LECTURE 2:

REGRESSION & CLASSIFICATION

University of Washington, Seattle

Fall 2024

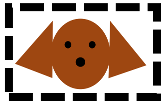
Previously in EEP 596…

Fixed model

$$x_n \rightarrow \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \rightarrow X_k$$

Learned model

$$x_n \rightarrow \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \rightarrow X_k$$

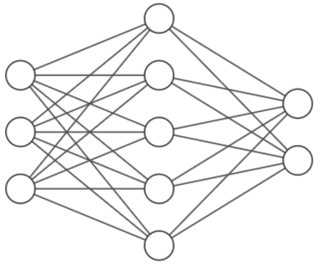


Model ☹

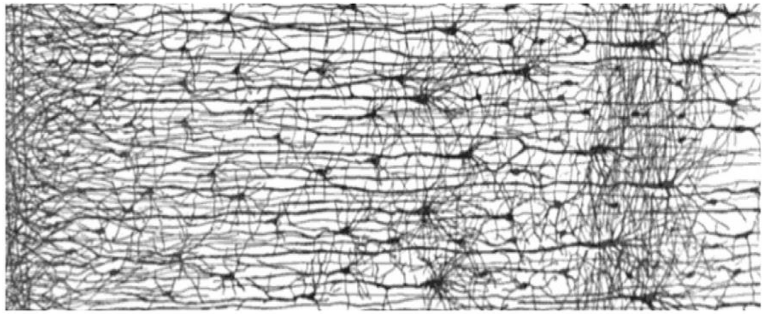
1: Café Poodle

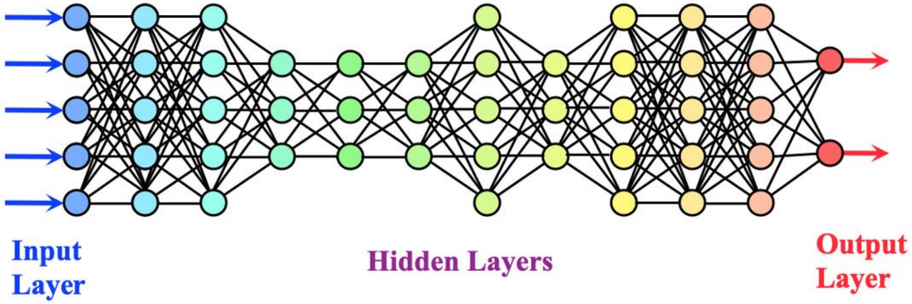
0: Not Café Poodle

Previously in EEP 596...



1: Café Poodle
0: Not Café Poodle





OUTLINE

Part 1: Regression

- Regression problem
- Linear regression
- Least square formula

Part 2: Binary Classification

- Binary Logistic Regression
- Linear vs Logistic regression
- Logistic regression and Neuron

Part 3: Training and Optimization

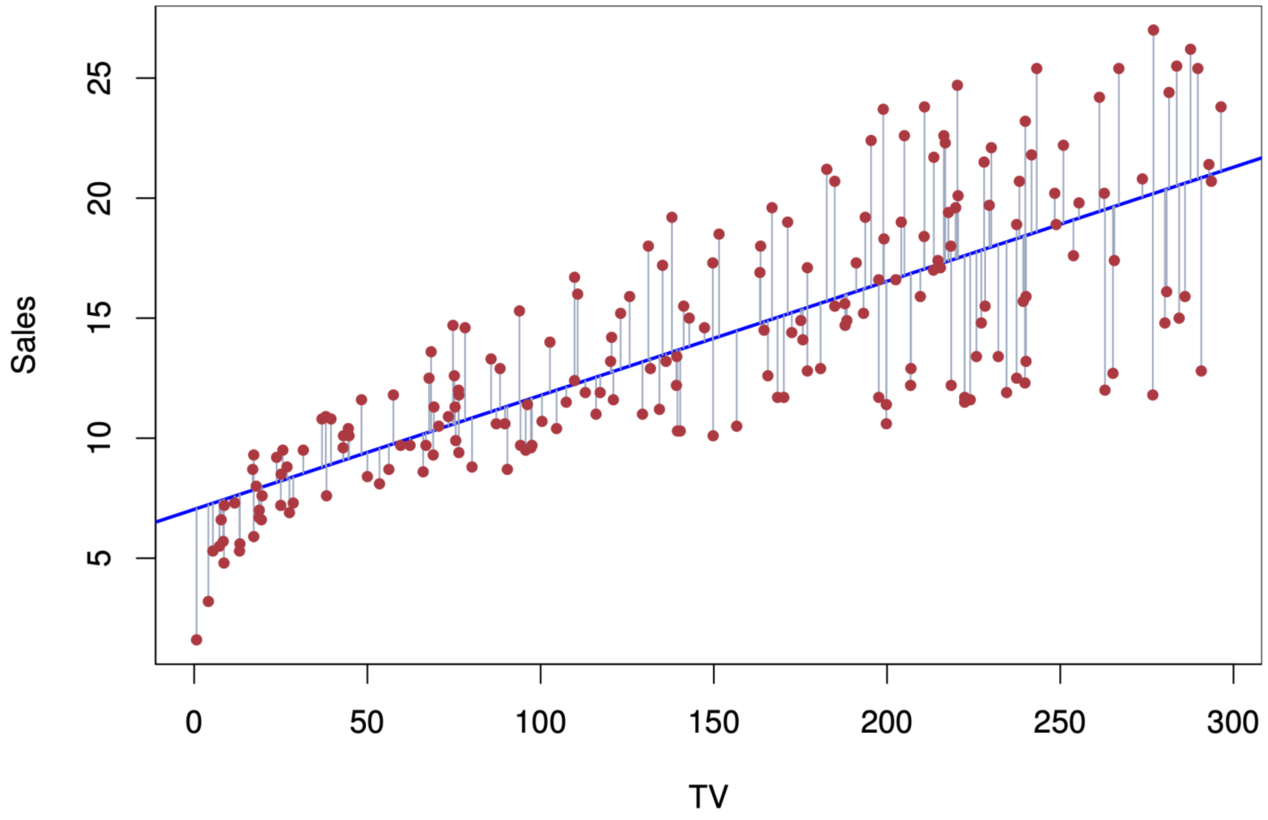
- Binary Cross Entropy Loss function
- Training Logistic Regression
- Gradient descent
- Composite Loss function and Chain rule
- Back-propagation algorithm
- Training terminologies

PART 1:

Regression

Regression problem

$$Y = f(X, W)$$



Regression problem

$\vec{x} \in \mathbb{R}^n$

$f(X, P)$

$y \in \mathbb{R}$

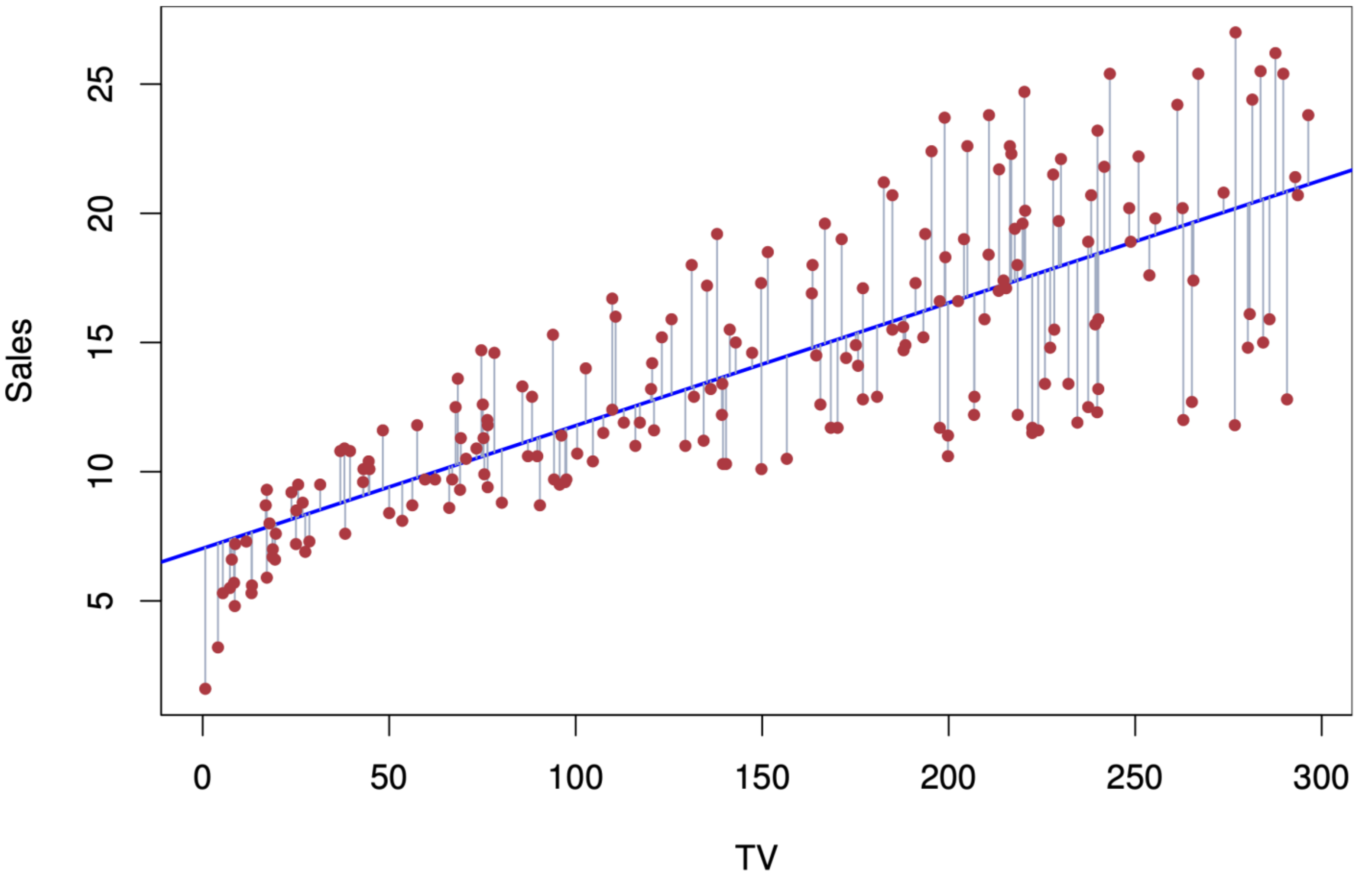
Regression model

Linear regression (polynomial fit)

$$y(x) = p_0 + p_1 x + \cdots + p_m x^m$$

$$y = \vec{p} \cdot \vec{x}$$

Polynomial fit – Linear regression



$$x = input\ data$$

$$y = output\ data$$

$$\hat{y}(x) = p_0 + p_1 x + \cdots + p_m x^m$$

Goal: Minimize $e = y - \hat{y}$

Polynomial fit – Linear regression

$\hat{y}(x) = p_0 + p_1 x + \cdots + p_m x^m$ Goal: Minimize $e = y - \hat{y}$

Polynomial fit – Linear regression

$\hat{y}(x) = p_0 + p_1 x + \cdots + p_m x^m$ Goal: Minimize $e = y - \hat{y}$

$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$

n-given points

$$X = \begin{array}{cc} \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{pmatrix} & \begin{matrix} \vec{\tilde{x}}_0 \\ \\ \\ \vec{\tilde{x}}_n \end{matrix} \end{array} \qquad P = \begin{pmatrix} p_0 \\ p_1 \\ . \\ . \\ . \\ p_m \end{pmatrix}$$

Polynomial fit – Linear regression

$\hat{y}(x) = p_0 + p_1 x + \cdots + p_m x^m$ Goal: Minimize $e = y - \hat{y}$

$(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$

n-given points

$$X = \begin{array}{cc} \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{pmatrix} & \begin{matrix} \vec{\tilde{x}}_0 \\ \\ \\ \vec{\tilde{x}}_n \end{matrix} \end{array} \qquad P = \begin{pmatrix} p_0 \\ p_1 \\ \cdot \\ \cdot \\ \cdot \\ p_n \end{pmatrix}$$

Fit ("Learn") the model:

Least squares error

$$\text{Cost } J = E_2 = \sum_{i=1}^{n} (\overbrace{\vec{p} \cdot \vec{\tilde{x}}_i}^{\hat{y}_i} - y_i)^2$$

Total error

Polynomial fit – Linear regression

Find the parameters minimizing the error

$$J = E_2 = \sum_{i=1}^{n} (\vec{p} \cdot \vec{\tilde{x}}_i - y_i)^2$$

$$\vec{p}^* = \arg\min_{\vec{p}} J(\vec{p})$$

Polynomial fit – Linear regression

Find the parameters minimizing the error

$$J = E_2 = \sum_{i=1}^{n} (\vec{p} \cdot \vec{\tilde{x}}_i - y_i)^2$$

$$\vec{p}^* = \arg\min_{\vec{p}} J(\vec{p})$$

$$\forall i : \quad \frac{\partial J}{\partial p_j} = 0 \qquad \frac{\partial J}{\partial p_j} = \sum_{i=1}^{n} 2(\vec{p} \cdot \vec{\tilde{x}}_i - y_i)\vec{\tilde{x}}_i$$

$$\vec{p}^* = \left(X^\top X\right)^{-1} X^\top \vec{y}$$

Polynomial fit – Linear regression

Find the parameters minimizing the error

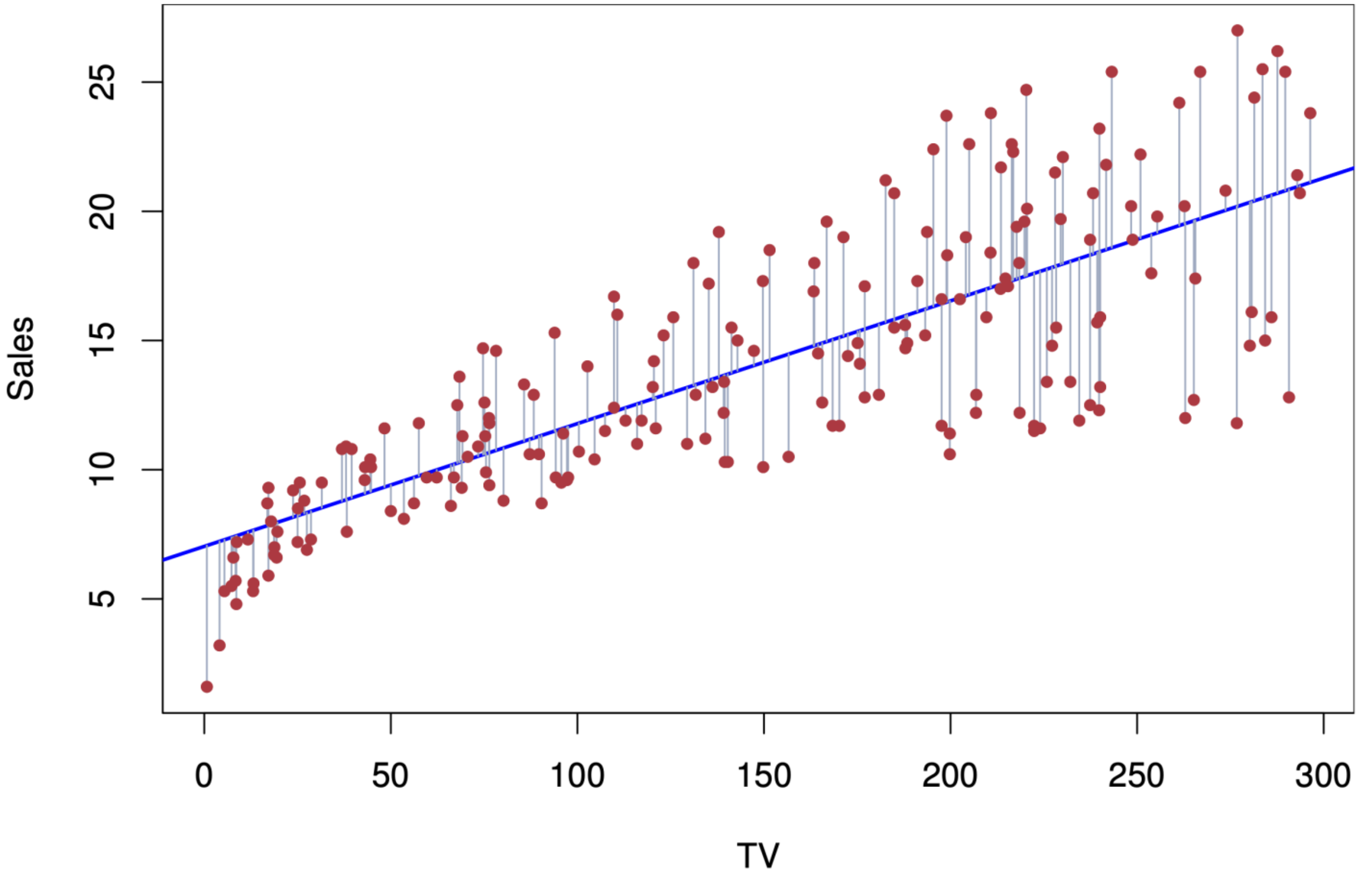
$$J = E_2 = \sum_{i=1}^{n} (\vec{p} \cdot \vec{\tilde{x}}_i - y_i)^2$$

$$\vec{p}^* = \underset{\vec{p}}{\arg\min}\, J(\vec{p})$$

$$\forall i : \quad \frac{\partial J}{\partial p_j} = 0 \qquad \frac{\partial J}{\partial p_j} = \sum_{i=1}^{n} 2(\vec{p} \cdot \vec{\tilde{x}}_i - y_i)\vec{\tilde{x}}_i$$

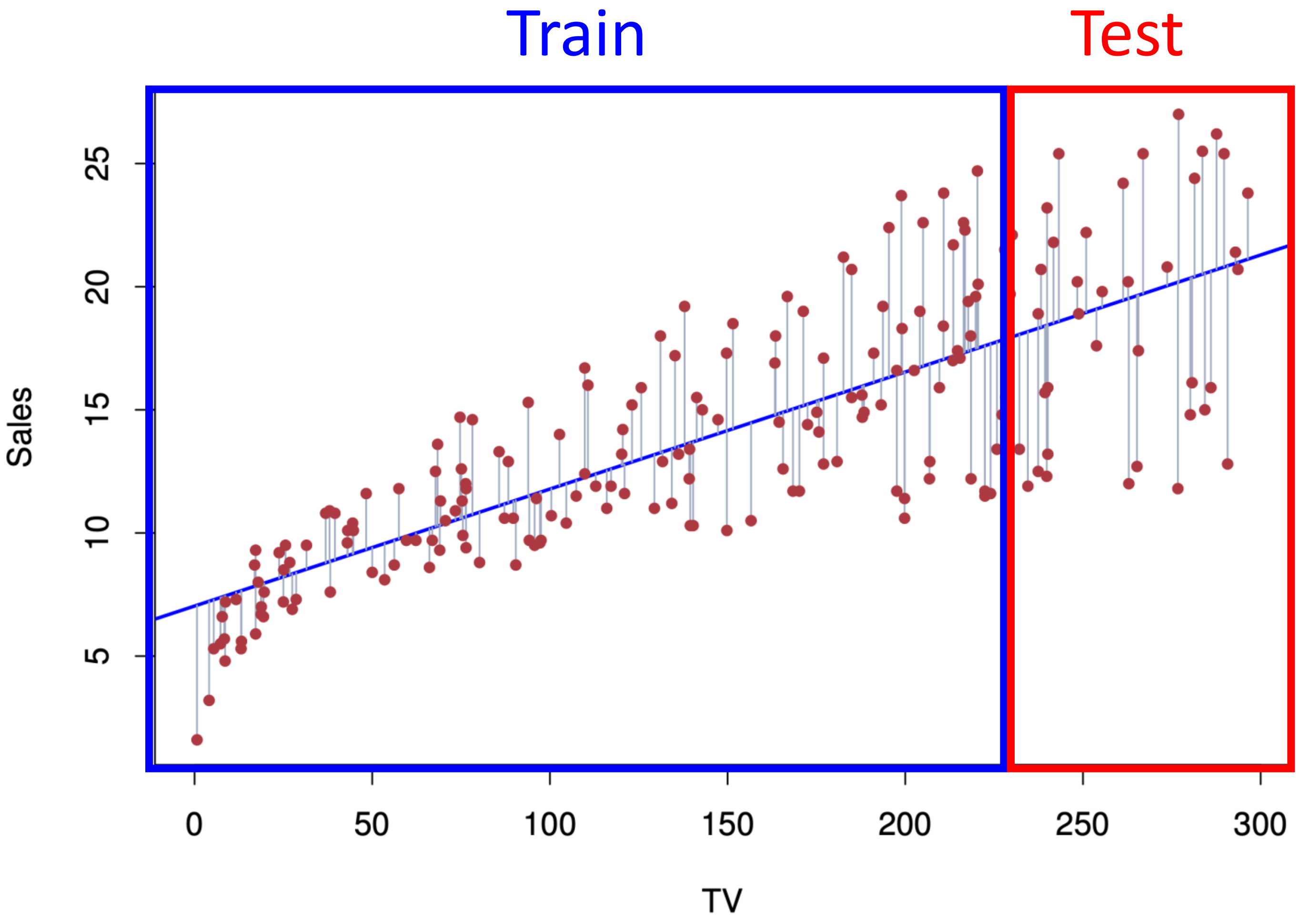
$$\vec{p}^* = \left(X^\intercal X\right)^{-1} X^\intercal \vec{y}$$ Linear least square formula

Linear regression is not perfect



$$\hat{y}(x) = p_0 + p_1 x + \cdots + p_m x^m$$

Linear regression is not perfect



$$\hat{y}(x) = p_0 + p_1 x + \cdots + p_m x^m$$

Model might perform worse in testing in the presence of outliers

Improving Linear Regression

$$W^* = \arg\min_{W} L(X, W)$$

$$J_2 = \frac{1}{n}\sum_{i=1}^{n}(f(\tilde{x}_i, W) - y_i)^2$$ Mean squared error

$$J_1 = \frac{1}{n}\sum_{i=1}^{n}|f(\tilde{x}_i, W) - y_i|$$ Mean absolute error

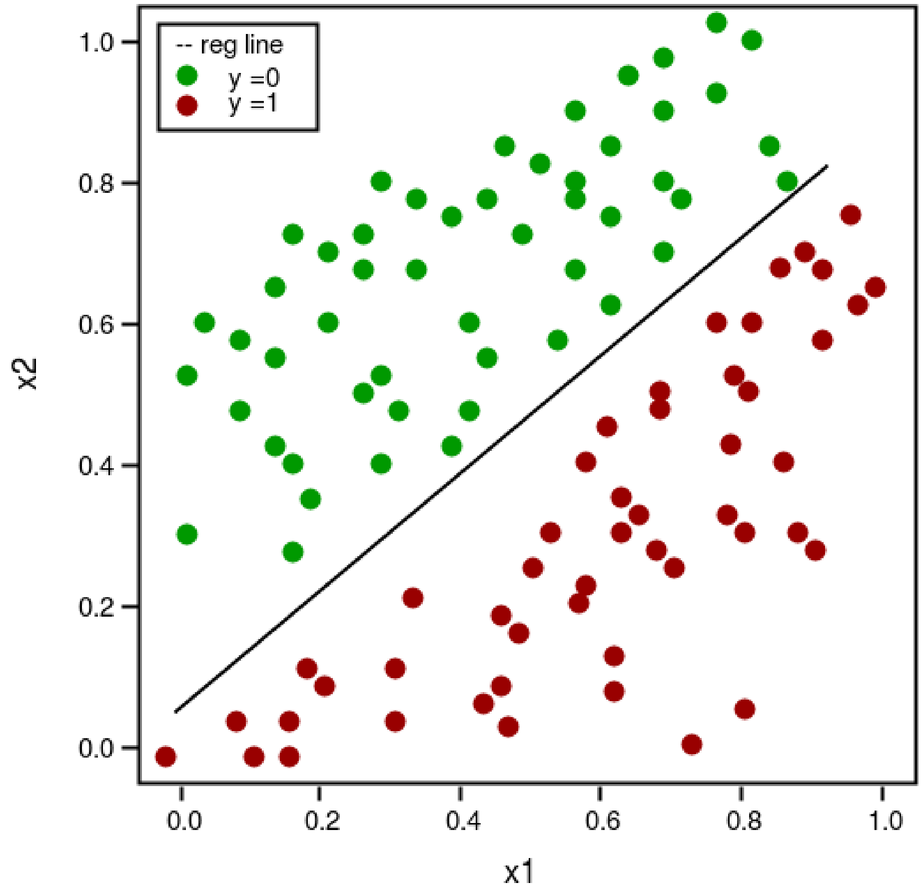
$$J_\infty = \max_{1<i<n}|f(\tilde{x}_i, W) - y_i|$$ Max error

PART 2:

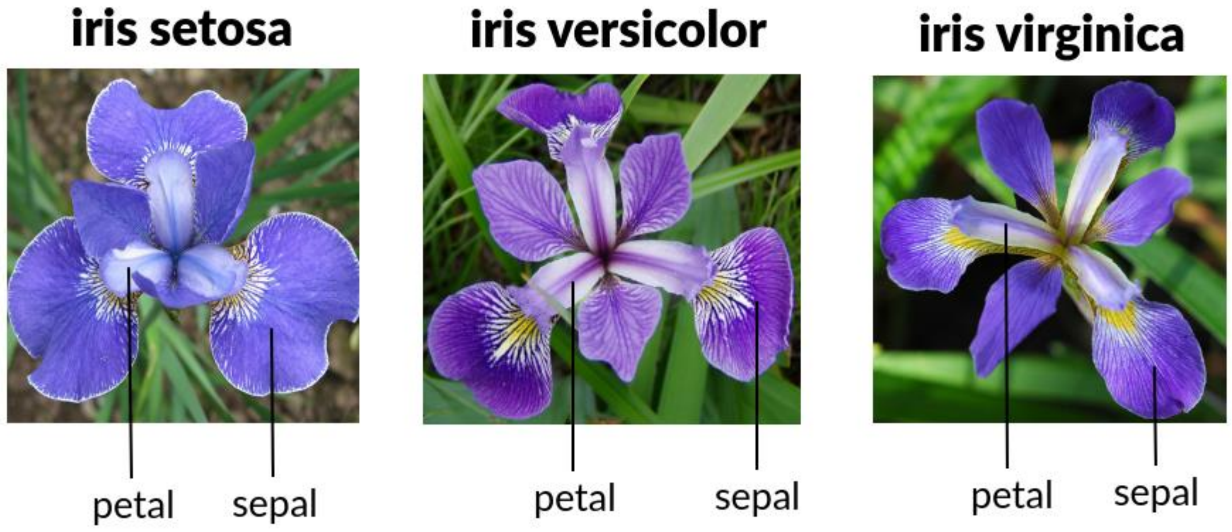
Classification

Binary classification problem

$$y = \sigma(\vec{w}^T \vec{x} + b)$$



Iris Flower Data

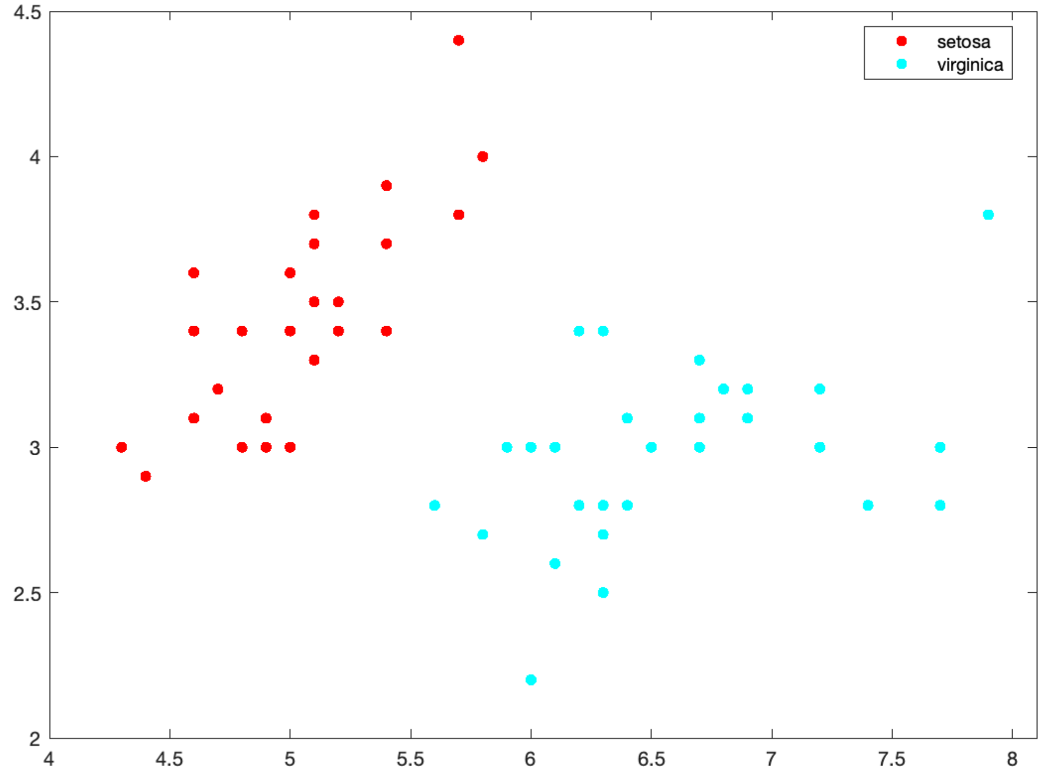


	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

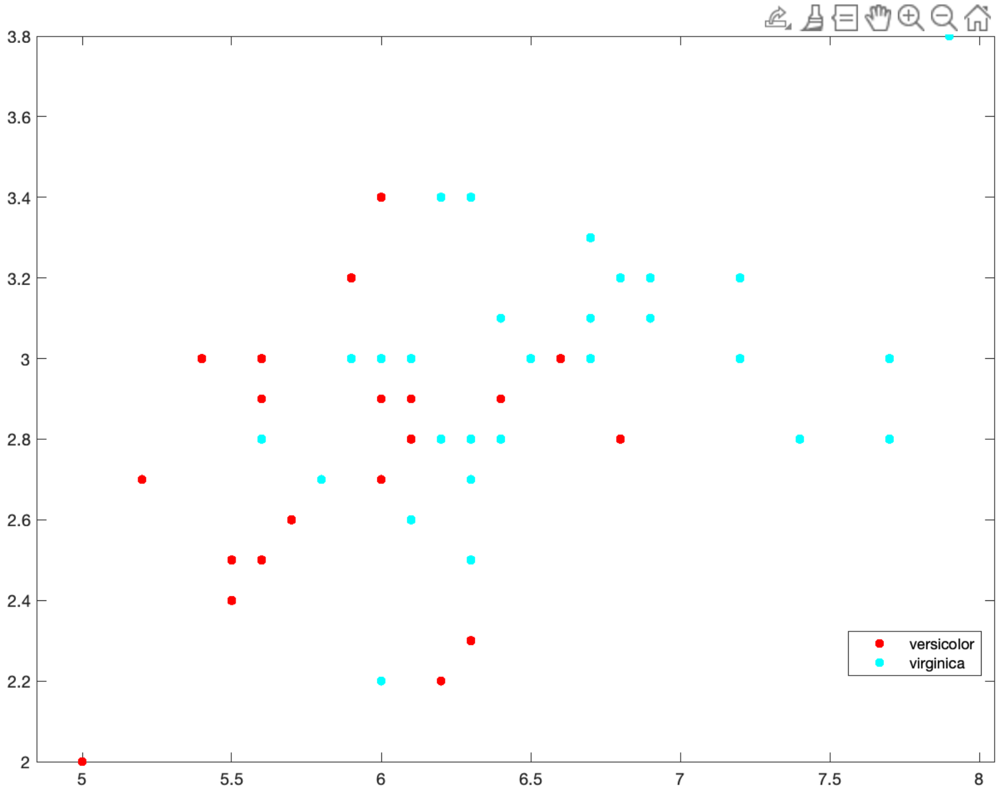
Features

Labels (Targets)

Linear vs Logistic regression

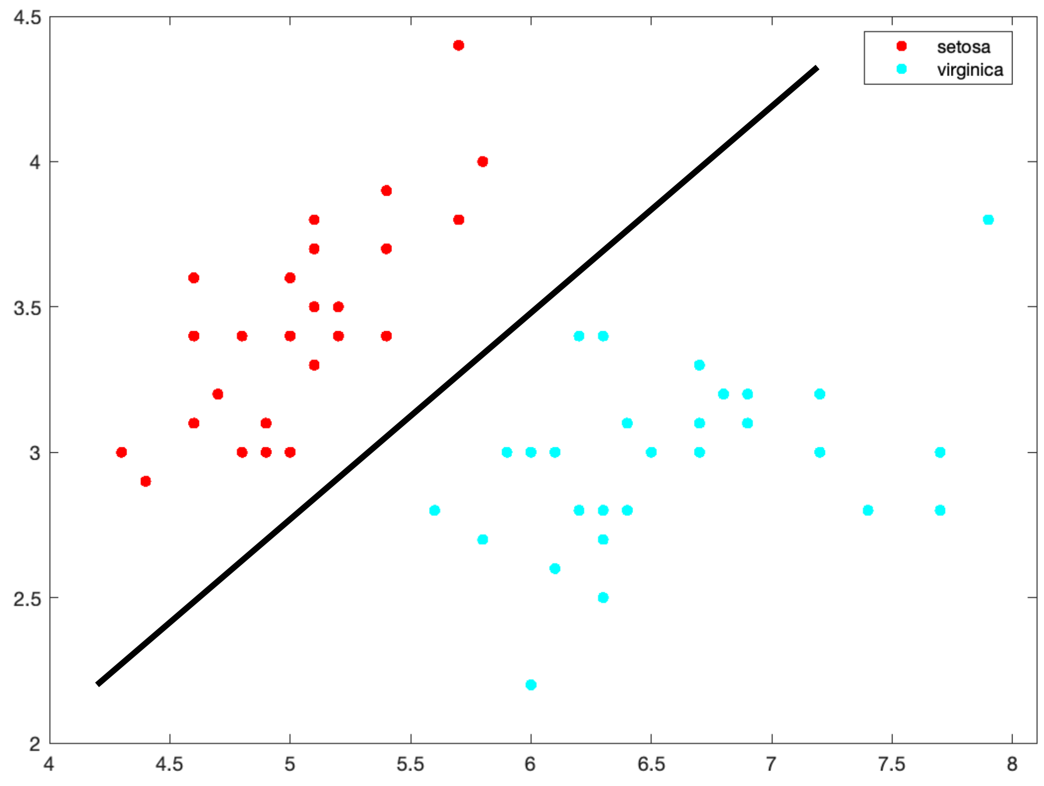


Setosa vs Virginica

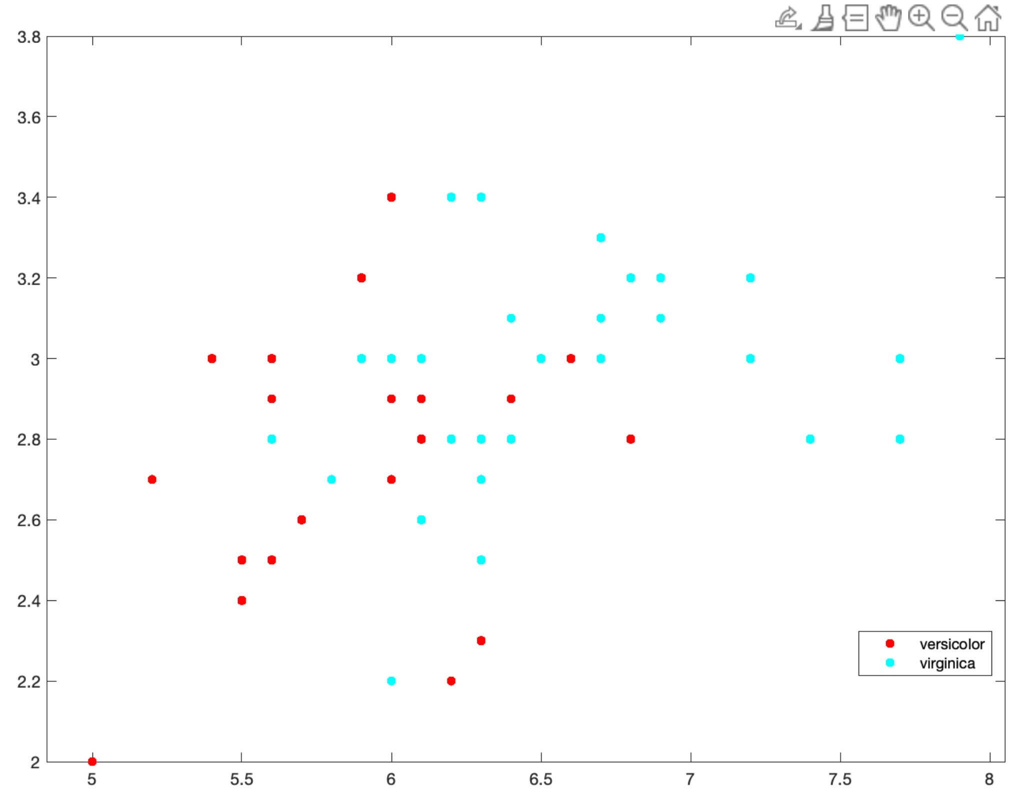


Virsicolor vs Virginica

Linear vs Logistic regression

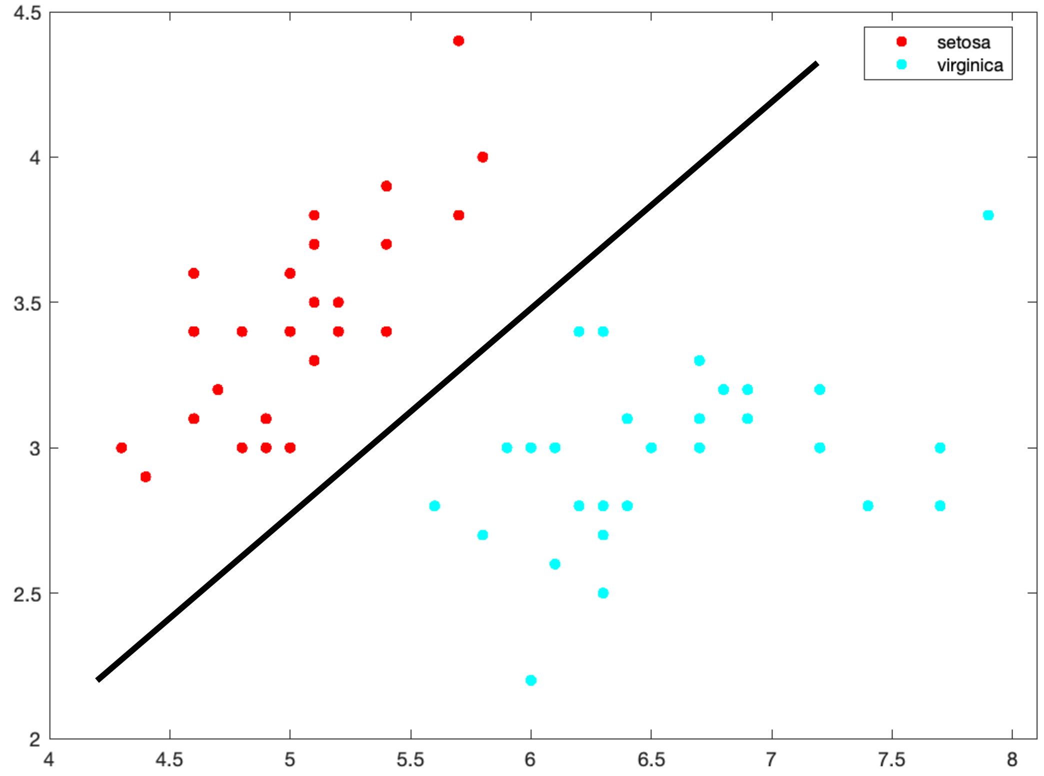


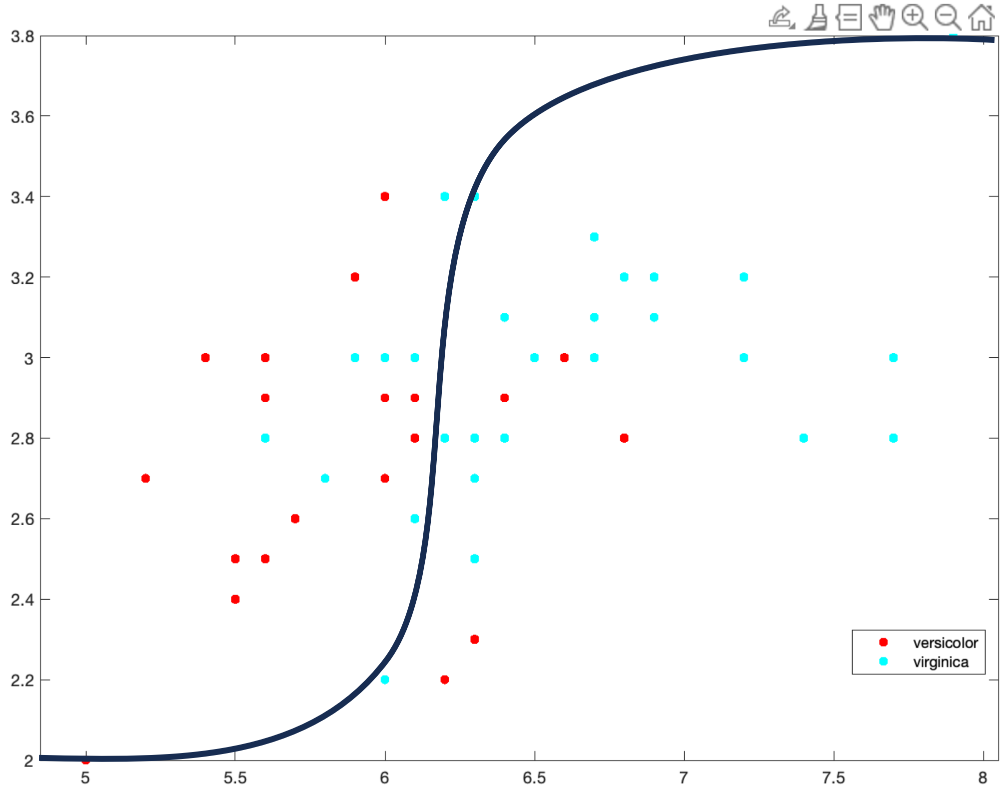
Setosa vs Virginica



Virsicolor vs Virginica

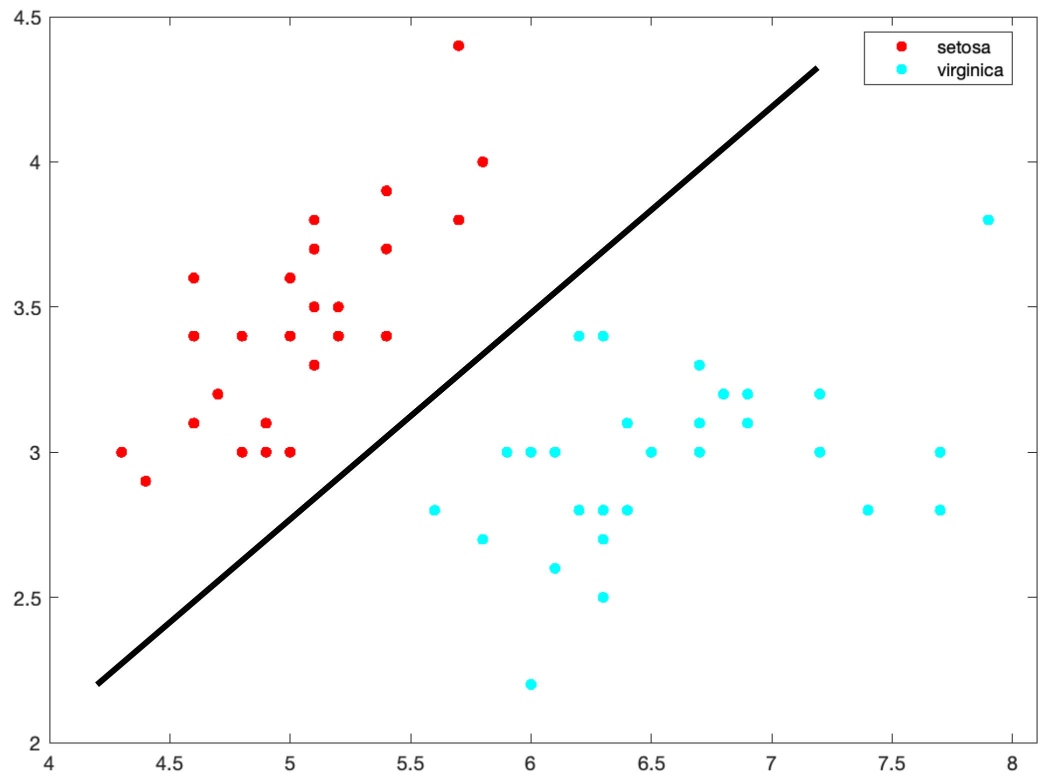
Linear vs Logistic regression

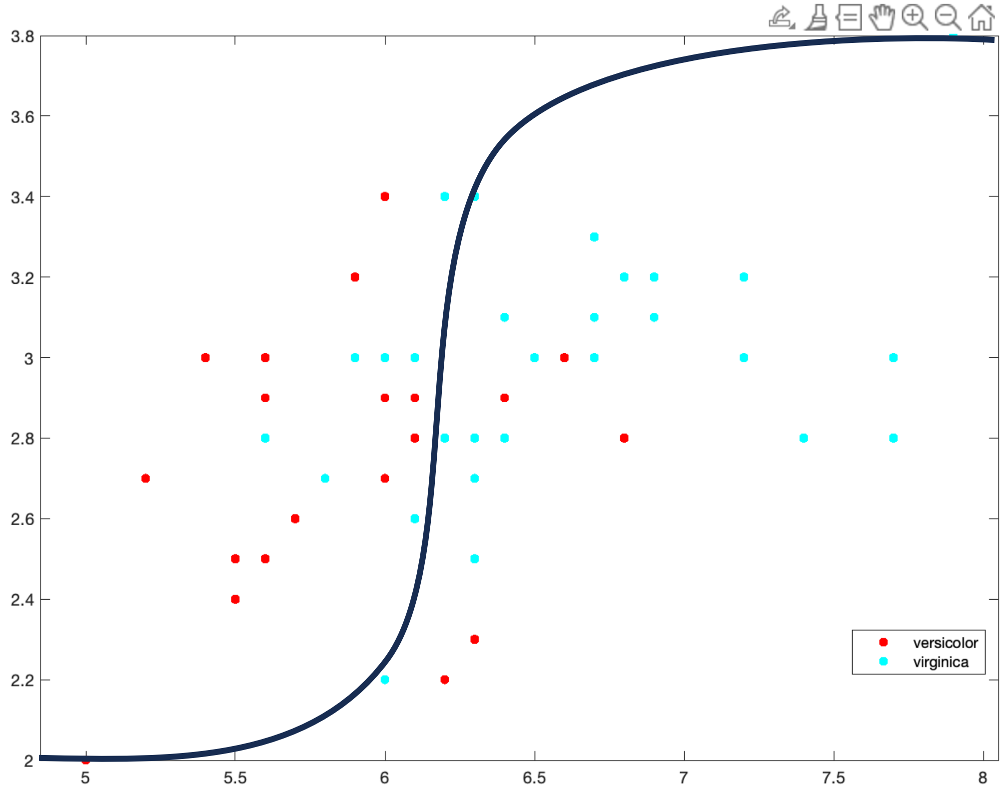




$$p = \frac{1}{1 + e^{-x}} \qquad 0 \leq p \leq 1$$

Linear vs Logistic regression



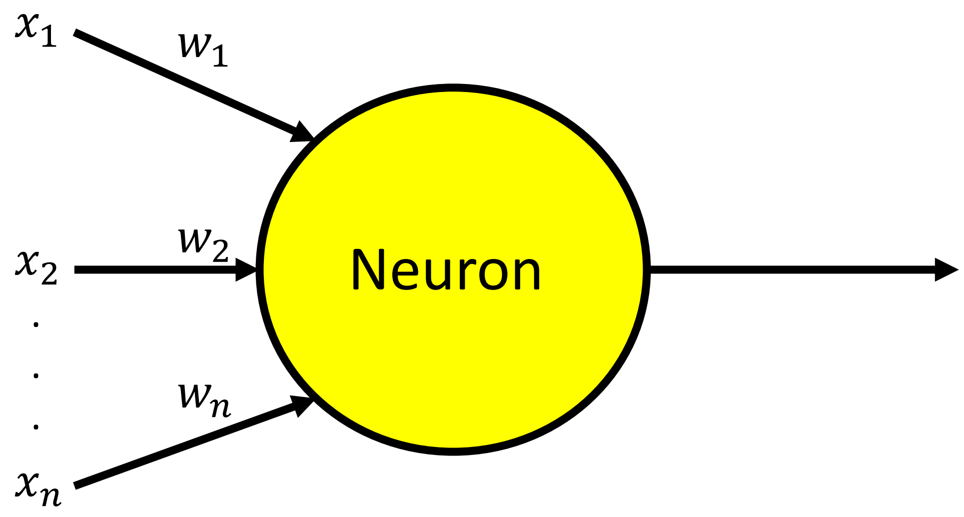


$p(Vir) \approx 0$

$p(Vir) \approx 1$

$$p = \frac{1}{1 + e^{-x}} \quad 0 \leq p \leq 1$$

Artificial Neuron as Logistic Regression Model



Input

$\vec{x} \in \mathbb{R}^n$

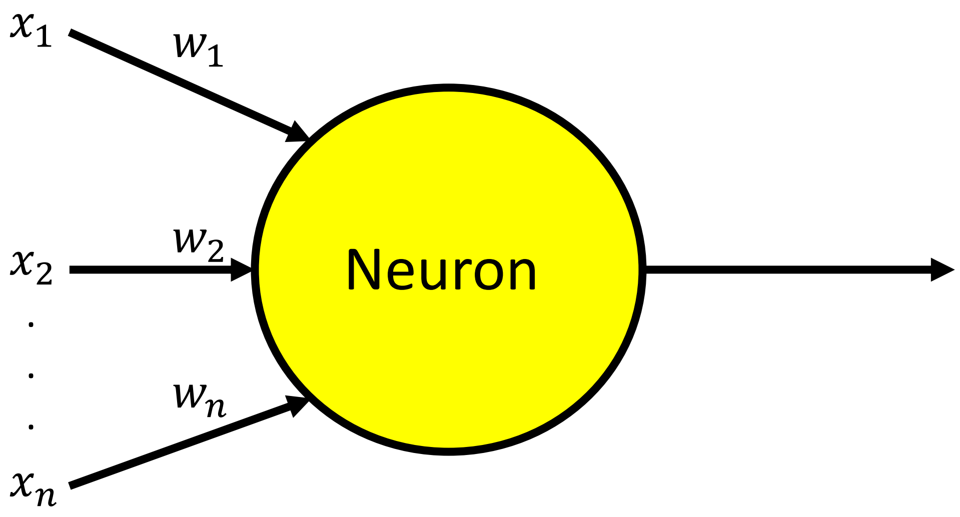
Activation

$$\sigma = \frac{1}{1 + e^{-z}}$$

Output

$$y = P(y = 1|\vec{x})$$
$$0 \leq y \leq 1$$

Artificial Neuron as Logistic Regression Model



Input

$$\vec{x} = \vec{w}^T\vec{x}$$

Activation

$$\sigma = \frac{1}{1 + e^{-z}}$$

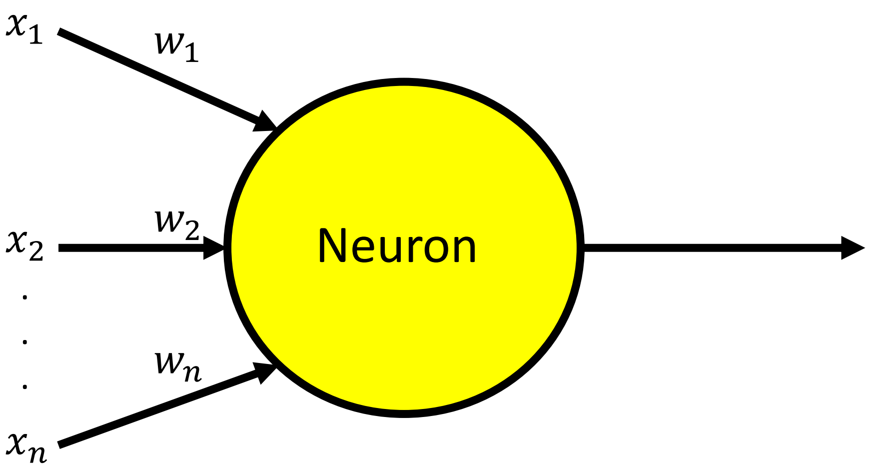
Output

$$y = \sigma(\vec{w}^T\vec{x} + b)$$

PART 3:

Training and Optimization

Cross Entropy Loss function (Binary)



Input

Activation

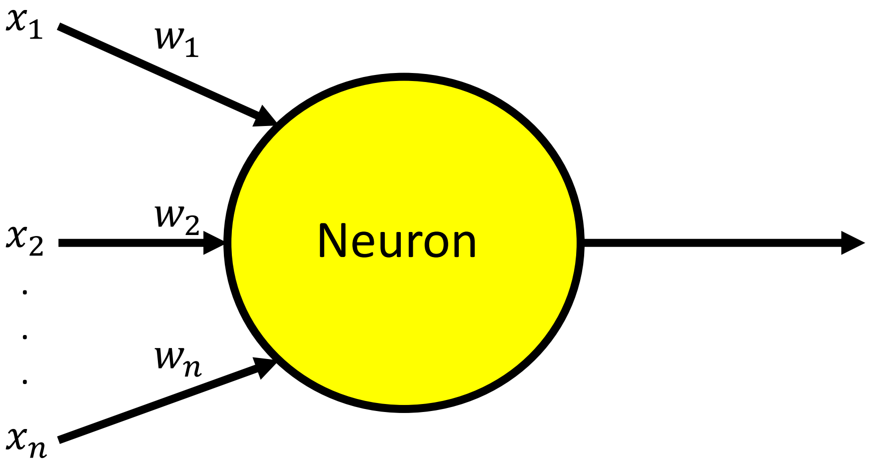
Output

$$\vec{x} = \vec{w}^{T}\vec{x}$$

$$\sigma = \frac{1}{1 + e^{-z}}$$

$$y = \sigma(\vec{w}^{T}\vec{x} + b)$$

Cross Entropy Loss function (Binary)

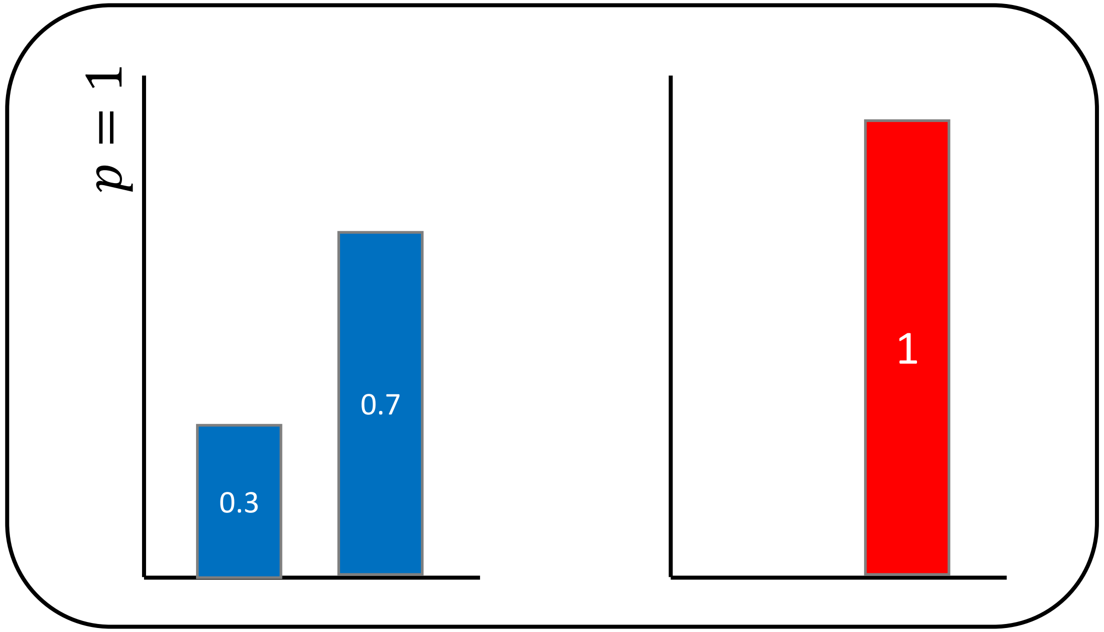


Input | Activation | Output

$$\vec{x} = \vec{w}^T \vec{x}$$

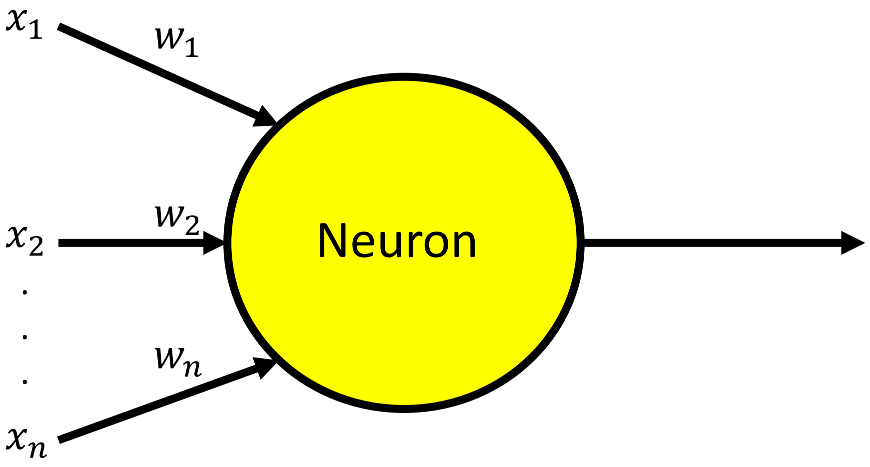
$$\sigma = \frac{1}{1 + e^{-z}}$$

$$y = \sigma(\vec{w}^T \vec{x} + b)$$



Model | Ground truth data

Cross Entropy Loss function (Binary)

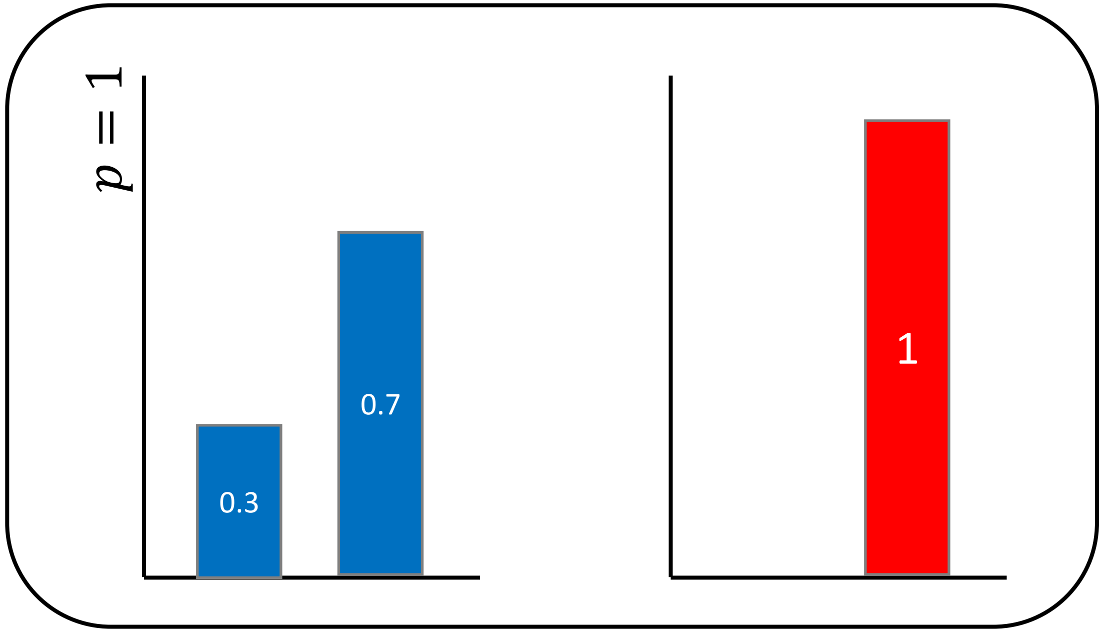


Input | Activation | Output

$\vec{x} = \vec{w}^T\vec{x}$

$\sigma = \dfrac{1}{1 + e^{-z}}$

$y = \sigma(\vec{w}^T\vec{x} + b)$



Model | Ground truth data

$J \approx$ Measure of difference in distributions

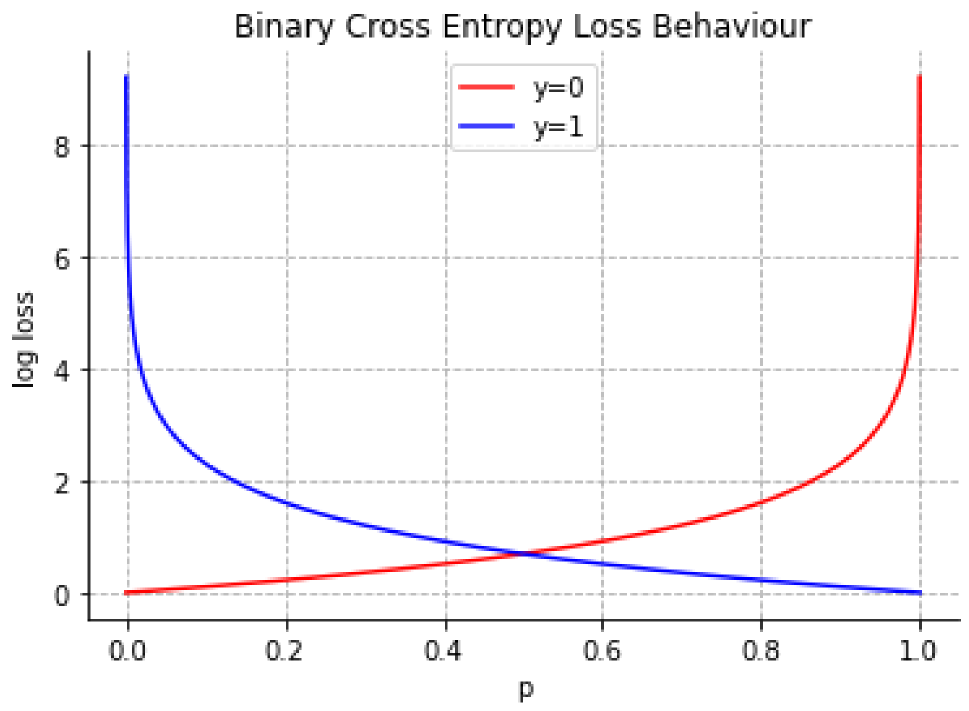
Cross Entropy Loss function (Binary)

$$L(\hat{y}, y) = f(x) = \begin{cases} -\,log(\hat{y}), & y = 1 \\ -log(1 - \hat{y}), & y = 0 \end{cases}$$

Cross Entropy Loss function (Binary)

$$L(\hat{y}, y) = f(x) = \begin{cases} -\,log(\hat{y})\,, & y = 1 \\ -log(1-\hat{y}), & y = 0 \end{cases}$$

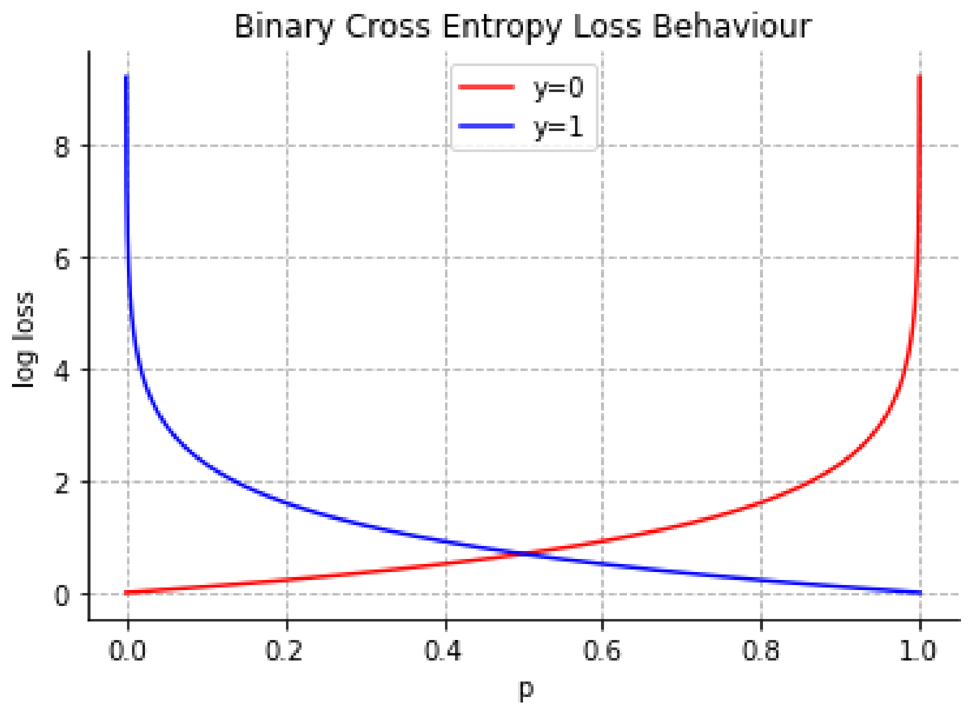
$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log (1-\hat{y}))$$



Cross Entropy Loss function (Binary)

$$L(\hat{y}, y) = f(x) = \begin{cases} -\,log(\hat{y}), & y = 1 \\ -log(1-\hat{y}), & y = 0 \end{cases}$$

$$L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log (1-\hat{y}))$$



Cross Entropy Loss is Convex

$\leftrightarrow L(\hat{y}, y)'' \geq 0$

The line segment between any two points does not lie below the graph

Logistic Regression Training

Training Set D

$$D : \{(\vec{x}^{(1)}, y^{(1)}), .., (\vec{x}^{(i)}, y^{(i)}), .., (\vec{x}^{(m)}, y^{(m)})\}$$

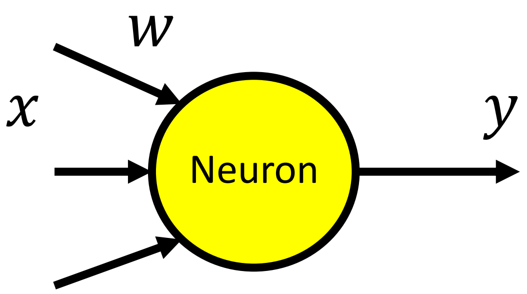
Cost J

$$J(\{\hat{y}\}^m, \{y\}^m; \{\vec{x}\}^m) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}))$$

Minimizing Loss using Gradient Descent

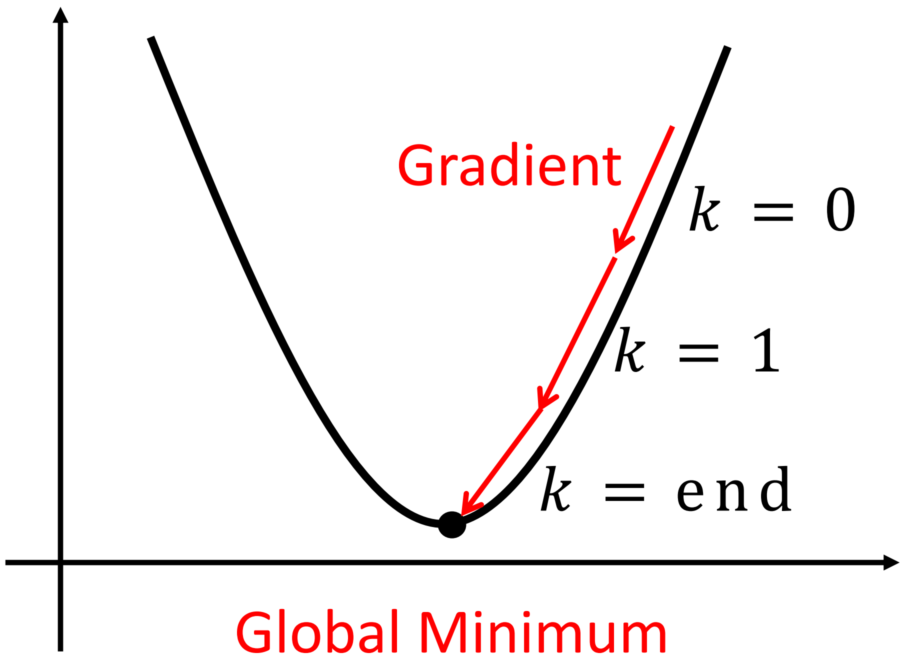
$$y = \sigma(\vec{w}^T\vec{x} + b)$$



$$J = L((\vec{w}, b), y)$$

$$\vec{w}_{k+1} = \vec{w}_k - \alpha\nabla_{\vec{w}}J(\vec{w}_k; b)$$

$$b_{k+1} = b_k - \alpha\frac{\partial}{\partial b}J(\vec{w}; b_k)$$



Minimizing Loss using Gradient Descent

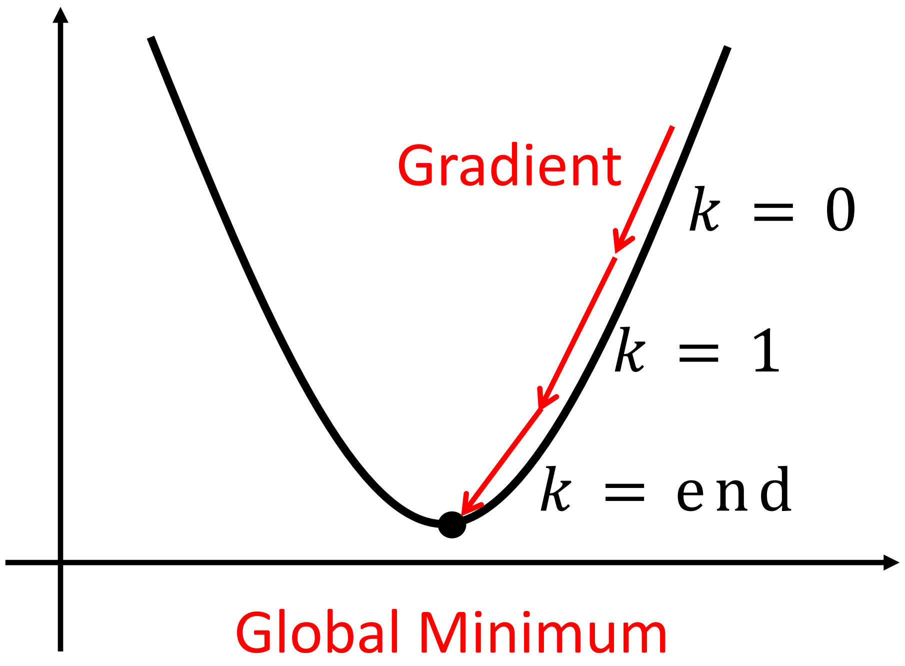
$$y = \sigma(\vec{w}^T\vec{x} + b)$$

Learning rate

$$\vec{w}_{k+1} = \vec{w}_k - \alpha\nabla_{\vec{w}}J(\vec{w}_k; b)$$

$$b_{k+1} = b_k - \alpha\frac{\partial}{\partial b}J(\vec{w}; b_k)$$

$$J = L((\vec{w}, b), y)$$



Iteratively adjust $(\vec{w}, b)$ until we reach the global minimum

Finding Gradient w.r.t. weights and biases

$$J = L(\hat{y}, y)$$

Finding Gradient w.r.t. weights and biases

$$J = L(\underbrace{\hat{y}}_{\hat{y} = \sigma(z)}, y)$$

$\hat{y} = \sigma(z)$ Activation function

Finding Gradient w.r.t. weights and biases

$$J = L(\hat{y}, y)$$

$$\hat{y} = \sigma(z)$$ Activation function

$$z = \vec{w}^{T}\vec{x} + b$$ Integrate Inputs

Finding Gradient w.r.t. weights and biases

$$J = L(\hat{y}, y)$$

$$\hat{y} = \sigma(z)$$

$$z = \vec{w}^T\vec{x} + b$$

$$J = L(\sigma(\vec{w}^T\vec{x} + b), y)$$

Finding Gradient w.r.t. weights and biases

$$J = L(\hat{y}, y)$$

$$\hat{y} = \sigma(z)$$

$$z = \vec{w}^T \vec{x} + b$$

$$J = L(\sigma(\boxed{\vec{w}}^T \vec{x} + b), y)$$

Using Chain Rule to Compute Gradients

$$J = L(\underbrace{\hat{y}}_{\hat{y} = \sigma(\underbrace{z}_{z = \vec{w}^T\vec{x} + b})}, y)$$

$$J = L(\sigma(\boxed{\vec{w}}^T\vec{x} + b), y)$$

$$P = f(g(h(x)))$$

$$\frac{dP}{dx} = \frac{df}{dg} * \frac{dg}{dh} * \frac{dh}{dx}$$

Using Chain Rule to Compute Gradients

$$J = L(\overbrace{\sigma(\underbrace{\vec{w}^T\vec{x} + b}_{z})}^{\hat{y}}, y)$$

$$P = f(g(h(x)))$$

$$\frac{dP}{dx} = \frac{df}{dg} * \frac{dg}{dh} * \frac{dh}{dx}$$

Using Chain Rule to Compute Gradients

$$J = L(\overbrace{\sigma(\underbrace{\vec{w}^T\vec{x} + b}_{z})}^{\hat{y}}, y) \qquad P = f(g(h(x)))$$

$$\frac{\partial J}{\partial \vec{w}} = \frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \frac{\partial z}{\partial \vec{w}} \qquad \frac{dP}{dx} = \frac{df}{dg} * \frac{dg}{dh} * \frac{dh}{dx}$$

$$\frac{\partial L}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z} * \nabla_{\vec{w}}\, z$$

Using Chain Rule to Compute Gradients

$$\nabla_{\vec{w}} L(\hat{y}, y) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \nabla_{\vec{w}} z$$

FWD

$z = \vec{w}^T \vec{x} + b \longrightarrow \hat{y} = \sigma(z) \longrightarrow L(\hat{y}, y)$

BWD

$\nabla_{\vec{w}} z \xleftarrow{\times} \frac{\partial \hat{y}}{\partial z}(z) \xleftarrow{\times} \frac{\partial L}{\partial \hat{y}}(\hat{y}, y)$

Using Chain Rule to Compute Gradients

BWD	$\nabla_{\vec{w}} z$	$\leftarrow \times$	$\frac{\partial \hat{y}}{\partial z}(z)$	$\leftarrow \times$	$\frac{\partial L}{\partial \hat{y}}(\hat{y}, y)$
$\nabla_{\vec{w}} L$	$\vec{x}$	$\leftarrow \times$	$\sigma(z)(1-\sigma(z))$	$\leftarrow \times$	$-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$
$\frac{\partial L}{\partial b}$	1	$\leftarrow \times$	$\sigma(z)(1-\sigma(z))$	$\leftarrow \times$	$-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$

Training Terminologies

- **Forward Propagation:**
  Computing the **loss through forward pass** for a single training example

- **Backward Propagation:**
  Computing **gradients of parameters** through backward pass for a single training example

- **Batch:**
  Training set could be divided into **smaller sets** called batches

- **Iteration:**
  When an **entire batch** is passed both **forward and backward**

- **Epoch:**
  When an **entire dataset** is passed both **forward and backward** through the NN once

Batch Gradient Descent

Training Set D

$$D : \{(\vec{x}^{(1)}, y^{(1)}), .., (\vec{x}^{(i)}, y^{(i)}), .., (\vec{x}^{(m)}, y^{(m)})\}$$

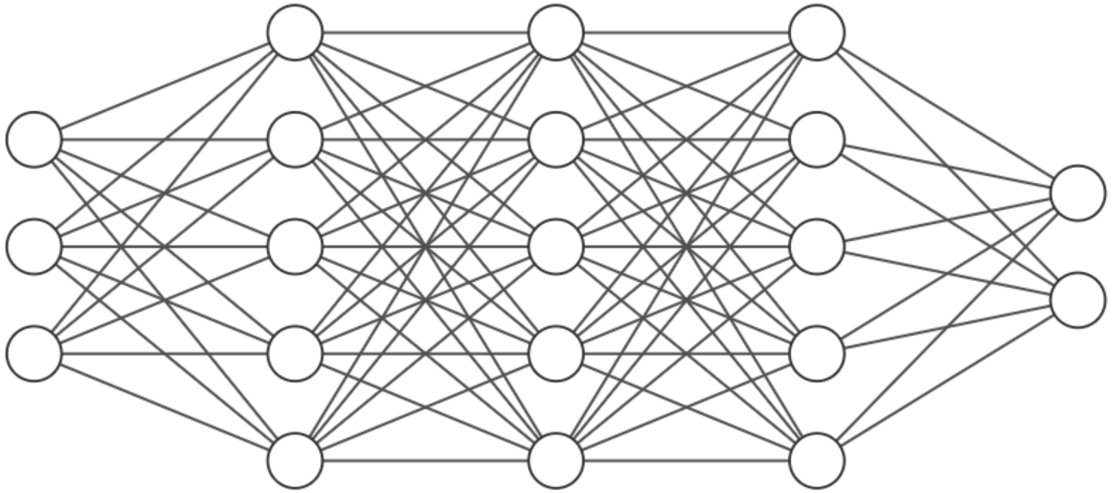
Cost J

$$J(\{\hat{y}\}^m, \{y\}^m; \{\vec{x}\}^m) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

$$= -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log (1 - \hat{y}^{(i)}))$$

Sum the gradients for all m-examples (Epoch)

Next episode in EE P 596…



Optimizations in Deep Learning