

# Machine Learning: Introduction to Linear Regression, Logistic Regression, and Neural Networks

# 2.1 Linear Regression: Mathematical Foundations

# Linear Regression: Mathematical Foundations

Goal of this Section:

- Present the mathematical foundations for the machine learning approach for linear regression, including:
  - Format of training data
  - Function structure and parameters
  - Loss function
  - Training algorithm
  - Prediction algorithm

# Linear Regression – Line Fitting

## Training Data:

- Input information: X values
- Output information: Y values

## Linear Regression Goal:

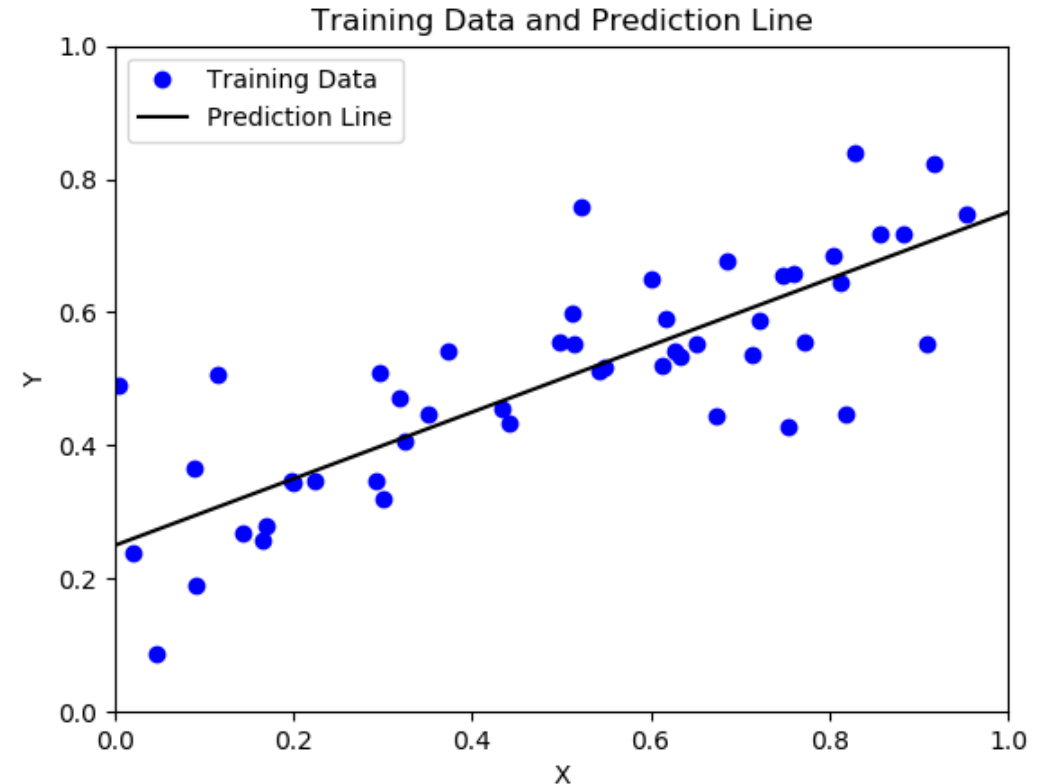
- Find straight line that best fits the training data

## Prediction:

- Use line to predict Y values given new input X values

## Why start with Linear Regression?

- Simple problem with well known solution
- This course will present a general approach that can also be applied to Logistic Regression and Neural Networks

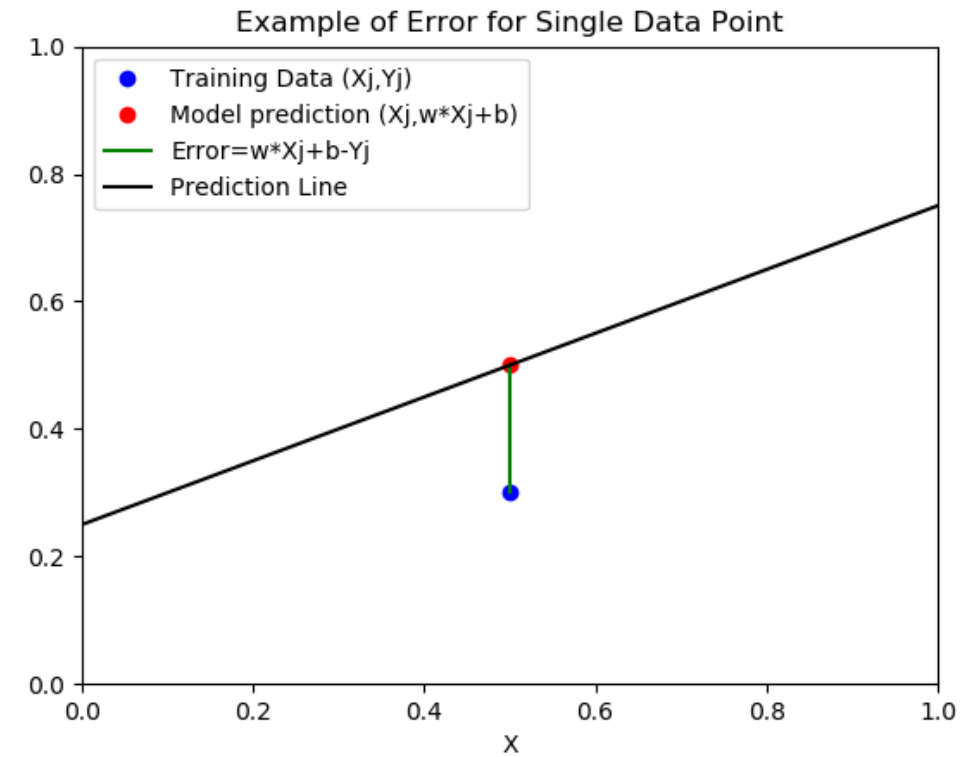


# Least Squares Approach

- Assume  $Y=WX+b$
- Least squares approach: find  $W$  and  $b$  that minimizes sum of squared error:

$$L = \sum_{j=0}^{m-1} (WX_j + b - Y_j)^2$$

- Error for a single input/output information pair
- Loss is sum of squares of error



# Summary of Least Squares Approach

## TRAINING DATA:

- Input/Output information pairs:  $(X_0, Y_0), (X_1, Y_1), \dots, (X_{m-1}, Y_{m-1})$

## FUNCTION STRUCTURE

- Assume line  $Y = W \cdot X + b$  (in this example  $W$  and  $b$  are scalars)

## LOSS:

- Measure accuracy of function structure using Loss function:  $L = \sum_{j=0}^{m-1} (WX_j + b - Y_j)^2$

## TRAINING PHASE:

- Find slope  $W$  and intercept  $b$  that minimize squared error function
- $W$  and  $b$  are solutions of the normal equations

## FUNCTION PARAMETERS/RULES:

- These are the  $W$  and  $b$  that minimize the squared error

## PREDICTION PHASE

- Given new input information  $X$ , use computed  $W$  and  $b$  to determine  $Y = W \cdot X + b$

# Normal Equations

- For the 1-dimensional problem input/output info:  $(X_0, Y_0), (X_1, Y_1), \dots, (X_{m-1}, Y_{m-1})$
- Let us define:

$$\hat{X} = \begin{bmatrix} X_0 & X_1 & \dots & X_{m-1} \\ 1 & 1 & \dots & 1 \end{bmatrix} \quad \text{and} \quad Y = [Y_0 \quad Y_1 \quad \dots \quad Y_{m-1}]$$

- Define the parameter vector as:

$$\hat{W} = [W \quad b]$$

- The least squares normal equations solution is (T signifies transpose and -1 is the inverse)

$$\hat{W} = (Y \hat{X}^T)(\hat{X} \hat{X}^T)^{-1}$$

(Note: this formula is different from what you have probably seen. In typical linear algebra courses,  $Y$  and  $w$  are column vectors. The above formula is the transpose of the typical formula from courses.)

- These ideas can be generalized to higher dimensions

# Linear Regression: General Approach

General approach has following components and phases:

(1) Training Data

(2) Function Structure

- Defines general form of the function with unknown parameters
- Process of applying function structure is called Forward Propagation

(3) Loss Function

- Used to measure effectiveness of function structure and choice of parameters

(4) Training Phase

- Uses optimization to determine function parameters that minimize loss function for training data
- Process of computing derivatives is called Back Propagation

(5) Prediction Phase

- Applies forward propagation using parameters determined in Training Phase to predict outputs when new input data is provided



# Training Data

- Consider more general regression problem where there are  $m$  data points, each consisting of a input information vector of length  $d$  and value  $Y$ :

- Data point  $j$ : input information (feature) vector:  $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \vdots \\ X_{d-1,j} \end{bmatrix}$  and output:  $Y_j$
- Define the feature matrix ( $d \times m$ ) and output vector ( $1 \times m$ ):

$$X = \begin{bmatrix} X_{00} & \dots & X_{0,m-1} \\ \dots & \dots & \dots \\ X_{d-1,0} & \dots & X_{d-1,m-1} \end{bmatrix} \quad Y = [Y_0 \quad \dots \quad Y_{m-1}]$$

# Training Data – Example Points in Plane

- For the 1-dimensional least squares problem in the motivating example, the training data consists of points in the plane:  $(X_0, Y_0), (X_1, Y_1), \dots, (X_{m-1}, Y_{m-1})$
- For example consider 4 samples in training set:  $(1,1), (0.5,2), (2,3), (4,2)$
- In this case each data point has 1 feature (the X value)
- Feature matrix and output vector are:

$$X = [1 \quad 0.5 \quad 2 \quad 3] \quad Y = [1 \quad 2 \quad 3 \quad 2]$$

# Training Data – Example Predicting House Prices

- Suppose we have input information (features) of a house and output information (price)

Feature 0: Lot area (in square feet)

Feature 1: House area (in square feet)

Feature 2: Number of bathrooms

Feature 3: Number of floors

Feature ...:

Feature d-1: Size of garage (number of cars)

- Data point j: feature vector:  $\begin{bmatrix} 5000 \\ 2000 \\ 3 \\ 2 \\ \dots \\ 2 \end{bmatrix}$  and output information (price):  $Y_j = 800,000$

- Collect all feature vectors and output values to create feature matrix and output vector

# Function Structure - Forward Propagation

Forward Propagation is name applied to process of estimating output values using function structure

Input: feature matrix  $X$  (d x m d features and m data points)

Assign: parameter vector  $W = [W_0 \quad W_1 \quad \dots \quad W_{d-1}]$  and bias  $b$

1. Linear part: for  $j=0, \dots, m-1$

$$Z_j = W_0 X_{0j} + W_1 X_{1j} + W_2 X_{2j} + \dots + W_{d-1} X_{d-1j} + b$$

In vector form  $Z = [Z_0 \quad Z_1 \quad \dots \quad Z_{m-1}]$  and  $Z = WX + b$

2. Activation: apply function  $f(z)$  to each component of  $Z$ :

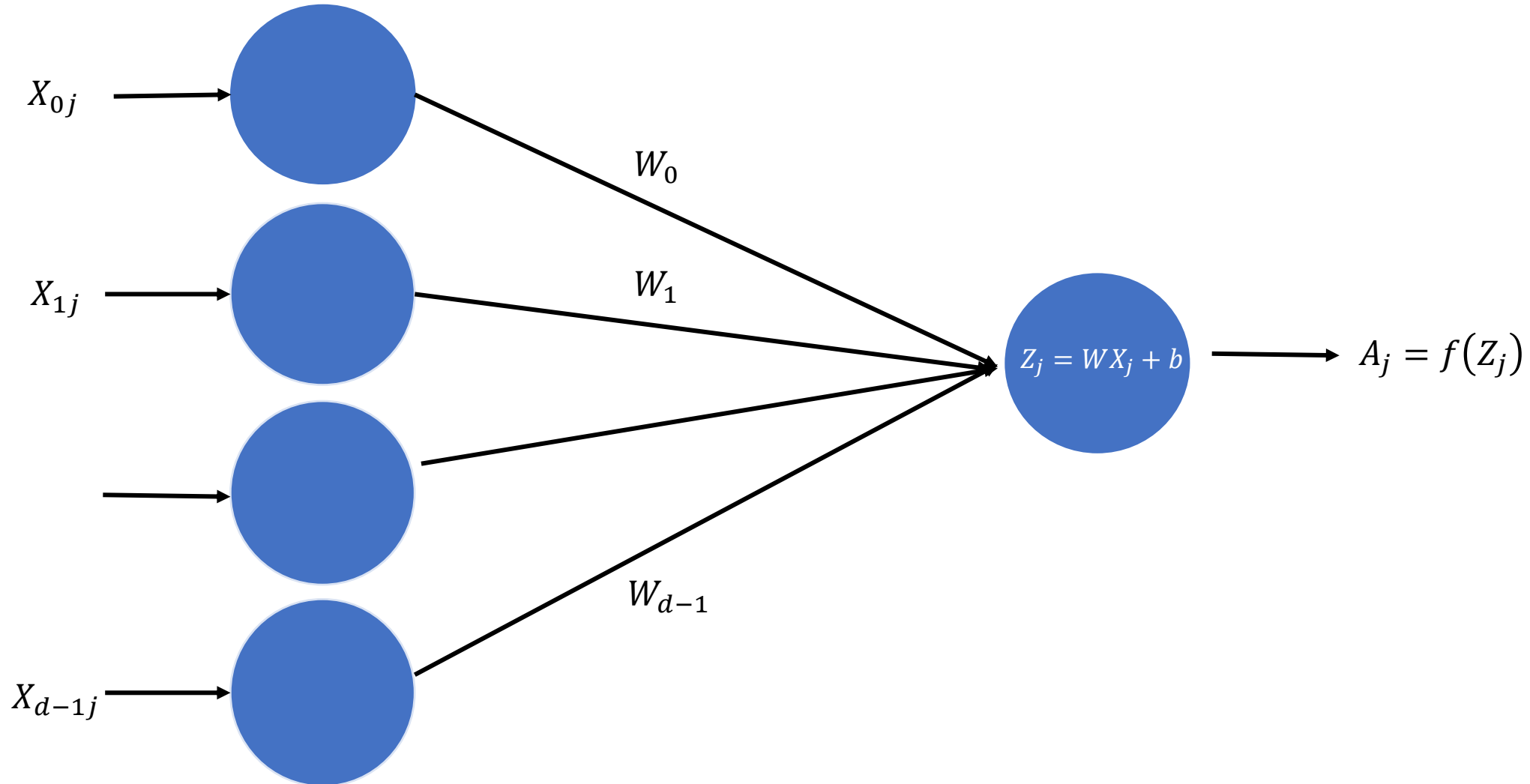
$$A_j = f(Z_j) \quad j = 0, \dots, m-1$$

For Linear Regression  $f(z) = z$ , so  $A_j = Z_j, j = 0, \dots, m-1$

$[A_0 \quad A_1 \quad \dots \quad A_{m-1}]$  is estimate of output values

Function Structure

# Forward Propagation Diagram



# Forward Propagation - Example

- Consider a case of 2 features and 3 data points ( $m=3$ )

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10]$$

- Assume that initial parameter values are:

$$W = [1 \quad 1] \quad b = [2]$$

- Forward Propagation:

$$Z = WX + b = [1 \quad 1] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [2] = [5 \quad 9 \quad 13]$$

$$A = Z = [5 \quad 9 \quad 13]$$

# Loss Function

- Loss function used to measure effectiveness of choice  $W$  and  $b$
- Standard approach for Linear regression is to use Mean Squared Error function:

$$Loss = L = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2$$

- $A_j - Y_j$  is the error between the original training data point and the prediction based on the function structure and forward propagation
- Loss is mean of squares of errors for all training points

# Loss Function - Example

- Consider a case of 2 features and 3 data points ( $m=3$ )

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10]$$

- From the Forward Propagation Example

$$A = [5 \quad 9 \quad 13]$$

- Loss function defined by

$$L = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} \sum_{j=0}^{m-1} [(5 - 8)^2 + (9 - 6)^2 + (13 - 10)^2] = 9$$



# Training Phase

- Training phase attempts to find suitable coefficients  $W$  and  $b$  by minimizing loss function when applied to training data
- From multi-variable calculus, Loss function has a local minimum when the gradients are 0

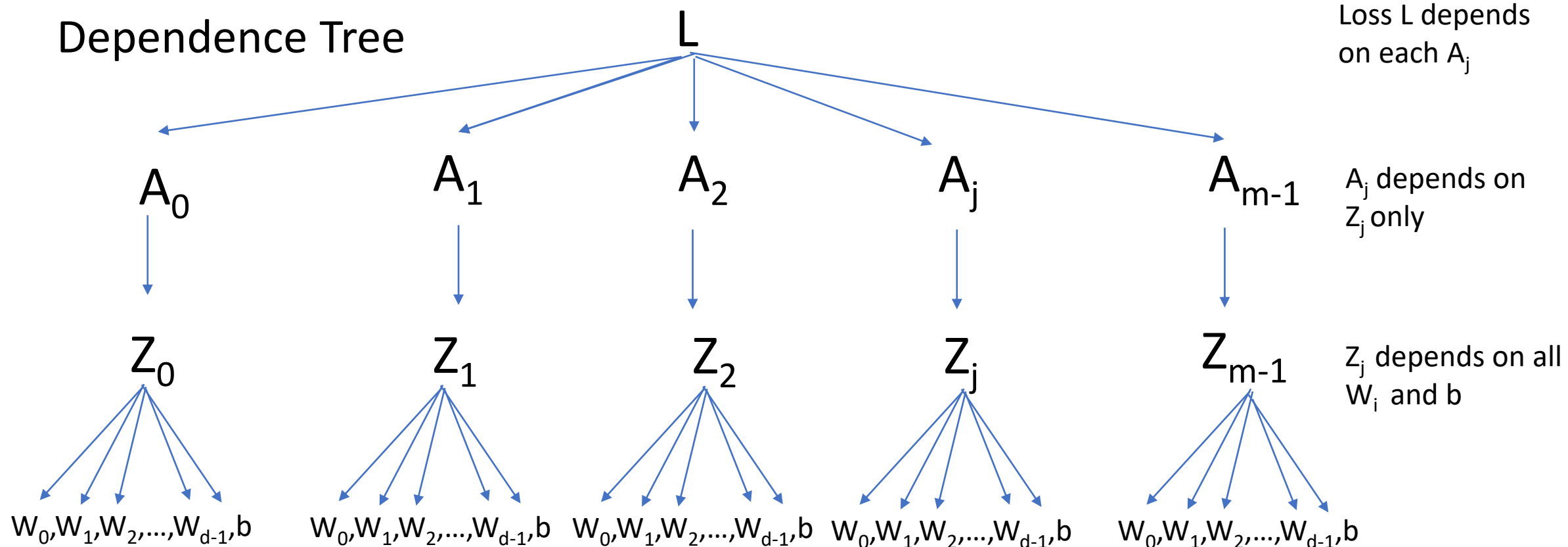
$$\nabla_W L = 0 \text{ and } \nabla_b L = 0$$

- Can solve these equations analytically for Linear Regression, but not in general case (Logistic Regression and Neural Networks)
- Use optimization algorithm (example: Gradient Descent) to minimize Loss function
  - Need to compute the above gradients

# Computing Gradients

Apply Chain Rule of Calculus to determine  $\nabla_W L$  and  $\nabla_b L$ .

Dependence Tree



# Multi-Variable Calculus – Chain Rule

Recall the chain rule lecture (Section 1.6)

- If  $L = L(Z_0, \dots, Z_{m-1})$  and  $Z = WX + b$ , then

$$\nabla_W L = \nabla_Z L X^T \quad \text{and} \quad \nabla_b L = \sum_{j=0}^{m-1} \nabla_Z L_j$$

- For linear regression, the loss function  $L$  depends on  $A_0, \dots, A_{m-1}$
- $A_0$  depends exclusively on  $Z_0$ ,  $A_1$  depends exclusively on  $Z_1$  and so on
- In fact  $\frac{\partial A_j}{\partial Z_j} = 1$  since the activation function is the identity  $f(z)=z$
- To compute the derivatives correctly, we need to again apply the chain rule

$$\nabla_Z L = \left[ \frac{\partial L}{\partial Z_0} \quad \dots \quad \frac{\partial L}{\partial Z_{m-1}} \right] = \left[ \frac{\partial L}{\partial A_0} \frac{\partial A_0}{\partial Z_0} \quad \frac{\partial L}{\partial A_1} \frac{\partial A_1}{\partial Z_1} \quad \dots \quad \frac{\partial L}{\partial A_{m-1}} \frac{\partial A_{m-1}}{\partial Z_{m-1}} \right]$$

$$\nabla_Z L = \left[ \frac{\partial L}{\partial A_0} \quad \frac{\partial L}{\partial A_1} \quad \dots \quad \frac{\partial L}{\partial A_{m-1}} \right] * \left[ \frac{\partial A_0}{\partial Z_0} \quad \frac{\partial A_1}{\partial Z_1} \quad \dots \quad \frac{\partial A_{m-1}}{\partial Z_{m-1}} \right] = \nabla_A L * [1 \quad 1 \quad \dots \quad 1]$$

Here  $*$  means component-wise multiplication

# Gradient of Mean Squared Error Function

- For the mean squared error function:

$$L = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2$$

- Gradient given by

$$\nabla_A L = \left[ \frac{\partial L}{\partial A_0} \quad \frac{\partial L}{\partial A_1} \quad \cdots \quad \frac{\partial L}{\partial A_{m-1}} \right] \text{ where } \frac{\partial L}{\partial A_j} = \frac{2}{m} (A_j - Y_j) \text{ for } j=0, \dots, m-1$$

# Back Propagation Algorithm

Back propagation is the process of computing  $\nabla_W L$  and  $\nabla_b L$

Assume that forward propagation has taken place so  $[A_0 \ A_1 \ \dots \ A_{m-1}]$  has been computed

Input: feature matrix  $X$  and value vector  $Y$

1. Compute gradient of  $L$  with respect to  $A$

$$\nabla_A L = \left[ \frac{\partial L}{\partial A_0} \quad \frac{\partial L}{\partial A_1} \quad \dots \quad \frac{\partial L}{\partial A_{m-1}} \right], \quad \frac{\partial L}{\partial A_j} = \frac{2}{m} (A_j - Y_j), \quad j = 0, \dots, m-1$$

2. Compute derivatives of  $A$ :  $\frac{\partial A_j}{\partial Z_j} = 1, j = 0, \dots, m-1$

3. Compute gradient  $L$  with respect to  $Z$ :

$$\nabla_Z L = \nabla_A L * \left[ \frac{\partial A_0}{\partial Z_0} \quad \frac{\partial A_1}{\partial Z_1} \quad \dots \quad \frac{\partial A_{m-1}}{\partial Z_{m-1}} \right] \text{ (component-wise multiplication)}$$

4. Compute gradient of  $L$  with respect to  $W$  and  $b$ :

$$\nabla_W L = \nabla_Z L X^T, \quad \nabla_b L = \sum_{j=0}^{m-1} \nabla_Z L_j$$

# Back Propagation - Example

- Consider a case of 2 features and 3 data points ( $m=3$ )

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10]$$

- Assume that initial parameter values are:

$$W = [1 \quad 1] \quad b = [2]$$

- From forward propagation example:

$$A = Z = [5 \quad 9 \quad 13]$$

- Gradient of Loss with respect to A:

$$\frac{\partial L}{\partial A_j} = \frac{2}{m} (A_j - Y_j) \text{ for } j=0, \dots, m-1 \quad \nabla_A L = \left[ \frac{2}{3} (5 - 8) \quad \frac{2}{3} (9 - 6) \quad \frac{2}{3} (13 - 10) \right] = [-2 \quad 2 \quad 2]$$

$$\left[ \frac{\partial A_0}{\partial Z_0} \quad \frac{\partial A_1}{\partial Z_1} \quad \frac{\partial A_2}{\partial Z_2} \right] = [1 \quad 1 \quad 1]$$

$$\nabla_Z L = \nabla_A L * \left[ \frac{\partial A_0}{\partial Z_0} \quad \frac{\partial A_1}{\partial Z_1} \quad \frac{\partial A_2}{\partial Z_2} \right] = [-2 \quad 2 \quad 2] * [1 \quad 1 \quad 1] = [-2 \quad 2 \quad 2] \text{ (component-wise multiplication)}$$

$$\nabla_W L = \nabla_Z L X^T = [-2 \quad 2 \quad 2] \begin{bmatrix} 1 & 2 \\ 2 & 5 \\ 4 & 7 \end{bmatrix} = [10 \quad 20]$$

$$\nabla_b L = \sum_{j=0}^{m-1} \nabla_Z L_j = 2 \text{ (sum of entries of } \nabla_b L)$$

# Training Algorithm

- Training algorithm uses Gradient Descent to find parameters  $W$  and  $b$  that minimize the Loss function
- At each step of gradient descent need to compute gradient of loss with respect to  $W$  and  $b$
- Computation of gradient involves both forward and back propagation

# Training Algorithm

Input training data: feature matrix  $X$  and values  $Y$

Make initial guess for parameters  $W_{\text{epoch}=0}$  and  $b_{\text{epoch}=0}$

Choose learning rate  $\alpha > 0$

1. Loop for epoch  $i = 1, 2, \dots$

- Forward Propagate using  $X$  to compute  $A_{\text{epoch}=i-1}$
- Back Propagate using  $X$ ,  $Y$ , and  $A_{\text{epoch}=i-1}$  to compute  $\nabla_W L_{\text{epoch}=i-1}, \nabla_b L_{\text{epoch}=i-1}$

- Update parameters: (Gradient Descent)

$$W_{\text{epoch}=i} = W_{\text{epoch}=i-1} - \alpha \nabla_W L_{\text{epoch}=i-1}$$

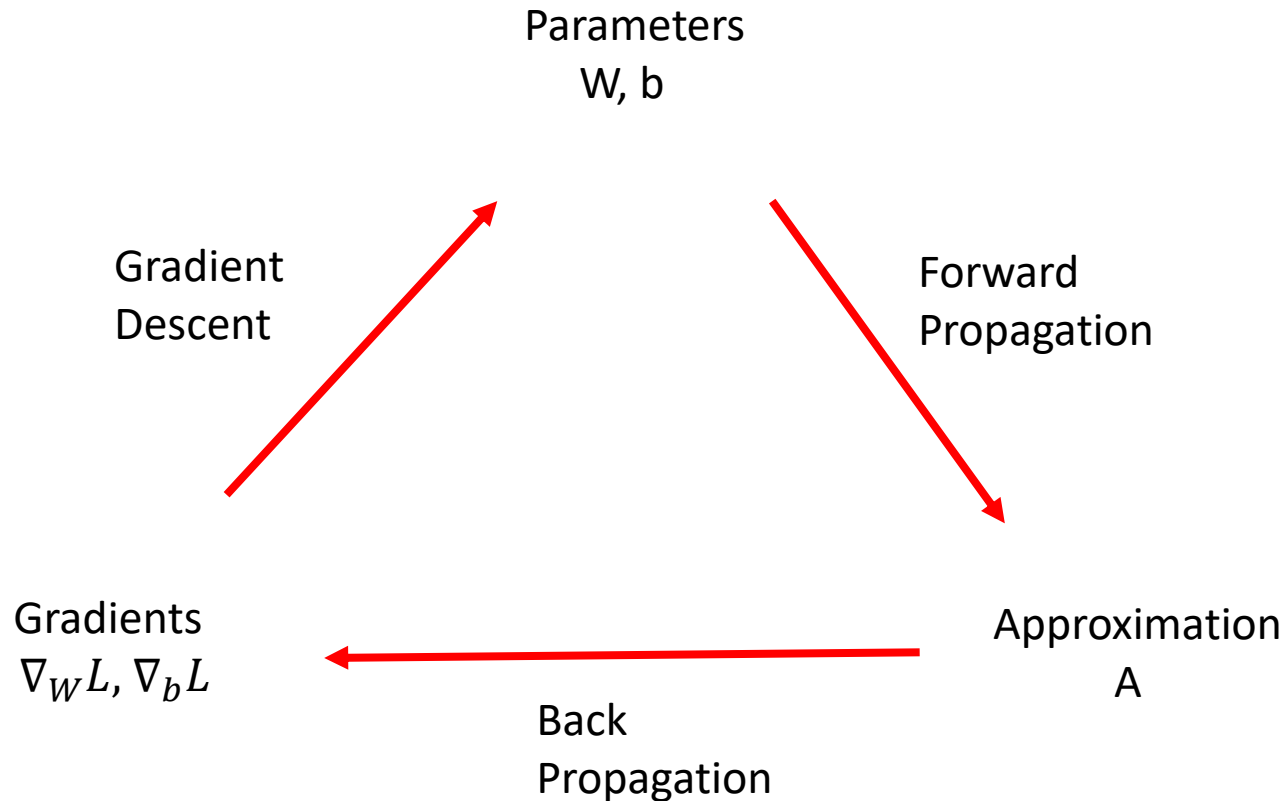
$$b_{\text{epoch}=i} = b_{\text{epoch}=i-1} - \alpha \nabla_b L_{\text{epoch}=i-1}$$

- Forward Propagate to compute  $A_{\text{epoch}=i}$
- Compute Loss at  $A_{\text{epoch}=i}$

Loop for fixed number of epochs (or if Loss reduced sufficiently)



# Training Algorithm



- With initial  $W$  and  $b$  use Forward Propagation to compute  $A$
- Use Back Propagation to compute gradients
- Use Gradient Descent to update parameters
- Process is repeated

# Training Algorithm - Example

- Consider a case of 2 features and 3 data points ( $m=3$ )

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10]$$

- Assume that initial parameter values are:

$$W_{epoch=0} = [1 \quad 1] \quad b_{epoch=0} = [2]$$

- Choose learning rate  $\alpha=0.01$

EPOCH 1

- From Forward Propagation Example:

$$A_{epoch=0} = [5 \quad 9 \quad 13]$$

- From Back Propagation Example:

$$\nabla_W L_{epoch=0} = [10 \quad 20], \quad \nabla_b L_{epoch=0} = [2]$$

- Update:

$$W_{epoch=1} = W_{epoch=0} - \alpha \nabla_W L_{epoch=0} = [1 \quad 1] - 0.01 * [10 \quad 20] = [0.9 \quad 0.8]$$

$$b_{epoch=1} = b_{epoch=0} - \alpha \nabla_b L_{epoch=0} = [2] - 0.01 * [2] = [1.98]$$

# Training Algorithm - Example

- Apply Forward Propagation with  $W_{epoch=1}$  and  $b_{epoch=1}$

$$A_{epoch=1} = Z_{epoch=1} = W_{epoch=1}X + b_{epoch=1} = [0.9 \quad 0.8] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [1.98] = [4.48 \quad 7.78 \quad 11.18]$$

$$Loss_{epoch=1} = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} [(-3.52)^2 + 1.78^2 + 1.18^2] = 5.6504$$

EPOCH 2

- Forward propagation has been applied above to compute  $A_{guess=1}$
- Apply Back Propagation to compute  $\nabla_W L_{epoch=1}, \nabla_b L_{epoch=1}$

$$\frac{\partial L}{\partial A_j} = \frac{2}{m} (A_j - Y_j) \text{ for } j=0, \dots, m-1 \quad \nabla_A L = \left[ \frac{2}{3} (4.48 - 8) \quad \frac{2}{3} (7.78 - 6) \quad \frac{2}{3} (11.18 - 10) \right] =$$

$$[-2.3467 \quad 1.1867 \quad 0.7867]$$

$$\begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \frac{\partial A_2}{\partial Z_2} \end{bmatrix} = [1 \quad 1 \quad 1]$$

$$\nabla_Z L = \nabla_A L * \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \frac{\partial A_2}{\partial Z_2} \end{bmatrix} = [-2.3467 \quad 1.1867 \quad 0.7867] * [1 \quad 1 \quad 1] = [-2.3467 \quad 1.1867 \quad 0.7867]$$

$$\nabla_W L = \nabla_Z L X^T = [-2.3467 \quad 1.1867 \quad 0.7867] \begin{bmatrix} 1 & 2 \\ 2 & 5 \\ 4 & 7 \end{bmatrix} = [3.1733 \quad 6.7467]$$

$$\nabla_b L = \sum_{j=0}^{m-1} \nabla_Z L_j = [-0.3733]$$

# Training Algorithm - Example

- Update:

$$W_{epoch=2} = W_{epoch=1} - \alpha \nabla_W L_{epoch=1} = [0.9 \quad 0.8] - 0.01 * [3.1733 \quad 6.7467] = [0.8683 \quad 0.7325]$$

$$b_{epoch=2} = b_{epoch=1} - \alpha \nabla_b L_{epoch=1} = [1.98] - 0.01 * [-0.3733] = [1.9837]$$

- Apply Forward Propagation with  $W_{epoch=2}$  and  $b_{epoch=2}$

$$A_{epoch=2} = Z_{epoch=2} = W_{epoch=2}X + b_{epoch=2} = [0.8686 \quad 0.7325] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [1.9837] \\ = [4.3171 \quad 7.3829 \quad 10.5845]$$

$$Loss_{epoch=2} = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} [(-3.6829)^2 + 1.3829^2 + 0.5845^2] = 5.2727$$

- Loss dropped from 5.6504 in EPOCH 1 to 5.2727 in EPOCH 2
- In general will use trial and error to adjust learning rate  $\alpha$

# Prediction Algorithm

Prediction algorithm makes use parameters computed in Training Algorithm

Input new input feature matrix  $\tilde{X}$  ( $d \times p$  -  $d$  features and  $p$  samples)

Use  $W$  and  $b$  computed by Training Algorithm

1. Perform Forward Propagation to compute output  $\tilde{A}$

Prediction is  $\tilde{A}$  ( $1 \times p$  values)

# Prediction Algorithm - Example

- From Training Algorithm Example:

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10]$$

- After 2 EPOCHs of Training Algorithm:

$$W_{epoch=2} = [0.8683 \quad 0.7325] \quad b_{epoch=2} = [1.9837]$$

- Prediction: Apply Forward Propagation with  $W_{epoch=2}$  and  $b_{epoch=2}$

$$\begin{aligned} A_{epoch=2} = Z_{epoch=2} &= W_{epoch=2}X + b_{epoch=2} = [0.8683 \quad 0.7325] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [1.9837] \\ &= [4.3170 \quad 7.3828 \quad 10.5844] \end{aligned}$$

# Accuracy Calculation

- Accuracy calculation compares value vector  $\tilde{Y}$  to prediction
- Can use Mean Squared Error function to measure accuracy for regression
  - Mean Squared Error is not as informative for Classification as for Regression so a different measure will be introduced later in the chapter
- In this section, use Mean Absolute Error

Assume Training has been performed

Assume Prediction Algorithm has been applied to yield value vector  $\tilde{A}$

1. Accuracy defined by mean absolute error

$$Accuracy = \frac{1}{m} \sum_{j=0}^{m-1} |\tilde{A}_j - \tilde{Y}_j|$$

# Accuracy Calculation - Example

- From example prediction example:

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10]$$

- Prediction:

$$A_{epoch=2} = [4.3170 \quad 7.3828 \quad 10.5844]$$

- Accuracy:

$$Accuracy = \frac{1}{m} \sum_{j=0}^{m-1} |A_j - Y_j| = \frac{1}{3} [| -3.6830 | + | 1.3828 | + | 0.5844 |] = 1.8834$$



# Linear Regression – Summary

Component	Algorithm	Details
Training Data		Input m data points: X (dxm-dimensional feature matrix) Y vector of values (1xm)
Function Structure	Forward Propagation	Linear: $Z = WX + b$ (Z is row vector of length m, W is row vector of length d, b is scalar) Activation function: $f(z) = z$ $A = f(Z)$ (1xm)
Loss Function		Mean Squared Error: $L = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2$
Derivative	Back Propagation	Compute gradients $\nabla_W L$ and $\nabla_b L$
Training	Fitting using Gradient Descent to minimize Loss	Find W, b that minimizes loss Initial guess: $W_{epoch=0}, b_{epoch=0}$ Choose Learning Rate: $\alpha > 0$ For epoch=1,2,3... (for fixed number of epochs or until Loss reduced sufficiently) apply forward and back propagation to compute $\nabla_W L_{epoch=i-1}, \nabla_b L_{epoch=i-1}$ $W_{epoch=i} = W_{epoch=i-1} - \alpha \nabla_W L_{epoch=i-1}$ $b_{epoch=i} = b_{epoch=i-1} - \alpha \nabla_b L_{epoch=i-1}$
Prediction	Apply Forward Propagation	Using computed W and b from Training Algorithm Given new input feature matrix $\tilde{X}$ Perform Forward Propagation to compute $\tilde{A}$ , the prediction for values

# Linear Regression – Jupyter Notebook Demo

- Open file IntroML/Examples/Chapter1/Chapter2.1\_LinearRegression.ipynb
- Has examples of
  - Forward Propagation
  - Loss Function
  - Backward Propagation
  - Training Algorithm
  - Prediction Algorithm
  - Accuracy Calculation

## 2.2 Derivative Testing

# Derivative Testing

Goal of this Section:

- Present testing algorithm for comparing gradients computed using forward/back propagation with approximate estimates

# Motivation for Derivative Testing

- The forward/back propagation approach for computing gradients described in the previous section is involved
- The formulas for neural networks presented in the next chapter are more complicated
- To gain confidence in machine learning training code, it is useful to provide a check of the gradients produced by forward/back propagation

# Difference Formula for Derivatives

- Let  $L = L(p_0, p_1, p_2, \dots, p_d)$ . Definition of partial derivative is:

$$\frac{\partial L}{\partial p_i} = \lim_{\varepsilon \rightarrow 0} \frac{L(p_0, p_1, \dots, p_i + \varepsilon, \dots, p_d) - L(p_0, p_1, \dots, p_i, \dots, p_d)}{\varepsilon}$$

- Forward difference formula: pick small  $\varepsilon$  (eg:  $10^{-5}$ ) - error proportional to  $\varepsilon$  or better as  $\varepsilon \rightarrow 0$ )

$$\frac{\partial L}{\partial p_i} \approx \frac{L(p_0, p_1, \dots, p_i + \varepsilon, \dots, p_d) - L(p_0, p_1, \dots, p_i, \dots, p_d)}{\varepsilon}$$

- Centered differences formula: (error is proportional to  $\varepsilon^2$  or better as  $\varepsilon \rightarrow 0$  – this is more accurate!)

$$\frac{\partial L}{\partial p_i} \approx \frac{L(p_0, p_1, \dots, p_i + \varepsilon, \dots, p_d) - L(p_0, p_1, \dots, p_i - \varepsilon, \dots, p_d)}{2\varepsilon}$$

- Apply centered differences approach for each variable (apply  $d+1$  times)

# Derivative Testing - Example

- Consider Backpropagation Example in Section 2.1

$$X = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} \quad Y = [8 \quad 6 \quad 10] \quad W = [1 \quad 1] \quad b = [2]$$

- From the Back Propagation example of Section 2.1, we have

$$\frac{\partial L}{\partial W_0} = 10, \frac{\partial L}{\partial W_1} = 20, \frac{\partial L}{\partial b} = 2$$

- Approximate  $\frac{\partial L}{\partial W_0}$  by bumping  $W_0$  by plus  $+\varepsilon$  and  $-\varepsilon$  (choose  $\varepsilon=0.1$ )

- $\varepsilon=0.1$  case:

$$Z = WX + b = [1.1 \quad 1] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [2] = [5.1 \quad 9.2 \quad 13.4] \quad A = Z$$

$$L_+ = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} ((-2.9)^2 + 3.2^2 + 3.4^2) = 10.07$$

- $\varepsilon=-0.1$  case:

$$Z = WX + b = [0.9 \quad 1] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [2] = [4.9 \quad 8.8 \quad 12.6] \quad A = Z$$

$$L_- = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} ((-3.1)^2 + 2.8^2 + 2.6^2) = 8.07$$

Hence:

$$\frac{\partial L}{\partial W_0} \approx \frac{L_+ - L_-}{2\varepsilon} = \frac{10.07 - 8.07}{2(0.1)} = 10 \quad (\text{this matches exact derivative})$$

# Derivative Testing - Example

- Approximate  $\frac{\partial L}{\partial W_1}$  by bumping  $W_1$  by plus  $+\varepsilon$  and  $-\varepsilon$  (choose  $\varepsilon=0.1$ )

- $\varepsilon=0.1$  case:

$$Z = WX + b = [1 \quad \mathbf{1.1}] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [2] = [5.2 \quad 9.5 \quad 13.7] \quad A = Z$$

$$L_+ = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} ((-2.8)^2 + 3.5^2 + 3.7^2) = 11.26$$

- $\varepsilon=-0.1$  case:

$$Z = WX + b = [1 \quad \mathbf{0.9}] \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + [2] = [4.8 \quad 8.5 \quad 12.3] \quad A = Z$$

$$L_- = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} ((-3.2)^2 + 2.5^2 + 2.3^2) = 7.26$$

Hence:

$$\frac{\partial L}{\partial W_1} \approx \frac{L_+ - L_-}{2\varepsilon} = \frac{11.26 - 7.26}{2(0.1)} = 20 \quad (\text{this matches exact derivative})$$



# Derivative Testing - Example

- Approximate  $\frac{\partial L}{\partial b}$  by bumping  $b$  by plus  $+\varepsilon$  and  $-\varepsilon$  (choose  $\varepsilon=0.1$ )
- $\varepsilon=0.1$  case:

$$Z = WX + b = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + \begin{bmatrix} 2.1 \end{bmatrix} = \begin{bmatrix} 5.1 & 9.1 & 13.1 \end{bmatrix} \quad A = Z$$

$$L_+ = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} ((-2.9)^2 + 3.1^2 + 3.1^2) = 9.21$$

- $\varepsilon=-0.1$  case:

$$Z = WX + b = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + \begin{bmatrix} 1.9 \end{bmatrix} = \begin{bmatrix} 4.9 & 8.9 & 12.9 \end{bmatrix} \quad A = Z$$

$$L_- = \frac{1}{m} \sum_{j=0}^{m-1} (A_j - Y_j)^2 = \frac{1}{3} ((-3.1)^2 + 2.9^2 + 2.9^2) = 8.81$$

Hence:

$$\frac{\partial L}{\partial b} \approx \frac{L_+ - L_-}{2\varepsilon} = \frac{9.21 - 8.81}{2(0.1)} = 2 \quad (\text{this matches exact derivative})$$

- Derivatives match approximations exactly – this occurs for linear regression as Loss depends quadratically on parameters  $W$  and  $b$ , but will not occur for Logistic Regression and Neural Network, in general, because activation and loss functions are more complicated

# Concatenating and Loading Parameters

- To efficiently bump each parameter it is convenient to store all parameter in a single vector
- Concatenation combines all parameters:

Original format:  $W = [W_0 \ W_1 \ \dots \ W_{d-1}]$  and scalar  $b$

Concatenated format:  $[W_0 \ W_1 \ \dots \ W_{d-1} \ b]$  (vector with all parameters)

- Process of loading takes parameters in concatenated form and puts parameters back into original format (separate  $W$  and  $b$ )
- Testing approach:
  - Use concatenated form to bump parameters
  - Put back into original format to perform forward propagate and compute Loss after parameters are bumped

# Derivative Testing Algorithm

Assign  $W$  and  $b$

Input Training Data: feature matrix  $X$  and values  $Y$

1. Perform Forward and Back Propagation to compute  $\nabla_W L, \nabla_b L$
2. Concatenate original parameters  $W$  and  $b$  and gradient vectors
3. Loop over  $i = 0, 1, \dots, d$  ( $d+1$  parameters in  $W$  and  $b$ )
  - Add  $\varepsilon$  to parameter  $i$  in original concatenated parameter list
  - Load parameters back into  $W$  and  $b$
  - Forward propagate and compute Loss  $L(p_i + \varepsilon)$
  - Subtract  $\varepsilon$  from parameter  $i$  in original concatenated parameter list
  - Load parameters back into  $W$  and  $b$
  - Forward propagate and compute Loss  $L(p_i - \varepsilon)$
  - Estimate partial derivative with respect to  $i$ 'th parameter is  $(L(p_i + \varepsilon) - L(p_i - \varepsilon))/2\varepsilon$
4. Compare estimated partial derivatives to those computed in Steps 1,2

# Derivative Testing – Jupyter Notebook Demo

- Open file  
IntroML/Examples/Chapter1/Chapter2.2\_DerivativeTesting.ipynb
- Has example of
  - Derivative Testing

## 2.3 Code Design Review

# Coding Overview

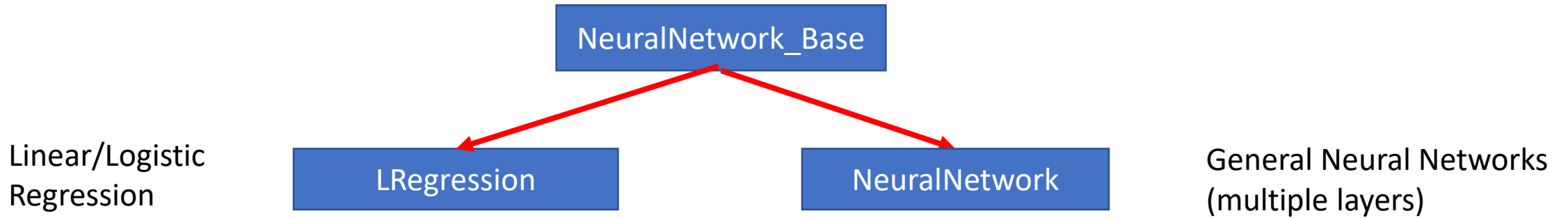
Goal of this Section:

- Present overview of code design, including
  - Principal classes employed
  - Numpy arrays

# Coding Overview

- Code employs object-oriented approach
  - Two principal classes
    - NeuralNetwork\_Base Class
    - Optimizer\_Base Class
- Additional codes
  - Activation functions
  - Loss functions
  - Plotting functions
  - Unit test
- Design considerations:
  - Numpy array is key variable type
  - Use interfaces for framework Tensorflow as a rough guide

# NeuralNetwork\_Base Class



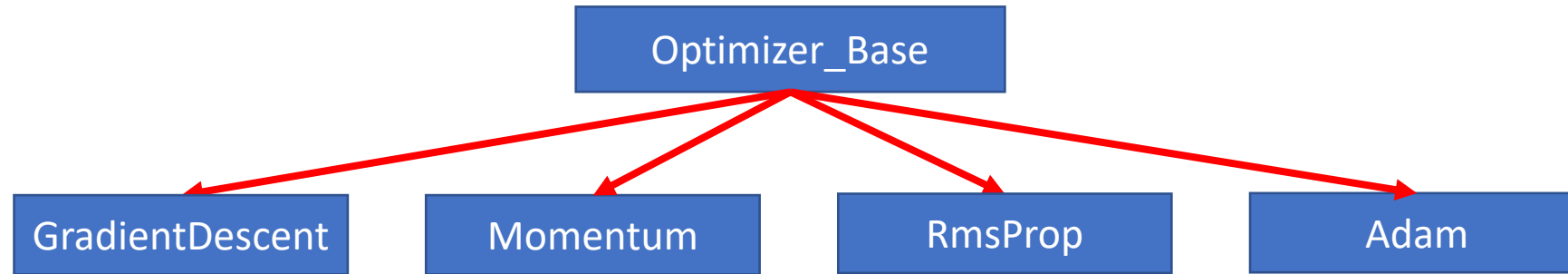
## Key Methods

- Forward Propagation
- Back Propagation
- Compute Loss
- Fit (for training)
- Predict
- Test Derivative
- Concatenate Parameters
- Load Parameters
- Compute accuracy

Will attempt to make methods general to handle both Linear/Logistic Regression and Neural Networks



# Optimizer\_Base Class



## Key Methods

- Update

# Code Design - Arrays

As part of the coding design, all relevant arrays will be defined explicitly as 2d numpy arrays

- Feature matrix  $X$ 
  - This naturally is a 2d object (number of features x number of samples)
- Value Vector  $Y$ 
  - This is a row vector – will be explicitly made to have dimensions  $(1 \times m)$
- Parameters  $W$  and  $b$  and gradients
  - For Linear/Logistic regression,  $W$  and  $\nabla_W L$  are row vectors of length  $d$  – will be explicitly made to have dimensions  $(1 \times d)$
  - $b$  is a scalar – will be explicitly made to have dimensions  $(1 \times 1)$
  - For neural networks  $W$  and  $\nabla_W L$  will naturally be 2d objects
  - For neural networks, in general,  $b$  and  $\nabla_b L$  will be column vectors – will be explicitly made to be column vectors – dimensions  $(n \times 1)$
- Computed Values  $A$  and  $Z$ 
  - For Linear/Logistic Regression,  $A$  and  $Z$  are row vectors – will have dimensions  $(1 \times m)$
  - For neural networks,  $A$  and  $Z$  will naturally be 2d objects

# 2.4 Code Walkthrough

## Version 1.1

# Coding Walkthrough: Version 1.1

Goal of this Section:

- Walkthrough creation of code to perform test of accuracy of forward/back propagation for computing gradients

# Coding Walkthrough: Version 1.1 To Do

File/Component	To Do
NeuralNetwork_Base	Create NeuralNetwork_Base class and methods to be used for both Logistic/Linear Regression and Neural Networks
LRegression	Create class derived from NeuralNetwork_Base with methods specific to Linear and Logistic Regression
functions_loss	Create functions for mean square error loss function and its derivative
functions_activation	Create functions for linear activation and its derivative
unittest_forwardbackprop	Create functions for performing test of derivative calculation

# NeuralNetwork\_Base Class – Attributes

Variable	Type	Description
nlayer	integer	Number of layers <ul style="list-style-type: none"><li>• Equals 1 for Linear and Logistic Regression</li><li>• In general greater than 1 for Neural Networks</li></ul>
info	list indices: 0,1,...,nlayer-1	info[k] is a dictionary containing information for layer = k Keys: <ul style="list-style-type: none"><li>• nIn: (integer) number of unit in previous layer (number of features for layer 0)</li><li>• nOut: (integer) number of units in current layer</li><li>• activation: (string) activation function type</li><li>• A: (numpy array) result after activation for current layer</li><li>• param: (dictionary) parameter matrices (keys W, b)</li><li>• param_der: (dictionary) derivatives of parameter matrices (keys W, b)</li><li>• optimizer: (dictionary) optimizer class objects (keys W,b)</li></ul>
loss_fun	string	Name of loss function

# NeuralNetwork\_Base Class – Methods

Method	Input	Description
get_A	layer (integer)	Return: A, the result of Forward Propagation for specified layer
get_param	layer (integer) order (string): “param” or “param_der” label (string): “W” or “b”	Return: parameters W or b or gradients $\nabla_W L$ , $\nabla_b L$ for specified layer, order, and label
compile	optimizer (dictionary) loss_fun (string)	Takes in loss function and optimizer information and constructs optimizer object for each parameter and layer Return: nothing
compute_loss	Y (numpy array)	Return: loss for output (label) vector Y assuming forward propagation has been performed
test_derivative	X (numpy array) Y (numpy array) eps (float)	Return: difference between exact (computed using forward/back propagation) and approximate (computed using centered differences with bump eps) gradients $\nabla_W L$ , $\nabla_b L$

# LRRegression – Methods

Method	Input	Description
<code>__init__</code>	nfeature (integer) activation (string)	Initialization routine that takes in the number of features and the activation function (“linear” for regression) Return: nothing
<code>forward_propagation</code>	X (numpy array)	Performs forward propagation using feature matrix X to compute approximation A and updates info variable for key “A” Returns: nothing
<code>back_propagation</code>	X (numpy array) Y (numpy array)	Performs back propagation using feature matrix X and label vector Y Returns: nothing
<code>concatenate_param</code>	order (string): “param” or “param_der”	Concatenates all parameters in W and b or in the their gradient vectors into a single numpy row vector
<code>load_param</code>	flat (numpy array) order (string): “param” or “param_der”	Takes values from flat (row vector) and puts them back into W or b or gradient objects



# Activation and Loss Functions

Function	Input	Description
functions_activation. activation	activation_fun (string) Z (numpy array)	Applies activation function $f(z)$ to entries in Z for specified function Return: $f(Z)$
functions_activation. activation_der	activation_fun (string) Z (numpy array)	Applies derivative of activation function to entries in Z for specified function Return: $f'(Z)$
functions_loss. loss	loss_fun (string) A (numpy array) Y (numpy array)	Computes loss function given approximation A, label vector Y, and specified function Return: Loss
functions_loss. loss_der	loss_fun (string) A (numpy array) Y (numpy array)	Computes gradient of loss function with respect to elements of A for approximation A, label vector Y, and specified function Return: $\nabla_A L$

# Unit Test Functionality

```
In [1]: import unittest

class Test(unittest.TestCase):
    def test1(self):
        x = 7
        y = 8
        z1 = (x+y)*(x+y)
        z2 = x*x + 2*x*y + y*y
        error = abs(z1-z2)
        self.assertLessEqual(error, 1e-7)

if __name__ == "__main__":
    #this is command in python when running in command window
    #unittest.main()
    # this is command in the jupyter notebook
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

.

---

Ran 1 test in 0.016s

OK

- Use functionality in unittest package
- Documentation at <https://docs.python.org/3.7/index.html>
- Create a class derived from unittest.TestCase
- Individual unit tests are set up as methods of the class
- Test should have “assert” command which determines pass or fail
- Use unittest.main to run tests
- Will get OK if test passes

# Unit Test – Jupyter Notebook Demo

- Open file IntroML/Examples/Chapter1/Chapter2.1\_UnittestExample.ipynb
- Has example of
  - Unit test

# Unit Test for Forward/Back Propagation

Unit test method has following components:

1. Preparation of Data
  - Create random X and Y
2. Creation of LRegression object
  - Create instance of the LRegression class
3. Compilation
  - Specify loss function and optimizer (None)
4. Run test\_derivative method of LRegression object
  - This will compare forward/back propagation derivatives to approximations
5. Assert
  - Check if error less than or equal to tolerance (will use  $10^{-7}$ )

# Code Version 1.1 Walkthrough

# 2.5 Code Walkthrough

## Version 1.2

# Coding Walkthrough: Version 1.2

Goal of this Section:

- Walkthrough creation of code to perform linear regression training and prediction

# Coding Walkthrough: Version 1.2 To Do

File/Component	To Do
NeuralNetwork_Base	Add methods for training, prediction, updating parameters, and computing accuracy of prediction
Optimizer_Base	Create Optimizer_Base class and a constructor to create optimizer objects
GradientDescent	Create GradientDescent class derived from Optimizer_Base
Plotting	Create function to plot training data, normal equations result, and linear regression prediction as well as accuracy and loss versus epoch
Driver	Create driver for linear regression



# NeuralNetwork\_Base Class – Methods

Method	Input	Description
update_param		Applies optimizer to update parameters Return: nothing
fit	X (numpy array) Y (numpy array) epochs (integer)	Applies training algorithm for specified number of epochs using feature matrix X and output information vector Y. Uses approach defined in optimizer input in compile method to compute updates. Return: history dictionary containing loss and accuracy at each epoch
predict	X (numpy array)	Applies prediction algorithm to compute output vector Y for input feature/information matrix X Return: predicted output values
accuracy	Y (numpy array) Y_pred (numpy array)	Computes accuracy comparing label vector Y to predicted results Y_pred Return: accuracy (float)

# GradientDescent – Methods

Method	Input	Description
<code>__init__</code>	learning_rate (float)	Takes relevant parameters Return: nothing
update	gradient (numpy array)	Computes update to be used by optimization algorithm. Input is gradient Return: update

# Linear Regression Driver

Driver has following components:

## 1. Preparation of Data

- Create data or load from external file or create in external program

## 2. Creation of Model Object

- Create instance of the LRegression class

## 3. Compilation

- Specify optimizer object and loss function

## 4. Training

- Input training data X and Y and specify number of epochs

## 5. Prediction

- Predict output for new input information X using learned parameters

# Plotting Functions

Will create functions for:

## 1. Plotting Loss and Accuracy

- Loss and accuracy are computed during training
- Plot these quantities as function of epoch on separate graphs

## 2. Plotting Training Data, Predicted Line, Normal Equations Line

- On the same graph plot:
  - Training data
  - Line predicted by training algorithm
  - Line predicted by the normal equations approach

# Code Version 1.2 Walkthrough

## 2.6 Logistic Regression: Mathematical Foundations

# Logistic Regression: Mathematical Foundations

Goal of this Section:

- Extend the mathematical foundations for linear regression to the case of logistic regression, including:
  - Format of input data
  - Function structure and parameters
  - Training algorithm
  - Prediction algorithm

# Logistic Regression and Linear Regression

- Linear Regression used for modeling real values ( $Y_j$  are real numbers)
- Logistic Regression for binary classification ( $Y_j$  are labels 0 or 1)
  - Binary classification 2 possibilities – arbitrarily assign 0 to one possibility and 1 to the other (eg cat is 0 and dog is 1, for x-rays 0 is normal and 1 is broken, etc)
- Underlying mathematics and code development for Linear Regression can be used for Logistic Regression
- Principal Differences between Logistic and Linear Regression:
  - Activation Function:
    - Need suitable activation function to produce 0 or 1 output
    - Linear activation function (can take on values from  $-\infty$  to  $\infty$ ) is not suitable
  - Loss Function
    - Need suitable loss function
    - Mean Squared Error loss function not suitable for Logistic Regression



# Binary Classification

## Training Data:

- Input Information: points in  $(x_0, x_1)$  plane
- Output Information: label 0 (red) or 1 (blue) for each point

## Goal:

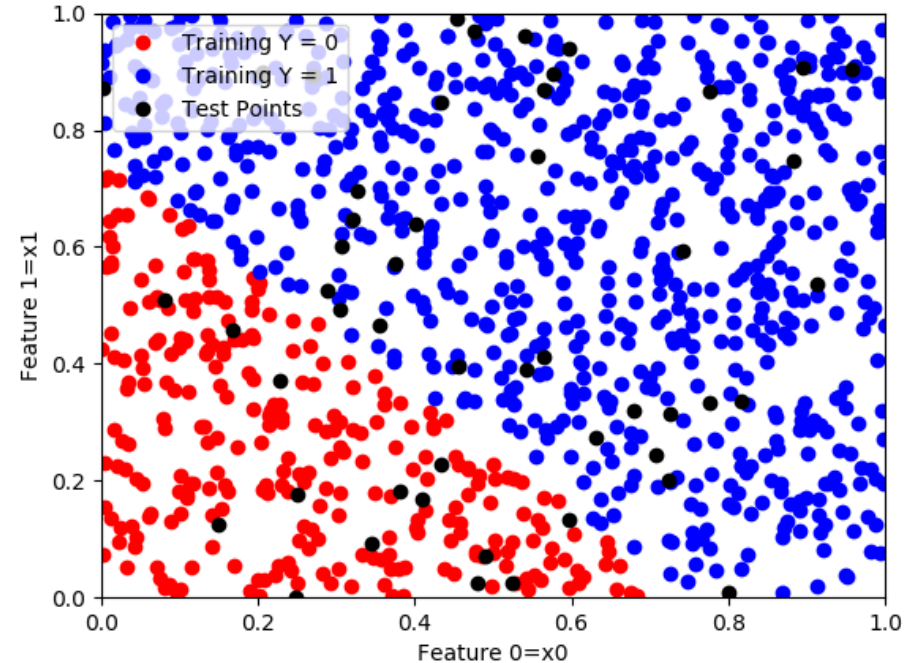
- Find function that best fits 0 and 1 labels in training data

## Prediction:

- Using function, determine label for new input test points (black points in picture)

## Logistic Regression:

- Simple approach for binary classification (builds on Linear Regression)



# Motivating Example

- Training data for sample  $j$ :

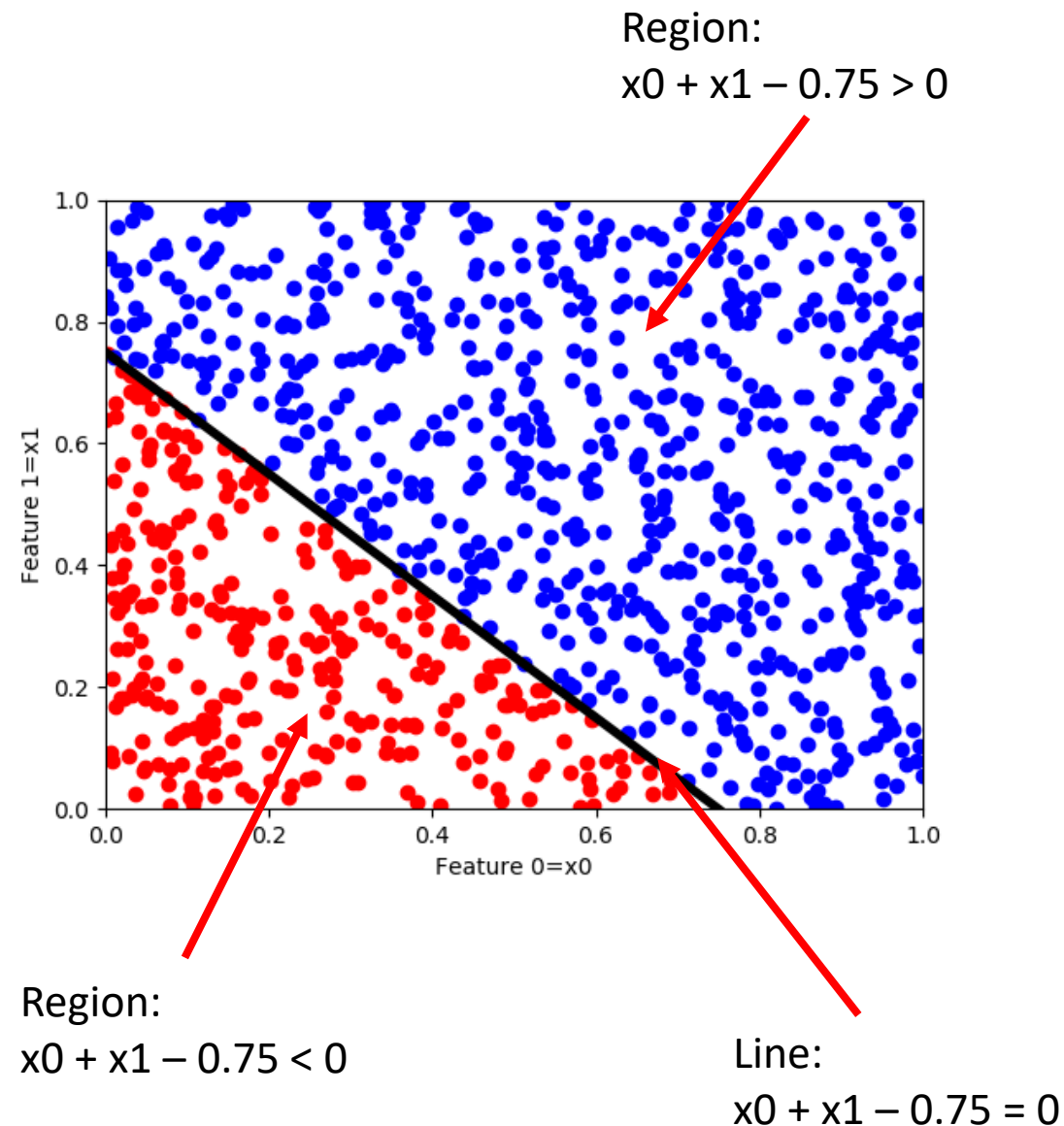
$$\left( \begin{bmatrix} X_{0j} \\ X_{1j} \end{bmatrix}, Y_j \right)$$

Here  $(X_{0j}, X_{1j})$  is the point in the plane and  $Y_j$  is the label 0 or 1.

- Define parameters  $W = [W_0 \ W_1]$  and  $b$   
 $Z_j = WX_j + b = W_0X_{0j} + W_1X_{1j} + b$

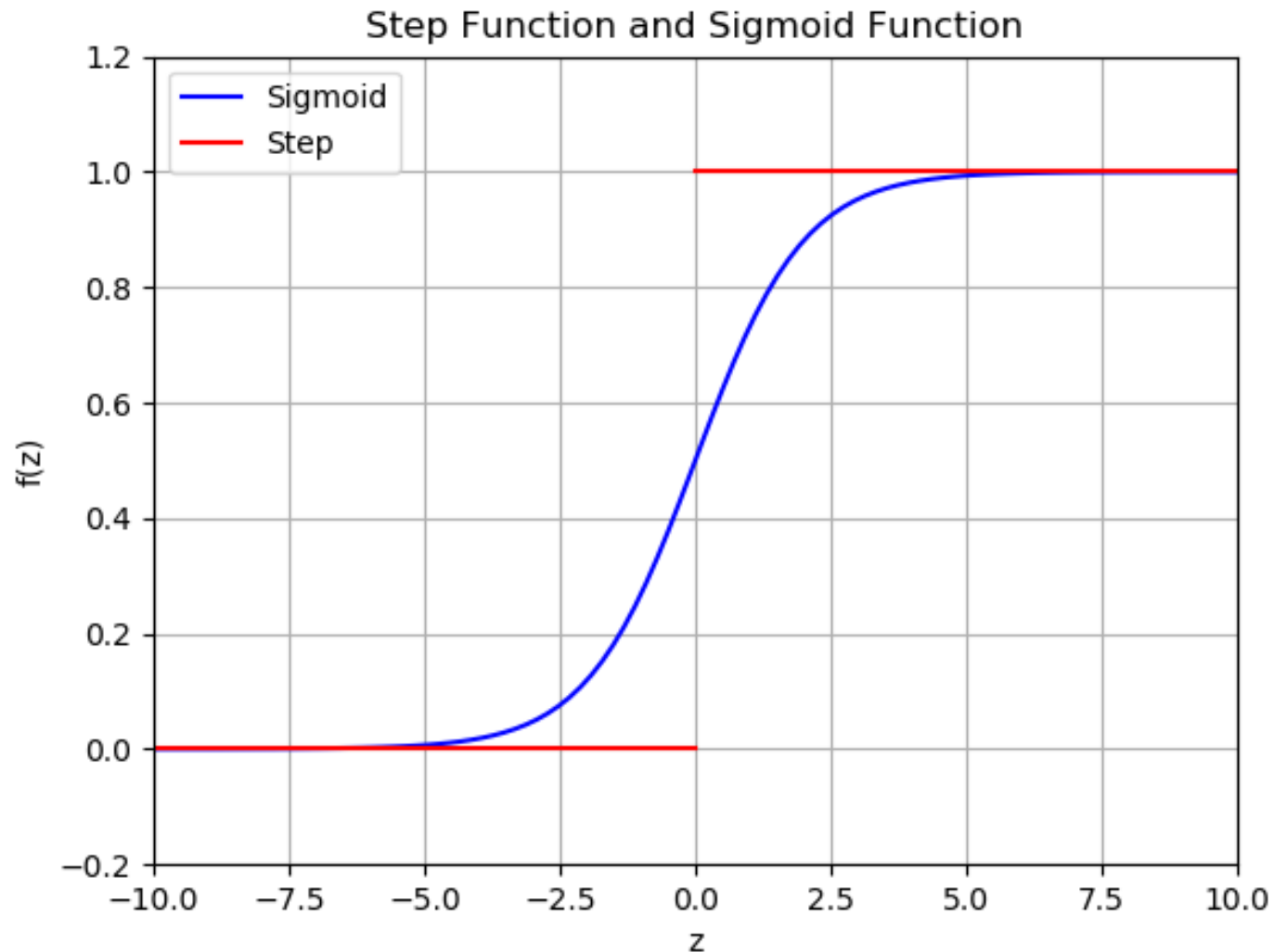
- If we choose  $W = [1 \ 1]$  and  $b = -0.75$ ,  
then appropriate activation function

$$f(Z_j) = \begin{cases} 1 & \text{if } Z_j \geq 0 \\ 0 & \text{if } Z_j < 0 \end{cases}$$



# Sigmoid Activation Function

- Step function is not best choice, as we want activation function to be differentiable
- Use instead the sigmoid activation function
$$f(z) = \frac{1}{1 + e^{-z}}$$
- Sigmoid function between 0 and 1  
 $f(z) \rightarrow 1$  as  $z \rightarrow \text{infinity}$   
 $f(z) \rightarrow 0$  as  $z \rightarrow -\text{infinity}$



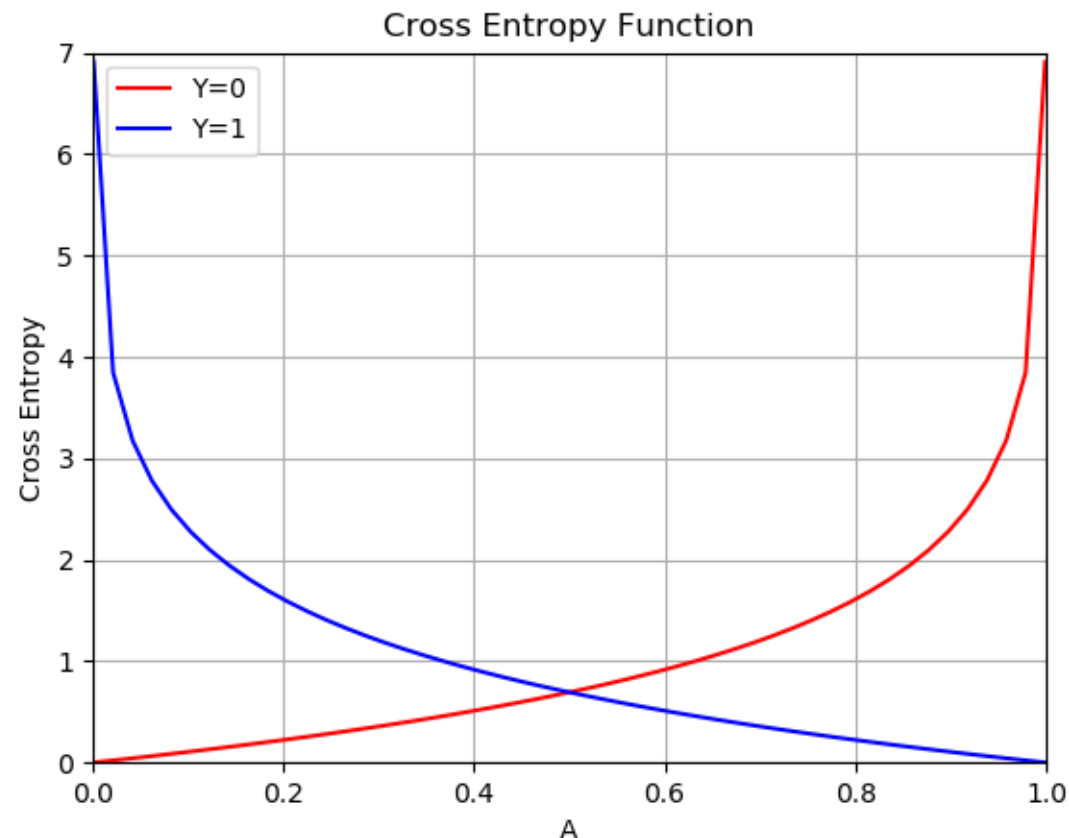
# Binary Cross Entropy Loss Function

- Mean Squared Error loss function is not appropriate for logistic regression

- Use Binary Cross Entropy loss function instead:

$$\text{Loss} = -Y\ln(A) + (1 - Y)\ln(1 - A)$$

- When  $Y = 0$ , Loss decreases as  $A$  goes to 0
- When  $Y = 1$  Loss decreases as  $A$  goes to 1



# Logistic Regression: General Approach

General approach has following components and phases:

(1) Training Data

(2) Function Structure

- Defines general form of the function with unknown parameters
- Process of applying function structure is called Forward Propagation

(3) Loss Function

- Used to measure effectiveness of function structure and choice of parameters

(4) Training Phase

- Uses optimization to determine function parameters that minimize loss function for training data
- Process of computing derivatives is called Back Propagation

(5) Prediction Phase

- Applies forward propagation using parameters determined in Training Phase to predict outputs when new input data is provided

# Training Data

- For general logistic regression problem there are  $m$  data points, each consisting of a input information vector of length  $d$  and value  $Y$ :

- Data point  $j$ : input information (feature) vector:  $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \vdots \\ X_{d-1,j} \end{bmatrix}$  and output:  $Y_j$
- Define the feature matrix ( $d \times m$ ) and output vector ( $1 \times m$ ):

$$X = \begin{bmatrix} X_{00} & \dots & X_{0,m-1} \\ \dots & \dots & \dots \\ X_{d-1,0} & \dots & X_{d-1,m-1} \end{bmatrix} \quad Y = [Y_0 \quad \dots \quad Y_{m-1}]$$

# Training Data – Example Points in Plane

- For points in plane with 0 and 1 labels in motivating example, training data consists of points in the plane  $(X_0, X_1)$  with label  $Y$
- Suppose 4 data samples with points and labels:  $(1,1)$  label=0,  $(0.5,2)$  label = 1,  $(2,3)$ , label = 1,  $(4,2)$  label 0
- In this case each sample has 2 features. Feature matrix and value vector are:

$$X = \begin{bmatrix} 1 & 0.5 & 2 & 4 \\ 1 & 2 & 3 & 2 \end{bmatrix} \quad Y = [0 \quad 1 \quad 1 \quad 0]$$

# Function Structure – Forward Propagation

Forward Propagation is name applied to process of estimating output values using function structure

1. Input: feature matrix  $X$  ( $d \times m$   $d$  features and  $m$  sample points), parameter vector  $W$  ( $1 \times d$  vector) and scalar bias  $b$

2. Linear part: for  $j=0, \dots, m-1$

$$Z_j = w_0 X_{0j} + w_1 X_{1j} + w_2 X_{2j} + \dots + w_{d-1} X_{d-1j} + b$$

In vector form

$$Z = WX + b$$

3. Activation: apply sigmoid function  $f(z)$  to each element of  $Z$ :

$$A_j = f(Z_j) \quad j = 0, \dots, m - 1$$

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



# Logistic Regression Forward Propagation - Example

- Consider a case of 2 features and 3 data points (m=3)

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = [0 \quad 1 \quad 0]$$

- Assume that initial parameter values are:

$$W = [0.1 \quad 0.1] \quad b = [0.2]$$

- Forward Propagation:

$$Z = WX + b = [0.1 \quad 0.1] \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} + [0.2] = [0.1 \quad -0.1 \quad -0.2]$$

$$A = f(Z) = \left[ \frac{1}{1+e^{-0.1}} \quad \frac{1}{1+e^{0.1}} \quad \frac{1}{1+e^{0.2}} \right] = [0.5250 \quad 0.4750 \quad 0.4502]$$

# Logistic Regression: Loss Function

- Loss function is average of binary cross entropy function over sample points

$$Loss = L = -\frac{1}{m} \sum_{j=0}^{m-1} Y_j \ln(A_j) + (1 - Y_j) \ln(1 - A_j)$$

- For each sample  $j$ , only one of  $Y_j \ln(A_j)$  or  $(1 - Y_j) \ln(1 - A_j)$  is non-zero, as  $Y_j$  is 0 or 1

# Training Phase

- Training phase attempts to find suitable coefficients  $W$  and  $b$  by minimizing loss function when applied to training data
- From multi-variable calculus, Loss function has a local minimum when the gradients are 0

$$\nabla_W L = 0 \text{ and } \nabla_b L = 0$$

- Can solve these equations analytically for Linear Regression, but not in general case (Logistic Regression and Neural Networks)
- Use optimization algorithm (example: Gradient Descent) to minimize Loss function
  - Need to compute the above gradients

# Derivative of Loss

- Loss function given by:

$$L = -\frac{1}{m} \sum_{j=0}^{m-1} Y_j \ln(A_j) + (1 - Y_j) \ln(1 - A_j)$$

- Differentiating the log terms, we get:

$$\frac{\partial L}{\partial A_j} = -\frac{1}{m} \left[ \frac{Y_j}{A_j} - \frac{1 - Y_j}{1 - A_j} \right], \quad j = 0, \dots, m - 1$$

# Derivative of Activation

$A_j$  related to  $Z_j$  by:


$$A_j = f(Z_j) = \frac{1}{1 + e^{-Z_j}}, \quad j = 0, \dots, m - 1$$

Derivatives given by

$$\frac{\partial A_j}{\partial Z_j} = \frac{e^{-Z_j}}{(1 + e^{-Z_j})^2}, \quad j = 0, \dots, m - 1$$

This can be simplified to:

As we will see when coding,  
this format saves memory  
as Z need not be saved

$$\frac{\partial A_j}{\partial Z_j} = \frac{e^{-Z_j}}{(1 + e^{-Z_j})^2} = \frac{1 + e^{-Z_j}}{(1 + e^{-Z_j})^2} - \frac{1}{(1 + e^{-Z_j})^2} = A_j - A_j^2$$


# Logistic Regression Back Propagation Algorithm

Input: feature matrix  $X$  and value vector  $Y$

1. Compute:

$$\nabla_A L = \left[ \frac{\partial L}{\partial A_0} \quad \frac{\partial L}{\partial A_1} \quad \cdots \quad \frac{\partial L}{\partial A_{m-1}} \right], \quad \frac{\partial L}{\partial A_j} = -\frac{1}{m} \left[ \frac{Y_j}{A_j} - \frac{1-Y_j}{1-A_j} \right], j = 0, \dots, m-1$$

3. Compute derivatives of A:  $\frac{\partial A_j}{\partial Z_j} = A_j - A_j^2, j = 0, \dots, m-1$

4. Compute gradient L with respect to Z:

$$\nabla_Z L = \nabla_A L * \left[ \frac{\partial A_0}{\partial Z_0} \quad \frac{\partial A_1}{\partial Z_1} \quad \cdots \quad \frac{\partial A_{m-1}}{\partial Z_{m-1}} \right] \text{ (component-wise multiplication)}$$

5. Compute gradient of L with respect to W and b (from Section 1.4):

$$\nabla_W L = \nabla_Z L X^T, \quad \nabla_b L = \sum_{j=0}^{m-1} \nabla_Z L_j$$

# Logistic Regression Back Propagation - Example

- Consider a case of 2 features and 3 data points ( $m=3$ )

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = [0 \quad 1 \quad 0]$$

- Assume that initial parameter values are:

$$W = [0.1 \quad 0.1] \quad b = [0.2]$$

- From Forward Propagation Example

$$A = [0.5250 \quad 0.4750 \quad 0.4502]$$

- Gradient of Loss with respect to A:

$$\nabla_A L = -\frac{1}{3} \left( \frac{Y}{A} - \frac{1-Y}{1-A} \right) = -\frac{1}{3} \begin{bmatrix} -\frac{1}{1-0.5250} & \frac{1}{0.4750} & -\frac{1}{1-0.4502} \end{bmatrix} = [0.7017 \quad -0.7017 \quad 0.6062]$$

$$\frac{\partial A_j}{\partial Z_j} = A_j - A_j^2 \quad \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \frac{\partial A_2}{\partial Z_2} \end{bmatrix} = [0.2494 \quad 0.2494 \quad 0.2475]$$

$$\nabla_Z L = \nabla_A L * \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \frac{\partial A_2}{\partial Z_2} \end{bmatrix} = [0.7017 \quad -0.7017 \quad 0.6062] * [0.2494 \quad 0.2494 \quad 0.2475] = [0.1750 \quad -0.1750 \quad 0.1501]$$

$$\nabla_W L = \nabla_Z L X^T = [0.1750 \quad -0.1750 \quad 0.1501] \begin{bmatrix} 1 & -2 \\ 2 & -5 \\ 4 & -8 \end{bmatrix} = [0.4252 \quad -0.6755]$$

$$\nabla_b L = \sum_{j=0}^{m-1} \nabla_Z L_j = 0.1501 \quad (\text{sum of entries of } \nabla_b L)$$

# Training Algorithm

- Other than change for activation function and loss function, Training Algorithm for Logistic Regression is the same as that for Linear Regression



# Prediction Algorithm

Prediction algorithm makes use of parameters computed in Training Algorithm

Input new input feature matrix  $\tilde{X}$

Use  $W$  and  $b$  computed by Training Algorithm

1. Perform Forward Propagation to determine  $\tilde{A}$ 
  - Round  $\tilde{A}$  to closest number 0 or 1 to get predicted label
- Can regard each entry of prediction vector  $\tilde{A}$  as probability that prediction is 1. ( $1 - \tilde{A}$  is probability that prediction is 0.)

# Logistic Regression Prediction - Example

- Consider a case of 2 features and 3 data points ( $m=3$ )

$$X = \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} \quad Y = [0 \quad 1 \quad 0]$$

- Assume that initial parameter values are:

$$W = [0.1 \quad 0.1] \quad b = [0.2]$$

- From the Forward Propagation Example slide

$$A = f(Z) = \left[ \frac{1}{1+e^{-0.1}} \quad \frac{1}{1+e^{0.1}} \quad \frac{1}{1+e^{0.2}} \right] = [0.5250 \quad 0.4750 \quad 0.4502]$$

- Round to 0 or 1 (round 0.5 up to 1) – predicted labels:  $[1 \quad 0 \quad 0]$

# Accuracy Calculation

Accuracy calculation compares actual vector label to predicted values

1. Perform Training
2. Let  $\tilde{X}$  denote feature matrix and  $\tilde{Y}$  denote related value vector (these may be same as used in training or completely different)
3. Apply prediction algorithm to  $\tilde{X}$  to get predicted value vector  $\tilde{P}$
4. Accuracy defined by:

$$Accuracy = \frac{1}{m} \sum_{j=0}^{m-1} (1 \text{ if } \tilde{P}_j = \tilde{Y}_j, 0 \text{ otherwise})$$

- Both value vector and predicted valued vector consist of 0 or 1 entries.  
Accuracy = 1 means prediction equals initial value vector for all entries.  
Accuracy = 0 means none of the entries in prediction vector match those in original value vector.

# Accuracy Calculation - Example

- Suppose that

$Y = [0 \quad 1 \quad 1 \quad 0 \quad 1]$  and predicted values after rounding =  $[1 \quad 0 \quad 1 \quad 0 \quad 1]$

- Accuracy calculation:  $Y$  matches predicted values for 3 out of 5 entries, so

*Accuracy* = 0.6

# Logistic Regression – Summary

Component	Subcomponent	Details
Training Data		Input m data points: X (dxm-dimensional feature matrix) Y vector of labels (0 or 1) (row vector of length m)
Function Structure	Forward Propagation	Linear: $Z = WX + b$ (Z is row vector of length m, W is row vector of length d, b is scalar) Activation function: $f(Z) = \frac{1}{1+e^{-Z}}$ $A = f(Z)$ (1xm vector)
Loss Function		Binary Cross Entropy: $L = -\frac{1}{m} \sum_{j=0}^{m-1} Y_j \log A_j + (1 - Y_j) \log(1-A_j)$
Derivative	Back Propagation	Compute $\nabla_W L$ and $\nabla_b L$
Training Algorithm	Train using Gradient Descent to minimize Loss	Find W, b that minimizes loss Initial epoch: $W_{epoch=0}, b_{epoch=0}$ Choose Learning Rate: $\alpha > 0$ For each epoch: (apply forward and back propagation to compute gradients) $W_{epoch=i} = W_{epoch=i-1} - \alpha \nabla_W L_{epoch=i-1}$ $b_{epoch=i} = b_{epoch=i-1} - \alpha \nabla_b L_{epoch=i-1}$ Perform fixed number of iterations or until Loss reduced sufficiently
Prediction Algorithm	Apply Forward Propagation	Using computed W and b from Training Algorithm Given new input feature matrix $\tilde{X}$ Perform Forward Propagation to compute $\tilde{A}$ and round to (0 or 1) to get predicted label

## 2.7 Code Walkthrough: Version 1.3

# Code Walkthrough Version 1.3

Goal of this Section:

- Walkthrough extension of linear regression codes to handle logistic regression

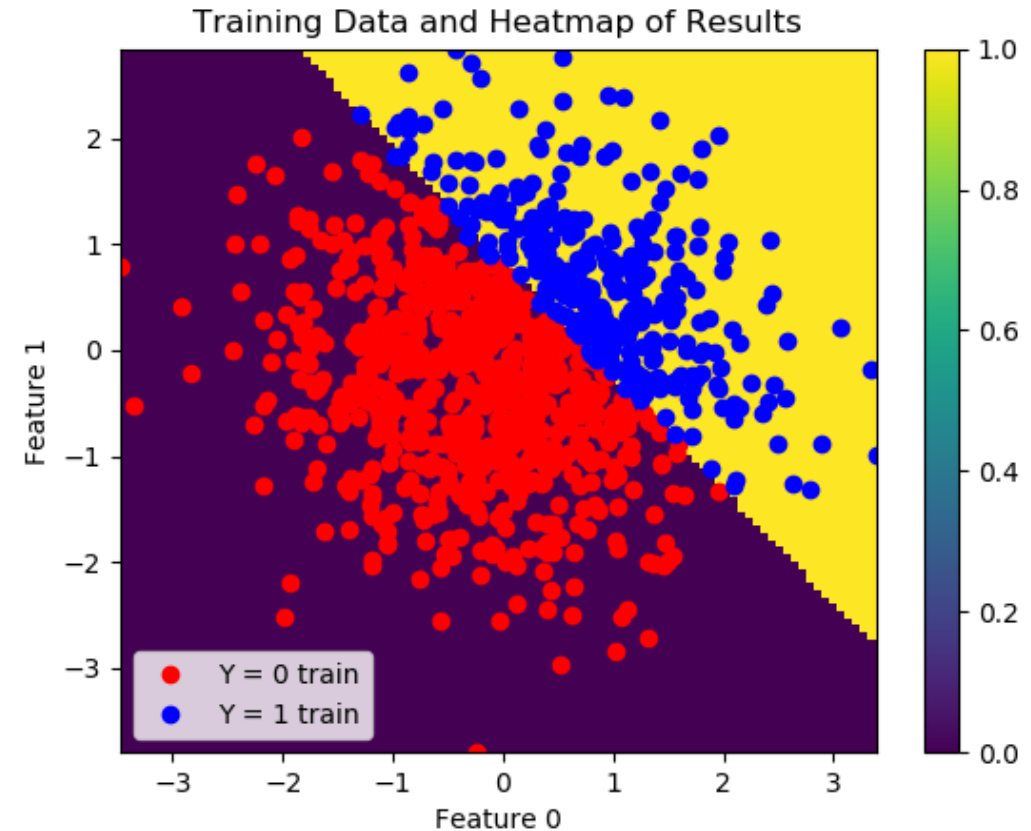
# Coding Walkthrough: Version 1.3 To Do

File/Component	To Do
NeuralNetwork_Base	Update accuracy method to handle logistic regression case
functions_loss	Add functions for binary cross entropy and its derivative
functions_activation	Add functions for sigmoid activation and its derivative
unittest_forwardbackprop	Add method for testing logistic regression case
driver	Add driver for logistic regression
plotting	Add routine for plotting training data and prediction



# Plotting Logistic Regression Results

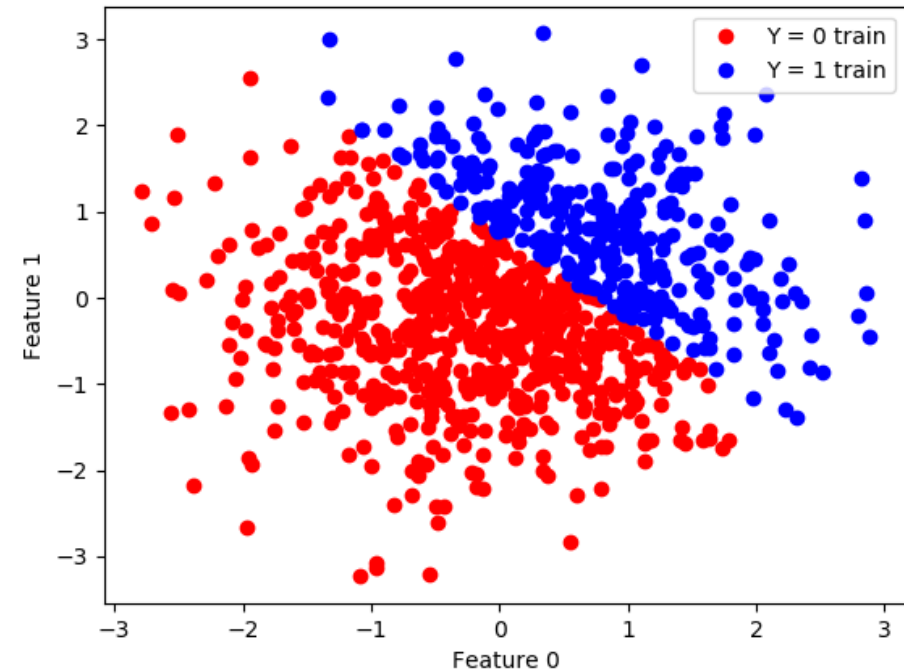
- Goal: produce plot of training data and predicted results to visually measure accuracy of predictions
- Training data: red and blue points
- Predicted Results: results predicted by model (purple is predicted 0 and yellow is predicted 1)



# Plotting Logistic Regression Training Data

Start with feature matrix  $X$  (2 features  $\times$   $m$  samples) and label vector  $Y$  ( $m$  samples)

1. Identify indexes for label = 0 – plot corresponding points red
2. Identify indexes for label = 1 – plot corresponding points blue



# Plotting Heatmap of Results

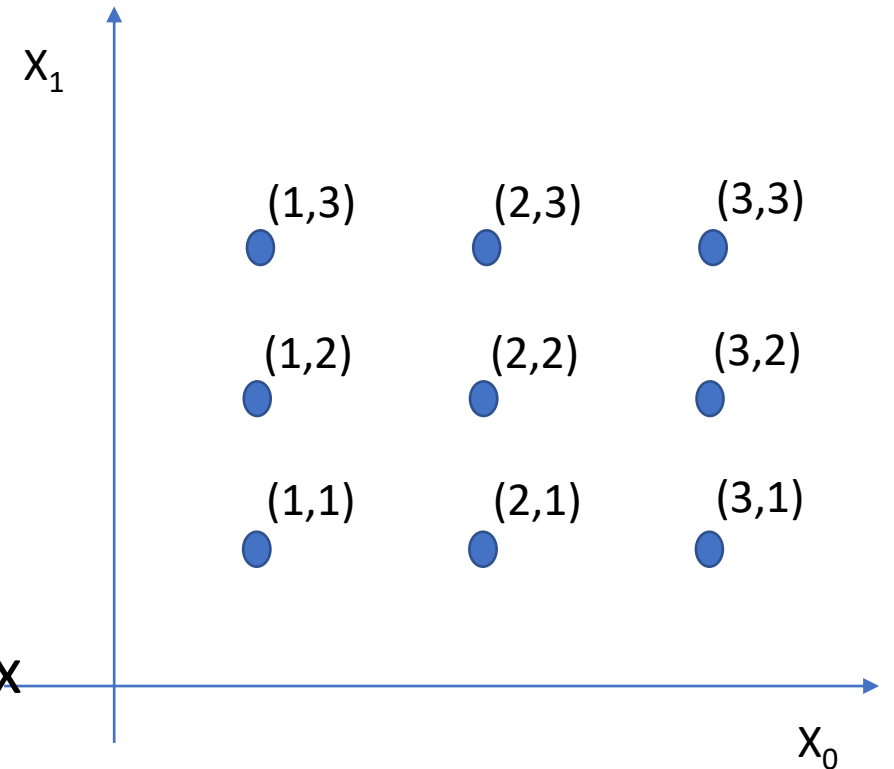
Assume that training algorithm has been performed with training data

1. Create grid of points similar to that on right
2. Points: (1,1), (2,1), (3,1), (1,2), (2,2), (3,2), (1,3), (2,3), (3,3)

3. Feature Matrix:

$$X = \begin{bmatrix} 1 & 2 & 3 & 1 & 2 & 3 & 1 & 2 & 3 \\ 1 & 1 & 1 & 2 & 2 & 2 & 3 & 3 & 3 \end{bmatrix}$$

4. Apply prediction algorithm with feature matrix representing grid of points to produce 0 or 1 label for each point
5. Convert prediction results to a 2d grid
6. Use pcolormesh function in matplotlib.pyplot applied to grid and labels to generate heatmap



# Code Version 1.3 Walkthrough