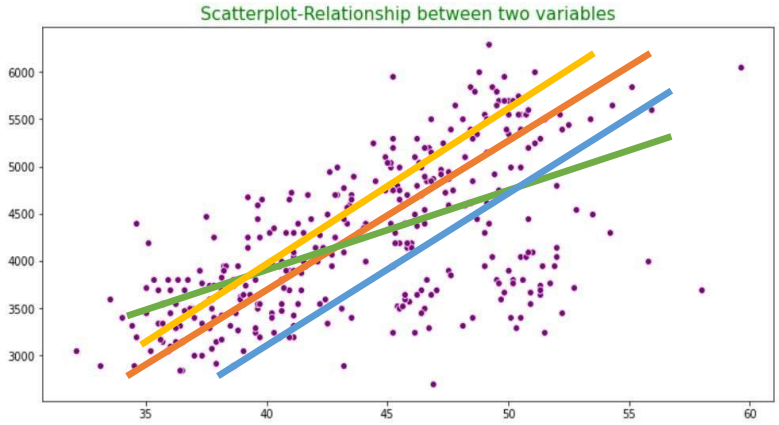
Loss Functions

Loss Functions

- Many ML/DL problems require us to optimize a function J of some variables w

- The line for which the error between the predicted values and the observed values is the minimum is called the best fit line
    - J: loss function which expresses how far off the mark our computed output is
    - Depends on the parameters of the model, e.g. weights of the neural network

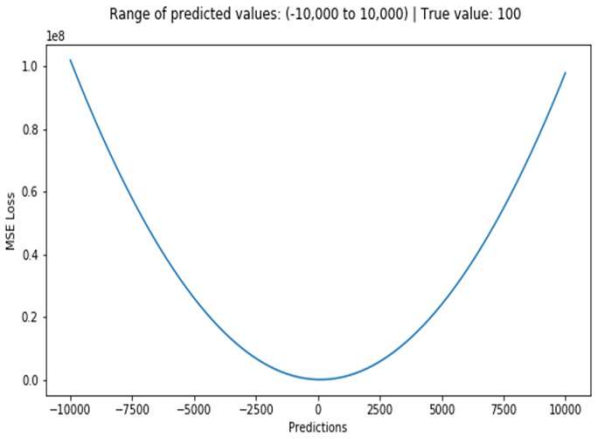


NOTE: The loss function is to capture the difference between the actual and predicted values for a single record whereas cost functions aggregate the difference for the entire training dataset.

Loss Function - Examples

❑ Regression:

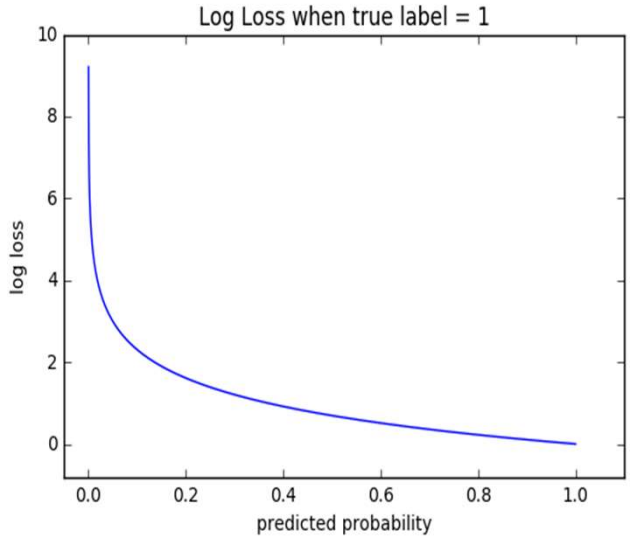
- Mean Squared Error: $\frac{1}{N}\sum_j (y_j^{actual} - y_j^{predicted})^2$



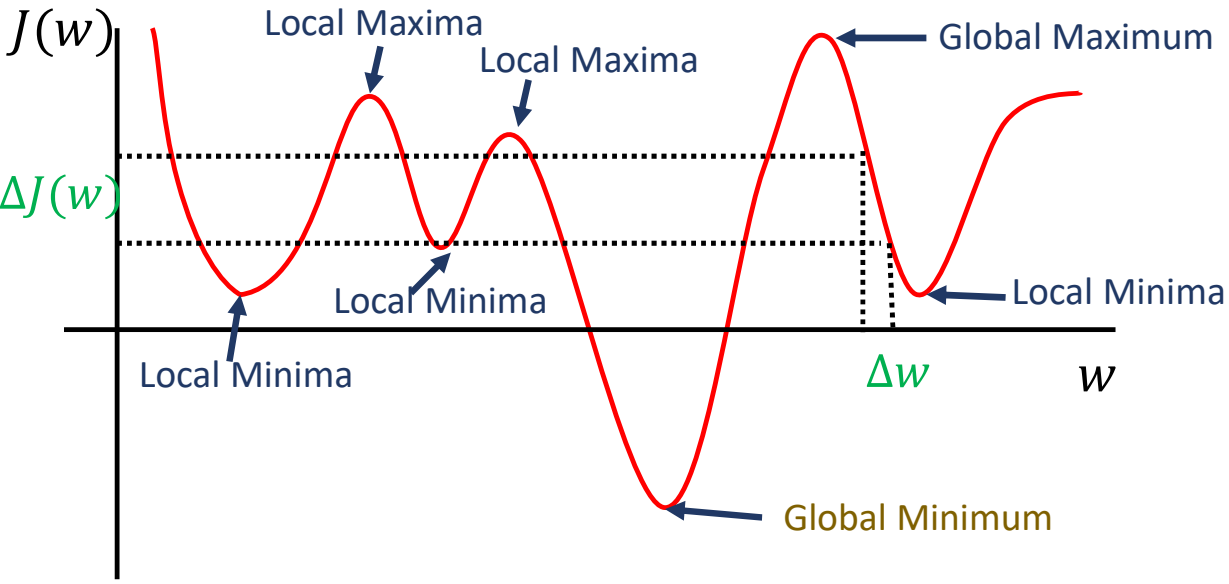
❑ Classification:

- Cross Entropy Loss: $-(y \log y' + (1-y)\ \log(1-y'))$

Actual Value (y)	Predicted Value (y')	Loss
0	0	0
0	1	∞
1	0	∞
1	1	0



Loss Functions - Characteristics



- Finding the optima requires the derivatives/gradients of the function $J(w)$
- Magnitude of derivative at a point $\left(\frac{dJ(w)}{dw}\right)$ is the rate of change of the function at that point

- Positive/Negative derivative means $J(w)$ is increasing/decreasing at w, if the value of w is increased by a very small amount
- Derivative becomes zero ($\nabla J(w) = 0$) at the points where $J(w)$ has its maxima or minima

Derivatives - Properties

- The function's optima can be determined from the second derivative and, also from how the derivative itself changes

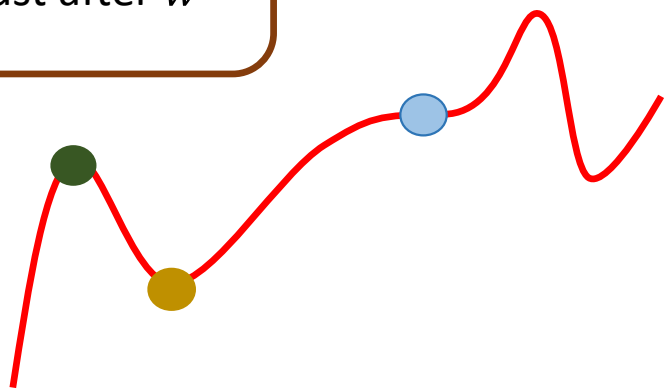
  - $\frac{dJ}{dw} = 0$, and $\frac{d}{dw}\left(\frac{dJ}{dw}\right) < 0$. w is a maxima

  - $\frac{dJ}{dw} = 0$, and $\frac{d}{dw}\left(\frac{dJ}{dw}\right) > 0$. w is a minima

  - $\frac{dJ}{dw} = 0$, and $\frac{d}{dw}\left(\frac{dJ}{dw}\right) = 0$. w may be a saddle

✓ $J'(w) = 0$ at w
✓ $J'(w) > 0$ just before w
✓ $J'(w) < 0$ just after w
w is a maxima

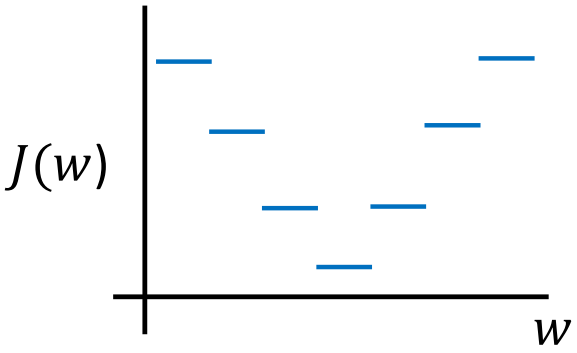
✓ $J'(w) = 0$ at w
✓ $J'(w) < 0$ just before w
✓ $J'(w) > 0$ just after w
w is a minima

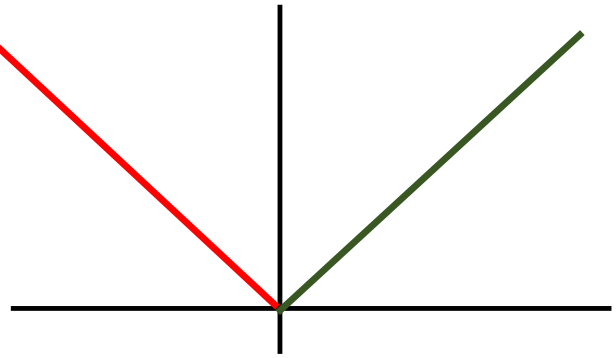


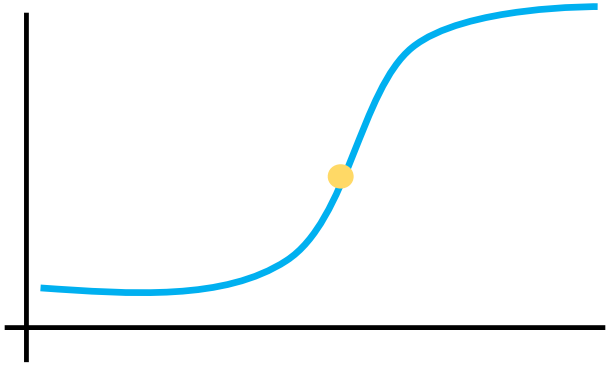
✓ $J'(w) = 0$ at w
✓ $J'(w) = 0$ just before w
✓ $J'(w) = 0$ just after w

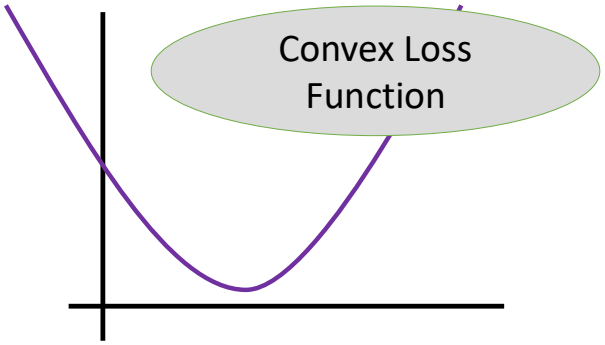
Loss Function - Properties

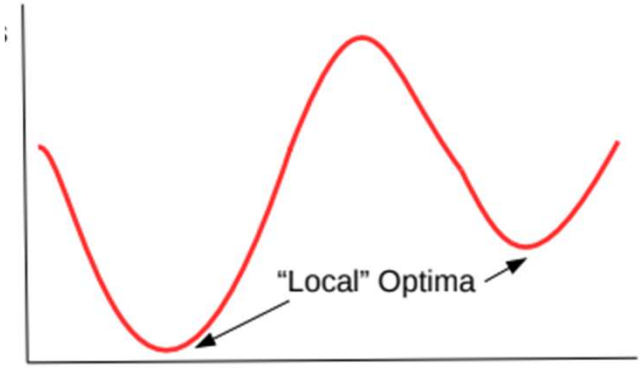
- The loss function should be Globally Continuous and Differentiable







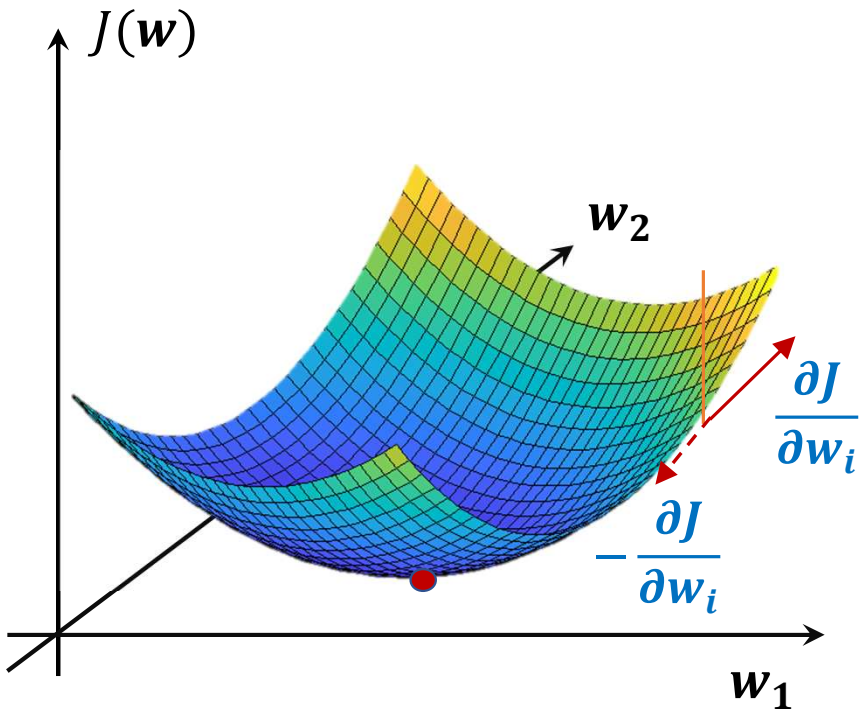




Point of inflection is where the curve changes from convex to concave, or vice versa. The second derivative at the point of inflection is zero. Note complicated loss functions can have this point like saddle point. In the simplest form it is best to avoid these

Gradient Descent

- Goal: Find the optimal weights that minimize the total Loss
- The weights w_i are modified iteratively to reduce $J(\boldsymbol{w})$ for each training sample
- The gradient of the loss function $\left(\frac{\partial J}{\partial w_i}\right)$ here gives the direction of fastest positive change of the loss function $J(\boldsymbol{w})$ when the parameters w_i are modified



Optimization by Gradient Descent

- Gradient Descent is an iterative algorithm, since it requires several steps/iterations to find the optimal solution

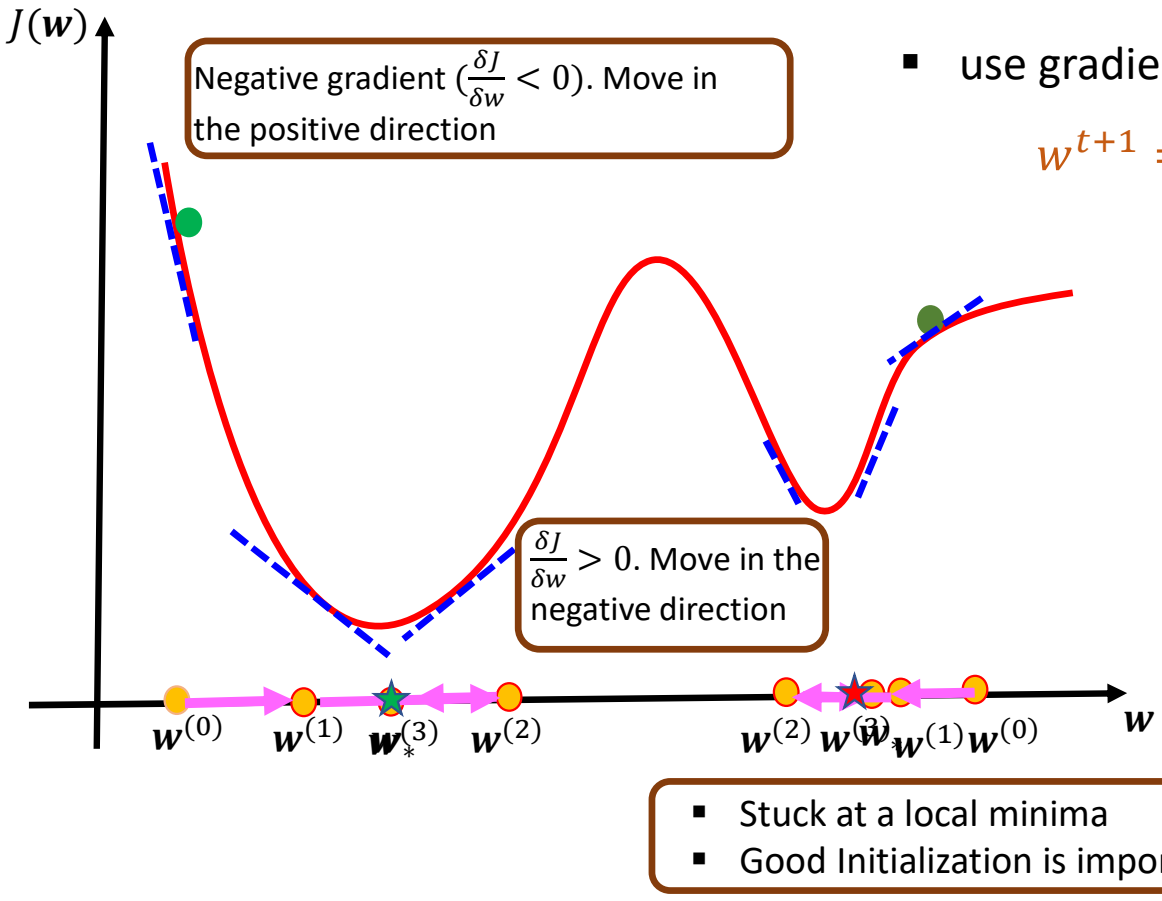
1. Randomly initialize w=[α , β]; call it: $\mathbf{w}^0$
2. Compute the gradient of the error function J at $\mathbf{w}$: $\nabla J = \frac{\partial \mathrm{J}}{\partial \mathbf{w}}$
3. $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla J$ [move in the opposite direction of the gradient, with η as learning rate]
4. Repeat steps 2 and 3 until convergence

- Convergence: when $\|\eta \nabla J\|$ becomes small

$$\mathrm{J} = \frac{1}{2}\sum_i (\beta x_i + \alpha - y_i)^2$$

$$\nabla J = \begin{bmatrix} \sum_i (\beta x_i + \alpha - y_i) \\ \sum_i x_i\,(\beta x_i + \alpha - y_i) \end{bmatrix}$$

Gradient Descent: An Illustration



- Can you use this approach to solve maximization problems?
- use gradient ascent

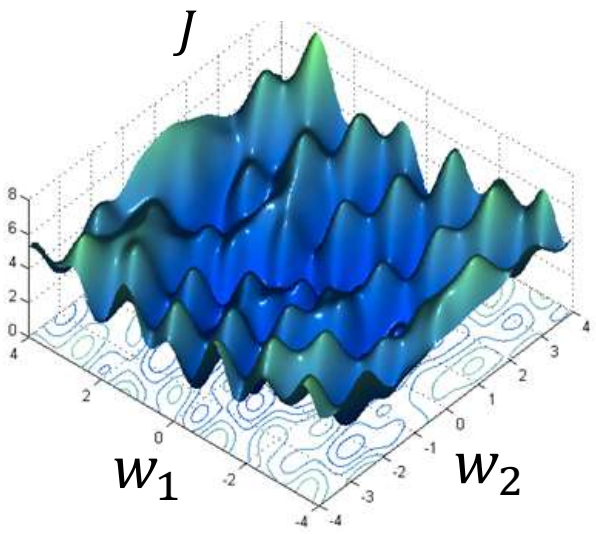
$$w^{t+1} = w^t + \eta \frac{\partial J}{\partial w}$$

Move in the direction of the gradient

- Stuck at a local minima
- Good Initialization is important

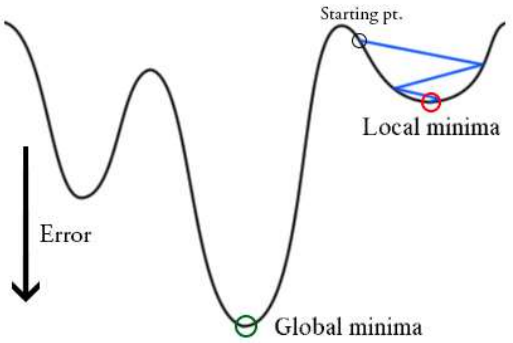
GDA and Local Minima

- The loss function for most tasks becomes highly complex and non-convex



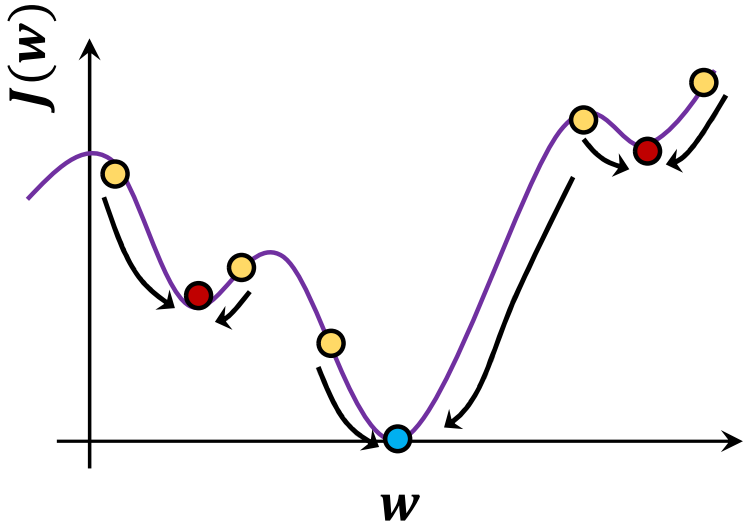
- Gradient Descent Algorithm stops when a local minimum of the loss surface is reached. Does not guarantee reaching a global minimum

- The time to obtain global optimum becomes longer as the model gets more complex



Solution: Initialization

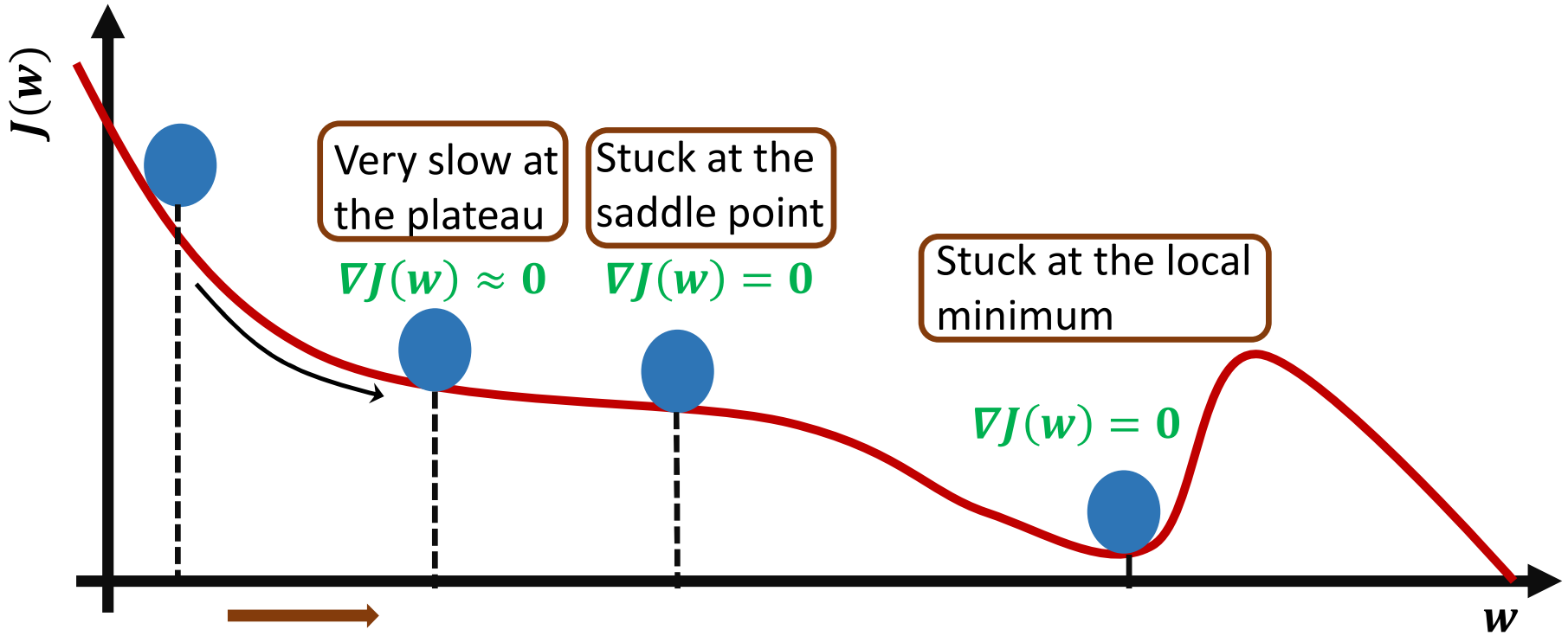
- Random initialization results in different initial parameters w^0
  - Gradient descent may reach different minima at every run
  - Therefore, NN will produce different predicted outputs
- Avoid bad initializations
  - Uniform/Normal random around 0 is reasonable in most cases
- Can reach global minima if initialization is close
  - Get approximate solutions; use them as initializations for GD



Currently, we don't have an algorithm that guarantees reaching a global minimum for an arbitrary loss function

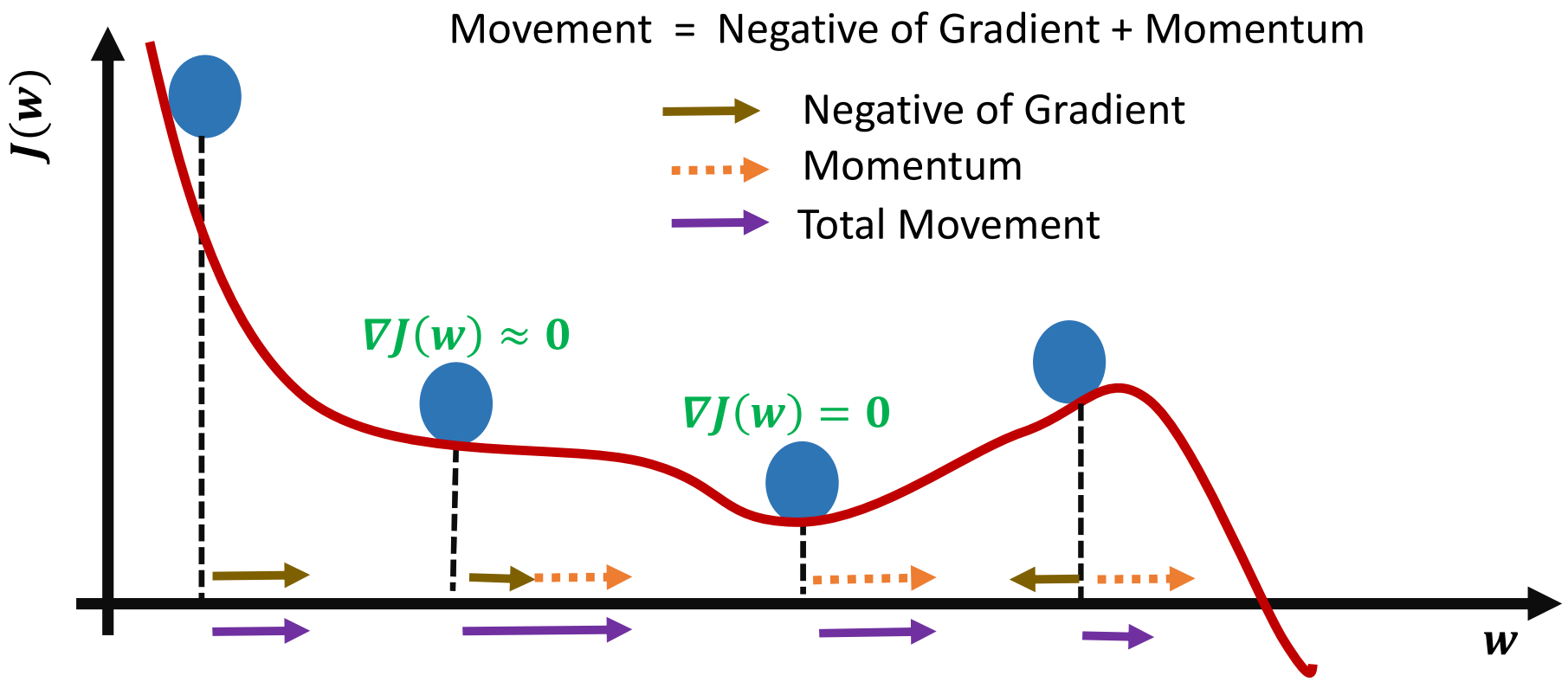
Gradient Descent - Problems

- The Gradient Descent algorithm can be very slow at plateau, or it can get struck at saddle points, local minimum



Solution - Momentum

- The Gradient Descent uses the momentum of the gradient for parameter optimization



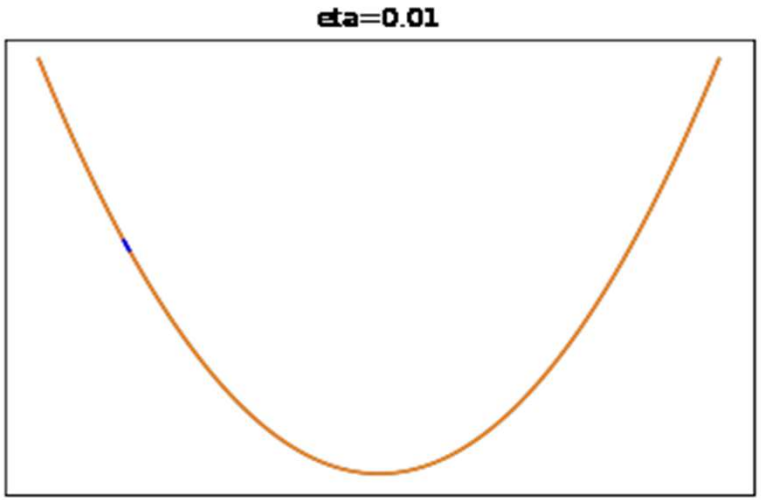
Gradient Descent with Momentum

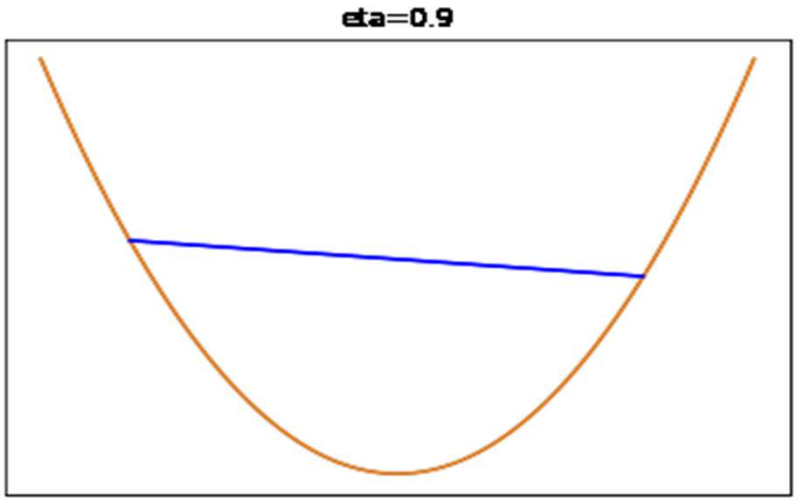
- Parameters update in GD with momentum : $\boldsymbol{w^{t+1} = w^t - V^t}$
  - Where: $\boldsymbol{V^t = \gamma V^{t-1} + \eta \nabla J(w^t)}$
- Compare to vanilla GD: $\boldsymbol{w^{t+1} = w^t - \eta \nabla J(w^t)}$
- The term V^t is called momentum
  - This term accumulates the gradients from the past several steps
  - It is similar to a momentum of a heavy ball rolling down the hill
- The parameter γ referred to as a coefficient of momentum
  - A typical value of the parameter γ is 0.9
- This method updates the parameters w in the direction of the weighted average of the past gradients

Learning Rate

- The gradient tells us the direction in which the loss has the steepest rate of increase, but it does not tell us how far along the opposite direction we should step

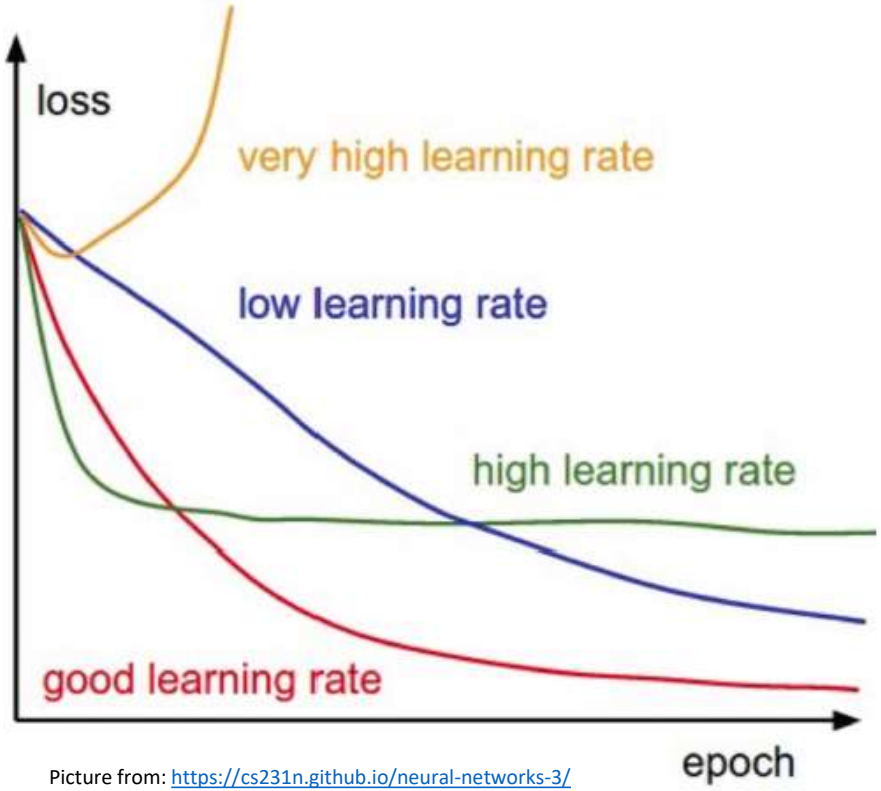
- Choosing the learning rate is one of the most important hyper-parameter setting





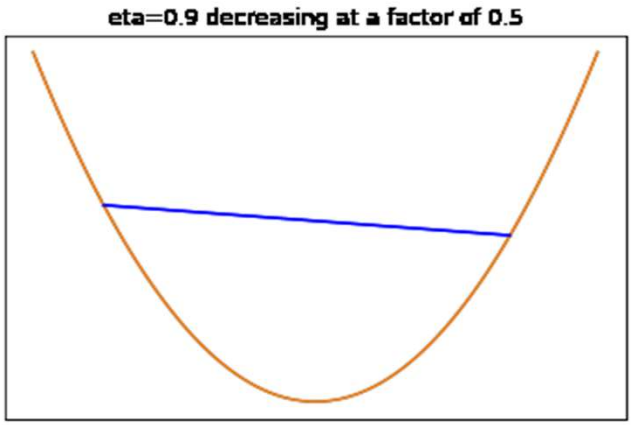
Learning Rate

- Higher learning rate may prevent from getting into a minimum
- Very high learning rates can cause divergence
  - The loss increases or plateaus too quickly
- Low learning rate can delay the learning process
  - The loss decreases too slowly and may take many epochs to reach a solution



Picture from: https://cs231n.github.io/neural-networks-3/

Learning Rate Schedule



- Reduce the learning rate over time (learning rate decay)

- Start with the typical learning rate
- Reduce the learning rate by some factor every few epochs
  - Typical values: reduce the learning rate by a half every 5 epochs, or by 10 every 20 epochs
  - Exponential decay reduces the learning rate exponentially over time
  - These numbers depend heavily on the type of problem and the model

- Reduce the learning rate by a constant whenever the validation loss stops improving
  - In TensorFlow: tf.keras.callbacks.ReduceLROnPleateau()
    - Monitor: validation loss
    - Factor: 0.1 (i.e., divide by 10)
    - Patience: 10 (how many epochs to wait before applying it)
    - Minimum learning rate: 1e-6 (when to stop)

Gradient Descent Variants

How often do we update?

Sample, Batch, Epoch

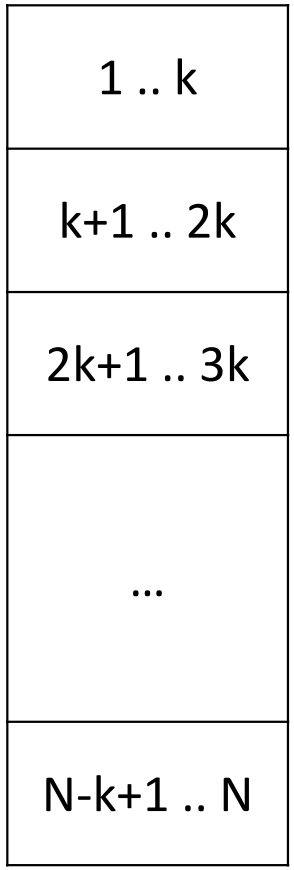
- A sample is a single row of data.
- The batch size defines the number of sample to work through, before updating the internal model parameters.
    - Depending on the batch size different learning algorithms are defined
- The number of epochs is a hyperparameter that defines the number of times the learning algorithm will work through the entire training dataset.
    - One epoch means that each sample in the training dataset was used to update the internal model parameters. An epoch is comprised of one or more batches.

An Epoch

One Epoch:

1. Randomly divide training set into $m = N/k$ batches
2. Use a batch of training samples to compute $\mathbf{J(w)}$.
3. Update w as: $\mathbf{w}^{t+1} = \mathbf{w}^{t} - \eta \frac{\partial \mathrm{J}}{\partial \mathbf{w}}$
4. Repeat 2 and 3 using different subsets all samples are used once.

- Question: What is the size of the batch?
  - 1 … k … N
  - May depend on hardware
- We repeat epochs until convergence



Batch Gradient Descent

- Training Set: e.g., ImageNet has 14M images

Approach:

  - Compute the loss $J(w)$ on the entire training, update the parameters w
  - At the next epoch, shuffle the training data, and repeat above process
- Typical batch size: Size of the Training Set

Mini-batch Gradient Descent

- It is wasteful to compute the loss over the entire set to perform a single parameter update for large datasets
  - e.g., ImageNet has 14M images
  - GD is replaced with mini-batch GD
- ***Mini-batch gradient descent***
  - Approach:
    - Compute the loss $\mathcal{L}(w)$ on a batch of images, update the parameters w, and repeat until all images are used
    - At the next epoch, shuffle the training data, and repeat above process
  - Mini-batch GD results in much faster training
  - Typical batch size: 32, 64, 128, and 256 samples
  - It works because the examples in the training data are correlated
    - i.e., the gradient from a mini-batch is a good approximation of the gradient of the entire training set

Stochastic Gradient Descent

Stochastic gradient descent

- SGD uses mini-batches that consist of a single input example
  - E.g., one image mini-batch, batch size = 1
- Although this method is very fast, it may cause significant fluctuations in the loss function
  - Therefore, it is less commonly used, and mini-batch GD is preferred
- In most DL libraries, SGD is typically a mini-batch SGD (with an option to add momentum)