Introduction to Neural Machine Translation & Recurrent Neural Nets

Jindřich Helcl, Jindřich Libovický

February 24, 2022




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

What is Machine Translation?

What is Machine Translation?

Time for discussion...

What We Think...

- MT does not care what translation is
- We believe people know what translation is and that it is captured in the data
- We evaluate how well we can mimic what humans do when they translate

From Statistical to Neural MT

Statistical

- Meaning of a sentence can be decomposed into phrases.
- Adequacy and fluency can be modeled separately.

Neural

- Sentences are sequence of words (or smaller discrete units).
- (And maybe something about words having independent meaning and compositionality, but this is difficult.)

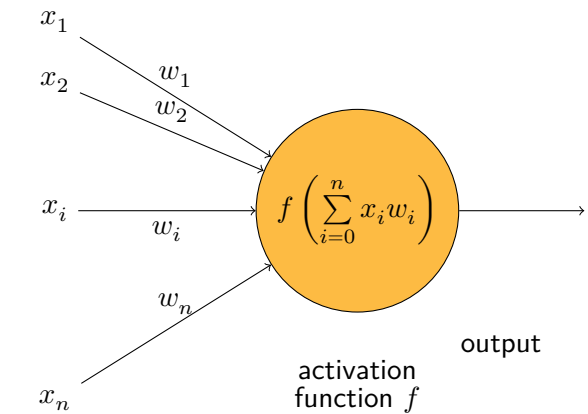
Neural Model are Black Boxes

⚠ Disclaimer ⚠

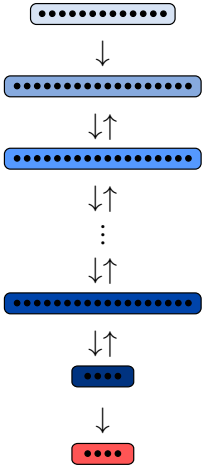
- Don't trust much the intuitions presented there: NNs can learn things different than we assume.
- All concepts used are from one half technical concepts, from the other half lose metaphors.

Deep Learning

Single Neuron



Neural Network



$$x$$

$$h_1 = f(W_1 x + b_1)$$

$$h_2 = f(W_2 h_1 + b_2)$$

$$h_n = f(W_n h_{n-1} + b_n)$$

$$o = g(W_o h_n + b_o)$$

$$E = e(o, t)$$

$$\rightarrow$$

$$\frac{\partial E}{\partial W_o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial W_o}$$

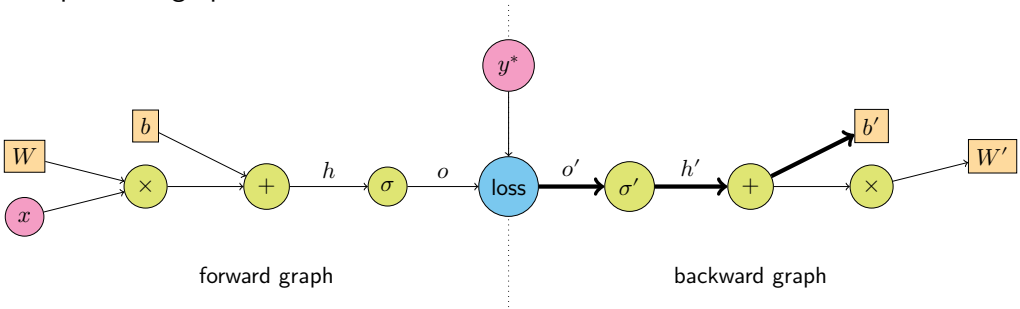
$$\frac{\partial E}{\partial o}$$

Implementation

Logistic regression:

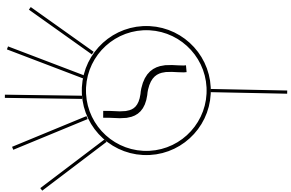
$$y = \sigma\left(Wx + b\right) \tag{1}$$

Computation graph:



Building Blocks

Single Neuron



- Computational model from 1940's
- Adds weighted inputs and transforms to input

Layer

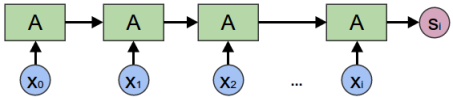
$$f(Wx + b)$$

... f nonlinearity, W ...weight matrix, b ...bias

- Having the network in layers allows using matrix multiplication
- Allows GPU acceleration
- Vector space interpretations

Encoder & Decoder

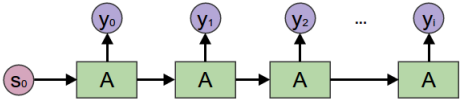
Encoder:



Functional fold (reduce) with function

`foldl a s xs`

Decoder:



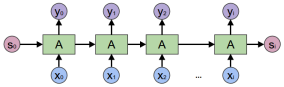
Inverse operation – functional unfold

`unfoldr a s`

Source: Colah's blog (`http://colah.github.io/posts/2015-09-NN-Types-FP/`)

RNNs & Convolutions

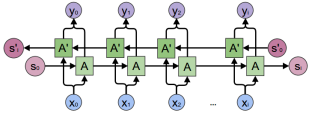
General RNN:



Map with accumulator

`mapAccumR a s xs`

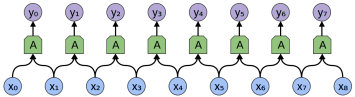
Bidirectional RNN:



Zip left and right accumulating map

`zip (mapAccumR a s xs) (mapAccumL a' s' xs)`

Convolution:



Zip neighbors and apply function

`zipWith a xs (tail xs)`

Source: Colah's blog (`http://colah.github.io/posts/2015-09-NN-Types-FP/`)

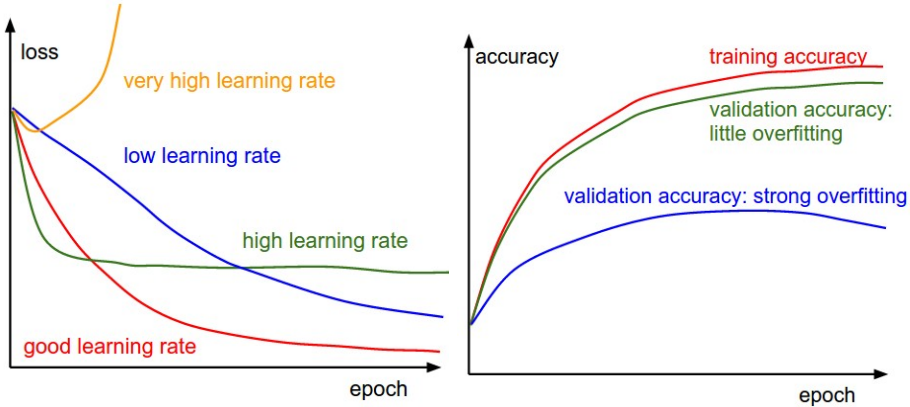
Optimization

- Data is constant, network is treated as function of parameters
- Differentiable error is function of parameters as well
- Clever variants of gradient descent algorithm

Deep Learning as Alchemy

- No rigorous manual for developing a good deep learning model – just rules of thumb
- Unclear how to interpret the weights the network has learned
- No theory that is able to predict results of experiments (as in physics), there are only experiments

Watching Learning Curves



Source: Convolutional Neural Networks for Visual Recognition at Stanford University (`http://cs231n.github.io/neural-networks-3/`)

Machine Translation and Deep Learning

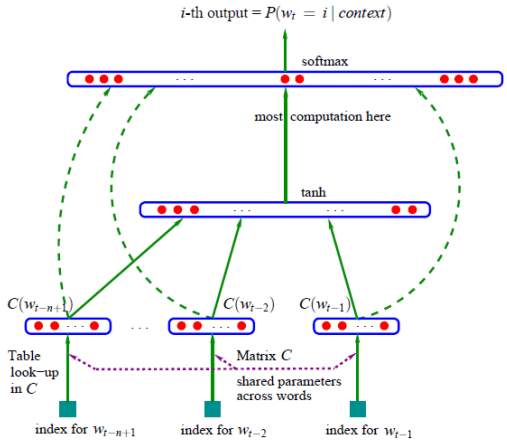
What's Strange on Neural MT

- We naturally think of translation in terms of manipulating with symbols
- Neural networks represent everything as real-space vectors
- Ignore pretty much everything we know about language

Symbol Embeddings

Discrete symbol vs. continuous representation

Simple task: predict next word given three previous:



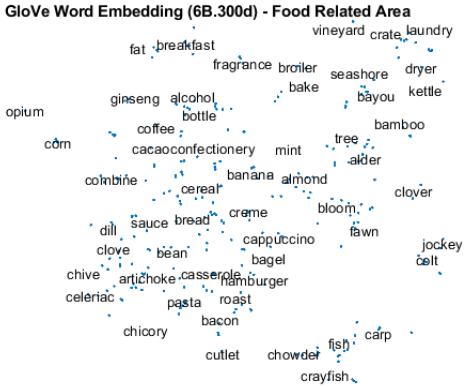
Source: Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.
http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

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

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

Properties of embeddings



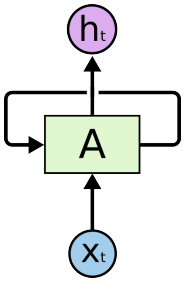
Source: https://blogs.mathworks.com/loren/2017/09/21/math-with-words-word-embeddings-with-matlab-and-text-analytics-toolbox/

Recurrent Networks

Why RNNs

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

General Formulation

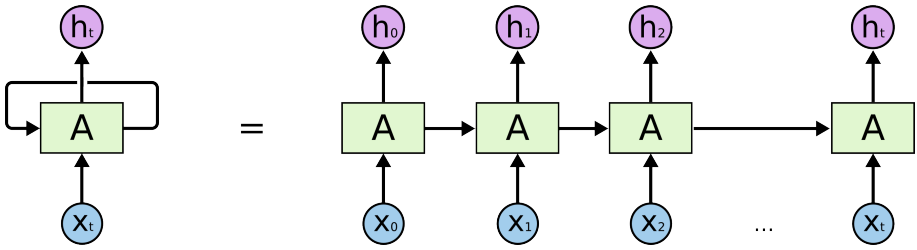


- inputs: $x_1, \ldots, x_T$
- initial state h_0:
  - **0**
  - result of previous computation
  - trainable parameter
- recurrent computation: $h_t = A(h_{t-1}, x_t)$

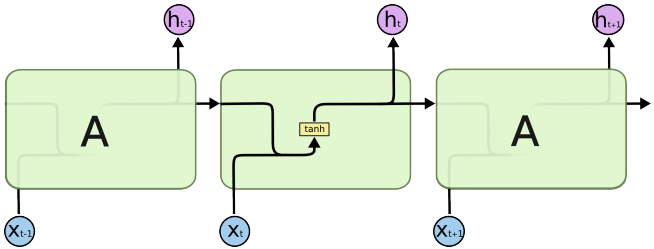
RNN as Imperative Code

```
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



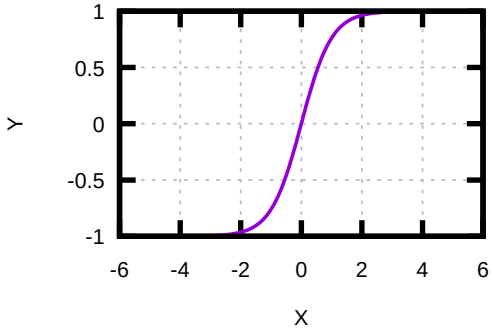
Vanilla RNN



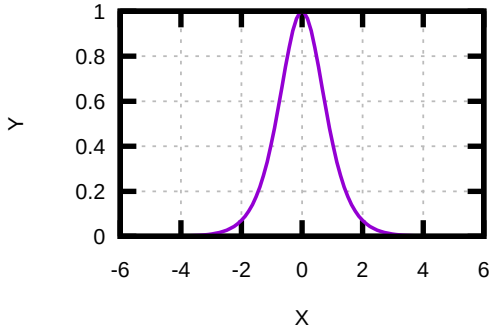
$$h_t = \tanh\left(W[h_{t-1}; x_t] + b\right)$$

- cannot propagate long-distance relations
- vanishing gradient problem

Vanishing Gradient Problem (1)



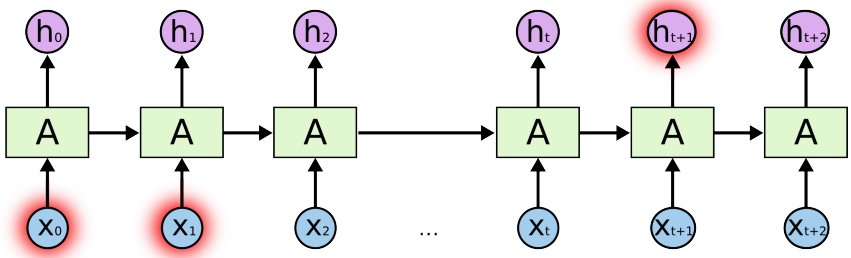
$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



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

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

Vanishing Gradient Problem (2)



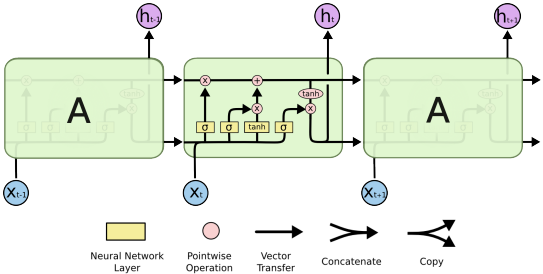
$$\frac{\partial E_{t+1}}{\partial b} = \frac{\partial E_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial b} \quad \text{(chain rule)}$$

Vanishing Gradient Problem (3)

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

LSTMs

LSTM = Long short-term memory

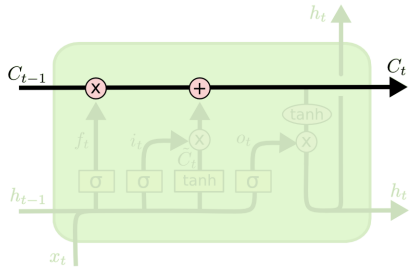


Control the gradient flow by explicitly gating:

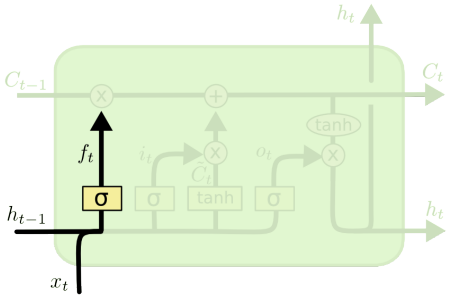
- what to use from input,
- what to use from hidden state,
- what to put on output

Hidden State

- two types of hidden states
- h_t — "public" hidden state, used as 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



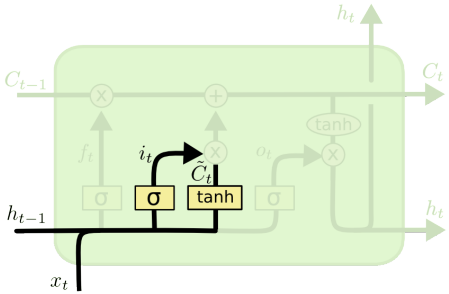
Forget Gate



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

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

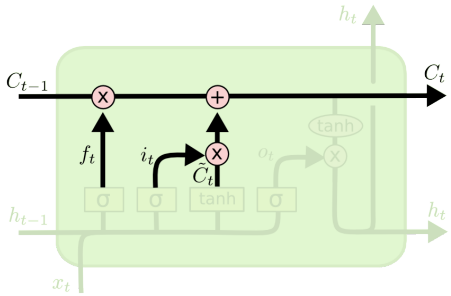
Input Gate



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

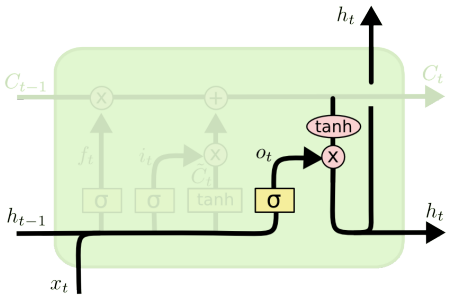
- $\tilde{C}$ — candidate what we may want to add to the memory
- i_t — decide how much of the information we want to store

Cell State Update



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

Output Gate



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

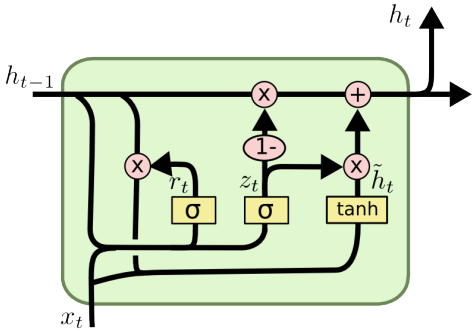
$$h_t = o_t \odot \tanh C_t$$

Here we are!

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

All gates are computed in a single matrix multiplication.

Gated Recurrent Units



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

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

Reading Assignment

Reading for the Next Week

Tao Lei et al., "Simple Recurrent Units for Highly Parallelizable Recurrence." EMNLP 2018. https://aclanthology.org/D18-1477/

Question:

Identify the information highway in simple recurrent units.

Given that all input embeddings are known, what part of the SRU computation can be fully parallelized and what part of the computation is sequential?