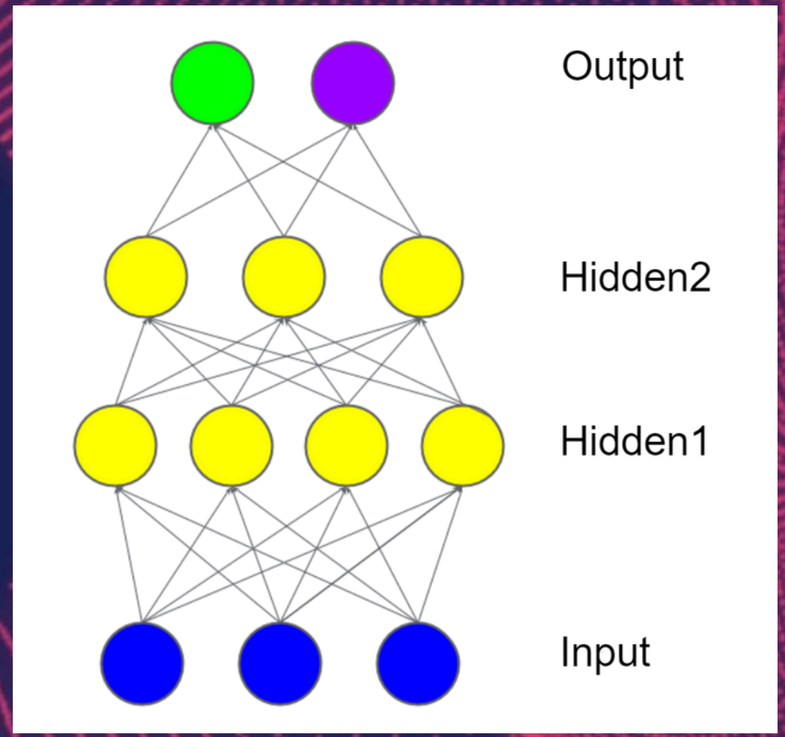
AI CLUB

Summer school session

OPTIMIZERS

NEURAL NETWORKS

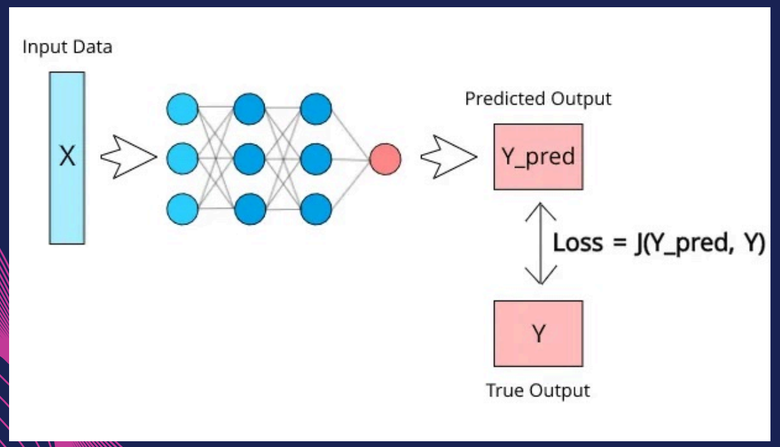
- Neural networks are a subset of machine learning and artificial intelligence inspired by the structure and function of the human brain.
- They are designed to recognize patterns and perform tasks such as classification, regression, and data processing.



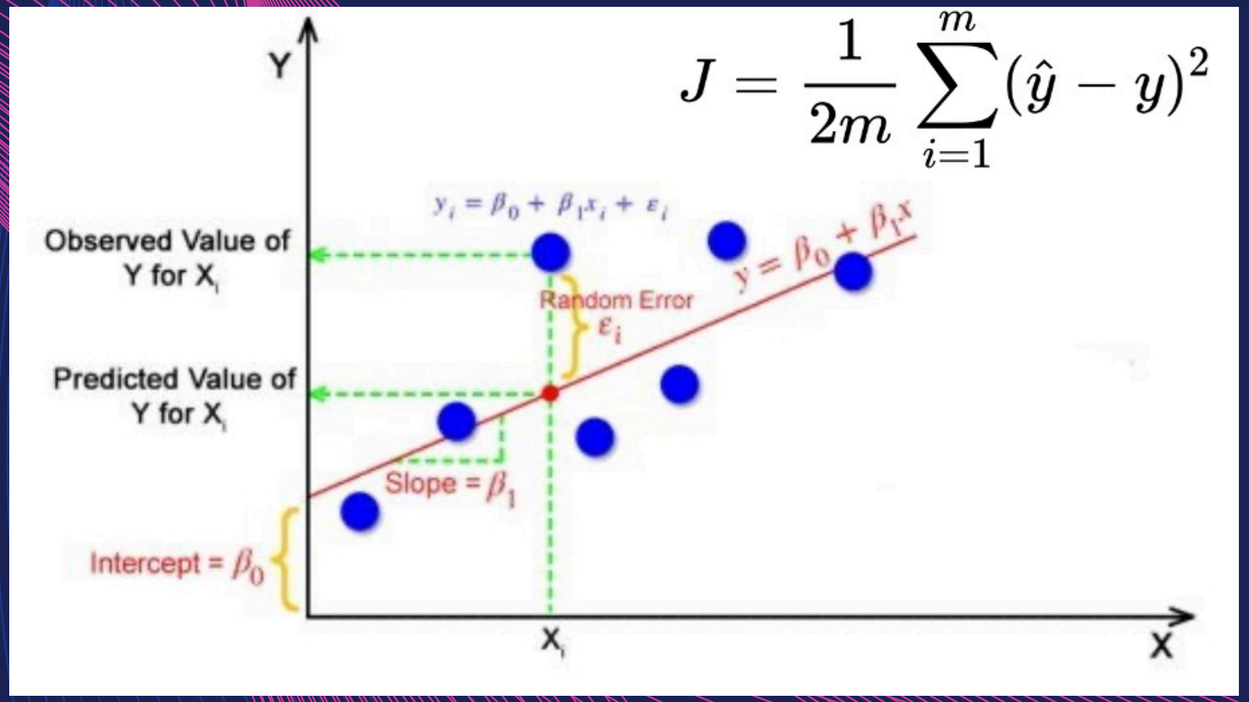
Structure of a neural network:

- Input Layer: Receives the input data. Each neuron in this layer represents a feature of the input data.
- Hidden Layers: Intermediate layers that process inputs from the previous layer. They apply transformations through weights and biases.
- Output Layer: Produces the final output. For classification, this layer often uses a softmax function to produce probabilities for each class.

WHAT IS A LOSS FUNCTION?



- Loss function (J) when provided with parameters such as the Y_pred and Y_true, would evaluate the deviation of the predicted Y value from the true value



- For a simple linear regression model, a loss function like J given here can be declared
- The function used here is mean squared error (MSE)

WHAT IS AN OPTIMIZER?

- Optimizers are algorithms or methods used to minimize an error function (loss function) or to maximize the efficiency of the model.

- 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.

- **How does the optimizer depend upon the model's learning parameters and optimize them at the same time?**

Okay so how are these loss function and optimizers related?

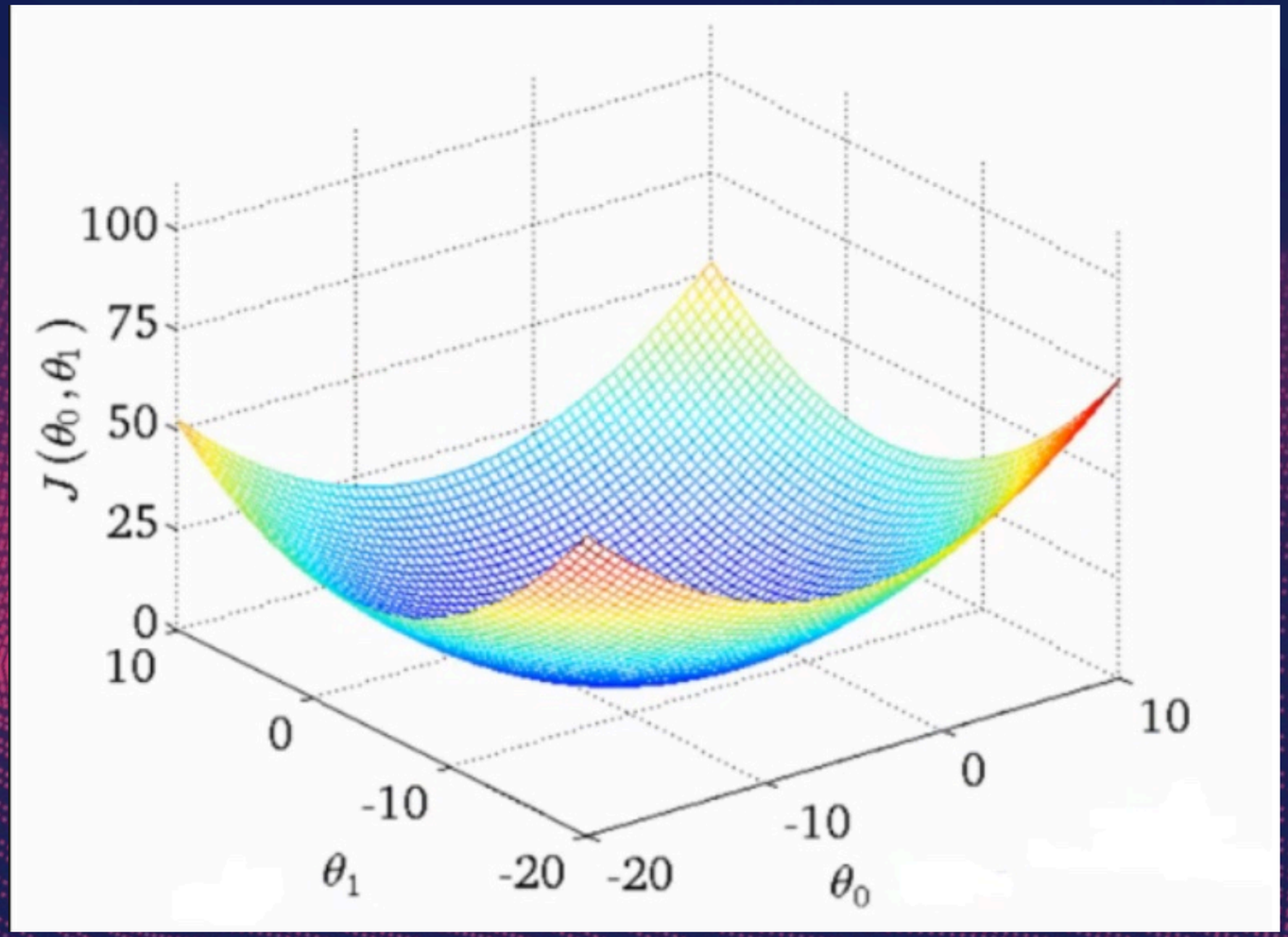
The main goal of deep learning is to reduce the loss or the error in a model, so how is it done?

It basically then comes down to loss function, since we need to minimize the loss as much as possible, it would be easier to graphically say that we need to determine the ***Global minima*** of the loss function

But, how do we determine it?

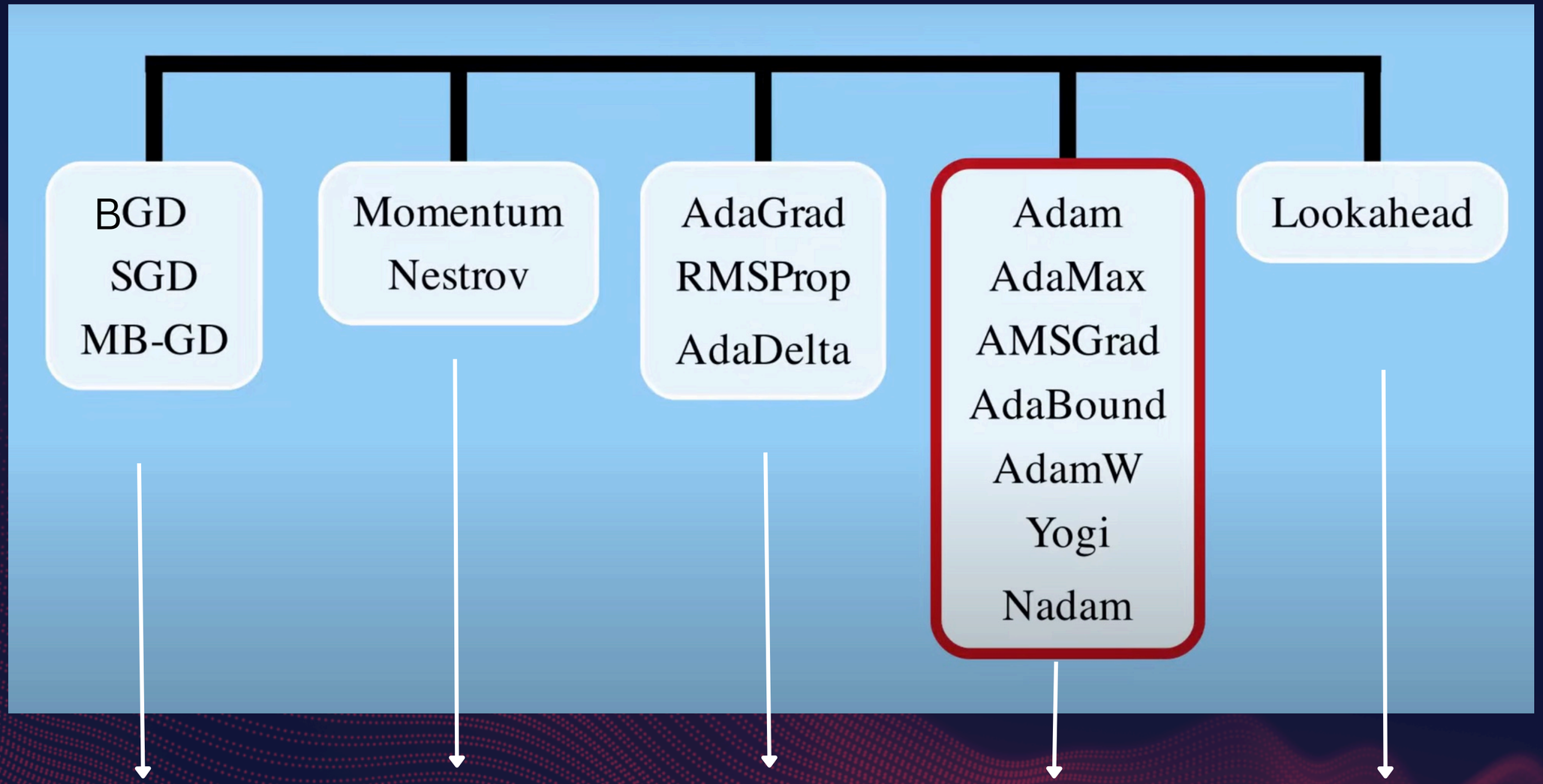
Now, thats where optimizers come in for the clutch

PLOT OF AN MSE LOSS FUNCTION



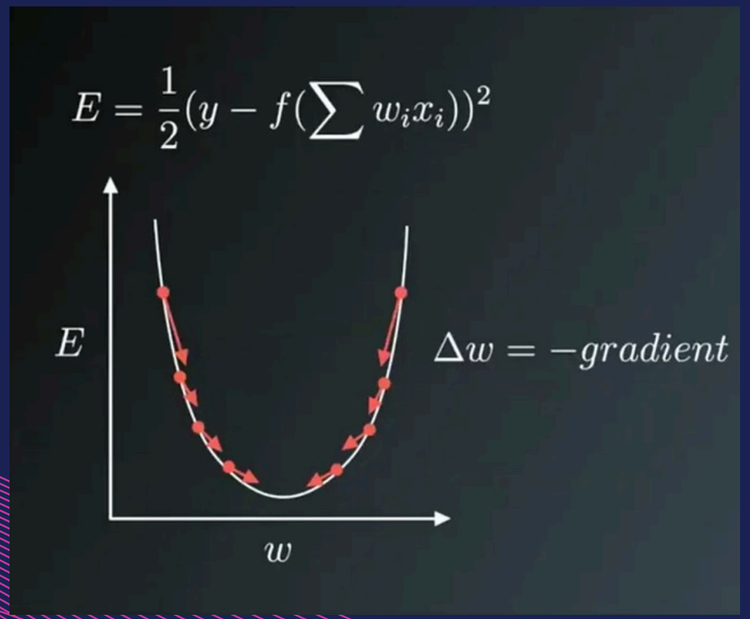
The optimize's job is to find the ideal values of θ_0 & θ_1 such that we attain the global minima of the given function

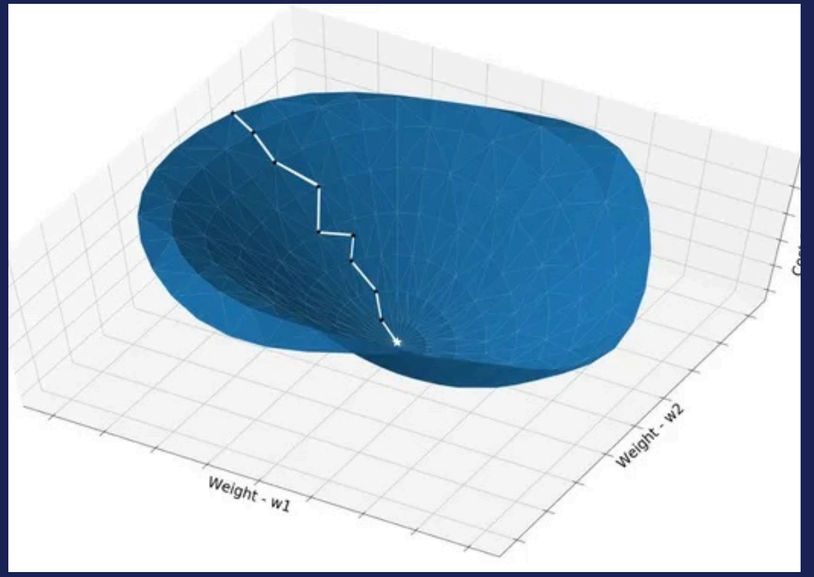
Types of Optimizers


BGD
SGD
MB-GD
Momentum
Nestrov
AdaGrad
RMSProp
AdaDelta
Adam
AdaMax
AMSGrad
AdaBound
AdamW
Yogi
Nadam
Lookahead
Gradient descent
Accelerated gradient descent
Adaptive learning rate methods
Adam family
Orthogonal to all the other optimizers

GRADIENT DESCENT

- Gradient Descent is the most basic but most used optimization algorithm. It’s used heavily in linear regression and classification algorithms.
- Movement in the direction of steepest descent till the global minima is found
- Similar to how hikers descend a hill



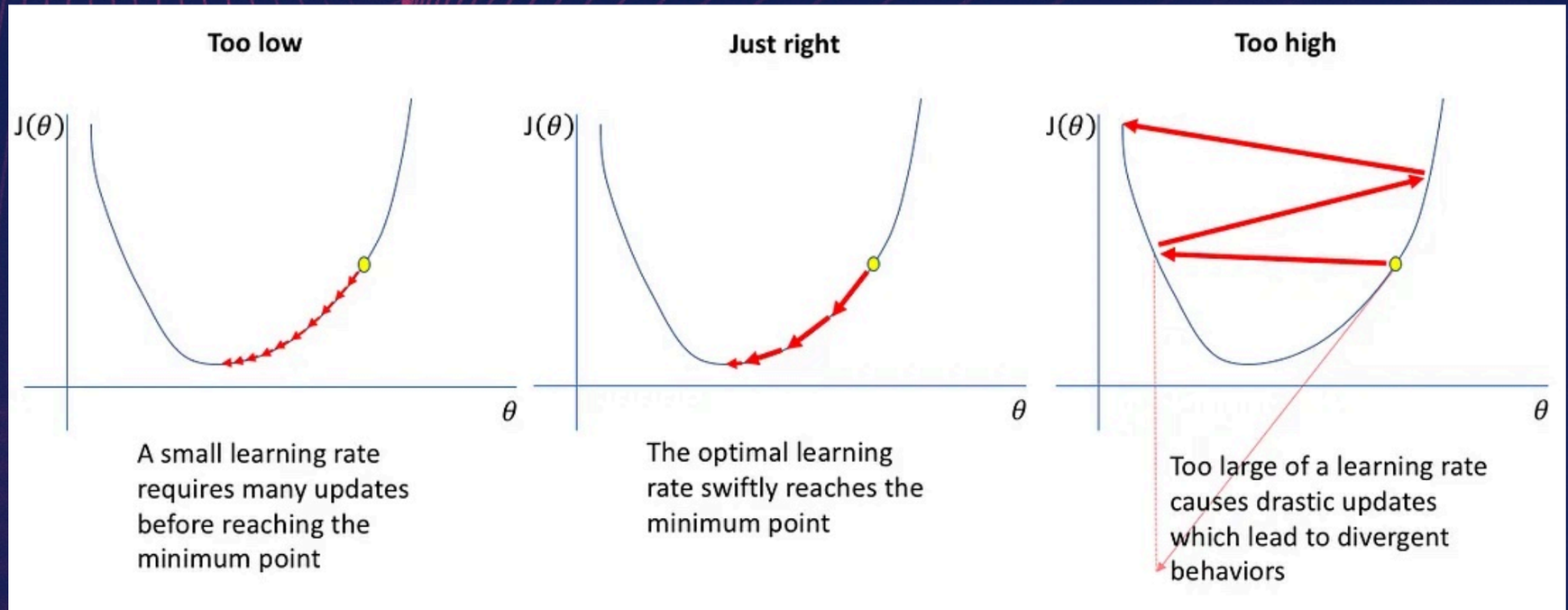


Learning rate :

Now that we know steps are gonna be taken in the direction of the steepest descent, how much should this step value be?

- **The steps we are taking in the direction opposite to the steepeset ascent is called as the learning rate**
- **Learning rate is a hyper-parameter, which means its specified by the user**

Now to answer our question on how much should this learning rate be;



Updating parameters

$$X = X - lr * \frac{d}{dX} f(X)$$

Where,

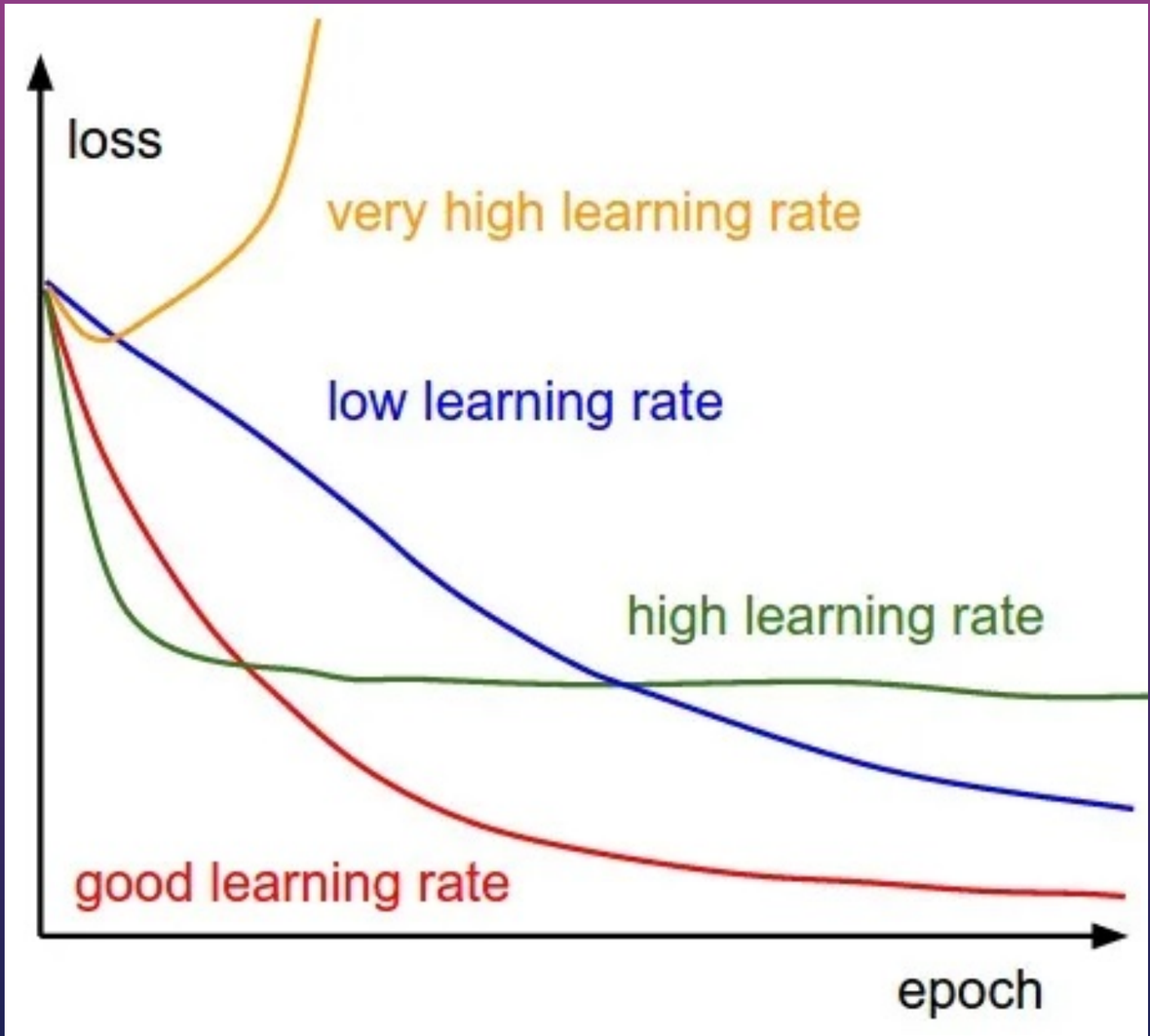
X = *input*

$F(X)$ = *output based on* X

lr = *learning rate*

- X can be any of the parameters i.e., weights or the biases.
- The d/dx term here represents the gradient of the function at that given point
- For the model to be optimized completely, all the parameters should be updated.

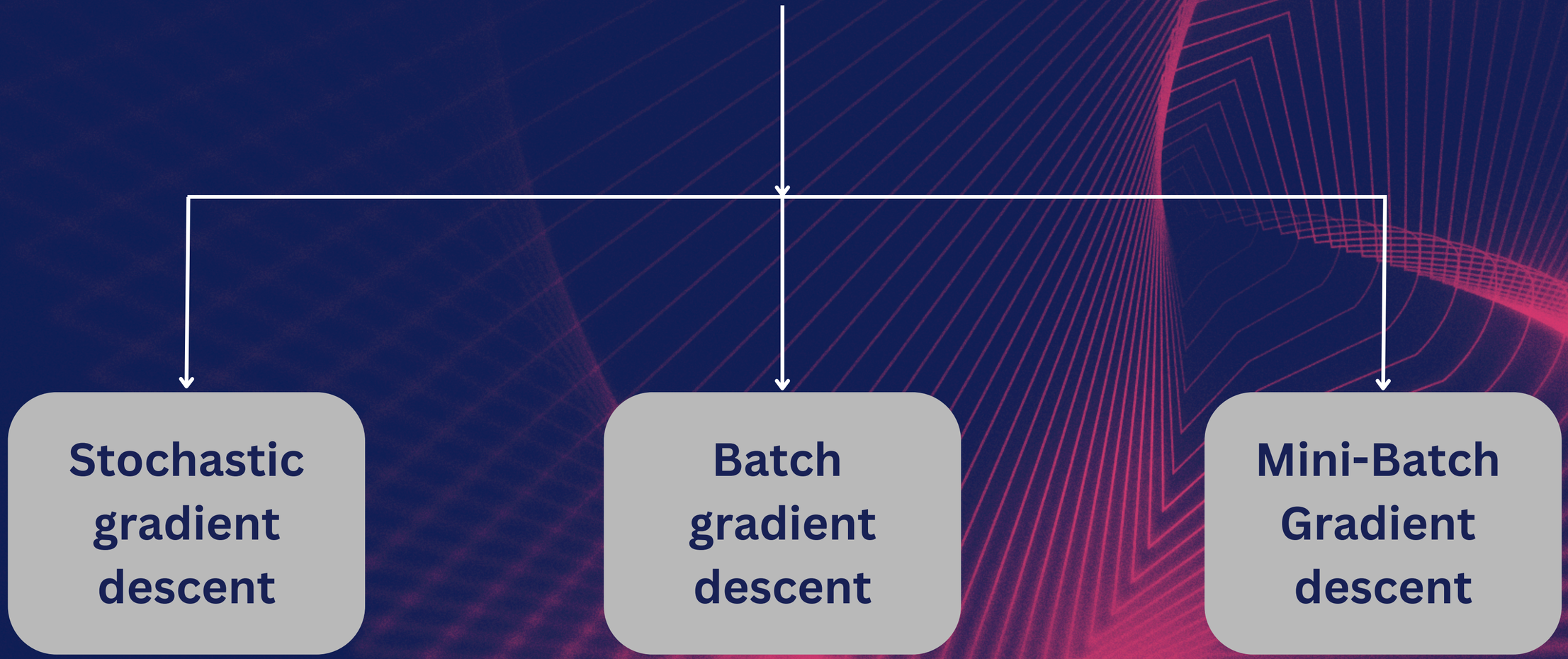
SLIDO



Epochs - It is the number of iterations a model has gone through to attain the global minima

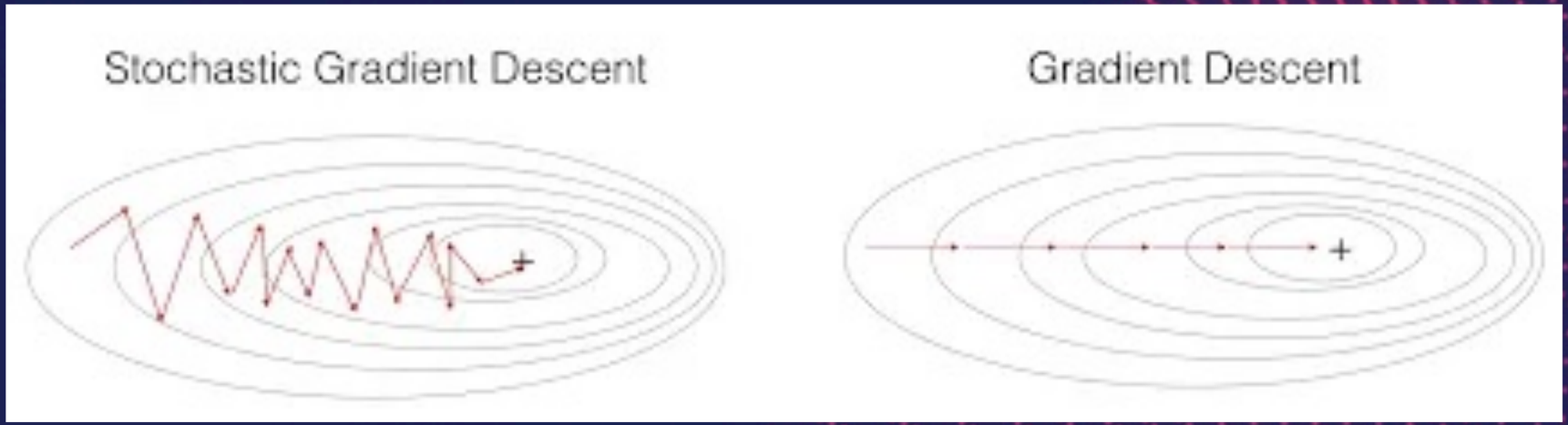
From this its clear that learning rate is a hyper parameter that should be fine tuned by the user so as to obtain an ideal value that maintains the balance between rate of learning and accuracy

Types of Gradient desccent



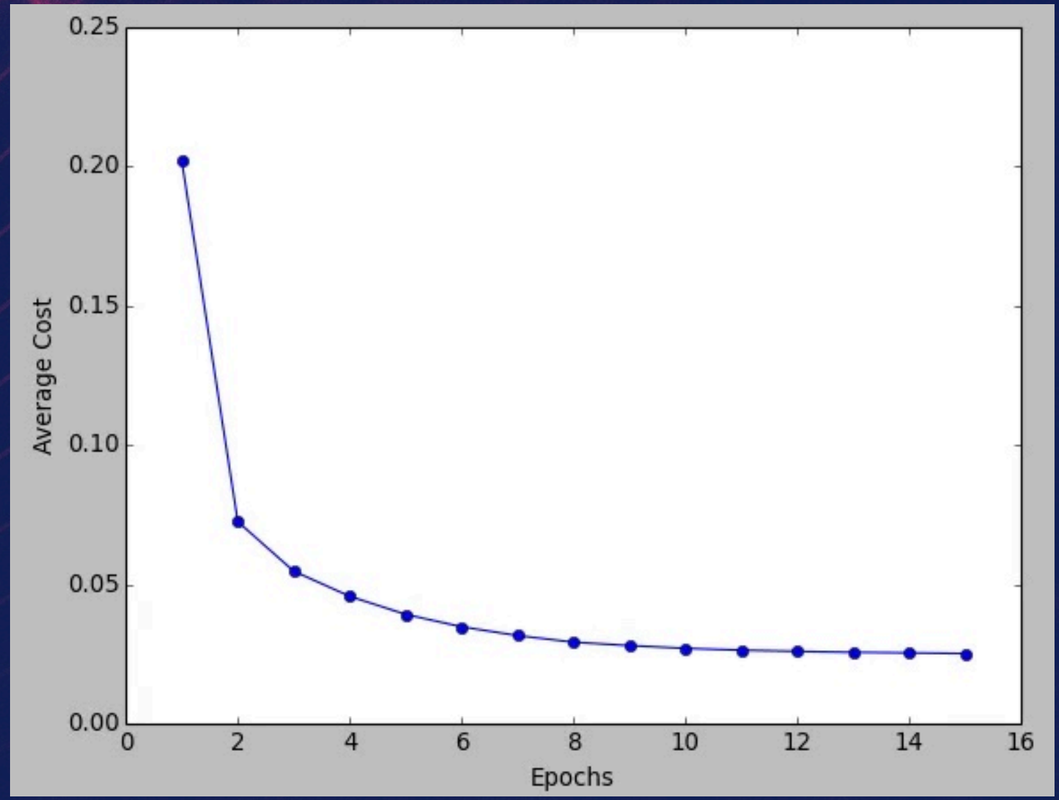
STOCHASTIC GRADIENT DESCENT

- The word ‘stochastic’ means a randomly defined process. In SGD, we randomly choose one point from the dataset, compute the gradient, then update the parameters and repeat until convergence.
- Basically we are trying to reduce the computation time by taking one datapoint from a million datapoints (perse) for a single epoch.
- This way the machine’s computation time would be reduced significantly



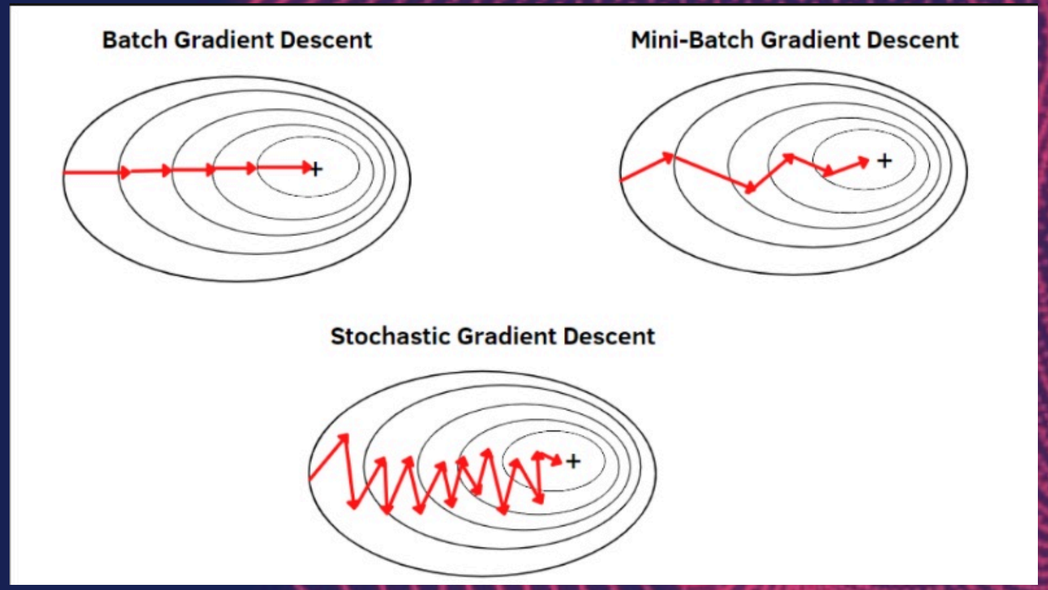
BATCH GRADIENT DESCENT

- In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters.
- Basically we are going to take the average of loss of a million datapoints and use that to upadate the parameters
- Here the effect of noise and outliers wont be much pronounced, which means the necessary amount of epochs are reduced over here.



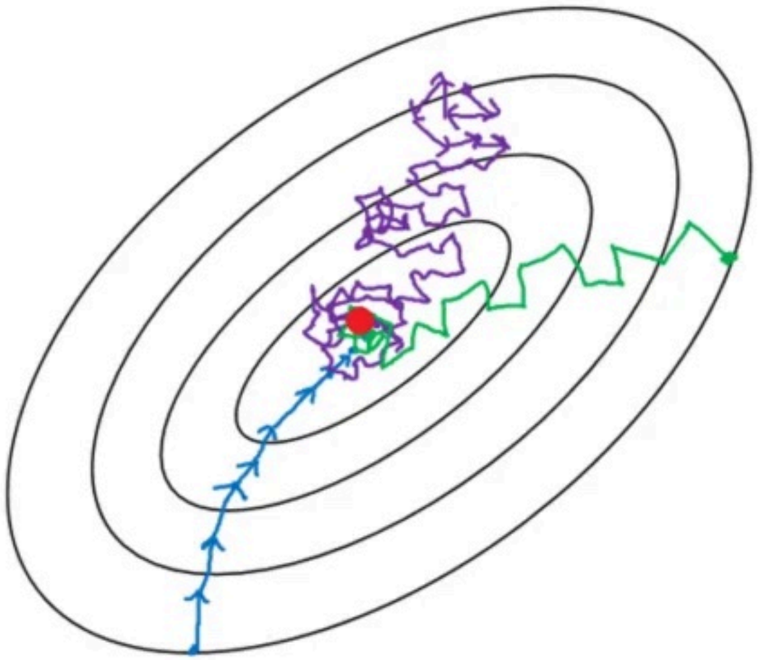
MINI - BATCH GRADIENT DESCENT

- Mini-batch GD = Batch GD + Stochastic GD
- A mini batch is a fixed subset of the original dataset.
- In mini-batch GD, the loss of mini-batches are averaged and the parameters are updated based on this for every epoch.



But WHY ?

- **SGD tends to be affected by noise and outliers due to the frequent updates since the variability in data is natural and the fact we are taking one point to represent a million other points**
- **BGD though accurate is pretty slow since it has to store all the values into its memory unlike SGD so as to evaluate the average of them**

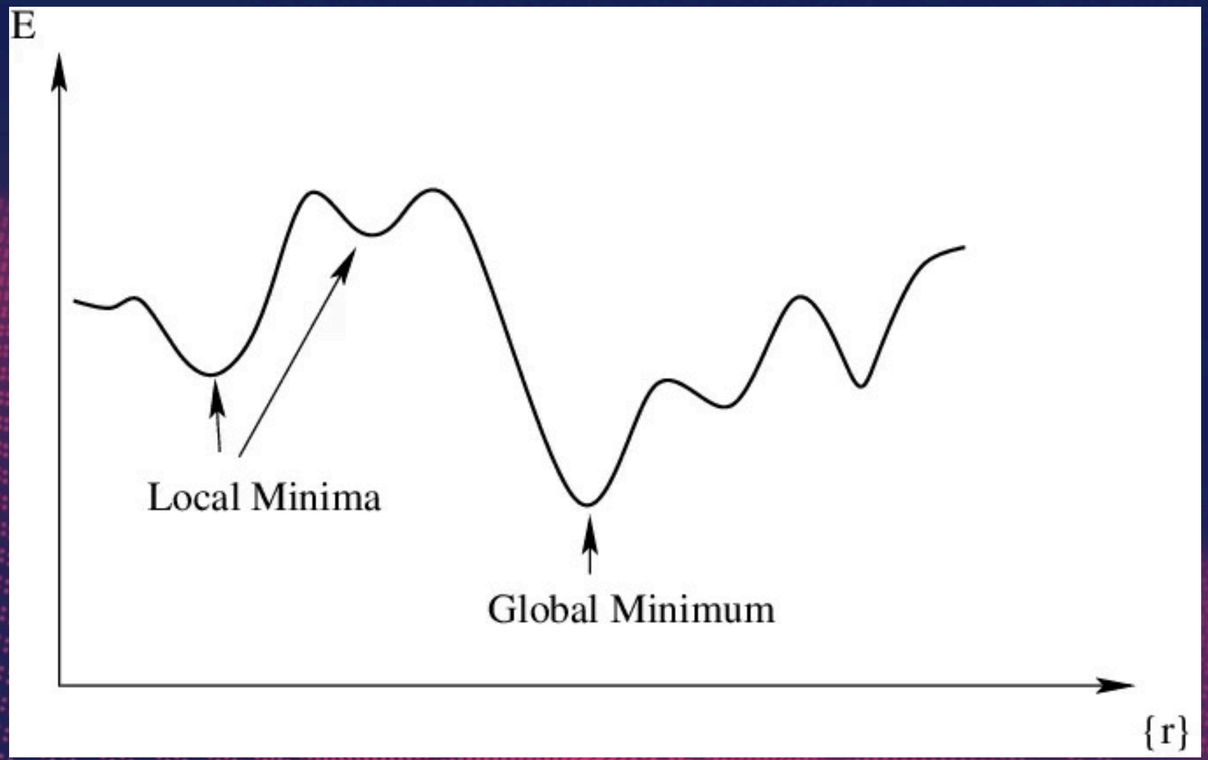


Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

PARAMETERS	BATCH GD ALGORITHM	MINI BATCH ALGORITHM	STOCHASTIC GD ALGORITHM
ACCURACY	HIGH	MODERATE	LOW
TIME CONSUMING	MORE	MODERATE	LESS

Pondering time

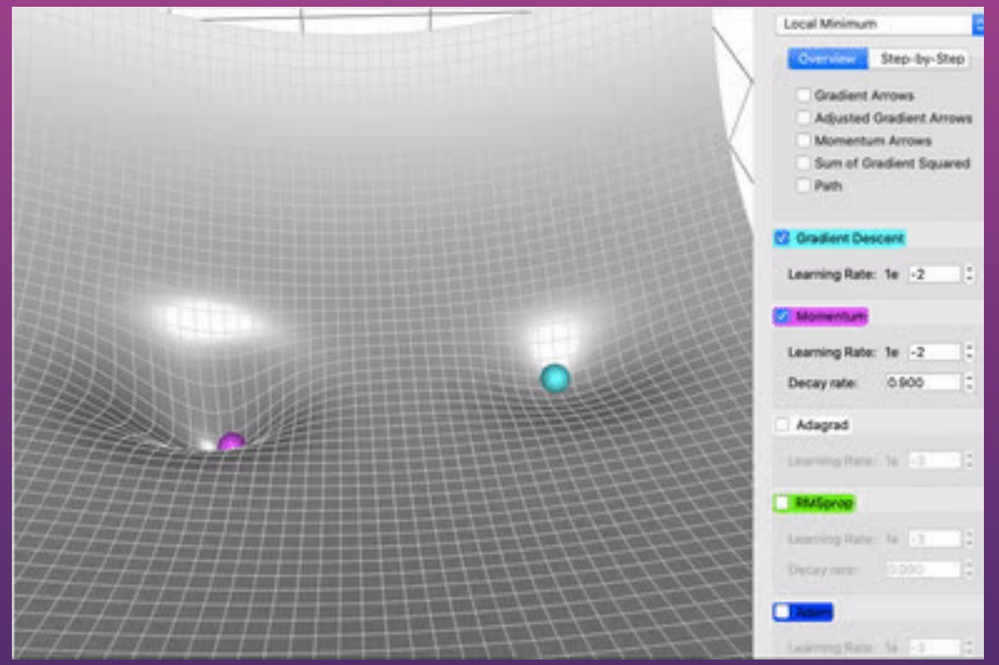
Will the convergence be affected in a gradient descent, if the starting points differ?



This is where momentum based optimizers come in for clutch

SGD WITH MOMENTUM

- As the name suggests, we are going to be introducing a momentum term to the SGD
- This is done so as to make sure the model has enough momentum to go right past all the local minima and attain the global minima



$$\theta_{t+1} = \theta_t - \alpha \mathbf{v}_{t+1}$$ ⟶ Parameter updation

Where,

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + (1-\beta)\nabla L(\theta_t, \mathbf{x}_t)$$ ⟶

- Vt can be thought of as the term that provides velocity
- β (momentum co efficient) decides how much of the velocity from the previous step should be retained.
- β should be between 0 and 1
- del(L) is the gradient as a function of parameters

There still is a good chance that this high speed from momentum might cause our model to overshoot the global minima

RMS PROP

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\mathbf{E}_{t+1}} + \epsilon}\mathbf{g}_t$$

→ Parameter updation

Where,

$$\mathbf{E}_{t+1} = \beta\mathbf{E}_t + (1-\beta)\mathbf{g}_t^2$$

- Again here β decides how much of the data should be remembered.
- More the value of β is, the slower we converge to minima
- ϵ is a constant which exists solely to prevent “division by zero” error

- It does not have a momentum term
- The learning rate is updated in each step by keeping a running average of the squared gradients
- The updation is such that when the gradient is high, Et+1 will be high, which in turn would decrease the learning rate
- This helps us because, the model is faster when it approaches local minimas (less gradients) thereby preventing us from getting stuck in the local minima

ADAM

- ADAM - Adaptive momentum estimation is the combination of both RMSprop and momentum optimizers
- We are going to introduce the momentum term to RMSprop
- The intuitive behind adam unlike RMSprop which aims to go fast enough to cross the local minima, is that we should slow down enough for determining whether or not are we approaching global minima

We are having two moments -

- **First moments is to make sure the model gains velocity (missing in RMSprop)**
- **Second moments is to make sure the model adapts the learning rate according to the gradient**

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

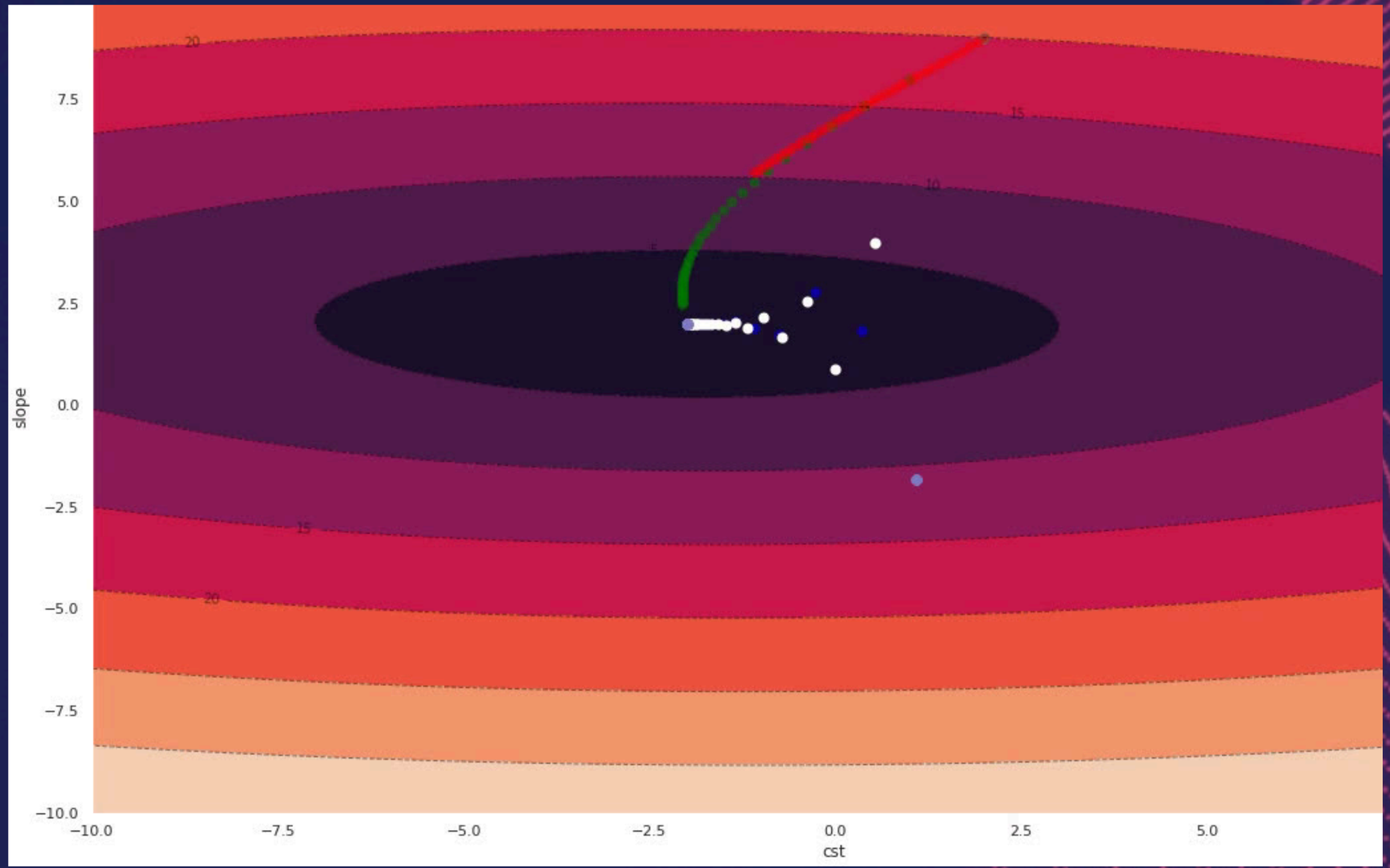
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

→ Bias correction

- First Moment m_t: Represents the exponentially decaying average of past gradients. It captures the trend or direction of gradients.
- Second Moment v_t: Represents the exponentially decaying average of past squared gradients. It captures the scale or magnitude of gradients.
- Bias Correction: Corrects for the bias introduced by initializing the moment estimates to zero, particularly during the early iterations of training.

SLIDO



CODE IMPLEMENTATION

Attendance QR