Recurrent neural networks
Modelling sequential data

Recurrent Networks

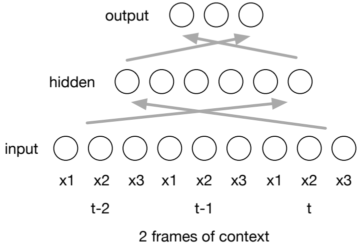
Steve Renals

Machine Learning Practical — MLP Lecture 9
16 November 2016

Introduction - Recurrent Neural Networks (RNNs)

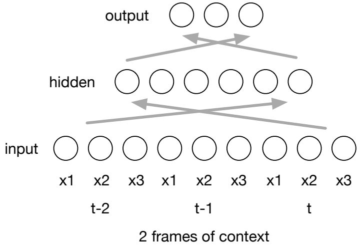
- Modelling sequential data
- Recurrent hidden unit connections
- Training RNNs: Back-propagation through time
- LSTMs
- Examples (speech and language)

Sequential Data



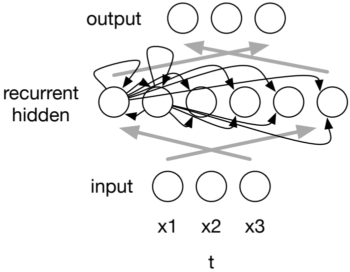
- Modelling sequential data with time dependences between feature vectors

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
    - **Finite** context determined by window width

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
  - **Finite** context determined by window width
- Model sequential inputs using *recurrent* connections to learn a *time-dependent state*
  - Potentially **infinite** context

Recurrent networks

If there was no external input... think of recurrent networks in terms of the dynamics of the recurrent hidden state

- Settle to a fixed point – stable representation
- 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 with external inputs:

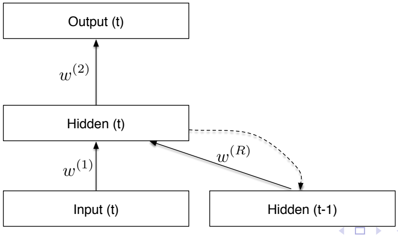
- Recurrent state as memory – remember things for (potentially) an infinite time
- Recurrent state as information compression – compress a sequence into a state representation

Vanilla RNNs

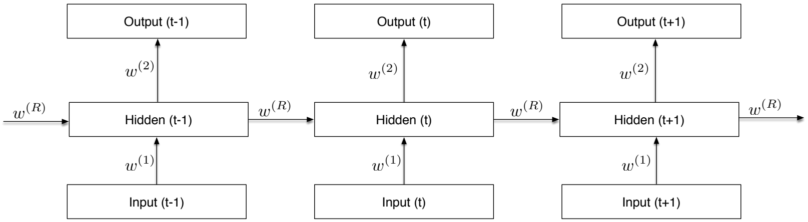
Simplest recurrent network

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

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

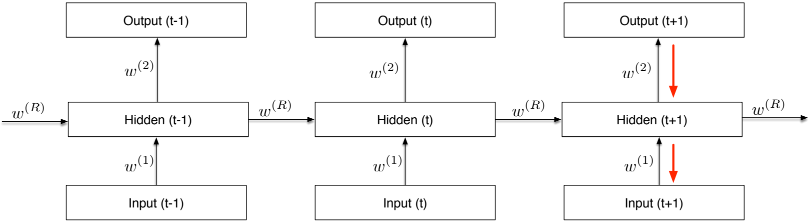


Recurrent network unfolded in time



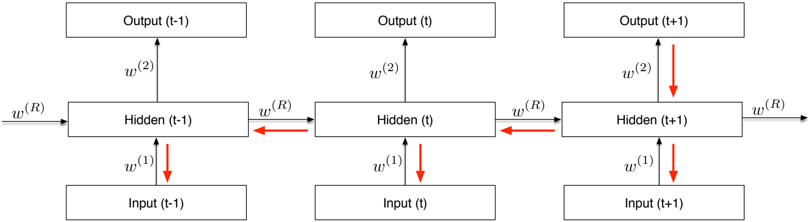
- An RNN for a sequence of T inputs can be viewed as a deep T-layer network with shared weights

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

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)

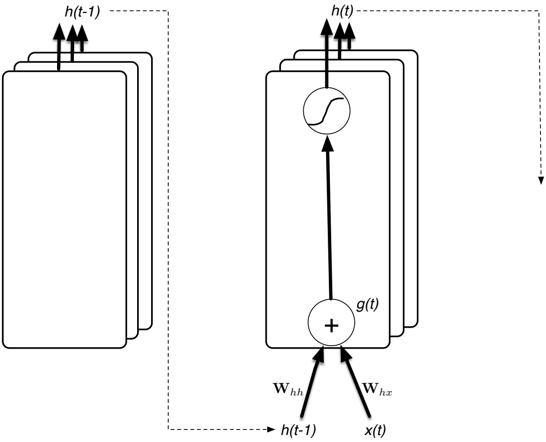
- We can train a network by unfolding and *back-propagating through time*, summing the derivatives for each weight as we go through the sequence
- More efficiently, run as a recurrent network
    - cache the unit outputs at each timestep
    - cache the output errors at each timestep
    - then backprop 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 (and similar algorithms) – normalise the gradient for each weight by average of it magnitude, with a 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 their inputs
    - Gated recurrent units

LSTM

Vanilla RNN

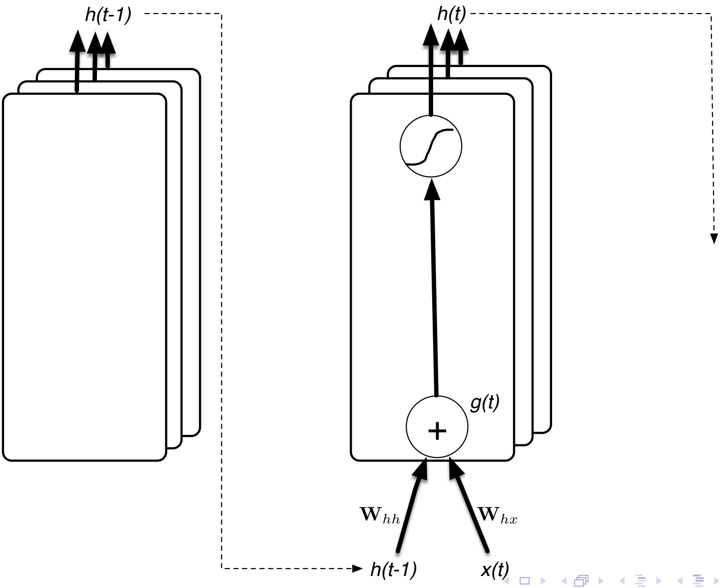


$$\mathbf{g}(t) = \mathbf{W}_{hx}\mathbf{x}(t) + \mathbf{W}_{hh}\mathbf{h}(t-1) + \mathbf{b}_h$$
$$\mathbf{h}(t) = \tanh\left(\mathbf{g}(t)\right)$$

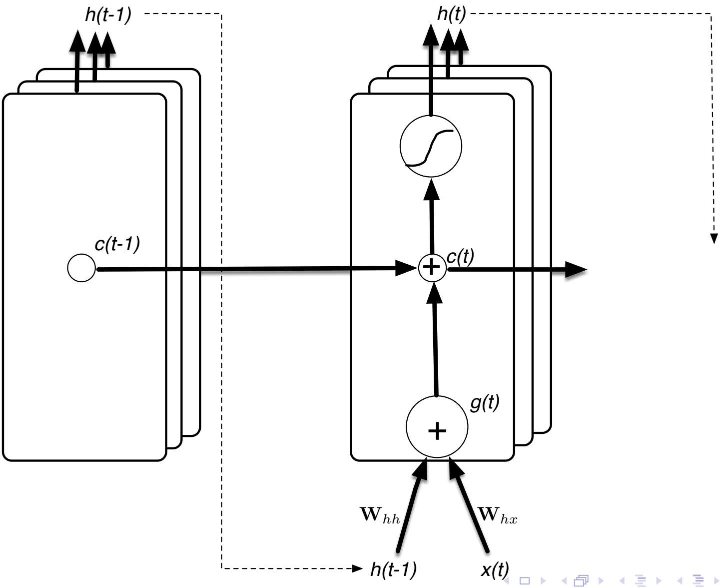
LSTM

- **Internal recurrent state** ("cell") $c(t)$ combines previous state $c(t-1)$ and LSTM input $g(t)$

LSTM



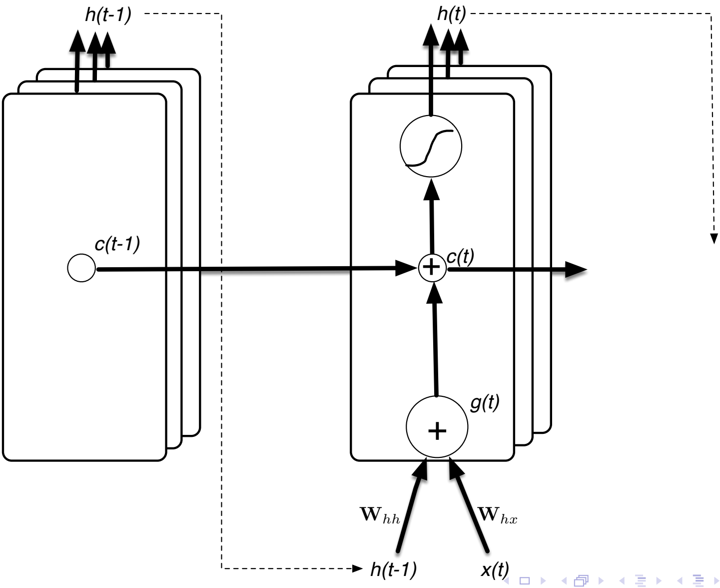
LSTM – Internal recurrent state



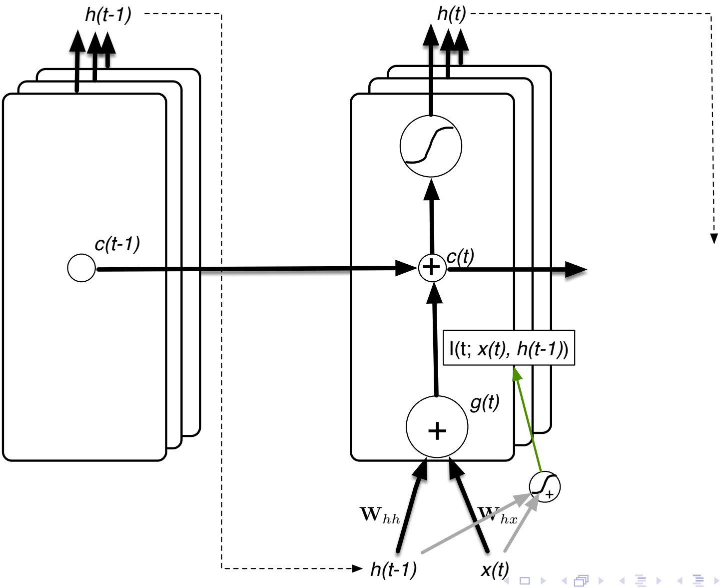
LSTM

- **Internal recurrent state** ("cell") $c(t)$ combines previous state $c(t-1)$ and LSTM input $g(t)$
- Gates - weights dependent on the current input and the previous state
- **Input gate**: controls how much input to the unit $g(t)$ is written to the internal state $c(t)$
- **Forget gate**: controls how much of the previous internal state $c(t-1)$ is written to the internal state $c(t)$
    - Input and forget gates together allow the network to control what information is stored and overwritten at each step

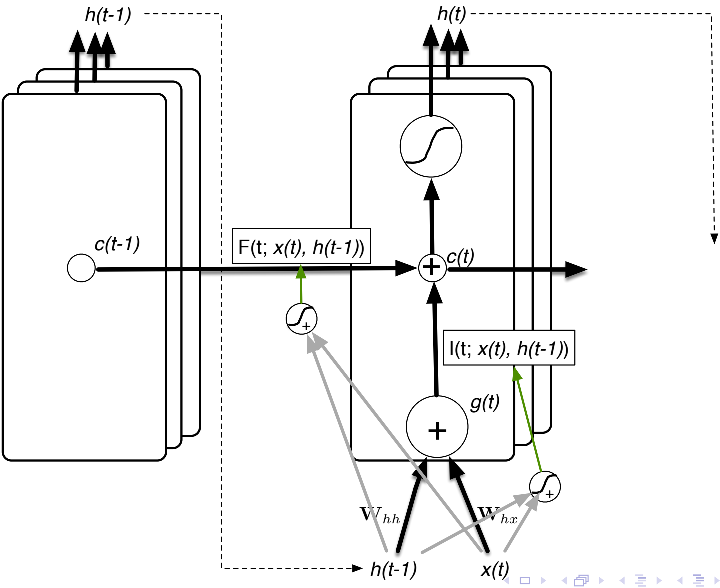
LSTM



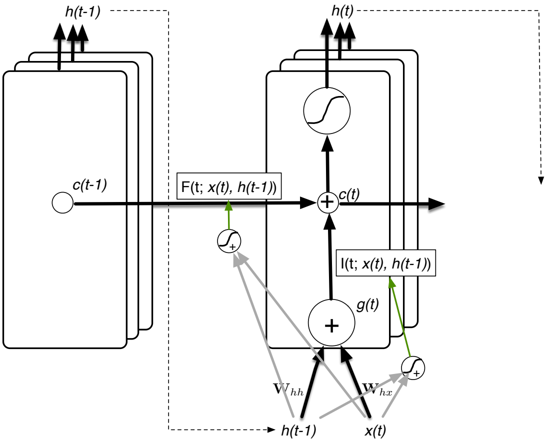
LSTM – Input Gate



LSTM – Forget Gate



LSTM – Input and Forget Gates



$$\mathbf{I}(t) = \sigma\left(\mathbf{W}_{ix}\mathbf{x}(t) + \mathbf{W}_{ih}\mathbf{h}(t-1) + \mathbf{b}_i\right)$$

$$\mathbf{F}(t) = \sigma\left(\mathbf{W}_{fx}\mathbf{x}(t) + \mathbf{W}_{fh}\mathbf{h}(t-1) + \mathbf{b}_f\right)$$

σ is the sigmoid function

$$\mathbf{g}(t) = \mathbf{W}_{hx}\mathbf{x}(t) + \mathbf{W}_{hh}\mathbf{h}(t-1) + \mathbf{b}_h$$

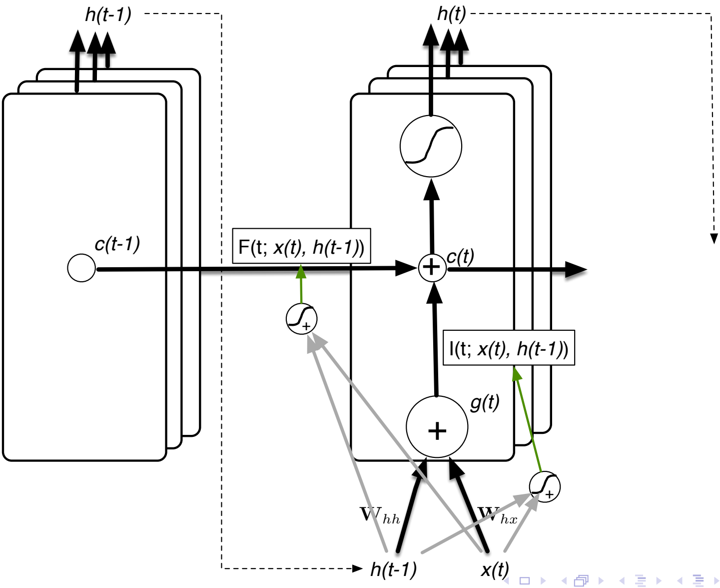
$$\mathbf{c}(t) = \mathbf{F}(t) \circ \mathbf{c}(t-1) + \mathbf{I}(t) \circ \mathbf{g}(t)$$

$\circ$ is element-wise vector multiply

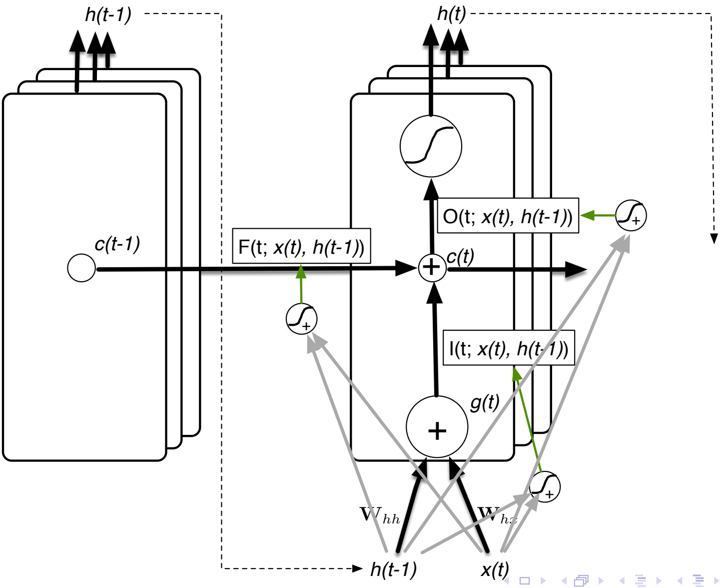
LSTM

- **Internal recurrent state** ("cell") $c(t)$ combines previous state $c(t-1)$ and LSTM input $g(t)$
- Gates - weights dependent on the current input and the previous state
- **Input gate**: controls how much input to the unit $g(t)$ is written to the internal state $c(t)$
- **Forget gate**: controls how much of the previous internal state $c(t-1)$ is written to the internal state $c(t)$
  - Input and forget gates together allow the network to control what information is stored and overwritten at each step
- **Output gate**: controls how much of each unit's activation is output by the hidden state – it allows the LSTM cell to kepp information that is not relevant at the current time, but may be relevant later

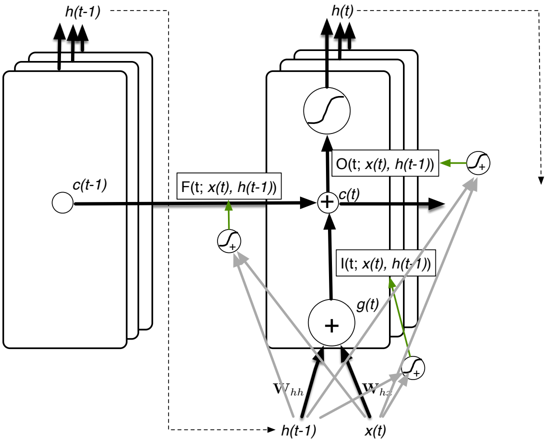
LSTM – Input and Forget Gates



LSTM – Output Gate

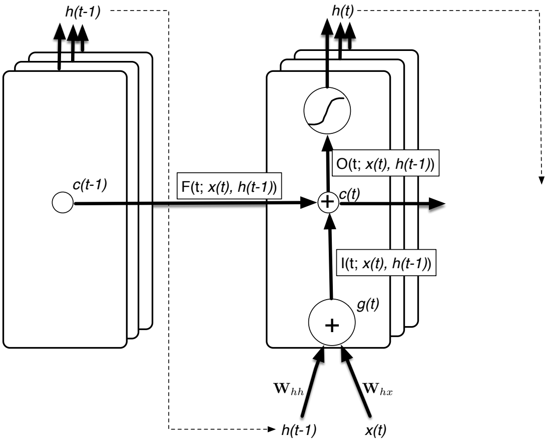


LSTM – Output Gate



$$\mathbf{O}(t) = \sigma\left(\mathbf{W}_{ox}\mathbf{x}(t) + \mathbf{W}_{oh}\mathbf{h}(t-1) + \mathbf{b}_o\right) \qquad \mathbf{h}(t) = \tanh\left(\mathbf{O}(t) \circ \mathbf{c}(t)\right)$$

LSTM



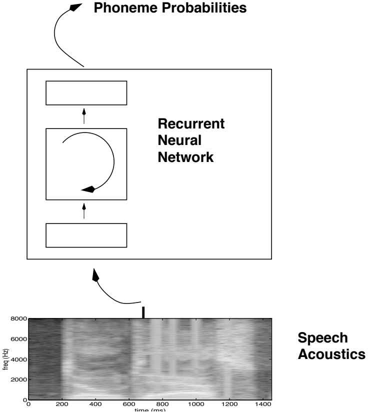
$$\mathbf{I}(t) = \sigma\left(\mathbf{W}_{ix}\mathbf{x}(t) + \mathbf{W}_{ih}\mathbf{h}(t-1) + \mathbf{b}_i\right) \qquad \mathbf{g}(t) = \mathbf{W}_{hx}\mathbf{x}(t) + \mathbf{W}_{hh}\mathbf{h}(t-1) + \mathbf{b}_h$$

$$\mathbf{F}(t) = \sigma\left(\mathbf{W}_{fx}\mathbf{x}(t) + \mathbf{W}_{fh}\mathbf{h}t - 1) + \mathbf{b}_f\right) \qquad \mathbf{c}(t) = \mathbf{F}(t) \circ \mathbf{c}(t-1) + \mathbf{I}(t) \circ \mathbf{g}(t)$$

$$\mathbf{O}(t) = \sigma\left(\mathbf{W}_{ox}\mathbf{x}(t) + \mathbf{W}_{oh}\mathbf{h}(t-1) + \mathbf{b}_o\right) \qquad \mathbf{h}(t) = \tanh\left(\mathbf{O}(t) \circ \mathbf{c}(t)\right)$$

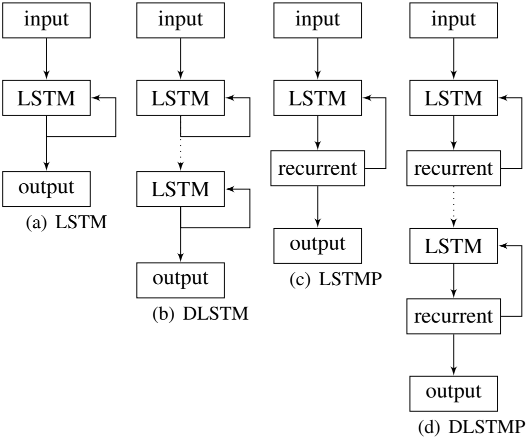
Example applications using RNNs

Example 1: speech recognition with recurrent networks



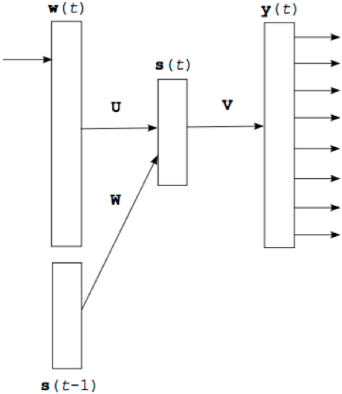
T Robinson et al (1996). "The use of recurrent networks in continuous speech recognition", in *Automatic Speech and Speaker Recognition Advanced Topics* (Lee et al (eds)), Kluwer, 233–258.

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*.

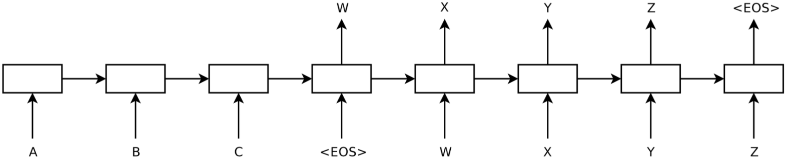
Example 3: recurrent network language models

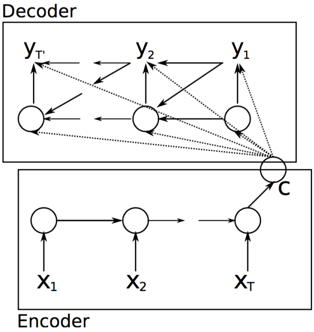


T Mikolov et al (2010). "Recurrent Neural Network Based Language Model", *Interspeech*

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
- More on recurrent networks next semester in NLU (and 1-2 lectures in ASR and MT)