Applied Data Science

Mod.8 – Neural Networks – day 2

CAMBRIDGE SPARK

@cambridgespark

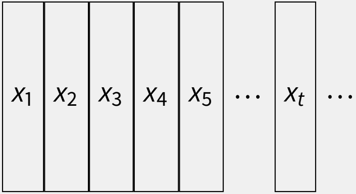
Outline

- CNN and Time Series
- RNN for classification
- RNN for regression

Remark: today is going to be mostly hands on work in the notebook using the Keras API and testing models.

Sequential inputs

- Consider a classification problem where the input is sequential —a sequence consisting of arbitrarily many steps , wherein at each step we have p features .



where each x_i has p-dimensions (features)

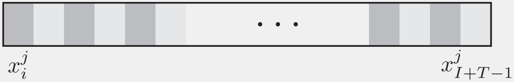
- The fully connected layers we used before expect a fixed-sized input and need to be adapted to work with sequences

Working in batches

- to bypass the problem of not having an fixed-sized input, we can work in batches containing a number T of subsequent time steps: $x_i, \ldots, x_{i+T-1}$
- each batch is now a fixed-sized $T \times p$ matrix
- we can now try to learn temporal features from these batches

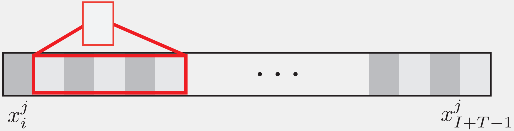
1D convolutions

- For each feature, within the T time steps of the batch, we can try to extract patterns
- A 1D convolution can allow to do this. Instead of looking at square patches of an image, we look at contiguous sections:



1D convolutions

- For each feature, within the T time steps of the batch, we can try to extract patterns
- A 1D convolution can allow to do this. Instead of looking at square patches of an image, we look at contiguous sections:



1D convolutions: exactly like 2D

Given a kernel k (now a vector of dimension w, the *window size*), the convolution is obtained by sliding a window of size w along the sequence and summing the element-wise products:

$$\sum_{\ell=1}^{w} x_{i_\ell}^{j} k_\ell$$

where

- k_ℓ is the ℓ-th entry of the kernel (this is *learned*)
- $x_{i_\ell}^{j}$ is the ℓ-th element of the current time window considered for the j-th feature; i is related to the batch

1D convolutions: exactly like 2D

Given a kernel k (now a vector of dimension w, the *window size*), the convolution is obtained by sliding a window of size w along the sequence and summing the element-wise products:

$$\sum_{\ell=1}^{w} x_{i_\ell}^{j} k_\ell$$

where

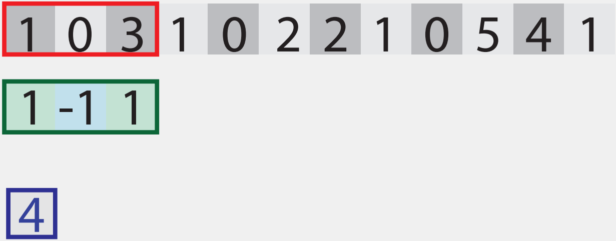
- k_ℓ is the ℓ-th entry of the kernel (this is *learned*)
- $x_{i_\ell}^{j}$ is the ℓ-th element of the current time window considered for the j-th feature; i is related to the batch
- *best to see this in action…*

Conv1D in action

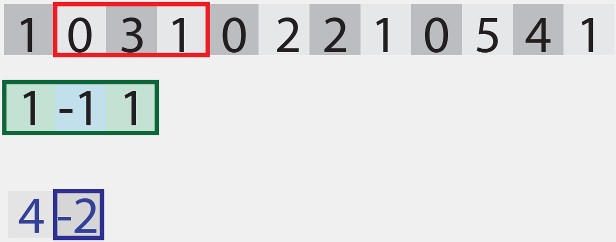
1	0	3	1	0	2	2	1	0	5	4	1

1	-1	1

Conv1D in action



Conv1D in action



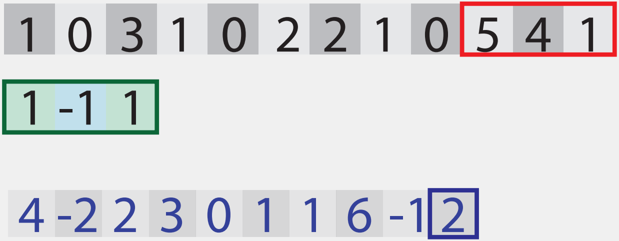
Conv1D in action

1 0 3 1 0 2 2 1 0 5 4 1

1 -1 1

4 -2 2 3 0 1 1 6 -1 2

Conv1D in action



- kernel width: 3
- stride: 1

The 1D convolution layer

For each feature:

- slide a window of a given width (e.g.: 3) with a given stride (e.g.: 1), compute the convolutions $\rightarrow$ vector of size L where L is the number of ways you can slide the window along the batch given width and stride (with *padding* if necessary)
- do this for a number of kernels (e.g.: 32)
- you now have a tensor of dimensions $32 \times L \times p$ which you can feed into another layer

Important remark

- window width $\leq$ batch size $\leq$ total number of time steps
- the window width limits the length of dependencies that can be captured (no *long term dependencies*)

Important remark

- window width $\leq$ batch size $\leq$ total number of time steps
- the window width limits the length of dependencies that can be captured (no *long term dependencies*)
- could you "just feed one big batch"? why?

Hands-on session

>>> CNN for time series pt. 1

A parenthesis: ROC curve

- you've seen how to measure the performances of a classifier via the confusion matrix
- this assumes that you have thresholded the output of the model if it was a probability. For example , if a Neural Network outputs 0.94 for class 1 it's tempting to threshold to "class 1".

A parenthesis: ROC curve

- you've seen how to measure the performances of a classifier via the confusion matrix
- this assumes that you have thresholded the output of the model if it was a probability. For example , if a Neural Network outputs 0.94 for class 1 it's tempting to threshold to "class 1".
- choosing a good threshold value is important especially for highly imbalanced problems

A parenthesis: ROC curve

- you've seen how to measure the performances of a classifier via the confusion matrix
- this assumes that you have thresholded the output of the model if it was a probability. For example , if a Neural Network outputs 0.94 for class 1 it's tempting to threshold to "class 1".
- choosing a good threshold value is important especially for highly imbalanced problems
- the Receiver Operating Characteristic curve helps you do this and is also a nice way to compare classifiers

ROC curve: True Positive vs False Positive rate

Recall the confusion matrix (except that now everything depends on the threshold)

	predicted 0	*predicted 1*
observed 0	TP	FP
observed 1	FN	TN

ROC curve: True Positive vs False Positive rate

Recall the confusion matrix (except that now everything depends on the threshold)

	predicted 0	*predicted 1*
observed 0	TP	FP
observed 1	FN	TN

- true positive rate (TPR): $TP/(TP + FN)$

ROC curve: True Positive vs False Positive rate

Recall the confusion matrix (except that now everything depends on the threshold)

	predicted 0	*predicted 1*
observed 0	TP	FP
observed 1	FN	TN

- true positive rate (TPR): $TP/(TP + FN)$
- false positive rate (FPR): $FP/(FP + TN)$

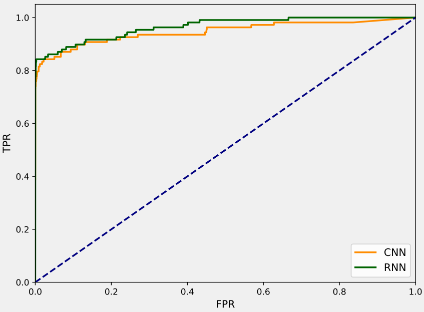
ROC curve: True Positive vs False Positive rate

Recall the confusion matrix (except that now everything depends on the threshold)

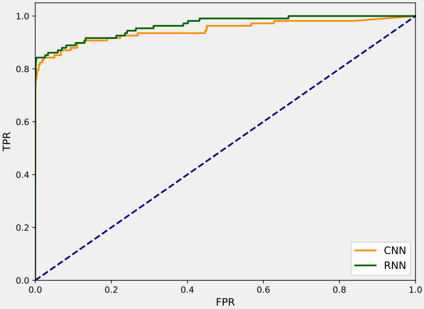
	predicted 0	*predicted 1*
observed 0	TP	FP
observed 1	FN	TN

- true positive rate (TPR): $TP/(TP + FN)$
- false positive rate (FPR): $FP/(FP + TN)$
- the ROC plots the TPR vs the FPR for a range of thresholds

Getting a feel for the ROC



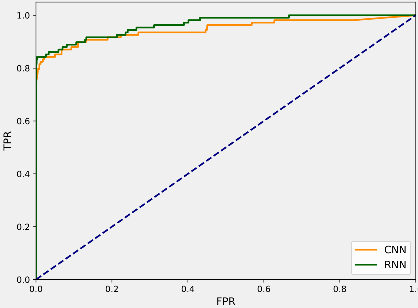
Getting a feel for the ROC



Notes:

- everything is contained within $[0,1] \times [0,1]$

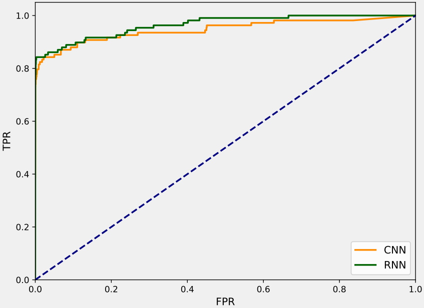
Getting a feel for the ROC



Notes:

- everything is contained within $[0,1] \times [0,1]$
- the diagonal corresponds to a random classifier (why?)

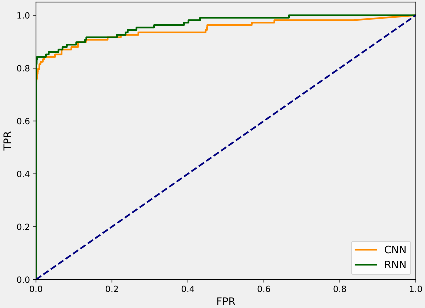
Getting a feel for the ROC



Notes:

- everything is contained within $[0, 1] \times [0, 1]$
- the diagonal corresponds to a random classifier (why?)
- what would the curve look like for a perfect classifier ?

Getting a feel for the ROC



Notes:

- everything is contained within $[0, 1] \times [0, 1]$
- the diagonal corresponds to a random classifier (why?)
- what would the curve look like for a perfect classifier ?
- the Area Under the Curve (AUC) is another metric for classifiers

Hands-on session

>>> CNN for Time Series pt. 2

Recurrent Neural Networks

The idea behind RNN

In quite a general sense, forecasting of a dynamic system usually assumes a form such as:

$$h_t \quad = \quad f(x_t, h_{t-1})$$

where

- x_t is the input to the system
- h_t is the output of the system at time t
- f is some unknown function computing the current output from the current input and the past output.

The idea behind RNN

In quite a general sense, forecasting of a dynamic system usually assumes a form such as:

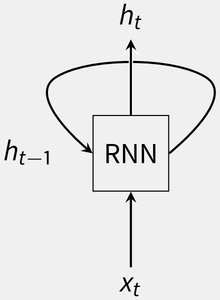
$$h_t \quad = \quad f(x_t, h_{t-1})$$

where

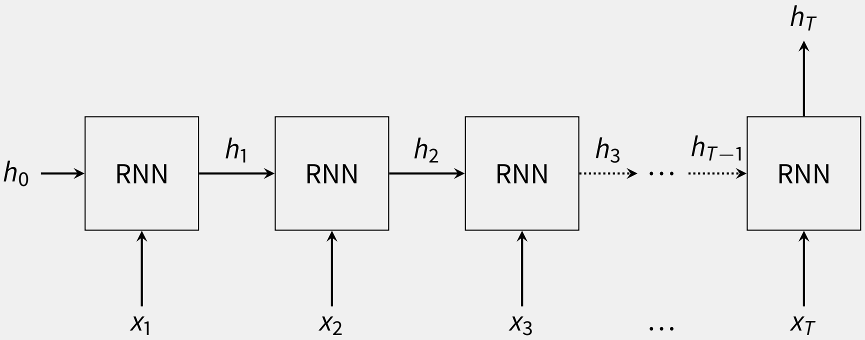
- x_t is the input to the system
- h_t is the output of the system at time t
- f is some unknown function computing the current output from the current input and the past output.
- we can try to learn f and Neural Networks are potentially very good to mimic arbitrary functions...

An RNN cell

from $h_t \quad = \quad f(x_t, h_{t-1})$ to…

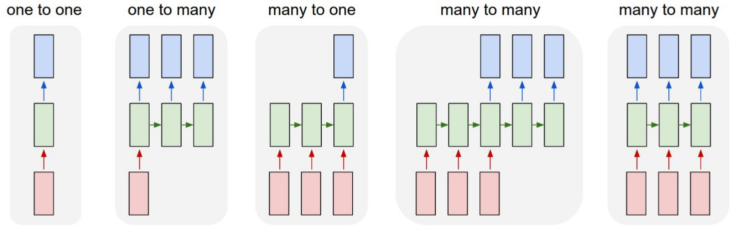


Unrolling the cell...

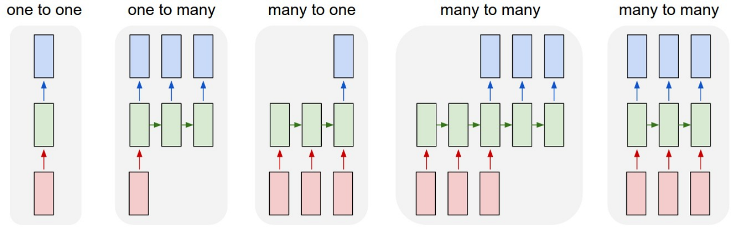


N.B. Every RNN block has the same parameters!

Multiple types of RNN output

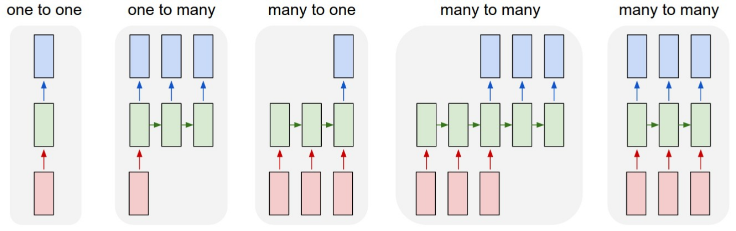


Multiple types of RNN output



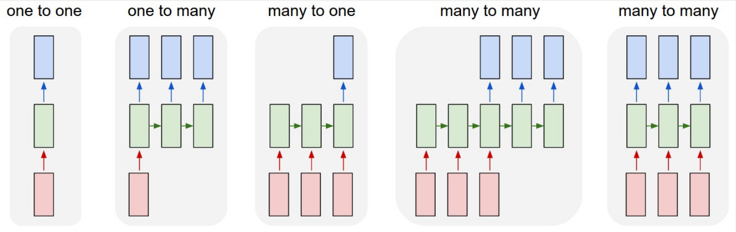
- 1-1 : not recurrent e.g.: image classif. (in= image, out= class)

Multiple types of RNN output



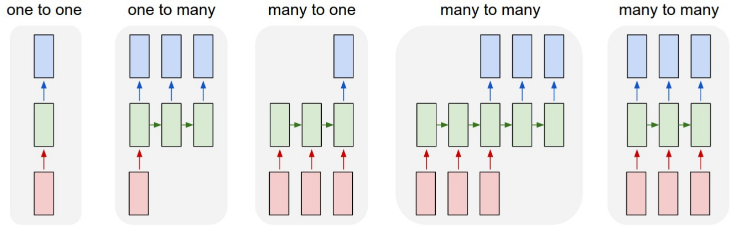
- 1-1 : not recurrent e.g.: image classif. (in= image, out= class)
- 1-M : e.g.: image captioning (in= image, out= sentence)

Multiple types of RNN output



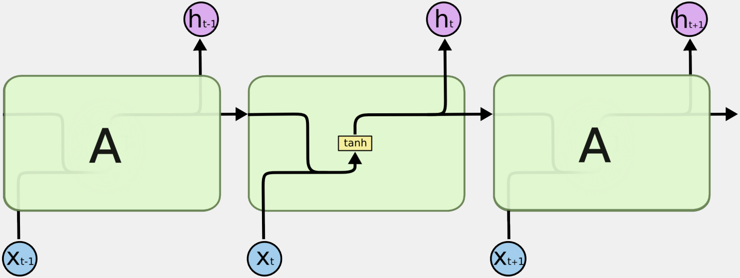
- 1-1 : not recurrent e.g.: image classif. (in= image, out= class)
- 1-M : e.g.: image captioning (in= image, out= sentence)
- M-1 : e.g.: sentiment analysis (in= sentence, out= label)

Multiple types of RNN output

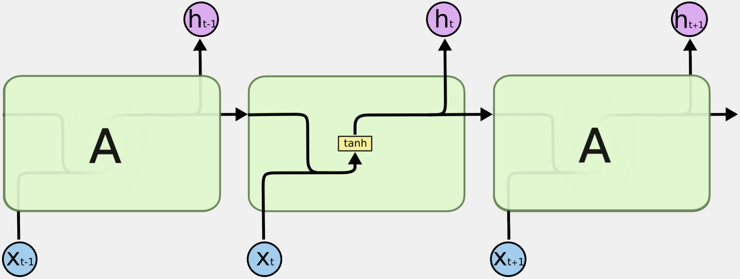


- 1-1 : not recurrent e.g.: image classif. (in= image, out= class)
- 1-M : e.g.: image captioning (in= image, out= sentence)
- M-1 : e.g.: sentiment analysis (in= sentence, out= label)
- M-M : e.g.: translation, video captioning

A Simple RNN (∼ early ‘90s)

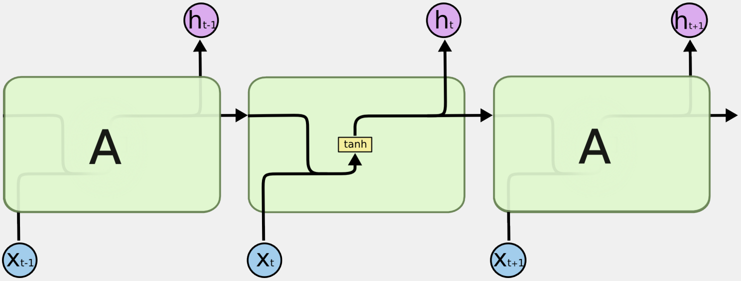


A Simple RNN ($\sim$ early '90s)



- concatenate the output h_{t-1} with x_t
- feed that vector into a single-layer NN with a number (say 32) of neurons and tanh activation function

A Simple RNN ($\sim$ early '90s)



- concatenate the output h_{t-1} with x_t
- feed that vector into a single-layer NN with a number (say 32) of neurons and tanh activation function
- the (only) parameters are the parameters of the 32 neurons.

Hands-on session

>>> getting a feel for the RNN

Issues when training an RNN

Recall that to train a NN, we follow a gradient-descent scheme:

$$\theta_{t+1} \quad \leftarrow \quad \theta_t - \kappa_t G_t$$

where G_t is (an approximation of) the gradient of the loss over the Neural Network parametrised with θ_t.

Issues when training an RNN

Recall that to train a NN, we follow a gradient-descent scheme:

$$\theta_{t+1} \quad \leftarrow \quad \theta_t - \kappa_t G_t$$

where G_t is (an approximation of) the gradient of the loss over the Neural Network parametrised with θ_t.

In the presence of loops (recurrence), the gradient of one RNN cell gets multiplied by itself at each loop. Simplifying a bit, it looks like:

$$\theta_{t+1} \quad \leftarrow \quad \theta_t - \kappa_t (G_t)^M$$

Issues when training an RNN

Recall that to train a NN, we follow a gradient-descent scheme:

$$\theta_{t+1} \quad \leftarrow \quad \theta_t - \kappa_t G_t$$

where G_t is (an approximation of) the gradient of the loss over the Neural Network parametrised with θ_t.

In the presence of loops (recurrence), the gradient of one RNN cell gets multiplied by itself at each loop. Simplifying a bit, it looks like:

$$\theta_{t+1} \quad \leftarrow \quad \theta_t - \kappa_t (G_t)^M$$

This will lead to exploding or vanishing gradients.

Two math points to understand: pt. 1

Take a function f and *compose* it with another function g: $h(x) = f(g(x))$. The gradient of that function is:

$$h'(x) \quad = \quad f'(g(x))g'(x)$$

Two math points to understand: pt. 1

Take a function f and *compose* it with another function g: $h(x) = f(g(x))$. The gradient of that function is:

$$h'(x) \quad = \quad f'(g(x))g'(x)$$

this is the chain rule .

Two math points to understand: pt. 1

Take a function f and *compose* it with another function g: $h(x) = f(g(x))$. The gradient of that function is:

$$h'(x) = f'(g(x))g'(x)$$

this is the chain rule .
Now if $g = f$ then you get a product of derivatives:

$$h'(x) = f'(f(x))f'(x)$$

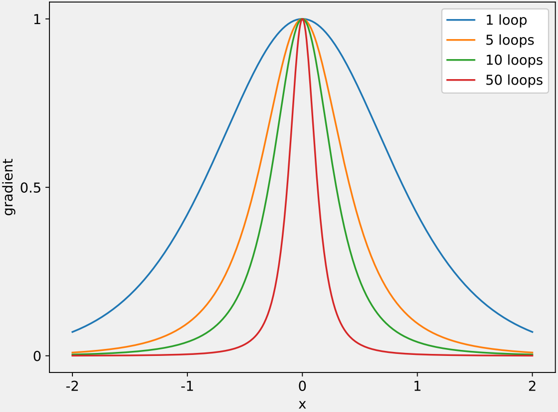
Tow math points to understand: pt. 2

If you take a positive number x to a power k with k increasingly large , you can have three situations:

1. $x < 1$ then $x^k \to 0$ (e.g. $0.5^5 \approx 0.03$ and $0.5^{10} \approx 0.001$)
2. $x = 1$ then $x^k = 1$
3. $x > 1$ then $x^k \to \infty$ (e.g. $2^5 = 32$ and $2^{10} = 1024$)

Tanh activation leads to vanishing gradient

Derivatives of $\tanh(\tanh(\ldots\tanh(x)))$



Dealing with exploding gradients

In more complex RNN architectures (i.e. not just a simple tanh), you could get both vanishing and exploding gradient components.

To deal with exploding gradients, you can clip or renormalise the gradient

Dealing with exploding gradients

In more complex RNN architectures (i.e. not just a simple tanh), you could get both vanishing and exploding gradient components.

To deal with exploding gradients, you can clip or renormalise the gradient

- clipping : setting every value of the gradient higher than some threshold to that threshold
- renormalising : dividing the gradient by its norm so that all elements are between 0 and 1

Dealing with exploding gradients

In more complex RNN architectures (i.e. not just a simple tanh), you could get both vanishing and exploding gradient components.

To deal with exploding gradients, you can clip or renormalise the gradient

- clipping : setting every value of the gradient higher than some threshold to that threshold
- renormalising : dividing the gradient by its norm so that all elements are between 0 and 1
- this is done automatically by Keras

Dealing with vanishing gradients

- vanishing gradients $\rightarrow$ cannot capture long-term dependencies

Dealing with vanishing gradients

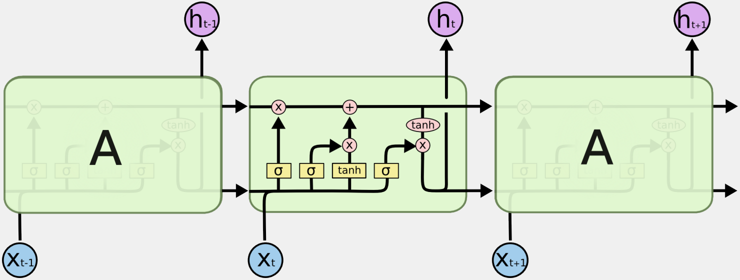
- vanishing gradients $\rightarrow$ cannot capture long-term dependencies
- the long short-term memory cell (LSTM) is a workaround introduced in the late ‘90s to try to keep track of both long and short term dependencies

Dealing with vanishing gradients

- vanishing gradients $\rightarrow$ cannot capture long-term dependencies
- the long short-term memory cell (LSTM) is a workaround introduced in the late ‘90s to try to keep track of both long and short term dependencies
- the LSTM *learns* the rate at which it should forget results of previous computations

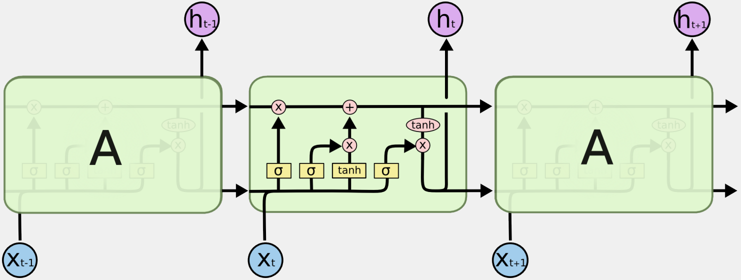
LSTM cell

There are variants, but a popular version looks like:



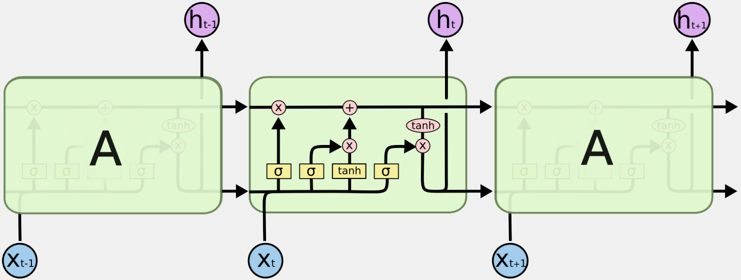
LSTM cell

There are variants, but a popular version looks like:



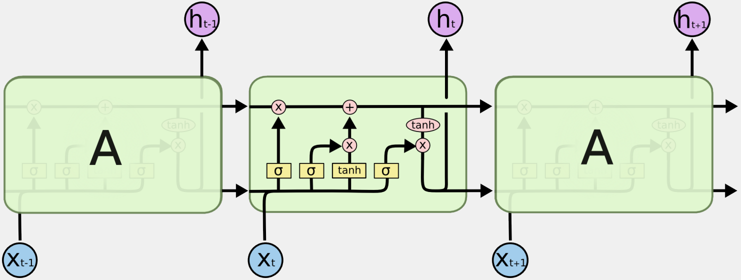
- yellow box = 1 layer of neurons with one type of activation (σ for logistic with $[0, 1]$ output, tanh with $[-1, 1]$ output)
- pink $\otimes$ for element-wise product of vectors
- pink $\oplus$ for element-wise addition of vectors

LSTM cell



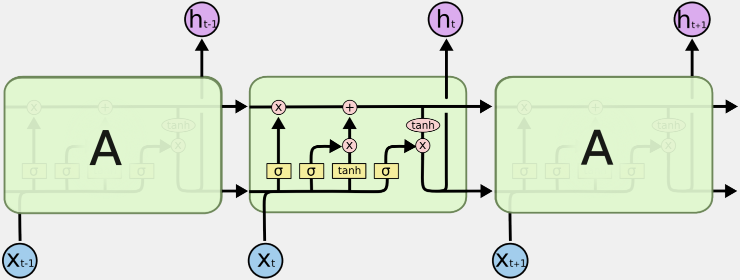
- A σ followed by a $\otimes$ is a *gate* : it multiplies a current vector by a vector with components between 0 and 1 thereby letting some elements pass (when close to 1) or inhibiting others (when close to 0)

LSTM: key components (think about a sentence)



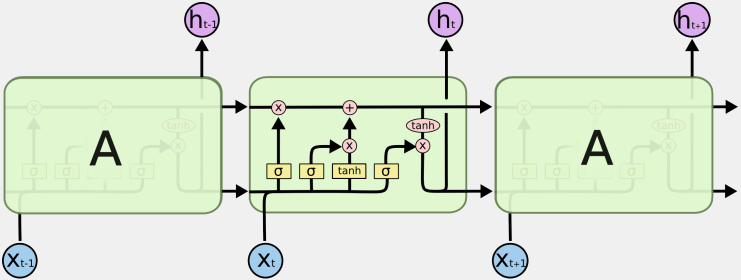
- the top horizontal arrow keeps track of long-term effects (e.g.: subject of the current sentence)

LSTM: key components (think about a sentence)



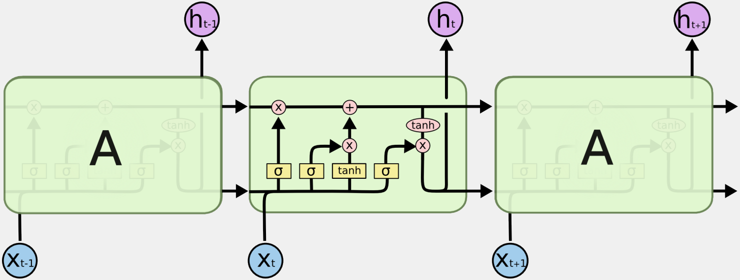
- the top horizontal arrow keeps track of long-term effects (e.g.: subject of the current sentence)
- the left vertical arrow indicates how much should be forgotten given the current input and short-term memory (e.g.: new subject)

LSTM: key components (think about a sentence)



- the middle vertical arrow updates the long-term effect given the current input and short-term memory

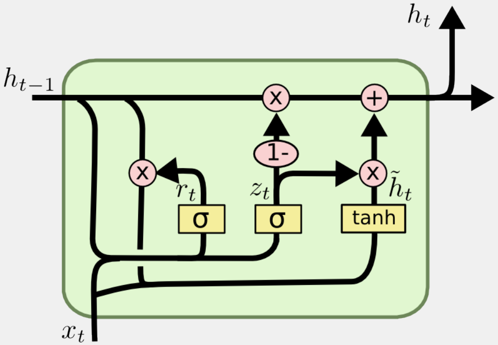
LSTM: key components (think about a sentence)



- the middle vertical arrow updates the long-term effect given the current input and short-term memory
- the last part on the right computes the new short-term memory (and output of the cell)

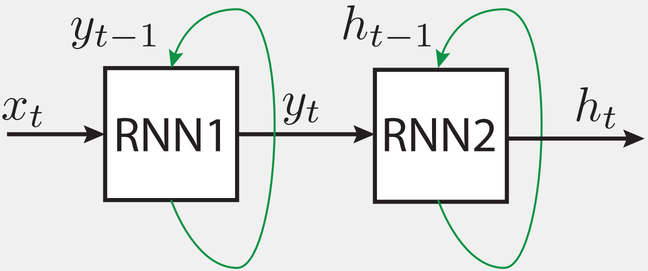
GRU cell

Another popular architecture with similar effects is the Gated Recurrent Unit (2014) which can work well and has fewer parameters than the LSTM



Deep RNNs

You can use intermediate outputs of RNNs and feed them into another batch of RNNs etc. It can work well but it's typically very hard to tune .



Regularisation with RNNs

RNNs are notoriously hard to tune but common techniques apply:

- component regularisation (ℓ_1, ℓ_2), this can help but tends to slow down the learning (need more epochs)
- dropout can be quite efficient as well

Regularisation with RNNs

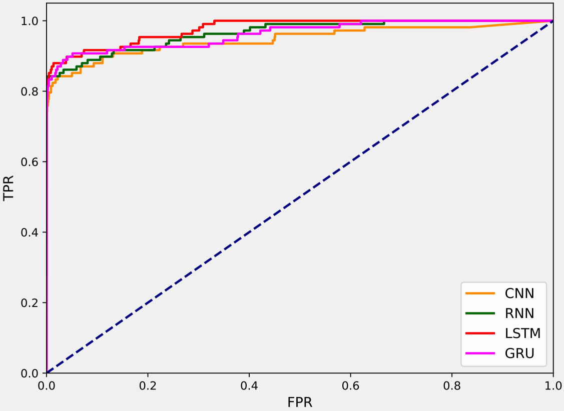
RNNs are notoriously hard to tune but common techniques apply:

- component regularisation (ℓ_1, ℓ_2), this can help but tends to slow down the learning (need more epochs)
- dropout can be quite efficient as well
- best is to try and see for yourself! (and ask questions)

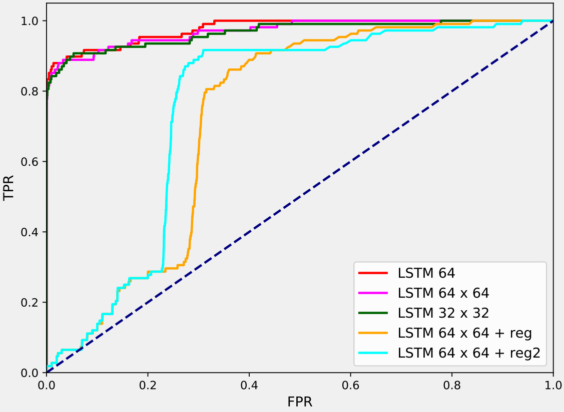
Hands-on session

>>> RNNs, LSTMs and regularisation

(sol) CNN v RNN v LSTM v GRU



(sol) LSTM 64 v 64x64 v 32x32 v 64x64+reg



(sol) LSTM causal vs LSTM bidir