Natural Language Understanding

Lecture 13: Long Short-term Memory

Frank Keller

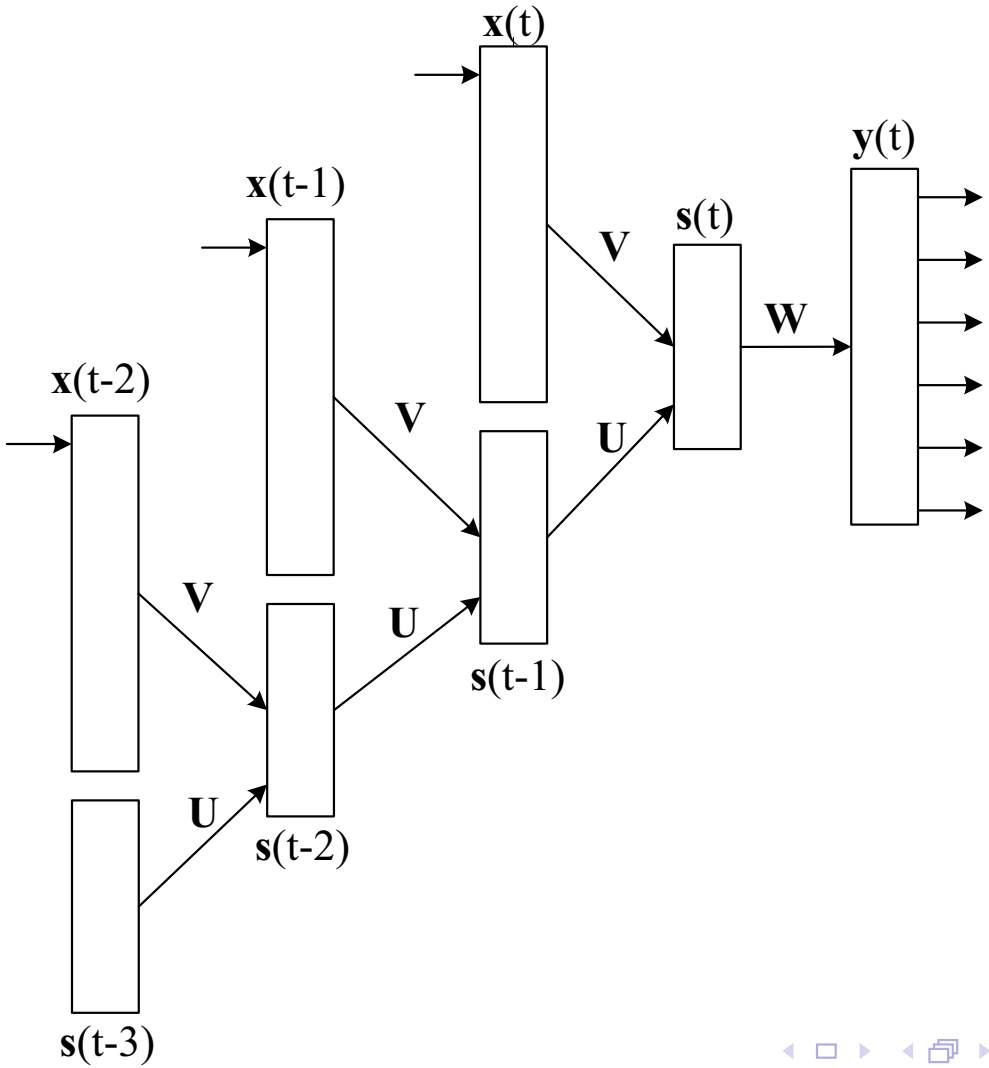
School of Informatics
University of Edinburgh
keller@inf.ed.ac.uk

March 10, 2017

Reading: Graves (2012: Ch. 4).
Additional background: Hochreiter and Schmidhuber (1997).

RNN Architecture

Recall the unfolded RNN from last lecture:



Standard Backpropagation

For output units, we update the weights **W** using:

$$\Delta w_{kj} = \eta \sum_{p}^{n} \delta_{pk} s_{pj} \qquad \delta_{pk} = (d_{pk} - y_{pk}) g'(net_{pk})$$

where d_{pk} is the desired output of unit k for training pattern p.
For hidden units, we update the weights **V** using:

$$\Delta v_{ji} = \eta \sum_{p}^{n} \delta_{pj} x_{pi} \qquad \delta_{pj} = \sum_{k}^{o} \delta_{pk} w_{kj} f'(net_{pj})$$

This is standard backprop, with the gradients highlighted.

Vanishing Gradients

We adjust **U** using backprop through time. For timestep t:

$$\Delta u_{ji} = \eta \sum_{p}^{n} \delta_{pj}(t) s_{ph}(t-1) \qquad \delta_{pj}(t) = \sum_{k}^{o} \delta_{pk} w_{kj} f'(net_{pj})$$

For timestep $t-1$:

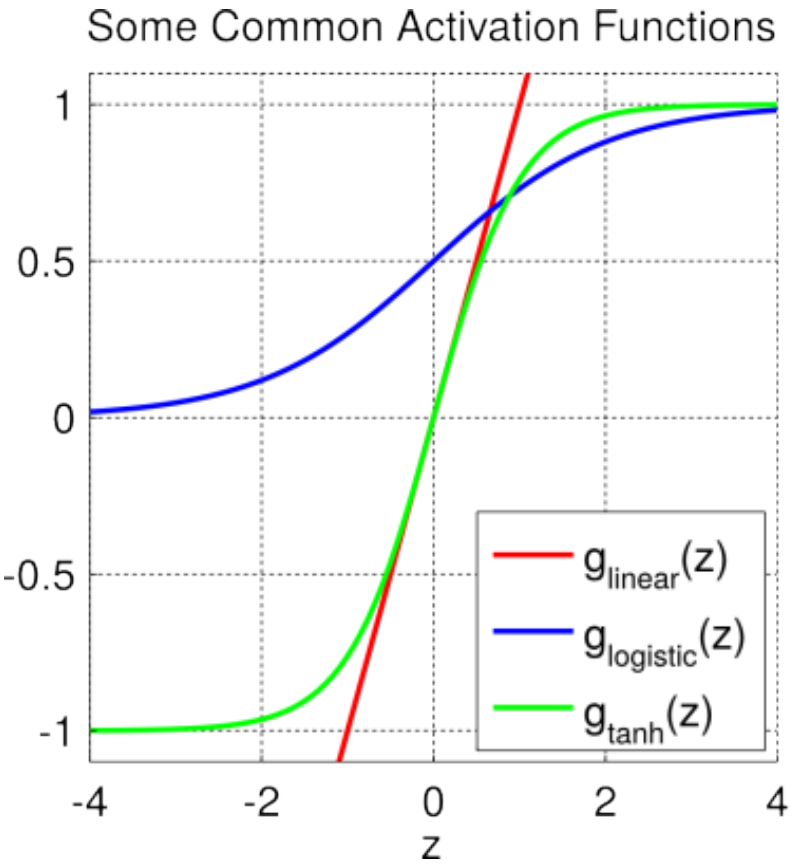
$$\delta_{pj}(t-1) = \sum_{h}^{m} \delta_{ph}(t) u_{hj} f'(s_{pj}(t-1))$$

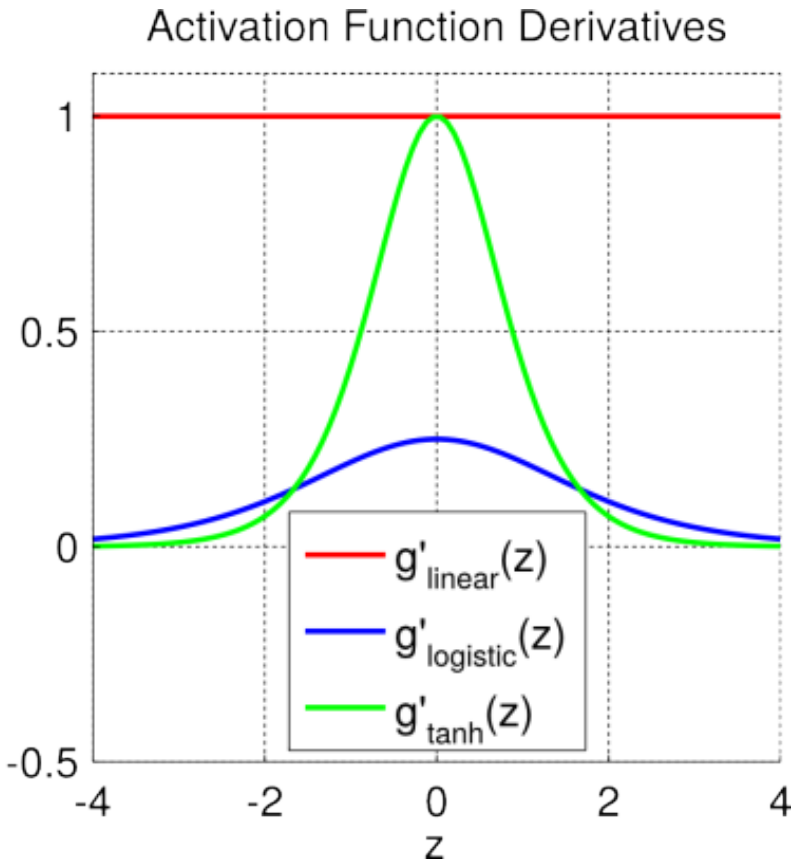
For time step $t-2$:

$$\begin{aligned} \delta_{pj}(t-2) &= \sum_{h}^{m} \delta_{ph}(t-1) u_{hj} f'(s_{pj}(t-2)) \\ &= \sum_{h}^{m} \sum_{h_1}^{m} \delta_{ph_1}(t) u_{h_1 j} f'(s_{pj}(t-1)) u_{hj} f'(s_{pj}(t-2)) \end{aligned}$$

Vanishing Gradients

At every time step, we multiply the weights with another gradient. The gradients are < 1 so the deltas become smaller and smaller.

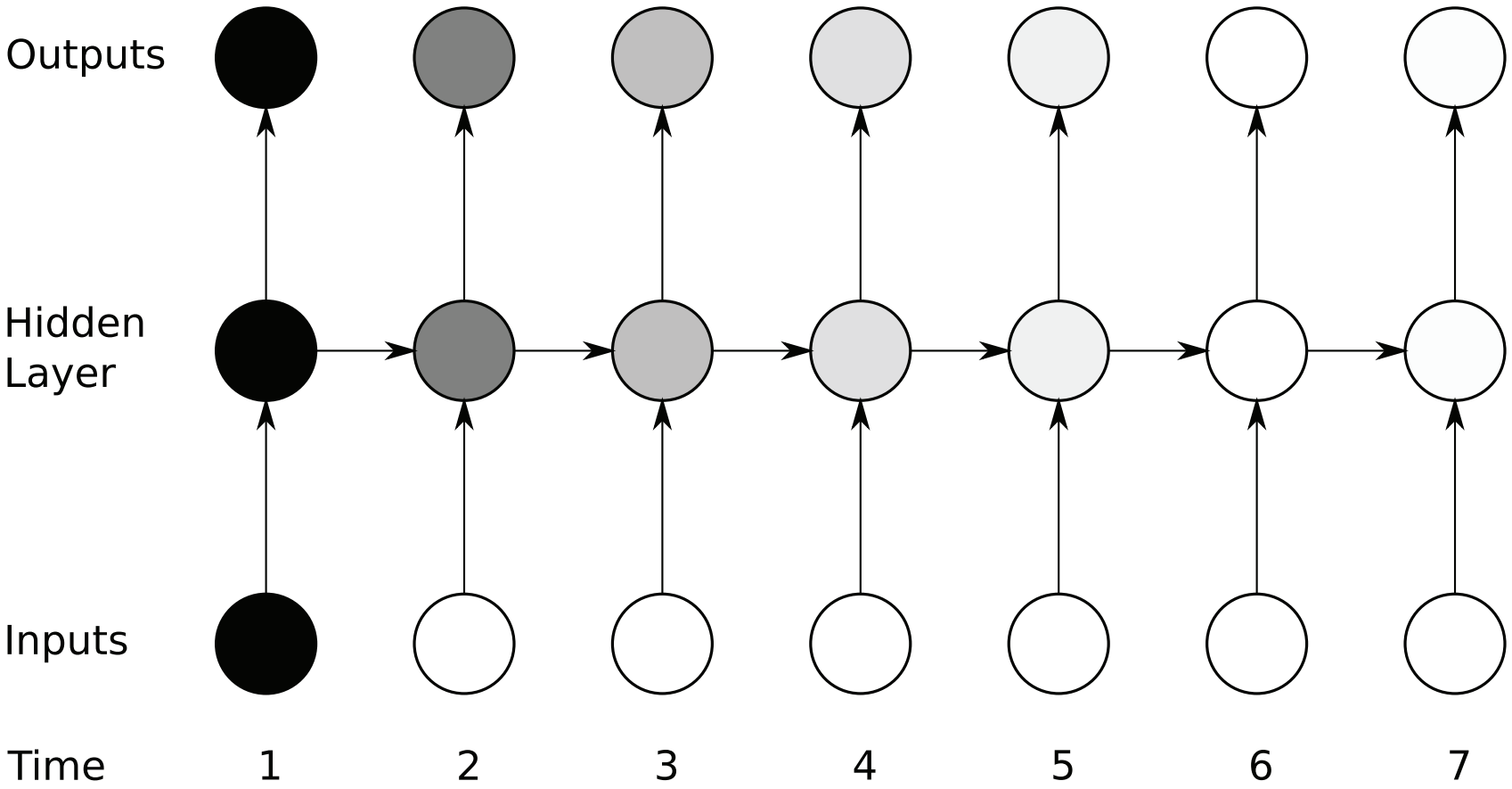




[Source: https://theclevermachine.wordpress.com/]

Vanishing Gradients

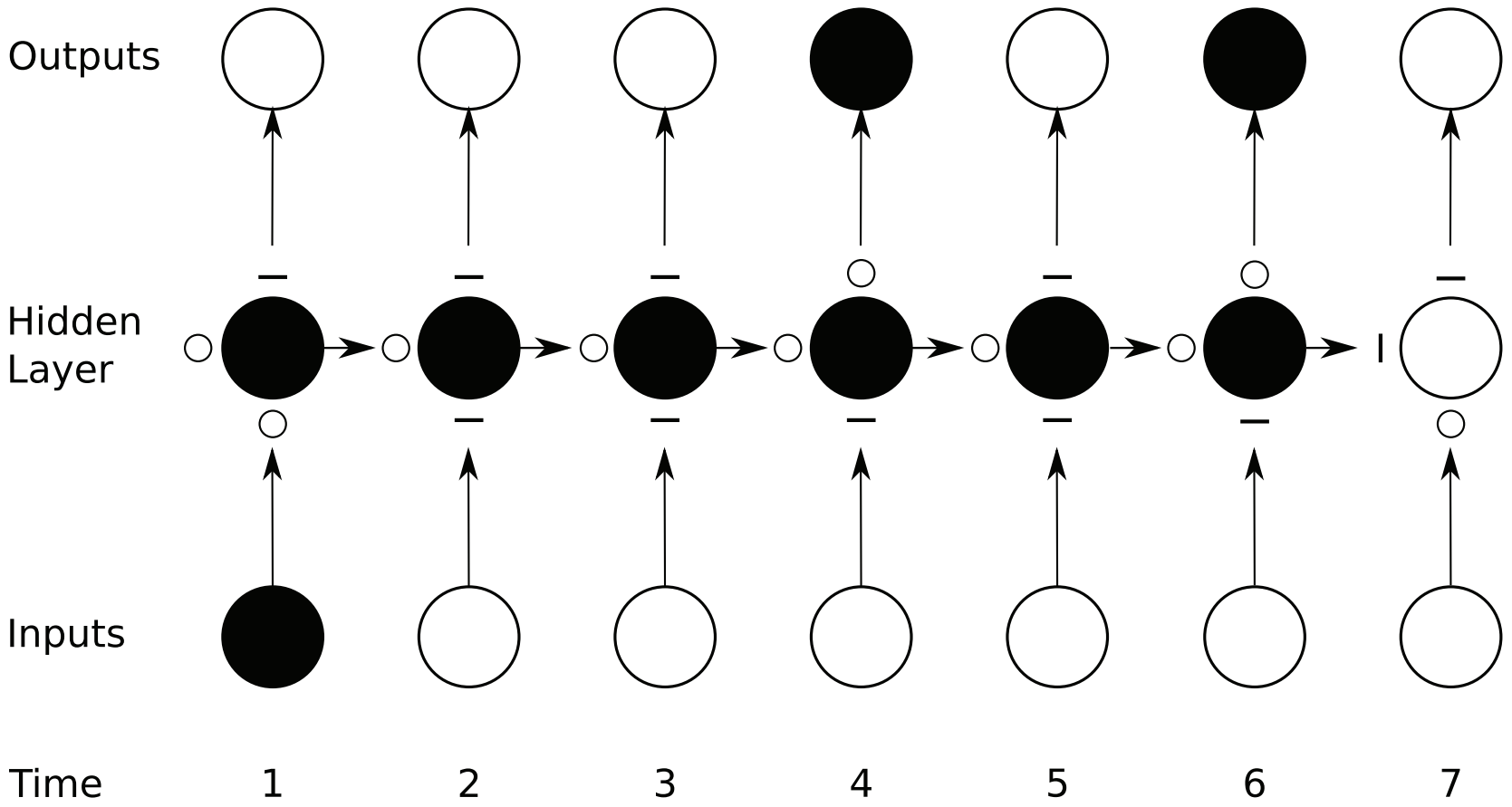
So in fact, the RNN is not able to learn long-range dependencies well, as the gradient vanishes: it rapidly "forgets" previous inputs:



[Source: Graves (2012).]

Long Short-term Memory

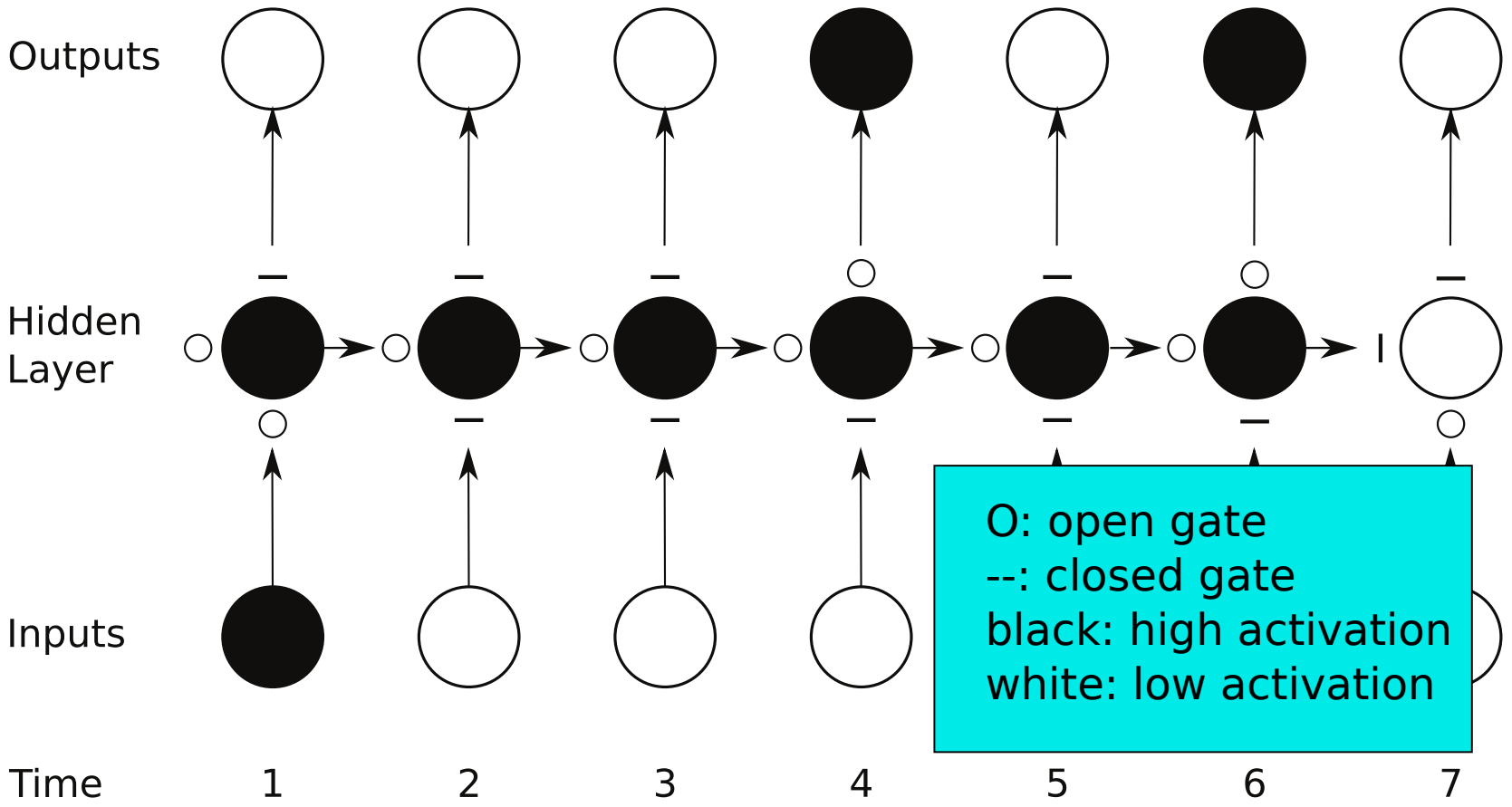
Solution: network can sometimes pass on information from previous time steps unchanged, so that it can learn from distant inputs:



[Source: Graves (2012).]

Long Short-term Memory

Solution: network can sometimes pass on information from previous time steps unchanged, so that it can learn from distant inputs:



[Source: Graves (2012).]

Architecture

To achieve this, we need to make the units of the network more complicated:

- LSTMs have a hidden layer of *memory blocks;*
- each block contains a recurrent *memory cell* and three multiplicative units: the *input, output and forget gates;*
- the gates are trainable: each block can learn whether to keep information across time steps or not.

In contrast, the RNN uses simple hidden units, which just sum the input and pass it through an activation function.

The Gates and the Memory Cell

Each memory block consists of four units:

Input gate: controls whether the input to is passed on to the memory cell or ignored;

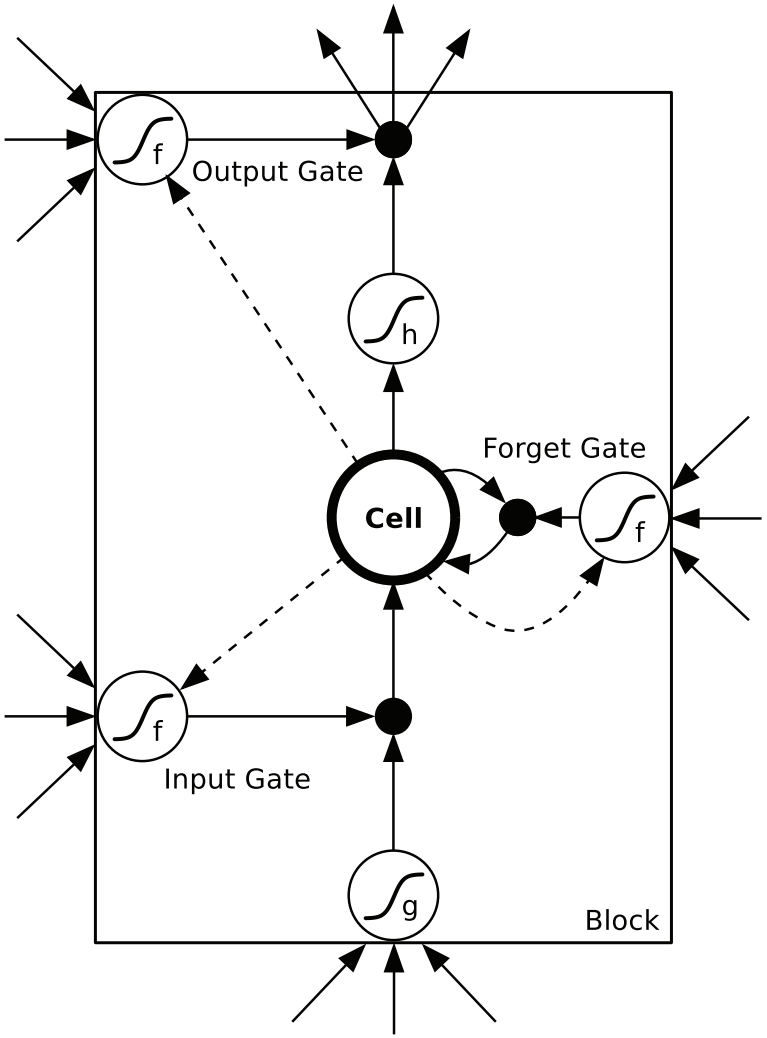
Output gate: controls whether the current activation vector of the memory cell is passed on to the output layer or not;

Forget gate: controls whether the activation vector of the memory cell is reset to zero or maintained;

Memory cell: stores the current activation vector; with recurrent connection to itself controlled by forget gate.

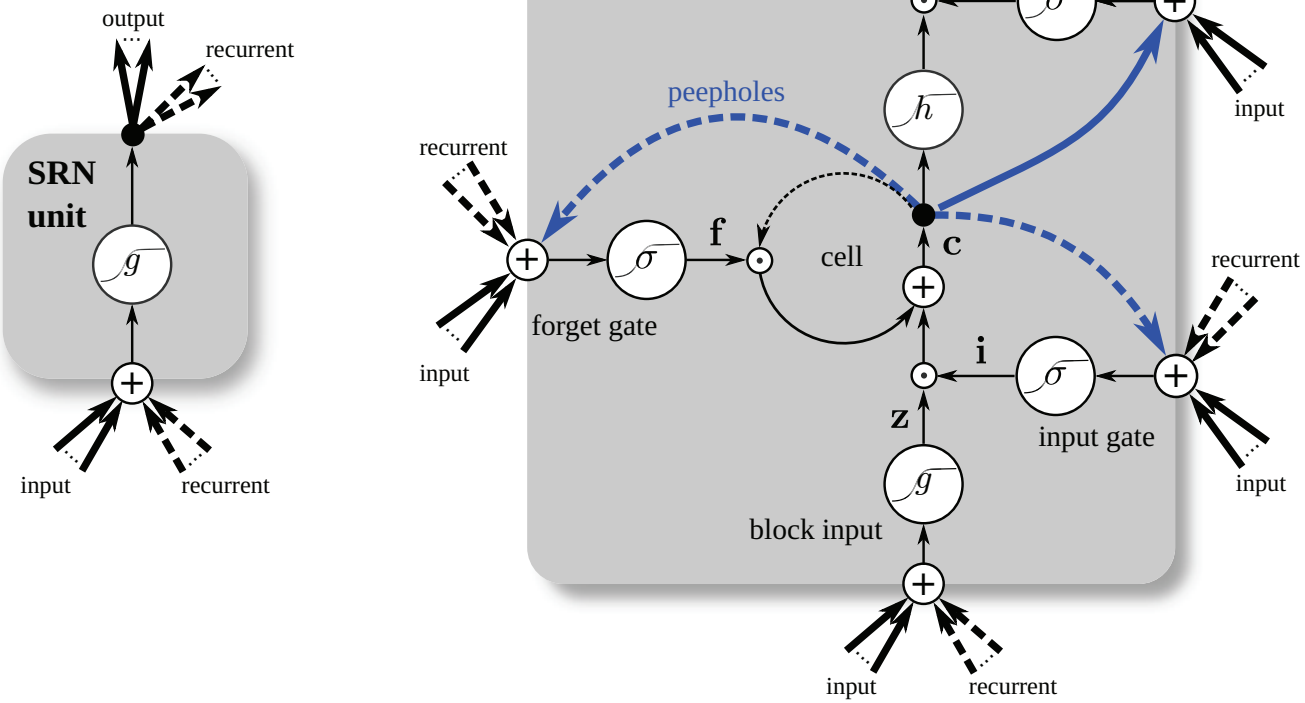
There are also peephole connections; we won't discuss these.

A Single LSTM Memory Block



[Source: Graves (2012).]

RNN Unit compared to LSTM Memory Block



Legend

- unweighted connection
- weighted connection
- connection with time-lag
- branching point
- mutliplication
- sum over all inputs
- gate activation function (always sigmoid)
- input activation function (usually tanh)
- output activation function (usually tanh)

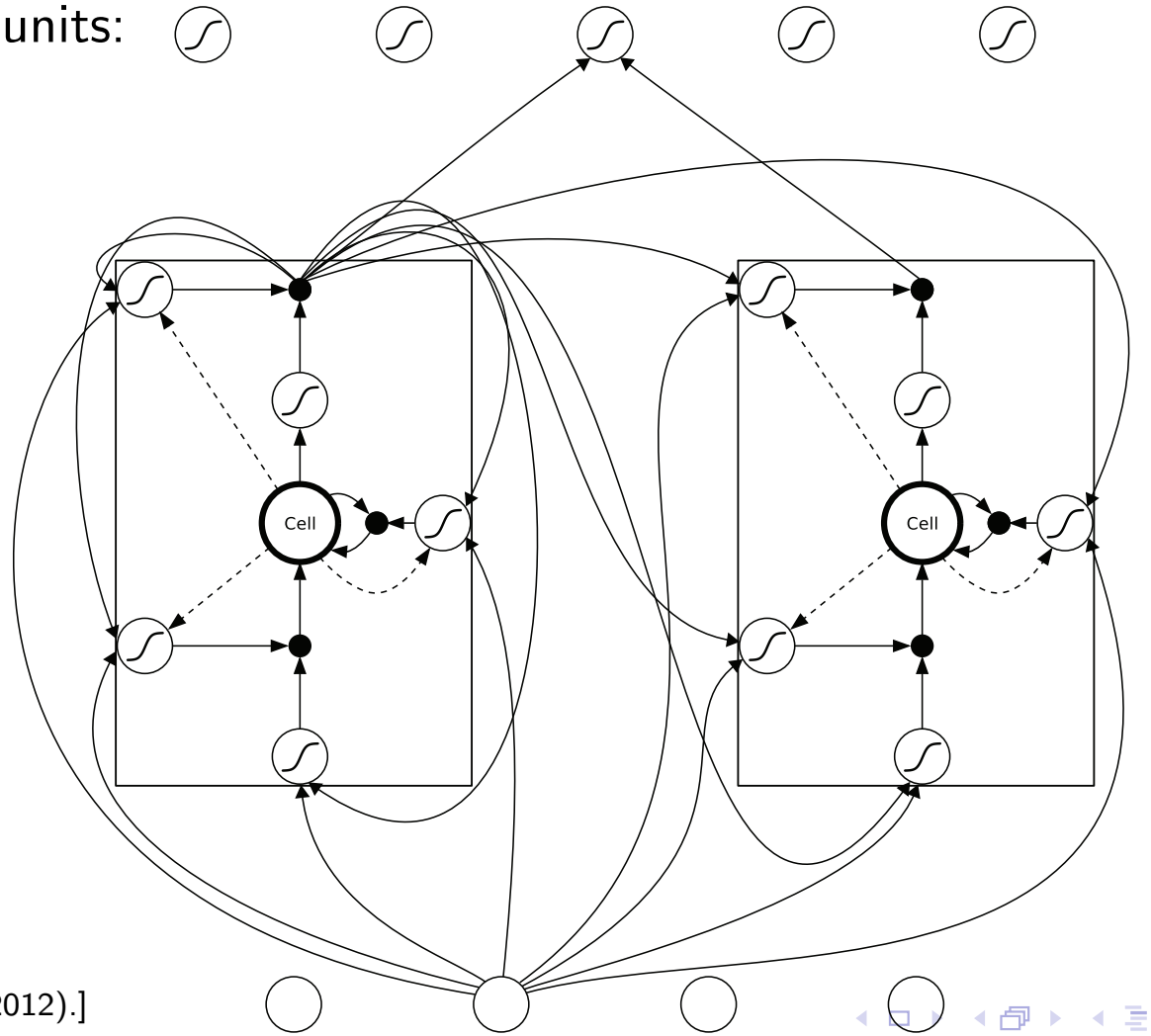
[Source: Klaus Greff et al.: LSTM: A Search Space Odyssey, 2015.]

The Gates and the Memory Cell

- Gates are *regular hidden units:* they sum their input and pass it through a sigmoid activation function;
- all four inputs to the block are the same: the *input layer and the recurrent layer* (hidden layer at previous time step);
- all gates have *multiplicative connections:* if the activation is close to zero, then the gate doesn't let anything through;
- the *memory cell* itself is linear: it has no activation function;
- but the block as a whole has input and output activation functions (can be tanh or sigmoid);
- all connections within the block are *unweighted:* they just pass on information (i.e., copy the incoming vector);
- the only output that the rest of the network sees is what the output gate lets through.

Putting LSTM Memory Blocks Together

Network with four input units, a hidden layer of two memory blocks and five output units:



[Source: Graves (2012).]

Vanishing Gradients Again

Why does this solve the vanishing gradient problem?

- the memory cell is linear, so its gradient doesn't vanish;
- an LSTM block can retain information indefinitely: if the forget gate is open (close to 1) and the input gate is closed (close to 0), then the activation of the cell persists;
- in addition, the block can decide when to output information by opening the output gate;
- the block can therefore retain information over an arbitrary number of time steps before it outputs it;
- the block learns when to accept input, produce output, and forget information: the gates have trainable weights.

Training

- The original LSTM (Hochreiter and Schmidhuber 1997) was trained with backprop through time;
- but BPTT was truncated after one step, assuming that long-range dependencies are dealt with by LSTM blocks;
- recall that the activation of the memory cell doesn't vanish (unlike the activation of standard recurrent connections);
- recent work has used untruncated BPTT, resulting in more accurate gradients.

We follow Graves's notation, which extends the RNN notation introduced in the last lecture.

Training: Computing Activations

Note that each LSTM block has four inputs (and four sets of input weights), but only one output.

Activations of the forget gate and the memory cell:

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} \qquad b_\phi^t = f(a_\phi^t)$$

$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \qquad s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t)$$

where a_j^t is the input of unit j at time t, b_j^t is the activation of j, w_{ij} is the weight from i to j, x_j^t is the input to j, s_c^t is the state of memory cell c, ϕ is the forget gate, ι is the input gate.

Training: Backpropagating Gradients

Deltas of the forget gate and the memory cell:

$$\delta_\phi^t = f'(a_\phi^t) \sum_{c=1}^{C} s_c^{t-1} \epsilon_s^t$$

$$\delta_c^t = b_\iota^t g'(a_c^t) \epsilon_s^t$$

where ϵ_s^t is the error of state s at time t, which is the sum of the error at the output gate and at the forget gate.

Note that we again ignore peephole connections. For full details see Graves (2012: Ch. 4.6).

Applications

Language modeling: use the LSTM just like a standard RNN:

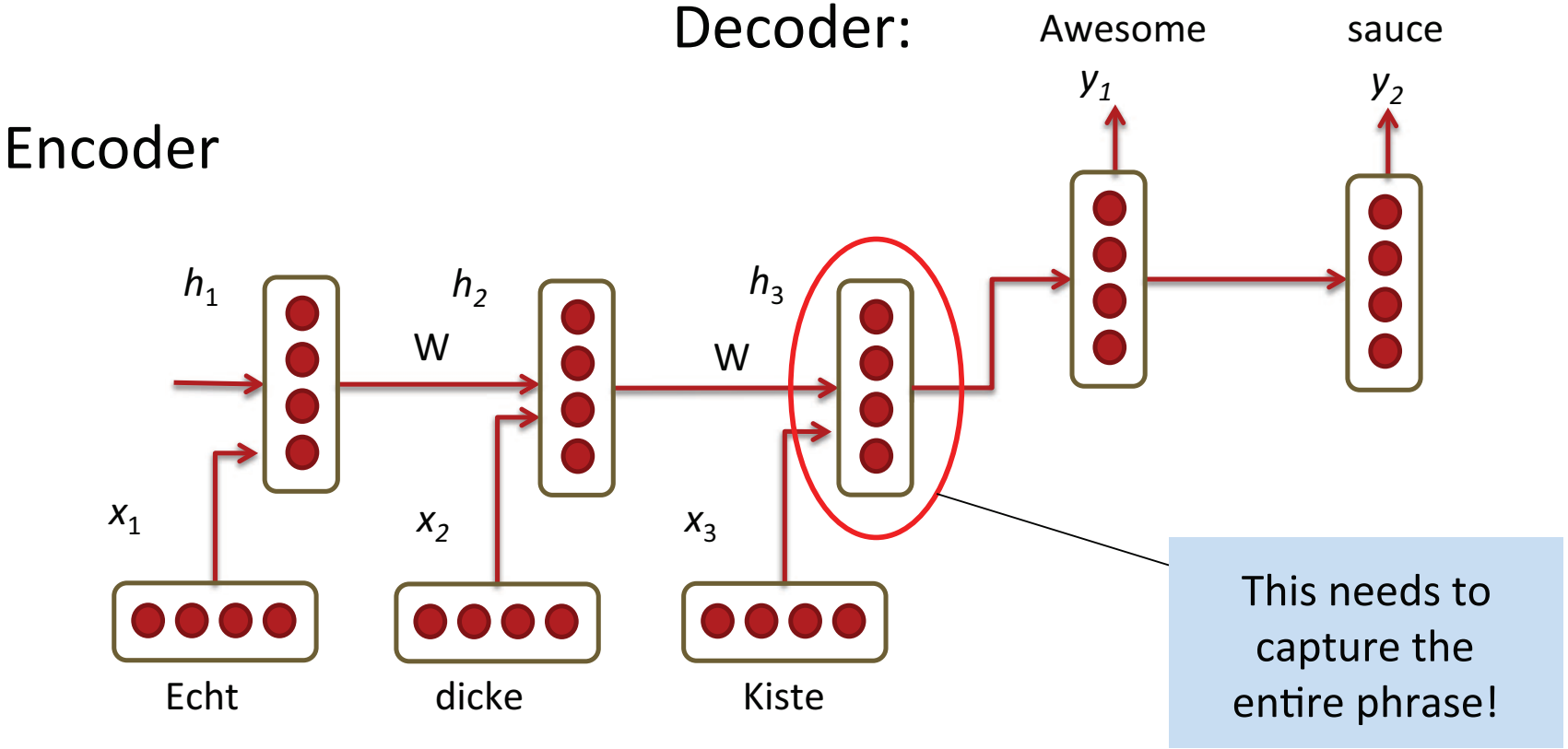
Method	Perplexity
LSTM (512 units)	68.8
IRNN (4 layers, 512 units)	69.4
IRNN (1 layer, 1024 units + linear projection)	70.2
RNN (4 layers, 512 tanh units)	71.8
RNN (1 layer, 1024 tanh units + linear projection)	72.5

[Source: Quoc V. Le et al. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units, 2015.]

IRNN: network of ReLUs (rectified linear units) initialized with identify matrix. The ReLU activation function is $f(z) = \max(z, 0)$.

Applications

Machine Translation: train the LSTM to memorize the source and the target sentence in sequence (encoder-decoder architecture):



[Source: Richard Socher: Fancy Recurrent Neural Networks for Machine Translation. Deep NLP Course, Stanford.]

Applications

LSTMs are useful for lots of sequence labeling tasks:

- part of speech tagging and parsing;
- semantic role labeling;
- opinion mining.

The encoder-decoder paradigm can be used for sequence-to-sequence mapping tasks:

- question answering;
- summarization;
- sentence compression and simplification.

We will see some of these applications in the rest of the course.

Summary

- Backprop through time with RNNs has the problem that gradients vanish with increasing timesteps;
- the LSTM is a way of addressing this problem;
- it replaces additive hidden units with complex memory blocks;
- a linear memory cell is the core of the each block;
- the input gate controls whether the cell receives input; the output gate controls whether the cell state is passed on; the forget gate determines whether the cell state is reset;
- the LSTM can be trained with standard backprop or BPTT;
- applications include sequence labeling and sequence-to-sequence mapping tasks.

References

Graves, Alex. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, Berlin.

Hochreiter, Sepp and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.