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

3.1 Neural Networks: Mathematical Foundations

Neural Networks: Mathematical Foundations

Goal of this Section:

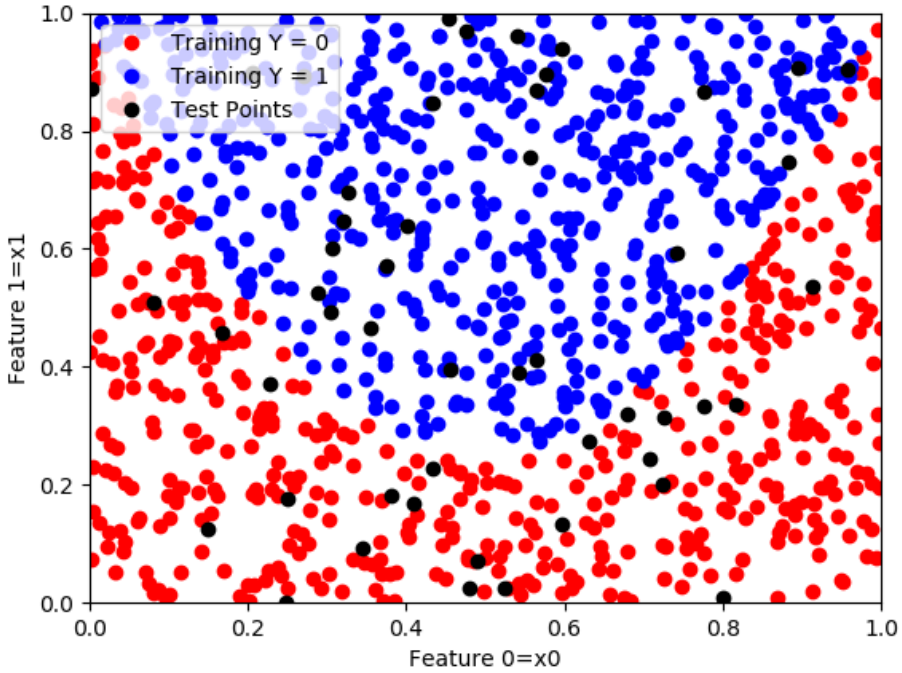
- Present the mathematical foundations for Neural Networks:
  - Format of training data
  - Function structure and parameters
  - Training algorithm
  - Prediction algorithm

Limitations of Linear/Logistic Regression

- Recall definition of Supervised Learning:
  - Process of learning a function that maps input information to labelled output information. The labelled input/output information is called the training data. The learned function is then used to predict outputs when new input information is provided.
- Linear and Logistic Regression function structures are inherently linear – limits ability to map input information to output information in most situations
- Need a more general function structure that can fit a larger range of training data sets

Motivating Example - Binary Classification

Training Data:
- Input Information: points in (x0,x1) plane
- Output Information: label 0 (red) or 1 (blue)



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)

Neural Network
- More suitable than Logistic Regression more sophisticated classification problems
- Can be used for classification with more than 2 classes

Neural Network: 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 general regression problem where there are m data points, each consisting of a input information vector of length d and value Y (0 or 1)

- Data point j: input information (feature) vector: $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \cdots \\ X_{d-1,j} \end{bmatrix}$ and output: Y_j

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

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

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

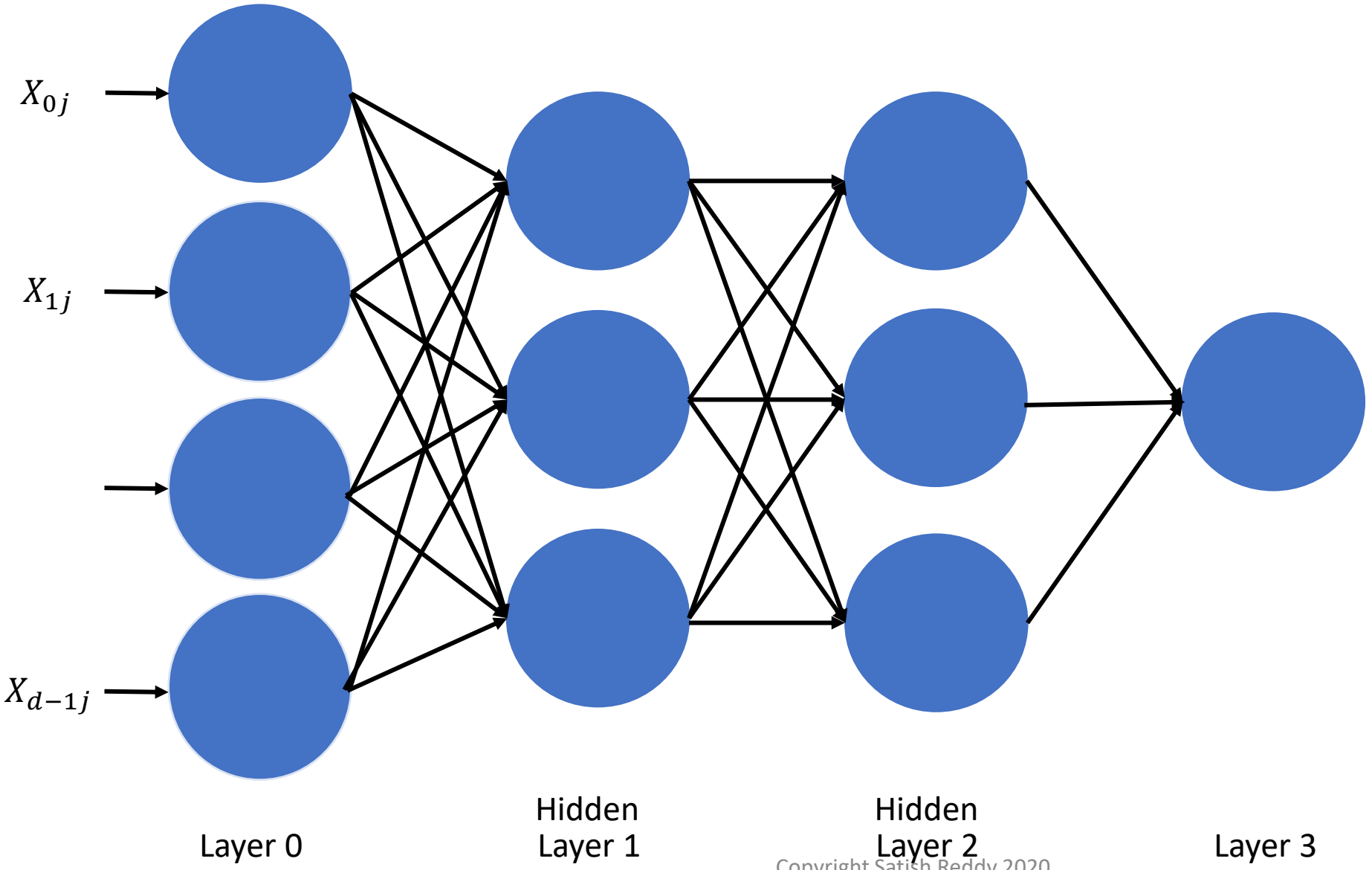
- In this case each sample has 2 features. Feature matrix and value vector are:

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

Neural Network Function Structure

- Function structure for entire Neural Network involves sequence of stages similar to the single stage for Linear/Logistic Regression
- Inputs:
  - Input feature matrix X (d x m) - X_{ij} = feature i for data point j (i=0,…,d-1, j=0,…,m-1)
  - Label vector Y (1xm) - Y_j label for data point j
- Number of layers
  - Assume N layers
- Number of units
  - Layer k=1,…,N has $n^{[k]}$ units - note that $n^{[0]} = d$ (number of features)
  - Final layer N has 1 unit
- Parameters:
  - $W^{[k]}$ is matrix of dimensions ($n^{[k]}$x $n^{[k-1]}$) for layer k
  - $b^{[k]}$ is vector of dimensions ($n^{[k]}$x 1) for layer k
- Activation functions
  - $f^{[k]}(z)$ is activation function for layer k

Neural Network Node Structure – 3 Layer



- Feature data is input at layer 0
- All nodes at layer k-1 are connected to all nodes at layer k
- Example of Feed Forward Neural Network as information moves in one direction only
- Inner layers 1 and 2 are called hidden
- Final layer has single node for binary classification. Later will consider multiple nodes at final layer for multi-class classification

Function Structure Forward Propagation Algorithm

Assume N layer Neural Network

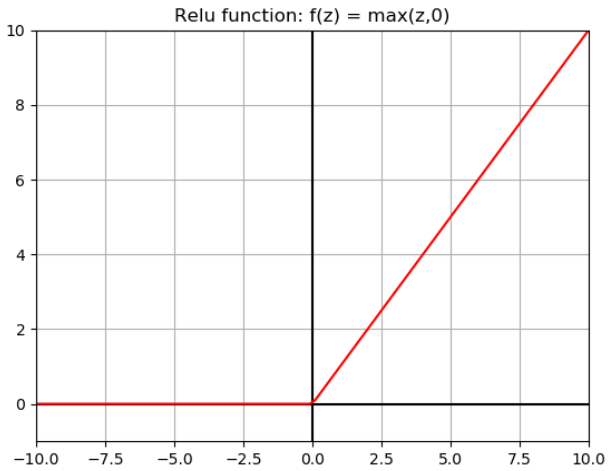
1. Input: feature matrix X (d features x m samples) and parameter matrices $W^{[k]}$, $b^{[k]}$ for k=1,…,N
2. Define: $A^{[0]} = X$
3. Loop for k=1,…,N (number of layers)
   - Linear part: $Z^{[k]} = W^{[k]}A^{[k-1]} + b^{[k]}$ $\quad \#\text{matrix of dimension } (n^{[k]}\text{x } m)$
   - Activation: $A^{[k]} = f^{[k]}(Z^{[k]})$ $\quad \#\text{matrix of dimension } (n^{[k]}\text{x } m)$

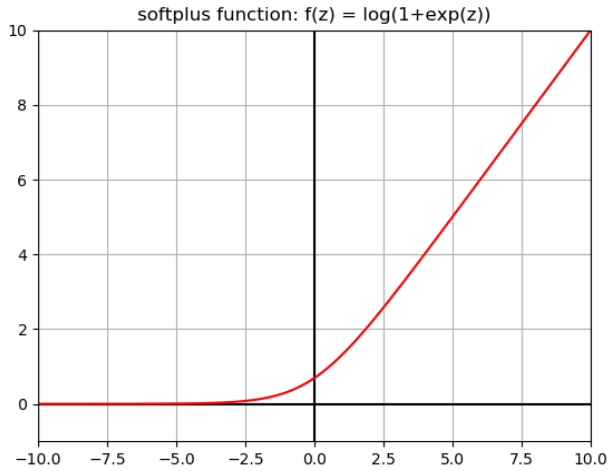
Notes:

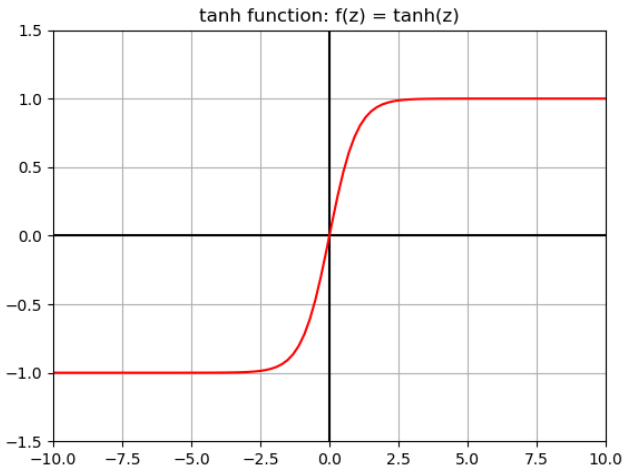
- Each layer k will have its own activation function $f^{[k]}(z)$
- For binary classification, final layer has 1 unit and uses sigmoid activation function

Activation Functions

- Can use different activation functions for each layer
- Examples of activation functions
  - $Relu(z) = \max(z, 0)$
  - $\text{softplus}(z) = \ln(1 + e^z)$
  - $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
  - See https://en.wikipedia.org/wiki/Activation_function for more examples







Neural Network 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 layer 1 has 2 units and that layer 2 has 1 unit
- Assume parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad W^{[2]} = \begin{bmatrix} -1 & 1 \end{bmatrix} \qquad b^{[2]} = \begin{bmatrix} -0.1 \end{bmatrix}$$

- Assume activation functions $f^{[1]}(z) = \tanh(z)$ and $f^{[2]}(z) = \frac{1}{1+e^{-z}}$

Forward Propagation:

- Layer 1:

$$Z^{[1]} = W^{[1]}X + b^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1.5 \\ 2 & 4 & 6.5 \end{bmatrix}$$

$$A^{[1]} = f(Z) = \begin{bmatrix} \tanh(0) & \tanh(-1) & \tanh(-1.5) \\ \tanh(2) & \tanh(4) & \tanh(6.5) \end{bmatrix} = \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix}$$

Neural Network Forward Propagation - Example

- Layer 2:

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} = [-1 \quad 1]\begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix} + [-0.1]$$

$$= [0.8640 \quad 1.6609 \quad 1.8051]$$

$$A^{[2]} = f^{[2]}\left(Z^{[2]}\right) = \left[\frac{1}{1+e^{-0.8640}} \quad \frac{1}{1+e^{-1.6609}} \quad \frac{1}{1+e^{-1.8051}}\right] = [0.7035 \quad 0.8404 \quad 0.8588]$$

Binary Cross Entropy Loss Function

- Loss function is same as for Logistic Regression
- Average of binary cross entropy applied to output of final layer

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

Neural Network Training Phase

- Training phase attempts to find suitable parameter matrices $W^{[k]}$, $b^{[k]}$ for k=1,…,N that minimize the loss function when applied to the training data
- Use optimization algorithm (example: Gradient Descent) to minimize Loss function
- Need to compute derivatives $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$ for k=1,…,N for optimization

Derivatives of Activation Functions

- Compute derivatives of Relu, Softplus, and Tanh
- $A = Relu(z) = \max(z, 0)$

$$\frac{\partial A}{\partial z} = 1 \text{ if A} \geq 0, \qquad 0 \; if \; A < 0$$

- $\text{A} = \text{softplus}(z) = \ln(1 + e^z)$

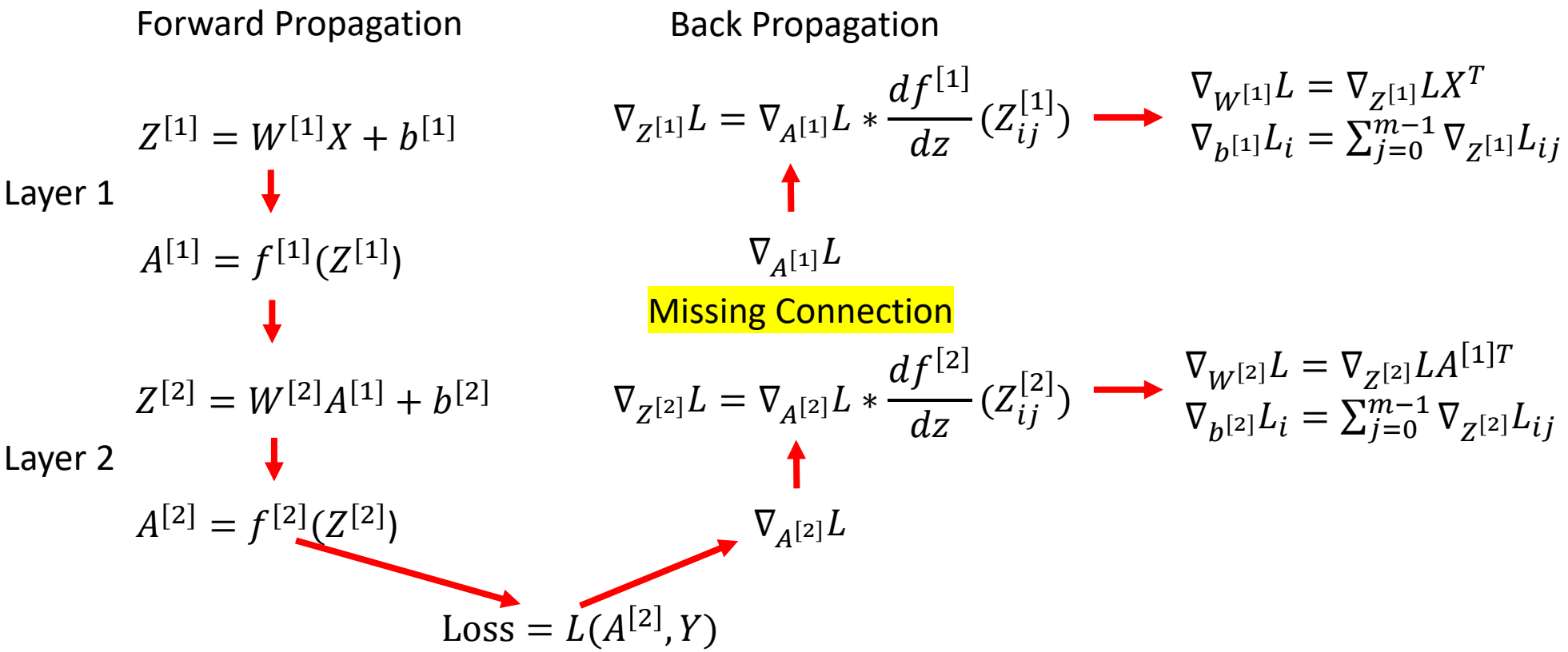
$$\frac{\partial A}{\partial z} = \frac{e^z}{1 + e^z} = \frac{e^A - 1}{e^A} = 1 - e^{-A}$$

- $A = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

$$\frac{\partial A}{\partial z} = \frac{e^z + e^{-z}}{e^z + e^{-z}} - \frac{(e^z - e^{-z})^2}{(e^z + e^{-z})^2} = 1 - A^2$$

Back Propagation – Connecting Layers

- Consider a neural network with 2 layers
- Goal: compute the four gradients $\nabla_{W^{[2]}} L, \nabla_{b^{[2]}} L, \nabla_{W^{[1]}} L, \nabla_{b^{[1]}} L$



Back Propagation – Connecting Layers

- Question from last slide: Given $\nabla_{Z^{[2]}} L$ what is $\nabla_{A^{[1]}} L$?
- We know

$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

- From chain rule analysis in Section 1.6

$$\nabla_{A^{[1]}} L = W^{[2]T} \nabla_{Z^{[2]}} L$$

- In general

$$\nabla_{A^{[k-1]}} L = W^{[k]T} \nabla_{Z^{[k]}} L$$

Back Propagation Algorithm

Assume N layers

Input: feature matrix X and and label vector Y and assume Forward Propagation has been performed

1. Compute $\nabla_{A^{[N]}} L$
2. Loop for k=N,…,1

- Compute $\frac{\partial A_{ij}^{[k]}}{\partial Z_{ij}^{[k]}} = \frac{df^{[k]}}{dz}(Z_{ij}^{[k]})$ for $i = 0, \dots, n^{[k]} - 1, j = 0, \dots, m - 1$
- Compute $\nabla_{Z^{[k]}} L = \nabla_{A^{[k]}} L * \frac{\partial A_{ij}^{[k]}}{\partial Z_{ij}^{[k]}}$ (pointwise-multiplication)
- $\nabla_{W^{[k]}} L = \nabla_{Z^{[k]}} L A^{[k-1]^T}$
- $\nabla_{b^{[k]}} L_i = \sum_{j=0}^{m-1} \nabla_{Z^{[k]}} L_{ij}\,, \quad i = 0, \dots, n^{[k]} - 1$
- If k>1:$\nabla_{A^{[k-1]}} L = W^{[k]^T} \nabla_{Z^{[k]}} L$

Neural Network Back Propagation - Example

See file IntroML/Examples/Chapter3/Chapter3.1_BackPropagation

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

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

- Assume that layer 1 has 2 units and that layer 2 has 1 unit
- Assume parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} -1 & 1 \end{bmatrix} \quad b^{[2]} = [-0.1]$$

- Assume activation functions $f^{[1]}(z) = \tanh(z)$ and $f^{[2]}(z) = \frac{1}{1+e^{-z}}$
- From Forward Propagation example:

$$A^{[1]} = \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix} \quad A^{[2]} = [0.7035 \quad 0.8404 \quad 0.8588]$$

Neural Network Back Propagation - Example

- Derivative of Loss Function

$$\nabla_{A^{[2]}} L = -\frac{1}{3}\left[\frac{Y}{A^{[2]}} - \frac{1-Y}{1-A^{[2]}}\right] = [1.1242 \quad -0.3967 \quad 2.3603]$$

Layer 2:

- Derivative of $A^{[2]}$ with respect to $Z^{[2]}$ (tanh activation function)

$$\frac{\partial A_j^{[2]}}{\partial Z_j^{[2]}} = 1 - A_j^{[2]} \quad \left[\frac{\partial A_0^{[2]}}{\partial Z_0^{[2]}} \quad \frac{\partial A_1^{[2]}}{\partial Z_1^{[2]}} \quad \frac{\partial A_2^{[2]}}{\partial Z_2^{[2]}}\right] = [0.2086 \quad 0.1342 \quad 0.1213]$$

$$\nabla_{Z^{[2]}} L = \nabla_{A^{[2]}} L * \left[\frac{\partial A_0^{[2]}}{\partial Z_0^{[2]}} \quad \frac{\partial A_1^{[2]}}{\partial Z_1^{[2]}} \quad \frac{\partial A_2^{[2]}}{\partial Z_2^{[2]}}\right] = [1.1242 \quad -0.3967 \quad 2.3603] * [0.2086 \quad 0.1341 \quad 0.1213] =$$

$$[0.2345 \quad -0.0532 \quad 0.2863]$$

$$\nabla_{W^{[2]}} L = \nabla_{Z^{[2]}} L \, A^{[1]T} = [0.2345 \quad -0.0532 \quad 0.2863] \begin{bmatrix} 0 & 0.9640 \\ -0.7616 & 0.9993 \\ -0.9051 & 1.0 \end{bmatrix} = [-0.2186 \quad 0.4591]$$

$$\nabla_{b^{[2]}} L_i = \sum_{j=0}^{m-1} \nabla_{Z^{[2]}} L_{ij} = [0.4675]$$

$$\nabla_{A^{[1]}} L = W^{[2]T} \nabla_{Z^{[2]}} L = \begin{bmatrix} -1 \\ 1 \end{bmatrix} [0.2345 \quad -0.0532 \quad 0.2863] = \begin{bmatrix} -0.2345 & 0.0532 & -0.2863 \\ 0.2345 & -0.0532 & 0.2863 \end{bmatrix}$$

Neural Network Back Propagation - Example

- Derivative of Loss Function

$$\nabla_{A^{[1]}} L = \begin{bmatrix} -0.2345 & 0.0532 & -0.2863 \\ 0.2345 & -0.0532 & 0.2863 \end{bmatrix}$$

Layer 1:

- Derivative of $A^{[1]}$ with respect to $Z^{[1]}$ (tanh activation function)

$$\frac{\partial A_{ij}^{[1]}}{\partial Z_{ij}^{[1]}} = 1 - A_{ij}^{[1]2} \quad \begin{bmatrix} \frac{\partial A_{00}^{[1]}}{\partial Z_{00}^{[1]}} & \frac{\partial A_{01}^{[1]}}{\partial Z_{01}^{[1]}} & \frac{\partial A_{02}^{[1]}}{\partial Z_{02}^{[1]}} \\ \frac{\partial A_{10}^{[1]}}{\partial Z_{10}^{[1]}} & \frac{\partial A_{11}^{[1]}}{\partial Z_{11}^{[1]}} & \frac{\partial A_{12}^{[1]}}{\partial Z_{12}^{[1]}} \end{bmatrix} = \begin{bmatrix} 1 & 0.4200 & 0.1807 \\ 0.0707 & 0.0013 & 0 \end{bmatrix}$$

$$\nabla_{Z^{[1]}} L = \nabla_{A^{[1]}} L * \begin{bmatrix} \frac{\partial A_{00}^{[1]}}{\partial Z_{00}^{[1]}} & \frac{\partial A_{01}^{[1]}}{\partial Z_{01}^{[1]}} & \frac{\partial A_{02}^{[1]}}{\partial Z_{02}^{[1]}} \\ \frac{\partial A_{10}^{[1]}}{\partial Z_{10}^{[1]}} & \frac{\partial A_{11}^{[1]}}{\partial Z_{11}^{[1]}} & \frac{\partial A_{12}^{[1]}}{\partial Z_{12}^{[1]}} \end{bmatrix} = \begin{bmatrix} -0.2345 & 0.0532 & -0.2863 \\ 0.2345 & -0.0532 & 0.2863 \end{bmatrix} * \begin{bmatrix} 1 & 0.4200 & 0.1807 \\ 0.0707 & 0.0013 & 0 \end{bmatrix} = \begin{bmatrix} -0.2345 & 0.0223 & -0.0517 \\ 0.0166 & -0.0001 & 0 \end{bmatrix}$$

$$\nabla_{W^{[1]}} L = \nabla_{Z^{[1]}} L X^T = \begin{bmatrix} -0.2345 & 0.0223 & -0.0517 \\ 0.0166 & -0.0001 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & -5 \\ 4 & -8 \end{bmatrix} = \begin{bmatrix} -0.3967 & 0.7711 \\ 0.0164 & -0.0328 \end{bmatrix}$$

$$\nabla_{b^{[1]}} L_{ij} = \sum_{j=0}^{m-1} \nabla_{Z^{[1]}} L_{ij} = \begin{bmatrix} -0.2639 \\ 0.0165 \end{bmatrix}$$

Neural Network Training Algorithm

Assume Neural Network with N layers

Input training data: feature matrix X and values Y

Make initial guess for parameters: $W_{epoch=0}^{[k]}$ and $b_{epoch=0}^{[k]}$ for k=1,…,N

Choose learning rate $\alpha>0$

1. Loop for epoch i = 1, 2, …
   - Forward Propagate using X to compute $A_{epoch=i-1}^{[k]}$ for k=1,…,N
   - Back Propagate using X, Y, and $A_{epoch=i-1}^{[k]}$ to determine $\nabla_{W^{[k]}} L_{epoch=i-1}$ and $\nabla_{b^{[k]}} L_{epoch=i-1}$ k=1,…N
   - Update parameters for k=1,…,N

   $$W_{epoch=i}^{[k]} = W_{epoch=i-1}^{[k]} - \alpha \nabla_{W^{[k]}} L_{epoch=i-1}$$
   $$b_{epoch=i}^{[k]} = b_{epoch=i-1}^{[k]} - \alpha \nabla_{b^{[k]}} L_{epoch=i-1}$$
   - Forward Propagate using X to compute $A_{epoch=i}^{[k]}$ for k=1,…,N
   - Compute Loss using $A_{epoch=i}^{[N]}$

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

Neural Network Training - Example

See File IntroML/Examples/Chapter3/Chapter3.1_TrainingAlgorithm.py

- From Back Propagation Example, start with parameter matrices

$$W^{[1]}_{epoch=0} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}, b^{[1]}_{epoch=0} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}, W^{[2]}_{epoch=0} = [-1 \quad 1], b^{[2]}_{epoch=0} = [-0.1]$$

- Pick learning rate $\alpha > 0.1$
- From Back Propagation Example:

$$\nabla_{W^{[1]}} L_{epoch=0} = \begin{bmatrix} -0.3967 & 0.7711 \\ 0.0164 & -0.0328 \end{bmatrix}, \nabla_{b^{[1]}} L_{epoch=0} = \begin{bmatrix} -0.2639 \\ 0.0165 \end{bmatrix}$$

$$\nabla_{W^{[2]}} L_{epoch=0} = [-0.2186 \quad 0.4591], \nabla_{b^{[2]}} L_{epoch=0} = [0.4675]$$

- Updating:

$$W^{[1]}_{epoch=1} = W^{[1]}_{epoch=0} - \alpha \nabla_{W^{[1]}} L_{epoch=0} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} - 0.1 \begin{bmatrix} -0.3967 & 0.7711 \\ 0.0164 & -0.0328 \end{bmatrix} = \begin{bmatrix} 0.5397 & 0.4229 \\ 0.4984 & -0.4967 \end{bmatrix}$$

$$b^{[1]}_{epoch=1} = b^{[1]}_{epoch=0} - \alpha \nabla_{b^{[1]}} L_{epoch=0} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} -0.2639 \\ 0.0165 \end{bmatrix} = \begin{bmatrix} 0.5264 \\ 0.4984 \end{bmatrix}$$

$$W^{[2]}_{epoch=1} = W^{[2]}_{epoch=0} - \alpha \nabla_{W^{[2]}} L_{epoch=0} = [-1 \quad 1] - 0.1 \, [-0.2186 \quad 0.4591] = [-0.9781 \quad 0.9541]$$

$$b^{[2]}_{epoch=1} = b^{[2]}_{epoch=0} - \alpha \nabla_{b^{[2]}} L_{epoch=0} = [-0.1] - 0.1[0.4675] = [-0.1468]$$

Prediction Algorithm

Prediction algorithm makes use parameters computed in Training

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

Use parameter matrices computed during training $W^{[k]}$ and $b^{[k]}$ for k=1,…,N

1. Perform Forward Propagation:
   - Get result of activation for each layer $\tilde{A}^{[k]}$ k=1,…,N

- Predicted labels are $\tilde{A}^{[N]}$ rounded to nearest (0 or 1)

Prediction Algorithm - 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 layer 1 has 2 units and that layer 2 has 1 unit
- Assume parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad W^{[2]} = \begin{bmatrix} -1 & 1 \end{bmatrix} \qquad b^{[2]} = \begin{bmatrix} -0.1 \end{bmatrix}$$

- Assume activation functions $f^{[1]}(z) = \tanh(z)$ and $f^{[2]}(z) = \frac{1}{1+e^{-z}}$
- From Forward Propagation example:

$$A^{[1]} = \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix} \qquad A^{[2]} = \begin{bmatrix} 0.7035 & 0.8404 & 0.8588 \end{bmatrix}$$

- Round entries of $A^{[2]}$ to nearest to 0 or 1 – predicted values [1 1 1]

Accuracy Calculation

- Accuracy calculation for binary classification with Neural Networks is same as that for Logistic Regression

Neural Network Binary Classification – Summary

Component	Subcomponent	Details
Training Data		Input m data points: X (dxm-dimensional feature matrix) Y vector of values (row vector of length m)
Function Structure	Forward Propagation	Assume N layers. For each layer k =1, …, N Linear: $Z^{[k]} = W^{[k]}A^{[k-1]} + b^{[k]} \quad A^{[0]} = X$ Activation: $A^{[k]} = f^{[k]}(Z^{[k]})$ (use sigmoid activation in layer N)
Loss Function		Binary Cross Entropy: $L = -\frac{1}{m}\sum_{j=0}^{m-1} Y_j \ln A_j^{[N]} + \left(1 - Y_j\right)\ln(1 - A_j^{[N]})$
Derivative	Back Propagation	For each layer k=1,…,N Compute $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$
Training Algorithm	Train using Gradient Descent to minimize Loss	Find W, b that minimizes loss Initial guess: $W^{[k]}, b^{[k]}$ for each layer k =1, …, N Choose Learning Rate: $\alpha>0$ Loop: i=1,2,… for fixed number of iterations or until Loss reduced sufficiently For k=1,…,N $W_{epoch=i}^{[k]} = W_{epoch=i-1}^{[k]} - \alpha\nabla_{W^{[k]}} L_{epoch=i-1}$ $b_{epoch=i}^{[k]} = b_{epoch=i-1}^{[k]} - \alpha\nabla_{b^{[k]}} L_{epoch=i-1}$ Perform fixed number or iterations or until Loss reduced sufficiently
Prediction Algorithm	Forward Propagation	Using $W^{[k]}, b^{[k]}$ for each layer k =1, …, N determined in Training Algorithm Given new input feature matrix $\tilde{X}$, perform Forward Propagation to compute $\tilde{A}^{[N]}$ Round entries to nearest (0 or 1) to predict label

3.2 Code Walkthrough Version 2.1

Coding Walkthrough: Version 2.1

Goal of this Section:

- Walkthrough creation of NeuralNetwork class and associated codes to perform binary classification using neural networks

Coding Walkthrough: Version 2.1 To Do

File/Component	To Duo
NeuralNetwork_Base	Add method to list layers and number of parameters
NeuralNetwork	Create derived NeuralNetwork class based on NeuralNetwork_Base class
functions_activation	Add additional activation functions
unittest_forwardbackprop	Add test case for neural networks
driver	Add driver for binary classification for neural network
example_classification	Create function for generating examples

NeuralNetwork_Base – Methods

Method	Input	Description
summary	Nothing	Prints following for each layer: Number of input units Number of output units Number of parameters (sum of number of entries in $W^{[k]}$ and $b^{[k]}$) Also prints total number of parameters Return: Nothing

NeuralNetwork – Methods

Method	Input	Description
__init__	nfeature (integer)	Initialization routine that takes in the number of features Return: nothing
add_layer	nunit (integer) activation (string)	Appends dictionary of information for the layer to info attribute Return: nothing
forward_propagation	X (numpy array)	Performs forward propagation to compute $A^{[k]}$ for k=1,…,N Returns: nothing
back_propagation	X (numpy array) Y (numpy array)	Performs back propagation to compute $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$ for k=1,…,N Returns: nothing
concatenate_param	order (string): “param” or “param_der”	Concatenates all parameters in $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$ into a single numpy row vector
load_param	flat (numpy array) order (string): “param” or “param_der”	Takes values from flat (row vector) and puts them back into $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$ for k=1,…,N

Activation and Loss Functions

Function	Input	Description
functions_activation. activation	activation_fun (string) Z (numpy array)	Add tanh, relu, and softplus cases Return: f(Z)
functions_activation. activation_der	activation_fun (string) Z (numpy array)	Add tanh, relu, and softplus cases Return: f’(Z)

Neural Network 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 Neural Network class
   - Create Neural Network with multiple layers using add_layer method
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

3.3 Multi-Class Classification: Mathematical Foundations

Multiclass Classification Mathematical Foundations

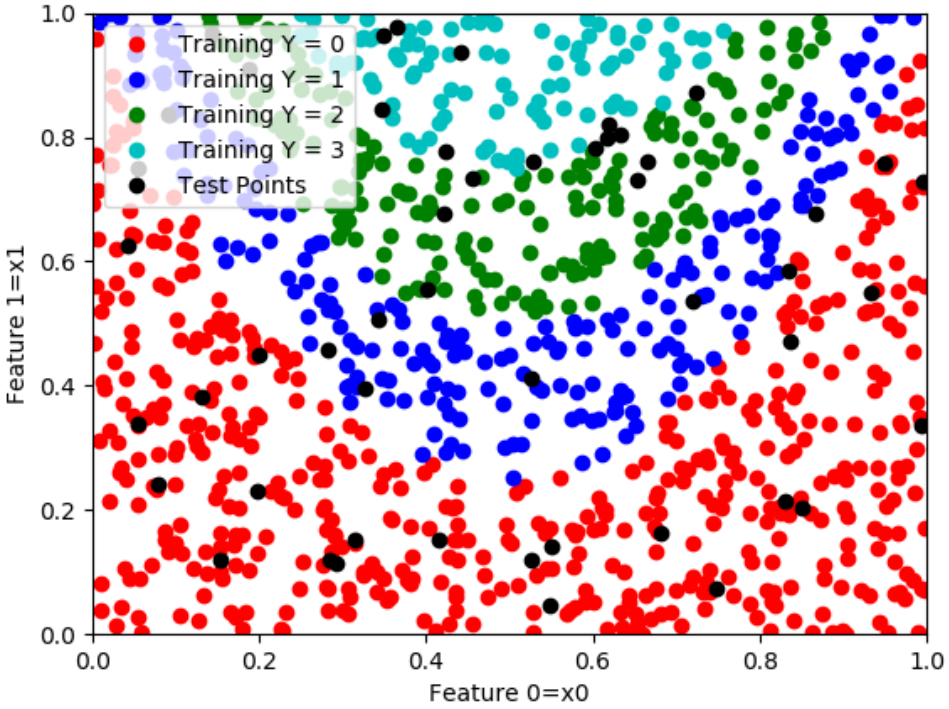
Goal of this Section:

- Present the mathematical foundations of neural networks for multi-class classification, including:
  - Format of training data
  - Function structure and parameters
  - Training algorithm
  - Prediction algorithm

Motivating Example

Training Data:

- Points in (x0,x1) plane each with label (red points are labelled 0, blued points 1, green points 2, cyan points 3)



Goal:

- Find function that best fits 0,1,2,3 labels in training data

Prediction:

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

Limits of Neural Network for Binary Classification

- Neural Network with single unit in final layer is not adequate to handle multi-class classification with more than 2 possible classes
- In this section, to handle this new classification problem
- We show modifications to previous approach are required for:
  - Neural Network structure in final layer
  - Activation function in final layer
  - Loss function

Neural Network: 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 problem where there are m data points, each consisting of a input information vector of length d and value Y, which is one of c classes
- Data point j: input information (feature) vector: $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \cdots \\ X_{d-1,j} \end{bmatrix}$ and output: Y_j (one of 0,1,…,c-1)
- Define the feature matrix (dxm) and output vector (1xm):

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

Training Data – Example Points in Plane

- For points in plane and 4 possible classes as in motivating example, training data consists of points in the plane (X_0,X_1) with label Y
- Suppose 6 data samples with points and labels:

(1,1) label=0

(0.5,2) label=3

(2,3), label=2

(4,2) label=0

(1,0.5) label=1

(-1,1) label=1

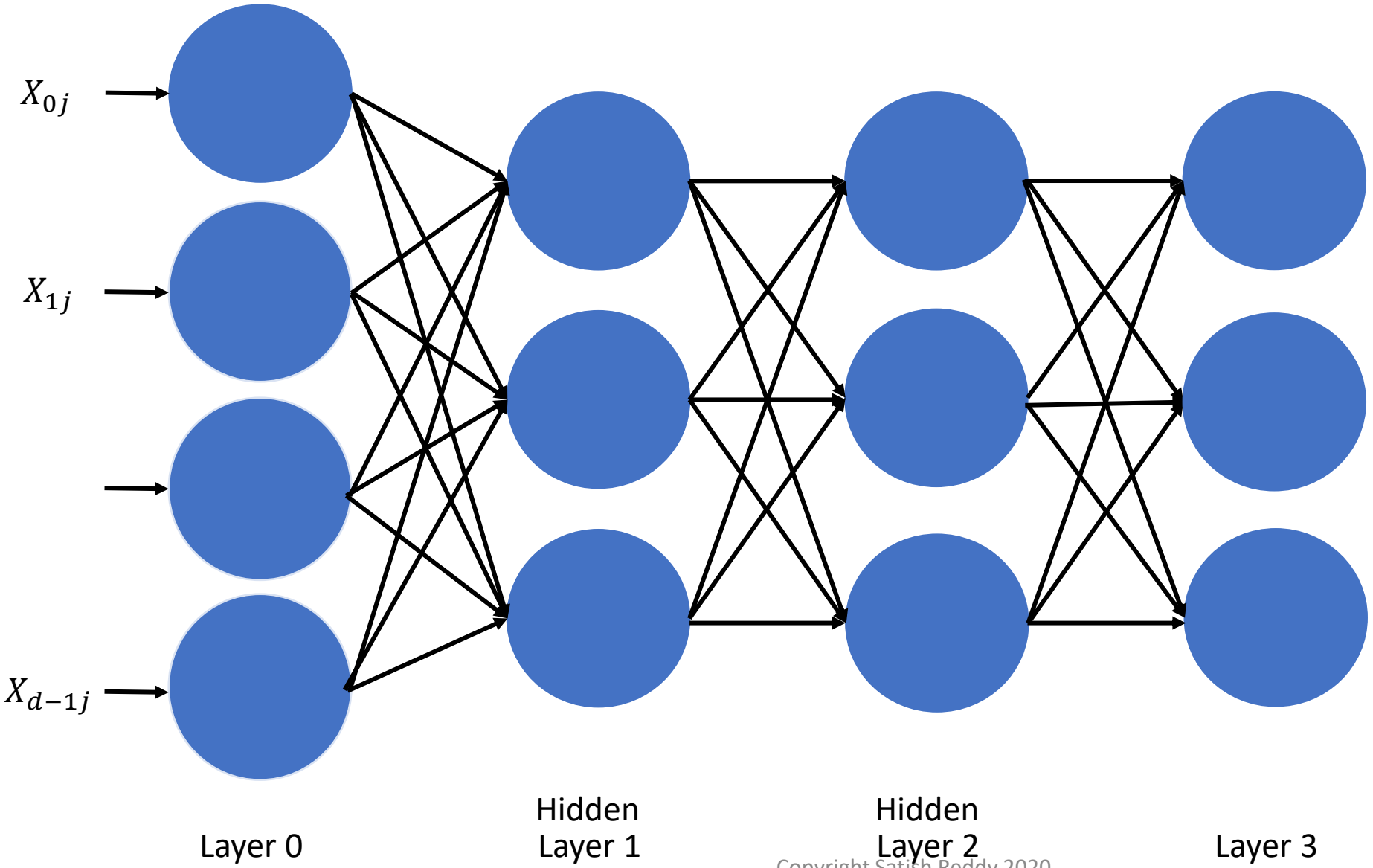
- In this case each sample has 2 features. Feature matrix and value vector are:

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

Neural Network Function Structure

- Inputs:
  - Input feature matrix X (d x m) - X_{ij} = feature i for data point j (i=0,…,d-1, j=0,…,m-1)
  - Label vector Y (1xm) - Y_j label for data point j (assume c classes)
- Number of layers
  - Assume N layers
- Number of units
  - Layer k=1,…,N has $n^{[k]}$ units - note that $n^{[0]} = d$ (number of features)
  - Final layer N has c units (= number of classes)
- Parameters:
  - $W^{[k]}$ is matrix of dimensions ($n^{[k]}$x $n^{[k-1]}$) for layer k
  - $b^{[k]}$ is vector of dimensions ($n^{[k]}$x 1) for layer k
- Activation functions
  - $f^{[k]}(z)$ is activation function for layer k

Neural Network Node Structure – 3 Layer



- Feature data is input at layer 0
- All nodes at layer k-1 are connected to all nodes at layer k
- Example of Feed Forward Neural Network as information moves in one direction only
- Inner layers 1 and 2 are called hidden
- Final layer has same number of nodes as classes

Function Structure Forward Propagation Algorithm

Assume N layer Neural Network

Input: feature matrix X (d features x m samples) and parameter matrices $W^{[k]}$, $b^{[k]}$ for k=1,…,N

1. Define: $A^{[0]} = X$
2. Loop for k=1,…,N (number of layers)
   - Linear part: $Z^{[k]} = W^{[k]}A^{[k-1]} + b^{[k]}$ #matrix of dimension ($n^{[k]}$x m)
   - Activation: $A^{[k]} = f^{[k]}(Z^{[k]})$ #matrix of dimension ($n^{[k]}$x m)

Notes:

- Each layer k will have its own activation function $f^{[k]}(z)$
- For multi-class classification use softmax activation in final layer

Activation Function for Final Layer - Softmax

- Consider matrix Z (c classes x m samples)

$$Z = \begin{bmatrix} Z_{00} & \cdots & Z_{0,m-1} \\ \cdots & \cdots & \cdots \\ Z_{c-1,0} & \cdots & Z_{c-1,m-1} \end{bmatrix}$$

- $A = softmax(Z)$ defined as:

$$A_{ij} = \frac{e^{Z_{ij}}}{\sum_{p=0}^{c-1} e^{Z_{pj}}}$$

- A_{ij} depends on all entries Z_{ij} in column j (not just single entry)
- Notice that A_{ij}>0 and $\sum_{i=0}^{c-1} A_{ij} = 1$, so can consider entries in each column $\{A_{ij}\}$ for i=0,…,c-1 as probability of getting class i

Softmax Activation - Example

See file IntroML/Examples/Chapter3/Chapter3.3_SoftmaxActivation.py

- Consider (4 classes x 3 samples) matrix Z

$$Z = \begin{bmatrix} 0.1 & -0.1 & -0.2 \\ -0.2 & 0.2 & 0.3 \\ -0.3 & 0.1 & 0.2 \\ 0.4 & -0.3 & -0.5 \end{bmatrix}$$

- Compute all entries in middle column:

Same Denominator

$$A_{01} = \frac{e^{Z_{01}}}{\sum_{p=0}^{3} e^{Z_{p1}}} = \frac{e^{-0.1}}{e^{-0.1} + e^{0.2} + e^{0.1} + e^{-0.3}} = 0.2278$$

$$A_{11} = \frac{e^{Z_{11}}}{\sum_{p=0}^{3} e^{Z_{p1}}} = \frac{e^{0.2}}{e^{-0.1} + e^{0.2} + e^{0.1} + e^{-0.3}} = 0.3075$$

$$A_{21} = \frac{e^{Z_{21}}}{\sum_{p=0}^{3} e^{Z_{p1}}} = \frac{e^{0.2}}{e^{-0.1} + e^{0.2} + e^{0.1} + e^{-0.3}} = 0.2782$$

$$A_{31} = \frac{e^{Z_{31}}}{\sum_{p=0}^{3} e^{Z_{p1}}} = \frac{e^{-0.3}}{e^{-0.1} + e^{0.2} + e^{0.1} + e^{-0.3}} = 0.1865$$

$$A = \begin{bmatrix} 0.2659 & 0.2278 & 0.2049 \\ 0.1970 & 0.3075 & 0.3378 \\ 0.1782 & 0.2782 & 0.3056 \\ 0.3589 & 0.1865 & 0.1518 \end{bmatrix}$$

Neural Network Forward Propagation - Example

See file IntroML/Examples/Chapter3/Chapter3.3_ForwardPropagation.py

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

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

- Assume that layer 1 has 2 units and that layer 2 has 3 unit
- Assume parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad W^{[2]} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -2 & 1 \end{bmatrix} \qquad b^{[2]} = [-0.1]$$

- Assume activation functions $f^{[1]}(z) = \tanh(z)$ and $f^{[2]}(z) = softmax(Z)$

Forward Propagation:

- Layer 1:

$$Z^{[1]} = W^{[1]}X + b^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ -2 & -5 & -8 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0 & -1 & -1.5 \\ 2 & 4 & 6.5 \end{bmatrix}$$

$$A^{[1]} = f(Z) = \begin{bmatrix} \tanh(0) & \tanh(-1) & \tanh(-1.5) \\ \tanh(2) & \tanh(4) & \tanh(6.5) \end{bmatrix} = \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix}$$

Neural Network Forward Propagation - Example

- Layer 2:

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix} + [-0.1]$$

$$= \begin{bmatrix} 0.8640 & 1.6609 & 1.8051 \\ -1.0640 & -1.8609 & -2.0051 \\ 0.8640 & 2.4225 & 2.7103 \end{bmatrix}$$

$$A^{[2]} = softmax(Z) = \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix}$$

One Hot Vector

- For Binary Classification label Y_j is 0 or 1
- For Multi-Class Classification with c classes, Y_j is one of 0,…,c-1
- Customary to represent as a One Hot vector
- Suppose Y_j =p, then one hot vector is:
- $Y_j^h = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ← 1 in position p
- Example: suppose there are 4 classes and $Y = [0 \quad 3 \quad 2 \quad 0]$, one-hot matrix is

$$Y^h = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Inverse of One-Hot Vector

- Given one-hot vector:
- $Y_j^h = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ← 1 in position p
- Inverse of one-hot vector is index of largest entry. In this case $Y_j = p$
- In numpy can use argmax function
- This applies to vectors that are not just 0 and 1 and to matrices (find index of largest entry for each column – first index in case of ties)

$$A = \begin{bmatrix} 0.2 & 0.3 & 0.4 & 0.3 \\ 0.4 & 0.5 & 0.1 & 0.2 \\ 0.0 & 0.1 & 0.3 & 0.4 \\ 0.4 & 0.1 & 0.2 & 0.1 \end{bmatrix}$$

$$argmax(over\ columns\ of\ A) = [1 \quad 1 \quad 0 \quad 2]$$

Cross Entropy Loss Function

- Loss function is

$$Loss = L = -\frac{1}{m}\sum_{j=0}^{m-1}\sum_{i=0}^{n^{[N]}-1} Y_{ij}^{h}\ln(A_{ij}^{[N]})$$

- Sum is over all units i=0,…, $n^{[N]} - 1$ in final layer and all samples j=0,…,m-1
- Note that for each sample j, only one of Y_{ij}^{h} is non-zero, as these are entries of the one-hot vector for that sample

Neural Network Training Phase

- Training phase attempts to find suitable parameter matrices $W^{[k]}$, $b^{[k]}$ for k=1,…,N that minimize the loss function when applied to the training data
- Use optimization algorithm (example: Gradient Descent) to minimize Loss function
- Need to compute derivatives $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$ for k=1,…,N

Cross Entropy Loss Function - Gradient

- Gradient of Cross Entropy loss function is:

$$\nabla_{A^{[N]}} L = -\frac{1}{m}\frac{Y^h}{A^{[N]}}$$

(pointwise division – each entry of one-hot matrix Y^h divided by corresponding entry of $A^{[N]}$)

Softmax Derivative

- To simplify notation, let's remove the sample axis and assume

$$Z = \begin{bmatrix} Z_0 \\ \vdots \\ Z_{c-1} \end{bmatrix}$$

- Define

$$A_k = \frac{e^{Z_k}}{\sum_{p=0}^{c-1} e^{Z_p}}$$

- Working out derivatives (consider 2 cases)

$$case\ k = i\text{:}\ \frac{\partial A_i}{\partial Z_i} = \frac{e^{Z_i}}{\sum_{p=0}^{c-1} e^{Z_p}} - \frac{e^{Z_i} e^{Z_i}}{\left[\sum_{p=0}^{c-1} e^{Z_p}\right]^2} = A_i - A_i^2$$

$$case\ k \neq i\text{:}\ \frac{\partial A_k}{\partial Z_i} = -\frac{e^{Z_i} e^{Z_k}}{\left[\sum_{p=0}^{c-1} e^{Z_p}\right]^2} = -A_i A_k$$

Chain Rule using Softmax

- With the notation of the last section, define loss function L(A) - where A is a vector of length c
- Given $\nabla_A L$ what is $\nabla_Z L$ when A is related to Z by the softmax function?
- Can't use formula for other activation functions as Z_i depends on all of $A_0,\ldots,A_{c-1}$ and not just A_i
- Using chain rule

$$\frac{\partial L}{\partial Z_i} = \sum_{k=0}^{c-1} \frac{\partial L}{\partial A_k}\frac{\partial A_k}{\partial Z_i} = \frac{\partial L}{\partial A_i}\frac{\partial A_i}{\partial Z_i} + \sum_{k=0,k\neq i}^{c-1} \frac{\partial L}{\partial A_k}\frac{\partial A_k}{\partial Z_i} = \frac{\partial L}{\partial A_i}(A_i - A_i^2) - \sum_{k=0,k\neq i}^{c-1} \frac{\partial L}{\partial A_k} A_i\, A_k$$

$$\frac{\partial L}{\partial Z_i} = \frac{\partial L}{\partial A_i} A_i - \sum_{k=0}^{c-1} \frac{\partial L}{\partial A_k} A_i\, A_k = A_i \frac{\partial L}{\partial A_i} - A_i \sum_{k=0}^{c-1} \frac{\partial L}{\partial A_k} A_k$$

Softmax Derivative – General Case

- The results of the previous slide can be generalized to case with sample axis

$$Z = \begin{bmatrix} Z_{00} & \cdots & Z_{0,m-1} \\ \cdots & \cdots & \cdots \\ Z_{c-1,0} & \cdots & Z_{c-1,m-1} \end{bmatrix}$$

- Define

$$A_{kj} = \frac{e^{Z_{kj}}}{\sum_{p=0}^{c-1} e^{Z_{pj}}}$$

- Index j identifies samples and samples are independent of each other
- It can be shown:

$$case\ k = i\text{:}\ \frac{\partial A_{ij}}{\partial Z_{ij}} = \frac{e^{Z_{ij}}}{\sum_{p=0}^{c-1} e^{Z_{pj}}} - \frac{e^{Z_{ij}} e^{Z_{ij}}}{\left[\sum_{p=0}^{c-1} e^{Z_{pj}}\right]^2} = A_{ij} - A_{ij}^2$$

$$case\ k \neq i\text{:}\ \frac{\partial A_{kj}}{\partial Z_{ij}} = -\frac{e^{Z_{ij}} e^{Z_{kj}}}{\left[\sum_{p=0}^{c-1} e^{Z_{pj}}\right]^2} = -A_{ij} A_{kj}$$

Chain Rule using Softmax – General Case

- Can generalize the chain rule for vectors to matrices
- Assume that loss is L(A) where A is (c classes x m samples) and A = softmax(Z)
- It can be shown:

$$\frac{\partial L}{\partial Z_{ij}} = A_{ij}\frac{\partial L}{\partial A_{ij}} - A_{ij}\sum_{k=0}^{c-1}\frac{\partial L}{\partial A_{kj}}A_{kj}$$

- The sum is over the product of entries in column j of $\nabla_A L$ and A
- If we denote $S_j = \sum_{k=0}^{c-1}\frac{\partial L}{\partial A_{kj}}A_{kj}$ for j=0,…,m-1, then

$$\frac{\partial L}{\partial Z_{ij}} = A_{ij}\frac{\partial L}{\partial A_{ij}} - A_{ij}S_j$$

In matrix form

$$\nabla_Z L = \nabla_A L * A - A * S$$

- The * corresponds to pointwise multiplication in the first case and pointwise multiplication with broadcasting in the second case
  - S is a row vector. Same entry in column j of S multiplies all entries in column j of A

Back Propagation Algorithm

Assume N layers and that Forward Propagation has been performed

Input: feature matrix X, label vector Y, parameter matrices $W^{[k]}$ and $b^{[k]}$ for k=1,…,N

1. Compute $\nabla_{A^{[N]}} L$
2. Loop for k=N,…,1

- For layer N (softmax activation)
- Compute $S_j = \sum_{k=0}^{n^{[N]}-1} \frac{\partial L}{\partial A_{kj}^{[N]}} A_{kj}^{[N]} \quad \text{for } j = 0, \dots, m-1$
- Compute $\nabla_{Z^{[k]}} L = \nabla_{A^{[k]}} L * A^{[N]} - A^{[N]} * S$
- For layers k=1,…,N-1:
- Compute $\frac{\partial A_{ij}^{[k]}}{\partial Z_{ij}^{[k]}} = \frac{df^{[k]}}{dz}(Z_{ij}^{[k]}) \text{ for } i = 0, \dots, n^{[k]} - 1, j = 0, \dots, m-1$
- Compute $\nabla_{Z^{[k]}} L = \nabla_{A^{[k]}} L * \frac{\partial A_{ij}^{[k]}}{\partial Z_{ij}^{[k]}}$ (pointwise-multiplication)
- $\nabla_{W^{[k]}} L = \nabla_{Z^{[k]}} L A^{[k-1]^T}$
- $\nabla_{b^{[k]}} L_i = \sum_{j=0}^{m-1} \nabla_{Z^{[k]}} L_{ij}, \quad i = 0, \dots, n^{[k]} - 1$
- If k>1:$\nabla_{A^{[k-1]}} L = W^{[k]T} \nabla_{Z^{[k]}} L$

Back Propagation - Example

- See file IntroML/Examples/Chapter3/Chapter3.3_BackPropagation.py
- Consider a case of 2 features and 3 data points (m=3)

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

- Assume that layer 1 has 2 units and that layer 2 has 3 unit
- Assume parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad W^{[2]} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -2 & 1 \end{bmatrix} \qquad b^{[2]} = \begin{bmatrix} -0.1 \end{bmatrix}$$

- Assume activation functions $f^{[1]}(z) = \tanh(z)$ and $f^{[2]}(z) = softmax(Z)$
- From the forward propagation example:

$$A^{[1]} = \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix} \qquad A^{[2]} = \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix}$$

Back Propagation - Example

- Derivative of Loss with respect to $A^{[2]}$ - gradient given by:

$$\nabla_{A^{[2]}} L = -\frac{1}{m}\frac{Y^h}{A^{[2]}}$$

- Y and its one-hot version are:

$$Y = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \qquad Y^h = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\nabla_{A^{[2]}} L = -\frac{1}{3}\begin{bmatrix} 1/0.4661 & 0/0.3153 & 0/0.2862 \\ 0/0.0678 & 1/0.0093 & 0/0.0063 \\ 0/0.4661 & 0/0.6753 & 1/0.7075 \end{bmatrix} = \begin{bmatrix} -0.7151 & 0 & 0 \\ 0 & -35.7788 & 0 \\ 0 & 0 & -0.4711 \end{bmatrix}$$

Back Propagation - Example

- Layer 2 (softmax activation)

$$\nabla_{A^{[2]}} L * A^{[2]} = \begin{bmatrix} -0.7151 & 0 & 0 \\ 0 & -35.7788 & 0 \\ 0 & 0 & -0.4711 \end{bmatrix} * \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix} = \begin{bmatrix} -0.3333 & 0 & 0 \\ 0 & -0.3333 & 0 \\ 0 & 0 & -0.3333 \end{bmatrix}$$

- Recall: $S_j = \sum_{k=0}^{c-1} \frac{\partial L}{\partial A_{kj}} A_{kj}$ - Summing the above over each column: $S = [-0.3333 \quad -0.3333 \quad -0.3333]$

- Gradient with respect to $Z^{[2]}$ given by

$$\nabla_{Z^{[2]}} L = \nabla_{A^{[2]}} L * A^{[2]} - A^{[2]} * S = \begin{bmatrix} -0.3333 & 0 & 0 \\ 0 & -0.3333 & 0 \\ 0 & 0 & -0.3333 \end{bmatrix} - \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix} * [-0.3333 \quad -0.3333 \quad -0.3333]$$

$$\nabla_{Z^{[2]}} L = \begin{bmatrix} -0.2609 & 0.0331 & 0.0273 \\ 0.0015 & -0.3333 & 0.0000 \\ 0.0724 & 0.1520 & -0.1665 \end{bmatrix}$$

$$\nabla_{W^{[2]}} L = \nabla_{Z^{[2]}} L \, A^{[1]T} = \begin{bmatrix} -0.2609 & 0.0331 & 0.0273 \\ 0.0015 & -0.3333 & 0.0000 \\ 0.0724 & 0.1520 & -0.1665 \end{bmatrix} \begin{bmatrix} 0 & 0.9640 \\ -0.7616 & 0.9993 \\ -0.9051 & 1 \end{bmatrix} = \begin{bmatrix} -0.0500 & -0.1911 \\ 0.2538 & -0.3316 \\ 0.0349 & 0.0553 \end{bmatrix}$$

- For $\nabla_{b^{[2]}} L$ sum $\nabla_{Z^{[2]}} L$ along each row

$$\nabla_{b^{[2]}} L = \begin{bmatrix} -0.2005 \\ -0.3318 \\ 0.0580 \end{bmatrix}$$

Back Propagation - Example

$$\nabla_{A^{[1]}} L = W^{[2]T} \nabla_{Z^{[2]}} L = \begin{bmatrix} -1 & 1 & -2 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -0.2609 & 0.0331 & 0.0273 \\ 0.0015 & -0.3333 & 0.0000 \\ 0.0724 & 0.1520 & -0.1665 \end{bmatrix} = \begin{bmatrix} 0.1176 & -0.6705 & 0.3057 \\ -0.1900 & 0.5185 & -0.1392 \end{bmatrix}$$

- Layer 1
- For f=tanh(z) activation function:

$\frac{\partial A_{ij}^{[1]}}{\partial Z_{ij}^{[1]}} = \frac{df^{[1]}}{dz}\left(Z_{ij}^{[1]}\right) = 1 - (A_{ij}^{[1]})^2$ this matrix is : $\begin{bmatrix} 1 & 0.4200 & 0.1807 \\ 0.0707 & 0.0013 & 0 \end{bmatrix}$

$$\nabla_{Z^{[1]}} L = \nabla_{A^{[1]}} L * \left[\frac{\partial A_{ij}^{[1]}}{\partial Z_{ij}^{[1]}}\right] = \begin{bmatrix} 0.1176 & -0.6705 & 0.3057 \\ -0.1900 & 0.5185 & -0.1392 \end{bmatrix} * \begin{bmatrix} 1 & 0.4200 & 0.1807 \\ 0.0707 & 0.0013 & 0 \end{bmatrix} = \begin{bmatrix} 0.1176 & -0.2816 & 0.0552 \\ -0.0134 & 0.0007 & 0 \end{bmatrix}$$

$$\nabla_{W^{[1]}} L = \nabla_{Z^{[1]}} L X^T = \begin{bmatrix} 0.1176 & -0.2816 & 0.0552 \\ -0.0134 & 0.0007 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & -5 \\ 4 & -8 \end{bmatrix} = \begin{bmatrix} -0.2246 & 0.7309 \\ -0.0120 & 0.0234 \end{bmatrix}$$

- For $\nabla_{b^{[1]}} L$ sum $\nabla_{Z^{[1]}} L$ along each row

$$\nabla_{b^{[1]}} L = \begin{bmatrix} -0.1088 \\ -0.0127 \end{bmatrix}$$

Neural Network Training Algorithm

- Neural Network Training Algorithm for multi-class classification is the same as the training algorithm for Neural Networks for binary classification, with minor modifications for back propagation as noted in previous slides.

Prediction Algorithm

Prediction algorithm makes use of parameters computed in Training

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

Use $W^{[k]}$ and $b^{[k]}$ for k=1,…,N determined in training

1. Perform Forward Propagation:
   - Get result of activation at final layer $\tilde{A}^{[N]}$(c classes x p samples)
2. Prediction:
   - For each sample j (column), $\tilde{A}_{ij}^{[N]}$ is probability of getting label i
   - For each sample j (column), predicted label row index i with max probability

   (In case of tie, choose first index with largest probability)

Prediction Algorithm - Example

- See file IntroML/Examples/Chapter3/Chapter3.3_Prediction.py
- Consider a case of 2 features and 3 data points (m=3)

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

- Assume that layer 1 has 2 units and that layer 2 has 3 unit
- Assume parameter matrices

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} -1 & 1 \\ 1 & -1 \\ -2 & 1 \end{bmatrix} \quad b^{[2]} = [-0.1]$$

- Assume activation functions $f^{[1]}(z) = \tanh(z)$ and $f^{[2]}(z) = softmax(Z)$
- From Forward Propagation Example slides

$$A^{[2]} = softmax\left(Z^{[2]}\right) = \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix}$$

Row index 0,2 correspond to max choose 0

Row index 2 corresponds to max

Row index 2 corresponds to max

$$pred = row\ indexofmax\ = [0 \quad 2 \quad 2]$$

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:

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

Accuracy Calculation - Example

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

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

- From Predication Algorithm Example:

$$\tilde{P} = \begin{bmatrix} 0 & 2 & 2 \end{bmatrix}$$

- Prediction $\tilde{P}$ matches $\tilde{Y}$ for 1 out of 3 entries, so

$$Accuracy = 0.3333$$

Neural Network Multi-class Classification – Summary

Component	Subcomponent	Details
Training Data		Input m data points: X (dxm-dimensional feature matrix) Y vector of values (row vector of length m)
Function Structure	Forward Propagation	Assume N layers. For each layer k =1, …, N Linear: $Z^{[k]} = W^{[k]}A^{[k-1]} + b^{[k]} \quad A^{[0]} = X$ Activation: $A^{[k]} = f^{[k]}(Z^{[k]})$ (use softmax activation in final layer)
Loss Function		Compute the one-hot matrix: Y^h from Y Cross Entropy: $L = -\frac{1}{m}\sum_{j=0}^{m-1} Y_{ij}^h \ln A_j^{[N]}$
Derivative	Back Propagation	For each layer k=1,…,N Compute $\nabla_{W^{[k]}} L$ and $\nabla_{b^{[k]}} L$
Training Algorithm	Train using Gradient Descent to minimize Loss	Find W, b that minimizes loss Initial guess: $W^{[k]}, b^{[k]}$ for each layer k =1, …, N Choose Learning Rate: $\alpha>0$ Loop: i=1,2,… for fixed number of iterations or until Loss reduced sufficiently For k=1,…,N $W_{epoch=i}^{[k]} = W_{epoch=i-1}^{[k]} - \alpha\nabla_{W^{[k]}} L_{epoch=i-1}$ $b_{epoch=i}^{[k]} = b_{epoch=i-1}^{[k]} - \alpha\nabla_{b^{[k]}} L_{epoch= i-1}$ Perform fixed number or iterations or until Loss reduced sufficiently
Prediction Algorithm	Forward Propagation	Using $W^{[k]}, b^{[k]}$ for each layer k =1, …, N determined in Training Algorithm Given new input feature matrix $\tilde{X}$, perform Forward Propagation to compute $\tilde{A}^{[N]}$ Predicted class label for each column of $\tilde{A}^{[N]}$ is row index with largest value

3.4 Code Walkthrough Version 2.2

Coding Walkthrough: Version 2.2

Goal of this Section:

- Walkthrough update of neural network codes to handle multi-class classification

Coding Walkthrough: Version 2.2 To Do

File/Component	To Do
NeuralNetwork	Minor fixes to handle softmax activation function
functions_activation	Add softmax activation function and derivative
functions_loss	Add cross entropy loss function and derivative
unittest_forwardbackprop	Add test case using cross entropy loss and softmax activation
driver	Add driver for multiclass classification
example_classification	Update to generate multiclass examples