Q No-1 : Define binary cross entropy as cost function

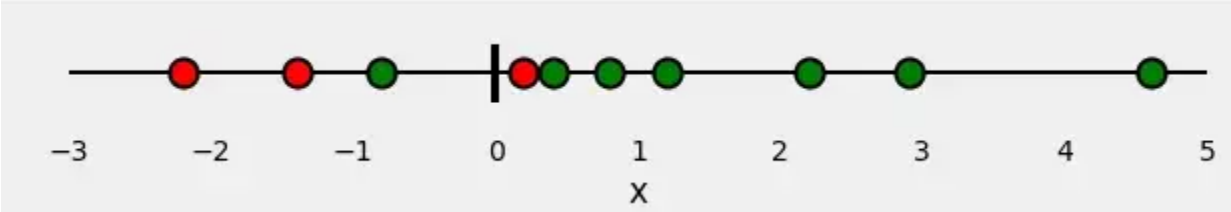
Answer: Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.

If we try to train a binary classification then there is a good chance to use **binary cross-entropy** / log loss as loss function.

Loss Function: Binary Cross-Entropy / Log Loss

$$\text{Log loss} = \frac{1}{N} \sum_{i=1}^{N} -\left(y_i * \log(p_i) + (1-y_i) * \log(1-p_i) \right)$$

Example :



Now let's assign some colors to our points red and green. These are our labels.

So, our classification problem is quite straightforward: given our **feature *x***, we need to predict its **label**: **red** or **green.**

If we fit a model to perform this classification, it will predict a probability of being green to each one of our points. Given what we know about the color of the points, how can we evaluate how good (or bad) are the predicted

probabilities? This is the whole purpose of the loss function! It should return high values for bad predictions and low values for good predictions. For a **binary classification** like our example, the **typical loss function** is the **binary cross-entropy / log loss**.

Now in the above binary cross-entropy, where **y** is the **label** (**1 for green** points and **0 for red** points) and **p(y)** is the predicted **probability of the point being green** for all **N** points.

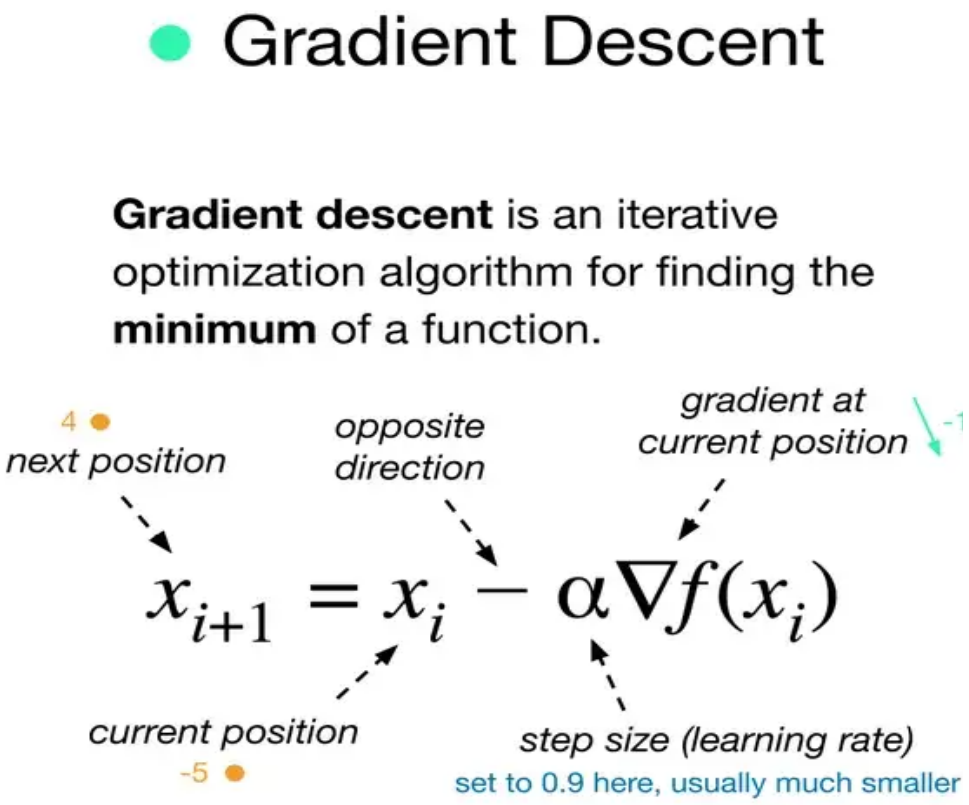
Reading this formula, it tells you that, for each **green** point (*y=1*), it adds *log(p(y))* to the loss, that is, the **log probability of it being green**. Conversely, it adds *log(1-p(y))*, that is, the **log probability of it being red**, for each **red** point (*y=0*).

Q No-2 : Difference between adam optimizer and gradient descent

Answer:

Gradient Descent:

Suppose we have a convex cost function of 2 input variables as shown above and our goal is to minimize its value and find the value of the parameters (x,y) for which f(x,y) is minimum. *What the gradient descent algorithm does is*, we start at a specific point on the curve and use the negative gradient to find the direction of steepest descent and take a small step in that direction and keep iterating till our value starts converging.



Adam Optimizer :

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the 'gradient descent with momentum' algorithm and the 'RMSP' algorithm.

Adam optimizer involves a combination of two gradient descent methodologies:

Momentum:

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the ‘exponentially weighted average’ of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

Root Mean Square Propagation (RMSP):

Root mean square prop or RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the ‘exponential moving average’.

Adam Optimizer inherits the strengths or the positive attributes of the above two methods and builds upon them to give a more optimized gradient descent.

Adam has been one of the most remarkable achievements in the grounds of optimization. Several incidents where the training of a large model required days have been reduced to hours since usage of Adam. Since its inception it has been made the default optimizer used in almost all deep learning libraries

So this optimization algorithm is a further extension of stochastic gradient descent to update network weights during training. Unlike maintaining a single learning rate through training in SGD, **Adam optimizer updates the learning rate for each network weight individually**