Loss Function :

1. Mean Squared Error/Squared loss/ L2 loss

$$MSE = \frac{1}{N}\sum_{i}^{N}(Y_i - \hat{Y}_i)^2$$

Advantage

- **Easy Interpretation:** The MSE is straightforward to understand.
- **Always Differential:** Due to the squaring, it is always differentiable.
- **Single Local Minimum:** It has only one local minimum.

Disadvantage

- **Error Unit in Squared Form:** The error is measured in squared units, which might not be intuitively interpretable.
- **Not Robust to Outliers:** MSE is sensitive to outliers.

  Can lead to overfitting in the presence of noise.

Preferred Activation Function

:

- **Linear Activation**: Typically used in the output layer for regression tasks since it can produce a range of real numbers.
- **ReLU Activation**: Can be used in hidden layers to introduce non-linearity, but not in the output layer.

2. Mean Absolute Error/ L1 loss Functions

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |Yi - \hat{Y}i|$$

Advantage

- **Intuitive and Easy:** MAE is easy to grasp.
- **Error Unit Matches Output Column:** The error unit is the same as the output column.
- **Robust to Outliers:** MAE is less affected by outliers.

Disadvantage

Graph Not Differential: The MAE graph is not differentiable, so gradient descent cannot be applied directly. Subgradient calculation is an alternative.

Less sensitive to large errors, which might not be ideal in all cases.

- **Preferred Activation Function**:
  - **Linear Activation**: Same as MSE, used in the output layer for regression tasks.
  - **ReLU Activation**: Commonly used in hidden layers for non-linearity.

Huber Loss

The Huber loss is used in robust regression and is less sensitive to outliers compared to squared error loss.

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2 \qquad |y_i - \hat{y}_i| \le \delta$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) \qquad |y_i - \hat{y}_i| > \delta$$

- **n:** The number of data points.
- **y:** The actual value (true value) of the data point.
- **ŷ:** The predicted value returned by the model.
- **δ:** Defines the point where the Huber loss transitions from quadratic to linear.

Advantag

- **Robust to Outliers:** Huber loss is more robust to outliers.
- **Balances MAE and MSE:** It lies between MAE and MSE.

Disadvantage

- **Complexity:** Optimizing the hyperparameter δ increases training requirements.

- Requires tuning the hyperparameter δ.

  δ\delta
- More complex to compute compared to MSE and MAE.

- **Preferred Activation Function**:
  - **Linear Activation**: Used in the output layer to provide a continuous output range.
  - **ReLU Activation**: Applied in hidden layers to capture complex patterns.

Classification Loss

1. Binary Cross Entropy/log loss Functions in machine learning models

It is used in binary classification problems like two classes. example a person has covid or not or my article gets popular or not.

Binary cross entropy compares each of the predicted probabilities to the actual class output which can be either 0 or 1. It then calculates the score that penalizes the probabilities based on the distance from the expected value. That means how close or far from the actual value.

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)$$

- yi – actual values
- yihat – Neural Network prediction

Advantage –

- A cost function is a differential.
- Suitable for probabilistic interpretation of outputs.
- Penalizes confident but wrong predictions more than less confident predictions.

Disadvantage –

- Multiple local minima
- Not intuitive
- Sensitive to class imbalance; performance can degrade if classes are highly imbalanced.

Preferred Activation Function

:

- **Sigmoid Activation**: Used in the output layer to produce probabilities between 0 and 1.
- **ReLU Activation**: Commonly used in hidden layers for its efficiency and non-linear properties.

Categorical Cross Entropy

Categorical Cross entropy is used for Multiclass classification and softmax regression.

loss function = -sum up to k(yjlagyjhat) where k is classes

$$\text{Loss} = -\sum_{j=1}^{K} y_j \log(\hat{y}_j)$$

where k is number of classes in the data

cost function = -1/n(sum upto n(sum j to k (yijloghijhat))

$$\text{Cost} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}[y_{ij}\log(\hat{y}_{ij})]$$

where

- k is classes,
- y = actual value
- yhat – Neural Network prediction

Note – In multi-class classification at the last neuron use the softmax activation function.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

if problem statement have 3 classes

- **Advantages**:
  - Effective for problems where the output is a probability distribution over multiple classes.
  - Provides a natural measure for how well the predicted probability distribution matches the true distribution.
- **Disadvantages**:
  - Requires the target labels to be one-hot encoded.
  - Can be sensitive to class imbalance.

- **Preferred Activation Function**:
  - **Softmax Activation**: Used in the output layer to produce a probability distribution over multiple classes.
  - **ReLU Activation**: Frequently used in hidden layers to add non-linearity.

Sparse Categorical Cross Entropy :

- **Advantages**:

- More computationally efficient than categorical cross entropy, especially for a large number of classes.
    - Does not require one-hot encoding of the labels.
- **Disadvantages**:
    - May be less intuitive to interpret compared to categorical cross entropy.
    - Still sensitive to class imbalance.

Preferred Activation Function

:

- **Softmax Activation**: Applied in the output layer for a probability distribution over classes.
- **ReLU Activation**: Often used in hidden layers to introduce non-linearity.

Summary of Activation Functions and Their Pairings

- **Linear Activation**:
    - **Loss Functions**: MSE, MAE, Huber Loss
    - **Use Case**: Output layer for regression tasks.
- **ReLU Activation**:
    - **Loss Functions**: MSE, MAE, Huber Loss, Binary Cross Entropy, Categorical Cross Entropy, Sparse Categorical Cross Entropy
    - **Use Case**: Hidden layers in most types of neural networks for introducing non-linearity and addressing the vanishing gradient problem.
- **Sigmoid Activation**:
    - **Loss Functions**: Binary Cross Entropy
    - **Use Case**: Output layer for binary classification tasks to produce probabilities.
- **Softmax Activation**:

- **Loss Functions**: Categorical Cross Entropy, Sparse Categorical Cross Entropy
- **Use Case**: Output layer for multiclass classification tasks to generate a probability distribution across multiple classes.

Considerations for Activation Functions:

- **Sigmoid**: While useful in the output layer for binary classification, it can suffer from vanishing gradient problems when used in hidden layers, leading to slow training and poor convergence.
- **ReLU**: Popular for hidden layers due to its ability to mitigate the vanishing gradient problem, but can suffer from dying ReLU problem where neurons output zero for all inputs.
- **Softmax**: Essential for multiclass classification in the output layer, providing a clear probabilistic interpretation of the model's predictions.
- **Linear**: Directly outputs values without applying any transformation, making it suitable for regression tasks.

Terms and Terminology:

1. Epoch:

- **Definition**: An epoch refers to one complete pass through the entire training dataset during the training of a machine learning model.
- **Usage**: During each epoch, the model sees and learns from the entire dataset, updating its parameters (weights and biases) based on the observed data and the optimization algorithm used (e.g., gradient descent).
- **Example**: If you train a neural network for 10 epochs on a dataset with 1000 samples, the model will go through the entire dataset 10 times, updating its parameters after each pass.

2. Batch:

- **Definition**: A batch is a subset of the training dataset used during one iteration of model training.
- **Usage**: Instead of updating the model's parameters after every individual sample (Stochastic Gradient Descent) or the entire dataset (Batch Gradient Descent), training is often performed in batches to balance computational efficiency and model stability.
- **Example**: If you have a training dataset of 1000 samples and you choose a batch size of 32, each training iteration will involve updating the model based on 32 samples at a time.

3. Iteration:

- **Definition**: An iteration refers to one update of the model's parameters based on one batch of training data.
- **Usage**: In each iteration, the model computes predictions for the current batch, calculates the loss, computes gradients, and updates the model's parameters accordingly.

4. Learning Rate:

- **Definition**: The learning rate is a hyperparameter that controls the size of the step taken during optimization when updating the model's parameters.
- **Usage**: A higher learning rate means larger updates to the parameters, potentially leading to faster convergence but with the risk of overshooting the optimal solution. Conversely, a lower learning rate ensures more conservative updates but may result in slower convergence.
- **Example**: Common learning rates range from 0.1 to 0.0001, depending on the problem and optimization algorithm.

Gradient Descent :

What is Gradient Descent?

Gradient Descent is an optimization algorithm used to minimize the cost (or loss) function in machine learning and deep learning models. The cost function measures how well a model's predictions match the actual data. Gradient Descent iteratively adjusts the model's parameters to find the values that minimize the cost function.

Why Are We Using Gradient Descent?

1. **Optimization**: Gradient Descent helps find the optimal parameters for the model, minimizing the error between predicted and actual values.
2. **Efficiency**: It is computationally efficient and works well with large datasets and high-dimensional parameter spaces.
3. **Scalability**: Gradient Descent can be used for both linear and non-linear models, making it versatile for various machine learning algorithms.
4. **Convergence**: It provides a systematic way to converge towards a minimum cost, improving the model's accuracy.

Why Is It Called Gradient Descent?

The term "Gradient Descent" comes from the following:

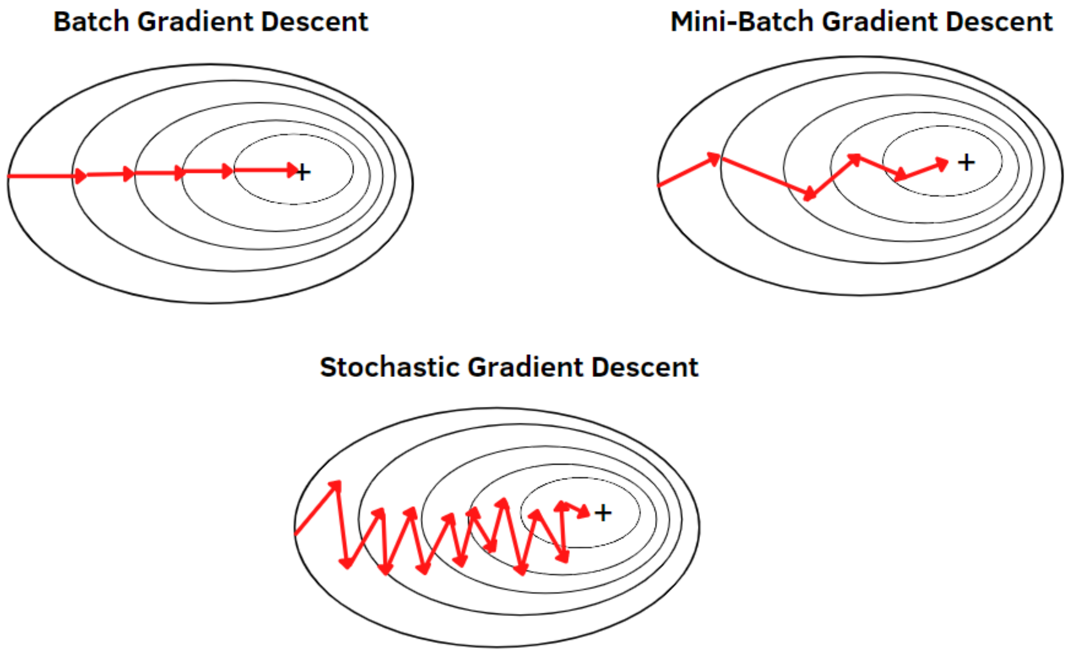
1. **Gradient**: The gradient (or slope) of the cost function with respect to the model's parameters indicates the direction and rate of the steepest increase in the cost function. Mathematically, the gradient is a vector of partial derivatives.
2. **Descent**: The algorithm aims to move in the opposite direction of the gradient (downhill) to decrease the cost function. Hence, it descends along the gradient.

How Gradient Descent Works

1. **Initialize Parameters**: Start with random values for the model parameters.
2. **Calculate Gradient**: Compute the gradient of the cost function with respect to each parameter.
3. **Update Parameters**: Adjust the parameters in the opposite direction of the gradient by a small step size (learning rate).
4. **Iterate**: Repeat the process until the cost function converges to a minimum or stops improving significantly.

Types of Gradient Descent

1. **Batch Gradient Descent**: Uses the entire dataset to compute the gradient and update parameters. It is computationally expensive for large datasets.
2. **Stochastic Gradient Descent (SGD)**: Updates parameters using only one data point at a time, making it faster but with more fluctuations.
3. **Mini-Batch Gradient Descent**: Combines aspects of both batch and stochastic gradient descent by updating parameters using a small batch of data points, balancing speed and stability.



Batch Gradient Descent (Vanilla Gradient Descent)

- **Procedure**: Calculates error for each example in the training dataset but updates the model only after all examples are processed (one epoch).
- **Advantages**:
    - Computationally efficient.
    - Produces a stable error gradient and stable convergence.
    - Suitable for parallel processing implementations.
- **Disadvantages**:
    - Can converge to suboptimal parameters.
    - Requires the entire dataset in memory.
    - Slow and requires more computational power for large datasets.

2. Stochastic Gradient Descent (SGD)

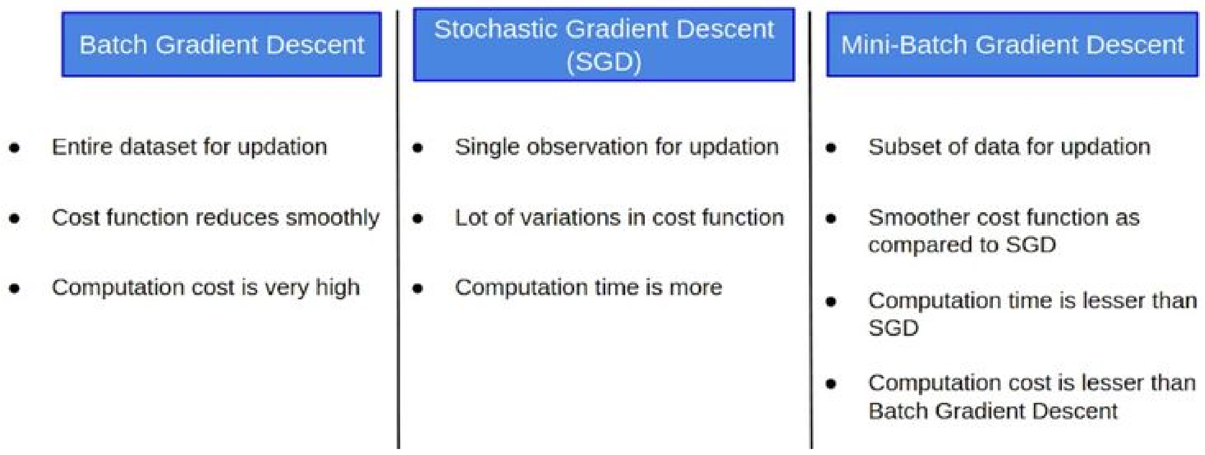
- **Procedure**: Updates model parameters for each training sample one at a time.
- **Advantages**:
    - Faster updates and less computational power required.
    - Immediate feedback on model performance.
    - Easier to implement for beginners.
    - Noisy updates help avoid local minima.
    - Suitable for larger datasets.
- **Disadvantages**:
    - More computationally intensive due to frequent updates.
    - Noisy gradient signals can cause high variance in the training process.
    - Difficult to converge to the minimum error due to the noisy learning process.

3. Mini-Batch Gradient Descent

- **Procedure**: Divides the training dataset into small batches and updates the model for each batch.
- **Advantages**:
    - More frequent updates than batch gradient descent.
    - More stable convergence than SGD.
    - Efficient use of memory and computational resources.
    - Suitable for parallel processing implementations.
- **Disadvantages**:
    - Requires setting an additional hyperparameter (batch size).
    - Error information should be accumulated over mini-batches.
    - Potentially more complex to implement.

Key Points

- **Mini-Batch Gradient Descent**: Recommended method as it balances the effectiveness of batch gradient descent with the durability of stochastic gradient descent.
- **Batch Gradient Descent**: Efficient but requires the entire dataset in memory and can be slow for large datasets.
- **Stochastic Gradient Descent**: Fast and suitable for large datasets but can produce noisy gradients and high variance in the training process.
- **Learning Rate**: Higher learning rates speed up training but can lead to non-optimal weights; lower learning rates are slower but can find more optimal weights.



Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini-Batch Gradient Descent
Entire dataset for updation	Single observation for updation	Subset of data for updation
Cost function reduces smoothly	Lot of variations in cost function	Smoother cost function as compared to SGD
Computation cost is very high	Computation time is more	Computation time is lesser than SGD
		Computation cost is lesser than Batch Gradient Descent

1. Speed

Fastest to Slowest:

1. **Stochastic Gradient Descent (SGD)**
   - **Speed**: Updates the model parameters after each training sample. This means it has the most frequent updates, leading to faster iterations, especially for large datasets.
   - **Pros**: Faster iteration per update, suitable for very large datasets.
2. **Mini-Batch Gradient Descent**
   - **Speed**: Balances between Batch Gradient Descent and SGD. It updates the model parameters after a small batch of samples, leading to more frequent updates than Batch Gradient Descent but less frequent than SGD.
   - **Pros**: Provides a good compromise, usually faster than Batch Gradient Descent but slower than SGD on a per-update basis.
3. **Batch Gradient Descent**
   - **Speed**: Updates the model parameters only after evaluating the entire training dataset. This leads to fewer updates but each update is more computationally expensive.
   - **Pros**: Each update is computationally intensive, slower on a per-update basis.

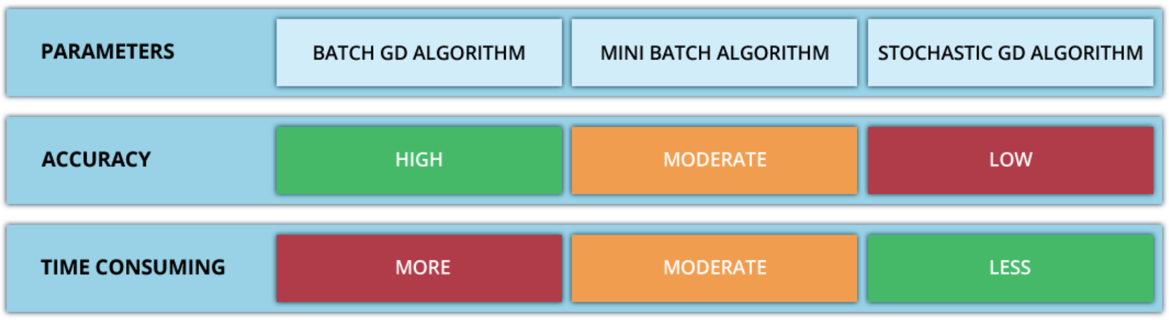
2. Convergence

Most Stable to Least Stable:

1. **Batch Gradient Descent**
    - **Convergence**: Most stable error gradient as it considers the entire dataset for each update, leading to smoother and more predictable convergence.
    - **Pros**: Less variance in updates, typically converges to the global minimum if a convex cost function.
    - **Cons**: Can converge to suboptimal local minima in non-convex cost functions, slow convergence for large datasets.
2. **Mini-Batch Gradient Descent**
    - **Convergence**: Provides a balance between stability and frequency of updates. Convergence is generally stable but may have some variance depending on batch size.
    - **Pros**: Less variance than SGD, more stable convergence, avoids some local minima due to smaller batch sizes.
    - **Cons**: Still might be influenced by batch size leading to fluctuations, generally more stable than SGD.
3. **Stochastic Gradient Descent (SGD)**
    - **Convergence**: Least stable due to high variance in updates as each update is based on a single training sample. Can make the convergence path noisy and fluctuating.
    - **Pros**: Can escape local minima due to high variance, potentially finding a better global minimum.
    - **Cons**: Highly noisy convergence path, requires more epochs to converge, less predictable.

Summary:

- **Speed**: SGD > Mini-Batch Gradient Descent > Batch Gradient Descent
- **Convergence Stability**: Batch Gradient Descent > Mini-Batch Gradient Descent > SGD



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

Batch Gradient Descent (BGD)

- **When to Use**:
    - **Small Datasets**: Suitable when the entire dataset can fit into memory, allowing for efficient computation of the gradient.
    - **Stable Convergence**: Preferred when you need a very stable error gradient and smooth convergence, especially for convex optimization problems.
    - **Parallel Processing**: When the implementation can leverage parallel processing to speed up computations.
- **Not Suitable For**:
    - **Large Datasets**: Not ideal for very large datasets due to the need to load the entire dataset into memory.
    - **Real-time Systems**: Where quick updates are necessary since it only updates after processing the entire dataset.

. Stochastic Gradient Descent (SGD)

- **When to Use**:
    - **Large Datasets**: Effective for large datasets where loading the entire dataset into memory is not feasible.

        - **Online Learning**: Suitable for real-time or online learning scenarios where the model needs to be updated frequently with new data points.
        - **Avoiding Local Minima**: When the problem has many local minima, the noise in SGD can help escape local minima and potentially find a better global minimum.
    - **Not Suitable For**:
        - **Stable Convergence Needed**: When you need a smooth and stable convergence path as SGD can be noisy.
        - **High Variance in Updates**: If the high variance in updates is a problem for the specific application.

3. Mini-Batch Gradient Descent (MBGD)

- **When to Use**:
    - **Balance of Speed and Stability**: Preferred when you need a balance between the fast updates of SGD and the stable convergence of BGD.
    - **Large Datasets**: Suitable for large datasets where batch size can be adjusted to fit into memory, allowing for efficient computation.
    - **Deep Learning**: Often used in training neural networks where mini-batch sizes can be tuned to match the GPU/CPU memory requirements.
- **Not Suitable For**:
    - **Very Small Datasets**: Where the benefits of mini-batch updates may not be significant.
    - **Real-time Systems**: If immediate updates are needed after each data point.