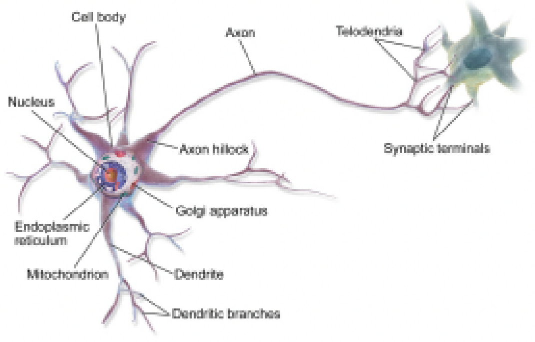
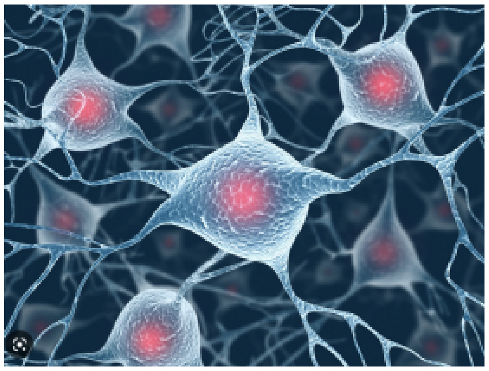
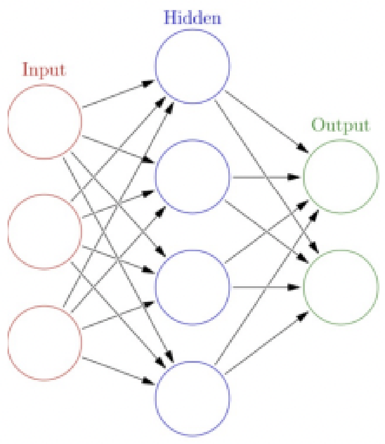
CAS CS 505
Lec 9

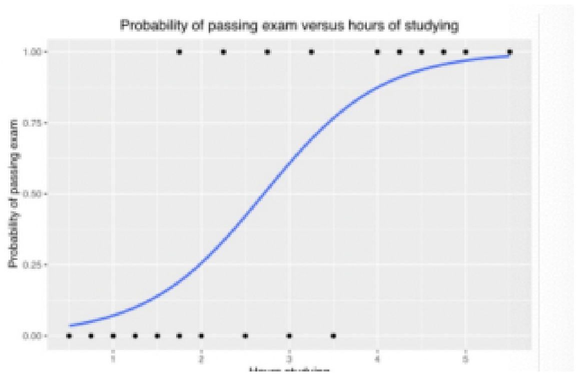
Deep Learning: Artificial Neural Networks: Design and Implementation; Workflow for Classification Tasks; Evaluating Classifiers

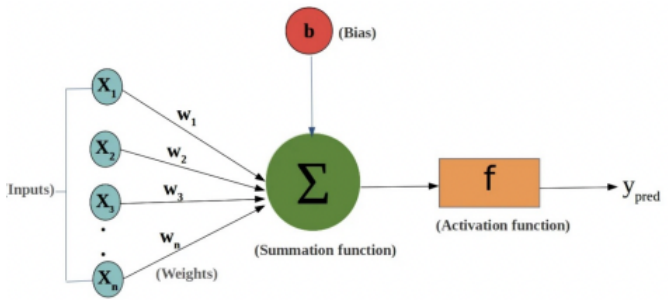
1. Introduction to Deep Learning
    a. Deep Learning refers to Supervised Learning using an Artificial Neural Network, which has the following features:
    b. It is a network/graph of small computation units called artificial neurons, loosely modeled on the neurons in our brains, which send signals to each other. The signals are floating-point numbers.
    c. The network is typically organized in layers: the first layer is the input layer, the last is the output layer, and others are called hidden layers.
    d. The input layer is array/vector of floats, and the output layer produces an array of floats. Thus, the network computes a function from vectors to vectors
    e. 

    f. 
    g. Each neuron:
        i. Is connected to each neuron in the previous layer, and each connection has a weight or parameter which determines the strength of the signal (importance of this input to the neuron);
        ii. Performs logistic regression, using the sigmoid or other non-linear activation function.

h. Gradient descent is used to learn the weights to minimize some cost function on the outputs.

i. 

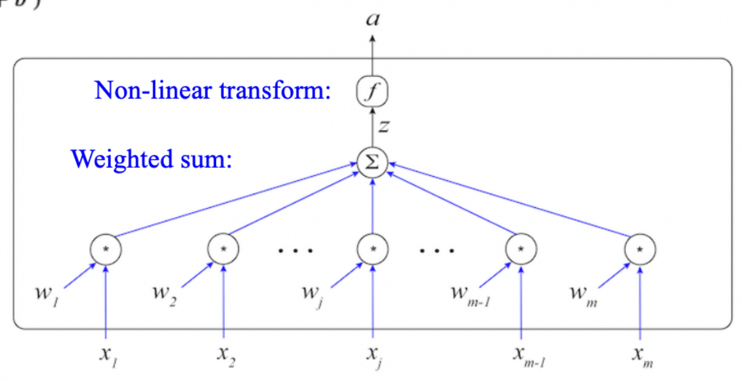
j. 

k. 

l. Features of artificial neural networks:
   i. Additional layers may perform data aggregation (e.g., convolution and pooling), dropout (forgets some of the weights), or other kinds of data manipulation (e.g., softmax = transforming the output into a probability distribution).
   ii. In a feedforward network, the network transforms an array of floats through the layers into another array of floats; in a sequence model, the inputs and outputs are sequences of vectors; and recurrent layers have cyclical connections which act as memory.
   iii. BERT, GPT, and other large networks learn to pay attention to complex patterns in the input sequence (e.g., words in a sentence).

A neuron is a higher-dimensional version of our logistic regression algorithm:

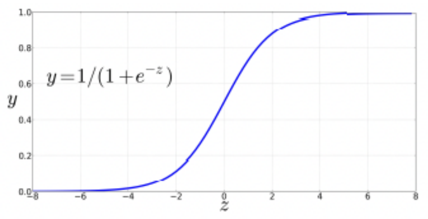
$\sigma(\, m * x + b \,)$



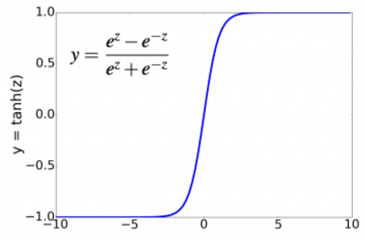
m.

n. One possible activation function f is the sigmoid from logistic regression:

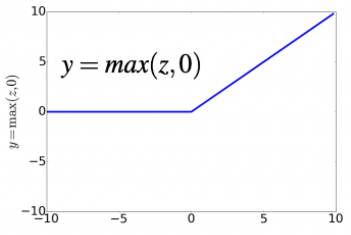
i.
$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



o. Both non-linear activation functions besides sigmoid are more often used

i.


Hyperbolic Tangent (Tanh)



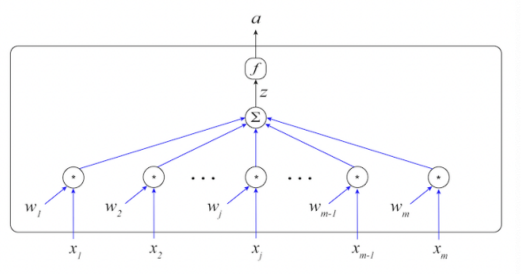
Rectified Linear Unit (Relu)

ii. Hyperbolic tangent works better than sigmoid in recurrent calls

p. A small amount of Linear Algebra can be used to compactly specify the logistic regression performed by a neuron.

q. Inputs and weights/parameters are just vectors:

i.
$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$



r. Often written as tuples

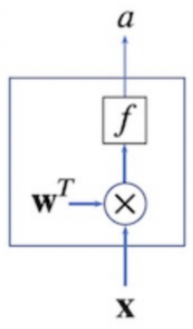
i. $\mathbf{x} = (x_1, x_2, \ldots x_m).$ $\qquad$ $\mathbf{w} = (w_1, w_2, \ldots w_m).$

s. Then the neuron performs logistic regression by performing a dot product of inputs x and weights w,

i. $$\mathbf{w} \cdot \mathbf{x} = (w_1 \cdot x_1) + \cdots + (w_k \cdot x_k) = \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

t. and then applying the activation function f:

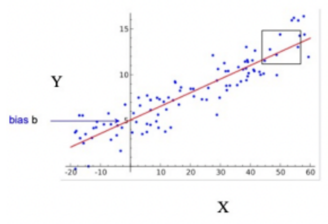
$\sigma(m * x + b)$ $\qquad$ $a = f(\mathbf{w}^T \mathbf{x}).$



i. $$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix}$$

u. A small but important detail: Each neuron has a bias term, because they are simply doing logistic regression! (each neuron will have a bias)

$\sigma(m * x + b * 1.0)$

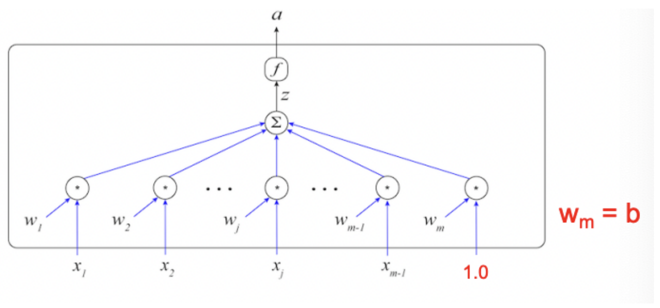
i. 

→ you always have a '1.0' to match 'b' in matrix multiplication

v. This connection has a weight (the value b) but is not connected to the inputs; it serves to scale the output, just as b does in linear regression.

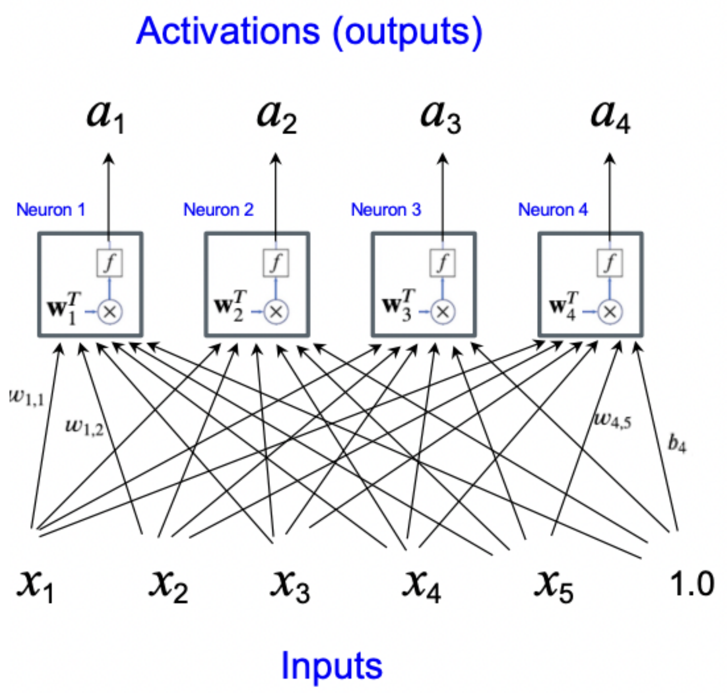
w. A convenient way to encode this is to assume that all input vectors to all neurons contain a constant 1.0 value:

i. $$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{m-1} \\ 1.0 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{m-1} \\ b \end{bmatrix}$$



x. The simplest network is simply a row of such neurons, where
    i. Each has its own weight/parameter vector, but
    ii. Shares the same input vector

y.



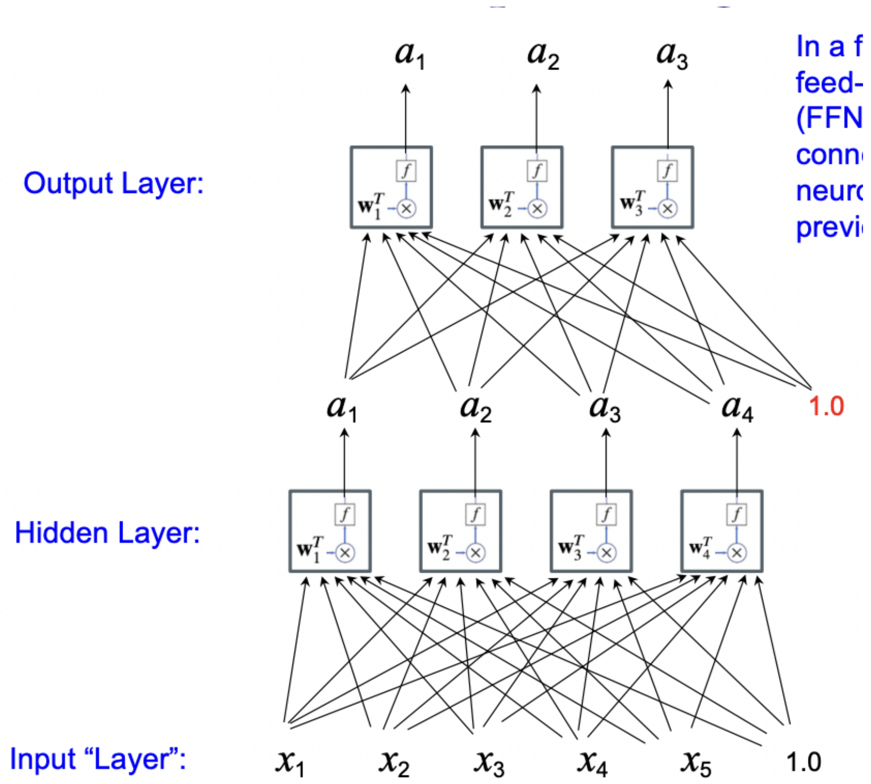
    i. They can work together to learn different thing about the inputs
    ii. Each will do logistic regression using its weights on the input and output an activation (each separately)
    iii. Vector-to-vector function
    iv. It works simultaneously (everything in one layer in principle)

z. The inputs and outputs for a layer are vectors
    i. $\mathbf{a} = (a_1, a_2, a_3, a_4)$

aa. The weights/parameters for a layer form a matrix

$$W\,\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} & b_1 \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} & b_2 \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} & b_3 \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & b_4 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_2 \\ x_4 \\ x_5 \\ 1.0 \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T\mathbf{x} \\ \mathbf{w}_2^T\mathbf{x} \\ \vdots \\ \mathbf{w}_n^T\mathbf{x} \end{bmatrix}.$$

    i. $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 1.0)$
    ii. The ws are weights that are learned

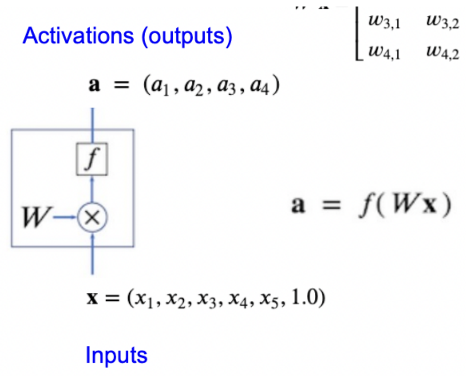


bb.
   i. In a fully connected feed-forward network (FFNN), each layer is connected to each neuron or input in the previous layer
   ii. Notice that each neuron has a bias weight
   iii. Each one of them is doing classification but they are working together
   iv. The bigger the network, there are problems introduced
      1. Cost of multiplication
      2. Searching in the highly dimensional space
      3. Vanishing weights/parameters → the probabilities are so small that they disappear/vanish (Same thing can happen here)
      4. Some numbers can get very small or big (happens on deeper networks but not usually in wider networks)

cc. Again, a little bit of linear algebra can make this a whole lot simpler:

$$W\,\mathbf{x} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & w_{1,5} & b_1 \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & w_{2,5} & b_2 \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & w_{3,5} & b_3 \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & w_{4,5} & b_4 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_2 \\ x_4 \\ x_5 \\ 1.0 \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^T\mathbf{x} \\ \mathbf{w}_2^T\mathbf{x} \\ \vdots \\ \mathbf{w}_n^T\mathbf{x} \end{bmatrix}$$

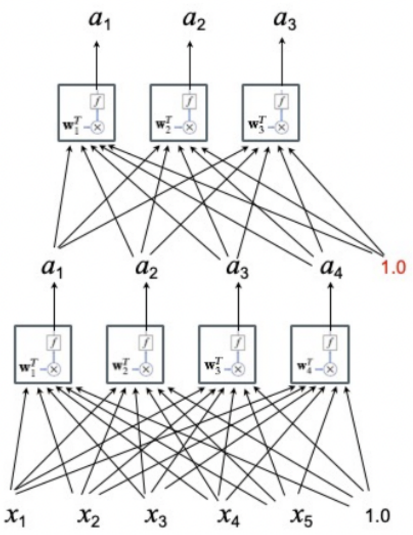
   i.



Activations (outputs)
$\mathbf{a} = (a_1, a_2, a_3, a_4)$
$w_{3,1}$ $w_{3,2}$
$w_{4,1}$ $w_{4,2}$
f
W
$\mathbf{a} = f(W\mathbf{x})$
$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 1.0)$
Inputs

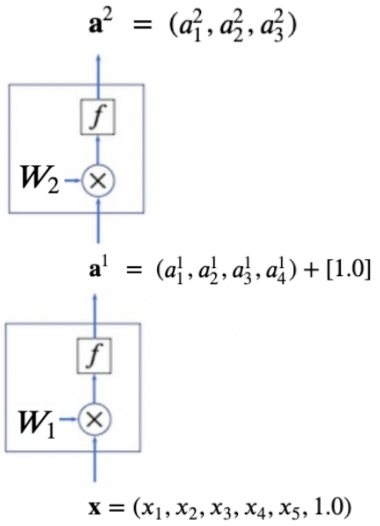
ii.

dd. Our FFNN:



a_1 a_2 a_3
a_1 a_2 a_3 a_4 1.0
x_1 x_2 x_3 x_4 x_5 1.0

i.

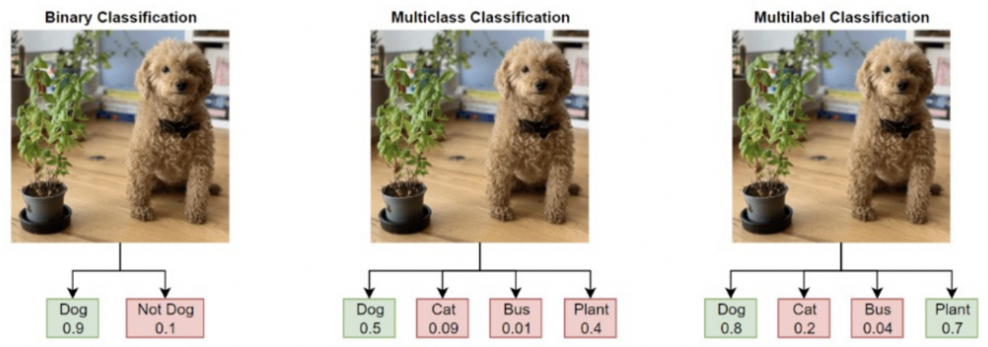


$\mathbf{a}^2 = (a_1^2, a_2^2, a_3^2)$
f
W_2
$\mathbf{a}^1 = (a_1^1, a_2^1, a_3^1, a_4^1) + [1.0]$
f
W_1
$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 1.0)$

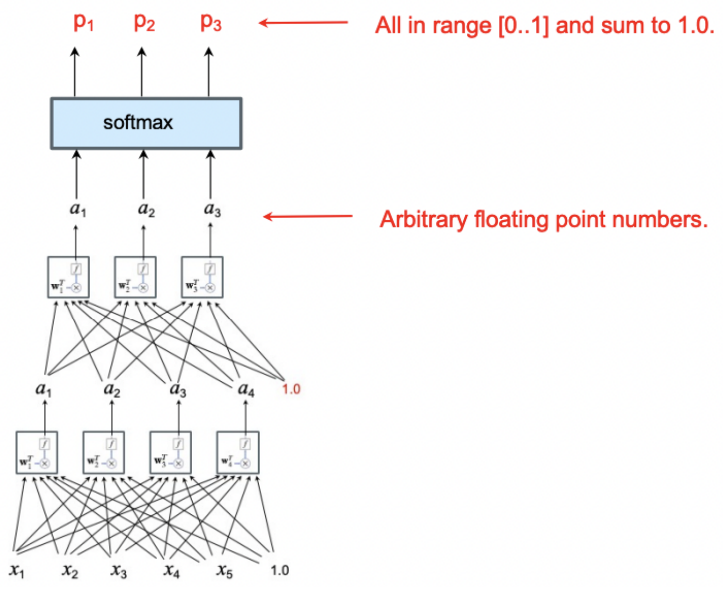
ii.

$$a^2 = f(W_2(f(W_2\mathbf{x}) + [1.0])$$

1. Do non-linear activation function every layer

2. Classification Methods: Supervised ML
    a. Input:
        i. a fixed set of classes C = {c1, c2,…, cJ}
        ii. a randomly-permuted set of labeled documents (d1,c1),....,(dn,cn) split into
            1. a training set (d1,c1),....,(dm,c)
            2. a testing set dm+1,....,dn (labels withheld)
    b. Output:
        i. A classifier γ : d → c trained the training set
        ii. The testing set with labels calculated by γ
        iii. Test results (confusion matrix, metrics, etc.)
3. Classification: Binary, Multiclass and Multilabel
    a. In Binary Classification, we have 2 labels, and we must choose one; often this is phrased as "something" or "not something" (spam or ham, misinformation or not, etc.)
    b. In Multiclass Classification, we have more than 2 labels, and our task is to assign a single label to each sample:
    c. In Multilabel Classification, we have more than 2 labels, and our task is to assign any appropriate labels (not just one):
    d. 

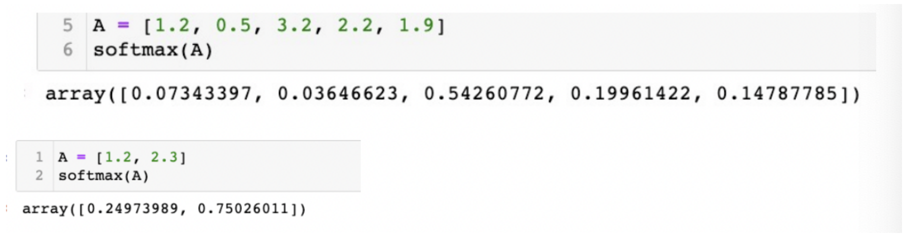
        i. Multilabel has a threshold → anything above the threshold is in the picture
4. Classification Methods: Supervised ML
    a. There are many different kinds of classifiers for labeled data
        i. Naïve Bayes
        ii. Logistic regression
        iii. Neural networks
5. Classification with Deep learning
    a. All of these types of classification are typically implemented by having the network output a probability distribution on all the labels.
    b. To convert the output values into a distribution, we used a generalization of the sigmoid called the softmax



p1 p2 p3
All in range [0..1] and sum to 1.0.
softmax
a1 a2 a3
Arbitrary floating point numbers.
a1 a2 a3 a4 1.0
x1 x2 x3 x4 x5 1.0

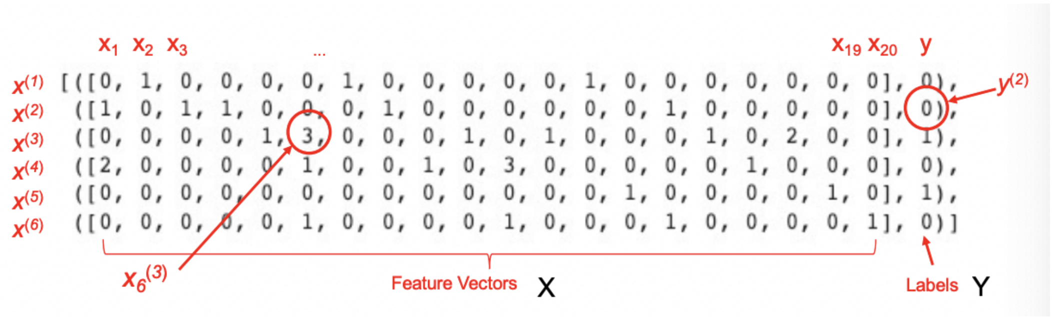
c.

   i. No guarantee that the numbers coming out are good probability distributions (might not be between 0 and 1 and add to 1 and etc.)
   ii. Use softmax
   iii. Take any sequence of floating numbers and turn it into a probability distribution (it will maintain the order but scale the value)
   iv. Softmax exaggeratest the differences

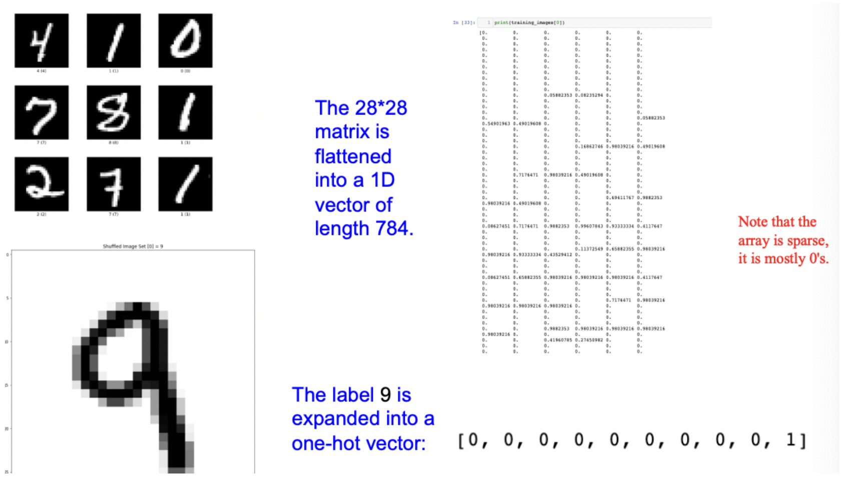
6. Introduction to Deep Learning
   a. Softmax = a generalization of sigmoid which scales k numbers into a probability distribution
      i.
$$s(z) = \frac{e^z}{e^z + 1}$$
   b. For a vector z of dimensionality k, the softmax is
      i.
$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k}\exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k}\exp(z_i)}\right]$$
      ii. Easy to manipulate, eazy to take derivatives, and etc.
   c. Example
      i.


```
A = [1.2, 0.5, 3.2, 2.2, 1.9]
softmax(A)
```
```
array([0.07343397, 0.03646623, 0.54260772, 0.19961422, 0.14787785])
```
```
A = [1.2, 2.3]
softmax(A)
```
```
array([0.24973989, 0.75026011])
```

         1. 3.2 is not that far away from 2.2 but the softmax value is quite far away (emphasizing the larger ones)

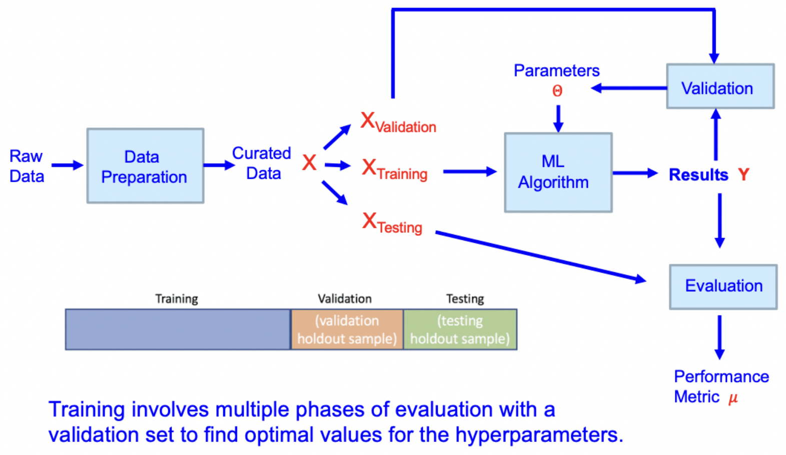
7. Data Format for Deep Learning Classifier
    a. The training set consists of a matrix of features X and a vector of labels Y.
    b. For each input observation x(i) , we have a vector of features [x1, x2, ... , xn]. Feature j for input x(i) is x_j, more precisely $x_j(i)$.
    c. 

```
[([0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0], 0),
 ([1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0], 0),
 ([0, 0, 0, 0, 1, 3, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 2, 0, 0], 1),
 ([2, 0, 0, 0, 0, 1, 0, 0, 1, 0, 3, 0, 0, 0, 0, 0, 1, 0, 0, 0], 0),
 ([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0], 1),
 ([0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1], 0)]
```

8. Multiclass Classification
    a. We will consider the MNIST database of handwritten digits: this is the “Hello World” of deep learning.
    b. The MNIST digit database consists of 70,000 28x28 BW pixel images, stored as 28*28=784 floating-point numbers in the range [0..1]; labels are integers 0..9:
    c. 

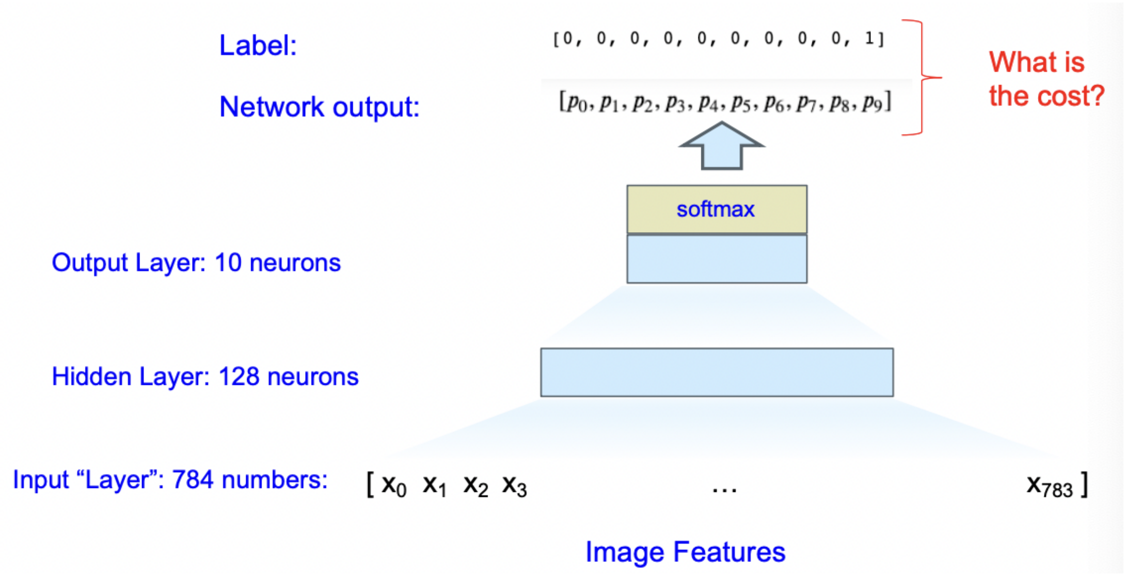
```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
```

The 28*28 matrix is flattened into a 1D vector of length 784.
Note that the array is sparse, it is mostly 0's.
The label 9 is expanded into a one-hot vector:

9. Recall: Supervised Machine Learning Workflow
    a. 

10. Cost/Loss Function for Classification

a. One more detail! What is the cost (loss) function used with the output of a classifier?


Label:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
Network output:
$[p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9]$
What is the cost?
softmax
Output Layer: 10 neurons
Hidden Layer: 128 neurons
Input "Layer": 784 numbers:
[x_0 x_1 x_2 x_3 … x_{783}]
Image Features

b.

c. We need to compare two probability distributions

Label:	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]	$\mathbf{y}$
Network output:	$[p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9]$	$\hat{\mathbf{y}}$

to measure how different they are. The function used to do this is called the Cross-Entropy Loss:

$$\hat{\mathbf{y}} = softmax(\mathbf{a}) \qquad CE(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \log(\hat{y}_i)$$

d. One great feature of this cost function is that the derivative of the softmax plus CE Loss function is absurdly simple: it is just the difference of the two distributions

i. $$\frac{\partial\, CE(\hat{y}_i, y_i)}{\partial\, z_i} = \hat{y}_i - y_i$$