Introduction to Neural Machine Translation & Recurrent Neural Nets

Jindřich Helcl, Jindřich Libovický

February 24, 2022
What is Machine Translation?
What is Machine Translation?

Time for discussion...
• 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.)
⚠ 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

The single neuron activation function is given by:

\[
f\left(\sum_{i=0}^{n} x_i w_i\right)
\]

where the inputs \(x_i\) are weighted by \(w_i\), and the output is a function of the weighted sum.
**Neural Network**

\[ x \]
\[ \downarrow \]
\[ h_1 = f(W_1 x + b_1) \]
\[ \downarrow \]
\[ \vdots \]
\[ \downarrow \]
\[ h_n = f(W_n h_{n-1} + b_n) \]
\[ \downarrow \]
\[ o = g(W_o h_n + b_o) \]
\[ \downarrow \]
\[ E = e(o, t) \]

\[ \frac{\partial E}{\partial W_o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial W_o} \]
Logistic regression:

\[ y = \sigma(Wx + b) \]  

(1)

Computation graph:

forward graph

backward graph
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
\[ \text{mapAccumR } a \ s \ xs \]

Bidirectional RNN:

Zip left and right accumulating map
\[ \text{zip } \left( \text{mapAccumR } a \ s \ xs \right) \left( \text{mapAccumL } a' \ s' \ xs \right) \]

Convolution:

Zip neighbors and apply function
\[ \text{zipWith } a \ xs \ (\text{tail } xs) \]

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
• 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
• 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:

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 ⇒ 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.
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: \(h_t = A(h_{t-1}, x_t)\)
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 Imperative Code

Introduction to Neural Machine Translation & Recurrent Neural Nets
RNN as a Fancy Image

Introduction to Neural Machine Translation & Recurrent Neural Nets
Vanilla RNN

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

- cannot propagate long-distance relations
- vanishing gradient problem
Vanishing Gradient Problem (1)

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

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

Weights initialized $\sim \mathcal{N}(0, 1)$ to have gradients further from zero.
Vanishing Gradient Problem (2)

\[ \frac{\partial E_{t+1}}{\partial b} = \frac{\partial E_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial b} \]  

(chain rule)
Vanishing Gradient Problem (3)

\[
\frac{\partial h_t}{\partial b} = \frac{\partial \tanh (W_h h_{t-1} + W_x x_t + b)}{\partial b} = z_t \text{ (activation)}
\]

(tanh' is derivative of tanh)

\[
= \tanh'(z_t) \cdot \left( \frac{\partial W_h h_{t-1}}{\partial b} + \frac{\partial W_x x_t}{\partial b} + \frac{\partial b}{\partial b} \right)
\]

\[
= W_h \underbrace{\tanh'(z_t)}_{\sim \mathcal{N}(0,1)} \frac{\partial h_{t-1}}{\partial b} + \tanh'(z_t)
\]

Introduction to Neural Machine Translation & Recurrent Neural Nets
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
\[ i_t = \sigma (W_i \cdot [h_{t-1}; x_t] + b_i) \]
\[ \tilde{C}_t = \tanh (W_c \cdot [h_{t-1}; x_t] + b_C) \]

- \( \tilde{C} \) — 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

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

\[ h_t = o_t \odot \tanh C_t \]
Here we are!

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

All gates are computed in a single matrix multiplication.
Gated Recurrent Units

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

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

GRU

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

Recurrent Networks

+ • correspond to intuition of sequential processing
  • theoretically strong

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

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?