Science & Technology Facilities Council
Rutherford Appleton Laboratory

# Introduction to Machine Learning

Emmanuel Olaiya

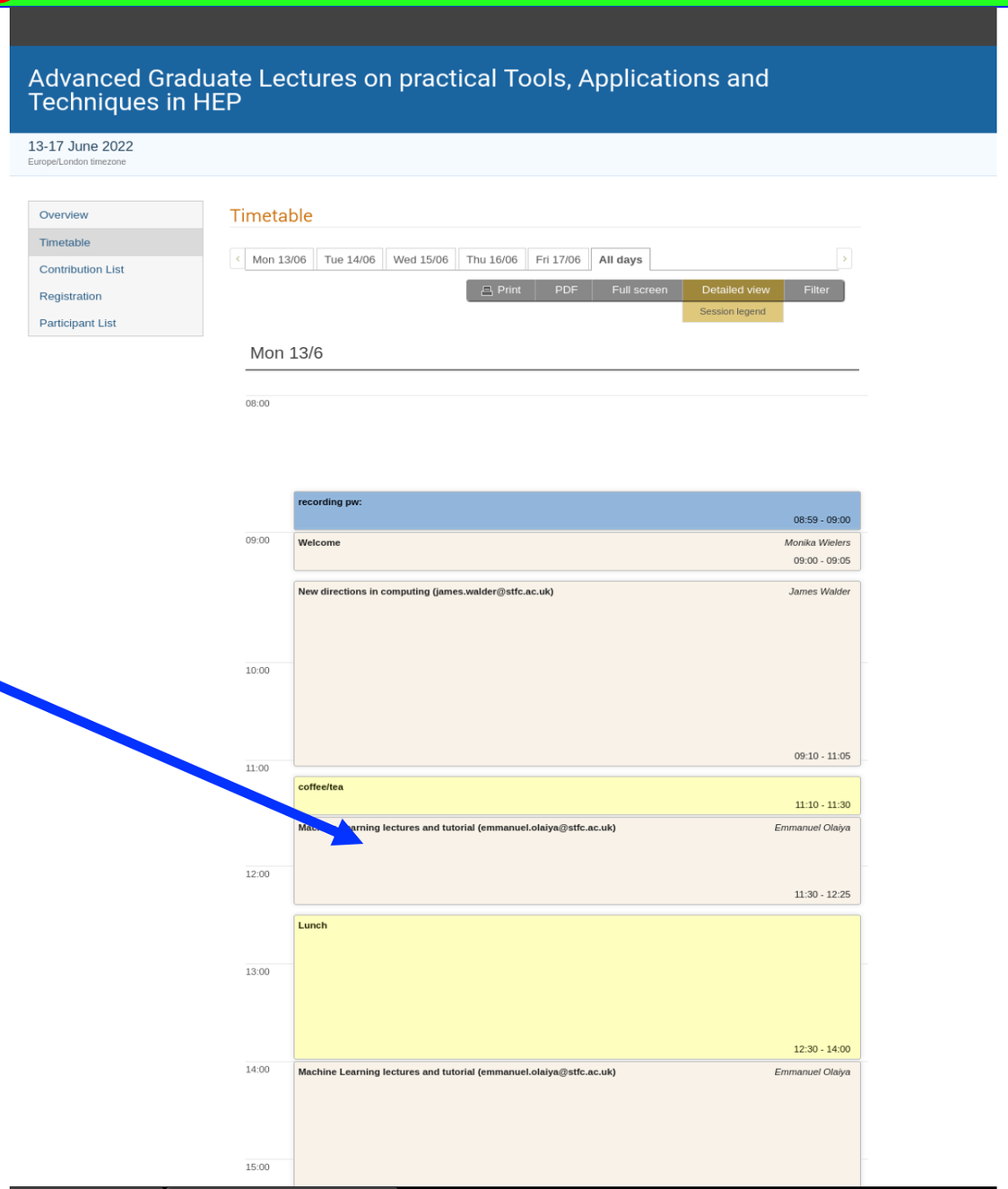
Rutherford Appleton Laboratory

13th June 2022

# Agenda

Today

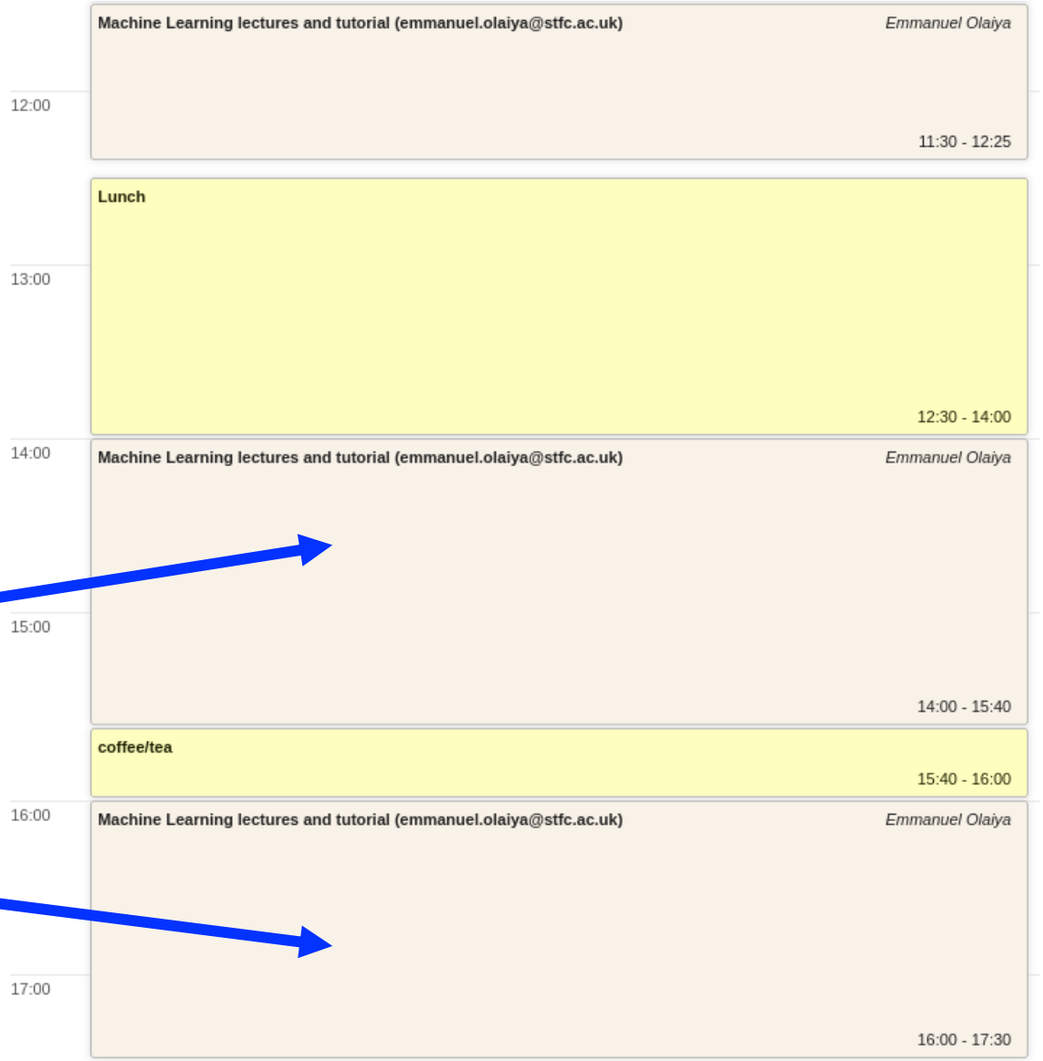
- 1130 – 1225: Lecture on Introduction to Machine Learning



# Agenda

Today, Monday 22nd May, this afternoon

- Tutorial walkthrough
- 1400 - 1540
- We will have breaks
- 1600 - 1730



# Content

- What is the goal of this lecture
  - To introduce you to Machine Learning
  - In one lecture you can't learn everything so the goal here is to summarise some main points
- The idea is for there to be a tutorial to follow (this afternoon) so you can implement and run Machine Learning models on a computer. Learn by doing!
  - There will be little to no implementation details in this lecture. This will be saved for the tutorial

# Contents

# What is AI/Machine learning

- AI (Artificial Intelligence) is a broad term that includes machine learning. AI is the idea that a machine can be similar to the human brain. Machine Learning (ML) is a product of AI's evolution
- Machine learning is the science of enabling machines to learn algorithms which are not explicitly programmed
  - Examples of ML use: face recognition, speech to text, internet search engines, spam filters, web browser personal recommendations, voice recognition. The list goes on ..
- This talk focuses on machine learning!
- Machine learning has been around for decades. So why is it taking off now?

# Why is Machine Learning Taking Off Now?

- Feasibility and accessibility
  - Data
    - Data is at the core of machine learning. Recent advances in data capture, data management and data storage have resulted in an exponential increase in data. Also disks are becoming faster,  cheaper with larger capacities
  - Hardware
    - Once you have the data you still need a large amount of computing power to do the number crunching. For a long time the required computing power was unavailable.
      - Enter the GPU (Graphics Processing Unit). Invented by Nvidia in 1999, GPUs can speed up significantly the training of neural nets (more about this in later slides)
  - Algorithms
    - With the hardware and data we are now seeing rapid development of sophisticated algorithms

# Types of Machine Learning

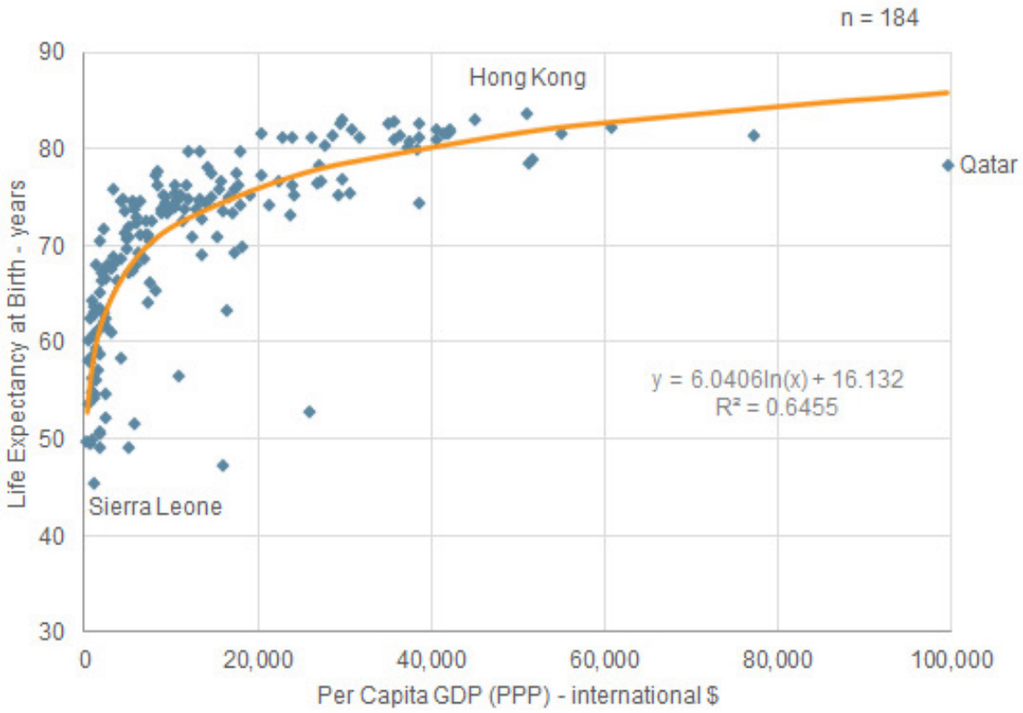
- Consider the types of Machine Learning
  - Supervised v Unsupervised
    - Supervised: the data you train your algorithm with also contains the solutions, called labels
      - Examples:k-Nearest Neighbors, Linear Regression, Logistic Regression, Decision Trees, …
    - Unsupervised: the data you train your algorithm on is unlabeled
      - Examples: K-Means, DBSCAN, t-SNE, ...
  - Batch v Online Learning
    - Batch: your algorithm is incapable of learning incrementally. All the data is used. If more data is acquired updating the algorithm requires retraining and then relaunching. This is offline learning.
    - Online: your algorithm is capable of learning incrementally. It can learn on the fly using new data. Online learning is great for systems that need to adapt or change rapidly eg. stock price modeling

# Types of Machine Learning

- Instance v Model-based Learning
  - Instance: the system learns the examples by heart and then generalises to new cases eg. t-SNE
  - Model-based:  build a model of these examples and the use that model to make predictions eg. Linear Regression, etc.

Model-based learning: Data of life expectancy v GDP. Fit a function to the data. Function can be used to make predictions

- Neural nets are model based
- Let's look at the architecture of Deep Neural Nets (DNNs) by taking Multilayered Perceptrons (MLPs) and Convolutional Neural Nets (CNNs) and Recurrent Neural Nets (RNNs) as examples



# Challenges with the Data

- Insufficient training data
  - Humans require little data in order to learn. This is not the case for Machine Learning. For example object recognition in images
    - Solution: Sorry, you just need more data
- Non representative data
  - You have to be careful that the training data is a good representation of the application data
    - Solution: ensure it is representative, eg. train on a subsample of the application data, compare test and run data
- Poor quality data
  - Significant errors, outliers, noise etc
    - Solution: Clean data. eg. remove outliers. Be careful not to bias data!

# Challenges with the Data

- Overfitting the data
  - A complex model that describes the training data very well but describes general cases poorly
    - Solution: simplify model, use regularisation, use dropout for DNN, the list goes on ….
- Underfitting the data
  - Model is too simple. Consequently doesn't describe the data well
    - Solution: Make the model more descriptive. Add more features (variables that describe the data), add more layers/nodes in the case of DNNs, more and larger features for CNNs, smaller feature strides for CNNs, the list goes on ….

# Testing and Validating

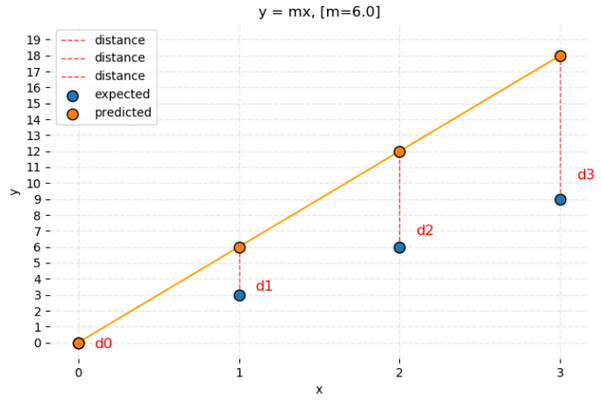
- How do you know how well your model will perform on new cases?
  - Test it out!
- One can split the data into a test set and a training set.
- Train with the training data and use the test set to see how the model will perform
- One risks overfitting to the training sample
  - We can do better than that
- Split the data into three samples, a training sample, a validation sample and a test sample
  - Train on the training sample but pick model parameters based on the best validation sample performance. Then run on the test sample to get an indication of performance in the real world
- How do we measure performance!

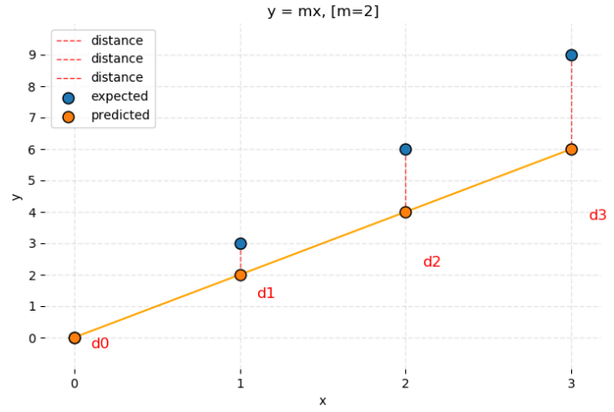
# A Performance Measure (Loss function)

- The Loss (Cost) function measures the performance of a Machine Learning model for given data
- The Loss function quantifies the error between the model's predicted values and the data's true values in the form of a single number
- You can define the error!

# A Performance Measure (Loss function)

- Lets take an extremely simple example
- We model some data with a straight line through the origin y = mx
- We try and find m for data x0 = 0, x1=1, x2 =2, x3 =3 such that y0 = 0, y1 = 3, y2 = 6, y3 = 9
- We take an initial guess of m = 6 followed by a guess of m = 2. We see disagreement. How do we express the disagreement mathematically, resulting in a single value? Mean Absolute Error (MAE)?

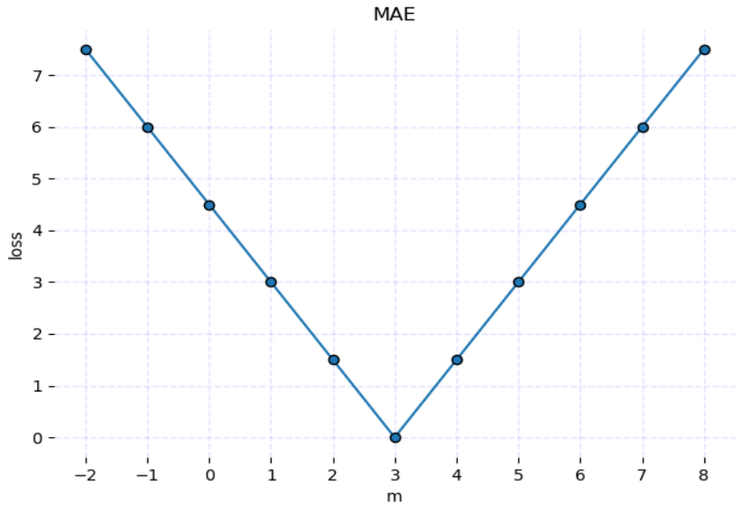


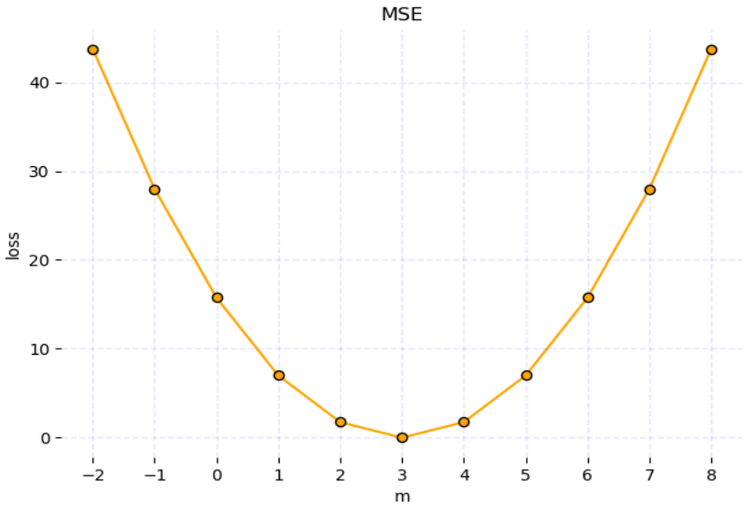




- Lets try Mean Absolute Error (MAE)? $\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$
- The smaller the value the better! MAE is 1.5 and 4.5 for m = 2 and 6 respectively. Consistent with observations!
- The ‘Absolute’ is necessary otherwise you can have negative values or even large positive and negative differences resulting in a small loss

# A Performance Measure (Loss function)

- What about other Loss functions. For example Mean Squared Error? $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$
- Lets compare the Loss between MAE and MSE for data and model in the previous example.

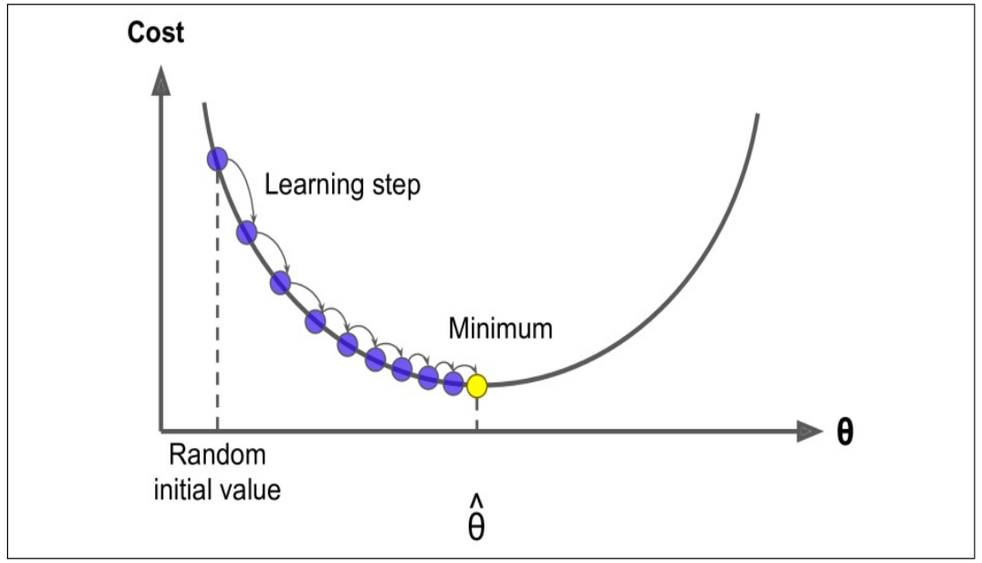



- The loss is minimal at m = 3 in both cases which is comforting
- MAE the loss contribution is linear compared to the error
- MSE the loss contribution from the error is quadratic. Data points with large errors can dominate the loss. Be careful when using MSE if you don't want modeling of the error tails to dominate
- Pick a loss function, that best minimises errors of the data as a whole
- One critical condition! The loss function must be differentiable everywhere!

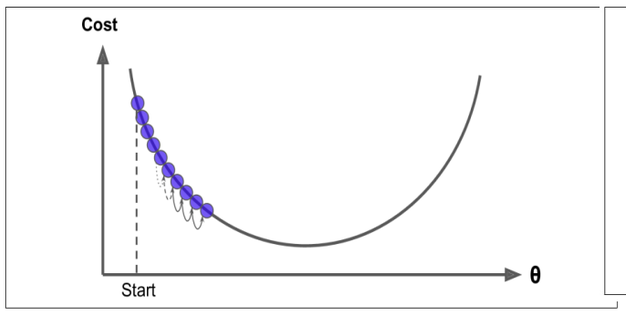
# Training

- In the previous model we saw that the correct answer for m has the smallest loss. How do we obtain the smallest (minimum) loss
- One option is to use Gradient Decent. Take the local loss gradient with respect to the parameter vector (in the previous case m) and step (learning rate) in the direction of the negative gradient. Repeat until the gradient is 0. You are at the minimum, best parameter. Same procedure is followed with a multi parameter space. This is how we train the multiple parameters of a neural net

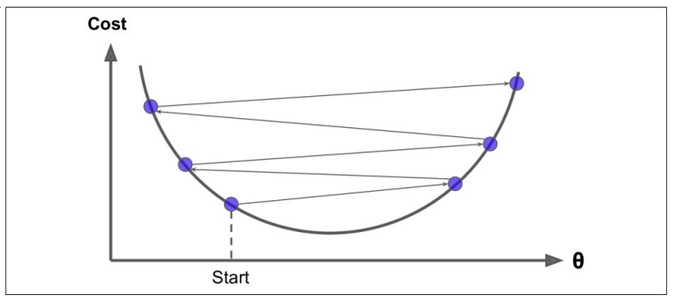



Things to consider

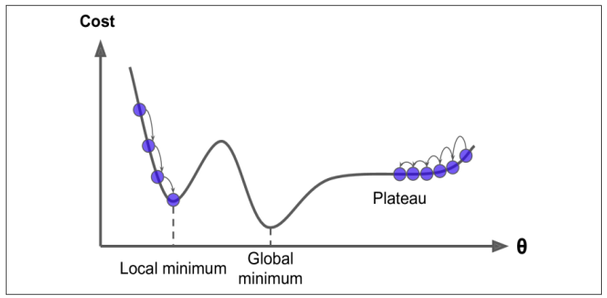
Learning rate too small. Takes a long time to get to the minimum




Learning rate too large and you will jump around the minimum

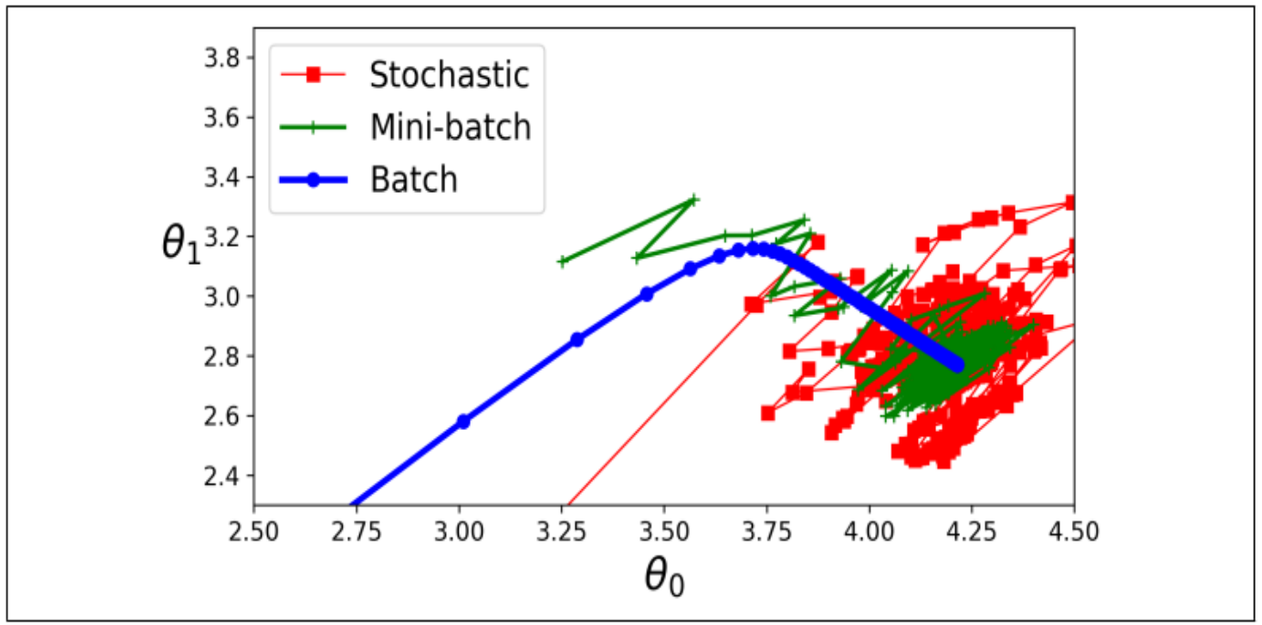



Be careful settling at local rather than global minima



# Training

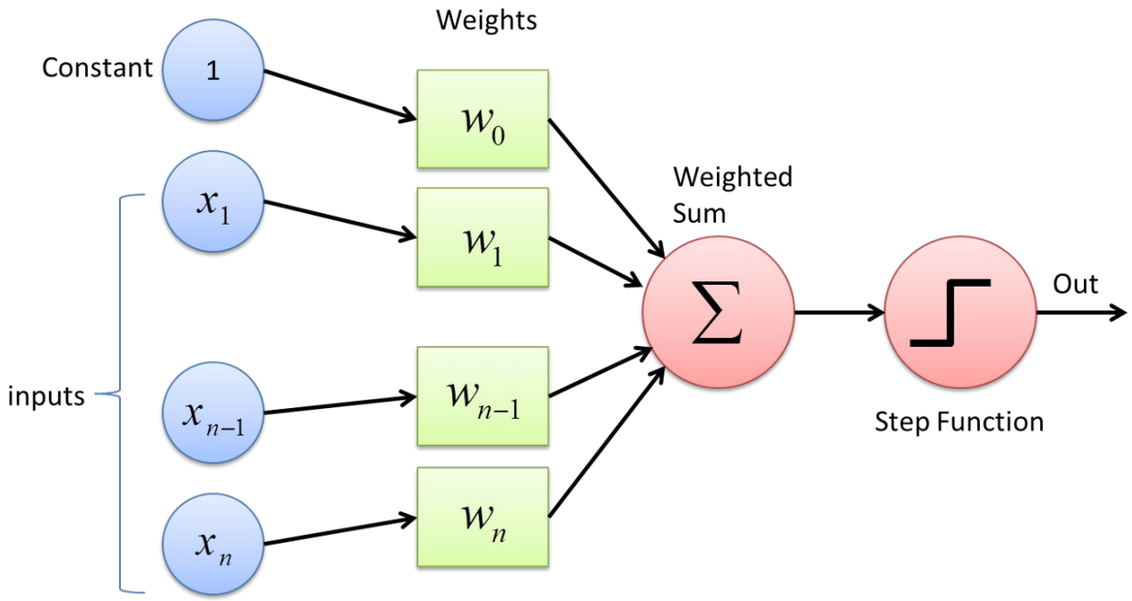
- Batch Gradient Decent calculates the partial derivative of the cost function using all the data at each step
  - For large datasets this is slow but you converge at a more precise minimum
- Stochastic Gradient Decent calculates the partial derivative using a random instance at each step
  - Much quicker but the gradients are more erratic. Once it gets to the minimum it will bounce around with the final value not being optimal
  - Good for getting out of local minima
- Mini batch Gradient Decent calculates the partial derivative based on a small random batch of events.
  - Fast and reasonable accurate



# Neural Networks and Deep Learning

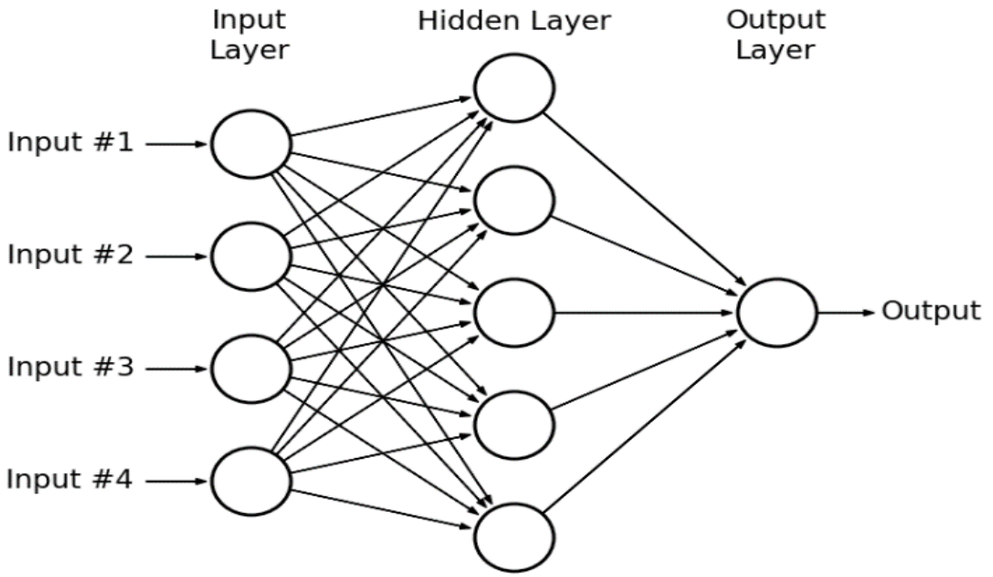
# The Perceptron

- The Perceptron inspired by the biological neuron is the simplest Artificial Neural Network (ANN)
- It consists of
  - Inputs
  - Weights and bias
  - Summed weights
  - Activation function (Step function)



# Multi-Layer Perceptron (MLP)

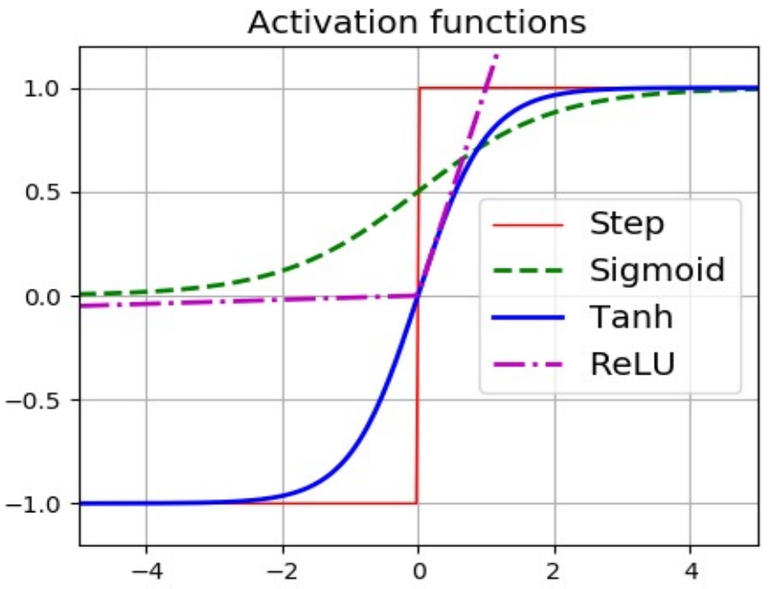
- The Perceptron could be used to solve simple problems such as simple linear binary classification
- However the Perceptron does struggle with some trivial problems
- What about having layers of neurons and stacking them (like in a biological neural network)?

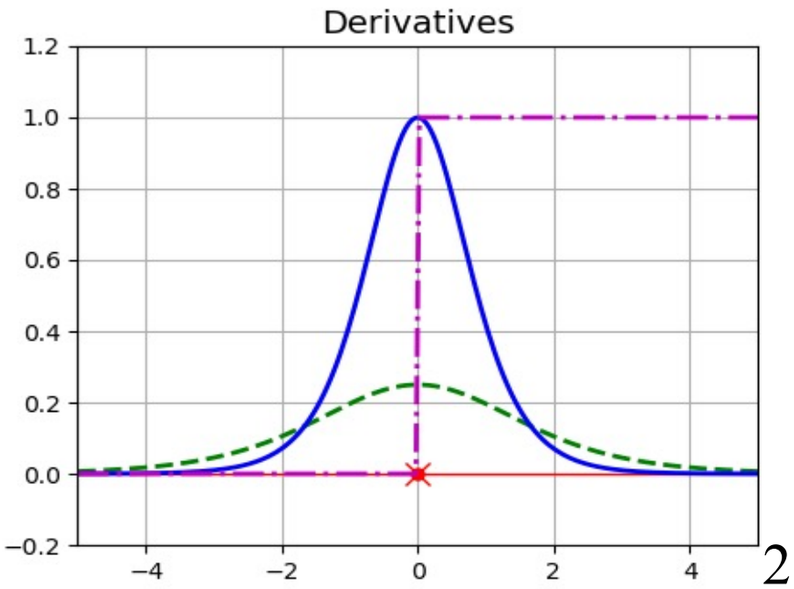



- This is a Multi-Layered Perceptron which is more sophisticated than the perceptron
- The MLP consists of an input layer, one or more layers of nodes (hidden layer) and an output layer. When a Neural Network consists of many hidden layers it is referred to as a Deep Neural Network (DNN)

# Activation Functions

- In order to train an MLP Gradient Decent is used where error calculations are propagated forward through the Neural Net and then back again.
- In order to use Gradient Decent the step activation function has to be replaced with something that doesn't have a zero gradient everywhere. You then have a powerful Neural Network!
- Below are some differentiable activation functions used to replace the step function
  - Classification: Sigmoid and tanh
  - Regression: ReLU



# Multilayer Perceptron (MLP)

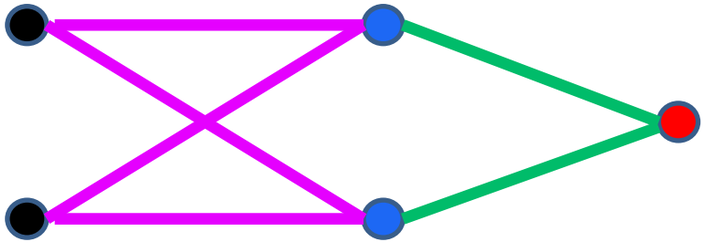
- Regression with DNN
  - Typically use ReLU or Leaky ReLU activation function for the nodes in the hidden layers
  - One or more outputs depending on the number values you want to estimate
  - The output node typically doesn’t have an activation function so the output is free to output any range. You can use activation functions on the output if you want to restrict the output value

# Multilayer Perceptron (MLP)

- Classification with DNN
  - The output node typically uses a sigmoid/logistic activation function constraining the output to a range between 0 and 1 which can be interpreted as a probability.
  - Can have multiple output nodes for multi classification tasks

# Black Box Myth

- One of the criticisms of Neural Nets is that they are a black box. “We have no idea what they are doing inside”
- This of course is not true
- They can be complex, but complexity is not obscurity!



$f_{NN} = \sigma(b_2 + W_2\sigma(b_1 + W_1 x))$

Equation for a classification NN with one layer of two nodes and an output

If we so wished we could train this NN and output its weights. Plug the weights into the adjacent equation and use the equation instead. They should behave the same!

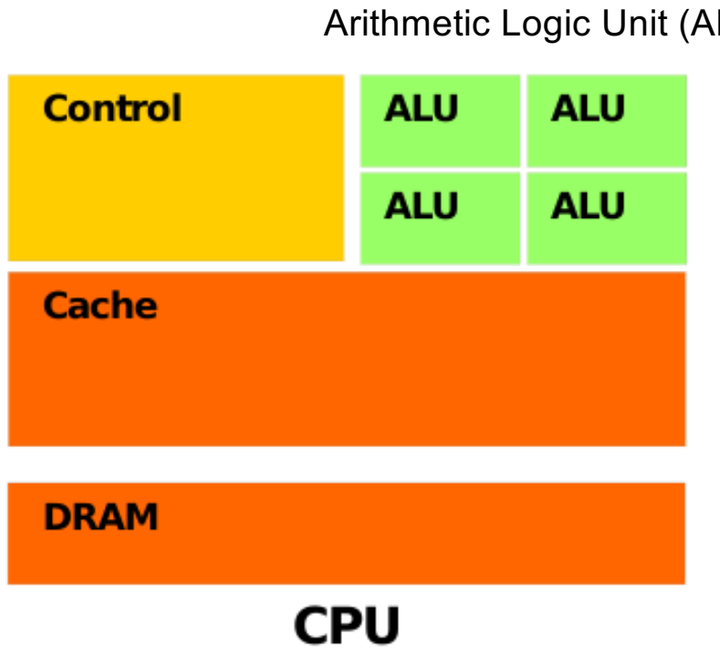
Activation (classification): $\sigma(x) = \tanh(x)$, Sigmoid $({}^{1}/_{1+\,e^{-x}})$

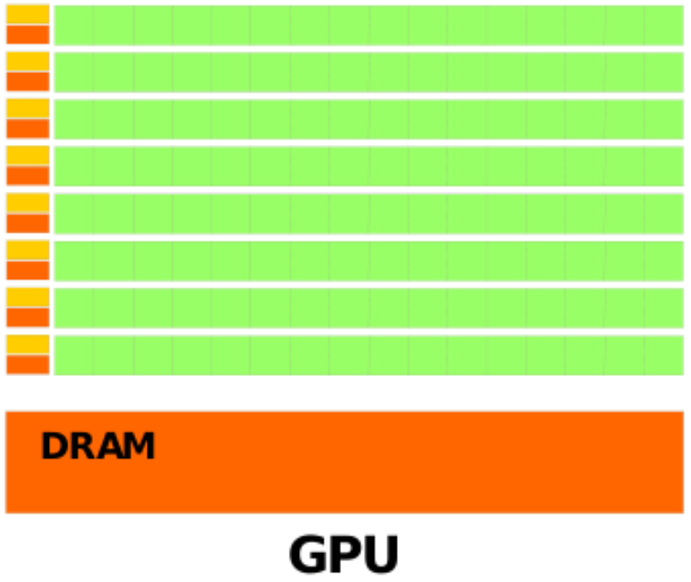
Activation (regression):RELU

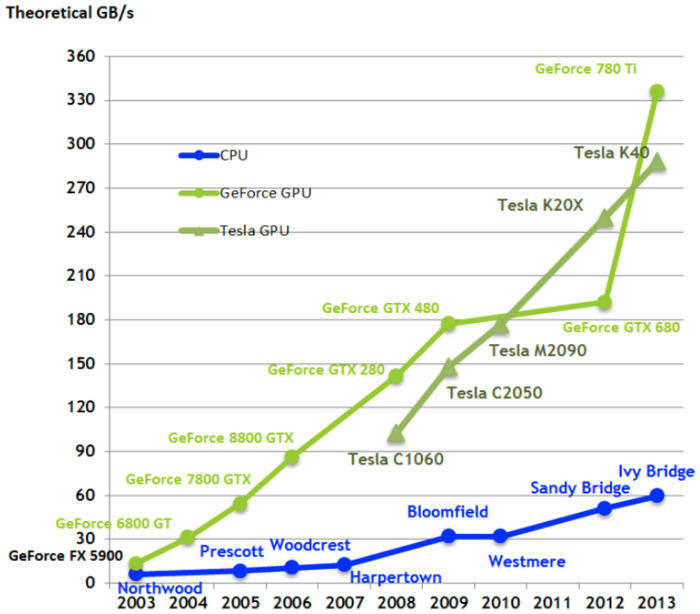
Input: $x = \begin{bmatrix} x_{1,1} \\ x_{2,1} \end{bmatrix}$ Weight: $W_1 = \begin{bmatrix} W_{1[1,1]} & W_{1[1,2]} \\ W_{1[2,1]} & W_{1[2,1]} \end{bmatrix}$ Bias: $b_1 = \begin{bmatrix} b_{1[1,1]} \\ b_{1[2,1]} \end{bmatrix}$

$$f_{NN} = \sigma_2\left(b_{2[1,1]} + \begin{bmatrix} W_{2[1,1]} & W_{2[1,2]} \end{bmatrix}\sigma_1\left(\begin{bmatrix} b_{1[1,1]} \\ b_{1[2,1]} \end{bmatrix} + \begin{bmatrix} W_{1[1,1]} & W_{1[1,2]} \\ W_{1[2,1]} & W_{1[2,2]} \end{bmatrix}\begin{bmatrix} x_{1,1} \\ x_{2,1} \end{bmatrix}\right)\right)$$

# GPU v CPU



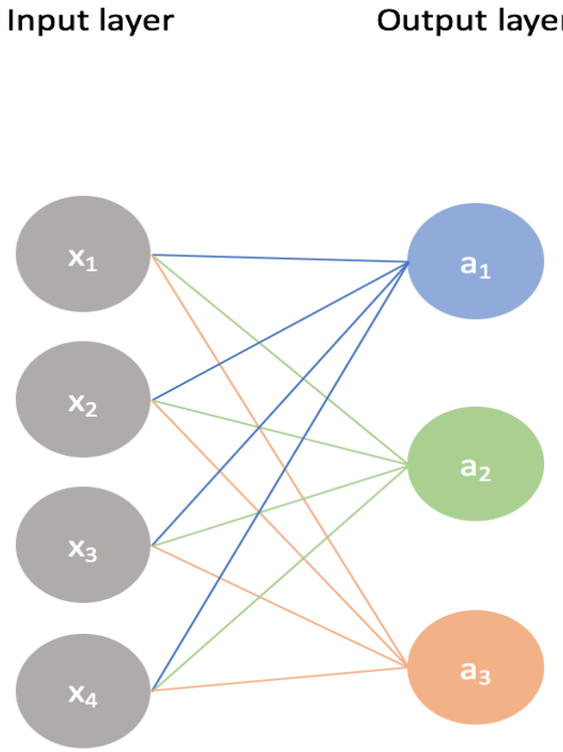




- Small number of compute cores
- MIMD (Multiple Instruction Multiple Data)
- Optimised for serial operations
- Low latency

- Large number of compute cores
- SIMD (Single Instruction Multiple Data)
- Built for parallel operations
- High throughput

Let's look at an MLP to demonstrate the high throughput of a GPU!

# GPU v CPU




## A simple neural network

$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \\ w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \end{bmatrix} \xrightarrow[activation]{} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

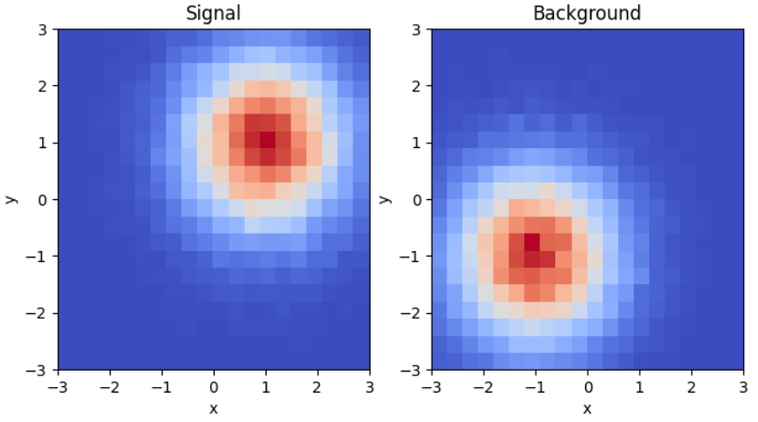
- Consider this output layer of the neural net
- Output is a vector calculated from the matrix equation on the LHS
- Each element of the vector can be calculated on a computing core
- In this simple case this can be done easily on a CPU
- However repeating this process sequentially over many data points is very slow on a CPU
- Because we want to run the same instruction, it is much faster to run in parallel on the many GPU cores

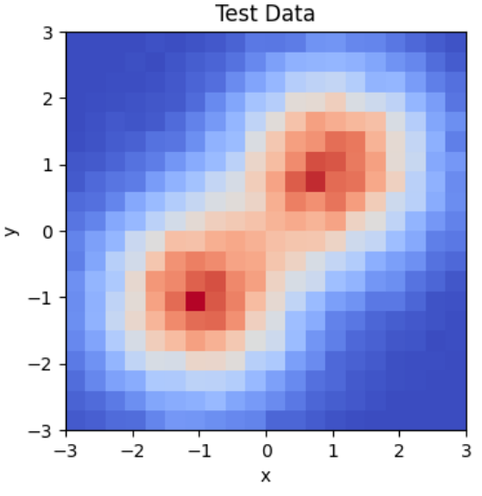
# Some Training Problems

- Vanishing/Exploding Gradients
  - When training calculated gradients can get smaller and smaller so the movement across the loss function and the training doesn't converge. This is the vanishing gradients problem
  - In some cases the opposite can happen and the gradient can get so large and the algorithm diverges. This is the exploding gradients problem
- Vanishing/Exploding Gradient aids
  - The weight initialisation (normal, mean 0, sigma of 1) was seen to be a problem. Variance of weights keeps increasing from layer to layer until the activation saturates at the top layers. Different initialisations are used with specific activation functions to help with the problem
  - Batch Normalisation, before or after the activation function, inputs are zero centred and normalised, then scaled and shifted
  - Choice of activation function. Activation functions that saturate with 0 derivatives can cause problems. Use alternative activation functions with this in mind such as Leaky ReLU instead of ReLU

# Simple DNN Classification example

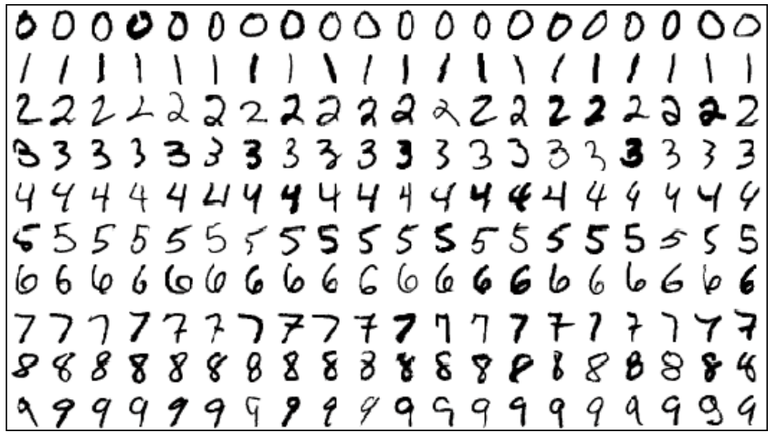
- Generate a data sample with two categories (we will walk through this in the tutorial)
  - Signal: Gaussian distributed with a mean x,y value of 1 and sigma of 1. Label events 1
  - Background: Gaussian distributed with a mean x,y value of -1 and sigma of 1. Label events 0
- Build a DNN
  - 2 inputs (x,y)
  - 1 hidden layer of 100 nodes. ReLU activation for each node
  - 1 output (1,0), signal or background. Sigmoid activation at node
- Train DNN
  - Run over data sample. Batch gradient decent 100 epochs
- Test!
  - Use trained DNN on a new data sample of signal and background. Can identify correct label 92% of the time



# Classification with the MNIST Dataset

- The MNIST (Modified National Institute of Standard and Technology ) database is a large database of handwritten digits that is commonly used for training various image processing systems



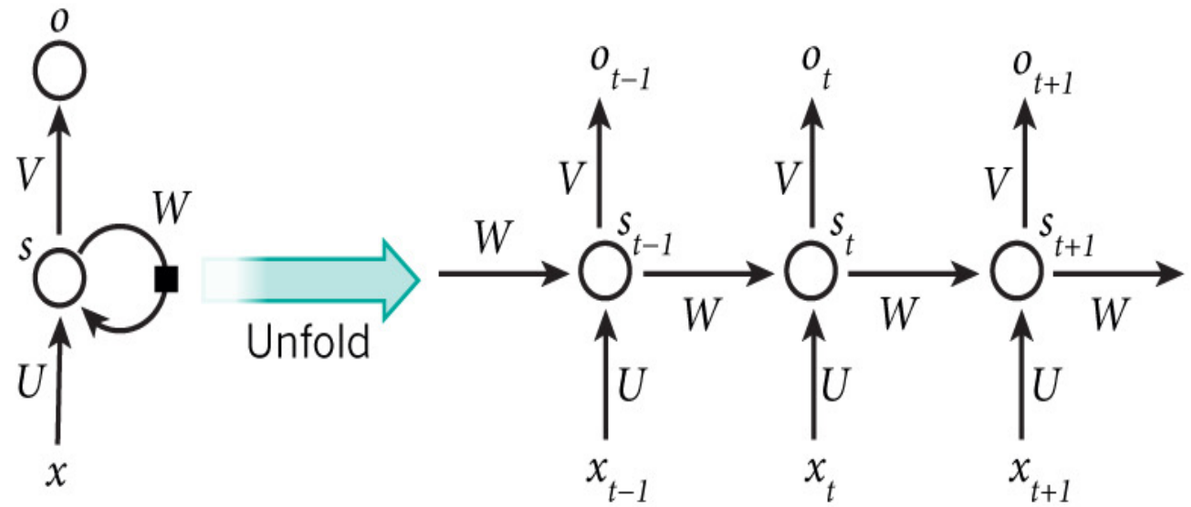
- Even though these are images we can still train a DNN to be able to identify them (tutorial walkthrough)
- Each image is 28x28 pixels so we just assign an input for each pixel
- For example we can construct a DNN as follows
  - Number of inputs = 28 x 28
  - Two hidden layers; 1st hidden layer = 300 nodes, 2nd = 100 nodes
  - Ten outputs (one for each number), sigmoid activation function
- With a training sample of 44000 events running over a large test sample the DNN has an accuracy > 90%
- Not bad! But we can do better!

# Deep v Wide Neural Nets

- A Neural Net with just a single hidden layer can model complex functions as long as you have enough neurons in a layer
- However for really complex functions deep networks have a higher parameter efficiency
  - For the same number of parameters, deeper networks are more powerful
    - Prefer deeper rather than wider NNs!

# Recurrent Neural Nets (RNNs)

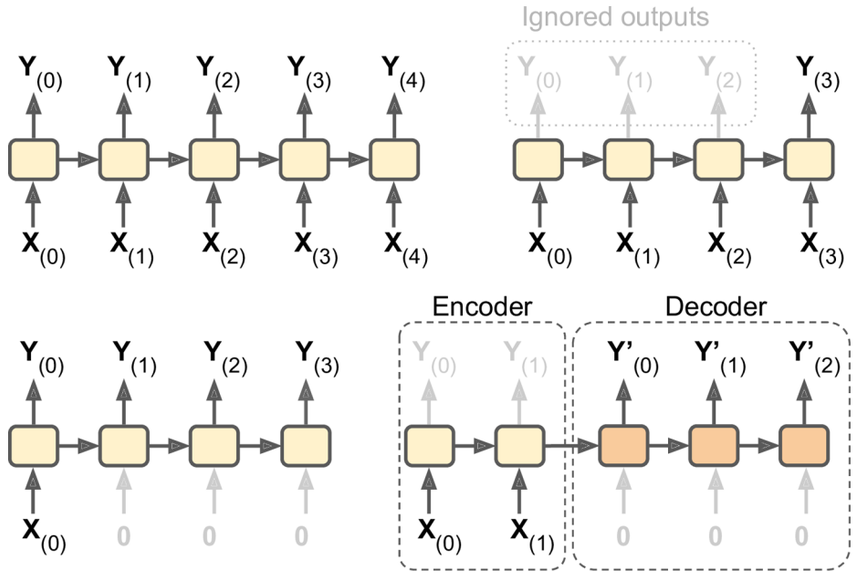
- In a MLP the operations flow in a forward direction, from input to output
- A Recurrent Neural Net (RNN) looks the same but also has connections that feed back from the output to the input
- Lets look at a simple RNN composed of one neuron




- At each time step (also called a frame) the neuron receives the inputs as well as the inputs from the previous step. Obviously there is no previous step initially
- The above image shows the RNN unfolded through time
- We have a network where each data output is related to the previous input
  - We have some information/memory of the past output

# Input Output Sequences

- A RNN can take a sequence of inputs and produce a sequence of outputs. This is known as a sequence-to-sequence network (top left). Useful for predicting time series such as stock prices
- Can feed in a sequence and ignore all outputs except the last one. This is a sequence-to vector network (top right). For example a book review. Was it a good or bad review
- Conversely you can input a vector and output a sequence (bottom left). This can be an image that is input and the output is a caption
- Also you can have a sequence-to-vector network followed by a vector-to-sequence network (bottom left). An example of this is translating one language to another. This is an Encoder-Decoder



# Recurrent Neural Networks

- Lets use an RNN to look at previous outputs to predict the next output
- Generate a batch of time series data using the equation

```
x(t) = 0.5 * sin((t - a) * (p * 10 + 10))  #  wave 1
          + 0.2 * np.sin((t - b) * (q * 20 + 20)) #  wave 2
          + 0.1 * (r - 0.5)   # noise
```

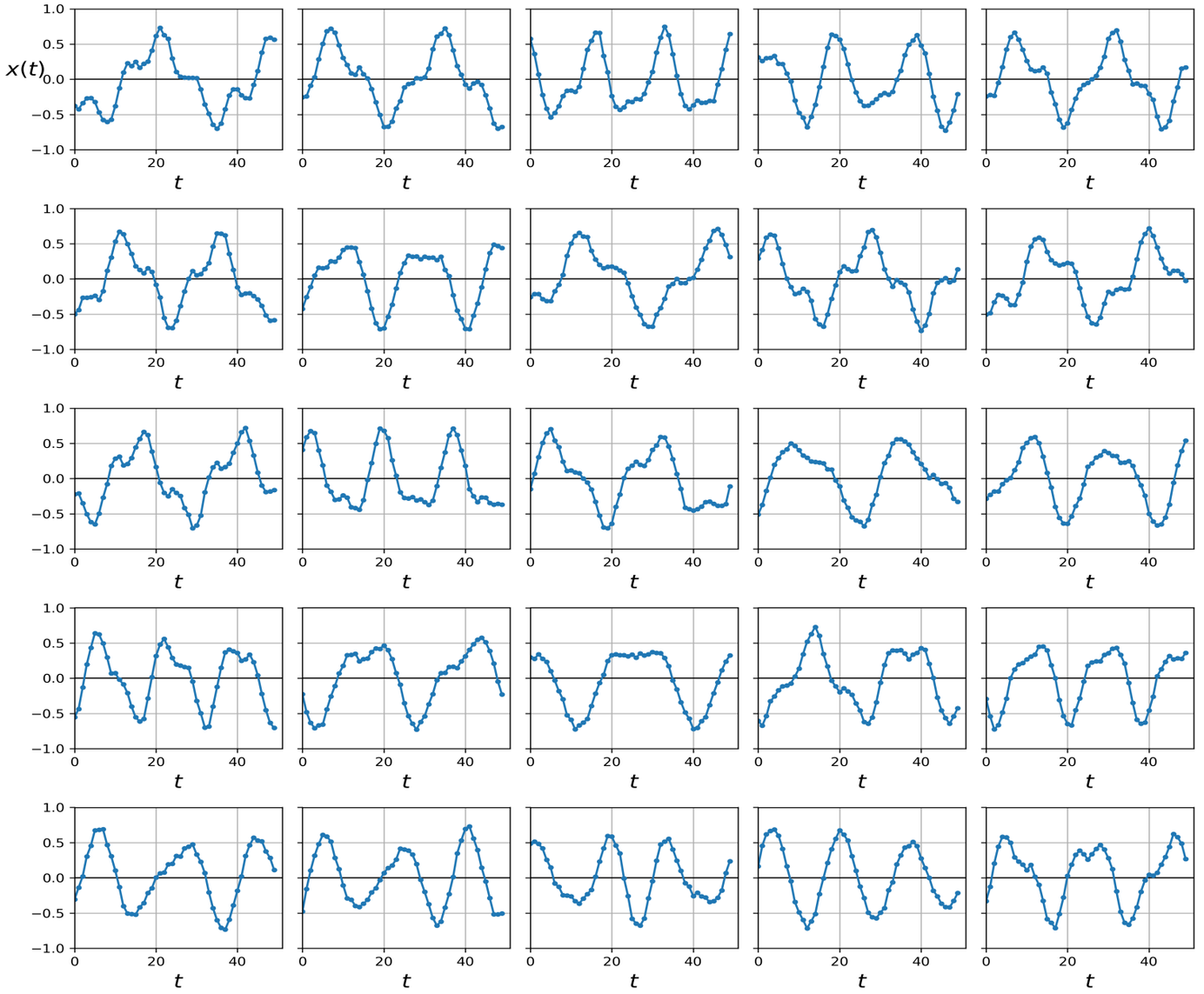
- The parameters, a,b,p,q and r are constants randomly set between 0 and 1 for each time series
  - Consequently each time series looks different
- Lets take a look at a few samples

# Recurrent Neural Networks

Each of the 25 time series is generated by the equation on the previous page

We generate thousands of these series to train the RNN

Can we model this series with an RNN? (tutorial walkthrough)

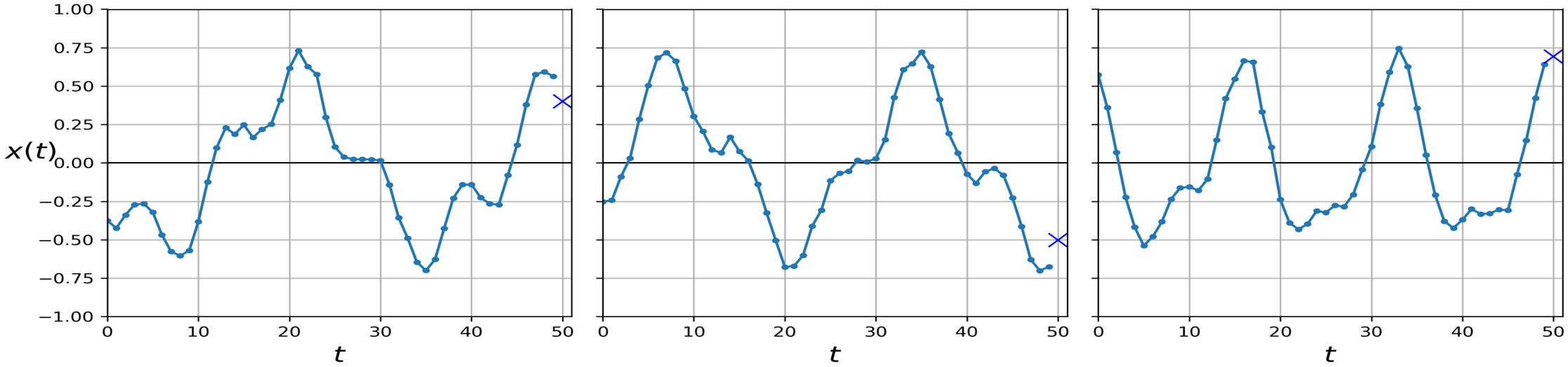


# Recurrent Neural Networks

- Data is a series of 50 points. Lets use an RNN to predict the next data point (51st). We have training data with the 51st point. Lets use the training data to train an RNN using the 50 data points against the 51st. This is a vector-to-vector network
- We can construct an RNN that looks like:
  - One input (x(t))
  - 2 layers, each with 20 nodes
  - One output (x(t+1))
- Whilst the RNN has an output at each point we only train against the last output. That is what we are interested in!

# Recurrent Neural Networks

- Lets take 3 previously unseen time series and predict the 51st data point
  - Looks good



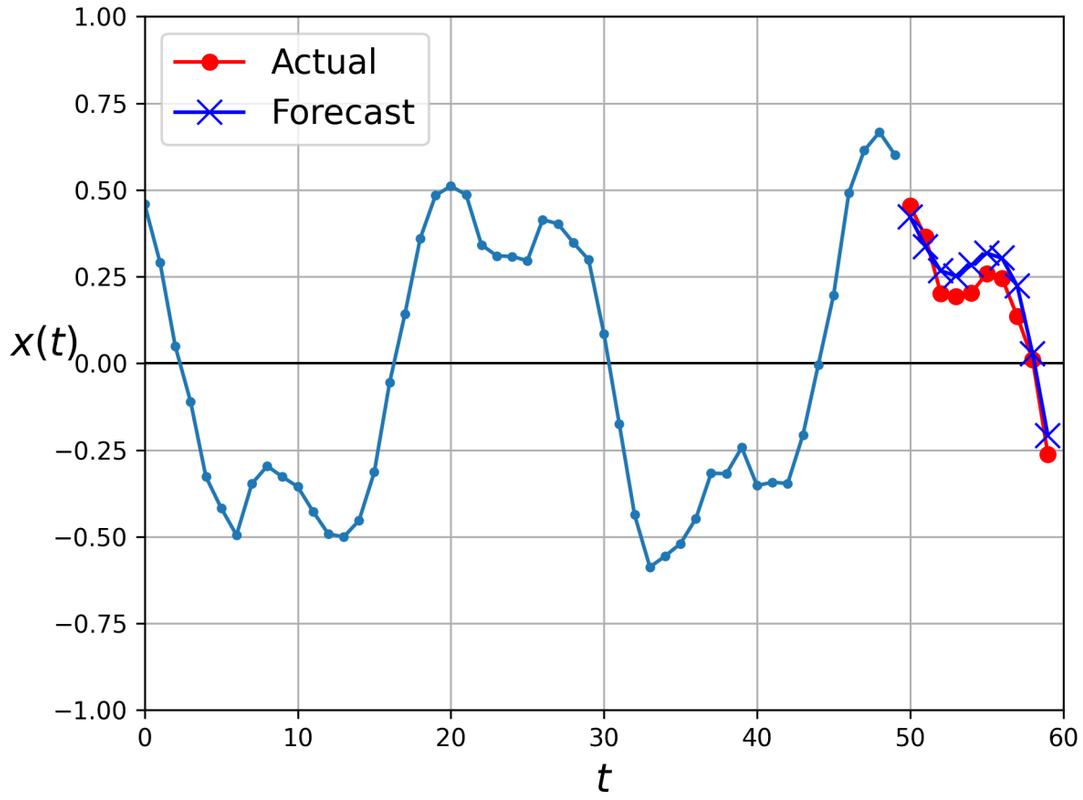

- We can't rely on looks alone. We compare the loss along with other test for many test samples

# Recurrent Neural Networks

- Ok, what about predicting the next 10 points
- We can construct an RNN that looks like:
  - One input (x(t))
  - 2 layers, each with 20 nodes
  - Ten outputs (x(t+1), x(t+2), ….., x(t+10))
- We train against outputs 51 to 60 . That is what we are interested in!

# Recurrent Neural Networks

- Lets look at a time series from our test data and predict the 51st to 60th data points
  - Looks good

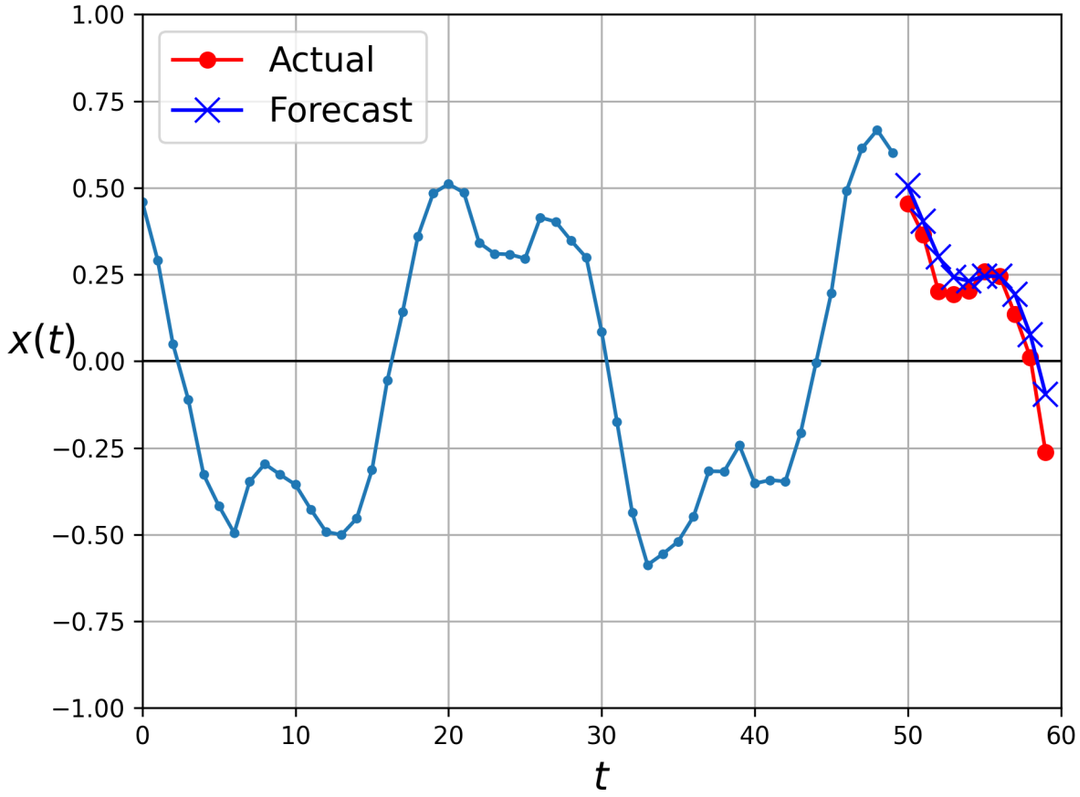


# Recurrent Neural Networks

- We can do better! Rather than training to predict the next 10 steps at the very last step we can train the model to predict the next 10 steps after every step. This is then a sequence-to-vector network
- There is more information and we should see a more accurate prediction
- Lets not worry about the implementation here. Just note we now look at all steps and train against the next 10 values
- Lets look at the results

# Recurrent Neural Networks

- Look at the test time series again and predict the 51st to 60th data points
  - You can see this looks better and trust me so is the loss!

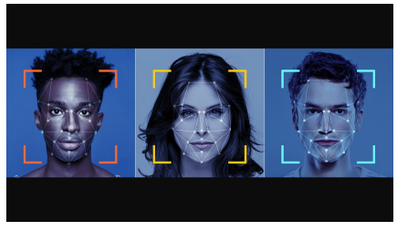


# Recurrent Neural Networks

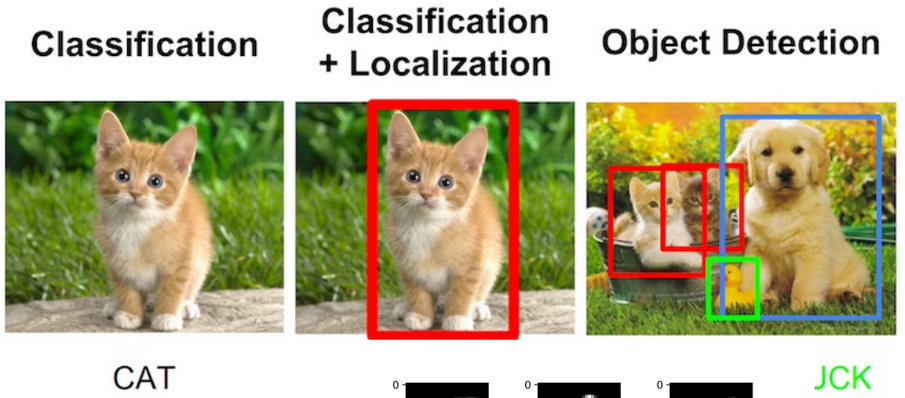
- RNNs are great for learning patterns and making predictions
  - Time series, NPL (talk about this another time) ...
- Even though they are not designed for this RNNs are also used for training against data that varies in size (number of steps)
- However for really long data series the RNN forgets about the early input by the end and breaks down
  - Don’t have time here but rather than simple RNN cells discussed here one can use memory cells such as Long Short-Term Memory (LSTM) cell and Gated Recurrent Unit (GRU) cell to prevent the break down
  - Convolutional layers can also be used to to improve the long term memory

# Convolutional Neural Nets

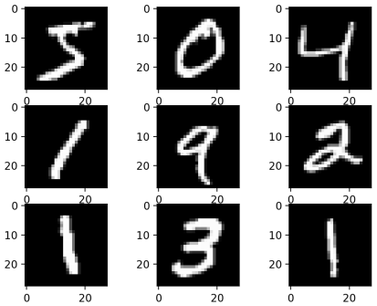
- A Convolutional Neural Net is a class of Neural Network mainly applied to analysing visual images
- We may be familiar with their use with
  - Face recognition



  - Object detection

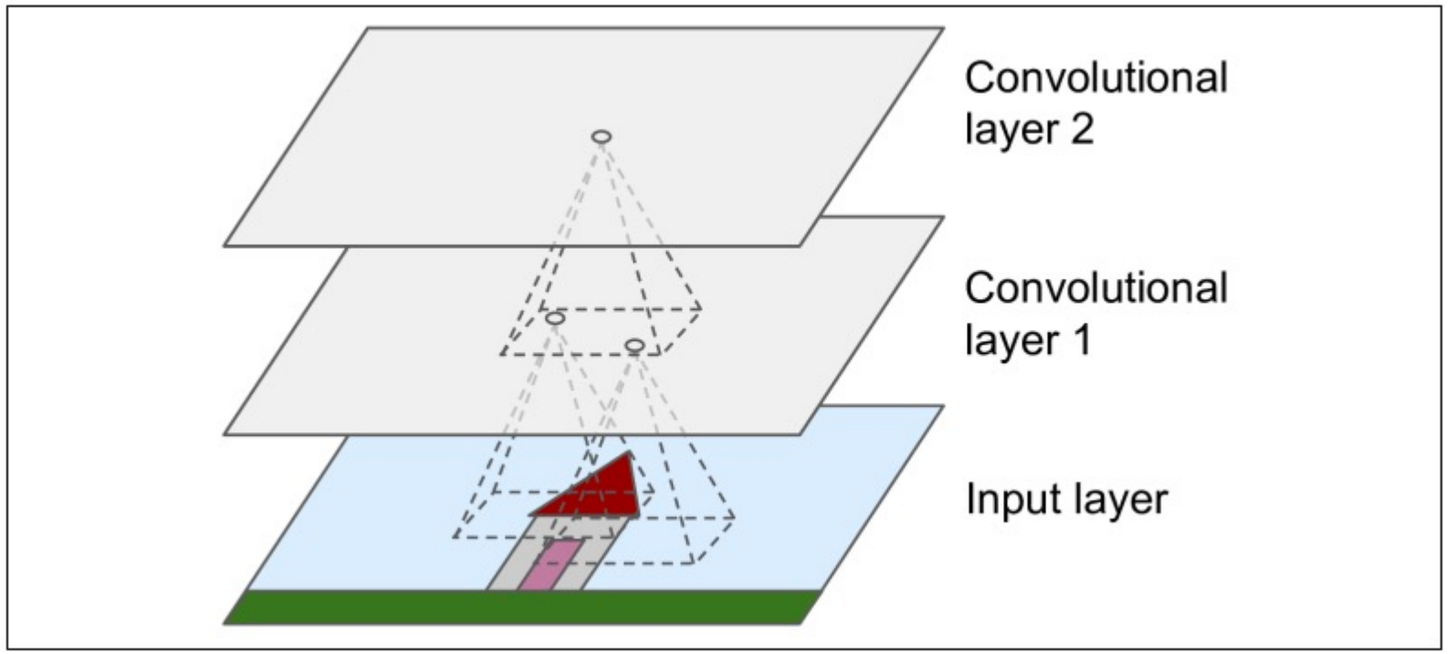



  - Identifying numbers/text in images



# Convolutional Neural Nets

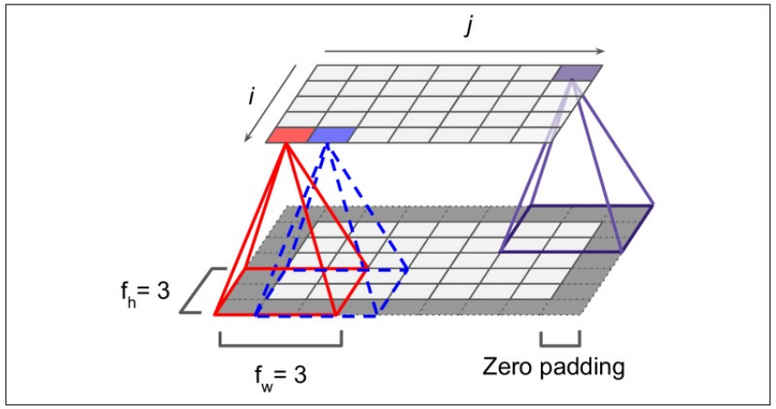
- How do CNNs work?



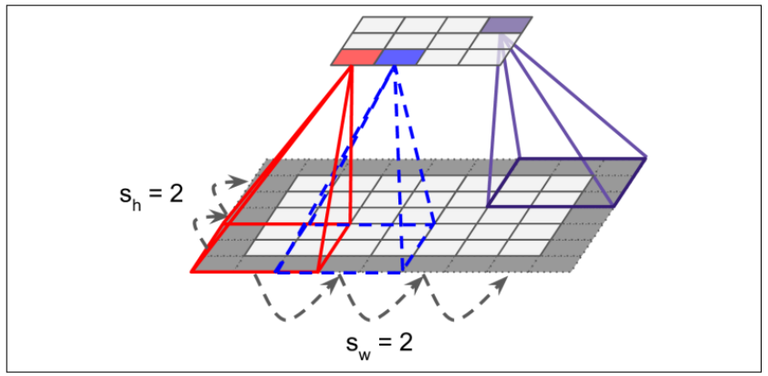

- An n dimension filter (n=2 for the image above) scans across your data/image taking the product. The output forms the first convolutional layer. This process is repeated with different filters on the first convolutional layer to output a second convolutional layer. We can proceed producing as many convolutional layers as we see fit
- What is the point of this?
  - Reducing the model's complexity

# Filters

- How do filters work?
  - Converting low level feature into higher level features




- Smaller filter is a larger approximation of the data
- With zero padding (adding zeros around the layer) and a step size of 1 the next convolutional layer will same size. Without zero padding the next convolutional layer is reduced in size

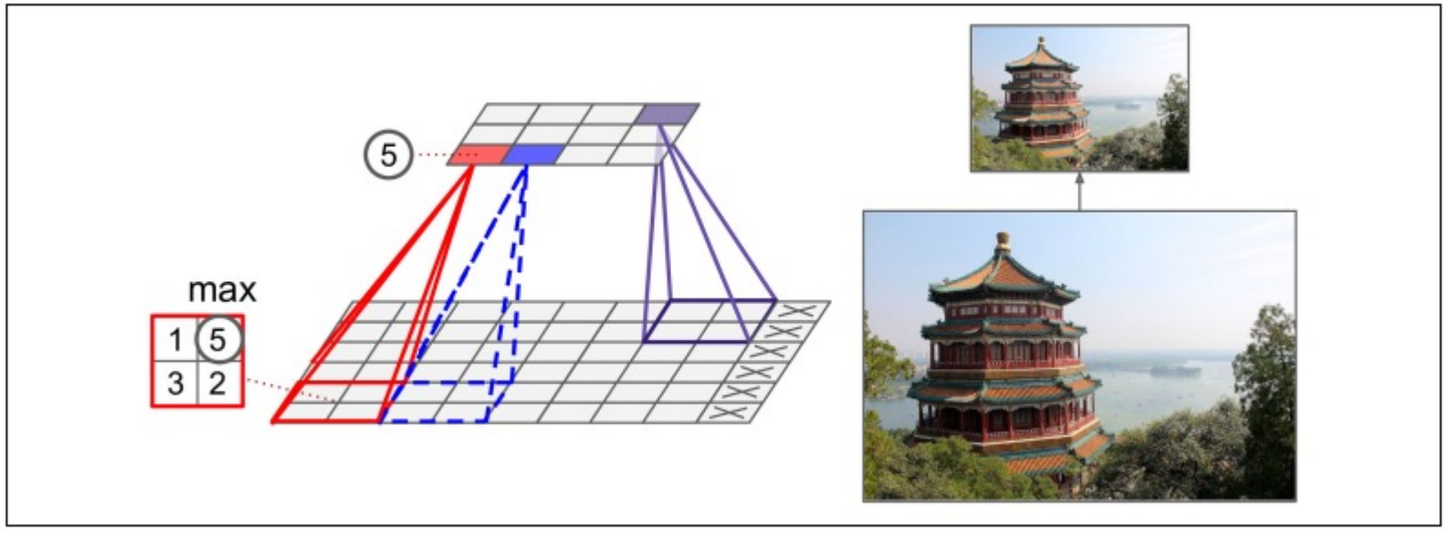



- You can also specify the step size (stride) of the scan. Increasing the step size will reduce the size the next convolutional layer
- The filters are the weights and the contents of the convolutional layers are referred to as the neurons. Unlike in a MLP you have a set of weights (filter) that is applied to all neurons in the layer. The filter values are set during training
- You can work with a less weights compared to a MLP

# Pooling

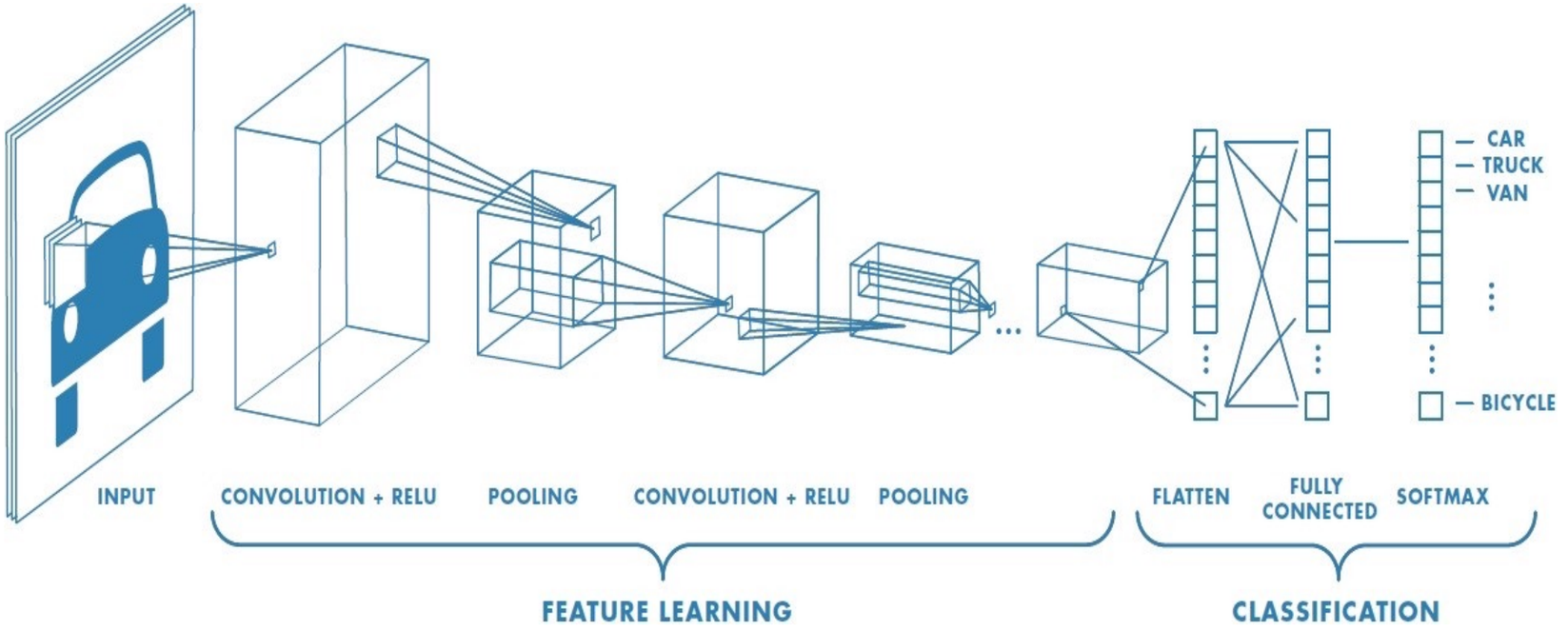
- You can reduce the load even further by pooling
  - This reduces the computational load, memory usage
  - The principle is the same as a filter. You specify a field and scan over the layer applying pooling and output the result to another layer
  - There no weights for pooling. For example you have:
    - Max pooling: Output the maximum value
    - Ave pool: Output the average value

Max pooling example



# CNNs Putting it all together

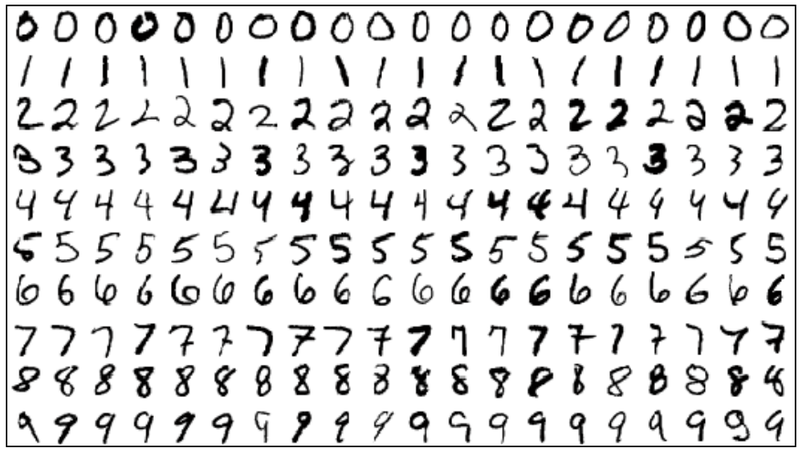
- The last layer is then fed to a MLP and trained on data where the output can be used for regression or classification




- Apply filters to the RGB components of the image to produce the 1st convolutional layer
- Apply pooling to the 1st convolutional layer to reduce the layer size
- Apply filters to the pooling output to produce the 2nd convolutional layer
- Apply pooling and then feed the output to a DNN with outputs for classification

# Classification with the MNIST Dataset

- Lets look at the MNIST database again (tutorial walkthrough)



- This time we will feed the images to a convolutional net that looks like:
  - Two convolutional layers of 32 and 64 feature maps respectively
  - Max pooling of the 2nd layer with a field size of 2x2
  - The pooling output is fed to a MLP hidden layer with 64 nodes with relu activation
  - The output has ten nodes with a logistic activation function
- A reasonable straightforward CNN!
- After training and testing this simple CNN has an accuracy of 99%

# Convolutional Layers

- Seen many of the components of a CNN before: deep neural nets and their activation functions.
- The most important building block is the convolutional layer
- Neurons in the first convolutional layer are not connected to every single pixel in the input image but only connected to the receptive field (filter). Unlike when compared to a DNN. Results in less weights!
- In return each neuron in the second convolutional layer is connected only to neurons in the receptive field of the first layer
- This architecture allow low level features in the input to be built into successively higher level features as you add convolutional layers

# Summary

- Machine Learning is a huge, expanding, exciting field
- Lots of other exciting topics, Decision trees, Support Vector Machines, Autoencoders, Graph Neural Networks …..
- Using Machine Learning we can build some powerful models for regression and classification
  - We looked at different methods (MLP, RNN, CNN) to perform classification and regression tasks
- Machine Learning is having a large impact on our lives today and I don’t think it is going anywhere soon!