**DEEP LEARNING :**

1. What kind of problems can neural networks solve? 👶

Artificial neural network or **ANN**are best suited for tasks that include pattern recognition.

These are:

1. Image recognition
2. Speech recognition
3. Natural language processing

It can also be great for computer vision. Automobile industry is probably best industry.

Another case is general signal processing. For instance processing any sort of signal into output. Distance sensors on self driven drones might be a good example of that.

1. How does a usual fully connected feedforward neural network work? ⭐️

Fully connected layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional [multi-layer perceptron](https://en.wikipedia.org/wiki/Multi-layer_perceptron) neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

1. Why do we need activation functions? 👶

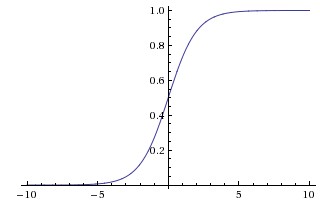
The purpose of an activation function is to add some kind of non-linear property to the function, which is a neural network. Without the activation functions, the neural network could perform only linear mappings from inputs x to the outputs y. Why is this so? Without the activation functions, the only mathematical operation during the forward propagation would be dot-products between an input vector and a weight matrix. Since a single dot product is a linear operation, successive dot products would be nothing more than multiple linear operations repeated one after the other. *And successive linear operations can be considered as a one single learn operation.*

In order to be able to compute really interesting stuff, neural networks must be able to approximate nonlinear relations from input features to output labels. Usually, the more complex the data is we are trying to learn something from, the more non-linear the mapping of features to the ground truth label is.

A neural network without any activation function would not be able to realize such complex mappings mathematically and would not be able to solve tasks we want the network to solve.

1. What are the problems with sigmoid as an activation function? ⭐️

It returns a value between 0 and 1. The two major problems with sigmoid activation functions are:



* **Sigmoid saturate and kill gradients:** The output of sigmoid saturates (i.e. the curve becomes parallel to x-axis) for a large positive or large negative number. Thus, the gradient at these regions is almost zero. During backpropagation, this local gradient is multiplied with the gradient of this gates’ output. Thus, if the local gradient is very small, it’ll kill the the gradient and the network will not learn. This problem of *vanishing gradient* is solved by ReLU.
* **Not zero-centered:** Sigmoid outputs are not zero-centered, which is undesirable because it can indirectly introduce undesirable zig-zagging dynamics in the gradient updates for the weights.

1. What is ReLU? How is it better than sigmoid or tanh? ⭐️

ReLu is a non-linear [activation function](https://deepai.org/machine-learning-glossary-and-terms/activation-function) that is used in multi-layer [neural networks](https://deepai.org/machine-learning-glossary-and-terms/neural-network) or deep neural networks. This function can be represented as:

https://images.deepai.org/glossary-terms/relu-7063560.jpg

where x = an input value

According to equation 1, the output of ReLu is the maximum value between zero and the input value. An output is equal to zero when the input value is negative and the input value when the input is positive. Thus, we can rewrite equation 1 as follows:

https://images.deepai.org/glossary-terms/relu-291307.jpg

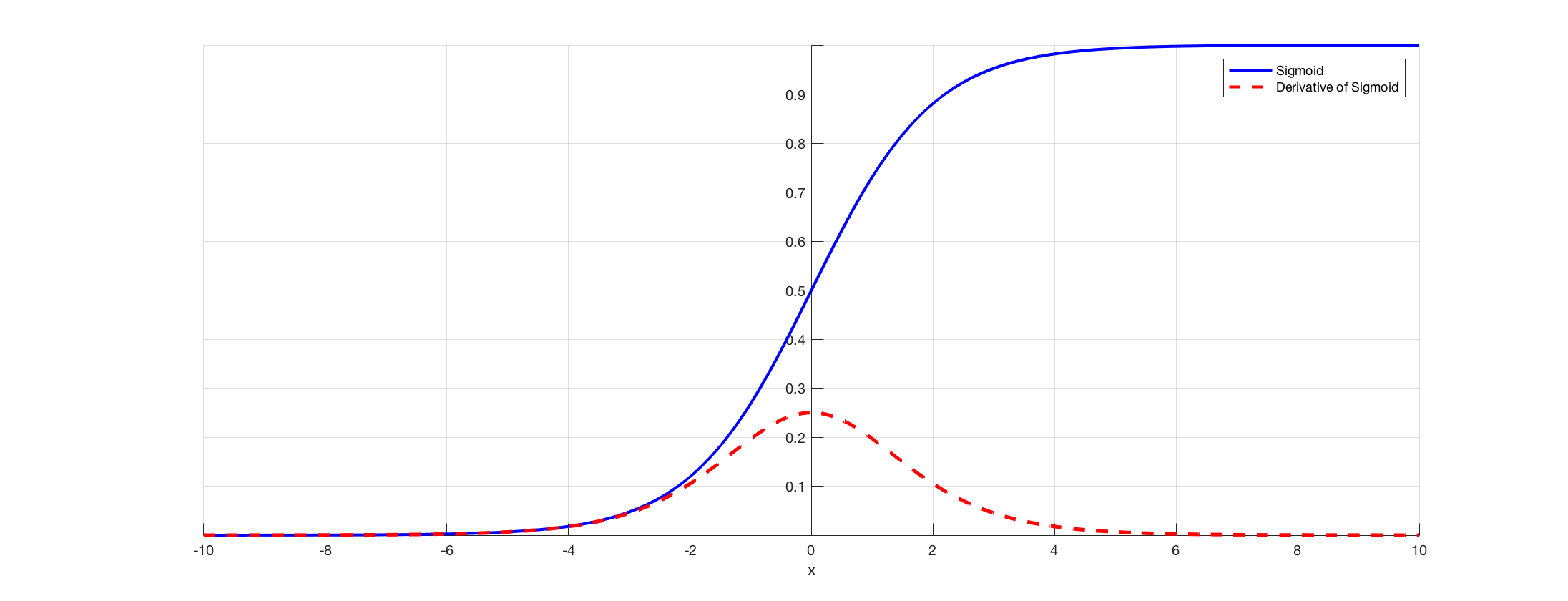
where x = an input value

1. How to initialize the weights of a neural network? ⭐️

**Zero initialization :**

In general practice biases are initialized with 0 and weights are initialized with random numbers, what if weights are initialized with 0?

In order to understand let us consider we applied sigmoid activation function for the output layer.



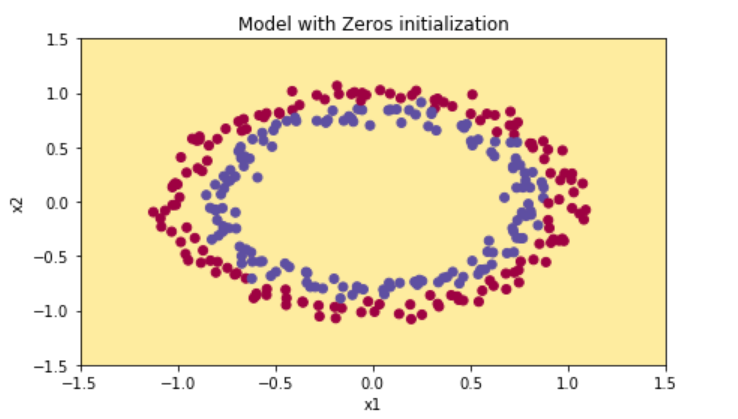
If all the weights are initialized with 0, the derivative with respect to loss function is the same for every w in W[l], thus all weights have the same value in subsequent iterations. This makes hidden units symmetric and continues for all the n iterations i.e. setting weights to 0 does not make it better than a linear model. An important thing to keep in mind is that biases have no effect what so ever when initialized with 0.

W[l] = np.random.zeros((l-1,l))

let us consider a neural network with only three hidden layers with ReLu activation function in hidden layers and sigmoid for the output layer.

Using the above neural network on the dataset “make circles” from sklearn.datasets, the result obtained as the following :

for 15000 iterations, loss = 0.6931471805599453, accuracy = 50 %



clearly, zero initialization isn’t successful in classification.

**Random initialization :**

Assigning random values to weights is better than just 0 assignment. But there is one thing to keep in my mind is that what happens if weights are initialized high values or very low values and what is a reasonable initialization of weight values.

**a)**If weights are initialized with very high values the term np.dot(W,X)+b becomes significantly higher and if an activation function like sigmoid() is applied, the function maps its value near to 1 where the slope of gradient changes slowly and learning takes a lot of time.

**b)** If weights are initialized with low values it gets mapped to 0, where the case is the same as above.

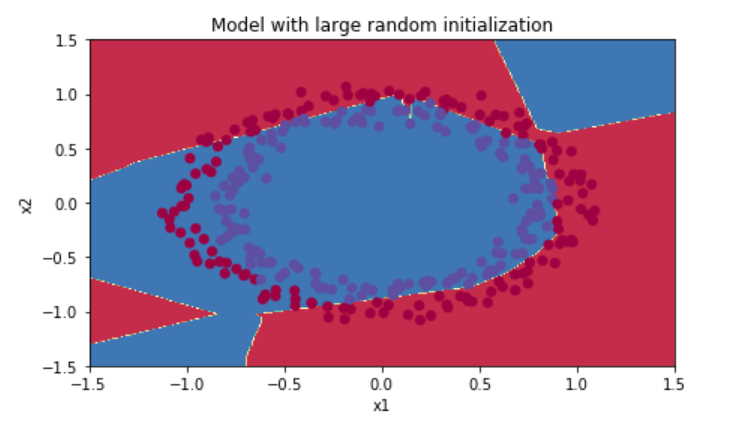
This problem is often referred to as the vanishing gradient.

To see this let us see the example we took above but now the weights are initialized with very large values instead of 0 :

W[l] = np.random.randn(l-1,l)\*10

Neural network is the same as earlier, using this initialization on the dataset “make circles” from sklearn.datasets, the result obtained as the following :

for 15000 iterations, loss = 0.38278397192120406, accuracy = 86 %



1. What if we set all the weights of a neural network to 0? ⭐️

If all the weights are initialized with 0, the derivative with respect to loss function is the same for every w in W[l], thus all weights have the same value in subsequent iterations. This makes hidden units symmetric and continues for all the n iterations i.e. setting weights to 0 does not make it better than a linear model.

1. What techniques for regulating neural networks do you know? ⭐️

Dropout

L1,L2 and elastic norm regularization

Early stopping

1. What is dropping out? Why is it useful? How it works? ⭐️

 Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By “ignoring”, I mean these units are not considered during a particular forward or backward pass.

**To prevent over-fitting:**A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

Regularization is way to prevent over-fitting. Regularization reduces over-fitting by adding a penalty to the loss function. By adding this penalty, the model is trained such that it does not learn interdependent set of features weights. Those of you who know Logistic Regression might be familiar with L1 (Laplacian) and L2 (Gaussian) penalties.

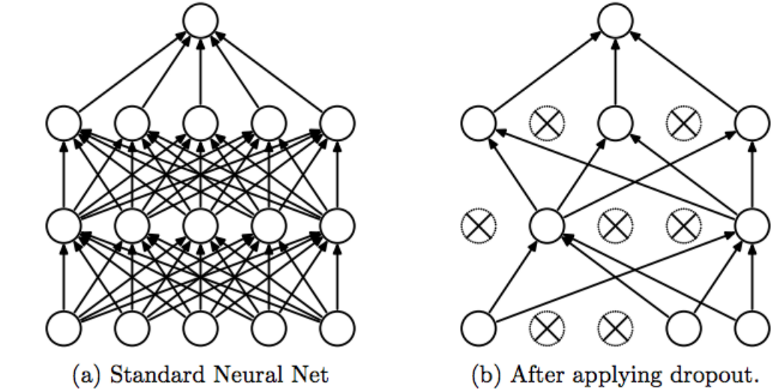
Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

**Training Phase:**

Training Phase: For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, *p*, of nodes (and corresponding activations).

**Testing Phase:**

Use all activations, but reduce them by a factor *p* (to account for the missing activations during training).



1. Optimization in neural networks

Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.

1. What is backpropagation? How it works? Why do we need it? ⭐️

**Back-propagation** is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

The **backpropagation algorithm works** by computing the gradient of the loss function with respect to each weight by the [chain rule](https://en.wikipedia.org/wiki/Chain_rule), computing the gradient one layer at a time, [iterating](https://en.wikipedia.org/wiki/Iteration) backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule.

**Backpropagation aims to minimize 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.

**why computing gradients**?-> The gradient shows how much the parameter x needs to change (in positive or negative direction) to minimize C.

12.What optimization techniques for training neural networks do you know? ⭐️

Gradient Descent

Stochastic Gradient Descent

Mini-batch Gradient Descent

Momentum

Nesterov Accelerated Gradient

Adagrad

AdaDelta

Adam

13.How do we use SGD (stochastic gradient descent) to form a neural network? ⭐️

Momentum

If the objective has the form of a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains. Momentum is one method for pushing the objective more quickly along the shallow ravine. The momentum update is given by,

vθ=γv+α∇θJ(θ;x(i),y(i))=θ−v

In the above equation v is the current velocity vector which is of the same dimension as the parameter vector θ. The learning rate α is as described above, although when using momentum α may need to be smaller since the magnitude of the gradient will be larger. Finally γ∈(0,1] determines for how many iterations the previous gradients are incorporated into the current update. Generally γ is set to 0.5 until the initial learning stabilizes and then is increased to 0.9 or higher.

14.What is a convolutional layer? ⭐️

Convolutional layers are the major building blocks used in convolutional neural networks.

A convolution is the simple application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input, such as an image.