MACHINE LEARNING FUNDAMENTALS

COEN 435

(Module_3: Neural Networks)

INSTRUCTOR: Dr. S. M. YUSUF

OBJECTIVE

❑ Software

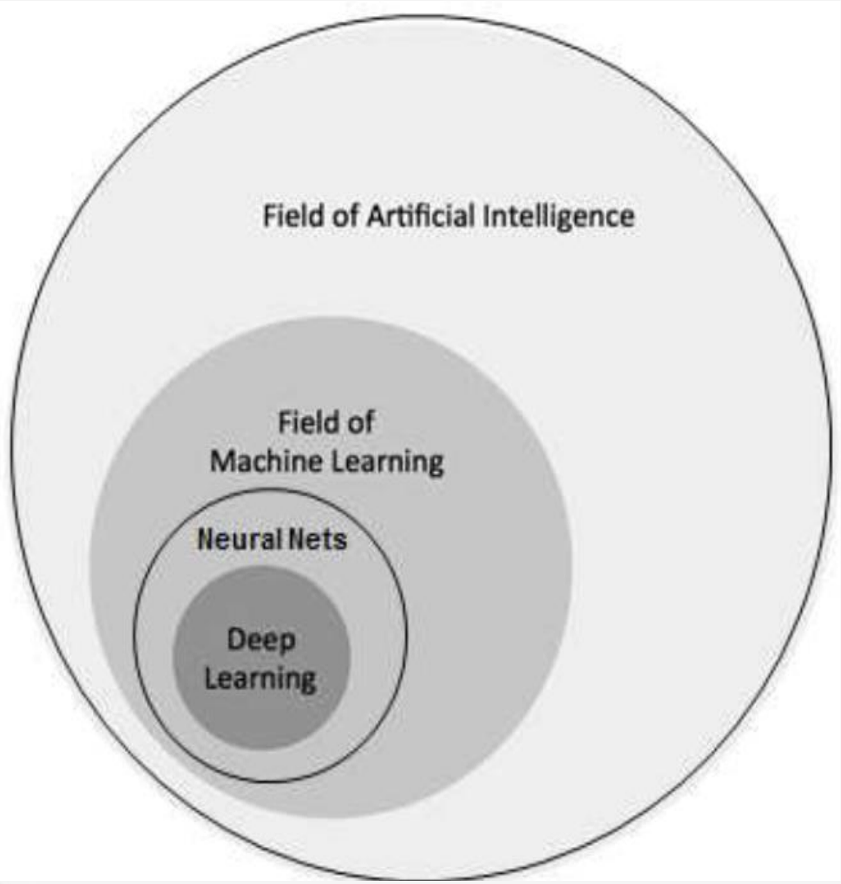
- ✓ Python (Assumption: Python programmer)
- ✓ Numpy (Pandas, Torch vision)
- ✓ TensorFlow (Keras, Pytorch)
- ✓ Sklearn
- ✓ Matplotlib

❑ Hardware

- ✓ 32-bit or 64-bit system architecture
- ✓ 2+ GHz CPU
- ✓ 4 GB RAM
- ✓ At least 10 GB of hard disk space

Introduction

- ✓ **subset of machine learning techniques that learns features directly from data by using several number of neurons organized in layers.**
- ✓ **This is a class of ML algorithms in the use a cascade of layers of processing units to extract features from data and make predictive guess about new data.**
- ✓ **Also known as Artificial Neural Network (ANN).**



Source: Machine Learning Guide, 9. Deep Learning

Neural Network (Cont.)

- ✓ Inspired by the structure of the neurons located in the human brain and how the brain works.
- ✓ Layers of neurons are interconnected in hierarchical manner.
- ✓ Learning is through progressive abstraction.

- ✓ The success of a subset of ANN (deep learning) is the availability of more training data (e.g. ImageNet) and,
  - ✓ Relatively low-cost GPUs for efficient numerical computation.

- ✓ Deep learning (DL) is utilized to analyze massive data of large industrial companies and an integral part of modern software production.

Artificial Neural Networks

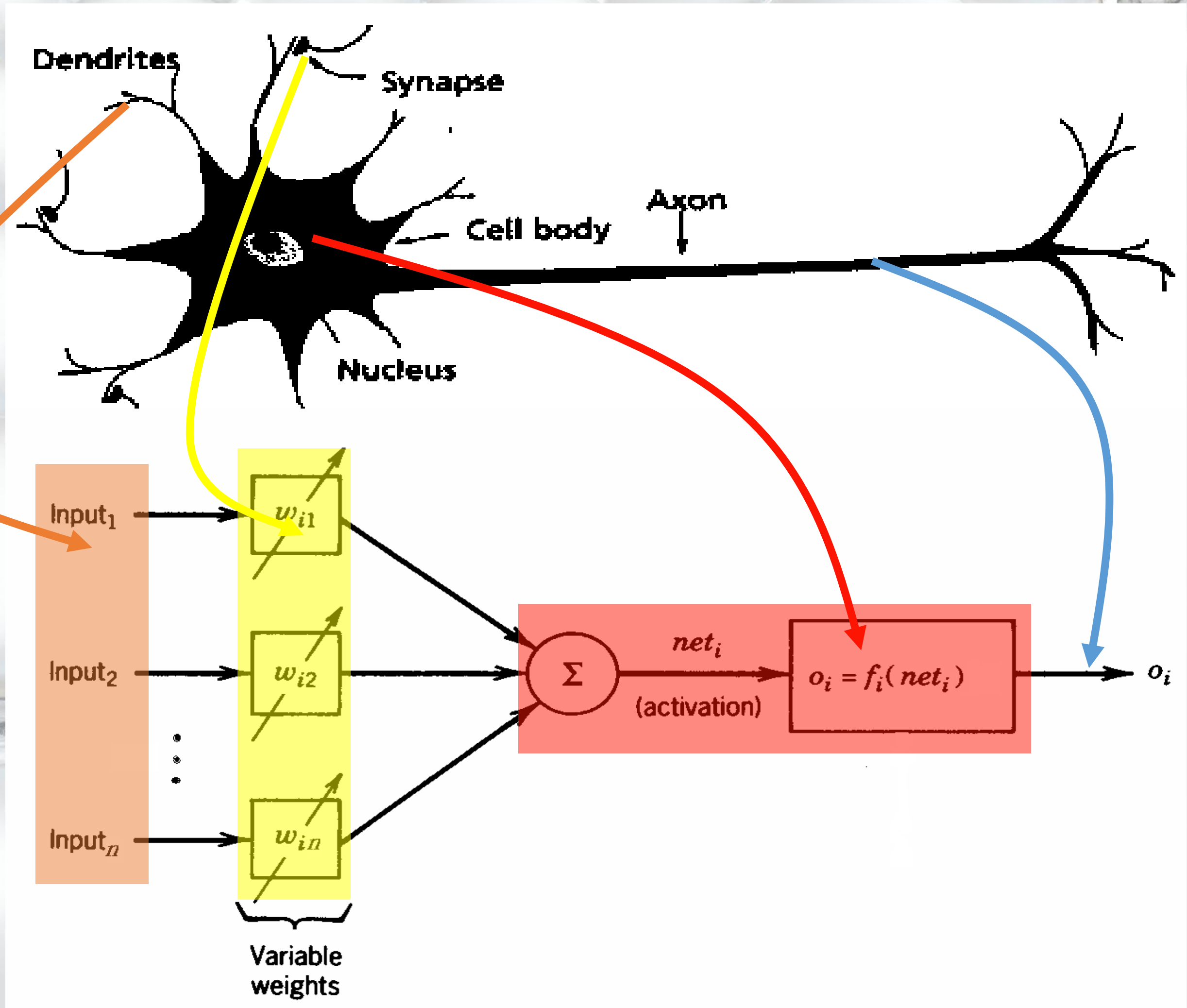
From experience: examples / training data

A physical neuron

Strength of connection between the neurons is stored as a weight-value for the specific connection.

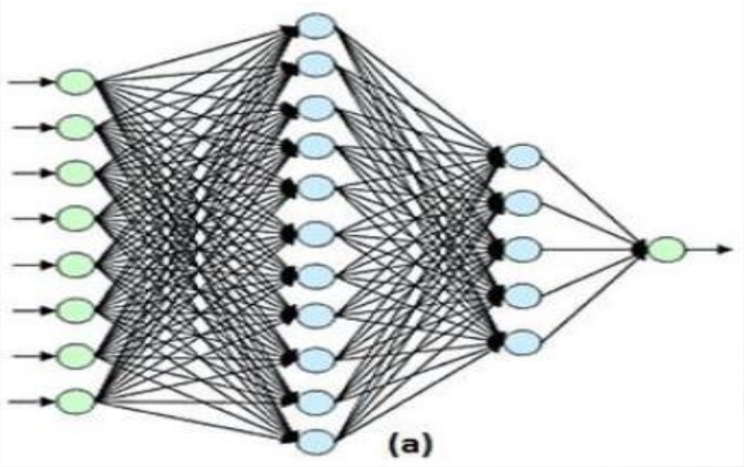
An artificial neuron

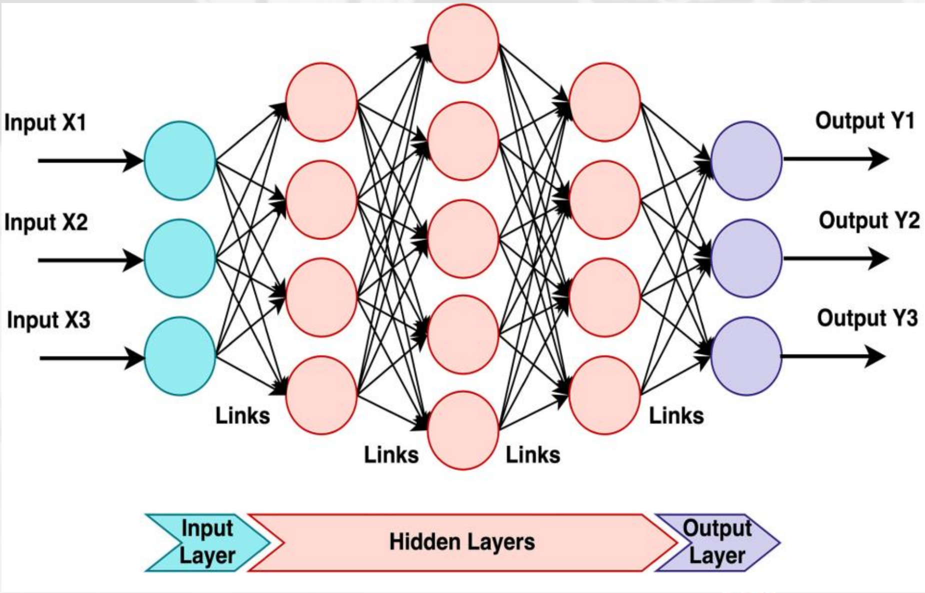
Learning the solution to a problem = changing the connection weights



ANN Cont.

- NN has 3 primary layers
  - ✓ Input layer
    - ✓ Contains input variables; number of nodes depends on the number of input variables; Connected to a hidden layer.
  - ✓ Hidden layer
    - ✓ Produces intermediate output; number depends on the nonlinearity (complexity) in the data.
  - ✓ Output layer



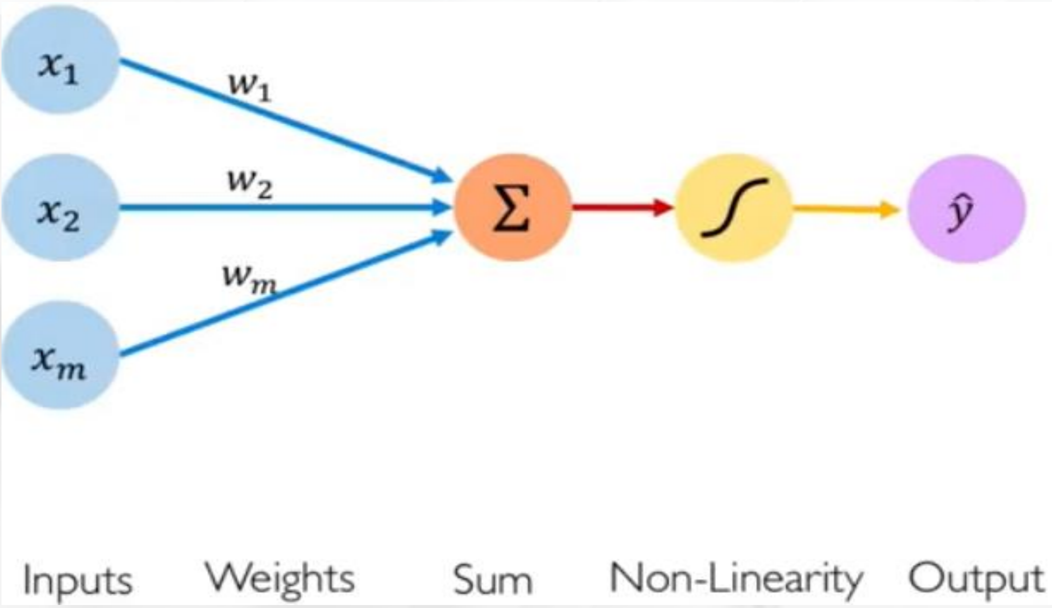


ANNs (a) Shallow Network (b) DNN (Khalil *et al.* 2019)

Types of Artificial Neural Networks

- **Includes**
  - ✓ Perceptron
  - ✓ Multi-Output Perceptron
  - ✓ Single Layer Neural Network
  - ✓ Multi-layered Perceptron (MLP)
- ✓ **Deep Learning**
  - ✓ Deep Neural Networks (DNN)
  - ✓ Convolutional Neural Networks (CNN)
  - ✓ Recurrent Neural Networks (RNN)
    - ✓ Long Short Term Memory (LSTM)
    - ✓ Bidirectional Long Short Term Memory (BLSTM)
    - ✓ Gated Recurrent Units (GRU)
  - ✓ Generative Adversarial Networks (GAN)

Forward Propagation in a Perceptron



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i \, w_i \right)$$

Non-linear activation function

Bias

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

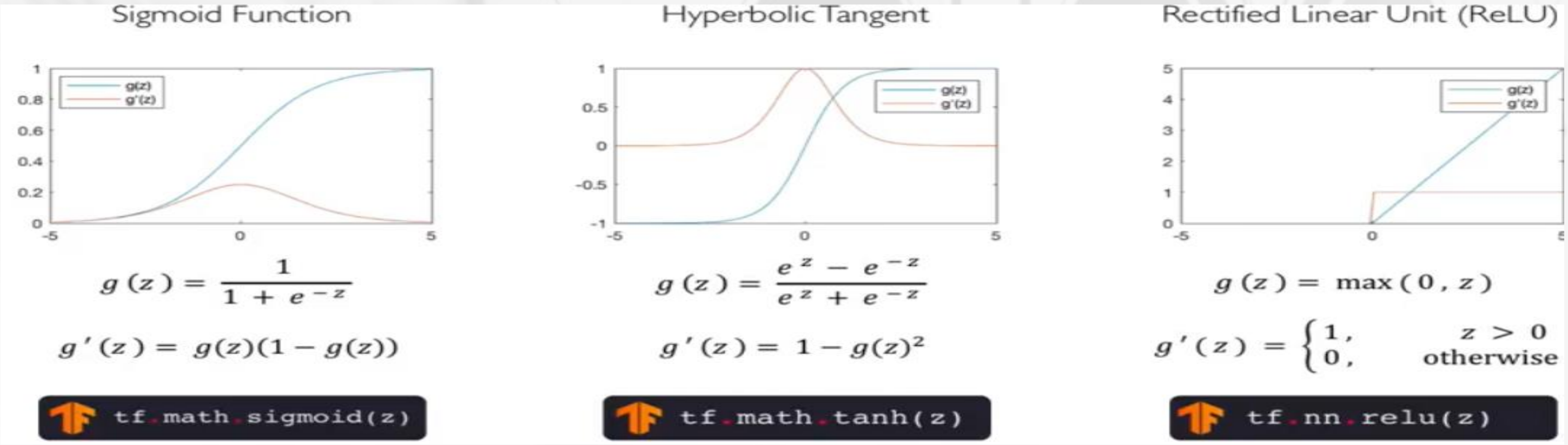
where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

- ❑ **The bias term shift the activation function left/ right regardless of the inputs.**
- ❑ **In vector form, the output of a single perceptron is the application of the activation function on the dot product of X and W.**

Activation Function

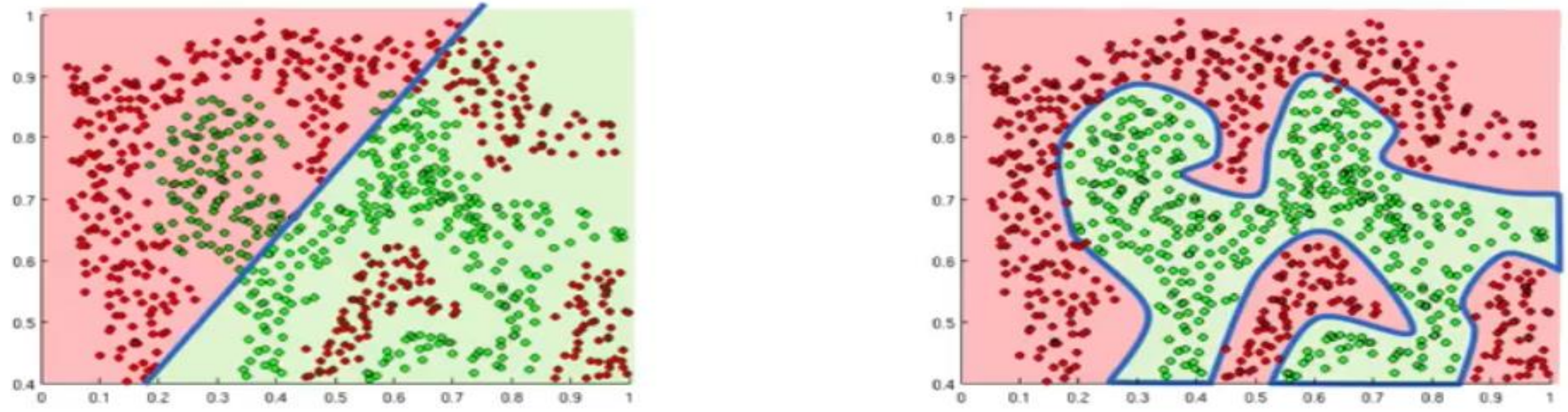
- Also called **Threshold** or **Transfer function**.
- **Functions that transforms the summed weighted inputs of a neuron, (Z) into an output, g(Z).**
- **Common activation functions includes; sigmoid, hyperbolic tangent, and Rectified linear units (ReLU).**



Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

- The purpose of activation function is to introduce non-linearities into the neural network.

- E.g: How do you distinguish between the red and green colored points?

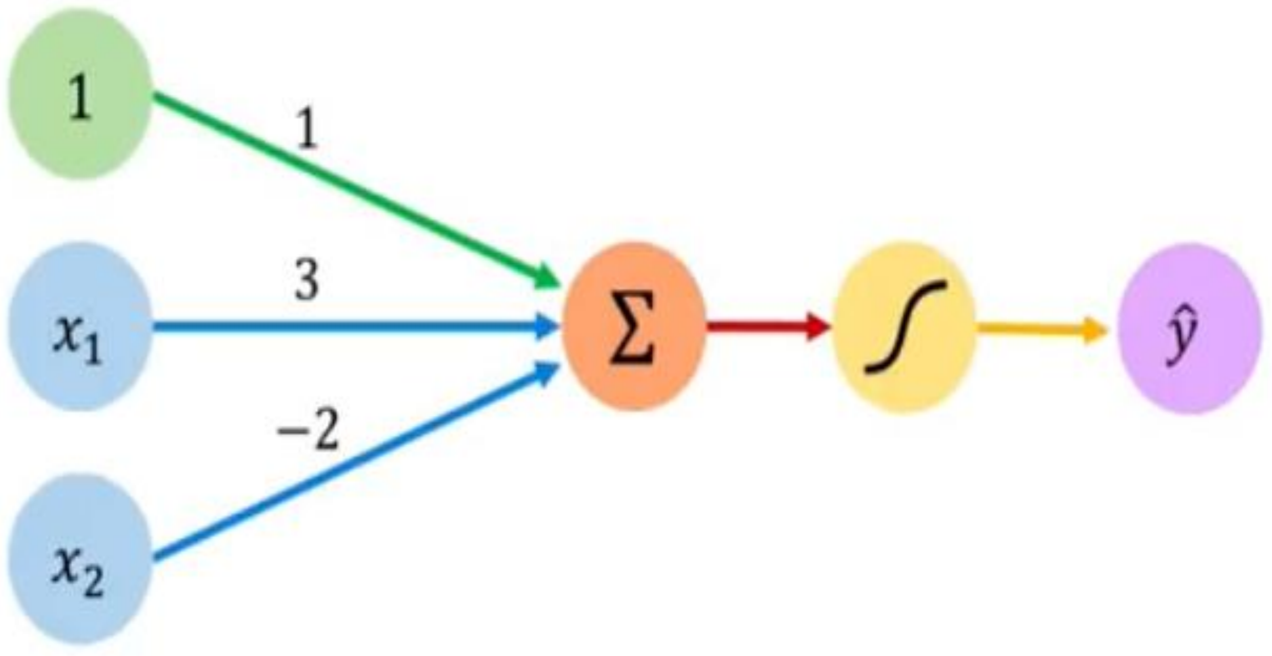


Linear activation functions produce linear decisions no matter the network size

Non-linearities allow us to approximate arbitrarily complex functions

Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Example 1



We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

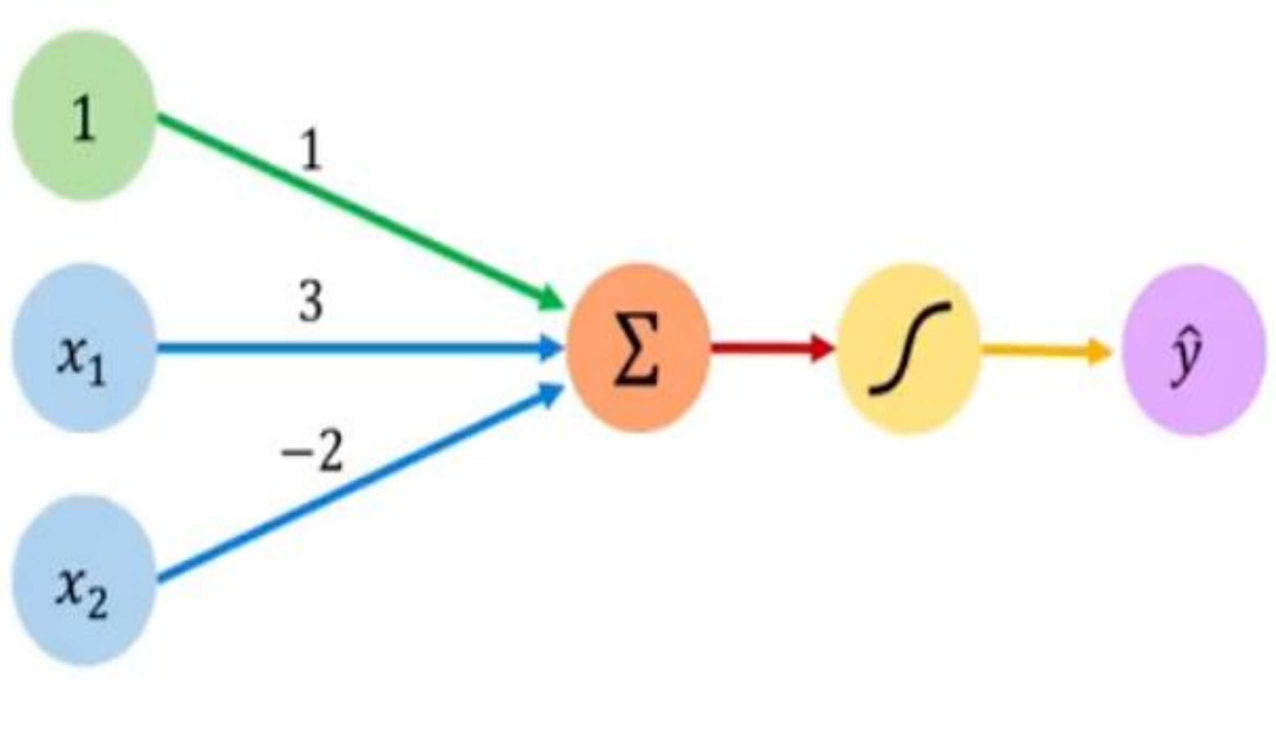
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

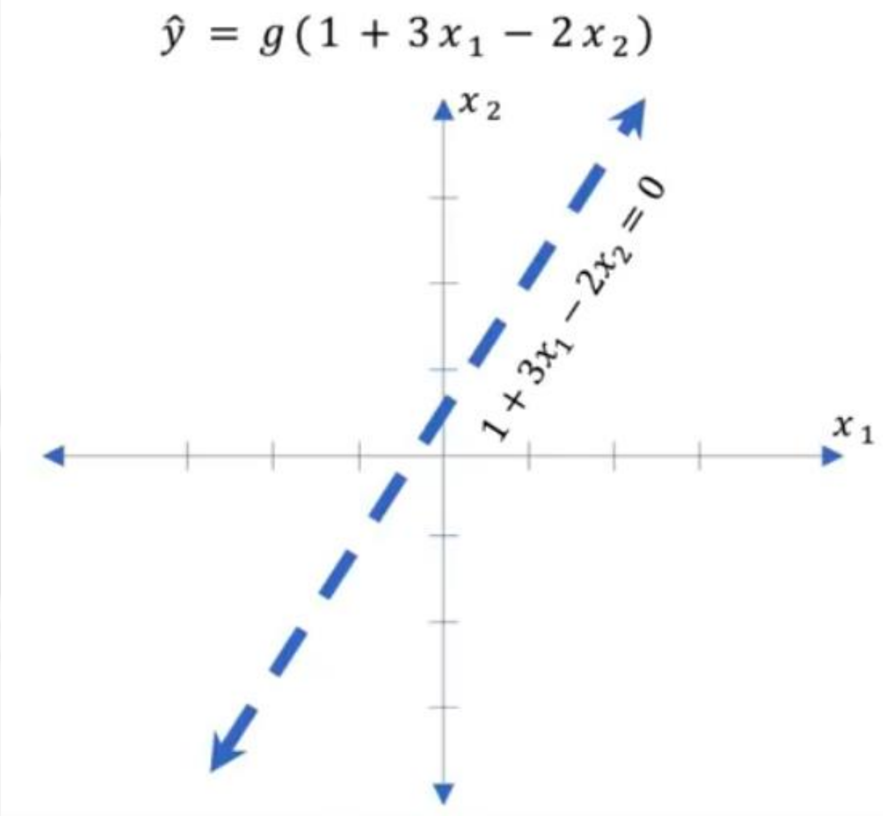
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

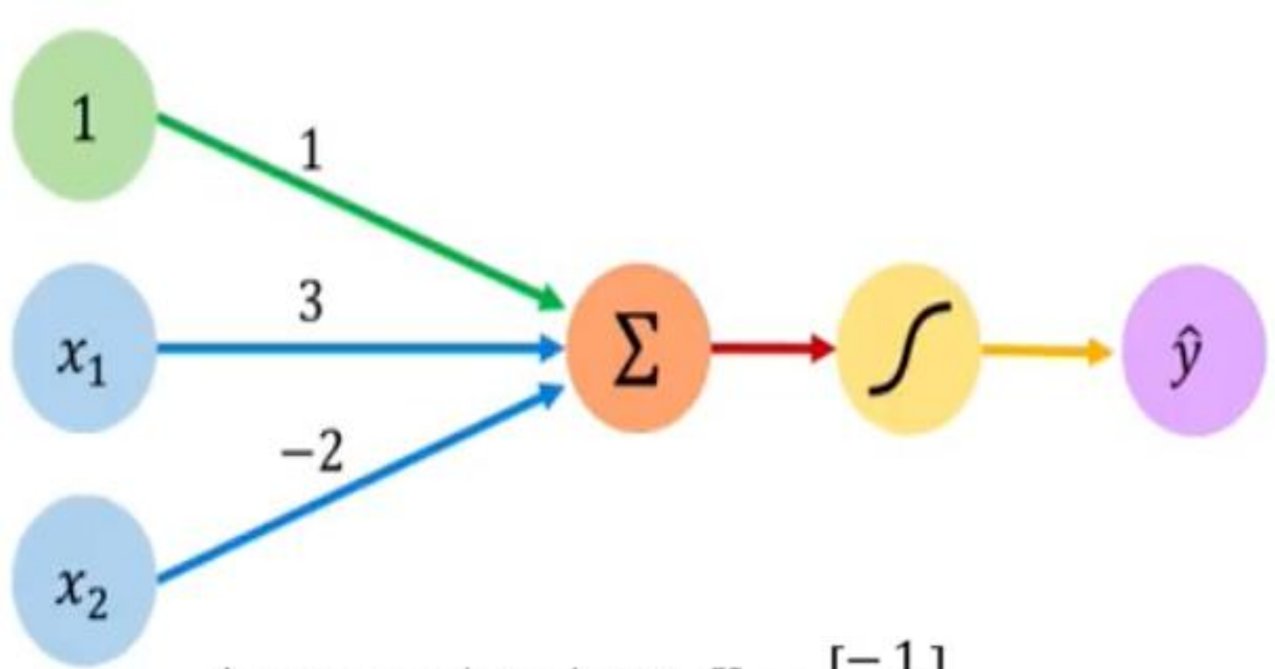
Example 1 Cont.





Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

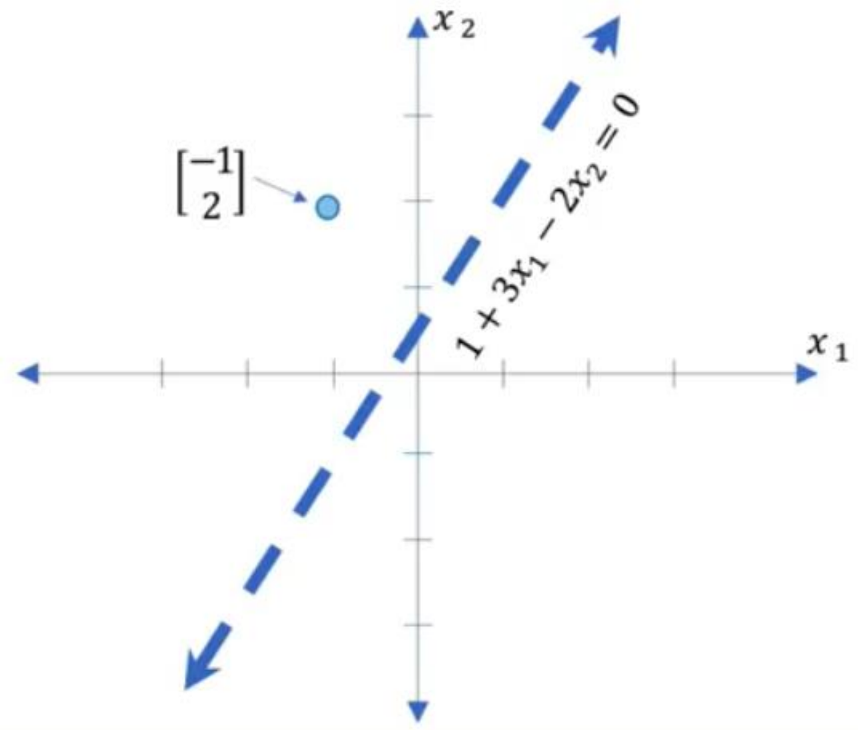
Example 1 Cont.



Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

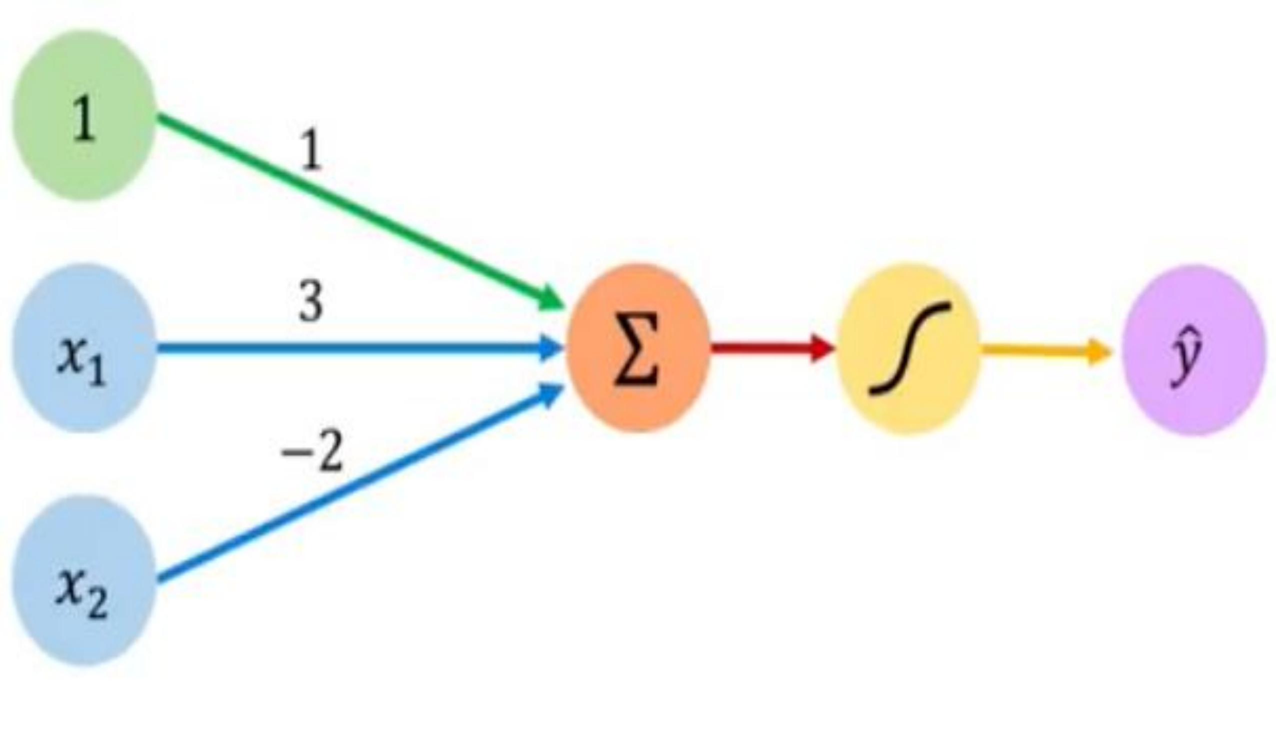
$$\hat{y} = g\left(1 + (3 * -1) - (2 * 2)\right)$$
$$= g(-6) \approx 0.002$$

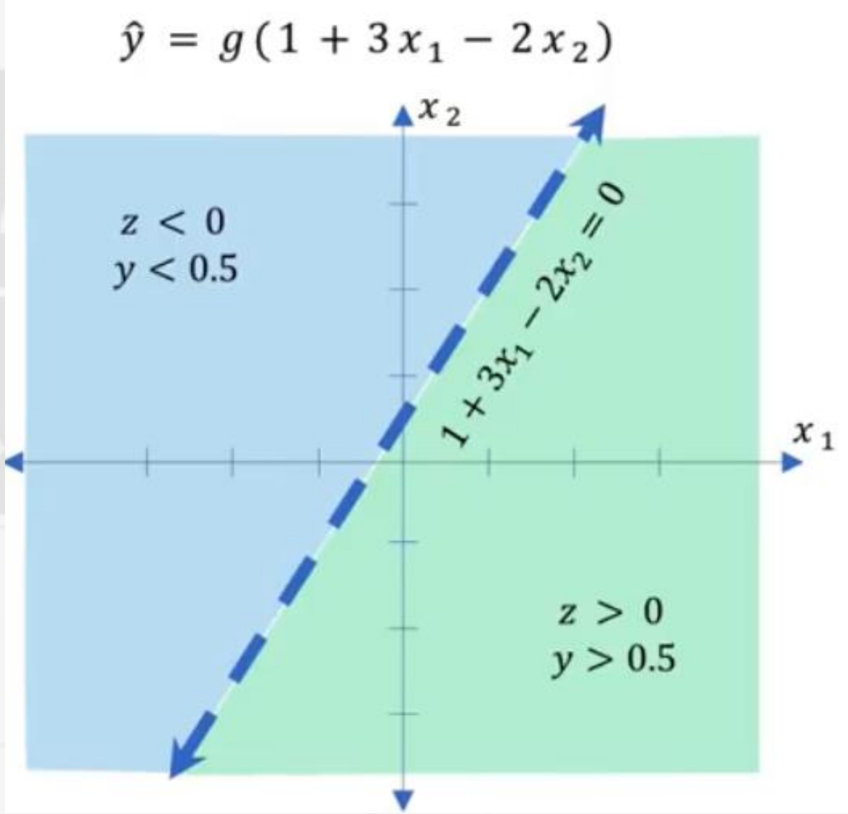
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

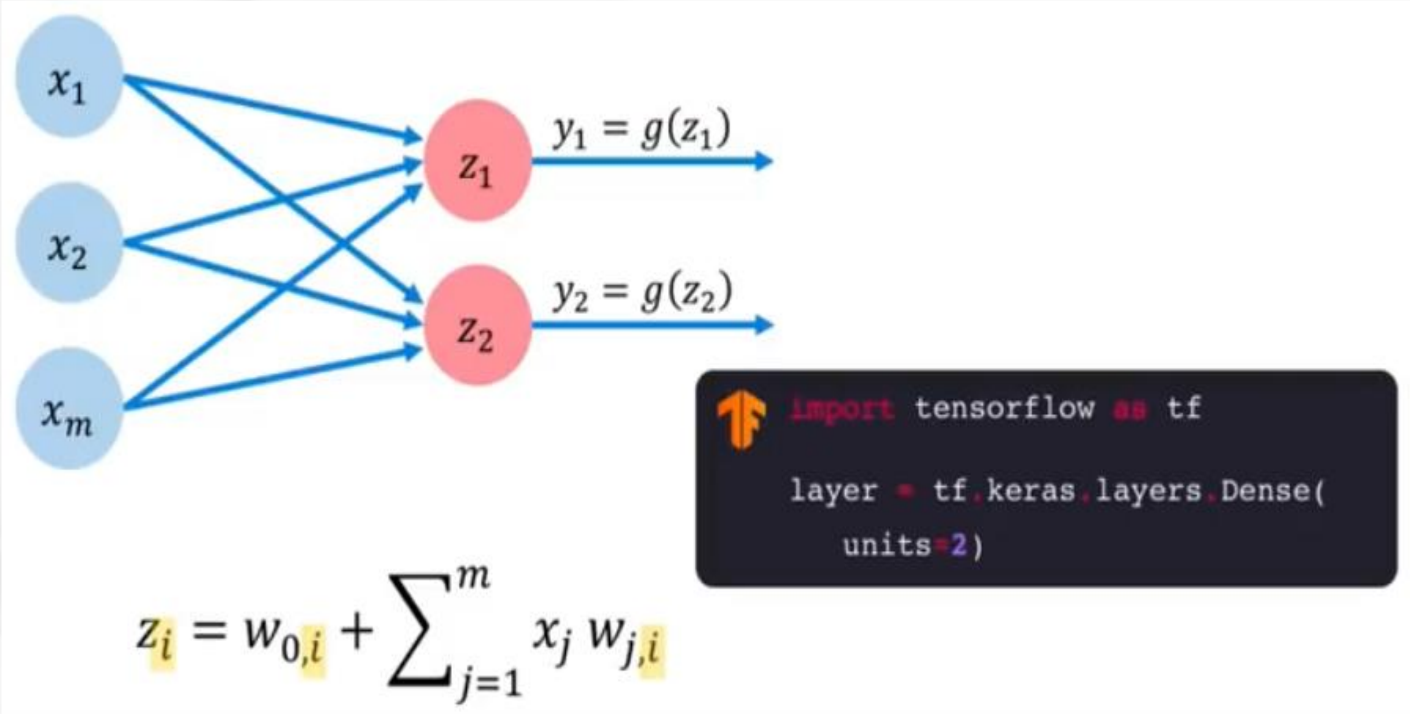
Example 1 Cont.





Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Multi-Output Perceptron



Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

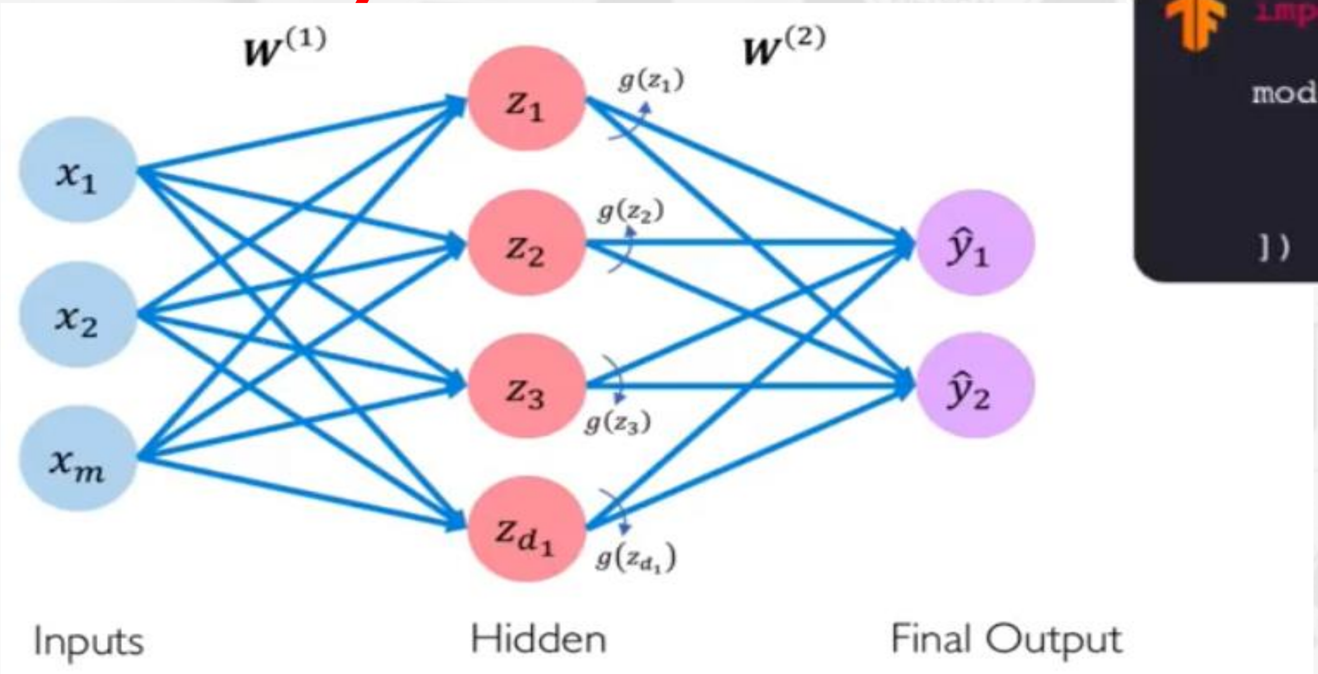
- ❑ **Since all inputs are fully connected to the layer of output neurons, the output layer is called a dense layer.**
- ❑ **A dense layer is a layer of neurons whose inputs are fully (densely) connected to outputs.**

Single Layer Neural Network

- This is a **NN** that is **made up of a single hidden layer of neurons** that feed into the output layer of neurons.
- **States of the hidden layer are learned.**

```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n),
    tf.keras.layers.Dense(2)
])
```



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

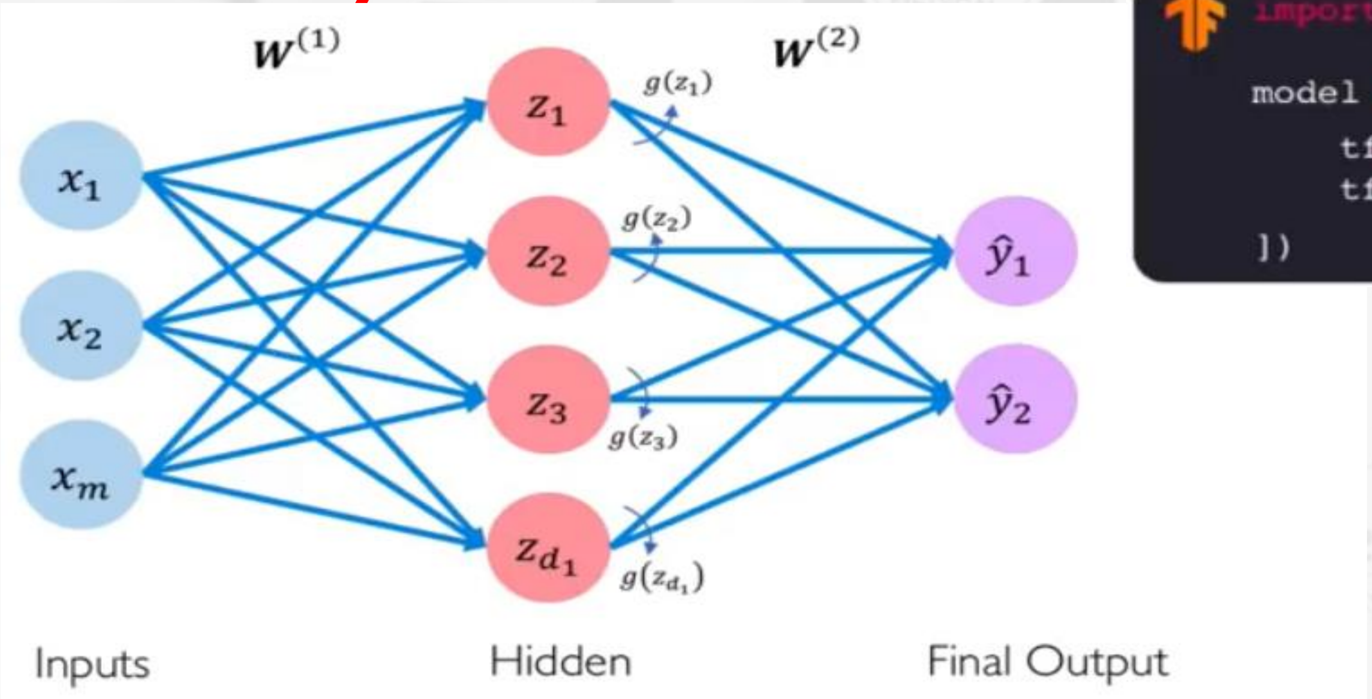
Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Single Layer Neural Network

- This is a network that is made up of a single hidden layer of neurons that feed into the output layer of neurons.
- States of the hidden layer are learned.

```python
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n),
    tf.keras.layers.Dense(2)
])
```

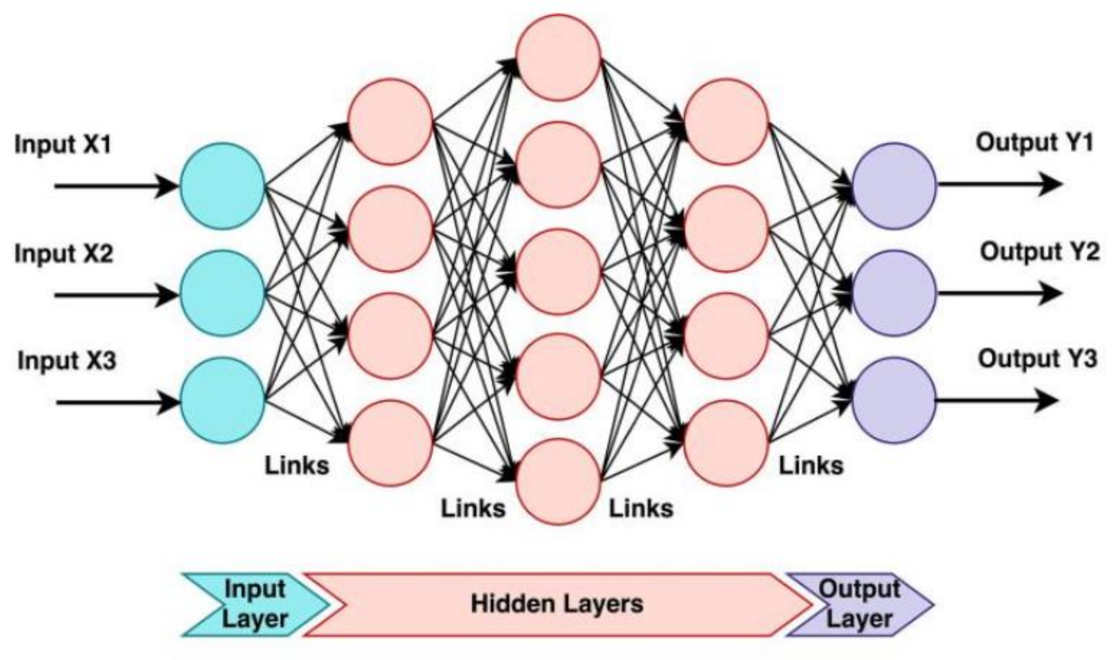


$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)}\right)$$

Source: Alexander Amini and Ava Soleimany, MIT 6.S191: Introduction to Deep Learning

Deep Neural Network

- This is a network that is made up of a stack of hidden layers.



Generic Deep Neural Network (DNN) Architecture (Khalil et al., 2019)

$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Training Neural Networks

- **Why?**
  - **Given a network wrong weights results to wrong predictions.**
  - **The network needs to know if its predictions are acceptable or not (based on historical data).**
- **Training a neural network is to teach it how to perform a task.**
  - **Fitting a model.**
- **Empirical loss function**
  - **Measures the cost incurred from incorrect predictions.**
  - **The type of loss function depends on the task at hand; classification or regression.**

Neural Network Algorithm

- ❑ Allows us to find the weights using backpropagation algorithm.
  - ✓ 5 steps in the algorithm
- ▪ Random Initialization
  - ✓ Initializes the weights by randomly selecting the values of the weights; using any prior information or standard distribution.
- ▪ Activation and feed forward (Forward propagation)
  - ✓ Involves weights multiplication with input variables, then summation, and applying activation function.
  - ✓ The predicted value is obtained at the end of this step.
- ▪ Error calculation & backward propagation
  - ✓ Finding the error between the actual and predicted value.
  - ✓ And propagating backward to find error contribution at the hidden layers is known as backpropagation.
    - ▪ Because error at the output layer is not entirely due to wrong weights connected to that layer.

Neural Network Algorithm Cont.

- Weight Updating
  - ✓ Adjusting the weight of a node so that the error at hidden nodes decreases, which in turn reduces the total error.

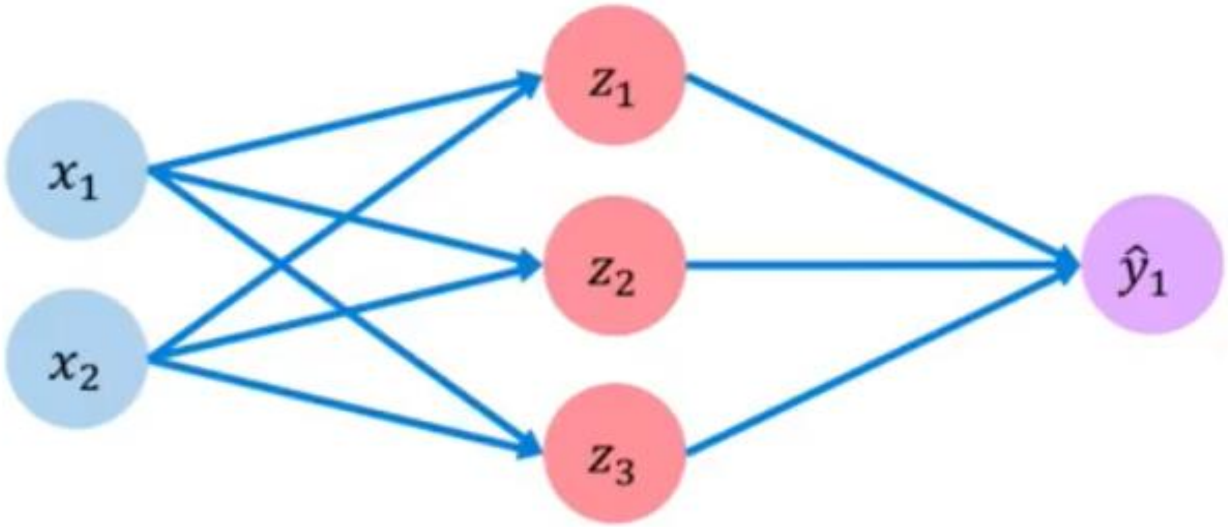
- Stopping Criteria
  - ✓ A full cycle of sending the whole data in a feedforward step followed by error calculation and backpropagation followed by weight updating is 1 epoch.
  - ✓ Repeat these epochs until we reach either zero error or when weights stop updating.

Training Neural Networks Cont.

❑ EMPIRICAL LOSS FUNCTION

The **empirical loss** measures the total loss over our entire dataset

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \end{array}$$

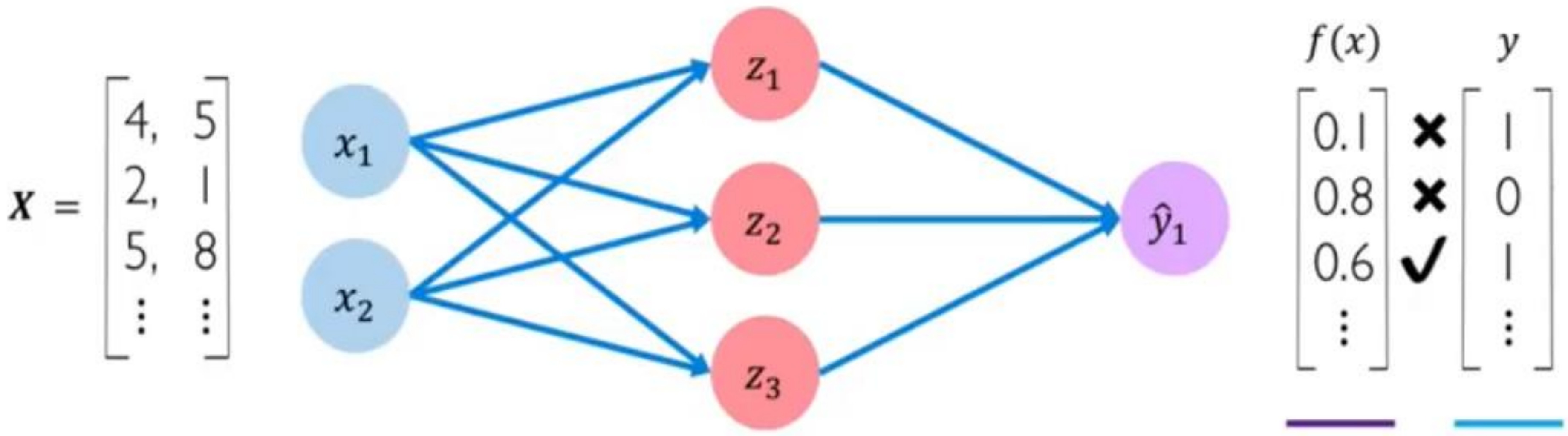
Also known as:

- Objective function
- Cost function
- Empirical Risk

$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\Big(\underbrace{f\big(x^{(i)}; W\big)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\Big)$$

❑ BINARY CROSSENTROPY LOSS FUNCTION

Cross entropy loss *can be used with models that output a probability between 0 and 1*



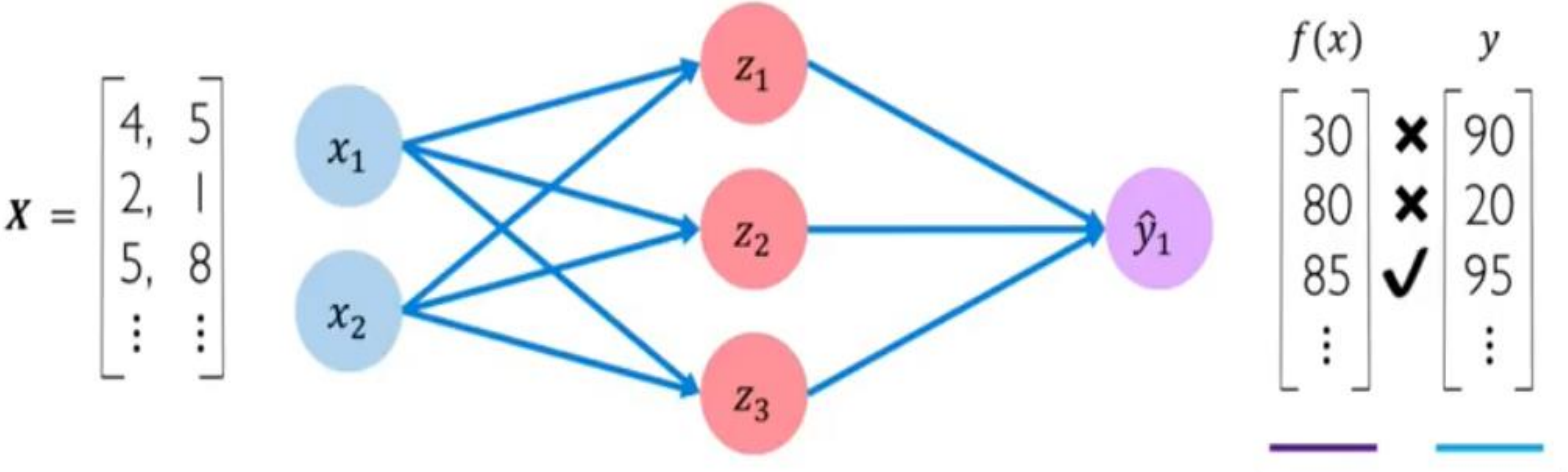
$$J(\mathbf{W}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)};\mathbf{W}\big)}_{\text{Predicted}}\Big) + (1-\underbrace{y^{(i)}}_{\text{Actual}})\log\Big(1-\underbrace{f\big(x^{(i)};\mathbf{W}\big)}_{\text{Predicted}}\Big)$$

```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

Training Neural Networks Cont.

❑ MEAN SQUARED ERROR LOSS FUNCTION

Mean squared error loss *can be used with regression models that output continuous real numbers*



Final Grades
(percentage)

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)}; W\big)}_{\text{Predicted}}\Big)^2$$

```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )
```

❑ LOSS OPTIMIZATION

- ✓ **Finding the network weights that achieve the lowest loss.**

$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

$$W^* = \underset{W}{\operatorname{argmin}}\, J(W)$$

$$W = \left\{W^{(0)}, W^{(1)}, \cdots\right\}$$

❑ What values of W gives us the minimum loss?

- ✓ **First, understand how the gradient (slope), $\frac{\partial J(W)}{\partial W}$, changes.**
- ✓ **Tells us how much we want to change the weights, in order to reduce the loss incurred on a particular training example.**
- ✓ **Compute using the gradient descent algorithm.**

Gradient Descent Optimization Algorithm

- ❑ **Initialize weights randomly** $\sim \mathrm{N}(0,\sigma^2)$
- ❑ **Loop until convergence:**
  - ✓ **Compute gradient using back propagation,** $\frac{\partial J(W)}{\partial W}$
  - ✓ **Update weights,** $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
- ❑ **Return weights**

- ❑ **Optimizers for Training Deep Neural Networks**
  - ❑ **Stocastic Gradient Descent (SGD)**
  - ❑ **Mini-batch SGD**
  - ❑ **SGDM**
  - ❑ **Adadelta**
  - ❑ **Adam**

❑ 4 Key Ingridents of developing DNN

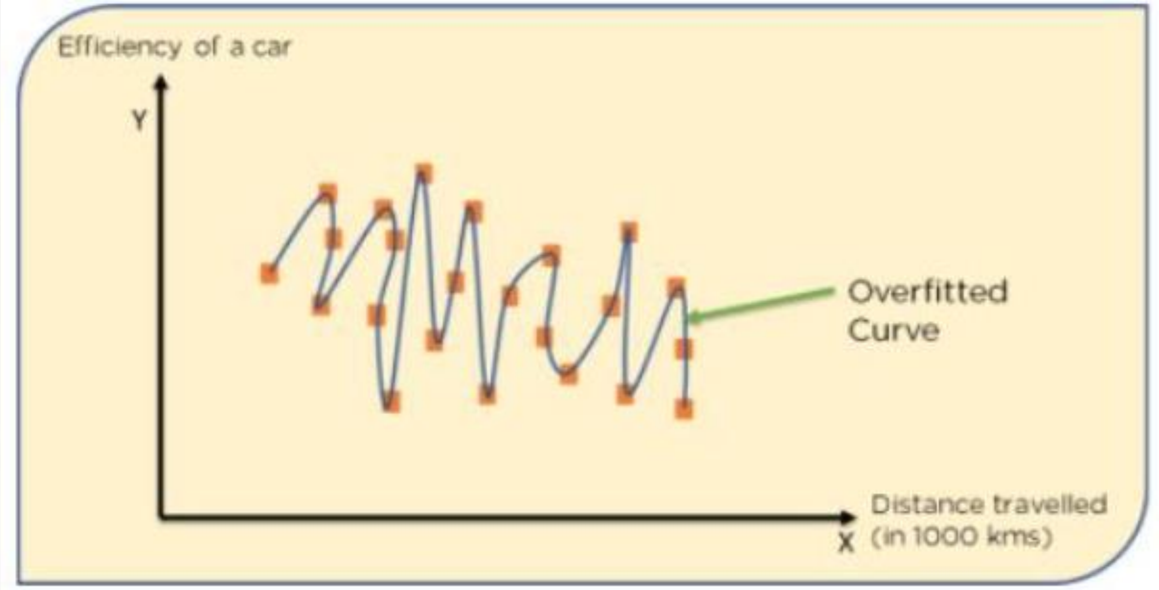
- ✓ **Dataset**
  - ✓ How many data points?
  - ✓ For regression/classification?
  - ✓ How many classes?
  - ✓ Multi-label/ multi-class/binary-class?
- ✓ **Loss function**
  - ✓ Binary Cross-entropy/categorical cross-entropy?
- ✓ **Model/ Architecture**
  - ✓ Type of DL algorithm?
  - ✓ No. of layers and nodes?
  - ✓ Make informed decision by experimenting different architectures
- ✓ **Optimization method**
  - ✓ SGD?
  - ✓ Batches / Adaptive step-size?
  - ✓ Also, set learning rate, regularization strength, No. of epochs, e.t.c

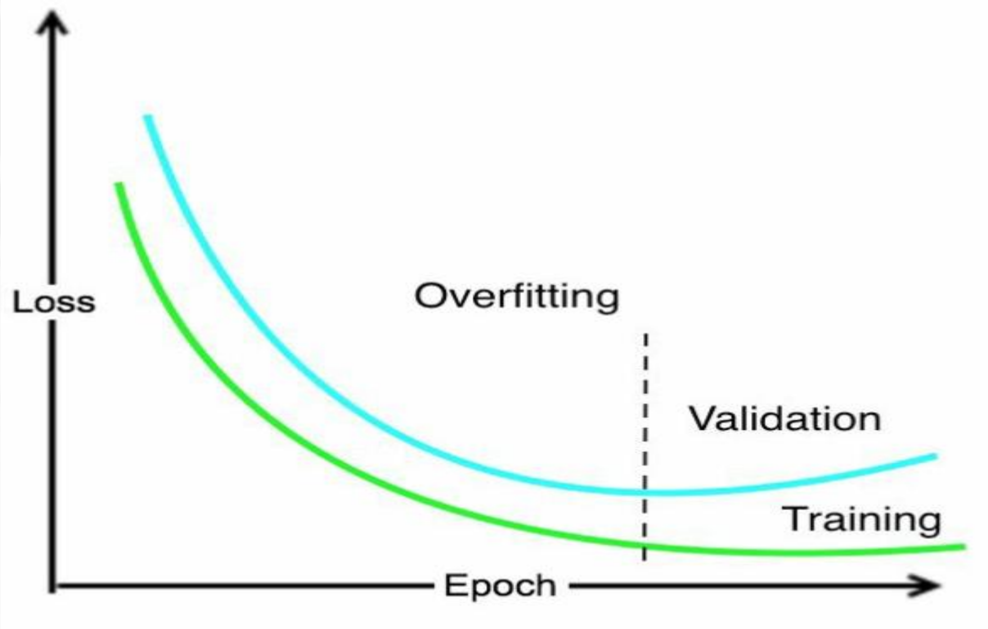
❑ SELECTING LOSS FUNCTIONS AND ACTIVATION FUNCTIONS

- ✓ **Loss function must fit activation function in the last layer.**
- ✓ **Squared loss and Hinge loss fit together with linear activation function.**
- ✓ **Negative log likelihood (NLL) loss fits together with sigmoid function.**
- ✓ **Negative log likelihood Multiclass (NLLM) loss fits together with Softmax function.**

OVER FITTING

- A scenario that occurs during training where the deep learning model learns from details along with noise and random points on the curve.
  - ✓ Tries to fit each data points on the curve.
  - ✓ Given a new data point, the model curve may not correspond to the patterns in the new data.
  - ✓ Thus, model cannot predict very well on test/validation set.





https://www.simplilearn.com/tutorials/

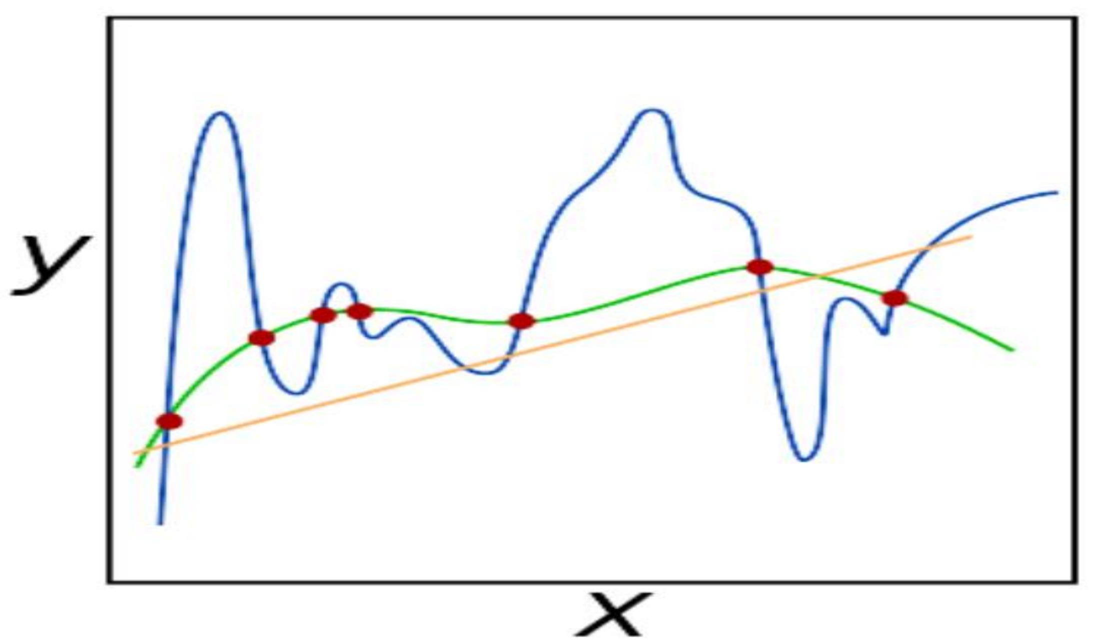
OVER FITTING Cont.

- **Reasons for Overfitting**
  - ✓ **Data utilized for training contain noise values in it.**
  - ✓ **Model has a high variance.**
  - ✓ **Limited/ scarce training data.**
  - ✓ **Model is too complex.**
- **NB:**
  - ✓ **There is the risk of overfitting when optimizing loss on the training data.**
  - ✓ **Overfitting is not a huge problem for large networks trained on large amount of data.**

UNDERFITTING

- A scenario that occurs where a Deep Learning model can **neither learn the relationship between variables in the data nor predict (or classify)** a new data point.
- Underfitted model performs poorly on the training data and is not able to model the relationship between the input data and output class labels.

- **Reasons for Underfitting**
  - ✓ Data utilized for training is not clean (contains garbage values).
  - ✓ Model has high bias.
  - ✓ Size of training data is not enough.
  - ✓ Model is too simple.



Overfit (Blue), Underfit (Orange) and Generalizing (Green) Models

REGULARIZATION

- These are strategies utilized to calibrate a deep learning network in order to minimize the loss as well as reduce overfitting or underfitting.
  - ✓ reduce the test error, possibly at the expense of increased training error.

- Regularization strategies help choose a set of parameters that help ensure deep learning model generalizes well.
  - ✓ Regularization helps control the DL model capacity, ensuring that the model is better at making (correct) classifications on data points that they were not trained on.

- Without Regularization, classifiers can easily become too complex and overfit to our training data.

- Second to learning rate, regularization is the most important parameter of DL model that you can tune.

REGULARIZATION Cont.

- Regularization function can be added to the loss function or explicitly or implicitly added to the network architecture.

- **Methods/ Strategies?**
  - Regularization penalties.
    - ✓ Utilized to update the loss function by adding an additional parameter to constrain the capacity of the model.
    - ✓ E.g. Weight Decay (L_1 , L_2 & Elastic Net)
    - ✓ Weight decay penalize the norm of all the weights.
  - Dropout (Explicit)
  - Batch normalization (Explicit)
  - Data Augmentation (implicit)
  - Early stopping (implicit)

- Too much of regularization can result to underfitting.

REGULARIZATION Cont.

❑ Weight Decay Regularization Penalty

- ✓ Let the empirical loss be, **L** (Loss over the entire training set)
  - ✓ Recall, **L** was originally;

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\Big(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\big(x^{(i)};W\big)}_{\text{Predicted}}\Big)^2$$

- ✓ Regularization penalty; commonly written as; **R(W)**
- ✓ **Penalizing the loss, J(W), the loss update;**

$$J(W) = L + \lambda * R(W)$$

$$\lambda = \text{Re}\,gularization\ term$$

$$\eta = Learning\ \ Rate$$

- ✓ **Consequently, the weight update will be of the form;**

$$W \leftarrow W - \eta\left(\frac{\partial J(W)}{\partial W} - \lambda W\right)$$

REGULARIZATION Cont.

- ✓ **Simplifying the weight update;**

$$W \leftarrow W(1-\lambda\eta)-\eta\frac{\partial J(W)}{\partial W}$$

- ✓ **The weight, W, decays by a factor of $(1-\lambda\eta)$ before taking a gradient step.**

❑ **For;**

- ✓ **L1 (Lasso) Regularization**
  - ✓ **Takes the sum of the absolute value of the weights.**

    i.e. $R(W)=\sum_i\sum_j\left|W_{i,j}\right|$
- ✓ **L2 (Ridge) Regularization**

$$R(W)=\|W\|=\sum_i\sum_j W_{i,j}^2$$

  - ✓ **Discourages large weights in the matrix W; preferring smaller ones.**
- ✓ **Elastic Net Regularization seeks to combine both L1 & L2 regularization.**

$$R(W)=\sum_i\sum_j \beta W_{i,j}^2+\left|W_{i,j}\right|$$

REGULARIZATION Cont.

❑ Early Stopping

- ✓ Easiest to implement and is in fairly common use.
- ✓ The idea is to train on your training set, but at every epoch, evaluate the loss of the current W on a validation set.
- ✓ Generally, loss on the training set reduces consistently with each iteration.
    - ✓ the loss on the validation set will initially decrease, but then begin to increase again.
- ✓ Stop training when validation loss increases and return the weights that had the lowest validation error.

REGULARIZATION Cont.

❑ Dropout

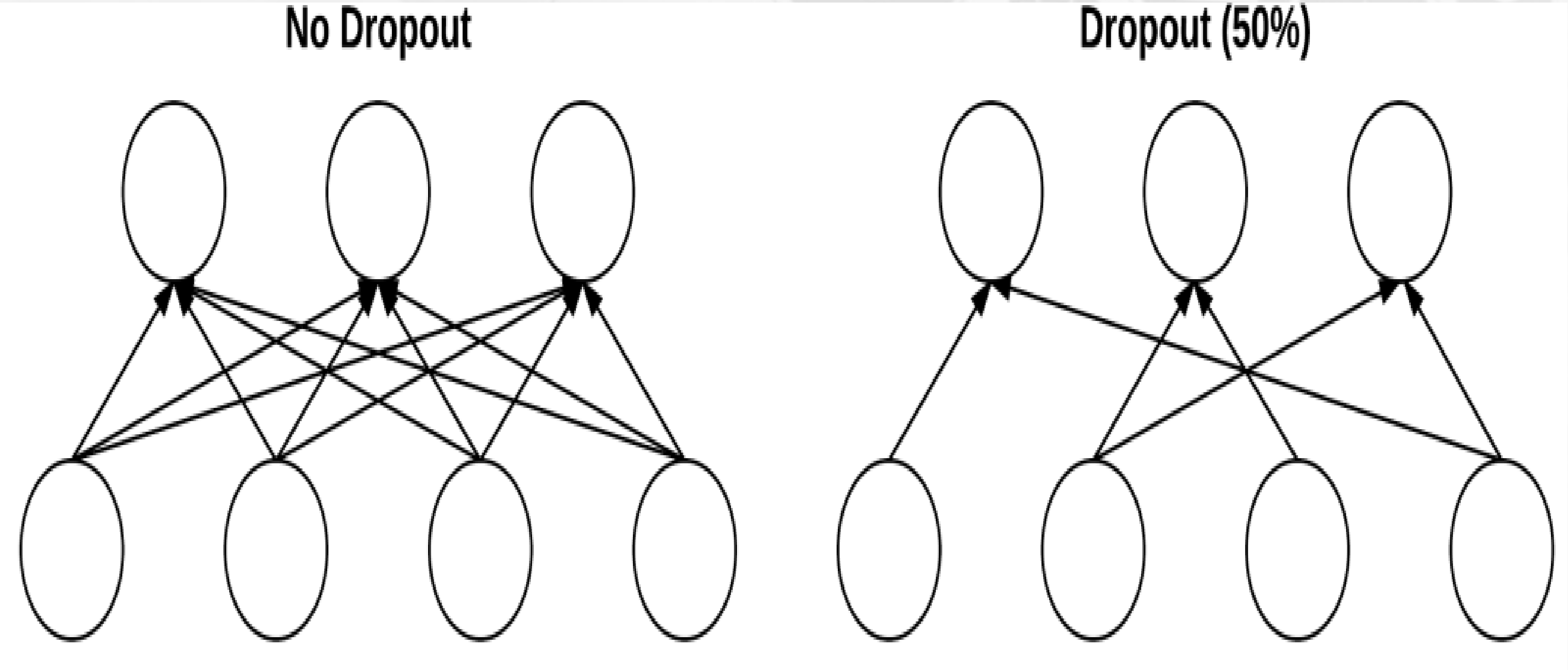
- ✓ Perturbing the neural network by randomly, on each training step, selecting randomly a set of units (neurons) in each layer and prohibiting them from participating.

- ✓ All of the units take a kind of "collective" responsibility for getting the answer right, and will not be able to rely on any small subset of the weights to do all the necessary computation.

- ✓ This tends also to make the network more robust to data perturbations.
  - ✓ aims to help prevent overfitting by increasing testing accuracy perhaps at the expense of training accuracy.
- ✓ Common to set p to 0.5.
- ✓ Experiment with different p_s to get good results on your problem and data.

REGULARIZATION Cont.

- Can apply dropout with smaller probabilities (i.e., p = 0.10 - 0.25) immediately after pooling (subsampling).
- Common to place dropout layers with p = 0.5 in-between FC layers of an architecture



REGULARIZATION Cont.

❑ Batch Normalization

- ✓ Tends to achieve better performance than Dropout regularization.
- ✓ The idea is to standardize the input values for each mini-batch.
- ✓ How?
  - ✓ Subtracting off the mean and dividing by the standard deviation of each input dimension.
- ✓ The scale of the inputs to each layer remains the same, no matter how the weights in previous layers change.
- ✓ NB: the batchwise mean and standard deviation is required to compute batch normalization.
- ✓ Add the BN transform immediately after the nonlinearity.

REGULARIZATION Cont.

❑ Data Augmentation

- ✓ Deep learning models need a large amount of training data in order to learn millions of parameters.
- ✓ Insufficient data inputs for training can contribute to overfitting.
- ✓ Data augmentation is a regularization approach that generates additional samples of the original training data.
- ✓ Augmentation can extract more information from the original training data and improve the performance of the deep learning network.
- ✓ Noise can be added to the original training inputs to generate new training samples.
- ✓ Assignment; Try and explore several data augmentation schemes.

Assignment 1

- ❑ **Install all the listed Python packages using Anaconda distribution.**
- ❑ **Download the following datasets using the link**
  - ✓ https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv
  - ✓ http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data
- ❑ **Mention and explain**
  - ✓ 4 **types of activation functions.**
  - ✓ 3 **types of backpropagation algorithms**
- ❑ **Explain the following DL concepts:**
  - ✓ **Shallow classifier**
  - ✓ **Deep neural networks**
  - ✓ **Forward pass**
  - ✓ **Backpropagation**

Assignment 2

- Explain the difference between Batches and Adaptive step-size as training strategies of DNN?
- Mention and explain 4 adaptive step size optimizers?
- For what value of k is mini-batch gradient decent equivalent to
  - ✓ stochastic gradient descent?
  - ✓ Batch gradient descent?
- Discuss the following loss functions?
  - ✓ Squared loss
  - ✓ Hinge loss
  - ✓ NLL
  - ✓ NLLM

Assignment 3

- ❑ **Practice.**
- ❑ **Mention and explain briefly**
  - ✓ **5 types of DL optimization algorithms.**
- ❑ **Explain the following DNN concepts:**
  - ✓ **Weight initialization**
  - ✓ **5 weight initialization methods**
  - ✓ **Regularization**
  - ✓ **Dropout**

谢谢

Thank you