Basic Sequence-to-Sequence (with Attention)

Ondřej Bojar

March 12, 2020

LANGTECH

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

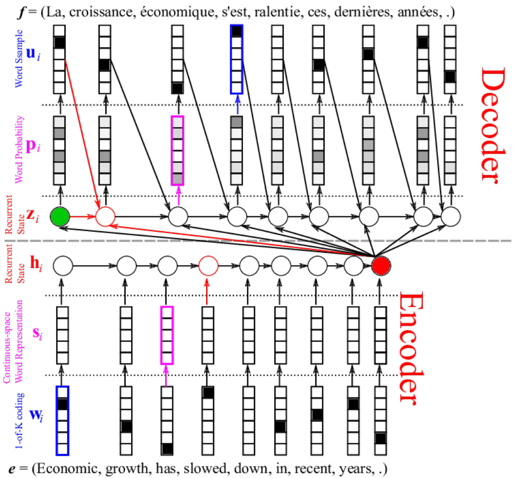
Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Outline

- 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



https://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-2/

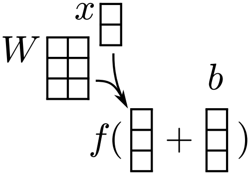
Basic NN Building Blocks

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.

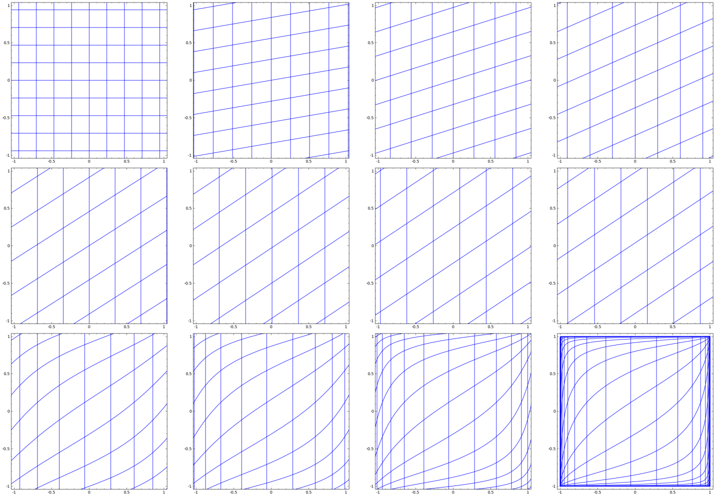


One Layer $\tanh(Wx + b)$, 2D→2D

Skew:
W

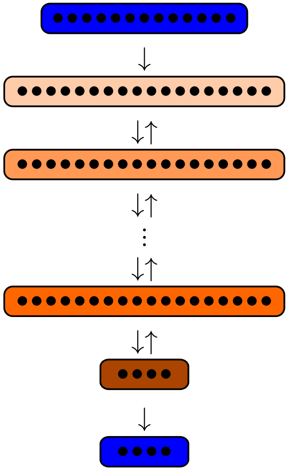
Transpose:
b

Non-lin.:
tanh



Animation by `http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`

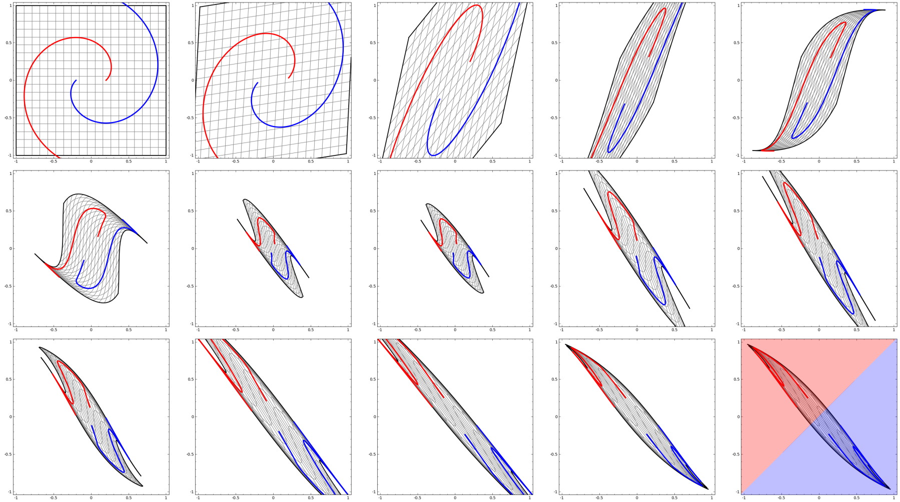
Feed-Forward Neural Network



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

blue: Training item (input x, output t), red: Trainable parameters ($W_1, b_1, \ldots$).

Four Layers, Disentagling Spirals



Animation by `http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`

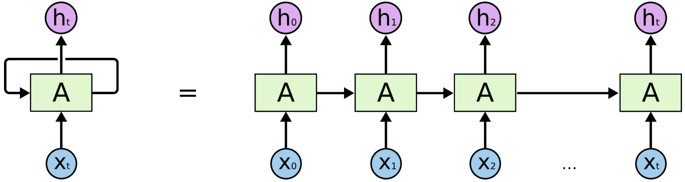
Variable-Length Inputs and Outputs

Variable-length input can be handled by recurrent NNs:

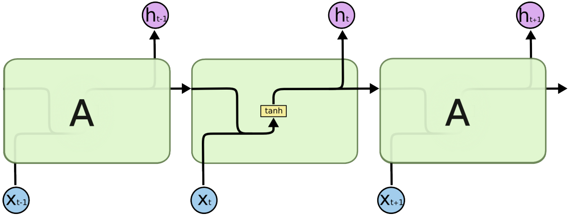
- Processing one input symbol at a time.
    - Initial state $h_0 = (0)$ (or some sentence representation).
    - The same (trained) transformation A used every time.

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

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



Vanilla RNN



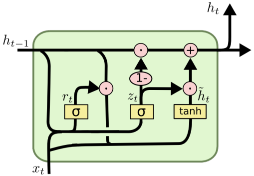
$$h_t = \mathsf{tanh}\left(W[h_{t-1}; x_t] + b\right) \tag{3}$$

$[h_{t-1}; x_t]$ is concatenation of h_{t-1} and x_t

- Vanishing gradient problem.
- Non-linear transformation always applied.
  $\Rightarrow$ 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):



$$z_t = \sigma\left(W_z[h_{t-1}; x_t] + b_z\right) \tag{4}$$
$$r_t = \sigma\left(W_r[h_{t-1}; x_t] + b_r\right) \tag{5}$$
$$\tilde{h}_t = \tanh\left(W[r_t \odot h_{t-1}; x_t]\right) \tag{6}$$
$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{7}$$

- 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."):

$$\text{cat} \mapsto (0, 0, \dots, 0, 1, 0, \dots, 0)$$

- Sentence is then a matrix:

		the	cat	is	on	the	mat
↑	a	0	0	0	0	0	0
	about	0	0	0	0	0	0
	...	...	...	...	...	...	...
	cat	0	**1**	0	0	0	0
Vocabulary size:	...	...	...	...	...	...	...
1.3M English	is	0	0	**1**	0	0	0
2.2M Czech	...	...	...	...	...	...	...
	on	0	0	0	**1**	0	0
	...	...	...	...	...	...	...
	the	**1**	0	0	0	**1**	0
	...	...	...	...	...	...	...
↓	zebra	0	0	0	0	0	0

Sub-Words to Reduce Vocabulary Size

- SMT struggles with productive morphology (>1M wordforms).

  nejneobhodpodařovávatelnějšími, Donaudampfschifffahrtsgesellschaftskapitän

- NMT can handle only 30–80k dictionaries.

⇒ Resort to sub-word units.

Orig	český politik svezl migranty
Syllables	čes ký ⊔ po li tik ⊔ sve zl ⊔ mig ran ty
Morphemes	česk ý ⊔ politik ⊔ s vez l ⊔ migrant y
Char Pairs	če sk ý ⊔ po li ti k ⊔ sv ez l ⊔ mi gr an ty
Chars	č e s k ý ⊔ p o l i t i k ⊔ s v e z l ⊔ m i g r a n t y
BPE 30k	český politik s@@ vez@@ l mi@@ granty

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.

$\Rightarrow$ Distribution over all possible target tokens.

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

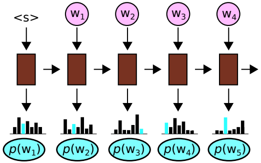
$$p(w)_t = \frac{\exp l(w)_t}{\sum_{w' \in V} \exp l(w')_t}$$

- $l(w)_t$ = logits/energies for word w in time t
- W_l: weight matrix (hidden state $\times$ voc. size)
  … this is **big**.
- Softmax normalization: $\frac{\exp \cdot}{\sum \exp \cdot}$
  … this is costly.
- Tricks what to do with it
  (negative sampling, hierarchical softmax)
  – not frequently used

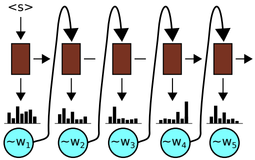
Neural Language Modeling

RNN Language Model

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



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

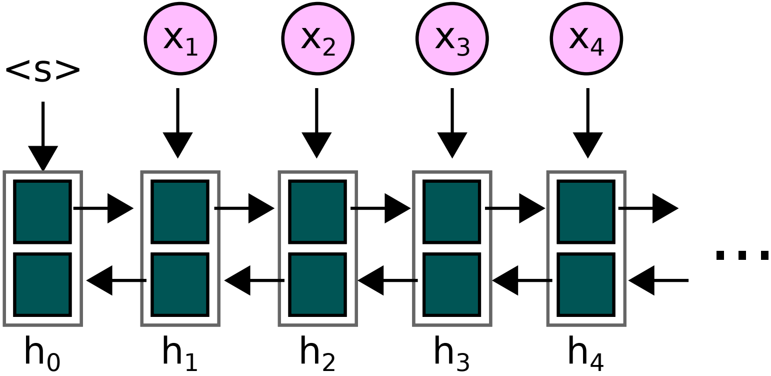


Two Views on RNN LM

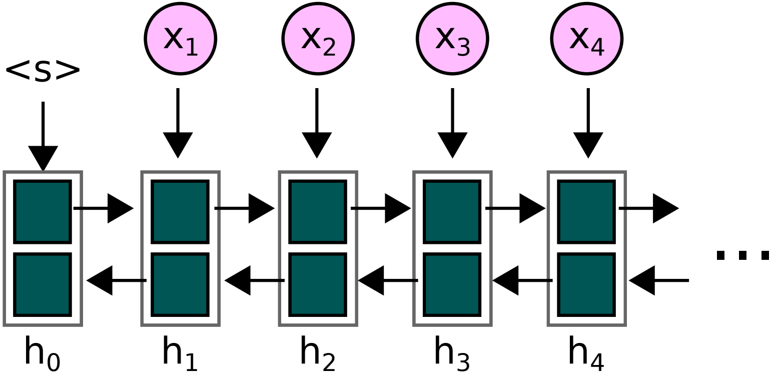
- RNN is a `for` loop / functional `map` over sequential data
- all outputs are conditional distributions
  → probabilistic distribution over sequences of words

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

Bidirectional RNN for Input

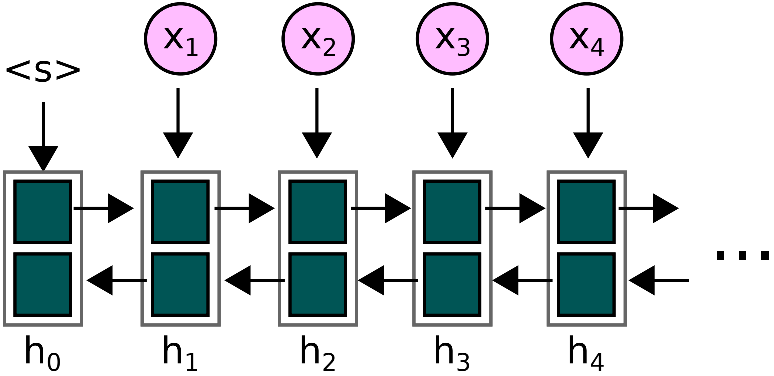


Bidirectional RNN for Input



- 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

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

Encoder-Decoder Architecture

- exploits the conditional LM scheme
- two networks

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

Encoder-Decoder Architecture

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

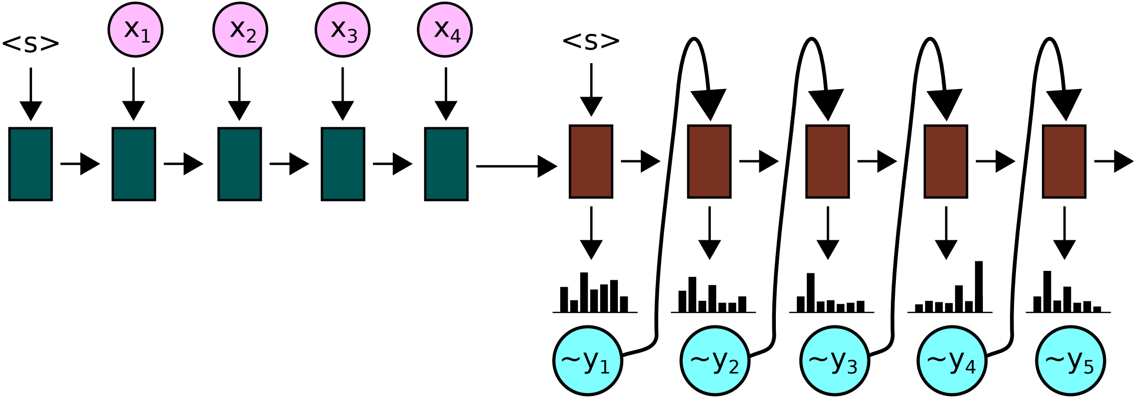
Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems. 2014.

Encoder-Decoder Architecture

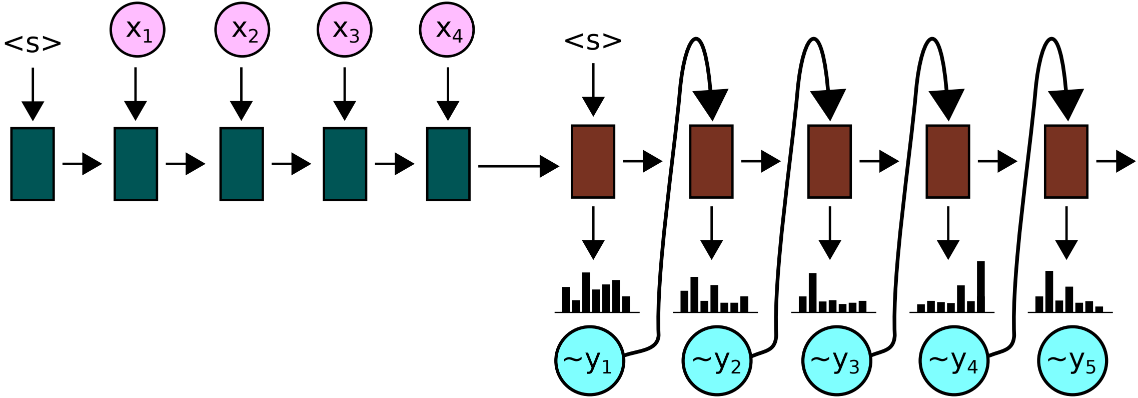
- exploits the conditional LM scheme
- two networks
    1. a network processing the input sentence into a single vector representation (encoder)
    2. a neural language model initialized with the output of the encoder (decoder)

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

Encoder-Decoder Model – Image



Encoder-Decoder Model – Image



source language input + target language LM

Encoder-Decoder Model – Code

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

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	$h_j = \mathrm{RNN}_{\mathrm{enc}}(h_{j-1}, x_j) = \tanh(U_e h_{j-1} + W_e E_e x_j + b_e)$
final state	h_{T_x}

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	$h_j = \mathrm{RNN}_{\mathrm{enc}}(h_{j-1}, x_j) = \tanh(U_e h_{j-1} + W_e E_e x_j + b_e)$
final state	h_{T_x}

Decoder

initial state	$s_0 = h_{T_x}$
i-th decoder state	$s_i = \mathrm{RNN}_{\mathrm{dec}}(s_{i-1}, \hat{y}_{i-1}) = \tanh(U_d s_{i-1} + W_d E_d \hat{y}_{i-1} + b_d)$
i-th word score	$t_i = \tanh(U_o s_i + W_o E_d \hat{y}_{i-1} + b_o)$ ("output projection")
output	$\hat{y}_i = \arg\max V_o t_i$

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

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:

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

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:

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

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

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/`

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

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

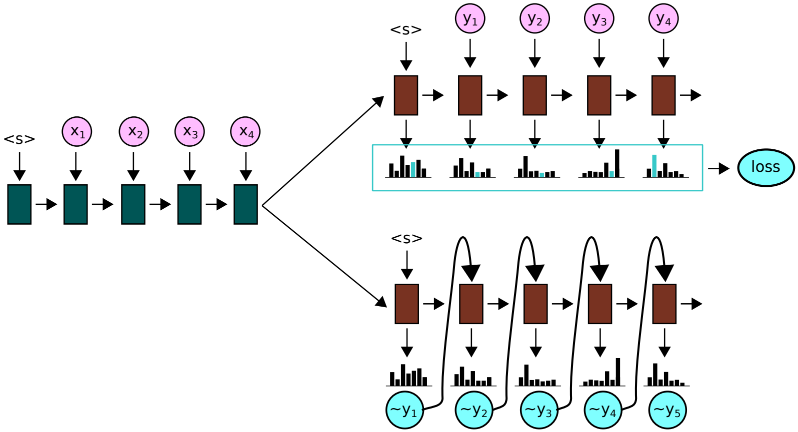
See `https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/`

...but we expect the model to produce

the exact word at the exact position!

Implementation: Runtime vs. Training

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



Encoder-Decoder Architecture

Decoding

Greedy Decoding

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

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

Greedy Decoding

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

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

- In greedy decoding:

$$y^* = \operatorname*{argmax}_{y \in V} p(y|h)$$

Greedy Decoding

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

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

- In greedy decoding:

$$y^* = \operatorname*{argmax}_{y \in V} p(y|h)$$

- Repeat, until an end-of-sentence symbol (</s>) is decoded.

Greedy Decoding — cont.

- Pros:
  - Fast and memory-efficient
  - Gives reasonable results
- Cons:
  - We are interested in the most probable sentence:

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

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

Beam Search

- Instead of taking the $\arg\max$ in every, step, keep a list (or 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 $\arg\max$ 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.

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

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
- 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 ψ_t:

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

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 → 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 ψ_t:

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

- Prefers shorter hypotheses → normalization necessary.

Beam Search — Algorithm

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:

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).

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.

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.

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.

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>.

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>.
4. Sort the hypotheses by their score and output the best one.

Attentive Sequence-to-Sequence Learning

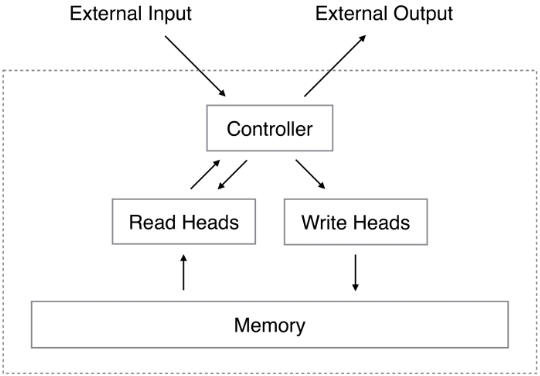
Main Idea

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

Goal of attention:

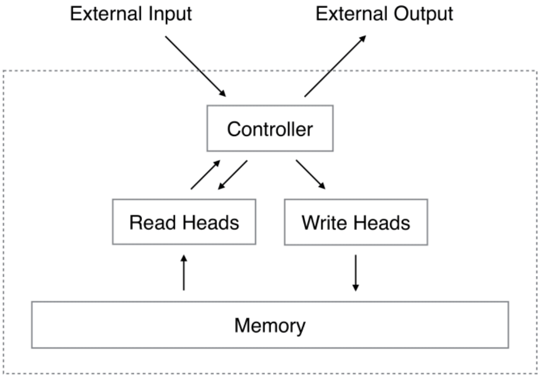
- Do not force the network to catch long-distance dependencies.
- Use decoder state only for:
  - target sentence dependencies (=LM) and
  - a as query for the source word sentence

Inspiration: Neural Turing Machine



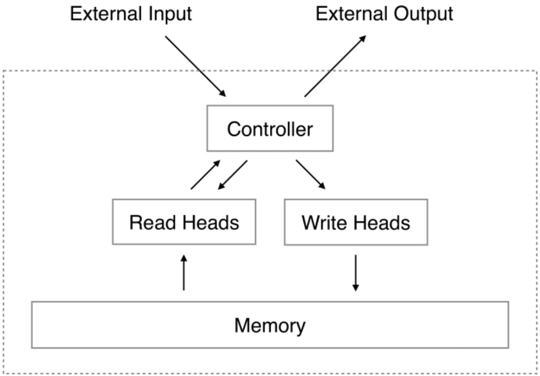
- general architecture for learning algorithmic tasks, finite imitation of Turing Machine

Inspiration: Neural Turing Machine



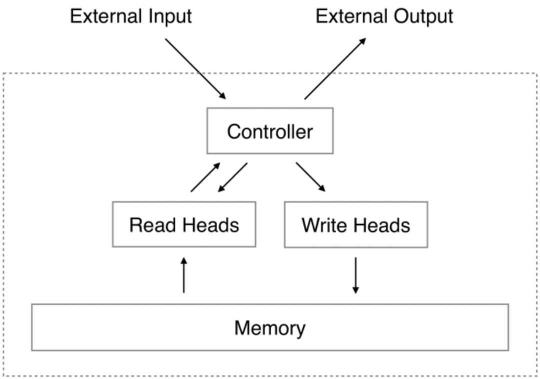
- general architecture for learning algorithmic tasks, finite imitation of Turing Machine
- needs to address memory somehow – either by position or by content

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



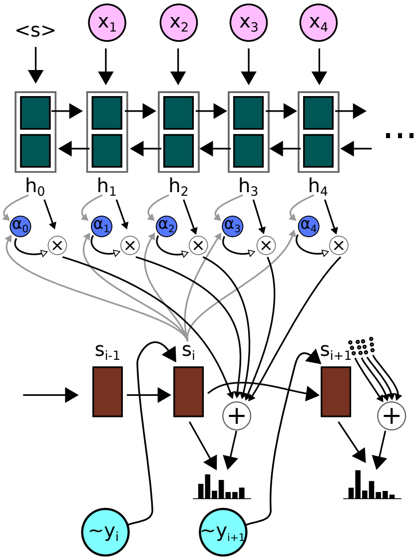
- general architecture for learning algorithmic tasks, finite imitation of Turing Machine
- needs to address memory somehow – either by position or by content

- 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



Attention Mechanism 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:

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

Context vector:

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

Attention Mechanism in Equations (2)

Decoder state:

$$s_i = \mathrm{tanh}(U_d s_{i-1} + W_d E_d \hat{y}_{i-1} + Cc_i + b_d)$$

Output projection:

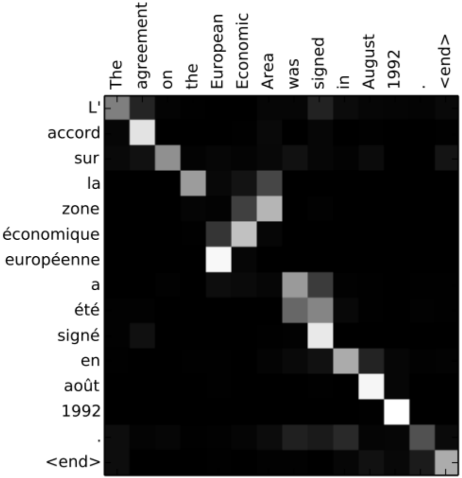
$$t_i = \mathrm{tanh}\left(U_o s_i + W_o E_d \hat{y}_{i-1} + C_o c_i + b_o\right)$$

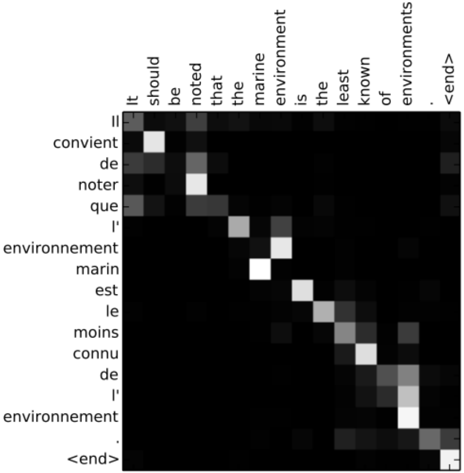
...context vector is mixed with the hidden state

Output distribution:

$$p\left(y_i = k \mid s_i, y_{i-1}, c_i\right) \propto \exp\left(W_o t_i\right)_k + b_k$$

Attention Visualization





Attentive Sequence-to-Sequence Learning

Attention vs. Alignment

Attention vs. Alignment

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

Attention vs. Alignment

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)	Alignment (SMT)
Probabilistic	Discrete

Attention vs. Alignment

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

Attention (NMT)	Alignment (SMT)
Probabilistic	Discrete
Declarative	Imperative

Attention vs. Alignment

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

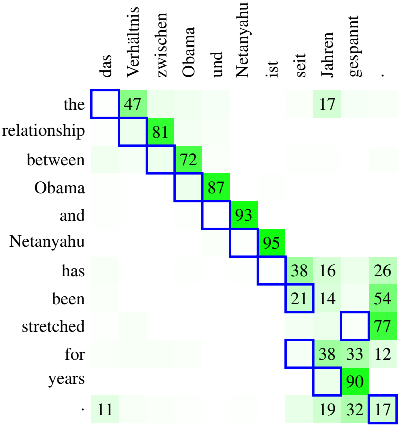
Attention (NMT)	Alignment (SMT)
Probabilistic	Discrete
Declarative	Imperative
LM generates	LM discriminates

Attention vs. Alignment

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

Attention (NMT)	Alignment (SMT)
Probabilistic	Discrete
Declarative	Imperative
LM generates	LM discriminates
Learnt with translation	Prerequisite

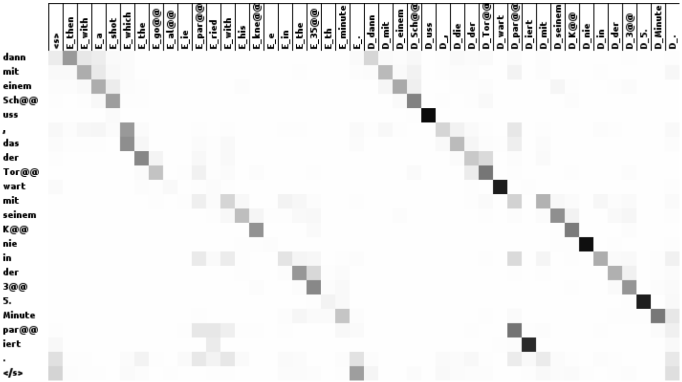
Attention Off by One



- Attention can appear on the neighbouring token.

Philipp Koehn and Rebecca Knowles (2017). Six Challenges for Neural Machine Translation. NMT workshop.

Attending to Two at Once



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

Jan Niehues, Eunah Cho, Thanh-Le Ha, and Alex Waibel. 2016. Pre-translation for neural machine translation.

Image Captioning

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.

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

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 $\rightarrow$ FR test set:

Model	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+ (2014) $\times$ 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 $\rightarrow$ FR test set:

Model	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 $\leftarrow$ **Why worse?**

Sutskever+ (2014) $\times$ Bahdanau+ (2014)

	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+ (2014) × Bahdanau+ (2014)

	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

Comparison with Bahdanau's model size:

method	BLEU score
encoder-decoder	13.9
attention model	28.5

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).

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In Proceedings of ICLR.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. CoRR, abs/1412.3555.

Jindřich Helcl and Ondřej Bojar. 2017. Deep Learning in MT / NMT. Tutorial at RANLP 2017, August.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. Neural Comput., 9(8):1735–1780, November.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. CoRR, abs/1301.3781.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In Advances in neural information processing systems, pages 3104–3112.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. CoRR, abs/1609.08144.