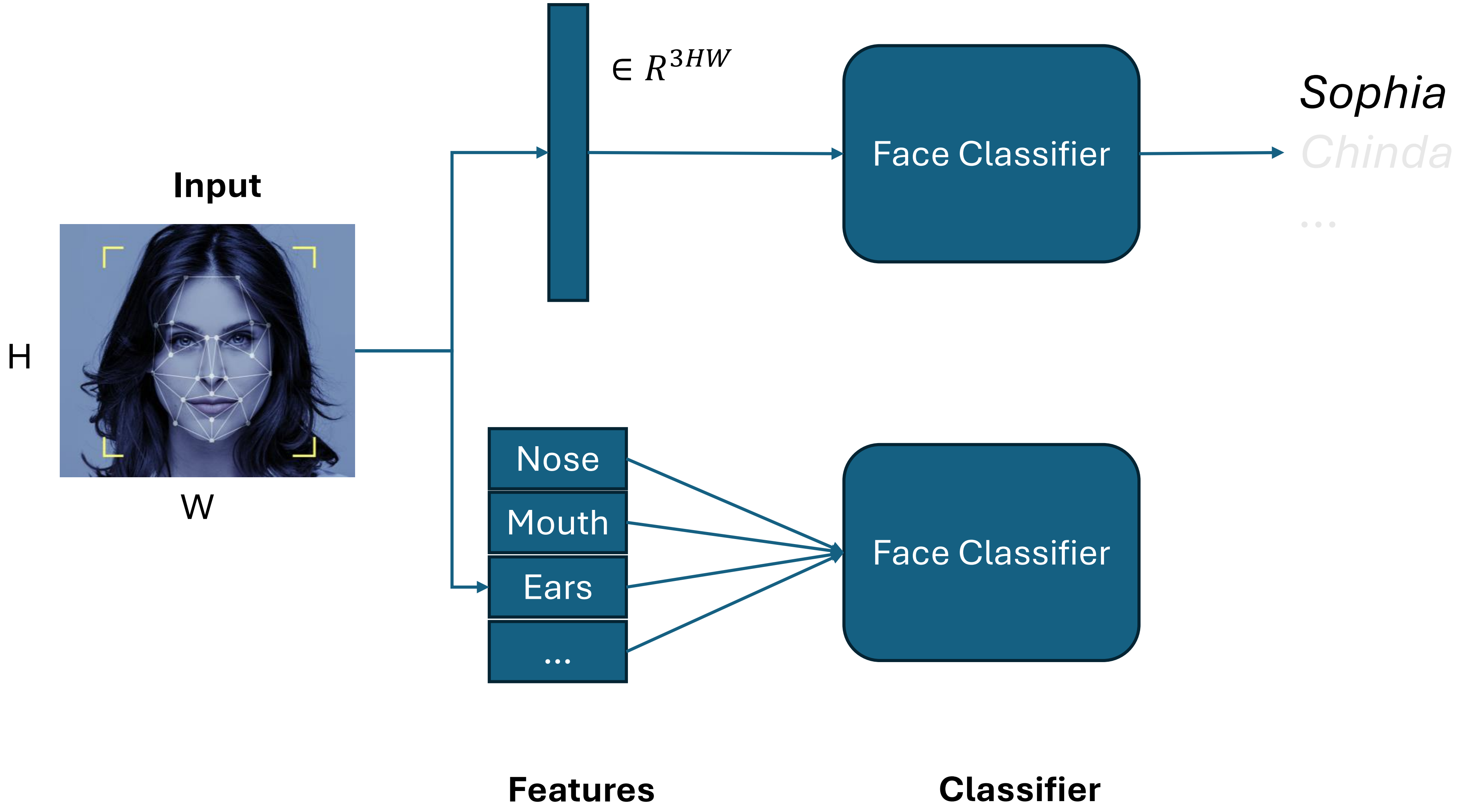
Recap - Intro. to Deep Learning

Rina BUOY

American University of Phnom Penh

Study Locally. Live Globally.

Why Deep Learning ?

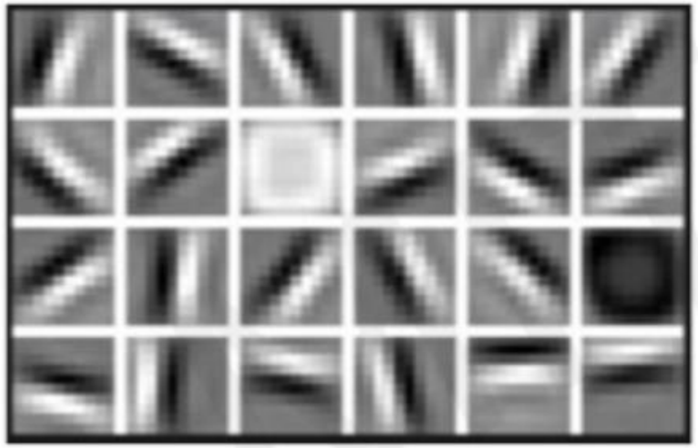


Why Deep Learning ?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

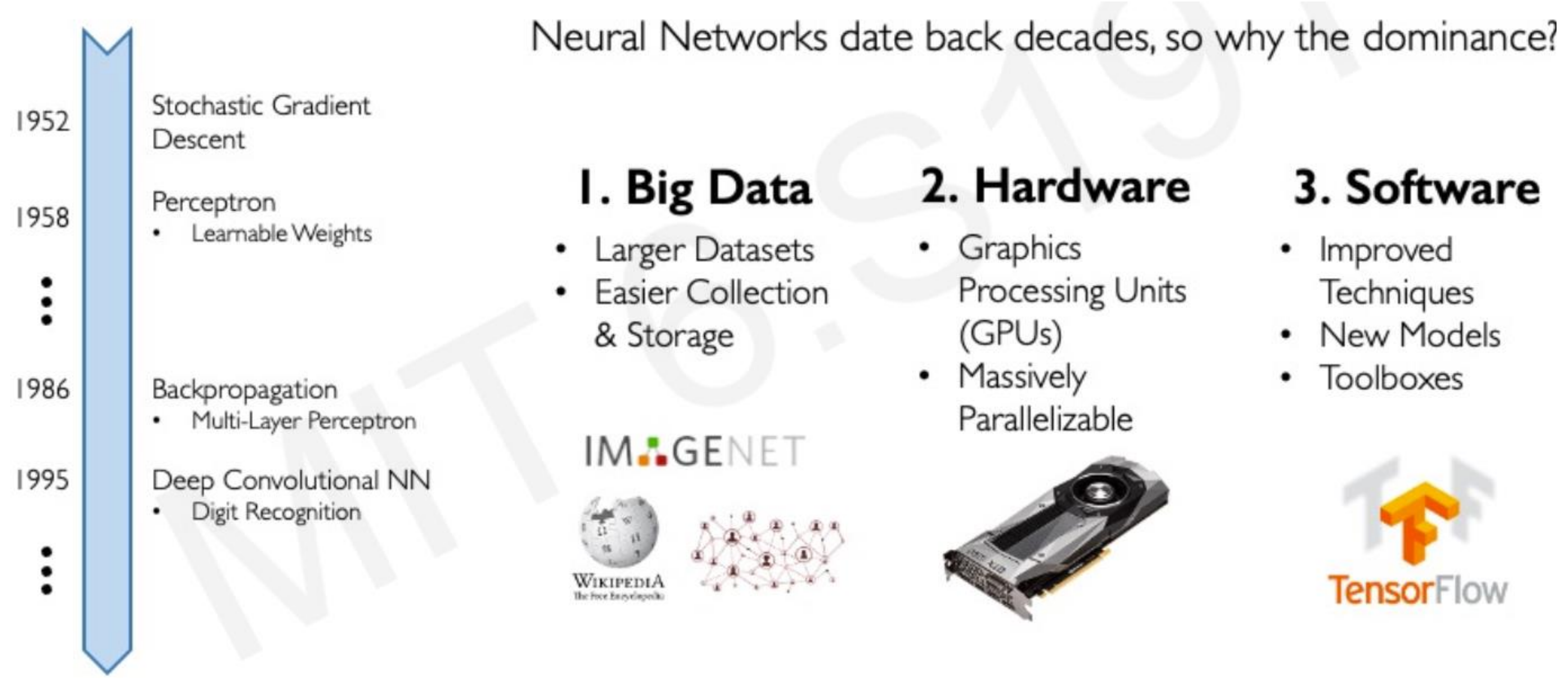
Mid Level Features

Eyes & Nose & Ears

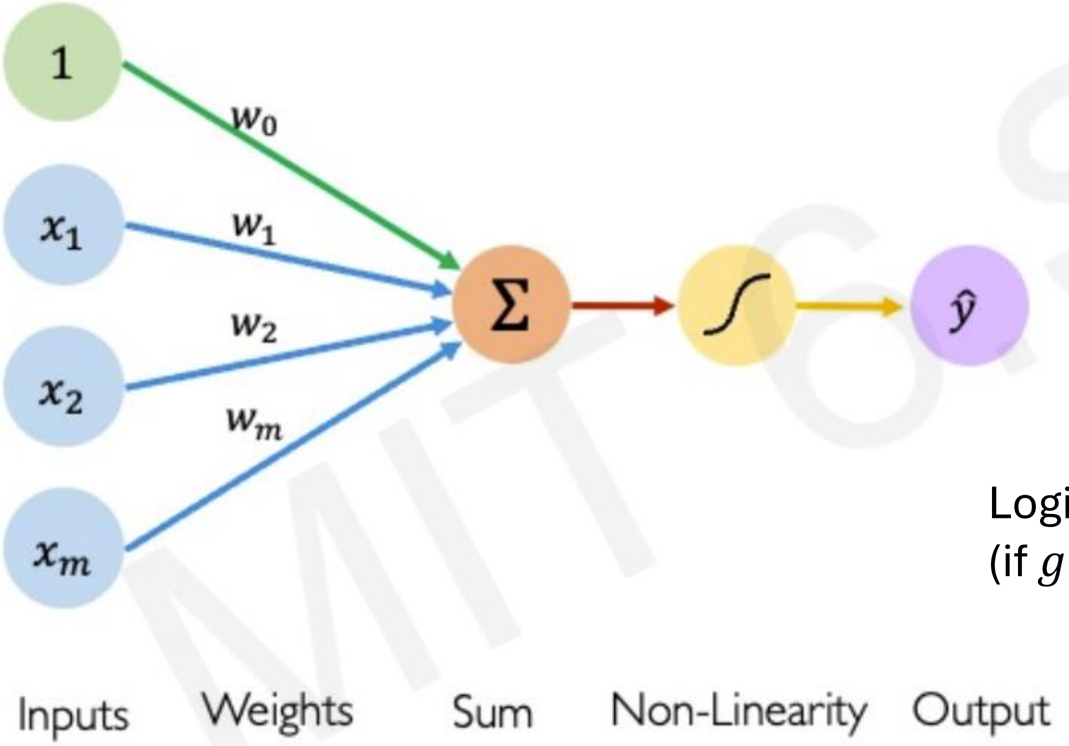
High Level Features

Facial Structure

Why Now ?


Neural Networks date back decades, so why the dominance?
1952
Stochastic Gradient Descent
1958
Perceptron
• Learnable Weights
1986
Backpropagation
• Multi-Layer Perceptron
1995
Deep Convolutional NN
• Digit Recognition
1. Big Data
• Larger Datasets
• Easier Collection & Storage
IMAGENET
WIKIPEDIA
2. Hardware
• Graphics Processing Units (GPUs)
• Massively Parallelizable
3. Software
• Improved Techniques
• New Models
• Toolboxes
TensorFlow

The Perceptron – A Neuron



Linear Regression

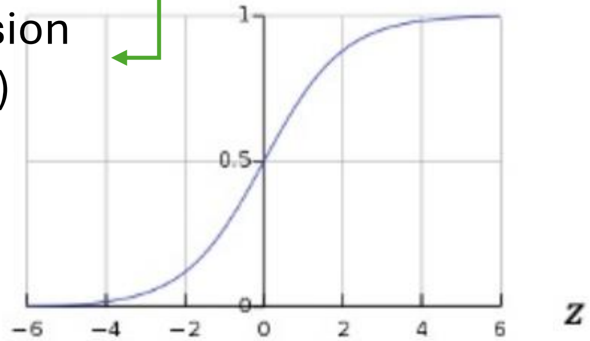
Activation Functions

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

- Example: sigmoid function

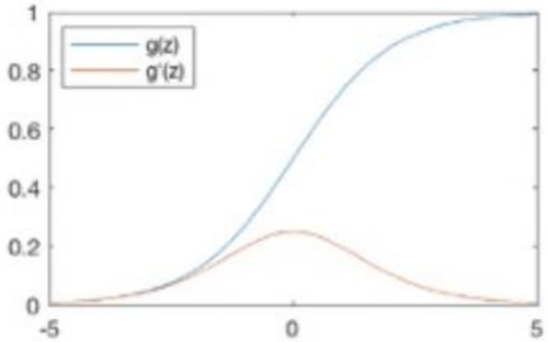
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

Logistic Regression
(if $g()$ is sigmoid)



Beyond Sigmoid - Activation Functions

Sigmoid Function

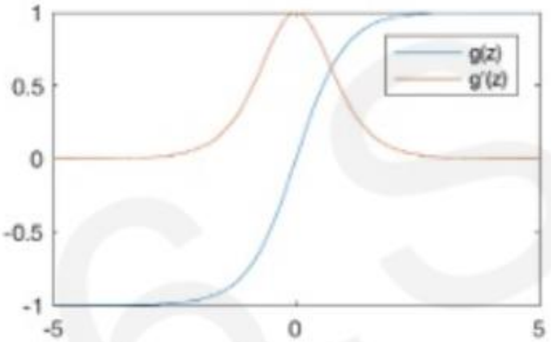


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

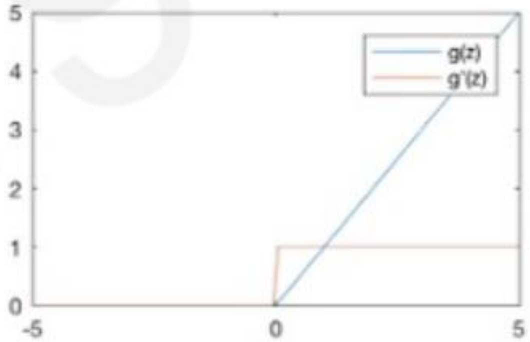


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)



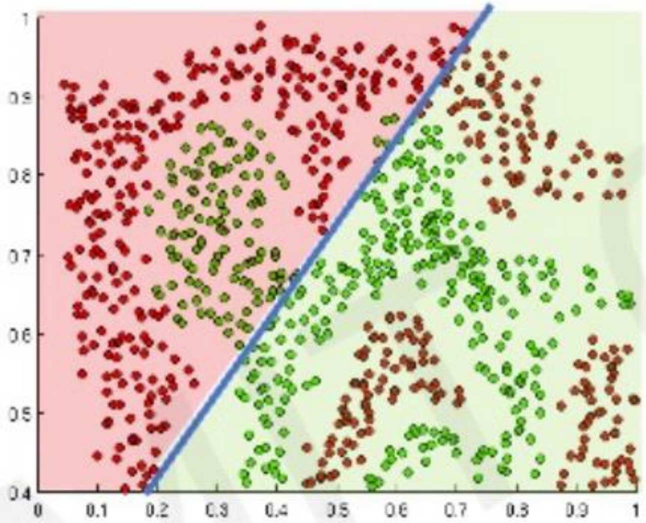
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

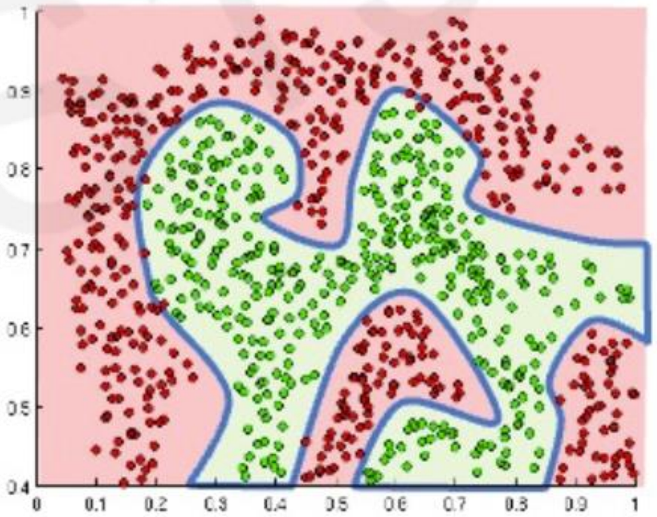
```
tf.nn.relu(z)
```

Activation Functions

The purpose of activation functions is to ***introduce non-linearities*** *into the network*



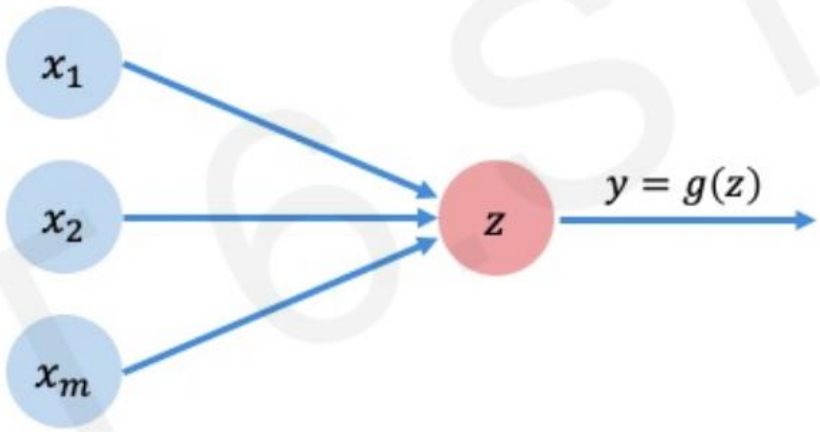
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

From Perceptron to Neural Network

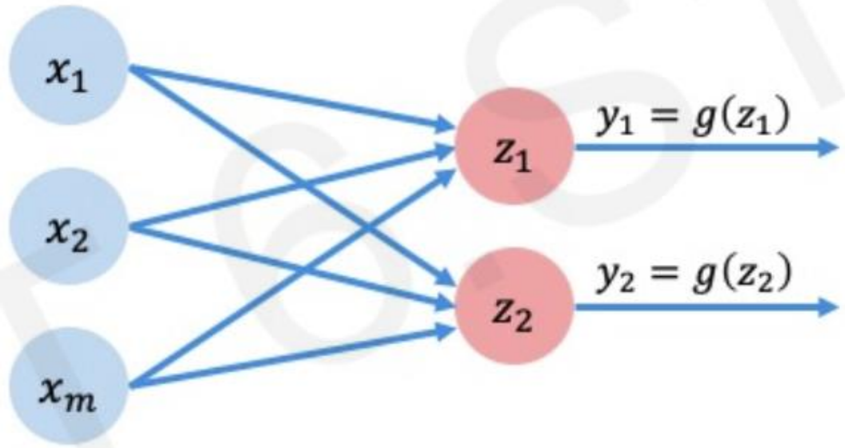
The Perceptron: Compact Notation



$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$

Beyond One Output – Multivariate Case

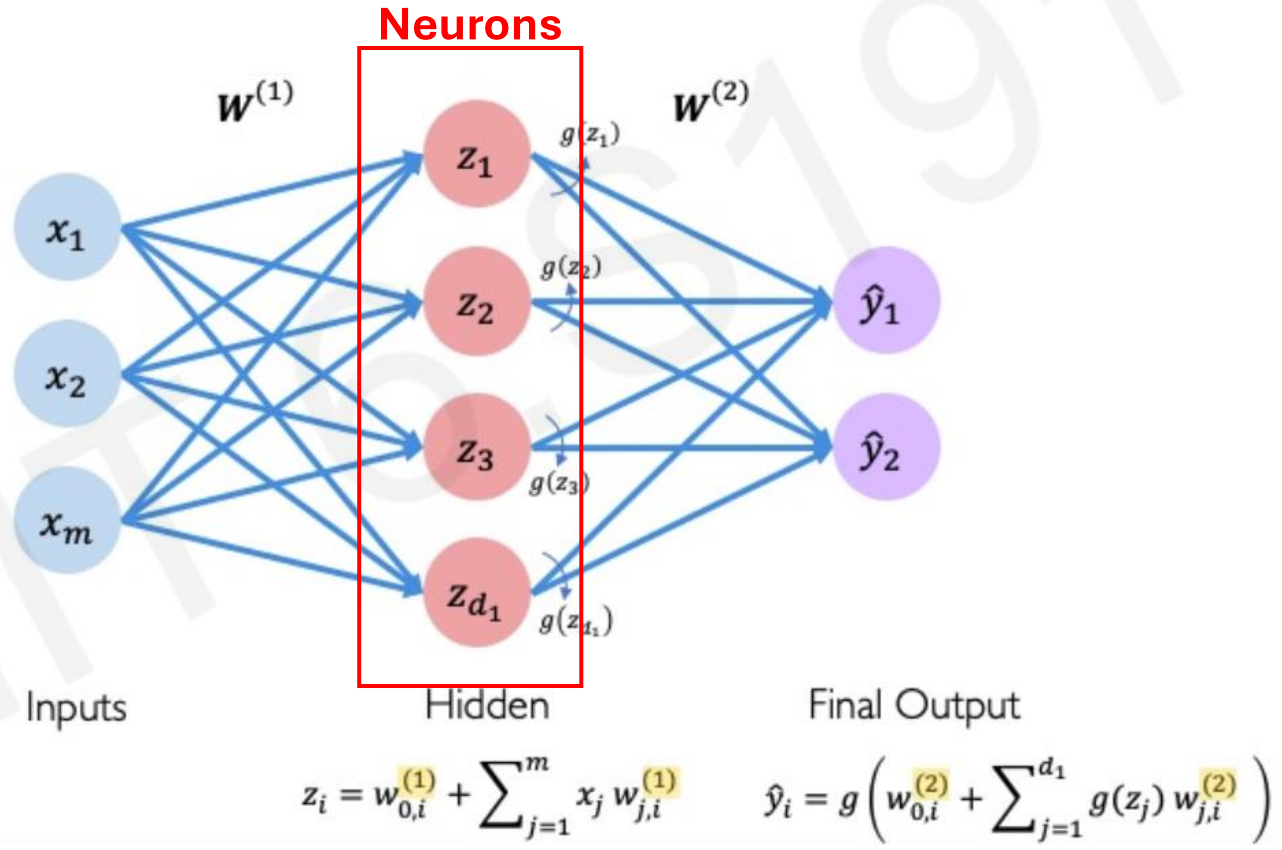
Because all inputs are densely connected to all outputs, these layers are called **Dense** layers



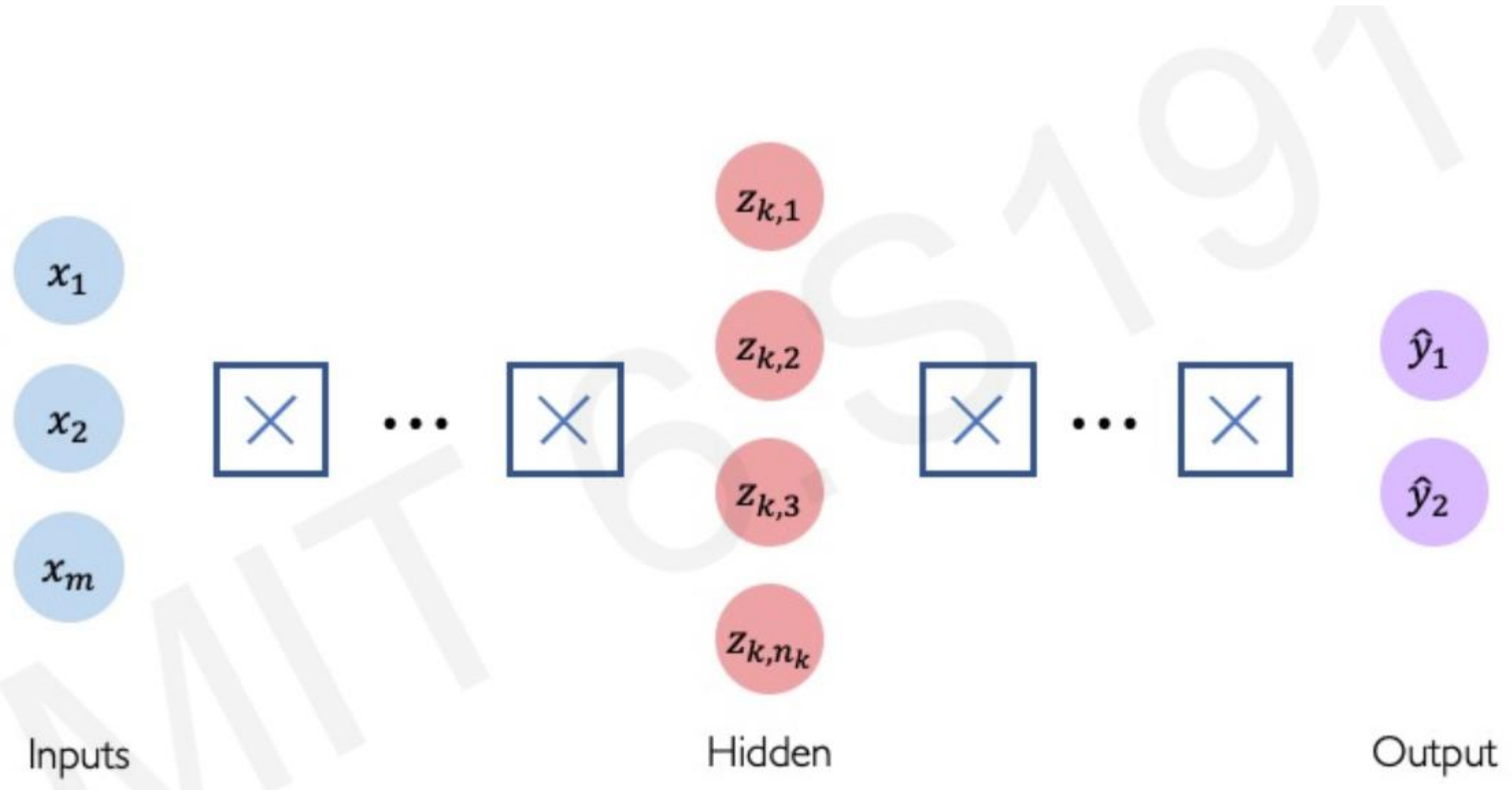
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Now, Neural Network

Called a single layer neural network because there is only one layer of hidden neurons.

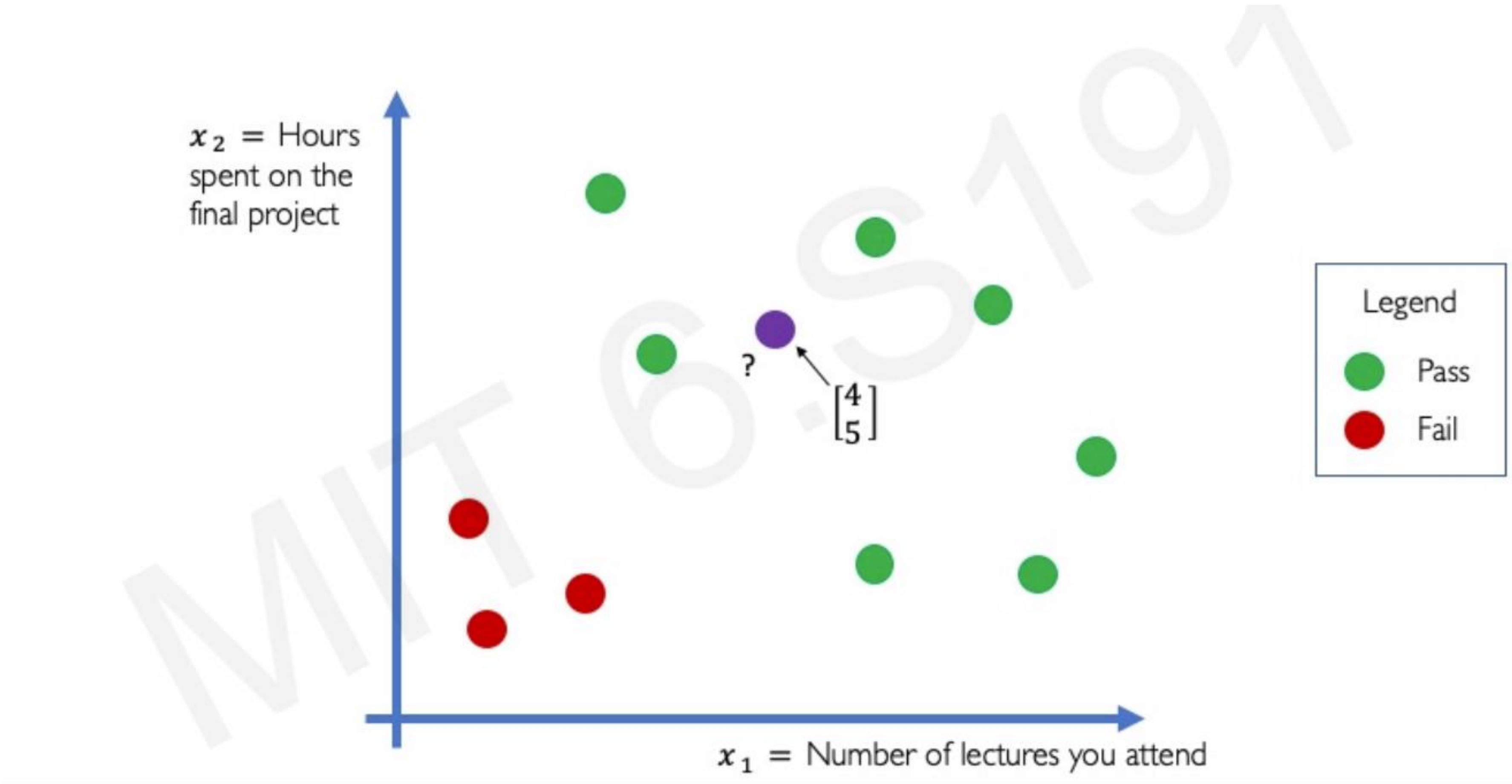


Beyond Single Layer - Deep Neural Network

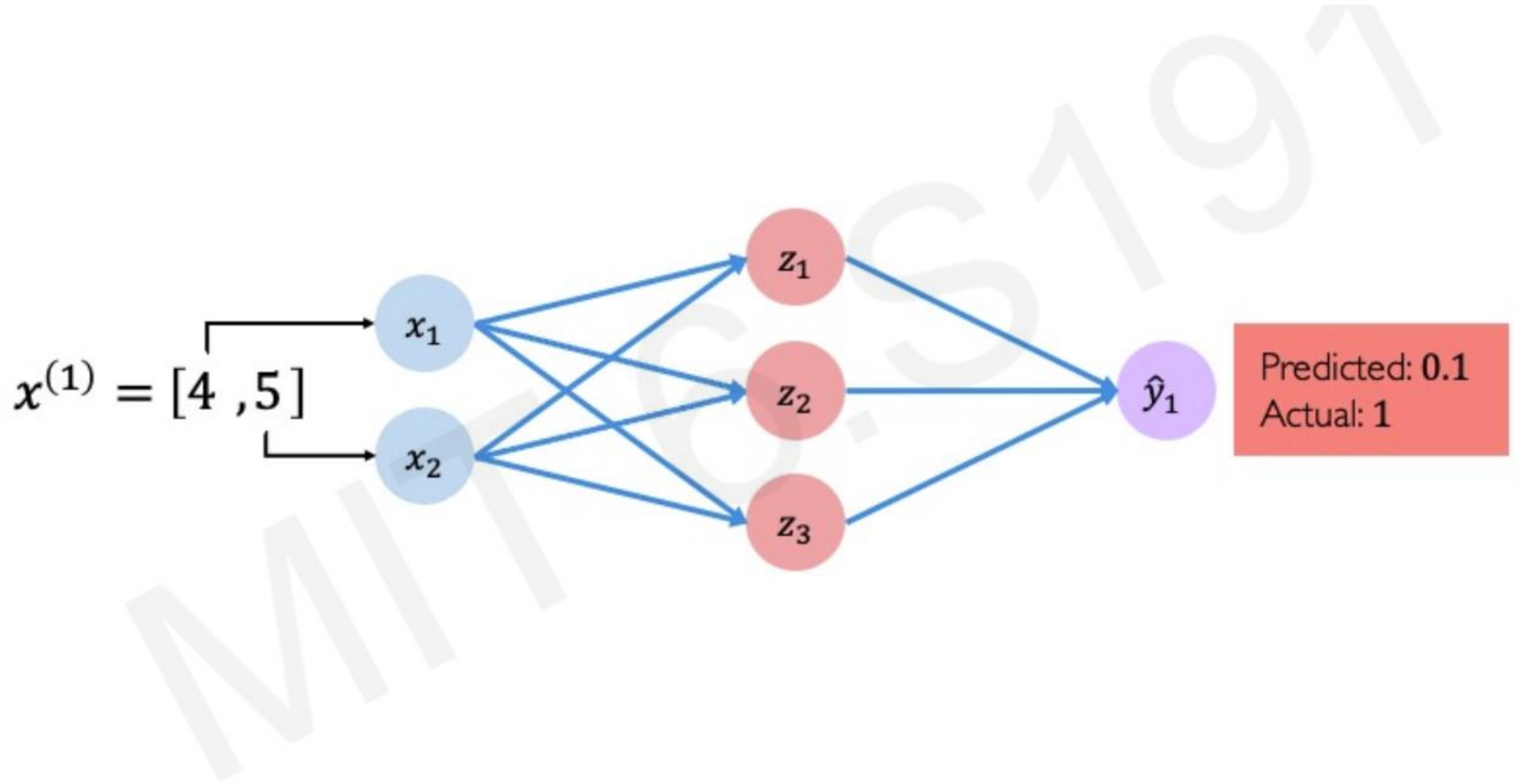


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Example Problem



Example Problem



Model Training

We want to find the network weights that ***achieve the lowest loss***

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

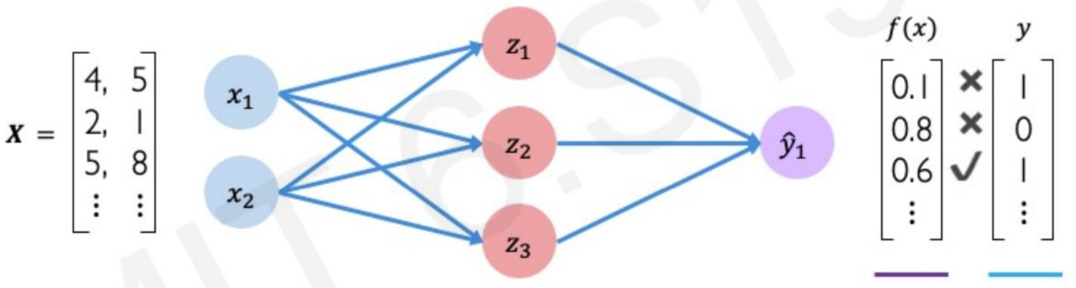
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\right\} \Rightarrow \begin{bmatrix} w_1 \\ \dots \\ w_m \end{bmatrix}$$

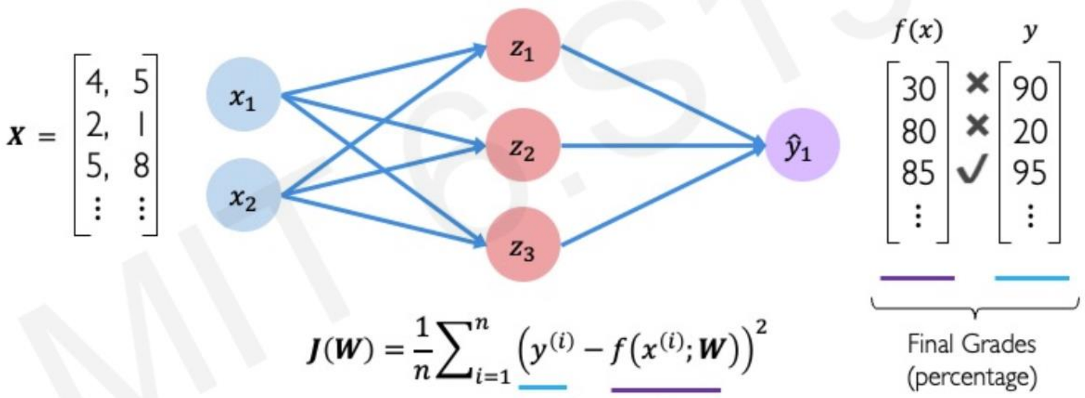
Loss Functions & Training

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



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

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



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

Algorithm

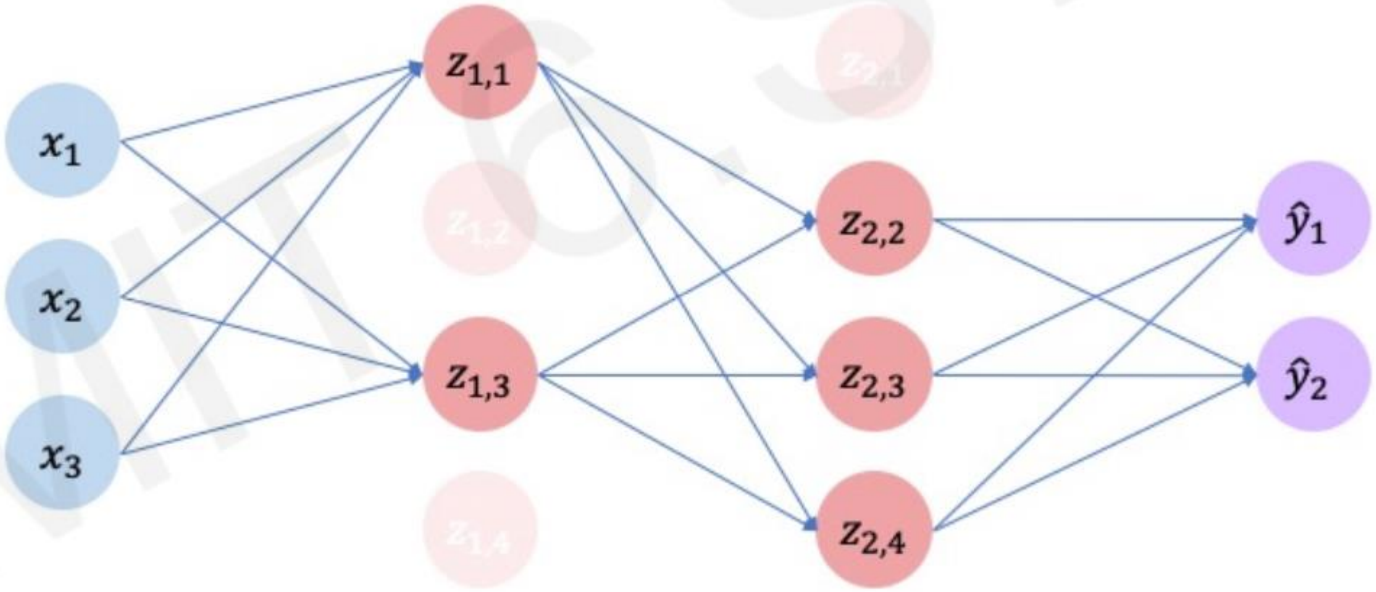
1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence:
3. $\boxed{\text{Compute gradient, } \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Practically, compute over a mini-batch.

Dropout

- During training, randomly set some activations to 0
  - Typically 'drop' 50% of activations in layer
  - Forces network to not rely on any 1 node

```
tf.keras.layers.Dropout(p=0.5)
```



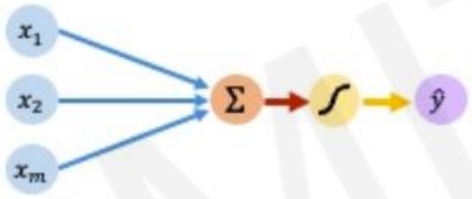
Early Stopping

- Stop training before we have a chance to overfit

Summary

The Perceptron

- Structural building blocks
- Nonlinear activation functions



Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation

Training in Practice

- Adaptive learning
- Batching
- Regularization