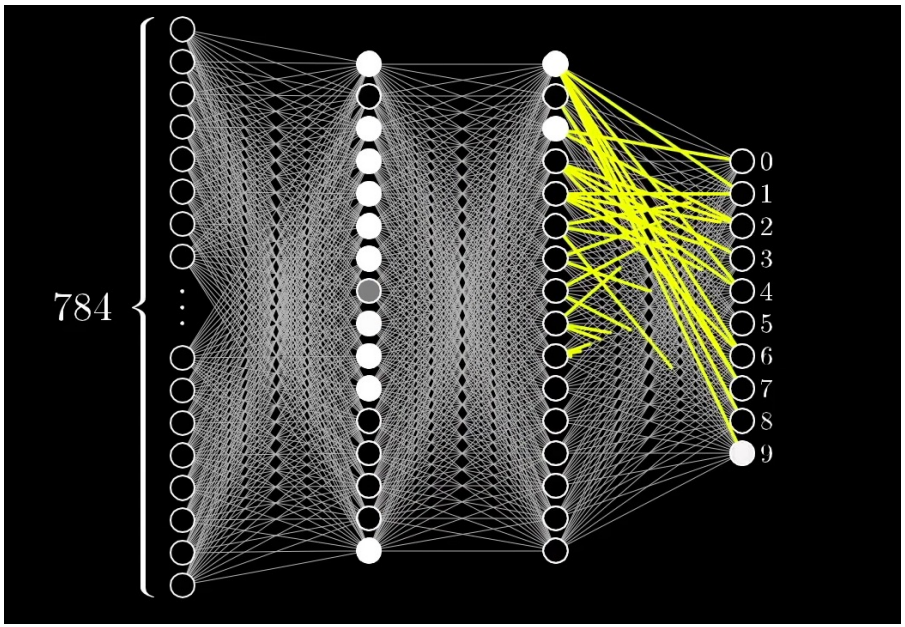
CS 505: Introduction to Natural Language Processing

Wayne Snyder
Boston University

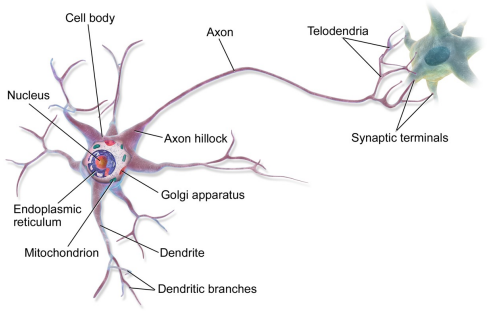
Lecture 10: **Deep Learning:** Artificial Neural Networks: Design and Implementation; Workflow for Classification Tasks; Evaluating Classifiers

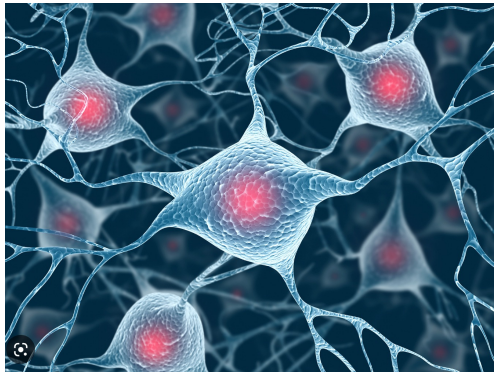


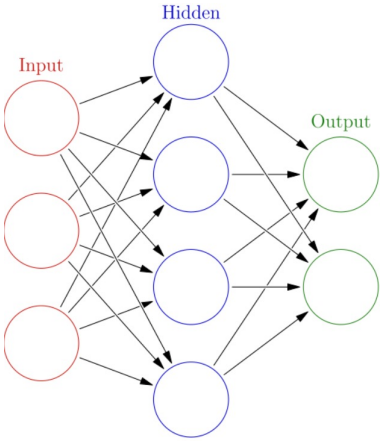
Introduction to Deep Learning

Deep Learning refers to Supervised Learning using an Artificial Neural Network, which has the following features:

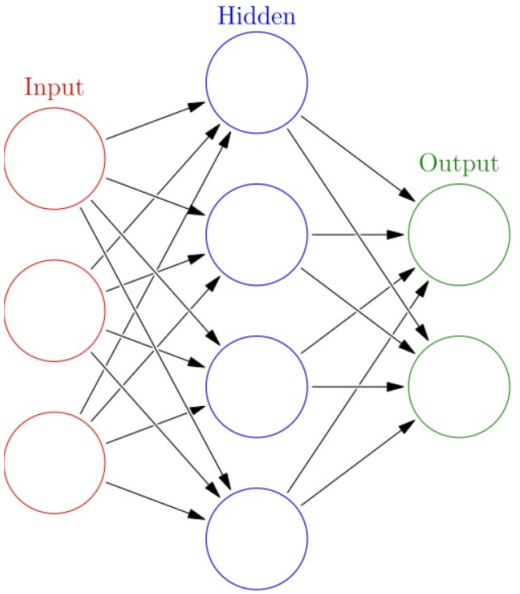
- It is a network/graph of small computation units called artificial neurons, loosely modeled on the neurons in our brains, which send signals to each other. The signals are floating-point numbers.
- The network is typically organized in layers: the first layer is the input layer, the last is the output layer, and others are called hidden layers.
- 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.





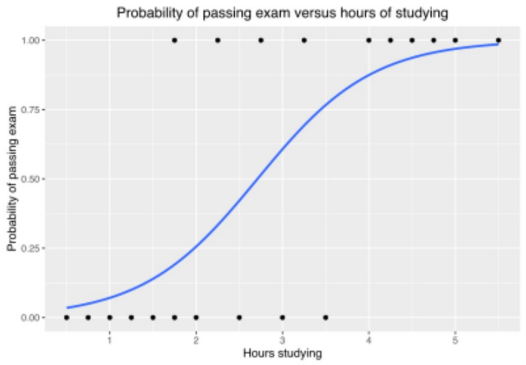


Introduction to Deep Learning

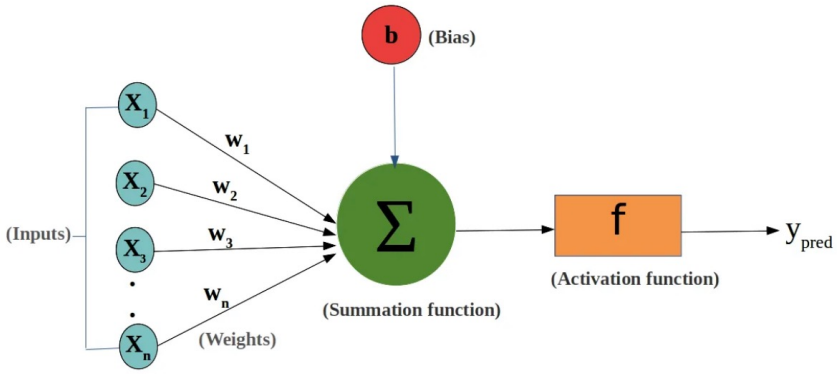


Each neuron:

- 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);
- Performs logistic regression, using the sigmoid or other non-linear activation function.



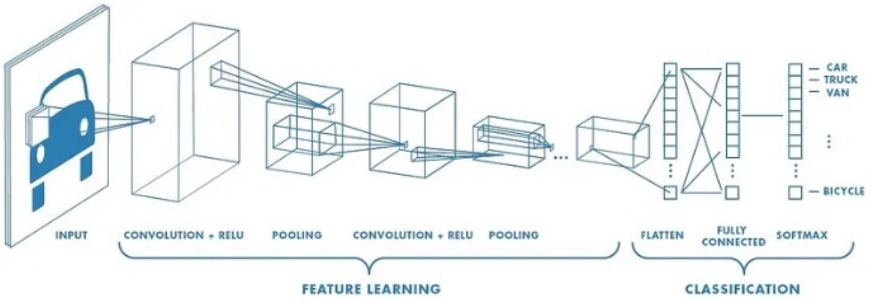
Gradient descent is used to learn the weights to minimize some cost function on the outputs.

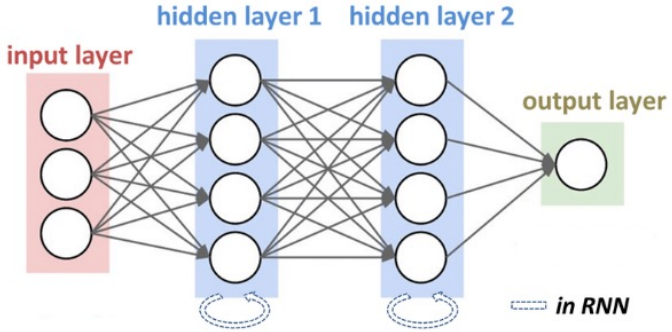


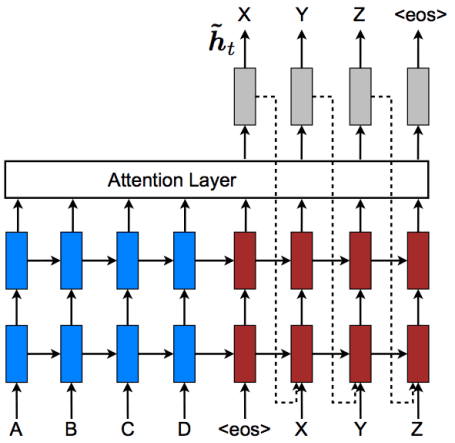
Introduction to Deep Learning

Features of artificial neural networks:

- Additional layers may perform data aggregation (e.g., convolution and pooling), dropout, or other kinds of data manipulation (e.g., softmax = transforming the output into a probability distribution).
- 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.
- BERT, GPT, and other large networks learn to pay attention to complex patterns in the input sequence (e.g., words in a sentence).



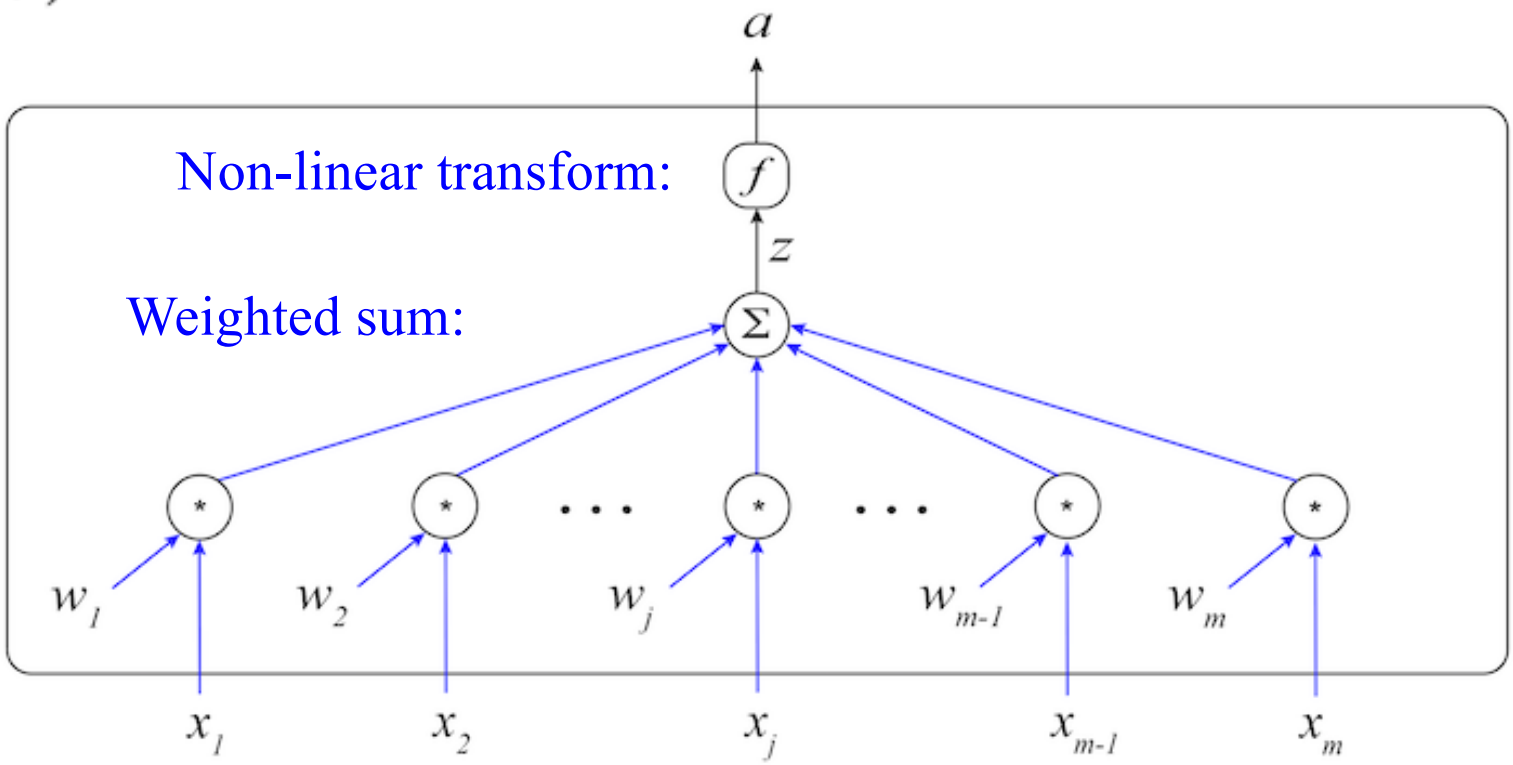




Introduction to Deep Learning

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

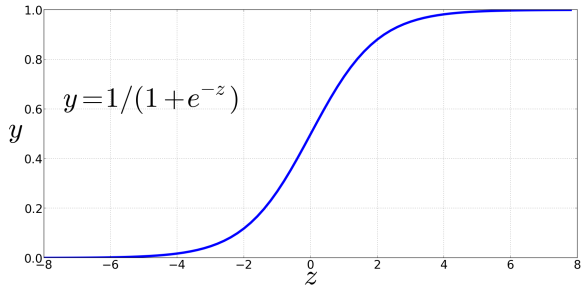
$$\sigma(m * x + b)$$



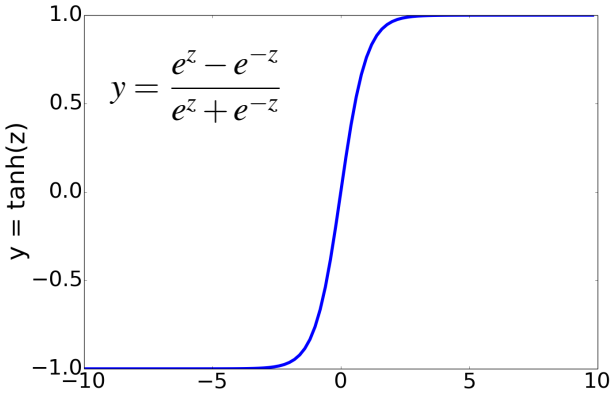
Introduction to Deep Learning

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

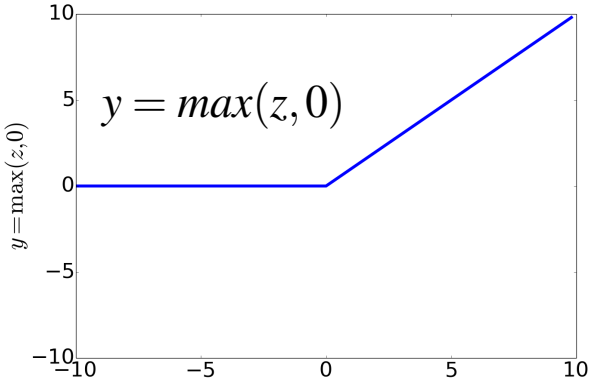
$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$



But non-linear activation functions besides sigmoid are more often used!



Hyperbolic Tangent (Tanh)



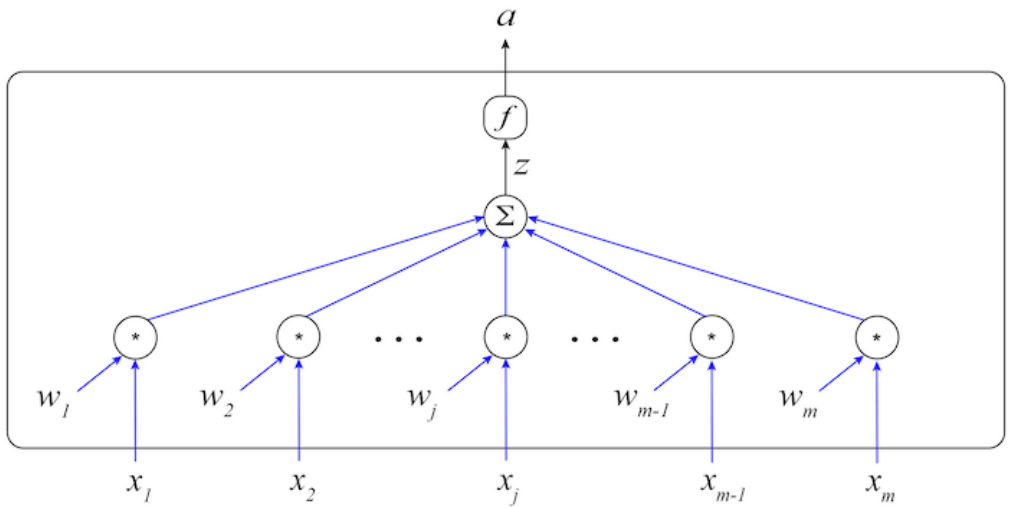
Rectified Linear Unit (Relu)

Introduction to Deep Learning

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

Inputs and weights/parameters are just vectors:

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



often written as tuples:

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

Introduction to Deep Learning

Then the neuron performs logistic regression by performing a dot product of inputs $\mathbf{x}$ and weights $\mathbf{w}$,

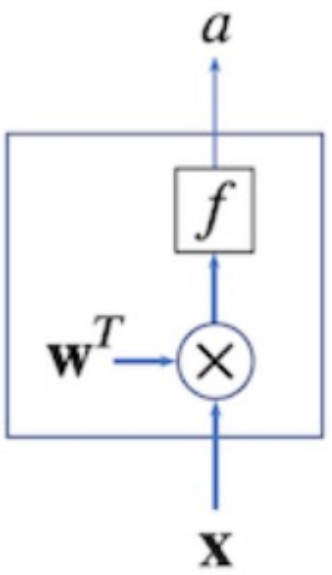
$$\mathbf{w} \cdot \mathbf{x} = (w_1 \cdot x_1) + \cdots + (w_k \cdot x_k) = \mathbf{w}^{\mathbf{T}}\mathbf{x}$$

and then applying the activation function f:

$\sigma(m * x + b)$

$$a = f(\mathbf{w}^T\mathbf{x}).$$

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



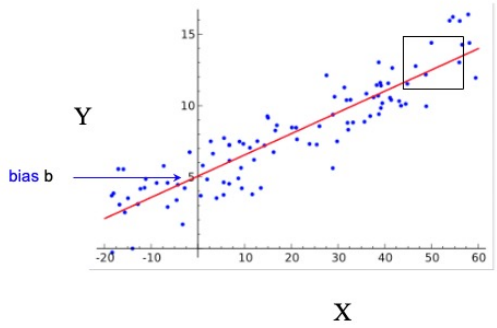
Introduction to Deep Learning

A small but important detail: Each neuron has a bias term, because they are simply doing logistic regression!

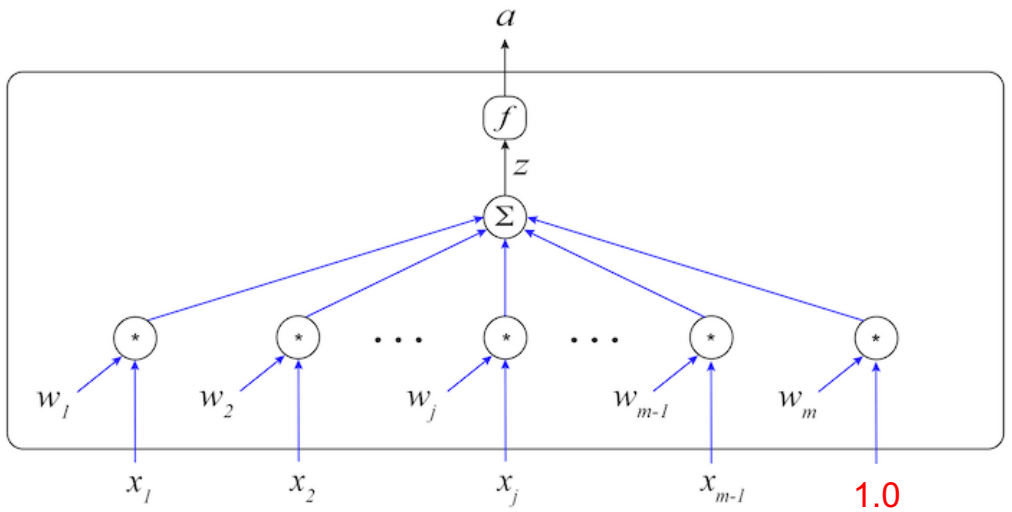
This connection has a weight (the value b) but is not connected to the inputs; it serves to scale the output, just as b does in linear regression.

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

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



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



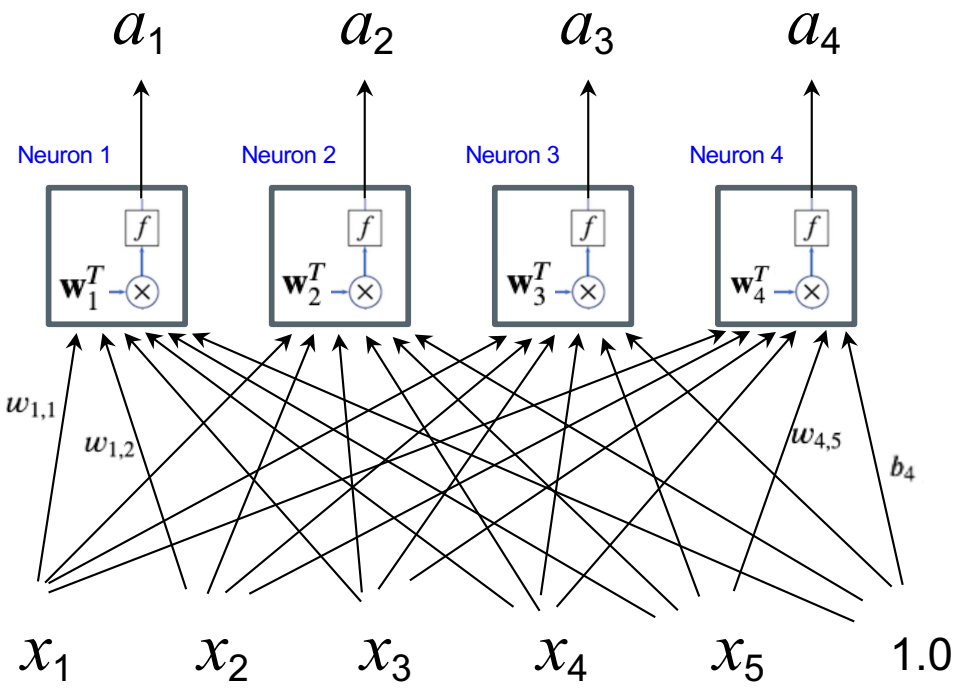
$w_m = b$

Introduction to Deep Learning

The simplest network is simply a row of such neurons, where

- Each has its own weight/parameter vector, but
- Shares the same input vector

Activations (outputs)



Inputs

The inputs and outputs for a layer are vectors:

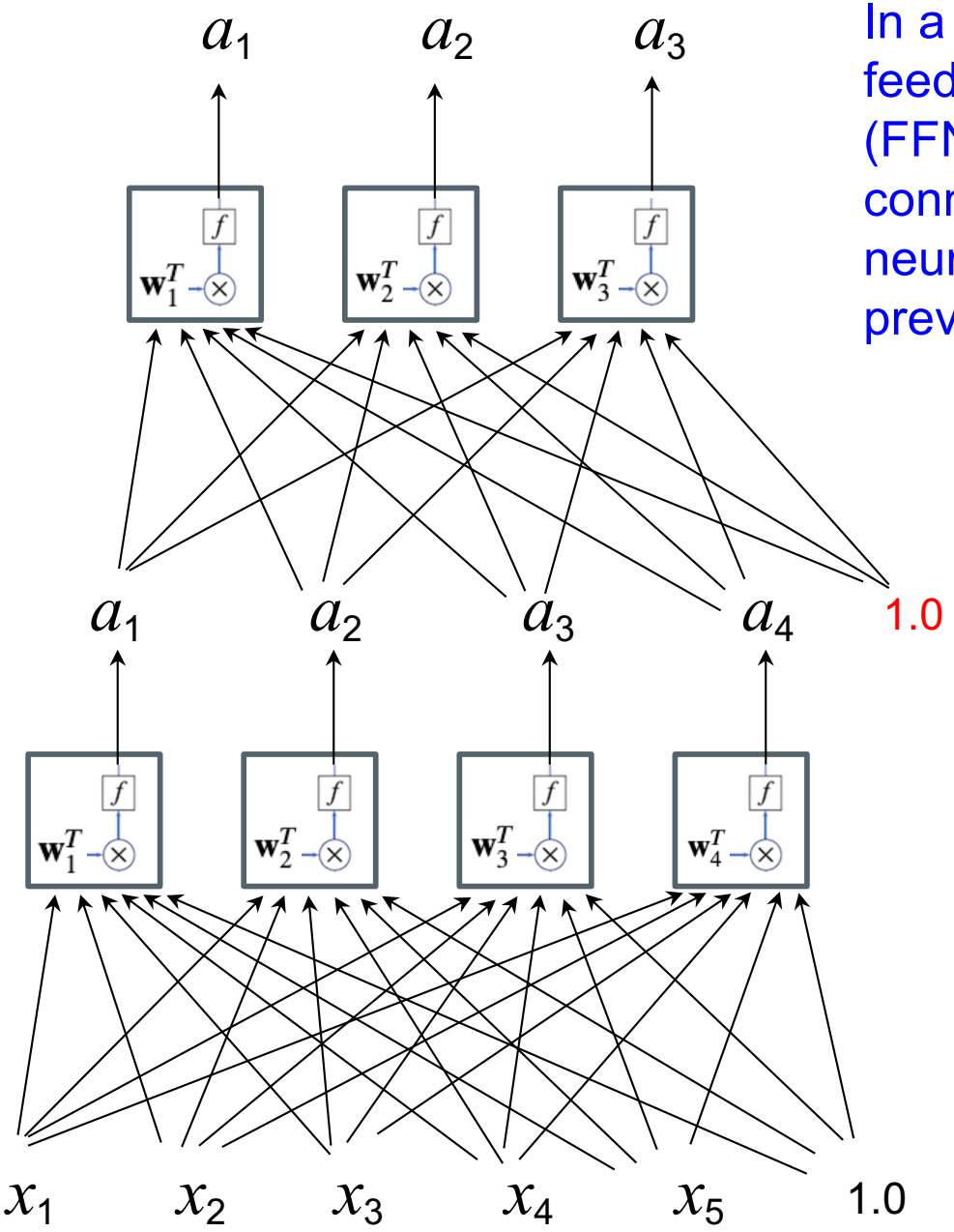
$$\mathbf{a} = (a_1, a_2, a_3, a_4)$$

The weights/parameters for a layer form a matrix:

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

$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 1.0)$$

Introduction to Deep Learning



In a fully-connected feed-forward network (FFNN), each layer is connected to each neuron or input in the previous layer.

Notice that each neuron has a bias weight!

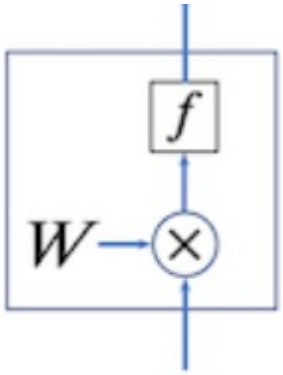
Introduction to Deep Learning

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

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

Activations (outputs)

$$\mathbf{a} = (a_1, a_2, a_3, a_4)$$



$$\mathbf{a} = f(W\mathbf{x})$$

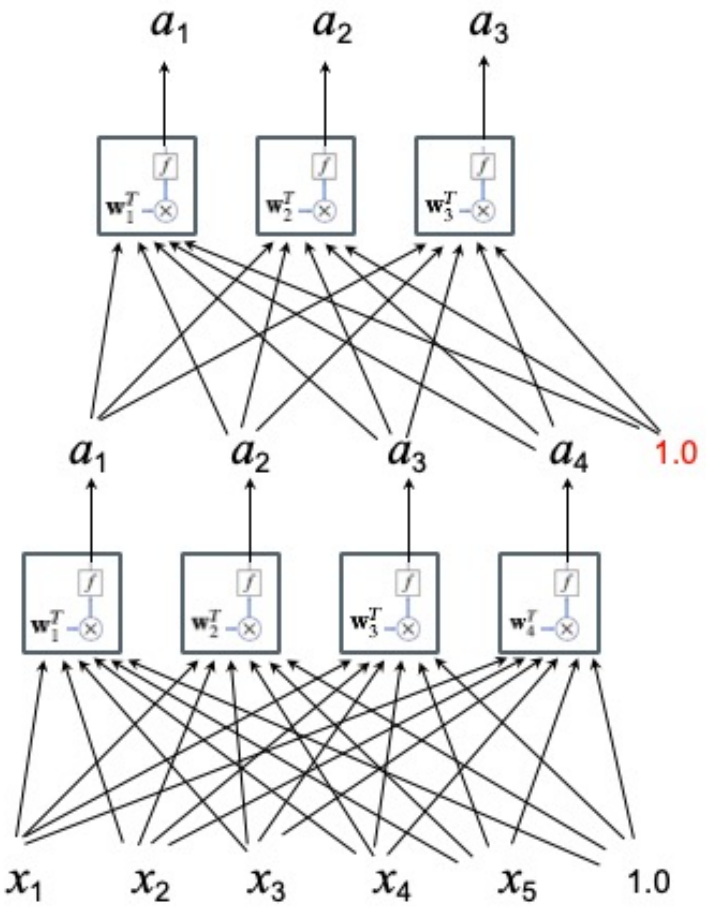
$$f\begin{pmatrix} \mathbf{w}_1^T\mathbf{x} \\ \mathbf{w}_2^T\mathbf{x} \\ \mathbf{w}_3^T\mathbf{x} \\ \mathbf{w}_4^T\mathbf{x} \end{pmatrix} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \mathbf{a}_4 \end{bmatrix}$$

$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 1.0)$$

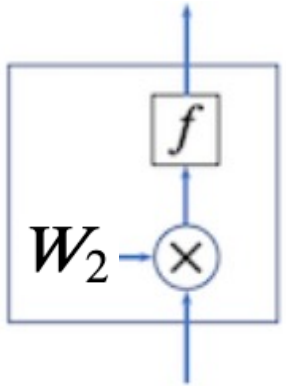
Inputs

Introduction to Deep Learning

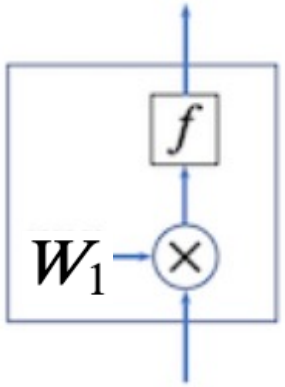
Our FFNN:



$$\mathbf{a}^2 = (a_1^2, a_2^2, a_3^2)$$



$$\mathbf{a}^1 = (a_1^1, a_2^1, a_3^1, a_4^1) + [1.0]$$



$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, 1.0)$$

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

Classification Methods: Supervised ML

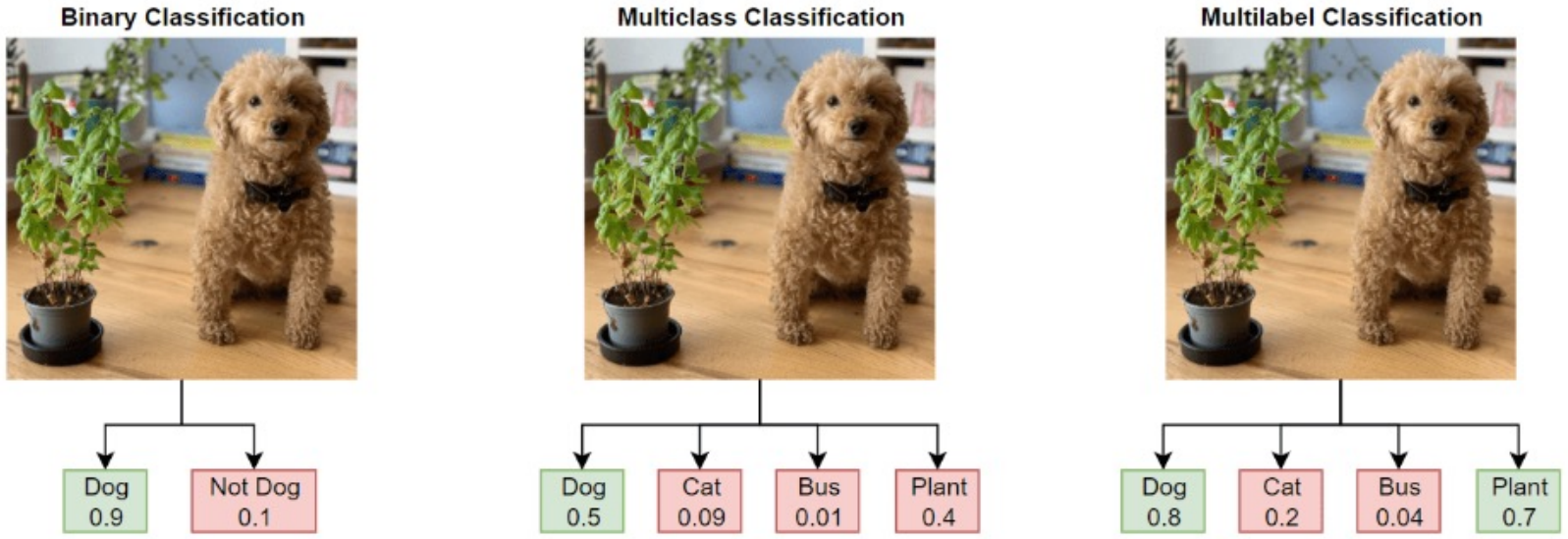
- ***Input:***
  - a fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
  - a randomly-permuted set of labeled documents $(d_1,c_1),\ldots,(d_n,c_n)$ split into
    - a training set $(d_1,c_1),\ldots,(d_m,c)$
    - a testing set $d_{m+1},\ldots,d_n$ (labels withheld)
- ***Output:***
  - A classifier $\gamma : d \rightarrow c$ trained the training set
  - The testing set with labels calculated by γ
  - Test results (confusion matrix, metrics, etc.)

Classification: Binary, Multiclass, and Multilabel

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.)

In Multiclass Classification, we have more than 2 labels, and our task is to assign a single label to each sample:

In Multilabel Classification, we have more than 2 labels, and our task is to assign any appropriate labels (not just one):



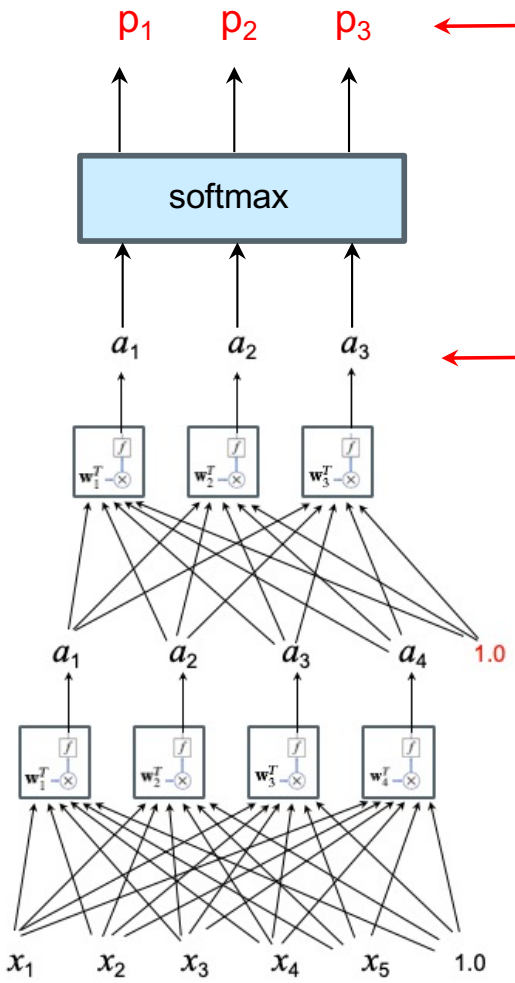
Classification Methods: Supervised ML

- **There are many different kinds of classifiers for labeled data**
  - Naïve Bayes
  - Logistic regression
  - Neural networks

Classification with Deep Learning

All of these types of classification are typically implemented by having the network output a probability distribution on all the labels.

To convert the output values into a distribution, we used a generalization of the sigmoid called the softmax:



All in range [0..1] and sum to 1.0.

Arbitrary floating point numbers.

Introduction to Deep Learning

Softmax = a generalization of sigmoid which scales k numbers into a probability distribution.

- For a vector *z* of dimensionality *k*, the softmax is:

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

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

- Examples:

```
5 A = [1.2, 0.5, 3.2, 2.2, 1.9]
6 softmax(A)
```

```
array([0.07343397, 0.03646623, 0.54260772, 0.19961422, 0.14787785])
```

```
1 A = [1.2, 2.3]
2 softmax(A)
```

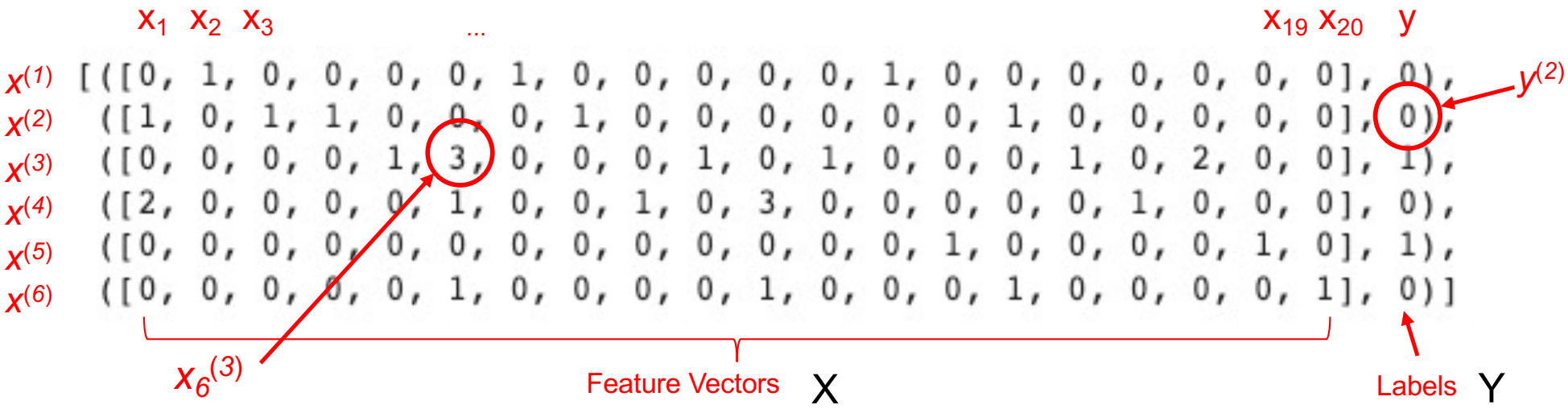
```
array([0.24973989, 0.75026011])
```

Data Format for Deep Learning Classifier

The training set consists of a matrix of features X and a vector of labels Y.

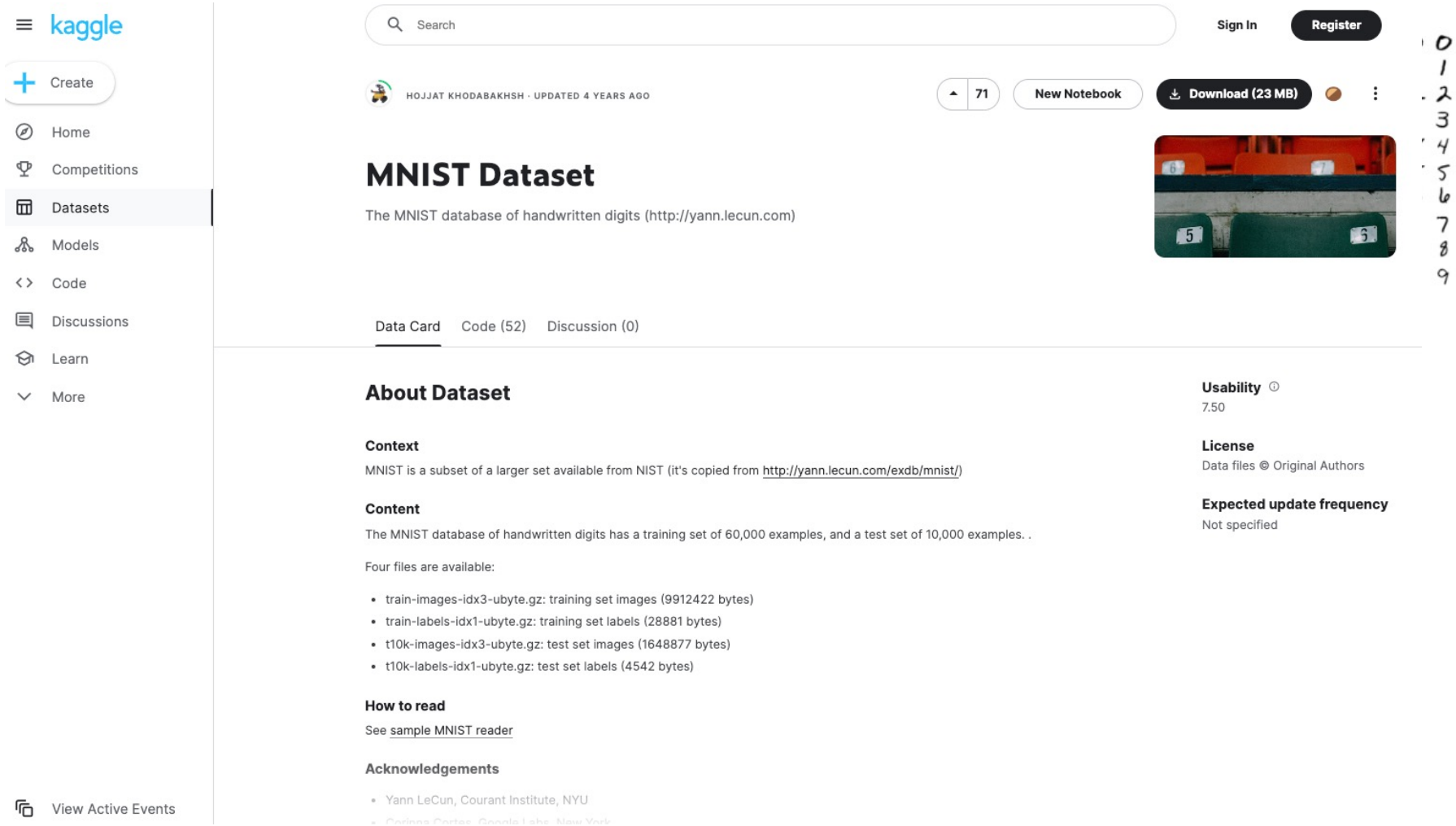
For each input observation $x^{(i)}$, we have a vector of features $[x_1, x_2, \ldots, x_n]$.

Feature j for input $x^{(i)}$ is x_j, more precisely $x_j^{(i)}$.



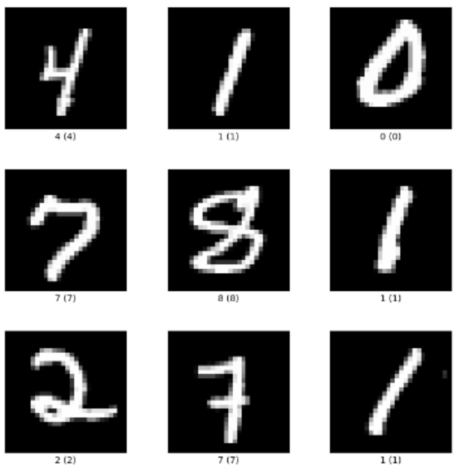
Multiclass Classification

We will consider the MNIST database of handwritten digits: this is the “Hello World” of deep learning.



Multiclass Classification

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:

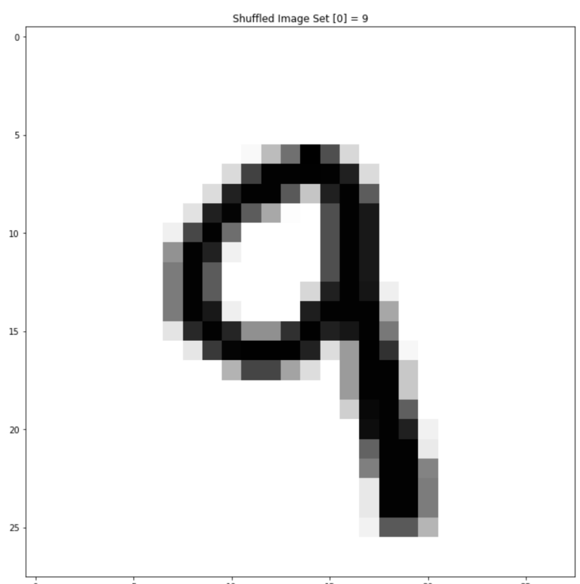


The 28*28 matrix is flattened into a 1D vector of length 784.

```
In [33]:  1 print(training_images[0])

[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.05882353 0.08235294 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.05882353
 0.54901963 0.49019608 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.16862746 0.98039216 0.49019608
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.7176471  0.98039216 0.49019608 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.69411767 0.9882353
 0.98039216 0.49019608 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.08627451 0.7176471  0.9882353  0.99607843 0.93333334 0.4117647
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.11372549 0.65882355 0.98039216
 0.98039216 0.93333334 0.43529412 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.08627451 0.65882355 0.98039216 0.98039216 0.98039216 0.4117647
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.7176471  0.98039216
 0.98039216 0.98039216 0.98039216 0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.9882353  0.98039216 0.98039216 0.98039216
 0.98039216 0.         0.         0.         0.         0.
 0.         0.         0.41960785 0.27450982 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
```

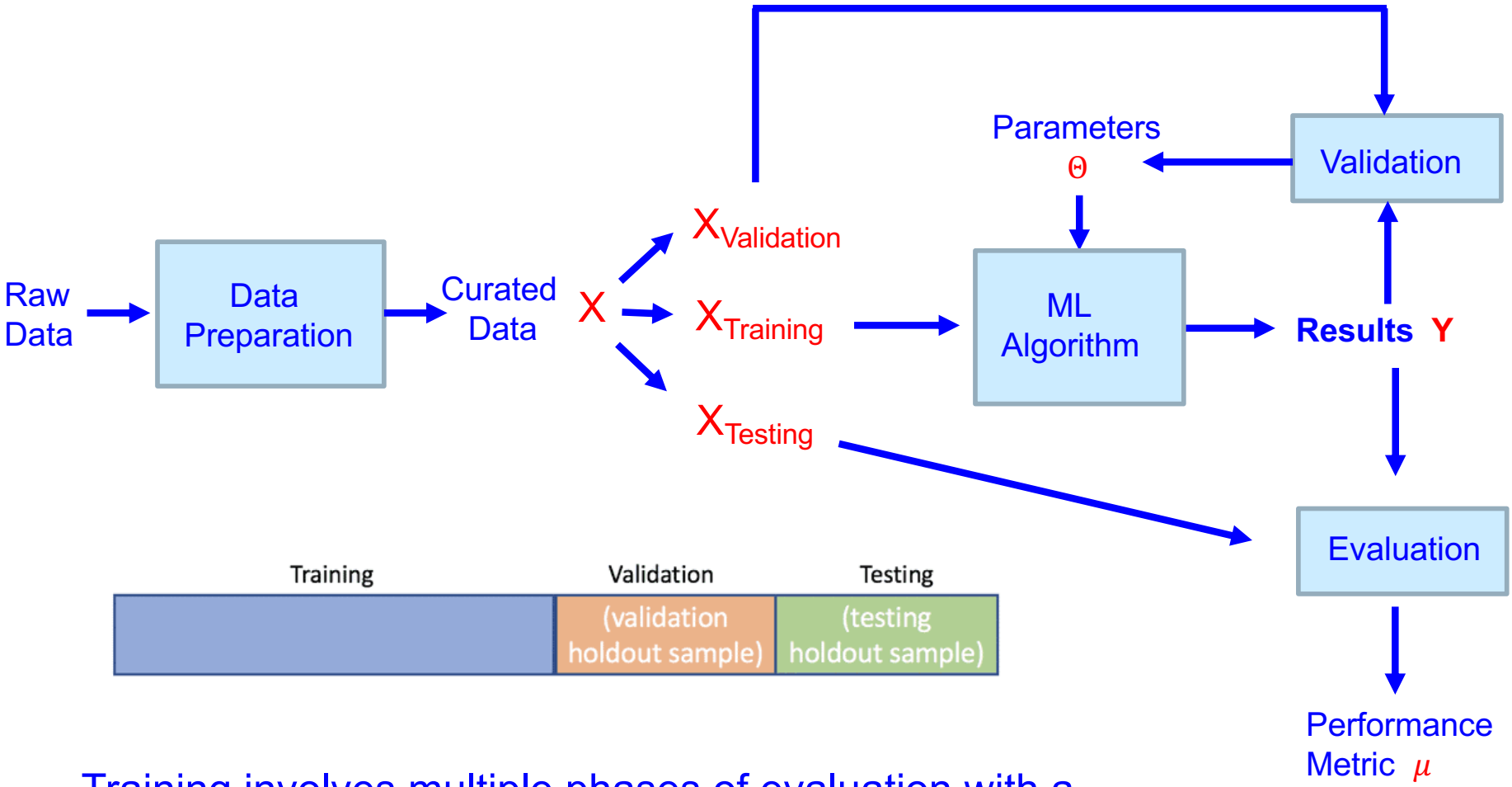
Note that the array is sparse, it is mostly 0's.



The label 9 is expanded into a one-hot vector:

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

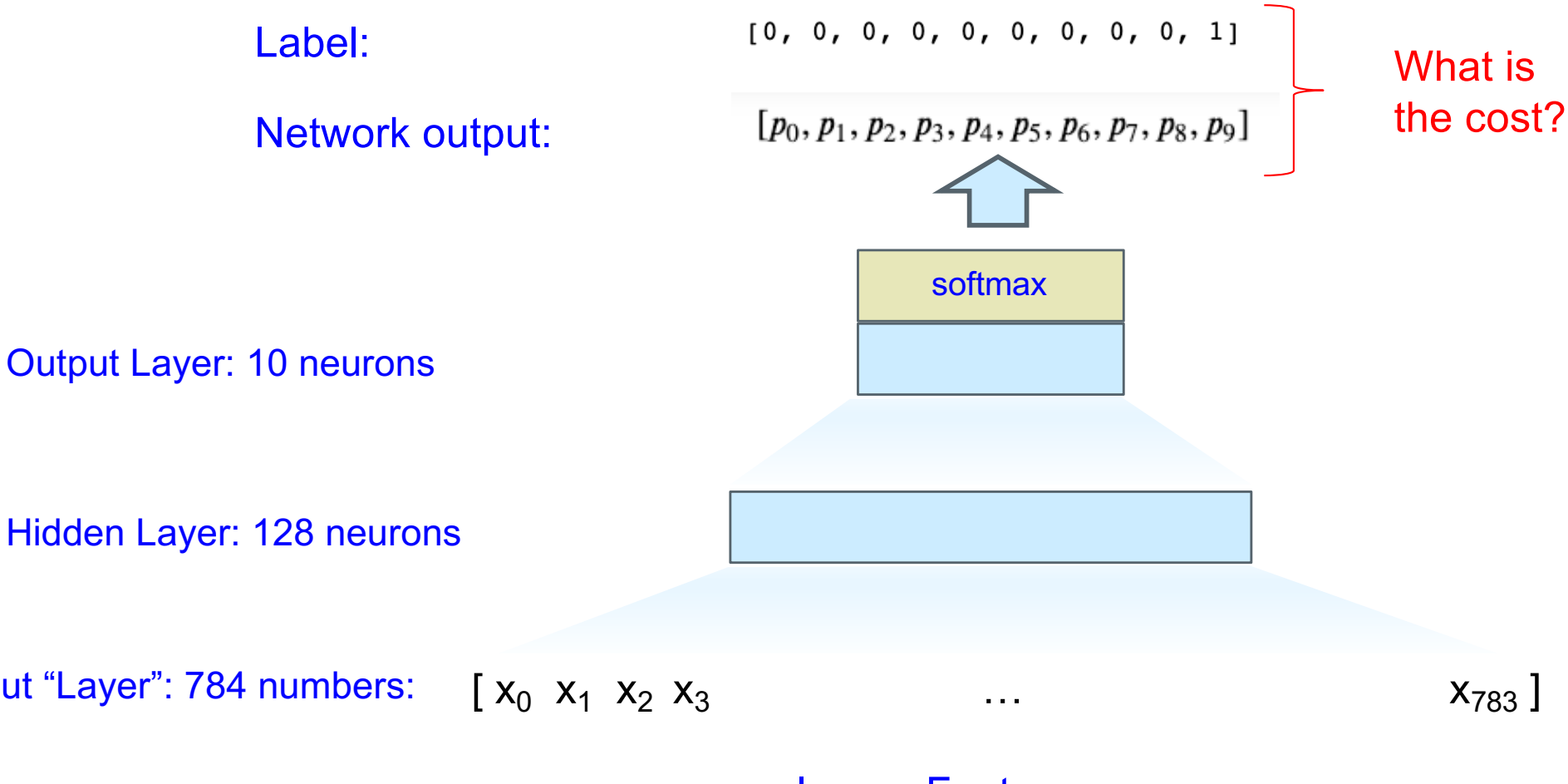
Recall: Supervised Machine Learning Workflow



Training involves multiple phases of evaluation with a validation set to find optimal values for the hyperparameters.

Cost/Loss Function for Classification

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



Cost/Loss Function for Classification

We need to compare two probability distributions

Label: `[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]` $\mathbf{y}$

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

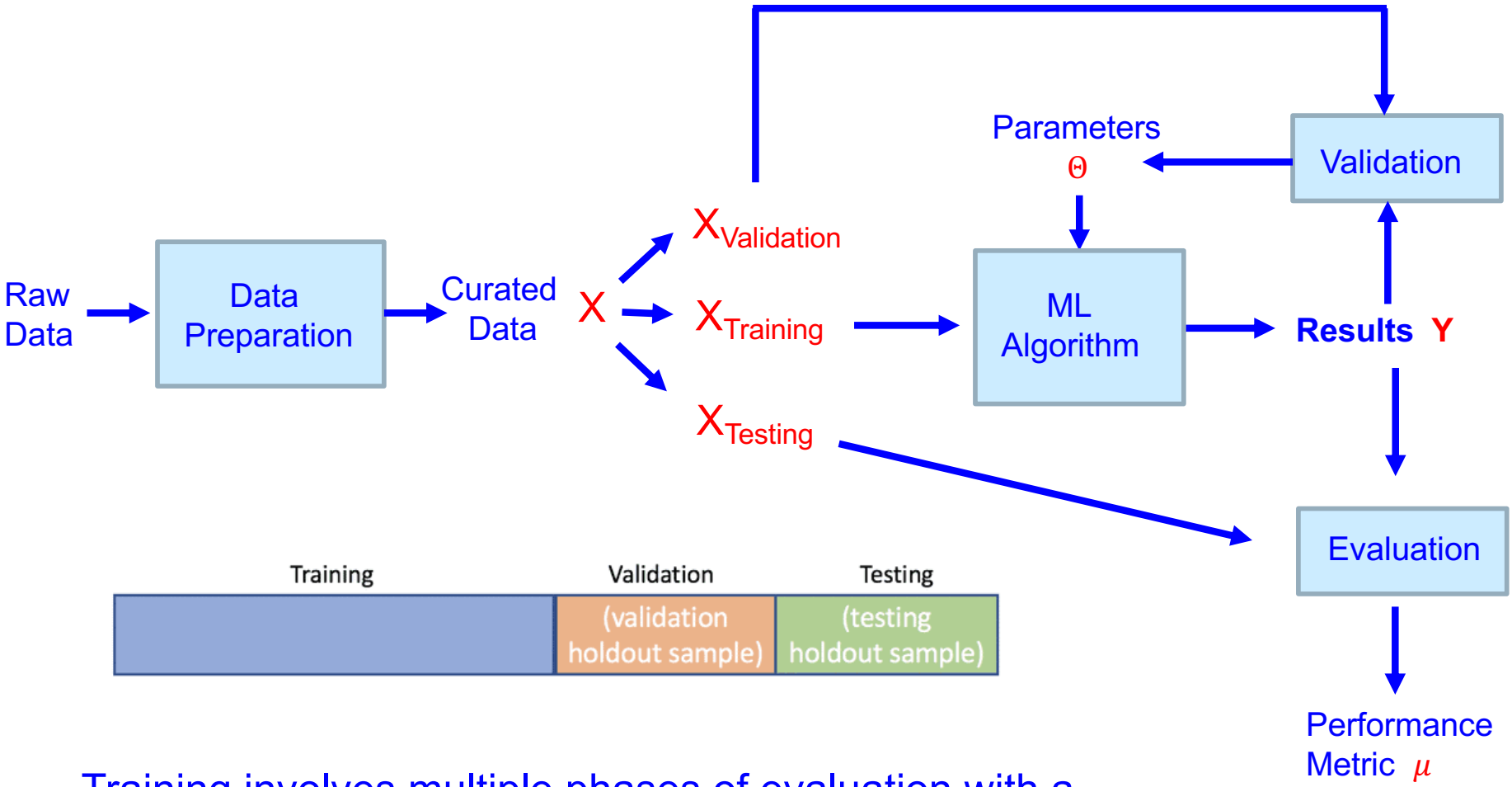
to measure how different they are. The function used to do this is called the Cross-Entropy Loss:

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

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:

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

Recall: Supervised Machine Learning Workflow



Training involves multiple phases of evaluation with a validation set to find optimal values for the hyperparameters.