Deep Learning

Introduction

A.I.

- Any technique which enables computer to mimic human behaviour
- it include expert systems, rule-based or other hard-coded algorithms.

Machine Learning

- AI techniques that give computer the ability to learn without being explicitly programmed to do so
- Algorithms are used to parse data, learn from it, and then make a decision.

Deep Learning

- A subset of ML which make the computation of multi-layer neural networks feasible
- It deals with multi-layer neural network with more neurones, more layers and more interconnectivity.
- Deep learning to learn rich features automatically through feature learning process.

	Machine Learning	Deep Learning
Feature Engineering	Features need to be identified by expert & hand coded	Learn features from data
Data Dependencies	ML algorithm can work with small amount of data because of handcrafted rule	Large data is required to understand problem perfectly
Hardware Dependencies	Low end machine can used	Large amount of matrix multiplication operations, to Optimise performance GPU is required
Problem Solving Approach	Modular approach	Solve problem end-to-end
Performance	Less training time but more testing time (large data set)	More training time but testing time is less

---

Deep Learning Models

Supervised Models

- Classic Neural Networks (Multilayer Perceptrons)
- Convolutional Neural Networks (CNNs)

- Recurrent Neural Networks (RNNs)

Unsupervised Models

- Self-Organising Maps (SOMs)
- Boltzmann Machines
- AutoEncoders

Classic Neural Networks (Multilayer Perceptrons)

A neural network is structured like the human brain and consists of artificial neurones, also known as nodes. These nodes are stacked next to each other in three layers:

- The input layer
- The hidden layer(s)
- The output layer

Data provides each node with information in the form of inputs. The node multiplies the inputs with random weights, calculates them, and adds a bias. Finally, nonlinear functions, also known as activation functions, are applied to determine which neurone to fire.

Convolutional Neural Networks (CNNs)

CNN consist of multiple layers and are mainly used for image processing and object detection. CNN is built to handle a greater amount of complexity and computation of data.

CNN's have multiple layers that process and extract features from data:

- Convolution Layer - Convolution layer that has several filters to perform the convolution operation, a process in which feature maps are created out of input data. A activation function(ReLU) is then applied to filter feature maps.
- Pooling Layer - The feature map next feeds into a pooling layer. Pooling is a down-sampling operation that reduces the dimensions of the feature map.
- Flattening layer - The flattening layer then converts the resulting two-dimensional arrays into a linear vector.
- Fully Connected Layer - A fully connected layer forms when the flattened layer output is fed as an input, which classifies and identifies the images.
- eg - Image Classification, Object Detection, Video Recognition, Medical Image Analysis, Recommender Systems

CNN limitation:

- Rely on examples being vectors of fixed length
- All inputs are independent of each other.

Recurrent Neural Networks (RNNs)

- Recurrent Neural Network used to process variable length sequences of inputs
- RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations.
- RNN have a memory which captures information about what has been calculated so far.
- eg - Machine Translation, Video Activity Recognition, Speech Recognition, Music Generation

eg - Machine Translation, Video Activity Recognition, Speech Recognition, Music Generation

Auto Encoder

- Auto-encoders are a specific type of feedforward neural network in which the input and output are identical.
- An auto-encoder is a feed-forward neural net used to reproduced the input
- Map high-dimensional data to lower dimensions for visualization
- Learn abstract features in an unsupervised way so these can apply them to a supervised task
- eg - Image Compression, Image De-noising, Image generation

Generative Adversarial Network (GAN)

- GANs are generative deep learning algorithms that create new data instances that resemble the training data. GANs have two components: a generator, which learns to generate fake data, and a discriminator, which learns from that false information.
- GANs can create images that look like photographs of human faces, even though the faces don't belong to any real person.
- eg - Image Editing, Synthetic data Generation, Game Design

Learning Approaches

Supervised learning

- Supervised learning describes a class of problem that involves using a model to learn a mapping between input examples and the target variable.
- Models are fit on training data comprised of inputs and outputs and used to make predictions on test sets where only the inputs are provided and the outputs from the model are compared to the withheld target variables and used to estimate the skill of the model.

Unsupervised learning

- Unsupervised learning describes a class of problems that involves using a model to describe or extract relationships in data.
- Compared to supervised learning, unsupervised learning operates upon only the input data without outputs or target variables. As such, unsupervised learning does not have a teacher correcting the model, as in the case of supervised learning.

Reinforcement learning

- Reinforcement learning describes a class of problems where an agent operates in an environment and must learn to operate using feedback.
- The use of an environment means that there is no fixed training dataset, rather a goal or set of goals that an agent is required to achieve, actions they may perform, and feedback about performance toward the goal.

Semi-Supervised Learning (Hybrid learning)

Semi-Supervised Learning (Hybrid learning)

- Semi-supervised learning is supervised learning where the training data contains very few labeled examples and a large number of unlabelled examples.
- The goal of a semi-supervised learning model is to make effective use of all of the available data, not just the labelled data like in supervised learning.

Self-Supervised Learning (Hybrid learning)

- Self-supervised learning refers to an unsupervised learning problem that is framed as a supervised learning problem in order to apply supervised learning algorithms to solve it.
- Supervised learning algorithms are used to solve an alternate or pretext task, the result of which is a model or representation that can be used in the solution of the original (actual) modelling problem.

Multi-Task Learning

- Multi-task learning is a type of supervised learning that involves fitting a model on one dataset that addresses multiple related problems
- Multi-task learning can be a useful approach to problem-solving when there is plenty of input data labeled for one task that can be shared with another task with much less labeled data.

Active Learning

- Active learning is a technique where the model is able to query a human user operator during the learning process in order to resolve ambiguity.
- Active learning is a type of supervised learning and seeks to achieve the same or better performance of so-called “passive” supervised learning, although by being more efficient about what data is collected or used by the model.

Online Learning

- Online learning involves using the data available and updating the model directly before a prediction is required or after the last observation was made.
- Online learning is appropriate for those problems where observations are provided over time and where the probability distribution of observations is expected to also change over time. Therefore, the model is expected to change just as frequently in order to capture and harness those changes.

Transfer Learning

- Transfer learning is a type of learning where a model is first trained on one task, then some or all of the model is used as the starting point for a related task.
- It is different from multi-task learning as the tasks are learned sequentially in transfer learning, whereas multi-task learning seeks good performance on all considered tasks by a single model at the same time in parallel.

Ensemble Learning

- Ensemble learning is an approach where two or more models are fit on the same data and the predictions from each model are combined.
- The objective of ensemble learning is to achieve better performance with the ensemble of models as compared to any individual model.

Vectorisation

- Process of converting an algorithm from operating on a single value at a time to operating on a set of values at one time.
- Used when require to perform same operation on each element in a data set.
- Vectorisation is the term for converting a scalar program to a vector program.

Normal vs Vector Instruction

- Scalar can only operate on pairs of operands at once whereas Vectorised programs can run multiple operations from a single instruction
- This take advantage of Data Level Parallelism (DLP) within an algorithm via vector processing i.e., processing batches of rows together.

Broadcasting

- Broadcasting refers to the ability (NumPy) to treat arrays of different shapes during arithmetic operations.
- Arithmetic operations on arrays are usually done on corresponding elements.
- If the dimensions of two arrays are dissimilar, element-to-element operations are not possible.
- Due to the broadcasting capability of Numpy Library operations on arrays of non-similar shapes is possible.
- The smaller array is broadcast to the size of the larger array so that they have compatible shapes.

Principles & Limitations of Broadcasting

Broadcasting works on 2 principles:

- The trailing dimensions of A and B should be equal or
- Trailing dimensions of either A or B should be 1.

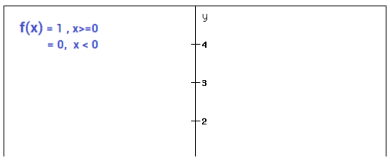
Limitations

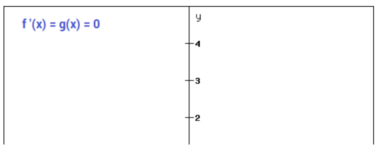
- It can only be performed when the shape of each dimension in the arrays are equal or one has the dimension size of 1.
- The dimensions are considered in reverse order, starting with the trailing dimension.

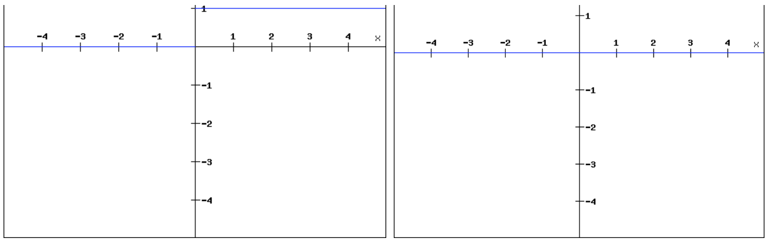
Activation Function

Binary Step Function

- If the input to the activation function is greater than a threshold, then the neurone is activated, else it is deactivated.
- This function will not be useful when there are multiple classes in the target variable.
- The gradient of the step function is zero which causes a hindrance in the back propagation process.

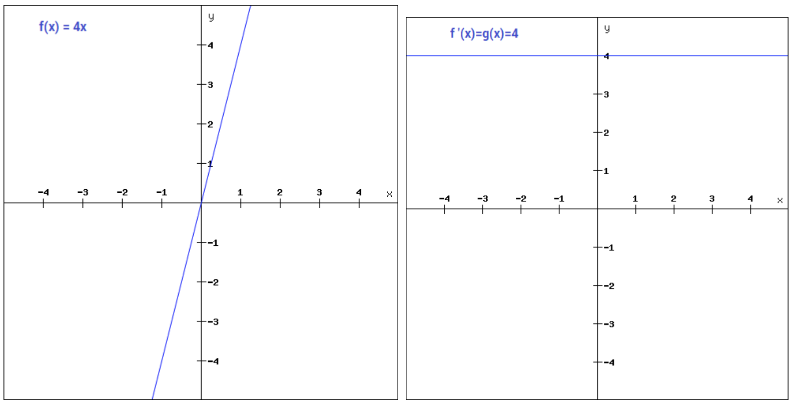






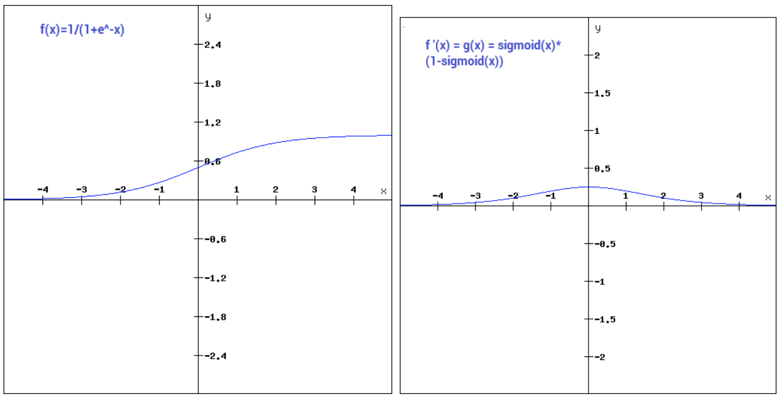
Linear Function

- Activation is proportional to the input
- Although the gradient here does not become zero, but it is a constant which does not depend upon the input value x at all.



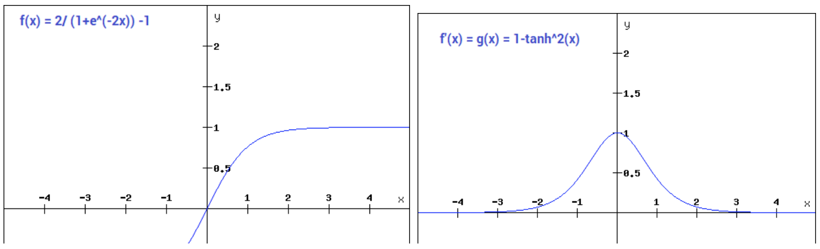
Sigmoid Function

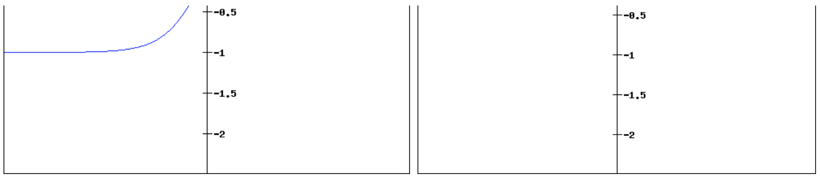
- Most widely used non-linear activation function.
- Sigmoid transforms the values between the range 0 and 1
- This is a smooth S-shaped function and is continuously differentiable.
- The sigmoid function is not symmetric around zero. So output of all the neurones will be of the same sign.



Tanh Function

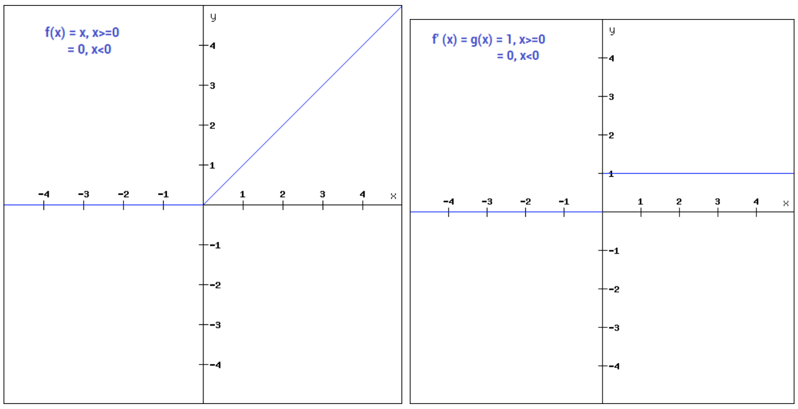
- The range of values in this case is from -1 to 1.
- Thus the inputs to the next layers will not always be of the same sign.
- The gradient of the Tan h function is steeper as compared to the sigmoid function.





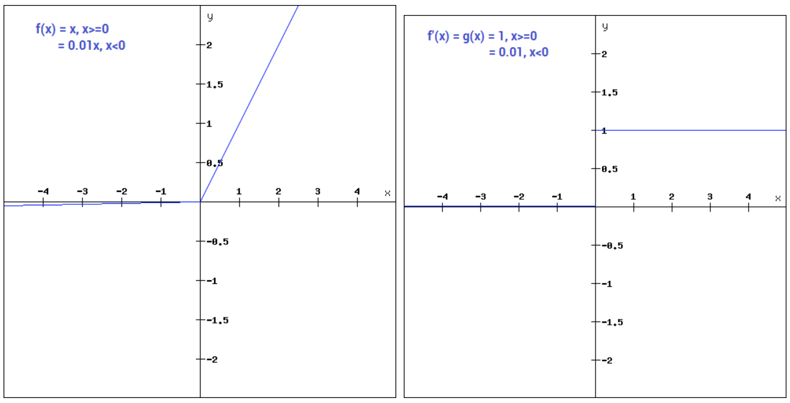
Rectified Linear Unit (ReLU) Function

- The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurones at the same time.
- The neurones will only be deactivated if the output of the linear transformation is less than 0
- During the back-propagation process, the weights and biases for some neurons are not updated. This can create dead neurones which never get activated.



Leaky ReLU Function

- Instead of defining the ReLU function as 0 for negative values of x, it is defined as an extremely small linear component of x
- The gradient of the left side of the graph comes out to be a non zero value.



Soft-max Function

- Sigmoid returns values between 0 and 1, which can be treated as probabilities of a data point belonging to a particular class. Thus sigmoid is widely used for binary classification problems.
- The Soft-max function can be used for multi-class classification problems. This function returns the probability for a datapoint belonging to each individual class.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Summary

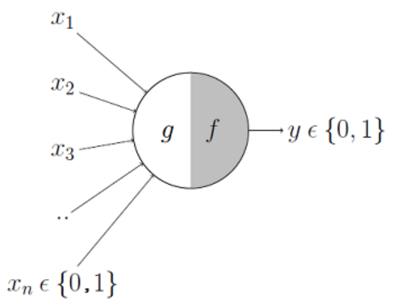
- Sigmoid functions and their combinations generally work better in the case of classifiers

- Sigmoid functions and their combinations generally work better in the case of classifiers
- Sigmoids and Tan h functions are sometimes avoided due to the vanishing gradient problem
- ReLU function is a general activation function and is used in most cases these days
- If we encounter a case of dead neurones in our networks the leaky ReLU function is the best choice
- Start with using ReLU function and then move over to other activation functions

---

McCulloch-Pitts Model

- It is very well known that the most fundamental unit of deep neural networks is called an artificial neurone/perceptron.
- The first computational model of a neurone was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943.

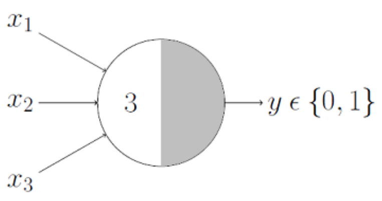


$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

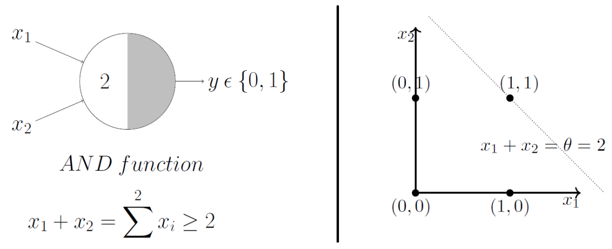
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

- The first part g takes an input and performs an aggregation and based on the aggregated value the second part f makes a decision.
- Let us suppose that I want to predict my own decision, whether to watch a random football game or not on TV. The inputs are all boolean i.e., {0,1} and my output variable is also boolean {0: Will watch it, 1: Won't watch it}.
- M-P neurone can be used to represent boolean function.

AND Function

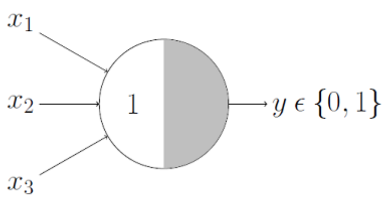


An AND function neurone would only fire when ALL the inputs are ON i.e., **g(x)** ≥ 3 i.e., ***x_1 + x_2 + x_3 = 3*** here.

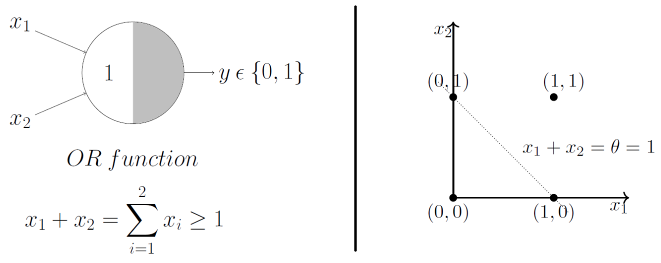


for two input |^|

OR Function



I believe this is self explanatory as we know that an OR function neuron would fire if ANY of the inputs is ON i.e., ***g*(x)** ≥ 1 i.e., ***x_1 + x_2 + x_3* ≥ 1** here.



for two input |^|

Perceptron

- Frank Rosen Blatt, an American psychologist, proposed the classical perceptron model (1958)
- A more general computational model than McCulloch{Pitts neurones)
- Perceptron is a single layer neural network
- Introduced numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values

Perceptron Training

- Training set S of examples {x, t}
  - x is an input vector and
  - t the desired target vector (Teacher)
  - Example: Logical AND
- Iterative process
  - Present a training example x , compute network output y , compare output y with target t, adjust weights and thresholds
- Learning rule
  - Specifies how to change the weights w of the network as a function of the inputs x, output y and target t.

Perceptron Learning Rule

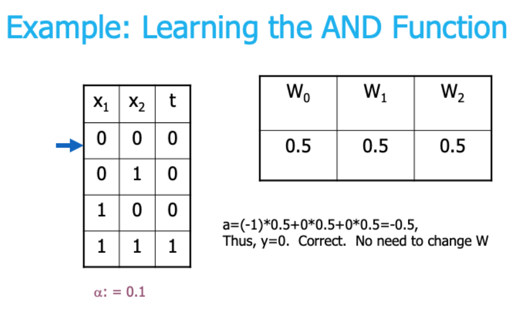
Perceptron Learning Rule

- wi :=wi +Dwi=wi +a(t-y)xi (i=1..n)
- The parameter a is called the learning rate.
- It determines the magnitude of weight updates Dwi .
- If the output is correct (t = y) the weights are not changed (Dwi =0).
- If the output is incorrect (t /= y) the weights wi are changed such that the output of the Perceptron for the new weights w'i is closer by considering input xi .

Perceptron Training Algorithm

```
Repeat
    for each training vector pair (x, t)
        evaluate the output y when x is the input
        if y/=t then
            form a new weight vector w' according
            to w'=w + α (t-y) x
        else
            do nothing
        end if
    end for
Until fixed number of iterations; or error less than a predefined value
```

Implement Basic AND Logic Gates with Perceptron



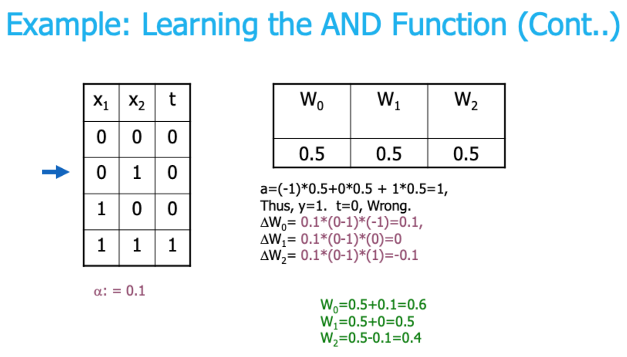
Example: Learning the AND Function

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

W_0	W_1	W_2
0.5	0.5	0.5

a=(-1)*0.5+0*0.5+0*0.5=-0.5,
Thus, y=0. Correct. No need to change W

α: = 0.1



Example: Learning the AND Function (Cont..)

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

W_0	W_1	W_2
0.5	0.5	0.5

a=(-1)*0.5+0*0.5 + 1*0.5=1,
Thus, y=1. t=0, Wrong.
ΔW_0= 0.1*(0-1)*(-1)=0.1,
ΔW_1= 0.1*(0-1)*(0)=0
ΔW_2= 0.1*(0-1)*(1)=-0.1

α: = 0.1

W_0=0.5+0.1=0.6
W_1=0.5+0=0.5
W_2=0.5-0.1=0.4

Example: Learning the AND Function (Cont..)

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

α: = 0.1

W_0	W_1	W_2
0.6	0.5	0.4

a=(-1)*0.6+1*0.5 + 0*0.4=-0.1,
Thus, y=0. t=0, Correct!

Example: Learning the AND Function (Cont..)

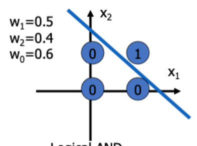
x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

α: = 0.1

W_0	W_1	W_2
0.6	0.5	0.4

a=(-1)*0.6+1*0.5 + 1*0.4=0.3,
Thus, y=1. t=1, Correct

Final Solution



$$a= 0.5x_1+0.4*x_2-0.6$$

Logical AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Note - α is the learning rate

The algorithm converges to the correct classification

- if the training data is linearly separable
- and learning rate is sufficiently small

The final weights in the solution w is not unique: there are many possible lines to separate the two classes.
XOR cannot be separated!

Multilayer Perceptron

- A network of interconnected Perceptrons in several layers
- First layer receives input, forwards to second layer etc.
- Succeeding hidden layers learn gradually more high-level features that are composed of previous hidden layer's features
- Hidden node computes a nonlinear transformation of its incoming inputs

Deep learning algorithms have high computational requirements. Code will run very slowly if we use for loops. To compute $z^{[1]} = W^{[1]} * x + b^{[1]}$ without for loop vectorise library such as numpy (python) is required.

Cont..

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} — W_1^{[1]T} — \\ — W_2^{[1]T} — \\ \vdots \\ — W_4^{[1]T} — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

$$\mathbb{R}^{n\times m} \qquad z^{[1]} = W^{[1]}x + b^{[1]}$$

- If we use linear activation function, this drops much of the representational power of the neural network .
- Without non-linear activation functions, the neural network will simply perform linear regression.

Binary Classification
Binary classification is the task of classifying the elements of a set into two groups on the basis of a classification rule.

Logistic Regression
- Logistic regression is the baseline supervised learning algorithm for classification, and also has a very close relationship with neural networks.
- A neural network can be viewed as a series of logistic regression classifiers stacked on top of each other.
- Logistic regression can be used to classify an observation into one of two classes or into one of many classes.
- Because the mathematics for the two-class case is simpler
- Logistic regression is a discriminative classifier

Logistic regression has two phases:
- Training: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.
- Test: Given a test example x we compute p(y/X) and return the higher probability label y = 1 or y = 0

Steps to solve classification problem
1) A feature representation of the input. Each input observation X(i), will be a vector of features [x1x2,........ , xn].
2) A classification function that computes y' the estimated class, via p(y/X)
3) An objective function for learning, usually involving minimising error on training examples
4) An algorithm for optimising the objective function

Logistic Regression Cost Function

Gradient Decent

- Gradient descent is an optimisation algorithm often used for finding the weights or coefficients of machine learning algorithms.
- It works by having the model make predictions on training data and using the error on the predictions to update the model in such a way as to reduce the error.
- The goal of the algorithm is to find model parameters (e.g. coefficients or weights) that minimise the error of the model on the training dataset. It does this by making changes to the model that move it along a gradient or slope of errors down toward a minimum error value. This gives the algorithm its name of "gradient descent."
- It is a first-order iterative optimisation algorithm for finding the minimum of a function
- In deep Learning the goal of the gradient descent is to minimise a loss function of the neural network.

Gradient descent can vary in terms of the number of training patterns used to calculate error; that is in turn used to update the model.
The number of patterns used to calculate the error includes how stable the gradient is that is used to update the model.

What is Stochastic Gradient Descent?
Stochastic gradient descent, often abbreviated SGD, is a variation of the gradient descent algorithm that calculates the error and updates the model for each example in the training dataset.

The update of the model for each training example means that stochastic gradient descent is often called an online machine learning algorithm.

Upsides

- The frequent updates immediately give an insight into the performance of the model and the rate of improvement.
- This variant of gradient descent may be the simplest to understand and implement, especially for beginners.
- The increased model update frequency can result in faster learning on some problems.
- The noisy update process can allow the model to avoid local minima (e.g. premature convergence).

Downsides

- Updating the model so frequently is more computationally expensive.
- The frequent updates can result in a noisy gradient signal, which may cause the model parameters and in turn the model error to jump around (have a higher variance over training epochs).
- The noisy learning process down the error gradient can also make it hard for the algorithm to settle on an error minimum for the model.

What is Batch Gradient Descent?
Batch gradient descent is a variation of the gradient descent algorithm that calculates the error for each example in the training dataset, but only updates the model after all training examples have been evaluated.

One cycle through the entire training dataset is called a training epoch. Therefore, it is often said that batch gradient descent performs model updates at the end of each training epoch.

Upsides
- Fewer updates to the model means this variant of gradient descent is more computationally efficient than stochastic gradient descent.
- The decreased update frequency results in a more stable error gradient and may result in a more stable convergence on some problems.
- The separation of the calculation of prediction errors and the model update lends the algorithm to parallel processing based implementations.

Downsides
- The more stable error gradient may result in premature convergence of the model to a less optimal set of parameters.
- The updates at the end of the training epoch require the additional complexity of accumulating prediction errors across all training examples.
- Commonly, batch gradient descent is implemented in such a way that it requires the entire training dataset in memory and available to the algorithm.
- Model updates, and in turn training speed, may become very slow for large datasets.

What is Mini-Batch Gradient Descent?
Mini-batch gradient descent is a variation of the gradient descent algorithm that splits the training dataset into small batches that are used to calculate model error and update model coefficients.

Implementations may choose to sum the gradient over the mini-batch which further reduces the variance of the gradient.

Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. It is the most common implementation of gradient descent used in the field of deep learning.

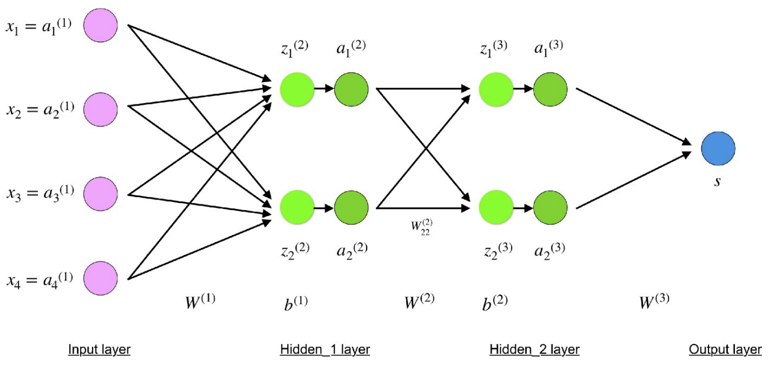
Upsides
- The model update frequency is higher than batch gradient descent which allows for a more robust convergence, avoiding local minima.
- The batched updates provide a computationally more efficient process than stochastic gradient descent.
- The batching allows both the efficiency of not having all training data in memory and algorithm implementations.

Downsides

- Mini-batch requires the configuration of an additional "mini-batch size" hyper-parameter for the learning algorithm.
- Error information must be accumulated across mini-batches of training examples like batch gradient descent.

Back Propagation

- The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, back propagation performs a backward pass while adjusting the model's parameters (weights and biases).



The final values at the hidden neurones, coloured in **green**, are computed using *z^l* — weighted inputs in layer *l*, and *a^l*— activations in layer *l*. For layer 2 and 3 the equations are:

- *l = 2*

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Equations for z^2 and a^2

- *l = 3*

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

Equations for z^3 and a^3

The equations form network's forward propagation. Here is a short overview:

$$x = a^{(1)} \qquad Input\ layer$$

$$z^{(2)} = W^{(1)}x + b^{(1)} \qquad neuron\ value\ at\ Hidden_1\ layer$$

$$a^{(2)} = f(z^{(2)}) \qquad activation\ value\ at\ Hidden_1\ layer$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)} \qquad \textit{neuron value at Hidden}_2 \textit{ layer}$$

$$a^{(3)} = f(z^{(3)}) \qquad \textit{activation value at Hidden}_2 \textit{ layer}$$

$$s = W^{(3)}a^{(3)} \qquad \textit{Output layer}$$

The final step in a forward pass is to evaluate the **predicted output s** *against an* ***expected output y** .*
Evaluation between *s* and *y* happens through a **cost function C = cost(s, y)**.

back-propagation aims to minimise the cost function by adjusting network's weights and biases. The level of adjustment is determined by the gradients of the cost function with respect to those parameters.

The derivative of the cost function measures the sensitivity to change of the output value with respect to a change in its input value

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$$

Compute those gradients happens using a technique called chain rule.
For a single weight *(w_jk)^l,* the gradient is:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \qquad \textit{chain rule}$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad \textit{by definition}$$

$$m \;-\; \textit{number of neurons in } l-1 \textit{ layer}$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad \textit{by differentiation (calculating derivative)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} \qquad \textit{final value}$$

Similar set of equations can be applied to *(b_j)^l*:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} \qquad \textit{chain rule}$$

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad \textit{by differentiation (calculating derivative)}$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} 1 \qquad \textit{final value}$$

The common part in both equations is often called *“local gradient”* and is expressed as follows:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \qquad local\ gradient$$

The gradients allow us to optimise the model’s parameters:

```
$while\ (termination\ condition\ not\ met)$
    $w := w - \epsilon \frac{\partial C}{\partial w}$
    $b := b - \epsilon \frac{\partial C}{\partial b}$
$end$
```

Learning rate determine the gradient’s influence

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \,.\, a_2^{(2)} = \frac{\partial C}{\partial a_2^{(3)}} \,.\, f'(z_2^{(3)}) \,.\, a_2^{(2)}$$

Computational Graph

• Every neural network represents a single mathematical function.

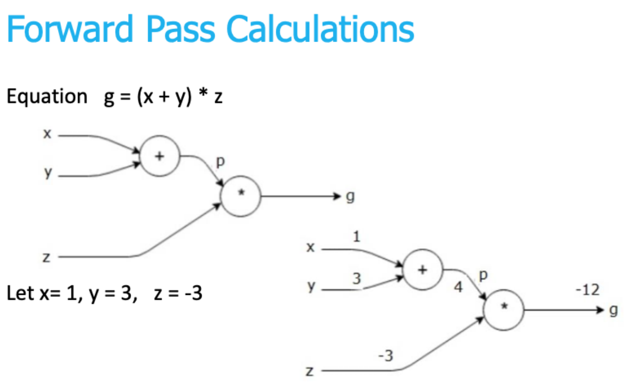
• when we set out to use a neural network for some task, our hypothesis is that there is some mathematical function that will approximate the observed behaviour reasonably well.

• In deep learning frameworks such as Tensor-flow, Theano, etc., back-propagation is implemented using computational graphs.

• Directed graph where the nodes correspond to mathematical operations.

• Computational graphs are a way of articulating and evaluating a mathematical expression.

Let us take an example, we have the following equation :- g=(x+y)∗zg=(x+y)∗z
The above equation is represented by the following computational graph.

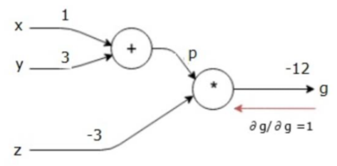


Backward Pass Calculations

Compute the gradients for each input with respect to the final output.
These gradients are needed for training the neural network

In the backward pass first derivative of the output variable computed with respect to itself

$$\frac{\partial g}{\partial g} = 1$$



Cont..

- Since *g = p*z*

$$\frac{\partial g}{\partial z} = p \qquad \frac{\partial g}{\partial z} = p = 4$$

$$\frac{\partial g}{\partial p} = z \qquad \frac{\partial g}{\partial p} = z = -3$$

- The gradient at *x* & *y* $\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}$
- By chain rule of differentiation $\frac{\partial g}{\partial x} = \frac{\partial g}{\partial p} * \frac{\partial p}{\partial x}$ $\frac{\partial g}{\partial y} = \frac{\partial g}{\partial p} * \frac{\partial p}{\partial y}$

Cont..

- since p directly depends on *x* and *y*

$$\frac{\partial g}{\partial y} = \frac{\partial g}{\partial p} * \frac{\partial p}{\partial y} = (-3)\,.1 = -3$$

- To calculate the gradient at x, we only used already computed values, and *dg/dx* (derivative of node output with respect to the same node's input).
- Used local information to compute a global value.

Model Parameters

Parameters (such as weights & bias) in the model must be determined using the training data set.
These are the fitted parameters.
m(slope) and c(intercept) in Linear Regression weights and biases in Neural Networks
• Required by the model when making predictions. • Estimated or learned from data. • Not set manually by the practitioner. • Often saved as part of the learned model.

Hyperparameters

• Adjustable parameters that must be tuned in order to obtain a model with optimal performance.
Learning rate in gradient descent Number of iterations in gradient descent Number of layers in a Neural

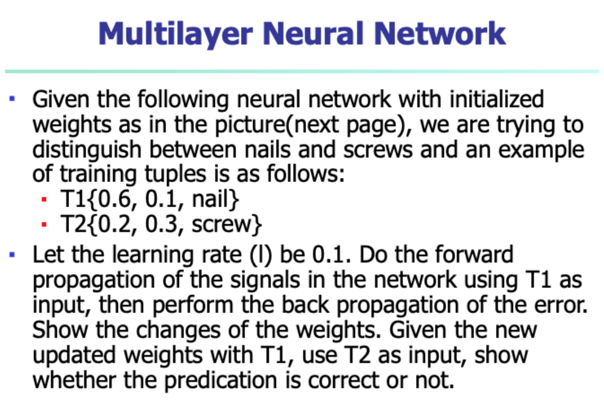
Network Number of neurones per layer in a Neural Network Number of clusters(k) in k means clustering

Parameters vs Hyperparameters

Parameters	Hyperparameters
Estimated during the training with historical data	Values are set beforehand
It is a part of the model	External to the model
The estimated value is saved with the trained model	Not part of the model & hence the values are not saved
The estimated value is saved with the trained model	Not part of the model & hence the values are not saved

---

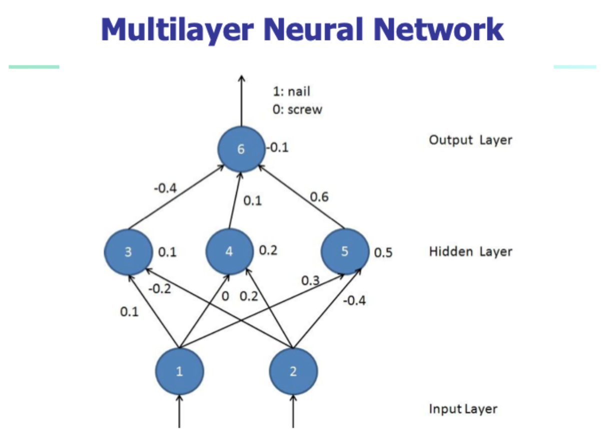
Back Propagation Example of how to solve ?

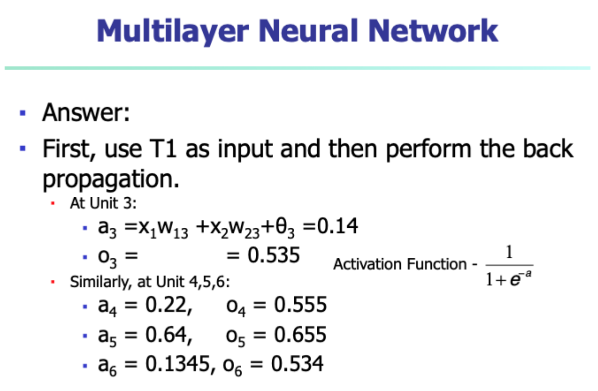


Multilayer Neural Network

- Given the following neural network with initialized weights as in the picture(next page), we are trying to distinguish between nails and screws and an example of training tuples is as follows:
  - T1{0.6, 0.1, nail}
  - T2{0.2, 0.3, screw}
- Let the learning rate (l) be 0.1. Do the forward propagation of the signals in the network using T1 as input, then perform the back propagation of the error. Show the changes of the weights. Given the new updated weights with T1, use T2 as input, show whether the predication is correct or not.

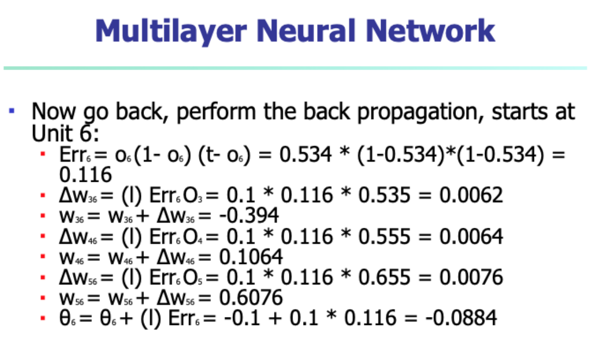
Multilayer Neural Network





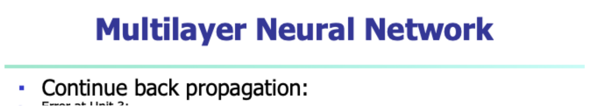
Multilayer Neural Network

- Answer:
- First, use T1 as input and then perform the back propagation.
  - At Unit 3:
    - $a_3 = x_1 w_{13} + x_2 w_{23} + \theta_3 = 0.14$
    - $o_3 = \qquad = 0.535$ Activation Function - $\frac{1}{1+e^{-a}}$
  - Similarly, at Unit 4,5,6:
    - $a_4 = 0.22, \quad o_4 = 0.555$
    - $a_5 = 0.64, \quad o_5 = 0.655$
    - $a_6 = 0.1345, o_6 = 0.534$



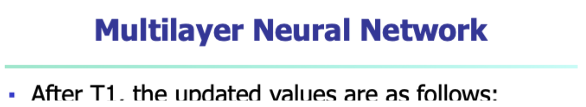
Multilayer Neural Network

- Now go back, perform the back propagation, starts at Unit 6:
  - $Err_6 = o_6(1- o_6)\ (t- o_6) = 0.534 * (1-0.534)*(1-0.534) = 0.116$
  - $\Delta w_{36} = (l)\ Err_6 O_3 = 0.1 * 0.116 * 0.535 = 0.0062$
  - $w_{36} = w_{36} + \Delta w_{36} = -0.394$
  - $\Delta w_{46} = (l)\ Err_6 O_4 = 0.1 * 0.116 * 0.555 = 0.0064$
  - $w_{46} = w_{46} + \Delta w_{46} = 0.1064$
  - $\Delta w_{56} = (l)\ Err_6 O_5 = 0.1 * 0.116 * 0.655 = 0.0076$
  - $w_{56} = w_{56} + \Delta w_{56} = 0.6076$
  - $\theta_6 = \theta_6 + (l)\ Err_6 = -0.1 + 0.1 * 0.116 = -0.0884$



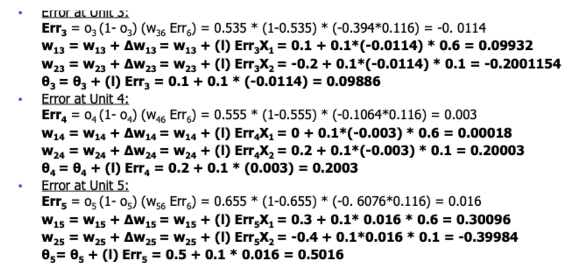
Multilayer Neural Network

- Continue back propagation:

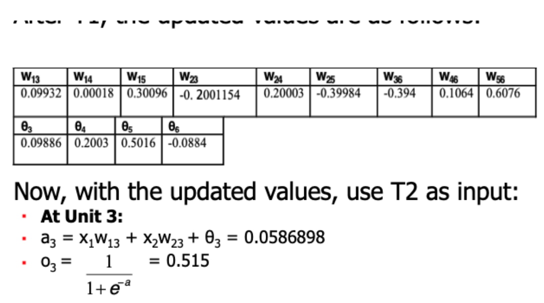


Multilayer Neural Network

- After T1, the updated values are as follows:



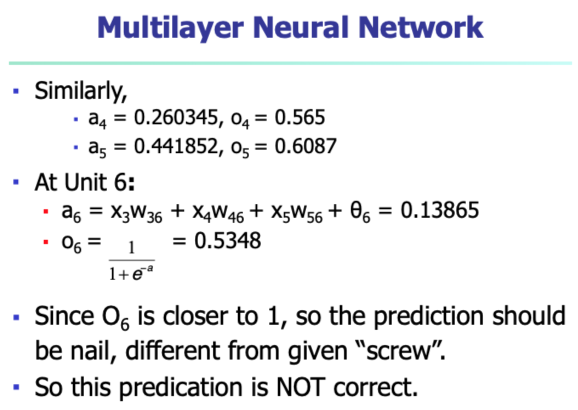
- Error at Unit 3:
  $\mathbf{Err_3} = o_3(1- o_3)(w_{36}\,Err_6) = 0.535 * (1-0.535) * (-0.394*0.116) = -0.\,0114$
  $\mathbf{w_{13} = w_{13} + \Delta w_{13} = w_{13} + (l)\,Err_3X_1 = 0.1 + 0.1*(-0.0114) * 0.6 = 0.09932}$
  $\mathbf{w_{23} = w_{23} + \Delta w_{23} = w_{23} + (l)\,Err_3X_2 = -0.2 + 0.1*(-0.0114) * 0.1 = -0.2001154}$
  $\mathbf{\theta_3 = \theta_3 + (l)\,Err_3 = 0.1 + 0.1 * (-0.0114) = 0.09886}$
- Error at Unit 4:
  $\mathbf{Err_4} = o_4(1- o_4)(w_{46}\,Err_6) = 0.555 * (1-0.555) * (-0.1064*0.116) = 0.003$
  $\mathbf{w_{14} = w_{14} + \Delta w_{14} = w_{14} + (l)\,Err_4X_1 = 0 + 0.1*(-0.003) * 0.6 = 0.00018}$
  $\mathbf{w_{24} = w_{24} + \Delta w_{24} = w_{24} + (l)\,Err_4X_2 = 0.2 + 0.1*(-0.003) * 0.1 = 0.20003}$
  $\mathbf{\theta_4 = \theta_4 + (l)\,Err_4 = 0.2 + 0.1 * (0.003) = 0.2003}$
- Error at Unit 5:
  $\mathbf{Err_5} = o_5(1- o_5)(w_{56}\,Err_6) = 0.655 * (1-0.655) * (-0.\,6076*0.116) = 0.016$
  $\mathbf{w_{15} = w_{15} + \Delta w_{15} = w_{15} + (l)\,Err_5X_1 = 0.3 + 0.1* 0.016 * 0.6 = 0.30096}$
  $\mathbf{w_{25} = w_{25} + \Delta w_{25} = w_{25} + (l)\,Err_5X_2 = -0.4 + 0.1*0.016 * 0.1 = -0.39984}$
  $\mathbf{\theta_5 = \theta_5 + (l)\,Err_5 = 0.5 + 0.1 * 0.016 = 0.5016}$



w_{13}	w_{14}	w_{15}	w_{23}	w_{24}	w_{25}	w_{36}	w_{46}	w_{56}
0.09932	0.00018	0.30096	-0. 2001154	0.20003	-0.39984	-0.394	0.1064	0.6076

θ_3	θ_4	θ_5	θ_6
0.09886	0.2003	0.5016	-0.0884

- Now, with the updated values, use T2 as input:
  - **At Unit 3:**
  - $a_3 = x_1w_{13} + x_2w_{23} + \theta_3 = 0.0586898$
  - $o_3 = \dfrac{1}{1+e^{-a}} = 0.515$



Multilayer Neural Network

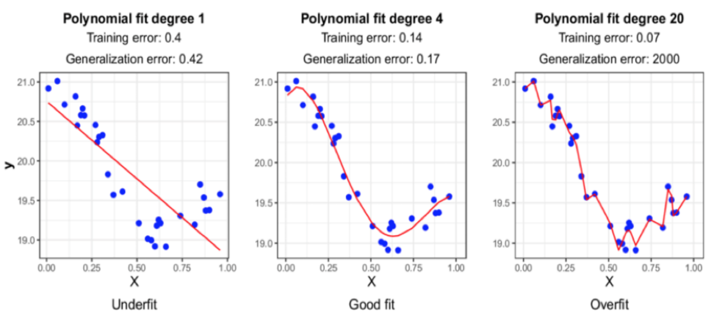
- Similarly,
  - $a_4 = 0.260345$, $o_4 = 0.565$
  - $a_5 = 0.441852$, $o_5 = 0.6087$
- At Unit 6**:**
  - $a_6 = x_3w_{36} + x_4w_{46} + x_5w_{56} + \theta_6 = 0.13865$
  - $o_6 = \dfrac{1}{1+e^{-a}} = 0.5348$
- Since O_6 is closer to 1, so the prediction should be nail, different from given "screw".
- So this predication is NOT correct.

Over-fitting and under-fitting

Once the DLN model has been trained, its true test is how well it is able to classify inputs that it has not seen before, which is also known as its Generalisation Ability. There are two kinds of problems that can afflict ML models in general:

1. Even after the model has been fully trained such that its training error is small, it exhibits a high test error rate. This is known as the problem of Overfitting.
2. The training error fails to come down in-spite of several epochs of training. This is known as the problem of Under-fitting.

Underfitting & Overfitting (Cont..)



https://images.app.goo.gl/HkTiWXBwFLBPesfCA

Underfitting & Overfitting (Cont..)

• Under-fitting happens when the learner cannot find a solution that fits the observed data well enough.

• For example, if the learner tries to fit a straight line to a training set whose examples exhibit a quadratic

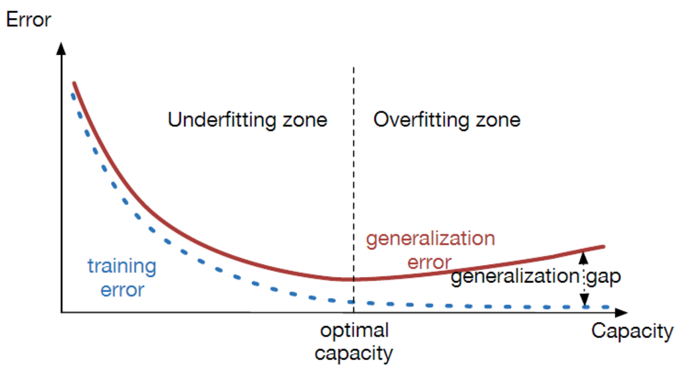
nature, then the learner under-fits the data.
• A learner that under-fits the training data will miss important aspects of the data.
• In overfitting, a statistical model describes random error or noise instead of the underlying relationship.
• Overfitting occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.
• A model that has been overfitted has poor predictive performance, as it overreacts to minor fluctuations in the training data

Generalisation
• Generalisation is a task a learning machine performs when it makes predictions about new inputs that it has not seen before.
• Generalisation performance is a measure of how well a learning machine performs on new inputs.

The following factors determine how well a model is able to generalise from the training dataset to the test dataset:

- The model capacity and its relation to data complexity: In general if the model capacity is less than the data complexity then it leads to under-fitting, while if the converse is true, then it can lead to overfitting. Hence we should try to choose a model whose capacity matches the complexity of the training and test datasets.
- Even if the model capacity and the data complexity are well matched, we can still encounter the overfitting problem. This is caused to due to an insufficient amount of training data.

The chances of encountering the overfitting problem increases as the model capacity grows, but decreases as more training data is available. Note that we are attempting to reduce the training error (to avoid the underfitting problem) and the test error (to avoid the overfitting problem) *at the same time*. This leads to conflicting demands on the model capacity, since training error reduces monotonically as the model capacity increases, but the test error starts to increase if the model capacity is too high. In general if we plot the test error as a function of model capacity, it exhibits a characteristic U shaped curve.



Bias

The bias of a predictor is proportional to the difference between the predictor's hypothesis and the true value of the parameter being estimated.

The higher is the difference between the predictor's hypothesis and the actual value being estimated, the higher is the bias of the predictor.

Variance

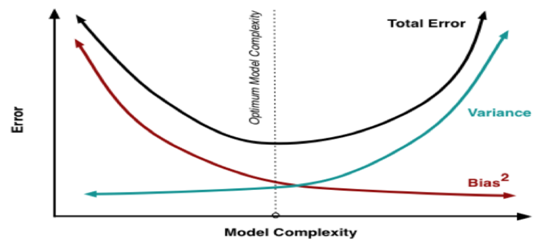
Prediction over one training set differ from the average predicted value over all the training set. Solutions for individual datasets vary around their averages

Noise

Noise cannot be reduced irrespective of what algorithm is used.

• The bias is error from erroneous assumptions in the learning algorithm.
• High bias can cause an algorithm to miss the relevant relations between features and target outputs (under-fitting).
• The variance is error from sensitivity to small fluctuations in the training set.
• High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

Bias & Variance Trade-Off



https://images.app.goo.gl/9NKQgAWDcxpyS88n9

Capacity

• Capacity is the percentage of training examples that the learner can fit almost perfectly.
• If a learner has high capacity, then it can fit a high percentage of the training examples very well.
• Capacity depends on the resources a machine learning algorithm has to model data
• If by increasing capacity we decrease generalisation error, then we are under-fitting, otherwise we are overfitting.
• If the error in representing the training set is relatively large and the generalisation error is large, then under-fitting; Need to increase capacity.
• If the error in representing the training set is relatively small and the generalisation error is large, then overfitting; • Need to decrease capacity.
• Many features but relatively small training set. • Need to decrease capacity and/or increase training set.

Regularisation

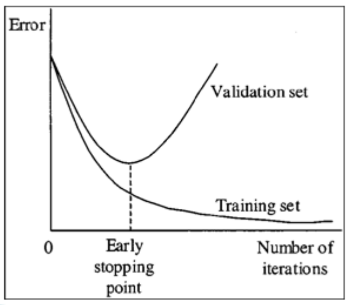
• When an algorithm performs well on the training set but performs poorly on the test set, the algorithm is said to be overfitted on the Training data.
• Techniques which are used to reduce the error on the test set at an expense of increased training error are known as Regularisation.
• Regularisation can be defined as any modification in a learning algorithm that is intended to reduce its

• Regularisation can be defined as any modification in a learning algorithm that is intended to reduce its generalisation error but not its training error.

Regularisation Techniques
• Early Stopping
• L1 Regularisation
• L2 Regularisation
• Dropout
• Training Data Augmentation • Batch Normalisation

Early Stopping

Use the validation data set to compute the loss function at the end of each training epoch, and once the loss stops decreasing, stop the training and use the test data to compute the final classification accuracy.
It is a fairly simple form of Regularisation, since it does not require any changes to the model or objective function which can change the learning dynamics of the system.



Data Augmentation

If the data set used for training is not large enough, A simple technique to get around this problem is by artificially expanding the training set. In the case of image data, this fairly straightforward by using following techniques:

- Translation: Moving the image by a few pixels in various directions
- Rotation
- Reflection
- Skewing
- Scaling: Changing the size of the image while preserving its shape
- Changing contrast or brightness

L2 Regularisation

L2 Regularisation is a commonly used technique in ML systems is also sometimes referred to as "Weight Decay". It works by adding a quadratic term to the Cross Entropy Loss Function l, called the Regularisation Term, which results in a new Loss Function lR given by:

$$\mathcal{L}_R = \mathcal{L} + \frac{\lambda}{2} \sum_{r=1}^{R+1} \sum_{j=1}^{P^{r-1}} \sum_{i=1}^{P^r} (w_{ij}^{(r)})^2$$

The Regularisation Term consists of the sum of the squares of all the link weights in the DLN, multiplied by a parameter λ called the Regularisation Parameter. This is another Hyper-parameter whose appropriate value needs to be chosen as part of the training process by using the validation data set. By choosing a value for this parameter, we decide on the relative importance of the Regularisation Term vs the Loss Function term. Note that the Regularisation Term does not include the biases, since in practice it has been found that their inclusion does not make much of a difference to the final result.

In order to gain an intuitive understanding of how L2 Regularisation works against overfitting, note that the net effect of adding the Regularisation Term is to bias the algorithm towards choosing smaller values for the link weight parameters. The value of the λ governs the relative importance of the Cross Entropy term vs the regularisation term and as λ increases, the system tends to favour smaller and smaller weight values.

L1 Regularisation

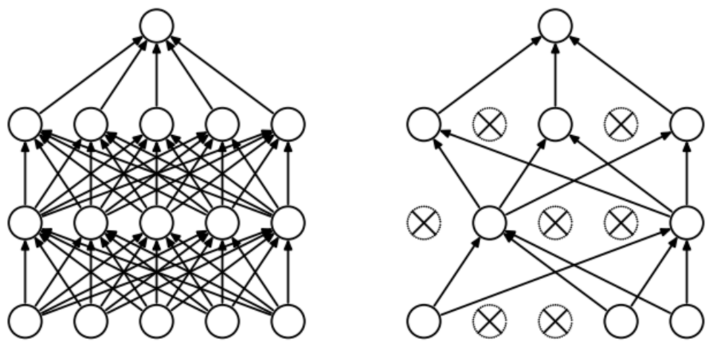
L1 Regularisation uses a Regularisation Function which is the sum of the absolute value of all the weights in DLN, resulting in the following loss function (l is the usual Cross Entropy loss):

$$\mathcal{L}_R = \mathcal{L} + \lambda \sum_{r=1}^{R+1} \sum_{j=1}^{P^{r-1}} \sum_{i=1}^{P^r} |w_{ij}^{(r)}|$$

At a high level L1 Regularisation is similar to L2 Regularisation since it leads to smaller weights. It results in the following weight update equation

Dropout Regularisation

Dropout is one of the most effective Regularisation techniques to have emerged in the last few years


(a) Standard Neural Net (b) After applying dropout.

- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically p = 0.5) for hidden nodes and p = 0.8 for visible nodes

Suppose a neural network has n nodes Using the dropout idea, other than output node each node can be

retained or dropped
Total number of thinned networks that can be formed? 2^n
Important Aspect
(1) Share the weights across all the networks
(2) Sample a different network for each training instance

Initialise all the parameters (weights) of the network and start training For the first mini.batch,
Apply dropout resulting in the thinned network compute the loss and back-propagate
Only active parameters will we update
• Apply dropout then train model by second batch
• Compute the loss and back-propagate to the active weights
• If the weight are updated by both the subnetworks then it received two updates
• If the weight is used only one of the training instances then it received only one updates
• Updated weights are used in training of new subnetwork (Parameter Sharing)
• At test time use the full Neural Network and scale the output of each node by the fraction of times it was on during training

---

Why CNN is preferred over MLP (ANN) for image classification

- MLPs use one perceptron for each input (and the amount of weights rapidly becomes unmanageable for large images.
- It includes too many parameters because it is fully connected.
- Difficulties arise whilst training and overfitting can occur which makes it loses the ability to generalise.
- Another common problem is that MLPs react differently to an input (images) and its shifted version — they are not translation invariant.
- MLPs are not the best idea to use for image processing
- Spatial information is lost when the image is flattened(matrix to vector) into an MLP.

Thus we need a way to leverage the spatial correlation of the image features (pixels) in such a way that we can see the object in our picture no matter where it may appear.
The benefit of using CNNs is their ability to develop an internal representation of a two-dimensional image.

Features of CNN

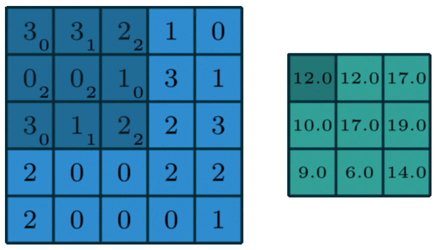
- In CNN local connectivity each filter is panned around the entire image [Convolution] according to certain size and stride(no. of pixels/blocks to shift around after every convolution step), allows the filter to find and match patterns no matter where the pattern is located in a given image).
- The weights are smaller and shared — less wasteful, easier to train than MLP and more effective too.
- Also go deeper. Layers are sparsely connected rather than fully connected
- It takes matrices as well as vectors as inputs
- CNN’s leverage the fact that nearby pixels are more strongly related than distant ones.
- Analyse the influence of nearby pixels by using a filter /Kernel and we move this across the image from top left to bottom right

top left to bottom right.

- This reduces the number of weights that the neural network must learn compared to an MLP
- We can set the stride and filter size
- CNN allows parameter sharing, weight sharing so that the filter looks for a specific pattern, and is location invariant — can find the pattern anywhere in an image
- CNNs are a special type of neural network whose hidden units are only connected to local receptive field.
- Less number of parameters needed by CNNs.

2D Convolutions: The Operation

The 2D convolution is a fairly simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an element-wise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.



In the above example, we have 5×5=25 input features, and 3×3=9 output features. If this were a standard fully connected layer, you’d have a weight matrix of 25×9 = 225 parameters, with every output feature being the weighted sum of *every single* input feature. Convolutions allow us to do this transformation with only 9 parameters, with each output feature, instead of “looking at” every input feature, only getting to “look” at input features coming from roughly the same location.

This operation of sliding the filter across the image, while computing the dot product at each position, is called a convolution.

Stride is defined as the number of pixels by which the filter is moved after each computation, either horizontally or vertically.

Pooling

- Pooling usually occurs after a Convolutional Layer
- Can be described as condensing the information from the layer into a smaller number of activations.
- Pooling does not introduce any new parameters to the CNN and the total number of parameters in the model are reduced considerably due to this operation.