SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Introduction to Neural Networks I

PROF LIM KWAN HUI

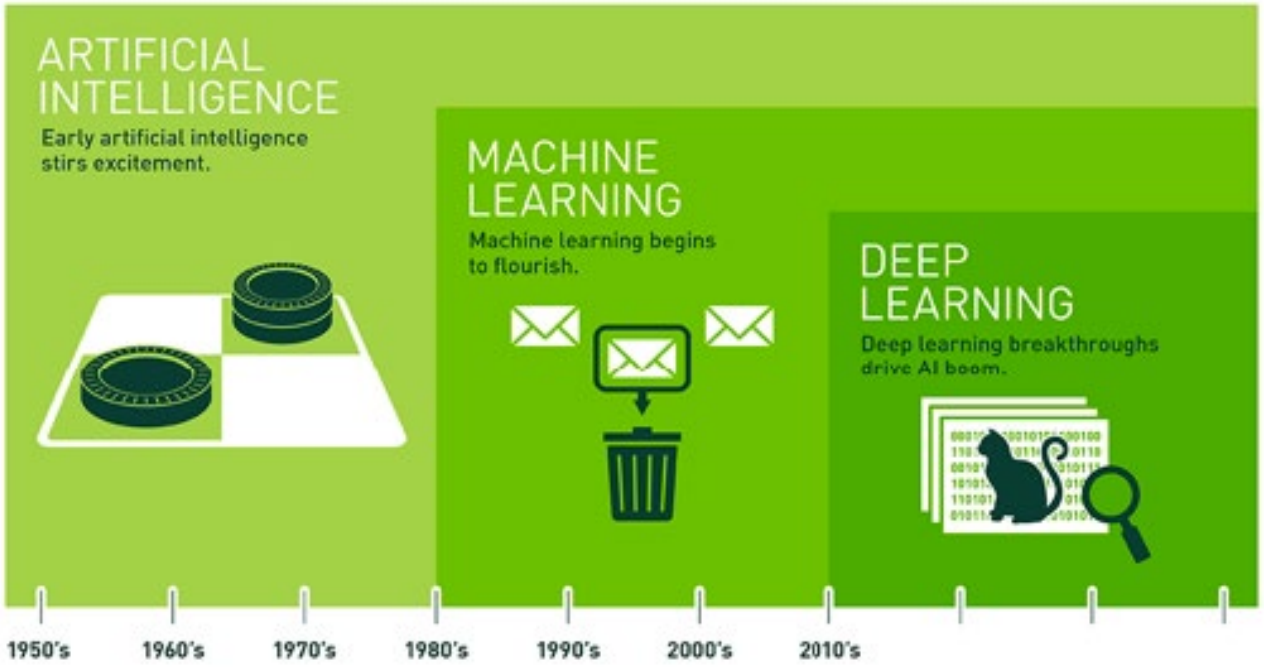
50.021 Artificial Intelligence

The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.

Outline & Objectives

- Understand how a perceptron works in the context of neural networks
- Understand how to train and use a neural network, including the general intuition behind activation functions, losses and optimizers
- Be aware of techniques to prevent overfitting in neural networks
- Be able to use a simple neural network to perform a certain task, e.g., compute the output for a prediction

Machine Learning



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Machine Learning

- "Learning is any process by which a system improves performance from experience." - Herbert Simon

- Definition by Tom Mitchell (1998):

  Machine Learning is the study of algorithms that:
  - Improve their performance P
  - At some task T
  - With experience E.

  A well-defined learning task is given by <P, T, E>.

ML is used when

- Human expertise does not exist (navigating on Mars)
- Humans can't explain their expertise (speech recognition)
- Models must be customized (personalized medicine)
- Models are based on huge amounts of data (genomics)
- Learning isn't always useful:
  - There is no need to "learn" to calculate payroll

3 components:

- *Representation*
- *Optimization*
- *Evaluation*

Types of learning

- Supervised (inductive) learning
  - Given: training data + desired outputs (labels)
- Unsupervised learning
  - Given: training data (without desired outputs)
- Semi-supervised learning
  - Given: training data a few desired outputs
- Reinforcement learning
  - Rewards from sequence of actions
- New: self-supervised learning: an elegant subset of unsupervised learning where you can generate output labels 'intrinsically' from data objects by exposing a relation between parts of the object, or different views of the object.

Why Deep Learning?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

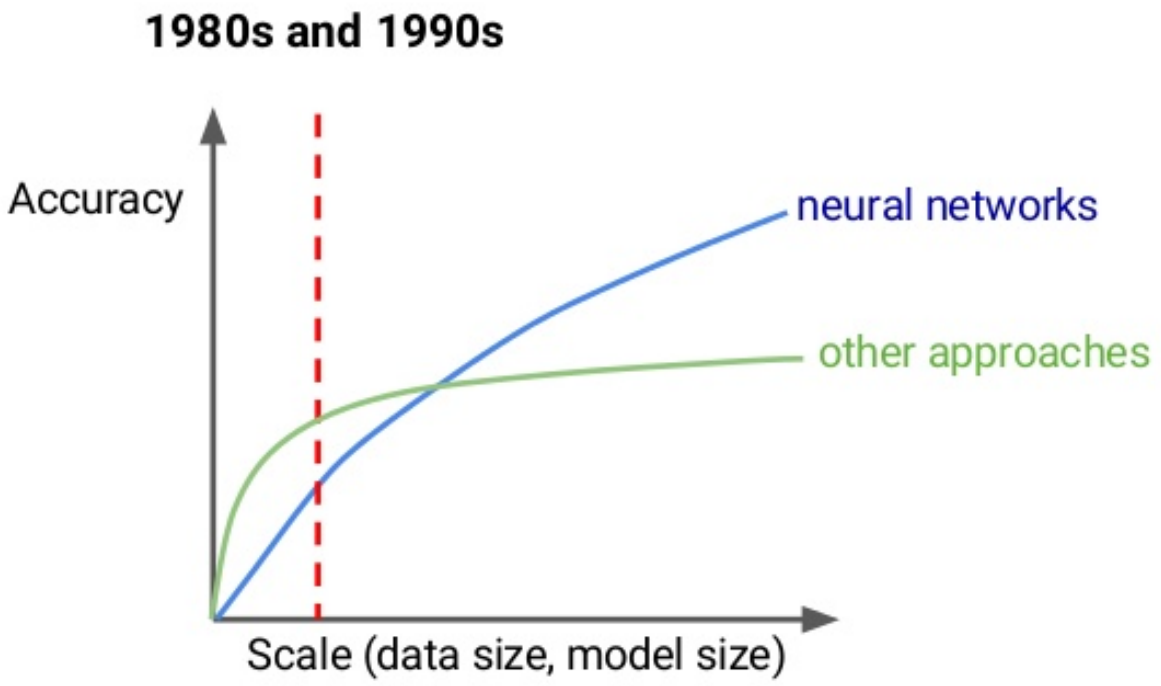
WIKIPEDIA The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

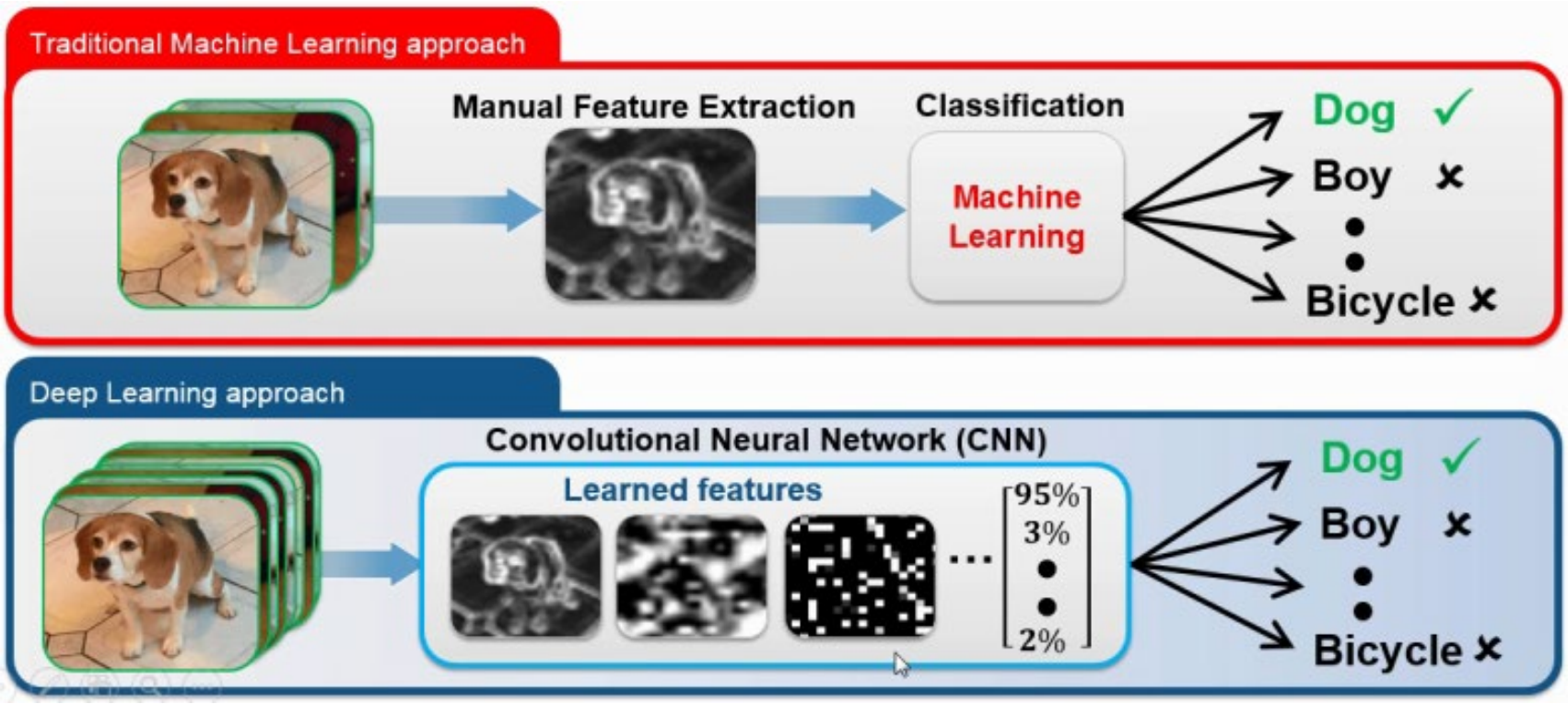
3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

Why Deep Learning?

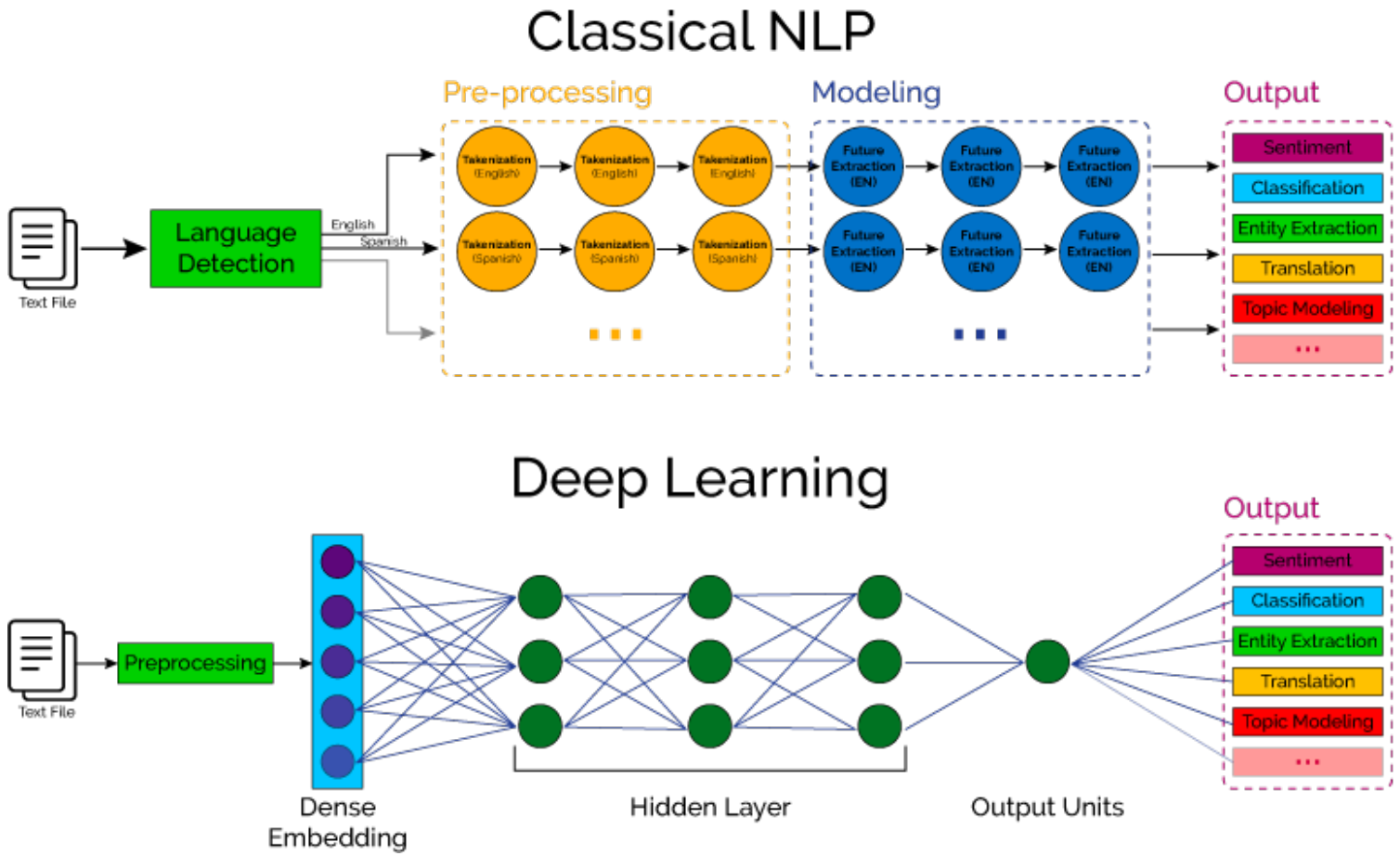


Differences

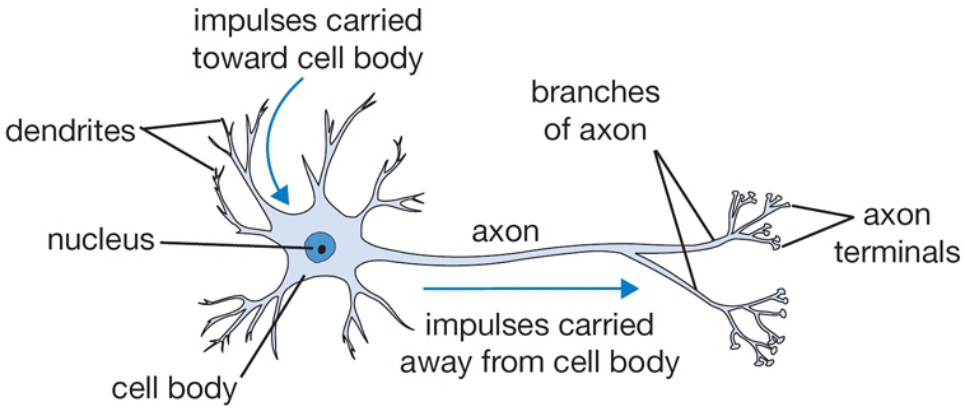


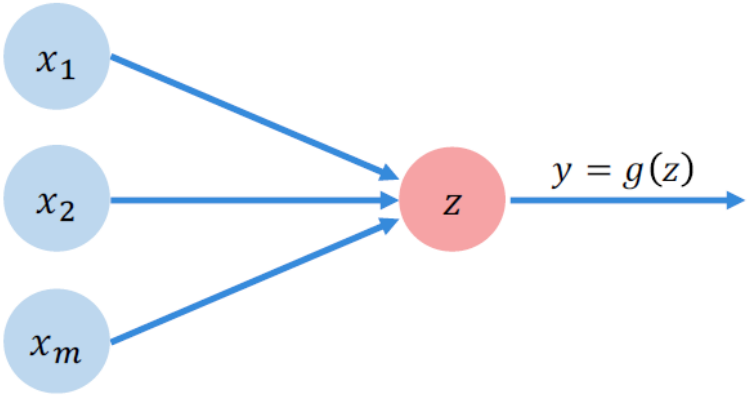
https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/deep_learning.html

Differences



Inspiration behind Neural Networks



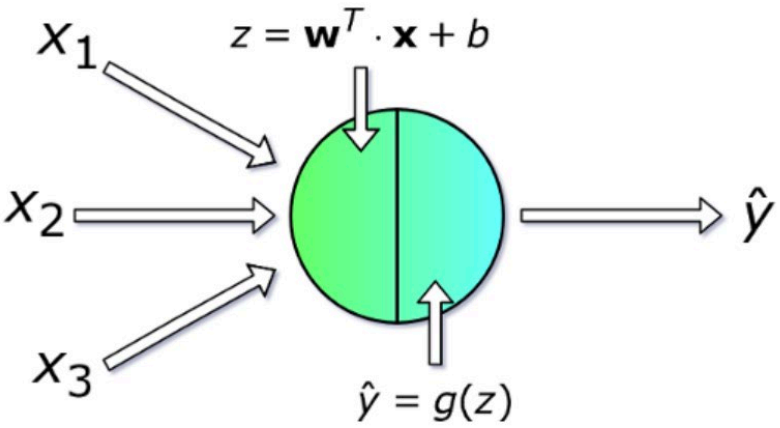


http://cs231n.github.io/neural-networks-1/

Single Perceptron

- Invented by Frank Rosenblatt, the perceptron was originally intended to be a custom-built mechanical hardware instead of a software function. The Mark 1 perceptron was a machine built for image recognition tasks by the US navy.

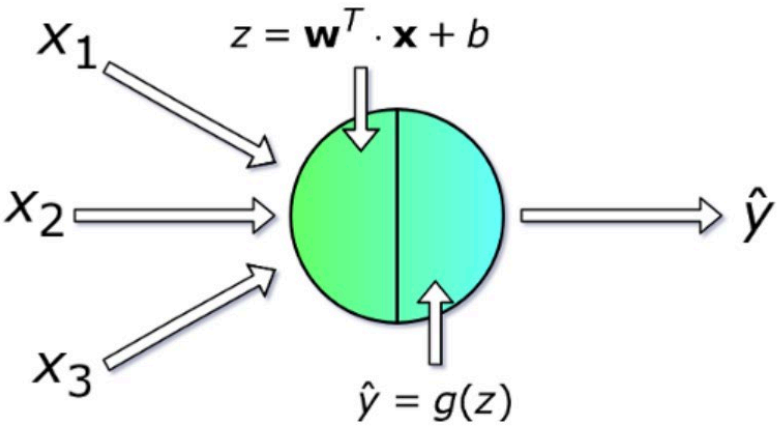
$$z = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n = \mathbf{w}^T \cdot \mathbf{x}$$



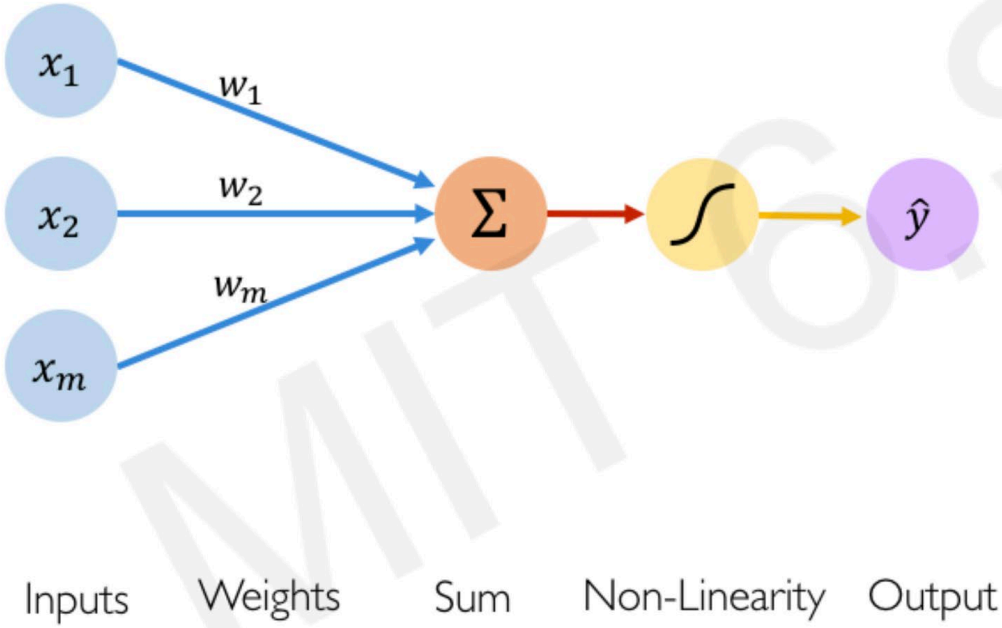
Single Perceptron

- Invented by Frank Rosenblatt, the perceptron was originally intended to be a custom-built mechanical hardware instead of a software function. The Mark 1 perceptron was a machine built for image recognition tasks by the US navy.

$$z = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n = \mathbf{w}^T \cdot \mathbf{x}$$



Single Perceptron



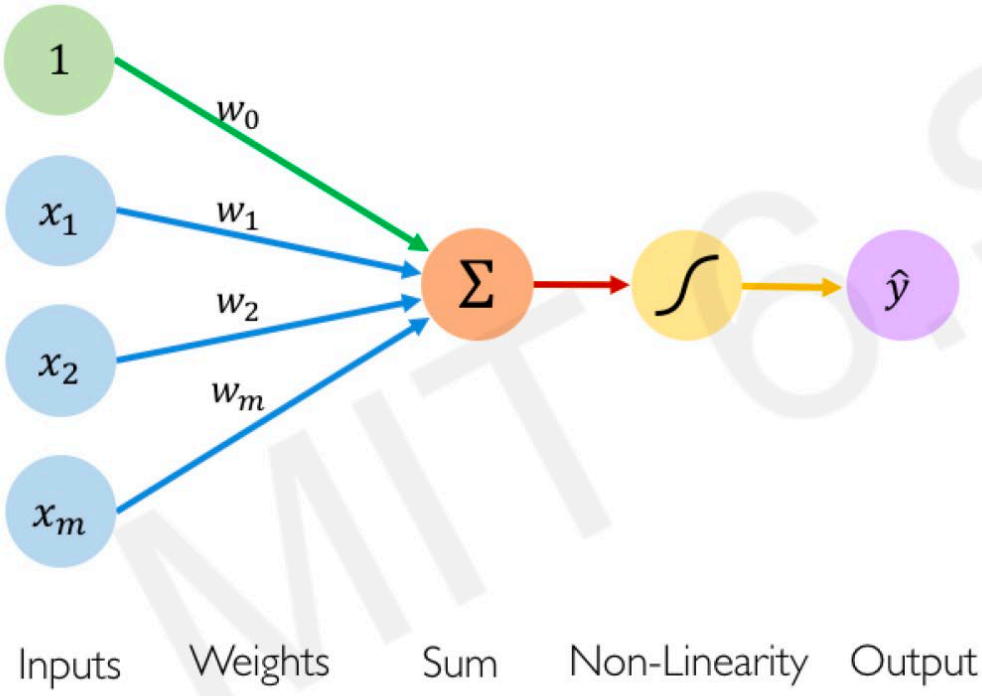
Output

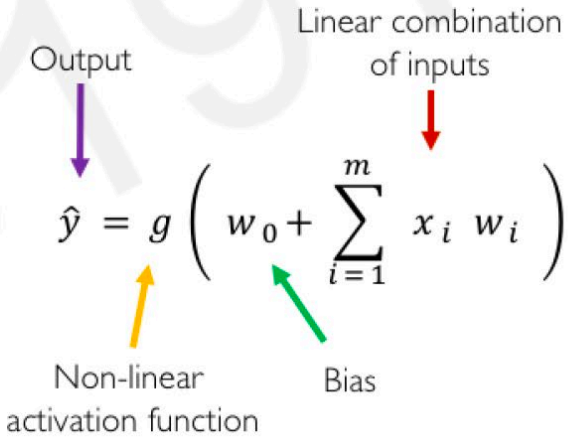
Linear combination of inputs

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

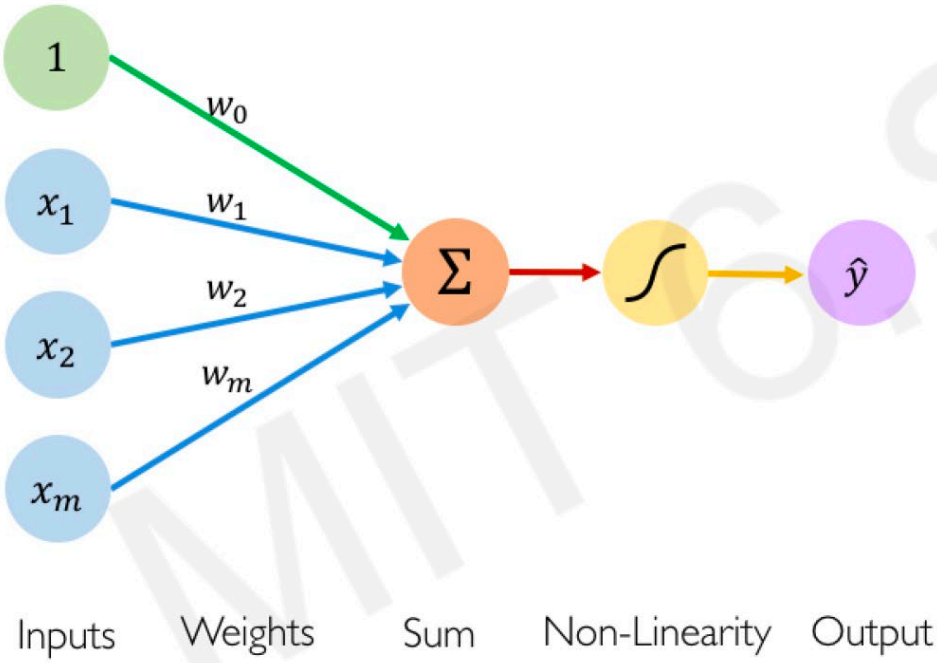
Non-linear activation function

Single Perceptron





Single Perceptron

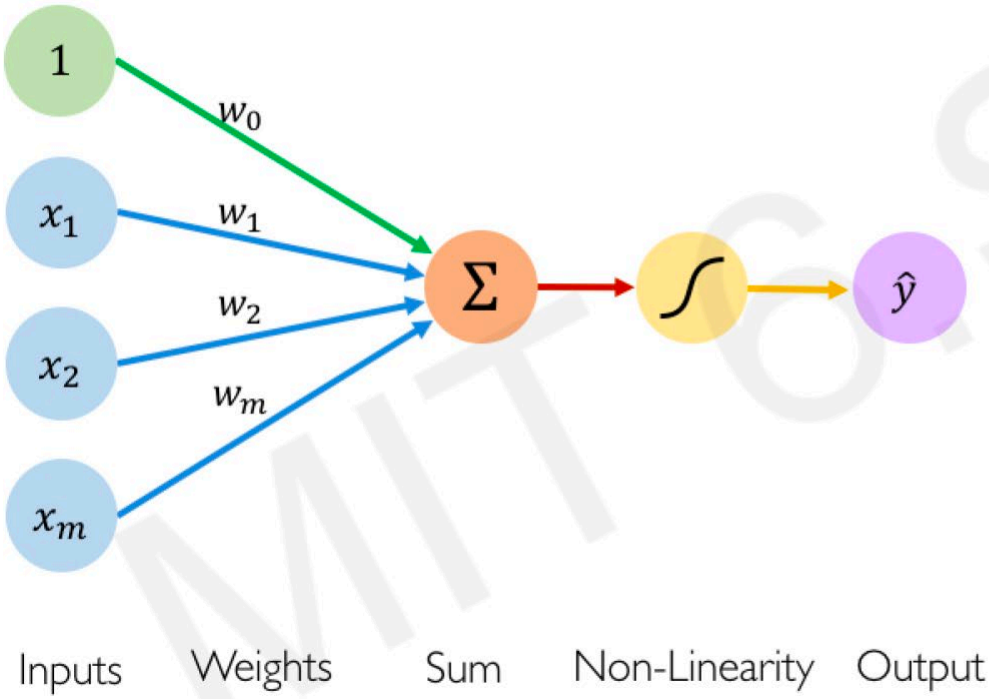


$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

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

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

Single Perceptron

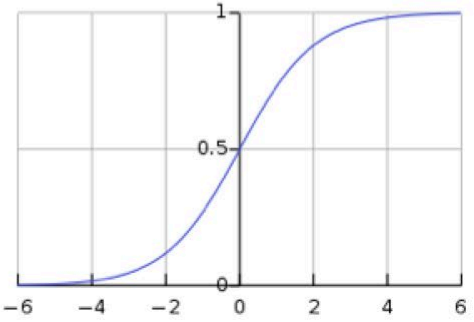


Activation Functions

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

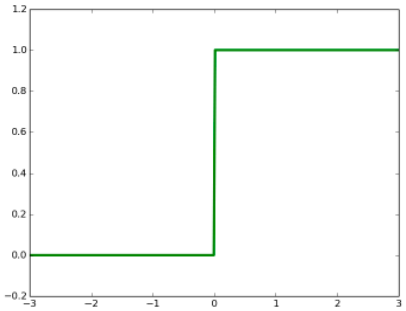
- Example: sigmoid function

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



Common Activation Functions

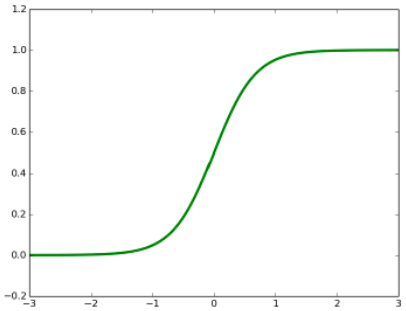
`torch.sigmoid(input)`



Threshold

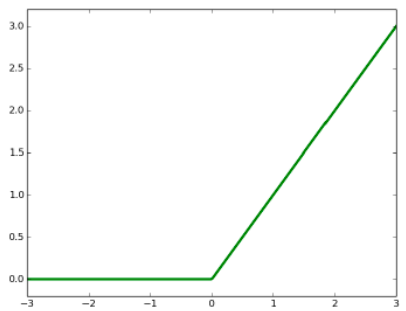
$$g(a) = [a > 0]$$

`torch.threshold(input)`



Sigmoid

$$g(a) = \frac{1}{1 + exp(-a)}$$



ReLU

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

`torch.relu(input)`

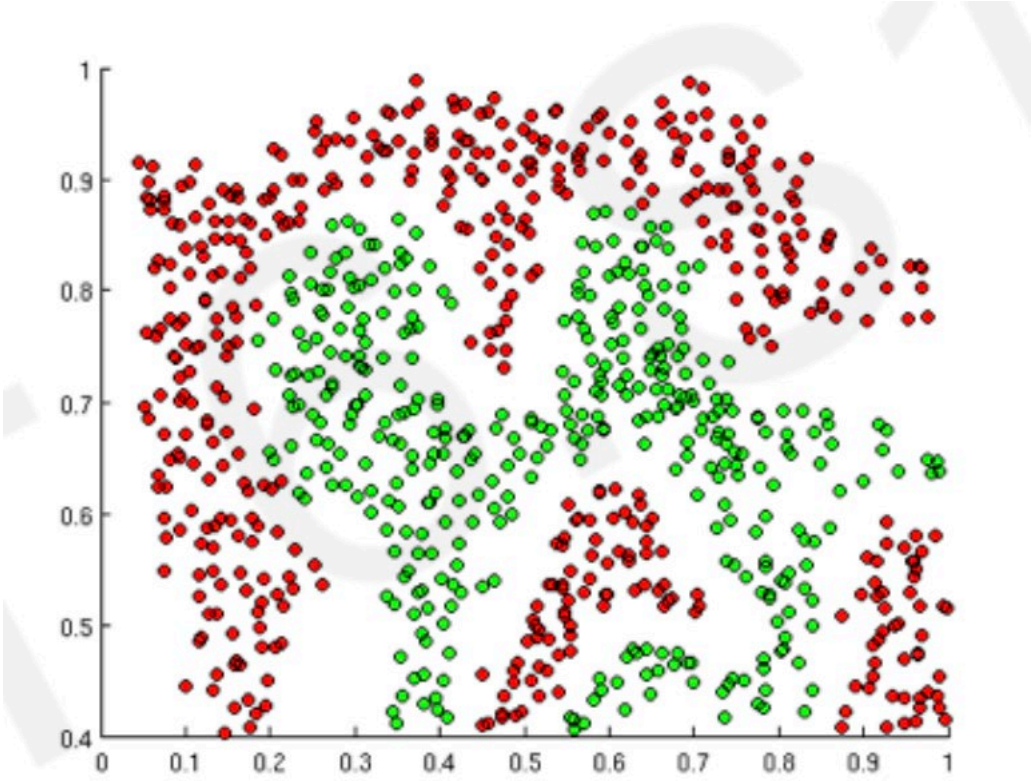
PyTorch documentation: https://pytorch.org/docs/stable/nn.functional.html

Activation Functions and Losses

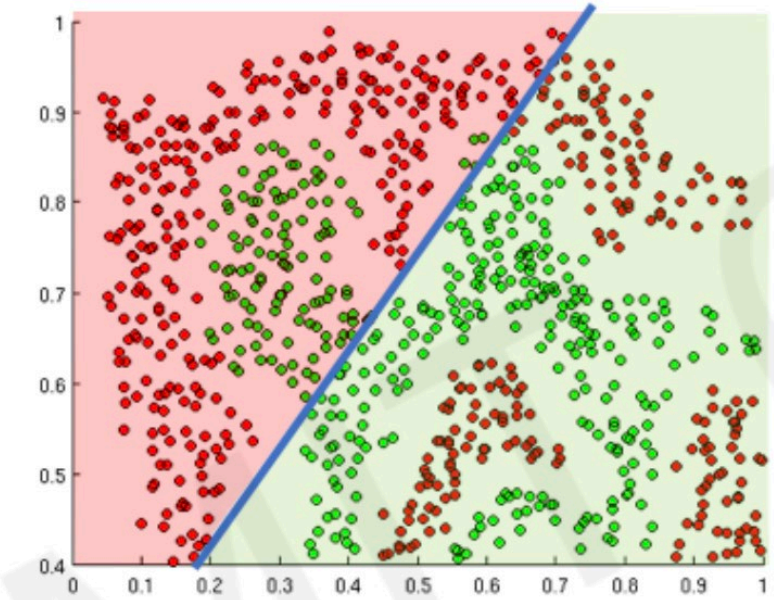
- Regression
  - Linear $(-\infty,+\infty)$ or ReLU $(0,+\infty)$
  - MSE Loss
- Binary classification
  - Sigmoid (0,1): confidence
  - Binary cross entropy loss
- Categorical
  - Softmax (0,1) for each class, total sum = 1
  - Cross entropy loss
- Multi label categorical
  - Sigmoid (0,1): confidence (for every class)
  - Binary cross entropy (for every class)

Importance of activation functions

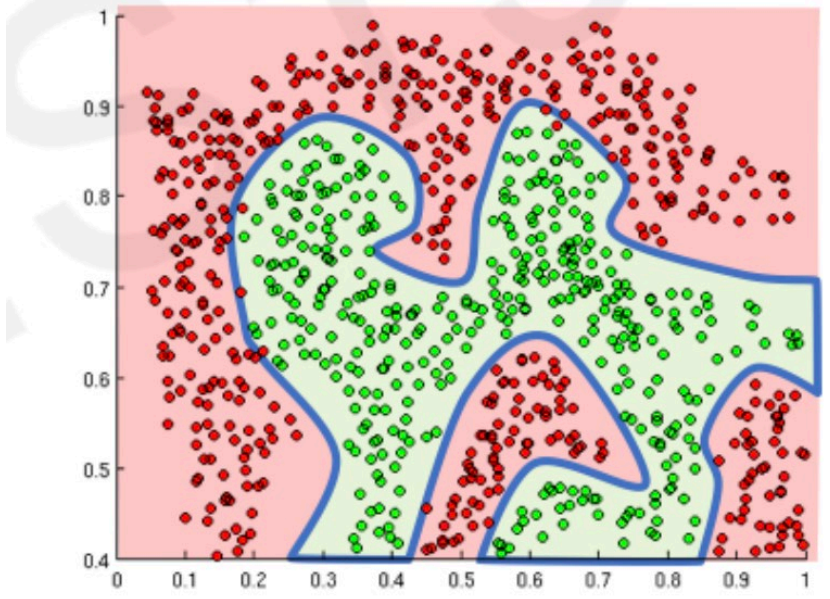
- To introduce non-linearities into the network



Importance of activation functions

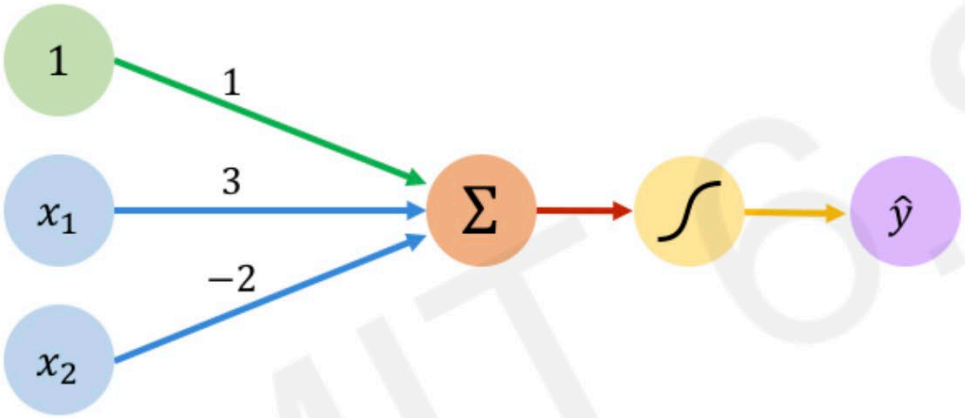


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

The perceptron: example



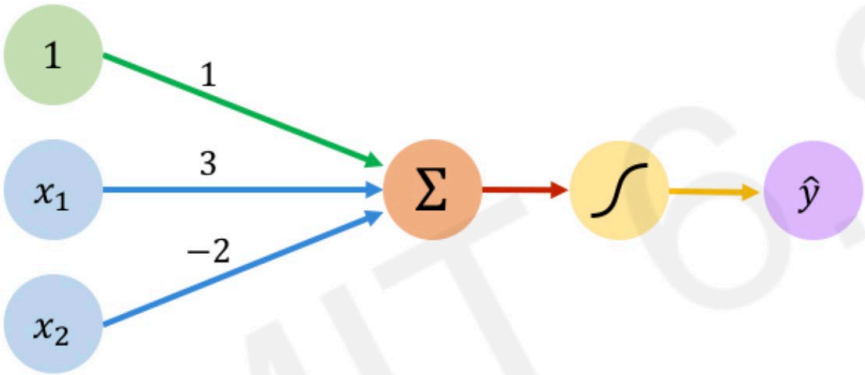
We have: $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

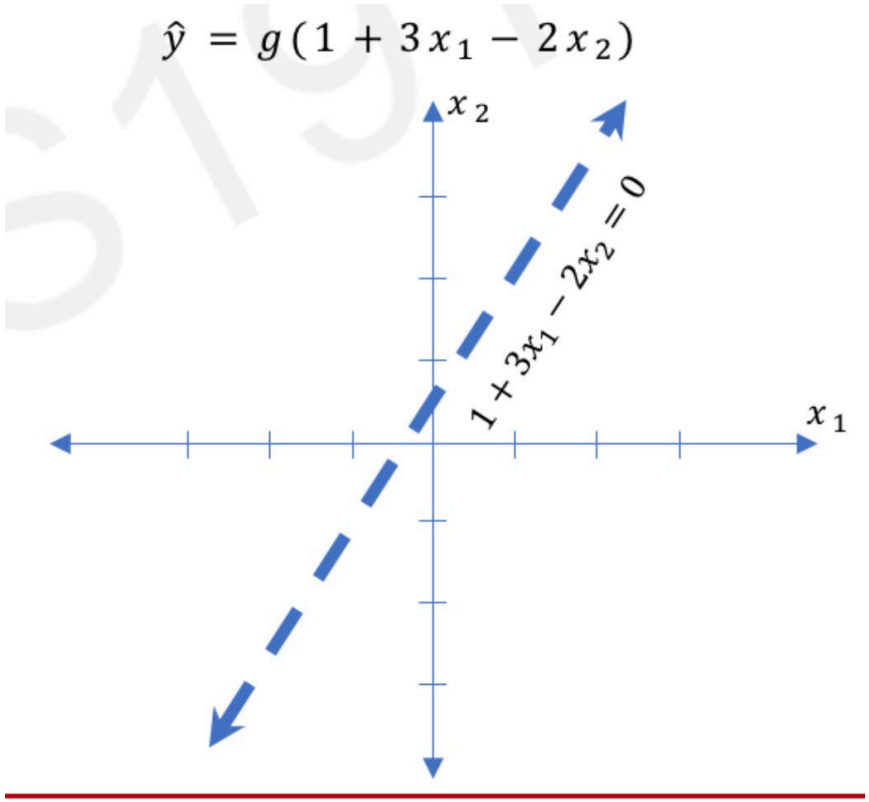
$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

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

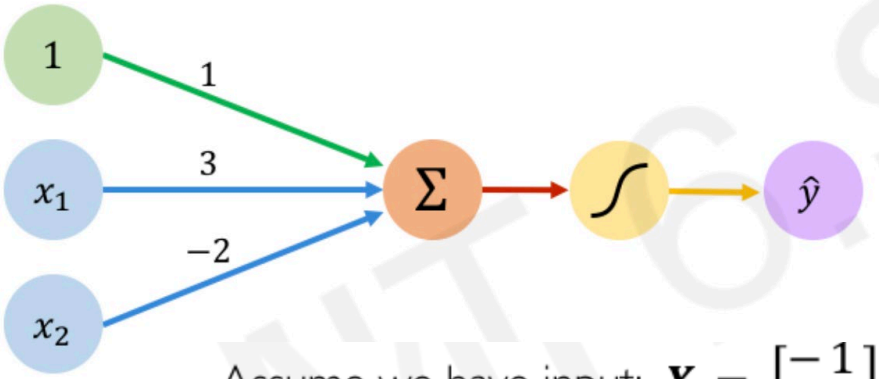
$$\hat{y} = g(\underbrace{1 + 3x_1 - 2x_2}_{\text{This is just a line in 2D!}})$$

The perceptron: example



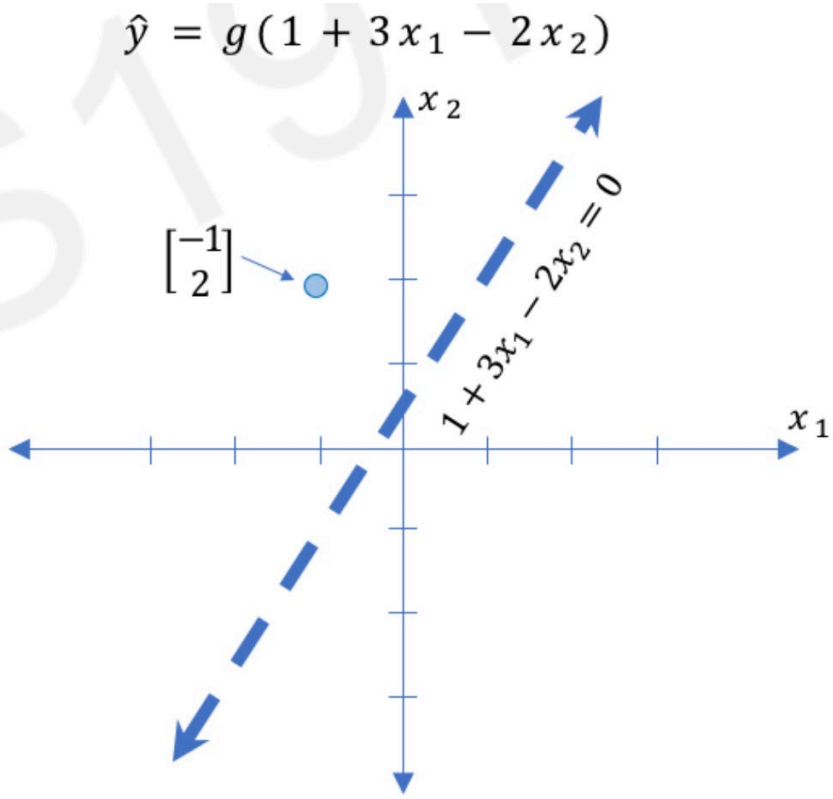


The perceptron: example

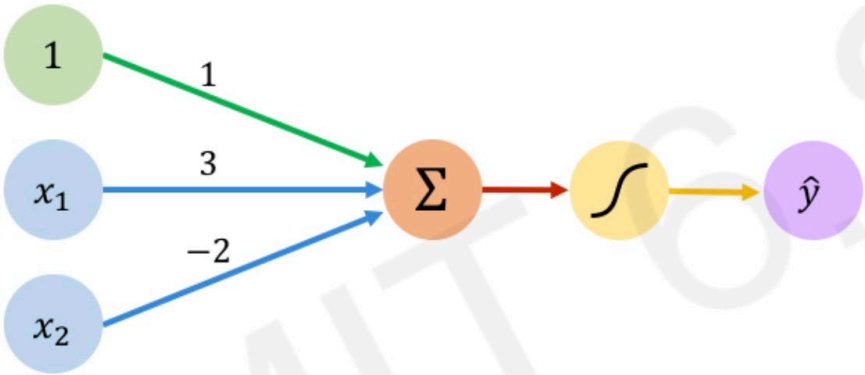


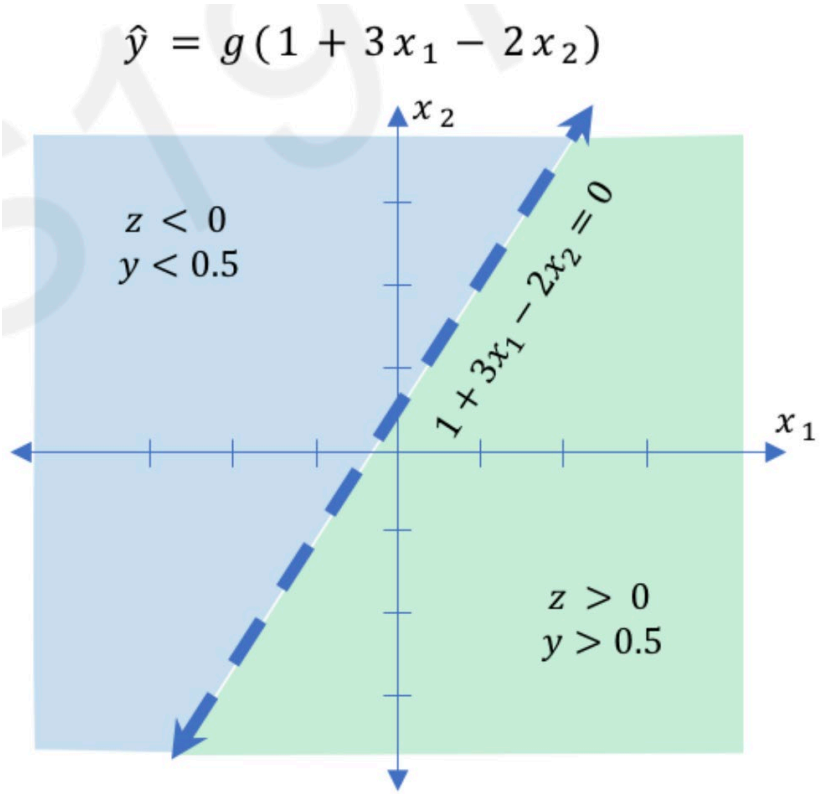
Assume we have input: $\boldsymbol{X} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

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

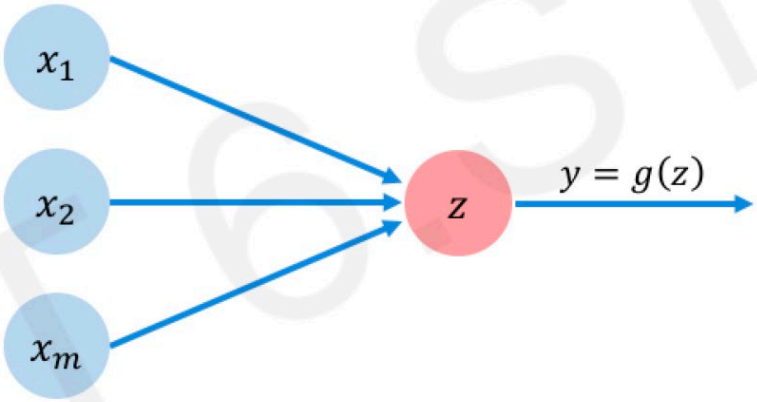


The perceptron: example





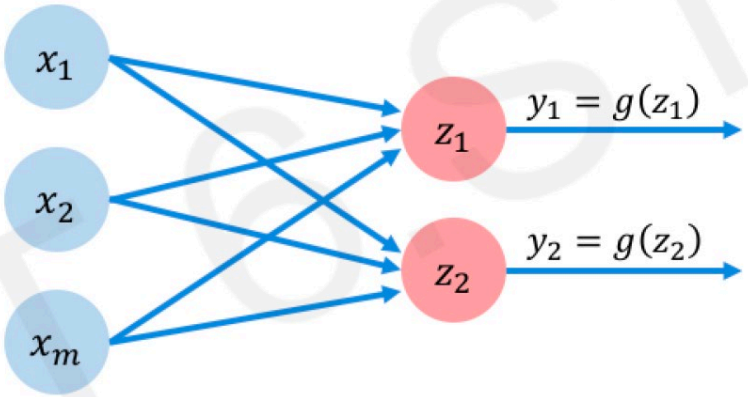
Building NN with Perceptrons



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

Dense layers

- All inputs are densely connected to all outputs

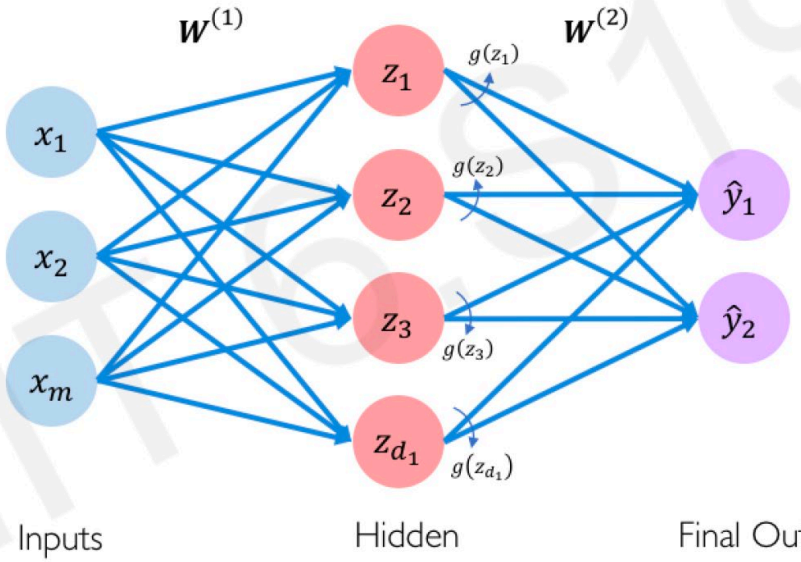


$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

```
torch.nn.Linear(in_features, out_features, bias=True)
```

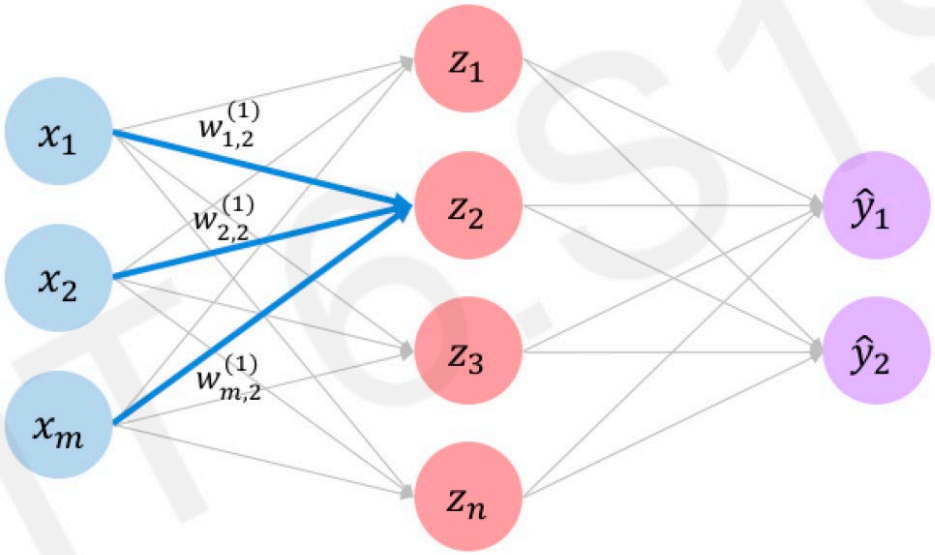
Single layer Neural Network

- Any directed graph built from neurons is a neural network!
- The art lies in: loss design, algorithms to update weights, data representation as inputs, data augmentation, model design.



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

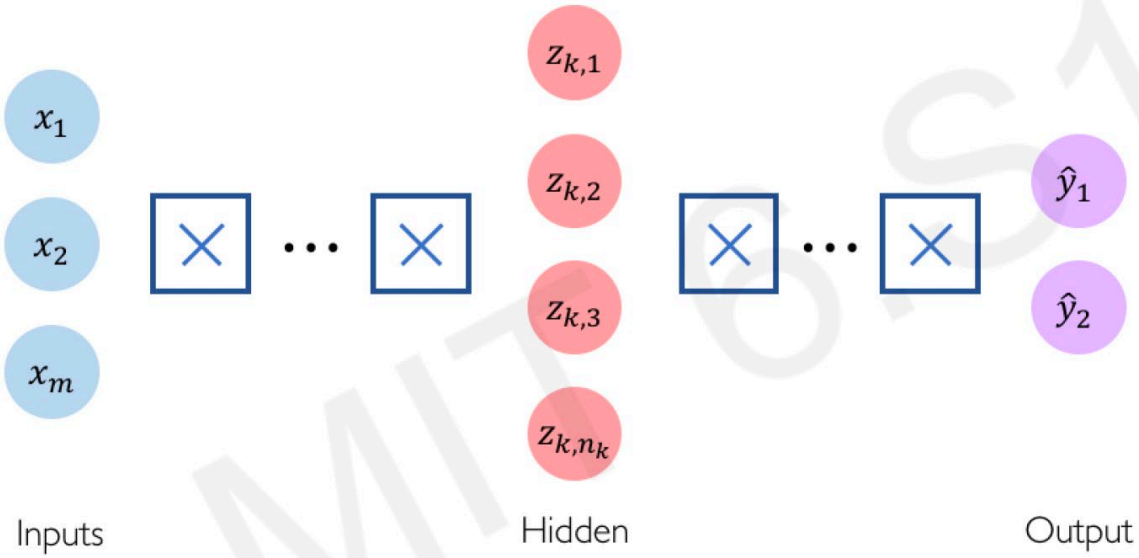
Example: z2



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Deep Neural Network



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

PyTorch 4 layered network

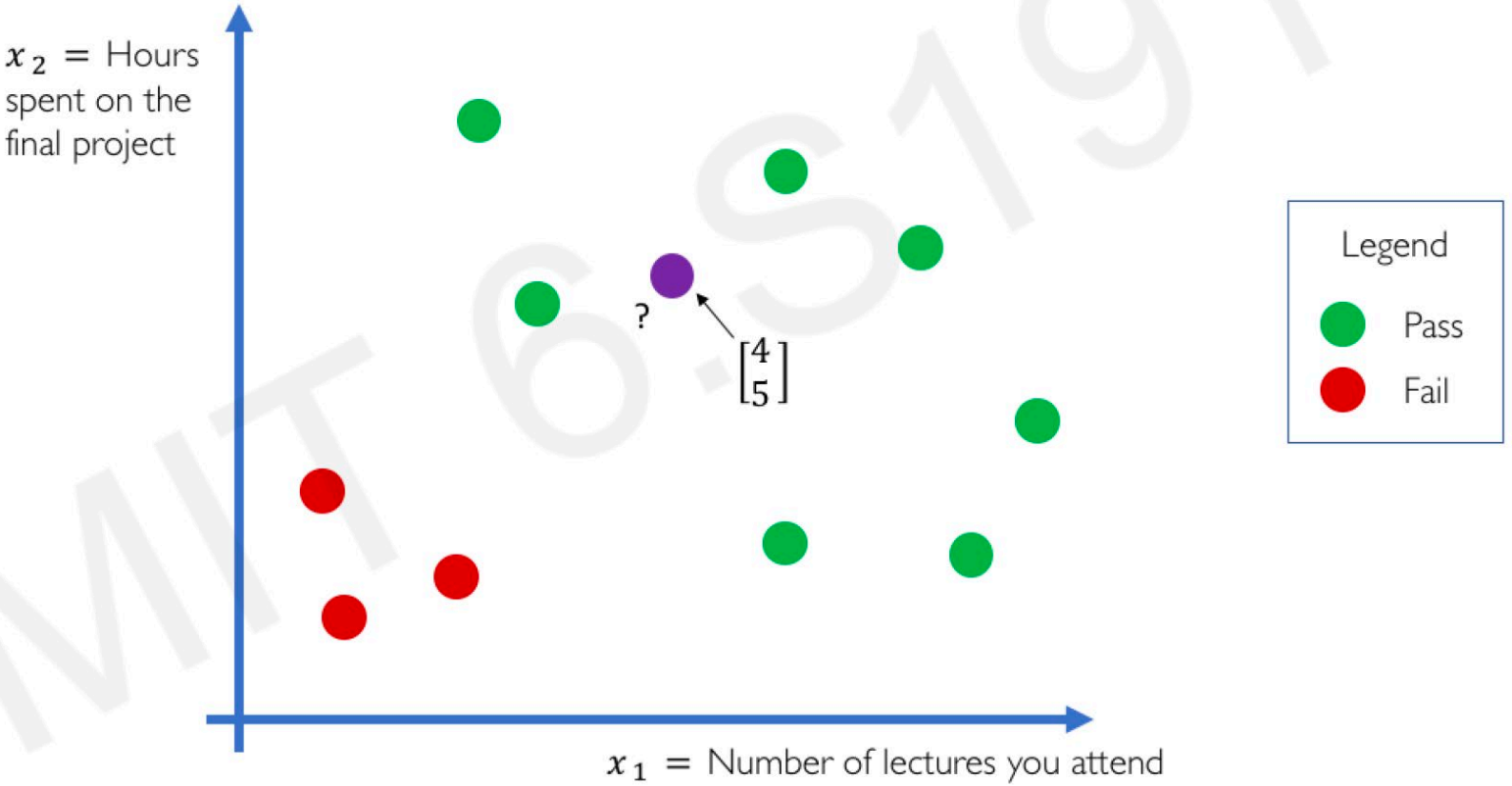
```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 200)
        self.fc2 = nn.Linear(200, 200)
        self.fc3 = nn.Linear(200, 10)

def forward(self, x):
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = self.fc3(x)
    return F.log_softmax(x)

net = Net()
print(net)
```

Example Neural Network

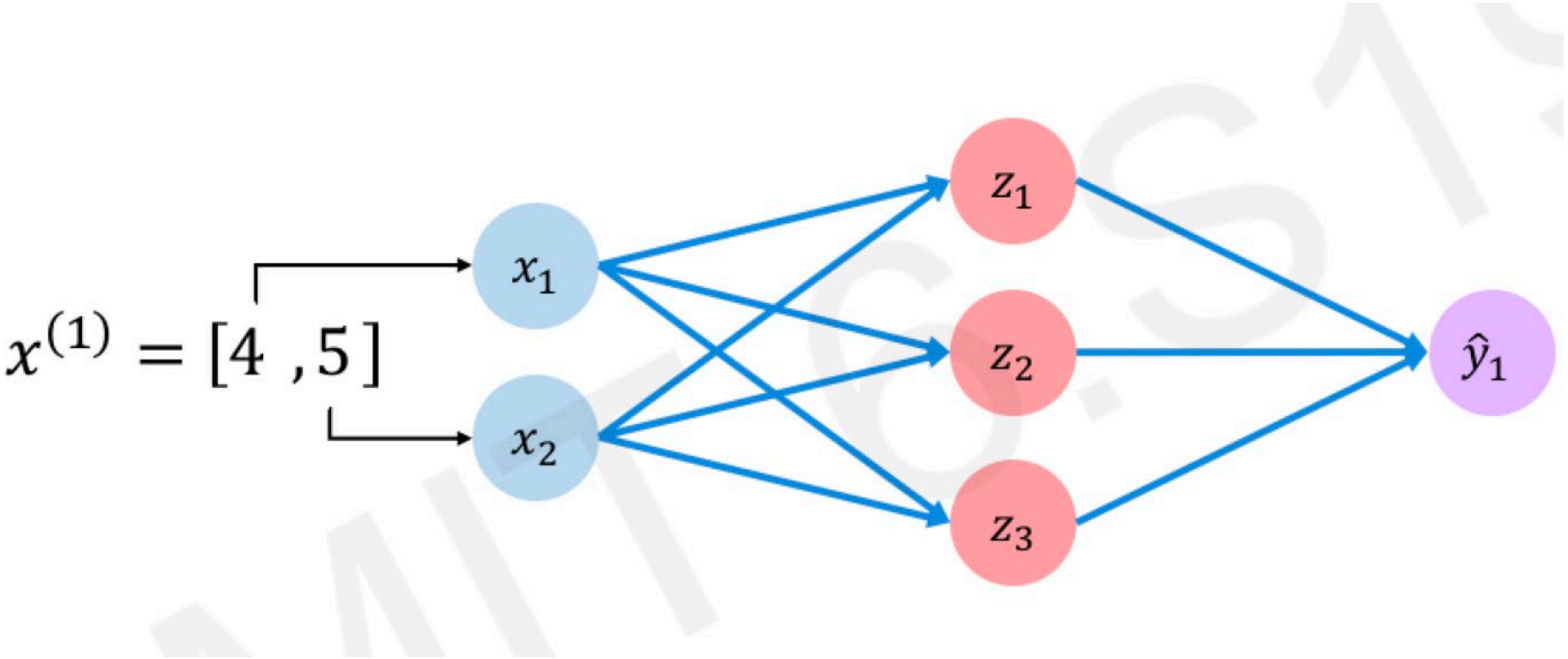
- Will I pass this class?

- Simple network with 2 features:
  - X_1 Number of lectures you attend
  - X_2 Number of hours spent on the final project

Will I pass this class?

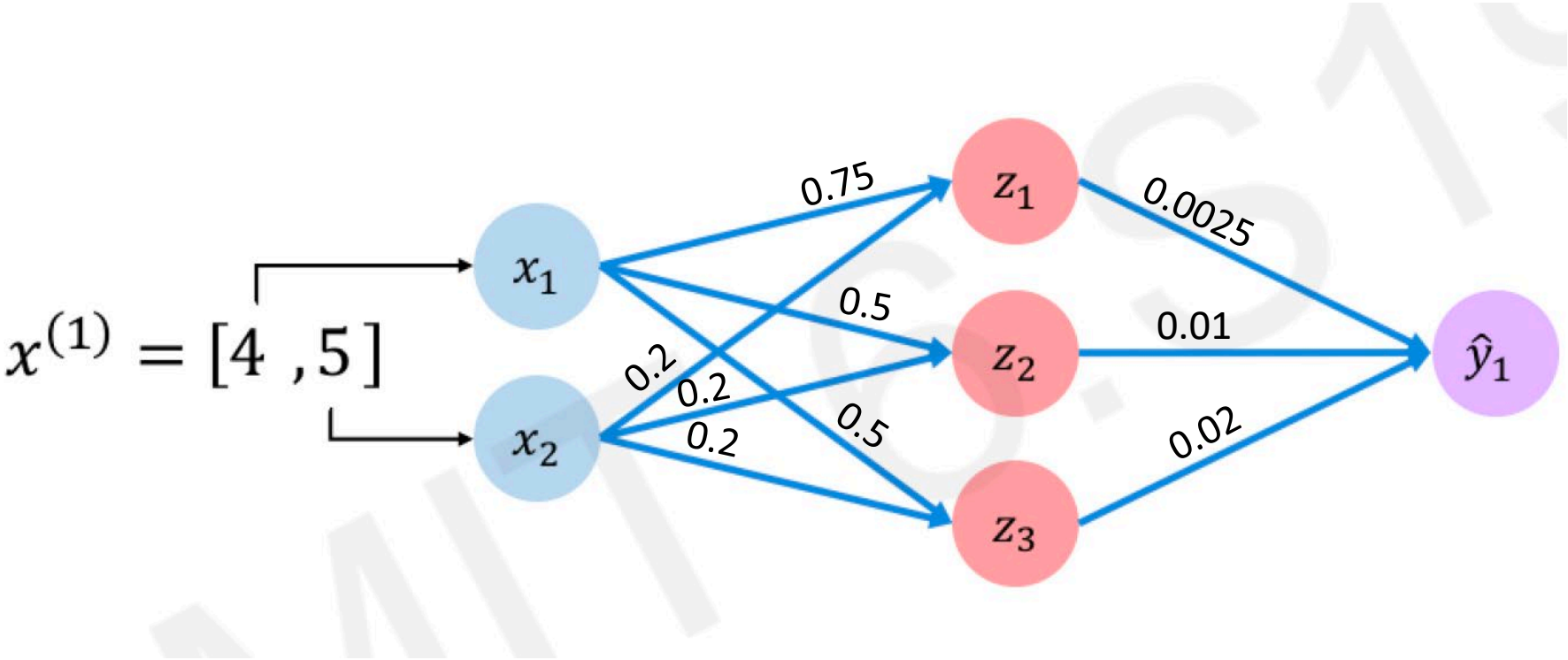


Will I pass this class?



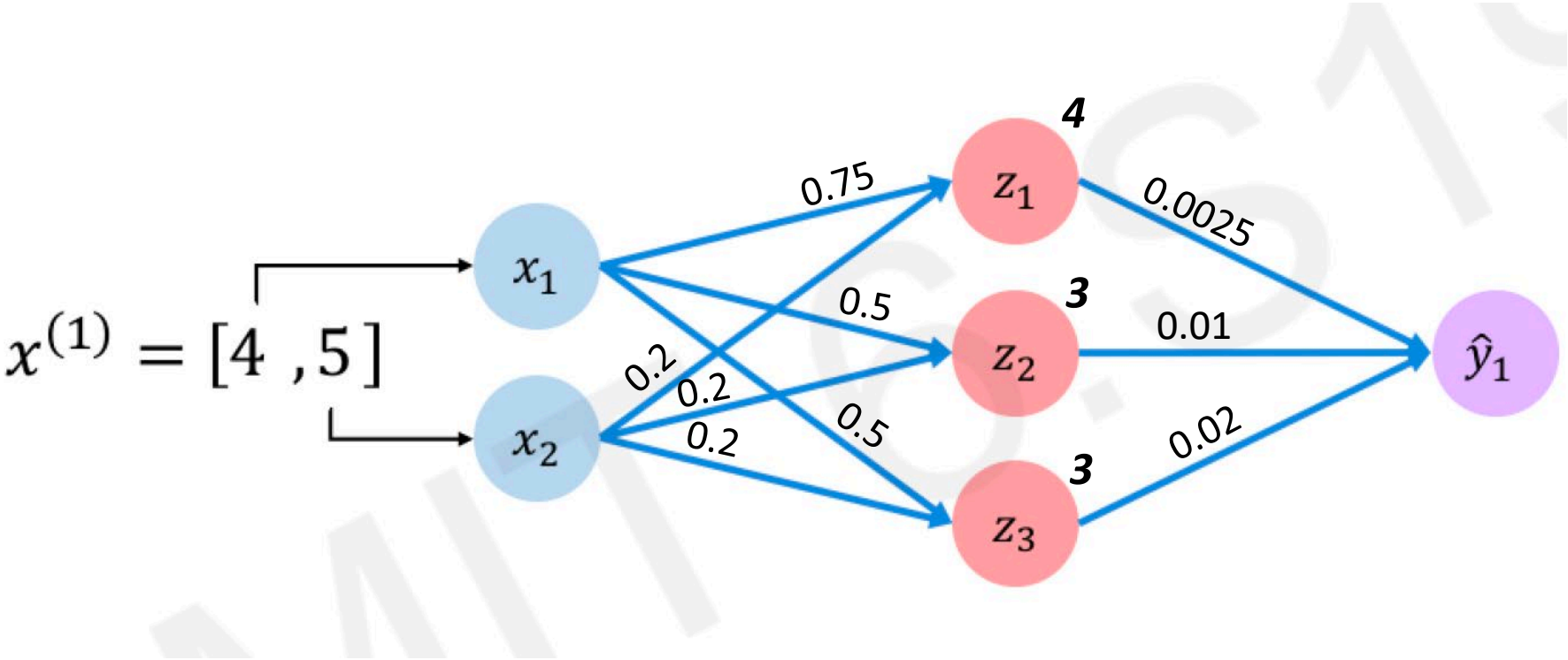
Exercise: Will I pass this class?

- Compute the value at y_1 based on the weights given. Ignore the bias term and assume a linear activation g(z)=z.



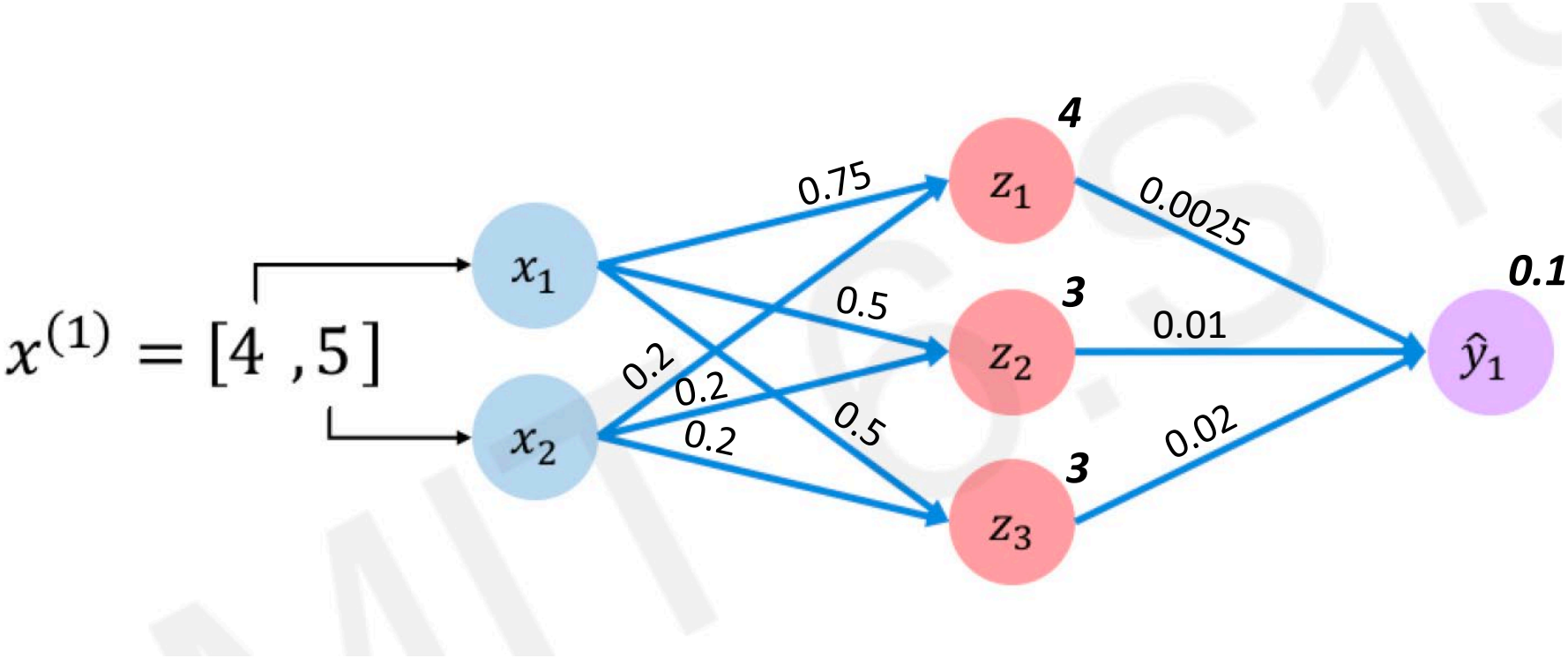
Exercise: Will I pass this class?

- Compute the value at y_1 based on the weights given. Ignore the bias term and assume a linear activation g(z)=z.

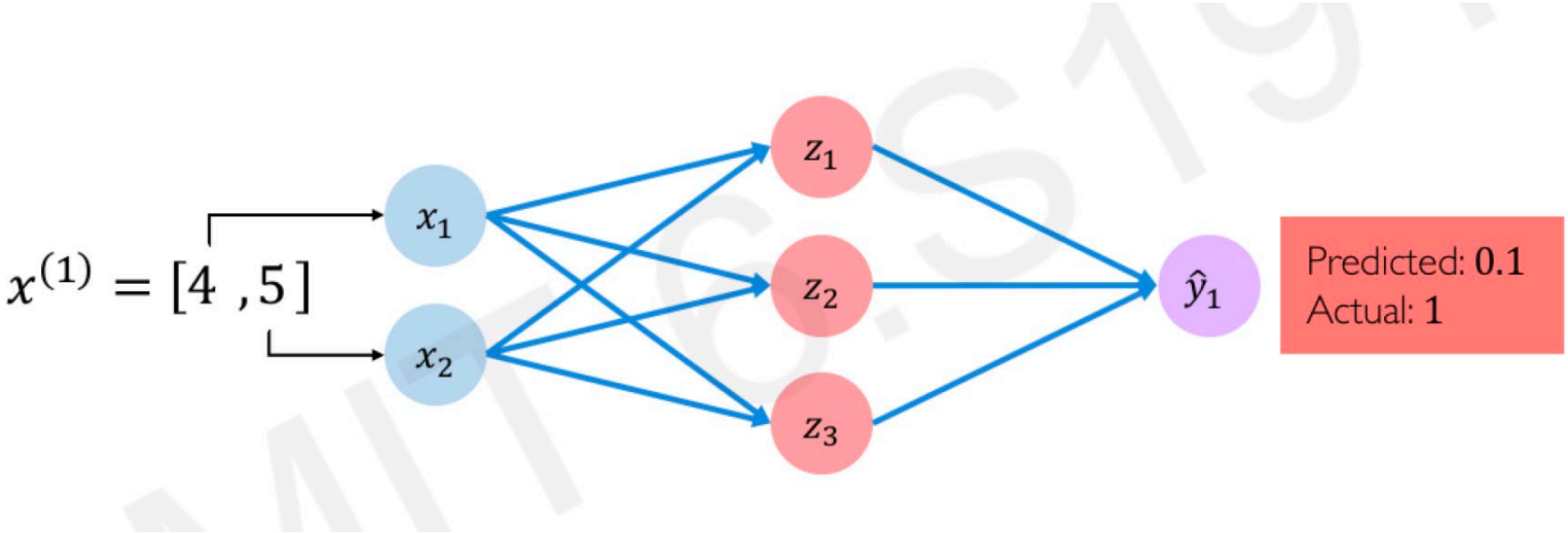


Exercise: Will I pass this class?

- Compute the value at y_1 based on the weights given. Ignore the bias term and assume a linear activation g(z)=z.

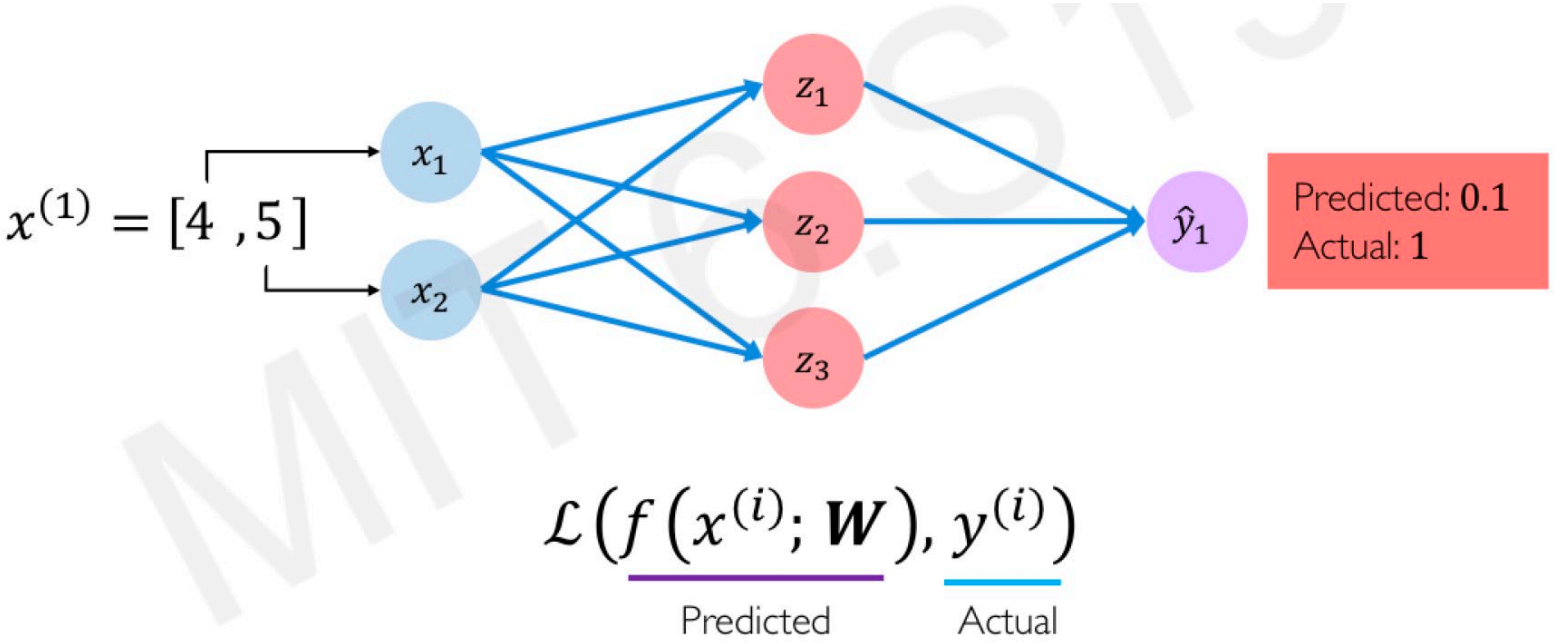


Exercise: Will I pass this class?



Quantifying loss

- The **loss** of our network measures the cost from mispredictions

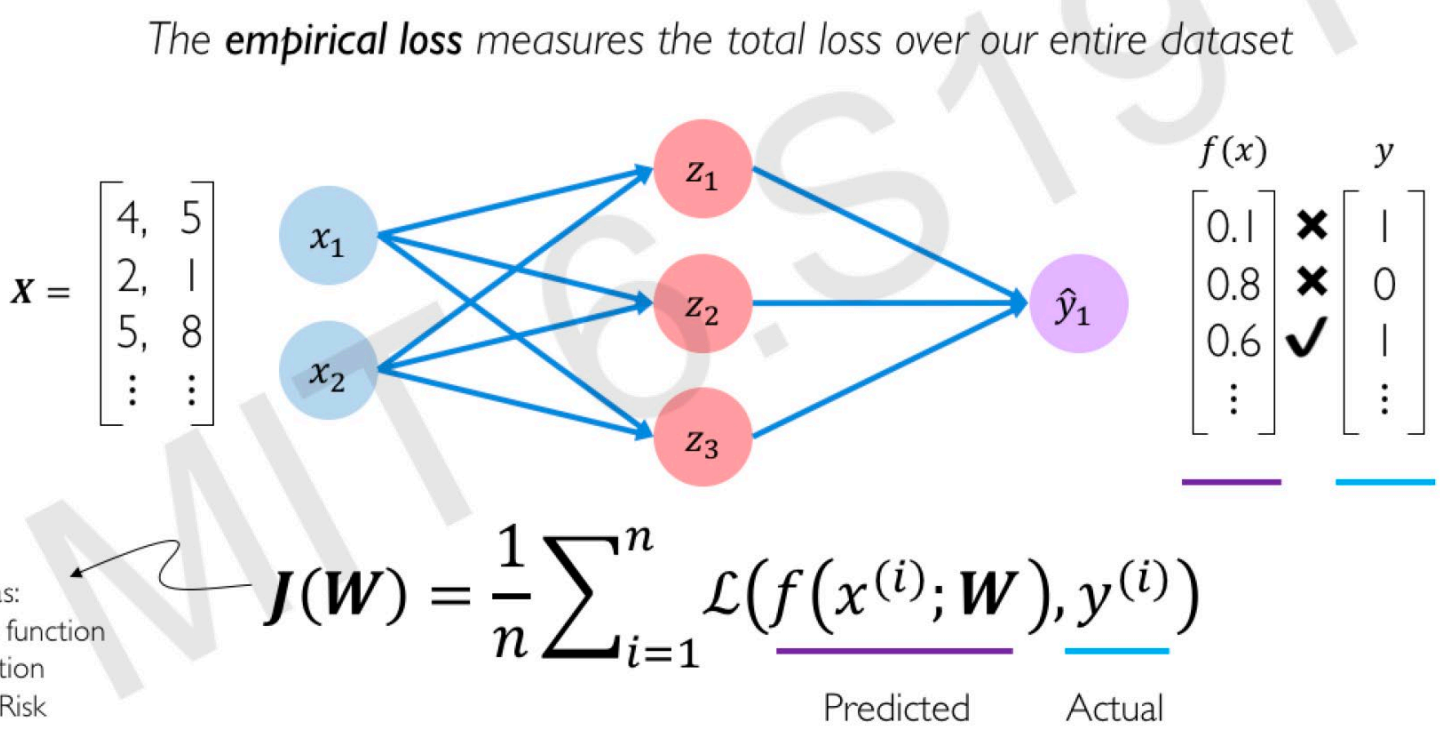


$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical loss

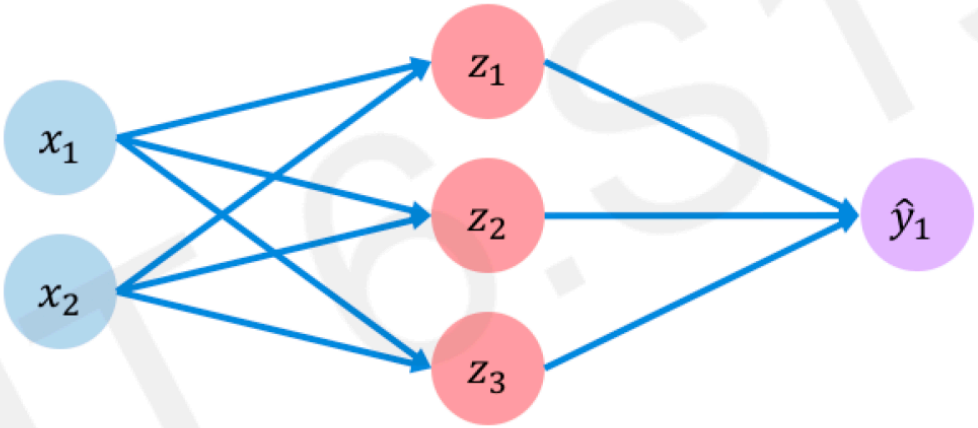
- The empirical **loss** measures the total loss over our entire dataset



Binary Cross-entropy Loss

- For models that output a probability between 0 and 1

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



$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \quad \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} \quad \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)}; \boldsymbol{W}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}; \boldsymbol{W}\right)\right)$$

Actual — Predicted — Actual — Predicted

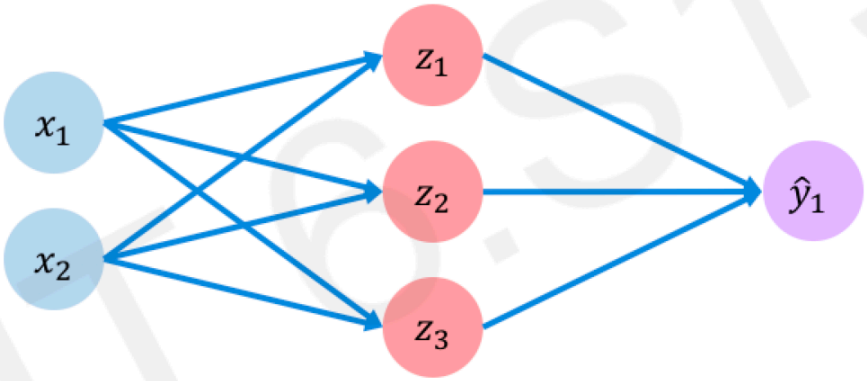
`torch.nn.torch.nn.BCELoss`

Mean Squared Error Loss

- For regression models that output continuous real numbers

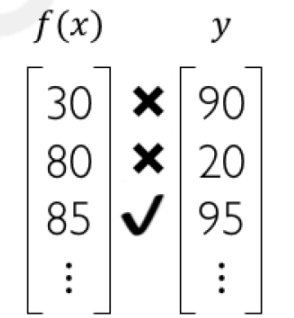
Mean squared error loss *can be used with regression models that output continuous real numbers*

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



$$\boldsymbol{J(W)} = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted



Final Grades (percentage)

```
torch.nn.MSELoss()
```

Next

- How to train a neural network
- How to prevent overfitting

Credits

- Images thanks to:
  - MIT 6.S191
  - https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba
  - https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
  - https://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - https://colah.github.io/