Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Objectives for today:
- Why are sequences important?
- Long-term dependencies in sequence data
- Sequence processing with feedforward models
- Sequence processing with recurrent models
- Vanishing Gradient Problem
- Long-Short-Term Memory (LSTM)
- Application: Music generation

Reading for this lecture:

Goodfellow et al.,2016 *Deep Learning*

- **Ch. 10** (except 10.6 and 10.8)

Further Reading for this Lecture:

Paper:
- F.A. Gers and J. Schmidhuber and F. Cummins (2000) *Learning to Forget: Continual Prediction with LSTM Neural Computation, 12, 2451–2471*
- Xu et al. (2015), *Show, attend and tell: Neural image caption generation…, ICML*

review: Artificial Neural Networks for classification

Given: Training data set

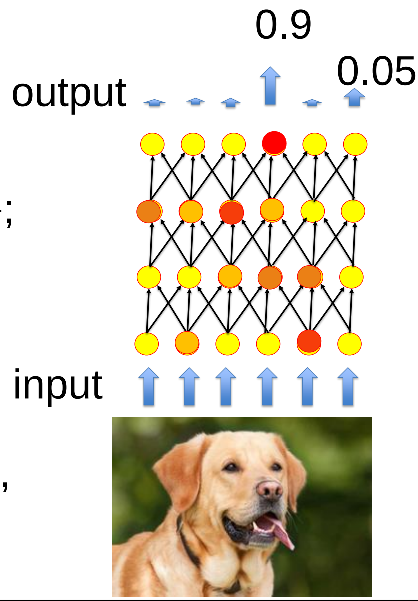
$\{ \ (\boldsymbol{x}^\mu, \boldsymbol{t}^\mu) \ , \quad 1 \leq \mu \leq P \ \};$

Aim of learning:

Adjust connections such that output $\boldsymbol{y}^\mu$ is correct

$\boldsymbol{y}^\mu = \boldsymbol{t}^\mu$

(for each static input image, $\boldsymbol{x}^\mu$)



Previous slide.

So far we considered classification using supervised learning. An input pattern was present and the network output was compared with a target class label.

review: Artificial Neural Networks for classification

Given: Training data set $\{ \quad (\boldsymbol{x}^{\mu}, t^{\mu}) \quad , \qquad 1 \leq \mu \leq P \quad \};$

Question:
is this really the most frequent situation in practice ?

No, for several reasons:
- difficult to get the labeled data!
- data is rarely static!

Previous slide.

But in practice, it is rare that we have such data because (i) most data is unlabeled and (ii) most data is dynamic rather than static.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences

Previous slide.

Let us look as sequences as an example of 'non-static' data.

1. Sequences: first example = video sequence

You have seen the past *n* frames, what is the next frame?

‘video frame prediction’

Screenshot from ‘Casablanca’

Previous slide.

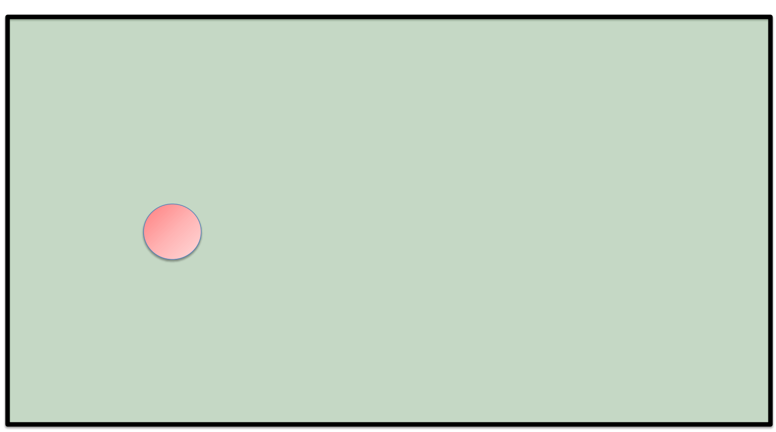
The task of video frame prediction is to predict the next frame of a video sequence, given earlier images of the same sequence.

1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

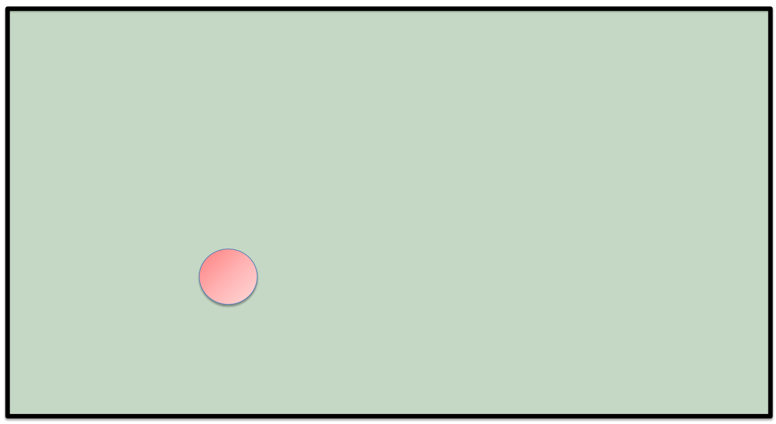
1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



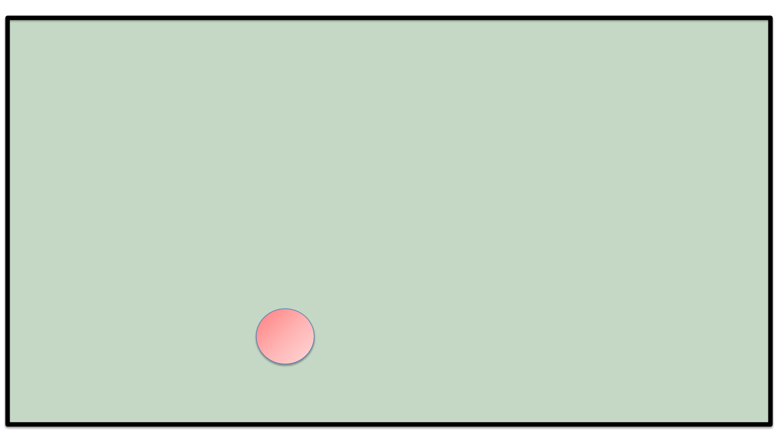
1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



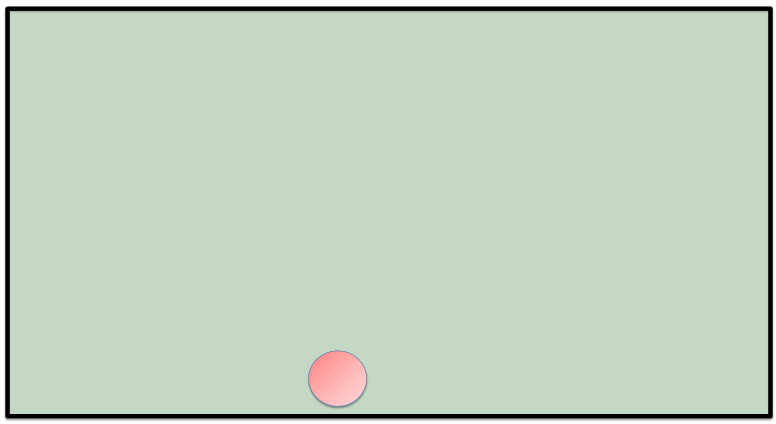
1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



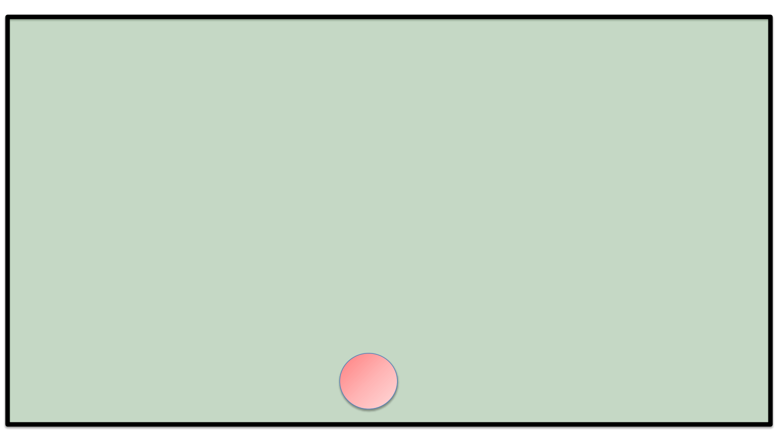
1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



1. Video Sequences/Video Frame Prediction

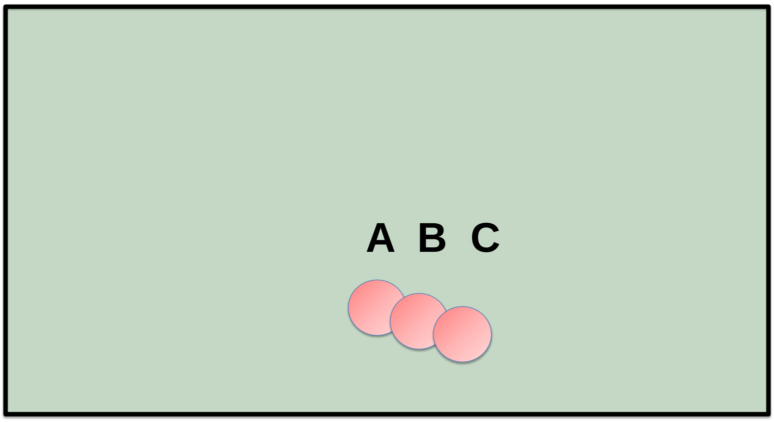
Bouncing Billiard Ball



1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

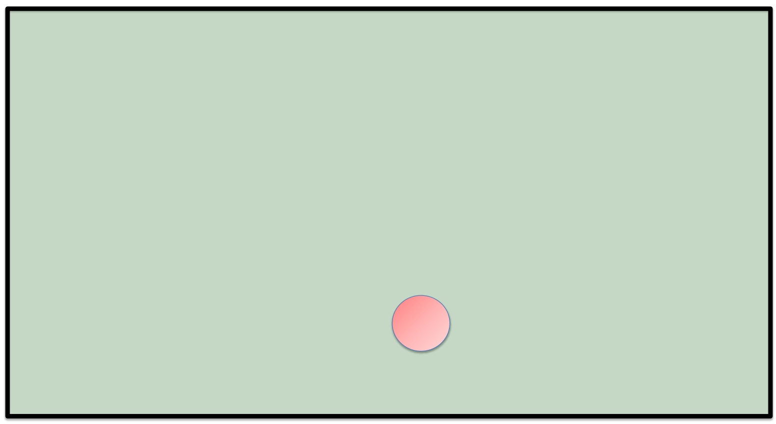
Predict position in next frame



Previous slide.
Given the 5 earlier images, what is the position of the billiard ball in the next frame?

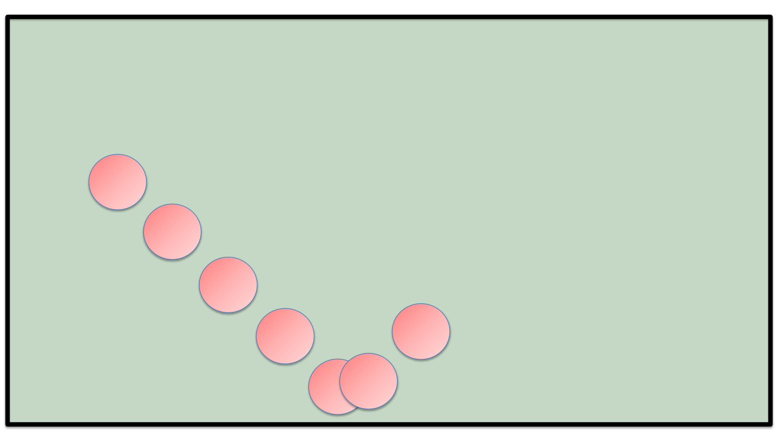
1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



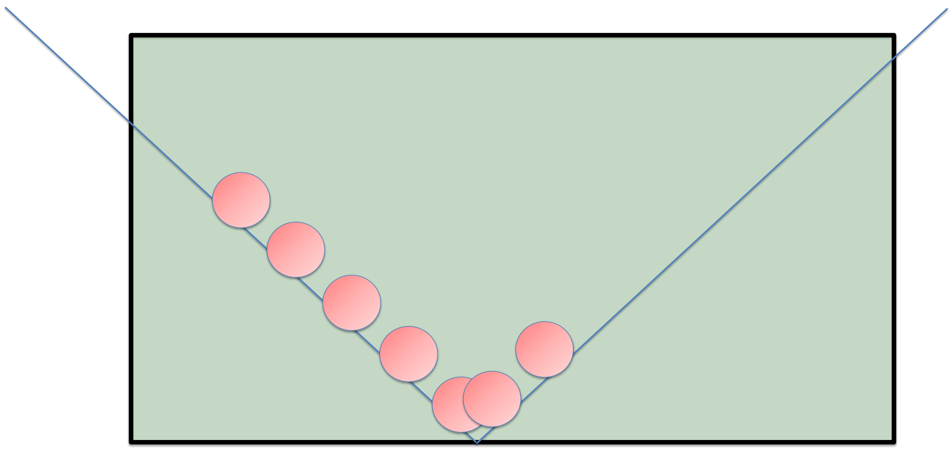
1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



1. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



Previous slide.
A good model of bouncing billiard balls will have understood that balls move on straight lines unless they touch the border or another ball.

1. Sequences: video frame prediction

1st example: video frame prediction

- Target of training is the next frame
  → lots of training data!!!

- Data consists of a temporal sequence,
  prediction needs more than 1 frame in the past
  → not the standard static input scenario

- Output is high-dimensional (pixels in one frame)

Previous slide.

Video frame prediction is interesting because

(i) There is lots of training data. The data is not a class label but simply the next image.

(ii) Data consists of a temporal sequences. Prediction requires to ‘understand’ the temporal structure.

(iii) The output is high-dimension: not 10 or 20 different classes, but THOUSANDS of real-valued PIXELs!

1. Sequences

- 1st example: video frame prediction

  Analogous: - move your arm while watching
  - observe movements of your neighbor and predict next move

- 2nd example: text prediction

Previous slide.

Try the following. You move your arm in various directions while watching your own arm. Looks easy to predict the movement.

Now try to predict the movements of your neighbor's arm.

1. Sequences: 2nd example - text prediction

Similar to Caltech, MIT, and GeorgiaTech which are considered top-level technical universities in the US, TUMunich, ETHZurich and …

Previous slide.

Sometimes text prediction looks easy, sometimes not.

1. Sequences: text prediction

2nd example: Text prediction

- Target of training is the next word
  → lots of training data!!!
- Data consists of a temporal sequence,
  prediction needs more than 1 word in the past
  → not the standard static input-output scenario
- Output is high-dimensional
  (ten-thousands of potential words)

Previous slide.

Same comments apply to text prediction as to video frame prediction.

1. Sequences

- 1st example: video frame prediction

- 2nd example: text prediction

  analogous: - text translation
  - speech (or phoneme) prediction
  - music prediction

- 3rd example: action planning

Previous slide.

Let us now turn to the third example.

1. Sequences: 3rd example – action planning and navigation

- Close your eyes
- Imagine how you would go
  to the library in the ‘learning center’

Previous slide.
Your imagination

1. Sequences

Summary:

- Sequences are everywhere

films, text, speech, body movement, action planning, navigation

- more common in reality than static input-output paradigms

We don't look at static photos in normal live

- target data (needed for supervised learning) is often cheap

e.g., target is next frame in video / next word in text/
next action in movement:
– all easy to observe

Previous slide.
Conclusion: in the real world, sequences are abundant, convenient, and much more 'normal' than classification of static patterns.

1. Sequences: Aim

First Question for today

how can we model and learn sequences in artificial neural networks?

Previous slide.

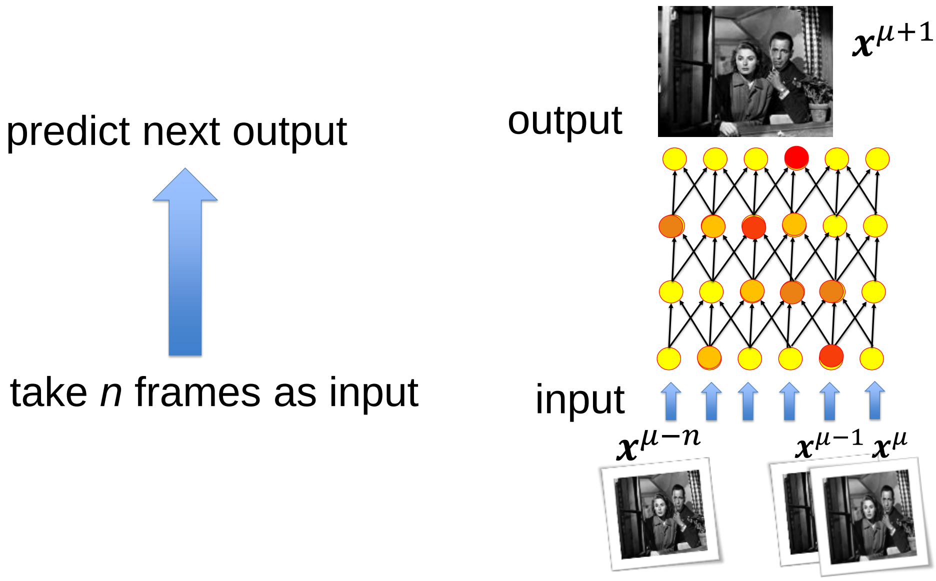
… and therefore we should develop neural networks that can deal with sequences.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve Neural Network implementation: increase number of inputs

Previous slide.
Let us first look at a naïve approach and treat sequences as high-dimensional input.

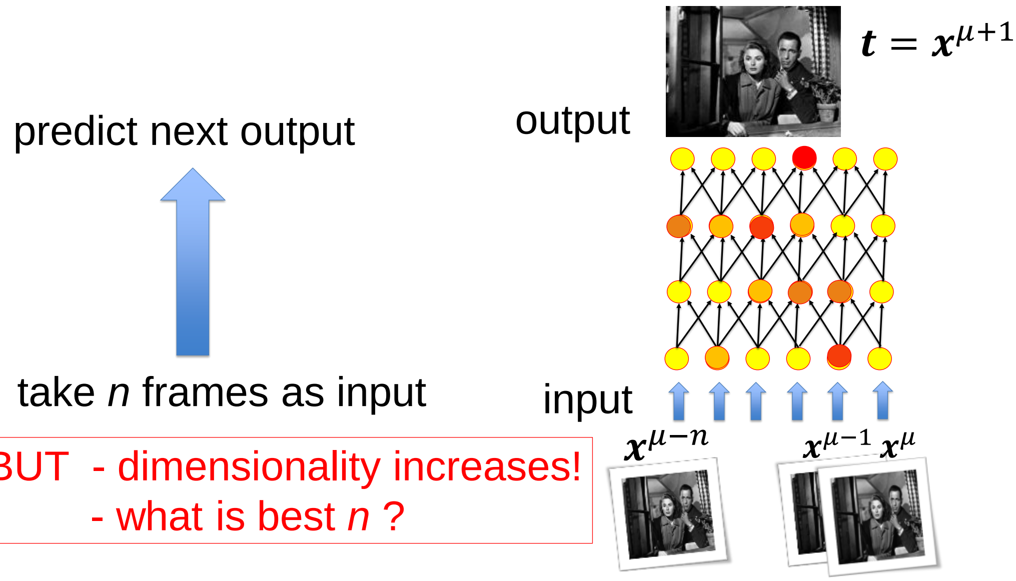
2. Naïve solution: increase number of inputs



Previous slide.

Instead of one image, we now take n frames of the video as input and train the network to predict the next image in the output. This way, the sequence problem has been transformed into a high-dimensional, but static problem.

2. Naïve solution: Problems



BUT - dimensionality increases!
- what is best *n* ?

Previous slide.

The two drawbacks are, first, that input dimension increases linearly with n; and second that we do not know the best n – but the input dimension is an important design parameter of the static network architecture.

2. Naïve solution: Problems

The naïve solution corresponds to implementing **n-grams** with a neural network, but

- dimensionality increases!
- what is best *n* ?

→ What is the relevant time scale?
(number of frames necessary for good prediction)

Previous slide.

The question of the best number of frames can be reformulated as a question of 'relevant time scale'.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve solution: increase number of inputs
3. Long-term Dependencies

Previous slide.

As we will see, the problem is that there is most often not a single time scale, but many potentially relevant time scales.

3. Dependencies in Video

Bouncing Billiard Ball

time = 1

Previous slide.

Let reconsider the bouncing billiard ball.
This time it can disappear behind a screen. Try to predict when it reappears.

3. Dependencies in Video

Bouncing Billiard Ball

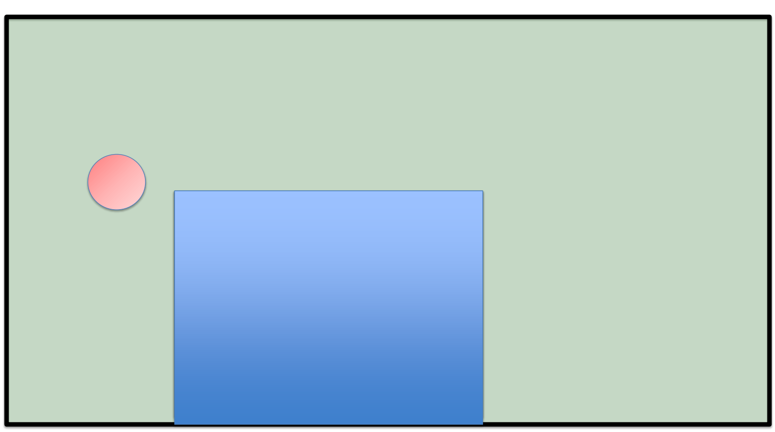
time = 1

Ball disappears behind blue screen.
Predict moment when ball **reappears**!

3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

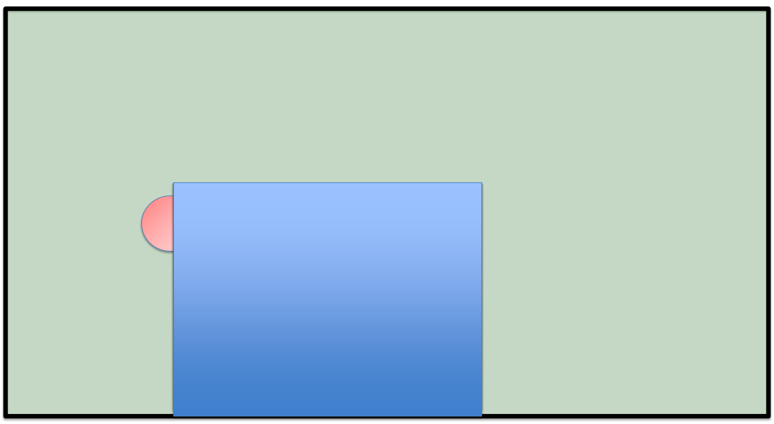
time = 1



3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

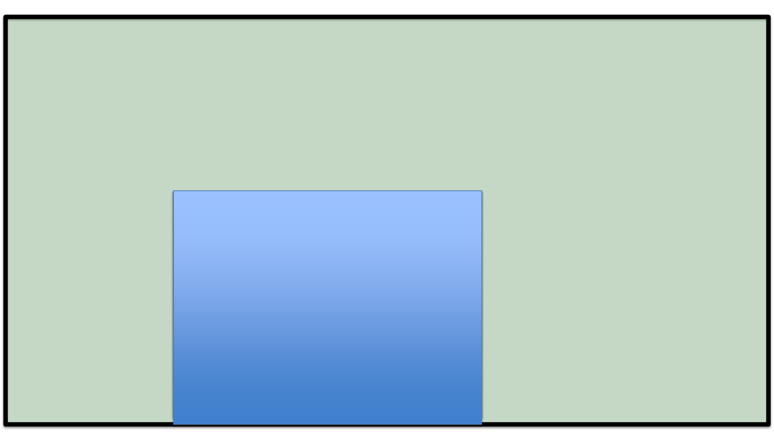
time = 2



3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

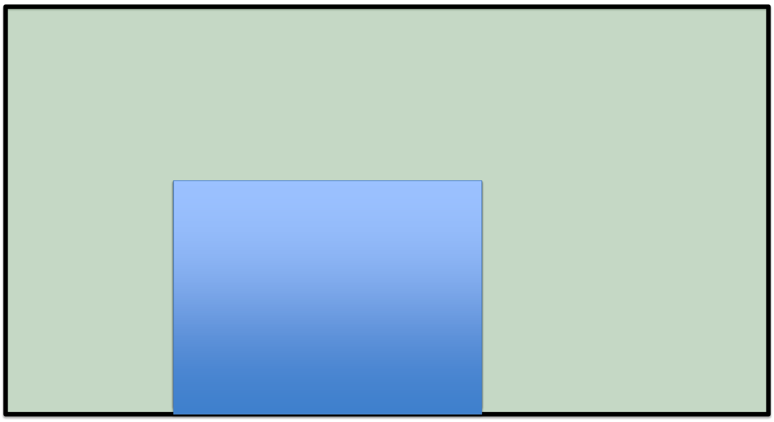
time = 3



3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

time = 4



3. Video Sequences/Video Frame Prediction

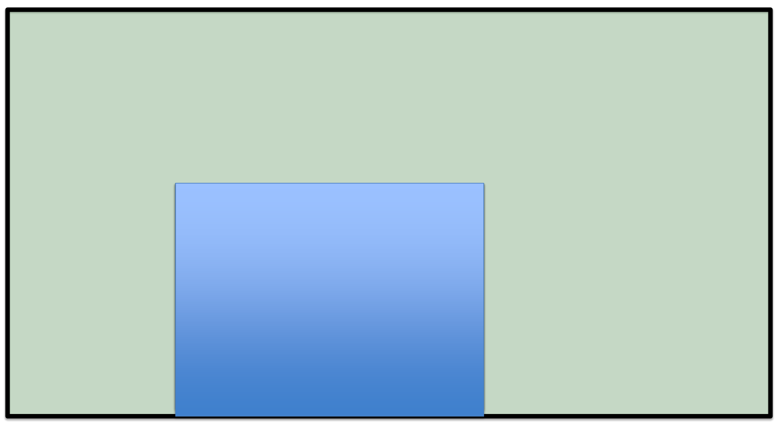
Bouncing Billiard Ball

time = 5

3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

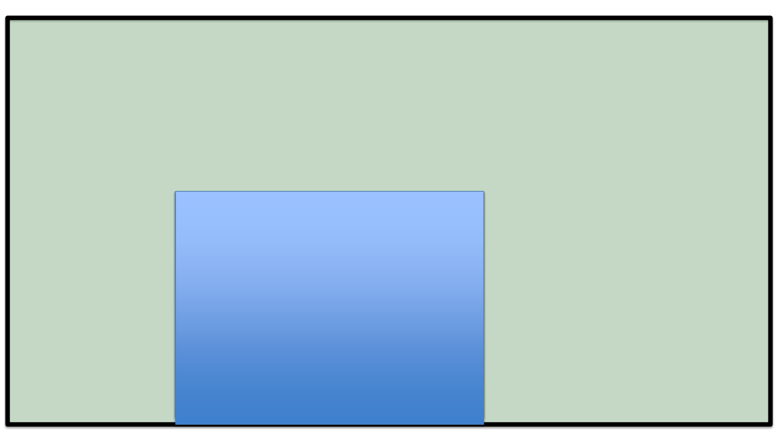
time = 6



3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

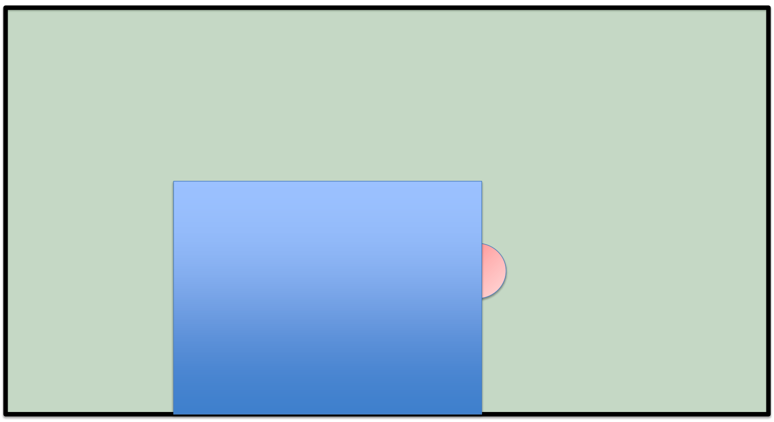
time = 7



3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball

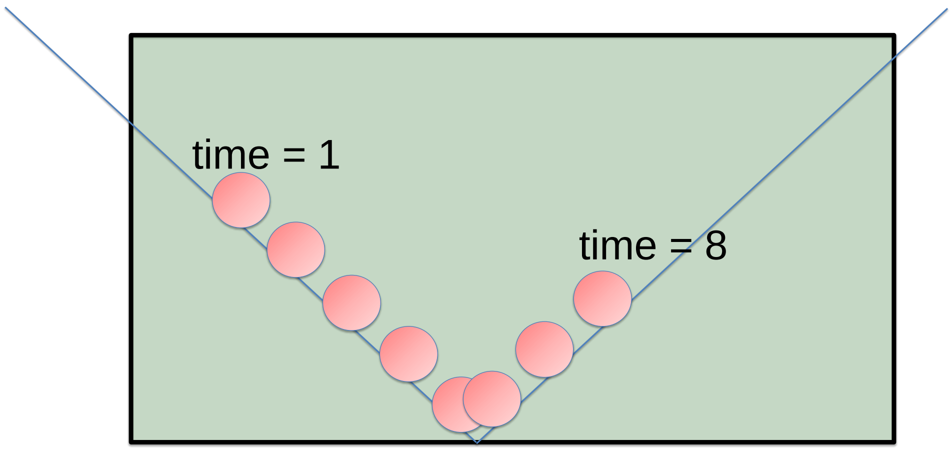
time = 8



Previous slide.
Did you succeed?

3. Video Sequences/Video Frame Prediction

Bouncing Billiard Ball



Previous slide.

Despite the fact that the movement is continuous and local (because it is given by Differential equation of physics, or, the discrete version by a Markov Process as often in Computer Science/Signal Processing), the screen transforms it into a problem with memory (a hidden Markov Process).
Worse, we don't know how long it remains behind the screen, therefore we do not know the length of the memory buffer that we should allocate.

3. Long-term dependencies in sequences

1st example: video frame prediction -
you potentially need a memory over MANY frames!

Extreme example:
- memory over a whole story, since entrance scene turns out to be important to predict the end
  → long time scale!!!
- but movements within one scene are on a fast time scale

Example: Actor with red shoes

→ You never know in advance how many frames you need
→ There might be several relevant time scales!

Previous slide.
Think of a movie where the actor goes out with red shoes, goes on a long walk, meets friends, gets into the rain, walks back home, and arrives with black shoes.
That would be a 'film mistake' that (some) people would notice.

3. Long-term dependencies in sequences

1st example: video frame prediction

2nd example: text prediction and text translation

Previous slide.

So far we have seen that video prediction can be hard.
Let us now turn to text prediction.

3. Long-term dependencies in text sequences

We are in 2013 and hear on the radio:

The international press writes that Mr. Obama who is starting today his second term as president of the United States is praised as one of the most influential world leaders.

We are in 2019 and remember:

In 2013 many international journals wrote that Mr. Obama who was then starting his second term as president of the United States was praised as one of the most influential world leaders.

Previous slide.

Past tense versus present tense.

3. Long-term dependencies in text sequences

Grammar rules create long-term dependencies

The international press writes that Mr. Obama who is starting today his second term as president of the United States is praised by the World Economic Forum as one of the most influential world leaders.

In 2013 many international journals wrote that Mr. Obama who was then starting his second term as president of the United States was praised by World Economic forum as one of the most influential world leaders.

Previous slide.

Because the second sentence is in past tense a few words (red) change.

3. Long-term dependencies in text sequences

Grammar rules create long-term dependencies
→ important for text translation

Previous slide.

Grammar rules induce long-term dependencies that can span more than 10 words. In other languages (such as German), the span can go over a full paragraph.

3. Long-term dependencies in text sequences

Ambiguities:

Tank as army vehicle
Tank as liquid container

Question: how can we disambiguate?

Previous slide.

Similarly, ambiguities of word meanings can only be dissolved in a given context.

What would be a good key word to disambiguate the context?

3. Long-term dependencies in text sequences

army vehicle / liquid container

There are a hundred liter of water in the tank.

Previous slide.

An example.

3. Long-term dependencies in text sequences

Grammar rules create long-term dependencies
→ important for text translation

Context resolves ambiguities
→ creates long-term dependencies
→ important for text translation

Previous slide.

These kind of long-term dependencies influence meaning. The challenges become obvious if you attempt to do 'translate' while somebody is speaking, a challenge that a simultaneous interpreter has to take up during international conferences in Geneva.

3. Long-term dependencies in text sequences

Depuis le mois de mars le nombre de vols à
l'aèroport de Genève a augmenté par 20 pourcent.

Previous slide.
Similar problems also arise in French.

3. Long-term dependencies in sequences

1st example: video frame prediction

2nd example: text prediction and text translation

→ You never know in advance how many words you need
→ There might be several relevant time scales!

Previous slide.

The long-term dependencies in video and text pose challenges, because you never know in advance whether 12 frames (or 12 words) are enough. Even if you design your computer program for 20 frames or words, you may run in a situation where this is not sufficient. In that sense, the time scale of dependencies is arbitrary.

3. Long-term dependencies in sequences

1st example: video frame prediction

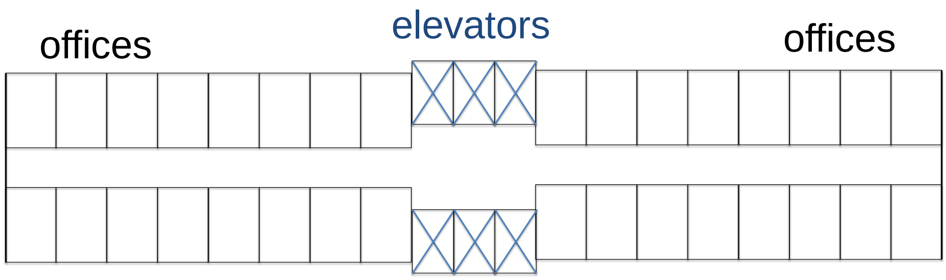
2nd example: text prediction and text translation

3rd example: action planning and navigation

Previous slide.

A similar problem also occurs during action planning and navigation through a city or a building.

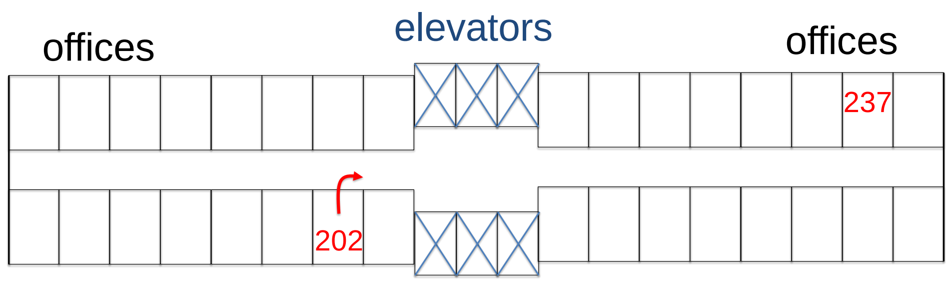
3. Long-term dependencies in action sequences



Previous slide.

Suppose we have a 10-floor building with two wings of offices and elevators in the middle.

3. Long-term dependencies in action sequences



start on floor 2, room 202

meeting on floor 8, room 837

Previous slide.

We start on floor 2 in room 202 and want to go to meet somebody on the 8th floor in room 837 – which is located right above room 237.
You take the elevator …

3. Long-term dependencies in action sequences

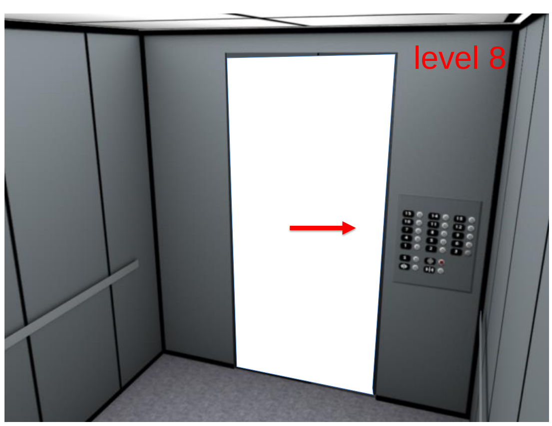
3. Long-term dependencies in action sequences

3. Long-term dependencies in action sequences

3. Long-term dependencies in action sequences

3. Long-term dependencies in action sequences

3. Long-term dependencies in action sequences

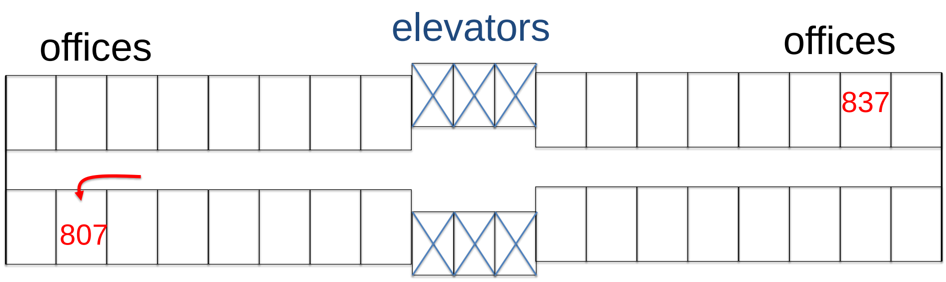


3. Long-term dependencies in action sequences

Previous slide.

And go up … and look around the floor.
At the end of the corridor you will have to turn (right or left????)

3. Long-term dependencies in action sequences



meeting on floor 8, room 837

Previous slide.

The building has a high symmetry. The only way to arrive at the correct office is to remember where you came from.

3. Long-term dependencies in sequences

1st example: video frame prediction

2nd example: text prediction and text translation

3rd example: action planning and navigation

Symmetries create ambiguities in space
Whether you should turn left or right depends on which elevator you took
→ Long-term dependencies
→ You do not know the time scale of dependency a priori

Previous slide.

The problem is the same as in the two earlier examples: you do not know the time scale of the temporal dependencies beforehand.

Quiz: Sequences

[ ] In texts, the longest temporal dependence is about 10-20 words.
[ ] Training data for text sequences is scarce and costly because it needs labeling.
[ ] Training data for video frame prediction is cheap, because there are thousands of videos on the internet and no labeling is needed
[ ] Target values in sequence tasks are always high-dimensional.
[ ] In video frame prediction, if I take the last 1000 frames as input, I am sure to be on the safe side (I am sure to cover all potential temporal dependencies)

Your notes.

3. Long-term dependencies in Sequences

Summary:

- Sequences are everywhere
- more common in reality than static input-output paradigms
- sequences contain dependencies on several time scales (fast as well as slow)
- Maximum time scale is hard to know at the beginning (or even impossible)

→ We need a memory in the model

Previous slide.

Since it is hard to know how long the relevant time scale in a sequence is, we need to build a model that learns to shift items into memory whenever necessary. And this is hard.

2. Long-term dependencies in sequences: Aim

Second Question for Today

how can we keep a memory of past events in artificial neural networks?

Previous slide.

The solution we discuss makes use of recurrent neural networks.

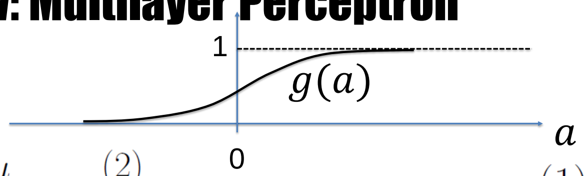
Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve solution: increase number of inputs
3. Long-term Dependencies
4. Recurrent Neural Networks

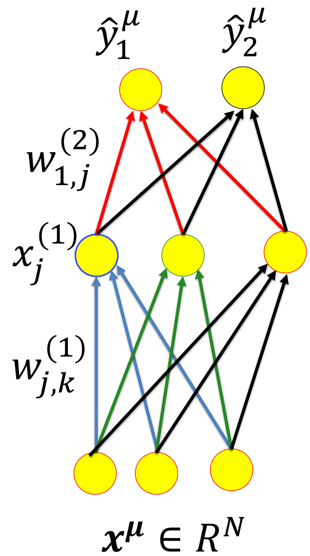
Previous slide.

So far our discussion has been limited to feedforward networks

Review: Multilayer Perceptron



$$
\begin{aligned}
\hat{y}_i^{\mu} &= x_i^{(2)} && (1)\\
&= g^{(2)}[a_i^{(2)}] && (2)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} x_j^{(1)}] && (3)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(a_j^{(1)})] && (4)\\
&= g^{(2)}[\sum_j w_{ij}^{(2)} g^{(1)}(\sum_k w_{jk}^{(1)} x_k^{\mu})] && (5)
\end{aligned}
$$



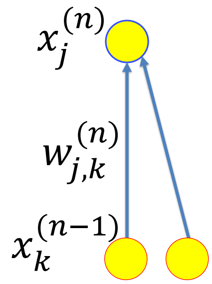
Previous slide.

… where input passes from the input layer to the output. Before we introduce recurrent neural networks we have to define a useful graphical notation.

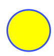
Review: graphical representation

threshold can be removed

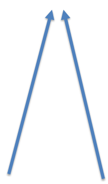
$$x_j^{(n)} = g\left(\sum_k w_{jk}\, x_k^{(n-1)} \cancel{-\vartheta}\right)$$



circle

 $= g(\,.\,)$

converging arrows

 $\sum_k$

Previous slide.

In the following, a circle means a nonlinearity (e.g., rectified linear or sigmoidal). Converging arrows mean summation of values across the input lines. We do not note the threshold explicitly.

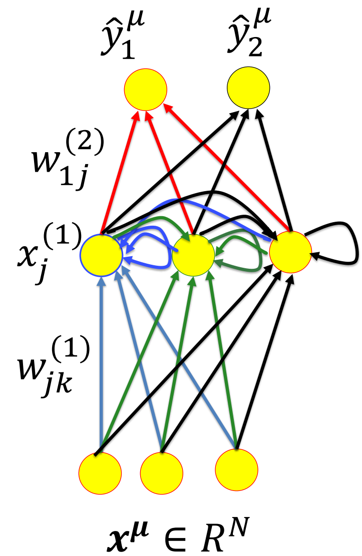
4. Recurrent Neural Networks

neurons in hidden layer
have lateral connections

$$x_j^{(1)} \leftarrow g\left(\sum_k w_{jk}^{(1)}\, x_k^{(0)} \quad + \sum_i w_{ji}^{(lat)}\, x_i^{(1)} \quad\right)$$

(formula can be read off from graph)

Blackboard 1



Previous slide.

With this graphical notation the formula of a simple graphical network can be read off from the graph. The value on the left-hand side is set to the value resulting from the calculation of the right-hand side.

4. Update in Recurrent Neural Network

Include timing information:
Discrete big time steps $t=1,2,\ldots$
Update rule for state of neuron

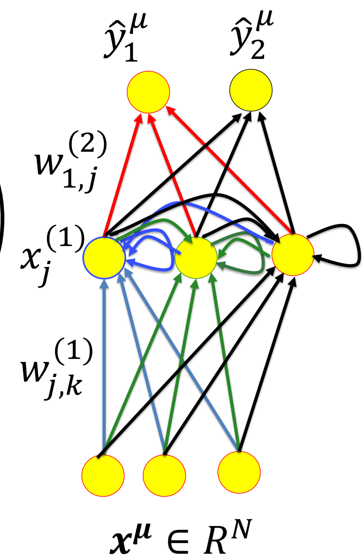
$$x_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\, x_i^{(1)}(t-1)\right)$$

Input at time t:

$\boldsymbol{x}^{\mu}$ with index $\mu = t$

component

$x_k^{(0)}(t) = x_k^t$



Previous slide.

The update can also be written by an explicit time index. We work in discrete time with time steps of 1.

By convention, the complete forward propagation pass is assigned to a SINGLE time step. Therefore, the feedforward input at time t influences the variable x in the SAME time step. This may looks strange at first sight, but arises quite natural when you program a feedforward network where one time step = processing of one input pattern.

However, lateral input from time step t arrives only at time step t=1. Again, this arises naturally because you cannot update neuron $x_i^{(n)}$ of neuron i with the value $x_j^{(n)}$ of neuron j and vice versa, if they are both in the same layer n.

Note that (just as in feedforward networks) we apply in each time step one pattern. However, the order of patterns is not random but fixed by the sequence.

4. Training data for Recurrent Neural Network

input

$\{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3 \ \ldots, \boldsymbol{x}^T\}$ single sequence of length T

target vector for output

$\{\boldsymbol{t}^1, \boldsymbol{t}^2, \boldsymbol{t}^3 \ \ldots, \boldsymbol{t}^{T-1}\}$

one example is: predict next input (e.g. video frame)

$\boldsymbol{t}^1 = \boldsymbol{x}^2$

$\boldsymbol{t}^2 = \boldsymbol{x}^3$

$\boldsymbol{t}^3 = \boldsymbol{x}^4$ ‘target at time step 3 is the input at time step 4’

Previous slide.

The data that we use for supervised learning has a strict temporal order - as opposed to the case of static classification, where we randomly draw patterns from the data base, one at a time (as seen so far in class).

For the case of video frame prediction, the input image at time step m+1 is the label $\boldsymbol{t}^m$ for time step m.

4. Training data for Recurrent Neural Network (text example)

'The grammar book of my friend. The first sentence often begins with a three-letter word, because the word 'the' is quite common. However much longer words are also possible as a first word of a sentence. Therefore this is just a rule of thumb. … '

$\boldsymbol{x}^1$ = character T in 1-hot coding

input

$$\{\boldsymbol{x}^1, \boldsymbol{x}^2, \boldsymbol{x}^3 \ \dots, \boldsymbol{x}^T\}$$

target vector for output

$$\{\boldsymbol{t}^1, \boldsymbol{t}^2, \boldsymbol{t}^3 \ \dots, \boldsymbol{t}^{T-1}\}$$

aim is: predict end of word symbol (text processing)

$$\boldsymbol{t}^1 = 0$$
$$\boldsymbol{t}^2 = 0$$
$$\boldsymbol{t}^3 = 1$$

'target at time step 3 is the 'blank' at time step 4'

Previous slide.

Example:
Suppose that the labels are one-dimensional (t = 0 or 1). The task is to predict in a written text, whether the current word has finished, i.e., predict that the next character is the 'blank-character'.

4. Update in Recurrent Neural Network (details)

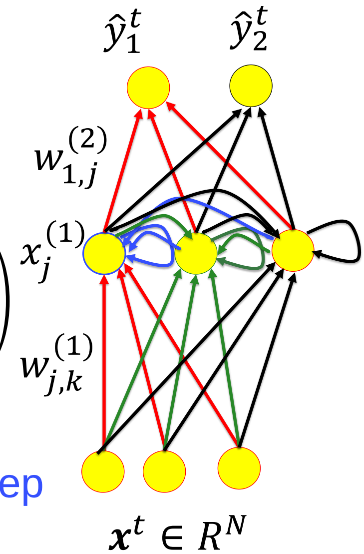
Discrete big time steps $t=1,2,$ ...

$$\hat{y}_i^t = \hat{y}_i\ (t) = g\left(\sum w_{ij}^{(2)} x_j^{(1)}(t)\right)$$

$$x_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1)}\ x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\ x_i^{(1)}(t-1)\right)$$

Feedforward processing
within the same time step
(feedforward pass)

Lateral input
from previous step



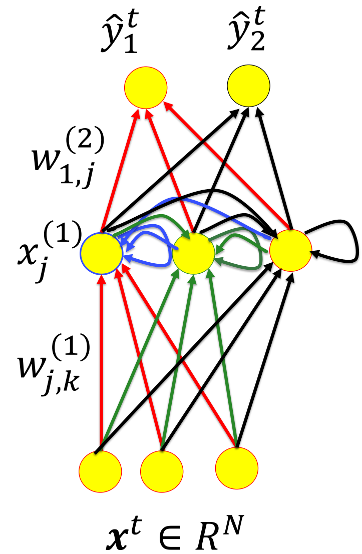
Previous slide.
Let us now study a recurrent network with one hidden layer in more detail.

4. Update in Recurrent Neural Network (details)

Update scheme looks complicated.

Question:

How does this work in practice?



Previous slide.

When we look more closely at the update rule of the network we find a process called 'unfolding in time' – which is the topic of the next section.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve solution: increase number of inputs
3. Long-term Dependencies
4. Recurrent Neural Networks
5. Unfolding the network in time

Previous slide.

To discover unfolding in time, you start with exercise 1.

5. Update in Recurrent Neural Network

Exercise 1
In Class (8min)

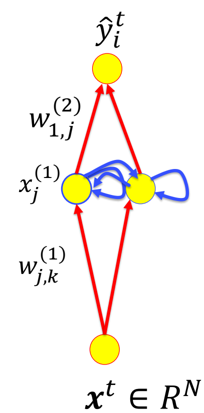
Discrete big time steps $t=1,2,\ \ldots$

$$\hat{y}_i^t = \hat{y}_i\ (t) = g\left(\sum w_{ij}^{(2)} x_j^{(1)}(t) - \vartheta\right)$$

$$x_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1)}\ x_k^{(0)}(t)\quad + \sum_i w_{ji}^{(lat)}\ x_i^{(1)}(t-1) - \vartheta_j\right)$$

Feedforward processing
within one big time step

Lateral input
from previous step



Previous slide.

Exercise 1. Unfolding in time (In Class)

We consider a neural network with one recurrent hidden layer as shown in class. The output is (we have suppressed the threshold ϑ)

$$\hat{y}_i(t) = g[\sum_j w_{ij}^{(2)} x_j^{(1)}(t)] \tag{1}$$

and neurons in the hidden layer have an activity

$$x_j^{(1)} = g[\sum_k w_{jk}^{(1)} x_k^{(0)}(t) + \sum_i w_{ji}^{(2)} x_i^{(1)}(t-1)] \tag{2}$$

a. Evaluate the output at time step $t = 4$ in terms of the weights and the inputs $x_k^{(0)}$ given at time steps $t = 1, 2, 3, 4$.

Hint: Insert the formula for $x_j^{(1)}$ into the formula for the output, and repeat the procedure recursively. Keep track of the time steps!

b. Construct an equivalent feedforward network.

5. Unfolding in time

Blackboard 2

Your notes.

5. Compact graphics for Recurrent Neural Network

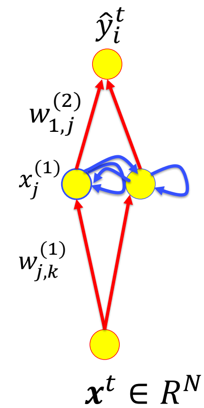
Discrete big time steps $t=1,2, \ldots$

$$\hat{y}_i^t = \hat{y}_i\ (t) = g\left(\sum w_{ij}^{(2)} x_j^{(1)}(t) - \vartheta\right)$$

$$x_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1)}\ x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\ x_i^{(1)}(t-1) - \vartheta_j\right)$$

Feedforward processing
within one big time step

Lateral input
from previous step



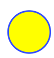
Previous slide.

Since the graphical representation of our recurrent network still looks complicated, we simplify it further.

Review: graphical representation

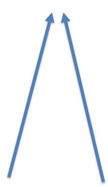
$$x_j^{(n)}(t) = g\left(\sum_k w_{jk}\, x_k^{(n-1)}(t) \quad + \sum_i w_{ji}\, x_i^{(n)}(t-1)\right)$$

circle

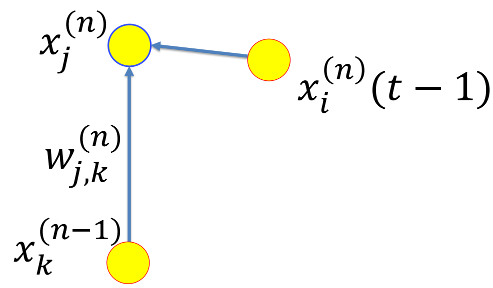


$= g(\,.\,)$

converging arrows



$\sum$



Previous slide.

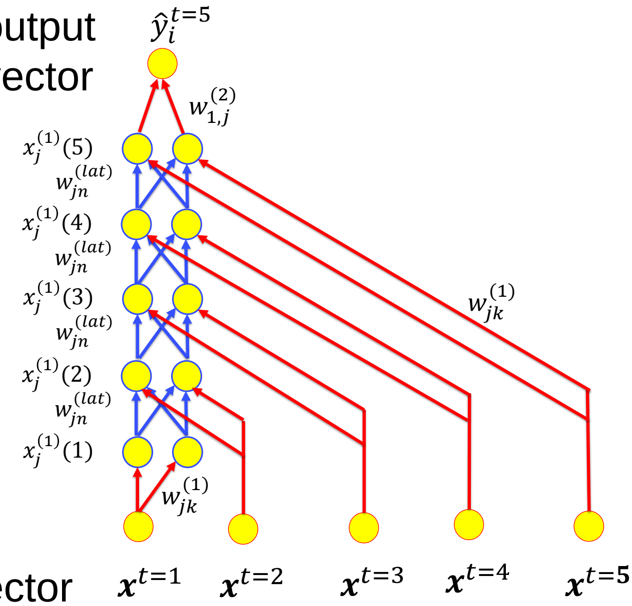
To do so, we reuse the same graphical elements that were introduced earlier.

5. unfolded graphics for Recurrent Neural Network

Discrete big time steps
t=1,2,3,4,5

equivalent
feedforward network
for 5 time steps

output
vector



input vector

Previous slide.

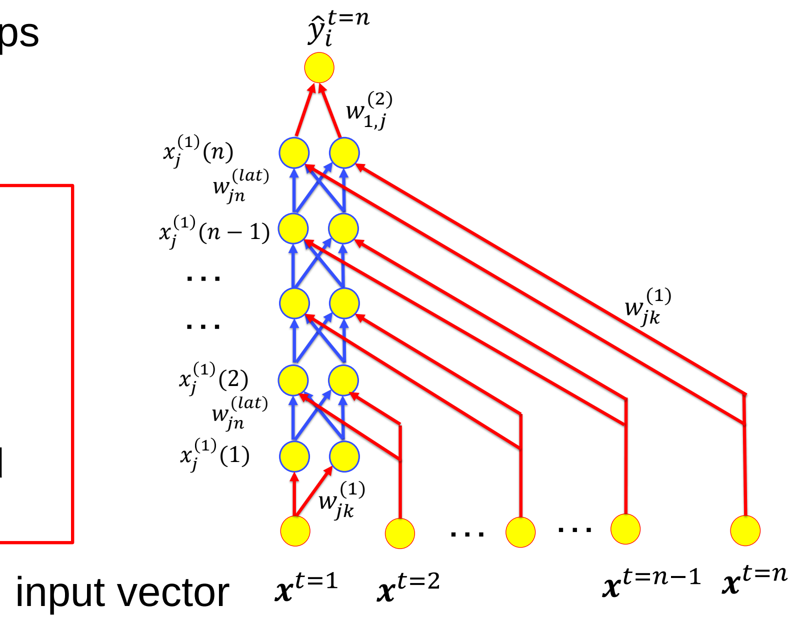
With this representation we find that the recurrent network (in time step 5) is equivalent to a feedforward network where input pattern $\boldsymbol{x}^{t=5}$ is injected in the last hidden layer, input pattern $\boldsymbol{x}^{t=4}$ in the previous one, and input pattern $\boldsymbol{x}^{t=1}$ in the first layer.

5. unfolded graphics for Recurrent Neural Network

Discrete big time steps

t=1,2,3,4,5, … , n

equivalent feedforward network for *n* time steps
→
n hidden layers with identical feedforward weights



Previous slide.

Moreover, the matrix of feedforward weights from layer n-1 to layer n is identical to that from layer n to n+1.

Quiz: Unfolding of Recurrent Networks

We process a **sequence of length *T*.**

[ ] When processing a sequence of length *T*,
a recurrent network with one hidden layer can always be reformulated as a deep feedforward network.

[ ] A recurrent network with one hidden layer of *n* neurons leads to an unfolded feedforward network with *n* layers of *n* neurons each.

[ ] A recurrent network with one hidden layer of *n* neurons leads to an unfolded feedforward network with *T* hidden layers

[ ] The unfolded network corresponds to a feedforward network with weight sharing.

[ ] The unfolded network corresponds to a feedforward network where inputs have direct short-cut connections to all hidden layers.

Your notes.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve solution: increase number of inputs
3. Long-term Dependencies
4. Recurrent Neural Networks
5. Unfolding the network in time
6. Backpropagation through time

Previous slide.

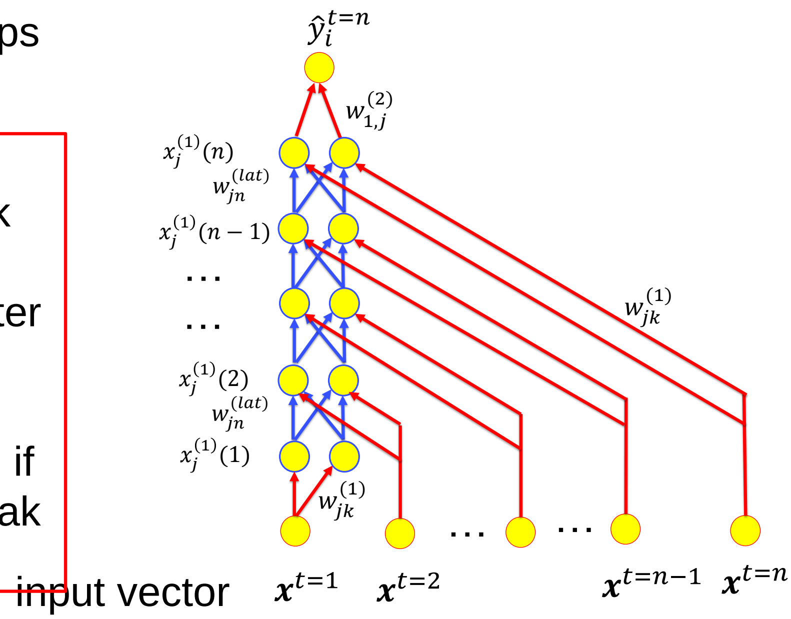
How can we update the weights?
The solution is called backpropagation through time.

6. Backpropogation through time

Discrete big time steps

t=1,2,3,4,5, … , n

- take the unfolded equivalent network
- apply backprop after each time step
- cut backward path if signal gets too weak



Previous slide.
In order to understand Backpropagation through time, we start with the unfolded equivalent feedforward network. For the feedforward network, we apply standard Backpropagation.

The name Backpropagation through time stems from the fact that the different layers of the equivalent feedforward network correspond to discrete time steps.

BackProp

0. **Initialization of weights**
1. **Choose pattern $\mathbf{x}^\mu$**

   input $x_k^{(0)} = x_k^\mu$
2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$
3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \quad (2)$$

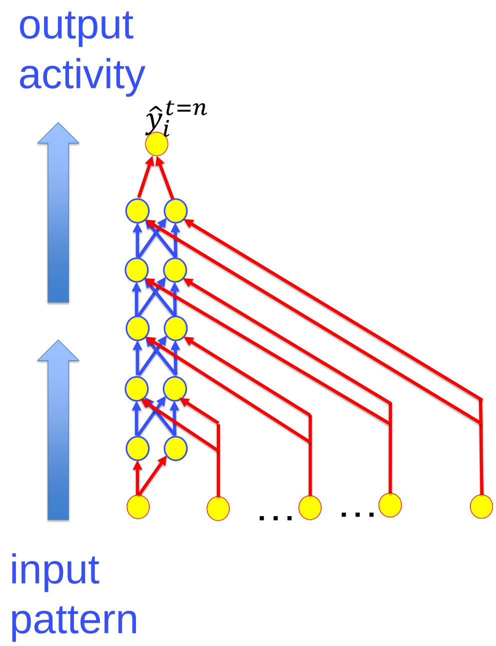
4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \quad (3)$$

5. **Update weights (for each (i,j) and all layers (n))**

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \quad (4)$$

6. **Return to step 1.**



BackProp

0. **Initialization of weights**
1. **Choose pattern $\mathbf{x}^\mu$**

   input $x_k^{(0)} = x_k^\mu$
2. **Forward propagation of signals** $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$
3. **Computation of errors in output**

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \quad (2)$$

4. **Backward propagation of errors** $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \quad (3)$$

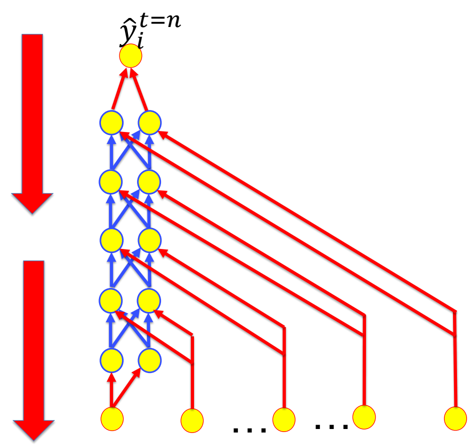
5. **Update weights (for each (i,j) and all layers (n))**

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \quad (4)$$

6. **Return to step 1.**

Calculate output error

δ



0. Initialization of weights

BackProp

1. Choose pattern $\mathbf{x}^{\mu}$

input $x_k^{(0)} = x_k^{\mu}$

2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

output $\hat{y}_i^{\mu} = x_i^{(n_{\max})}$

3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^{\mu} - \hat{y}^{\mu}] \quad (2)$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \quad (3)$$

5. Update weights (for each (i, j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \quad (4)$$

6. Return to step 1.

update **all** weights

$$\Delta w_{i,j}^{(n)} = \delta_i^{(n)} x_j^{(n-1)}$$

Previous slide.

Therefore BackProp through time is just the standard BackProp algorithm – but applied to the equivalent feedforward network.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve solution: increase number of inputs
3. Long-term Dependencies
4. Recurrent Neural Networks
5. Unfolding the network in time
6. Backpropagation through time
7. The vanishing Gradient Problem

Previous slide.

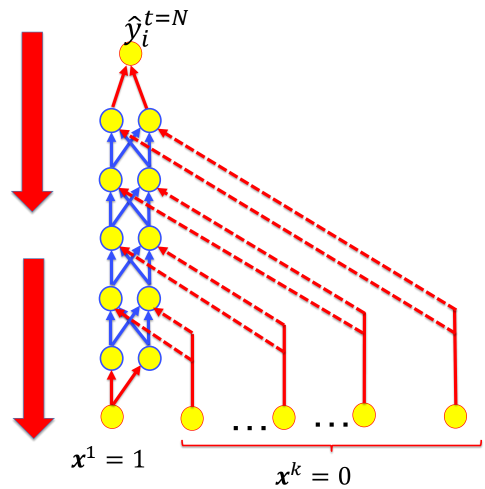
The vanishing gradient problem that we have seen in one of the previous lectures is also a problem for recurrent networks.

7. Vanishing gradient problem

- Assume strong input at time *t=1*
- Assume no further input up to time $t{=}N$
- Calculate error in output
- Backpropagate over N layers to find the effect of earlier input on the output now

Calculate output error

δ



Previous slide.

Suppose we are in time step t= N and observe a mismatch between our network output and the target value.

The last non-zero input occurred in time step t=1.

Question: can BackProp learn to connect the output at time step N with the input at time step 1?

In the backward pass, the

BackProp

0. Initialization of weights
1. Choose pattern $\mathbf{x}^\mu$

   input $x_k^{(0)} = x_k^\mu$
2. Forward propagation of signals $x_k^{(n-1)} \longrightarrow x_j^{(n)}$

$$x_j^{(n)} = g^{(n)}(a_j^{(n)}) = g^{(n)}(\sum w_{jk}^{(n)} x_k^{(n-1)}) \quad (1)$$

   output $\hat{y}_i^\mu = x_i^{(n_{\max})}$
3. Computation of errors in output

$$\delta_i^{(n_{\max})} = g'(a_i^{(n_{\max})})\ [t_i^\mu - \hat{y}^\mu] \quad (2)$$

4. Backward propagation of errors $\delta_i^{(n)} \longrightarrow \delta_j^{(n-1)}$

$$\delta_j^{(n-1)} = g'^{(n-1)}(a^{(n-1)}) \sum_i w_{ij}\, \delta_i^{(n)} \quad (3)$$

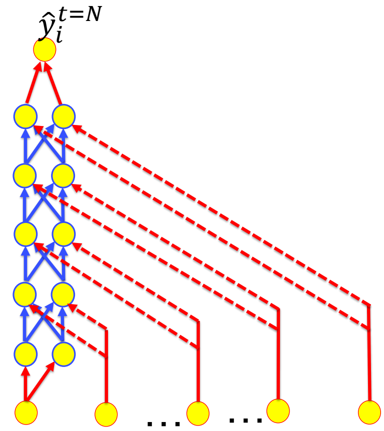
5. Update weights (for each (i, j) and all layers (n))

$$\Delta w_{ij}^{(n)} = \eta\, \delta_i^{(n)}\, x_j^{(n-1)} \quad (4)$$

6. Return to step 1.

Calculate output error

δ



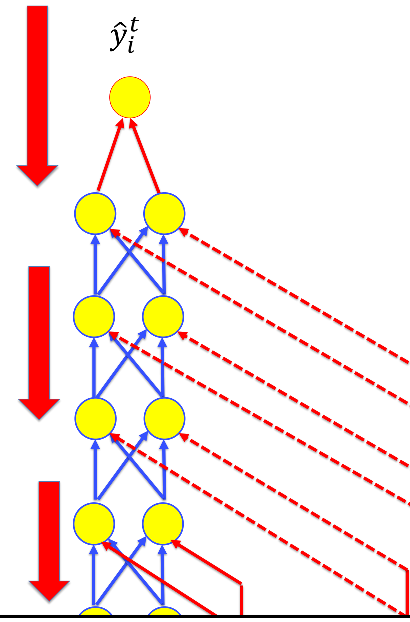
Previous slide.

For each layer of the backward pass, we get a derivative g’ and a sum of weights. But the weights in layer n are copies of those in layer n+1. Therefore the delta-error information will either blow up or decay. The latter is the reason for the name ‘vanishing gradient problem’. See also exercises this week.

7. Vanishing gradient problem

- Assume strong input at time *t-N*,
- Assume no further input up to time t
- Calculate error in output
- Backpropagate over N layers to find the effect of input

$$\delta_i^{(n-1)} = \sum_j w_{ji}^{(lat)} g'^{(n-1)}(a_i^{(n-1)}) \delta_j^{(n)}$$

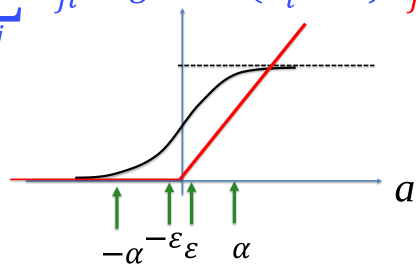


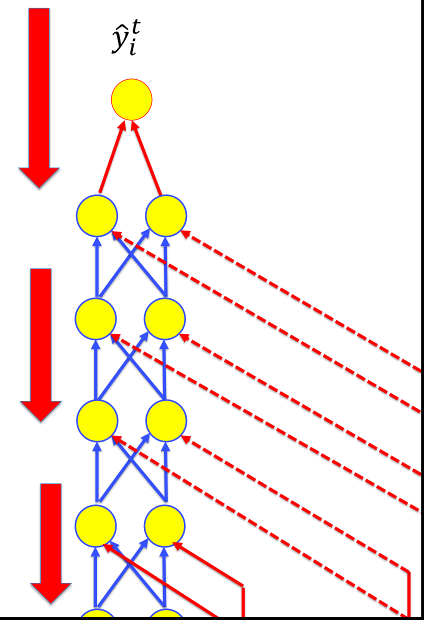
Previous slide.

This slide is just a copy of the vanishing gradient argument from an earlier lecture (only the image has been changed.

7. Vanishing gradient problem

$$\delta_i^{(n-1)} = \sum_j w_{ji}^{(lat)} g'^{(n-1)}(a_i^{(n-1)}) \delta_j^{(n)}$$





After N layers: each path contributes

$$\delta_i^{(t-N)} \sim g'^{(1)} w_{ji}^{(lat)} g'^{(2)} w_{ji}^{(lat)} \dots g'^{(N-1)} w_{ji}^{(lat)} \delta_j^{(N)}$$

Many terms to be summed,
but most terms vanish if |g'w| < 1

Previous slide.

The sum can be decomposed in many different paths, but the contributions of most paths is very, very small.

Quiz: Vanishing Gradient Problem

The vanishing gradient problem of recurrent network means that

[ ] the derivative of the gain function vanishes: $g' = 0$

[ ] that the output error at time t contains only very little information about input at an earlier time step $t\text{-}k$ if $k>10$

[ ] that $|g' \, w_{ji}^{(lat)}|^k \approx 0$ for $k>10$

Your notes. The value k>10 is somewhat arbitrary. I could also have written k>>1.

7. Summary: Vanishing Gradient Problem

It is hard to learn long-term dependencies of sequence data with a (normal) recurrent neural network
using backpropagation.

Your notes.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

1. Sequences
2. Naïve solution: increase number of inputs
3. Long-term Dependencies
4. Recurrent Neural Networks
5. Unfolding the network in time
6. Backpropagation through time
7. The vanishing Gradient Problem
8. Long Short-Term Memory (LSTM)

Previous slide.

We now consider a variant of a recurrent network that avoids the vanishing gradient problem. It has been called the Long Short-term Memory (LSTM) and invented by Sepp Hochreiter and Jurgen Schmidhuber. The LSTM or modern variants of it are the basis of modern networks for speech recognition or text translation. The 'Neural Turing Machines' can be seen also seen as modifications of the same basic ideas.

8. Long short-term memory (LSTM)

Two basic ideas

(i) Hard to keep memory in a recurrent network
→ define explicit memory units

(ii) Avoid the vanishing gradient problem
→ make sure that $g'^{(1)} w_{ji}^{(lat)} = 1$

Previous slide.

The are two different motivations for the introduction of LSTMs.

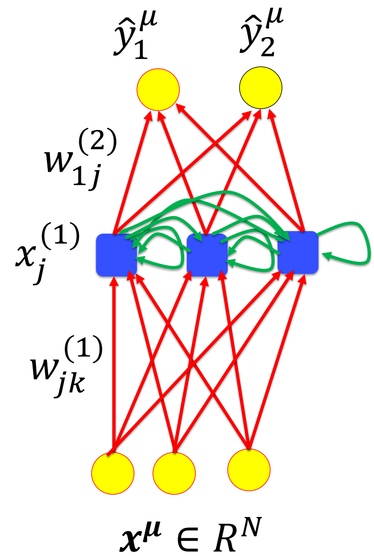
The first motivation is a functional one: recurrent neural networks are used to link events at time step t with earlier events several time steps before. The most natural way to do this would be explicit memory units.

The second one is related to the vanishing gradient problem. To avoid the vanishing gradient problem we have to make sure that $g'^{(1)} w_{ji}^{(lat)} = 1$

8. Long short-term memory (LSTM)

Replace neurons in hidden layer
by memory units

= 1 memory unit
= 1 LSTM unit

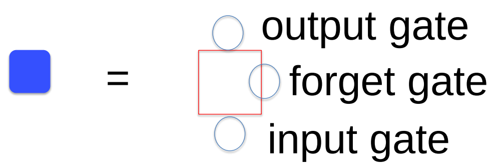


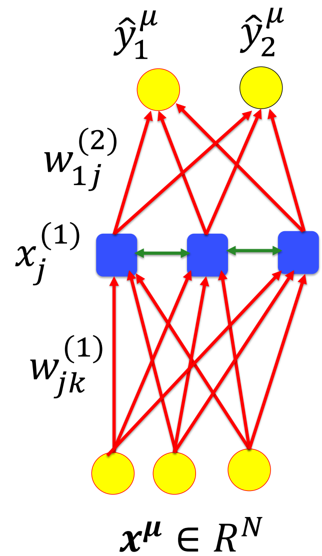
Previous slide.

To introduce explicit memory, we replace each neuron in the hidden layer by a 'memory unit' also called LSTM unit.

8. Long short-term memory (LSTM)

Replace neurons in hidden layer by LSTM units





Previous slide.

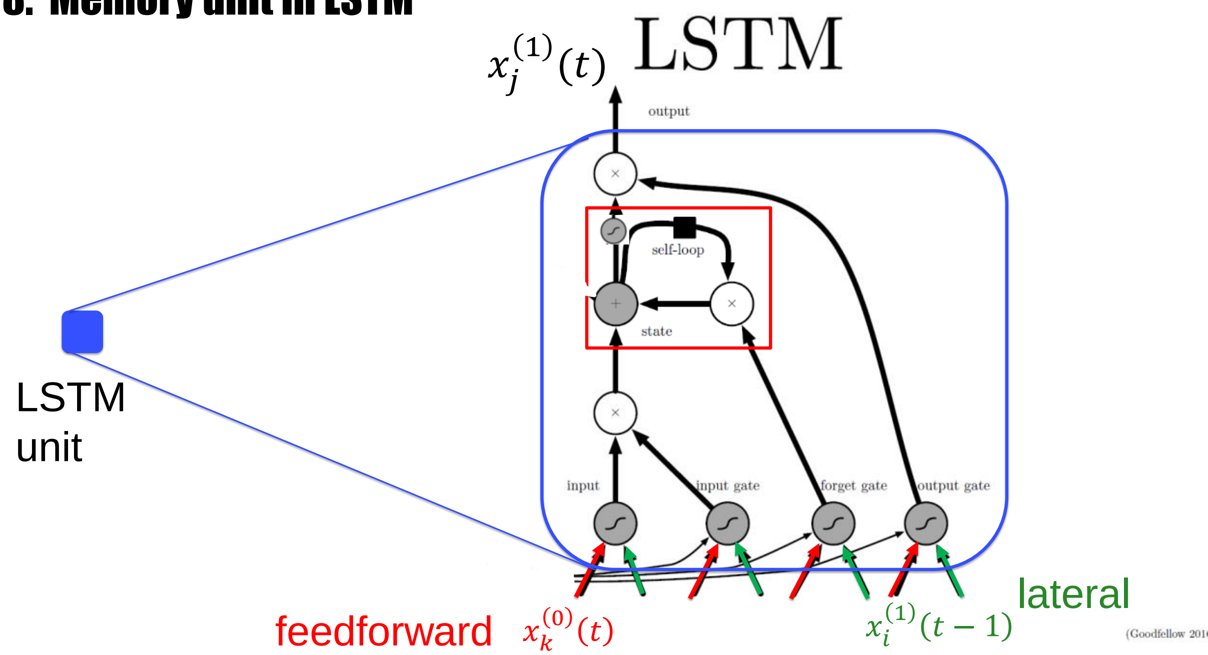
If we zoom in and look a little bit more closely, we see that each LSTM has three gates:

An input gate that controls when some input is added to the memory ('write to memory')

An output gate that controls when the current value is read out form the memory ('read memory)

And a forget gate that controls when the current value is suppressed ('reset memory')

8. Memory unit in LSTM



Previous slide.

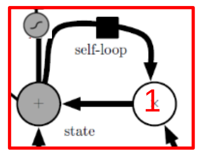
An even closer look shows that the gates are themselves controlled by feedforward (red) and lateral (green) inputs.

The input gate multiplies the input with the value of the gate and adds the result to the memory.
The output gate multiplies the current value of the memory with the value of the gate and conveys the result further on along the links to other units.
The forget gate multiplies the current value of the memory with the value of the gate and sends the result back to the memory unit (and overwrites its old value).

Black square: delay of one time step.
In the next few slides we will work our way, step by step, through this image.

Image adapted from Gers et al. 2000 (Neural Computation) and Goodfellow et al (Deep Learning, MIT Press).

8. Memory unit in LSTM



Internal state s of memory

$$s_j^{(1)}(t) = 1 \cdot s_j^{(1)}(t-1)$$

Compare: $x_j^{(1)}(t) = g[w \cdot s_j^{(1)}(t-1)]$ set $g(a)=a$ and $w=1$

Previous slide.

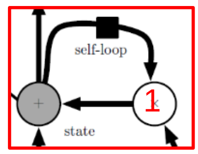
Let us focus on the core of the memory unit.
(For the moment we disregard the forget gate and set the multiplication factor to 1).
The memory unit keeps its value from one time step to the next, by circling it around.

This is equivalent to saying that we work with a small recurrent network consisting of a single neuron with a linear gain function of slope one and recurrent weight one. Hence there is no vanishing gradient problem.

8. Memory unit in LSTM: writing into memory

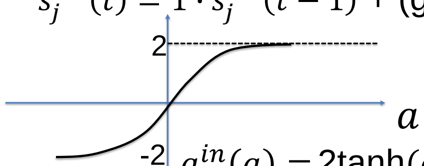
'write in memory when useful for the task'

Internal state s of memory



$s_j^{(1)}(t) = 1 \cdot s_j^{(1)}(t-1)$ + (gated) ***input***

$s_j^{(1)}(t) = 1 \cdot s_j^{(1)}(t-1)$ + (gated) $g^{in}[\sum_k w_{jk}^{(1)}\, x_k^{(0)}(t) \;\; + \sum_i w_{ji}^{(lat)}\, x_i^{(1)}(t-1) - \vartheta_j]$



$g^{in}(a) = 2\tanh(a)$

$x_k^{(0)}(t)$ feedforward

input

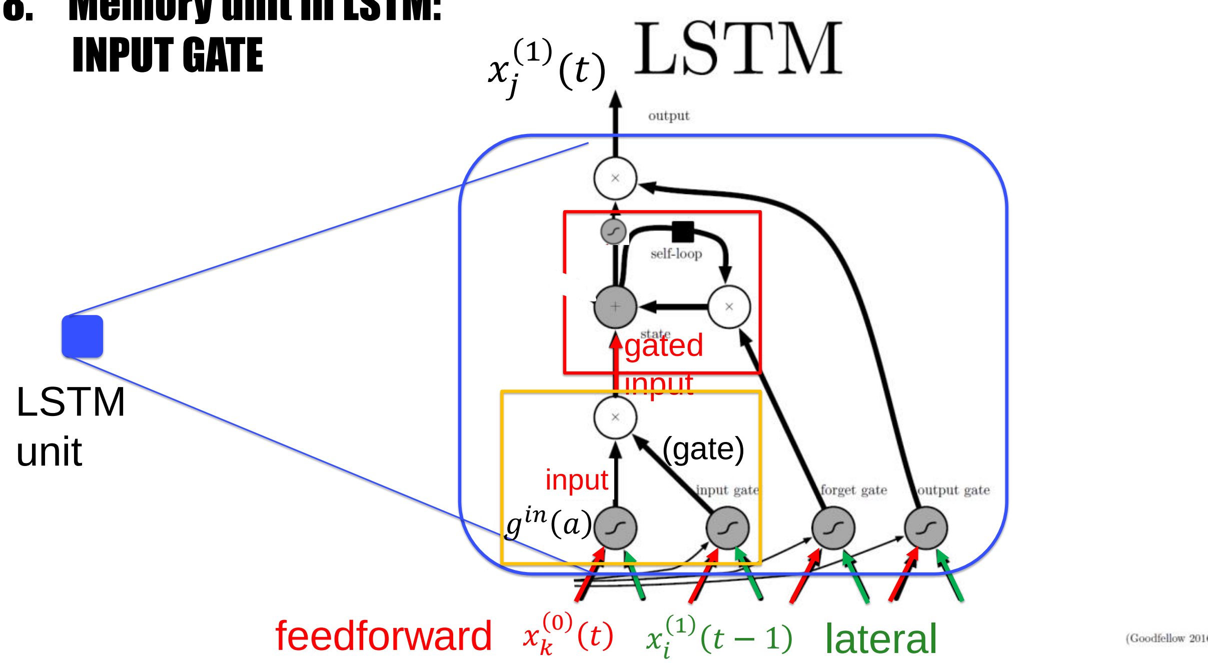
$x_i^{(1)}(t-1)$ lateral

Previous slide.

If the input gate is open (value = +1), then g(a) is added to the value s of the meory unit. Here a is the weighted sum over feedforward input and lateral input.

Gers and Schmidhuber use in the input pathway a gain function $g^{in}(a)$ which ranges from -2 to +2.
The next slide shows where this gain function sits in the input pathway.

8. Memory unit in LSTM: INPUT GATE



Previous slide.

Let us now focus on the input gate (orange box).
The value of the input pathway is multiplied with the gating value:
(gate) = Y = g(a)

Whether the gating value Y is nonzero depends on its activation value a.
The function g(a) is a standard sigmoidal between zero and 1.
The details of the equation Y=g(a) = … are given on the next slide.

8. LSTM – input gate

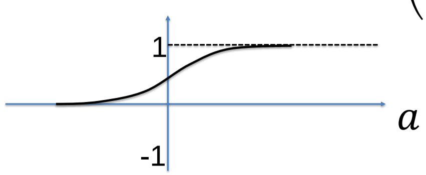
Internal state s of memory

$$s_j^{(1)}(t) = 1\cdot s_j^{(1)}(t-1) + \text{(gated) } g^{in}\overbrace{\left[\sum_k w_{jk}^{(1)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\, x_i^{(1)}(t-1) - \vartheta_j\right]}^{\text{input}}$$

$$s_j^{(1)}(t) = 1\cdot s_j^{(1)}(t-1) + \quad Y_j^{(1)} g^{in}\left[\sum_k w_{jk}^{(1)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\, x_i^{(1)}(t-1) - \vartheta_j\right]$$

Gating variable Y of input

$$Y_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1,Y)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat,Y)}\, x_i^{(1)}(t-1) - \vartheta_j^{(1,Y)} - \mathbf{1}\right)$$



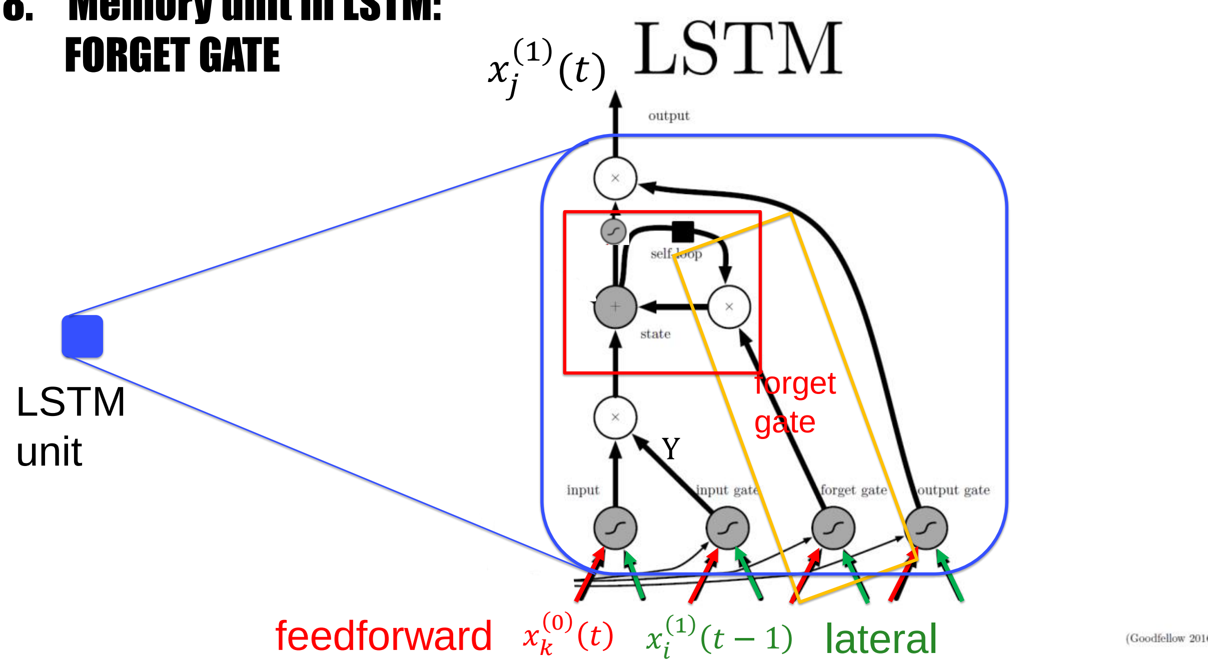
$x_k^{(0)}(t)$ ***input*** $x_i^{(1)}(t-1)$

feedforward lateral

Previous slide.
The input gate is characterized by a gating variable Y = g(a).
where a depends on weights which are learnable.

The parameters of the input gate are initialized with negative bias (highlighted by red color of the bias value -1): therefore the gate has to learn WHEN to write into the cell.
Default is that the memory is idling and not changed.

8. Memory unit in LSTM: FORGET GATE



Previous slide.

Let us now focus on the forget gate (orange box, diagonal).

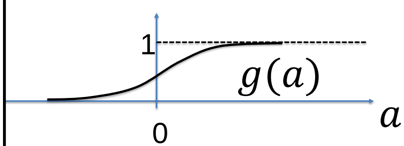
8. LSTM – Forgetting gate (initialize at 1 or close to 1)

$$s_j^{(1)}(t) = 1 \cdot s_j^{(1)}(t-1) + \quad Y_j^{(1)}\, g[\textstyle\sum_k w_{jk}^{(1)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\, x_i^{(1)}(t-1) - \vartheta_j]$$

$$s_j^{(1)}(t) = f \cdot s_j^{(1)}(t-1) + \quad Y_j^{(1)}\, g[\textstyle\sum_k w_{jk}^{(1)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat)}\, x_i^{(1)}(t-1) - \vartheta_j]$$

Gating variable f for forgetting

$$f_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1,f)}\, x_k^{(0)}(t) \quad + \sum_i w_{ji}^{(lat,f)}\, x_i^{(1)}(t-1) - \vartheta_j^{(1,f)} + 1\right)$$



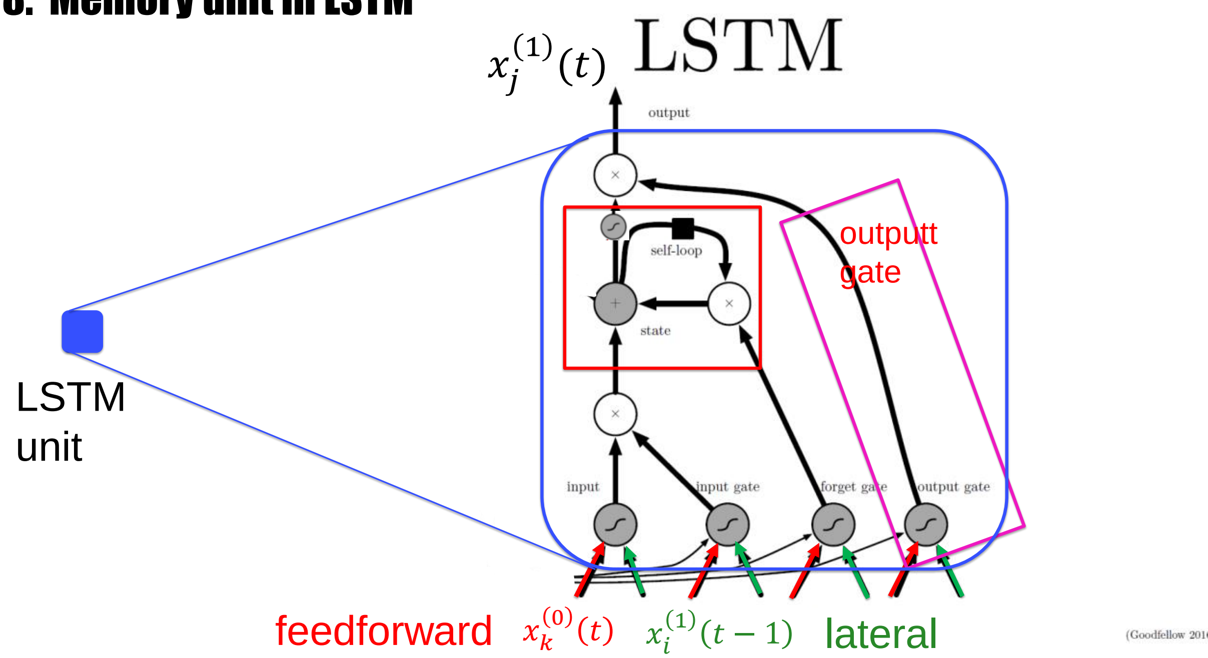
$x_k^{(0)}(t)$ **_input_** $x_i^{(1)}(t-1)$

feedforward lateral

Previous slide.
The forget unit is described by the variable f.

Its bias is initialized at 1 or close to 1. The idea is that normally the memory unit should keep the memory. However, thanks to the weights, the forget unit can learn, WHEN the memory should be forgotten. Forgetting occurs if the value of f is close to zero.

8. Memory unit in LSTM

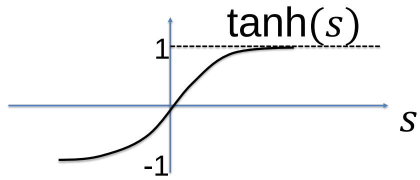


Previous slide.

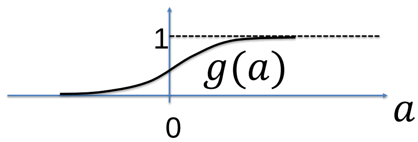
Let us now focus on the output gate (pink box).

8. LSTM –output gate

$$x_j^{(1)}(t) = \; q_j^{(1)} \tanh[s_j^{(1)}(t)]$$



Gating variable q for output $g(a) = 0.5[1 + tanh(a)]$



$$q_j^{(1)}(t) = g\left(\sum_k w_{jk}^{(1,q)} \, x_k^{(0)}(t) \; + \sum_i w_{ji}^{(lat,q)} \, x_i^{(1)}(t-1) - \vartheta_j^{(1,f)} - 1 \right)$$

$x_k^{(0)}(t)$ ***input*** $x_i^{(1)}(t-1)$

feedforward lateral

Previous slide.

The value of the output gate is denoted by a variable q.
Similar to the two other gates, it is controlled by feedforward and lateral inputs.

8. Memory unit in LSTM

Remark (memory block):
1 LSTM unit can have several state variables, controlled by a shared gates

LSTM

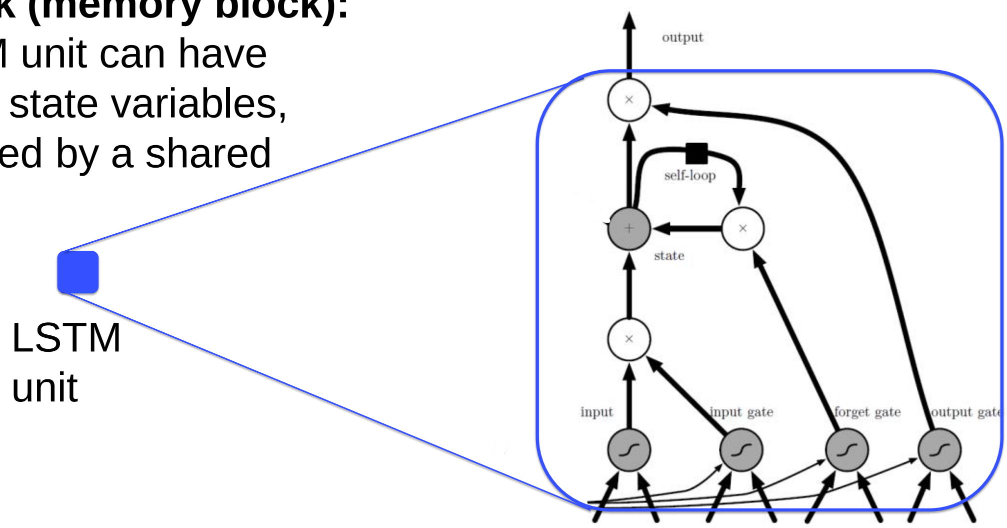
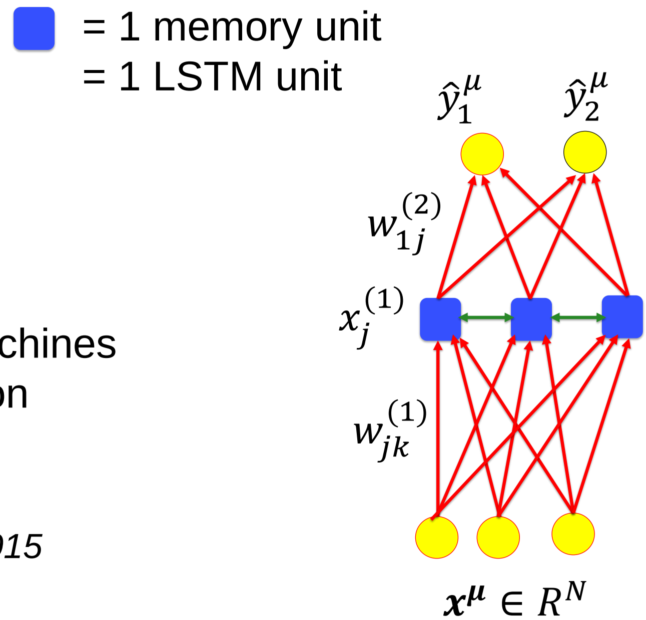


Figure 10.16

Previous slide.

The state variable s of one LSTM can be a vector (i.e., several variables, connected to several inputs). As long as all components are controlled by the same input and output gates, we call it a SINGLE LSTM unit. Gers and Schmidhuber call it an LSTM block.

8. LSTM networks



Trained with BackProp

State of the art for
- text translation by machines
- handwriting recognition
- speech recognition
- image captioning

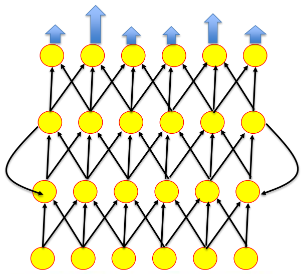
e.g. Xu et al. 2015

Previous slide.

And now we take many of these LSTM units and build networks. LSTM networks are the state-of-the-art approach for all applications that involve sequences.

Deep networks with recurrent connections (Lecture 1)

'a man sitting on a couch with a dog'



Network desribes the image with the words:

'a man sitting on a couch with a dog'

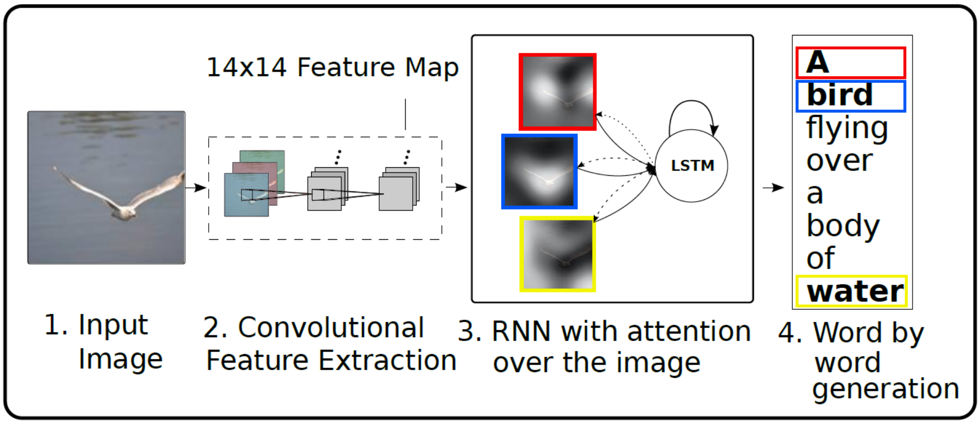
(Fang et al. 2015)

Previous slide.

For example the caption-generating network presented at the beginning of lecture 1 used a recurrent network of LSTM units to generate the text of the caption.

8. LSTM in Deep networks with recurrent connections

Figure 1. Our model learns a words/image alignment. The visualized attentional maps (3) are explained in Sections 3.1 & 5.4



Xu et al. (2015), *Show, attend and tell: Neural image caption generation…, ICML*

Previous slide.

Some of the applications used a slightly simplified variant of LSTM units, but the ideas are the same.

Artificial Neural Networks: Lecture 6
Sequences and Recurrent Networks

Wulfram Gerstner
EPFL, Lausanne, Switzerland

Objectives for today:

- Why are sequences important?
  they are everywhere; labeling is (mostly) for free
- Long-term dependencies in sequence data
  unknown time scales, fast and slow
- Sequence processing with feedforward models
  corresponds to n-gram=finite memory
- Sequence processing with recurrent models
  potentially unlimited memory, but:
- Vanishing Gradient Problem
  error information does not travel back beyond a few steps
- Long-Short-Term Memory (LSTM)
  explicit memory units keep information beyond a few steps
- Application: Music generation

The end