t &= \sigma\left(W_i \cdot [h_{t-1};x_t] + b_i\right)\\
o_t &= \sigma\left(W_o \cdot [h_{t-1};x_t] + b_o\right)\\
\tilde{C}_t &= \tanh\left(W_c \cdot [h_{t-1};x_t] + b_C\right)\\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t\\
h_t &= o_t \odot \tanh C_t
\end{aligned}
$$

GRU

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

GRU or LSTM?

- GRU preserves the information highway property
- GRU has less parameters, should learn faster
- LSTM more general (although both Turing complete)
- empirical results: it's task-specific

Chung, Junyoung, et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:412.3555 (204).
Irie, Kazuki, et al. "LSTM, GRU, highway and a bit of attention: an empirical overview for language modeling in speech recognition." Interspeech, San Francisco, CA, USA (206).

Recurrent Networks

+

- correspond to intuition of sequential processing
- theoretically strong

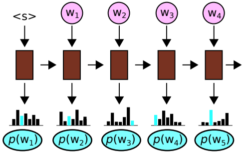
-

- cannot be parallelized, always need to wait for previous state

Neural Network Language Models

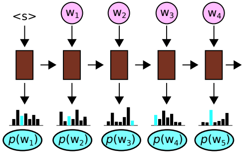
RNN Language Model

- Train RNN as classifier for next words (unlimited history)



RNN Language Model

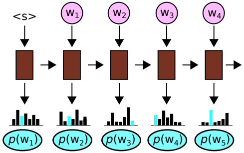
- Train RNN as classifier for next words (unlimited history)



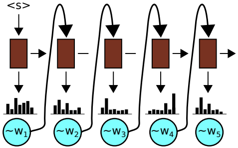
- Can be used to estimate sentence probability / perplexity $\rightarrow$ defines a distribution over sentences

RNN Language Model

- Train RNN as classifier for next words (unlimited history)



- Can be used to estimate sentence probability / perplexity $\rightarrow$ defines a distribution over sentences
- We can sample from the distribution



Two views on RNN LM

- RNN is a `for` loop (functional `map`) over sequential data
- All outputs are conditional distributions $\rightarrow$ probabilistic distribution over sequences of words:

$$P(w_1, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_{i-1}, \ldots, w_1)$$

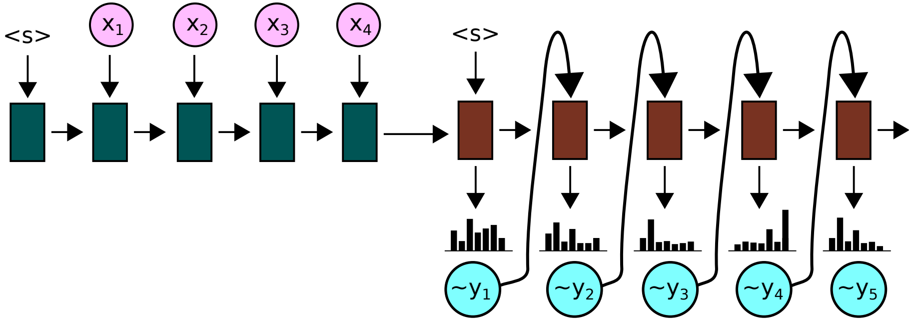
Vanilla Sequence-to-Sequence Model

Encoder-Decoder NMT

- 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*)

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

Encoder-Decoder – Image



Source language input + target language LM

Encoder-Decoder Model – Code

```
state = np.zeros(EMB_SIZE)
for w in input_words:
    input_embedding = source_embeddings[w]
    state, _ = enc_cell(state, input_embedding)

prev_w = "<s>"
while prev_w != "</s>":
    prev_w_embeding = target_embeddings[prev_w]
    state, dec_output = dec_cell(state, prev_w_embeding)
    logits = output_projection(dec_output)
    prev_w = np.argmax(logits)
    yield prev_w
```

Encoder-Decoder Model – Formal Notation

Data

input embeddings (source language)	$\mathbf{x} = (x_1, \dots, x_{T_x})$
output embeddings (target language)	$\mathbf{y} = (y_1, \dots, y_{T_y})$

Encoder-Decoder Model – Formal Notation

Data

input embeddings (source language)	$\mathbf{x} = (x_1, \dots, x_{T_x})$
output embeddings (target language)	$\mathbf{y} = (y_1, \dots, y_{T_y})$

Encoder

initial state	$h_0 \equiv \mathbf{0}$
j-th state	$h_j = \mathrm{RNN}_{\mathrm{enc}}(h_{j-1}, x_j)$
final state	h_{T_x}

Encoder-Decoder Model – Formal Notation

Data

input embeddings (source language)	$\mathbf{x} = (x_1, \dots, x_{T_x})$
output embeddings (target language)	$\mathbf{y} = (y_1, \dots, y_{T_y})$

Encoder

initial state	$h_0 \equiv \mathbf{0}$
j-th state	$h_j = \mathrm{RNN}_{\mathrm{enc}}(h_{j-1}, x_j)$
final state	h_{T_x}

Decoder

initial state	$s_0 = h_{T_x}$
i-th decoder state	$s_i = \mathrm{RNN}_{\mathrm{dec}}(s_{i-1}, y_i)$
i-th word score	$t_{i+1} = U_o s_{i+1} + V_o E y_i + b_o$, (or multi-layer projection)
output	$\hat{y}_{i+1} = \arg\max t_{i+1}$

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)

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 $p_i \equiv \mathbf{1}[y_i]$

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 $p_i \equiv \mathbf{1}\left[y_i\right]$

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

$$\mathcal{L} = H(\hat{p}, p) = \mathbf{E}_p\left(-\log \hat{p}\right)$$

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 $p_i \equiv \mathbf{1}\left[y_i\right]$

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

$$\mathcal{L} = H(\hat{p}, p) = \mathbf{E}_p\left(-\log \hat{p}\right) = -\log \hat{p}(y_i)$$

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 $p_i \equiv \mathbf{1}\left[y_i\right]$

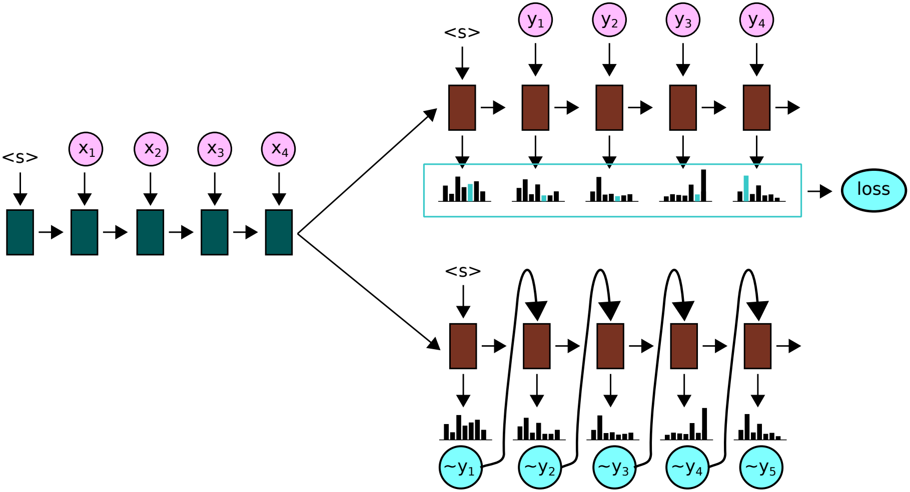
Cross entropy $\approx$ distance of $\hat{p}$ and p:

$$\mathcal{L} = H(\hat{p}, p) = \mathbf{E}_p \left(-\log \hat{p}\right) = -\log \hat{p}(y_i)$$

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

Implementation: Runtime vs. training

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



Sutskever et al., “Sequence-to-Sequence Learning with Neural Networks”, 2014

- Reverse input sequence
- Impressive empirical results – made researchers believe NMT is way to go

Evaluation on WMT14 EN $\rightarrow$ FR test set:

method	BLEU score
vanilla SMT	33.0
tuned SMT	37.0
Sutskever et al.: reversed	30.6
–”–: ensemble + beam search	34.8
–”–: vanilla SMT rescoring	36.5
Bahdanau’s attention	28.5

Sutskever et al., “Sequence-to-Sequence Learning with Neural Networks”, 2014

- Reverse input sequence
- Impressive empirical results – made researchers believe NMT is way to go

Evaluation on WMT14 EN $\rightarrow$ FR test set:

method	BLEU score
vanilla SMT	33.0
tuned SMT	37.0
Sutskever et al.: reversed	30.6
–”–: ensemble + beam search	34.8
–”–: vanilla SMT rescoring	36.5
Bahdanau’s attention	28.5

Why is better Bahdanau’s model worse?

Sutskever et al. × Bahdanau et al.

	Sutskever et al.	Bahdanau et al.

Sutskever et al. × Bahdanau et al.

	Sutskever et al.	**Bahdanau et al.**
vocabulary	160k enc, 80k dec	30k both

Sutskever et al. $\times$ Bahdanau et al.

	Sutskever et al.	**Bahdanau et al.**
vocabulary	160k enc, 80k dec	30k both
encoder	4$\times$ LSTM, 1,000 units	bidi GRU, 2,000

Sutskever et al. $\times$ Bahdanau et al.

	Sutskever et al.	**Bahdanau et al.**
vocabulary	160k enc, 80k dec	30k both
encoder	4$\times$ LSTM, 1,000 units	bidi GRU, 2,000
decoder	4$\times$ LSTM, 1,000 units	GRU, 1,000 units

Sutskever et al. $\times$ Bahdanau et al.

	Sutskever et al.	**Bahdanau et al.**
vocabulary	160k enc, 80k dec	30k both
encoder	4$\times$ LSTM, 1,000 units	bidi GRU, 2,000
decoder	4$\times$ LSTM, 1,000 units	GRU, 1,000 units
word embeddings	1,000 dimensions	620 dimensions

Sutskever et al. $\times$ Bahdanau et al.

	Sutskever et al.	Bahdanau et al.
vocabulary	160k enc, 80k dec	30k both
encoder	4$\times$ LSTM, 1,000 units	bidi GRU, 2,000
decoder	4$\times$ LSTM, 1,000 units	GRU, 1,000 units
word embeddings	1,000 dimensions	620 dimensions
training time	7.5 epochs	5 epochs

Sutskever et al. × Bahdanau et al.

	Sutskever et al.	**Bahdanau et al.**
vocabulary	160k enc, 80k dec	30k both
encoder	4× LSTM, 1,000 units	bidi GRU, 2,000
decoder	4× LSTM, 1,000 units	GRU, 1,000 units
word embeddings	1,000 dimensions	620 dimensions
training time	7.5 epochs	5 epochs

With Bahdanau's model size:

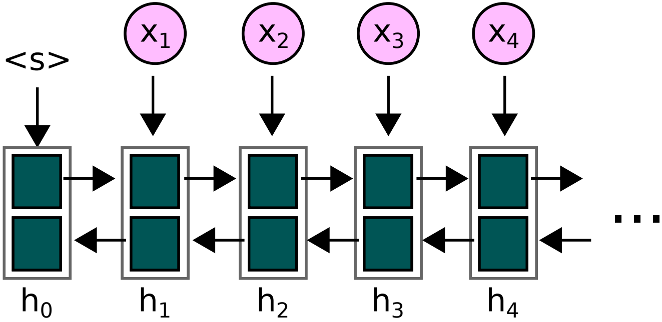
method	BLEU score
encoder-decoder	13.9
attention model	28.5

Attentive Sequence-to-Sequence Learning

Main Idea

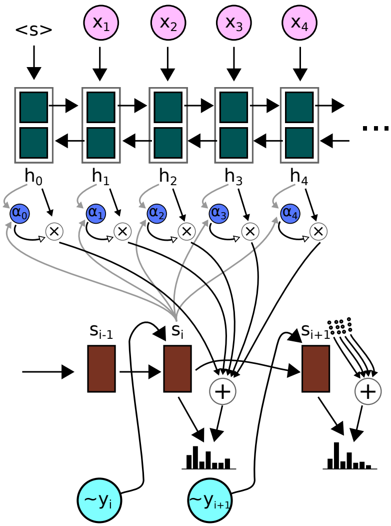
- Same as reversing input: do not force the network to catch long-distance dependencies
- Use decoder state only for target sentence dependencies and as a query for the source word sentence
- RNN can serve as LM — it can store the language context in their hidden states

Small Trick before We Start



Bidirectional network

- read the input sentence from both sides
- every h_i contains in fact information from the whole sentence



<s>
x_1
x_2
x_3
x_4
...
h_0
h_1
h_2
h_3
h_4
α_0
α_1
α_2
α_3
α_4
s_{i-1}
s_i
s_{i+1}
$\sim y_i$
$\sim y_{i+1}$

Attention Model in Equations (1)

Inputs:

decoder state s_i

encoder states $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right] \quad \forall i = 1 \ldots T_x$

Attention Model in Equations (1)

Inputs:

decoder state s_i

encoder states $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right] \quad \forall i = 1 \dots T_x$

Attention energies:

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

Attention Model in Equations (1)

Inputs:

decoder state s_i

encoder states $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right] \quad \forall i = 1 \dots T_x$

Attention energies:

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

Attention distribution:

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

Attention Model in Equations (1)

Inputs:

decoder state s_i

encoder states $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right] \quad \forall i = 1 \ldots T_x$

Attention energies:

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

Attention distribution:

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

Context vector:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Attention Model in Equations (2)

Output projection:

$$t_i = \mathrm{MLP}\left(U_o s_{i-1} + V_o E y_{i-1} + C_o c_i + b_o\right)$$

...context vector is mixed with the hidden state

Attention Model in Equations (2)

Output projection:

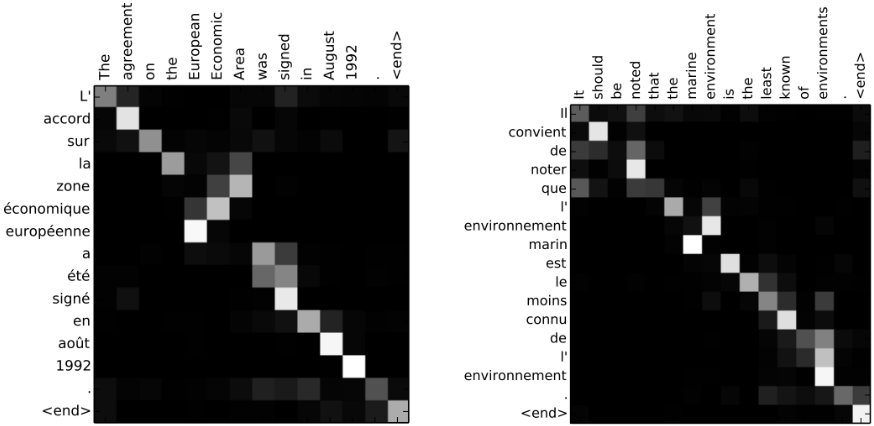
$$t_i = \mathrm{MLP}\left(U_o s_{i-1} + V_o E y_{i-1} + C_o c_i + b_o\right)$$

...context vector is mixed with the hidden state

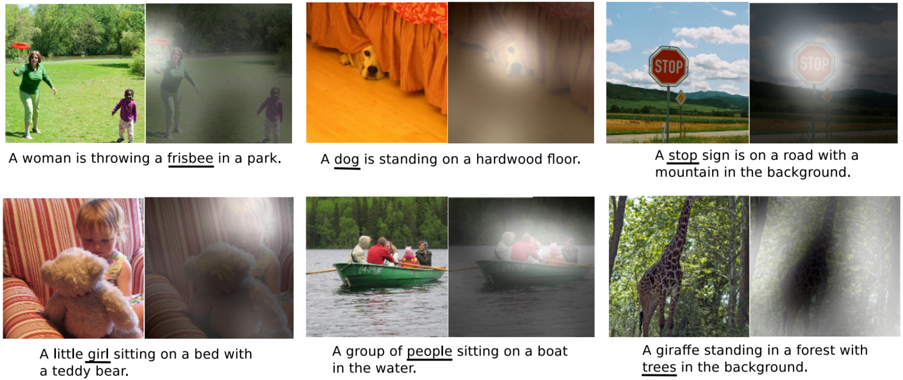
Output distribution:

$$p\left(y_i = w \middle| s_i, y_{i-1}, c_i\right) \propto \exp\left(W_o t_i\right)_w + b_w$$

Attention Visualization



Attention over CNN for image classification:



Source: Xu, Kelvin, et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention." *ICML*. Vol. 14. 2015.

Reading Assignment

Reading for the Next Week

Vaswani, Ashish, et al. "Attention is all you need." Advances in Neural Information Processing Systems. 2017.
`http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`

Question:
The model uses the scaled dot-product attention which is a non-parametric variant of the attention mechanism. Why do you think it is sufficient in this setup? Do you think it would work in the recurrent model as well?

The way the model processes the sequence is principally different from RNNs. Does it agree with your intuition of how language is processed?