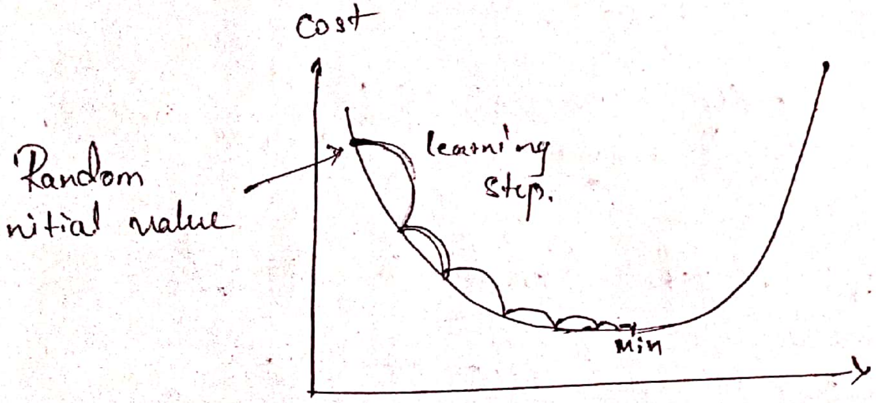
⑥

# Gradient Descent

→ A generic optimization algo to find optimal sol$^{ns}$ to wide range of pblms.

→ used to tweak parameters iteratively in order to minimize a cost function.



→ Model params are initialized randomly and get tweaked repeatedly to minimize the cost function. The steps gets smaller as the cost approaches the minimum.

→ when using Gradient descent, ensuring feature are all scaled (Standard Scaler), or else it will take longer to converge.

→ Training a model mean searching for a combination of model parameters that minimizes the cost function.

## How Gradient Descent works

Step 1 :- initializing the parameters of the neural network includes weight & biases

Step 2 :- perform a forward pass through the neural network. This involves passing the input data through each neural network applying transformations (activation) at each neuron until we reach the output layer. The output of the network is the predicted values for the input data.

Step 3 :- loss calculation :-
predicted output − actual output.
like MSE (mean squared error) for regression

Step 4 :- Backpropagation :- Compute the gradient of the loss function w.r.t each parameter of the neural network. This is done by chain rule of calculus which allows us to calculate how much each parameter contributed to error.

Step 5 :- Gradient update :- update the params of the neural network in the opposite direction of the gradients to minimige the loss function. Size of the update is determined by the learning rate.

## Different Types of Gradient Descent

### 1) Batch Gradient descent (BGD)

→ Entire training dataset is used to compute the gradient of the loss function

→ More accurate, takes time and computationally expensive.

### 2) Stochastic Gradient Descent (SGD)

→ only a single random data point from the training dataset is used to compute the gradient at each iteration.

→ After computing the gradient, a parameter update is performed.

→ Computationally efficient but it can have high variance in the parameter updates and may oscillate around the minimum.

→ Convergence is much faster ~~compared~~ ~~to~~ in SGD compared to GD

→ Time taken to complete the number of ephocs

GD → much faster
SGD → slower bcs it needs to update ephocs & n times.

→ Convergence is faster in SGD compared to GD. ~~em~~

→ SGD → Batch size = 1.

→ The spiky behavior helps to come out of the local minima and reach Global minima

### ③ Mini Batch Gradient descent

→ Compromise b/w BGD & SGD.

→ It divides the training dataset into small batches of fixed size (32 - 256)

→ the gradient is computed by averaging the gradients of the loss function computed on each batch.

→ number of batches = $\frac{n}{\text{batch size}}$

→ 4, 8, 32, 64, 128, 256 → ~~use~~ to use ram effectively. — binary

# Optimizers

Optimizers refers to the process of adjusting the parameters of a neural network to minimize the loss function.

## 1) Momentum Gradient.

Aims to accelerate GD in the relevant direction and dampen oscillations. It computes an exponentially weighted average of the past gradients and uses it to update the weights.

→ Helps to converge faster, especially in case of sparse gradients.

→ Dampens oscillations and helps escape local minima.

→ might overshoot the minimum due to momentum.

→ Time taking bcs of oscillations.

## 2) Nesterov Accelerated Gradient (NAG)

→ Nag is a modification of the momentum.

→ it computes the gradient of the objective function not at the current position but at an approximate future position

→ Converges faster than standard momentum.

→ Exhibits better behavior near the minimum compared to standard momentum.

→ requires additional computations.

## 3) AdaGrad :- Adaptive Gradient.

→ Adagrad adapts the learning rate for each parameter based on the historical gradients. It divides the learning rate by the squared root of the sum of historical squared gradients for each parameter.

→ works great even when the input data is on different scales. works well on sparse data.

→ mostly not used in complex neural network but used in linear regression problems.

→ Learning rate becomes too small over time hindering convergence. $\frac{\eta}{\sqrt{V_t}}$, $V_t$ becomes larger over the epochs.

→ Cannot reach to Global minima.

## 4) RMS Prop

→ Extension of AdaGrad that addresses its rapidly diminishing learning rates. it divides the learning rate by the exponentially decaying average of squared gradients

→ AdaGrad uses all the previous gradients where as RMS prop uses only recent epoch gradients.

→ No disadvantages, it is one of the best optimization techniques used in neural networks until Adam optimizer. Adam gives slightly better results but still the RMS prop competes with adam till today

## 5) Adam - Adaptive Moment Estimation

→ Adam combines the advantages of both momentum & RMS prop. It computes adaptive learning rates for each parameter by considering both the first & second momentum of the gradients.

→ The most powerful optimizer as of today.

Varients = Adamax

adamax replaces the $l_2$ norm with $l_\infty$ norm.
→ more stable than Adam.

For momentum -

tf. Keras.optimizers. SGD(lr = 0.001, momentum= 0.9)

NAG — tf.keras.opti ———— (lr; ————, nesterov= True)

RMS prop - ———————. RMSprop(lr, rho = 0.9)

Adam = t ————, Adam( lr, beta_1 = 0.9, beta_2 = 0.999)

Nadam → Combines adam plus Nesterov trick.
Converge slightly faster than Adam.

AdamW → incorporates weight decay directly into the update step. weight decay is a form of regularization that penalizes large weights to prevent overfitting.