FMAN-45: Machine Learning

Lecture 8:

Recurrent Neural Networks

David Nilsson, Cristian Sminchisescu

RNNs

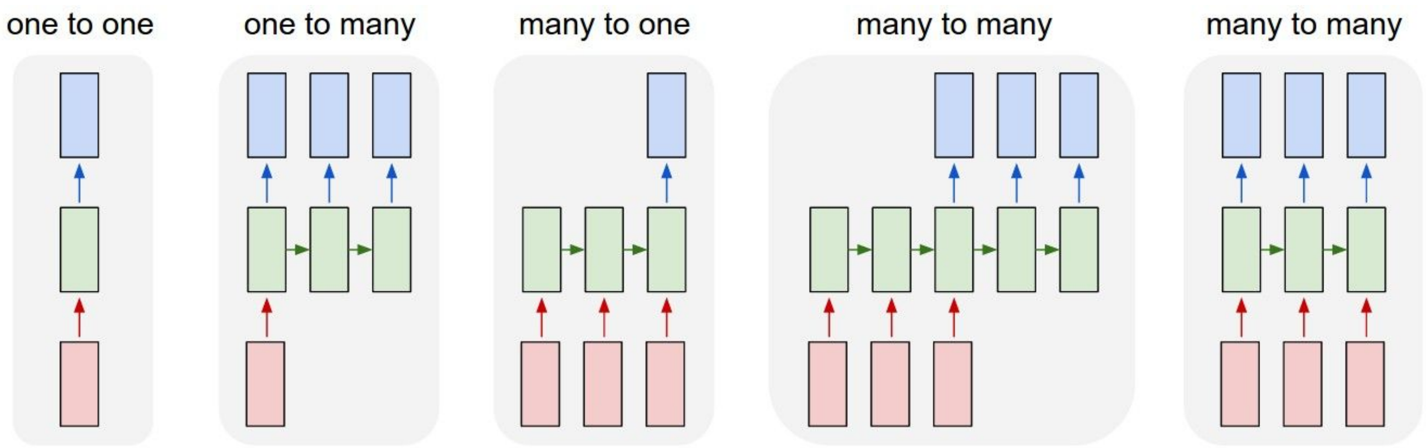


Image -> class

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNNs

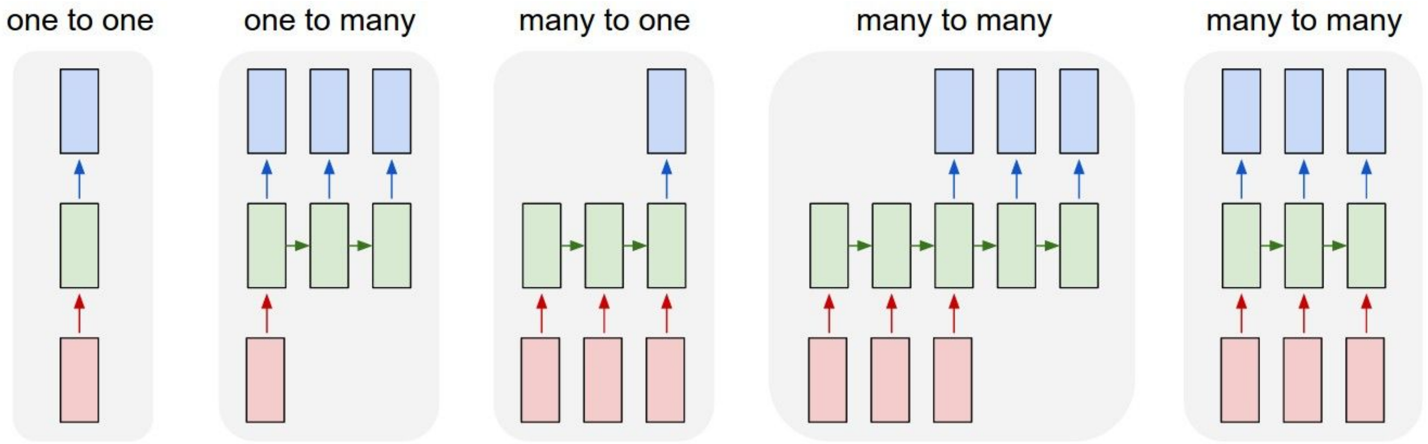
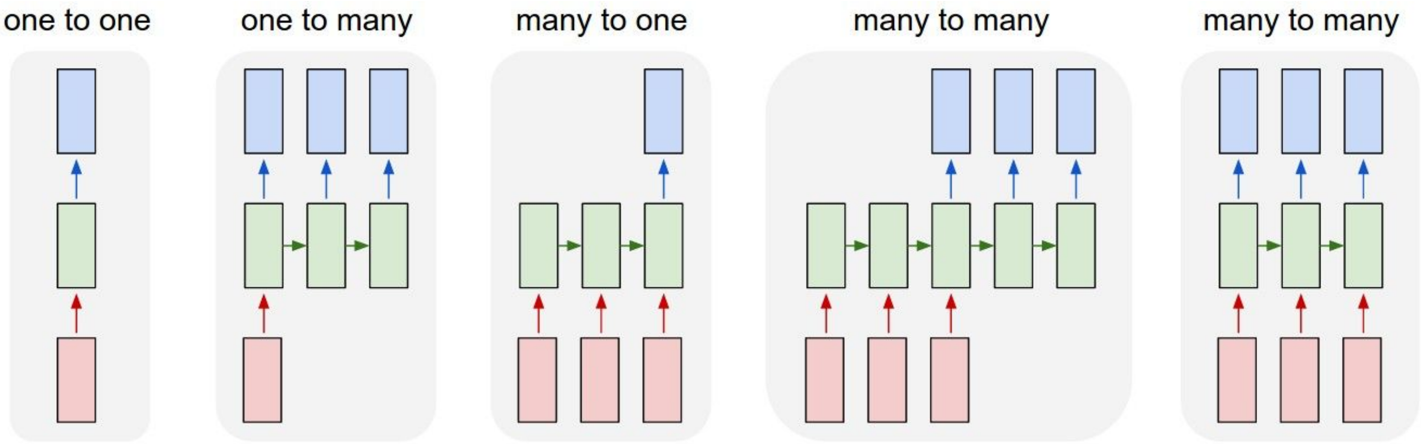


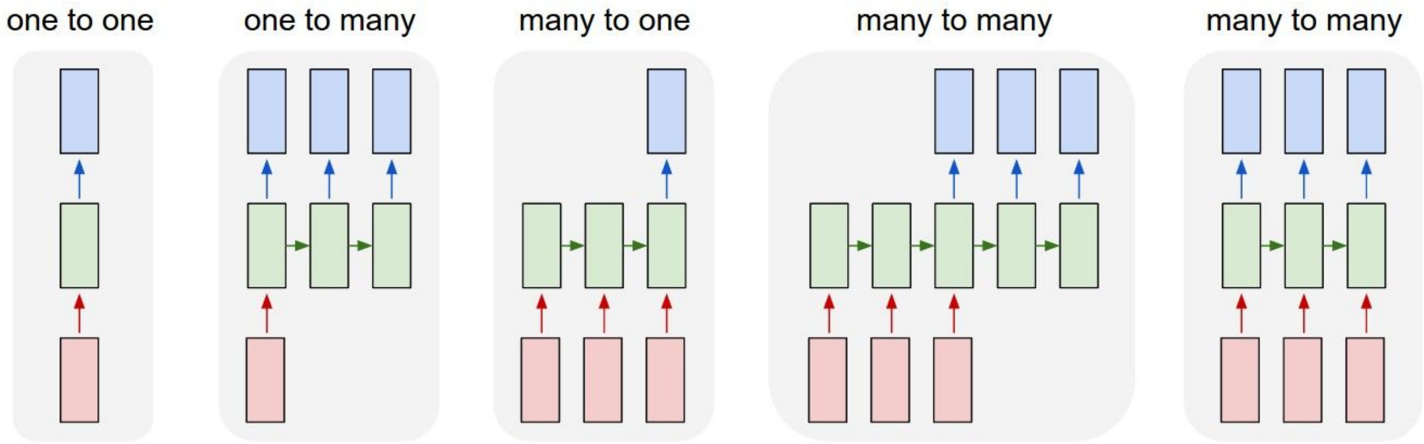
Image -> caption
Output one word at a time

RNNs



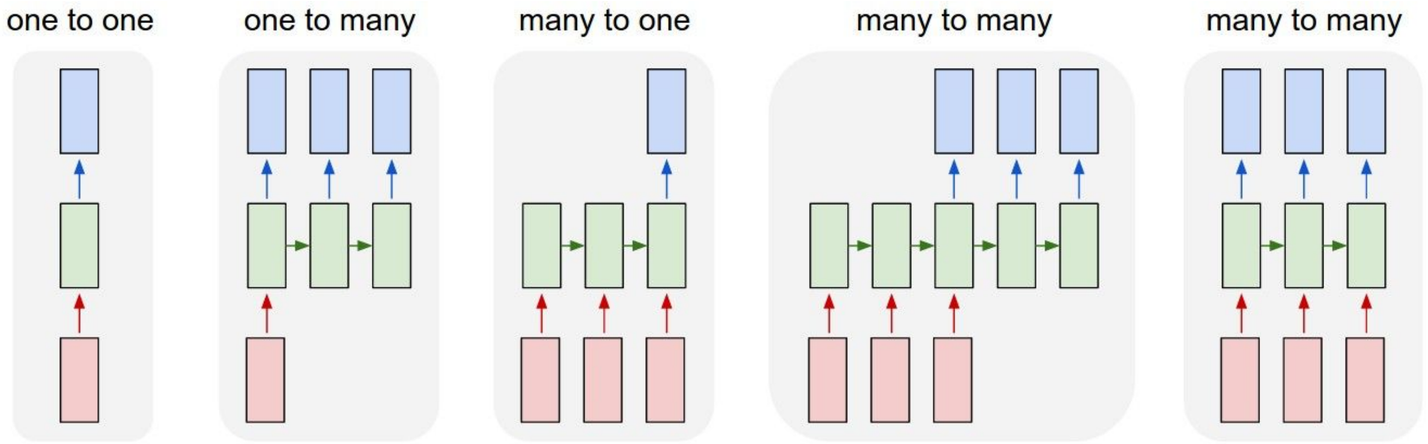
Video -> action (e.g. run, walk, jump, …)

RNNs



Machine translation
Sequence of words -> sequence of words

RNNs



Video segmentation

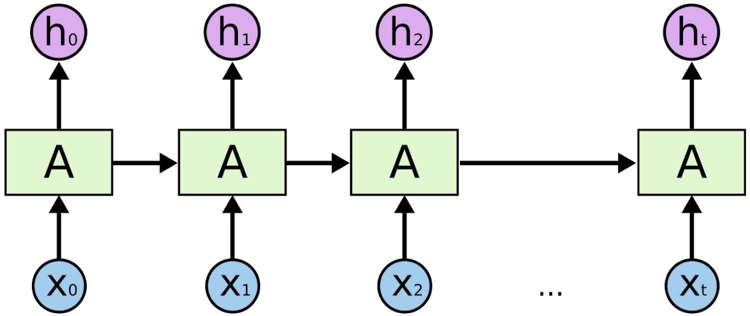
RNN

Process sequential data.

The RNN stores a state vector with relevant features.

Every time an input is processed, update the state vector.

The weights of the recurrent units are shared through time. There is only one set of parameters to learn.

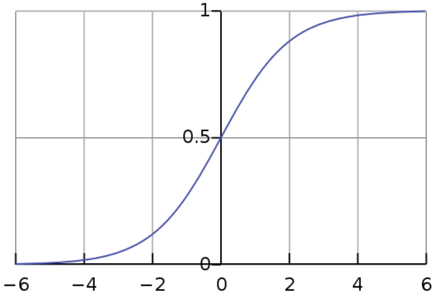


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNNs - Gating

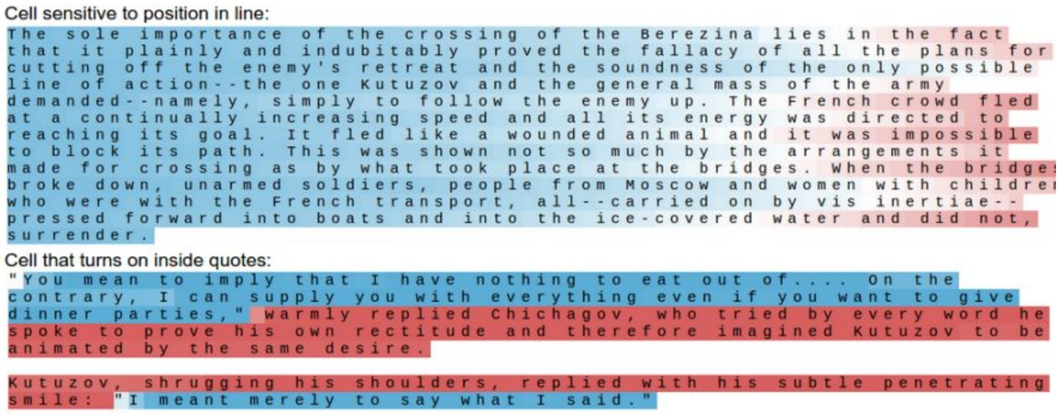
We want the hidden states to turn on and off during processing.

Multiply with 0/1, computed adaptively using the data, is the obvious modelling of how to do it. The problem is that all neural networks layers must be differentiable. We use the sigmoid function followed by element-wise multiplication to model gatings.



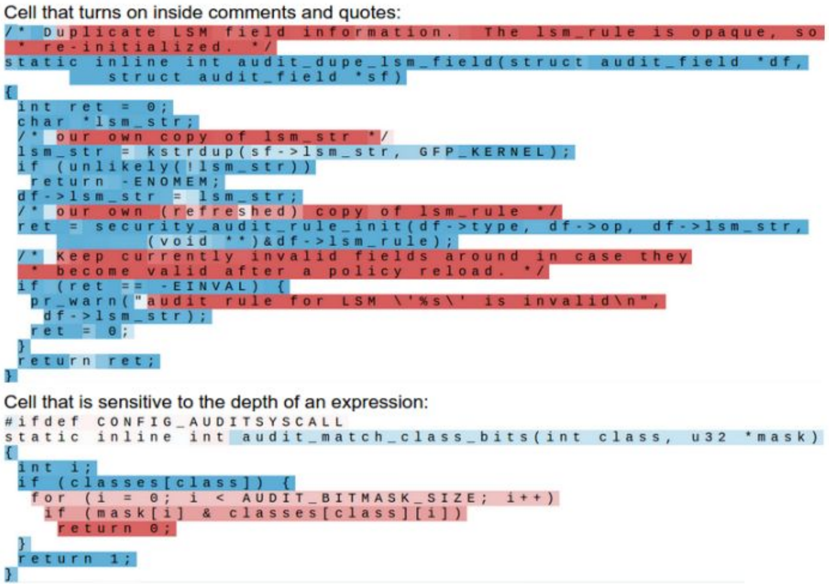
Sigmoid function

RNNs - Gating



https://arxiv.org/pdf/1506.02078.pdf

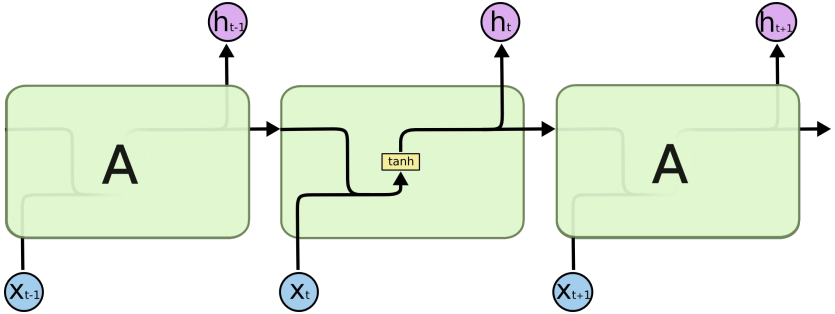
RNNs - Gating



https://arxiv.org/pdf/1506.02078.pdf

RNNs

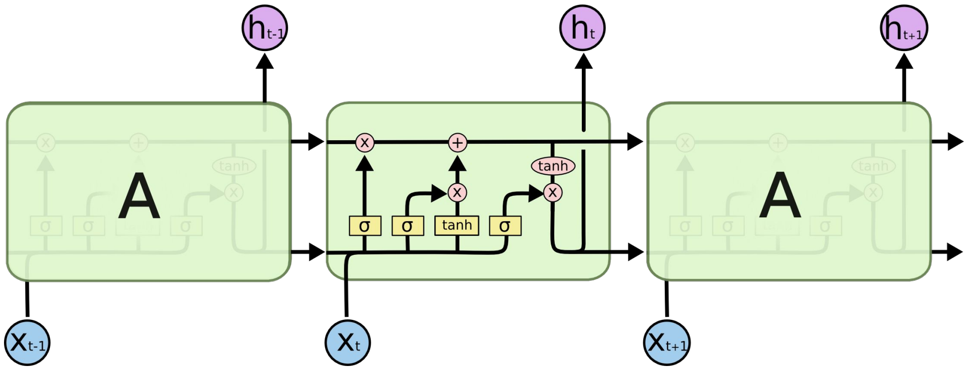
For every timestep the hidden states pass through a tanh non-linearity. This makes the optimization problem very hard and it will not learn long term dependencies.



$$h_t = \tanh(Wh_{t-1} + Ux_t + b)$$
$$y_t = \mathrm{softmax}(Vh_t + c)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM - Long Short-Term Memory

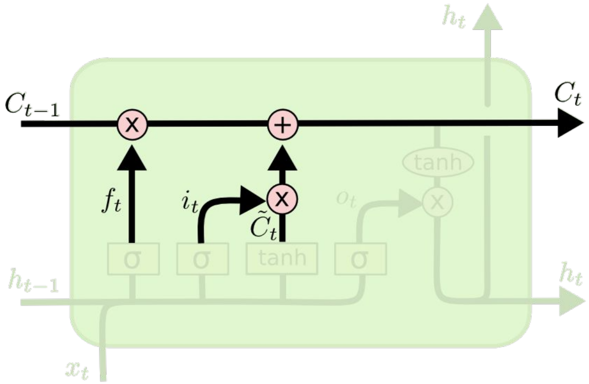


LSTM

Memory state.

The reason why the gradients are stable and why an LSTM can learn long-term dependencies.

We forget what should be forgotten and add what should be added.

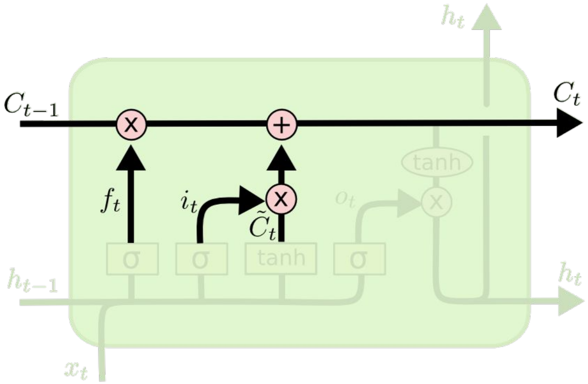


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM

If the state is unchanged, the mapping is an identity mapping and the same errors are back-propagated.

The formula holds for one hidden state. It holds element-wise for many hidden states.



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

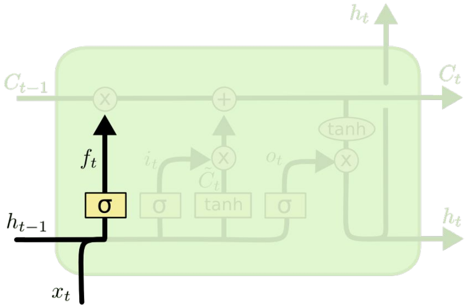
$$\frac{\partial L}{\partial C_{t-1}} = \frac{\partial L}{\partial C_t} f_t$$

$$\left(f_t = 1 \implies \frac{\partial L}{\partial C_{t-1}} = \frac{\partial L}{\partial C_t} \right)$$

LSTM

Forget gate.

Example: We have a feature that increases the longer it has been since a line break. When a line break is obtained in x_t, then the feature is reset to 0.



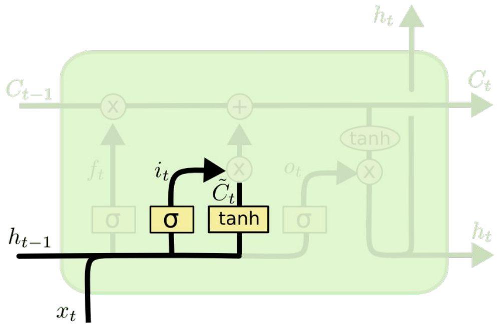
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

LSTM

Input gate and candidate memory state.

Given new information in x_t, the memory state C_t should be updated.

Example: If x_t is a dot, then the memory state should contain information that we are about to start a new sentence.



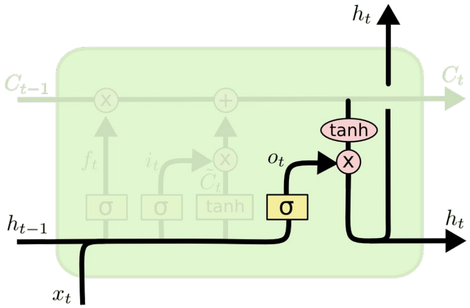
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTM

Output gate.

The memory state C_t and the hidden state h_t have different time scales. The hidden states are relevant for short term processing and the memory gate stores relevant long-term information.

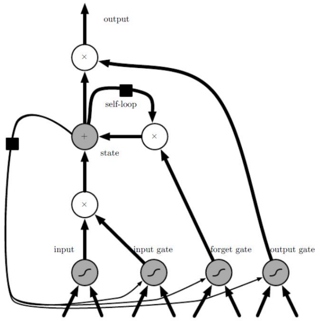


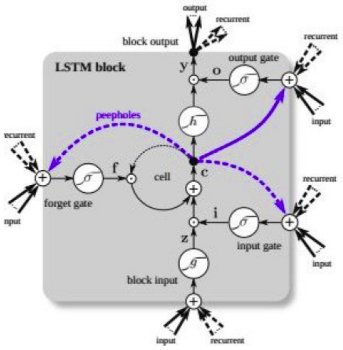
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

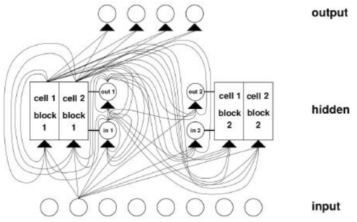
$$h_t = o_t * \tanh\left(C_t\right)$$

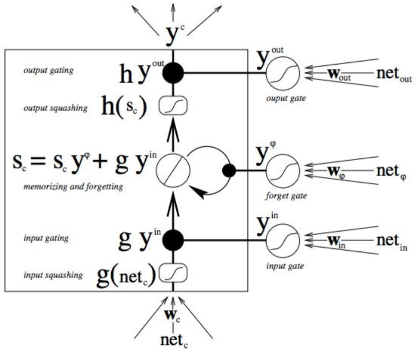
LSTM

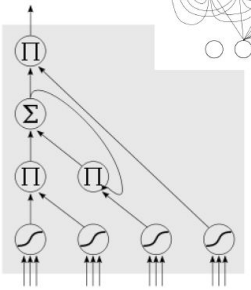
Be aware of crazy LSTM diagrams. See the [webpage](#) where the figures in the previous slides are taken from for a very good description.

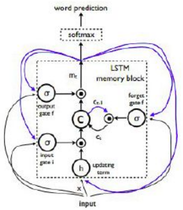












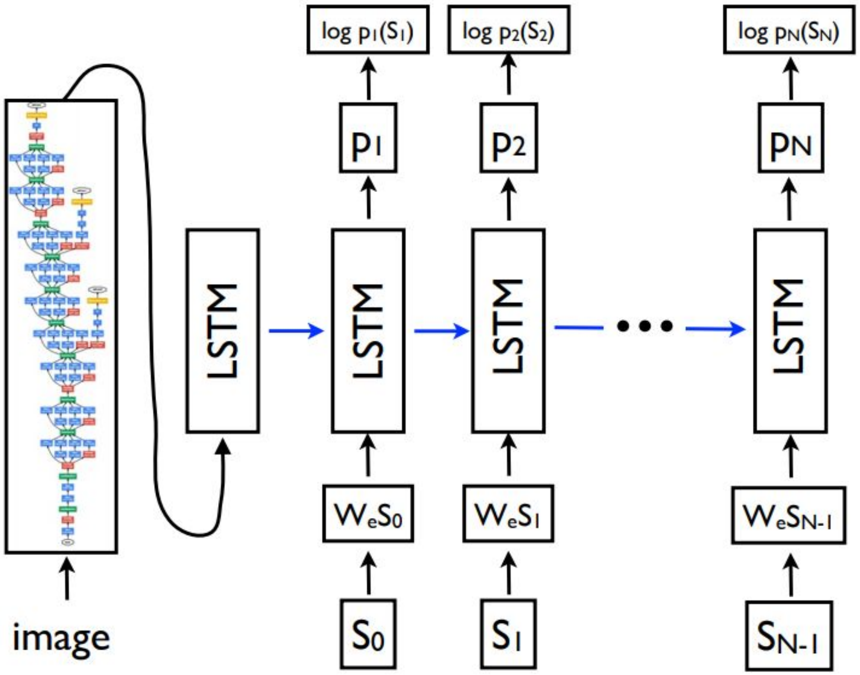
RNNs

Vinyals et al. CVPR 2015

RNNs

Process the image through a deep CNN to get a 512-dimensional feature vector, used to initialize the state vector.

Send in one word embedding at a time, process it through the LSTM, compute/sample the next word.

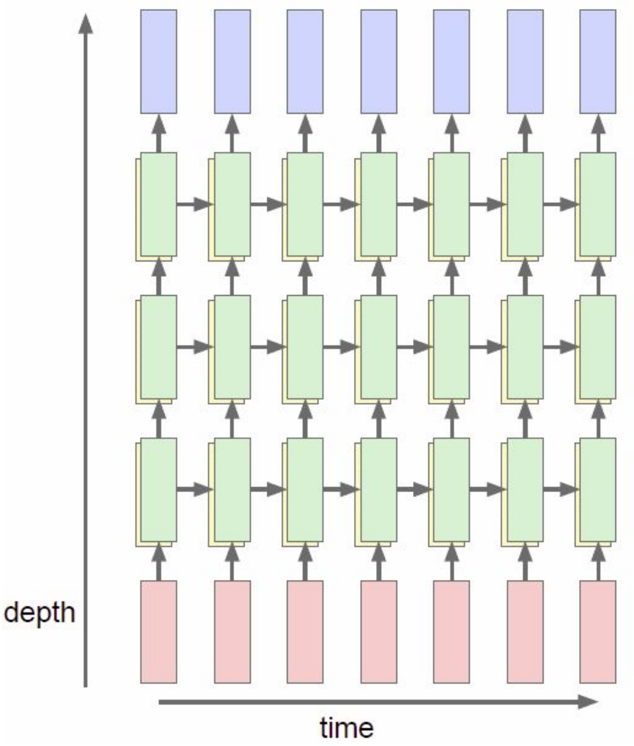


Vinyals et al. CVPR 2015

Stack LSTMs

We can make RNNs that are arbitrarily deep.

Notice similar dependencies on inputs and hidden states from the same layer.



Stack LSTMs

Google’s translation system.

Process the word embedding in a sequential manner to get an encoding of the sentence to translate.

Send the encoding to a decoder that outputs the translation.

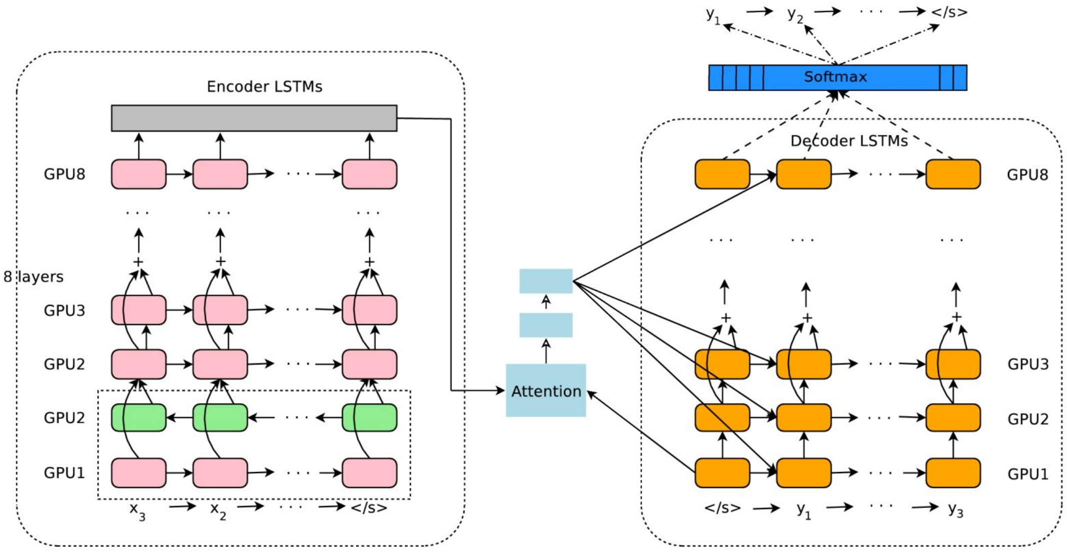
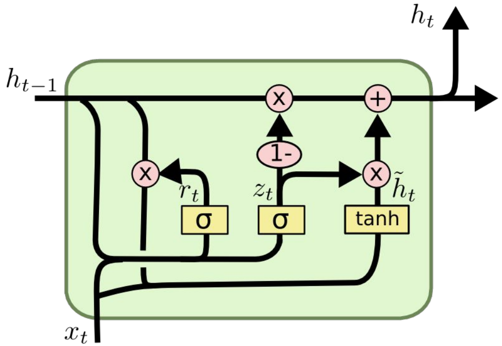


Figure 1: The model architecture of GNMT, Google’s Neural Machine Translation system. On the left

https://arxiv.org/pdf/1609.08144.pdf

“it takes around 6 days to train a basic model using 96 NVIDIA K80 GPUs”

GRU - Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Assignment 3

Derive and implement forward and backward passes for fully connected layers, relu-layers and softmax-losses. Be very careful with indices!

For problem 1-4, start with the provided formulas. Do not use a Jacobian approach where it is not applicable, or preferably not at all. The Jacobian is defined for a mapping from a vector to a vector and not for mappings from matrices to matrices. dA/dB=?

Implement gradient descent with momentum.

Train and design neural networks for image classification.

Readings

Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning, MIT Press 2016 (online at http://www.deeplearningbook.org/)

CNNs

- Section 9.1-9.3 in the deep learning book

LSTMs

- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Section 10.10 in the deep learning book