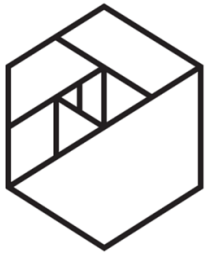


Neural networks

INTRODUCTION TO DATA SCIENCE – FALL 2018

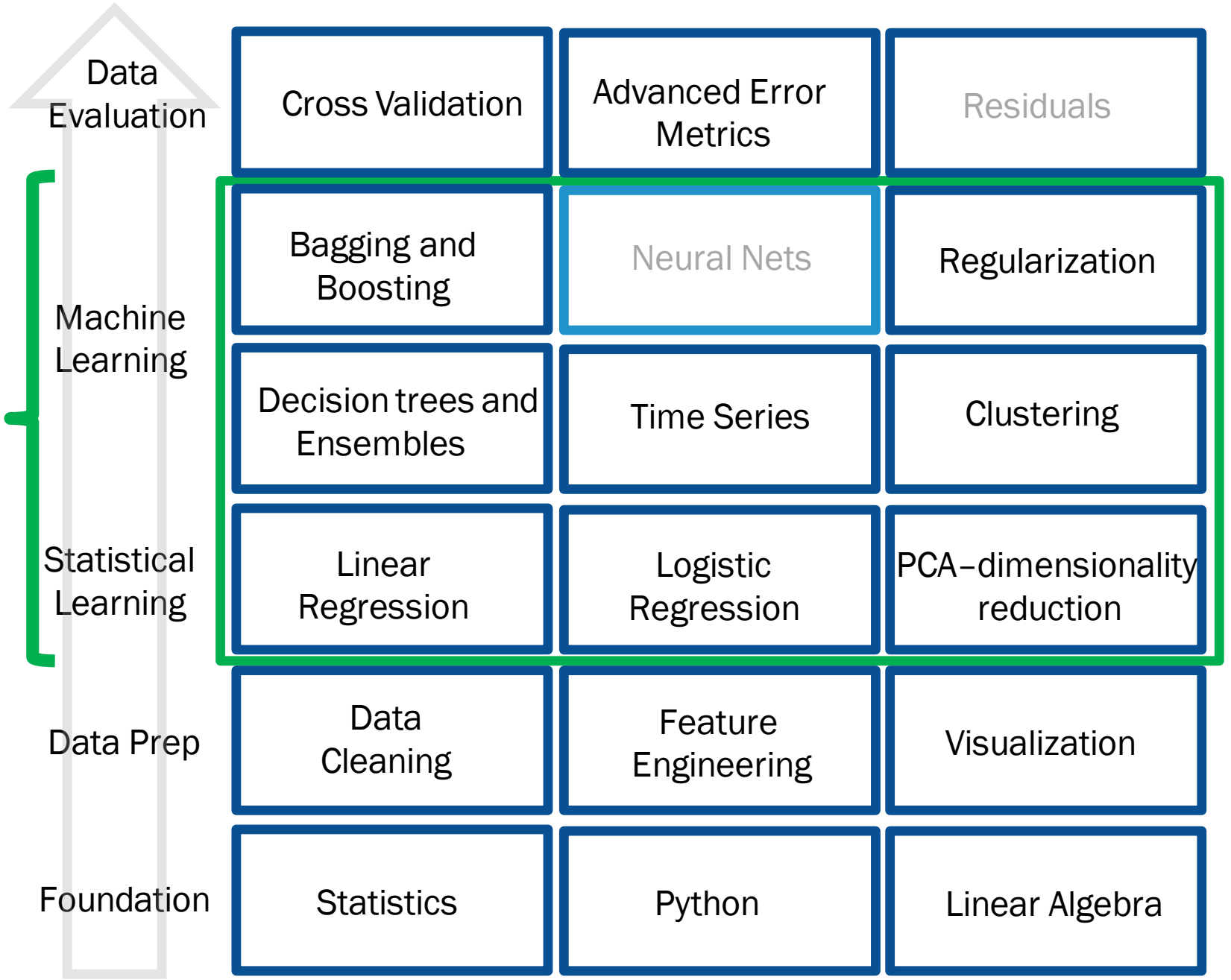
SESSION 11

AGENDA

1. Artificial neural nets
2. Convolutional neural nets
3. Recurrent neural nets

Introduction to Data Science

- Learning the steps in the Data Science Process
- Learning multiple model methodologies

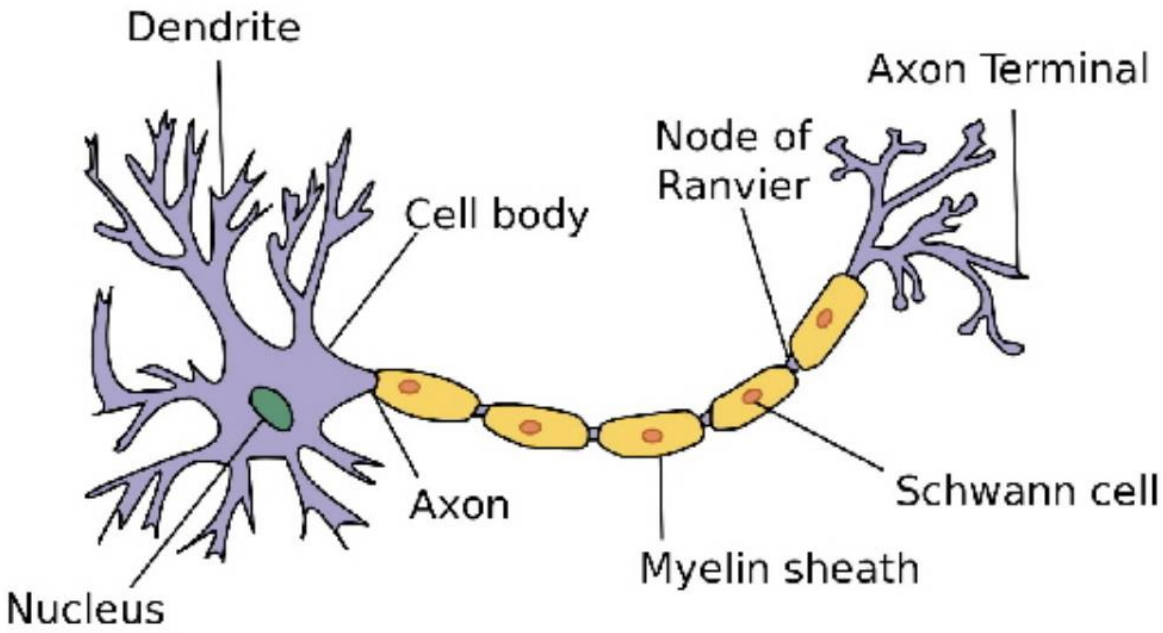


Artificial neural nets

TECHNOLOGY'S VERSION OF A NEURON

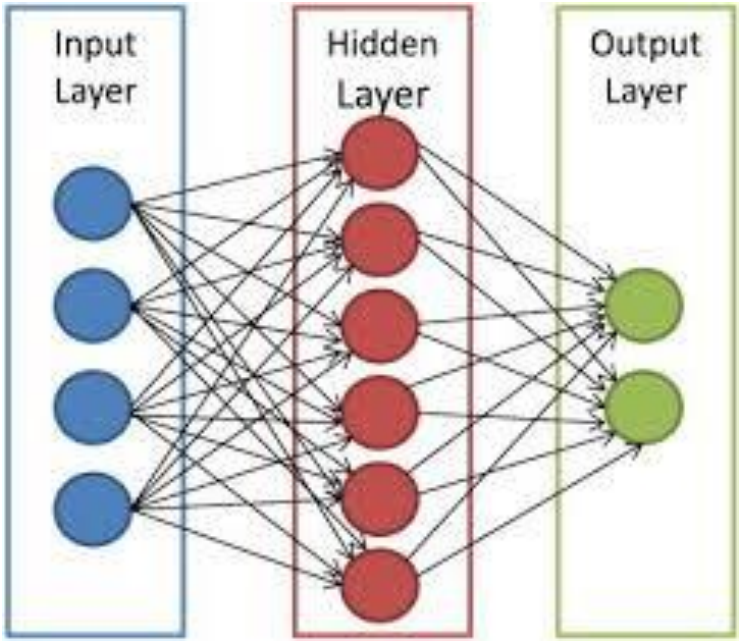
Neural nets are based on human neurons

- Dendrite receives information from the previous neuron and transfers it to the Cell body
- The Cell body contains the Nucleus,
- Axon conducts signals away from the cell body to the next dendrite



Artificial neural networks

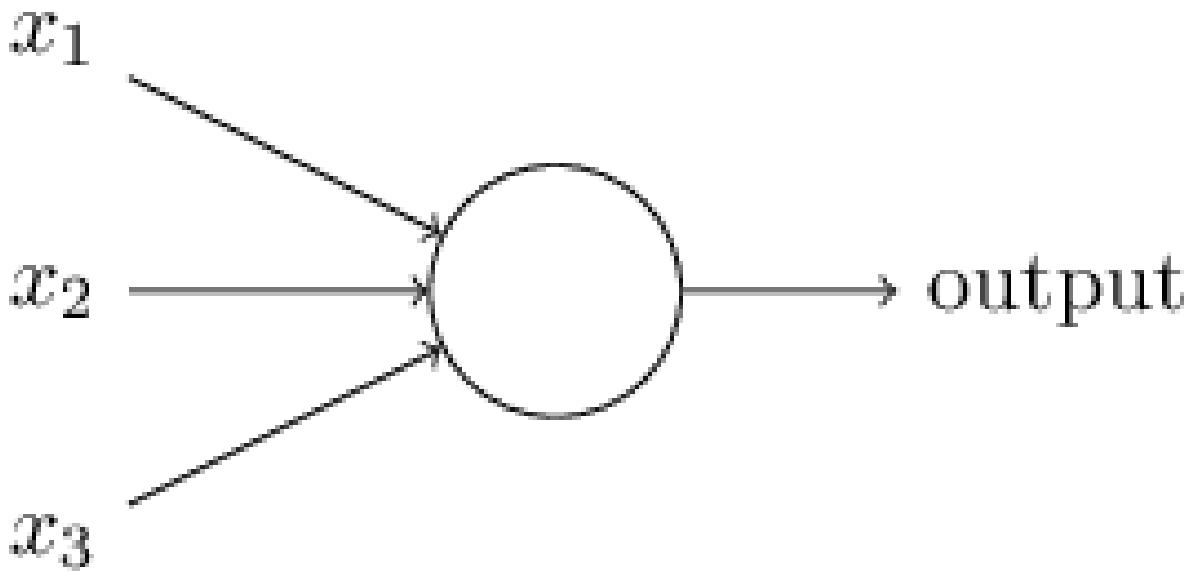
- Designed to mimic the work of biological neurons
- These methods are used to estimate or approximate functions of large number of inputs, detect patterns or classify observations
- ANNs are parameterized with coefficients (weights) that are estimated and tuned during training



Interconnected and exchanging information

Single layer perceptron

- Developed in 1950s by Frank Rosenblatt
- Takes several binary outputs and produces a single binary output (linear classifier)
- Weights express the importance of inputs to the output
- The neuron's output is 0 or 1



Structure of a single layer perceptron

Perceptron learning algorithm

1. Initialize the weights
2. Present an input vector X and calculate output y
3. Update the weights according to

$$w_j(t+1) = w_j(t)+\eta(d-y)x$$

d is the desired output
t is the iteration number
eta is the learning rate

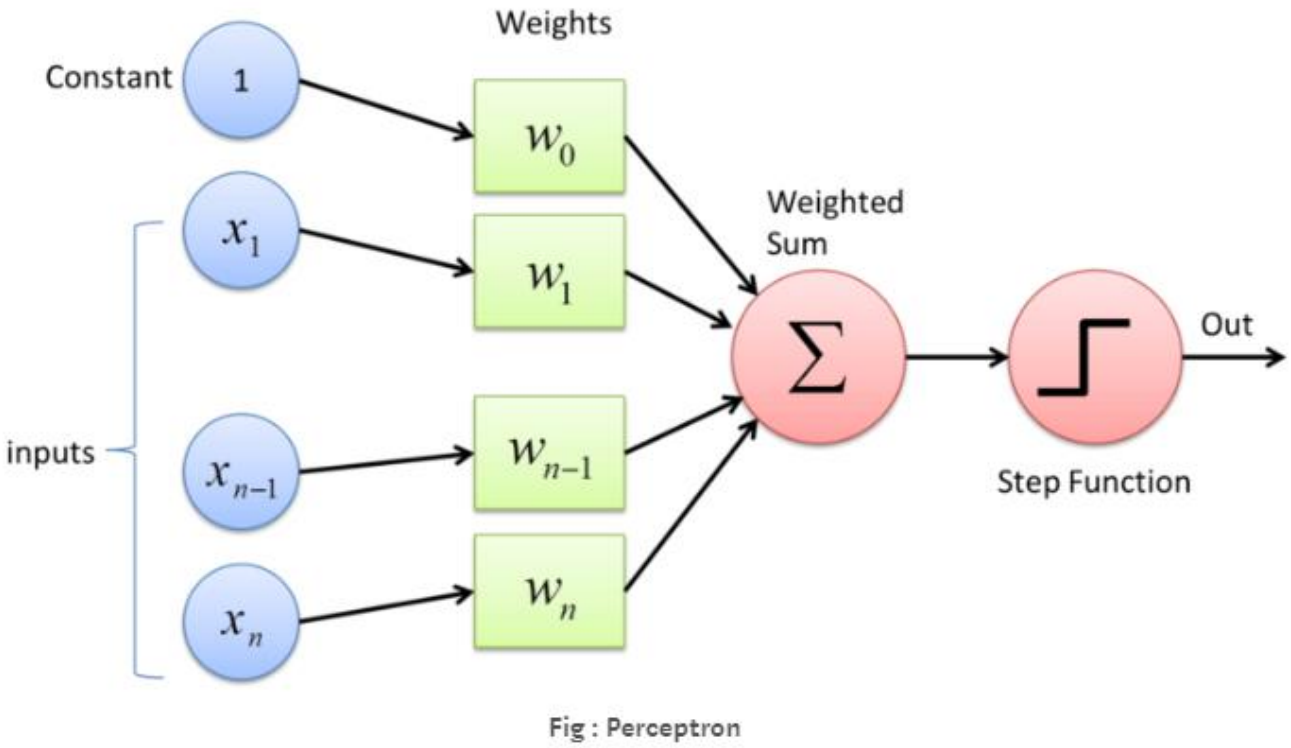
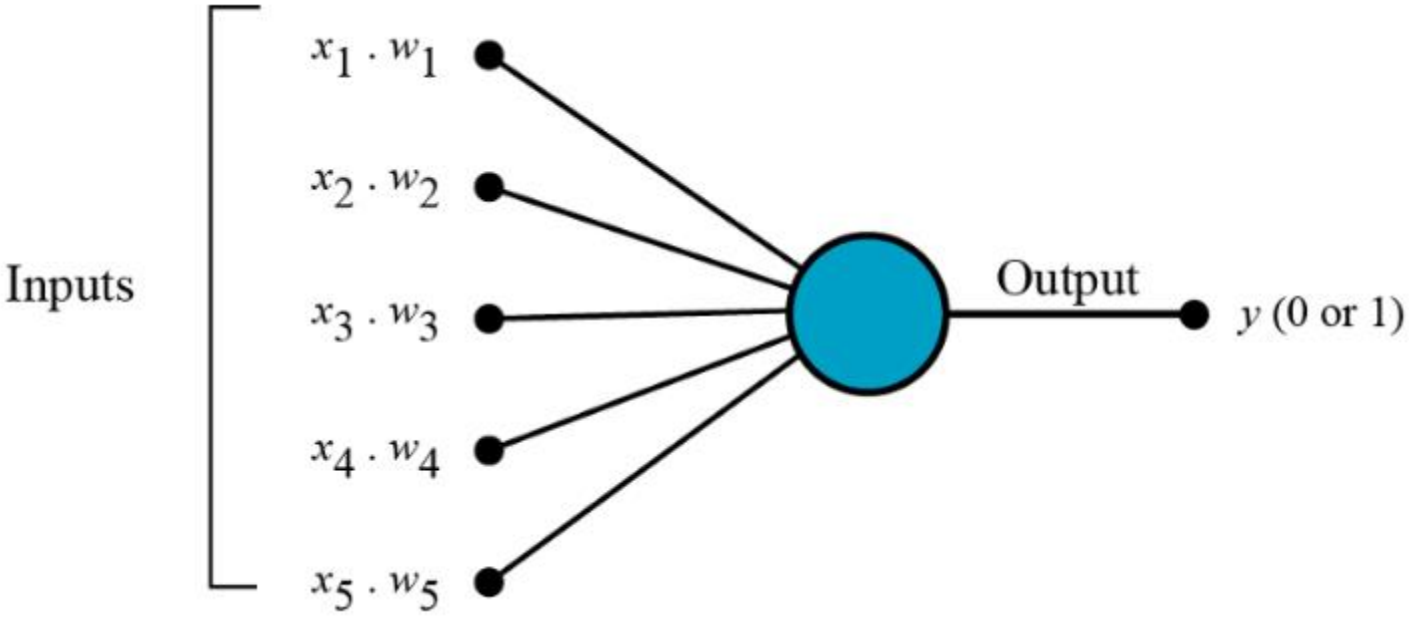


Fig : Perceptron

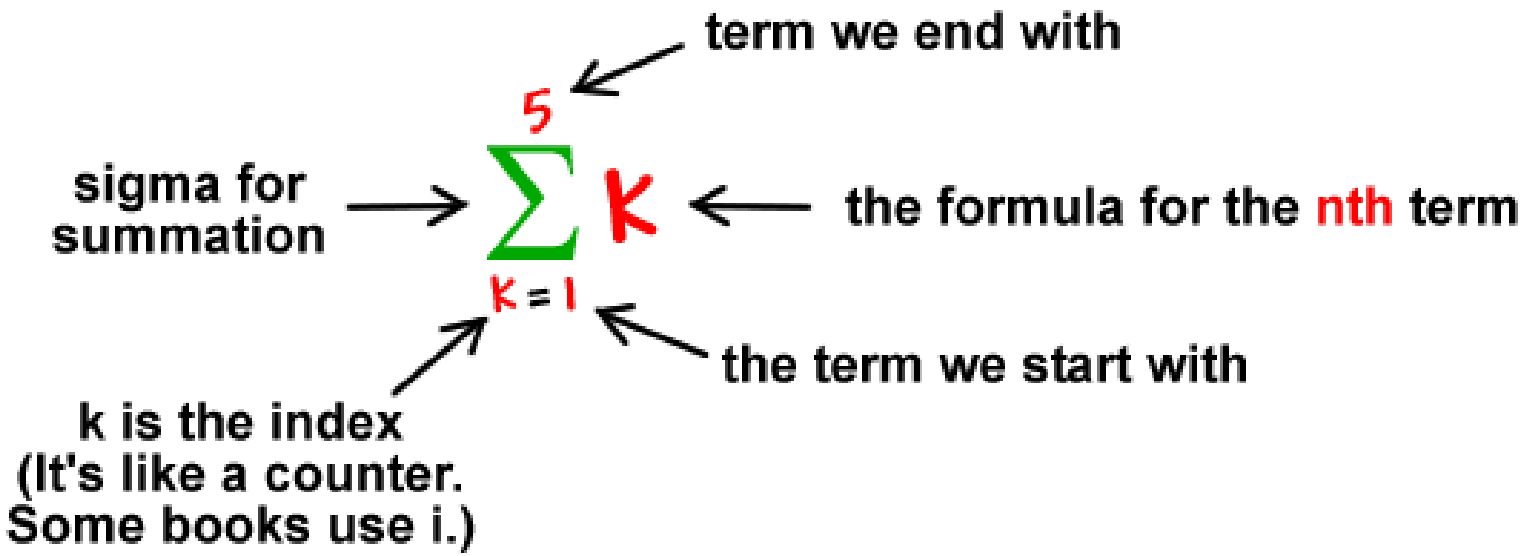
Step 1 – multiply weights

- All the inputs **x** are multiplied with their weights **w**
- Weights show the strength of the particular node
- Bias value allows you to shift the activation function curve up or down



Step 2 – weighted sum

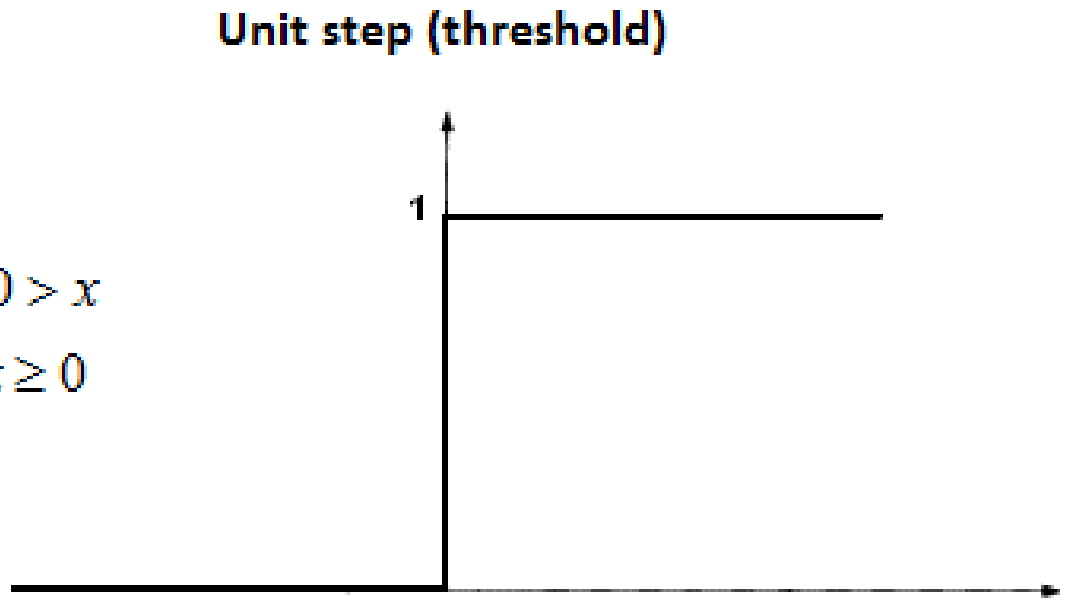
Add all the multiplied values



Step 3 – activation function

- Apply that weighted sum to the correct Activation Function
- Used to map the input between the required values like (0,1) or (-1,1)

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$

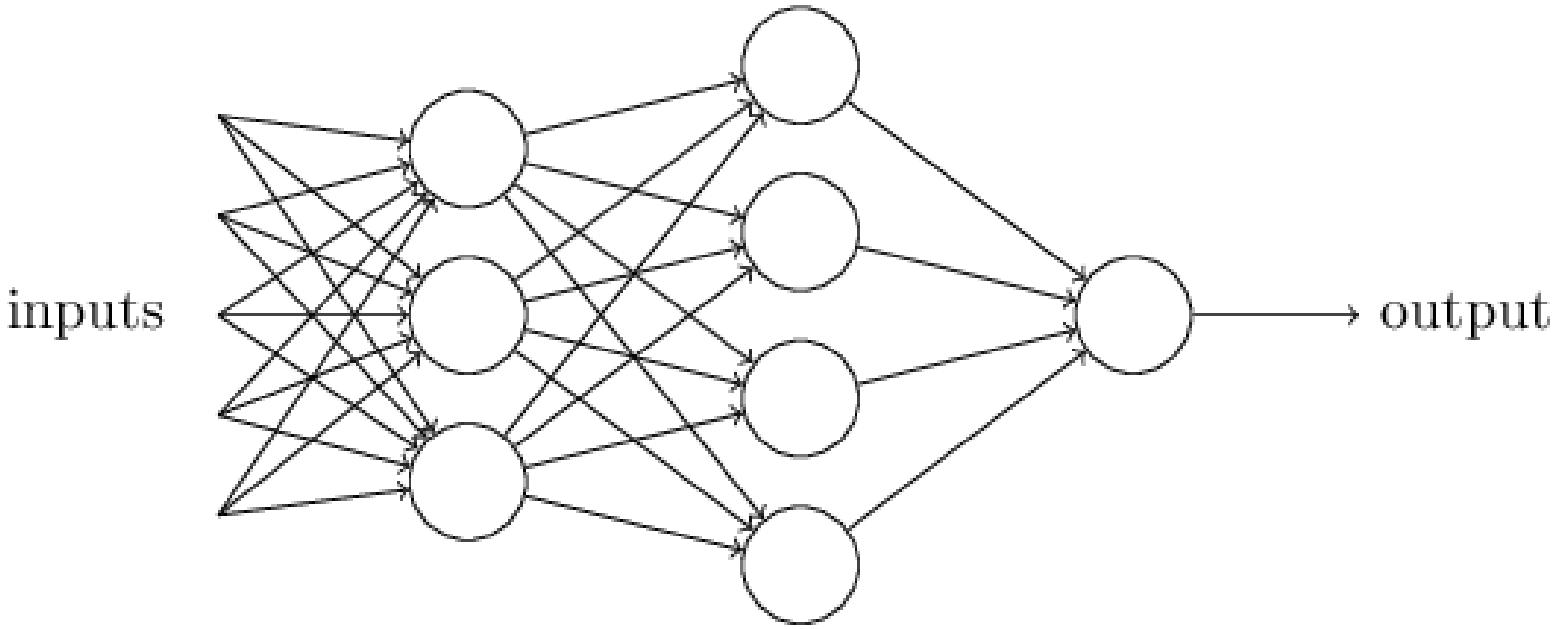


Feedforward Neural Networks

- Historically the first and simplest type of artificial neural network
- The information moves in only one direction
- Single perceptron has limited learning ability, but a **single layer of parallel units** can learn any continuous function from any compact interval into [-1,1]
- A network of **several layers of units** has more flexibility. Can approximate any map of real interval [a,b] into real interval [c,d]

Multi-layer perceptron

- First column (layer) is making three simple decisions by weighing the inputs
- Second layer is making a decision based on the outputs of the first layer



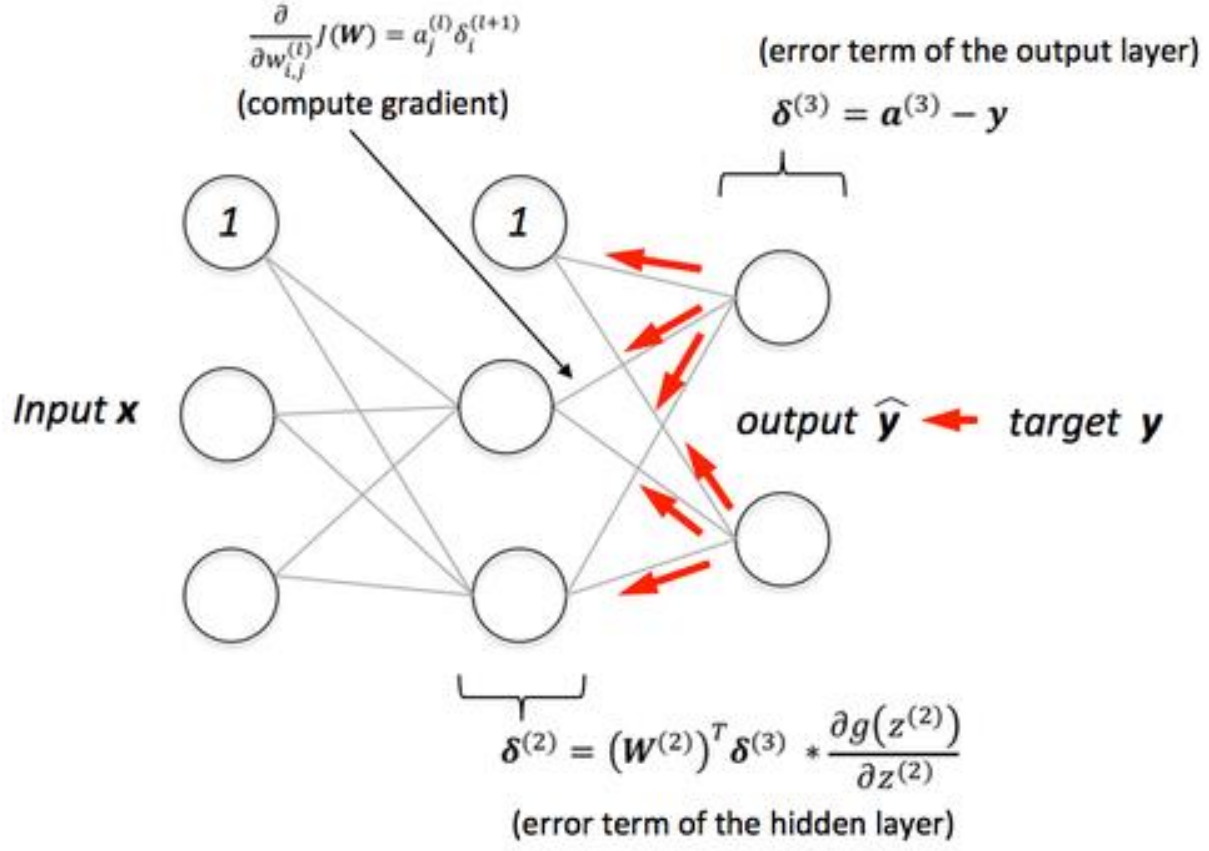
Structure of a multi-layer perceptron

Backpropagation

- Means the backward propagation of errors, this is a method of training multilayer feedforward artificial neural networks
- The method is based on calculation of the gradient of a loss function, with respect to coefficients of the network
- The gradient is returned to the learning process and used by it to find a new iteration of the coefficients that improve the fit
- Backpropagation is a supervised learning method: it requires knowing a output for training sample

Backpropagation Algorithm

1. Propagation
2. Weight adjustment
3. Repeat until convergence



Step 1 - propagation

1. Propagate the given input forward through the network to obtain the predicted output
2. Calculate the loss function as a measure of distance between the true and predicted output
3. Propagate the loss back to find loss associated with each synapse

Step 2 - weight adjustment

1. For each unit use the adjustment rule

$$w_j(t) = w_j(t-1) + \eta(d-y)x_j$$

Where w is the weight number ***j*** of the network associated with the unit, ***t*** is the iteration, ***d*** is the target output for the unit, ***y*** is the predicted output for the unit, ***x*** is the input for the unit, and ***eta*** is the "learning rate", coefficients showing the speed of the adjustment

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Moving toward deep learning

- Early 21st century has revolutionized data science with the significant advances in technology, computer science and the application of tensor flow calculations
- Google has played the leading role by developing major concepts and opening their projects to the public, like MapReduce (Spark) and TensorFlow
- TensorFlow is based on very efficient C++ library including sophisticated optimization techniques. Very powerful when working with neural networks

Activation functions

- Step function from Rosenblatt's perceptron had serious weaknesses
  - 1969 Minsky-Papert article proved it couldn't solve XOR

$$(0,0) \rightarrow 0,\ (1,1) \rightarrow 0,\ (0,1) \rightarrow 1,\ (1,0) \rightarrow 1$$

- Backpropagation method from 1970s revived optimism
  - Logistic function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

  - Hyperbolic tangent function

$$\tanh(z) = 2\sigma(z) - 1$$

Challenges of deep learning

- Increasing the number of layers in neural nets brought new challenges
- Most important of these are
  - Vanishing gradients and exploding gradients
  - With number of layers, complexity of connections between them grows tremendously which slows down the training process
  - Increased number of parameters increases the risk of overfitting

Vanishing or exploding gradients

- Gradients have tendency to either vanish or explode as they get to the lower layers, close to the input
  - Vanishing gradients – leave weights unchanged, slowing convergence
  - Exploding gradients – makes too large updates, divergence of algorithm
- In general, deep networks tend to have a problem of unstable gradients meaning that different layers learn at different speeds

Overcoming instability of gradients

- In 2010, Xavier Glorot and Yoshua Bengio showed that the main reasons for the instability of gradients are the shape of the sigmoid function and commonly used practice of initiation of weights using the standard normal distribution.
- The variance of outputs of each layer is much larger than the variance of its inputs
- Layer outputs saturate in areas of sigmoid function where gradient is zero. When backpropagation starts there is not enough gradient for all layers to adjust weights sufficiently.
- Series of recommendations became known as Xavier and He Initialization

Xavier and He initialization

- Variance of outputs is kept as close as possible to variance of inputs at each layer
- Variance of gradients in reverse flow of back propagation is also kept as constant as possible before and after each layer
- Sigmoid activation function is more commonly replaced with ReLu

$$h_{w,b}(x) = \max(x \cdot w + b, 0)$$

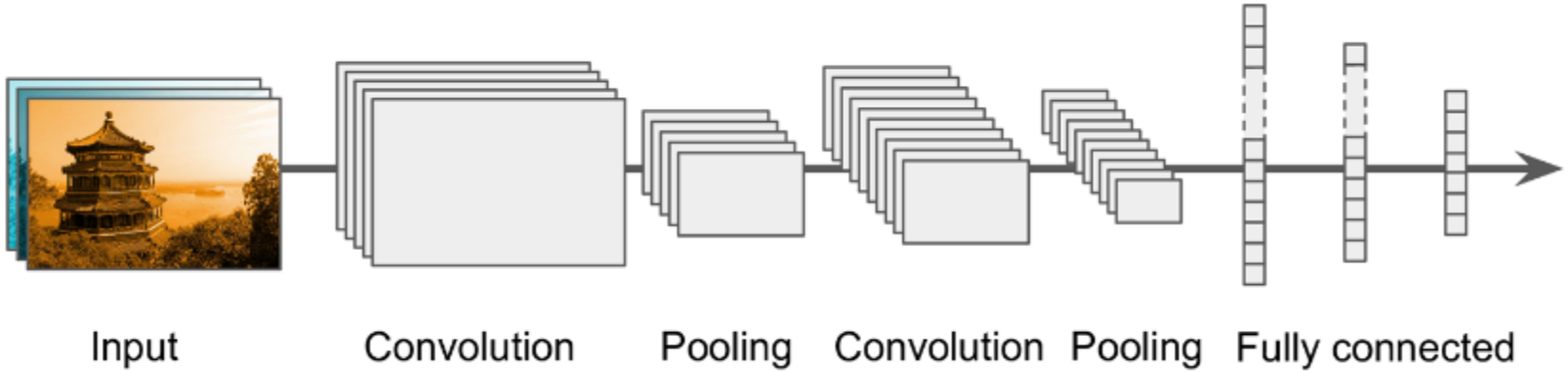
- Not perfect because it is flat for negative arguments
- Another measure increasing stability is Batch Normalization
  - Each batch of observations gets normalized to avoid saturation

Convolutional neural nets

IMAGE RECOGNITION

Applications for computer vision

CNN architecture



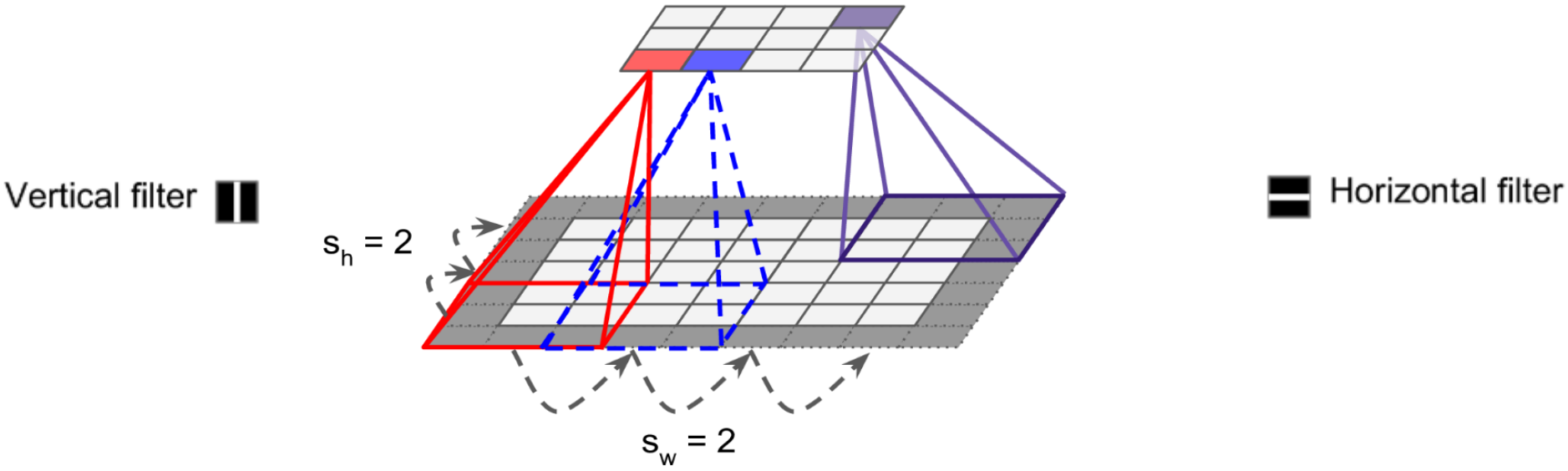
Input: The image is composed of pixels and possibly color channels (RGB). The image gets smaller and smaller as it progresses through the network, also deeper and deeper

Convolution: Creating feature maps using a hierarchical scanning process

Pooling: Downsizing feature maps by one or several pooling kernels, like max or mean

Fully Connected: Regular feedforward neural network, final layer (softmax) estimates class probabilities

Convolutional layer



Stride: Distance between two consecutive receptive fields (horizontal and vertical)

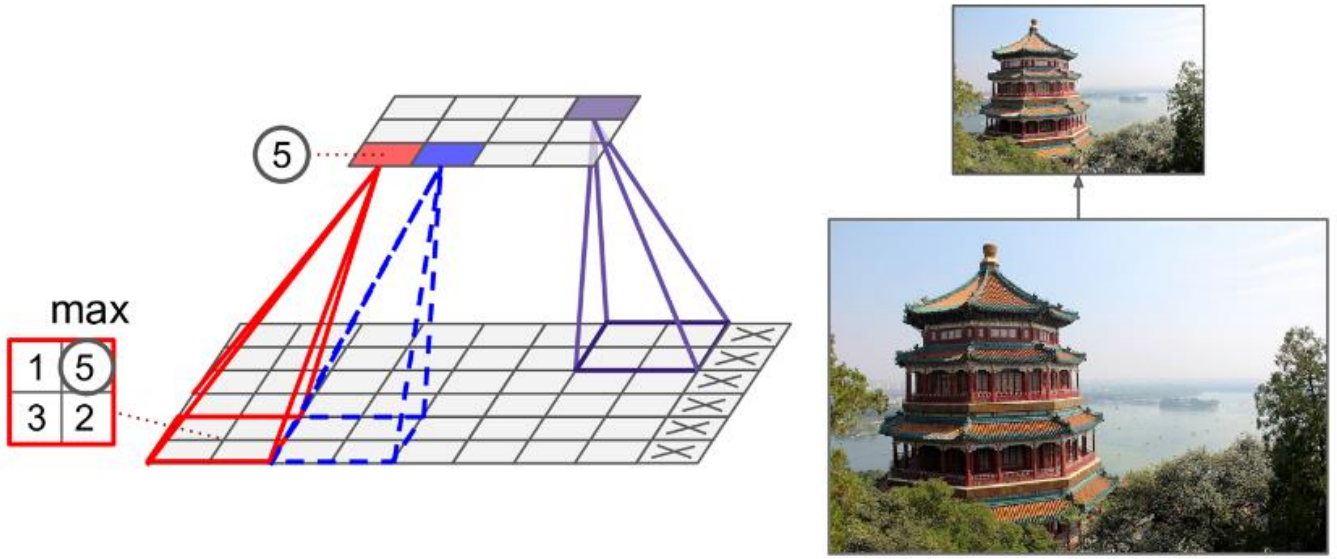
Zero Padding: Adding zeros around the inputs

Neuron: Only connected to pixels in their receptive fields

Filter (convolutional kernel): Neuron's weights, same across the entire filter. Typically learned during training, can be combined into more complex patterns.

Feature Maps: Resulting data from use of the filter

Pooling layer



- Subsample the input image in order to reduce the computational load, memory usage and number of parameters (reduce risk of overfitting)
- Pooling also requires size and stride parameters.
- Works to aggregate data. Common methods are max or mean pooling

Recurrent neural nets

IMAGE RECOGNITION

RECURRENT NEURAL NETWORKS

Used to examine data where time or sequence matter

- Forecasting based off a time series
- Text or Speech data
- Classification or regression

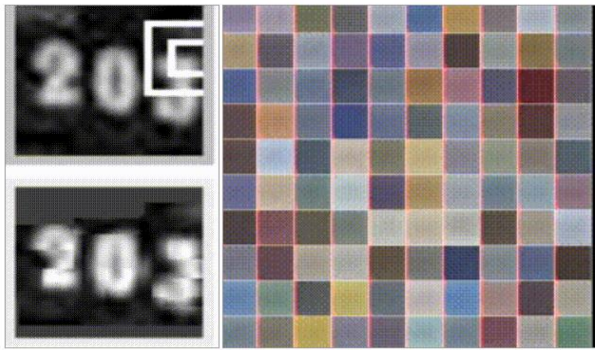
Image data input

Tabular datasets/csv

- https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

RECURRENT NEURAL NETWORKS

Used to examine data where time or sequence matter



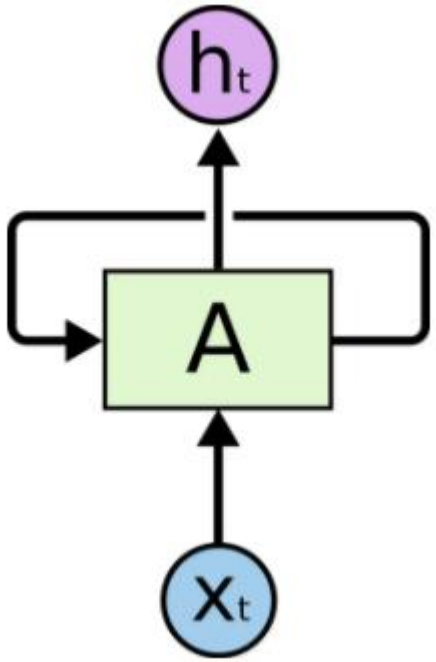
- Sequence to sequence (financial time series)
- Sequence to vector (document to a vector of sentiment scores)
- Vector to sequence (image results in a sequence of features)
- Sequence to vector then vector to sequence (encoder/decoder)
- Sentence in one language to a vector representation, then decoder converts vector into a sequence of words in another language

- https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/

RECURRENT NEURAL NETWORKS

Looks like usual FFN but has loops

- x_t is an input
- h_t is an output



$$h_t = W_t \times X_t + b$$

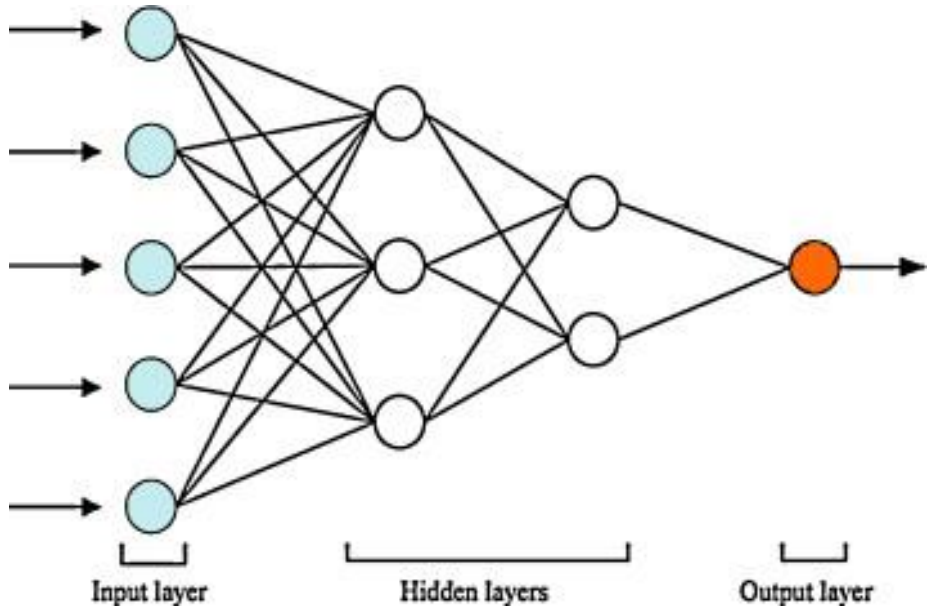
- The state (loop) allows information to be passed from one step of the network to the next

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

TERMS AND CONCEPTS

Before we dig in

- **Neural Network Structure**
- Activation functions
  - Sigmoid
  - Tanh
- Time Series Data
- Matrix/vector multiplication

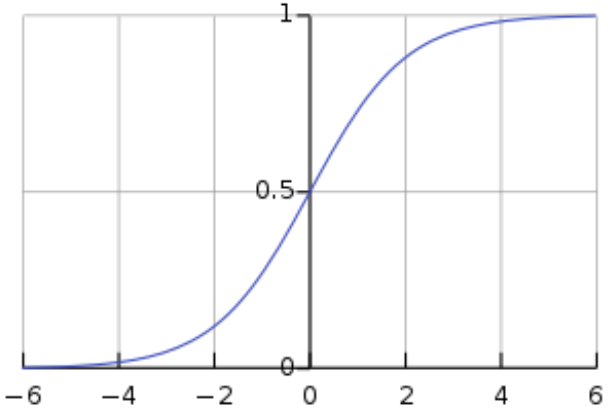


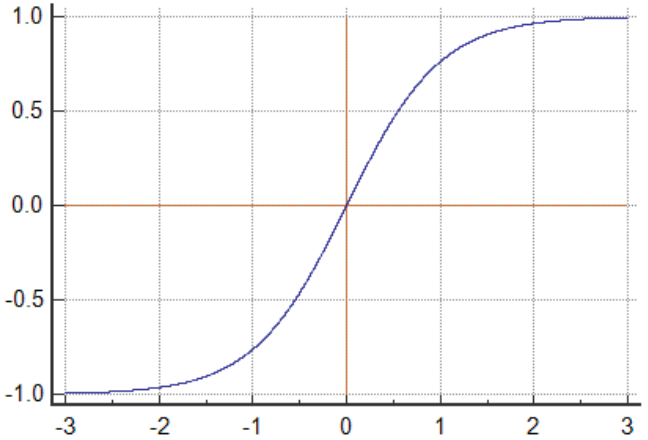
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

TERMS AND CONCEPTS

Before we dig in

- Neural Network Structure
- **Activation functions**
  - Sigmoid
  - Tanh
- Time Series Data
- Matrix/vector multiplication

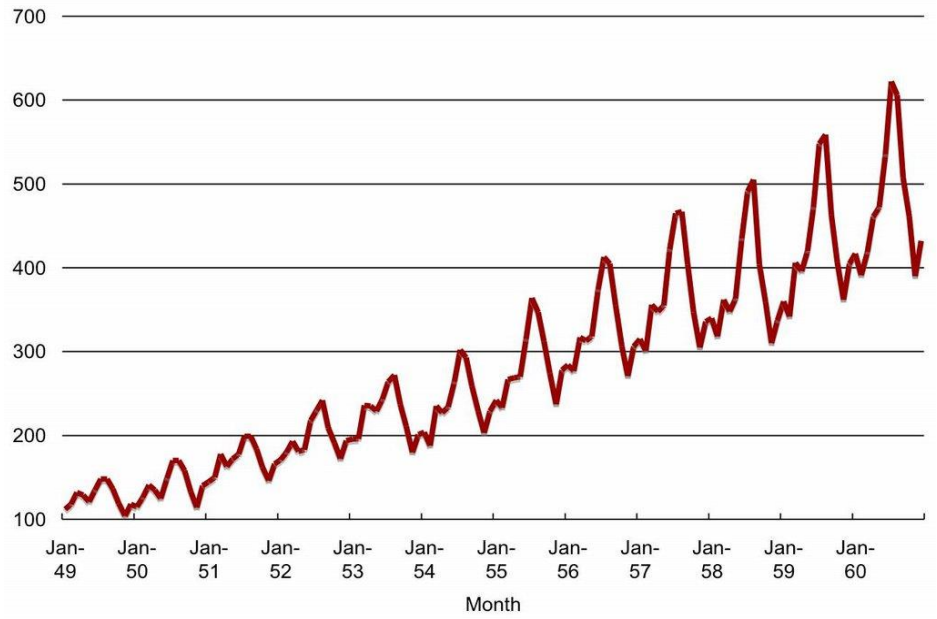




- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

TERMS AND CONCEPTS

- Neural Network Structure
- Activation functions
  - Sigmoid
  - Tanh
- **Time Series Data**
- Matrix/vector multiplication



	bankname	bank	year	quarter	quarters	beta	leverage
178	Sparebank SMN	3	2011	4	2011q4	.7119	12.9143
179	Sparebank SMN	3	2012	1	2012q1	.0361	12.528
180	Sparebank SMN	3	2012	2	2012q2	.6157	12.3613
181	Sparebank SMN	3	2012	3	2012q3	.3987	12.5357
182	Sparebank SMN	3	2012	4	2012q4	.4382	11.5395
183	Sparebank SMN	3	2013	1	2013q1	.804	11.436

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

TERMS AND CONCEPTS

Before we dig in

- Neural Network Structure
- Activation functions
  - Sigmoid
  - Tanh
- Time Series Data
- **Matrix/vector multiplication**

$$\begin{bmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}$$

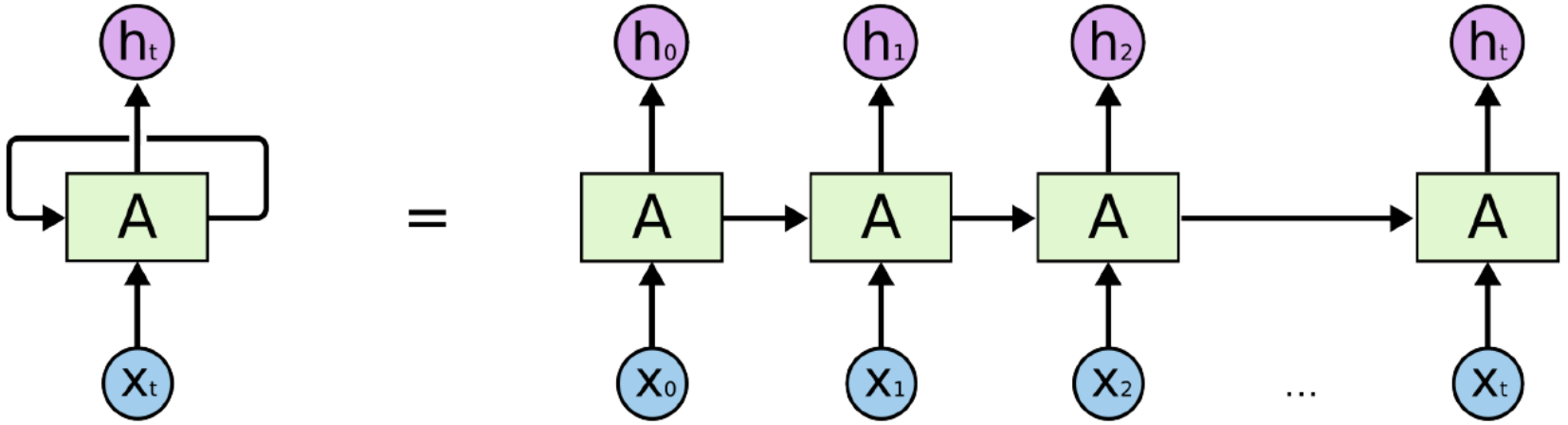
$$a_x = a_1 x_1 + a_2 x_2 + a_3 x_3$$

$$a_y = a_1 y_1 + a_2 y_2 + a_3 y_3$$

$$a_z = a_1 z_1 + a_2 z_2 + a_3 z_3$$

- https://colah.github.io/posts/2015-08-Understanding-LSTMs/

UNROLLING THE NEURON THROUGH TIME

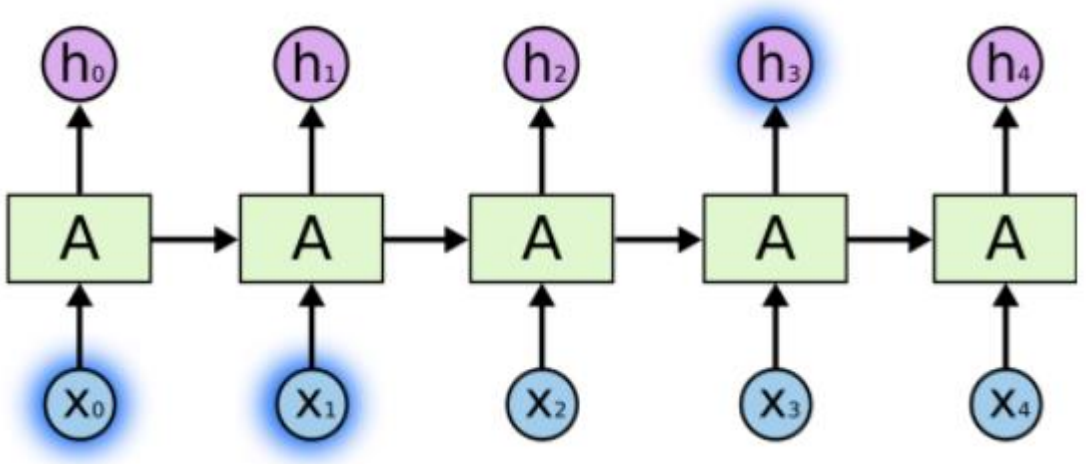


- Recurrent neuron is like multiple copies of the same neuron, which pass information to each other
- The neuron receives two inputs at each time t

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

PROBLEMS WITH LONG-TERM DEPENDENCIES

Predicting language – Short example

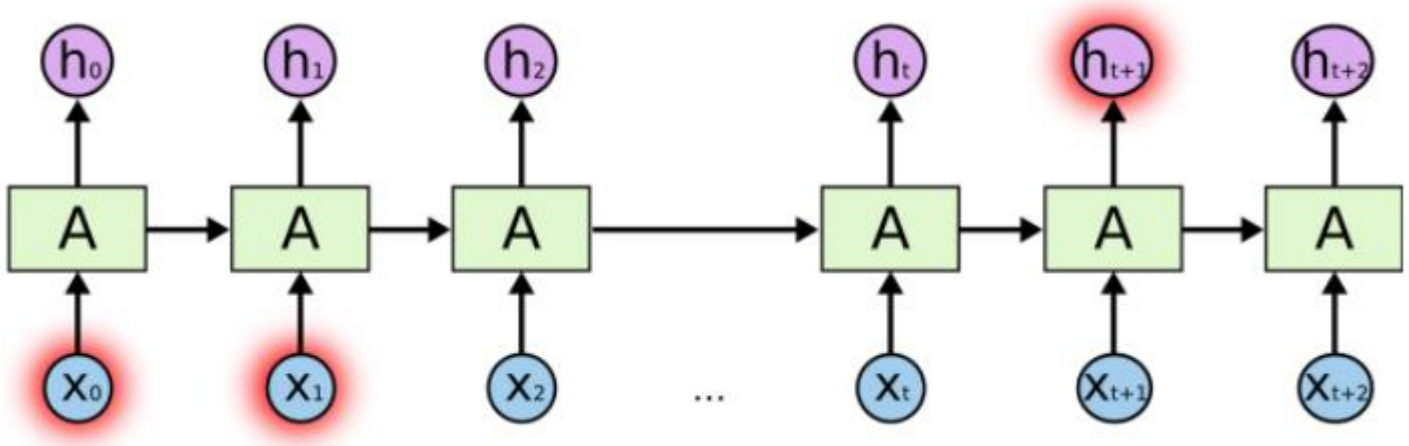


"the clouds are in the *sky*"

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

PROBLEMS WITH LONG-TERM DEPENDENCIES

Predicting language – Long example



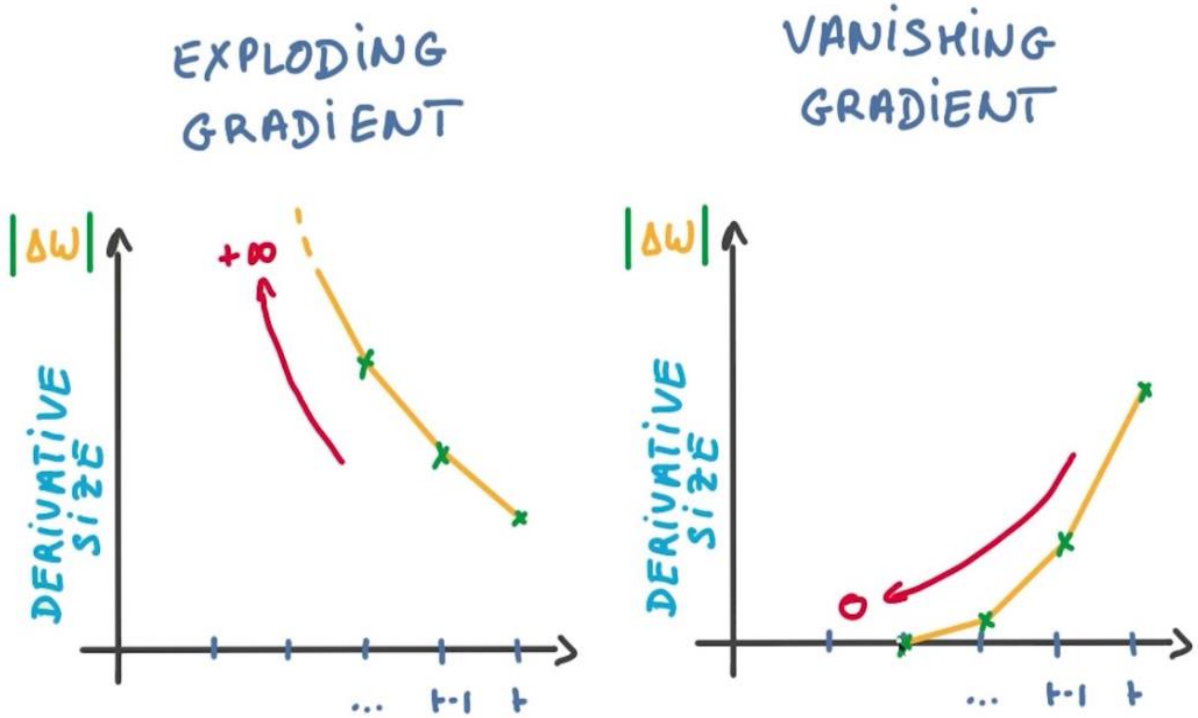
"I grew up in France…I speak fluent *French*"

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

PROBLEMS WITH LONG-TERM DEPENDENCIES

Exploding/vanishing gradients

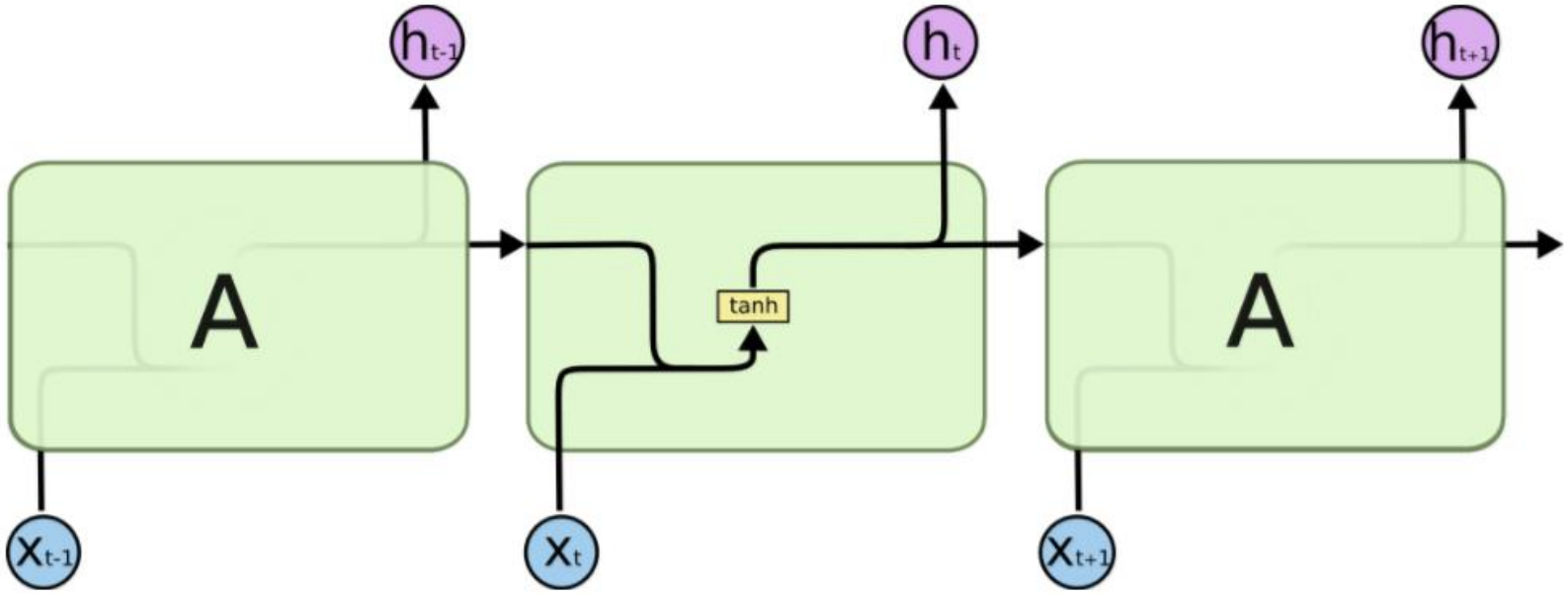
- 1991 Sepp Hochreiter
  - *Fundamental Deep Learning Problem*
- 1997 Sepp Hochreiter
  - *Long Short-Term Memory*



http://people.idsia.ch/~juergen/fundamentaldeeplearningproblem.html
https://www.bioinf.jku.at/publications/older/2604.pdf

LSTM Networks

Special RNNs that can learn long-term dependencies

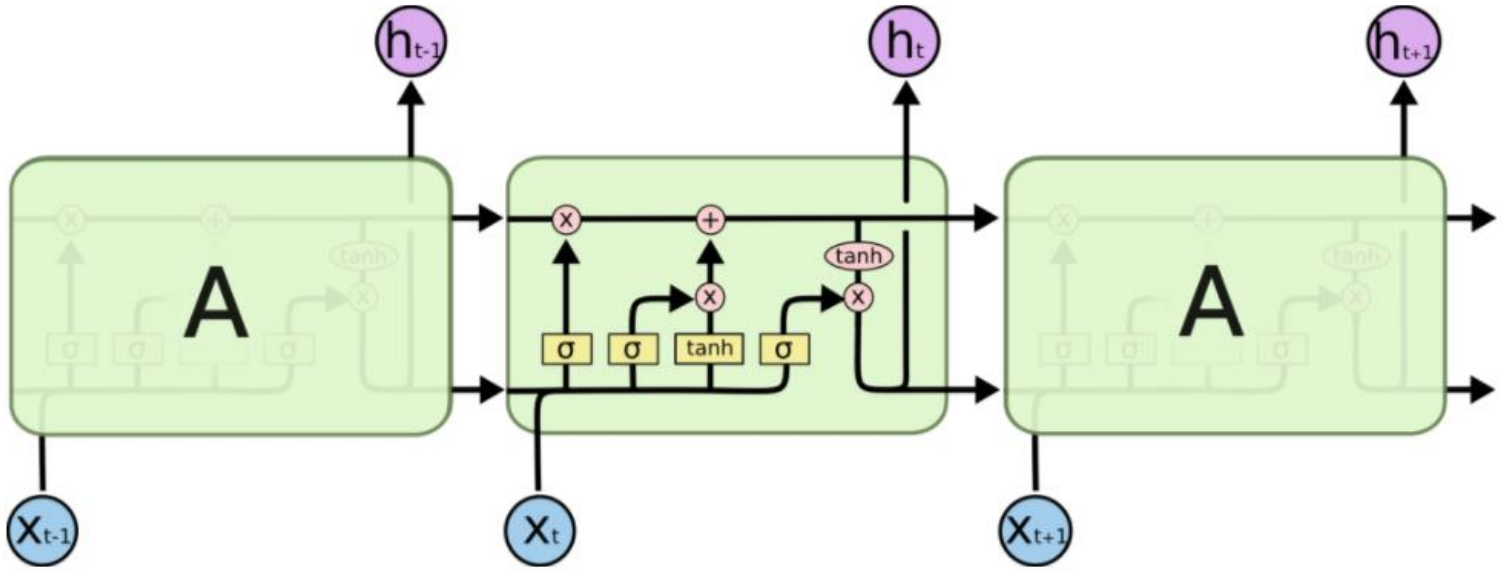


The repeating module in a standard RNN contains a single layer.

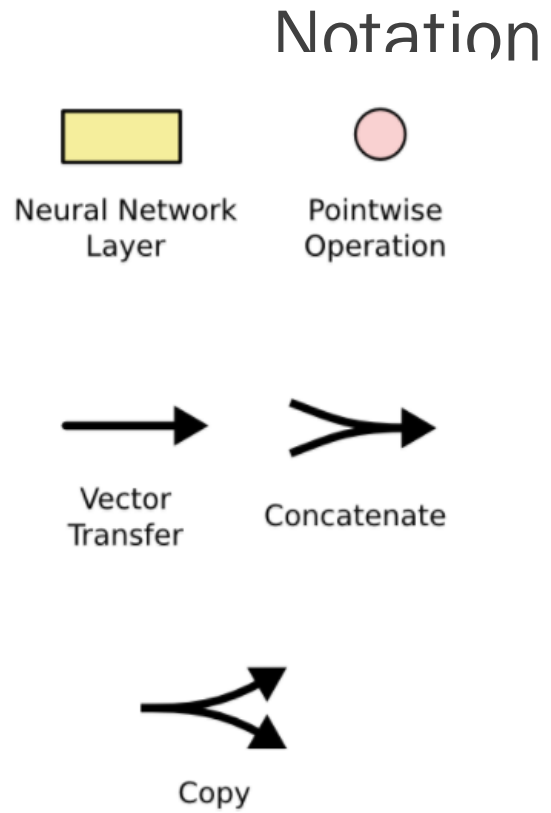
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Networks

Special RNNs that can learn long-term dependencies



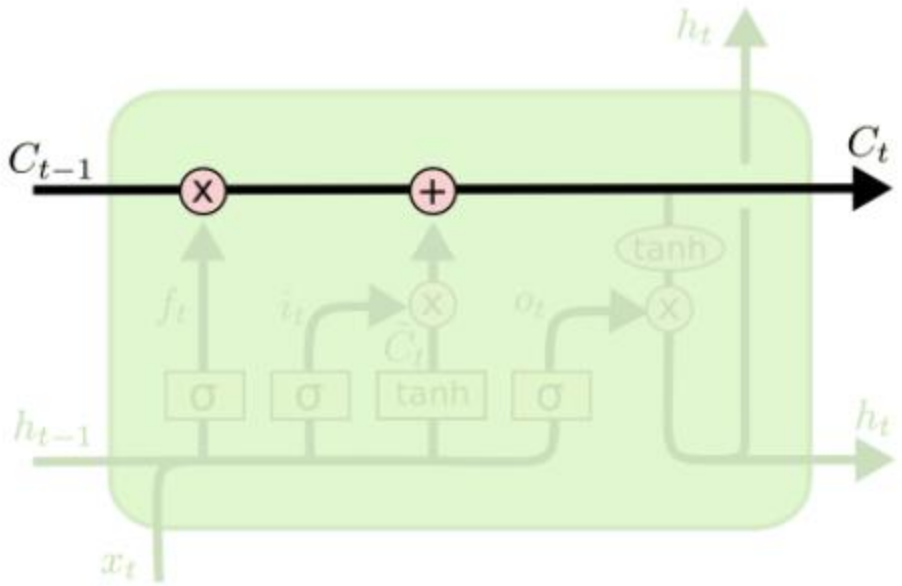
The repeating module in an LSTM contains four interacting layers.



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

THE KEY IS THE CELL STATE

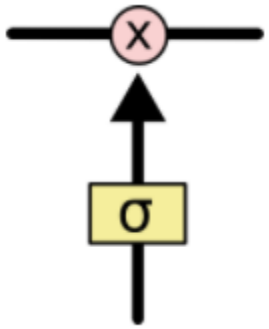
A conveyor belt of information



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

GATES REGULATE INFORMATION

An LSTM has three gates to affect the cell state

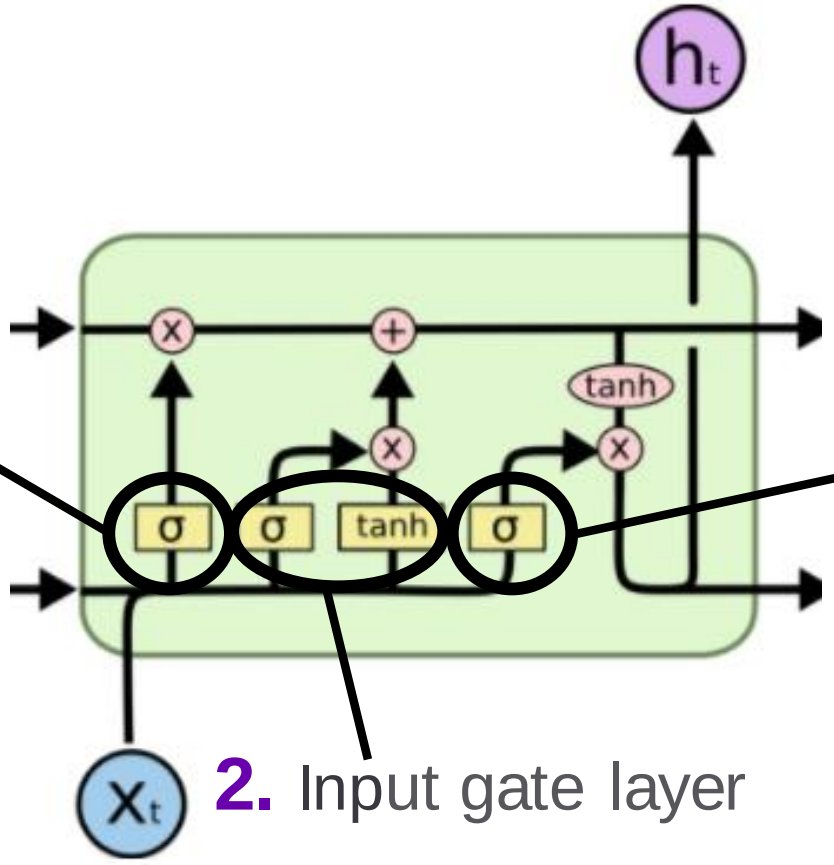


- Gates are composed of a sigmoid neural net layer and a pointwise multiplication operation
- The output of a sigmoid layer is a number between zero and one

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

GATES REGULATE INFORMATION

An LSTM has three gates to affect the cell state



1. Forget gate layer

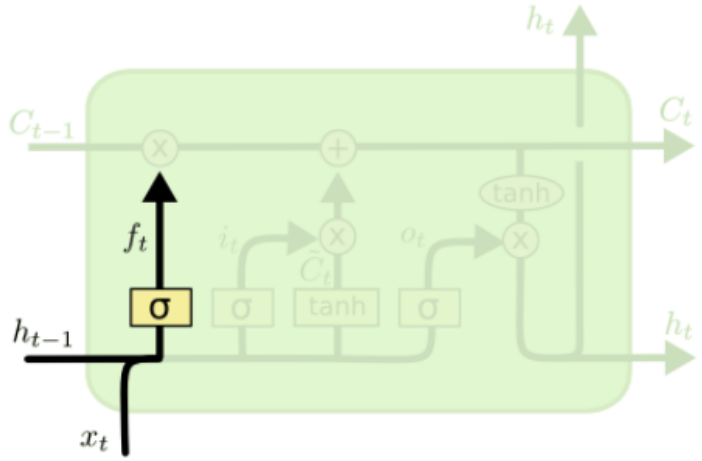
2. Input gate layer

3. Output gate layer

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

STEP 1: FORGETTING

The forget gate layer decides what we are going to get rid of



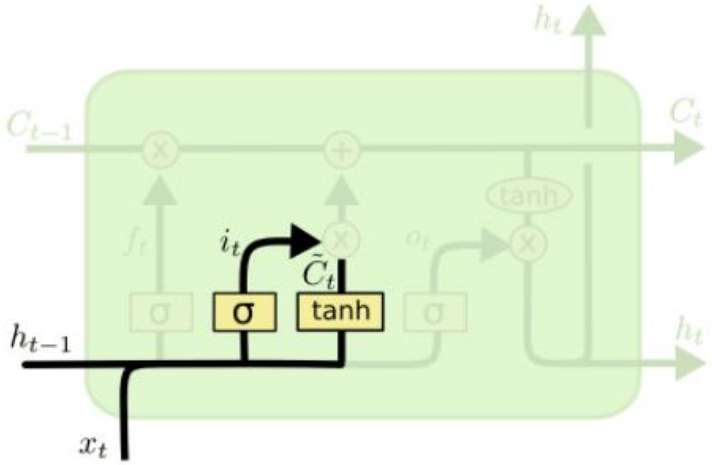
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- Forget gate looks at h_{t-1} and x_t
- Sigmoid outputs a value between 0 and 1
- Ex: the cell state might include gender, maybe we want to forget the gender of the old subject

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

STEP 2: INPUTTING

The input gate layer decides which values to update



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

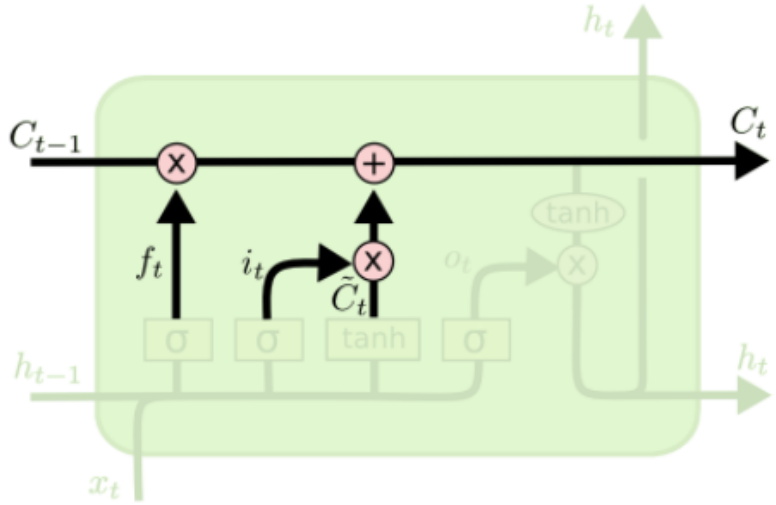
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

- First, sigmoid decides which values to update
- Next, tanh creates a vector, $\tilde{C}_t$, of new candidate values to be added to the state
- Ex: We want to add the gender of the new subject to the cell state, to replace the old one we have forgotten

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

STEP 2.5: UPDATING

Update the old cell state, C_{t-1}, into the new cell state C_t



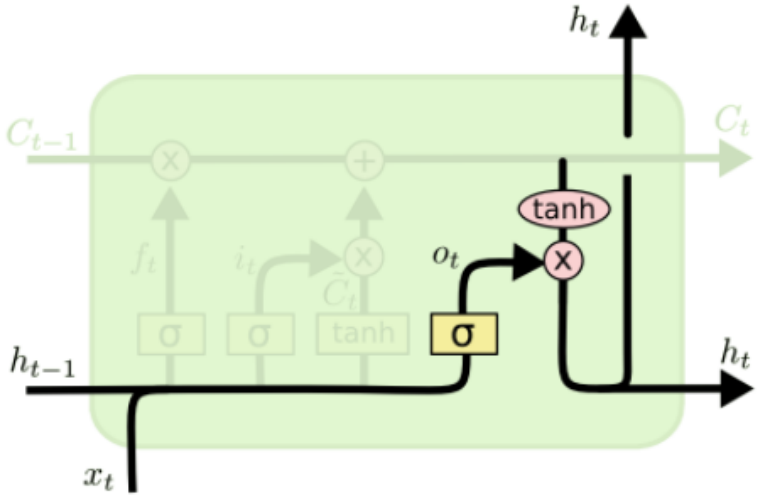
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Multiply the old state by f, - forgetting things
- Next, add $i_t * \tilde{C}_t$ - This is the new candidate values scaled by input gate
- Ex: The step where we would actually drop the old subject's gender and add the new information

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

STEP 3: OUTPUTTING

Outputting a filtered version of our current Cell state



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

- Sigmoid decides which part of the cell state we're going to ouput
- Next, put cell state through tanh (-1,1)
- Multiply together, so we only output the parts we decided to
- Ex: Since it just saw a subject, it may want to output information relevant to a verb. If the subject is singular/plural it will know how to conjugate

https://colah.github.io/posts/2015-08-Understanding-LSTMs/

WHAT IS IT GOOD FOR

RNNs can make use of sequential information

- Image captioning
- Generating creative copy
- KPI forecasting
- Predicting probabilities

Appendix

Acknowledgements

Material for these slides are taken from but not limited to the following sources:

Balasanov, Yuri Machine Learning and Predictive Analytics Session 8

https://lcn.epfl.ch/tutorial/english/perceptron/html/learning.html

https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/lecture-videos/lecture-12a-neural-nets/

http://neuralnetworksanddeeplearning.com/chap1.html#perceptrons

https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

https://colah.github.io/posts/2015-08-Understanding-LSTMs/