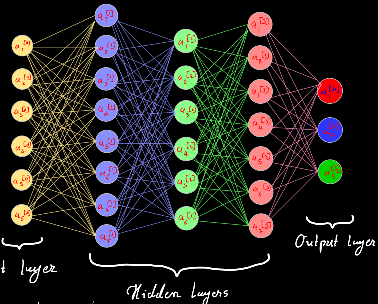
Forward Propogation & Backward Propogation in Neural Network



where $a_n^{[m]}$ — n : neuron number in a layer; m : layer number of a neuron

$$a_1^{[1]} = af(w_{11}^1 \times a_1^0 + w_{12}^1 \times a_2^0 + w_{13}^1 \times a_3^0 + w_{14}^1 \times a_4^0 + w_{15}^1 \times a_5^0 + w_{16}^1 \times a_6^0 + B)$$

Forward Propogation

$$Z_1 = W_1 \times A_0 + B_1 \qquad Z_2 = W_2 \times A_1 + B_2 \qquad Z_3 = W_3 \times A_2 + B_3$$
$$A_1 = f(Z_1) \rightarrow A_2 = f(Z_2) \rightarrow A_3 = f(Z_3) = \text{O/p prediction}$$

Process

Assign random weight (w) at initially ⟹ Model Training ⟹ Update the value of weights in such a way so that it can predict more accurately.

Cost function

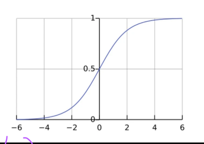
$$W_{new} = W_{old} - \alpha * \frac{\partial cost}{\partial w}$$

Where
α = learning rate
$\frac{\partial cost}{\partial w}$ = slope

Activation functions

1. Sigmoid function

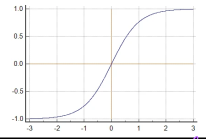
$$f(x) = \frac{1}{1 + e^{-x}}$$



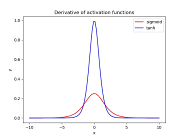
→ Output range 0 to 1 hence we can use it on output layer. (Binary classification)

2. Tanh function

$$\tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



→ Also called as hyperbolic tangent function.
→ Range -1 to 1.
→ We need to take derivative of activation functions in order to calculate derivative of cost function.

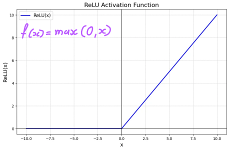


→ As this image shows the derivatives of both function.
→ Here tanh has more derivative than sigmoid that's why by using tanh in hidden layers of NN, we can accelerate the training process much faster.
→ In tanh, average of data is close to '0' (data is normalized centre around '0')
→ Hence if we pass normalized data to next layer, it makes training much more easier.

Note: Both sigmoid & tanh has smaller derivative value at both corner side. this will leads to slow learning also called as vanishing gradient problem. this problem can be solved by next activation function.

3. ReLU (Rectified Linear Unit)

Created with Notewise



→ Overcomes the Vanishing Gradient Problem

→ $\dfrac{\partial f(x)}{\partial x} = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$

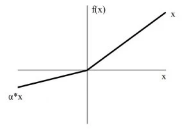
→ also called as Piece-wise linear.

→ Advantages of both linear & non-linear property.

→ linearity of a function can helps to overcome vanishing gradient problem and also makes training faster.

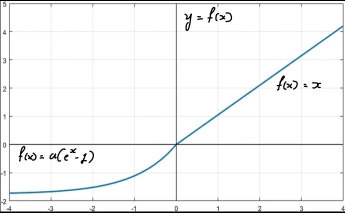
Variations of ReLU

[a] Leaky ReLU



$f(x) = max(0.01 * x, x)$

[b]. ELU [Exponential Linear Unit]

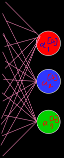


$f(x) = max(\alpha^*(e^x - 1, x), x)$

4. Softmax function

→ For multi-class classification.

first method...



→ 7

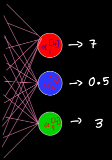
→ 0.5

→ 3

converting into probability

$$\begin{bmatrix} 7 \\ 0.5 \\ 3 \end{bmatrix} \Rightarrow \begin{bmatrix} e^7 \\ e^{0.5} \\ e^3 \end{bmatrix} \Rightarrow \begin{bmatrix} \dfrac{e^7}{e^7 + e^{0.5} + e^3} \\[2ex] \dfrac{e^{0.5}}{e^7 + e^{0.5} + e^3} \\[2ex] \dfrac{e^3}{e^7 + e^{0.5} + e^3} \end{bmatrix} \Rightarrow \begin{bmatrix} 0.9782 \\ 0.00147 \\ 0.0179 \end{bmatrix}$$

Second method.



$$\begin{bmatrix} 7 \\ 0.5 \\ 3 \end{bmatrix} \Longrightarrow \begin{bmatrix} \dfrac{7}{7 + 0.5 + 3} \\ \dfrac{0.5}{7 + 0.5 + 3} \\ \dfrac{3}{7 + 0.5 + 3} \end{bmatrix} \Longrightarrow \begin{bmatrix} 0.6096 \\ 0.0435 \\ 0.2608 \end{bmatrix}$$

→ We can also calculate probability as shown in above, but by using exponential method, it will expend the value hence we can get more accuracy.

$$\text{Softmax } f_i(x) = \frac{e^{a_i}}{\sum e^{a_k}}$$

Note: If the o/p of neural network is continues in this case, it is not sutable to use any kind of non-linear activation function at o/p neuron.

→ In this situation, we will just take the linear o/p without activation function.

Cost functions

Regression / Linear regression

Binary Classification

Multi-class classification.

1. Regression problem

↳ ex. House prize prediction

$$\text{cost} = \underbrace{\frac{1}{m}\sum_{i=1}^{m} |\hat{y}_i - y_i|}_{\text{Mean Absolute error}} \quad \text{or} \quad \underbrace{\frac{1}{2m}\sum_{i=1}^{m} (\hat{y}_i - y_i)^2}_{\text{Mean Squared error}}$$

$\text{Loss} = |\hat{y}_i - y_i|$ for i^{th} observation

2. For binary classification

$$\text{error} = -[y_i \cdot \log(a_i) + (1-y_i)\log(1-a_i)] \quad \text{for 1 observation}$$

$$\text{cost} = -\frac{1}{m}\sum[y_i \cdot \log(a_i) + (1-y_i)\log(1-a_i)]$$

Note if $a_i \to 0$ then error will be ↓

$a_i \to 1$ then error will be ↑

$a_i \to y_i$ then we will get less error.

$! a_i \to y_i$ then we will get more error.

→ This method is also called as 'Binary Cross Entropy'.

3. For multi-class classification

Suppose

$$y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad a_i = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.3 \\ 0.2 \end{bmatrix}$$

error =

$$[y_{i1} \cdot \log(a_{i1}) + y_{i2} \cdot \log(a_{i2}) + y_{i3} \cdot \log(a_{i3}) + y_{i4} \cdot \log(a_{i4})]$$

$$\vdots$$

$$\frac{-[y_{i2} \cdot \log(a_{i2})]}{m} = \frac{-\log(0.5)}{m}$$

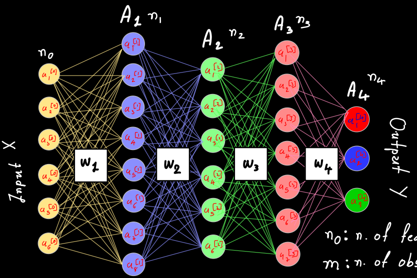
$$\boxed{\text{Cost} = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * \log(a_{ij}))}$$

n: Total number of neurons in o/p layer.

m: Total number of observations.

Created with Notewise

Derivation of back propogation



n_0 : n. of features

m : n. of observations

Shape of $W_1 = (n_1, n_0)$ Shape of $B_1 = (n_1, 1)$ Shape of $X = (n_0, m)$

Shape of $W_2 = (n_2, n_1)$ Shape of $B_2 = (n_2, 1)$ Shape of $A_1 = (n_1, m)$

Shape of $W_3 = (n_3, n_2)$ Shape of $B_3 = (n_3, 1)$ Shape of $A_2 = (n_2, m)$

Shape of $W_4 = (n_4, n_3)$ Shape of $B_4 = (n_4, 1)$ Shape of $A_3 = (n_3, m)$

Shape of $A_4 = (n_4, m)$

Shape of $Y = (n_4, m)$

Activation functions

$i = [1, 2, 3]$ (ReLU / tanh)

$A_i = f_i(Z_i)$

For O/P layer

A_4 (Sigmoid / Softmax)

$$Z_k = W_k \cdot A_{k-1} + B_k$$

Process of backward Propogation

$\frac{\partial L}{\partial W_4}$ L = Loss of i^{th} data point

$\underbrace{L \to a_4 \to z_4 \to W_4}$

$$\text{loss} = -[y_i \cdot \log(a_i) + (1-y_i)\log(1-a_i)]$$

a_4 : Sigmoid (z_4) $z_4 = W_4 \cdot A_3 + B_4$

$$\frac{\partial L}{\partial W_4} = \frac{\partial L}{\partial a_4} \cdot \frac{\partial a_4}{\partial z_4} \cdot \frac{\partial z_4}{\partial W_4}$$

Steps of NN training

1. Initialize parameters randomly.
2. Repeat the process
   - Forward Propogation
   - Activation function
   - Calculating cost
   - Backward Propogation
   - Updating weights & bias