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

Chapter 5 Neural Networks

Neural Networks

Section	Title	Description
5.1	Neural Networks: Mathematical Foundations	This section presents the mathematical foundations for Neural Networks for binary classification. It builds upon the foundations for Linear and Logistic Regression.
5.2	Implementation of Activation Functions	This section discusses details of implementation of activation functions to avoid numerical overflow
5.3	Code Walkthrough Version 2.1	Walkthrough of updates to machine learning framework for Neural Networks.
5.4	Softmax Activation	This section introduces the softmax activation function used for multi-class classification.
5.5	One-hot Matrix	This section introduces the one-hot matrix used for multi-class classification.
5.6	Multi-class Classification: Mathematical Foundations	This section presents the mathematical foundations for Neural Networks for multi-class classification.
5.7	Code Walkthrough Version 2.2	Walkthrough of updates to machine learning framework for multi-class classification.

5.1 Neural Networks: Mathematical Foundations

Neural Networks: Mathematical Foundations

Goal of this Section:

- Present the mathematical foundations for Neural Networks for binary classification

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 limited in their ability to map input information to output information
- Need a more general function structure that can fit a larger range of training data sets

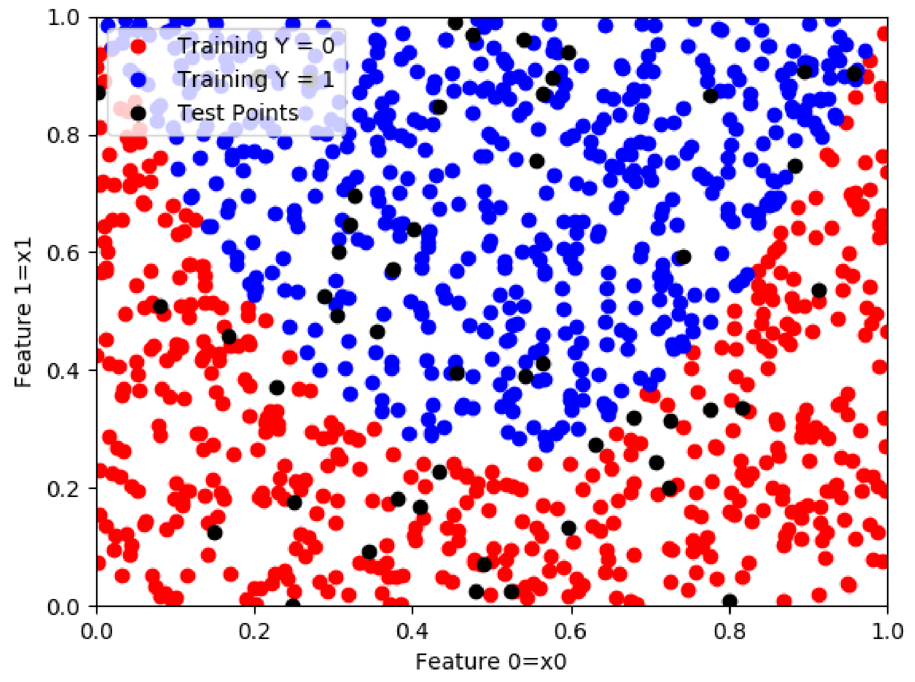
Motivating Example - Binary Classification

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 for complicated classification problems
- Also can be used for classification with more than 2 classes



Neural Network: Binary Classification

General approach has following components and phases:

1. Training Data
2. Function Structure
3. Loss Function
4. Training Phase
5. Prediction Phase

Training Data

Assume training input information has d features

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

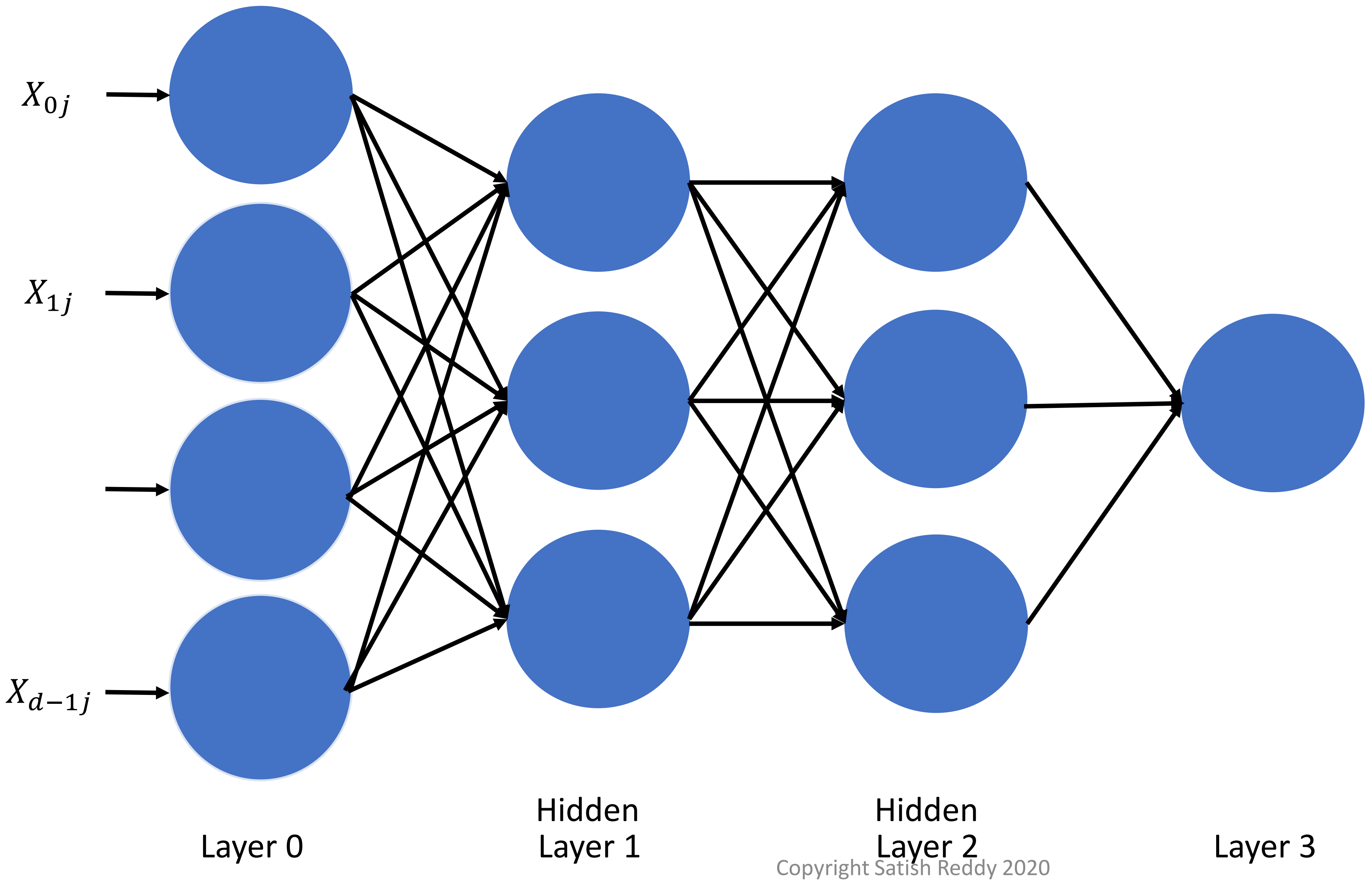
$$X = \begin{bmatrix} X_{00} & \dots & X_{0,m-1} \\ \dots & \dots & \dots \\ X_{d-1,0} & \dots & X_{d-1,m-1} \end{bmatrix} \qquad Y = [Y_0 \quad \dots \quad Y_{m-1}]$$

- This is the same as for Logistic Regression

Neural Network Function Structure

- 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



- Input info entered at layer 0 – number of units = number of features
- 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.

Neural Network Layers - Example

Consider the Neural Network from previous slide:

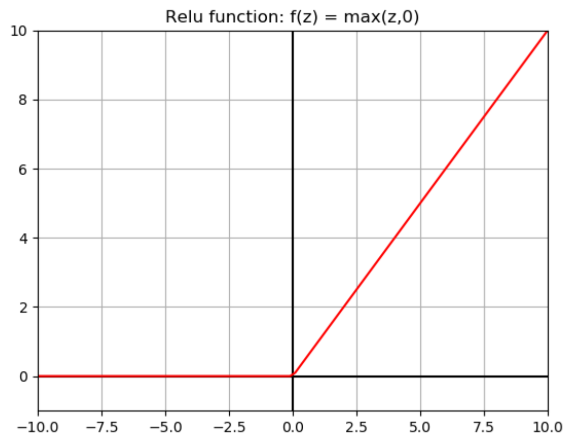
- Number of features = 4
- Number of units layer 1 = 3
- Number of units layer 2 = 3
- Number of units layer 3 = 1

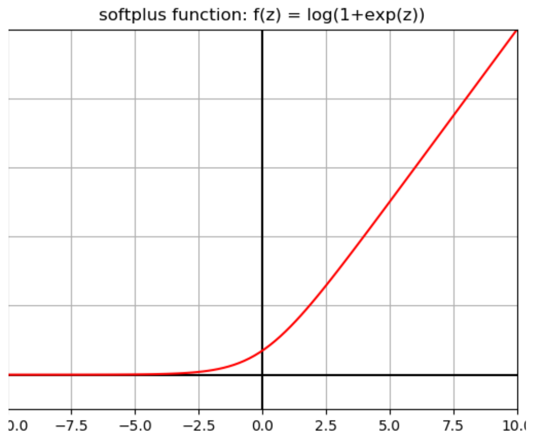
What are dimensions of $W^{[k]}$ and $b^{[k]}$ for each layer k=1,2,3?

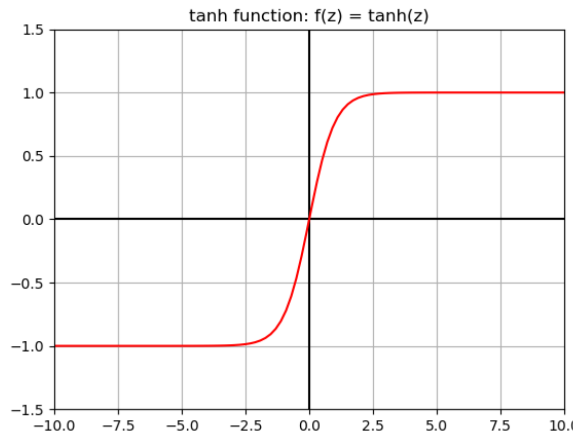
Layer	Units: Previous Layer	Units : Current Layer	Dimension W	Dimension b	Total Number Of Parameters
1	4	3	3x4	3x1	12+3 = 15
2	3	3	3x3	3x1	9 + 3 = 12
3	3	1	1x3	1x1	3 + 1 = 4
Total					31

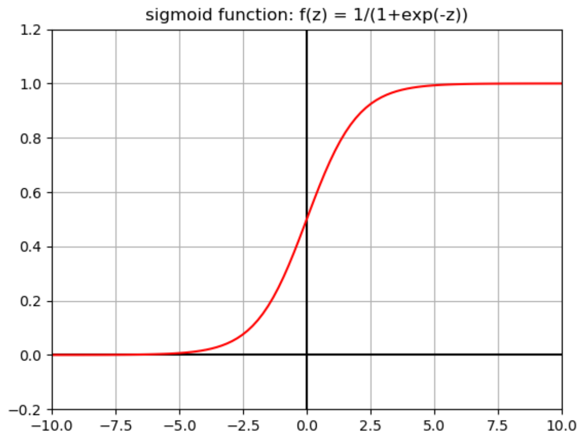
Activation Functions

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









Vanishing Gradients

- Recall that in Training Algorithm, update to W and b depends on gradients $\nabla_W L$ and $\nabla_b L$
- For sigmoid and tanh activation functions, for a wide range of Z values -> gradients will be extremely close to 0 (vanishing gradient). If gradients are close to 0, then training will be slow.
- Vanishing gradient issue often arises in neural networks with multiple layers
- Addressing vanishing gradient is motivation for using Relu, Softplus, and related activation functions, where derivative is close to 1 for positive Z
- See following for more details: https://en.wikipedia.org/wiki/Vanishing_gradient_problem

Function Structure Forward Propagation Algorithm

Assume N layer Neural Network

Input:

- Feature matrix X (d features x m samples)
- 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 binary classification, final layer has 1 unit with sigmoid activation

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 units with parameter matrices

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

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

Forward Propagation:

- Layer 1:

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

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

Neural Network Forward Propagation - Example

- Layer 2:

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} = \begin{bmatrix} -1 & 1 \end{bmatrix} \begin{bmatrix} 0 & -0.7616 & -0.9051 \\ 0.9640 & 0.9993 & 1.0 \end{bmatrix} + \begin{bmatrix} -0.1 \end{bmatrix}$$
$$= \begin{bmatrix} 0.8640 & 1.6609 & 1.8051 \end{bmatrix}$$

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

Binary Cross Entropy Loss Function

- Loss function is same as for Logistic Regression
- Average of binary cross entropy applied to activation after 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

Derivatives of Activation Functions

- Compute derivatives of Relu, softplus, tanh, and sigmoid
- $A = Relu(z) = \max(z, 0)$

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

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

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

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

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

- $A = \mathrm{sigmoid}(z) = \frac{1}{1 + e^{-z}}$

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

Back Propagation – Connecting Layers

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

$$\begin{tikzcd}
 & \text{Forward Propagation} & \text{Back Propagation} & \\
 & Z^{[1]} = W^{[1]}X + b^{[1]} \arrow[dd] & \nabla_{Z^{[1]}}L_{ij} = \nabla_{A^{[1]}}L_{ij} * \frac{\partial A^{[1]}_{ij}}{\partial Z^{[1]}_{ij}} \arrow[r] & \begin{gathered} \nabla_{W^{[1]}}L = \nabla_{Z^{[1]}}LX^{T} \\ \nabla_{b^{[1]}}L_i = \sum_{j=0}^{m-1} \nabla_{Z^{[1]}}L_{ij} \end{gathered} \\
\text{Layer 1} & & & \\
 & A^{[1]} = f^{[1]}(Z^{[1]}) \arrow[dd] & \nabla_{A^{[1]}}L \arrow[uu] & \\
 & & \text{Missing Connection} & \\
 & Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \arrow[dd] & \nabla_{Z^{[2]}}L_{ij} = \nabla_{A^{[2]}}L_{ij} * \frac{\partial A^{[2]}_{ij}}{\partial Z^{[2]}_{ij}} \arrow[r] & \begin{gathered} \nabla_{W^{[2]}}L = \nabla_{Z^{[2]}}LA^{[1]T} \\ \nabla_{b^{[2]}}L_i = \sum_{j=0}^{m-1} \nabla_{Z^{[2]}}L_{ij} \end{gathered} \\
\text{Layer 2} & & & \\
 & A^{[2]} = f^{[2]}(Z^{[2]}) \arrow[dr] & \nabla_{A^{[2]}}L \arrow[uu] & \\
 & & \text{Loss} = L(A^{[2]}, Y) \arrow[u] &
\end{tikzcd}$$

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 3.2

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

- In general for k>1

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

Back Propagation Algorithm

Assume N layers

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

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 $\nabla_{Z^{[k]}} L_{ij} = \nabla_{A^{[k]}} L_{ij} * \frac{\partial A_{ij}^{[k]}}{\partial Z_{ij}^{[k]}}$ (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

- 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

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

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

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

Neural Network Back Propagation - Example

- Derivative of Loss Function

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

Layer 2:

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

$$\frac{\partial A_j^{[2]}}{\partial Z_j^{[2]}} = A_j^{[2]} - A_j^{[2]2} \quad \begin{bmatrix} \frac{\partial A_0^{[2]}}{\partial Z_0^{[2]}} & \frac{\partial A_1^{[2]}}{\partial Z_1^{[2]}} & \frac{\partial A_2^{[2]}}{\partial Z_2^{[2]}} \end{bmatrix} = [0.2086 \quad 0.1342 \quad 0.1213]$$

$$\nabla_{Z^{[2]}}L = \nabla_{A^{[2]}}L * \begin{bmatrix} \frac{\partial A_0^{[2]}}{\partial Z_0^{[2]}} & \frac{\partial A_1^{[2]}}{\partial Z_1^{[2]}} & \frac{\partial A_2^{[2]}}{\partial Z_2^{[2]}} \end{bmatrix} = [1.1242 \quad -0.3967 \quad 2.3603] * [0.2086 \quad 0.1341 \quad 0.1213] = [0.2345 \quad -0.0532 \quad 0.2863]$$

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

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

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

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)

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

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

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

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

Neural Network Training Algorithm

Assume Neural Network with N layers

Input training data: feature matrix X and values Y

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

Choose learning rate $\alpha > 0$

1. Loop for epoch i = 1, 2, …
   - Forward Propagate using X to compute $A^{[k]}_{epoch=i-1}$ for k=1,…,N
   - Back Propagate using X, Y, and $A^{[k]}_{epoch=i-1}$ 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

   $$W^{[k]}_{epoch=i} = W^{[k]}_{epoch=i-1} - \alpha \nabla_{W^{[k]}} L_{epoch=i-1}$$
   $$b^{[k]}_{epoch=i} = b^{[k]}_{epoch=i-1} - \alpha \nabla_{b^{[k]}} L_{epoch=i-1}$$

   - Forward Propagate using X to compute $A^{[k]}_{epoch=i}$ for k=1,…,N
   - Compute Loss using $A^{[N]}_{epoch=i}$

Loop for fixed number of iterations

Neural Network Training - Example

- 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

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

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

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

- 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	Initial guess: $W_{epoch=0}^{[k]}, b_{epoch=0}^{[k]}$ for each layer k =1, …, N Choose Learning Rate: α>0 Loop: i=1,2,… for fixed number of iterations Perform forward propagation Perform back propagation to compute $\nabla_{W^{[k]}} L_{epoch=i-1}$ and $\nabla_{b^{[k]}} L_{epoch=i-1}$ for k=1,…,N $W_{epoch=i}^{[k]} = W_{epoch=i-1}^{[k]} - \alpha \nabla_{W^{[k]}} L_{epoch=i-1}$ for k=1,…,N $b_{epoch=i}^{[k]} = b_{epoch=i-1}^{[k]} - \alpha \nabla_{b^{[k]}} L_{epoch=i-1}$ for k=1,…,N
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

5.1 Neural Network – Jupyter Notebook DEMO

- Open file IntroML/Examples/Chapter5/NeuralNetworkBinary.ipynb
- Has examples of
  - Forward Propagation
  - Back Propagation
  - Training Algorithm
  - Prediction Algorithm
  - Accuracy Calculation

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

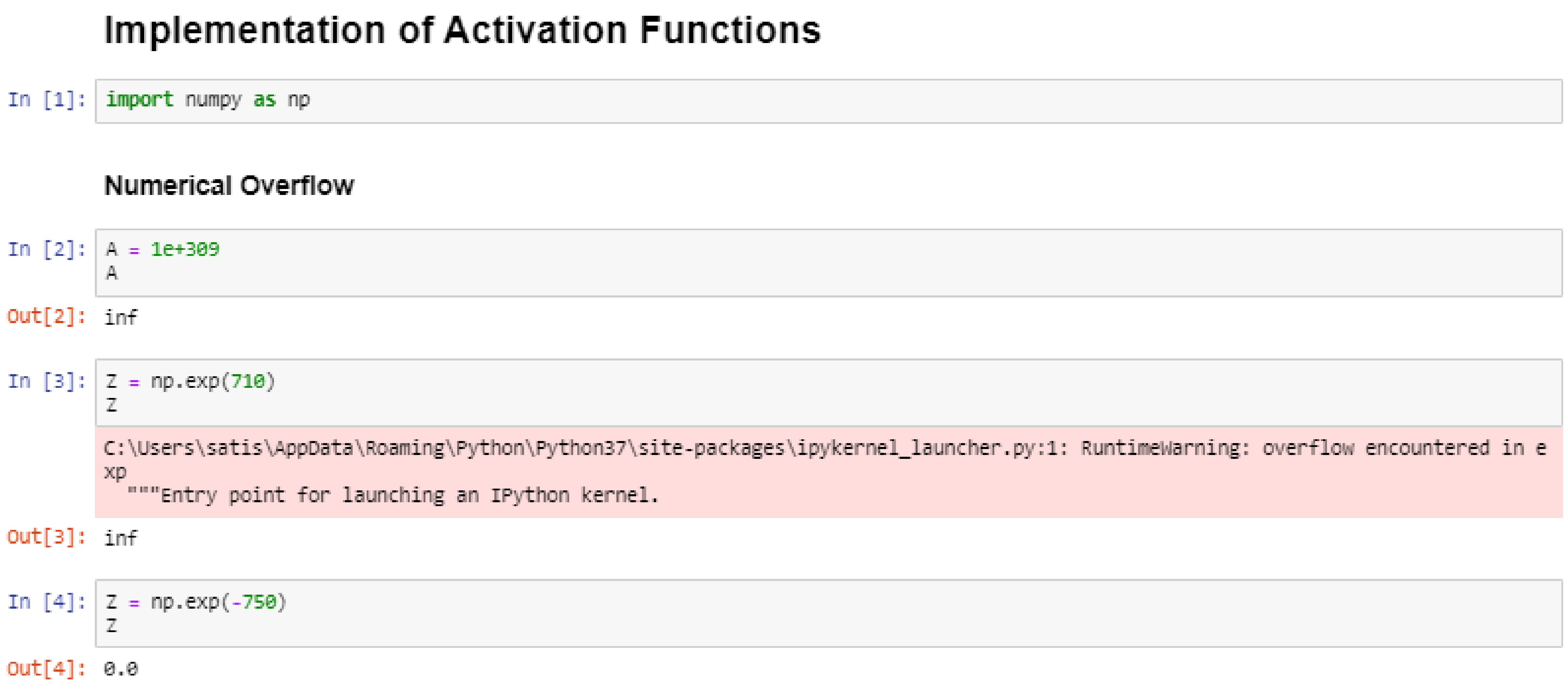
5.2 Implementation of Activation Functions

Implementation of Activation Functions

Goal of this Section:

- This section discusses how to implement activation functions to avoid numerical overflow

Numerical Overflow: Example



Implementation of Activation Functions

```
In [1]: import numpy as np
```

Numerical Overflow

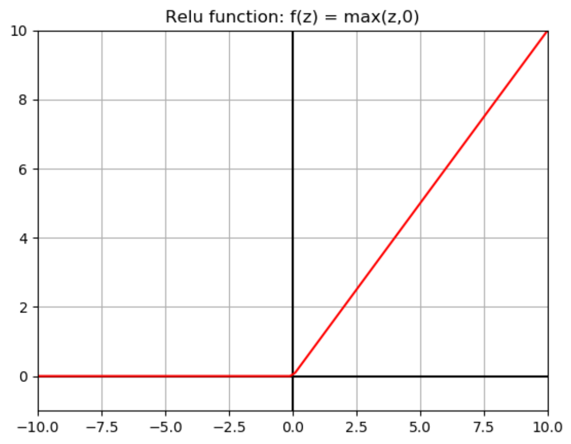
```
In [2]: A = 1e+309
        A

Out[2]: inf

In [3]: Z = np.exp(710)
        Z

C:\Users\satis\AppData\Roaming\Python\Python37\site-packages\ipykernel_launcher.py:1: RuntimeWarning: overflow encountered in e
xp
  """Entry point for launching an IPython kernel.

Out[3]: inf

In [4]: Z = np.exp(-750)
        Z

Out[4]: 0.0
```

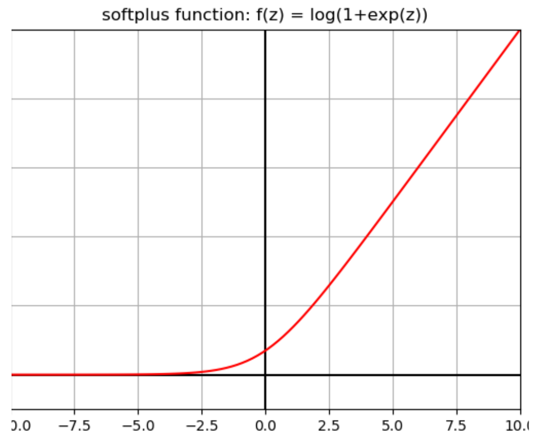
Numerical Overflow

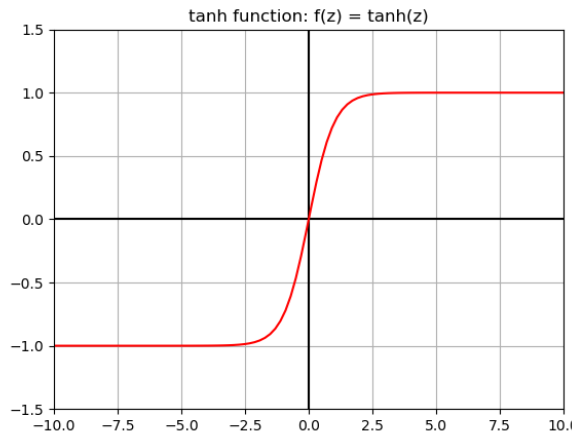
- Detailed discussion of representation of floating point (real) numbers on a computer is beyond the scope of this course. See following site: https://en.wikipedia.org/wiki/Double-precision_floating-point_format
- Can't represent number larger than roughly 10^{308}
  - Larger numbers represented as Inf
- Can get Inf overflow warning by taking numpy exponential of large number
  - Example: exp(709) is okay, but exp(710) leads to overflow warning
- To avoid these warnings, need to make minor adjustments to activation functions to avoid taking exponentials of large numbers
- Note that underflow can be an issue with some systems. Does not appear to be an issue with numpy exponential
  - Example: exp(-750) = 0

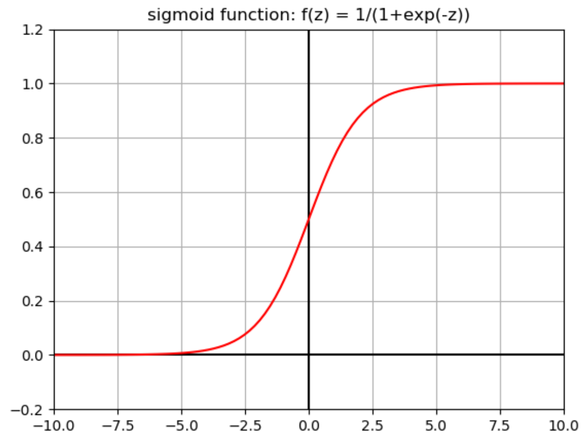
Activation Functions

- $Relu(Z) = \max(Z, 0)$
- $\text{softplus}(Z) = \ln(1 + e^{Z}) = \ln\left(e^{Z}(e^{-Z} + 1)\right) = \ln(e^{Z}) + \ln(e^{-Z} + 1) = \text{Z} + \ln(e^{-Z} + 1)$
- $\tanh(Z) = \frac{e^{Z} - e^{-Z}}{e^{Z} + e^{-Z}}$
- sigmoid(Z) = $\frac{1}{1 + e^{-Z}}$









Implementation of Activation Functions

Activation Function	Original Format	Adjusted Format	Comments

sigmoid	A = 1/(1+exp(-Z))	Z = max(Z,-50)
		A = 1/(1+exp(-Z))
			Z very negative -> overflow issue
			Make sure Z >=-50 to avoid overflow

softplus	A = log(1+exp(Z))	Z = max(Z,-50)
		A = Z + log(exp(-Z) + 1)
			Z very negative -> overflow issue
			Make sure Z >=-50 to avoid overflow

tanh	A = tanh(Z)	A = tanh(Z)	No overflow issues: Z very positive -> tanh(Z) = 1 in numpy Z very negative -> tanh(Z)=-1 in numpy
relu	A = max(Z,0)	A = max(Z,0)	Won't make adjustment

5.2 Activation Functions – Jupyter Notebook DEMO

- Open file IntroML/Examples/Chapter5/ActivationFunctions.ipynb
- Has examples of:
    - Adjustments to computation of activation functions to avoid overflow

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

5.3 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 a neural network

Coding Walkthrough: Version 2.1 To Do

File/Component	To Do
NeuralNetwork_Base	Add method to list layers and number of parameters
NeuralNetwork	Create derived NeuralNetwork class from NeuralNetwork_Base class
functions_activation	Add additional activation functions
unittest_forwardbackprop	Add test case for binary classification using a neural network
driver_neuralnetwork_binary	Add driver for binary classification using a neural network

NeuralNetwork_Base – Methods

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

Derivative Testing: Concatenation and Loading

- Derivative Testing will be performed using the test_derivative method in NeuralNetwork_Base
- For Neural Network create methods concatenate_param and load_param to be used for derivative testing:
  - concatenate_param: converts parameter matrices into single row vector
  - load_param: converts row vector to parameter matrices

- Example: Original format of parameter matrices (2 layers)

$$W^{[1]} = \begin{bmatrix} W_{00}^{[1]} & W_{01}^{[1]} \\ W_{10}^{[1]} & W_{11}^{[1]} \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} b_{00}^{[1]} \\ b_{10}^{[1]} \end{bmatrix} \quad W^{[2]} = \begin{bmatrix} W_{00}^{[2]} & W_{01}^{[2]} \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} b_{00}^{[2]} \end{bmatrix}$$

- Concatenated format:

$$RowVector = \begin{bmatrix} W_{00}^{[1]} & W_{01}^{[1]} & W_{10}^{[1]} & W_{11}^{[1]} & b_{00}^{[1]} & b_{10}^{[1]} & W_{00}^{[2]} & W_{01}^{[2]} & b_{00}^{[2]} \end{bmatrix}$$

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 $W^{[k]}$ and $b^{[k]}$ or $\nabla_{W^{[k]}}L$ and $\nabla_{b^{[k]}}L$ into a single numpy row vector Returns: row vector
load_param	flat (numpy array) order (string): “param” or “param_der”	Takes values from flat (row vector) and puts them back into $W^{[k]}$ and $b^{[k]}$ or $\nabla_{W^{[k]}}L$ and $\nabla_{b^{[k]}}L$ for k=1,…,N Returns: nothing

Activation Function

Function	Input	Description
functions_activation. activation	activation_fun (string) Z (numpy array)	Make adjustment to avoid overflow for sigmoid activation and add relu, softplus and tanh cases
functions_activation. activation_der	activation_fun (string) A (numpy array) grad_A_L (numpy array)	Add relu, softplus, and tanh cases Return: $\nabla_Z L$

5.3 Code Version 2.1 Walkthrough DEMO

- Code located at: IntroML/Code/Version2.1
- How to proceed:
  - Using information from previous videos, see if you can make updates to framework using an original Version1.3 as starting point
  - Alternatively, watch this video, which walks through updates and then use as a guide to make updates
  - The examples in the Jupyter notebooks indicate how we will use numpy functionality to convert algorithms into Python code

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

5.4 Softmax Activation Function

Softmax Activation Function

Goal of this Section:

- Review Softmax Activation Function used for multi-class classification

Softmax Activation

- softmax activation is used in final layer of neural network for multi-class classfication
- Consider matrix Z (c classes x m samples)

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

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

$$A_{ij} = \frac{e^{Z_{ij}}}{\boxed{\sum_{p=0}^{c-1} e^{Z_{pj}}}} \longleftarrow \text{Sum of entries in column j of exp(Z)}$$

- A_{ij} depends on all entries Z_{ij} in column j (not just single entry)

Softmax Activation - Example

- 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 softmax(Z) in 3 steps:

(1) Compute e^Z

$$e^Z = \begin{bmatrix} e^{Z_{00}} & e^{Z_{01}} & e^{Z_{02}} \\ e^{Z_{10}} & e^{Z_{11}} & e^{Z_{12}} \\ e^{Z_{20}} & e^{Z_{21}} & e^{Z_{22}} \\ e^{Z_{30}} & e^{Z_{31}} & e^{Z_{32}} \end{bmatrix} = \begin{bmatrix} e^{0.1} & e^{-0.1} & e^{-0.2} \\ e^{-0.2} & e^{0.2} & e^{0.3} \\ e^{-0.3} & e^{0.1} & e^{0.2} \\ e^{0.4} & e^{-0.3} & e^{-0.5} \end{bmatrix}$$

(2) Compute sum of e^Z down each column:

$$sum = [e^{0.1} + e^{-0.2} + e^{-0.3} + e^{0.4} \quad e^{-0.1} + e^{0.2} + e^{0.1} + e^{-0.3} \quad e^{-0.2} + e^{0.3} + e^{0.2} + e^{-0.5}]$$

Softmax Activation - Example

(3) Divide entries in row j of exp(Z) by column j of Sum:

Example: consider entries in middle column:

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

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

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

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

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

Same Denominator

Probabilistic Interpretation

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

- Notice that A_{ij}>0 and $\sum_{i=0}^{c-1} A_{ij} = 1$
- For column j, can interpret $\{A_{ij}\}$ for i=0,…,c-1 as probability of getting class i
- For example from last slide, can confirm that each column sums to 1

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

Softmax Activation – Avoiding Numerical Overflow

If entries of Z are large, then numerical overflow may occurring when computing the exponential

Use the following adjustment to avoid overflow

(0) Determine the maximum of Z in each column Z_m (this is a row vector)

(1) Compute e^{Z-Z_m} this involves broadcasting

(2) Compute Sum of e^{Z-Z_m} down each column

(3) Divide entries in row j of e^{Z-Z_m} by column j of Sum:

Avoiding Numerical Overflow - Example

- Consider example:

$$Z = \begin{bmatrix} 1000 & -1000 & 0 \\ -500 & 500 & 250 \end{bmatrix}$$

$$softmax1(Z) = \begin{bmatrix} \frac{e^{1000}}{e^{1000}+e^{-500}} & \frac{e^{-1000}}{e^{-1000}+e^{500}} & \frac{e^{0}}{e^{0}+e^{250}} \\ \frac{e^{-500}}{e^{1000}+e^{-500}} & \frac{e^{500}}{e^{-1000}+e^{500}} & \frac{e^{250}}{e^{0}+e^{250}} \end{bmatrix}$$

Overflow when computing e^{1000}

- Adjusted calculation: Z_m= $[1000 \quad 500 \quad 250]$

$$Z - Z_m = \begin{bmatrix} 1000 & -1000 & 0 \\ -500 & 500 & 250 \end{bmatrix} - [1000 \quad 500 \quad 250] = \begin{bmatrix} 0 & -1500 & -250 \\ -1500 & 0 & 0 \end{bmatrix}$$

$$softmax2(Z) = \begin{bmatrix} \frac{e^{0}}{e^{0}+e^{-1500}} & \frac{e^{-1500}}{e^{-1500}+e^{0}} & \frac{e^{-250}}{e^{-250}+e^{0}} \\ \frac{e^{-1500}}{e^{0}+e^{-1500}} & \frac{e^{0}}{e^{-1500}+e^{0}} & \frac{e^{0}}{e^{-250}+e^{0}} \end{bmatrix} \approx \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

No overflow issues as $Z - Z_m$ does not have any large positive numbers

- Note softmax1(Z) = softmax(2) - to show this divide numerator and denominator of each entry in softmax1(Z) in column 0 by e^{1000} in column 1 by e^{500} and column 2 by e^{250}

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

$$A_k = \frac{e^{Z_k}}{\sum_{p=0}^{c-1} e^{Z_p}} = \frac{e^{Z_k}}{e^{Z_0} + e^{Z_1} + \cdots e^{Z_{c-1}}}$$

- 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,…,A_{c-1} and not just A_i
- Using chain rule

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

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

Chain Rule using Softmax – General Case

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

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

- Denote $S_j = \sum_{k=0}^{c-1}\frac{\partial L}{\partial A_{kj}}A_{kj}$ for j=0,…,m-1. This term can be computed by performing pointwise multiplication $\nabla_A L * A$ and then summing down each column

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

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

5.4 Softmax – Jupyter Notebook DEMO

- Open file IntroML/Examples/Chapter5/Softmax.ipynb

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

5.5 One-Hot Matrix

One-Hot Matrix

Goal of this Section:

- Review One-Hot Matrix used for multi-class classification

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
- For Multi-Class Classification, represent Y_j as a One-Hot vector
- Suppose there are c classes and Y_j =p, then one-hot vector is of length c
- $Y_j^h = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \longleftarrow \text{1 in row index p}$
- Example: suppose c = 4 and $Y_j = 2$
- $Y_j^h = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$

One-Hot Matrix

- Can extend one-hot vector concept to matrices
- Assume that there are c classes and $Y = [Y_0 \quad \dots \quad Y_{m-1}]$ where each Y_j is one of 0,…,c-1, then Y^h is one-hot matrix where column j is the one-hot vector for Y_j
- Example: let $Y = [0 \quad 3 \quad 2 \quad 0]$ and assume 4 classes:

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

Inverse of One-Hot Matrix

- Given one-hot vector:
- $Y_j^h = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \xleftarrow{\text{1 in row index p}}$
- Inverse of one-hot vector is row 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 row of largest entry for each column – first index in case of ties)

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

$$onehot\ inverse(A) = [1 \quad 1 \quad 0 \quad 2]$$

5.5 One Hot Matrix – Jupyter Notebook DEMO

- Open file IntroML/Examples/Chapter5/Onehot.ipynb

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

5.6 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
  - Loss function
  - Training algorithm
  - Prediction algorithm

Motivating Example – 4 Classes

Training Data:

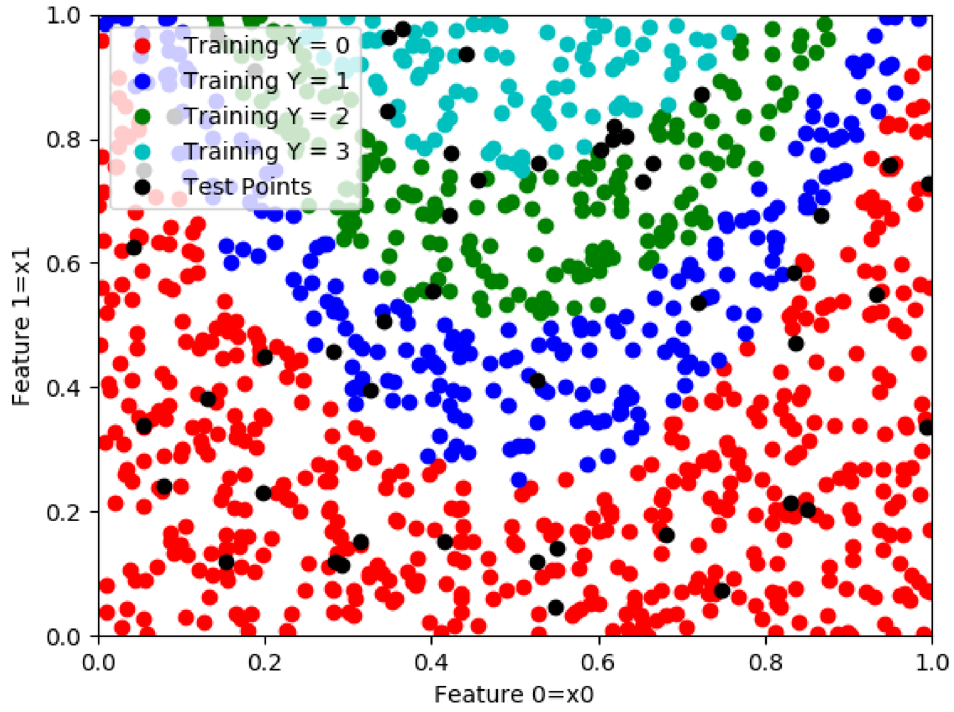
- Input Info: Points in (x0,x1) plane
- Ouptut Info: Labels (red =0, blue=1, green=2,cyan=3)

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)



Neural Network for Multi-class 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 we show the modifications to the Neural Network approach for the multi-class case:
  - Training data – output info Y is one of c classes
  - Neural Network structure in final layer – use multiple units
  - Activation function in final layer – use softmax
  - Loss function – use cross entropy function
  - Prediction Algorithm: use inverse onehot to predict class

Neural Network: General Approach

General approach has following components and phases:

1. Training Data
2. Function Structure
3. Loss Function
4. Training Phase
5. Prediction Phase

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: $\begin{bmatrix} X_{0,j} \\ X_{1,j} \\ \dots \\ X_{d-1,j} \end{bmatrix}$ and output: Y_j (one of 0,1,…,c-1)
- Define the feature matrix (dxm) and output vector (1xm):

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

Training Data – Example Points in Plane

Consider the motivating example of points in the (X_0,X_1) with 4 classes

- Suppose 4 data samples:

(1,1) label=0

(0.5,2) label=3

(2,3), label=2

(4,2) label=0

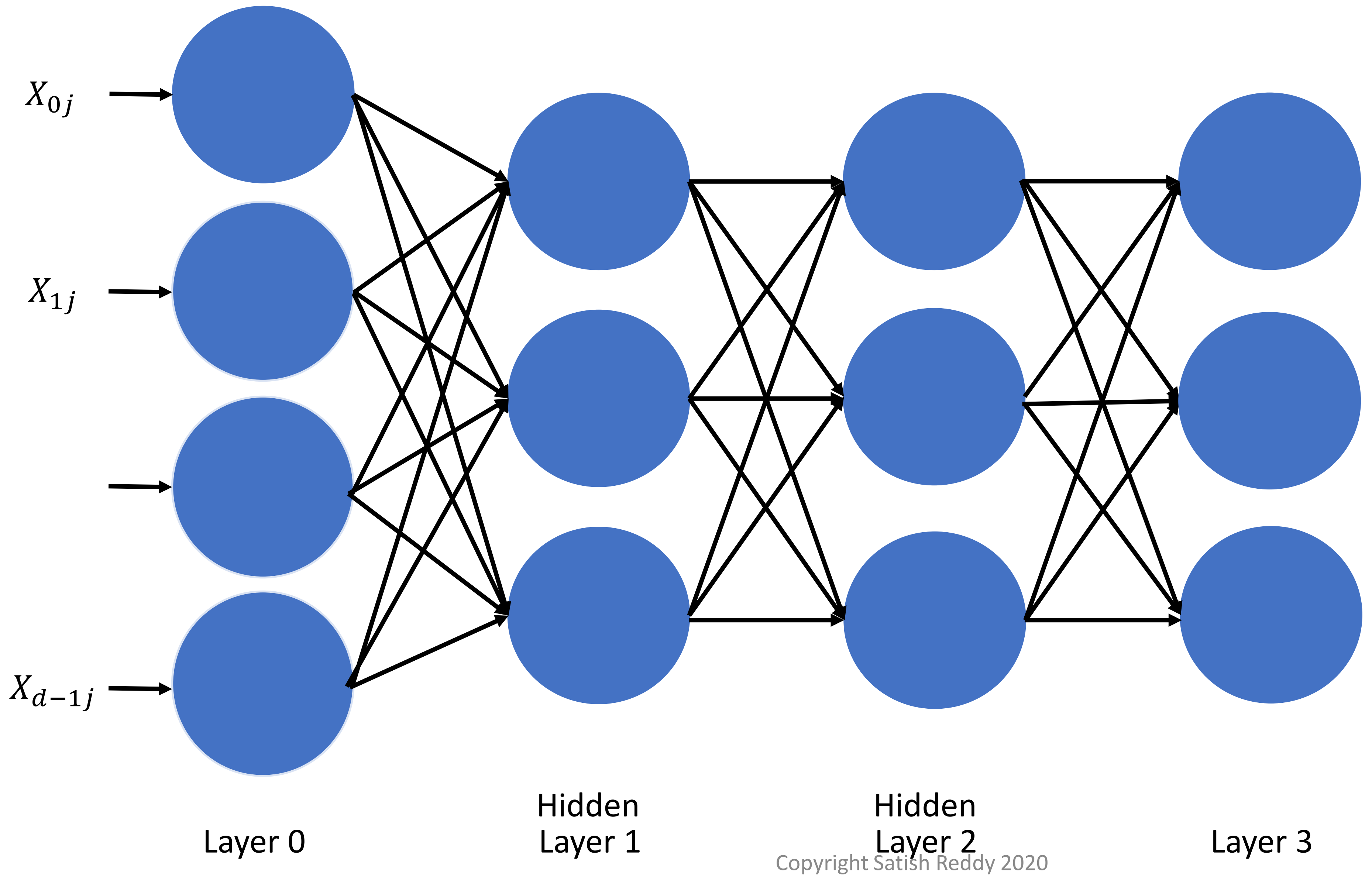
- In this case each sample has 2 features. Feature matrix and value vector are:

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

Neural Network Function Structure

- 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
- 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: $Z^{[k]} = W^{[k]}A^{[k-1]} + b^{[k]}$ $\quad$ #matrix of dimension $(n^{[k]}$x $m)$
   - Activation: $A^{[k]} = f^{[k]}(Z^{[k]})$ $\quad$ #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

Neural Network Forward Propagation - Example

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

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

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

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

- 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:

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

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

- Compute softmax of $Z^{[2]}$

$$e^{Z^{[2]}} = \begin{bmatrix} 2.3727 & 5.2642 & 6.0808 \\ 0.3451 & 0.1555 & 0.1346 \\ 2.3727 & 11.2742 & 15.0337 \end{bmatrix}$$ Sum of $e^{Z^{[2]}}$ down each column $= [5.0905 \quad 16.6939 \quad 21.2492]$

$$A^{[2]} = softmax(Z^{[2]}) = \frac{e^{Z^{[2]}}}{sum} = \begin{bmatrix} 2.373 & 5.2642 & 6.0808 \\ 0.3451 & 0.1555 & 0.1346 \\ 2.3723 & 11.2742 & 15.0337 \end{bmatrix} / [5.0905 \quad 16.6939 \quad 21.2492]$$

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

Cross Entropy Loss Function

- Loss function is

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

- Y^h is the one-hot matrix version of Y
- Sum is over all units i=0,…, $n^{[N]} - 1$ in final layer and all samples j=0,…,m-1

Cross Entropy Loss Function - Example

- From Forward Propagation Example:

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

$$A^{[2]} = \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix} \qquad \ln(A^{[2]}) = \begin{bmatrix} -0.7633 & -1.1541 & -1.2512 \\ -2.6914 & -4.6760 & -5.0615 \\ -0.7633 & -0.3925 & -0.3460 \end{bmatrix}$$

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

Compute $Y^h * \ln\left(A^{[2]}\right)$ (pointwise multiplication)

$$Y^h * \ln\left(A^{[2]}\right) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -0.7633 & -1.1541 & -1.2512 \\ -2.6914 & -4.6760 & -5.0615 \\ -0.7633 & -0.3925 & -0.3460 \end{bmatrix} = \boxed{\begin{bmatrix} -0.7633 & 0 & 0 \\ 0 & -4.6760 & 0 \\ 0 & 0 & -0.3460 \end{bmatrix}}$$

Sum entries

Loss=$-\frac{1}{m}\sum_{j=0}^{m-1}\sum_{i=0}^{n^{[N]}-1} Y_{ij}^h \ln(A_{ij}^{[N]}) = -\frac{1}{3}(-0.7633 - 4.6760 - 0.3460) = 1.9284$

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

- Cross Entropy Loss function is:

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

- Partial derivatives are given by:

$$\frac{\partial L}{\partial A_{ij}^{[N]}} = -\frac{1}{m}\frac{Y_{ij}^{h}}{A_{ij}^{[N]}}$$

- 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 $A^{[N]}$)

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 $\nabla_{A^{[N]}}L$ using one-hot matrix version of Y
2. Loop for k=N,…,1

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

Back Propagation - Example

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

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

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

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

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

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

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:

$$Y = \begin{bmatrix} 0 & 1 & 2 \end{bmatrix} \qquad Y^h = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad A^{[2]} = \begin{bmatrix} 0.4661 & 0.3153 & 0.2862 \\ 0.0678 & 0.0093 & 0.0063 \\ 0.4661 & 0.6753 & 0.7075 \end{bmatrix}$$

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

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: $S_j = \sum_{k=0}^{c-1} \frac{\partial L}{\partial A_{kj}^{[2]}} A_{kj}^{[2]}$ - Summing the above over each column: $S = [-0.3333 \quad -0.3333 \quad -0.3333]$

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

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

$$\nabla_{Z^{[2]}}L = \begin{bmatrix} -0.1780 & 0.1051 & 0.0954 \\ 0.0226 & -0.3302 & 0.0021 \\ 0.1554 & 0.2251 & -0.0975 \end{bmatrix}$$

$$\nabla_{W^{[2]}}L = \nabla_{Z^{[2]}}L \, A^{[1]T} = \begin{bmatrix} -0.1780 & 0.1051 & 0.0954 \\ 0.0226 & -0.3302 & 0.0021 \\ 0.1554 & 0.2251 & -0.0975 \end{bmatrix} \begin{bmatrix} 0 & 0.9640 \\ -0.7616 & 0.9993 \\ -0.9051 & 1 \end{bmatrix} = \begin{bmatrix} -0.1664 & 0.0289 \\ 0.2496 & -0.3061 \\ -0.0832 & 0.2772 \end{bmatrix}$$

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

$$\nabla_{b^{[2]}}L = \begin{bmatrix} 0.0225 \\ -0.3055 \\ 0.2830 \end{bmatrix}$$

Back Propagation - Example

$$\nabla_{A^{[1]}} L = W^{[2]T} \nabla_{Z^{[2]}} L = \begin{bmatrix} -1 & 1 & -2 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} -0.1780 & 0.1051 & 0.0954 \\ 0.0226 & -0.3302 & 0.0021 \\ 0.1554 & 0.2251 & -0.0975 \end{bmatrix} = \begin{bmatrix} -0.1102 & -0.8856 & 0.1017 \\ -0.0452 & 0.6605 & -0.0042 \end{bmatrix}$$

- 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 : $\begin{bmatrix} 1 & 0.4200 & 0.1807 \\ 0.0707 & 0.0013 & 0 \end{bmatrix}$

$$\nabla_{Z^{[1]}} L = \nabla_{A^{[1]}} L * \left[\frac{\partial A_{ij}^{[1]}}{\partial Z_{ij}^{[1]}}\right] = \begin{bmatrix} -0.1102 & -0.8856 & 0.1017 \\ -0.0452 & 0.6605 & -0.0042 \end{bmatrix} * \begin{bmatrix} 1 & 0.4200 & 0.1807 \\ 0.0707 & 0.0013 & 0 \end{bmatrix} = \begin{bmatrix} -0.1102 & -0.3719 & 0.0184 \\ -0.0032 & 0.0009 & 0 \end{bmatrix}$$

$$\nabla_{W^{[1]}} L = \nabla_{Z^{[1]}} L X^T = \begin{bmatrix} -0.1102 & -0.3719 & 0.0184 \\ -0.0032 & 0.0009 & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 \\ 2 & -5 \\ 4 & -8 \end{bmatrix} = \begin{bmatrix} -0.7805 & 1.9329 \\ -0.0014 & 0.0020 \end{bmatrix}$$

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

$$\nabla_{b^{[1]}} L = \begin{bmatrix} -0.4637 \\ -0.0023 \end{bmatrix}$$

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

- Assume 2 layer neural network and 3 classes
- Assume results of activation at layer 2 are given by:

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

Row index 0,2 correspond to max choose 0

Row index 2 corresponds to max

Row index 2 corresponds to max

- Probabilistic Interpretation:
  - Sum of each column is 1
  - Entry of row index i in column is probability of getting class I
  - Apply inverse onehot to $A^{[2]}$ to get prediction
  - Predicted class for each column = row index with highest probability (choose first in case of ties)

$$prediction = onehot\ inverse\left(A^{[2]}\right) = \begin{bmatrix} 0 & 2 & 2 \end{bmatrix}$$

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:

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

This is the same as for Binary classification

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	Initial guess: $W_{epoch=0}^{[k]}, b_{epoch=0}^{[k]}$ for each layer k =1, …, N Choose Learning Rate: α>0 Loop: i=1,2,… for fixed number of iterations or until Loss reduced sufficiently Perform forward propagation Perform back propagation to compute $\nabla_{W^{[k]}}L_{epoch=i-1}$ and $\nabla_{b^{[k]}}L_{epoch=i-1}$ For k=1,…,N $W_{epoch=i}^{[k]} = W_{epoch=i-1}^{[k]} - \alpha\nabla_{W^{[k]}}L_{epoch=i-1}$ $b_{epoch=i}^{[k]} = b_{epoch=i-1}^{[k]} - \alpha\nabla_{b^{[k]}}L_{epoch=\ i-1}$
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

5.6 Multiclass Classification – Jupyter Notebook DEMO

- Open file IntroML/Examples/Chapter5/NeuralNetworkMulticlass.ipynb
- Has examples of
  - Forward Propagation
  - Back Propagation
  - Prediction Algorithm
  - Accuracy Calculation

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/

5.7 Code Walkthrough Version 2.2

Coding Walkthrough: Version 2.2

Goal of this Section:

- Walkthrough update of machine learning framework to handle multi-class classification

Coding Walkthrough: Version 2.2 To Do

File/Component	To Do
NeuralNetwork_Base	Update to prediction and accuracy methods to handle “crossentropy” loss case
functions_activation	Add softmax activation function and derivative
functions_loss	Add “crossentropy” loss function and derivative
onehot	Create functions to convert vector of labels to one-hot matrix and create inverse function to convert back to a vector of labels
unittest_forwardbackprop	Add test case using cross entropy loss and softmax activation
driver_neuralnetwork_multiclass	Add driver for multiclass classification

NeuralNetwork_Base

method	Input	Description
accuracy	Y (numpy array) Y_pred (numpy array)	Add case for “crossentropy” loss Return: accuracy See IntroML/Examples/Chapter5/NeuralNetworkMulticlass.ipynb
predict	X (numpy array)	Add case for “crossentropy” loss Return: predicted labels See IntroML/Examples/Chapter5/NeuralNetworkMulticlass.ipynb

Activation and Loss Functions

Function	Input	Description
functions_activation.activation	activation_fun (string) Z (numpy array)	Add softmax case Return: f(Z) See IntroML/Examples/Chapter5/Softmax.ipynb
functions_activation.activation_der	activation_fun (string) A (numpy array) grad_A_L (numpy array)	Add softmax case Return: $\nabla_Z L$ See IntroML/Examples/Chapter5/NeuralNetworkMulticlass.ipynb
functions_loss.loss	loss_fun (string) A (numpy array) Y (numpy array)	Add crossentropy loss case Return: Loss See IntroML/Examples/Chapter5/NeuralNetworkMulticlass.ipynb
functions_loss.loss_der	loss_fun (string) A (numpy array) Y (numpy array)	Add crossentropy loss case Return: $\nabla_A L$ See IntroML/Examples/Chapter5/NeuralNetworkMulticlass.ipynb

Onehot

Function	Input	Description
onehot.onehot	Y (numpy array) nclass (integer)	Computes one-hot matrix given vector of labels Return: one-hot matrix See IntroML/Examples/Chapter5/Onehot.ipynb
onehot.onehot_ inverse	Y_onehot (numpy array)	Converts matrix into vector of labels Return: vector of labels See IntroML/Examples/Chapter5/Onehot.ipynb

5.7 Code Version 2.2 Walkthrough DEMO

- Code located at: IntroML/Code/Version2.2
- How to proceed:
  - Using information from previous videos, see if you can make updates to framework using an original Version2.1 as starting point
  - Alternatively, watch this video, which walks through updates and then use as a guide to make updates
  - The examples in the Jupyter notebooks indicate how we will use numpy functionality to convert algorithms into Python code

Course Resources at:

- https://github.com/satishchandrareddy/IntroML/