CS 615 – Deep Learning

Basic Architectures

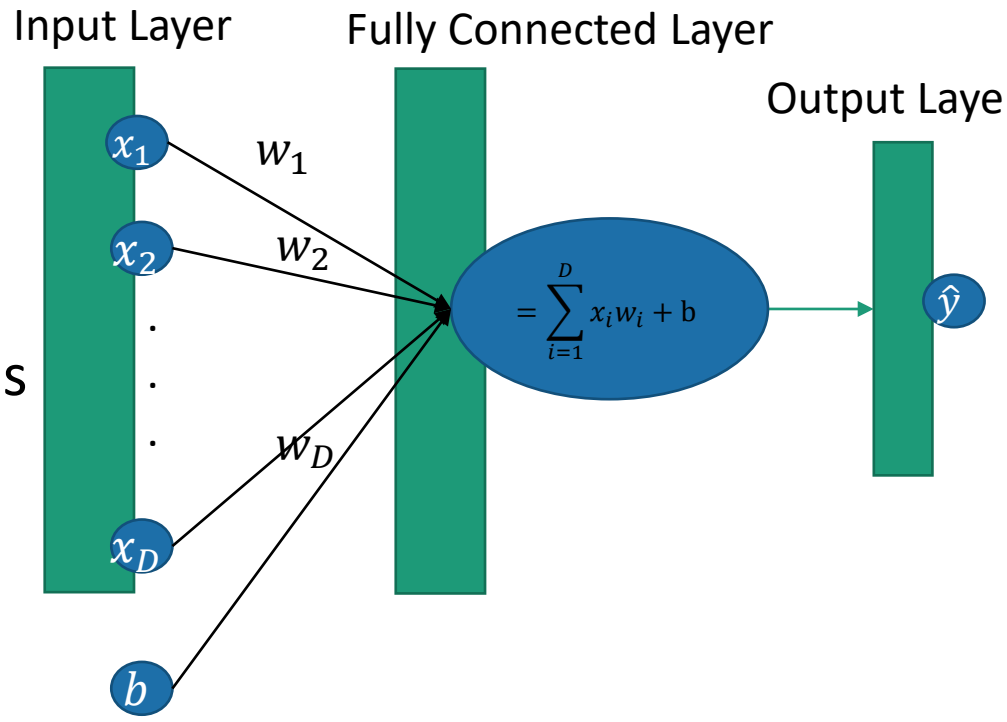
Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Example architectures:
  - Regression (linear regression)
  - Binary Classification (logistic classification)
  - Multi-Class Classification
  - ANNs/MLPs

Linear Regression Architecture

- Let's actually start building, optimizing (learning/training), and using some common architectures.
- We'll start off with one called *linear regression.*
- Linear regression is used to compute a continuous value is as the weighed some of the inputs.
- To that end, its architecture is as follows:
  - Input layer.
  - Fully connected layer with one output.
  - Squared error objective function.



Linear Regression

- Let's perform forward-backwards propagation on this architecture using batched gradient learning.
- We will assume we have an observation matrix X that has N observations, with D features per observation (so X is a $N \times D$ matrix) and that we have a column vector of target values, Y.
- Let's start with forward propagation!
  - $H^{(1)} = Input(X)$
    - $H^{(1)}$ is a $N \times D$ matrix z-scored with the training data
  - $H^{(2)} = FC\left(H^{(1)}\right) = H^{(1)}W + b$
    - $H^{(2)}$ is now a $N \times 1$ matrix
  - $\hat{Y} = H^{(2)}$
    - All done forward propagating. Of course, $\hat{Y}$ is also a $N \times 1$ matrix.

Linear Regression

- Time to backwards propagate!
- Output layer:
  - $\delta = \frac{\partial J}{\partial \hat{Y}} = 2\left(\hat{Y} - Y\right)$
    - δ is now a $N \times 1$ matrix
- Fully connected layer
  - $\frac{\partial J}{\partial W} = \delta \cdot \frac{\partial \hat{Y}}{\partial W}$
    - But we'll compute $\frac{\partial J}{\partial W}$ as $\frac{\partial J}{\partial W} = \frac{1}{N} H^{(1)^T} \delta \in \mathbb{R}^{D \times 1}$
    - Update $W = W + \eta\left(-\frac{\partial J}{\partial W}\right)$

Linear Regression

Continued from prior slide

- $\frac{\partial J}{\partial b} = \delta \cdot \frac{\partial \hat{Y}}{\partial b}$
  - Again, since we're doing a batch, we'll compute it $\frac{\partial J}{\partial b}$ as the mean of $H^{(1)}$ over the observations.
  - Update $b = b + \eta\left(-\frac{\partial J}{\partial b}\right)$
- $\delta = \delta \cdot \frac{\partial \hat{Y}}{\partial X} = \delta \cdot W^T$
  - Since δ is a $N \times 1$ matrix and W^T is a $1 \times D$ matrix, this is a $N \times D$ matrix
- Now we're at the input layer, so we're all done!

Linear Regression Example

- Let's do an example an actual quantitative!
- We are given exam data for 25 students.
- This data includes their score on exam 1, exam 2, exam 3, and the final exam.
- Our goal is to build a system that can take the exam 1, exam 2, exam 3 data a predict the final exam score.
- Since the final exam score is continuous, this is a regression problem.
- Model:

$$Final = w_1 Exam1 + w_2 Exam2 + w_3 Exam3 + b$$

Note: Final exam out of 200

EXAM1	EXAM2	EXAM3	FINAL
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177
78	83	77	159
82	86	90	177
86	82	89	175
78	83	85	175
76	83	71	149
96	93	95	192

Example

- Validation Set: The first 8 samples
- Training Set: The rest (next 17 samples)
- Settings:
  - Seed the random number generate to zero (for reproducibility and debugging)
  - Z-score the data
  - Initialize each parameter to some random number in the range of $\pm 10^{-4}$
  - Use full batch gradient descent
  - Use learning rate of $\eta = 10^{-3}$
  - Terminate when change in objective function (SE) is < 10^{-4}

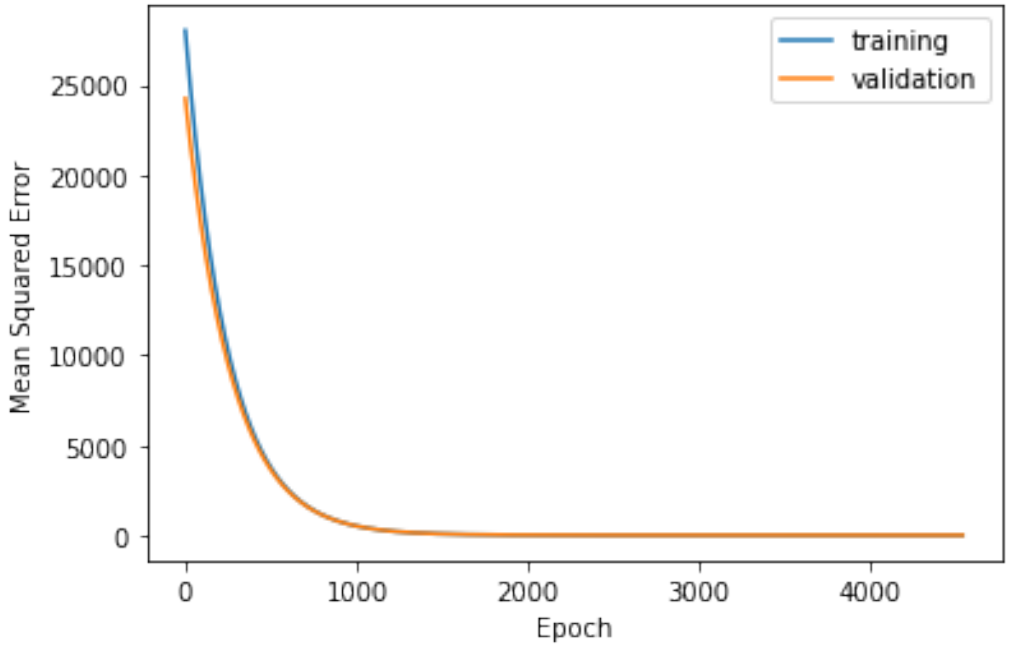
EXAM1	EXAM2	EXAM3	FINAL
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177
78	83	77	159
82	86	90	177
86	82	89	175
78	83	85	175
76	83	71	149
96	93	95	192

Results

- Model:

$$Final = 3.81Exam1 + 4.45Exam2 + 10.17Exam3 + 166.51$$

- Number of epochs: 4,537
- Final training mean squared error: 5.89
- Final validation mean squared error: 8.55



Evaluating Regression

- While we're at it, let's talk a little more about reporting error for regression.
- Using the mean of the squared error isn't that logical/natural.
- Instead, we typically think of the square root of this.
- This is known as the *root mean squared error (RMSE)* :

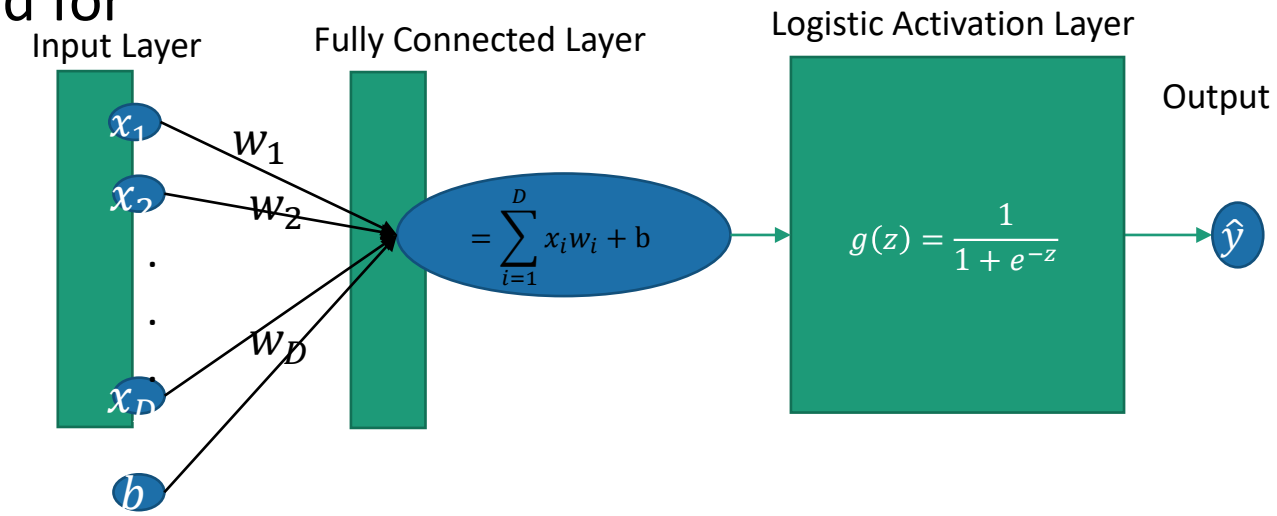
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \widehat{Y}_i\right)^2} = \sqrt{\frac{1}{N}\left(Y - \widehat{Y}\right)^T\left(Y - \widehat{Y}\right)}$$

- If we're interested in a scale-independent metric, perhaps we look at the mean absolute percent errors:

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{Y_i - \widehat{Y}_i}{Y_i}\right|$$

Logistic Regression

- The next common architecture we'll look at is *logistic regression.*
- Logistic regression is commonly used for *binary classification*.
- Its architecture consists of:
  - Input Layer
  - Fully Connected Layer (single output)
  - Logistic Activation Layer
  - Log Loss Objective Function
- Let's do a similar exercise to what we just did!



Logistic Regression

- We'll once again do batch gradient learning and our X and Y matrices will have the same shapes.
- However, now our Y matrix will have values of either zero or one (for binary classification)
- Ok, off to forward propagation!
  - $H^{(1)} = Input(X)$
    - $H^{(1)}$ is a $N \times D$ matrix z-scored with the training data
  - $H^{(2)} = FC\left(H^{(1)}\right) = H^{(1)}W + b$
    - $H^{(2)}$ is now a $N \times 1$ matrix
  - $H^{(3)} = Sigmoid\left(H^{(2)}\right)$
    - $H^{(3)}$ has the same shape as $H^{(2)}$, i.e $N \times 1$
  - $\hat{Y} = H^{(3)}$
    - All done forward propagating. Of course, $\hat{Y}$ is also a $N \times 1$ matrix.

Logistic Regression

- Time to backwards propagate!
- Output layer:
  - $\delta = \frac{\partial J}{\partial \hat{Y}} = \frac{1-Y}{1-\hat{Y}} - \frac{Y}{\hat{Y}}$
    - δ is now a $N \times 1$ matrix
- Logistic/Sigmoid activation layer
  - $\delta = \delta \circ \frac{\partial H^{(3)}}{\partial H^{(2)}}$
    - Since $\delta \in \mathbb{R}^{N\times 1}$ and $\frac{\partial H^{(3)}}{\partial H^{(2)}} \in \mathbb{R}^{N\times 1}$, when we do the Hadamard product, we get a matrix $\in \mathbb{R}^{N\times 1}$

Logistic Regression

- Fully connected layer
  - Update the weights:
    - $\frac{\partial J}{\partial W} = \frac{1}{N} {H^{(1)}}^T \cdot \delta$
      - Since ${H^{(1)}}^T$ is a $D \times N$ matrix and δ is a $N \times 1$ matrix, this is a $D \times 1$ matrix
        - NOTE: This is the same size as W!
        - NOTE: This multiplication did the sum of the gradients for us!
      - Update $W = W + \eta\left(-\frac{\partial J}{\partial W}\right)$
    - $\frac{\partial J}{\partial b} = \frac{1}{N}\sum_{n=1}^{N} \delta_n$
      - Where δ_n is the n^{th} row of the incoming gradient.
      - Update $b = b + \eta\left(-\frac{\partial J}{\partial b}\right)$
  - $\delta = \delta \cdot W^T$
    - Since δ is a $N \times 1$ matrix and W^T is a $1 \times D$ matrix, this is a $N \times D$ matrix
- Now we're at the input layer, so we're all done!

Logistic Regression Example

- Let's classifying whether a person will buy a product or not

	Y	X-Variables							
Obs. No.	Buy	Income	Is Female	Is Married	Has College	Is Pro-fessional	(Omit-ted Vari-ables)	Prev Child Mag	Prev Parent Mag
1	0	24000	1	0	1	1	. . .	0	0
2	1	75000	1	1	1	1	. . .	1	0
3	0	46000	1	1	0	0	. . .	0	0
4	1	70000	0	1	0	1	. . .	1	0
5	0	43000	1	0	0	0	. . .	0	1
6	0	24000	1	1	0	0	. . .	0	0
7	0	26000	1	1	1	0	. . .	0	0
8	0	38000	1	1	0	0	. . .	0	0
9	0	39000	1	0	1	1	. . .	0	0
10	0	49000	0	1	0	0	. . .	0	0
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
654	0	10000	1	0	0	0	. . .	0	0
655	1	75000	0	1	0	1	. . .	0	0
656	0	72000	0	0	1	0	. . .	0	0
657	0	33000	0	0	0	0	. . .	0	0
658	0	58000	0	1	1	1	. . .	0	0
659	1	49000	1	1	0	0	. . .	0	0
660	0	27000	1	1	0	0	. . .	0	0
661	0	4000	1	0	0	0	. . .	0	0
662	0	40000	1	0	1	1	. . .	0	0
663	0	75000	1	1	1	0	. . .	0	0
664	0	27000	1	0	0	0	. . .	0	0
665	0	22000	0	0	0	1	. . .	0	0
666	0	8000	1	1	0	0	. . .	0	0
667	1	75000	1	1	1	0	. . .	0	0
668	0	21000	0	1	0	0	. . .	0	0
669	0	27000	1	0	0	0	. . .	0	0
670	0	3000	1	0	0	0	. . .	0	0
671	1	75000	1	1	0	1	. . .	0	0
672	1	51000	1	1	0	1	. . .	0	0
673	0	11000	0	1	0	0	. . .	0	0

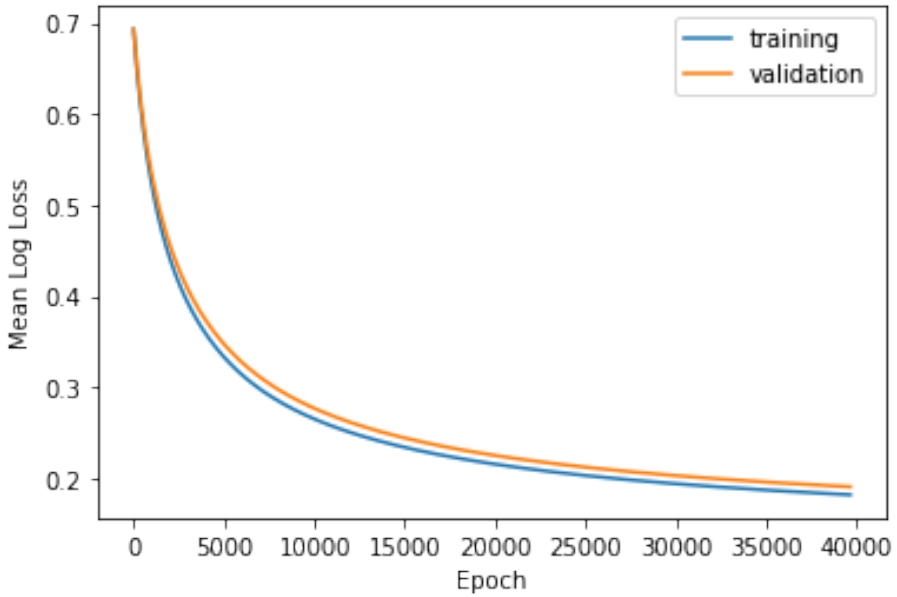
KidCreative.csv

Logistic Regression Example

- Design decisions:
  - Randomize (shuffle rows) data
  - Use 2/3 training, 1/3 validation
  - Z-score features using training data
  - Initialize parameters to random values in the range $\pm 10^{-4}$
  - Terminate when change in mean of the log loss of the training data is less than 10^{-6}
  - Full batch
  - Set the learning rate to $\eta = 10^{-3}$

Example

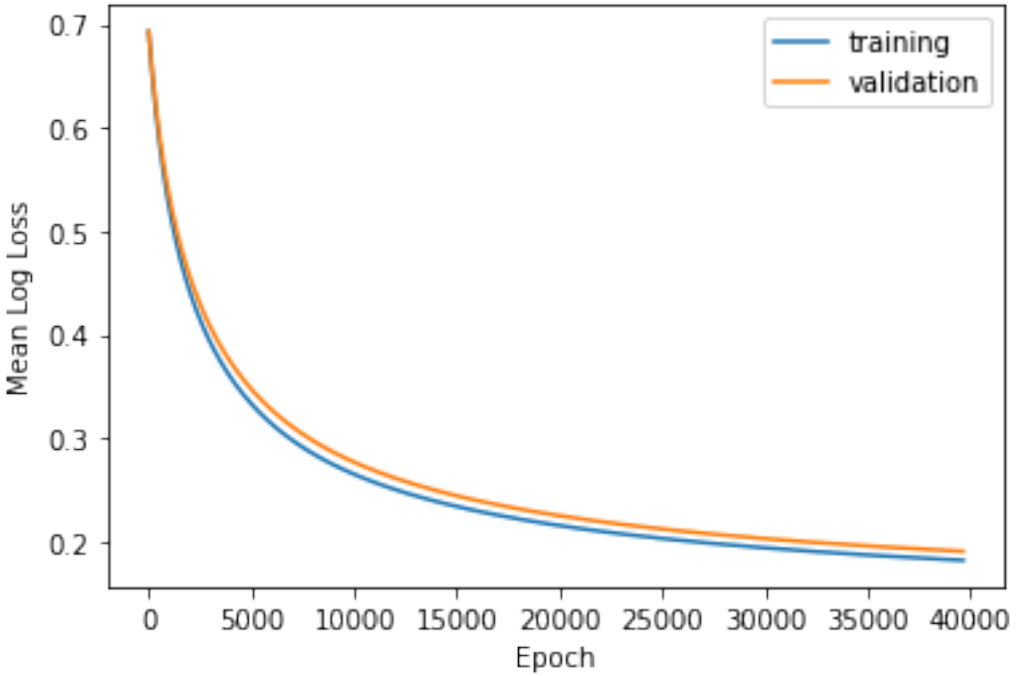
- Number of epochs: 39,639
- Final training mean log loss: 0.18
- Final validation mean log loss: 0.19



Evaluating Classifiers

- Of course, the log-loss isn't really all that natural of a thing to think about as far as quality of a system.
- And the root mean error doesn't really make sense when we have categorical/enumerated targets.
- The most natural way to evaluate a classifier is to report the percentage of observations you "got right".
- We call this the **accuracy**.

Validation Accuracy: ~92%
Training Accuracy: 93%



Multi-Class Classification

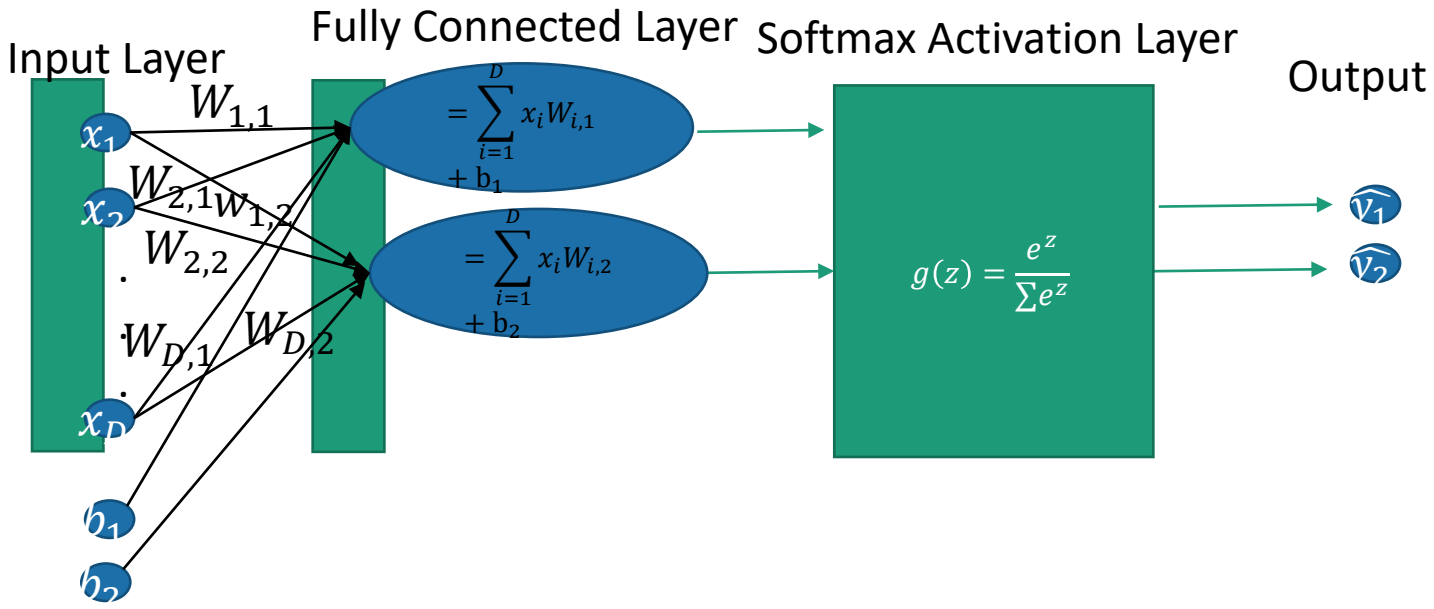
- What if we have several classes?
  - We call this multi-class classification
- We could create several binary classifiers and do a one-vs-one scheme.
- But is there a convenient way to do this in a single system?
- Yes, that's where the *cross-entropy* objective function comes it.
- Recall that the cross-entropy objective function compares two probability distributions.
- So, our predictions and our targets must be valid distributions.
- We can convert our predictions to a valid distribution using a softmax activation layer!
- To convert our targets to valid distributions, we do something called *one-hot encoding.*

One-Hot Encoding

- To convert our target class labels to a distribution we can do **one-hot encoding.**
- With one-hot encoding we convert our target class vector into a target class **matrix** with K columns, one per class.
- We want each row to be a valid distribution (sum to one, all values in $[0,1]$), so to do this we just put zeros everywhere and then a value of one in the column pertaining to the target class.
- Example:
  - What is the one-hot encoding for the target labels: $Y = [0,1,0,2]^T$?

Multi-Class Classification

- Let's wrap up with another common multi-class classification architecture:
  - Input layer
  - Fully connected layer (one output per class)
  - Softmax activation layer
  - Cross-entropy objective function
- And let's go through our drill again!



Multi-Class Classification

- We'll once again do batch gradient learning.
- Our X matrix will still be a $N \times D$ matrix, but now our Y matrix is a $N \times K$ one-hot encoded binary matrix.
- And the weights in our fully connected layer are now $W \in \mathbb{R}^{(D \times K)}$, $b \in \mathbb{R}^{(1 \times K)}$
- Very well then, off to forward propagation!
  - $H^{(1)} = Input(X)$
    - $H^{(1)}$ is a $N \times D$ matrix z-scored with the training data
  - $H^{(2)} = FC\left(H^{(1)}\right) = H^{(1)}W + b$
    - $H^{(2)}$ is now a $N \times K$ matrix
  - $H^{(3)} = Softmax\left(H^{(2)}\right)$
    - $H^{(3)}$ has the same shape as $H^{(2)}$, i.e $N \times K$
  - $\hat{Y} = H^{(3)}$
    - All done forward propagating. Of course, $\hat{Y}$ is also a $N \times K$ matrix.

Multi-Class Classification

- Time to backwards propagate!
- Output layer:
  - $\delta = \frac{\partial J}{\partial \hat{Y}} = -\frac{Y}{\hat{Y}}$
    - Due to multiple outputs, δ is now a $N \times K$ **matrix**
- Softmax activation layer
  - No weights to update
  - $\delta = \delta \otimes \frac{\partial H^{(3)}}{\partial H^{(2)}}$
    - Since the gradient of the softmax function is a tensor, we used the tensor product.
    - Since $\delta \in \mathbb{R}^{N \times K}$ and $\frac{\partial H^{(3)}}{\partial H^{(2)}} \in \mathbb{R}^{N \times K \times K}$, $\delta \otimes \frac{\partial H^{(3)}}{\partial H^{(2)}} \in \mathbb{R}^{N \times K}$

Multi-Class Classification

- Fully connected layer
  - Update the weights:
    - $\frac{\partial J}{\partial W} = \frac{1}{N} {H^{(1)}}^T \cdot \delta$
      - Since ${H^{(1)}}^T$ is a $D \times N$ matrix and δ is a $N \times K$ matrix, this is a $D \times K$ matrix
        - NOTE: This is the same size as W!
      - Update $W = W + \eta\left(-\frac{\partial J}{\partial W}\right)$
    - $\frac{\partial J}{\partial b} = \frac{1}{N}\sum_{n=1}^{N} \delta_n$
      - Since δ is a $N \times K$ matrix, this is a $1 \times K$ matrix
      - Update $b = b + \frac{\eta}{N}\left(-\frac{\partial J}{\partial b}\right)$
  - Backpropagate:
    - $\delta = \delta \cdot W^T$
      - Since δ is a $N \times K$ matrix and W^T is a $K \times D$ matrix, this is a $N \times D$ matrix
- Now we're at the input layer, so we're all done!

Multi-Class Classification

- How do we evaluate multi-class classification.
- We'll stick with *accuracy.*
- Given the output for an observation $\hat{y} \in \mathbb{R}^{1\times K}$, we choose our "best guess" for the correct class as the maximum of these values:

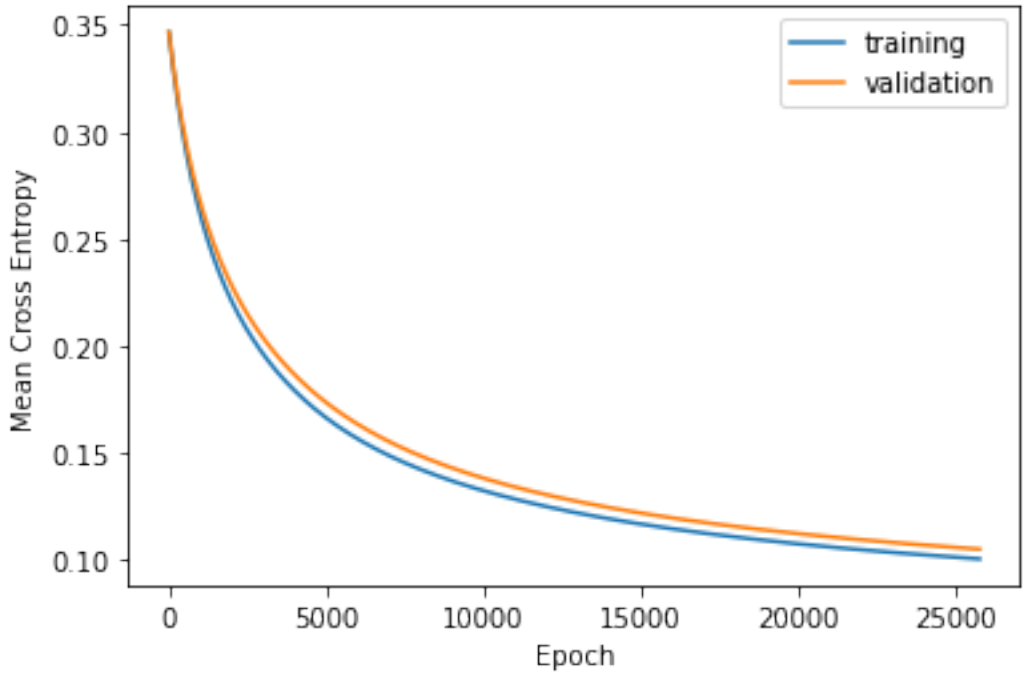
$$\hat{y}^* = argmax(\hat{y})$$

- Let $\hat{Y}^*$ be our best guess for each class.
- Then we just compute the percentage of times our guess was correct!

$$accuracy = \frac{1}{N}\sum_{n=1}^{N}\left(Y_n == \hat{Y}_n{}^*\right)$$

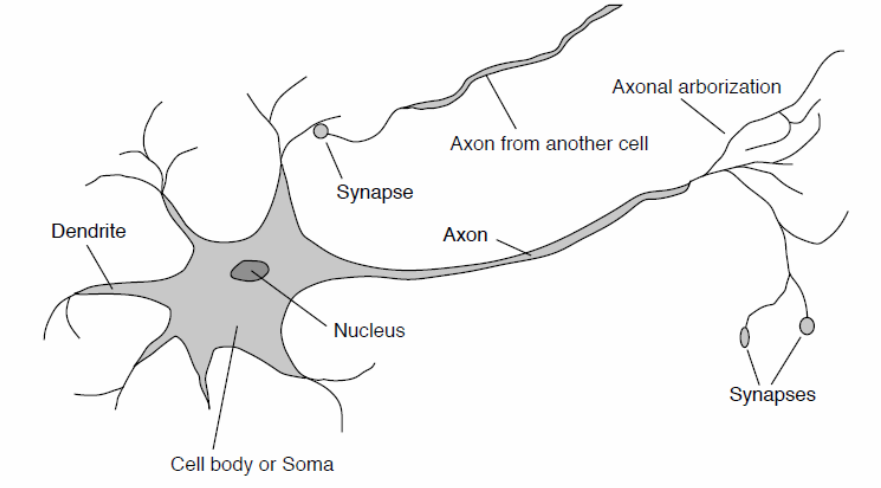
Example

- Same example as the w/ logistic regression, but one-hot encoded the target values (so Y now has two columns).
- All the same hyperparameters.
- Number of epochs: 25,738
- Final training mean cross entropy: 0.10
- Final validation mean cross entropy: 0.11
- Final training accuracy: 93.5%
- Final validation accuracy: 92%



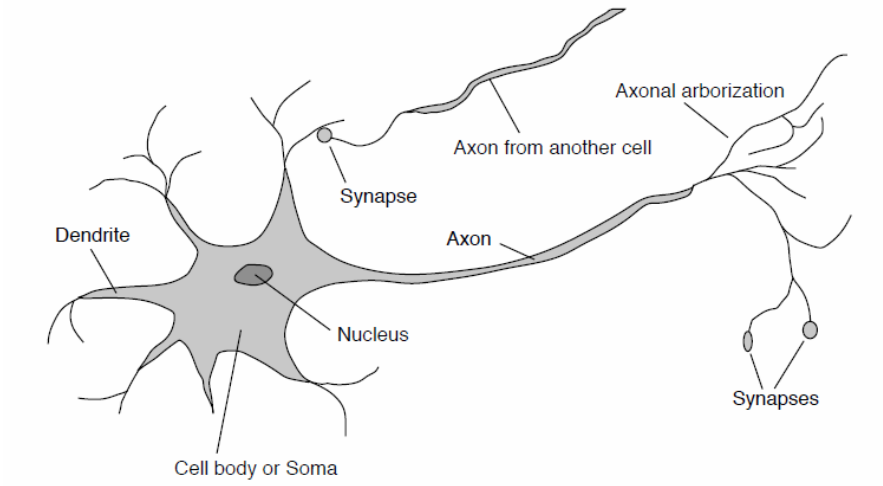
Artificial Neural Network (ANN)

- From an evolutionary standpoint, the previous architectures acted as building blocks for artificial neural networks (ANNs), which acted as the building blocks for deep networks.
- In fact, ANNs could be considered some of the first deep networks.



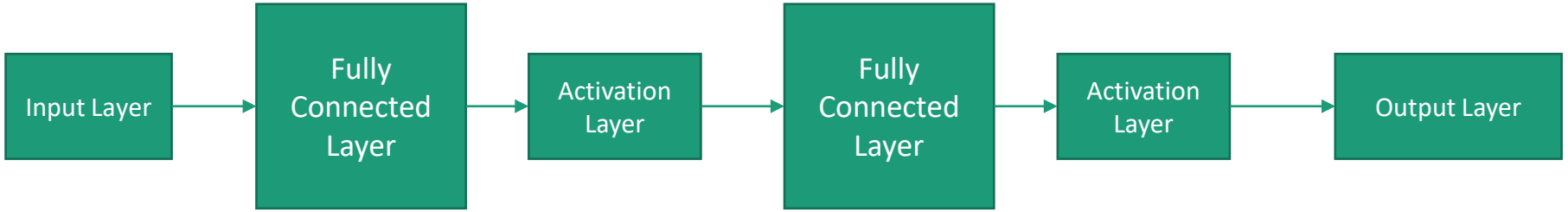
Artificial Neural Network (ANN)

- Originally, the structure of ANNs were "inspired" by the brain
  - Many interconnections between neurons (think connected layers)
  - Simple processing within neurons (think activation functions)
- But in the context of what we're doing, it's really just another assembly of our modules!



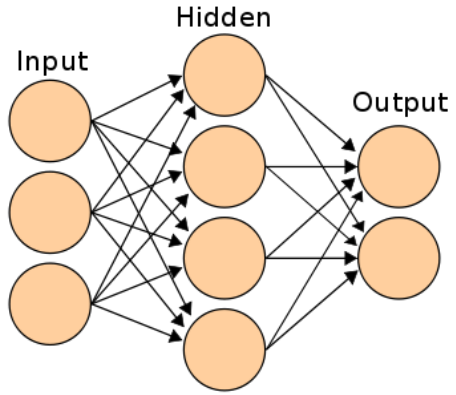
Artificial Neural Network (ANN)

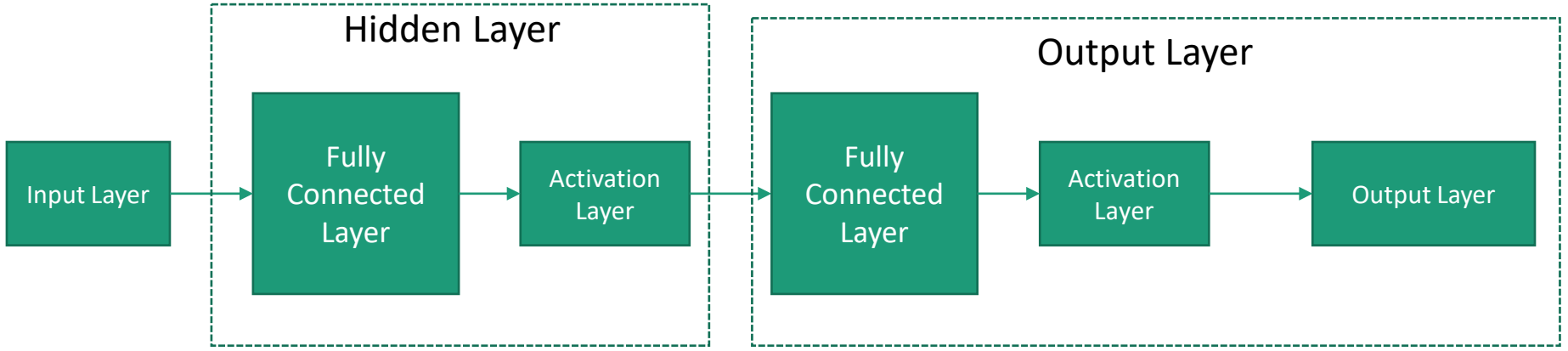
- A basic ANN has an architecture of:
  1. Input Layer
  2. Fully connected layer
  3. Activation Layer
  4. Fully Connected Layer
  5. Activation Layer
  6. Output Layer (w/ objective function)



Artificial Neural Network (ANN)

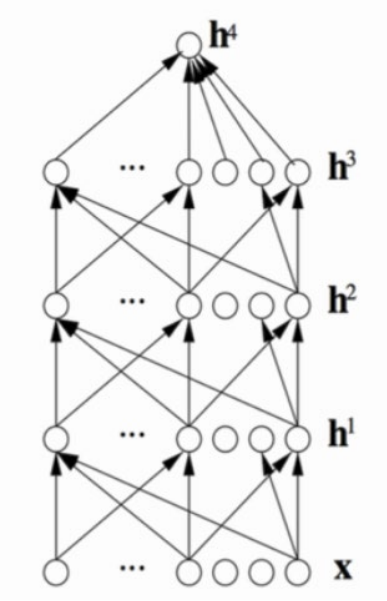
- However, most literature group a sequence of a fully connected layer followed by an activation layer.
- If this is connected to the output, then this is what they call the *output layer.*
- Otherwise, this sequence is considered a *hidden layer.*
  - This are “hidden” since they are neither the input or the output (which are the two things we have access to in a “black box” model).

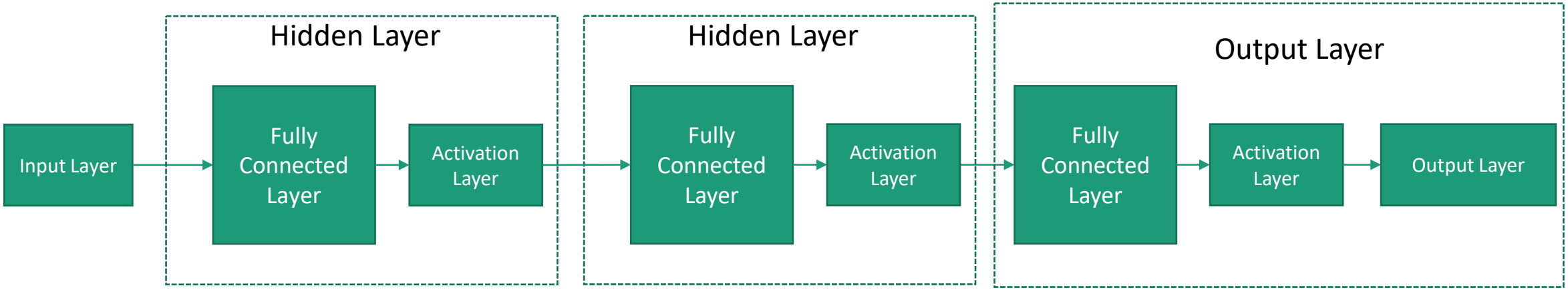




Multi-Layer Perceptron (MLP)

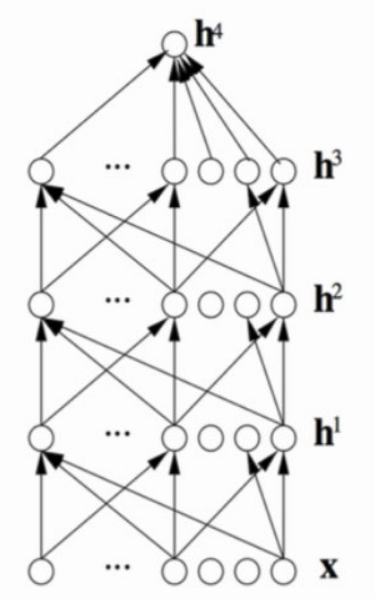
- An ANN with multiple hidden layers is often referred to as a *multi-layer perceptron (MLP).*





MLP Design Decisions

- With multiple layers, there's much responsibility!?
  - How many layers?
  - How many outputs in each non-output fully-connected layer?
  - What activation function to use at each layer?
- Unfortunately, making these decisions, for now, is largely based on empirical results.



References

- Textbook
  - Chapter 6 Deep Feedforward Networks