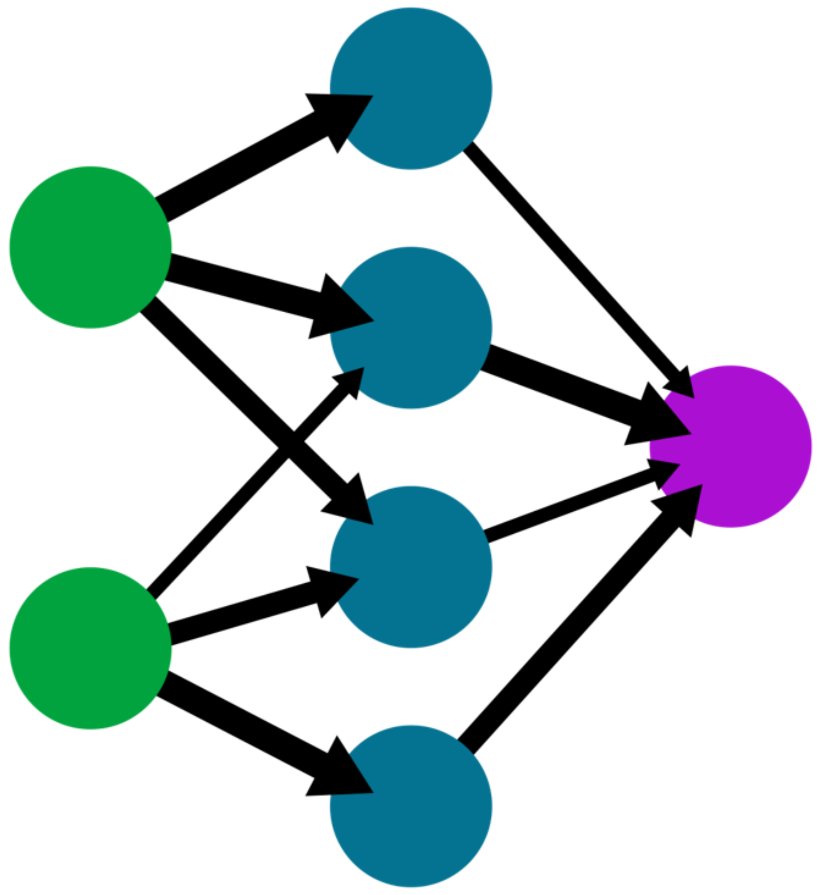
Faculty of Engineering and Technology
School of Computer Science and Mathematics

7144COMP Deep Learning Concepts and Techniques

Dr Paul Fergus, Dr Carl Chalmers
{p.fergus, c.chalmers}@ljmu.ac.uk
Room 714, 629 Byrom Street

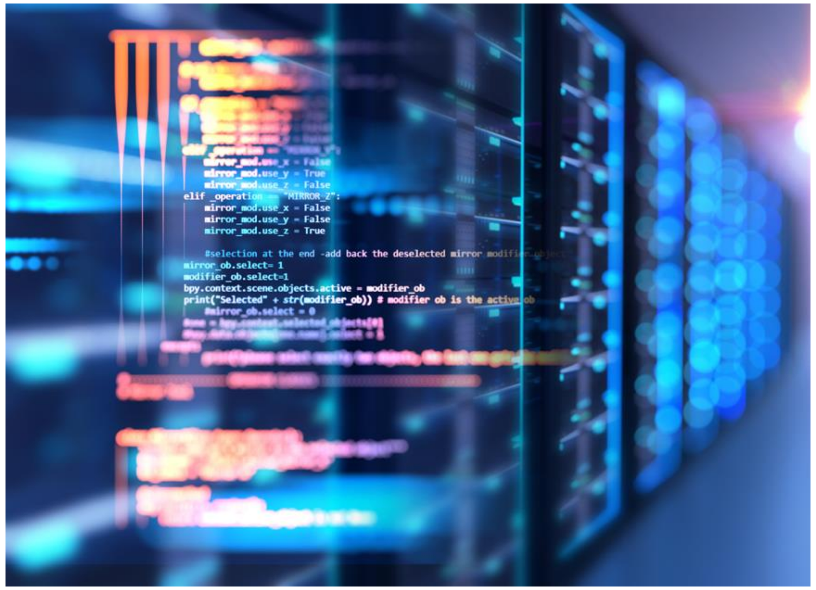
Lecture 3

Optimisation
(Hyperparameter Tuning)



In this session…

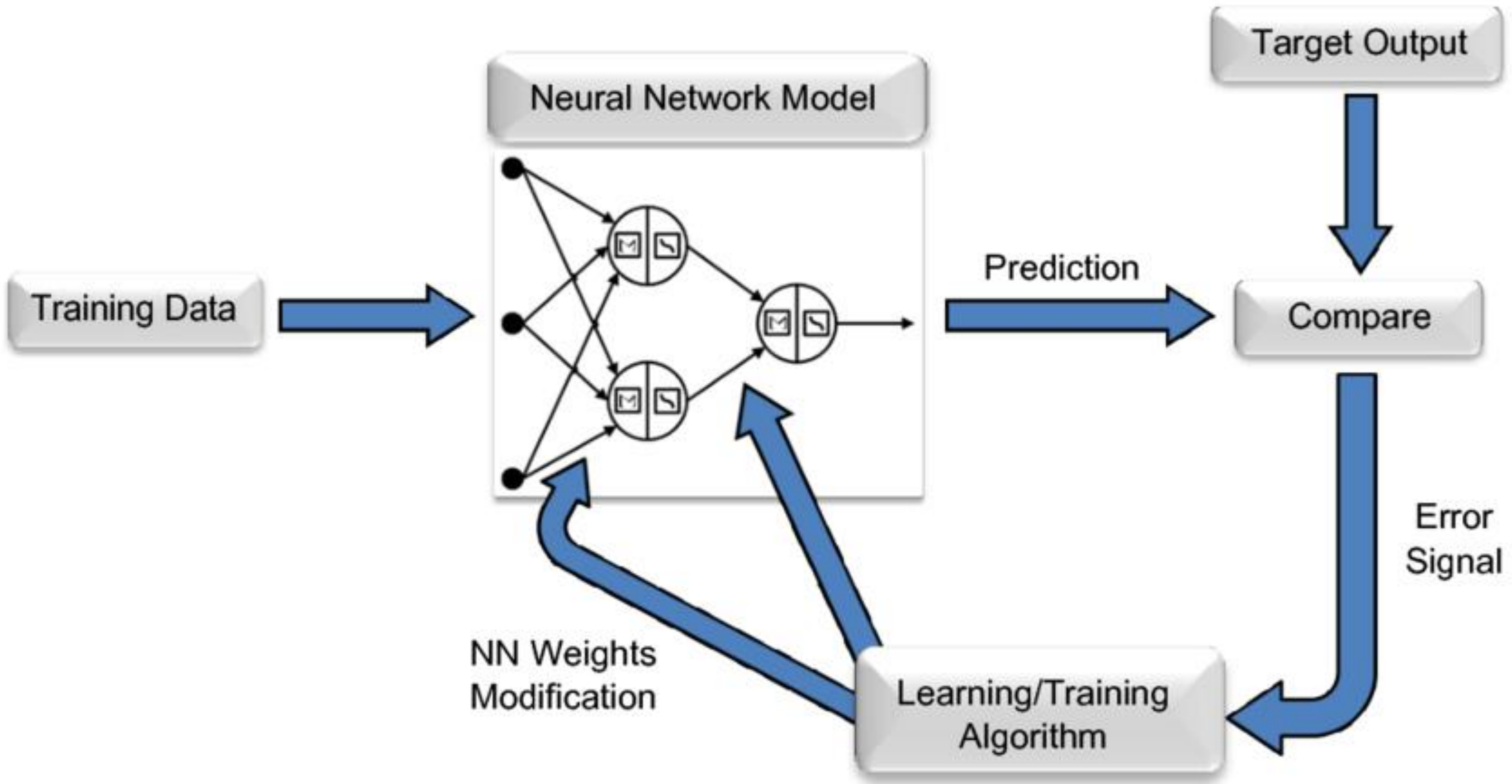
- We will cover:
  - Loss Functions
  - Optimisation Algorithms (SGD, ADAM etc.)
  - Local and Global Minima, local Maxima, Global Maxima
  - Learning Rate
  - Configuring Hyperparameters



Cost Functions and Optimisers

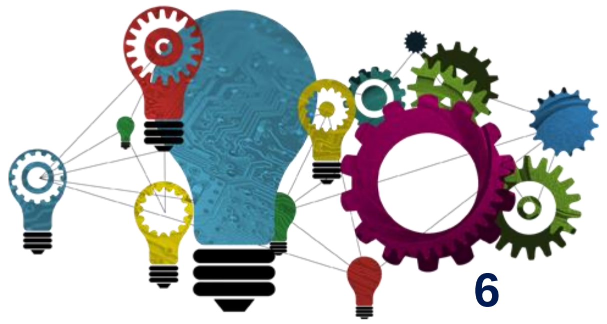
- The network provides an estimation of what it predicts the output class to be
- One of the key challenges in ANNs is how to efficiently evaluate the model's prediction and determine how close or far off the model is from the correct class (this done by comparing the predicted output to the ground truth)
- Atter this evaluation there is a requirement to update the networks weights and biases to improve the model's output and reduce its error
- This is known as the learning phase of the process and is done in similar way to how humans learn whereby; we study and update our own internal models as we gain a better understanding

Cost Functions and Optimisers



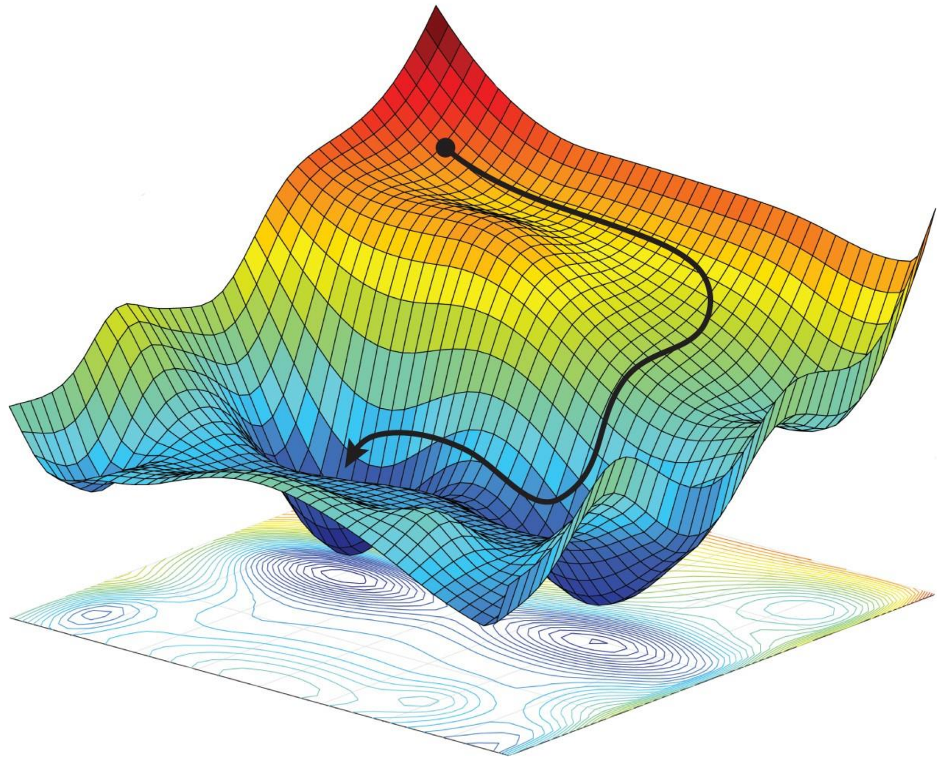
Cost Functions and Optimisers

- With ANN we need to take the estimated outputs of the network and then compare them to the real values of the label
- In this stage we are using the training dataset during the fitting/training of the model
- The cost function (often referred to the loss function must be an average so it can output a single value)
- We need to keep track of our loss during the training phase to monitor network performance
- Cost functions enable us the evaluate the performance of a perceptron by telling us how far off we are from the target value



Cost Functions

- There are different cost functions, but the most popular is Cross Entropy
- The benefits of Cross Entropy are that enable faster training
- Typically, the larger the difference, the faster the neuron can learn
- The next step is to use our neurons and our measurement of error (cost function) to correct the networks prediction (this is the learning part of the process)
- This is known as optimisation and is undertaken by optimisation algorithms

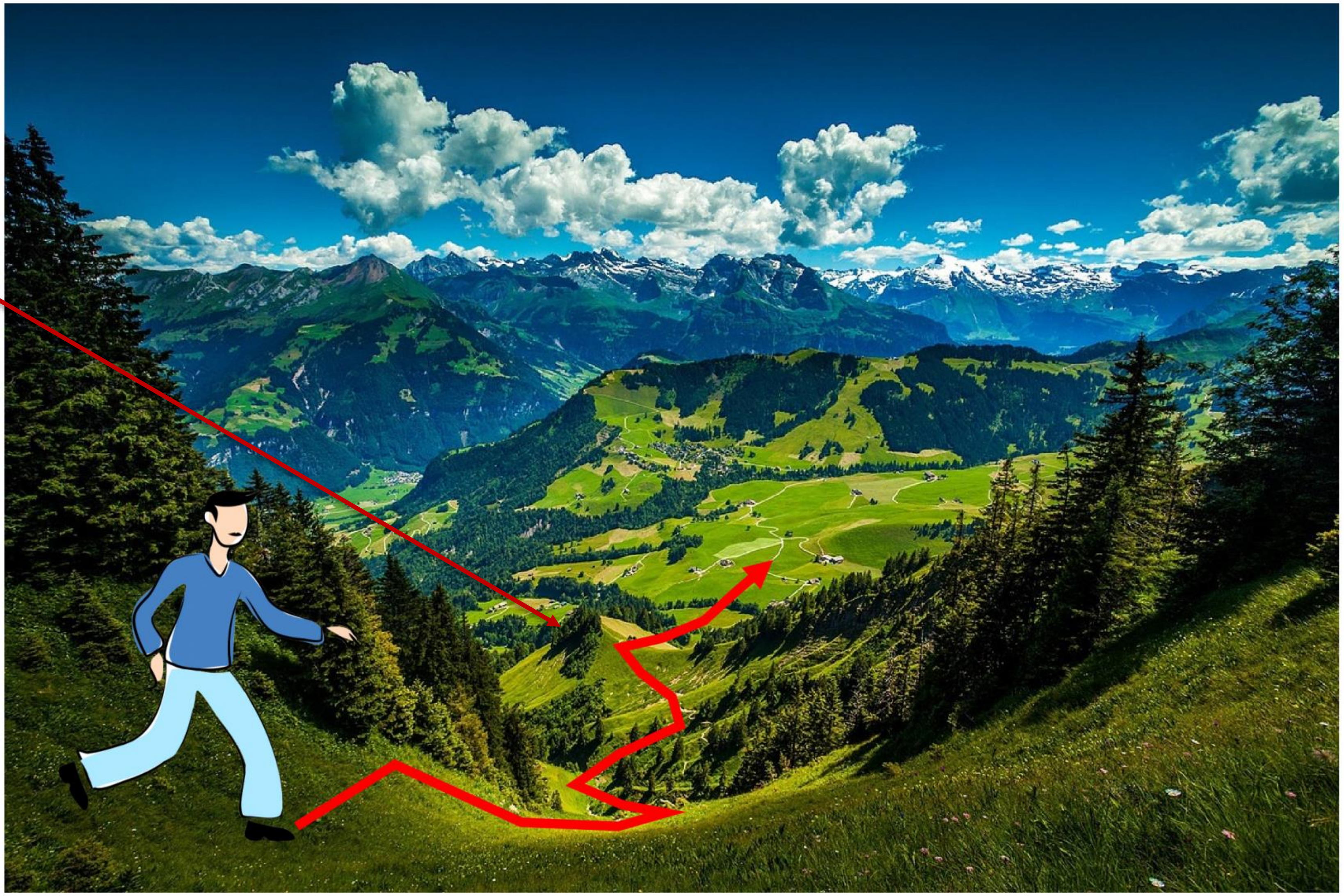


Optimisation Example

- Imagine we are baking a loaf a bread and you need to weight 380g of flower
- You start by dumping an amount of flour into the scale and reading the digital display to see how much the flower weighs. As with most scenarios the amount of flower added will be either over or under your target value of 380g
- To correct this, you gradually add or remove spoonful’s of flower unit you get to your target of 380g (note that this might take a number of iterations (epochs) until you reach the desired target).
- This scenario is essentially how optimisation and the cost function work in an ANN

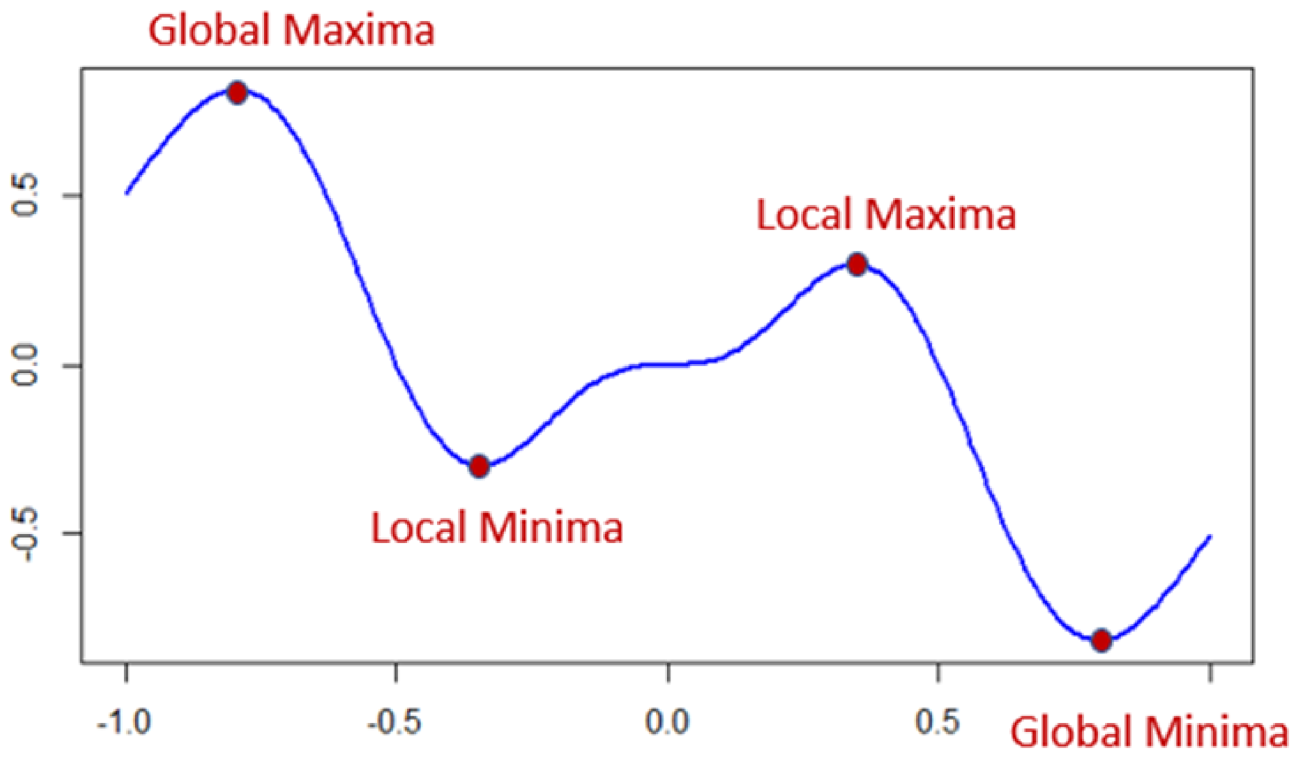
Optimisation

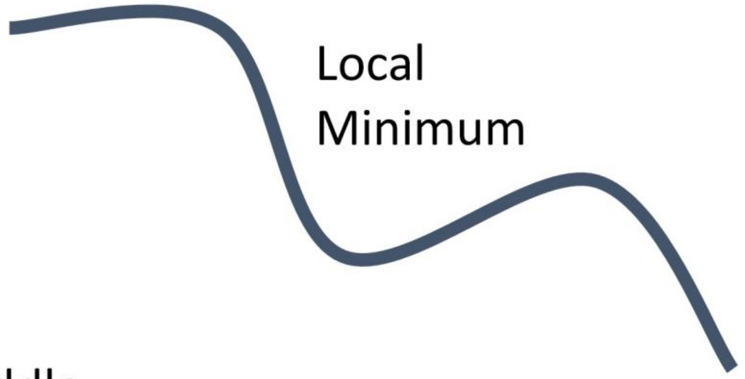
Each x,y point on the ground is the Value of our wright Matrix (w)

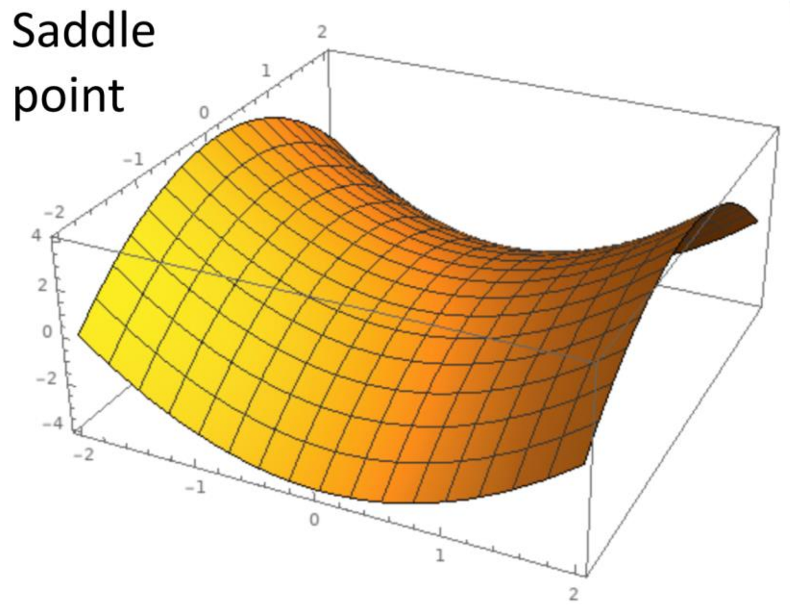


Height of peaks
Is the value of cost function L(w)

Global and Local Minima

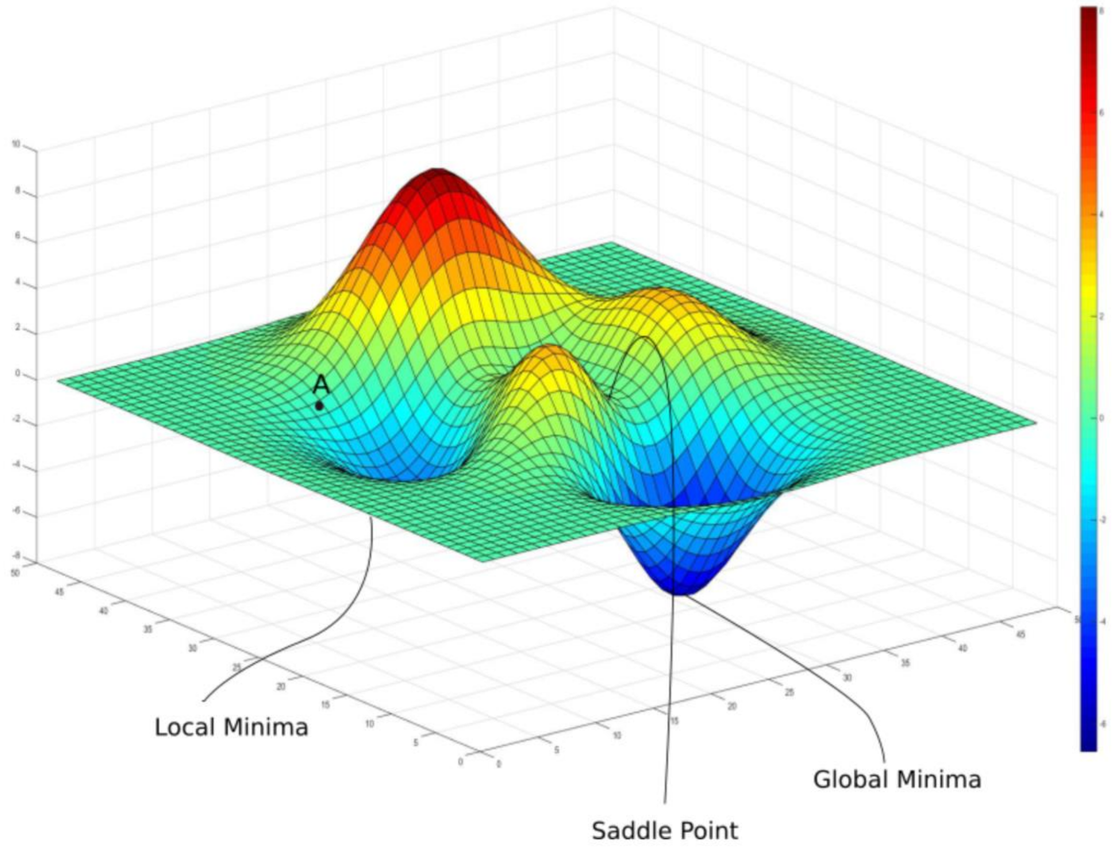






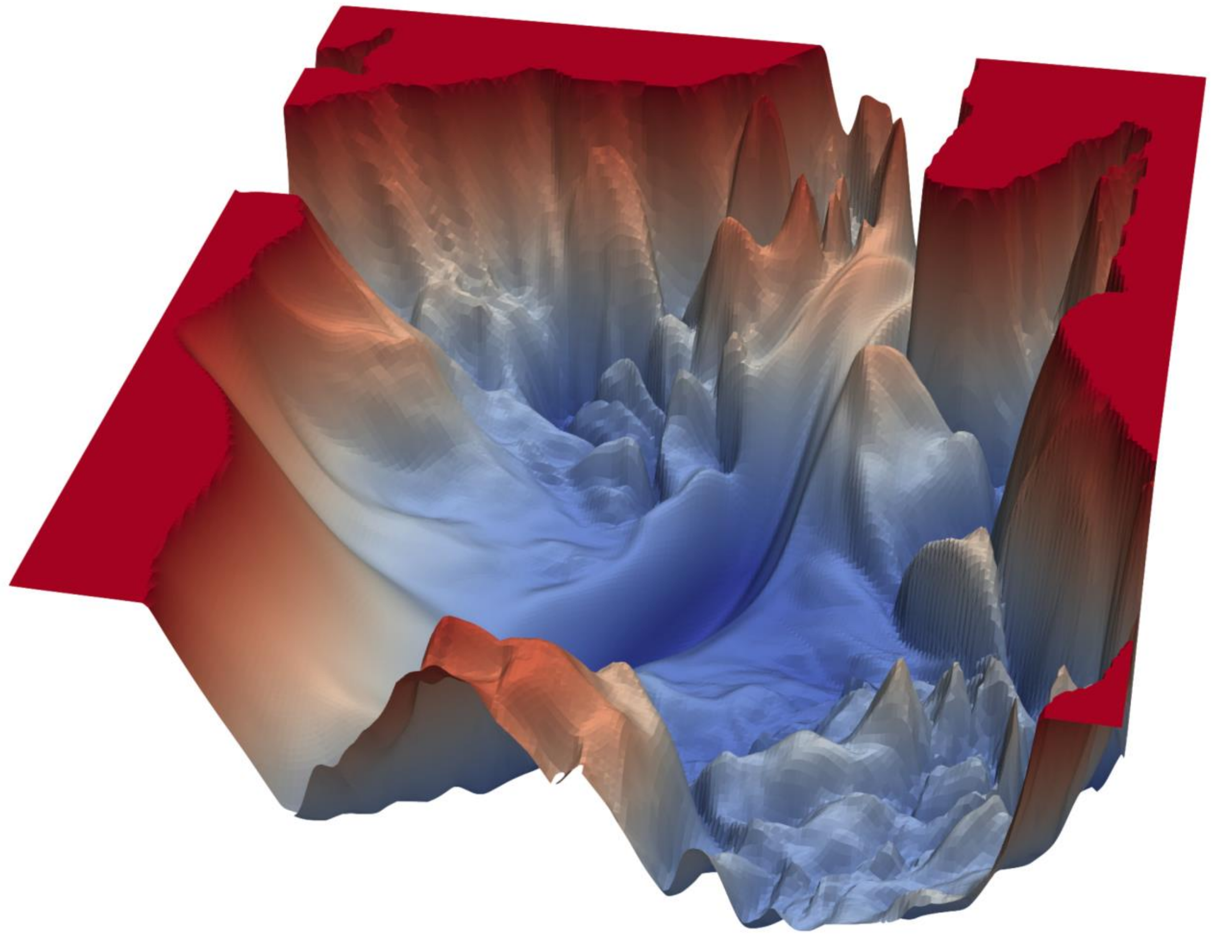
Global and Local Minima

- Often there are many local minima's however these are often close to the global minima
- This might provide an advantage when it comes to overfitting as the global minima is attained on the training set
- Look at the initialization of A were not going to be able to reach the global minima



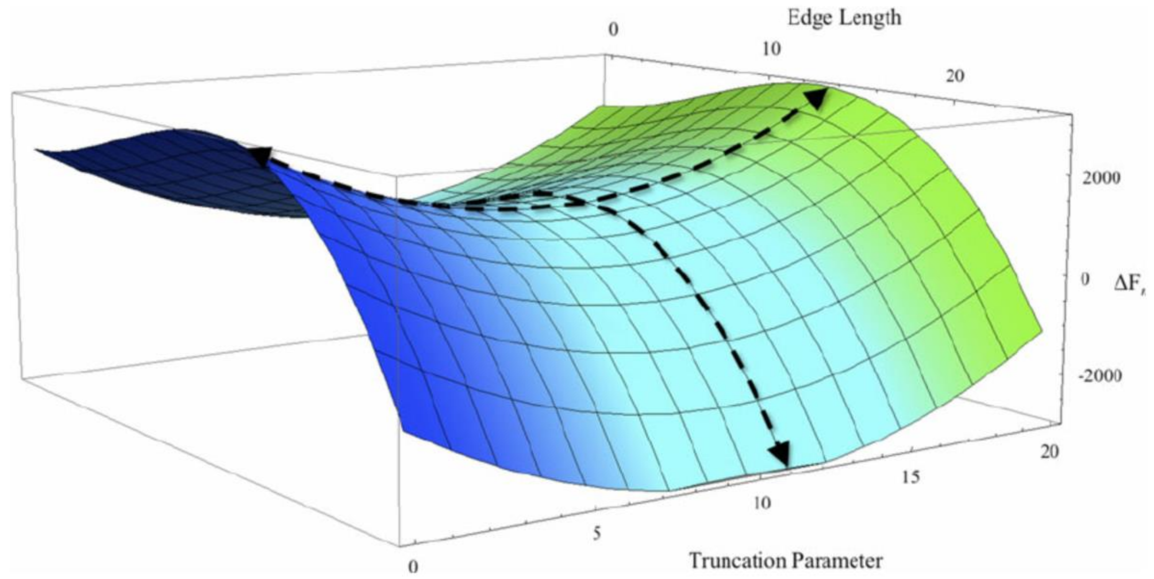
Global and Local Minima

- A complicated loss landscape



Saddle Point

- it's a minima in one direction (x), it's a local maxima in another direction, and if the contour is flatter towards the x direction, GD would keep oscillating to and from in the y – direction
- This gives the illusion that we have converged to a minima
- Not a problem for SGD due to randomness

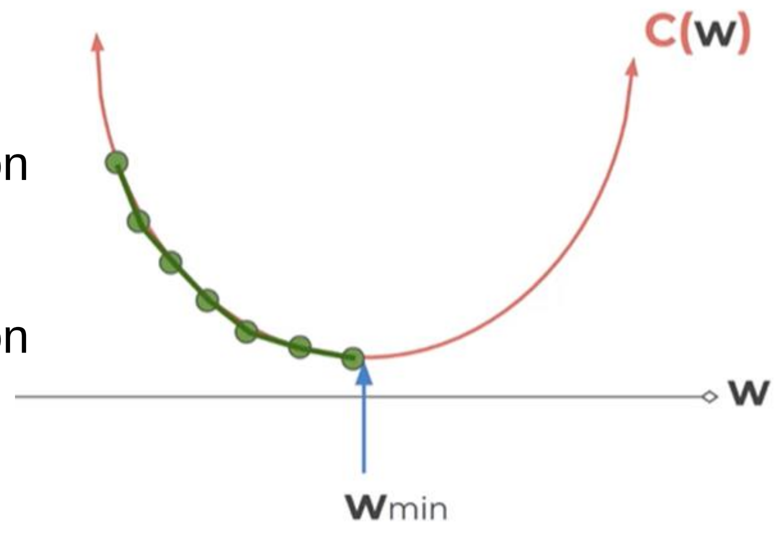


Convex and non-convex functions

1) Convex Functions
Occurs in most machine learning problems

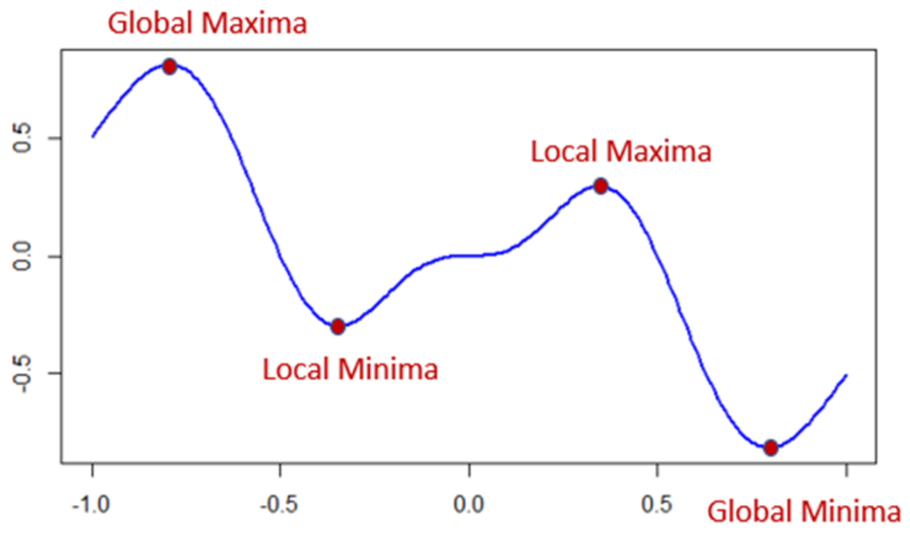
Linear
Regression

Logistic
Regression
etc.



Has only one global minima

2) Non-Convex Functions
Common in Deep Learning



There are so many weights and
hyperparameters at play

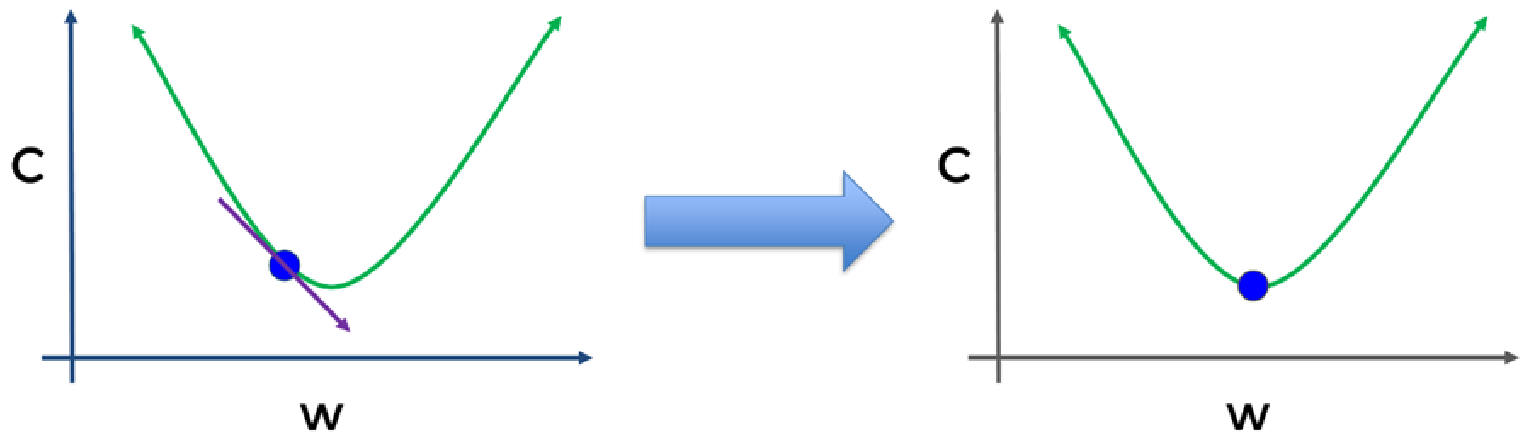
So how do we get to the bottom (convergence)?

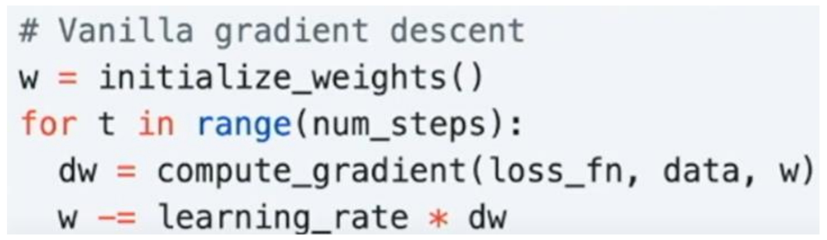
- In some situations we know the answer. For certain types of optimisation problems in linear regression we have an explicit equation
- For most problems this doesn't work so we need to use iterative methods (epochs)
- Modern algorithms follow the slope to achieve the desired performance
- This is where we can use derivatives to find the slope
- In 1-dimension, the derivative of a function gives the slope

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Optimisation Algorithms Gradient Decent

- In order to calculate and reduce the loss we use a set of algorithms known as optimisation algorithms
- One of the most popular optimisation algorithms is Gradient Decent which is an optimisation algorithm for finding the minimum of a function (in our case the cost function)
- Iteratively step in the direction of the negative gradient (direction of local steepest descent)





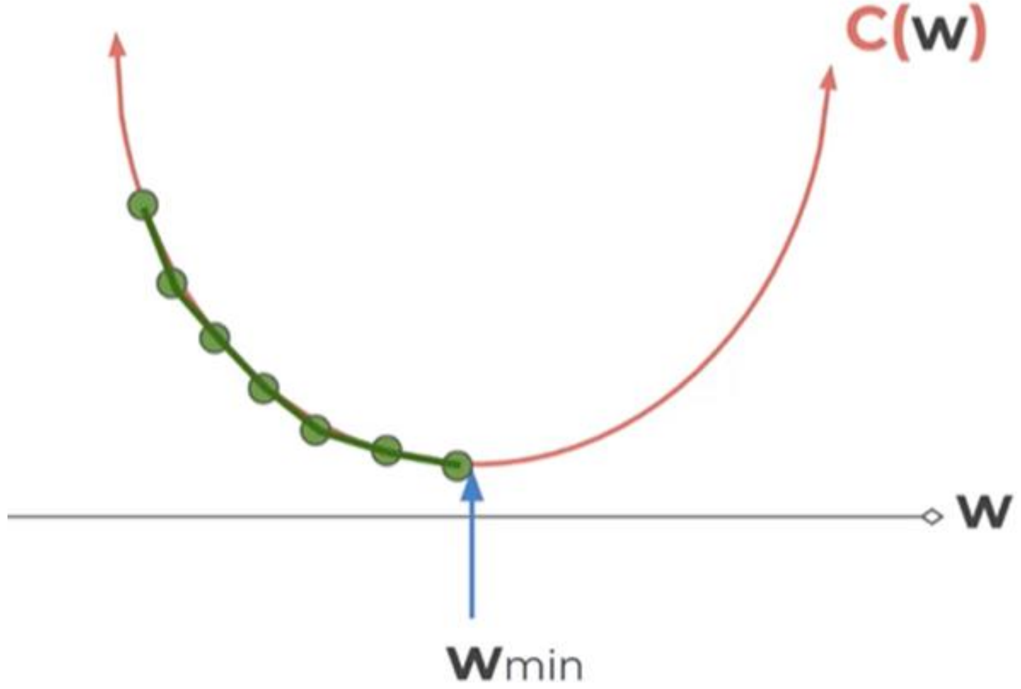
```
# Vanilla gradient descent
w = initialize_weights()
for t in range(num_steps):
  dw = compute_gradient(loss_fn, data, w)
  w -= learning_rate * dw
```

Optimisation Algorithms Gradient Decent

- Gradient Decent works by calculating the slop as a given point and moving in the downwards direction of the slope
- This is repeated until we converge to zero indicating the minimum. Smaller step sizes take longer to find the minimum
- The step size is known as the learning rate. One of the key challenges in gradient descent is setting the optimal learning rate
- If you set it too small, it may take a significant amount of time before your network converges (reaches the global minimum). However, if you set the learning too high you will create a ping pong effect in the bowl whereby the learner overshoots the global minima and never reaches it

Gradient Decent Learning Rate

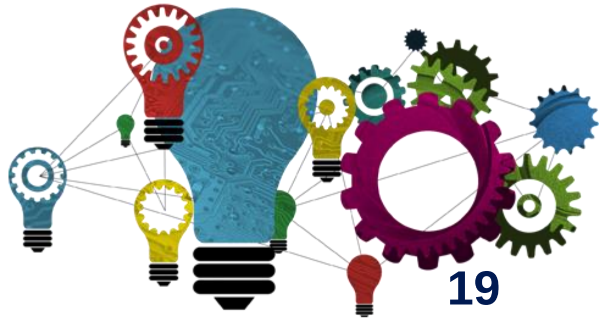
- One of the issues with traditional gradient decent is that to can get stick in a local minimum instead of the global minima
- It is possible to alter the learning to address this issue, however, several optimisation techniques have been developed to overcome this issue



Hyperparameters:
- Weight initialization method
- Number of steps
- Learning rate

Gradient Decent

- With gradient decent we calculate the gradient of the cost function by summing the cost for each example. This can be inefficient, if we had 3 million examples we would have to loop through 3 million times!!
- So to move a single step we would have to calculate the cost 3 million times
- Becomes difficult to train the models quickly if you have large training set especially in case of Deep learning
- But it does guarantee convergence with the local minima
- Which means that every update will decrease the error
- Only really used in convex functions

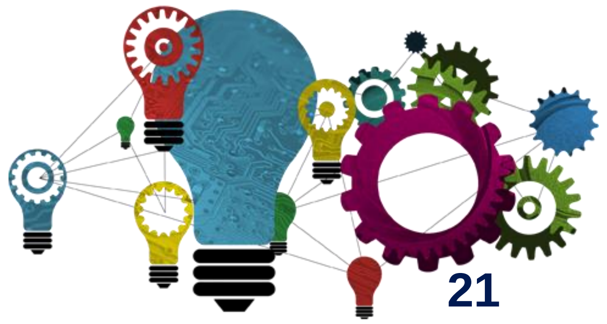


Gradient Decent (Batch/Online Learning)

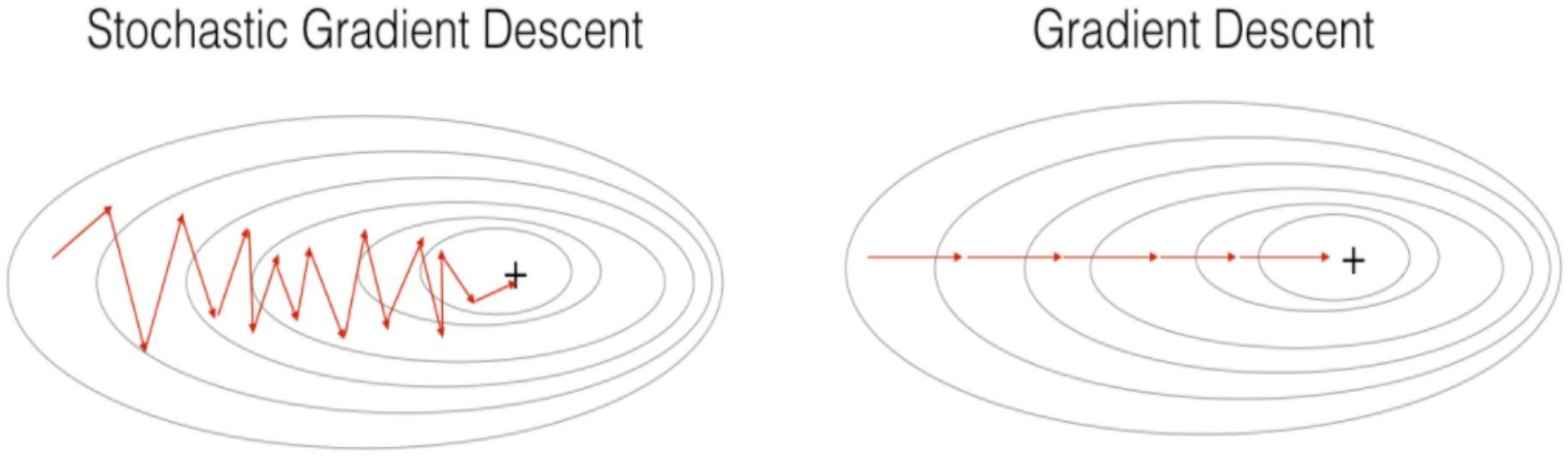
- In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data
- This will generally take a lot of time and computing resources, so it is typically done offline
- First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning
- In online learning, you train the system incrementally by feeding it data instances sequentially, either individually or by small groups called mini-batches. Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives
- Gradient decent cant be efficiently used for online learning

Stochastic Gradient Decent

- With SGD, we are using the cost gradient of 1 example at each iteration, instead of using the sum of the cost gradient of ALL examples
- Depending on the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update
- SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily

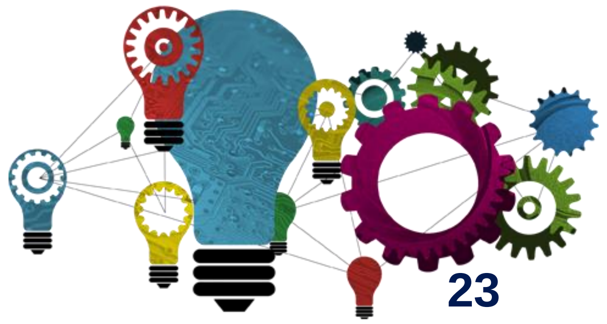


Stochastic Gradient Decent



Mini-Batch Gradient Decent

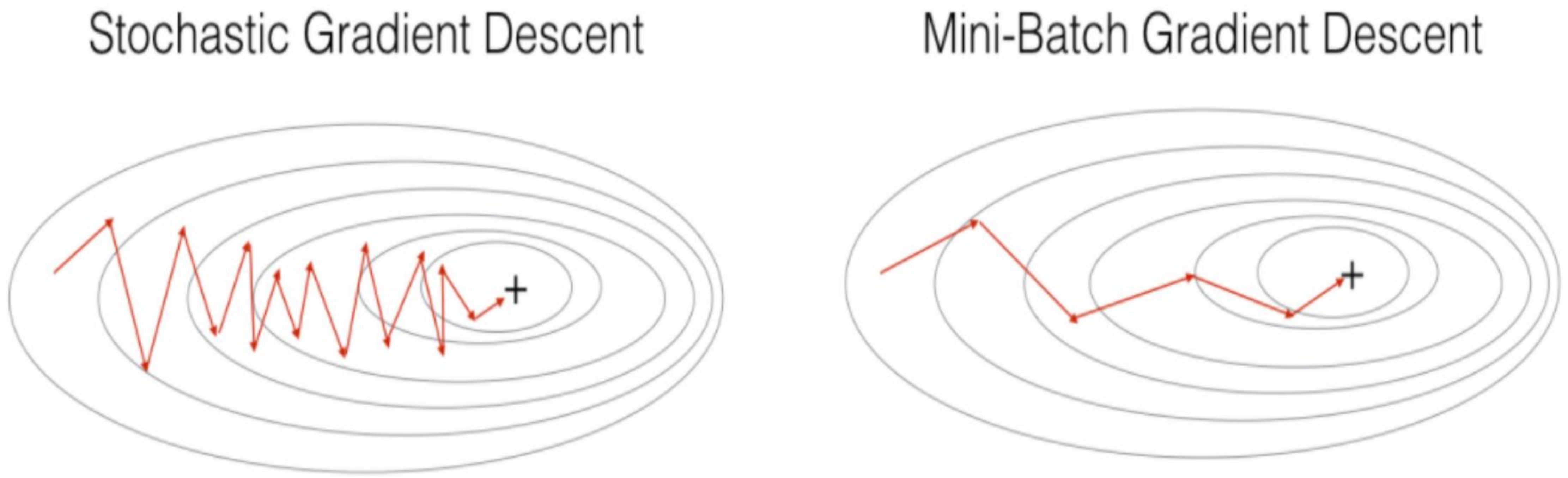
- Instead of implementing gradient descent on the entire training set, we can split our training set into smaller sets and implement gradient descent on each batch one after the other
- This makes the algorithm faster and more efficient
- In contrast with batch gradient descent where cost function smoothly decays, mini-batch gradient has a noisy cost function but still trends downwards
- The most important thing is to select the size of each mini batch. If size is very large then it will again behave as batch gradient descent (very slow) and if it is very small then each example will be a mini batch and it becomes stochastic gradient descent with very large noise
- Samples need to be randomised



Mini-Batch Gradient Decent

- Mini-Batch Gradient Decent can be computed in parallel across multiple compute nodes
- We can then do the weighted sum and update
- Small batches offer a form of regularisation
- Efficient approach for deep learning. Fewer updates to the model means this variant of gradient descent is more computationally efficient than stochastic gradient descent
- Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm

Mini-Batch Gradient Decent



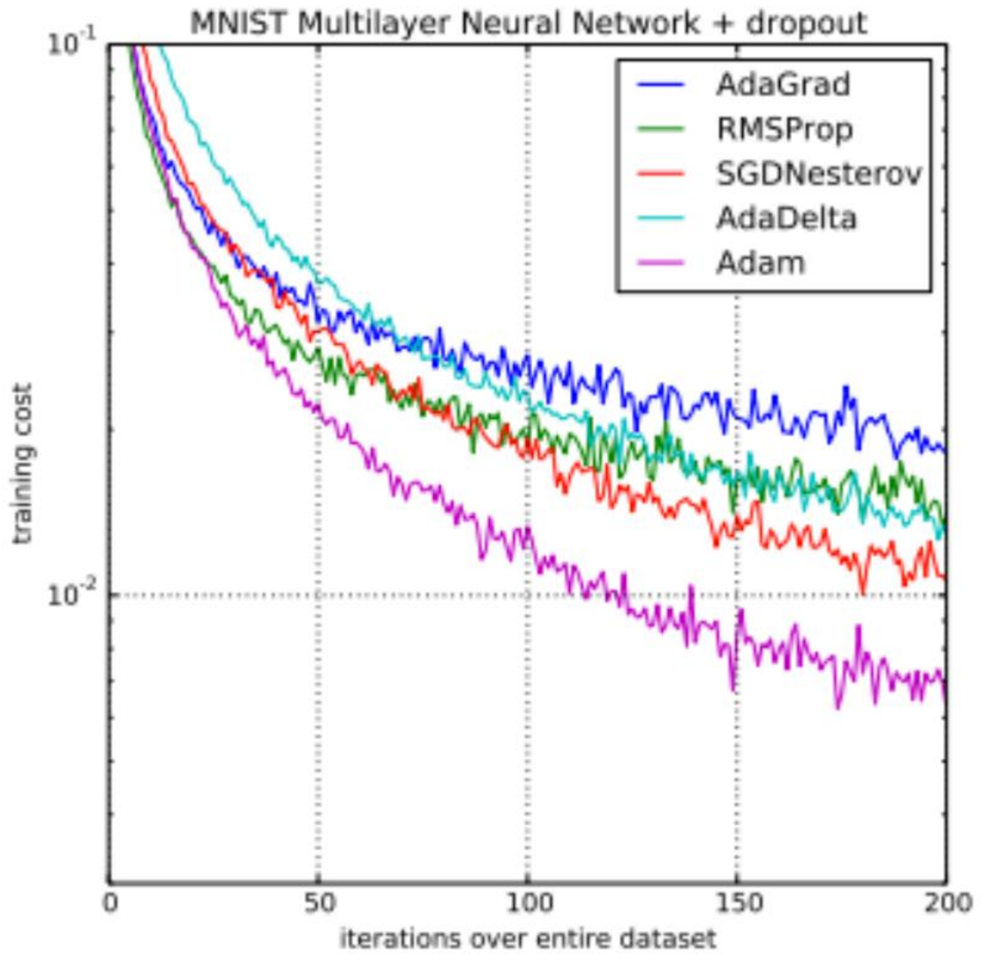
Stochastic Gradient Decent Learning Rate

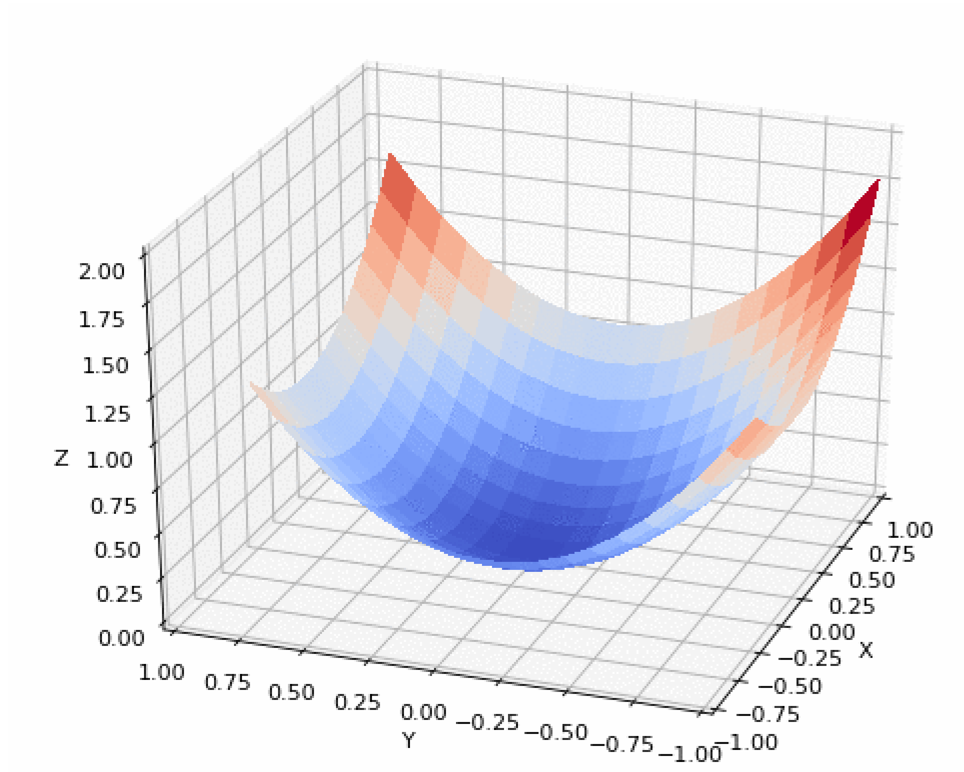
- With SGD the learning rate is fixed
- The learning rate determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley
- We can use other optimisers which can cleverly adapt our learning rate (step size) automatically
- We could start with larger step and then reduce them as the gradient begins to level off (this is known as adaptive gradient decent)

Adam

- One of the most common optimisers used in ANN's is Adaptive Moment Estimation (Adam)
- Adam has recently seen broader adoption for deep learning applications in computer vision and natural language processing
- Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training
- It computes adaptive learning rates for each parameter
- Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled “Adam: A Method for Stochastic Optimization”

Adam


MNIST Multilayer Neural Network + dropout
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam
training cost
iterations over entire dataset
10^{-1}
10^{-2}
0
50
100
150
200


Z
X
Y
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

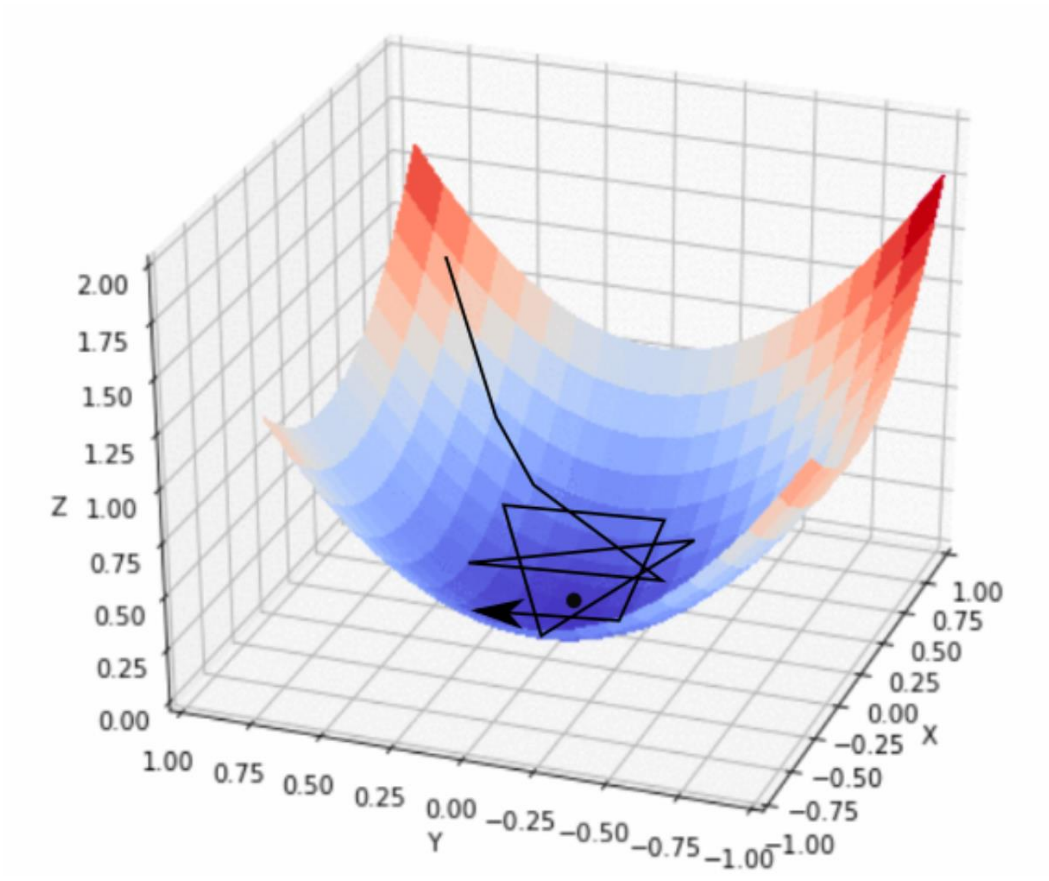
Optimiser Comparisons

Algorithm	Tracks first moments (Momentum)	Tracks second moments (Adaptive learning rates)	Leaky second moments	Bias correction for moment estimates
SGD	✗	✗	✗	✗
SGD+Momentum	✓	✗	✗	✗
Nesterov	✓	✗	✗	✗
AdaGrad	✗	✓	✗	✗
RMSProp	✗	✓	✓	✗
Adam	✓	✓	✓	✓

Configuring Hyperparameters

- Pipeline.config

```
train_config: {
  batch_size: 1
  sync_replicas: true
  startup_delay_steps: 0
  replicas_to_aggregate: 8
  num_steps: 25000
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        cosine_decay_learning_rate {
          learning_rate_base: .0004
          total_steps: 25000
          warmup_learning_rate: .0004
          warmup_steps: 2000
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
```



Next Session

- Introduction to backpropagation
- Chain rule
- Computational graphs
- The vanishing gradient
- Overcoming the vanishing gradient issue
- Weight Initialisation
- Regularisation