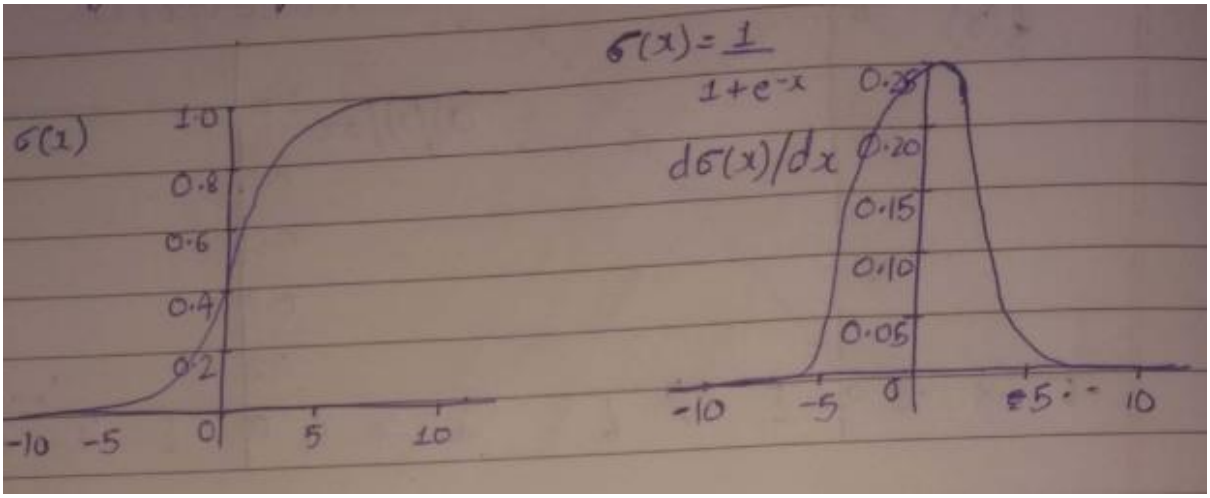
Sigmod function

The function formula and chart are as follows



The Sigmoid function is the most frequently used activation function in the beginning of deep learning. It is a smoothing function that is easy to derive.

In the sigmoid function, we can see that its output is in the open interval (0,1). We can think of probability, but in the strict sense, don't treat it as probability. The sigmoid function was once more popular. It can be thought of as the firing rate of a neuron. In the middle where the slope is relatively large, it is the sensitive area of the neuron. On the sides where the slope is very gentle, it is the neuron's inhibitory area.

The function itself has certain defects.

1) When the input is slightly away from the coordinate origin, the gradient of the function becomes very small, almost zero. In the process of neural network backpropagation, we all use the chain rule of differential to calculate the differential of each weight w. When the backpropagation passes through the sigmod function, the differential on this chain is very small. Moreover, it may pass through many sigmod functions, which will eventually cause the weight w to have little effect on the loss function, which is not conducive to the optimization of the weight. This The problem is called gradient saturation or gradient dispersion.

2) The function output is not centered on 0, which will reduce the efficiency of weight update.

3) The sigmod function performs exponential operations, which is slower for computers.

Advantages of Sigmoid Function : -

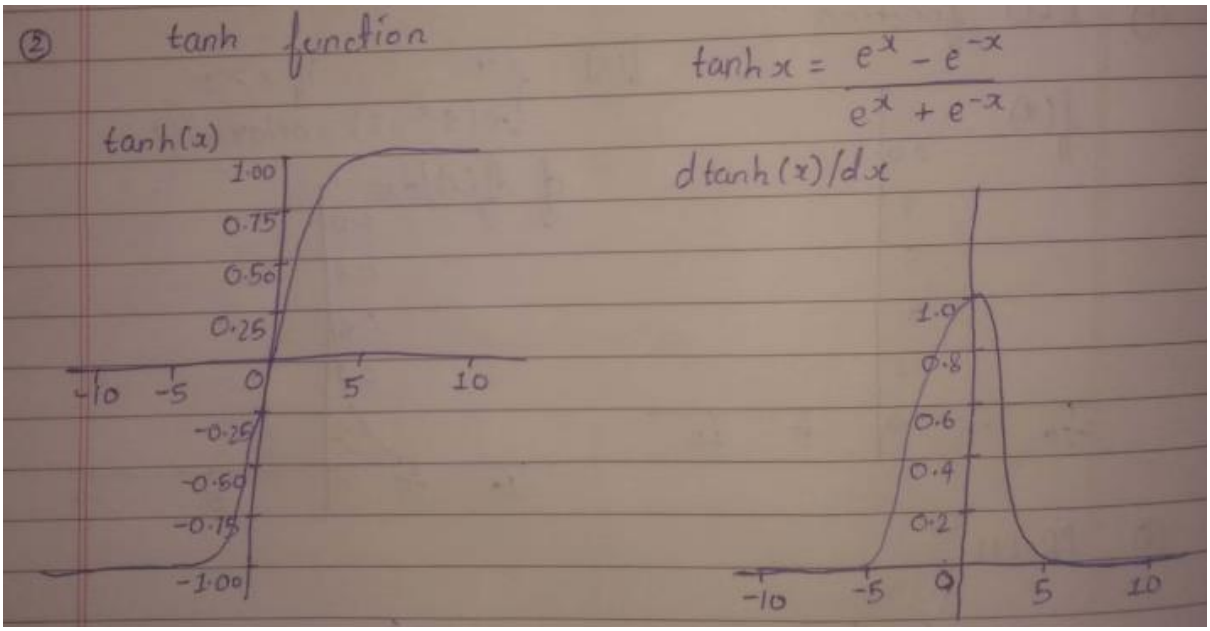
1. Smooth gradient, preventing “jumps” in output values.
2. Output values bound between 0 and 1, normalizing the output of each neuron.
3. Clear predictions, i.e very close to 1 or 0.

Sigmoid has three major disadvantages:

- Prone to gradient vanishing
- Function output is not zero-centered
- Power operations are relatively time consuming

tanh function

The tanh function formula and curve are as follows



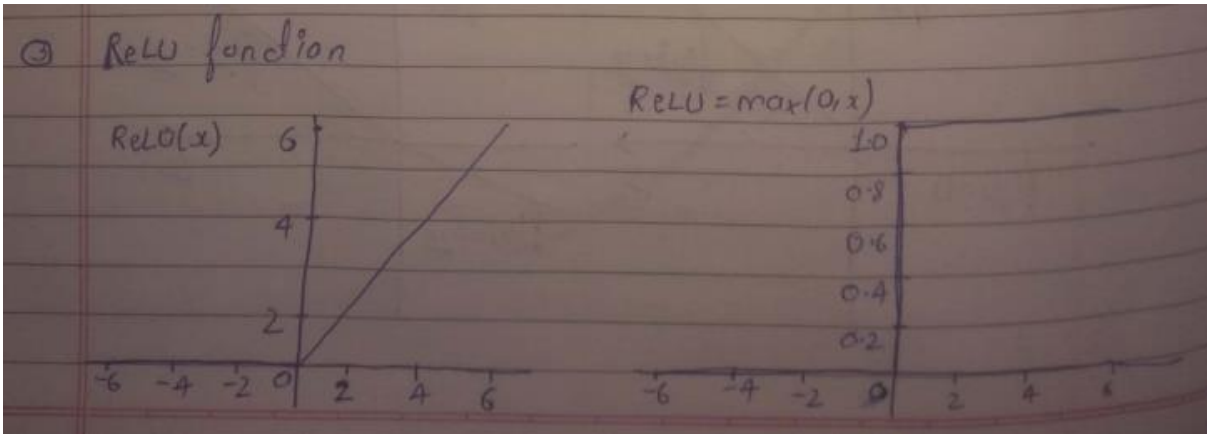
Tanh is a hyperbolic tangent function. The curves of tanh function and sigmod function are relatively similar. Let ’s compare them. First of all, when the input is large or small, the output is almost smooth and the gradient is small, which is not conducive to weight update. The difference is the output interval.

The output interval of tanh is 1), and the whole function is 0-centric, which is better than sigmod.

In general binary classification problems, the tanh function is used for the hidden layer and the sigmod function is used for the output layer. However, these are not static, and the specific activation function to be used must be analyzed according to the specific problem, or it depends on debugging.

ReLU function

ReLU function formula and curve are as follows



ReLU function is actually a function that takes the maximum value. Note that this is not fully interval-derivable, but we can take sub-gradient, as shown in the figure above. Although ReLU is simple, it is an important achievement in recent years.

The ReLU (Rectified Linear Unit) function is an activation function that is currently more popular. Compared with the sigmod function and the tanh function, it has the following advantages:

1) When the input is positive, there is no gradient saturation problem.

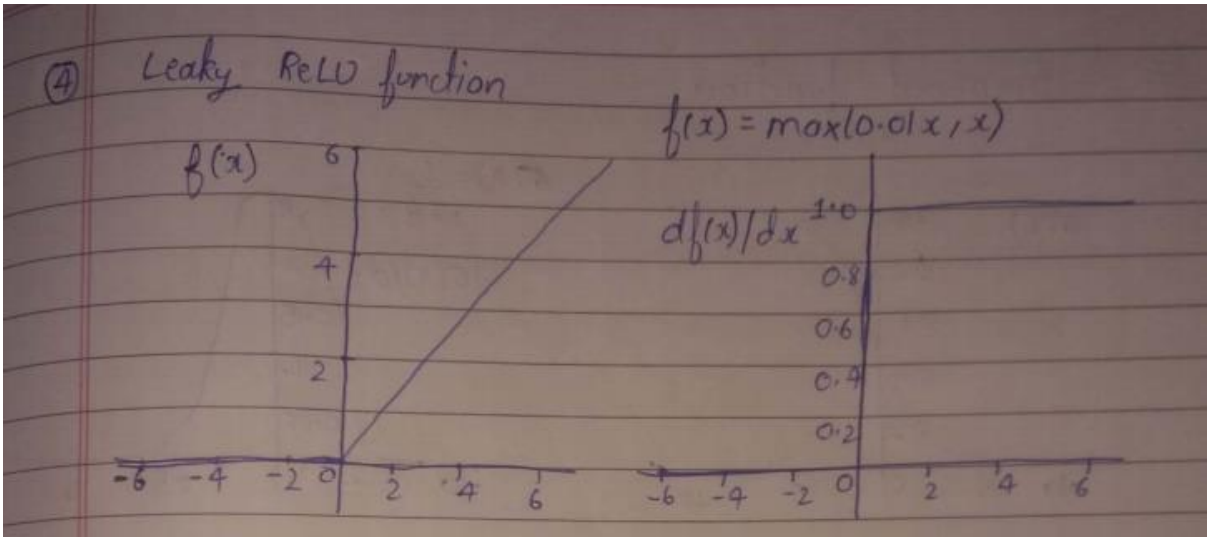
2) The calculation speed is much faster. The ReLU function has only a linear relationship. Whether it is forward or backward, it is much faster than sigmod and tanh. (Sigmod and tanh need to calculate the exponent, which will be slower.)

Ofcourse, there are disadvantages:

1) When the input is negative, ReLU is completely inactive, which means that once a negative number is entered, ReLU will die. In this way, in the forward propagation process, it is not a problem. Some areas are sensitive and some are insensitive. But in the backpropagation process, if you enter a negative number, the gradient will be completely zero, which has the same problem as the sigmod function and tanh function.

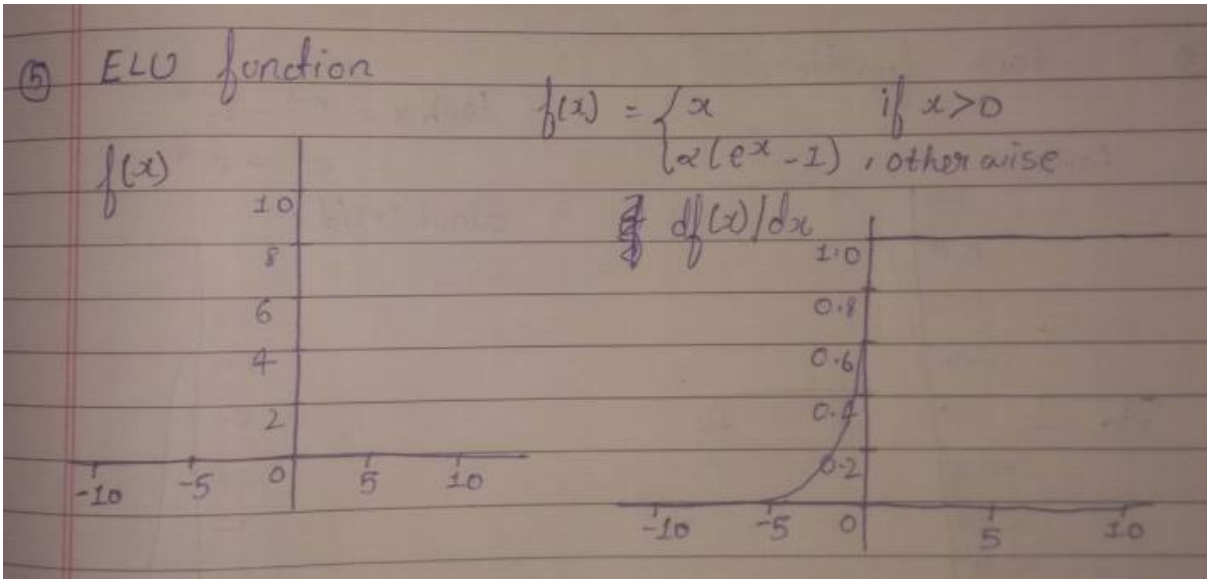
2) We find that the output of the ReLU function is either 0 or a positive number, which means that the ReLU function is not a 0-centric function.

Leaky ReLU function¶



In order to solve the Dead ReLU Problem, people proposed to set the first half of ReLU 0.01x instead of 0. Another intuitive idea is a parameter-based method, Parametric ReLU : f(x)= max(alpha x,x), which alpha can be learned from back propagation. In theory, Leaky ReLU has all the advantages of ReLU, plus there will be no problems with Dead ReLU, but in actual operation, it has not been fully proved that Leaky ReLU is always better than ReLU.

ELU (Exponential Linear Units) function



ELU is also proposed to solve the problems of ReLU. Obviously, ELU has all the advantages of ReLU, and:

- No Dead ReLU issues
- The mean of the output is close to 0, zero-centered

One small problem is that it is slightly more computationally intensive. Similar to Leaky ReLU, although theoretically better than ReLU, there is currently no good evidence in practice that ELU is always better than ReLU.

Softmax

$$S(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}, j = 1, 2, \ldots, K$$

for an arbitrary real vector of length K, Softmax can compress it into a real vector of length K with a value in the range (0, 1), and the sum of the elements in the vector is 1.

It also has many applications in Multiclass Classification and neural networks. Softmax is different from the normal max function: the max function only outputs the largest value, and Softmax ensures that smaller values have a smaller probability and will not be discarded directly. It is a "max" that is "soft".

The denominator of the Softmax function combines all factors of the original output value, which means that the different probabilities obtained by the Softmax function are related to each other. In the case of binary classification, for Sigmoid, there are:

$$p(y=1|x)=\frac{e^{\theta_1^T x}}{e^{\theta_0^T x}+e^{\theta_1^T x}}=\frac{1}{1+e^{(\theta_0^T-\theta_1^T)x}}=\frac{1}{1+e^{-\beta x}}$$

$$p(y=0|x)=\frac{e^{\theta_0^T x}}{e^{\theta_0^T x}+e^{\theta_1^T x}}=\frac{e^{(\theta_0^T-\theta_1^T)x}}{1+e^{(\theta_0^T-\theta_1^T)x}}=\frac{e^{-\beta x}}{1+e^{-\beta x}}$$

For Softmax with K = 2, there are:

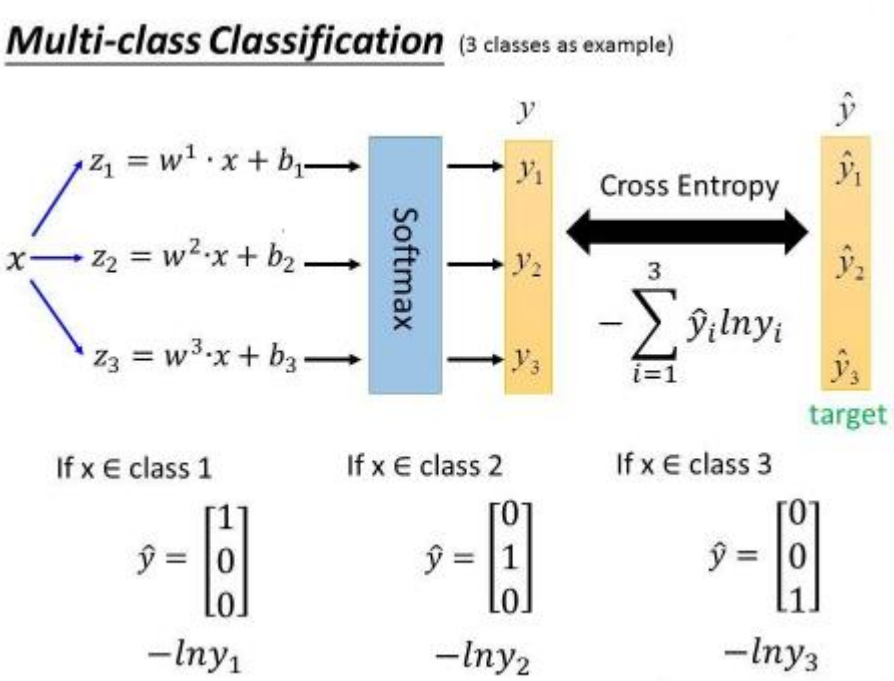
$$p(y=1|x)=\frac{1}{1+e^{-\theta^T x}}$$

$$p(y=0|x)=1-p(y=1|x)=\frac{e^{-\theta^T x}}{1+e^{-\theta^T x}}$$

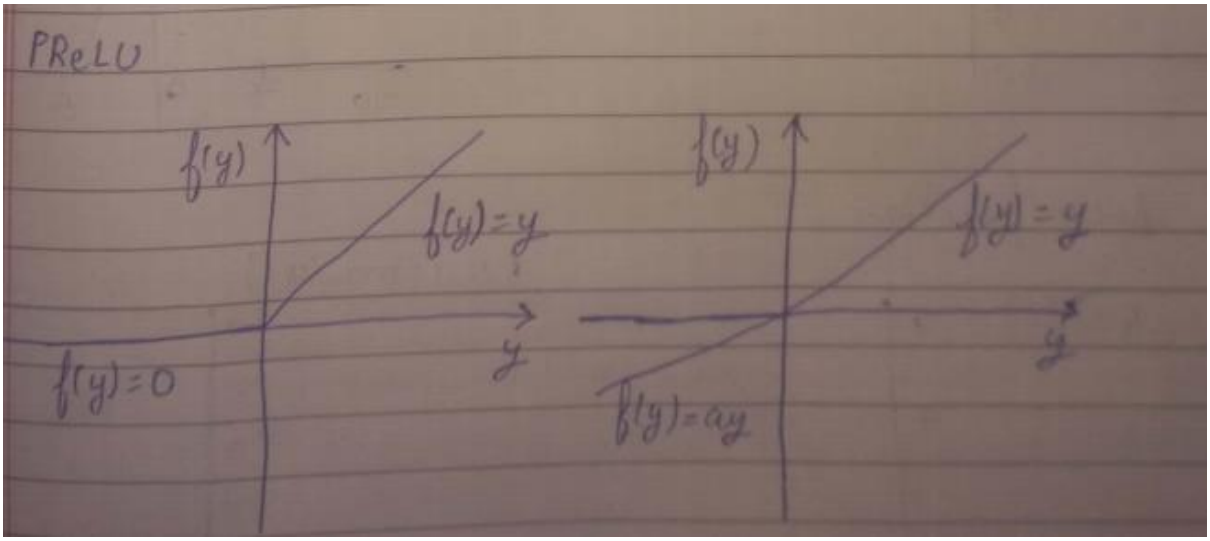
Among them: It

$$\beta=-(\theta_0^T-\theta_1^T)$$

can be seen that in the case of binary classification, Softmax is degraded to Sigmoid.



PRelu (Parametric ReLU)¶



PReLU is also an improved version of ReLU. In the negative region, PReLU has a small slope, which can also avoid the problem of ReLU death. Compared to ELU, PReLU is a linear operation in the negative region. Although the slope is small, it does not tend to 0, which is a certain advantage.

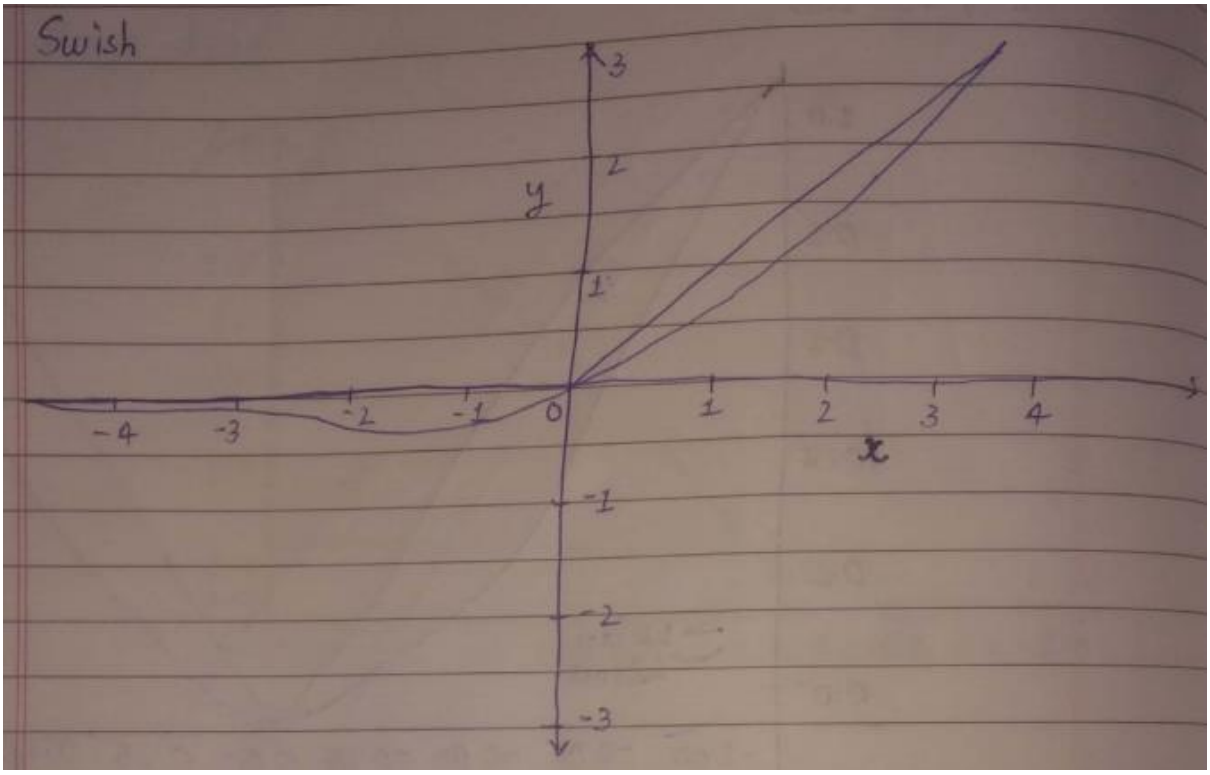
$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}$$

We look at the formula of PReLU. The parameter α is generally a number between 0 and 1, and it is generally relatively small, such as a few zeros. When α = 0.01, we call PReLU as Leaky Relu , it is regarded as a special case PReLU it.

Above, y_i is any input on the ith channel and a_i is the negative slope which is a learnable parameter.

- if a_i=0, f becomes ReLU
- if a_i>0, f becomes leaky ReLU
- if a_i is a learnable parameter, f becomes PReLU

Swish (A Self-Gated) Function



The formula is: **y = x * sigmoid (x)**

Swish's design was inspired by the use of sigmoid functions for gating in LSTMs and highway networks. We use the same value for gating to simplify the gating mechanism, which is called **self-gating**.

The advantage of self-gating is that it only requires a simple scalar input, while normal gating requires multiple scalar inputs. This feature enables self-gated activation functions such as Swish to easily replace activation functions that take a single scalar as input (such as ReLU) without changing the hidden capacity or number of parameters.

1) Unboundedness (unboundedness) is helpful to prevent gradient from gradually approaching 0 during slow training, causing saturation. At the same time, being bounded has advantages, because bounded active functions can have strong reguairzation, and larger negative inputs will be resolved.

2) At the same time, smoothness also plays an important role in optimization and generalization.

Maxout

The Maxout activation function is defined as follows

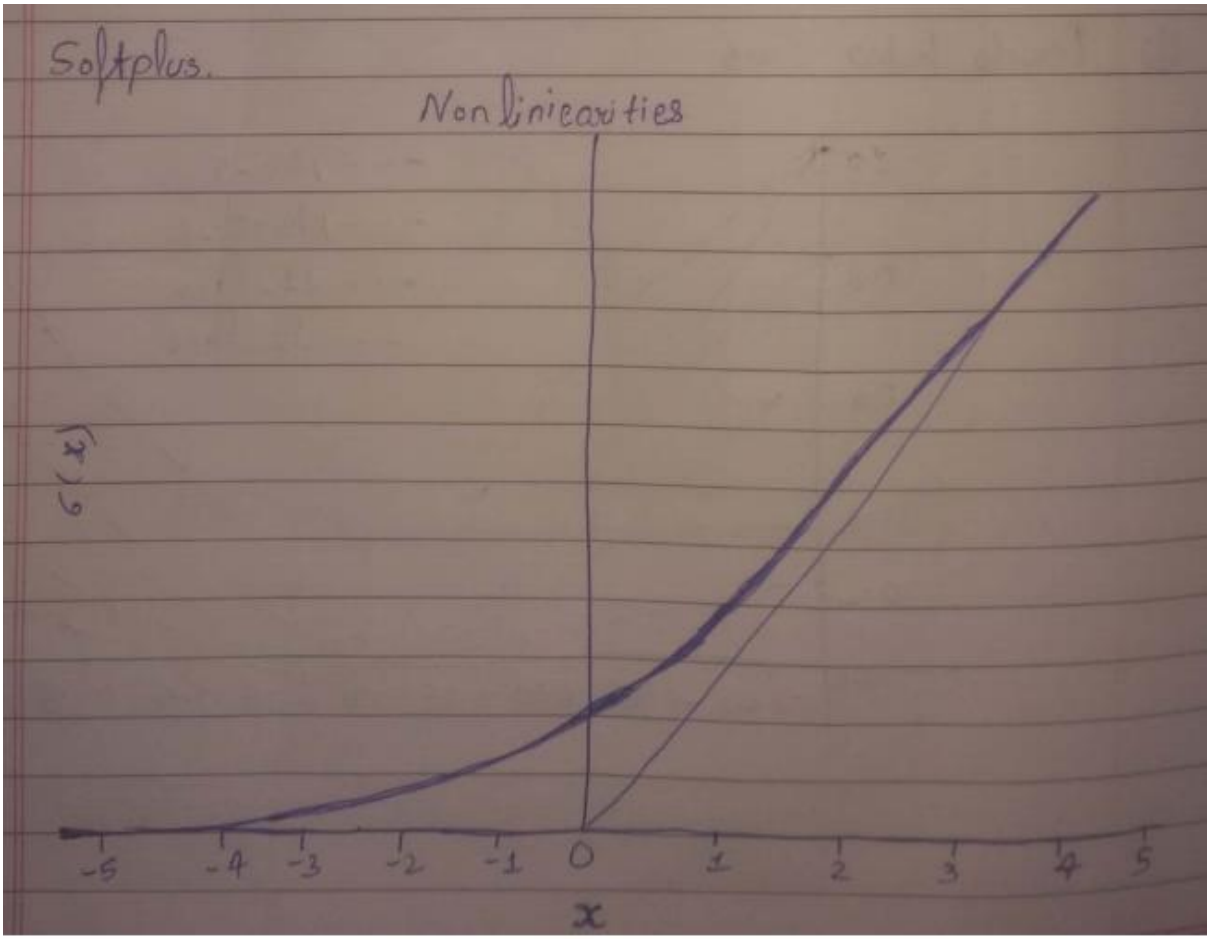
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

One relatively popular choice is the Maxout neuron (introduced recently by Goodfellow et al.) that generalizes the ReLU and its leaky version. Notice that both ReLU and Leaky ReLU are a special case of this form (for example, for ReLU we have w1,b1 =0).The Maxout neuron therefore enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks

The Maxout activation is a generalization of the ReLU and the leaky ReLU functions. It is a learnable activation function.

Maxout can be seen as adding a layer of activation function to the deep learning network, which contains a parameter k. Compared with ReLU, sigmoid, etc., this layer is special in that it adds k neurons and then outputs the largest activation value. value.

Softplus



The softplus function is similar to the ReLU function, but it is relatively smooth.It is unilateral suppression like ReLU.It has a wide acceptance range (0, + inf).

Softplus function: **f(x) = ln(1+exp x)**

Loss Functions

1. L1 and L2 loss

L1 and *L2* are two common loss functions in machine learning which are mainly used to minimize the error.

L1 loss function are also known as **Least Absolute Deviations** in short **LAD**. **L2 loss function** are also known as **Least square errors** in short **LS**.

Let's get brief of these two

L1 Loss function

It is used to minimize the error which is the sum of all the absolute differences in between the true value and the predicted value.

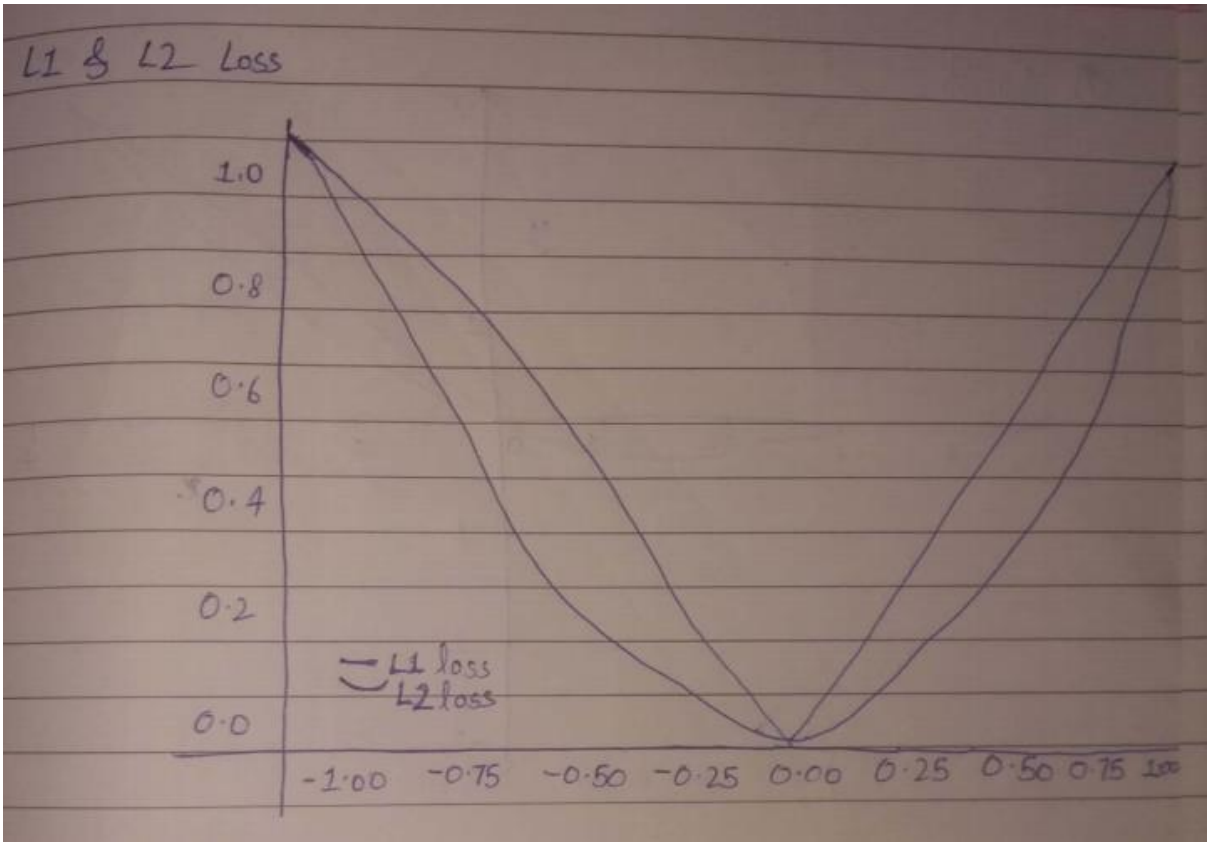
$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

L2 Loss Function

It is also used to minimize the error which is the sum of all the squared differences in between the true value and the pedicted value.

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

The disadvantage of the **L2 norm** is that when there are outliers, these points will account for the main component of the loss. For example, the true value is 1, the prediction is 10 times, the prediction value is 1000 once, and the prediction value of the other times is about 1, obviously the loss value is mainly dominated by 1000.



Huber Loss

Huber Loss is often used in regression problems. Compared with L2 loss, Huber Loss is less sensitive to outliers(because if the residual is too large, it is a piecewise function, loss is a linear function of the residual).

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for} |y - f(x)| \le \delta, \\ \delta |y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Among them, δδ is a set parameter, yy represents the real value, and $f(x)$f(x) represents the predicted value.

The advantage of this is that when the residual is small, the loss function is L2 norm, and when the residual is large, it is a linear function of L1 norm

Pseudo-Huber loss function

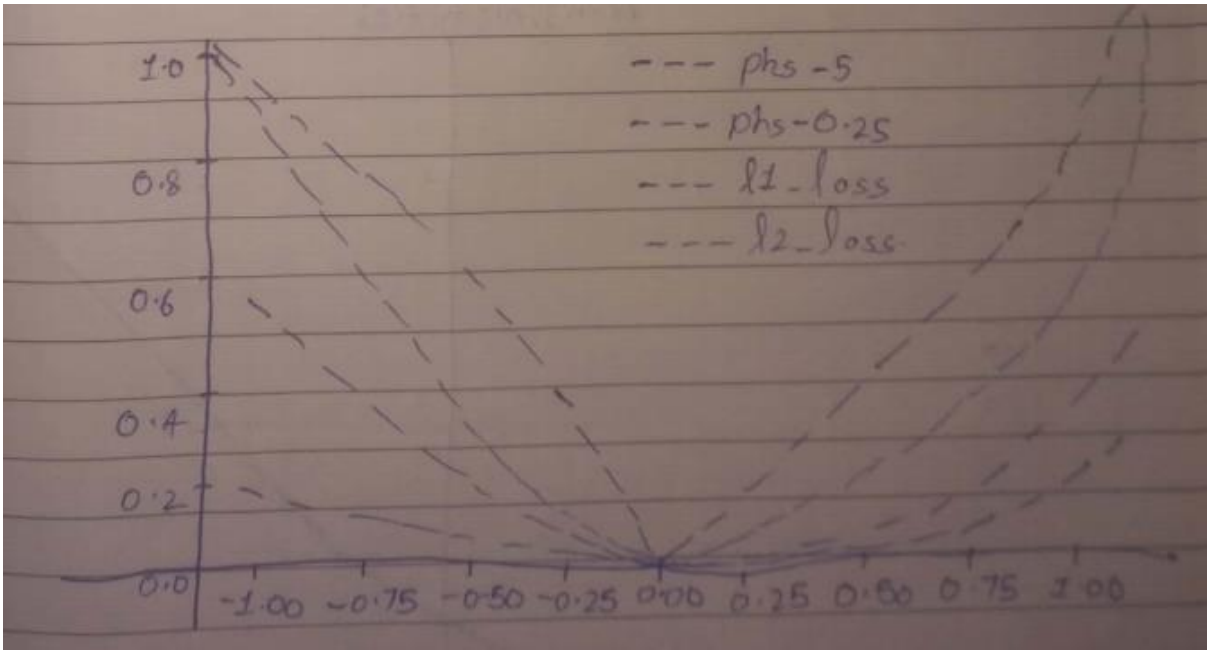
A smooth approximation of Huber loss to ensure that each order is differentiable.

$$L_\delta(a) = \delta^2\left(\sqrt{1 + (a/\delta)^2} - 1\right).$$

As such, this function approximates $a^2/2$ for small values of a, and approximates a straight line with slope δ for large values of a.

While the above is the most common form, other smooth approximations of the Huber loss function also exist.[5]

Where δδ is the set parameter, the larger the value, the steeper the linear part on both sides.



Hinge Loss

Hinge loss is often used for binary classification problems, such as ground true: t = 1 or -1, predicted value y = wx + b

In the svm classifier, the definition of hinge loss is

Hinge loss

From Wikipedia, the free encyclopedia

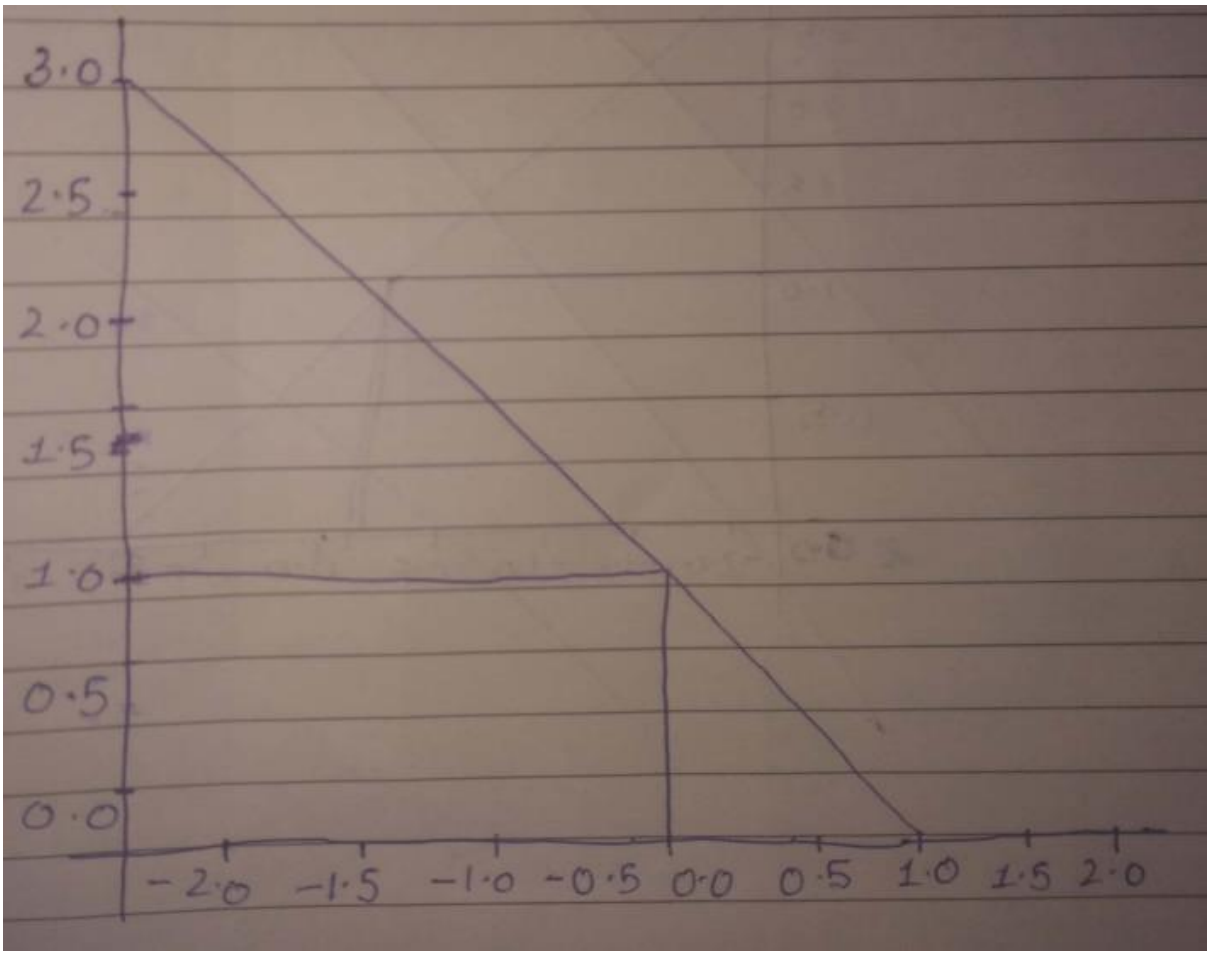
In machine learning, the **hinge loss** is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs).[1] For an intended output $t = \pm 1$ and a classifier score y, the hinge loss of the prediction y is defined as

$$\ell(y) = \max(0, 1 - t \cdot y)$$

Note that y should be the "raw" output of the classifier's decision function, not the predicted class label. For instance, in linear SVMs, $y = \mathbf{w} \cdot \mathbf{x} + b$, where $(\mathbf{w}, b)$ are the parameters of the hyperplane and $\mathbf{x}$ is the point to classify.

It can be seen that when t and y have the same sign (meaning y predicts the right class) and $|y| \geq 1$, the hinge loss $\ell(y) = 0$, but when they have opposite sign, $\ell(y)$ increases linearly with y (one-sided error).

In other words, the closer the y is to t, the smaller the loss will be.



Cross-entropy loss

Cross-entropy loss function and logistic regression [edit]

Cross entropy can be used to define a loss function in machine learning and optimization. The true probability p_i is the true label, and the given distribution q_i is the predicted value of the current model.

More specifically, consider logistic regression, which (among other things) can be used to classify observations into two possible classes (often simply labelled 0 and 1). The output of the model for a given observation, given a vector of input features x, can be interpreted as a probability, which serves as the basis for classifying the observation. The probability is modeled using the logistic function $g(z) = 1/(1+e^{-z})$ where z is some function of the input vector x, commonly just a linear function. The probability of the output $y = 1$ is given by

$$q_{y=1} = \hat{y} \equiv g(\mathbf{w}\cdot\mathbf{x}) = 1/(1+e^{-\mathbf{w}\cdot\mathbf{x}}),$$

where the vector of weights $\mathbf{w}$ is optimized through some appropriate algorithm such as gradient descent. Similarly, the complementary probability of finding the output $y = 0$ is simply given by

$$q_{y=0} = 1-\hat{y}$$

Having set up our notation, $p \in \{y, 1-y\}$ and $q \in \{\hat{y}, 1-\hat{y}\}$, we can use cross entropy to get a measure of dissimilarity between p and q:

$$H(p,q) = -\sum_i p_i \log q_i = -y\log\hat{y} - (1-y)\log(1-\hat{y})$$

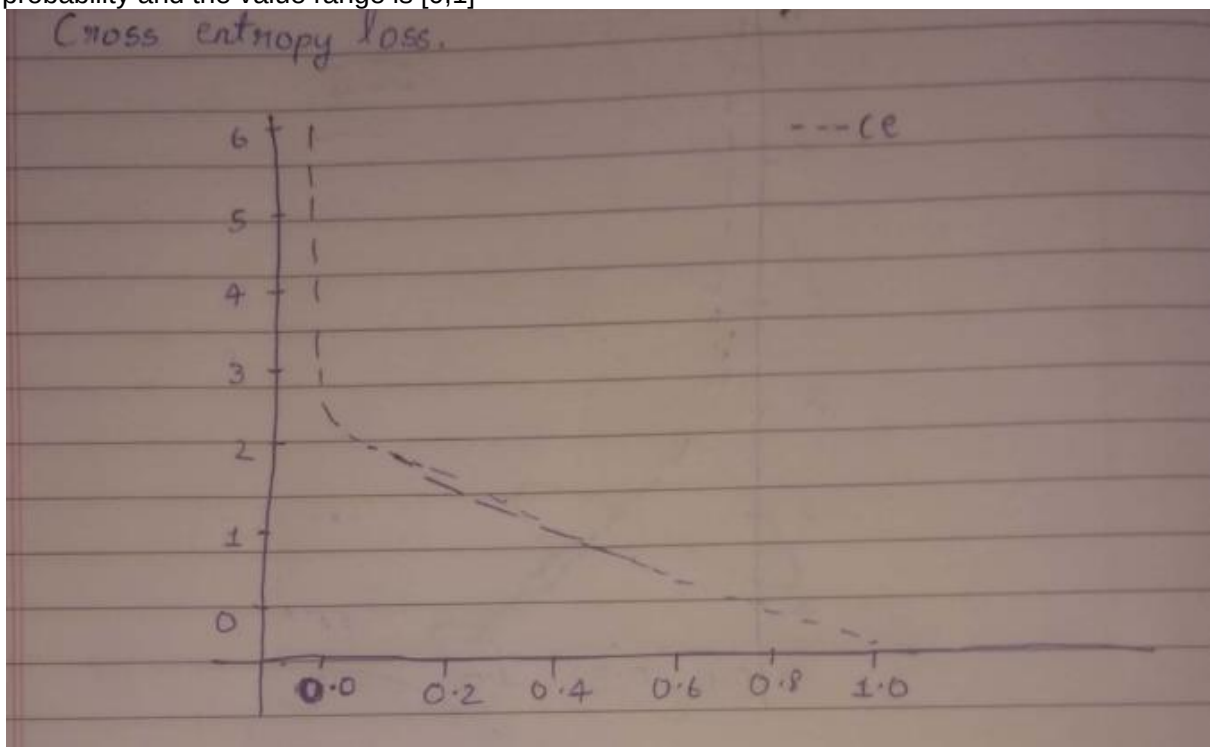
Logistic regression typically optimizes the log loss for all the observations on which it is trained, which is the same as optimizing the average cross-entropy in the sample. For example, suppose we have N samples with each sample indexed by $n = 1, \ldots, N$. The average of the loss function is then given by:

$$J(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log\hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right],$$

where $\hat{y}_n \equiv g(\mathbf{w}\cdot\mathbf{x}_n) = 1/(1+e^{-\mathbf{w}\cdot\mathbf{x}_n})$, with $g(z)$ the logistic function as before.

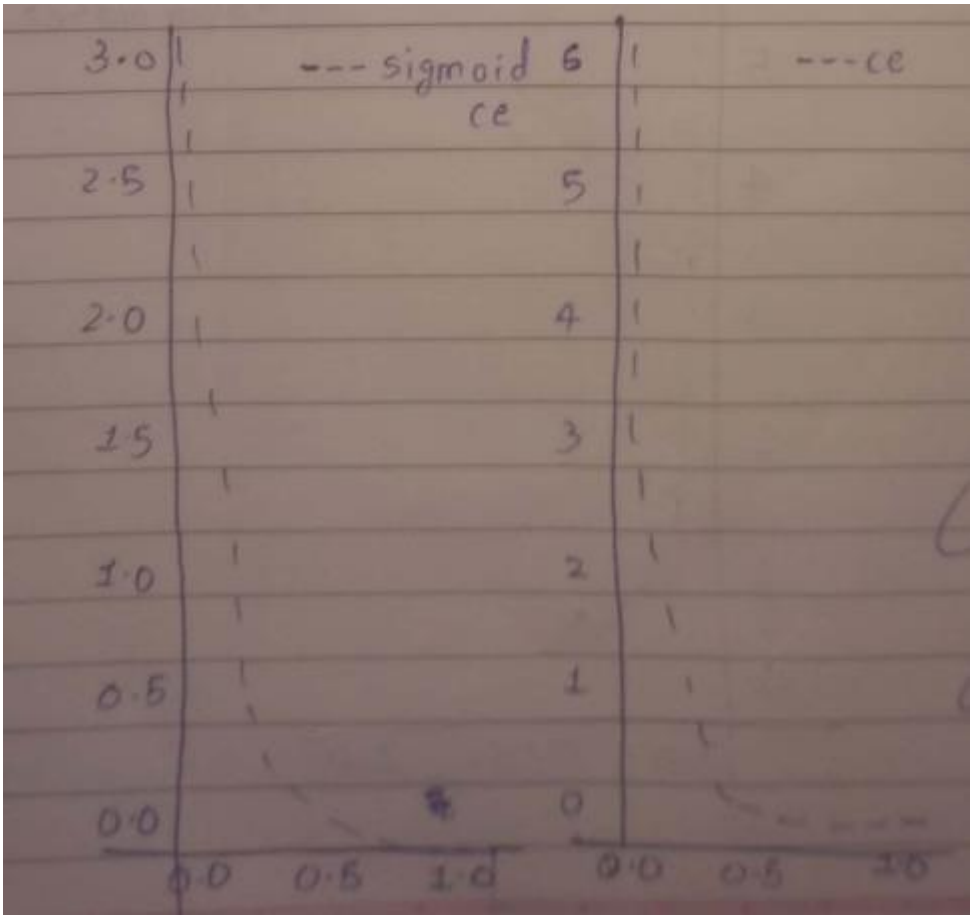
The logistic loss is sometimes called cross-entropy loss. It is also known as log loss (In this case, the binary label is often denoted by {-1,+1}).[2]

The above is mainly to say that cross-entropy loss is mainly applied to binary classification problems. The predicted value is a probability value and the loss is defined according to the cross entropy. Note the value range of the above value: the predicted value of y should be a probability and the value range is [0,1]



Sigmoid-Cross-entropy loss

The above cross-entropy loss requires that the predicted value is a probability. Generally, we calculate $scores = x * w + b$scores=x∗w+b. Entering this value into the sigmoid function can compress the value range to (0,1).



It can be seen that the sigmoid function smoothes the predicted value(such as directly inputting 0.1 and 0.01 and inputting 0.1, 0.01 sigmoid and then entering, the latter will obviously have a much smaller change value), which makes the predicted value of sigmoid-ce far from the label loss growth is not so steep.

Softmax cross-entropy loss

First, the softmax function can convert a set of fraction vectors into corresponding probability vectors. Here is the definition of softmax function

> In mathematics, the **softmax function**, or **normalized exponential function**,[1]:198 is a generalization of the logistic function that "squashes" a K-dimensional vector $\mathbf{z}$ of arbitrary real values to a K-dimensional vector $\sigma(\mathbf{z})$ of real values in the range (0, 1] that add up to 1. The function is given by
>
> $$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

As above, softmax also implements a vector of 'squashes' k-dimensional real value to the [0,1] range of k-dimensional, while ensuring that the cumulative sum is 1.

According to the definition of cross entropy, probability is required as input.Sigmoid-cross-entropy-loss uses sigmoid to convert the score vector into a probability vector, and softmax-cross-entropy-loss uses a softmax function to convert the score vector into a probability vector.

According to the definition of cross entropy loss.

$$H(p,q) = -\sum_{x} p(x)\log q(x)$$

where $p(x)$p(x) represents the probability that classification xx is a correct classification, and the value of pp can only be 0 or 1. This is the prior value
$q(x)$q(x) is the prediction probability that the xx category is a correct classification, and the value range is (0,1)
So specific to a classification problem with a total of C types,
then $p(x_j)$p(xj), $(0<=j<=C)$(0<=j<=C) must be only 1 and C-1 is 0(because there can be only one correct classification, correct the probability of classification as correct classification is 1, and the probability of the remaining classification as correct classification is 0)

Then the definition of softmax-cross-entropy-loss can be derived naturally.

Here is the definition of softmax-cross-entropy-loss.

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \qquad \text{or equivalently} \qquad L_i = -f_{y_i} + \log\sum_j e^{f_j}$$

Where $f_j$fj is the score of all possible categories, and f_{y_i}fyi is the score of ground true class