NPFL116 Compendium of Neural Machine Translation

Sequence-to-Sequence Learning

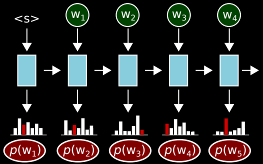
March 15, 2017

Jindřich Libovický, Jindřich Helcl

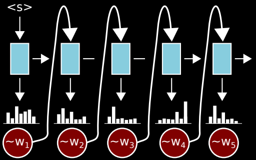
Charles Univeristy in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

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

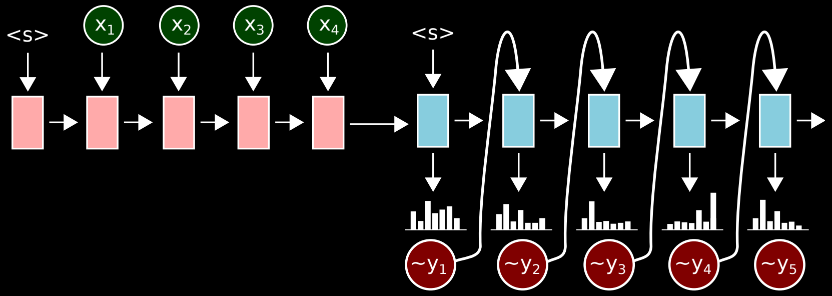


Two views on RNN LM

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

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

Encoder-Decoder – Image



source language LM + 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(encoder_state,
                        input_embedding)

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

Encoder-Decoder Model – Formal Notation

Data

input embeddings (source language) $\mathbf{x} = (x_1, \ldots, x_{T_x})$

output embeddings (target language) $\mathbf{y} = (y_1, \ldots, y_{T_y})$

Encoder

initial state $h_0 \equiv \mathbf{0}$

j-th state $h_j = \text{RNN}_{\text{enc}}(h_{j-1}, x_j)$

final state h_{T_x}

Decoder

initial state $s_0 = h_{T_x}$

i-th decoder state $s_i = \text{RNN}_{\text{dec}}(s_{i-1}, \hat{y}_i)$

i-th word score $t_{i+1} = U_o + V_o E y_i + b_o$,

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

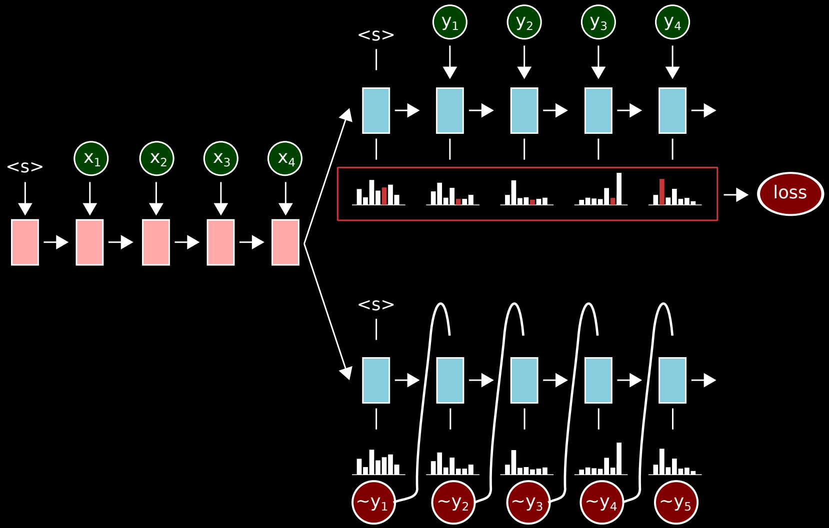
Cross entropy $\approx$ distance of $\hat{p}$ and p:

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

...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.

- 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. $\times$ 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

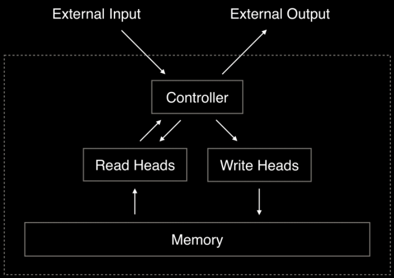
With Bahdanau's model size:

method	BLEU score
encoder-decoder	13.9
attention model	28.5

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 a as query for the source word sentence
- RNN can serve as LM — it can store the language context in their hidden states

Inspiration: Neural Turing Machine

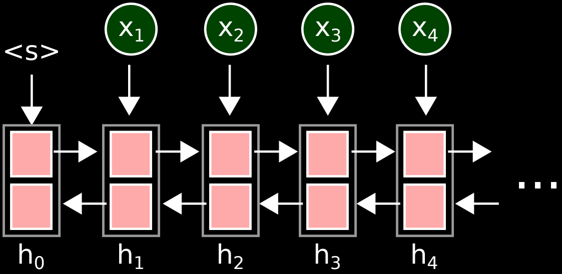


- general architecture for learning algorithmic tasks, finite imitation of Turing Machine
- needs to address memory somehow – either by position or by content

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

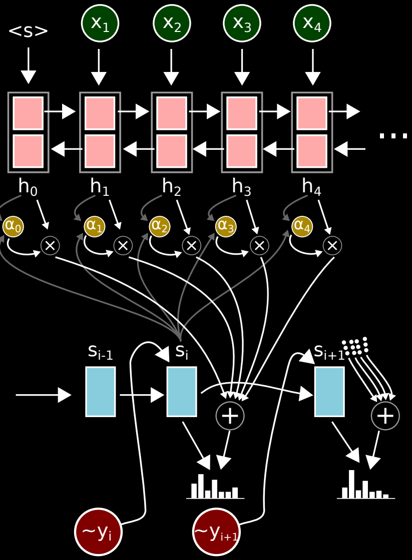
Small Trick before We Start

Bidirectional network



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

Attention Model



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:

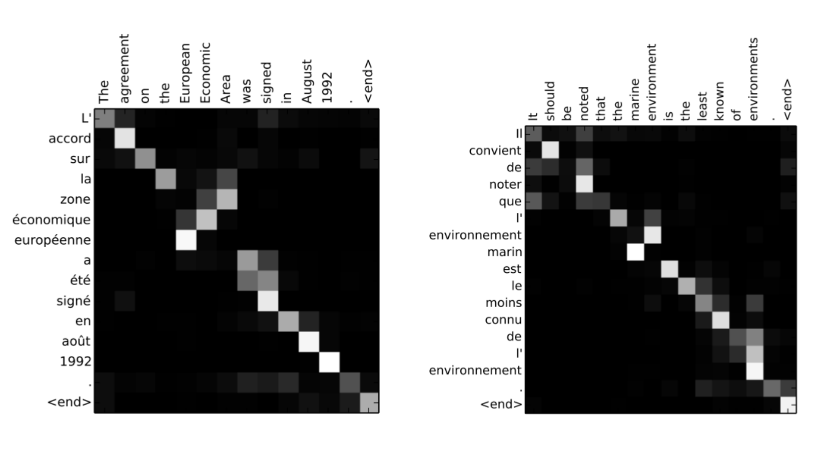
$$t_i = \mathrm{MLP}\left(U_o s_{i-1} + V_o E y_{i-1} + C_o c_i + b_o\right)$$

...attention is mixed with the hidden state

Output distribution:

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

Attention Visualization



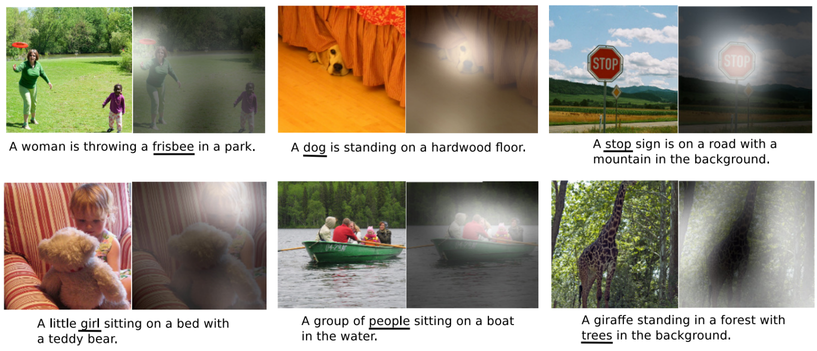
Attention vs. Alignment

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

attention (NMT)	alignment (SMT)
probabilistic	discrete
declarative	imperative
LM generates	LM discriminates

Image Captioning

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 for the Next Week

Chung, Junyoung, Kyunghyun Cho, and Yoshua Bengio. ”A character-level decoder without explicit segmentation for neural machine translation.” arXiv preprint arXiv:1603.06147 (2016).
`https://arxiv.org/pdf/1603.06147.pdf`

Question:
What are the reasons authors do not use character-level encoder? How would you improve the architecture such that it would allow character level encoding?