Artificial Intelligence and Machine Learning
Lecture #5

Amin Noroozi

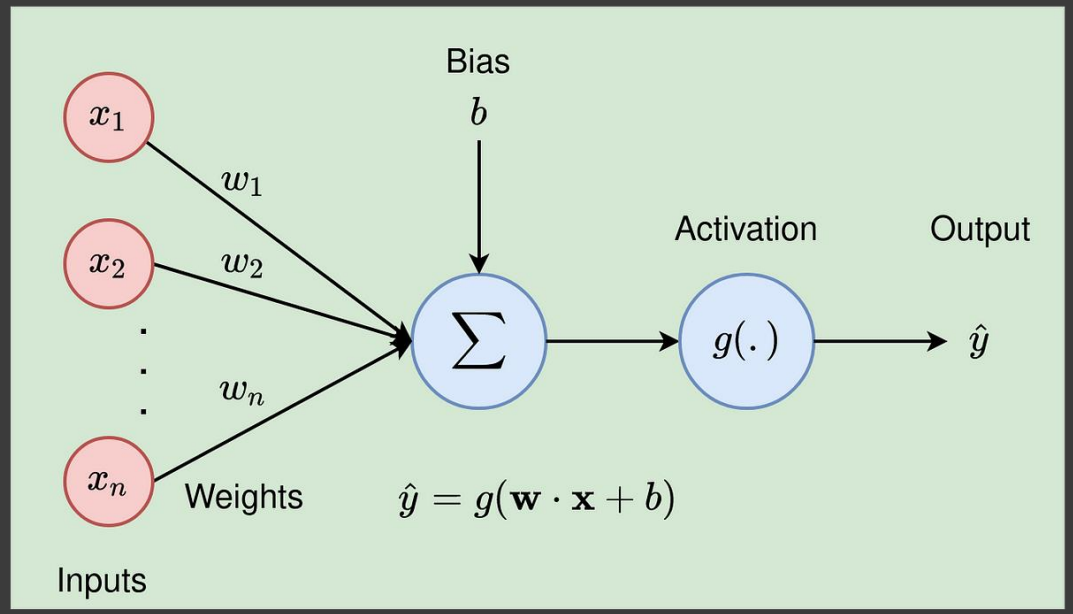
University of Wolverhampton

a.noroozifakhabi@wlv.ac.uk

https://www.linkedin.com/in/amin-n-148350218/

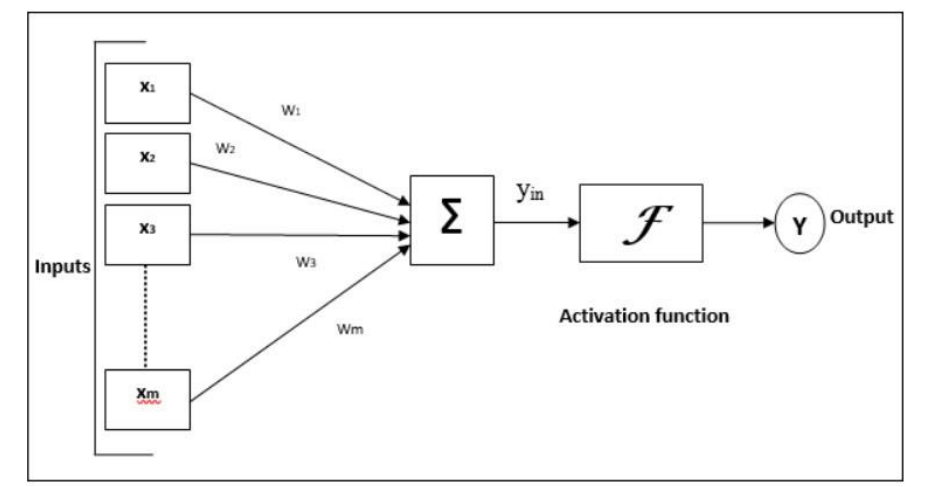
❖ Recap

A single neuron can be modelled using a linear regression model and an activation function.



❖ Recap

In a simplified expression, we can drop the bias term



$$y_{in} = x_1.w_1 + x_2.w_2 + x_3.w_3 \ldots x_m.w_m$$

i.e., Net input $y_{in} = \sum_i^m x_i.w_i$

The output can be calculated by applying the activation function over the net input.

$$Y = F(y_{in})$$

❖ Recap

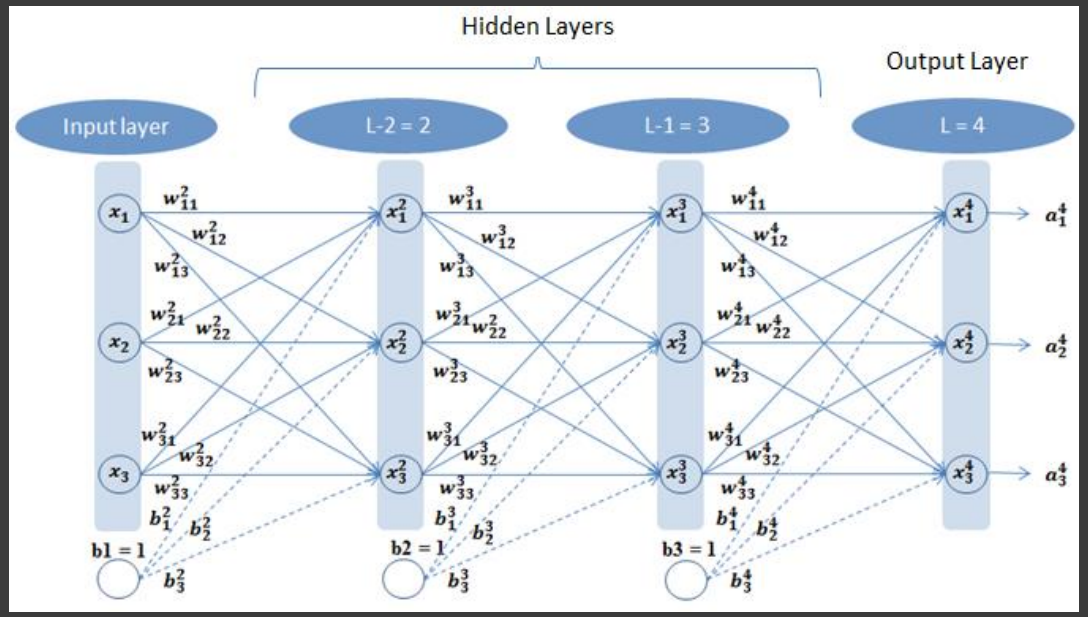
In a simplified expression, we can drop the bias term

$$\hat{y} = Xw$$

$$\boldsymbol{X} = \begin{pmatrix} x_1^0 & x_1^1 & \dots & x_1^n \\ x_2^0 & x_2^1 & \dots & x_2^n \\ \vdots & \vdots & \vdots & \vdots \\ x_m^0 & x_m^1 & \dots & x_m^n \end{pmatrix} \quad w = \begin{pmatrix} w^1 \\ w^2 \\ \vdots \\ w^n \end{pmatrix} \quad \hat{\boldsymbol{y}} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$$

❖ Recap

We can form a neural network by arranging the neurons in different layers



$$Y = f_2(W_2 f_1(W_1 X + b_1) + b_2)$$

where f_i are activation functions

$$W_1 = \begin{bmatrix} w_{11}^2 & w_{21}^2 & w_{31}^2 \\ w_{12}^2 & w_{22}^2 & w_{32}^2 \\ w_{13}^2 & w_{23}^2 & w_{33}^2 \end{bmatrix}$$

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

❖ Recap

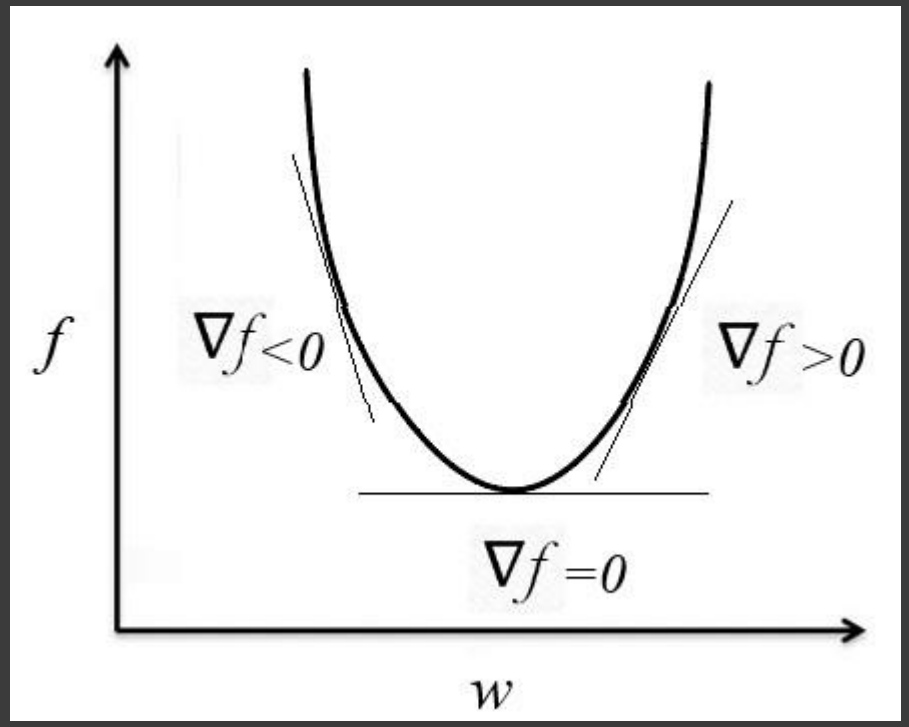
- The goal is to find the network weights such that the following function is minimised:

$$f = \frac{1}{N} \sum_{i=1}^{N} \|t_i - y_i\|^2$$

where t_i and y_i are real targets and network predictions, respectively.

There are two methods to find the weights: Analytical solution and gradient descent

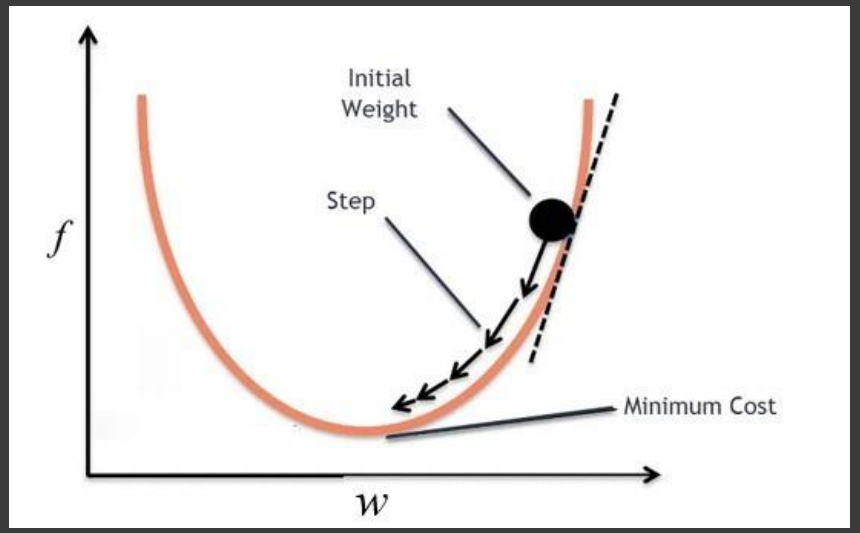
❖ Recap



In the analytical solution, we set $\nabla f=0$ to find the weights. But calculating the gradient is not always possible

❖ Recap

- **Gradient descent**



In the gradient descent method, we initialise the weights randomly, and then iteratively update the weights to gradually move them in the opposite direction of the gradient until we get to the optimum weights (the minimum cost point)

❖ Recap

➢ Gradient descent

Step 1: Initialise the network weights randomly (k=1, i. e. iteration one). Set a threshold $\mathcal{E}$ (this threshold is used later to stop the algorithm)

Step 2: Update the wights in the opposite direction of the gradient of the cost function *f* (aka objective or error function) using the following formula

$$w^{k+1} = w^k - \eta^k \nabla f(w^k, x)$$

Step 3: Calculate the cost function

Step 4: If the difference between the cost function value in the current iteration and the previous iteration is smaller than ε, then terminate the algorithm. Otherwise, go to Step 2.

❖ Recap

➢ Gradient descent

Question: Which x value should we use to update the weights in Step 2?

$$w^{k+1} = w^k - \eta^k \nabla f(w^k, x)$$

Batch gradient descent (BGD): Calculate the gradient for all x values and then calculate the average

Mini Batch gradient descent: Use a batch of data to calculate the average gradient value

Stochastic gradient descent (SGD): use a random x to calculate the gradient value

❖ Regularisation

Like linear regression, we can use regularisation to adjust weights in neural networks.

$$f = \sum_{i=1}^{N} (t_i - y_i)^2 + \lambda \sum_{i=1}^{M} (w^j)^2$$

We have two types of regularisation in neural networks: L1 and L2.

The cost function formula is slightly different in the two regularisation methods but the logic behind them is the same.

❖ Regularisation

- **Different regularisation for neural networks**

We have the regression equation $y=Wx+b$, where x is the input, W the weights matrix and b the bias.

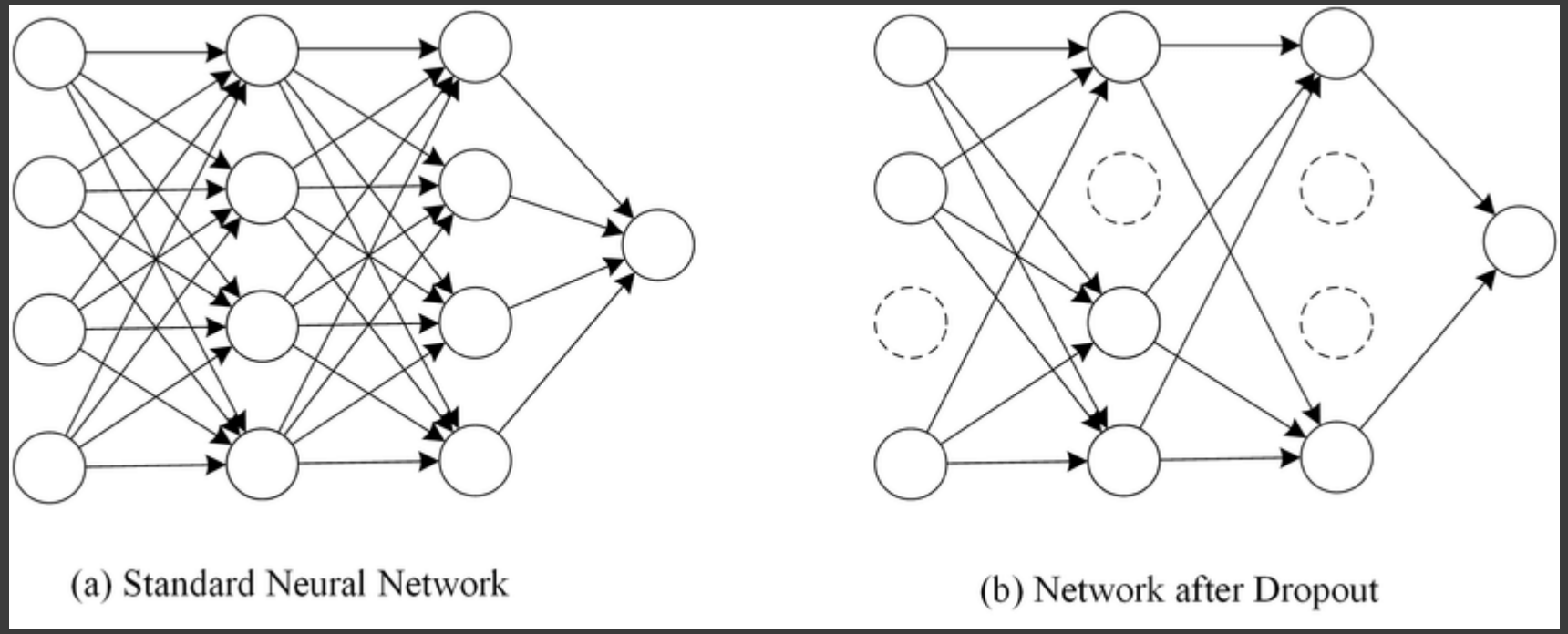
Kernel Regularizer: Tries to reduce the weights W (excluding bias).

Bias Regularizer: Tries to reduce the bias b.

Activity Regularizer: Tries to reduce the layer's output y, thus reducing the weights and adjusting bias so $Wx+b$ is the smallest.

❖ Drop out

Alternatively, we can use the drop out method to adjust the weights



❖ Drop out

The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network

All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.

❖ Drop out

Let's try to understand with a given input x: {1, 2, 3, 4, 5} to the fully connected layer.

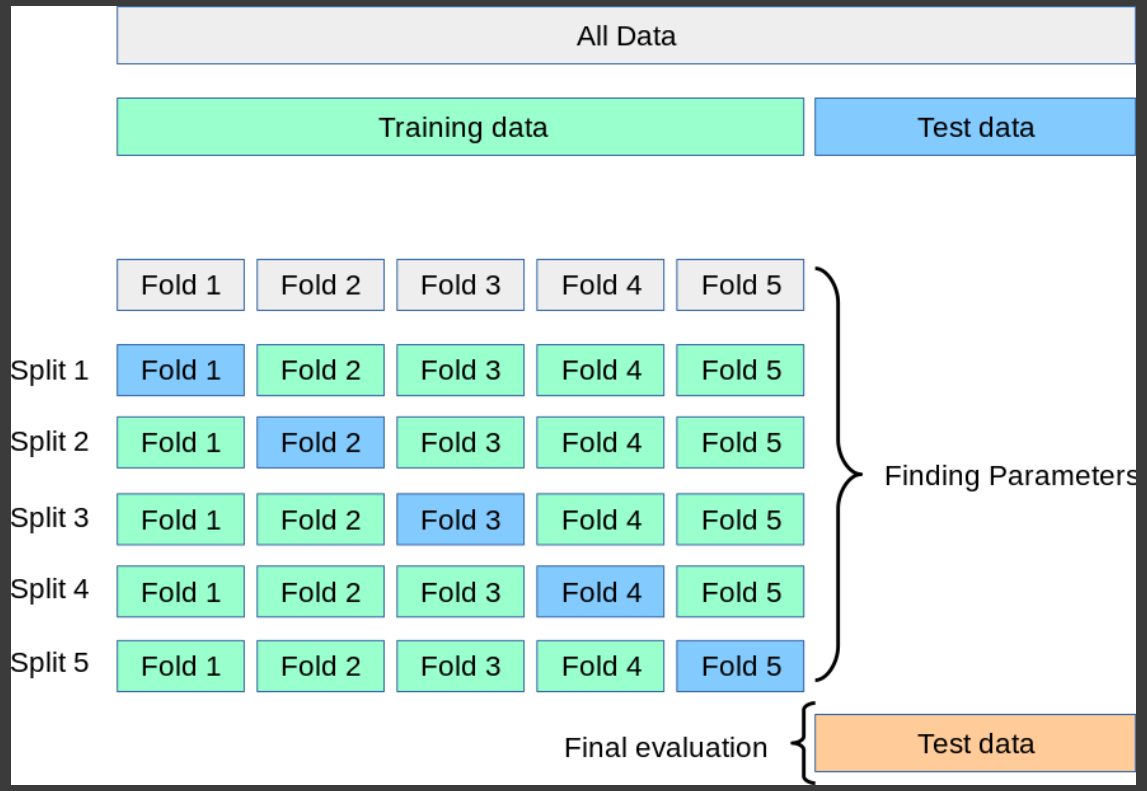
Assume we have a dropout layer with probability p = 0.2 (or keep probability = 0.8).

During the forward propagation (training) from the input x, 20% of the nodes would be dropped, i.e. the x could become {1, 0, 3, 4, 5} or {1, 2, 0, 4, 5} and so on. Similarly, it is applied to the hidden layers

❖ Drop out

For instance, if the hidden layers have 1000 neurons (nodes) and a dropout is applied with drop probability = 0.5, then 500 neurons would be randomly dropped in every iteration (batch).

❖ Cross validation



❖ Cross validation

In the cross-validation (CV), the training data is split into K folds

In each iteration, one of the folds is reserved for testing (validation) and the rest of the folds are merged to train the model

This process is repeated until all folds are used for testing (validation) at least once

The final accuracy will be the average of the model's accuracies over all folds