Focus for this lecture

Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

- Linear Model
- Gradient Descent
- Perceptron
- Neural Network

Linear Models

Recap: The Machine Learning Framework

$$y = f(x)$$

Output **Prediction function** **Feature representation**

Training: Given a training set, estimate the prediction function, $f(\,)$, by minimizing the prediction error

Testing: Apply $f(\,)$ to unknown test sample x and predicted value(output) is y

Recap: The Machine Learning Framework

$$y = f(W, x)$$

Output **Prediction function** **Feature representation**

Training: Given a training set, estimate the prediction function, $f(\ )$, by minimizing the prediction error

Testing: Apply $f(W, .\,.)$ to unseen test sample x and predicted value(output) is y

Parameter: Primary problem is to find the parameters, W

Recap: Sample and Representation

- A sample is easy to visualize in 2D.
- $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$
- And sometime in 3D with some effort, $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$
- And we often need much larger dimensionality in practice.
- $x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_{100} \end{bmatrix}$ $\qquad x = \begin{bmatrix} x_1 \\ x_2 \\ . \\ x_d \end{bmatrix}$

Recap: Decision Boundary

- Decision boundary: Hyperplane

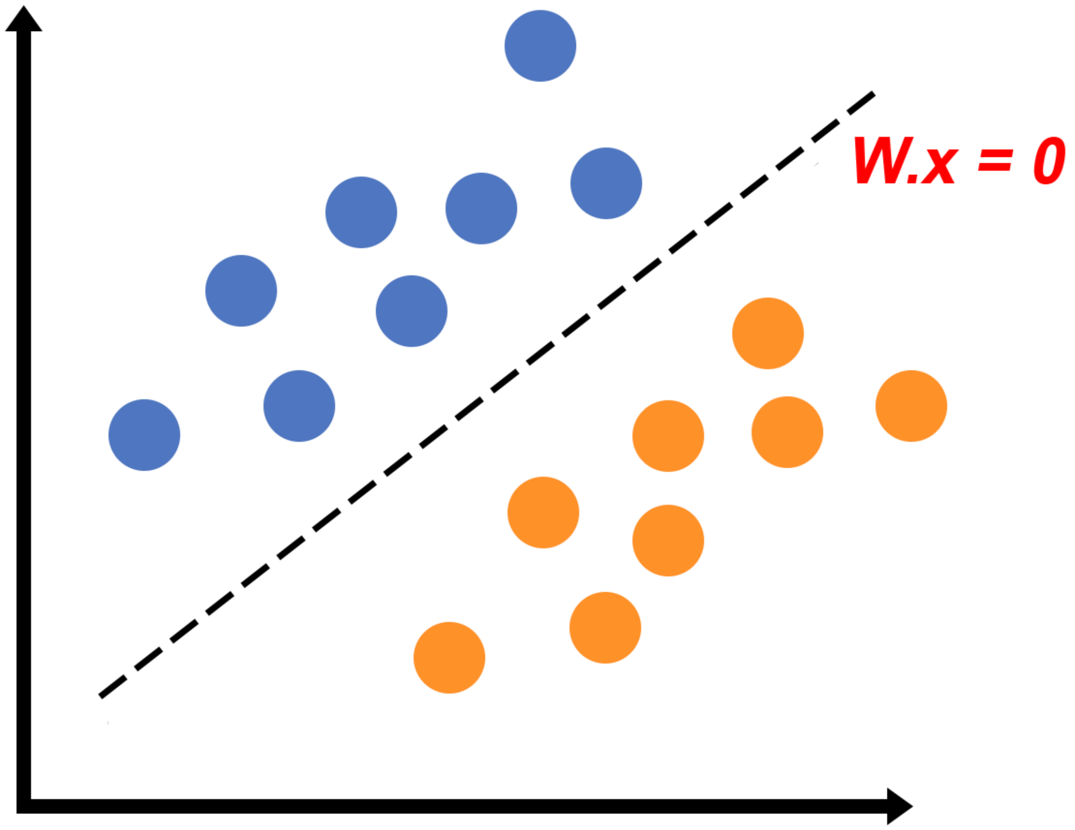
$$W \cdot x = 0$$

- Class 1 lies on the positive side

$$W \cdot x \geq 0$$

- Class 0 lies on the negative side

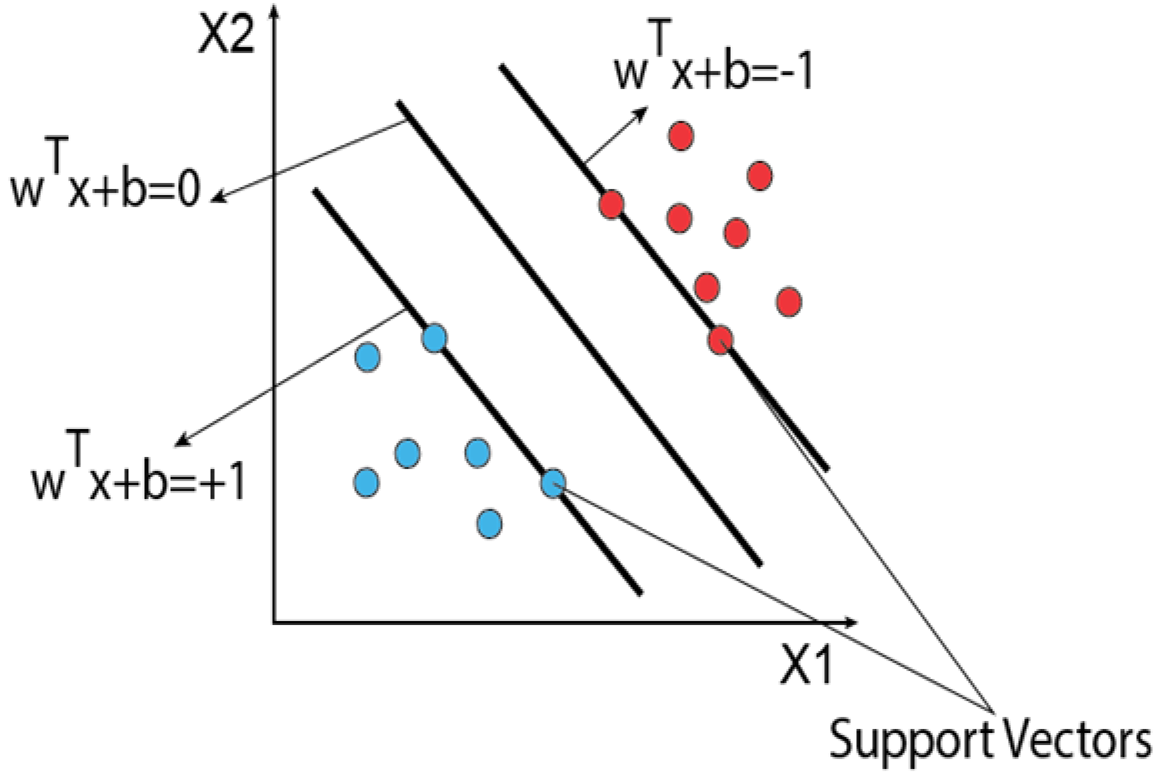
$$W \cdot x < 0$$



Trailer: Support Vector Machines

- SVMs maximize the margin around the separating hyperplane.
  - A.k.a. large margin classifiers

- The decision boundary is fully specified by a subset of training samples, the support vectors.



The Problem

- Find $\boldsymbol{w}$, given examples: $(x_i, y_i), i = 1,2, \dots, n$
- Supervised situation: output label y; is available for all training n samples
- **Objective:** predict y values for a novel input x that we have not seen before
  - Called generalization in Machine Learning
- How do we find w? **Ans: Gradient descent!**

Loss Function

- **Error/Loss (L):** A function of the difference between the actual value or label (y_i) and the predicted value ($f(w, x_i)$)

- **Total Loss:**

$$J = \sum_{i=1}^{n} L(y_i, f(w, x_i))$$

the sum of loss over n labelled training samples: (x_i, y_i)

- **Strategy:** Start with some initial values for w and bring the predicted values closer to the corresponding labels (or minimize J) by adjusting w.

Example: Loss Function

- Loss function could be the squared difference between predicted values and labels

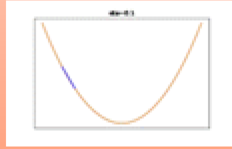
$$J = \sum_{i=1}^{n} \left(y_i, f(w, x_i)\right)^2$$

- Final objective is to minimize J, the sum of losses over each training sample

Questions?

Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

- ✓ Linear Model
- Gradient Descent

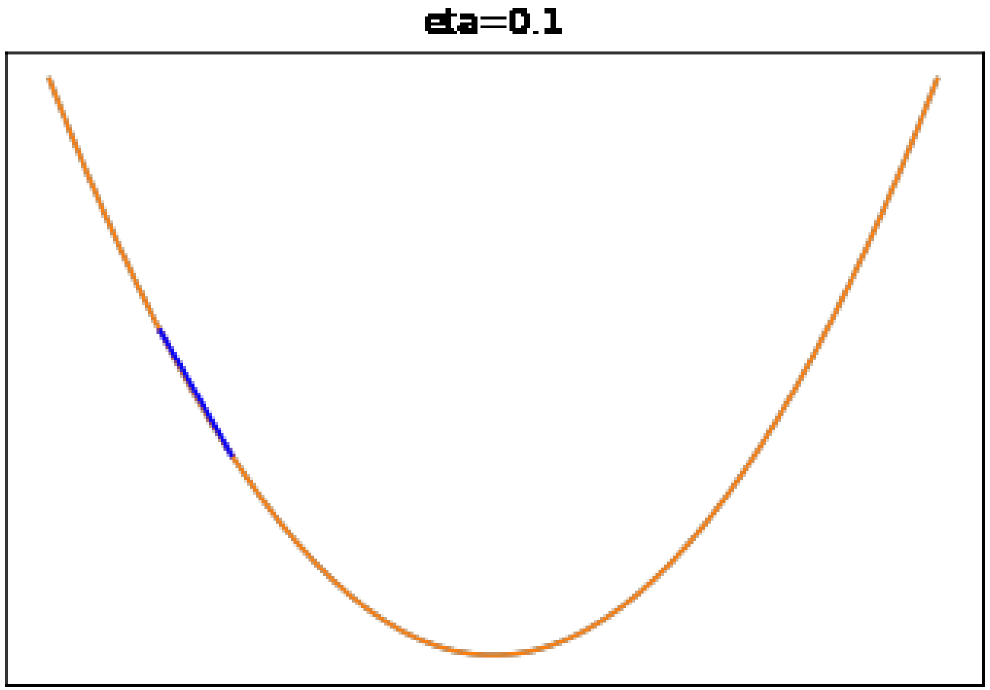


Gradient Descent

Optimization Algorithm

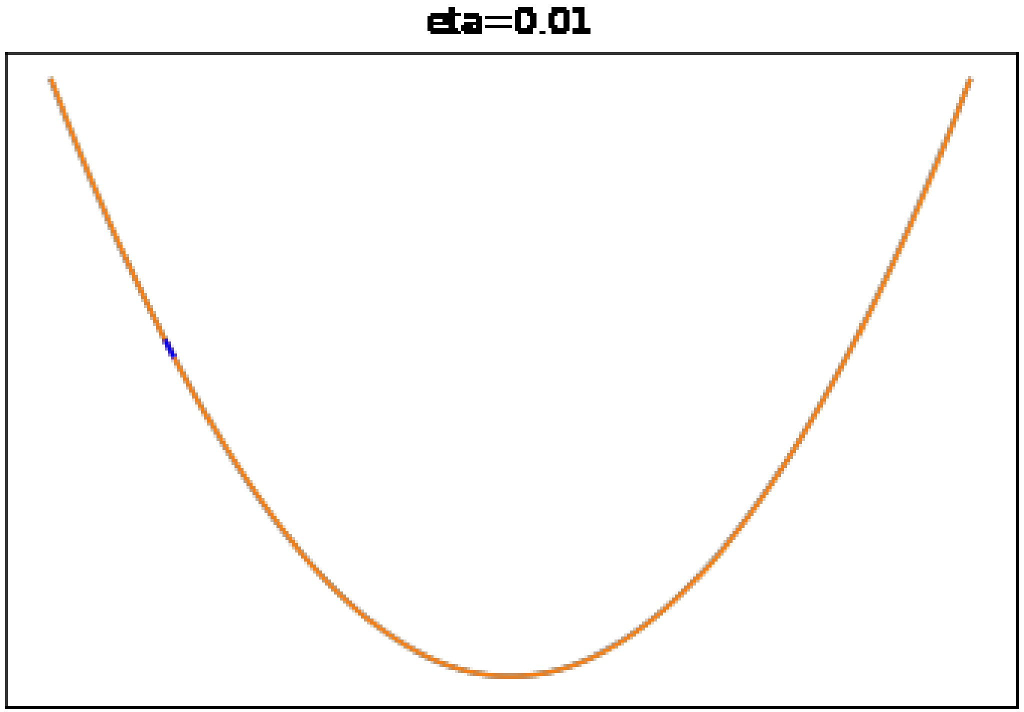
Gradient Descent

- Minimum of the function lies in the opposite direction of gradient
- Start with a guess
- Take a step against gradient:
- $w' = w - \eta \nabla J(w)$



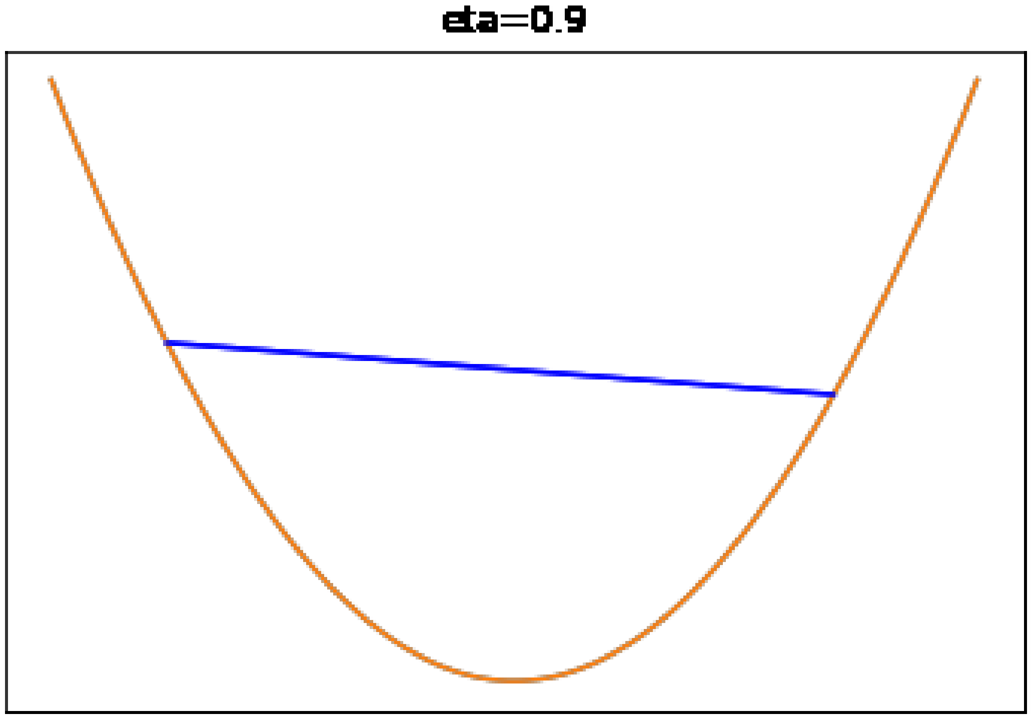
Very Small Learning Rate

- Step size too small
- Gradient descent slow to converge



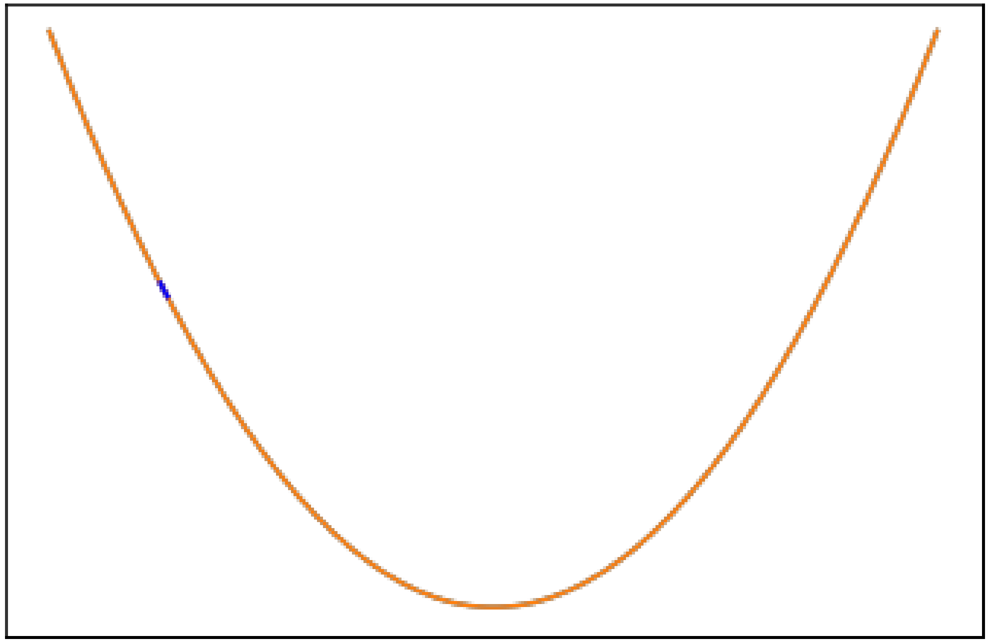
Very High Learning Rate

- Step size too large
- Gradient descent Oscillating

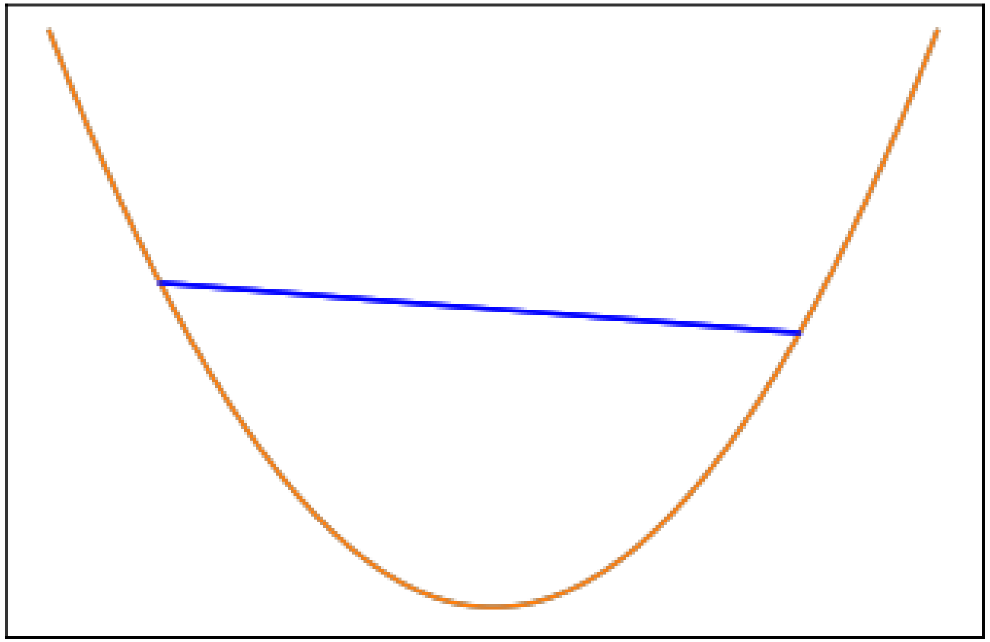


Gradient Descent: Importance of Learning rate

eta=0.01

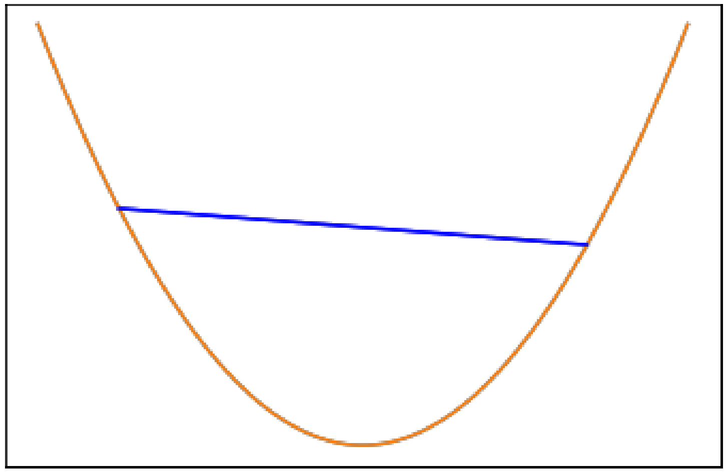


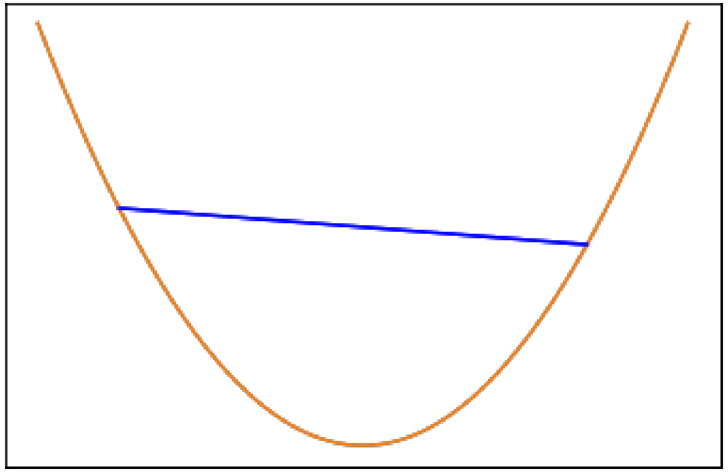
eta=0.9



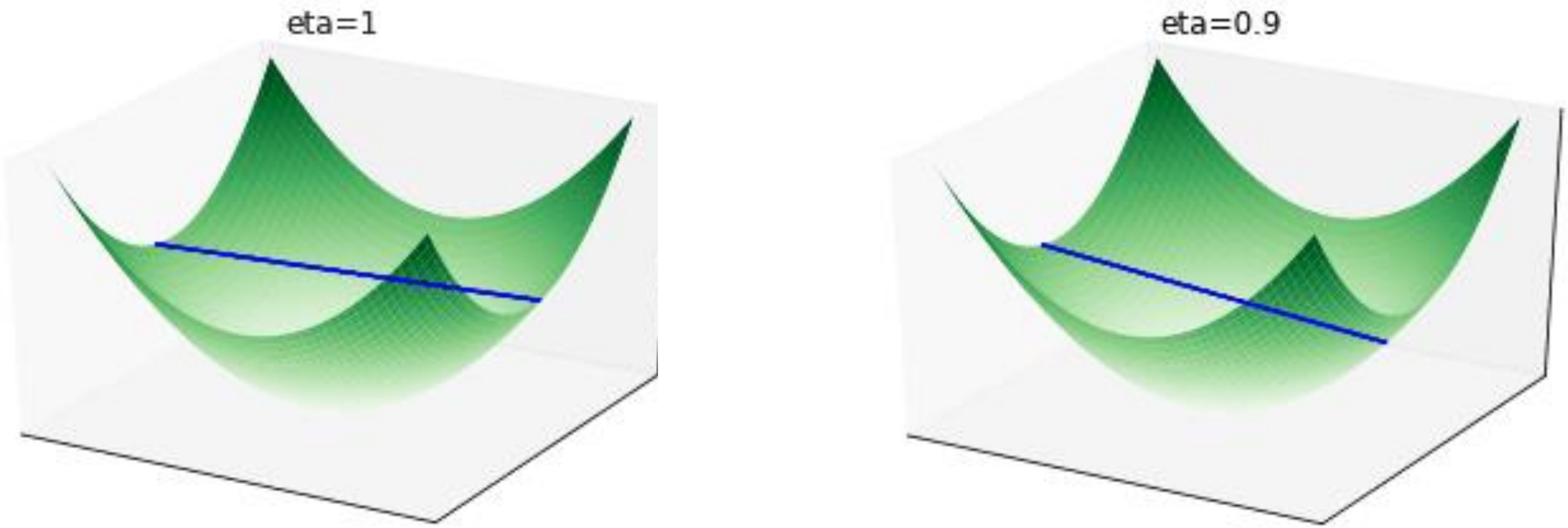
Gradient Descent - Learning rate

- Learning rate is critical
  - Start high to make rapid strides
  - Reduce for smoother convergence

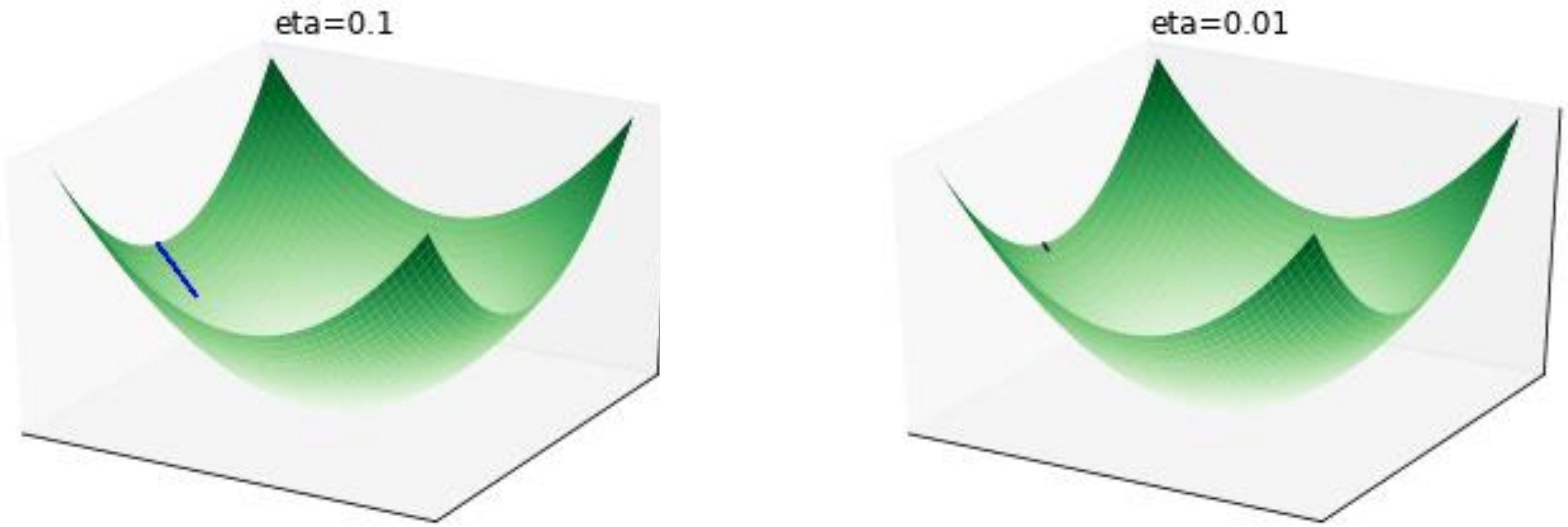




High Learning rates



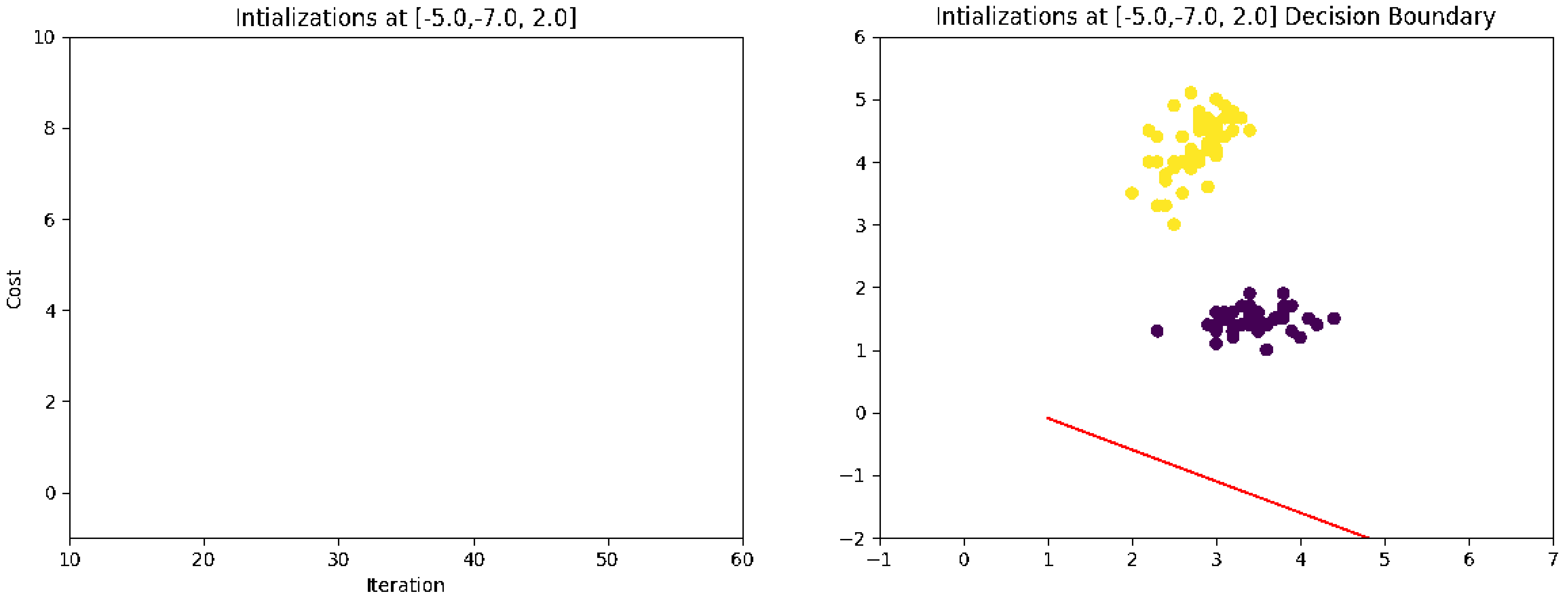
Low Learning rates



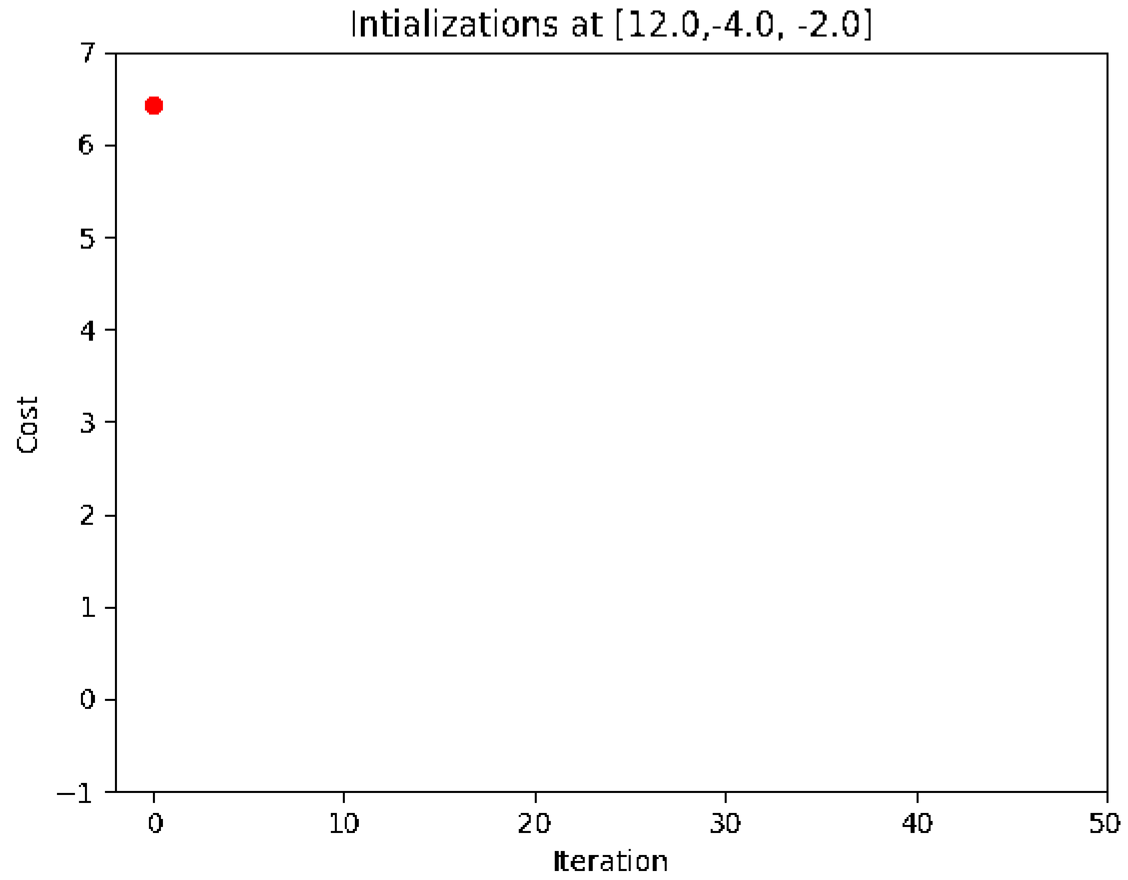
Gradient Descent - Initialization

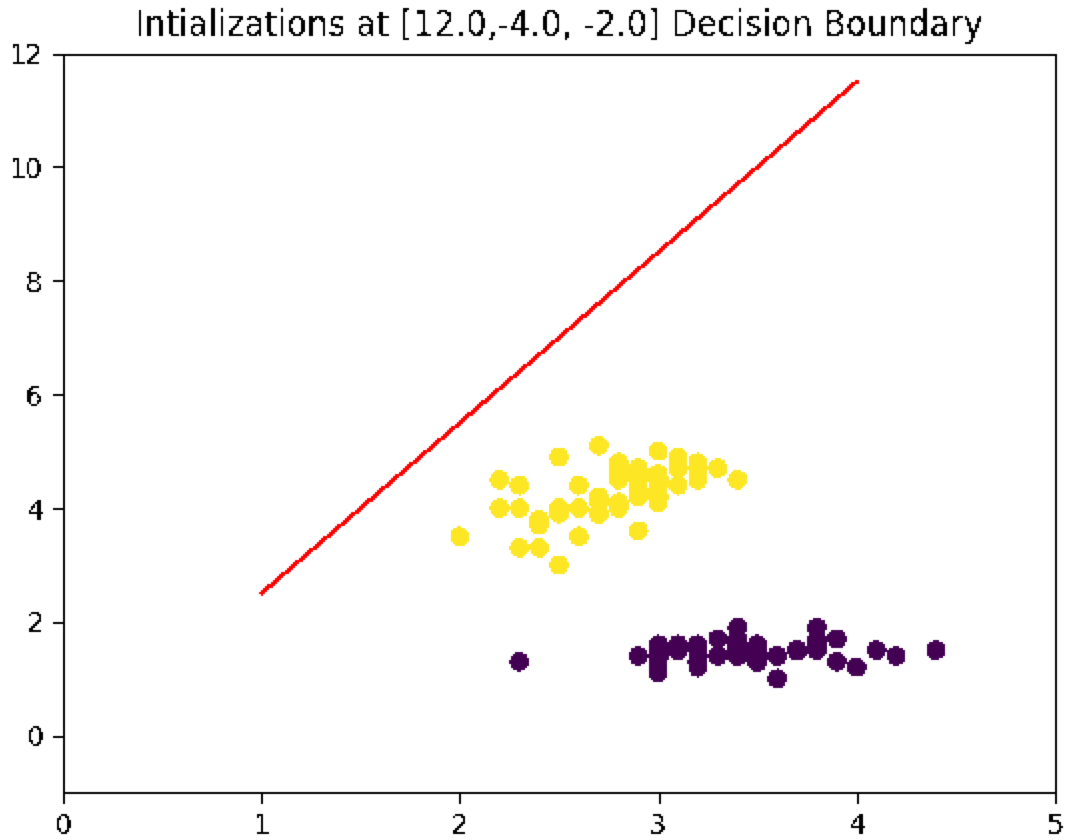
- Effects of Initialization of weights
- Good initialization: Converge fast

Gradient Descent - Initialization



Gradient Descent - Initialization



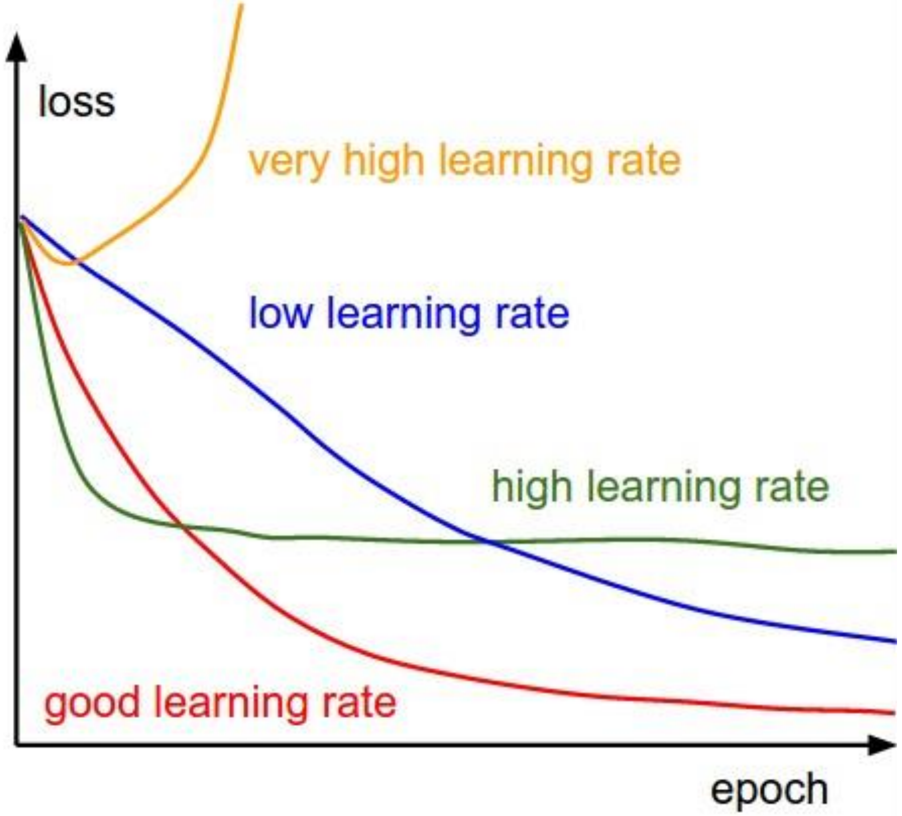


Scale varied for better visibility

Fine Points

- When do we stop the iterations?
  - When the gradient value is too low ($< \boldsymbol{\varepsilon}$)
  - When the change in loss function is too small
- How about the step size?
  - Ensure smooth convergence

Gradient Descent - Learning rate



http://cs231n.github.io/neural-networks-3/#baby

Batch Gradient Descent

Pseudo Code

```
model = initialization(...)
train_data = load_training_data()
for i in 1...n:
    error = 0
    for X, y in train_data :
        prediction = predict(X, model)
        error += calculate_error(y,prediction)
    gradient  = differentiate(error)
    model = update_model(model, gradient)
```

Mini Batch Gradient Descent

Pseudo Code

```
model = initialization(...)
train_data = load_training_data()
Tb1, Tb2, Tb3, ...............,Tbm = split_training_batches(train_data)
for i in 1...n:
        error = 0
        for Tb in Tb1, Tb2, Tb3, ...............,Tbn :
                for X, y in Tb :
                        predictions = predict(X, model)
                        error += calculate_error(y,predictions)
                gradient  = differentiate(error)
                model = update_model(model, gradient)
```

Stochastic Gradient Descent

Pseudo Code

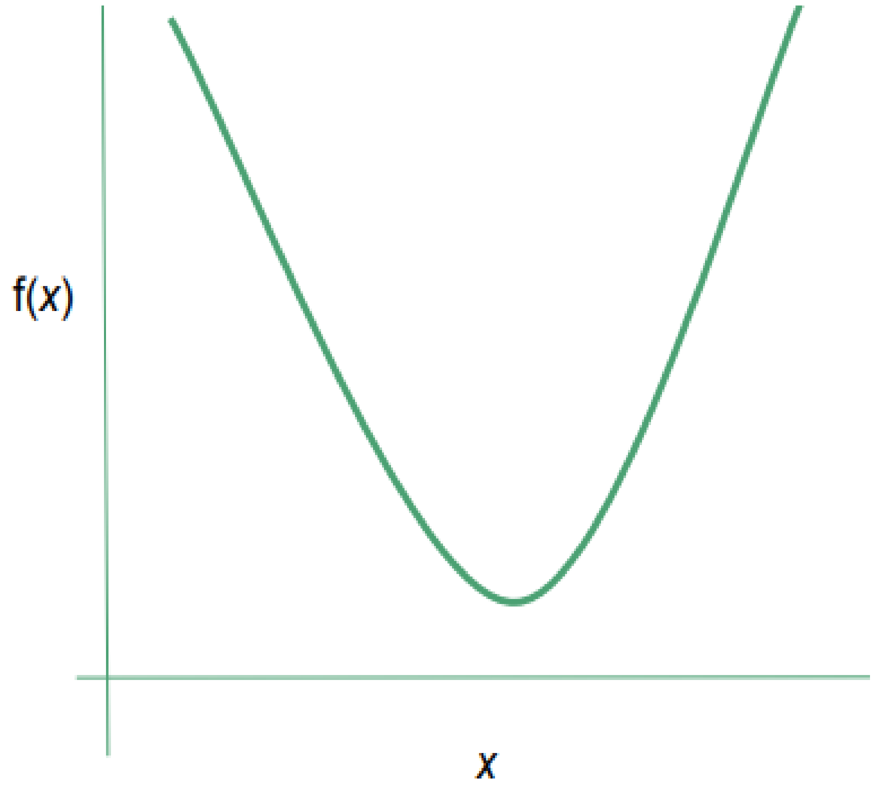
```
model = initialization(...)
train_data = load_training_data()
for i in 1...n:
    train_data = random_shuffle(train_data)
    error = 0
    for X, y in train_data :
        prediction = predict(X, model)
        error = calculate_error(y,prediction)
        gradient  = differentiate(error)
        model = update_model(model, gradient)
```

Trend

Use Stochastic Mini Batch Gradient Descent

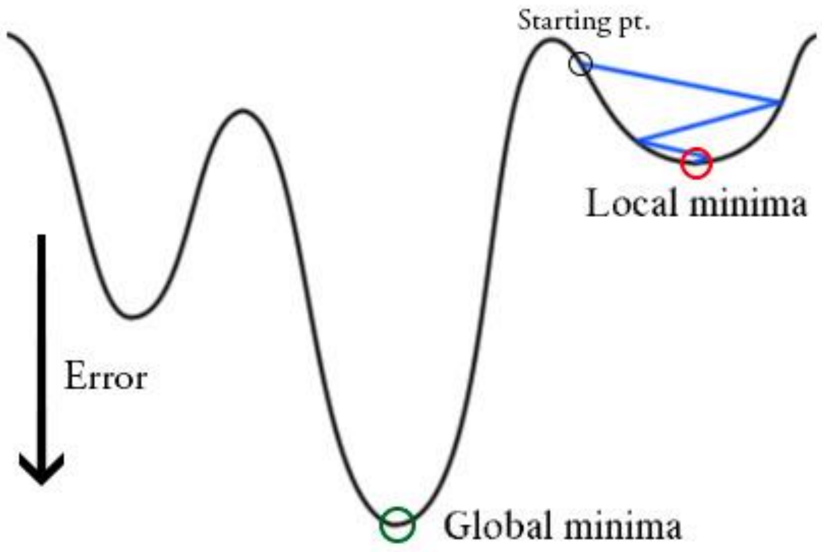
Comments: Convex Objective

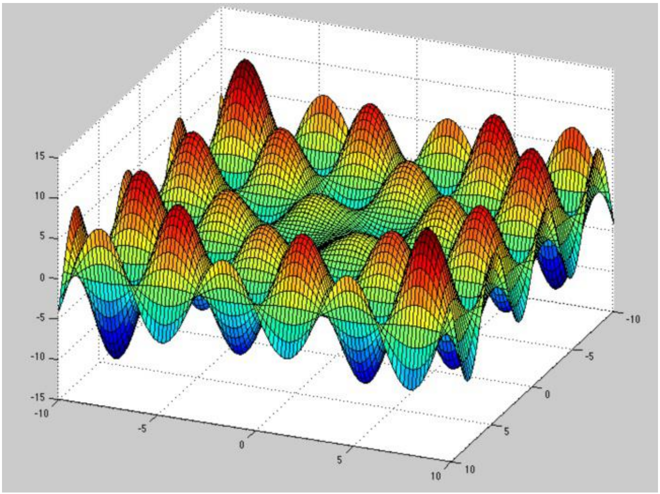
- Local minima = global minima
- Strong theoretical guarantees
- Only one optimal solution, easier in intuition
- Takes lesser time to converge to optima



Comments: Non - Convex Objective

- Multiple local minima
- Multiple local minima != Global minima
- It can take a lot of time to identify whether the problem has no solution or if the solution is global
- Takes longer to obtain global optima

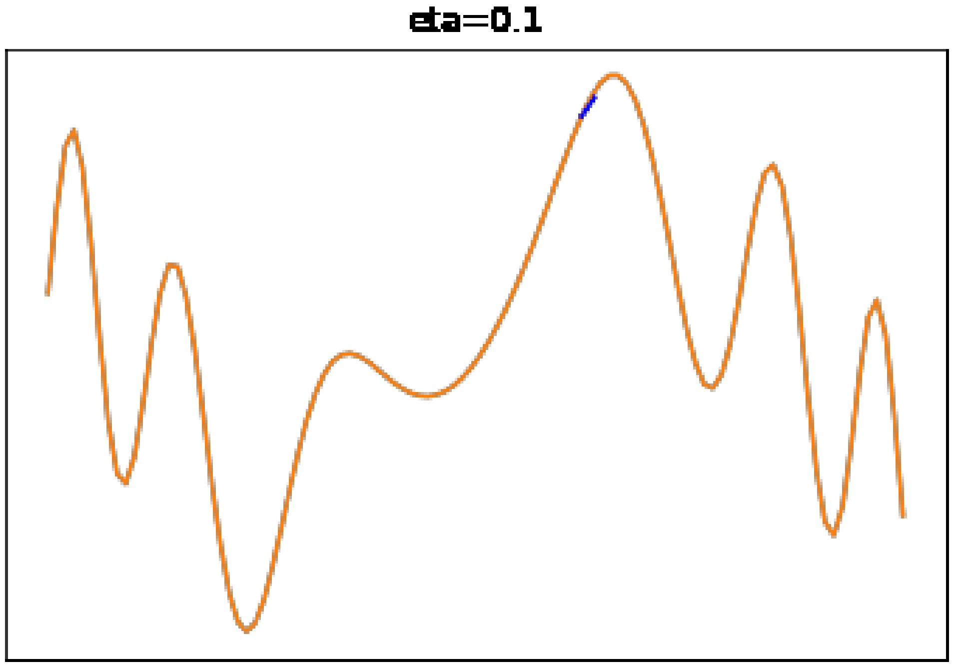




Comments: Non - Convex Objective

- Typically viewed as highly non-convex function but more recently it's believed to have smoother surfaces but with many saddle regions!

Visualization of loss function



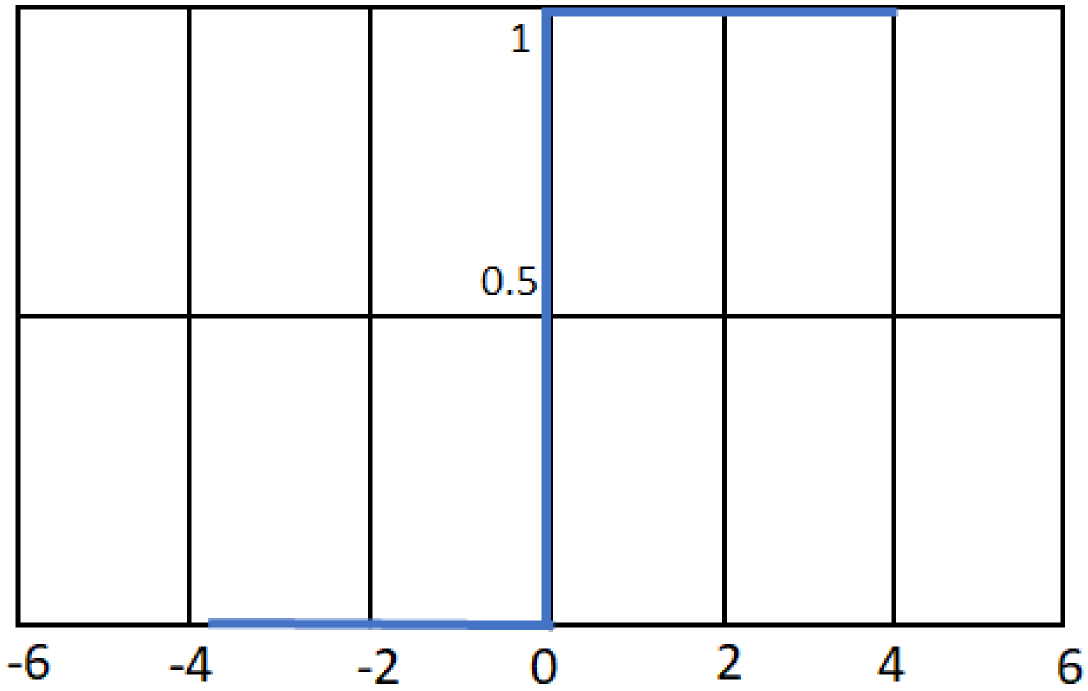
Comments: Step Function

For binary classification,

$$y = f(w, x)$$

$$y = \begin{cases} 1 & if\ f(w,x) \geq 0 \\ 0 & if\ f(w,x) < 0 \end{cases}$$

- This is not differentiable

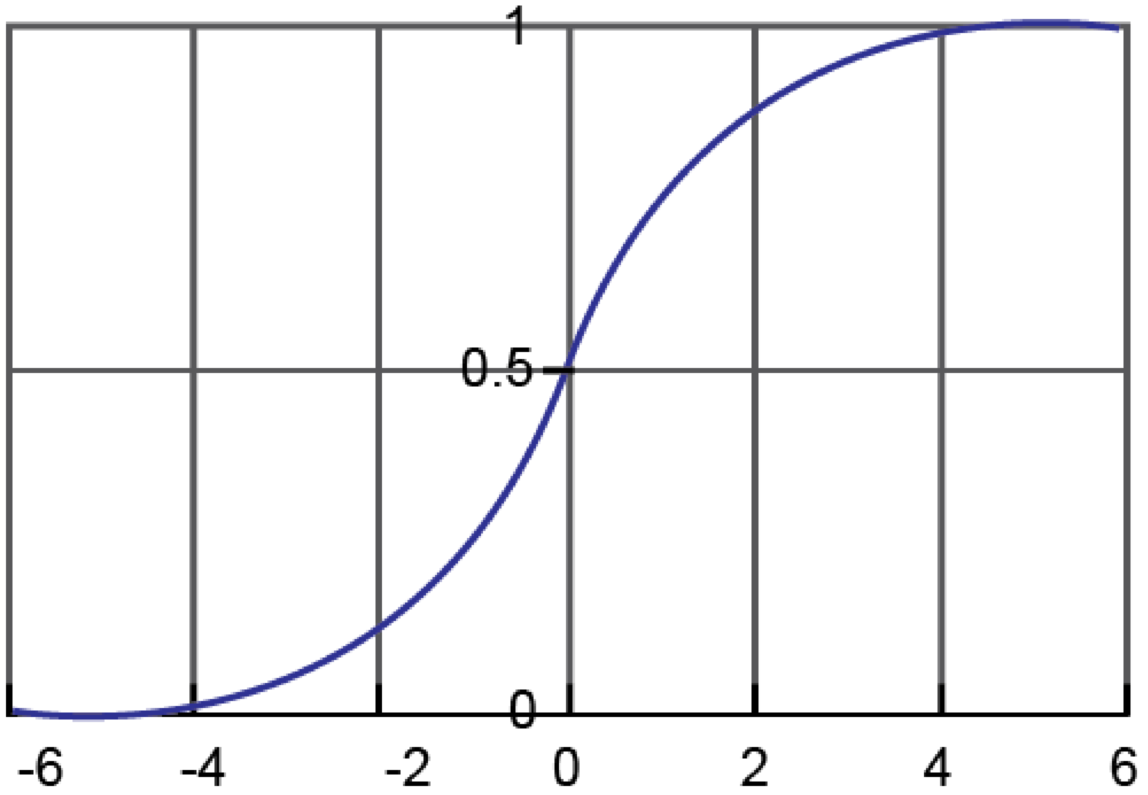


Comments: Logistic Regression

- Also called as Sigmoid Function
- To make the loss “differentiable” (see next few slides)

$$y = f(w, x) = \frac{1}{1 + e^{-w \cdot x}}$$

- (Nice probabilistic interpretation; between 0 and 1)



Comments: Logistic Function

- The logistic function
  - Probability of an outcome

$$f(z) = \frac{1}{1 + e^{z}}$$

- Has an interesting derivative form

$$f'(z) = f(z)(1 - f(z))$$

- Connect with linear classifier:

$$f(w, x) = \frac{1}{1 + e^{-w \cdot x}}$$

$$z = w \cdot x$$

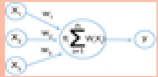
Summary

- Linear classifiers are simple to train
- Gradient descent is a powerful technique to optimize convex and non-convex objective functions

Questions?

Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

- ✓ Linear Model
- ✓ Gradient Descent
- Perceptron



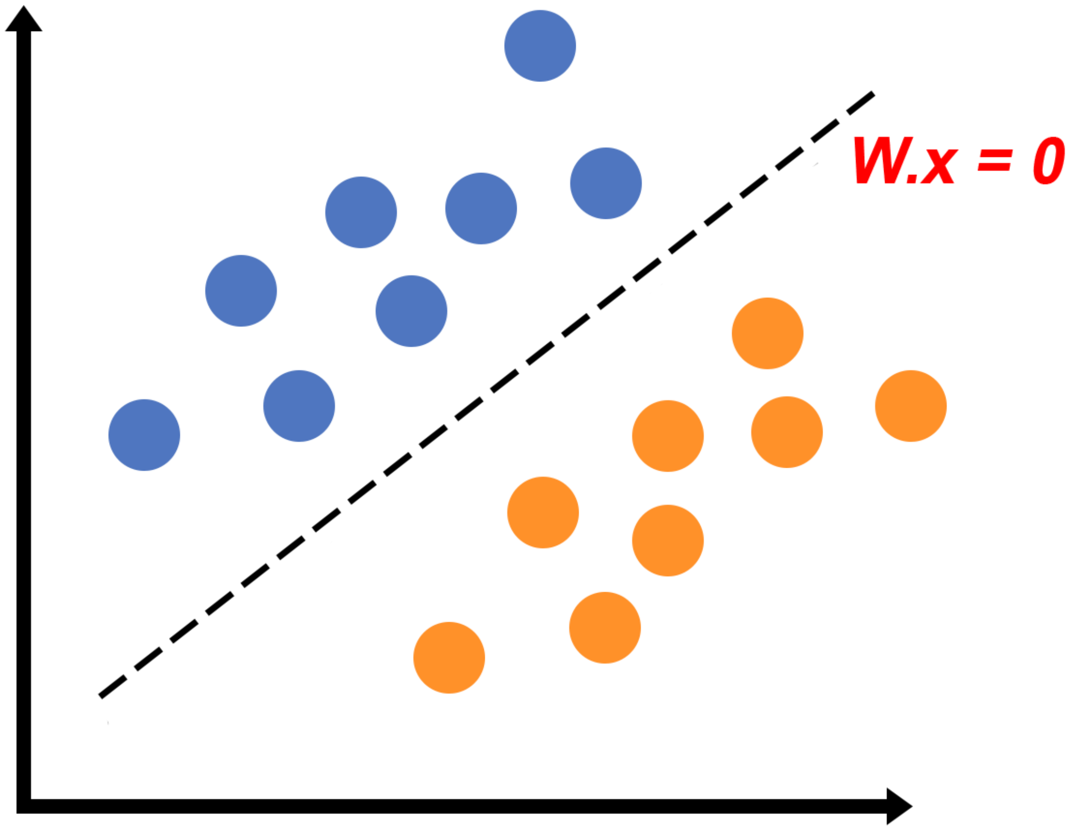
- Neural Network

Perceptron

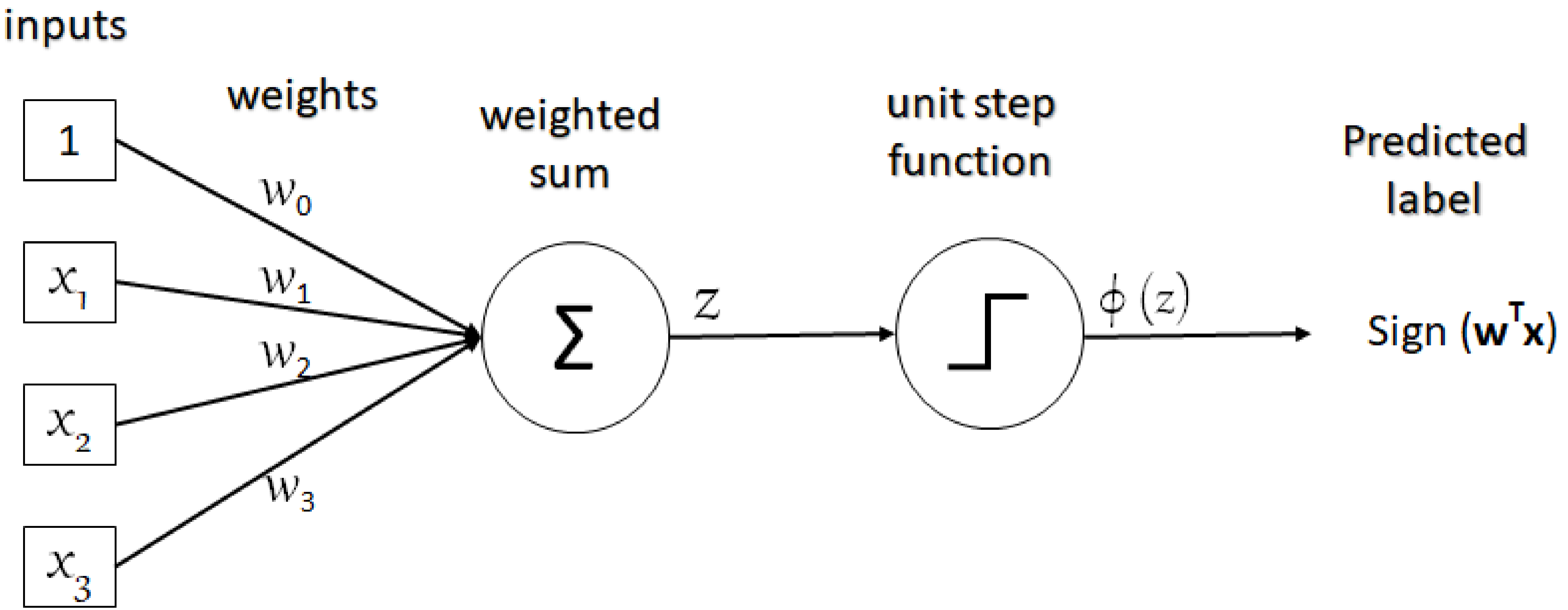
Neuron Network Perspective

Linear Classifier

It has linear partition or decision boundaries.

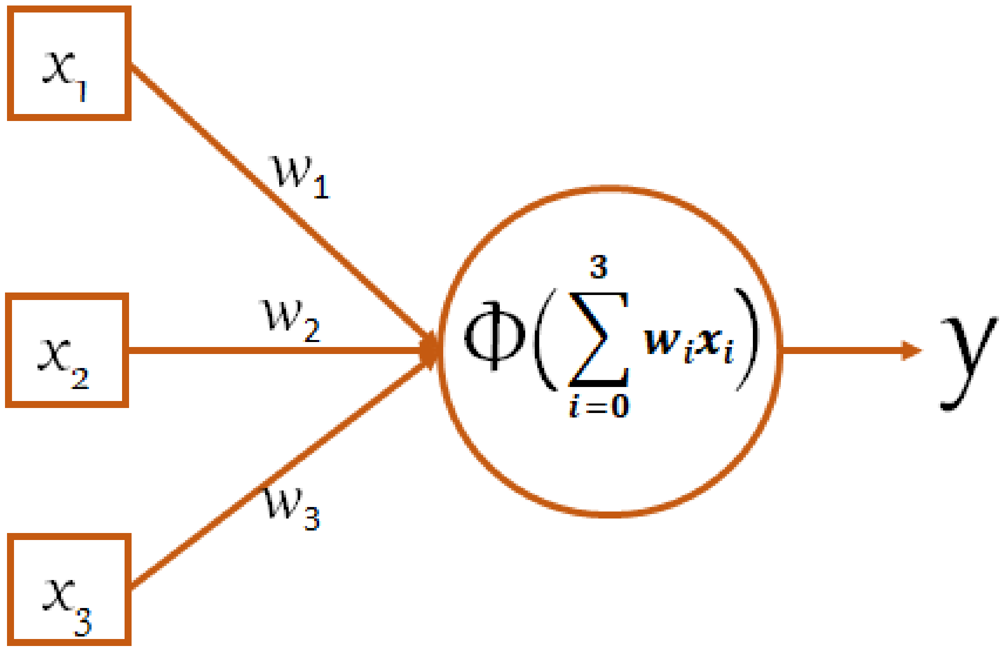


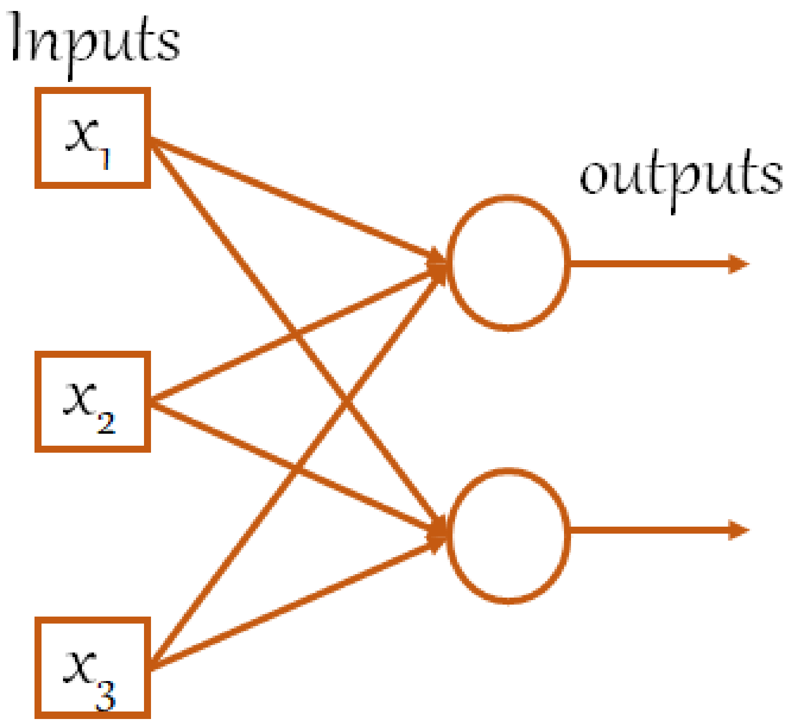
A “Neural Network” Perspective



Simple "Neuron" and Single Layer Perceptron

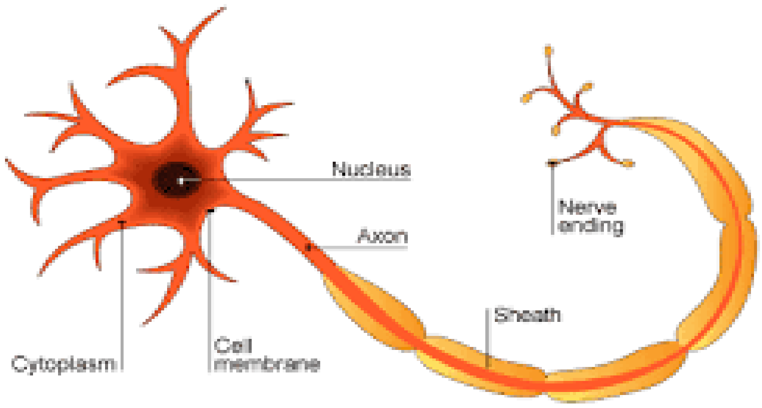
$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

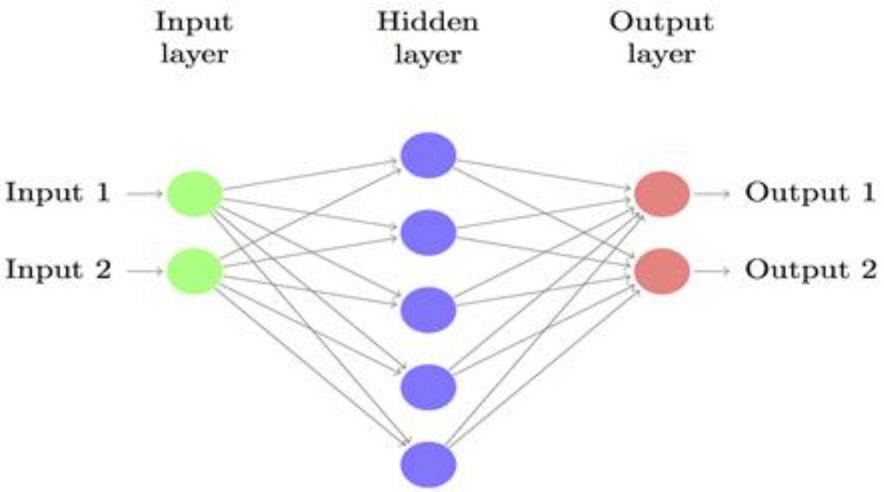




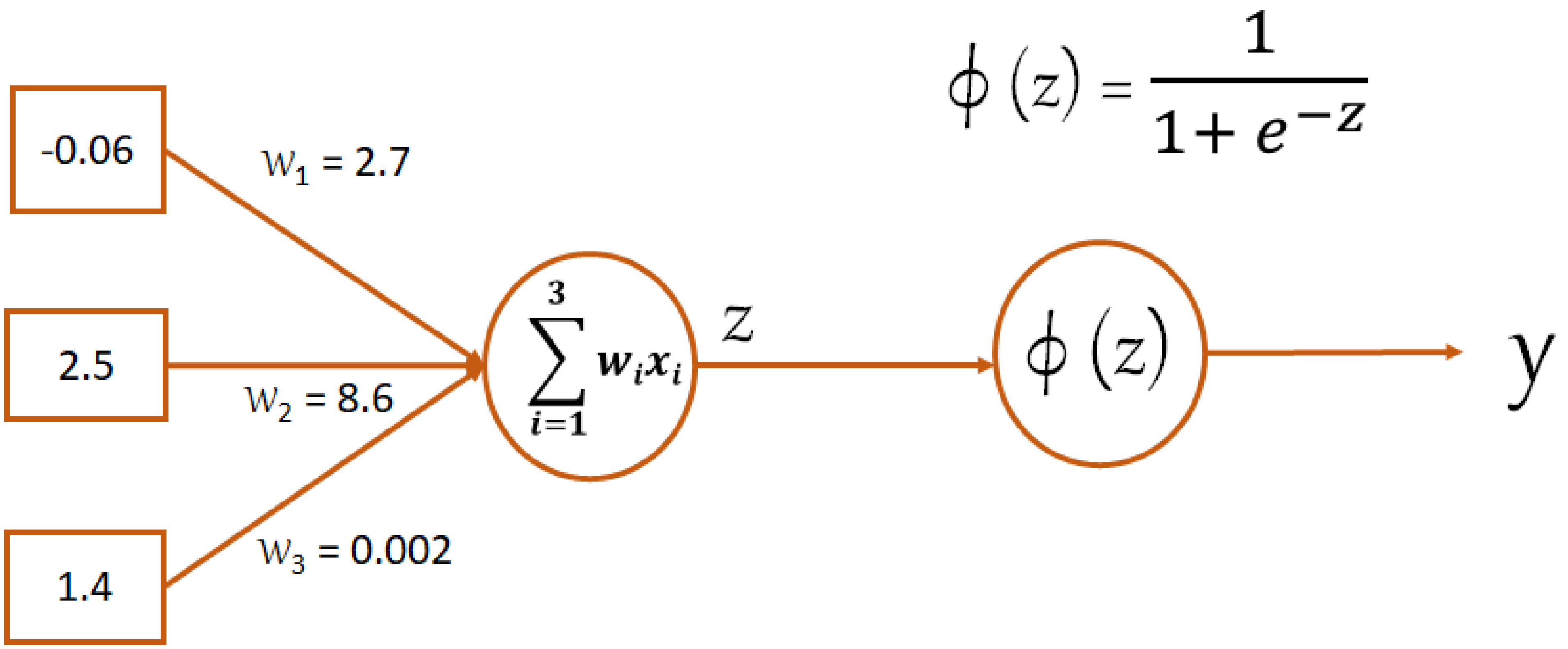
Neural Networks

- Biologically inspired networks
- Complex function approximation through composition of functions
- Can learn arbitrary Nonlinear decision boundary





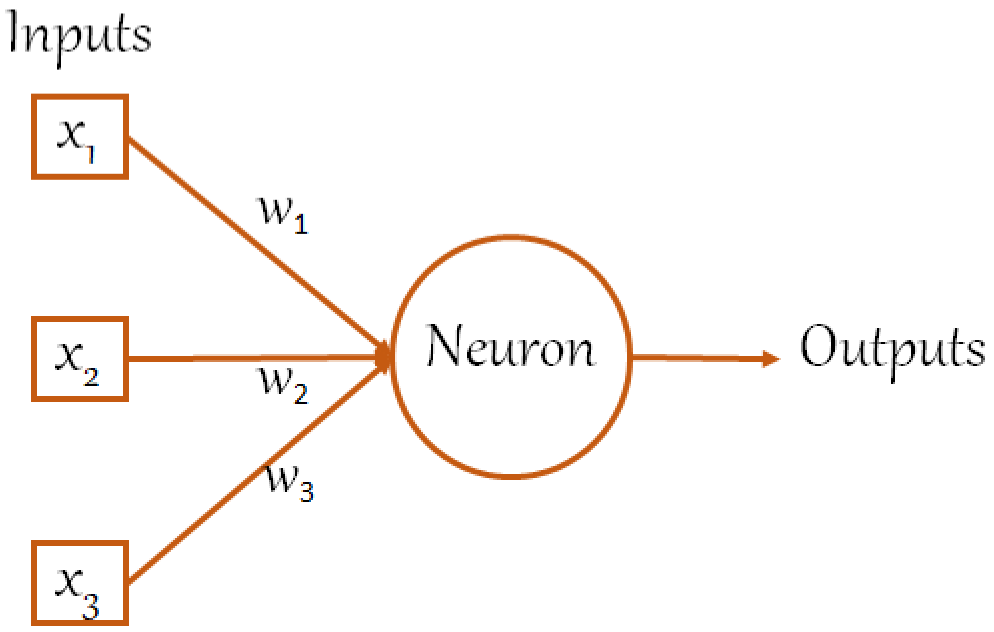
Learning in Neural Networks

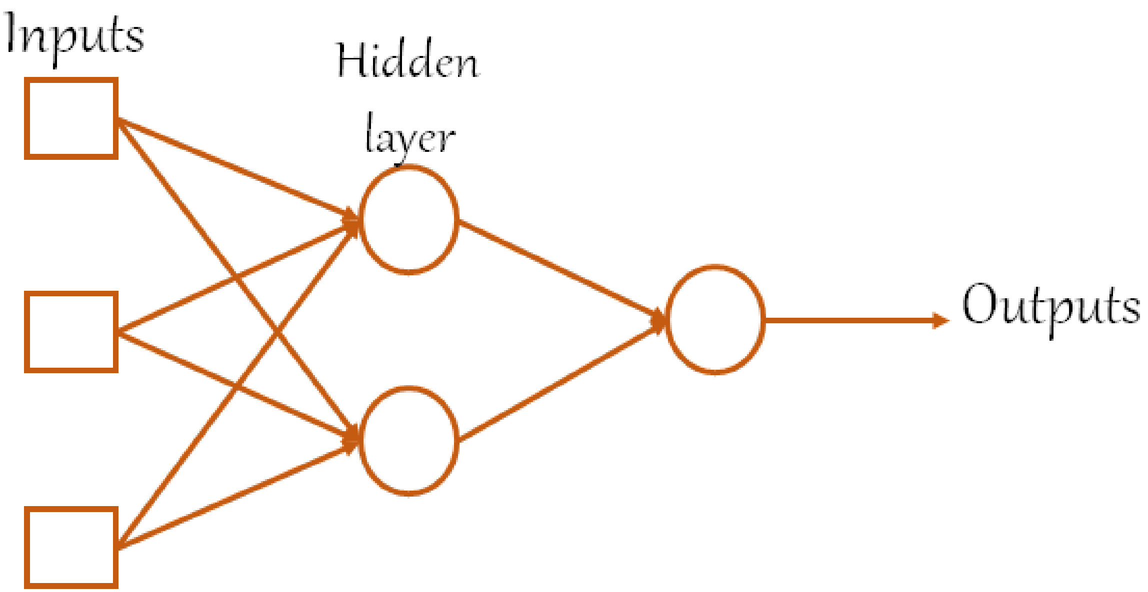


$$z = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$$

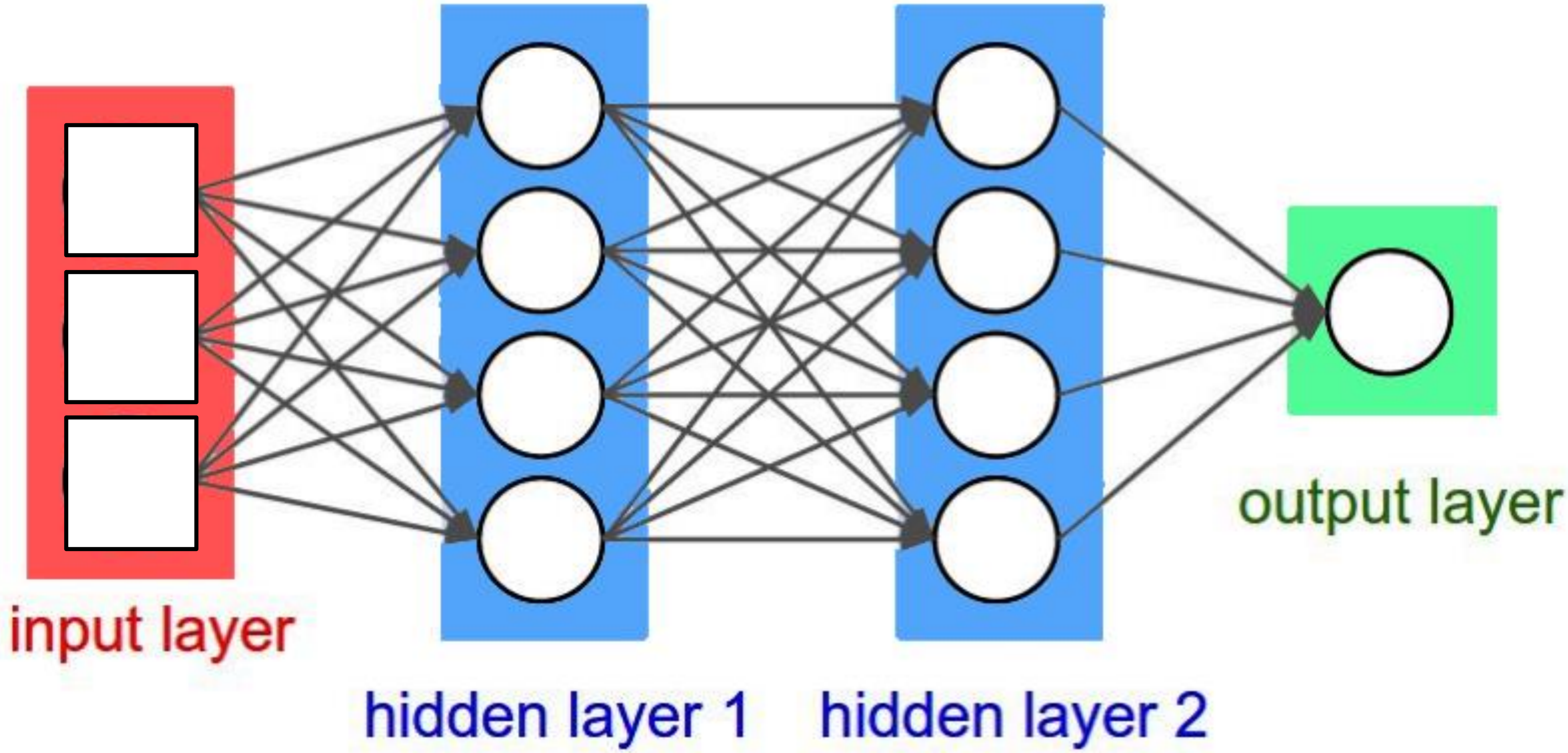
Learning = Finding the best or optimal weights

Single Layer Perceptron & Multi Layer Perceptron





Deep Neural Networks (Multi Layer Perceptron)



Summary

Acquire Raw Data → Prepare / Clean Data, Visualization → Feature Engineering → Pick Model and Hyper-params for Task → Model Training / Optimization → Evaluate Model Performance → Deploy Model

- ✓ Linear Model
- ✓ Gradient Descent
- ✓ Perceptron
- ✓ Neural Network

Thanks!!

Questions?