**Loss Functions**

A loss function in Machine Learning is a measure of how accurately your ML model is able to predict the expected outcome i.e the ground truth.

The loss function will take two items as input: the output value of our model and the ground truth expected value. The output of the loss function is called the loss which is a measure of how well our model did at predicting the outcome.

A high value for the loss means our model performed very poorly. A low value for the loss means our model performed very well.

It can be categorized into two groups which are –

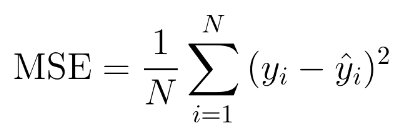
* Classification (discrete values, 0,1,2…)
* Regression (continuous values)

**Regression**

MSE

The Mean Squared Error (MSE) is perhaps the simplest and most common loss function. To calculate the MSE, you take the difference between your model’s predictions and the ground truth, square it, and average it out across the whole dataset.

The MSE will never be negative, since we are always squaring the errors. The MSE is formally defined by the following equation:



Where N is the number of samples we are testing against.

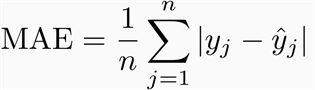
Advantage: The MSE is great for ensuring that our trained model has no outlier predictions with huge errors, since the MSE puts larger weight on theses errors due to the squaring part of the function.

Disadvantage: If our model makes a single very bad prediction, the squaring part of the function magnifies the error. Yet in many practical cases we don’t care much about these outliers and are aiming for more of a well-rounded model that performs good enough on the majority.

MAE

To calculate the MAE, you take the difference between your model’s predictions and the ground truth, apply the absolute value to that difference, and then average it out across the whole dataset.

The MAE, like the MSE, will never be negative since in this case we are always taking the absolute value of the errors. The MAE is formally defined by the following equation:

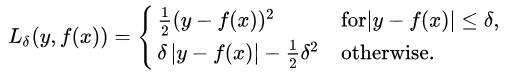


Advantage: The beauty of the MAE is that its advantage directly covers the MSE disadvantage. Since we are taking the absolute value, all of the errors will be weighted on the same linear scale. Thus, unlike the MSE, we won’t be putting too much weight on our outliers and our loss function provides a generic and even measure of how well our model is performing.

Disadvantage: If we do in fact care about the outlier predictions of our model, then the MAE won’t be as effective. The large errors coming from the outliers end up being weighted the exact same as lower errors. This might results in our model being great most of the time, but making a few very poor predictions every so-often.

Huber Loss

The Huber Loss offers the best of both worlds by balancing the MSE and MAE together. We can define it using the following piecewise function:



What this equation essentially says is: for loss values less than delta, use the MSE; for loss values greater than delta, use the MAE. This effectively combines the best of both worlds from the two loss functions!

**Classification**

Cross-entropy

This function comes from information theory where the goal is to measure the difference between two averages of the number of bits of distribution of information.

Entropy is the number of bits required to transmit a randomly selected event from a probability distribution. A skewed distribution has low entropy, whereas a distribution where events have equal probability has a larger entropy.

The cross-entropy is a class of Loss function most used in machine learning because that leads to better generalization models and faster training.

Cross-entropy can be used with binary and multiclass classification problems (many classes with one label, different from many classes with multilabel called multilabel classification).

Types of cross-entropy:

* Binary cross-entropy: for binary classification problem
* Categorical cross-entropy: binary and multiclass problem, the label needs to be encoded as categorical, one-hot encoding representation (for 3 classes: [0, 1, 0], [1,0,0]…)
* Sparse cross-entropy: binary and multiclass problem (the label is an integer — 0 or 1 or … n, depends on the number of labels)

Kullback Leibler Divergence Loss

The KL divergence is the score of two different probability distribution functions. The KL difference between a PDF of q(x) and a PDF of p(x) is noted KL(Q||P) where || means divergence (it is not symmetric KL(P||Q) != KL(Q||P)).

KL(Q||P) = -sum( q(x) \* log(p(x)/q(x)) or sum(q(x)\*log(q(x)/p(x))

The sum is for a discrete case and integral for continuous. This means that the divergence increases if the PDF of q(x) is large and the PDF of p(x) is small for the same data. In machine learning, you can represent this as the difference between the prediction and the ground truth.

**Optimizers**

Deep learning is a great advancement over machine learning in terms of flexibility, higher accuracy, and a wide range of possibilities in industry applications. Deep learning finds its usage in almost every sector. With this much usage, it becomes important that these algorithms run under minimum resources so we can reduce recurring costs and provide efficient results in less time.

An optimizer is a method or algorithm to update the various parameters that can reduce the loss in much less effort.

Gradient Descent (GD)

This is the most basic optimizer that directly uses the derivative of the loss function and learning rate to reduce the loss and achieve the minima. This approach is also adopted in backpropagation in neural networks where the updated parameters are shared between different layers depending upon when the minimum loss is achieved. It is easy to implement and interpret the results, but it has various issues.

The weights are updated when the whole dataset gradient is calculated, which slows down the process. It also requires a large amount of memory to store this temporary data, making it a resource-hungry process. Though the idea behind this algorithm is well suited, it needs to be tweaked.

Stochastic Gradient Descent

This is a changed version of the GD method, where the model parameters are updated on every iteration. It means that after every training sample, the loss function is tested and the model is updated. These frequent updates result in converging to the minima in less time, but it comes at the cost of increased variance that can make the model overshoot the required position.

But an advantage of this technique is low memory requirement as compared to the previous one because now there is no need to store the previous values of the loss functions.

Nesterov Accelerated Gradient (NAG)

The momentum-based GD gave a boost to the currently used optimizers by converging to the minima at the earliest, but it introduced a new problem. This method takes a lot of u-turns and oscillates in and out in the minima valley adding to the total time. The time taken is still way too less than normal GD, but this issue also needs a fix and this is done in NAG.

The approach followed here was that the parameters update would be made with the history element first and then only the derivative is calculated which can move it in the forward or backward direction. This is called the look-ahead approach.

Adagrad

Till now we are only focusing on how the model parameters are affecting our training, but we haven’t talked about the hyper-parameters that are assigned constant value throughout the training. One such important hyper-parameter is learning rate and varying this can change the pace of training.

The solution for this is to have an adaptive learning rate that can change according to the input provided. Adagrad optimizer tries to offer this adaptiveness by decaying the learning rate in proportion to the updated history of the gradients.It means that when there are larger updates, the history element is accumulated, and therefore it reduces the learning rate and vice versa. One disadvantage of this approach is that the learning rate decays aggressively and after some time it approaches zero.

RMSProp

It is an improvement to the Adagrad optimizer. This aims to reduce the aggressiveness of the learning rate by taking an exponential average of the gradients instead of the cumulative sum of squared gradients. Adaptive learning rate remains intact as now exponential average will punish larger learning rate in conditions when there are fewer updates and smaller rate in a higher number of updates.

Adam

Adaptive Moment Estimation combines the power of RMSProp (root-mean-square prop) and momentum-based GD. In Adam optimizers, the power of momentum GD to hold the history of updates and the adaptive learning rate provided by RMSProp makes Adam optimizer a powerful method. It also introduces two new hyper-parameters beta1 and beta2 which are usually kept around 0.9 and 0.99 but you can change them according to your use case.

Conclusion

Picking a winner among these is highly subjective to the use case and the problem you are dealing with but one can surely rank Adam Optimizer on the top because of its combination with the momentum concept that changed how the model parameters should be updated and adapting the changing learning rate for different scenarios enabling efficient processing of any types of inputs.