**Urmi Dedhia**

**Synapse Task 5**

**Loss Functions :** Loss function measures the difference between the true target value of a sample and the estimated target value. If the predicted values deviate too much from the actual results, loss function would end up producing a very large number. Gradually, with the help of some optimization function, the error in predictions is reduced and hence loss function decreases. Different loss functions cater to different models. They are hence classified into two categories for regression and classification models respectively :

Loss functions for regression models :

* Squared loss (ypred - y)2 : This function generally works well for a regression problem but doesn’t work well with a dataset containing several outliers as the squaring increases the error due to the outliers. We can compute the gradient during gradient descent easily with this loss function.
* Absolute loss | ypred – y| : This function doesn’t greatly alter the loss value due to the outliers and hence can be used when we want to ignore the outliers. Compared to the squared loss it is difficult to compute the gradient during gradient descent due to the point of discontinuity.
* Pseudo-Huber loss : This function is a combination of squared and absolute loss functions where if the data point has relatively low error it will take its squared loss and if it is an outlier it will take its absolute loss.

Loss functions for classification models :

* Cross entropy loss : Cross-entropy calculates the number of bits required to transmit an average event from one distribution compared to another distribution.
* KL divergence : The difference between the entropy and cross entropy is known as KL divergence.
* Hinge loss : It splits the datapoints to classify them while maintaining the maximum distance possible from the datapoints. It maximizes the minimum distance from the boundary line.

**Optimizers :** An optimizer is a function that modifies the attributes of the neural network such as weights and learning rate and reduces the loss function value.

* Gradient Descent : In gradient descent, the weights are repeatedly iterated to find the weights for which the loss value is minimum. But sometimes this method falls short as it calculates the values for new weights only after each epoch, which may lead to large jumps in values for weight and optimum weight values might get missed.
* Batch Gradient Descent (BGD) : It calculates and updates the weights only after the entire batch of training examples is processed. Hence SGD is preferred over it as it is computationally less tedious.
* Stochastic Gradient Descent (SGD) : It calculates and updates the weights after each datapoint’s iteration. Hence it is quicker than BGD as newer examples are being processed with better weights which were updated from the iterations before.
* Mini Batch Gradient Descent : It is a middle ground between SGD and BGD as it processes a batch of examples in each iteration.

We can process and reach the optimum weights even faster by using SGD and momentum. The momentum makes the model focus on data points which have a similar pattern and helps learn faster. But this results in it ignoring points which do not match the pattern hence not being trained correctly. We can counter this by involving an acceleration which can increase and decrease the momentum based on the model recognizing whether a point follows the pattern or not.

The above algorithms have a fixed learning rate. There are adaptive optimizers like adagrad, adadelta and adam which do not have a fixed learning rate and they keep updating it with each iteration. Adagrad uses adaptive loss to optimize the data but here, the model tends to learn very slowly after a certain point. To counter this we use adadelta which is a modified version of adagrad in which the learning rate doesn’t suddenly shoot down to 0, hence it is faster than adagrad. Adam is a modified version of adadelta which can update momentum as well along with updating learning rate. It is widely used due to its speed and accuracy.