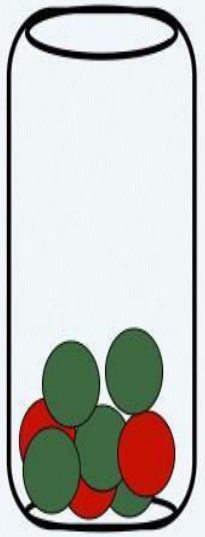
Probability Basics-Gradient based optimization

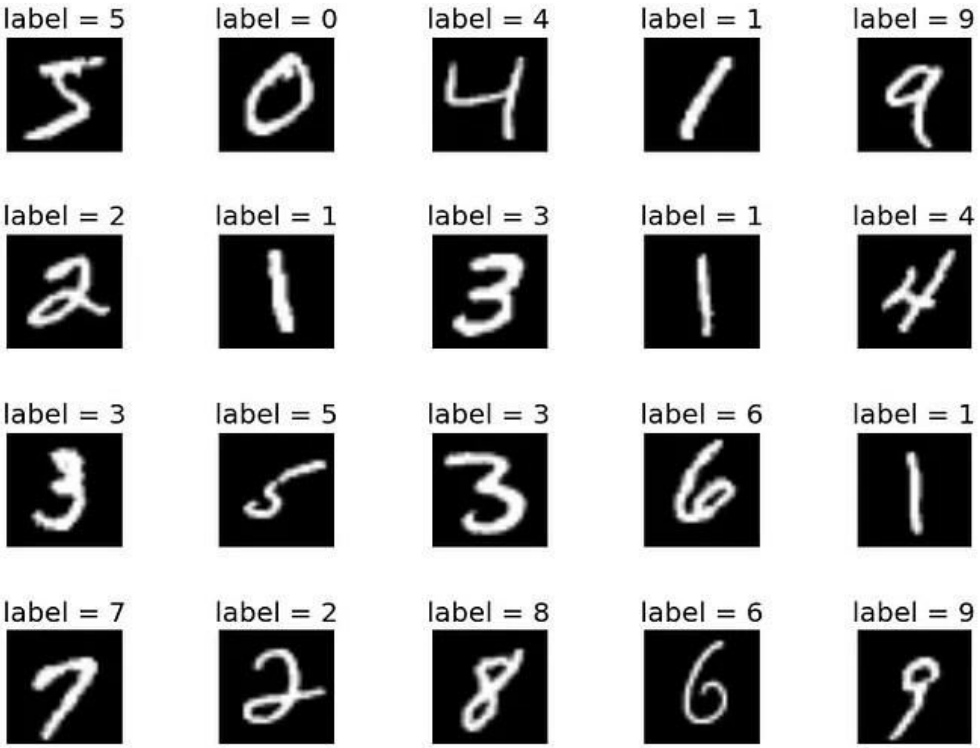
Probability

- Probability is the science of quantifying uncertain things.
- Most machine learning and deep learning systems utilize a lot of data to learn about patterns in the data.
- *Whenever data is utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.*

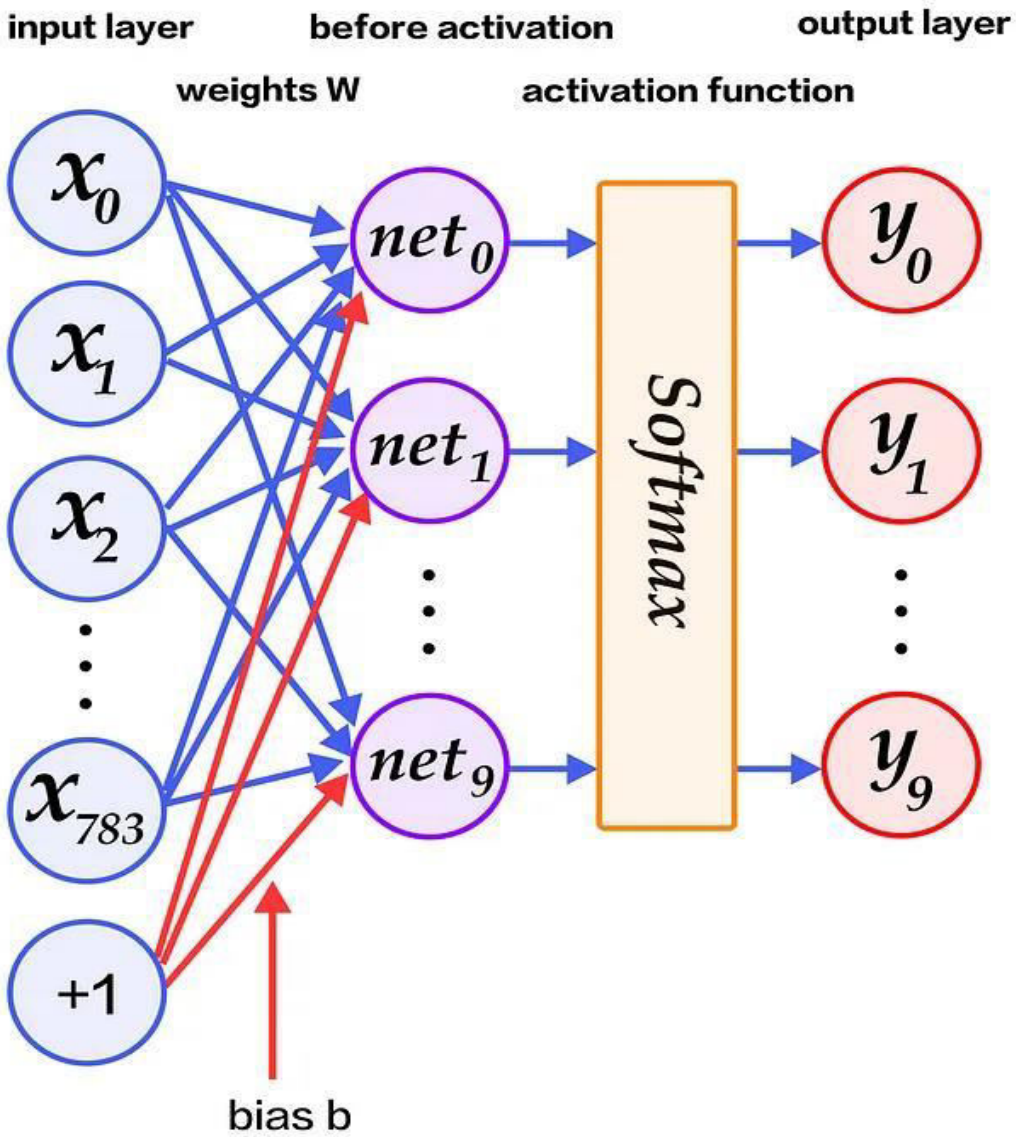


Example

MNIST handwritten digit recognition task



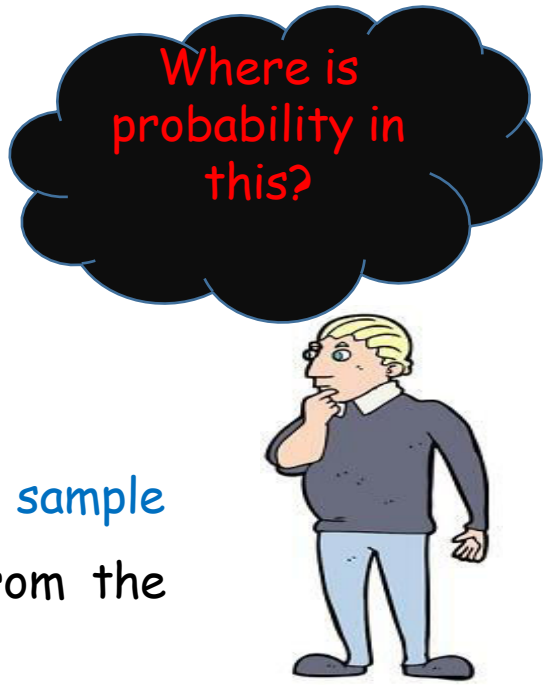
The images are 28*28 pixel images



Procedure

- The input layer is a flattened vector of the size of the input image(28*28=784).
- It is passed to a layer, where the input vector is multiplied by the weights and added with the bias vector. This layer has 10 neurons.
- This is the implication that there are 10 digits. Then they go through a softmax activation function.
- After this step they do not output the exact digit but a vector of length 10 with each element being a probability value for each digit.
- We use argmax to get the index of the probability with the highest value in the output vector.(which is the prediction)

Where probability?



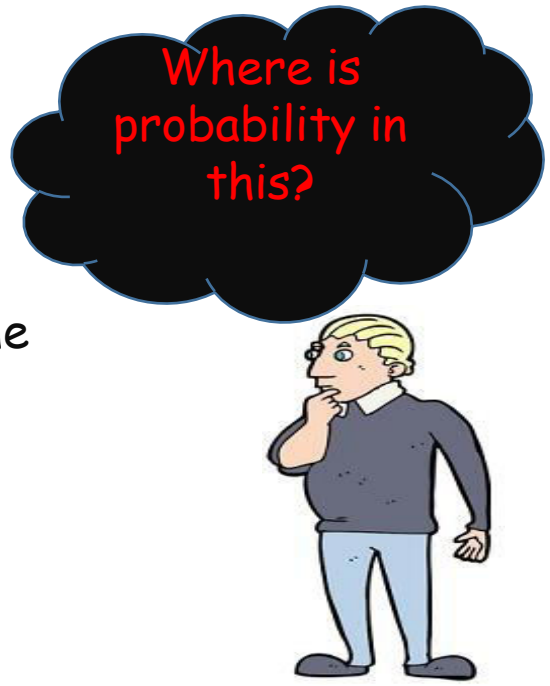
- vector ***y*** = [y0, y1, y2, y3, y4, y5, y6, y7, y8 , y9]

- Sample space: The set of all possible values in an experiment.
  - In the above example, the input can be from a set of images, thus it is the sample space for the input, similarly, the output prediction can take any value from the digits 0 to 9, thus the digits are the sample space for the output prediction.

Random variable: A variable that can take different values of the sample space randomly.

  - In the above neural network, the input vector ***x*** is a random variable, the output 'prediction' is a random variable, the weights of the neural network is also a random variable.

Where probability? Cont'd



vector ***y*** = [y0, y1, y2, y3, y4, y5, y6, y7, y8 , y9]

Probability distribution: Probability distribution is a description of how likely the random variable is to take on different values of the sample space.

- In the neural network, the weights are initialized from a probability distribution.
- The output vector y follows softmax distribution which is also a probability distribution that shows the probability of X taking different digit values.
- In this example, the probability distribution y is discrete(having 10 discrete values)
- Note : Some other cases, it may be continuous(the sample space is also continuous)

Where probability? Cont'd

- *If you see some instance of the output vector*

$$y = [0.03,0.5,0.07,0.04,0.06,0.05,0.05,0.06,0.04,0.1]$$

- If you look closely they all add up to 1.0 and the argmax shows that index 1 has a maximum value of 0.5 indicating the value should be 1.
- This property of adding up to 1.0 is called normalization. Also, the values must be between 0 and 1. An impossible event is denoted by 0 and a sure event is denoted by 1.

3 Basic Definitions of Probability

- **Joint Probability**: *What is the probability of two events occurring simultaneously .denoted by P(y=y,x=x) or p(y and x).* Example: probability of seeing sun and moon **at the same time** is very low.
- **Conditional probability**: *What is the probability of some event y happening, given that other event x had happened .denoted by P(y = y | x =x).* since the other event x had occurred, it's probability can't be zero.
- **Marginal probability**: *what is the probability of a subset of random variables from a superset of them*

Joint, Conditional and Marginal Probability

- Consider this example.
- Let us see how to do all?

	Male	Female	Total
Football	120	75	195
Rugby	100	25	125
Other	50	130	180
	270	230	500

Probability distribution

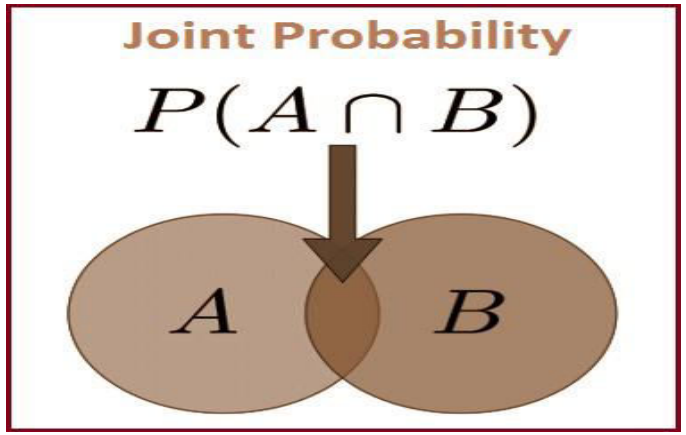
- No of observations / total gives the probability distribution

	Male	Female	Total
Football	0.24	0.15	0.39
Rugby	0.2	0.05	0.25
Other	0.1	0.26	0.36
	0.54	0.46	1

Joint Probability

- The Joint probability is a statistical measure that is used to calculate the probability of two events occurring together at the same time — P(A and B)

	Male	Female	Total
Football	0.24	0.15	0.39
Rugby	0.2	0.05	0.25
Other	0.1	0.26	0.36
	0.54	0.46	1



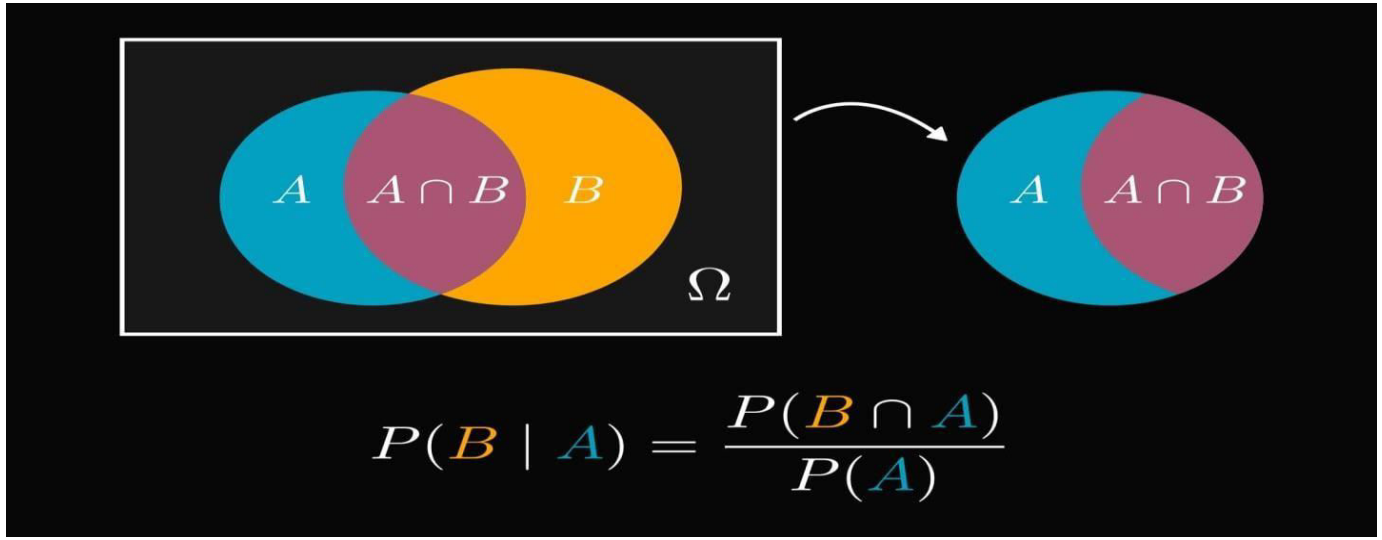
Marginal Probability

- Marginal distribution of a subset of a collection of random variables is the probability distribution of the variables contained in the subset.

	Male	Female	Total
Football	0.24	0.15	0.39
Rugby	0.2	0.05	0.25
Other	0.1	0.26	0.36
	0.54	0.46	1

Conditional Probability

- It defines the probability of one event occurring given that another event has occurred (by assumption, presumption, assertion or evidence).



- P(Rugby | Female) = P(Rugby ,Female) / P(Female)
- = 0.05 / ------ = -----

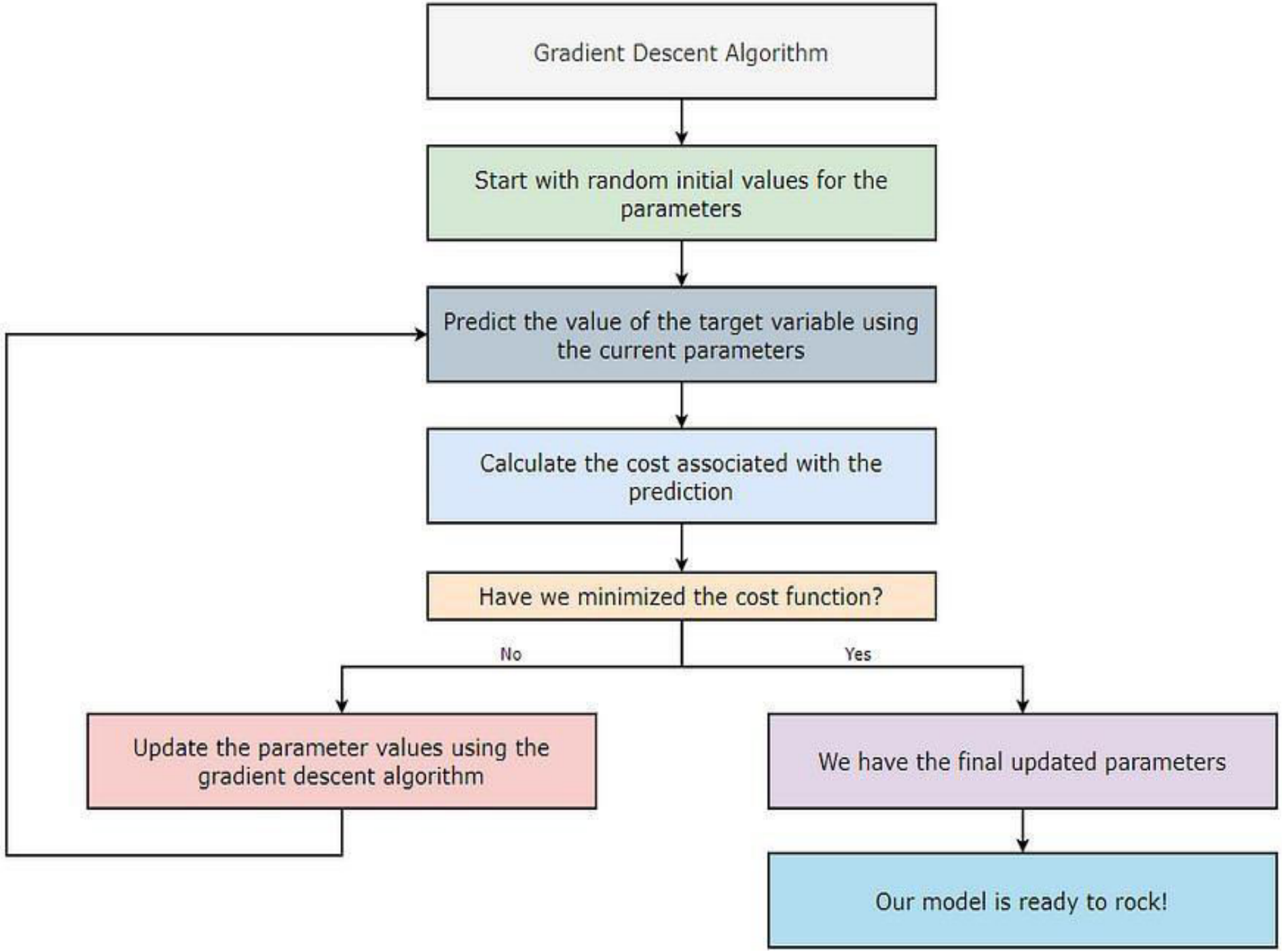
Gradient-Based Optimization

Gradient-Based Optimization

- Optimization refers to the task of either minimizing or maximizing some function f(x) by altering x.
- Optimization problems in terms of minimizing f (x) could be done.
- Maximization may be accomplished via a minimization algorithm by minimizing –f (x)
- The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function
- Value that minimizes or maximizes a function is denoted with a superscript
- for example, we might say $x^* = \arg\min f(x)$

Gradient Descent

- Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models.
- Gradient descent is also used to train Neural Networks.
- It minimizes errors between actual and expected results.


Gradient Descent Algorithm
Start with random initial values for the parameters
Predict the value of the target variable using the current parameters
Calculate the cost associated with the prediction
Have we minimized the cost function?
No
Yes
Update the parameter values using the gradient descent algorithm
We have the final updated parameters
Our model is ready to rock!

Gradient Descent Cont'd

The Formula of the Gradient Descent Algorithm:

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

Where,

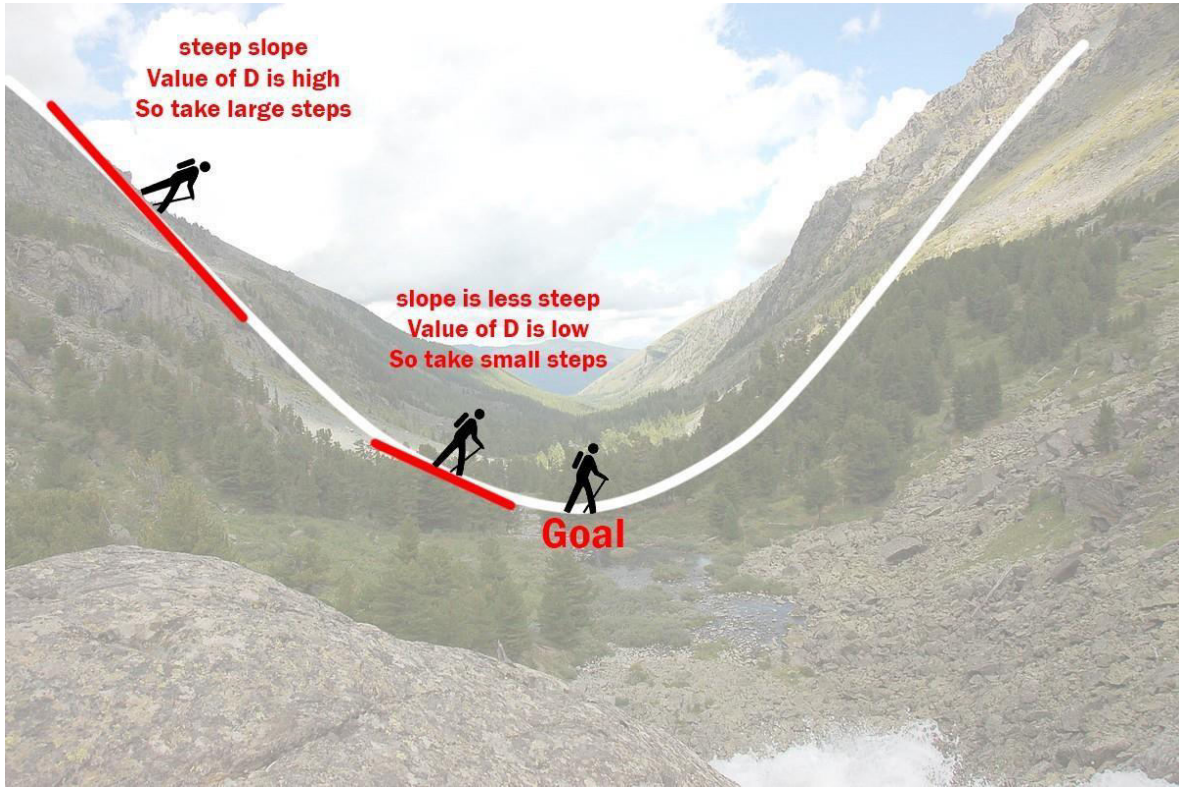
$X = parameters\ to\ be\ optimized$

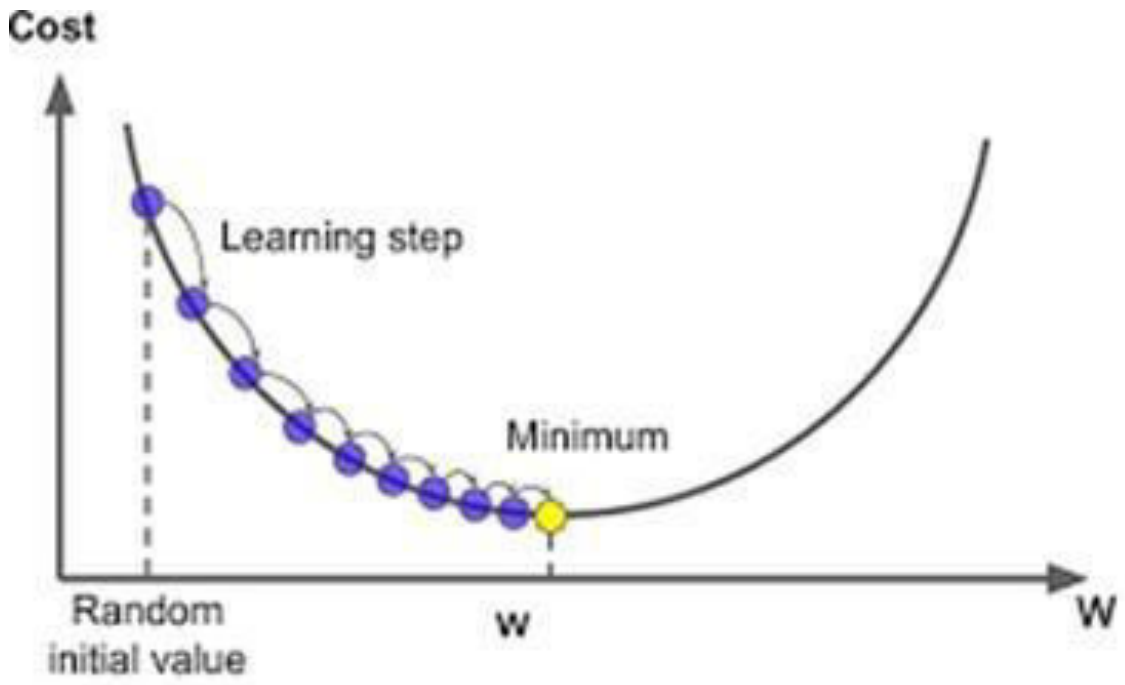
$f(X) = cost\ function$

$lr = learning\ rate$

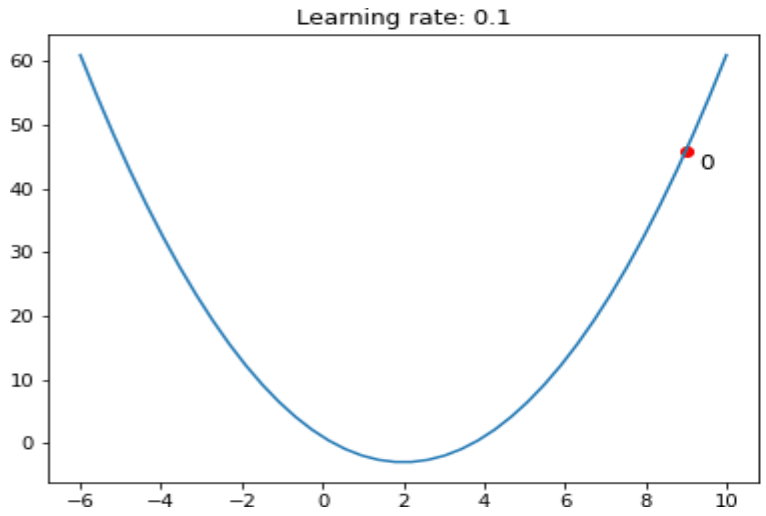
- *Gradients are nothing but a vector whose entries are partial derivatives of a function.*

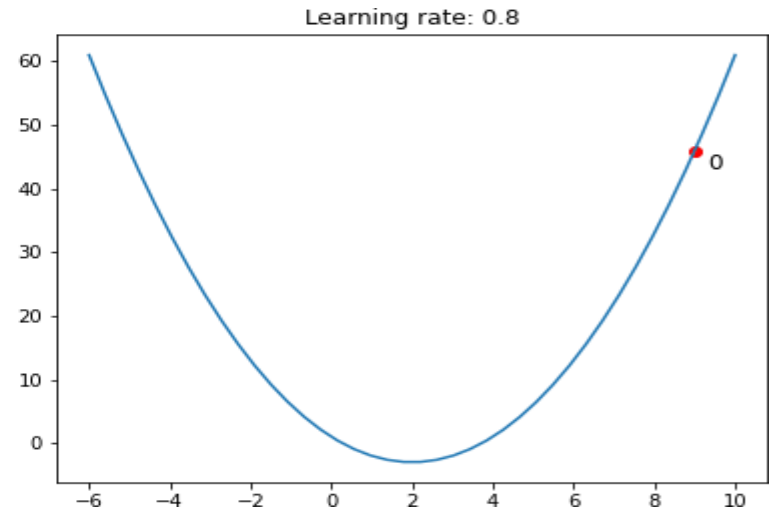
Understanding Gradient Descent

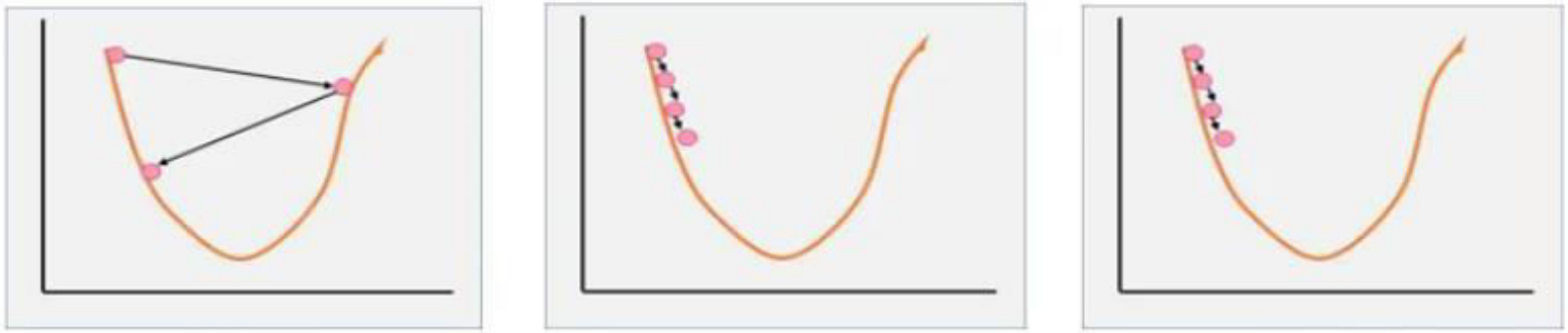




Learning rate Difference







(a) Large learning rate, (b) Small learning rate, (c) Optimum learning rate

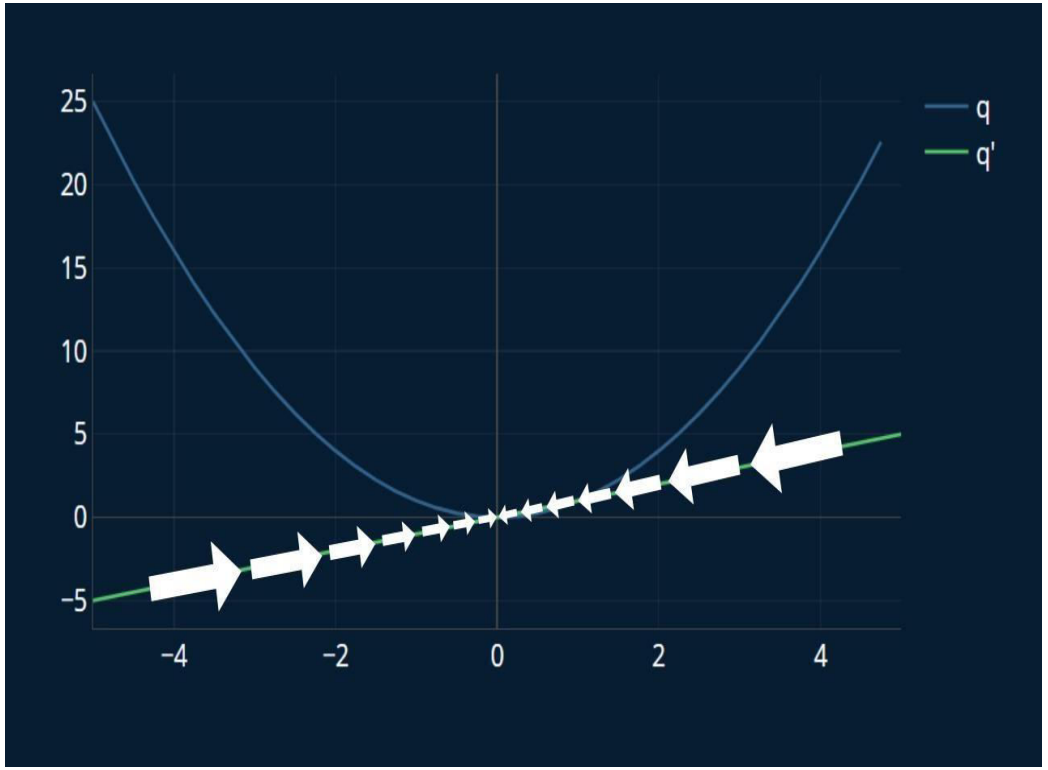
Learning Rate Selection

1. A **large learning rate** (a) causes the optimizer to overshoot the minima, leading to instability or failure to converge.
2. A **small learning rate** (b) makes the training slow, requiring many iterations to reach the minima.
3. An **optimal learning rate** (c) allows the model to converge efficiently to the minima.

Learning rate Difference

So the important points to remember are

- **Positive derivative** → Reduce the function value.
- **Negative derivative** → Increase the function value.
- **High absolute derivative** → Take a large step
- **Low absolute derivative** → Take a small step.



A **derivative** measures how a function changes as its input changes. It's like asking, "How fast is the function's value increasing or decreasing at a particular point?"

Suppose you're driving, and the distance you travel depends on time, say $d(t) = t^2$, where t is time in hours and $d(t)$ is the distance in kilometers.

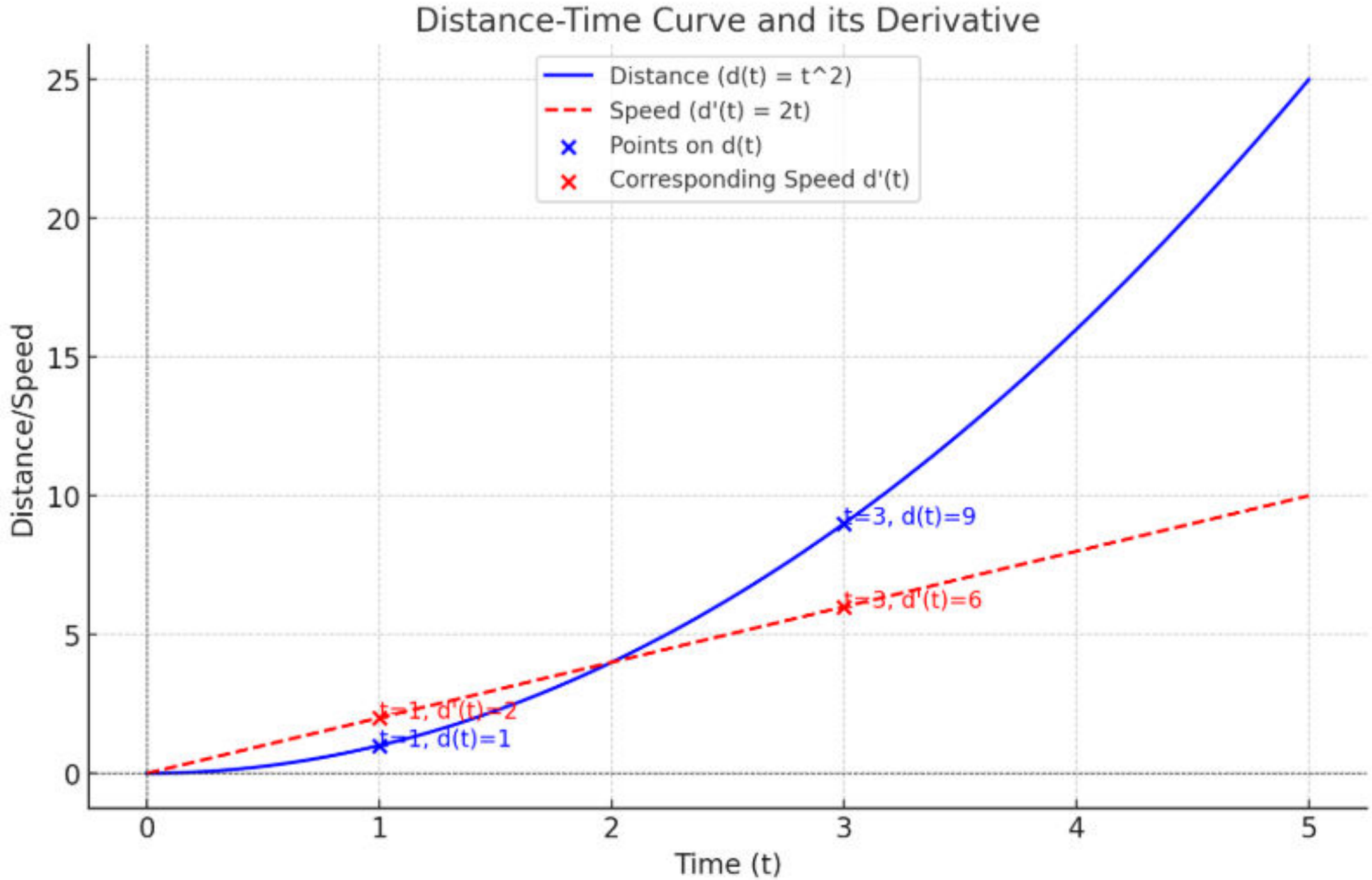
- The derivative of $d(t)$, written as $d'(t)$ or $\frac{d}{dt}[t^2]$, tells us the speed (rate of change of distance).
- For $d(t) = t^2$, the derivative is:

$$d'(t) = 2t$$

This means that at any given time t, your speed is $2t$ km/h.

1. **At $t = 1$ hour**, your speed is $2(1) = 2$ km/h.
2. **At $t = 3$ hours**, your speed is $2(3) = 6$ km/h.

The derivative $2t$ shows how your speed changes as time increases.


Distance-Time Curve and its Derivative
Distance (d(t) = t^2)
Speed (d'(t) = 2t)
Points on d(t)
Corresponding Speed d'(t)
t=3, d(t)=9
t=3, d'(t)=6
t=1, d'(t)=2
t=1, d(t)=1
Distance/Speed
Time (t)
0
5
10
15
20
25
0
1
2
3
4
5

Global Minimum

In the case of the linear regression model, there is only one minimum and it is the global minimum

Equation for linear regression (straight line): y=wx+b

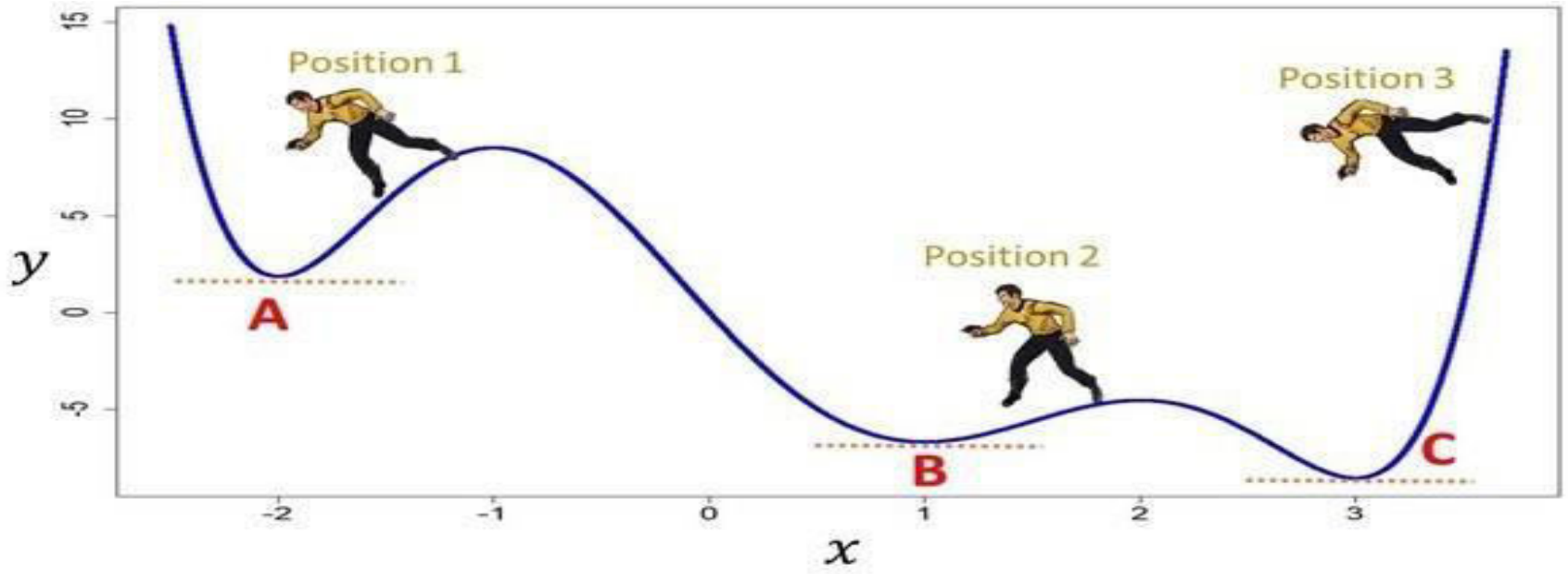
Equation for error(loss function): $$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (wx_i + b))^2$$

Squaring the error ensures we only get positive values.

Error curve: We adjust the slope (w) and intercept (b) to get a better fit. The error (MSE) forms a "U-shaped" curve. The bottom of this curve is where the error is smallest. This is the global minimum.

Global Minimum

In the case of the linear regression model, there is only one minimum and it is the global minimum



The local minimum reached depends on the initial coefficients taken into consideration. Here, point A, B are termed Local Minimum and point C is Global Minimum.

Different Types of Gradient Descent Algorithms

- Batch gradient descent: When the weight update is calculated based on all examples in the training dataset, it is called batch gradient descent.

- Stochastic gradient descent: When the weight update is calculated incrementally after each training example or a small group of training examples, it is called as stochastic gradient descent.

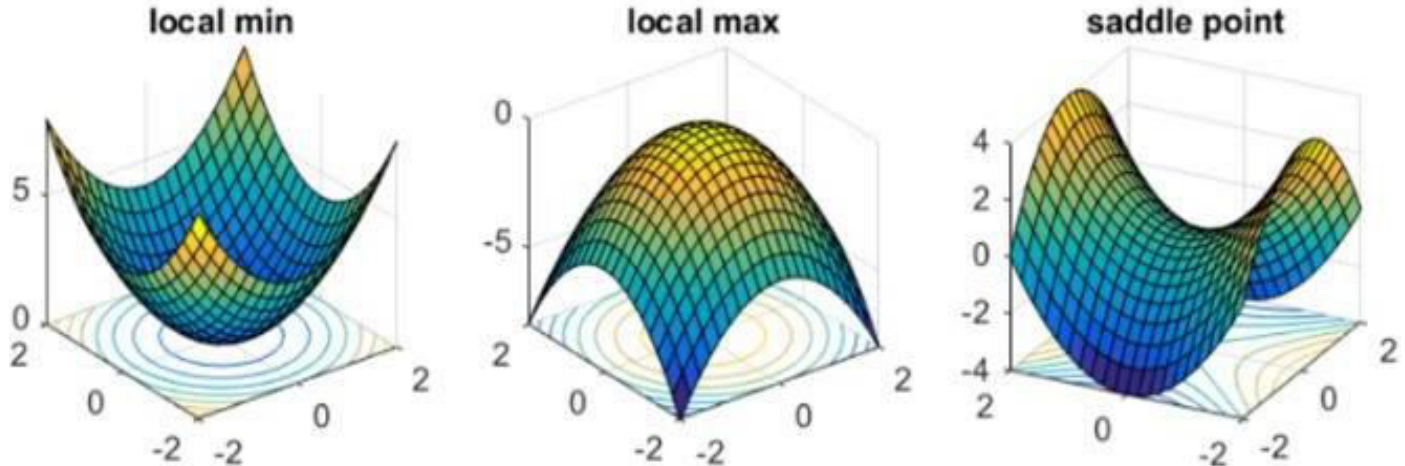
- Mini-batch gradient descent is a gradient descent modification that divides the training dataset into small batches that are used to compute model error and update model coefficients.

1. **Batch Gradient Descent**: Uses the entire dataset for each update. Slow but stable.
2. **Stochastic Gradient Descent**: Uses one training example for each update. Fast but noisy.
3. **Mini-batch Gradient Descent**: Uses a small batch of training examples for each update. A balance between speed and stability.

In practice, mini-batch gradient descent is the most commonly used approach due to its efficiency and ability to strike a balance between speed and convergence stability.

Issues that might occur

- When training a deep neural network with gradient descent and backpropagation, we calculate the partial derivatives by moving across the network from the final output layer to the initial layer.
- With the chain rule, layers that are deeper in the network go through continuous matrix multiplications to compute their derivatives.
- Due to this process, vanishing gradient, exploding gradient and saddle point occurs



Vanishing Gradient Problem

- **What it is**: In deep neural networks, as we move backward through the layers during backpropagation, the gradients (which are used to update the weights) can become very small, especially in the earlier layers.
- **Cause**: This issue usually arises when activation functions like **sigmoid** or **tanh** squash their inputs into small ranges, leading to gradients that are close to zero.
- **Impact**: If the gradients are too small, the weight updates for the earlier layers become insignificant, and these layers stop learning. As a result, the network struggles to capture features in the early layers.
- Solution: ReLU - The ReLU activation function is often used to mitigate this problem because it doesn't squash values into a small range (it outputs either the input or zero).

Exploding Gradient Problem

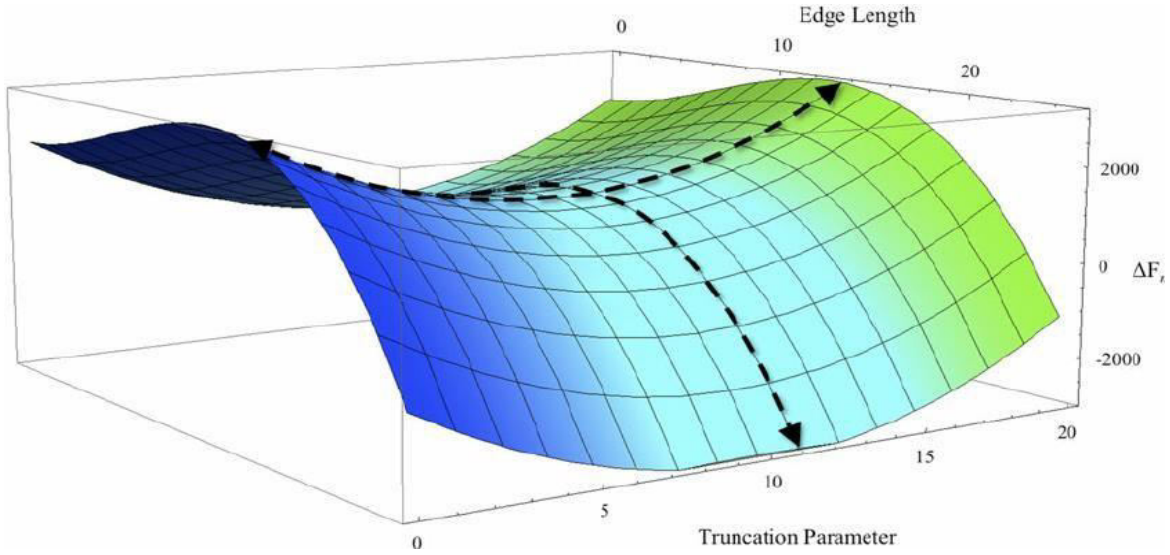
- **What it is:** Opposite to vanishing gradients, the exploding gradient problem occurs when gradients become excessively large during backpropagation, making weight updates too large.
- **Cause:** This can happen when the weights in a network are initialized with very large values, or the network is deep, causing the gradients to grow exponentially as they propagate backward.
- **Impact:** Exploding gradients cause the model to diverge, as the updates become too large and the model weights become unstable, leading to a loss in performance.
- Solution: Gradient clipping - One common method is gradient clipping, where gradients are capped at a certain value to prevent them from becoming too large.

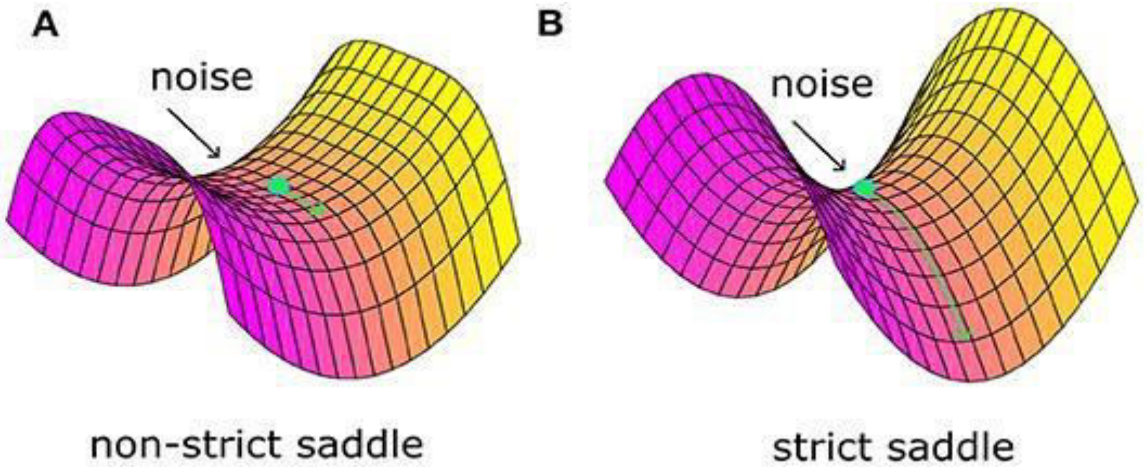
Saddle Point Problem

- **What it is:** A saddle point is a point in the loss function where the gradient is zero, but it is not a minimum or maximum. Instead, it's a point where the surface curves up in some directions and down in others.
- **Cause:** In high-dimensional spaces (which deep networks have), the loss function might have many saddle points that are not optimal, making it difficult for gradient descent to escape them.
- **Impact:** The model may get stuck in these saddle points during optimization because the gradient at these points is zero, making it hard for the optimizer to proceed. While the gradient is zero, the loss is not necessarily at a minimum, so the network can fail to improve.
- Solution: SGD, Momentum, Adaptive learning rate (Adam, RMSProp)

Saddle point

- Saddle point injects confusion into the learning process.
- Learning of the model becomes slow.
- It means that the crucial point achieved is the maximum cost value.
- This saddle point gets the focus when the gradient descent works in multi-dimensions.





Solutions

- Changing the architecture
  - This solution could be used in both the exploding and vanishing gradient problems but requires a good understanding and outcomes of the change.
  - For example, if we reduce the number of layers in our network, Model complexity is reduced.
- Gradient Clipping for Exploding Gradients
  - Carefully monitoring and limiting the size of the gradients whilst our model trains is yet another solution. This requires some deep knowledge of how the changes could impact the overall performance.
- Careful Weight Initialization
  - A more careful initialization of the model parameters for our network is a partial solution since it does not solve the problem completely.

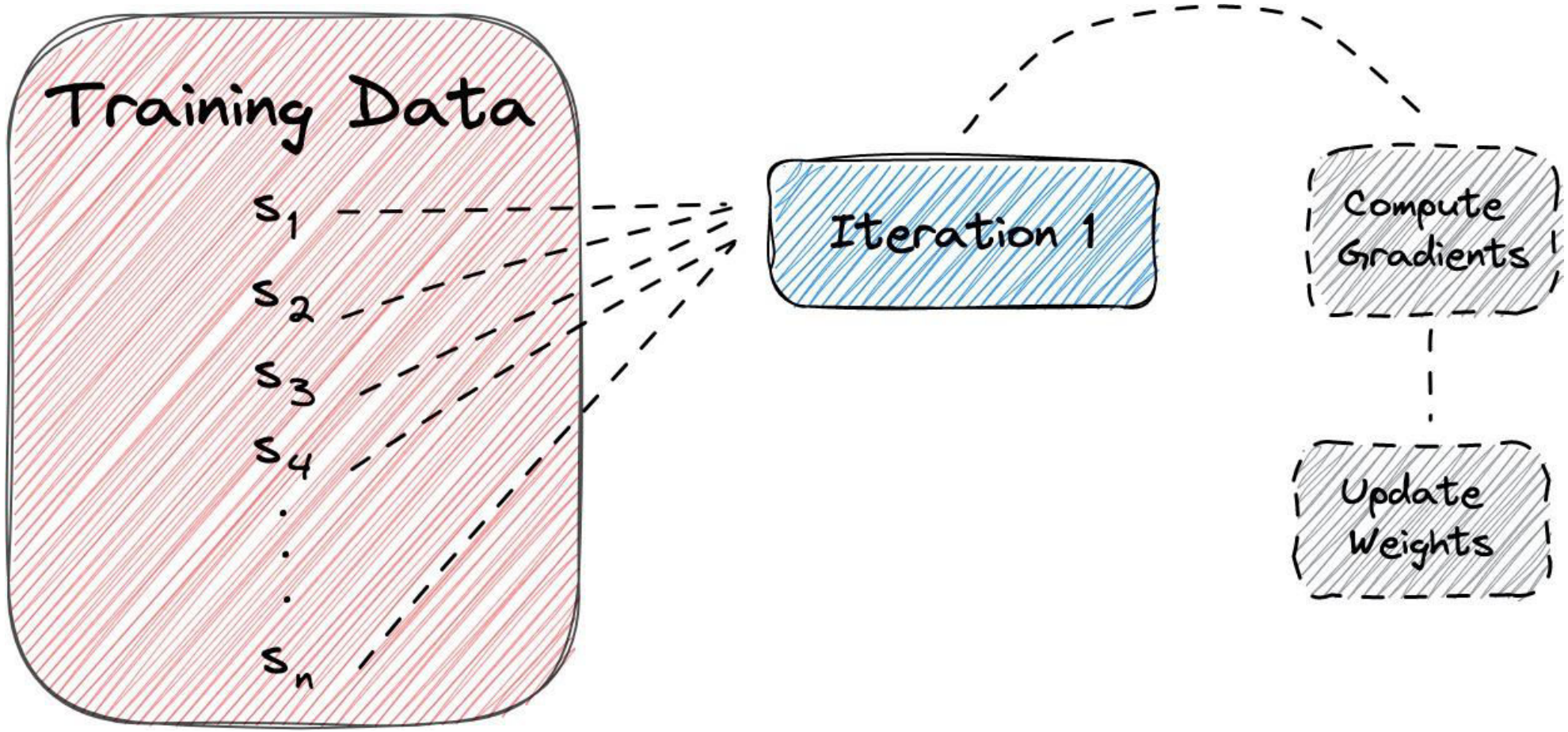
Limitations

- For a good generalization we should have a large training set, which comes with a huge computational cost.
- i.e., as the training set grows to billions of examples, the time taken to take a single gradient step becomes long.

Choosing Gradient Descent ?

1. **Vanishing Gradient**: Gradients shrink as they move backward, causing learning in earlier layers to stop. Mitigated by ReLU, proper initialization, and batch normalization.
2. **Exploding Gradient**: Gradients grow excessively, making updates too large and causing instability. Mitigated by gradient clipping, weight regularization, and proper initialization.
3. **Saddle Points**: Points where gradients are zero but are not minima, making optimization difficult. Mitigated by using SGD, momentum, and adaptive learning rates.

Batch Gradient Descent



Batch Gradient Descent Cont'd

- In batch gradient descent, we use all our training data in a single iteration of the algorithm.
- So, we first pass all the training data through the network and compute the gradient of the loss function for each sample. Then, we take the average of the gradients and update the parameters using the computed average.

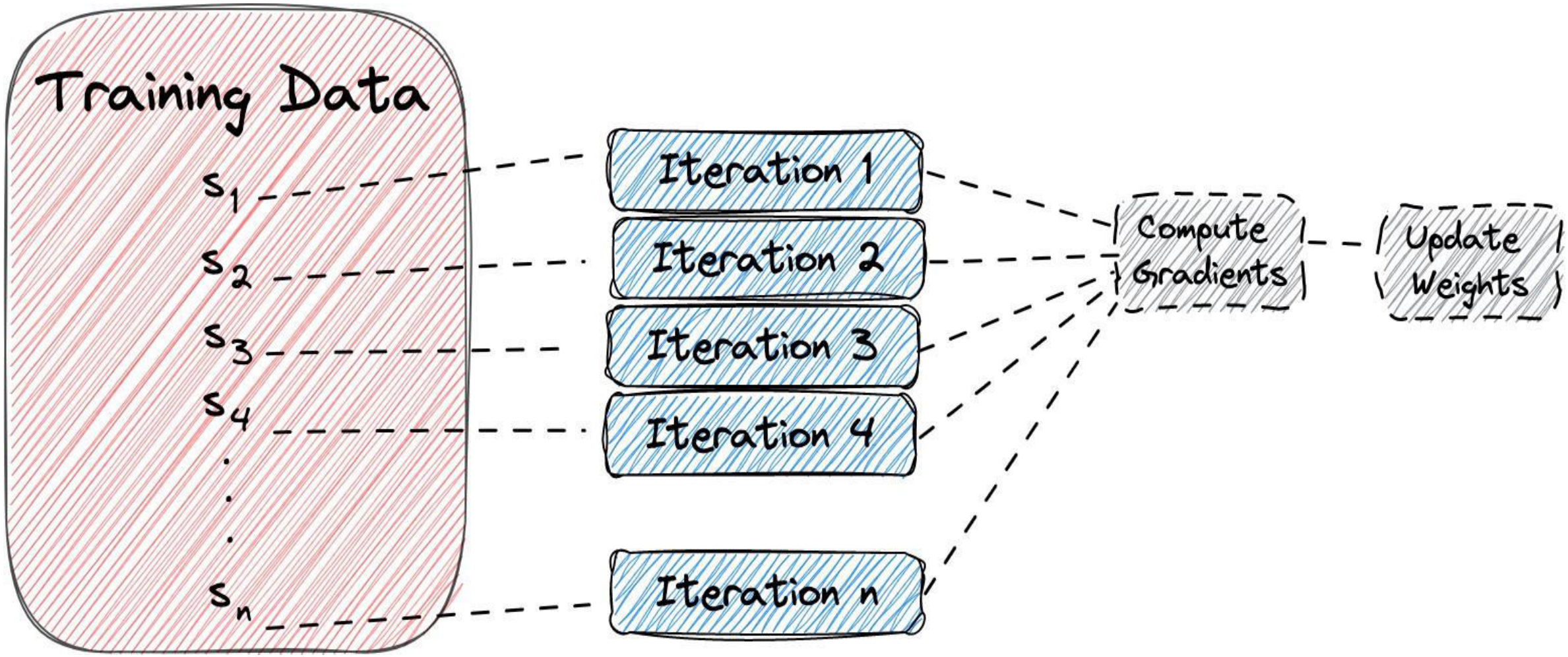
Stochastic Gradient Descent

- SGD is a variant of the optimization algorithm that saves us both time and computing space while still looking for the best optimal solution
- **Stochastic gradient descent** is a variant of gradient descent.
- The process simply takes one random stochastic gradient descent example, iterates, then improves before moving to the next random example.
- However, because it takes and iterates one example at a time, it tends to result in more noise than we would normally like.

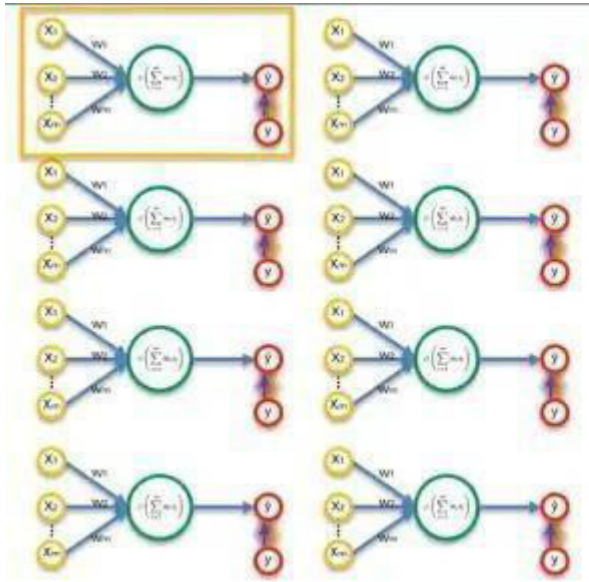
$$for\ i\ in\ range(m):$$

$$\theta_j = \theta_j - \alpha(\hat{y}^i - y^i)x_j^i$$

Stochastic Gradient Descent

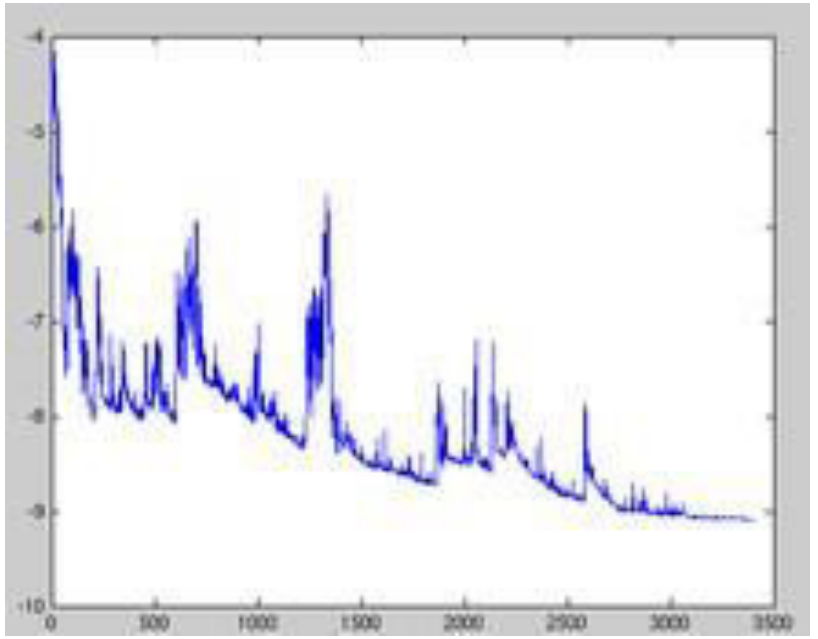


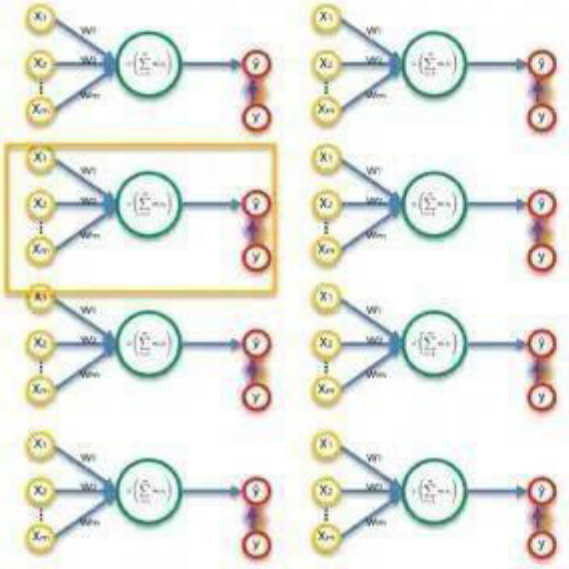
One sample will be used



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$C = \sum \frac{1}{2}(\hat{y} - y)^2$





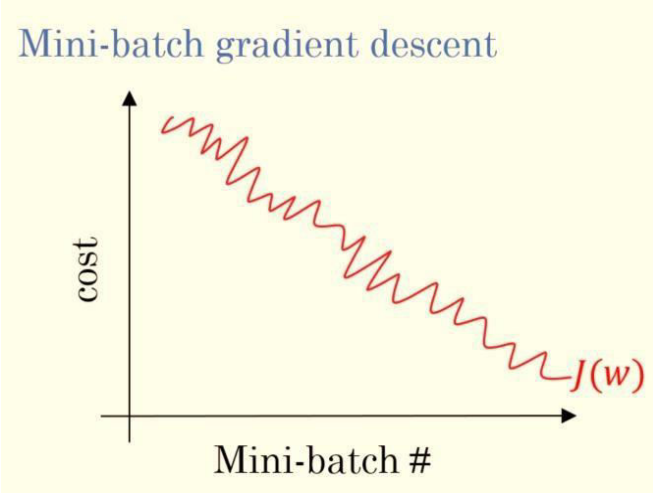
Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$C = \sum \frac{1}{2}(\hat{y} - y)^2$

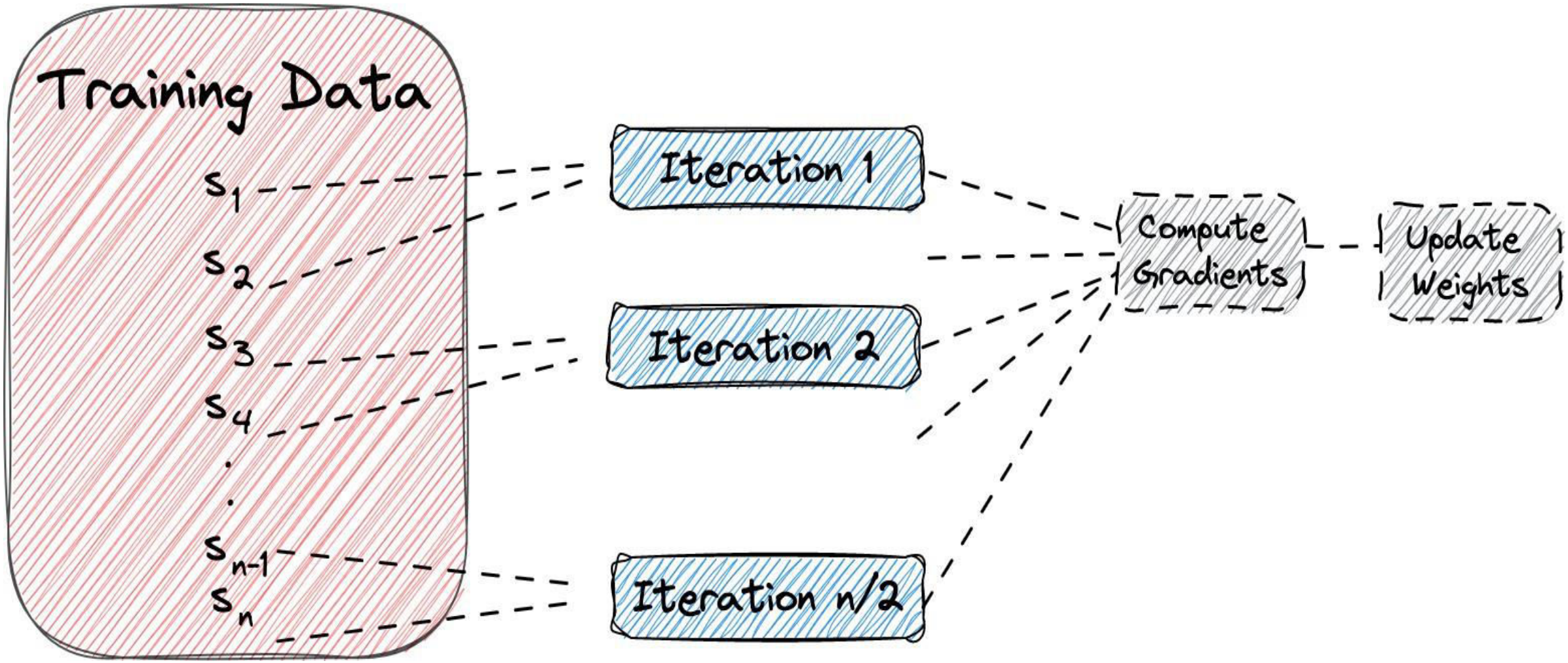
Mini-Batch Descent

- Instead of going through the complete dataset or choosing one random parameter, Mini-batch gradient descent divides the entire dataset into randomly picked batches and optimizes it.

- The mini-batch is a fixed number of training examples that is less than the actual dataset. So, in each iteration, we train the network on a different group of samples until all samples of the dataset are used.



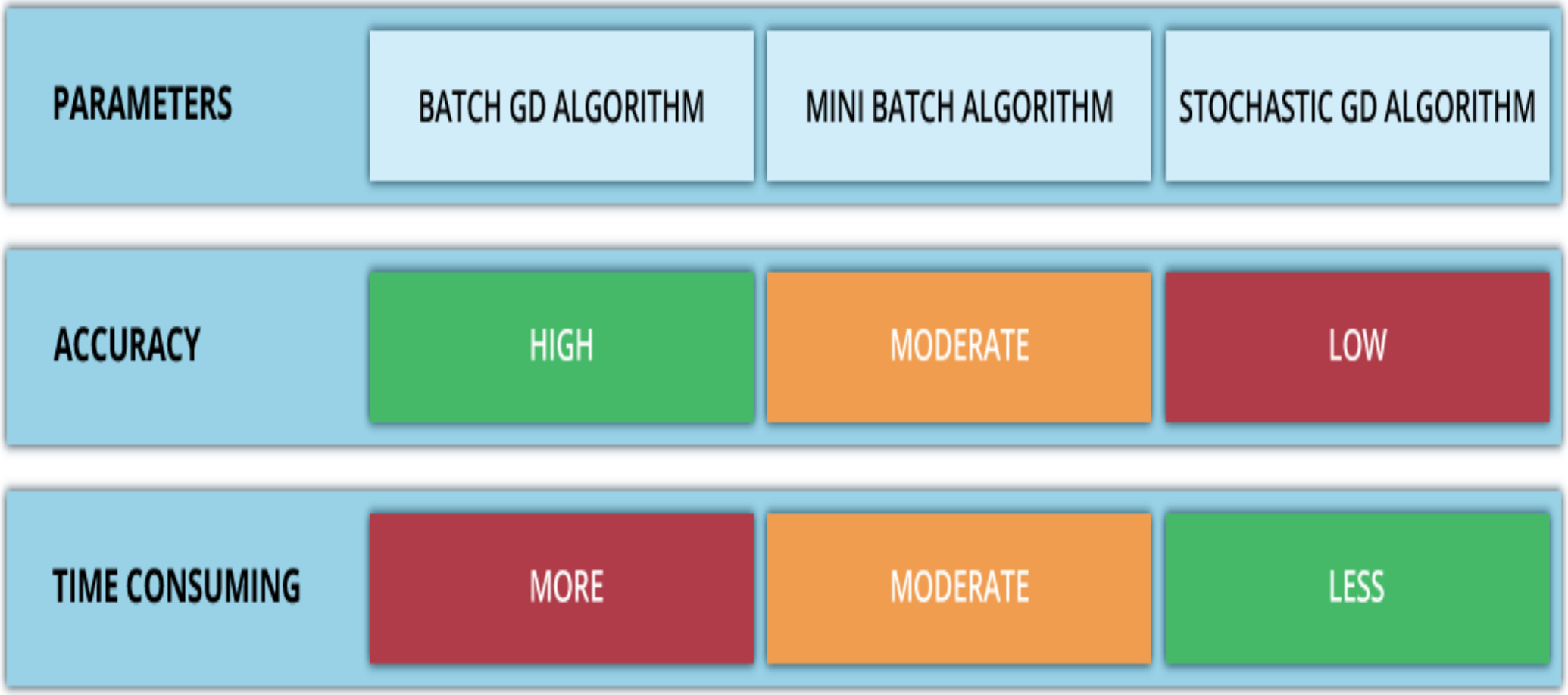
Mini Batch Gradient

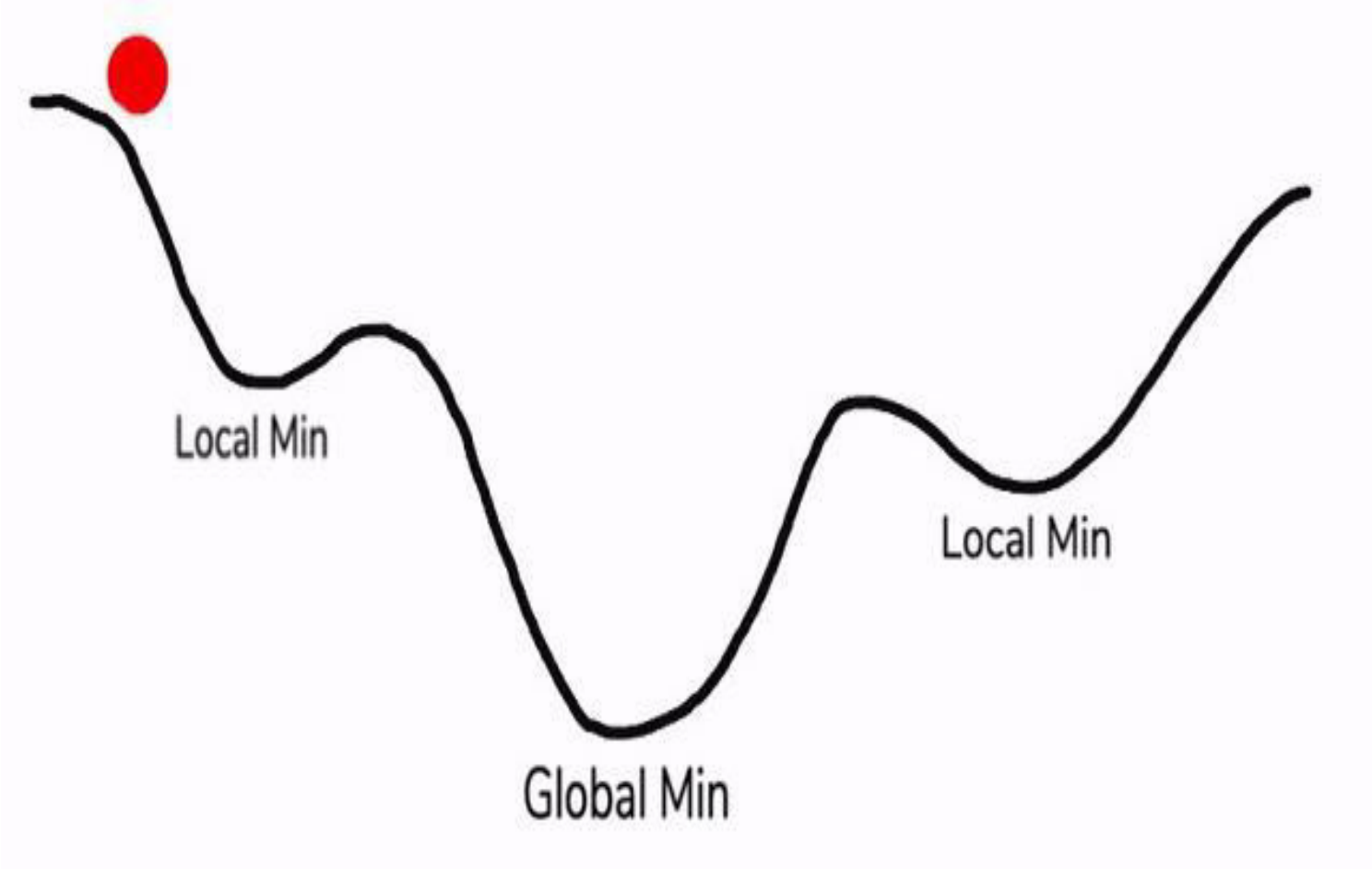


Batch GD vs SGD vs Mini Batch

Batch Descent	Stochastic Descent	Mini-Batch Descent
Uses all training samples while training the dataset and only after the complete cycle makes the required update in the training dataset.	It uses one random training sample and makes the required update in the training dataset.	Divides the dataset into batches and, after completion of every batch makes the required update in the dataset.
Requires too much time and computational space for one epoch.	Faster than Batch descent and requires moderate computational space.	It is the fastest and requires minimal computational space.
Best to use for small training datasets	Best for training big datasets with less computational	Best for large training datasets.

Batch GD vs SGD vs Mini Batch

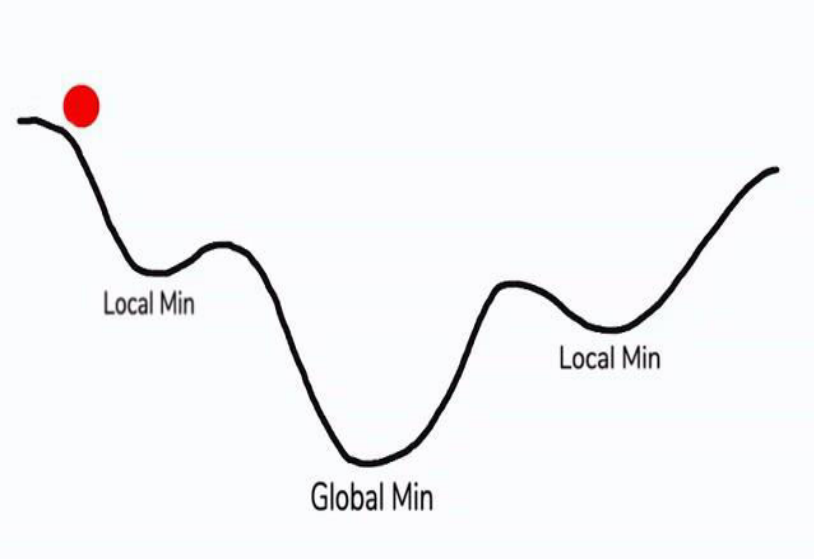




Issue with GD is accidently getting stuck in local minima, where our loss can still be HUGE

Momentum

- Momentum adds to gradient descent by considering previous gradients (the slope of the hill *prior* to where the ball is *currently* at).
- So in the previous case, instead of stopping when the gradient is 0 at the first local minimum, momentum will continue to move the ball forward because it takes into consideration how steep the slope before it was.



Momentum Cont'd

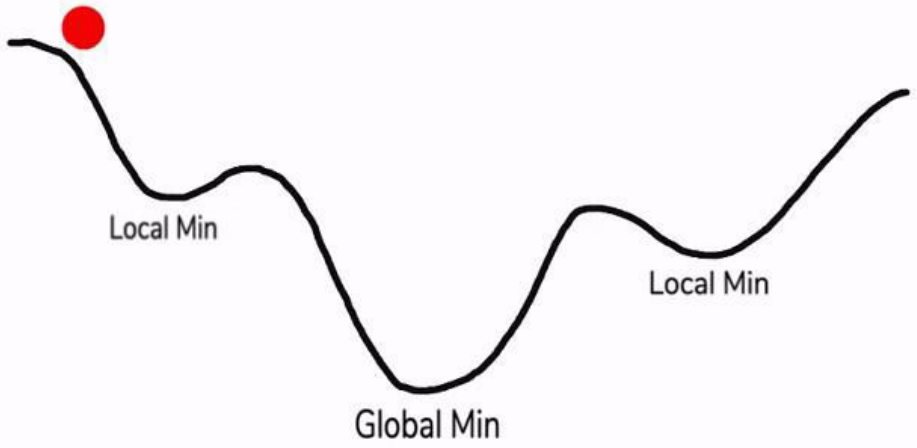
- Momentum is all about speeding up and smoothening the process of gradient descent.
- Notice how the ball is "speeds" up *after* steeper slopes.
- That's momentum taking into consideration previous steep gradients and convincing itself to continue moving, regardless of the local minimum.
- Momentum is a good way to prevent getting stuck in local minima.
- Since momentum constantly considers previous gradients, we can say that momentum calculates *moving averages*.

Adam optimizers Cont'd

- Adam is a gradient-based optimization algorithm, making use of the stochastic gradient extensions of AdaGrad and RMSProp, to deal with machine learning problems involving large datasets and high-dimensional parameter spaces.
- Benefits
  - Straightforward to implement
  - Computationally efficient
  - Little memory requirements
- Drawback
- It does not always converge to an optimal solution, in which switching to stochastic gradient descent with momentum might be better.

Adam optimizers

- Adam is fast, and is quick to settle down.
- It doesn't dwell on his losses, instead, he gets over them quickly and keeps moving until he knows it's finally time to settle down



Adam optimizers Cont'd

Formula

- mt = Aggregate of gradients at time t [Current] (Initially, mt = 0)
- mt-1 = Aggregate of gradients at time t-1 [Previous]
- Wt = Weights at time t
- Wt+1 = Weights at time t+1
- αt = Learning rate at time t
- ∂L = Derivative of Loss Function
- ∂Wt = Derivative of weights at time t
- β = Moving average parameter (Constant, 0.9)

Mathematically:

$$w_{t+1} = w_t - \alpha m_t$$

Where,

$$m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta w_t}\right]$$

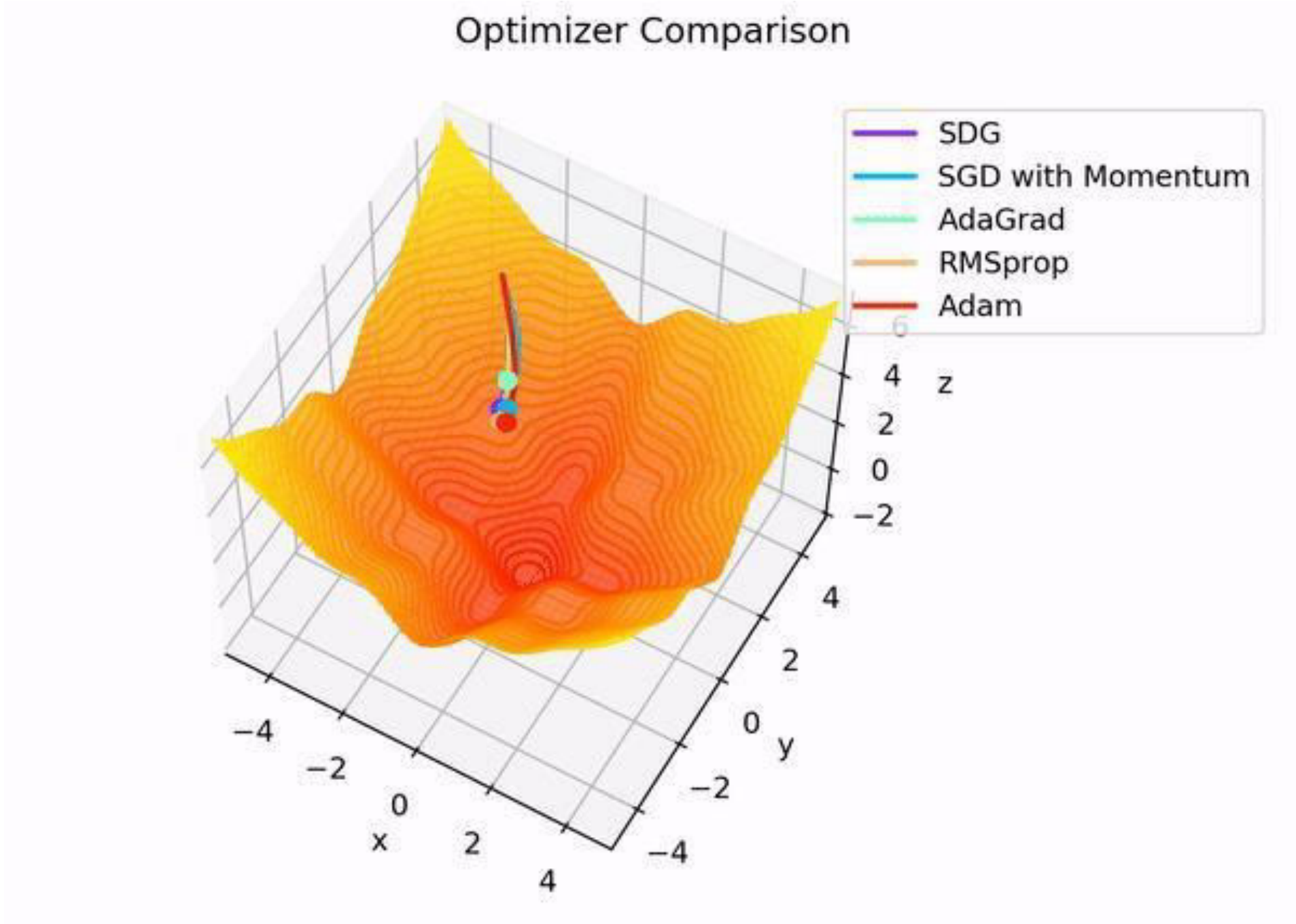
Key points (Epoch, batch, sample)

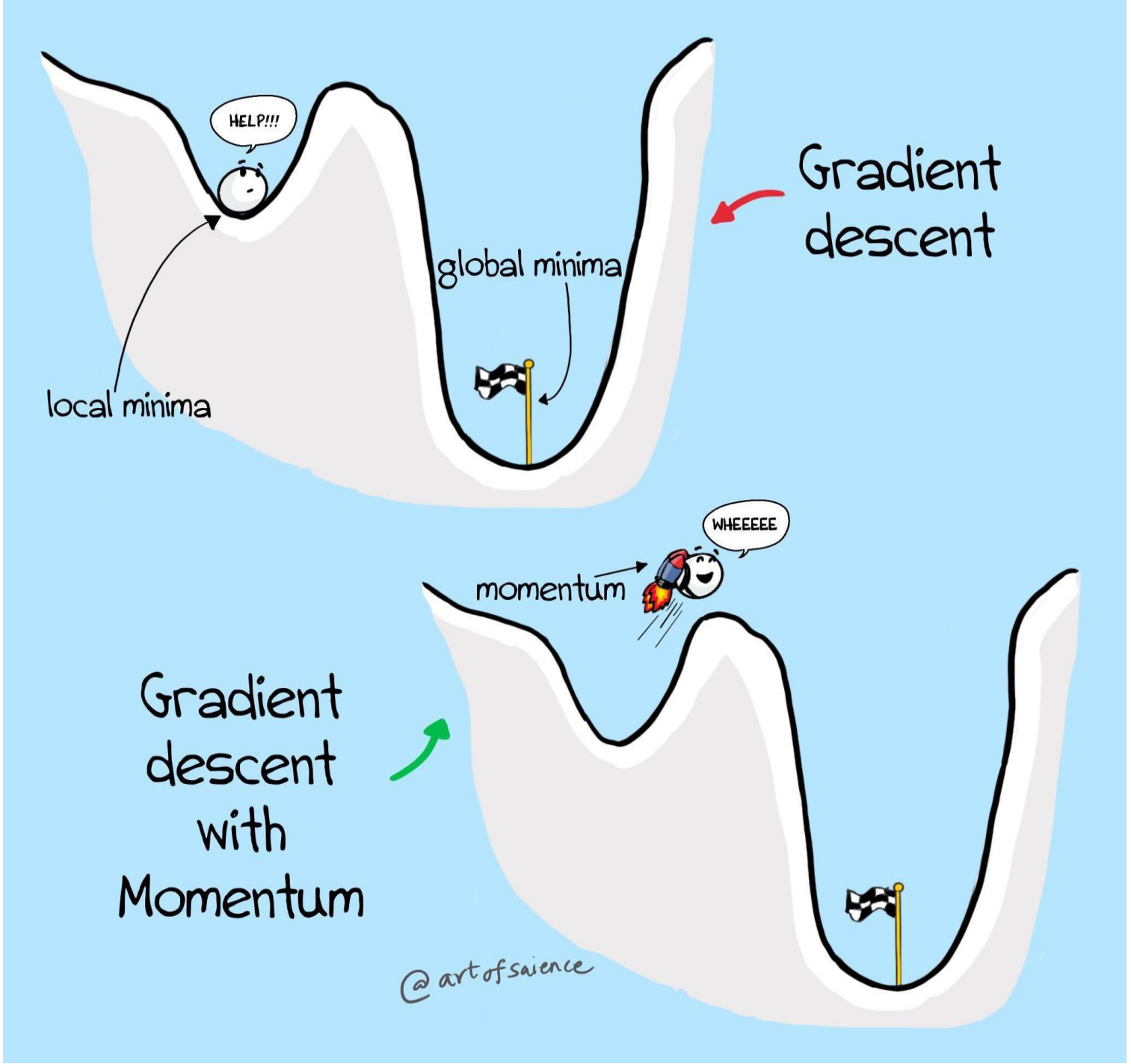
- Example: Assume there are 70 000 samples. Let's arbitrarily choose a batch size of 100 and a number of epochs equal to 500.

$$\#Batches = \frac{\#Samples}{Batch\ size} = \frac{70000}{100} = 700$$

- One epoch will represent a 100 update of the model's parameters. 500 epochs, the dataset takes 500 times. It represents a total of 50 000 batches.

$$Total\ \#Batches = \#Epochs \times Batch\ size = 500 \times 100 = 50000$$


Optimizer Comparison
SDG
SGD with Momentum
AdaGrad
RMSprop
Adam
6
4
2
0
−2
z
−4
−2
0
2
4
x
4
2
0
−2
−4
y


HELP!!!
local minima
global minima
Gradient descent
WHEEEEE
momentum
Gradient descent with Momentum
@artofsaience