Recurrent Neural Networks

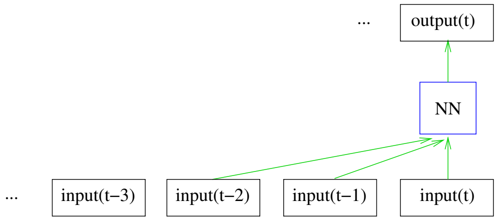
Modelling sequences

Typical problems:

- **turn an input sequence into an output sequence** (possibly in a different domain):
  - \- translation between different languages
  - \- speech/sound recognition
  - \- ...
- **predict the next term in a sequence**
  The target output sequence is the input sequence with an advance of 1 step. Blurs the distinction between supervised and unsupervised learning.
- **predict a result from a temporal sequence of states**
  Typical of Reinforcement learning, and robotics.

Memoryless approach

Compute the output as a result of a fixed number of elements in the input sequence



Used e.g. in

- - Bengio's (first) predictive natural language model
- - Qlearning for Atari Games

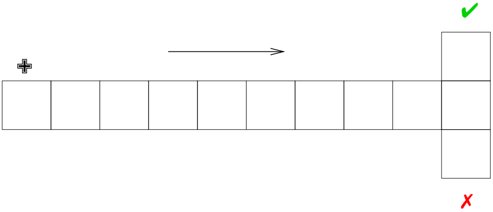
Difficult to deal with very long-term dependencies.

Simple problems requiring memory

arithmetical sum

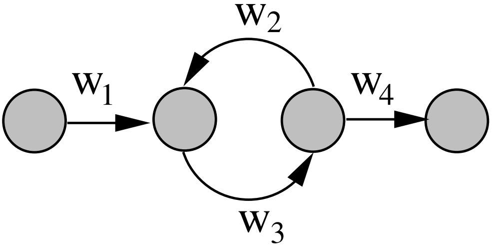
$$
\begin{array}{cccccccc}
 & & & \longleftarrow & & & & \\
\ldots & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\
\ldots & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
\hline
\ldots & 0 & 1 & 0 & 1 & 0 & 0 & 1
\end{array}
$$

the T-maze



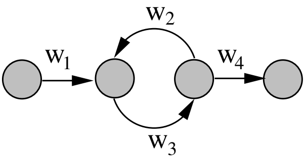
Recurrent Neural Networks

In presence of backward connections, hidden states depend on the past history of the net



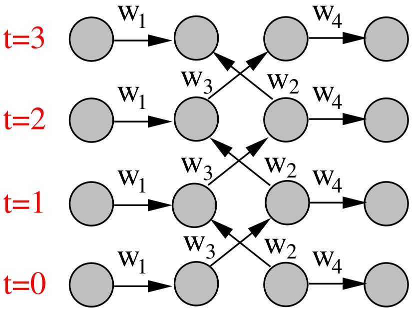
The hidden state may be updated in complex ways, according to the usual non-linear dynamics of neural nets.

Temporal unfolding



Weights are updated at precise times steps

The recurrent net is just a layered net that keeps reusing the same weights



Sharing weights through time

It is easy to modify the backprop algorithm to incorporate equality constraints between weights.

We compute the gradients as usual, and then average gradients so that they induce a same update.

If the initial weights started satisfied the constraints, they will continue to do.

To constrain $w_1 = w_2$
we need $\Delta w_1 = \Delta w_2$

compute $\frac{\partial E}{\partial w_1}$ and $\frac{\partial E}{\partial w_2}$

and use $\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$

to update both w_1 and w_2

Backpropagation through time - BPTT

- think of the recurrent net as a layered, feed-forward net with shared weights and train the feed-forward net with weight constraints

- reasoning in the time domain:
  - the forward pass builds up a stack of the activities of all the units at each time step
  - the backward pass peels activities off the stack to compute the error derivatives at each time step
  - finally we add together the derivatives at all the different times for each weight.

Hidden state initialization

We need to specify the initial activity state of all the hidden and output units.

The best approach is to treat them as parameters, learning them in the same way as we learn the weights:

- start off with an initial random guess for the initial states
- at the end of each training sequence, backpropagate through time all the way to the initial states to get the gradient of the error function with respect to each initial state
- adjust the initial states by following the negative gradient

Why training Recurrent Neural Nets is difficult

(based on Hinton's course)

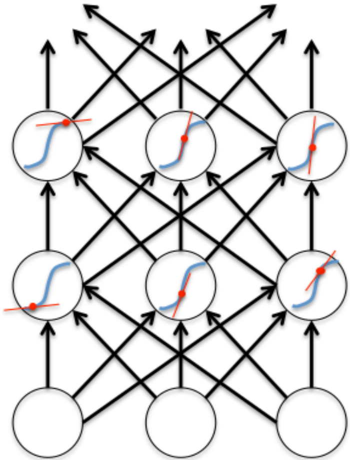
The backward pass is linear

There is a big difference between the forward and backward passes

In the forward pass we use squashing functions (like the logistic) to prevent the activity vectors from exploding

The backward pass is linear. If you double the error derivatives at the final layer, all the error derivatives will double.

The forward pass determines the slope of the linear function used for backpropagating through each neuron



Vanishing or exploding gradients

What happens to the magnitude of the gradients as we backpropagate through many layers?

- if the weights are small, the gradients shrink exponentially
- if the weights are big the gradients grow exponentially

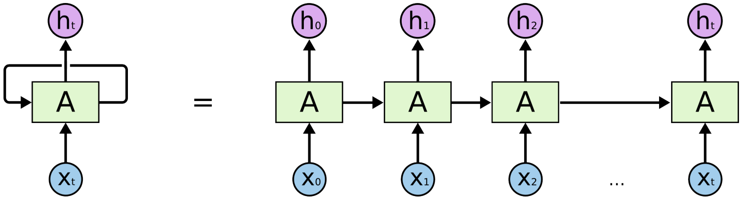
Typical feed-forward neural nets can cope with these exponential effects because they only have a limited number of layers.

In an RNN trained on long sequences (e.g. 100 time steps) the gradients can easily explode or vanish.

Long-Short Term Memory (LSTM)

Largely based on Colah's blog

Unrolling recurrent nets

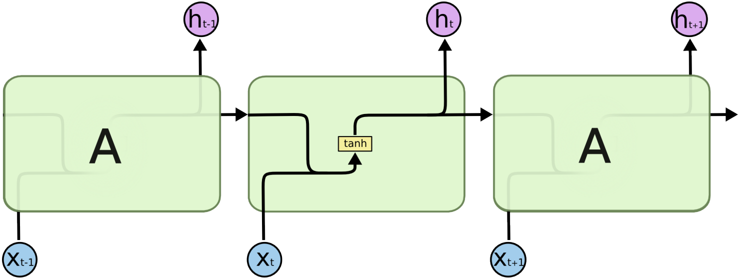


In the following, we shall mostly depict RNN in unrolled form.

A forward link between two units muts be understood as a looping connection.

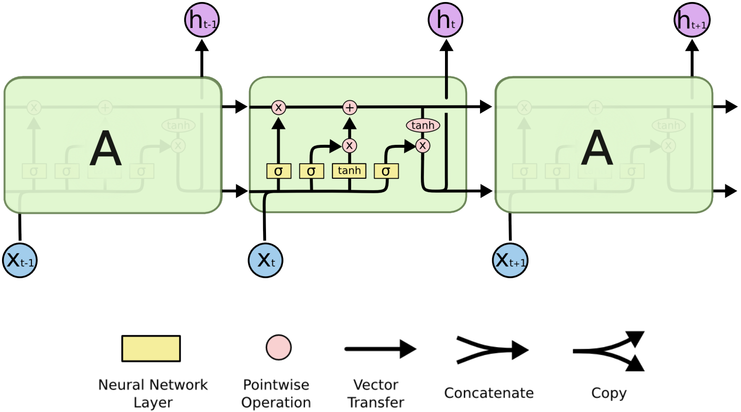
A simple, traditional RNN

The content of the memory cell C_t, and the input x_t are combined through a simple neural net to produce the output h_t that coincides with the new content of the cell C_{t+1}.



Why $C_{t+1} = h_t$? Better trying to preserve the memory cell, letting the neural net learn how and when to update it.

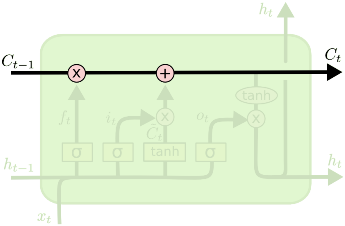
The overall structure of a LSTM



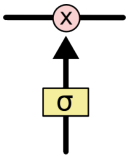
C-line and gates

The LSTM has the ability to remove or add information to the cell state, in a way regulated by suitable gates.

Gates are a way to optionally let information through: the product with a sigmoid neural net layer simulates a boolean mask.

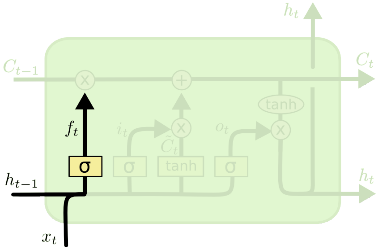


the C-line



a gate

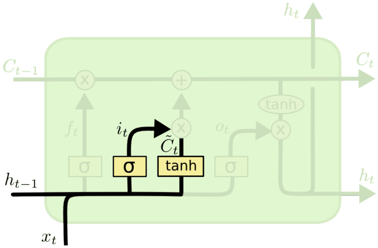
The forget gate



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

The forget gate decides what part of the memory cell to preserve

The update gate



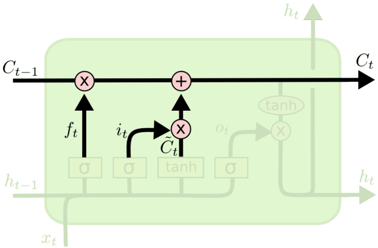
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

The input gate decides what part of the input to preserve.

The tanh layer creates a vector of new candidate values $\tilde{C}_t$ to be added to the state.

Cell updating



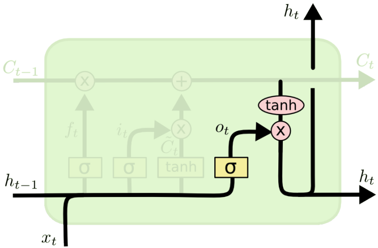
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

We multiply the old state by the boolean mask f_t.

Then we add $i_t * \tilde{C}_t$.

output gate

The output h_t is a filtered version of the content of the cell.



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

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

The output gate decides what parts of the cell state to output.

The tanh function is used to renormalize values in the interval $[-1, 1]$.

Thousands of variants

Essential bibliography

- S.Hochreiter, J. Schmidhuber. "Long short-term memory". Neural Computation. 9 (8): pp.1735-1780. 1997
- F.A.Gers, Jürgen Schmidhuber, F.Cummins. "Learning to Forget: Continual Prediction with LSTM". Neural Computation. 12 (10), pp.2451-2471. 2000.
- F.A.Gers, E.Schmidhuber. "LSTM recurrent networks learn simple context-free and context-sensitive languages". IEEE Transactions on Neural Networks. 12 (6): pp. 1333-1340. 2001.
- Y.Chung, C.Gulcehre, K.Cho, Y.Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". arXiv:1412.3555. 2014