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

Copyright Satish Reddy 2020

2.1 Linear Regression: Mathematical Foundations

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

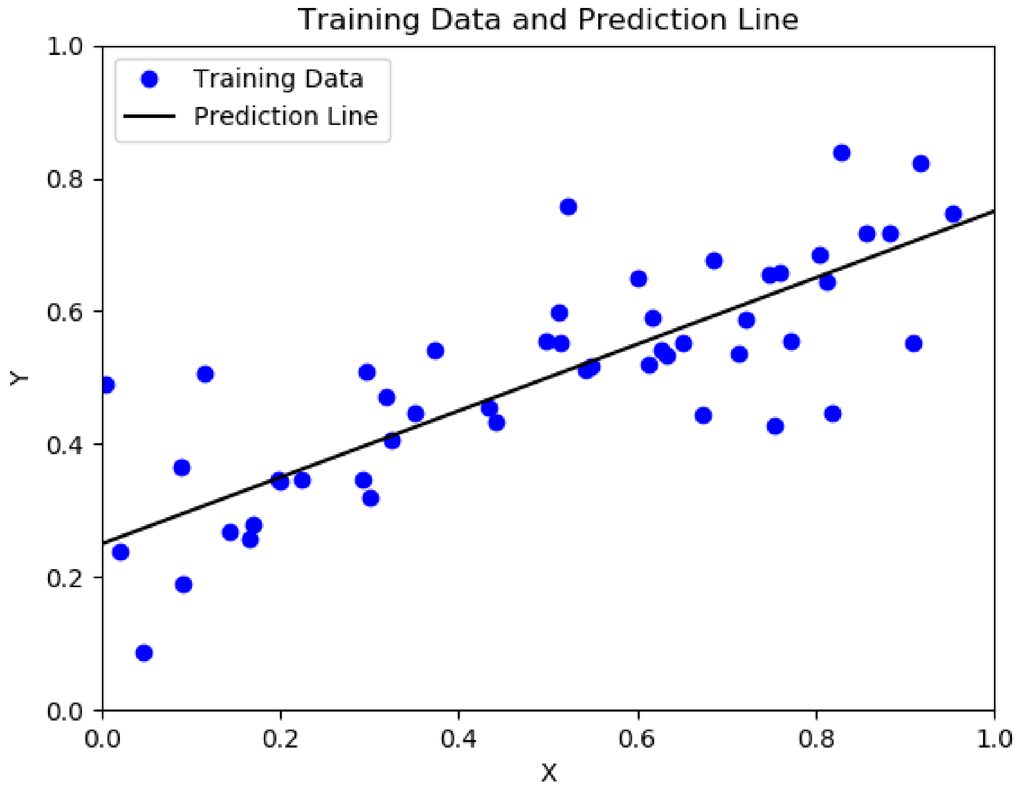
Linear Regression – Line Fitting

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



Copyright Satish Reddy 2020

Least Squares Approach

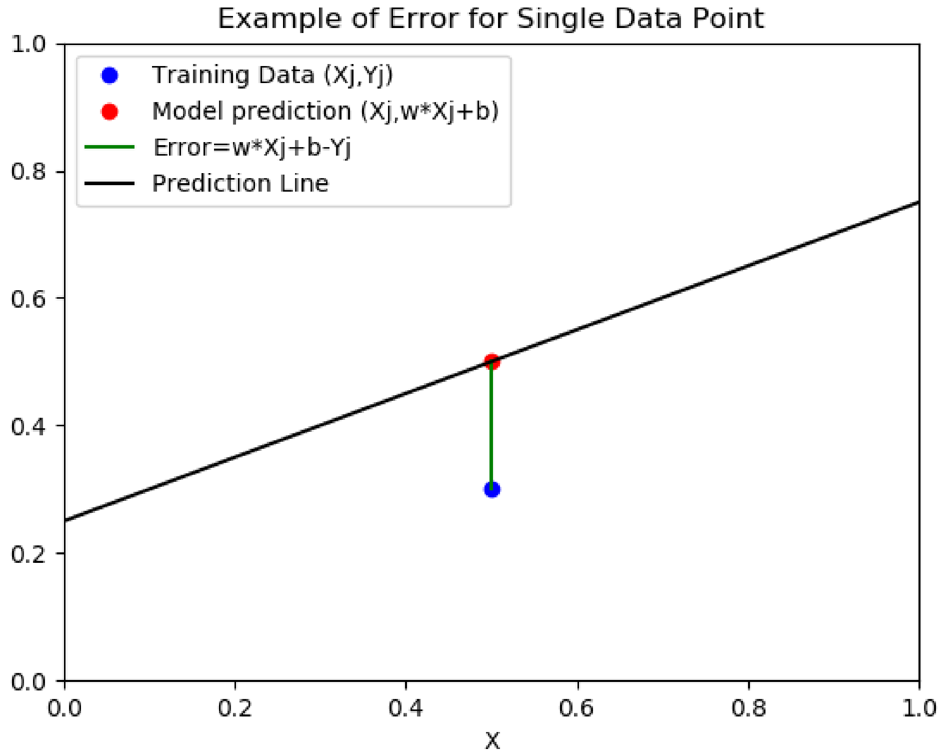
- Assume Y=WX+b

- 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



Copyright Satish Reddy 2020

Summary of Least Squares Approach

TRAINING DATA:

- Input/Output information pairs: (X_o,Y_o), (X_1,Y_1), …, (X_{m-1},Y_{m-1})

FUNCTION STRUCTURE

- Assume line Y = W*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*X + b

Copyright Satish Reddy 2020

Normal Equations

- For the 1-dimensional problem input/output info: (X_o, Y_o), (X_1, Y_1), …, (X_{m-1}, Y_{m-1})
- Let us define:

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

- Define the parameter vector as:

$$\widehat{W} = \begin{bmatrix} W & b \end{bmatrix}$$

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

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

(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

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

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: $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \ldots \\ X_{d-1,j} \end{bmatrix}$ and output: Y_j

- Define the feature matrix (dxm) and output vector (1xm):

$$X = \begin{bmatrix} X_{00} & \ldots & X_{0,m-1} \\ \ldots & \ldots & \ldots \\ X_{d-1,0} & \ldots & X_{d-1,m-1} \end{bmatrix} \qquad Y = \begin{bmatrix} Y_0 & \ldots & Y_{m-1} \end{bmatrix}$$

Copyright Satish Reddy 2020

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_o,Y_o), (X_1,Y_1), …, (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:

$$X = \begin{bmatrix} 1 & 0.5 & 2 & 3 \end{bmatrix} \qquad Y = \begin{bmatrix} 1 & 2 & 3 & 2 \end{bmatrix}$$

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

Function Structure - Forward Propagation

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

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

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

1. Linear part: for j=0,…,m-1

$$Z_j = W_o X_{oj} + W_1 X_{1j} + W_2 X_{2j} + \cdots + W_{d-1} X_{d-1j} + b$$

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

Function Structure

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

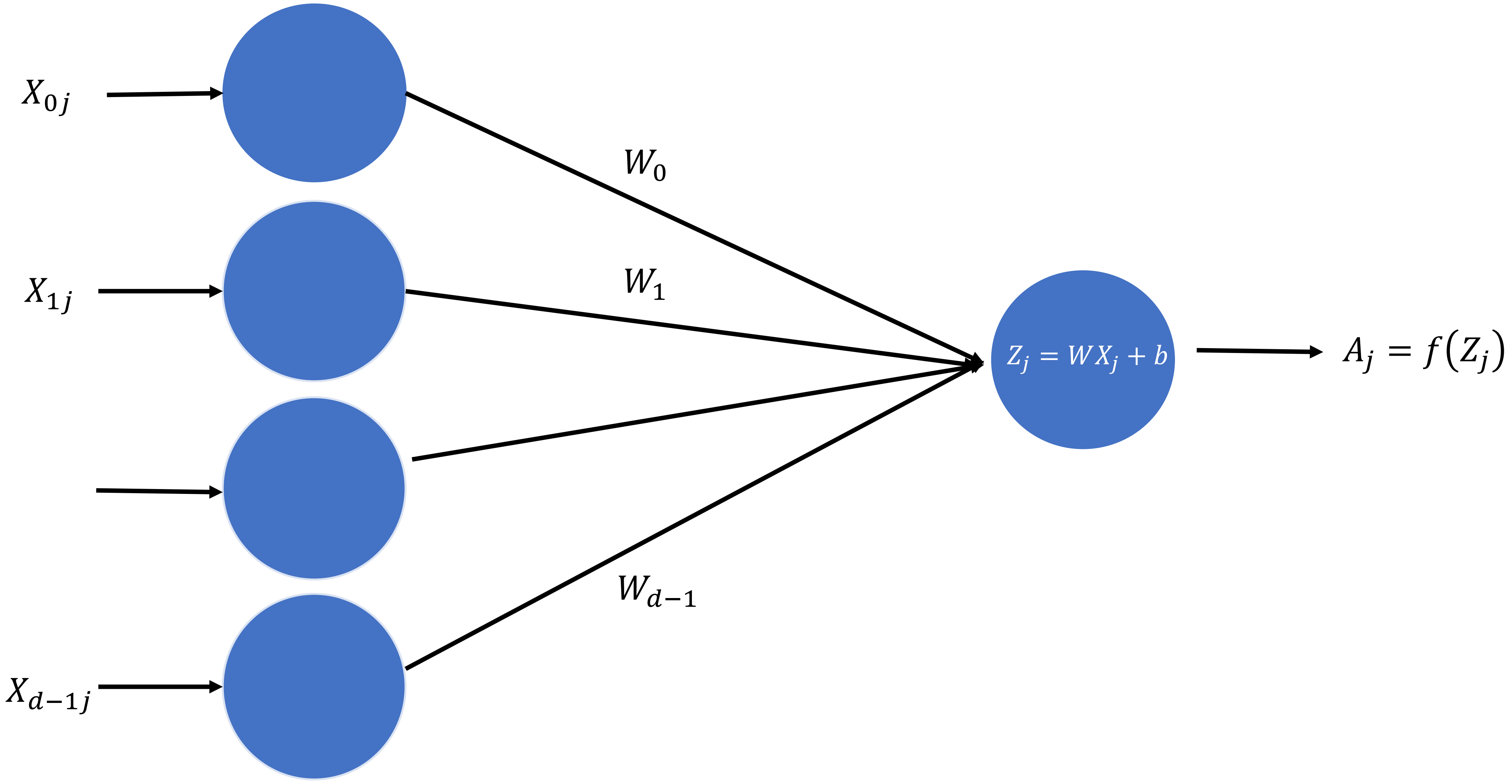
$$A_j = f(Z_j) \qquad j = 0, \ldots, m-1$$

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

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

Copyright Satish Reddy 2020

Forward Propagation Diagram



Copyright Satish Reddy 2020

Forward Propagation - Example

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

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

- Assume that initial parameter values are:

$$W = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathrm{b} = [2]$$

- Forward Propagation:

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

$$A = Z = \begin{bmatrix} 5 & 9 & 13 \end{bmatrix}$$

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

Loss Function - Example

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

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

- From the Forward Propagation Example

$$A = \begin{bmatrix} 5 & 9 & 13 \end{bmatrix}$$

- Loss function defined by

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

Copyright Satish Reddy 2020

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

$\nabla_W L = 0$ and $\nabla_b L = 0$

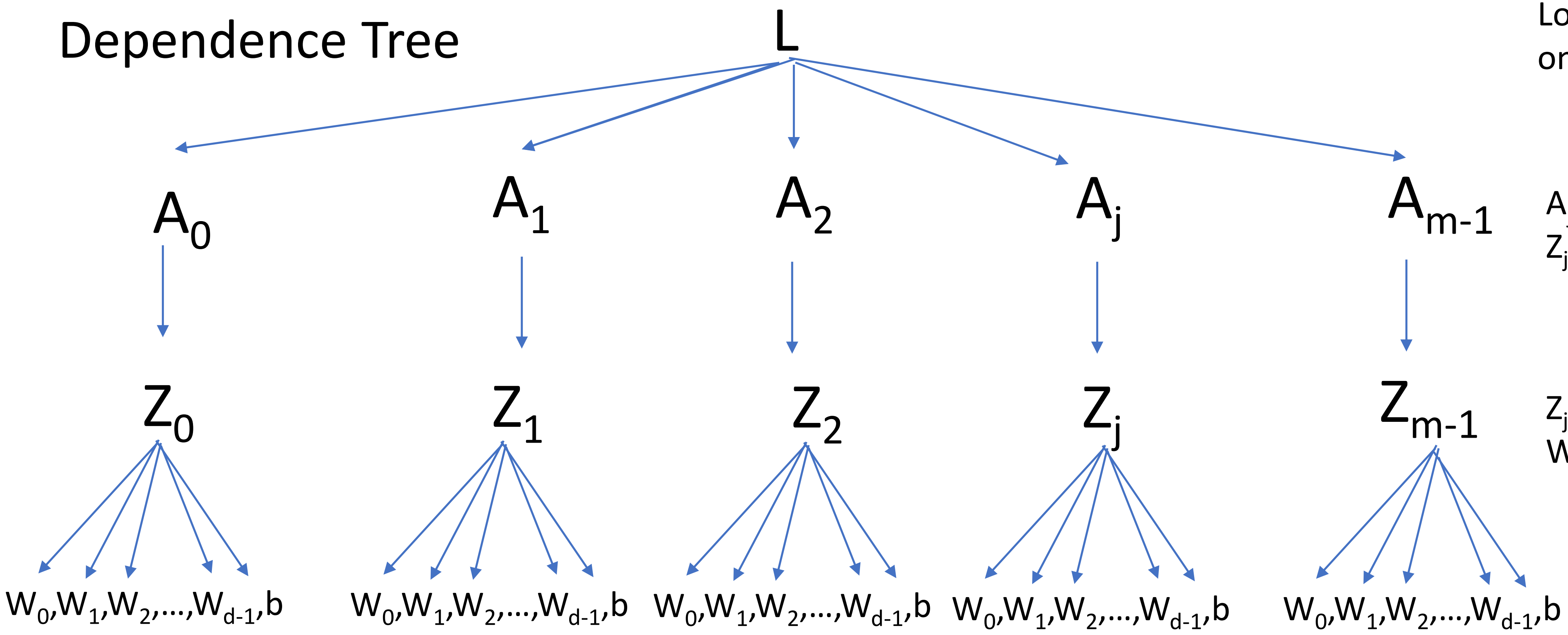
- Can solve these equations analytically for Linear Regression, but not in general case (Logistic Regression and Neural Networks)
- Use optimization algorithm (example: Gradient Descent) to minimize Loss function
    - Need to compute the above gradients

Copyright Satish Reddy 2020

Computing Gradients

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

Dependence Tree



Loss L depends on each A_j

A_j depends on Z_j only

Z_j depends on all W_i and b

Copyright Satish Reddy 2020

Multi-Variable Calculus – Chain Rule

Recall the chain rule lecture (Section 1.6)

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

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

- For linear regression, the loss function L depends on A_0 ,…,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

$$\nabla_Z L = \begin{bmatrix} \frac{\partial L}{\partial Z_0} & \dots \frac{\partial L}{\partial Z_{m-1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial A_0}\frac{\partial A_0}{\partial Z_0} & \frac{\partial L}{\partial A_1}\frac{\partial A_1}{\partial Z_1} & \dots & \frac{\partial L}{\partial A_{m-1}}\frac{\partial A_{m-1}}{\partial Z_{m-1}} \end{bmatrix}$$

$$\nabla_Z L = \begin{bmatrix} \frac{\partial L}{\partial A_0} & \frac{\partial L}{\partial A_1} & \dots \frac{\partial L}{\partial A_{m-1}} \end{bmatrix} * \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \dots & \frac{\partial A_{m-1}}{\partial Z_{m-1}} \end{bmatrix} = \nabla_A L * \begin{bmatrix} 1 & 1 & \dots & 1 \end{bmatrix}$$

Here * means component-wise multiplication

Copyright Satish Reddy 2020

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

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

Copyright Satish Reddy 2020

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 $[A_0 \;\; A_1 \;\; \ldots \;\; A_{m-1}]$ has been computed

Input: feature matrix X and value vector Y

1. Compute gradient of L with respect to A

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

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

3. Compute gradient L with respect to Z:

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

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

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

Copyright Satish Reddy 2020

Back Propagation - Example

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

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

- Assume that initial parameter values are:

$W = \begin{bmatrix} 1 & 1 \end{bmatrix} \qquad \mathrm{b} = [2]$

- From forward propagation example:

$A = Z = \begin{bmatrix} 5 & 9 & 13 \end{bmatrix}$

- Gradient of Loss with respect to A:

$\frac{\partial L}{\partial A_j} = \frac{2}{m}(A_j - Y_j)$ for j=0,…,m-1 $\quad \nabla_A L = \begin{bmatrix} \frac{2}{3}(5-8) & \frac{2}{3}(9-6) & \frac{2}{3}(13-10) \end{bmatrix} = \begin{bmatrix} -2 & 2 & 2 \end{bmatrix}$

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

$\nabla_Z L = \nabla_A L * \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \frac{\partial A_2}{\partial Z_2} \end{bmatrix} = \begin{bmatrix} -2 & 2 & 2 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 2 & 2 \end{bmatrix}$ (component-wise multiplication)

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

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

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

Training Algorithm

Input training data: feature matrix X and values Y

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

Choose learning rate $\alpha > 0$

1. Loop for epoch i = 1, 2, …
   - Forward Propagate using X to compute $A_{epoch=i-1}$
   - Back Propagate using X, Y, and $A_{epoch=i-1}$ to compute $\nabla_W L_{epoch=i-1}, \nabla_b L_{epoch=i-1}$
   - Update parameters: (Gradient Descent)

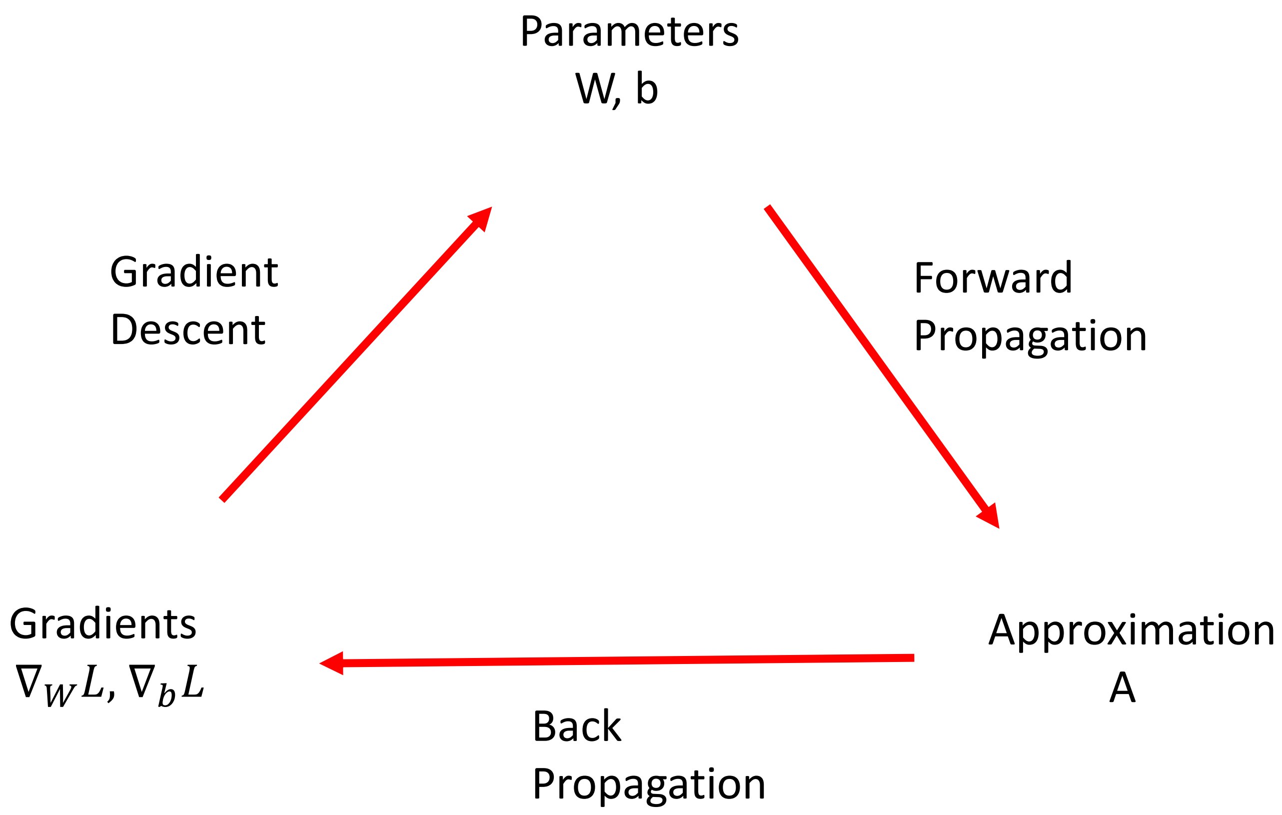
     $W_{epoch=i} = W_{epoch=i-1} - \alpha \nabla_W L_{epoch=i-1}$

     $b_{epoch=i} = b_{epoch=i-1} - \alpha \nabla_b L_{epoch=i-1}$
   - Forward Propagate to compute $A_{epoch=i}$
   - Compute Loss at $A_{epoch=i}$

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

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

Training Algorithm - Example

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

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

- Assume that initial parameter values are:

$$W_{epoch=0} = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad b_{epoch=0} = [2]$$

- Choose learning rate α=0.01

EPOCH 1

- From Forward Propagation Example:

$$A_{epoch=0} = \begin{bmatrix} 5 & 9 & 13 \end{bmatrix}$$

- From Back Propagation Example:

$$\nabla_W L_{epoch=0} = \begin{bmatrix} 10 & 20 \end{bmatrix}, \; \nabla_b L_{epoch=0} = [2]$$

- Update:

$$\mathrm{W}_{epoch=1} = \mathrm{W}_{epoch=0} - \alpha \nabla_W L_{epoch=0} = \begin{bmatrix} 1 & 1 \end{bmatrix} - 0.01 * \begin{bmatrix} 10 & 20 \end{bmatrix} = \begin{bmatrix} 0.9 & 0.8 \end{bmatrix}$$

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

Copyright Satish Reddy 2020

Training Algorithm - Example

- Apply Forward Propagation with $\mathrm{W}_{epoch=1}$ and $\mathrm{b}_{epoch=1}$

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

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

EPOCH 2

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

$\frac{\partial L}{\partial A_j} = \frac{2}{m}(A_j - Y_j)$ for j=0,…,m-1 $\quad \nabla_A L = \left[\frac{2}{3}(4.48 - 8) \quad \frac{2}{3}(7.78 - 6) \quad \frac{2}{3}(11.18 - 10)\right] =$
$[-2.3467 \quad 1.1867 \quad 0.7867]$

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

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

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

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

Copyright Satish Reddy 2020

Training Algorithm - Example

- Update:

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

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

- Apply Forward Propagation with $\mathrm{W}_{epoch=2}$ and $\mathrm{b}_{epoch=2}$

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

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

Copyright Satish Reddy 2020

Prediction Algorithm

Prediction algorithm makes use parameters computed in Training Algorithm

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

Use W and b computed by Training Algorithm

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

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

Copyright Satish Reddy 2020

Prediction Algorithm - Example

- From Training Algorithm Example:

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

- After 2 EPOCHs of Training Algorithm:

$$W_{epoch=2} = \begin{bmatrix} 0.8683 & 0.7325 \end{bmatrix} \quad b_{epoch=2} = \begin{bmatrix} 1.9837 \end{bmatrix}$$

- Prediction: Apply Forward Propagation with $\mathrm{W}_{epoch=2}$ and $\mathrm{b}_{epoch=2}$

$$A_{epoch=2} = Z_{epoch=2} = \mathrm{W}_{epoch=2}X + b_{epoch=2} = \begin{bmatrix} 0.8683 & 0.7325 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 7 \end{bmatrix} + \begin{bmatrix} 1.9837 \end{bmatrix}$$
$$= \begin{bmatrix} 4.3170 & 7.3828 & 10.5844 \end{bmatrix}$$

Copyright Satish Reddy 2020

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

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

Copyright Satish Reddy 2020

Accuracy Calculation - Example

- From example prediction example:

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

- Prediction:

$$A_{epoch=2} = \begin{bmatrix} 4.3170 & 7.3828 & 10.5844 \end{bmatrix}$$

- Accuracy:

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

Copyright Satish Reddy 2020

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: $f(z) = z$ $\mathrm{A} = f(Z)$ (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: α>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

Copyright Satish Reddy 2020

Linear Regression – Jupyter Notebook Demo

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

Copyright Satish Reddy 2020

2.2 Derivative Testing

Copyright Satish Reddy 2020

Derivative Testing

Goal of this Section:

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

Copyright Satish Reddy 2020

Motivation for Derivative Testing

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

Copyright Satish Reddy 2020

Difference Formula for Derivatives

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

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

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

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

- Centered differences formula: (error is proportional to ε^2 or better as $\varepsilon \to 0$ – 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)

Copyright Satish Reddy 2020

Derivative Testing - Example

- Consider Backpropagation Example in Section 2.1

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

- 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 +ε and –ε (choose ε=0.1)
- ε=0.1 case:

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

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

- ε=–0.1 case:

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

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

Hence:

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

Copyright Satish Reddy 2020

Derivative Testing - Example

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

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

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

- ε=–0.1 case:

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

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

Hence:

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

Copyright Satish Reddy 2020

Derivative Testing - Example

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

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

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

- ε=–0.1 case:

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

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

Hence:

$$\frac{\partial L}{\partial b} \approx \frac{L_{+} - L_{-}}{2\varepsilon} = \frac{9.21 - 8.81}{2(0.1)} = 2$$ (this matches back propagation derivative exactly)

- 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

Copyright Satish Reddy 2020

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: $W = [W_0 \;\; W_1 \;\; \ldots \;\;\; W_{d-1}]$ and scalar b

Concatenated format: $[W_0 \;\; W_1 \;\; \ldots \;\;\; W_{d-1} \;\; b]$ (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

Copyright Satish Reddy 2020

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, b and gradient vectors $\nabla_W L, \nabla_b L$
3. Loop over i = 0,1, .. d (d+1 parameters in W and b)

- Add ε 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 ε 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

Copyright Satish Reddy 2020

Derivative Testing – Jupyter Notebook Demo

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

Copyright Satish Reddy 2020

2.3 Code Design Review

Copyright Satish Reddy 2020

Coding Design Review

Goal of this Section:

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

Copyright Satish Reddy 2020

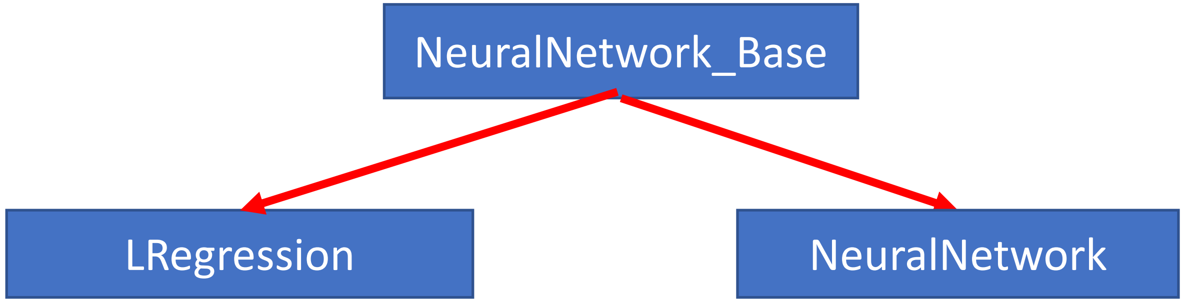
Coding Overview

- Code employs object-oriented approach
  - Two principal classes
    - NeuralNetwork_Base Class
    - Optimizer_Base Class (used to compute update in training algorithm)
- Additional codes
  - Activation functions
  - Loss functions
  - Plotting functions
  - Unit test
  - Drivers
  - Load Data functions
- Design considerations:
  - Numpy array is key building block
  - Use interfaces for framework Tensorflow as a rough guide

Copyright Satish Reddy 2020

NeuralNetwork_Base Class

Linear/Logistic Regression



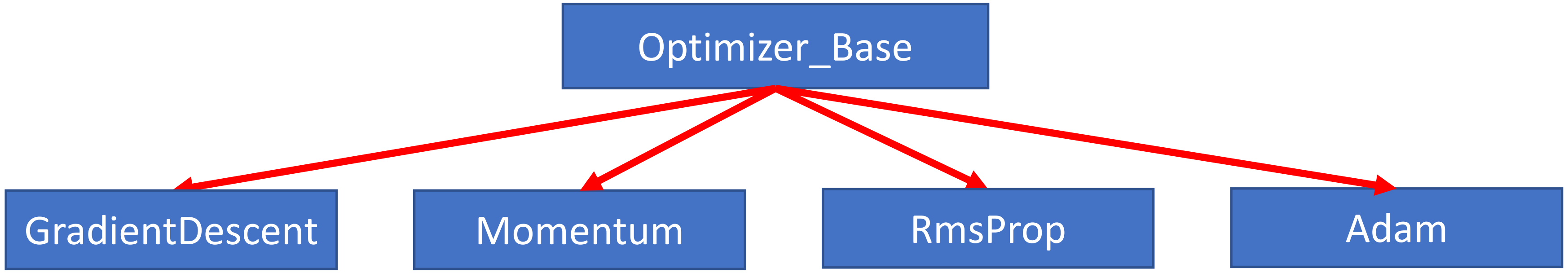
General Neural Networks (multiple layers)

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

Will attempt to put as many methods as possible in NeuralNetwork_Base to handle both Linear/Logistic Regression and Neural Networks

Copyright Satish Reddy 2020

Optimizer_Base Class



Key Methods

- Update

Copyright Satish Reddy 2020

Code Design - Arrays

All relevant arrays will be defined explicitly as 2d numpy arrays

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

Copyright Satish Reddy 2020

2.4 Code Walkthrough
Version 1.1

Copyright Satish Reddy 2020

Coding Walkthrough: Version 1.1

Goal of this Section:

- Walkthrough of code necessary to perform unit test of forward/back propagation for Linear Regression

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

NeuralNetwork_Base Class – Attributes

<table>
<tr><th>Variable</th><th>Type</th><th>Description</th></tr>
<tr><td>nlayer</td><td>integer</td><td>Number of layers
• Equals 1 for Linear and Logistic Regression
• In general greater than 1 for Neural Networks</td></tr>
<tr><td>info</td><td>list
indices:
0,1,…,nlayer-1</td><td>info[k] is a dictionary containing information for layer = k
Keys:
• nIn: (integer) number of unit in previous layer (number of features for layer 0)
• nOut: (integer) number of units in current layer
• activation: (string) activation function type
• A: (numpy array) result after activation for current layer
• param: (dictionary) parameter matrices (keys W, b)
• param_der: (dictionary) derivatives of parameter matrices (keys W, b)
• optimizer: (dictionary) optimizer class objects (keys W,b)</td></tr>
<tr><td>loss_fun</td><td>string</td><td>Name of loss function</td></tr>
</table>

Copyright Satish Reddy 2020

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 $\nabla_W L$, $\nabla_b L$

Copyright Satish Reddy 2020

LRegression – Methods

Method	Input	Description
__init__	nfeature (integer) activation (sting)	Initialization routine that takes in the number of features and the activation function ("linear" for regression) Return: nothing
forward_propagation	X (numpy array)	Performs forward propagation using feature matrix X to compute approximation A and updates info variable for key "A" Returns: nothing
back_propagation	X (numpy array) Y (numpy array)	Performs back propagation using feature matrix X and label vector Y Returns: nothing
concatenate_param	order (string): "param" or "param_der"	Concatenates all entries in W and b or in the their gradient into a single row vector
load_param	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

Copyright Satish Reddy 2020

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 activation 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 activation A, label vector Y, and specified function Return: $\nabla_A L$

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

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})

Copyright Satish Reddy 2020

Unit Test – Jupyter Notebook Demo

- Open file IntroML/Examples/Chapter2/Chapter2.4_UnittestExample.ipynb
- Has example of
  - Unit test

Copyright Satish Reddy 2020

Code Version 1.1 Walkthrough

- Code for walkthrough located at:

IntroML/Code/Version1.1

Copyright Satish Reddy 2020

2.5 Code Walkthrough
Version 1.2

Copyright Satish Reddy 2020

Coding Walkthrough: Version 1.2

Goal of this Section:

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

Copyright Satish Reddy 2020

Coding Walkthrough: Version 1.2 To Do

File/Component	To Duo
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

Copyright Satish Reddy 2020

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)

Copyright Satish Reddy 2020

GradientDescent – Methods

Method	Input	Description
__init__	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

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

Code Version 1.2 Walkthrough

- Code for this walkthrough located at:

IntroML/Code/Version1.2

Copyright Satish Reddy 2020

2.6 Logistic Regression: Mathematical Foundations

Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

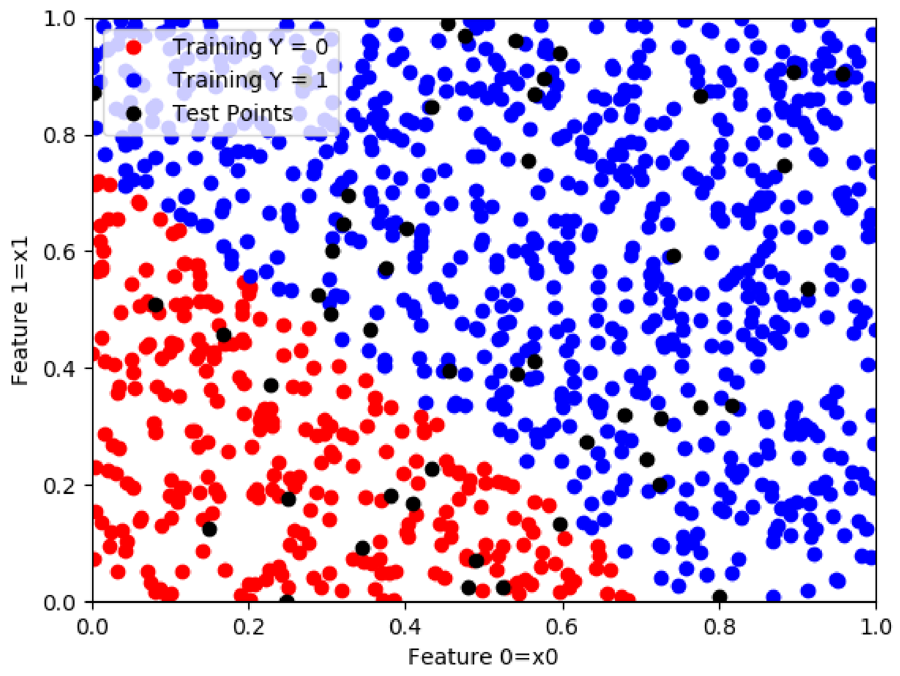
Logistic Regression and Linear Regression

- 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 extended to Logistic Regression
- Principal Differences between Linear and Logistic Regression:
  - Activation Function:
    - Need suitable activation function to produce 0 or 1 output
    - Linear activation function (can take on values from –infinity to infinity) is not suitable
  - Loss Function
    - Need suitable loss function
    - Mean Squared Error loss function not suitable for Logistic Regression

Copyright Satish Reddy 2020

Motivating Example: Binary Classification

Training Data:
- Input Information: points in (x0,x1) 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)

Copyright Satish Reddy 2020

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.

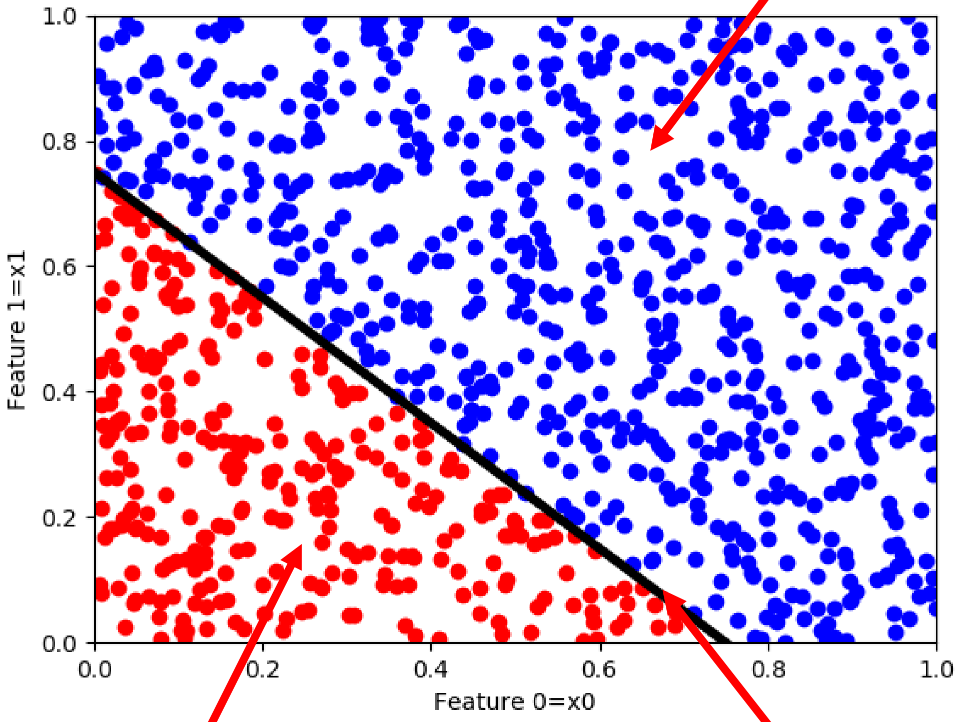
- Define parameters W = [W_0 W_1] and b

$$Z_j = WX_j + b = W_0X_{0j} + W_1X_{1j} + b$$

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

$$f(Z_j) = \begin{Bmatrix} 1 \;\; if Z_j \geq 0 \\ 0 \;\; if Z_j < 0 \end{Bmatrix}$$

Region:
x0 + x1 – 0.75 > 0



Region:
x0 + x1 – 0.75 < 0

Line:
x0 + x1 – 0.75 = 0

Copyright Satish Reddy 2020

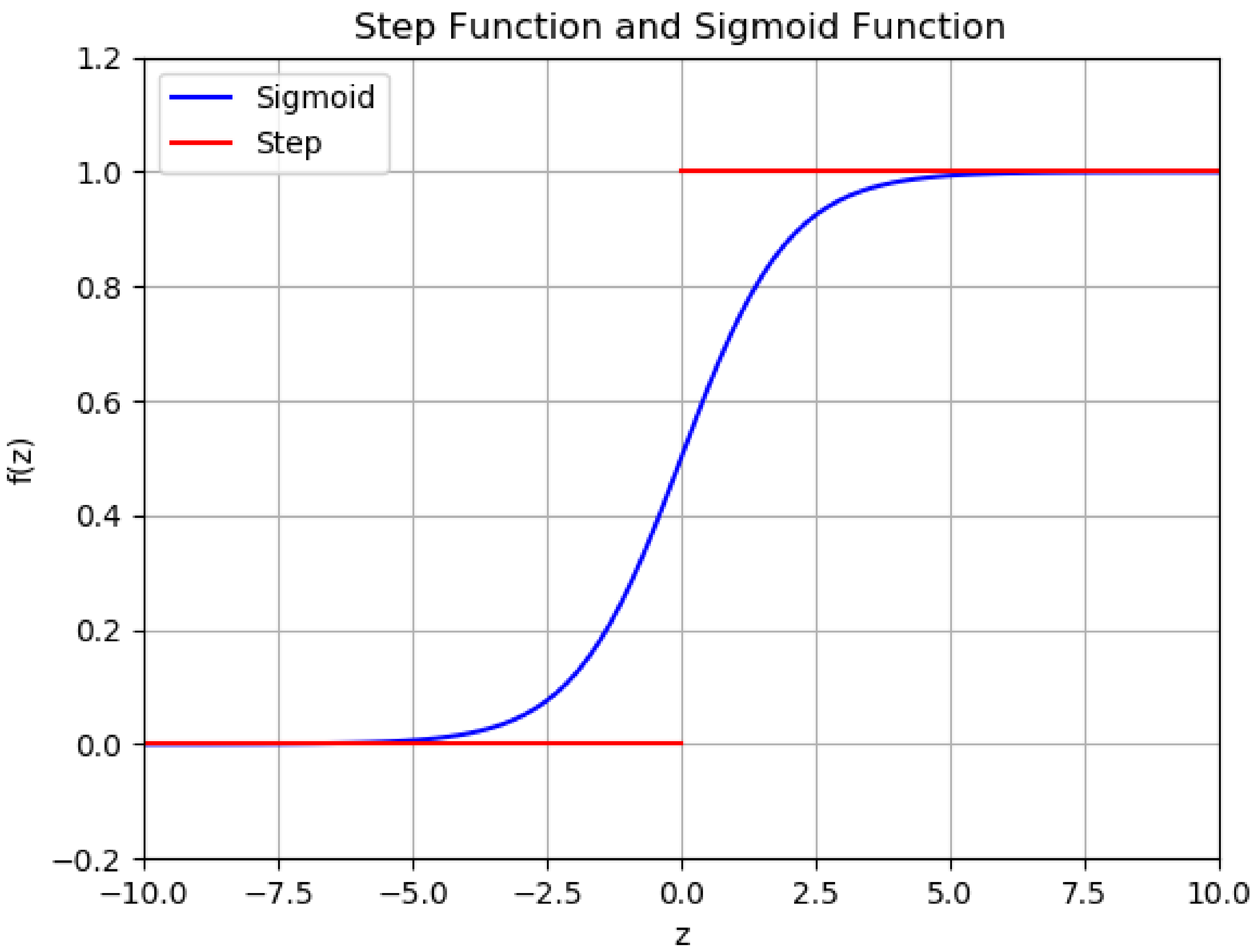
Sigmoid Activation Function

- 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) -> 1 as z->infinity
  f(z) -> 0 as z->-infinity



Copyright Satish Reddy 2020

Binary Cross Entropy Loss Function

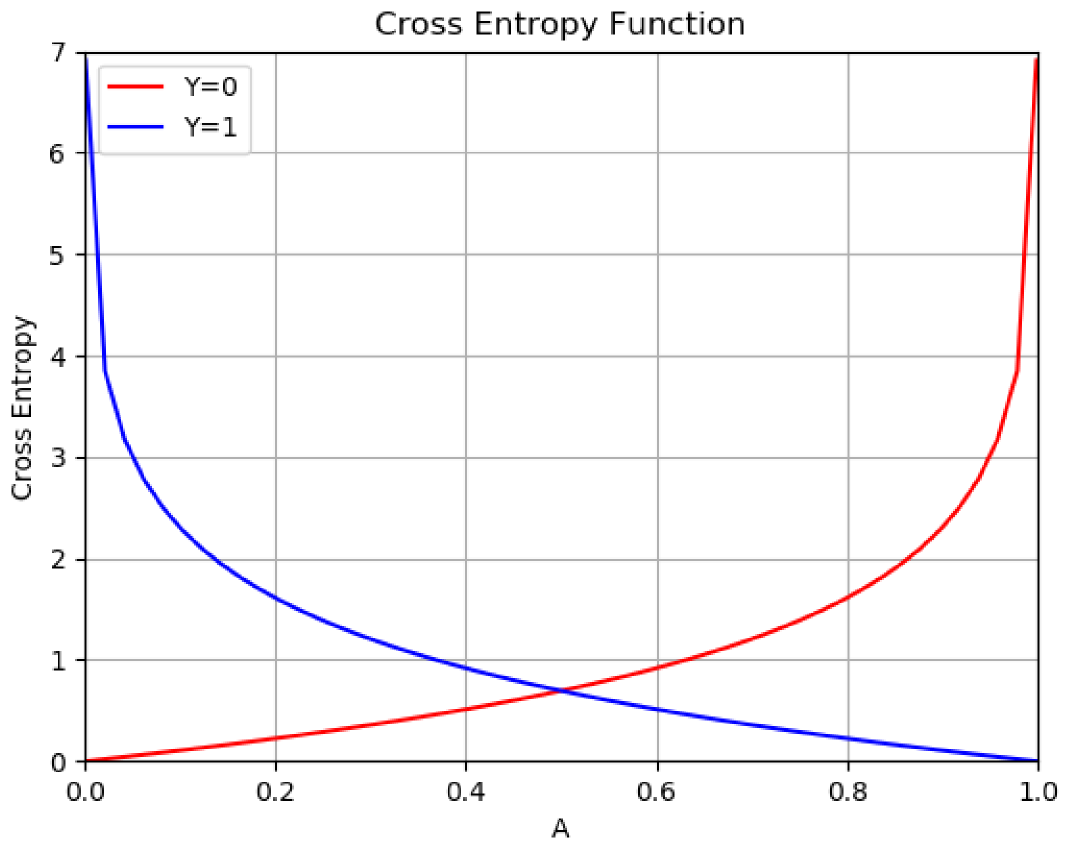
- Mean Squared Error loss function is not appropriate for logistic regression

- Use Binary Cross Entropy loss function instead:

$$Loss = -[Y ln(A) + (1 - Y)\ln(1 - A)]$$

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

- When Y = 1 Loss decreases as A goes to 1



Copyright Satish Reddy 2020

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

Copyright Satish Reddy 2020

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: $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \dots \\ X_{d-1,j} \end{bmatrix}$ and output: Y_j

- Define the feature matrix (dxm) and output vector (1xm):

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

Copyright Satish Reddy 2020

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:

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

Copyright Satish Reddy 2020

Function Structure – Forward Propagation

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

1. Input: feature matrix X (dxm d features and m sample points), parameter vector $W = [W_0 \quad \dots \quad W_{d-1}]$and bias b
2. Linear part: for j=0,…,m-1

$$Z_j = W_o X_{oj} + W_1 X_{1j} + W_2 X_{2j} + \cdots + W_{d-1} X_{d-1j} + b$$

In vector form

$$Z = WX + b$$

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

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

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

Copyright Satish Reddy 2020

Logistic Regression Forward Propagation - Example

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

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

- Assume that initial parameter values are:

$$W = \begin{bmatrix} 0.1 & 0.1 \end{bmatrix} \qquad \mathrm{b} = [0.2]$$

- Forward Propagation:

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

$$A = f(Z) = \begin{bmatrix} \frac{1}{1+e^{-0.1}} & \frac{1}{1+e^{0.1}} & \frac{1}{1+e^{0.2}} \end{bmatrix} = \begin{bmatrix} 0.5250 & 0.4750 & 0.4502 \end{bmatrix}$$

Copyright Satish Reddy 2020

Logistic Regression: Loss Function

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

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

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

Copyright Satish Reddy 2020

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

$\nabla_W L = 0$ 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

Copyright Satish Reddy 2020

Derivative of Loss

- Loss function given by:

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

- Differentiating the log terms, we get:

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

Copyright Satish Reddy 2020

Derivative of Activation

A_j related to Z_j by:

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

Derivatives given by

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

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

This can be simplified to:

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

Copyright Satish Reddy 2020

Logistic Regression Back Propagation Algorithm

Input: feature matrix X and value vector Y

1. Compute:

$$\nabla_A L = \begin{bmatrix} \frac{\partial L}{\partial A_0} & \frac{\partial L}{\partial A_1} & \dots & \frac{\partial L}{\partial A_{m-1}} \end{bmatrix}, \frac{\partial L}{\partial A_j} = -\frac{1}{m}[\frac{Y_j}{A_j} - \frac{1-Y_j}{1-A_j}], j = 0, \dots, m-1$$

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:

$\nabla_Z L = \nabla_A L * \begin{bmatrix} \frac{\partial A_0}{\partial Z_0} & \frac{\partial A_1}{\partial Z_1} & \dots & \frac{\partial A_{m-1}}{\partial Z_{m-1}} \end{bmatrix}$ (component-wise multiplication)

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

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

Copyright Satish Reddy 2020

Logistic Regression Back Propagation - Example

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

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

- Assume that initial parameter values are:

$$W = \begin{bmatrix} 0.1 & 0.1 \end{bmatrix} \qquad \mathrm{b} = \begin{bmatrix} 0.2 \end{bmatrix}$$

- From Forward Propagation Example

$$A = \begin{bmatrix} 0.5250 & 0.4750 & 0.4502 \end{bmatrix}$$

- Gradient of Loss with respect to A:

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

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

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

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

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

Copyright Satish Reddy 2020

Training Algorithm

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

Copyright Satish Reddy 2020

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.)

Copyright Satish Reddy 2020

Logistic Regression Prediction - Example

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

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

- Assume that initial parameter values are:

$$W = \begin{bmatrix} 0.1 & 0.1 \end{bmatrix} \qquad \mathrm{b} = [0.2]$$

- From the Forward Propagation Example slide

$$A = f(Z) = \begin{bmatrix} \frac{1}{1+e^{-0.1}} & \frac{1}{1+e^{0.1}} & \frac{1}{1+e^{0.2}} \end{bmatrix} = \begin{bmatrix} 0.5250 & 0.4750 & 0.4502 \end{bmatrix}$$

- Round to 0 or 1 (round 0.5 up to 1) – predicted labels: $\begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$

Copyright Satish Reddy 2020

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:

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

- 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.

Copyright Satish Reddy 2020

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

$Accuracy = 0.6$

Copyright Satish Reddy 2020

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: $f(Z) = \frac{1}{1+e^{-Z}}$ $\mathrm{A} = f(Z)$ (1xm vector)
Loss Function		Binary Cross Entropy: $L = -\frac{1}{m}\sum_{j=0}^{m-1} Y_j log\, A_j + (1 - Y_j)\log(1\text{-}A_j)$
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: α>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 or 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

Copyright Satish Reddy 2020

Logistic Regression – Jupyter Notebook Demo

- Open file IntroML/Examples/Chapter2/Chapter2.6_LogisticRegression.ipynb
- Has example of
  - Forward Propagation
  - Back Propagation
  - Prediction
  - Accuracy calculation

Copyright Satish Reddy 2020

2.7 Code Walkthrough: Version 1.3

Copyright Satish Reddy 2020

Code Walkthrough Version 1.3

Goal of this Section:

- Walkthrough extension of linear regression codes to handle logistic regression

Copyright Satish Reddy 2020

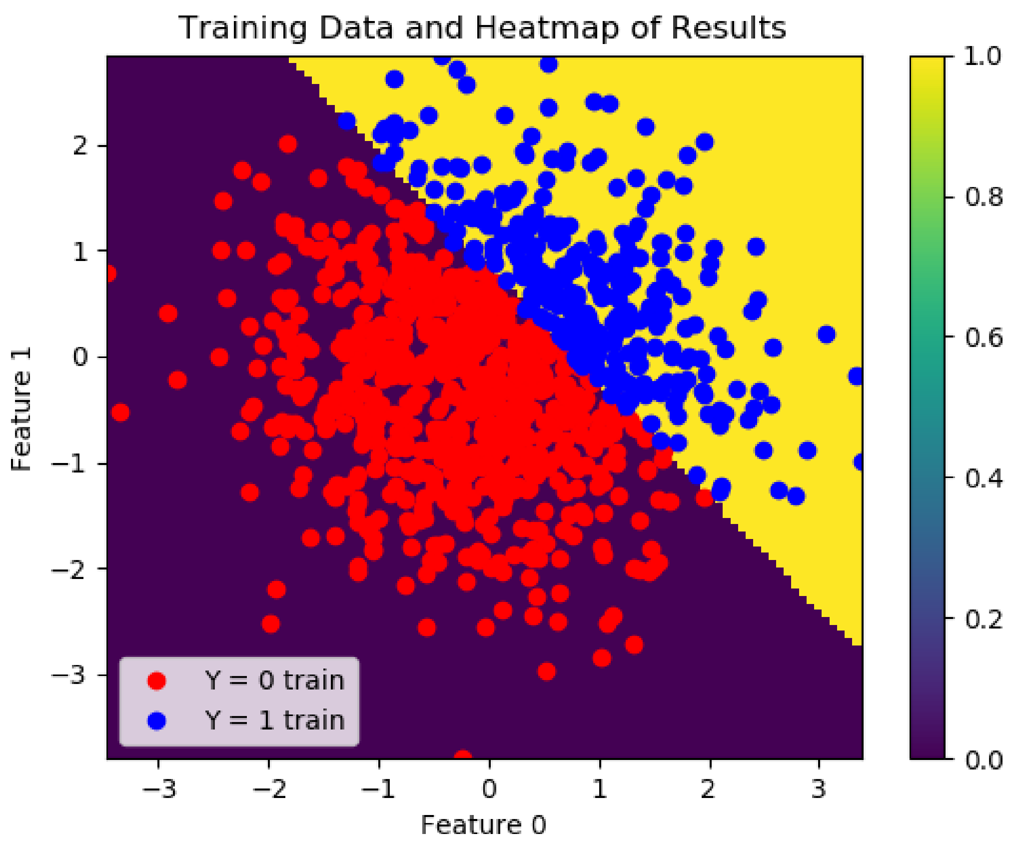
Coding Walkthrough: Version 1.3 To Do

File/Component	To Duo
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

Copyright Satish Reddy 2020

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)

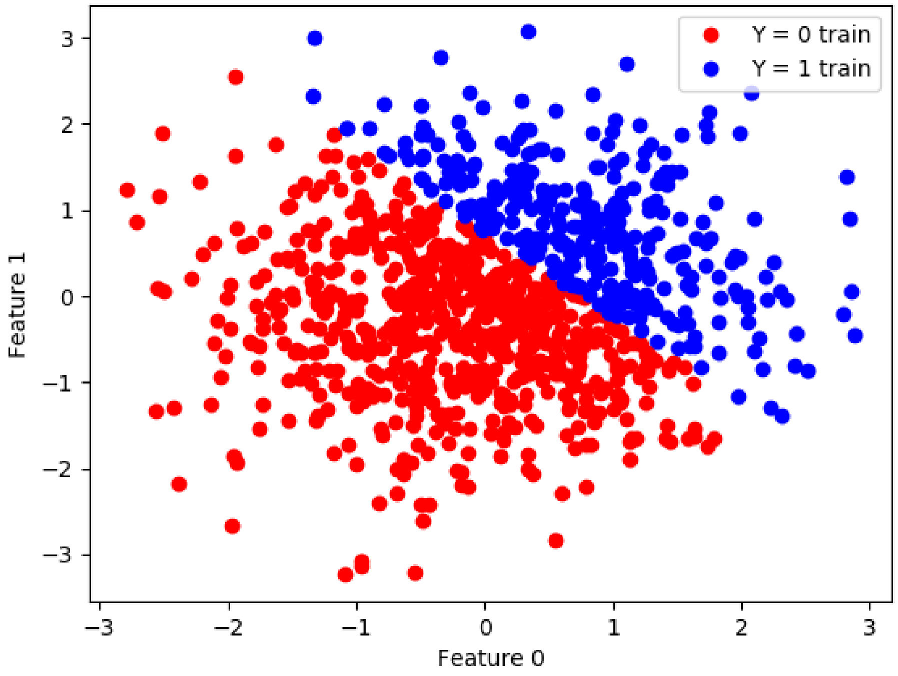


Copyright Satish Reddy 2020

Plotting Logistic Regression Training Data

Start with feature matrix X (2 features x 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



Copyright Satish Reddy 2020

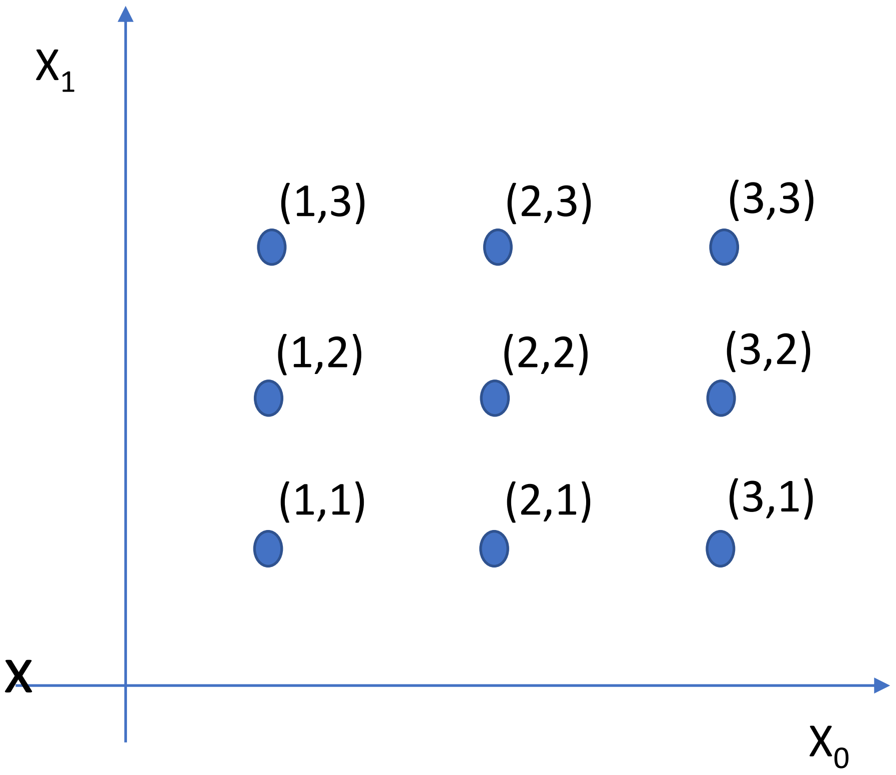
Plotting Heatmap of Results

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



Copyright Satish Reddy 2020

Code Version 1.3 Walkthrough

Copyright Satish Reddy 2020