Recurrent Networks

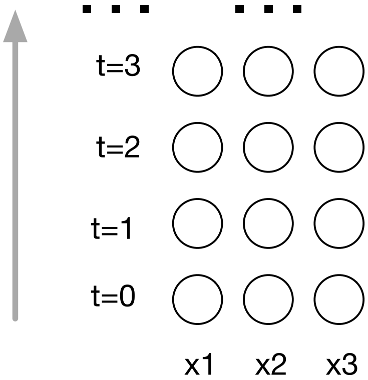
Steve Renals

Machine Learning Practical — MLP Lecture 9 (extra)
4 December 2015

Introduction

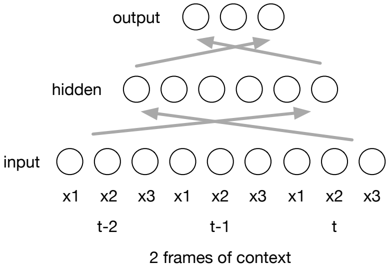
- Modelling sequential data
- Recurrent hidden unit connections
- Training RNNs: Back-propagation through time
- LSTMs
- Examples (speech and language)

Sequential Data



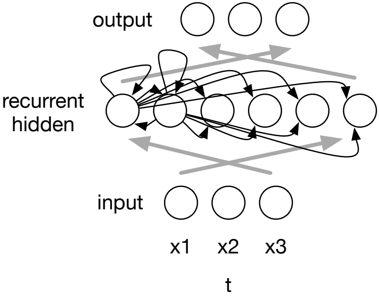
- Modelling sequential data with time dependences between feature vectors

Sequential Data



- Modelling sequential data with time dependences between feature vectors
- Can model fixed context with a feed-forward network with previous time input vectors added to the network input (in signal processing this is called FIR – **finite** input response)

Sequential Data



- Modelling sequential data with time dependences between feature vectors
- Can model fixed context with a feed-forward network with previous time input vectors added to the network input (in signal processing this is called FIR – **finite** impulse response)
- Model sequential inputs using *recurrent* connections to learn a *time-dependent state* (in signal processing this is called IIR – **infinite** impulse response)

Recurrent networks

Can think of recurrent networks in terms of the dynamics of the recurrent hidden state

- Settle to a fixed point – stable representation for a sequence (e.g. machine translation)
- Regular oscillation ("limit cycle") – learn some kind of repetition
- Chaotic dynamics (non-repetitive) – theoretically interesting ("computation at the edge of chaos")

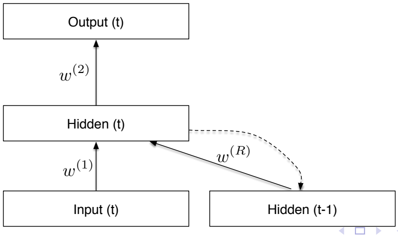
Useful behaviours of recurrent networks:

- Recurrent state as memory – remember things for (potentially) an infinite time
- Recurrent state as information compression – compress a sequence into a state representation

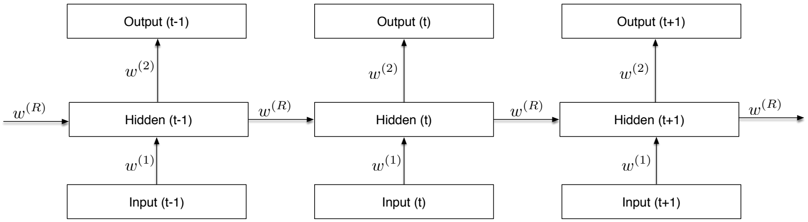
Simplest recurrent network

$$y_k(t) = \text{softmax}\left(\sum_{r=0}^{H} w_{kr}^{(2)} h_r(t)\right)$$

$$h_j(t) = \text{sigmoid}\left(\sum_{s=0}^{d} w_{js}^{(1)} x_s(t) + \underbrace{\sum_{r=0}^{H} w_{jr}^{(R)} h_r(t-1)}_{\text{Recurrent part}}\right)$$



Recurrent network unfolded in time



- An RNN for a sequence of T inputs can be viewed as a deep T-layer network with shared weights
- We can train an RNN by doing backprop through this unfolded network, making sure we share the weights
- Weight sharing
  - if two weights are constrained to be equal ($w_1 = w_2$) then they will stay equal if the weight changes are equal ($\partial E/\partial w_1 = \partial E/\partial w_2$)
  - achieve this by updating with ($\partial E/\partial w_1 + \partial E/\partial w_2$) (cf Conv Nets)

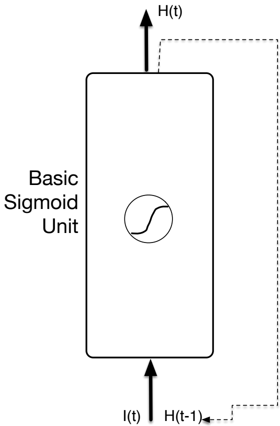
Back-propagation through time (BPTT)

- We can train a network by unfolding and *back-propagating through time*, summing the derivatives for each weight as we go through the sequence
- More efficiently, run as a recurrent network
  - cache the unit outputs at each timestep
  - cache the output errors at each timestep
  - then backprop from the final timestep to zero, computing the derivatives at each step
  - compute the weight updates by summing the derivatives across time
- Expensive – backprop for a 1,000 item sequence equivalent to a 1,000-layer feed-forward network
- Truncated BPTT – backprop through just a few time steps (e.g. 20)

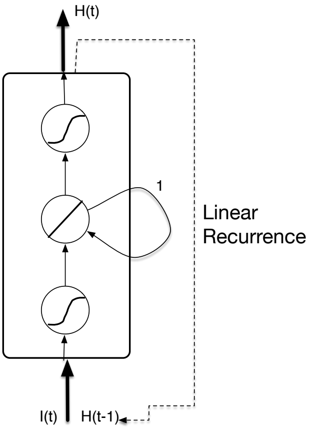
Vanishing and exploding gradients

- BPTT involves taking the product of many gradients (as in a very deep network) – this can lead to vanishing (component gradients less than 1) or exploding (greater than 1) gradients
- This can prevent effective training
- Modified optimisation algorithms
  - RMSProp (normalise the gradient for each weight by average of it magnitude, learning rate for each weight)
  - Hessian-free – an approximation to second-order approaches which use curvature information
- Modified hidden unit transfer functions
  - Long short term memory (LSTM)
    - Linear self-recurrence for each hidden unit (long-term memory)
    - Gates - dynamic weights which are a function of the inputs
  - ReLUs

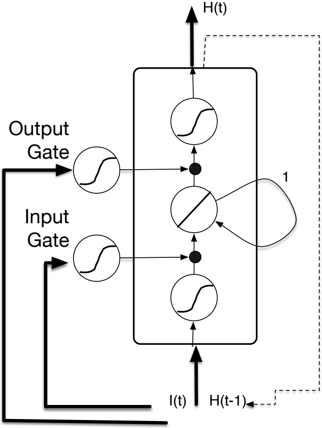
LSTM v1



LSTM v1

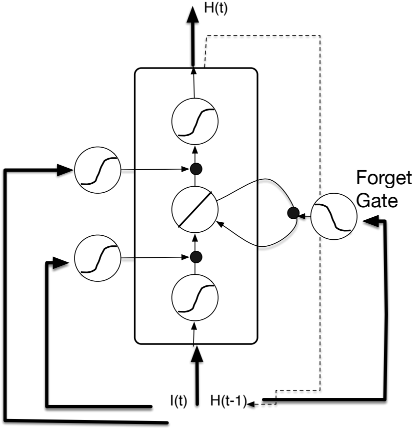


LSTM v1



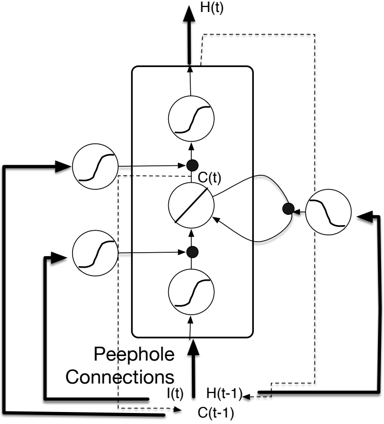
S Hochreiter and J Schmidhuber (1997). "Long Short-Term Memory", *Neural Computation*, 9:1735–1780.

LSTM v2



FA Gers et al (2000). “Learning to Forget: Continual Prediction with LSTM”, *Neural Computation*, 12:2451–2471.

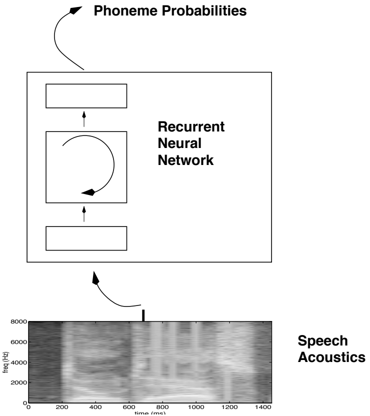
LSTM v3



LSTM equations

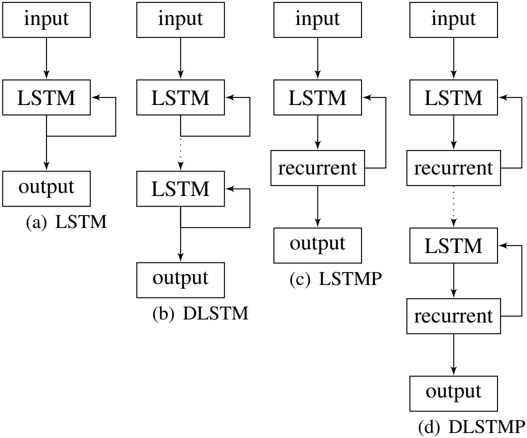
$$
\begin{align}
i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1} + W_{ic}c_{t-1} + b_i) \tag{1}\\
f_t &= \sigma(W_{fx}x_t + W_{mf}m_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2}\\
c_t &= f_t \odot c_{t-1} + i_t \odot g(W_{cx}x_t + W_{cm}m_{t-1} + b_c) \tag{3}\\
o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1} + W_{oc}c_t + b_o) \tag{4}\\
m_t &= o_t \odot h(c_t) \tag{5}\\
y_t &= W_{ym}m_t + b_y \tag{6}
\end{align}
$$

Example 1: speech recognition with recurrent networks



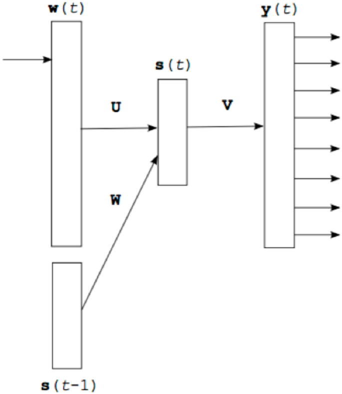
T Robinson et al (1996). "The use of recurrent networks in continuous speech recognition", in *Automatic Speech and Speaker Recognition Advanced Topics* (Lee et al (eds)), Kluwer, 233–258.

Example 2: speech recognition with stacked LSTMs



H Sak et al (2014). "Long Short-Term Memory based Recurrent Neural Network Architectures for Large Scale Acoustic Modelling", *Interspeech*.

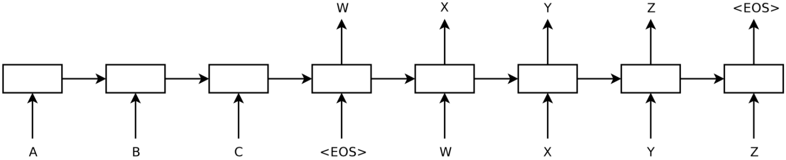
Example 3: recurrent network language models

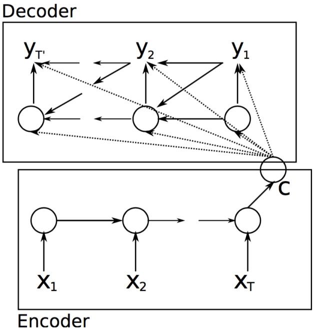


T Mikolov et al (2010). "Recurrent Neural Network Based Language Model", *Interspeech*

Example 4: recurrent encoder-decoder

Machine translation





- I Sutskever et al (2014). "Sequence to Sequence Learning with Neural Networks", *NIPS*.
- K Cho et al (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", *EMNLP*.

Summary

- RNNs can model sequences
- Unfolding an RNN gives a deep feed-forward network
- Back-propagation through time
- LSTM
- RNNs are useful for more than sequence learning! – see recent Google DeepMind work on using RNNs to locate items of interest in images
- More on recurrent networks next semester in NLU (and 1-2 lectures in ASR and MT)