Deep Learning

Jawwad Shamsi

February 8, 2024

1 Fundamentals of Deep Neural Networks contd

This is a continuation of the previous notes.

1.1 Activation Functions

Activation Functions add non-linearity to the o/p Following activation functions were discussed

1. Sigmod

2. Softmax

3. tanh

4. RELU

5. Leaky Relu

They are explained in detail in the previous section. Below is a summary extracted from the book

Activation functions

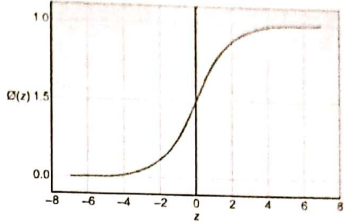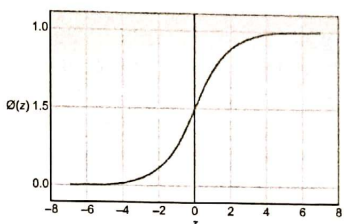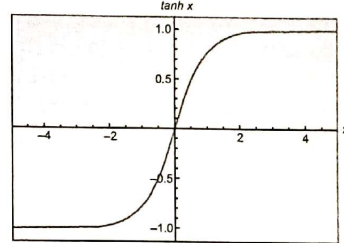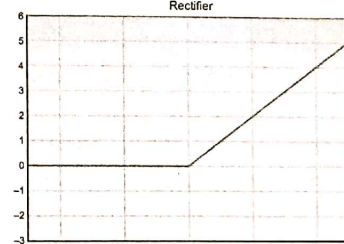**Table 2.1   A cheat sheet of the most common activation functions**
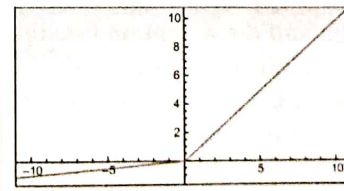
Activation function	Description	Plot	Equation
Sigmoid/ logistic function	Squishes all the values to a probability between 0 and 1, which reduces extreme values or outliers in the data. Usually used to classify two classes.		$\sigma(z) = \dfrac{1}{1+e^{-z}}$
Softmax function	A generalization of the sigmoid function. Used to obtain classification probabilities when we have more than two classes.		$\sigma(x_j) = \dfrac{e^{x_j}}{\sum_i e^{x_i}}$
Hyperbolic tangent function (tanh)	Squishes all values to the range of −1 to 1. Tanh almost always works better than the sigmoid function in hidden layers.		$\tanh(x) = \dfrac{\sinh(x)}{\cosh(x)}$ $= \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$
Rectified linear unit (ReLU)	Activates a node only if the input is above zero. Always recommended for hidden layers. Better than tanh.		$f(x) = \max(0, x)$
Leaky ReLU	Instead of having the function be zero when $x < 0$, leaky ReLU introduces a small negative slope (around 0.01) when (x) is negative.		$f(x) = \max(0.01x, x)$

Figure 1: Activation Functions

1.2   Multi-label vs multi-class

: Multi-class means a classification problem with more than two classes. These classes are distinct, i.e., only one of them could occur at once. e.g. in a corpus of images, we can distinguish b/w dog,cat, and fish. Since sum of all probabilities is 1 here, we should use softmax activation function.

In comparison, multi-label classification means labels which are not distinct and can occur together. For instance, in a chect x-ray scan system, multiple chest diseases such as T.B, cancer, and CoVID can co-exist. Here sum of all probabilities can be greater than one. We will use sigmoid activation function.

1.3   Error Function

In Machine Learning, error is a difference between the actual value and the estimated value.

$$Error = |\hat{y} - y_i|$$

Error is always +ve. This is because we are interested in the absolute value.

Remember that the estimated value depends upon the weights and bias. So in other words, Error is a function of weights and bias. It is also referred as the cost function.

1.3.1   Types of Errors

For continuous o/p variable. For Regression Problems

Mean Square Error $E(W,b) = \frac{1}{n}\Sigma_{i=1}^{n}(\hat{y} - y_i)^2$
MSE penalizes outliers
Or Absolute Mean Error $E(W,b) = \frac{1}{n}\sum_{t=1}^{n}|\hat{y} - y_i|$
For categorical values. For Classification Problems
Cross Entropy
Cross Entropy is used to quantify the problem as it quantifies the difference between two probability distributions. We can use the following equation to compute the cross entropy loss.
$E(W,b) = -\Sigma_{i=1}^{m}\hat{y}_i log(p_i)$
cross-entropy over all training examples can be computed as follows:
$E(W,b) = -\Sigma_{i=1}^{n}\Sigma_{i=1}^{m}\hat{y}_i log(p_i)$
Here is the target probability (p) is the predicted probability and (m) is the number of classes.

Example 1.1 *Understanding Cross Entropy For instance, we have a data set to distinguish between an Elephant, Lion, and Giraffe. Suppose the training probability distribution is as follows:*
Class A: Elephant 0.0 Class B: Lion 1.0 Class C: Giraffe 0.0
This implies that the particular training instance has 100% probability of Class B (Lion) and 0 % probability of other two classes.
Now suppose during testing our algorithm predicts the following distribution: Class A: Elephant 0.3 Class B: Lion 0.2 Class C: Giraffe 0.5
E= -(0.0 * log (.3) + 1.0 * log(0.2)+ 0.0 *log (0.5))= -1.609
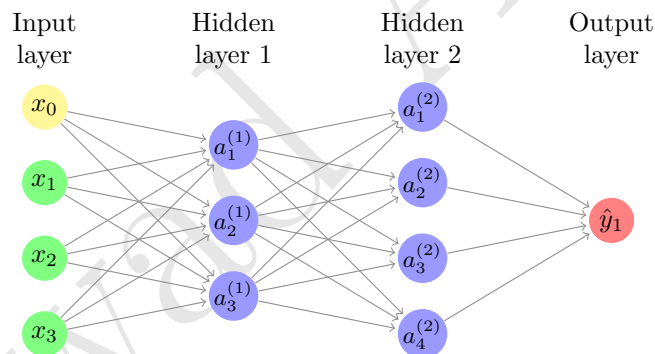*Let's compute error for better predictions*
Class A: Elephant 0.2 Class B: Lion 0.6 Class C: Giraffe 0.2

1.4   Adjusting Error

In order to reduce error, we have to adjust weights. There are two possibilities, a brute force approach or some algorithm to determine a possible set of weights.

1.4.1   Brute Force

Let's analyze brute force first.

In the above diagram, there are 28 possible weights and eight possible values of bias. This is a simple DNN. If we increase hidden layers or no. of neurons in the hidden layers then it will eventually increase the no. of neurons (and no of weights and bias). If we only consider 28 weights, and assume that each weight could have 1000 different values then we will have 10^{84} possible values of weights (1000X1000 X1000 ....1000, 25 times). This will increase with the increase in weights.

Certainly, the brute force approach is not scalable. This is because as the no. of neurons and no. of layers increase, no. of weights also increase as well.

So we need a mechanism to adjust weights

1.4.2   Local Minima Vs. Global Minima

Before diving deeper into the algorithm, let's recall the difference between the local minima and the global minima. The former is the value of the function (if plotted on a graph) which is less than all other values near it (local context). Whereas, the latter is a value of the function, which is less then all the other values of the function (global minima) The word minima is used as a plural and indicates the possibility of multiple values.
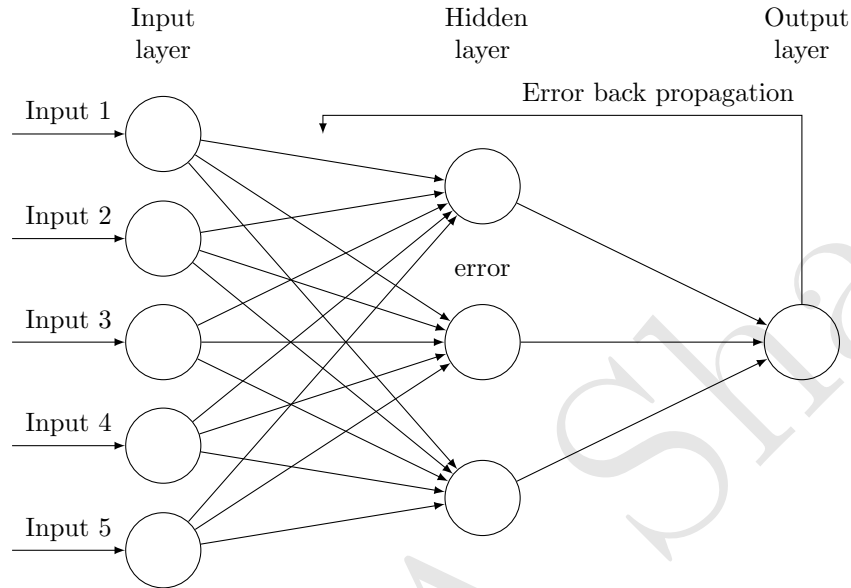
> **Example 1.2 _Why do we have multiple local minima_** _In a DNN, there are multiple combination of weights.So multiple minima are possible_

> **Example 1.3 _Why do we compute absolute mean error over multiple points_** _We have multiple data points. In order to comprehensively evaluate our model, we need to evaluate it over multiple data points.So mean absolute error is used to compute error over multiple data points. Another option is to compute root mean square error._

1.5   Back Propagation

Back Propagation is the process of adjusting weights based on the error. Remember that error is the difference between actual output and the esstimated output from the NN.

So as we move backwards from output layer to hidden layers and then to the input layers, we adjust weights which will give them a close to expected output. The figure below explains the process.

Input   Hidden   Output
layer   layer   layer

Error back propagation

Input 1

Input 2

error

Input 3

Input 4

Input 5

Three algorithms for back propagation are used.

1. Gradient Descent

2. Stochastic Gradient Descent

3. Mini Batch Gradient Descent

The latter two algorithms are variants of the Gradient Descent Algorithm. We will first study the gradient descent algorithm.

1.5.1   Gradient Descent Algorithm

The gradient descent algorithm allows us to adjust weights in order to reduce error.

The algorithm works as follows:

1. Select initial weights

2. Compute slope of the error function

3. choose the steepest slope.

4. Choose the descent direction (adjust weights) opposite to the direction of the slope

5. the next value of weight is computed using the product of slope and the learning rate, where the learning rate is the magnitude of adjustment.

6. 1-5 iterate until convergence

The algorithm starts with an initial point and then repeatedly takes a step opposite to the gradient direction of the function at the current point. The algorithm can be written as follows $f(x)$ is as follows:

for $k = 0, 1, 2, \ldots$ **do**
    $g_k \leftarrow \nabla f(x_k)$
 $x_{k+1} \leftarrow x_k - t_k g_k$
end for

In summary, difference of weights (or weights adjustments) can be mentioned as follows:

$$\Delta W_i = -\alpha \frac{\partial E}{\partial W_i}$$

In the above equation, α gives us the amount of step size; whereas,

$$\frac{\partial E}{\partial W_i}$$

helps in determining the slope. The goal is to select the steepest slope in order to get the desired path to global minima. The slope helps in finding the direction towards the global minima.
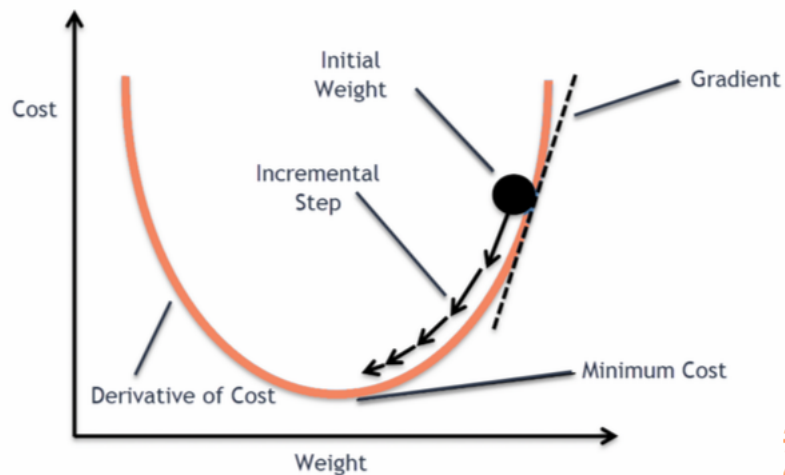
Figure 2: Gradient Descent

Figure 2 explains gradient descent algorithm. The error function (cost function) has been plotted as a function weights. Initial weights are randomly selected. Incremental steps are shown. The magnitude of these steps are determined by α, the learning rate, whereas the direction is selected opposite to the gradient. The algorithm will converge after determining the minimum cost.

In addition to local minima and the (rare) possibility of getting stuck at a saddle point, there are other issues that should be taken into consideration.

For example, if the step size t_k remains large, it may lead to an oscillatory behavior that does not converge.

Another issue is that, depending on the function and starting point, gradient descent could continue indefinitely because there is no minimum. Consider for example minimizing e^x: there is no finite x for which $\frac{d}{dx}e^x = 0$. Other functions could have such asymptotic minima but also a global minimum of lower value; gradient descent, depending on its starting point, might forever chase the asymptote, unaware of the true answer elsewhere in the search space.

> **Example 1.4** *Why do we adjust weights in the opposite direction of slope* *Our aim is to minimize the value of loss function. Therefore, we would like to move in the direction where the loss function decreases the most. This is the opposite direction of slope.*

1.5.2   Variants of Gradient Descent

:

Optimization algorithm will be used to determine how to adjust weights. We will study three variants for optimization algorithm:

1. **Batch Gradient Descent**: Compute gradient descent over all the points in a dataset. That is, update weights after completing the iteration over the whole batch. For instance, if the training dataset contains 1000 data examples, then weights are adjusted after completing the whole iteration over 1000 data points.

   Pros

   (a) Stable descent
   (b) Less Oscillations

   Cons

   (a) May get stuck in local minima
   (b) Extremely slow

2. **Stochastic Gradient Descent**: Select a random point and compute gradient w.r.t that point.

   Pros

(a) Faster than Batch Gradient Descent

(b) Can converge faster

(c) Likely to find error which is near to the global minima

Cons

(a) Higher Oscillations

(b) Faster than gradient descent but may not reach global minima

3. **Mini Batch Gradient Descent**: Divide training into mini batches and select random training examples from these mini batches.

Example 1.5 *Differentiate between epochs, iterations, and Batch*
Epoch:An epoch consists of one or more batches. It implies one forward pass and one backward pass of all the training examples.
***Batch size:**This denotes the number of training examples in one forward/backward pass.*
***Iterations:** This denotes the number of passes using the batch size, where one pass is equal to forward and backward pass.*
***For instance:** With Batch size of 500 and given 2000 training examples, we need four iterations to complete one epoch.*

1.5.3   Chain Rule

Suppose that we have two functions $f(x)$ and $g(x)$ and they are both differentiable.

1. If we define $F(x) = (f \circ g)(x)$ then the derivative of $F(x)$ is,

$$F'(x) = f'(g(x)) \; g'(x)$$

2. If we have $y = f(u)$ and $u = g(x)$ then the derivative of y is,

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

Figure 3: Chain rule explained

(a) $f(x) = \sin(3x^2 + x)$ *Hide Solution* ▼

It looks like the outside function is the sine and the inside function is $3x^2+x$. The derivative is then.

$$f'(x) = \underbrace{\cos}_{\substack{\text{derivative of} \\ \text{outside function}}} \underbrace{(3x^2 + x)}_{\substack{\text{leave inside} \\ \text{function alone}}} \underbrace{(6x + 1)}_{\substack{\text{times derivative} \\ \text{of inside function}}}$$

Or with a little rewriting,

$$f'(x) = (6x + 1)\cos(3x^2 + x)$$

(b) $f(t) = \left(2t^3 + \cos(t)\right)^{50}$ *Hide Solution* ▼

In this case the outside function is the exponent of 50 and the inside function is all the stuff on the inside of the parenthesis. The derivative is then.

$$f'(t) = 50\left(2t^3 + \cos(t)\right)^{49}\left(6t^2 - \sin(t)\right)$$
$$= 50\left(6t^2 - \sin(t)\right)\left(2t^3 + \cos(t)\right)^{49}$$

Figure 4: Chain rule examples

For the above figure, it is worth noting that derivative of cosx is -sinx, whereas the derivative of sinx is cosx.

1.5.4   Learning BackPropagation

[2][t!]   [Backpropagation of errors through the network.]Once evaluated for all output units, the errors $\delta_i^{(L+1)}$ can be propagated backwards..
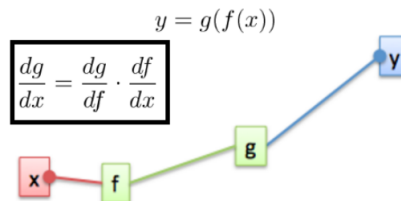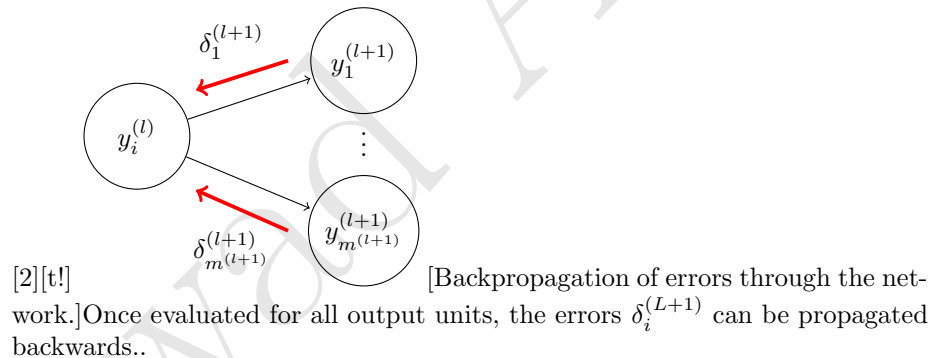

Figure 5: back propagation example