

Recurrent Neural Networks

DR RAYMOND LEE
ASSOCIATE PROFESSOR
DIVISION OF SCIENCE AND TECHNOLOGY
BNU-HKBU UNITED INTERNATIONAL COLLEGE

RoadMap

- Introduction
- Motivation
- RNN architecture
- Long Short Term Memory (LSTM) Network
- How LSTM solves the problem
- Conclusions

Introduction

- RNN were introduced in the late 80' s.
- Hochreiter discovers the 'vanishing gradients' problem in 1991.
- Long Short Term Memory published in 1997.
- LSTM a recurrent network to overcome these problems.

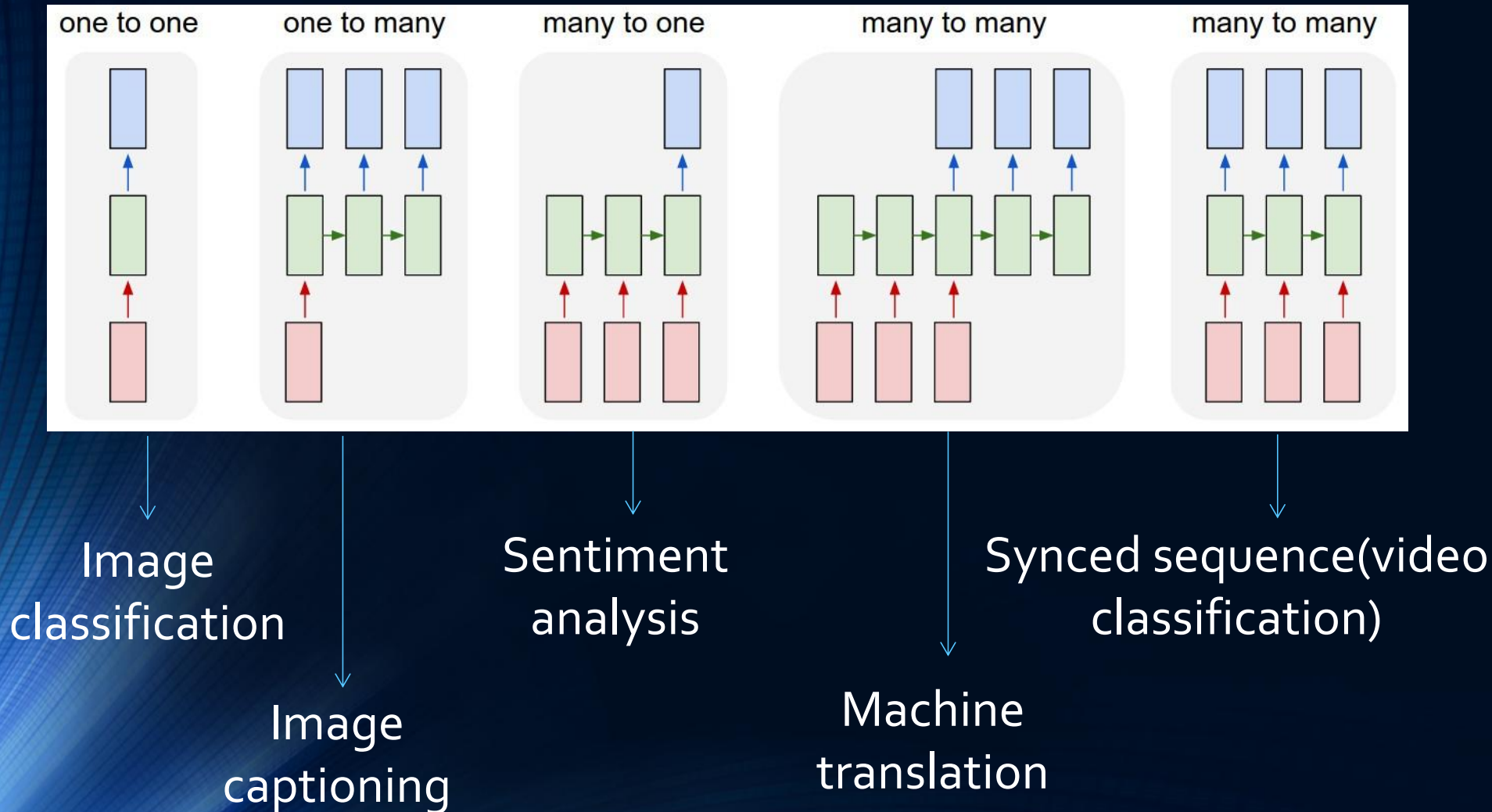
RoadMap

- Introduction
- **Motivation**
- RNN architecture
- LSTM
- LSTM experiments
- Conclusions

Motivation

- Feed forward networks accept a fixed-sized vector as input and produce a fixed-sized vector as output
- Fixed amount of computational steps
- Recurrent nets allow us to operate over *sequences* of vectors

Motivation

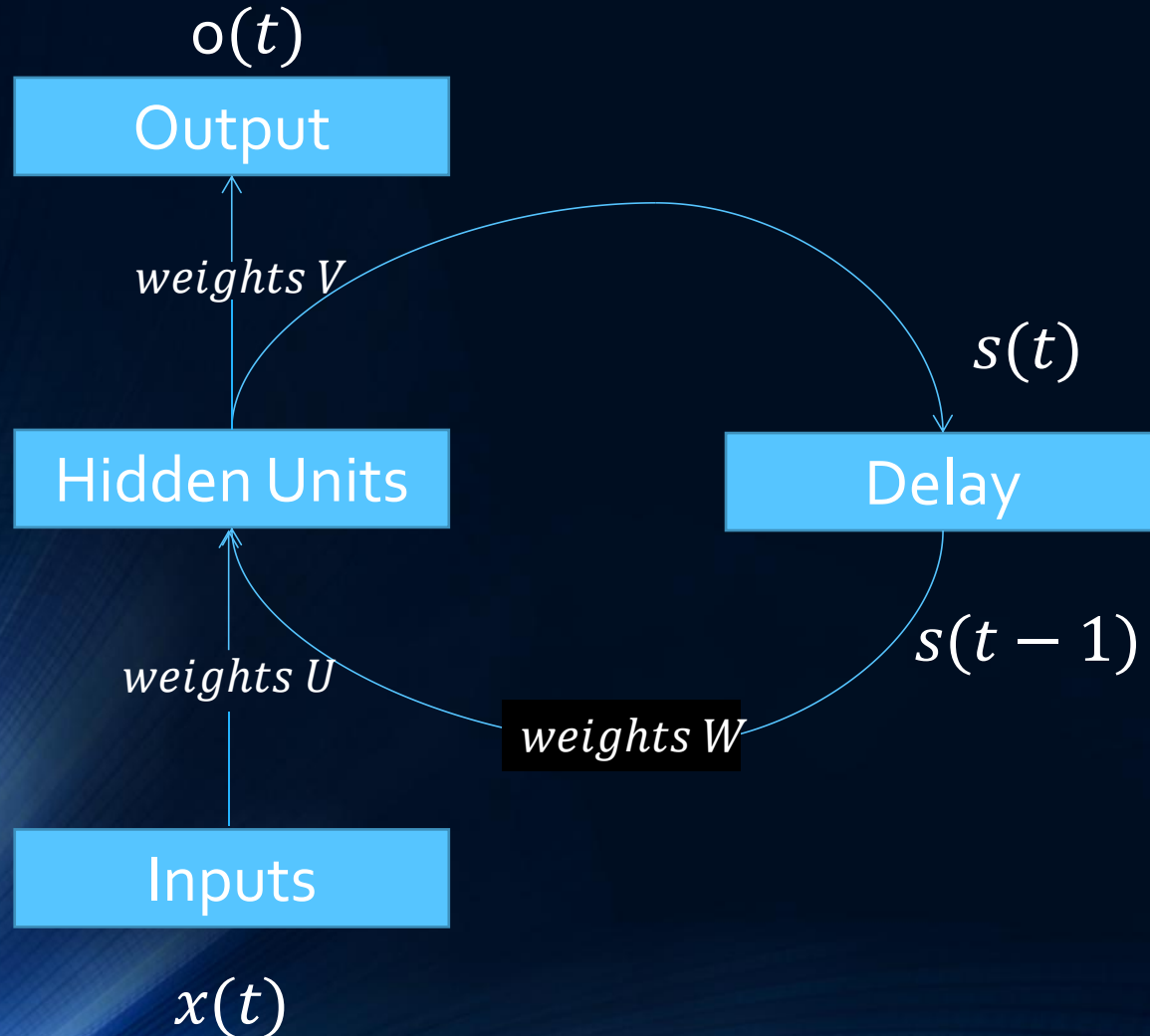


- The idea behind RNNs is to make use of sequential information.
- In a traditional neural network we assume that all inputs (and outputs) are independent of each other.
- But for many tasks that's a very bad idea. If you want to predict the next word in a sentence you better know which words came before it.

RoadMap

- Introduction
- Motivation
- RNN architecture
- Long Short Term Memory (LSTM) Network
- LSTM experiments
- Conclusions

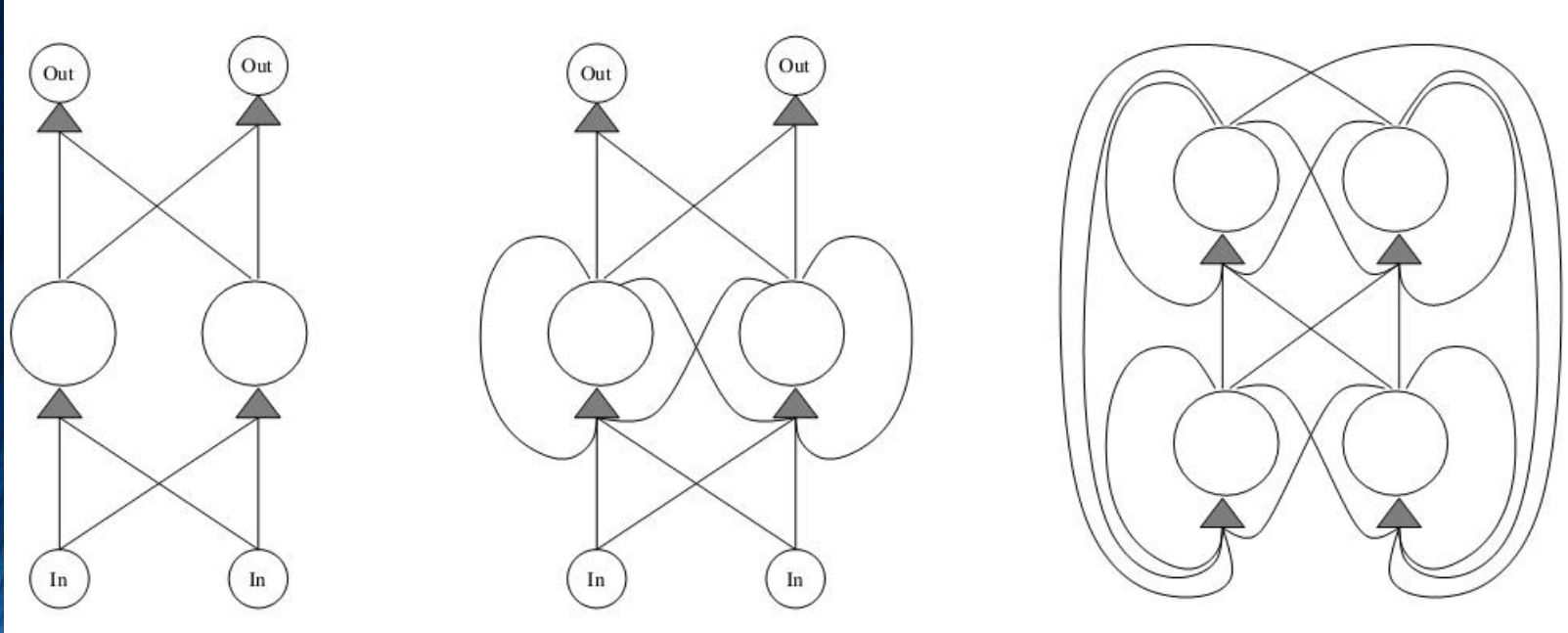
RNN Architecture



Note:

- RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations (memory).
- Inputs $x(t)$ outputs $y(t)$ hidden state $s(t)$ the memory of the network
- A delay unit is introduced to hold activation until they are processed at the next step
- The decision a recurrent net reached at time step $t-1$ affects the decision it will reach one moment later at time step t .
- So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data

RNN Architecture



Note:

- RNN topologies range from partly recurrent to fully recurrent.
- Partly recurrent is a layered network with distinct input and output layers where the recurrence is limited to the hidden layer.
- In fully recurrent networks each node gets inputs from all other nodes.

Left: feed forward neural network

Middle: a simple recurrent neural network

Right: Fully connected recurrent neural network

RNN Forward Pass

- The network input at time t:

$$a_h(t) = Ux(t) + Ws(t - 1)$$

- The activation of the input at time t:

$$s(t) = f_h(a_h(t))$$

- The network input to the output unit at time t:

$$a_o(t) = Vs(t)$$

- The output of the network at time t is:

$$o(t) = f_o(a_o(t))$$

Note:

The complete sequence of hidden activations can be calculated starting at $t=1$ and recursively applying these equations incrementing t at each step.

RNN Architecture

- If a network training sequence starts at time t_0 and ends at t_1 , the total **loss function** is the sum over time of the square error function

$$[e(t)]_k = [d(t)]_k - [o(t)]_k$$

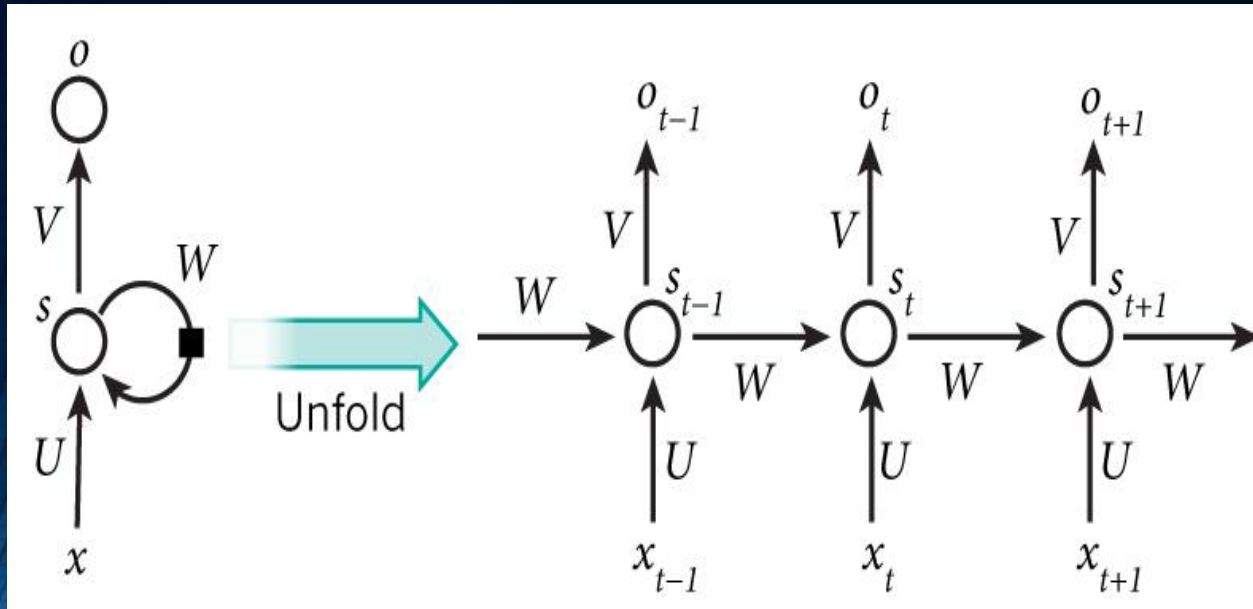
$$E(t) = \frac{1}{2} e(t)^T e(t)$$

$$E_{total} = \sum_t E(t)$$

Note:

- U, W, V weight matrices, F_h, F_o hidden and output activation functions
- For k outputs at time t
 $[e(t)]_k$ is a vector size k
- This type of error is better for regression and not classification (for classification a softmax is better)
- For RNN another cost function which is used in language models is the cross entropy (sum over label $\ln(y)$) used with a softmax activation function.

RNN Architecture



- The recurrent network can be converted into a feed forward network by **unfolding over time**

Note:

- This diagram shows a RNN being unrolled (or unfolded) into a full network.
- By unrolling we simply mean that we write out the network for the complete sequence.
- For example, if the sequence we care about is a sentence of 5 words, the network would be unrolled into a 5-layer neural network, one layer for each word.
- This means all the earlier theory about feed forward networks learning follows through
- In RNN errors can be propagated more than 2 layers in order to capture longer history information.
- This process is usually called unfolding.
- In an unfolded RNN the recurrent weight is duplicated for an arbitrary number of time steps (referred to as t)
- The above diagram has outputs at each time step, but depending on the task this may not be necessary.
- For example, when predicting the sentiment of a sentence we may only care about the final output, not the sentiment after each word.
- Similarly, we may not need inputs at each time step.
- The main feature of an RNN is its hidden state, which captures some information about a sequence.

Back Propagation Through Time

- BPTT learning algorithm is an extension of standard backpropagation that performs gradients descent on an unfolded network.
- The gradient descent weight updates have contributions from each time step.
- The errors have to be back-propagated through time as well as through the network

Note:

- The key difference is that we sum up the gradients for at each time step. In a traditional NN we don't share parameters across layers, so we don't need to sum anything.
- Backpropagation in feedforward networks moves backward from the final error through the outputs, weights and inputs of each hidden layer, assigning those weights responsibility for a portion of the error by calculating their partial derivatives – $\partial E / \partial w$, or the relationship between their rates of change. Those derivatives are then used by our learning rule, gradient descent, to adjust the weights up or down, whichever direction decreases error.

RNN Backward Pass

- For recurrent networks, the loss function depends on the activation of the hidden layer through its influence on the output layer **and** through its influence on the hidden layer at the next step.

$$\delta_j(t) = \frac{\partial E}{\partial a_j(t)}, j \in \{o, h\} \quad , \odot \text{ is the Hadamard product}$$

RNN Backward Pass

- For recurrent networks, the loss function depends on the activation of the hidden layer through its influence on the output layer **and** through its influence on the hidden layer at the next step.

$$\frac{\partial E}{\partial a_o(t)} = \frac{\partial E}{\partial o(t)} \frac{\partial o(t)}{\partial a_o(t)} = \delta_o(t) = (d(t) - o(t))f'_o(a_o(t))$$

$$\delta_j(t) = \frac{\partial E}{\partial a_j(t)}, j \in \{o, h\} \quad , \odot \text{ is the Hadamard product}$$

RNN Backward Pass

- For recurrent networks, the loss function depends on the activation of the hidden layer through its influence on the output layer **and** through its influence on the hidden layer at the next step.

$$\frac{\partial E}{\partial a_o(t)} = \frac{\partial E}{\partial o(t)} \frac{\partial o(t)}{\partial a_o(t)} = \delta_o(t) = (d(t) - o(t))f'_o(a_o(t))$$

$$\frac{\partial E}{\partial a_h(t)} = \delta_h(t) = f'_h(a_h(t)) \odot (V^T \delta_o(t) + W^T \delta_h(t+1))$$

$$\delta_j(t) = \frac{\partial E}{\partial a_j(t)}, j \in \{o, h\}, \odot \text{ is the Hadamard product}$$

RNN Backward Pass

- We sum over the whole sequence to get the derivatives w.r.t the network weights:

$$E = \sum_{t=0}^T E_t \rightarrow \frac{\partial E}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial a_h} \frac{\partial a_h}{\partial W} = \sum_{t=0}^T \delta_h(t) \frac{\partial a_h}{\partial W}$$

RNN Backward Pass

- We sum over the whole sequence to get the derivatives w.r.t the network weights:

$$E = \sum_{t=0}^T E_t \rightarrow \frac{\partial E}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial a_h} \frac{\partial a_h}{\partial W} = \sum_{t=0}^T \delta_h(t) \frac{\partial a_h}{\partial W}$$

- $\frac{\partial E}{\partial V} = s(t) \cdot \delta_o(t); \frac{\partial E}{\partial U} = x(t) \cdot \delta_h(t); \frac{\partial E}{\partial W} = s(t-1) \cdot \delta_h(t)$

RNN Backward Pass

- We sum over the whole sequence to get the derivatives w.r.t the network weights:

$$E = \sum_{t=0}^T E_t \rightarrow \frac{\partial E}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial W} = \sum_{t=0}^T \frac{\partial E_t}{\partial a_h} \frac{\partial a_h}{\partial W} = \sum_{t=0}^T \delta_h(t) \frac{\partial a_h}{\partial W}$$

- $\frac{\partial E}{\partial V} = s(t) \cdot \delta_o(t); \frac{\partial E}{\partial U} = x(t) \cdot \delta_h(t); \frac{\partial E}{\partial W} = s(t-1) \cdot \delta_h(t)$

- Updating the weights:
$$V(t+1) = V(t) + \alpha \frac{\partial E}{\partial V}$$
$$U(t+1) = U(t) + \alpha \frac{\partial E}{\partial U}$$
$$W(t+1) = W(t) + \alpha \frac{\partial E}{\partial W}$$

Back Propagation Through Time

$$\sum_{t=0} \frac{\partial E_t}{\partial W} = \sum_{t=0} \frac{\partial E_t}{\partial a_h} \frac{\partial a_h}{\partial W}$$

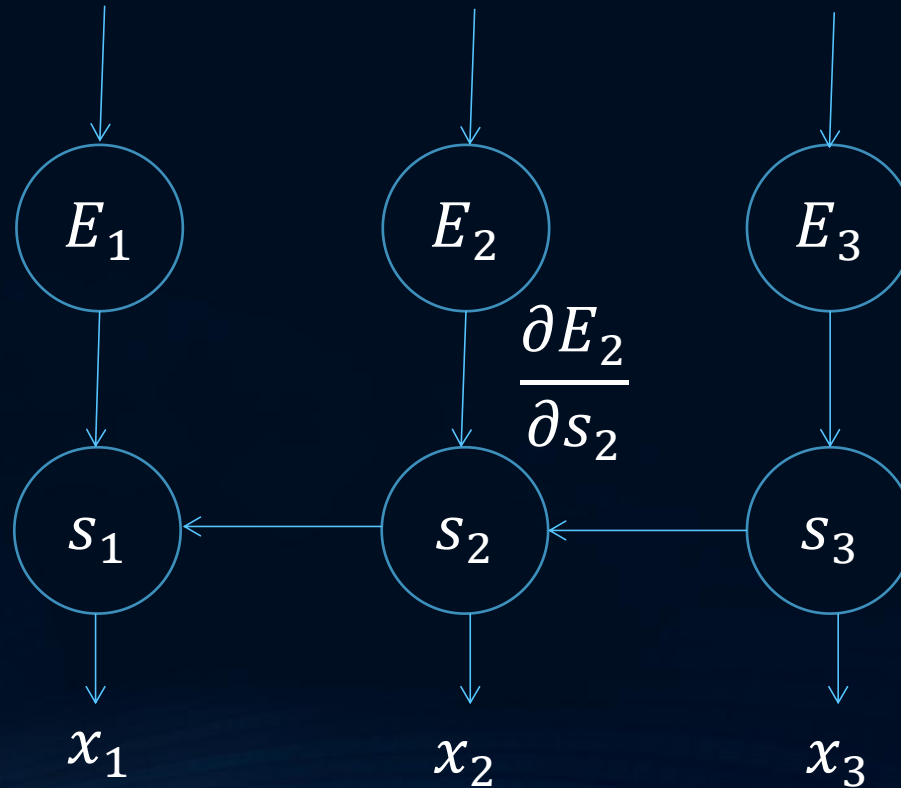
$$\frac{\partial E_t}{\partial W} = \sum_k^t \frac{\partial E_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}; \quad \frac{\partial s_t}{\partial s_k} = \prod_{t>i>k} \frac{\partial s_i}{\partial s_{i-1}}$$

$$\frac{\partial E_t}{\partial W} = \sum_k^t \frac{\partial E_t}{\partial s_t} \left(\prod_{j=k+1}^t \frac{s_j}{s_{j-1}} \right) \frac{\partial s_k}{\partial a_h} \frac{\partial a_h}{\partial W_{20}}$$

Back Propagation Through Time

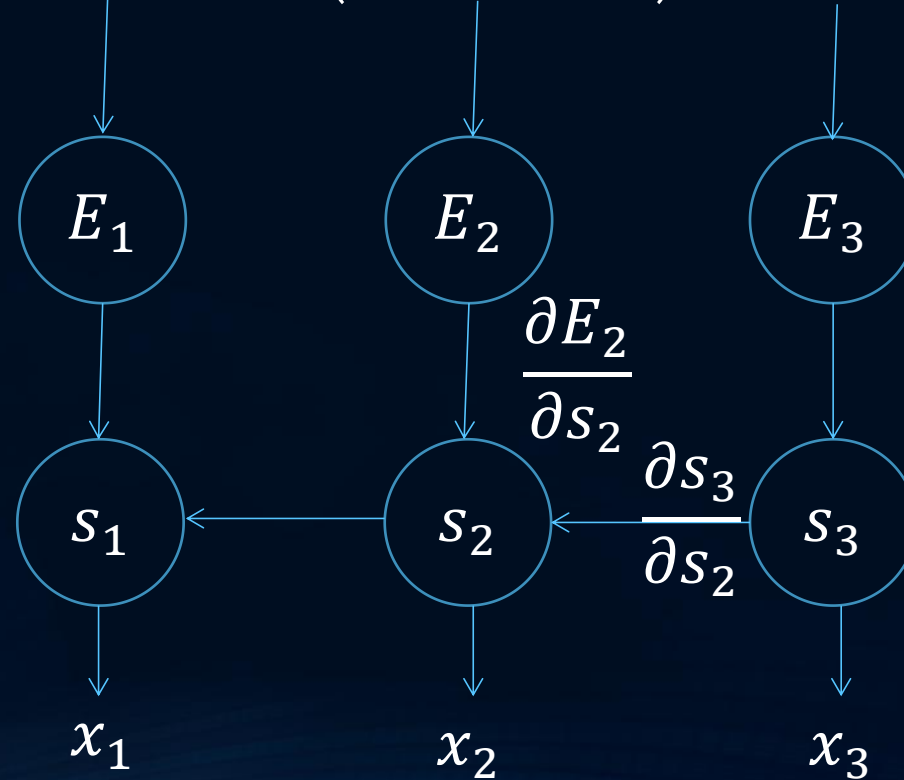
Remember:

$$s_2 = f(Ux_t + Ws_1)$$

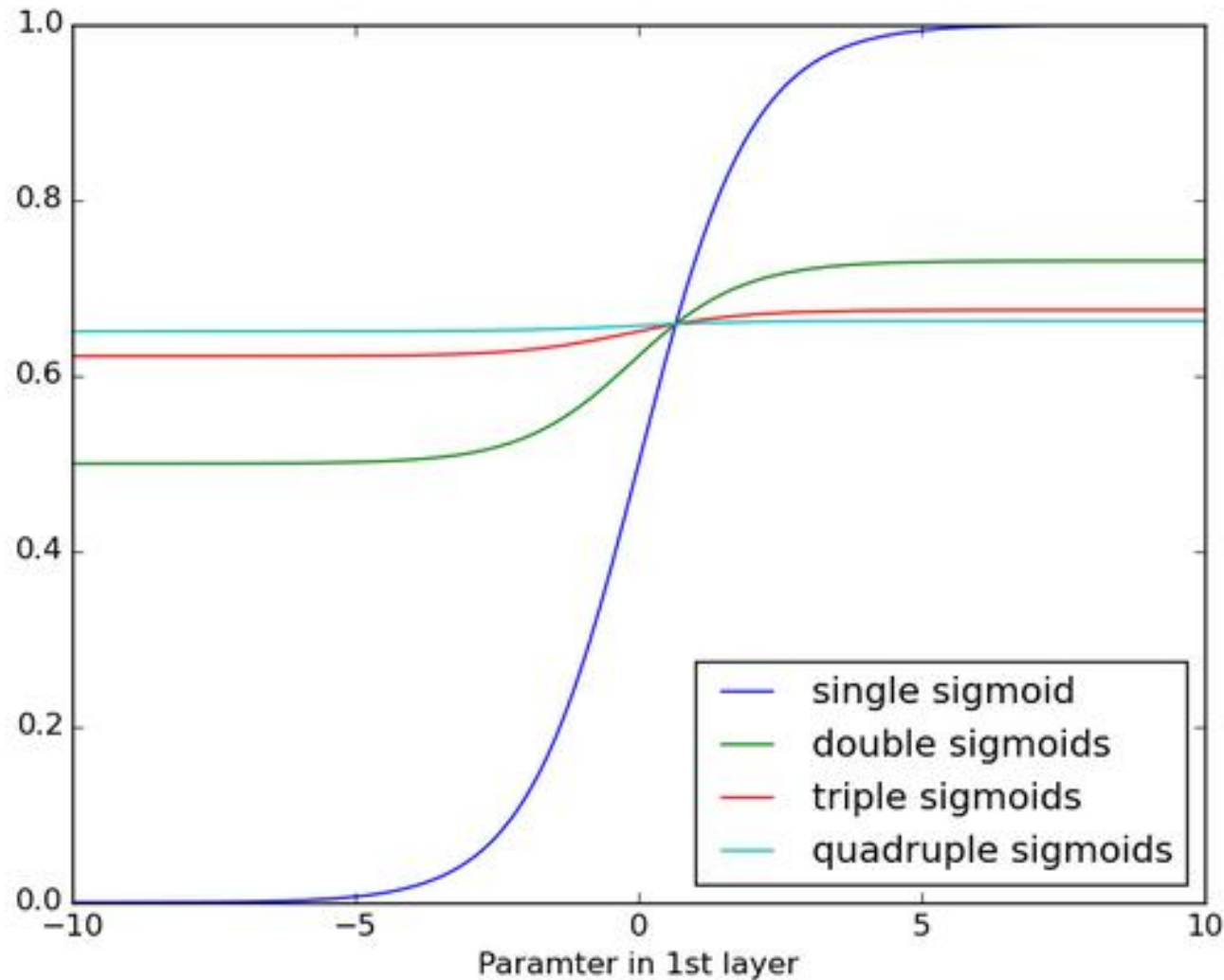


Back Propagation Through Time

$$\frac{\partial E_2}{\partial W} = \sum_{k=0}^2 \frac{\partial E_2}{\partial s_2} \left(\prod_{j=k+1}^2 \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial a_{hk}} \frac{\partial a_{hk}}{\partial W}$$

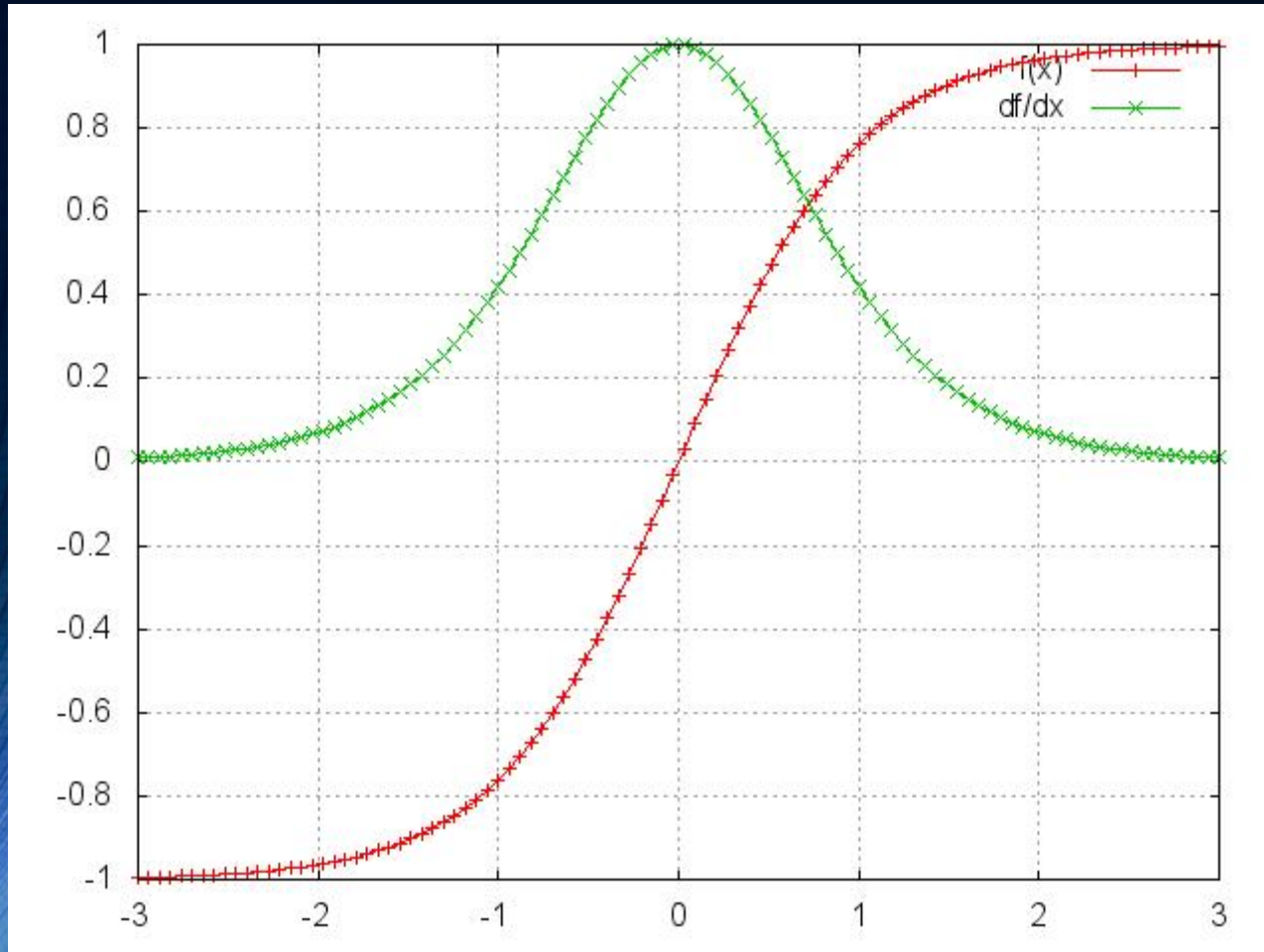


Vanishing Gradients



- RNNs trained with BPTT have difficulties learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem.
- You see the effects of applying a sigmoid function over and over again. The data is flattened until, for large stretches, it has no detectable slope. This is analogous to a gradient vanishing as it passes through many layers.
- Exploding gradients treat every weight as though it were the proverbial butterfly whose flapping wings cause a distant hurricane. Those weights' gradients become saturated on the high end; i.e. they are presumed to be too powerful. But exploding gradients can be solved relatively easily, because they can be truncated or squashed. Vanishing gradients can become too small for computers to work with or for networks to learn – a harder problem to solve.

Vanishing Gradients



- You can see that the sigmoid functions have derivatives of 0 at both ends. They approach a flat line.
- When this happens we say the corresponding neurons are saturated. They have a zero gradient and drive other gradients in previous layers towards 0.
- Thus, with small values in the matrix and multiple matrix multiplications (t-k in particular) the gradient values are shrinking exponentially fast, eventually vanishing completely after a few time steps.
- Gradient contributions from “far away” steps become zero, and the state at those steps doesn’t contribute to what you are learning: You end up not learning long-range dependencies.
- Sigmoid function becomes very flat when sigma is 0 or 1 (for tanh -1 or 1)

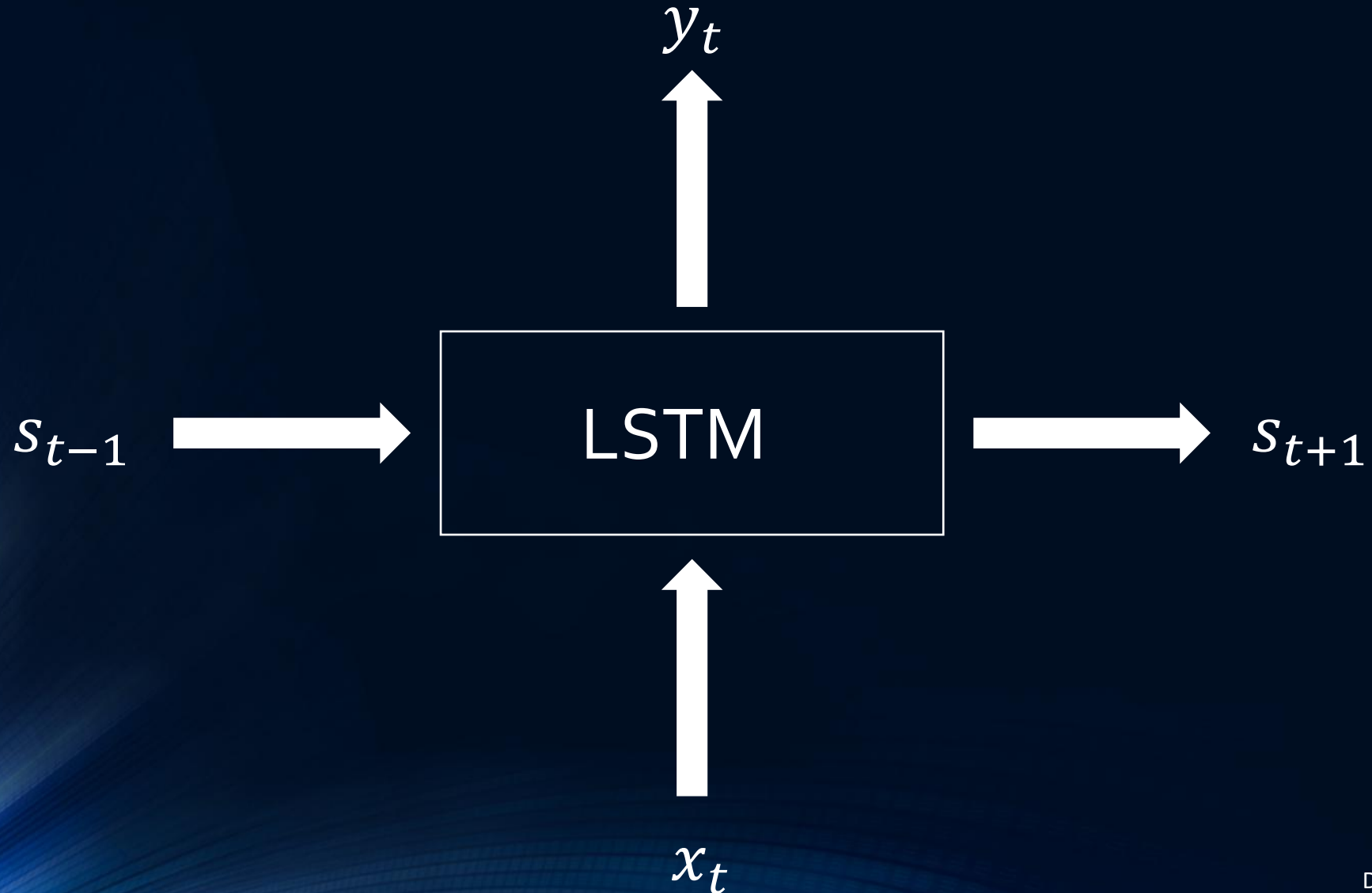
RoadMap

- Introduction
- Motivation
- RNN architecture
- Long Short Term Memory (LSTM) Network
- LSTM experiments
- Conclusions

LSTM - introduction

- LSTM was invented to solve the vanishing gradients problem.
- LSTM maintain a more constant error flow in the backpropagation process.
- LSTM can learn over more than 1000 time steps , and thus can handle large sequences that are linked remotely.

LSTM – same idea as RNN



LSTM Architecture

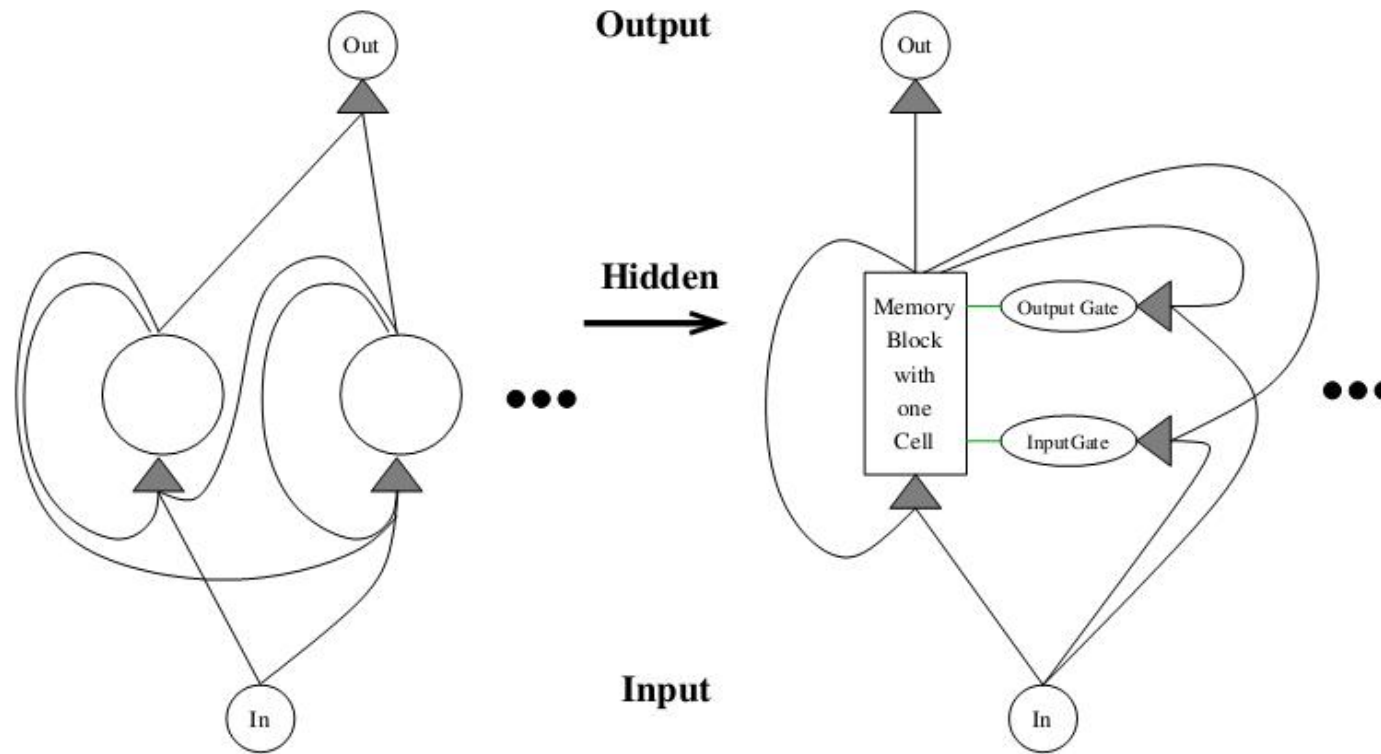
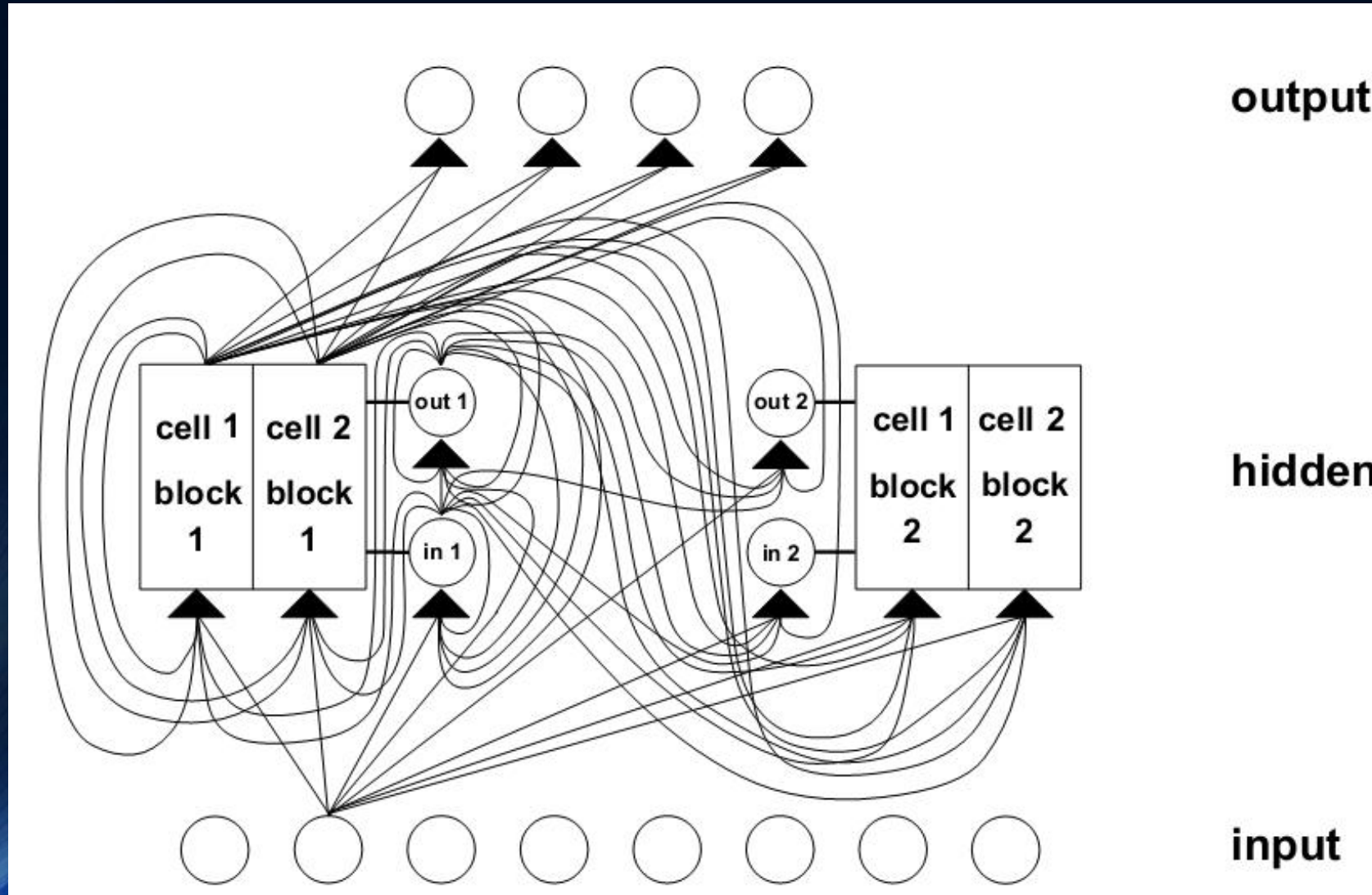


Figure 2.1: Left: RNN with one fully recurrent hidden layer. Right: LSTM network with memory blocks in the hidden layer (only one is shown).

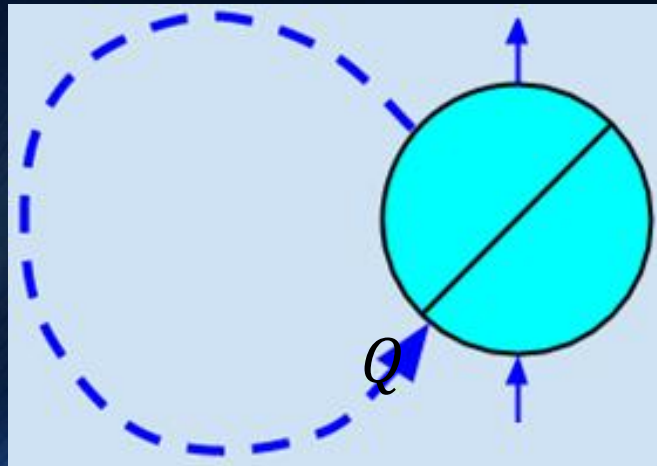
- The basic unit in the hidden layer of an LSTM network is a memory block, it replaces the hidden unit in a traditional RNN.
- A memory block contains one or more memory cell and a pair of adaptive multiplicative gating units which gates input and output to all cells in the block.
- Memory blocks allow cells to share the same gates thus reducing the number of parameters.
- Each cell has in its core a recurrently self connected linear unit called the “Constant error carousel” whose activation we call the cell state.

LSTM Architecture



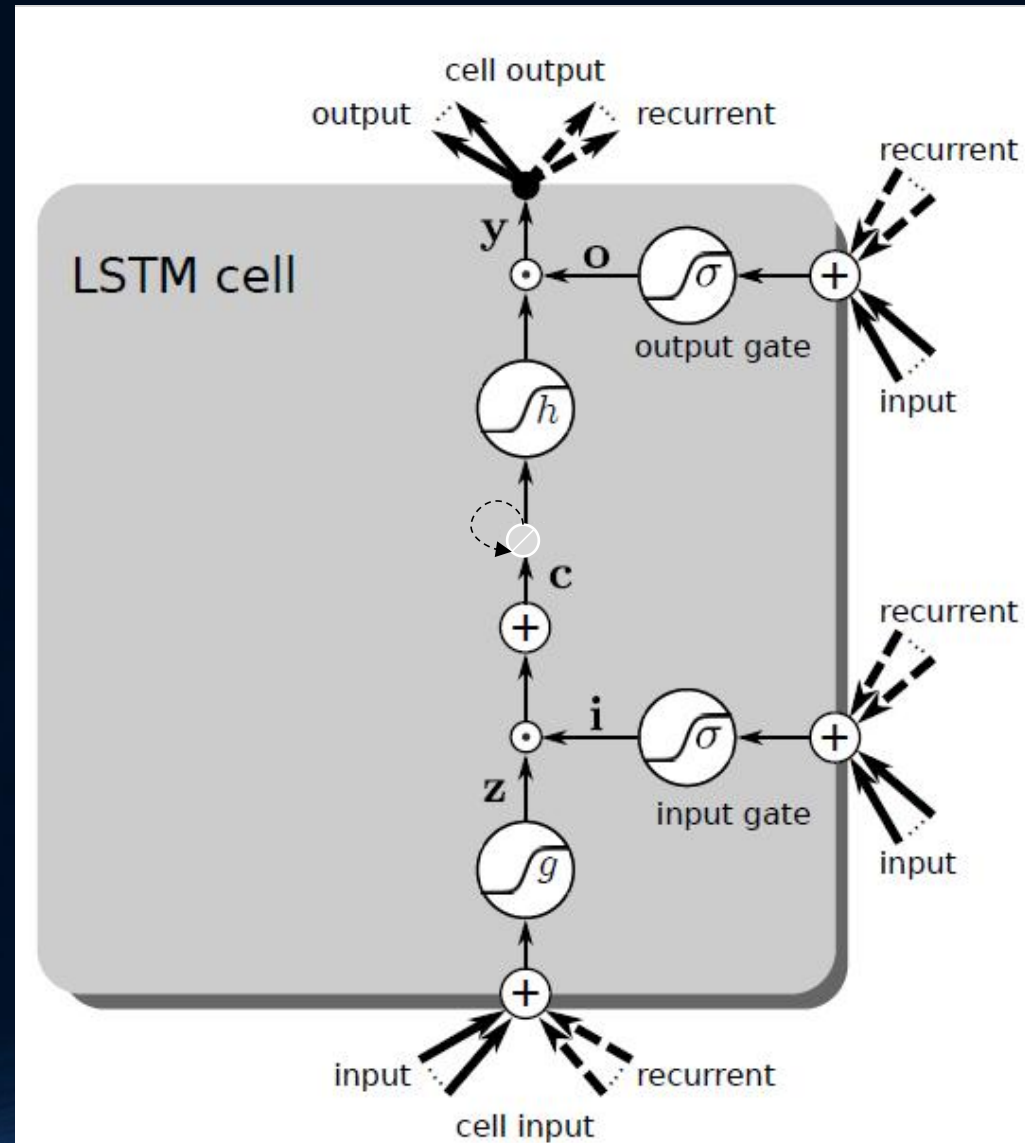
LSTM Architecture

Each memory cell contains a node with a self-connected recurrent edge of fixed weight one, ensuring that the gradient can pass across many time steps without Vanishing – which is called CEC (constant error carousel)

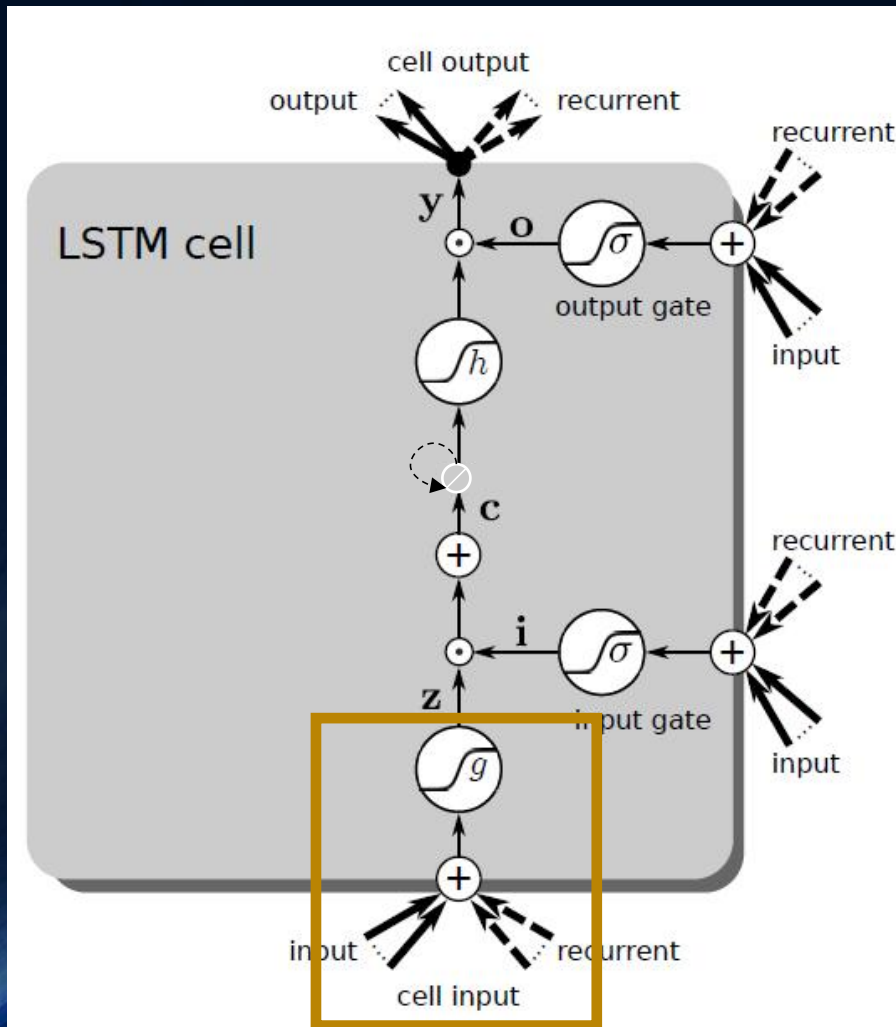


$$\begin{aligned}\delta_c(t-1) &= \delta_c(t) \rightarrow \\ f'_c(Qc(t-1)) \odot Q &= 1 \rightarrow \\ f_c(Qc(t-1)) &= \frac{Qc(t-1)}{Q} \\ c(t) &= f_c(Qc(t-1)) = c(t-1) \rightarrow \\ Q &= 1, f_c(x) = x\end{aligned}$$

LSTM Architecture - overview



LSTM Architecture – input

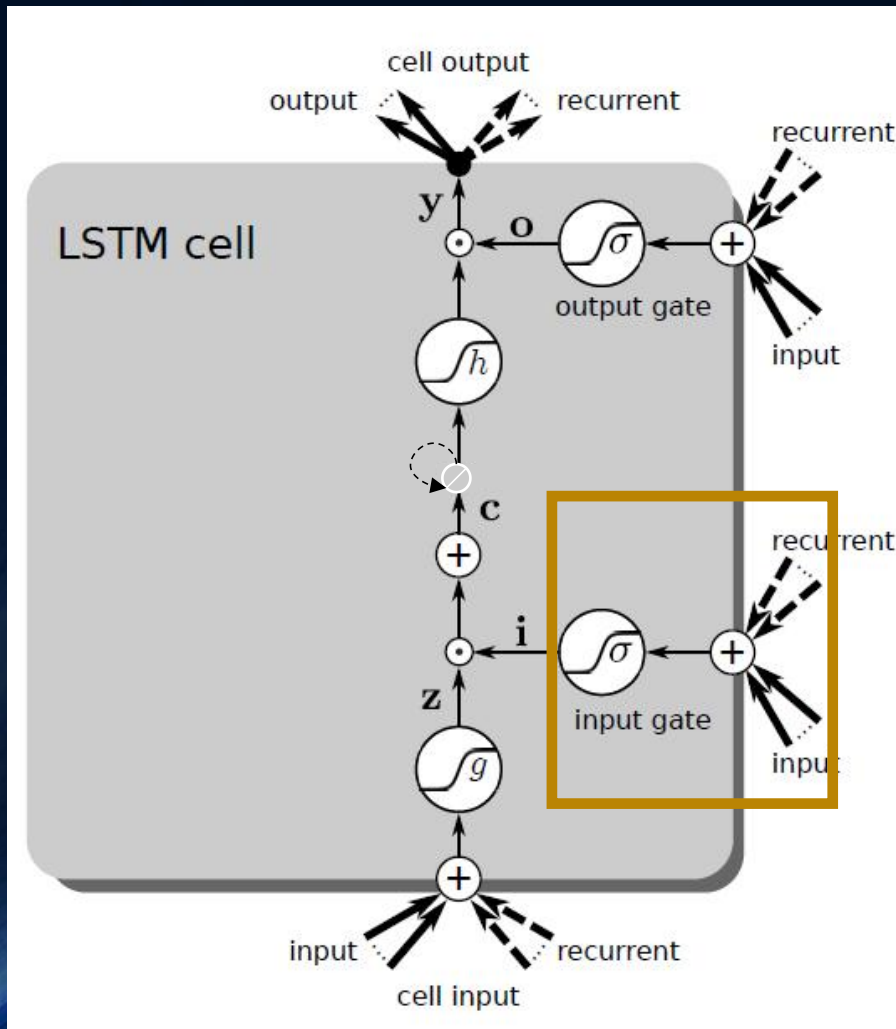


$$a_z(t) = W_z x(t) + R_z y(t - 1)$$
$$z(t) = g(a_z(t))$$

Note:

- The layer decides which information from the input should be forwarded into the LSTM cell.
- Respectively by multiplying y_{t-1} with the recurrent weight matrix R_z the layer decides which information from the previous time step should be forwarded into the LSTM cell

LSTM Architecture – input gate

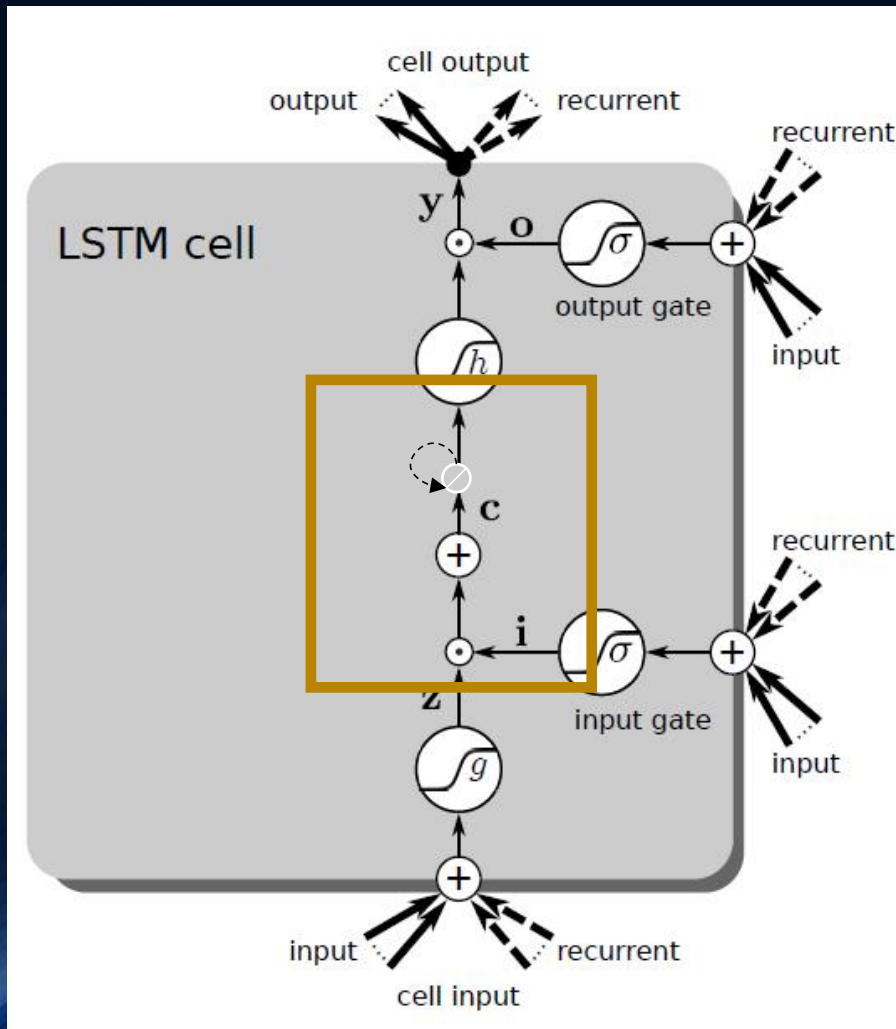


$$a_{in}(t) = W_{in}x(t) + R_{in}y(t - 1)$$
$$i(t) = \sigma(a_{in}(t))$$

Note:

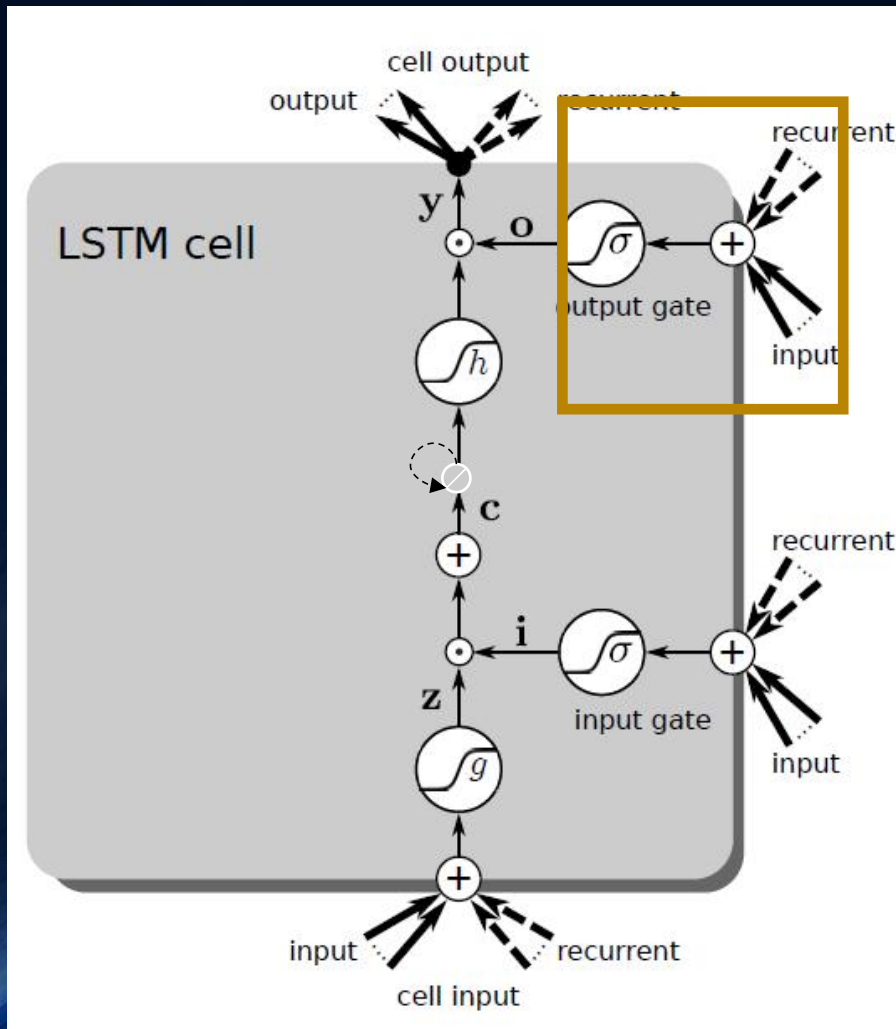
- The input gate controls write accesses to memory cells.
- This is realized by using the result of the squashing function σ as a factor which will be later multiplied by the squashed cell input z .
- The range of function σ is $(0,1)$ and can be interpreted in case of 0 as write access denied and in case of 1 as write access granted.
- Note that all values in between are also possible

LSTM Architecture – CEC



$$c(t) = z(t) \odot i(t) + c(t - 1)$$

LSTM Architecture – Output gate

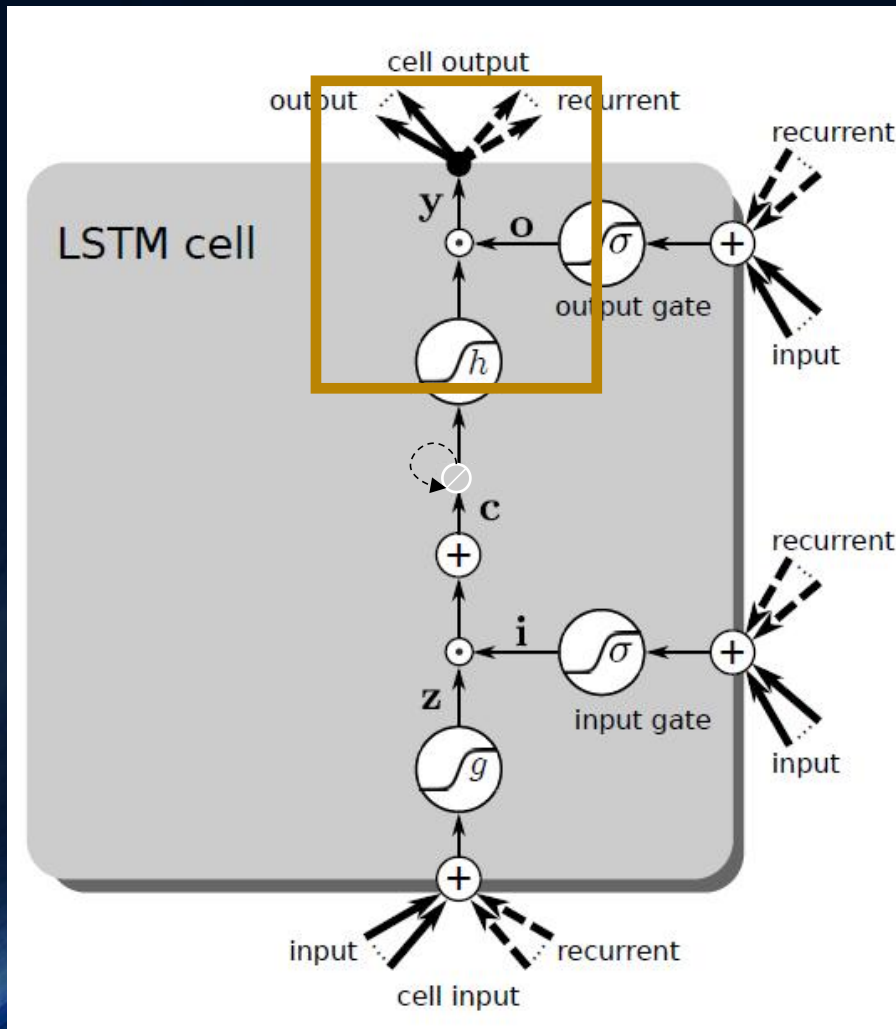


$$a_{out}(t) = W_{out}x(t) + R_{out}y(t - 1)$$
$$o(t) = \sigma(a_{out}(t))$$

Note:

- The output gate controls read accesses from memory cells.
- This is realized by using the result of the squashing function σ as a factor which will be later multiplied by the squashed cell content $h(c\ t)$.
- The range of function σ is $(0,1)$ and can be interpreted in case of 0 as read access denied and in case of 1 as read access granted.
- Note that all values in between are also possible.

LSTM Architecture – Output gate



$$y(t) = h(c(t)) \odot o(t)$$

Note:

- The value stored in the cell c is squashed by function h .
- Whether this information gets output is decided by the output gate via o .
- This is done by point wise multiplication of both vectors.

LSTM Forward Pass

- The cell state c is updated based on its current state and 3 inputs: a_z , a_{in} , a_{out}

$$a_z(t) = W_z x(t) + R_z(y(t-1)), z(t) = g(a_z(t))$$

$$a_{in}(t) = W_{in} x(t) + R_{in}(y(t-1)), i(t) = \sigma(a_{in}(t))$$

$$c(t) = z(t) \odot i(t) + c(t-1)$$

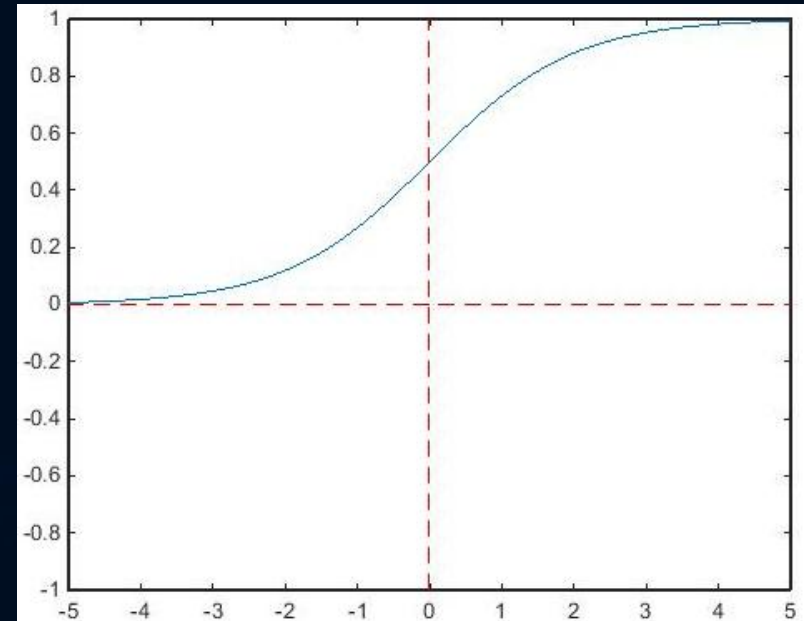
$$a_{out}(t) = W_{out} x(t) + R_{out}(y(t-1)), o(t) = \sigma(a_{out}(t))$$

$$y(t) = h(c(t)) \odot o(t)$$

LSTM – activation functions

σ is the activation function of both gates:

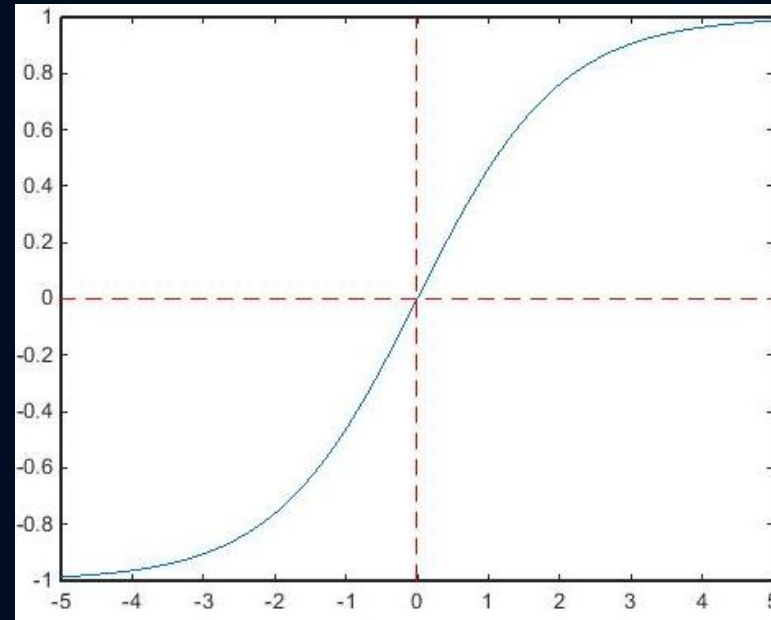
$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$



LSTM – activation functions

g is the activation function of the input:

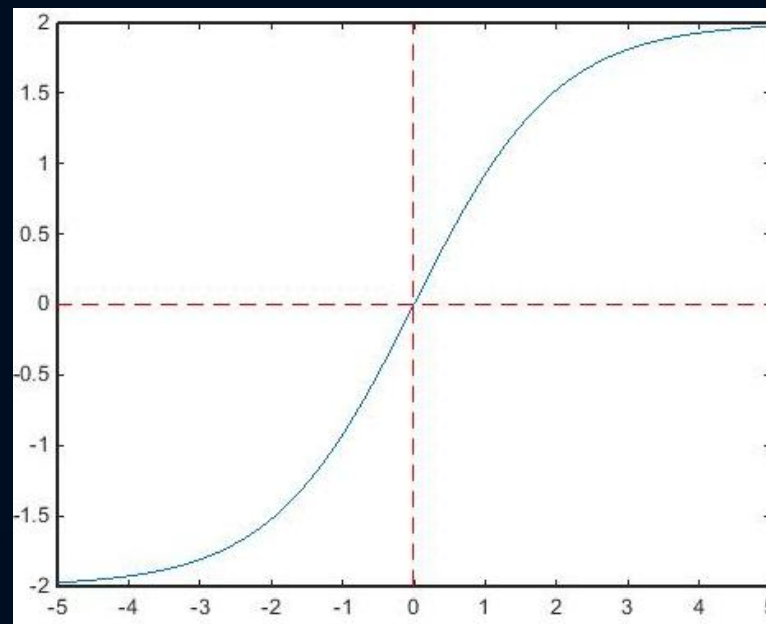
$$g(x) = \frac{2}{1 + \exp(-x)} - 1$$



LSTM – activation functions

h is the activation function of the output:

$$h(x) = \frac{4}{1 + \exp(-x)} - 2$$



LSTM Backward Pass

- Errors arriving at cell outputs are propagated to the CEC
- Errors can stay for a long time inside the CEC
- This ensures non-decaying error
- Can bridge time lags between input events and target signals

LSTM Backward Pass

$$\delta_y(t) = \underbrace{\frac{\partial E_t}{\partial y(t)}}_{\text{Error from the current output}} + \underbrace{R_z^T \delta_z(t+1) + R_i^T \delta_i(t+1) + R_o^T \delta_o(t+1)}_{\text{Error from the next cell output}}$$

Error from the current output

Error from the next cell output

LSTM Backward Pass

○ In forward pass $y(t) = h(c(t)) \odot o(t)$

■ Given $\delta_y(t) = \frac{\partial E}{\partial a_y(t)} \rightarrow$ find $\delta_c(t) = \frac{\partial E}{\partial a_c(t)}$, $\delta_o(t) = \frac{\partial E}{\partial a_o(t)}$

$$\frac{\partial E}{\partial a_o(t)} = \frac{\partial E}{\partial y(t)} \frac{\partial y(t)}{\partial o(t)} \frac{\partial o(t)}{\partial a_o(t)} = \delta_y(t) \odot h(c(t)) \odot \sigma'(a_o(t))$$

$$\frac{\partial E}{\partial c(t)} = \frac{\partial E}{\partial y(t)} \frac{\partial y(t)}{\partial c(t)} = \delta_y(t) h'(c(t)) \odot o(t)$$

We also need to add the gradient from the next step

$$\delta_c(t) = \delta_y(t) h'(c(t)) \odot o(t) + \delta_c(t+1)$$

LSTM Backward Pass

- In forward pass $c(t) = z(t) \odot i(t) + c(t - 1)$
- Given $\delta_c(t) = \frac{\partial E}{\partial c(t)} \rightarrow$ find $\delta_i(t) = \frac{\partial E}{\partial a_i(t)}$, $\delta_z(t) = \frac{\partial E}{\partial a_z(t)}$

$$\frac{\partial E}{\partial a_i(t)} = \frac{\partial E}{\partial c(t)} \frac{\partial c(t)}{\partial i(t)} \frac{\partial i(t)}{\partial a_i(t)} = \delta_c(t) \odot z(t) \odot \sigma'(a_i(t))$$

$$\frac{\partial E}{\partial a_z(t)} = \frac{\partial E}{\partial c(t)} \frac{\partial c(t)}{\partial z(t)} \frac{\partial z(t)}{\partial a_z(t)} = \delta_c(t) \odot z(t) \odot g'(a_z(t))$$

$$\delta_y(t - 1) = \frac{\partial E_{t-1}}{\partial y(t - 1)} + R_z^T \delta_z(t) + R_i^T \delta_i(t) + R_o^T \delta_o(t)$$

LSTM Backward Pass

$$\Delta W_z = \sum_t^T \delta_z(t) x(t)^T$$

$$\Delta W_i = \sum_t^T \delta_i(t) x(t)^T$$

$$\Delta W_o = \sum_t^T \delta_o(t) x(t)^T$$

$$\Delta R_z = \sum_t^T \delta_z(t+1) y(t)^T$$

$$\Delta R_i = \sum_t^T \delta_i(t+1) y(t)^T$$

$$\Delta R_o = \sum_t^T \delta_o(t+1) y(t)^T$$

RoadMap

- Introduction
- Motivation
- RNN architecture
- Long Short Term Memory (LSTM) Network
- **LSTM experiments**
- Conclusions

LSTM Experiments

- All experiments involve long minimal time lags
- Complex tasks that cannot be solved by simple strategies such as weight guessing.
- Comparison to other RNNs
- Weight initialization $\rightarrow [-0.2, 0.2], [-0.1, 0.1]$

LSTM Experiments – predict next symbol

Prediction of the next symbol in a sequence of symbols

method	hidden units	# weights	learning rate	% of success	success after
RTRL	3	≈ 170	0.05	“some fraction”	173,000
RTRL	12	≈ 494	0.1	“some fraction”	25,000
ELM	15	≈ 435		0	>200,000
RCC	7-9	$\approx 119-198$		50	182,000
LSTM	4 blocks, size 1	264	0.1	100	39,740
LSTM	3 blocks, size 2	276	0.1	100	21,730
LSTM	3 blocks, size 2	276	0.2	97	14,060
LSTM	4 blocks, size 1	264	0.5	97	9,500
LSTM	3 blocks, size 2	276	0.5	100	8,440

- Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." *Neural computation* 9.8 (1997): 1735-1780.

Note: The nets task is to read strings one symbol at a time and to permanently predict the next symbol.
LSTM almost always learns to solve the task, learns much faster.

LSTM Experiments

T	minimal lag	# weights	# wrong predictions	Success after
100	50	93	1 out of 2560	74,000
500	250	93	0 out of 2560	209,000
1000	500	93	1 out of 2560	853,000

- Each element of each input sequence is a pair of components
- First component is a real value randomly chosen between $[-1,1]$ the second is either 1,0,-1.
- The target is $0.5 + \frac{X_1 + X_2}{4.0}$ when X_1, X_2 are the first two components with $X_2 = 1$

LSTM Experiments - classification

- Sequence of letters –
 $\{a,b,\dots\{X|Y\},\dots a,b\dots\{X|Y\}\dots a,b\dots\{X|Y\}\dots\}$
- Goal is to find the class of the sequence
- For Example : YXX,XXY,XYX... 2^3 classes
- Length of the sequence 100 letters

task	# weights	# wrong predictions	Success after
Task 6a	156	1 out of 2560	31,390
Task 6b	308	2 out of 2560	571,100

Note: The goal is to classify sequences. LSTM is able to extract information conveyed by the temporal order of widely separated inputs.

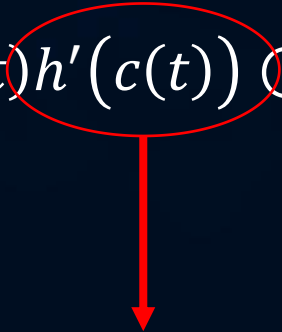
Advantages of LSTM

- Non-decaying error backpropagation.
- For long time lag problems, LSTM can handle noise and continuous values.
- No parameter fine tuning.
- Memory for long time periods

Limitations of LSTM

$$c(t) = z(t) \odot i(t) + c(t-1) \rightarrow \text{grows linearly}$$

For a continues input stream \rightarrow
 $c(t)$ may grow in an unbounded fashion \rightarrow
can cause a saturation in $h(t)$

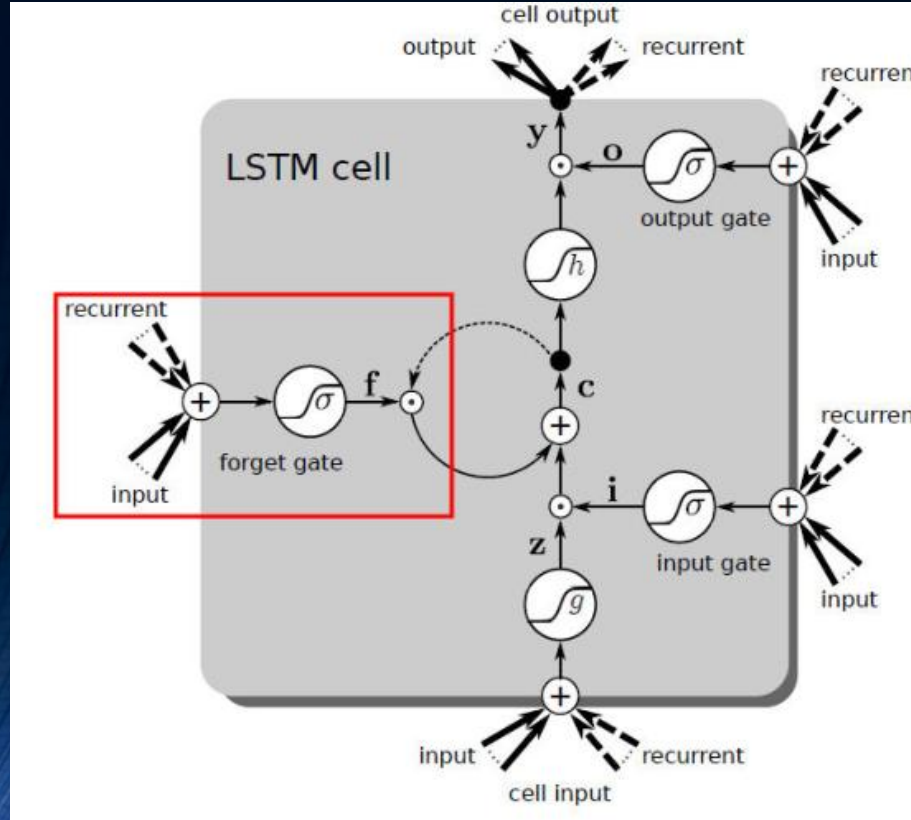
$$\delta_c(t) = \delta_y(t) h'(c(t)) \odot o(t)$$


Small gradients

Note:

- LSTM allows information to be stored across arbitrary time lags and error signals to be carried far back in time.
- This potential strength however, can contribute to a weakness: the cells states S_c often tend to grow linearly during the presentation of a time series.
- If we present a continues input stream the cell state may grow in unbounded fashion causing saturation of the output squashing function h . saturation will make h 's derivative vanish thus blocking incoming errors, and make the cell output equal the output gate activation - the cell will degenerate to an ordinary BPTT so the cell will cease functioning as a memory.

LSTM possible remedy



Note:

- LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory.
- The cell makes decisions about what to store, and when to allow reads, writes and erasures, via gates that open and close. Unlike the digital storage on computers, however, these gates are analog, implemented with element-wise multiplication by sigmoids, which are all in the range of 0-1. Analog has the advantage over digital of being differentiable, and therefore suitable for backpropagation
- Those gates act on the signals they receive, and similar to the neural network's nodes, they block or pass on information based on its strength and import, which they filter with their own sets of weights.
- Those weights, like the weights that modulate input and hidden states, are adjusted via the recurrent networks learning process.
- That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error, and adjusting weights via gradient descent

$$f(t) = \sigma(W_f x(t) + R_f(y(t-1)))$$

$$c(t) = z(t) \odot i(t) + f(t) \odot c(t-1)$$

RoadMap

- Introduction
- Motivation
- RNN architecture
- Long Short Term Memory (LSTM) Network
- LTSM experiments
- Conclusions

LSTM conclusions

- RNNs - self connected networks
- Vanishing gradients and long memory problems
- LSTM - solves the vanishing gradient and the long memory limitation problem
- LSTM can learn sequences with more than 1000 time steps.