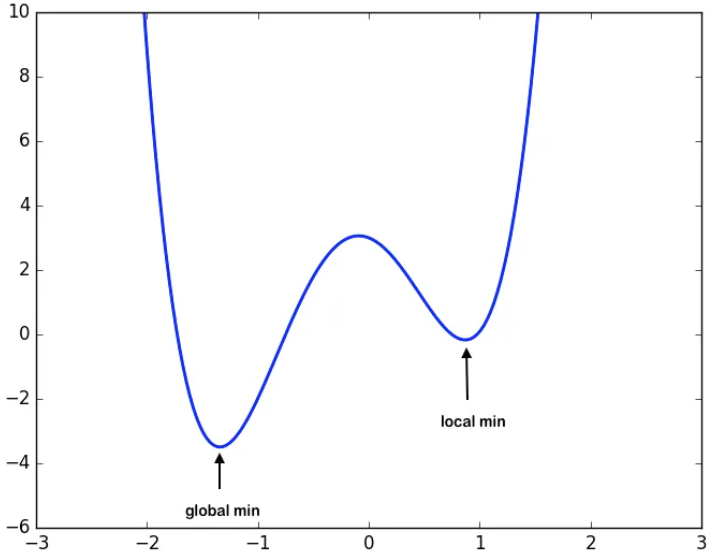
DIFFERENT TYPES OF OPTIMIZER IN ANN

An optimizer in a neural network is an algorithm that updates the network's weights during training. The goal of an optimizer is to find the best weights to minimize the loss function, which measures how well the network predicts the target variable.



How does an optimizer work?

* **Loss function**: The loss function measures how well the network is performing. A lower loss means the network is performing better.
* **Trial and error**: Optimizers use trial and error to find the best weights.
* **Gradient descent**: A popular optimization algorithm that uses gradients to find the best step to take to reduce the loss.

Why is an optimizer important?

* The choice of optimizer can significantly impact the performance of the model.
* Optimizers help the network go from making random guesses to making accurate guesses

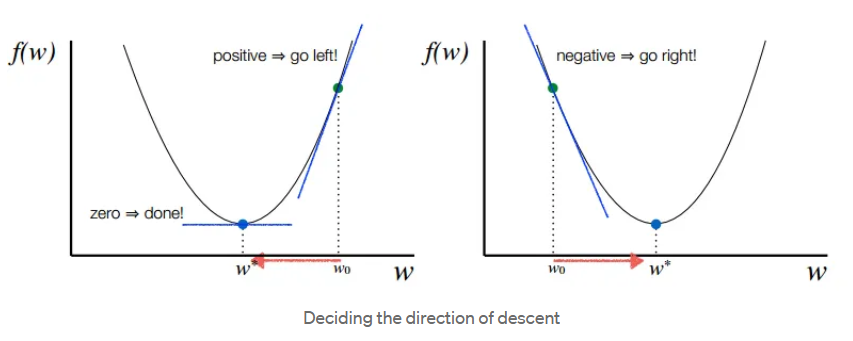
***TYPES OF OPTIMIZERS :***

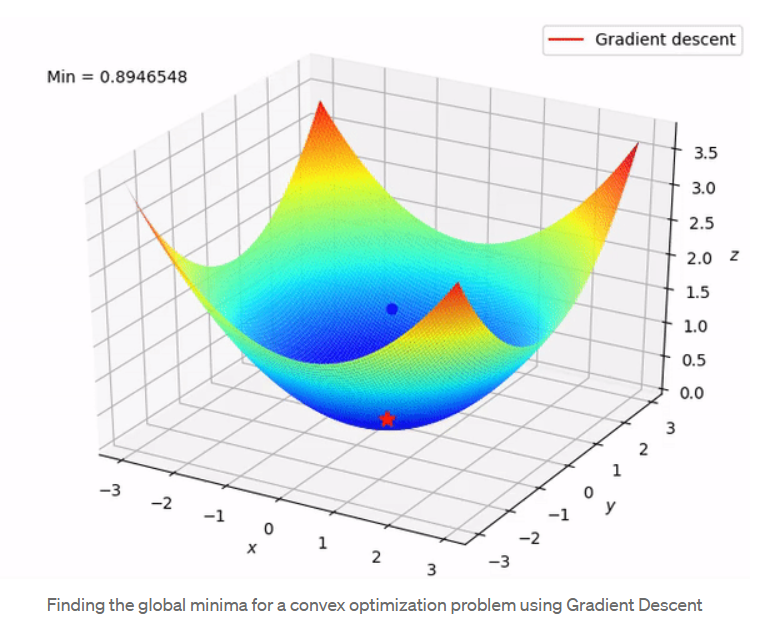
1. Gradient Descent
2. Stochastic Gradient Descent
3. Adagrad
4. Adadelta
5. RMSprop
6. Adam

G**radient Descent :**

This is one of the oldest and the most common optimizer used in neural networks, best for the cases where the data is arranged in a way that it possesses a convex optimization problem. It will try to find the least cost function value by updating the weights of your learning algorithm and will come up with the best-suited parameter values corresponding to the Global Minima.

This is done by moving down the hill with a negative slope, increasing the older weight, and positive slope reducing the older weight.



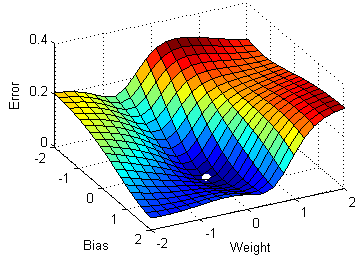


S***tochastic Gradient Descent :***

This is another variant of the Gradient Descent optimizer with an additional capability of working with the data with a non-convex optimization problem. The problem with such data is that the cost function results to rest at the local minima which are not suitable for your learning algorithm.

Rather than going for batch processing, this optimizer focuses on performing one update at a time. It is therefore usually much faster, also the cost function minimizes after each iteration (EPOCH). It performs frequent updates with a high variance that causes the objective function(cost function) to fluctuate heavily. Due to which it makes the gradient to jump to a potential Global Minima.

However, if we choose a learning rate that is too small, it may lead to very slow convergence, while a larger learning rate can make it difficult to converge and cause the cost function to fluctuate around the minimum or even to diverge away from the global minima.



Converging at the global minima using SGD for non-convex data

A***dagrad :***

This is the Adaptive Gradient optimization algorithm, where the learning rate plays an important role in determining the updated parameter values. Unlike Stochastic Gradient descent, this optimizer uses a different learning rate for each iteration(EPOCH) rather than using the same learning rate for determining all the parameters.

Thus it performs smaller updates(lower learning rates) for the weights corresponding to the high-frequency features and bigger updates(higher learning rates) for the weights corresponding to the low-frequency features, which in turn helps in better performance with higher accuracy. Adagrad is well-suited for dealing with sparse data.

So at each iteration, first the alpha at time t will be calculated and as the iterations increase the value of t increases, and thus alpha t will start increasing.

**3. Adaptive Optimizers**

* Adjust learning rates dynamically based on past gradients.
* **Adagrad (Adaptive Gradient Algorithm)**
  + Adapts learning rate per parameter.
  + Effective for sparse data but may slow down due to aggressive learning rate reduction.
* **RMSprop (Root Mean Square Propagation)**
  + Improves Adagrad by maintaining an exponentially weighted average of past squared gradients.
  + Used for non-stationary objectives (e.g., training RNNs).
* **Adam (Adaptive Moment Estimation)**
  + Combines momentum and RMSprop.
  + Uses first-moment (mean) and second-moment (variance) estimates.
  + Works well in most deep learning applications.
* **AdaDelta**
  + Improvement over Adagrad; prevents rapid learning rate decay.
  + Uses an adaptive learning rate without explicit learning rate tuning.
* **AdamW**
  + Similar to Adam but incorporates weight decay, improving generalization.
* **Nadam (Nesterov-accelerated Adaptive Moment Estimation)**
  + Combines NAG with Adam for faster convergence.