Ayesha Ashraf

Task # 26

Understanding Optimizers, last –layer Activation Loss Function and Evaluation Metrics

What is an optimizer?

Optimizers are algorithms or methods used to minimize an error function(*loss function*)or to maximize the efficiency of production. Optimizers are mathematical functions which are dependent on model's learnable parameters i.e Weights & Biases. Optimizers help to know how to change weights and learning rate of neural network to reduce the losses.

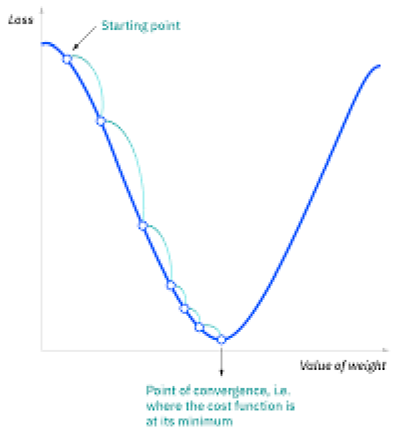
This post will walk you through the optimizers and some popular approaches.

Types of optimizers

Let's learn about different types of optimizers and how they exactly work to minimize the loss function.

Gradient Descent

Gradient descent is an optimization algorithm based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum. Gradient Descent iteratively reduces a loss function by moving in the direction opposite to that of steepest ascent. It is dependent on the derivatives of the loss function for finding minima. uses the data of the entire training set to calculate the gradient of the cost function to the parameters which requires large amount of memory and slows down the process.



Gradient Descent

$$W_{new} = W_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

Advantages of Gradient Descent

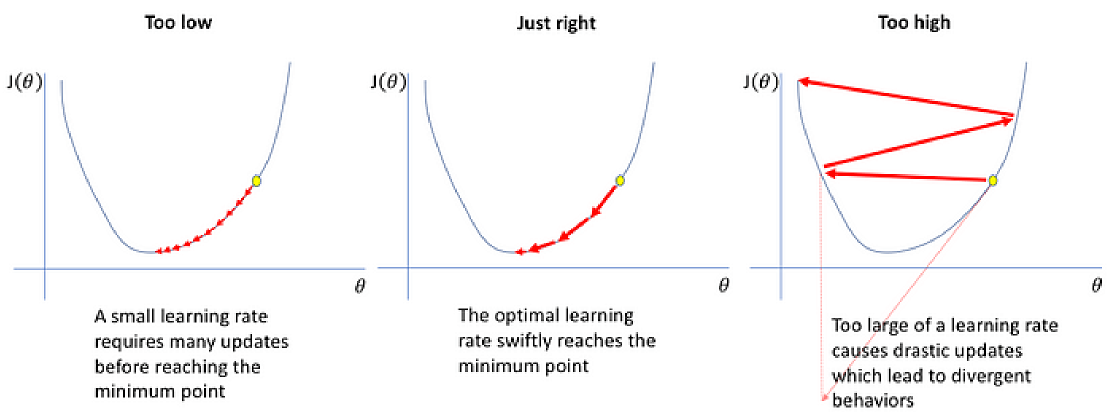
1. Easy to understand
2. Easy to implement

Disadvantages of Gradient Descent

1. Because this method calculates the gradient for the entire data set in one update, the calculation is very slow.
2. It requires large memory and it is computationally expensive.

Learning Rate

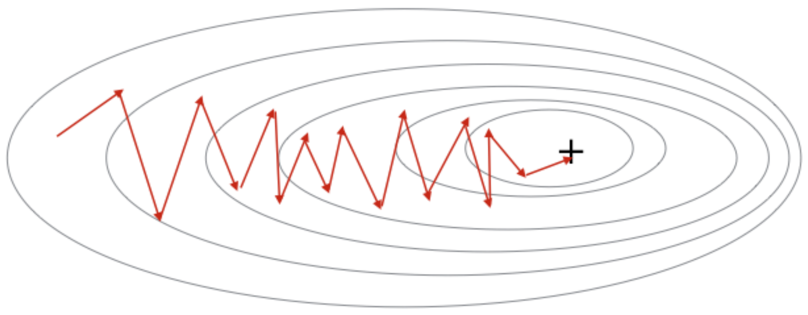
How big/small the steps are gradient descent takes into the direction of the local minimum are determined by the learning rate, which figures out how fast or slow we will move towards the optimal weights.


Too low
Just right
Too high
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Learning Rate

Stochastic Gradient Descent

It is a variant of Gradient Descent. It update the model parameters one by one. If the model has 10K dataset SGD will update the model parameters 10k times.



Stochastic Gradient Descent

Advantages of Stochastic Gradient Descent

1. Frequent updates of model parameter

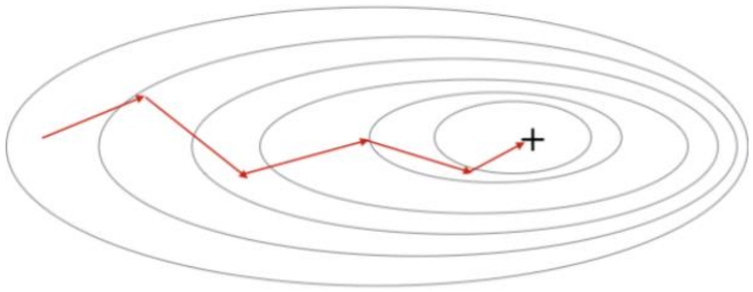
2. Requires less Memory.
3. Allows the use of large data sets as it has to update only one example at a time.

Disadvantages of Stochastic Gradient Descent

1. The frequent can also result in noisy gradients which may cause the error to increase instead of decreasing it.
2. High Variance.
3. Frequent updates are computationally expensive.

Mini-Batch Gradient Descent

It is a combination of the concepts of SGD and batch gradient descent. It simply splits the training dataset into small batches and performs an update for each of those batches. This creates a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent. it can reduce the variance when the parameters are updated, and the convergence is more stable. It splits the data set in batches in between 50 to 256 examples, chosen at random.


Mini Batch Gradient Descent

Advantages of Mini Batch Gradient Descent:

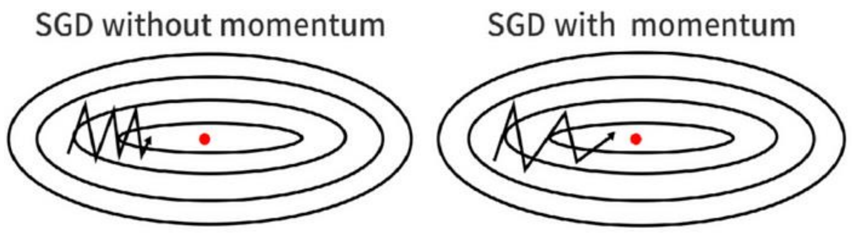
1. It leads to more stable convergence.
2. more efficient gradient calculations.
3. Requires less amount of memory.

Disadvantages of Mini Batch Gradient Descent

1. Mini-batch gradient descent does not guarantee good convergence,
2. If the learning rate is too small, the convergence rate will be slow. If it is too large, the loss function will oscillate or even deviate at the minimum value.

SGD with Momentum

SGD with Momentum is a stochastic optimization method that adds a momentum term to regular stochastic gradient descent. Momentum simulates the inertia of an object when it is moving, that is, the direction of the previous update is retained to a certain extent during the update, while the current update gradient is used to fine-tune the final update direction. In this way, you can increase the stability to a certain extent, so that you can learn faster, and also have the ability to get rid of local optimization.



SGD with Momentum

$$\nu_{new} = \eta * \nu_{old} - \alpha * \frac{\partial(Loss)}{\partial(W_{old})}$$

Momentum Formula

Advantages of SGD with momentum

1. Momentum helps to reduce the noise.
2. Exponential Weighted Average is used to smoothen the curve.

Disadvantage of SGD with momentum

1. Extra hyperparameter is added.

AdaGrad(Adaptive Gradient Descent)

In all the algorithms that we discussed previously the learning rate remains constant. The intuition behind AdaGrad is can we use different Learning Rates for each and every neuron for each and every hidden layer based on different iterations.

$$W_{new} = W_{old} + \frac{\alpha}{\sqrt{cache_{new}} + \epsilon} * \frac{\partial(Loss)}{\partial(W_{old})}$$

Advantages of AdaGrad

1. Learning Rate changes adaptively with iterations.

2. It is able to train sparse data as well.

Disadvantage of AdaGrad

1. If the neural network is deep the learning rate becomes very small number which will cause dead neuron problem.

RMS-Prop (Root Mean Square Propagation)

RMS-Prop is a special version of Adagrad in which the learning rate is an exponential average of the gradients instead of the cumulative sum of squared gradients. RMS-Prop basically combines momentum with AdaGrad.

$$cache_{new} = \gamma * cache_{old} + (1 - \gamma) * (\frac{\partial(Loss)}{\partial(W_{old})})^2$$

Advantages of RMS-Prop

1. In RMS-Prop learning rate gets adjusted automatically and it chooses a different learning rate for each parameter.

Disadvantages of RMS-Prop

1. Slow Learning

AdaDelta

Adadelta is an extension of Adagrad and it also tries to reduce Adagrad’s aggressive, monotonically reducing the learning rate and remove decaying learning rate problem. In Adadelta we do not need to set the default learning rate as we take the ratio of the running average of the previous time steps to the current gradient.

Advantages of Adadelta

1. The main advantage of AdaDelta is that we do not need to set a default learning rate.

Disadvantages of Adadelta

1. Computationally expensive

Adam(Adaptive Moment Estimation)

Adam optimizer is one of the most popular and famous gradient descent optimization algorithms. It is a method that computes adaptive learning rates for each parameter. It stores both the decaying average of the past gradients , similar to momentum and also the decaying average of the past squared gradients , similar to RMS-Prop and Adadelta. Thus, it combines the advantages of both the methods.

$$\mathrm{w}_{\mathrm{t}} = \mathrm{w}_{\mathrm{t}-1} - \frac{\eta}{\sqrt{\mathrm{S}_{\mathrm{dw}_{\mathrm{t}}} - \varepsilon}} * \mathrm{V}_{\mathrm{dw}_{\mathrm{t}}}$$

$$b_{\mathrm{t}} = b_{\mathrm{t}-1} - \frac{\eta}{\sqrt{\mathrm{S}_{db_{\mathrm{t}}} - \varepsilon}} * \mathrm{V}_{db_{\mathrm{t}}}$$

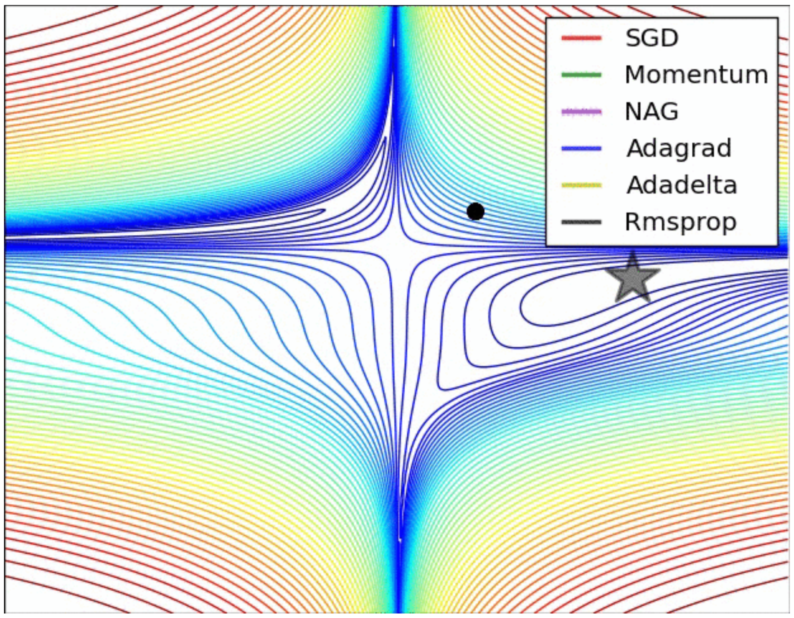
Advantages of Adam

1. Easy to implement

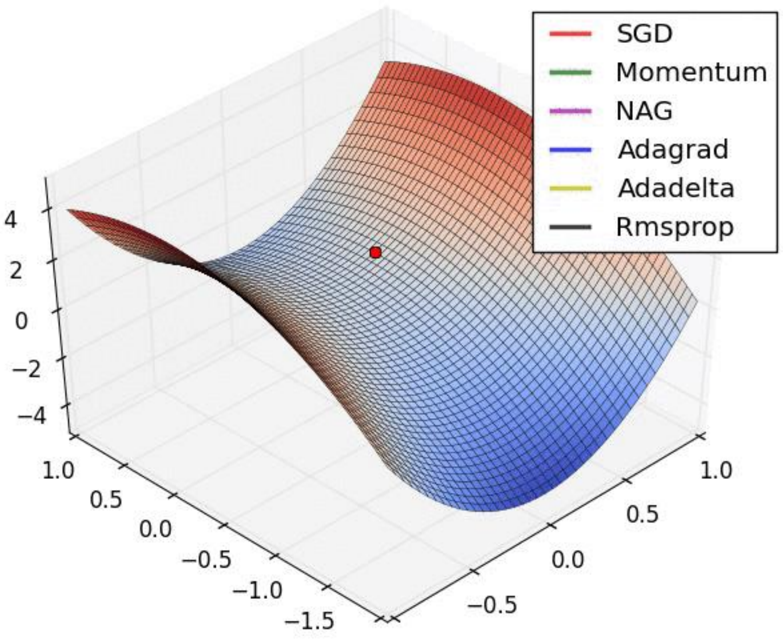
2. Computationally efficient.

3. Little memory requirements.

Comparison



Optimizers Comparison



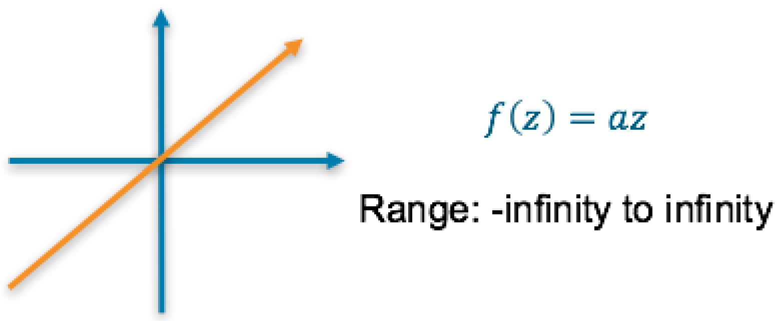
Optimization on saddle point

How to choose optimizers?

- If the data is sparse, use the self-applicable methods, namely Adagrad, Adadelta, RMSprop, Adam.
- RMSprop, Adadelta, Adam have similar effects in many cases.
- Adam just added bias-correction and momentum on the basis of RMSprop,
- As the gradient becomes sparse, Adam will perform better than RMSprop.

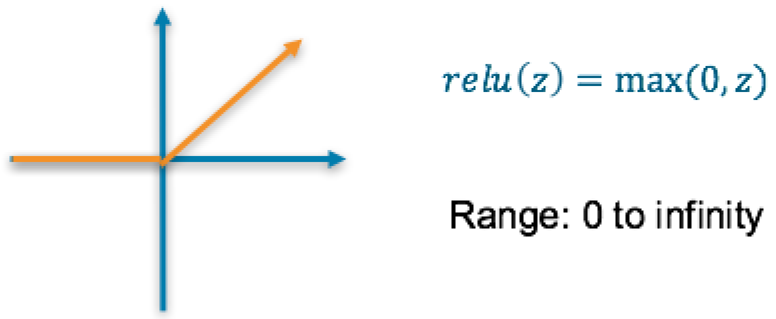
Final Activation Function

Linear — This results in a numerical value which we require



or

ReLU — This results in a numerical value greater than 0



Loss Function

Mean squared error (MSE) — This finds the average squared difference between the predicted value and the true value

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

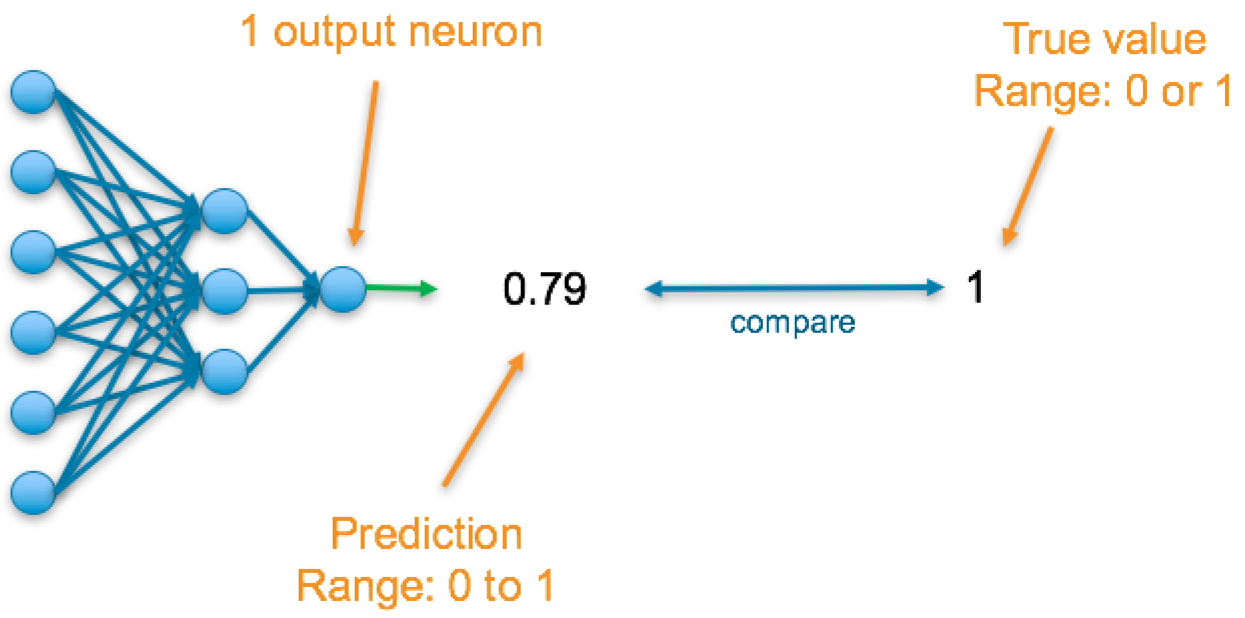
Where $\hat{y}$ is the predicted value and y is the true value

Categorical: Predicting a binary outcome

E.g. predicting a transaction is fraud or not

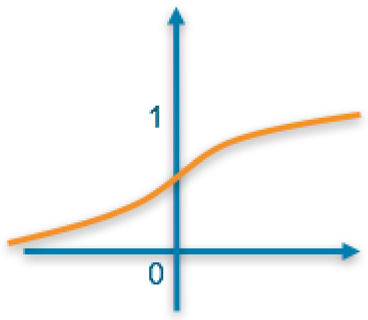
The final layer of the neural network will have one neuron and will return a value between 0 and 1, which can be inferred as a probably.

To understand the accuracy of the prediction, it is compared with the true value. If the data is that class, the true value is a 1, else it is a 0.



Final Activation Function

Sigmoid — This results in a value between 0 and 1 which we can infer to be how confident the model is of the example being in the class



$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

Range: 0 to 1

Loss Function

Binary Cross Entropy — Cross entropy quantifies the difference between two probability distribution. Our model predicts a model distribution of {p, 1-p} as we have a binary distribution. We use binary cross-entropy to compare this with the true distribution {y, 1-y}

$$\text{Binary cross entropy} = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

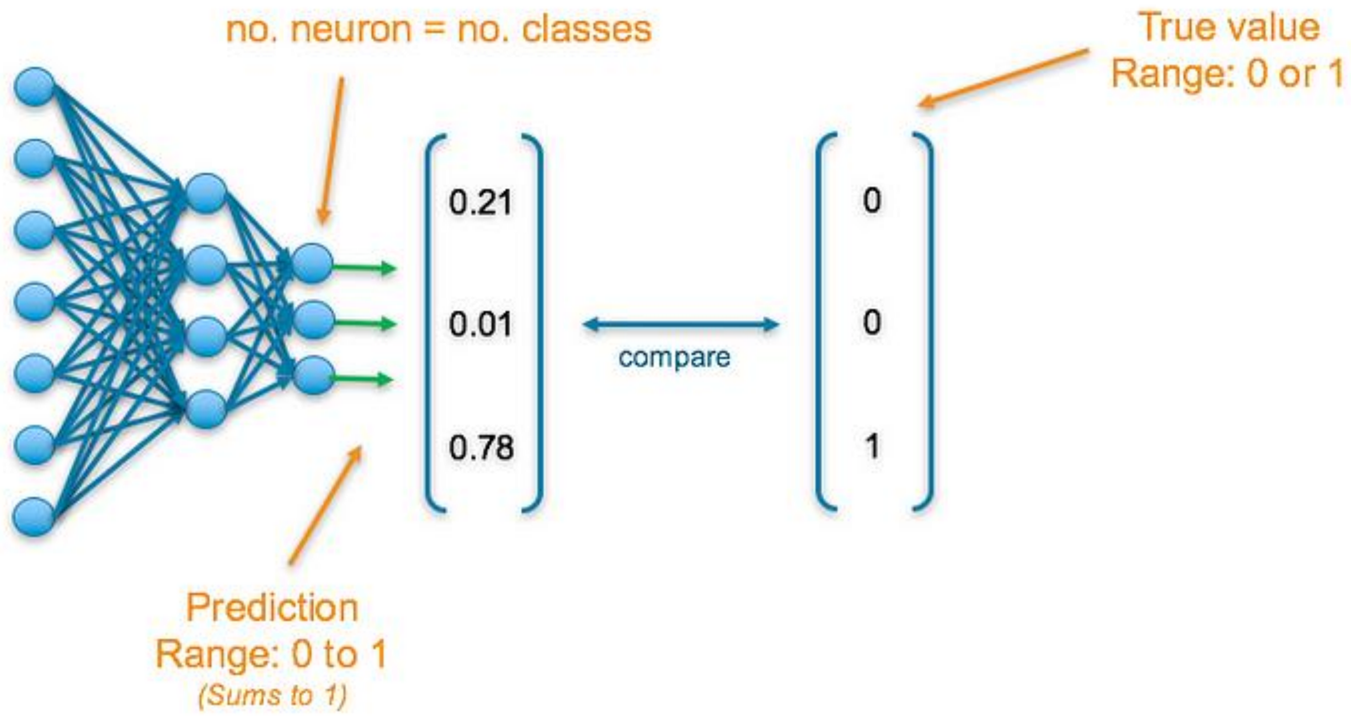
Where $\hat{y}$ is the predicted value and y is the true value

Categorical: Predicting a single label from multiple classes

E.g. predicting the document's subject

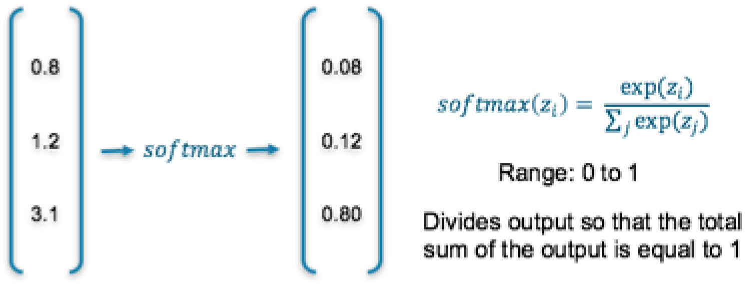
The final layer of the neural network will have one neuron for each of the classes and they will return a value between 0 and 1, which can be inferred as a probably. The output then results in a probability distribution as it sums to 1.

To understand the accuracy of the prediction, each output is compared with its corresponding true value. True values have been one-hot-encoded meaning a 1 appears in the column corresponding to the correct category, else a 0 appears



Final Activation Function

Softmax — This results in values between 0 and 1 for each of the outputs which all sum up to 1. Consequently, this can be inferred as a probability distribution


0.8
1.2
3.1
softmax
0.08
0.12
0.80
$softmax(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$
Range: 0 to 1
Divides output so that the total sum of the output is equal to 1

Loss Function

Cross Entropy — Cross entropy quantifies the difference between two probability distribution. Our model predicts a model distribution of {p1, p2, p3} (where p1+p2+p3 = 1). We use cross-entropy to compare this with the true distribution {y1, y2, y3}

$$\text{Cross entropy} = -\sum_i^M y_i \log(\hat{y}_i)$$

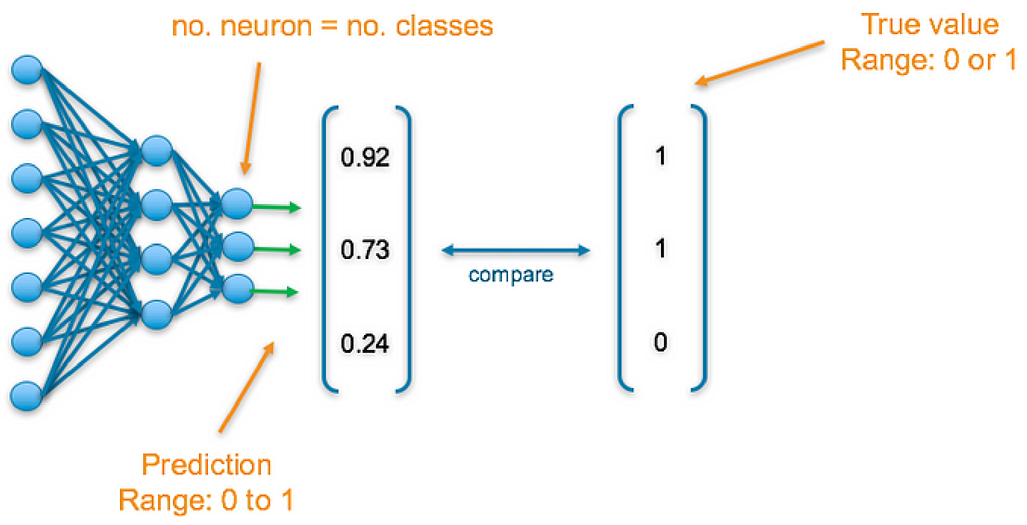
Where $\hat{y}$ is the predicted value, y is the true value and M is the number of classes

Categorical: Predicting multiple labels from multiple classes

E.g. predicting the presence of balls in an image

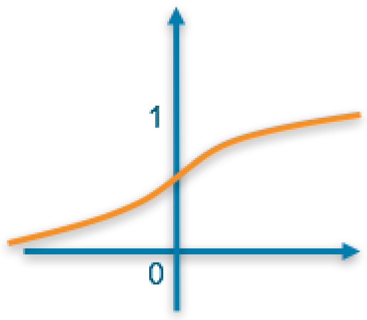
The final layer of the neural network will have one neuron for each of the classes and they will return a value between 0 and 1, which can be inferred as a probably.

To understand the accuracy of the prediction, each output is compared with its corresponding true value. If 1 appears in the true value column, the category it corresponds to is present in the data, else a 0 appears.



Final Activation Function

Sigmoid — This results in a value between 0 and 1 which we can infer to be how confident it is of it being in the class



$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

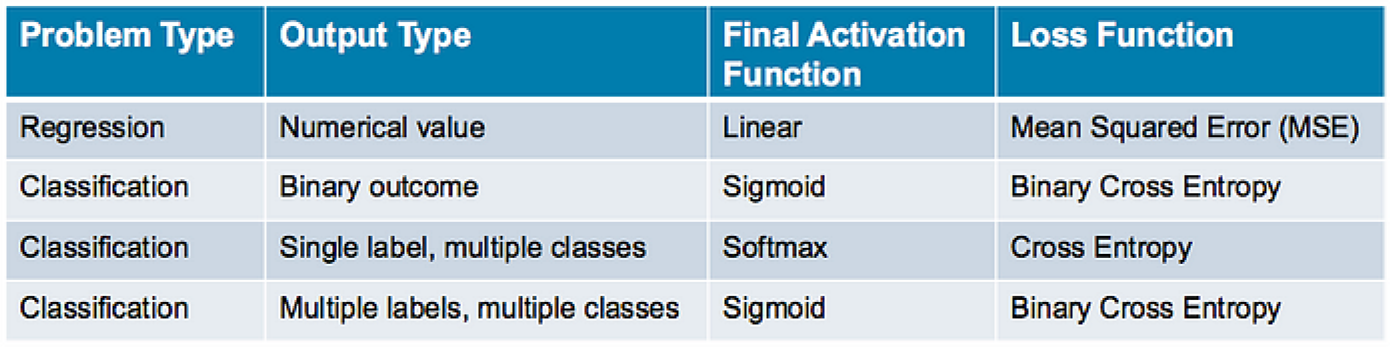
Range: 0 to 1

Loss Function

Binary Cross Entropy — Cross entropy quantifies the difference between two probability distribution. Our model predicts a model distribution of {p, 1-p} (binary distribution) for each of the classes. We use binary cross-entropy to compare these with the true distributions {y, 1-y} for each class and sum up their results

Binary cross entropy = $-\sum_i^M (y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$
Where $\hat{y}$ is the predicted value and y is the true value

The following table summarizes the above information to allow you to quickly find the final layer activation function and loss function that is appropriate to your use-case



Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

What are Evaluation Metrics?

Evaluation metrics are used to measure the quality of the statistical or machine learning model. Evaluating machine learning models or algorithms is essential for any project. There are many different types of evaluation metrics available to test a model. These include classification accuracy, logarithmic loss, confusion matrix, and others. Classification accuracy is the ratio of the number of correct predictions to the total number of input samples, which is usually what we refer to when we use the term accuracy. Logarithmic loss, also called log loss, works by penalizing the false classifications. A confusion matrix gives us a matrix as output and describes the complete performance of the model. There are other evaluation metrics that can be used that have not been listed. Evaluation metrics involves using a combination of these individual evaluation metrics to test a model or algorithm.

Why is this Useful?
It is very important to use multiple evaluation metrics to evaluate your model. This is because a model may perform well using one measurement from one evaluation metric, but may perform poorly using another measurement from another evaluation metric. Using evaluation metrics are critical in ensuring that your model is operating correctly and optimally.

Applications of Evaluation Metrics

- Statistical Analysis
- Machine Learning