CSED!

"A Place To Invent And Learn"

Deep Learning Course

The Rise of Deep Learning

'Deep Voice' Software Can Clone Anyone's Voice With Just 3.7 Seconds of Audio

Using snippets of voices, Baidu's 'Deep Voice' can generate new speech, accents, and tones.

Let There Be Sight: How Deep Learning Is Helping the Blind 'See'

AI Can Help In Predicting Cryptocurrency Value

chnology outpacing security ures

Facial Recognition | Features and Interviews

AI beats docs in cancer spottin

A new study provides a fresh example of machine learning as an importa diagnostic tool. Paul Biegler reports.

DEEPMIND I STARCRAFT TRIUMPH FO

'Creative' AlphaZero leads way for chess computers and, maybe, science

Former chess world champion Garry Kasparov likes what he sees of computer that could be used to find cures for diseases

How an A.I. 'Cat-and-Mouse Game' Generates Believable Fake Photos

By CADE METZ and KEITH COLLINS JAN. 2, 2018

Stock Predictions Based On AI: Is the Market Truly Predictable?

Neural networks everywhere

New chip reduces neural networks' power consumption by up to 95 percent, making them practical for battery-powered devices.

Wed, 01/16/2019 - 8:00am 1 Comment by Kenny Walter - Digital Reporter - @RandDMagazine

MIT researchers have developed a special-purpose chip that increases the speed of neural-network computations by three to seven times over its predecessors, while reducing power consumption 93 to 96 percent. That could make it practical to run neural networks locally on smartphones or even to embed them in household appliances.

e faces show how far AI image generation has nced in just four years

ple on the right aren't real; they're the product of machine learning

on, Faked Data

SELF-DRIVING STARTUP

After Millions of Trials, These Simulated Humans Learned to Do Perfect Backflips and Cartwheels

George Dvorsky

Deep L

Automation And Algorithms: De-Risking Manufacturing With Artificial Intelligence

Sarah Goehrke Contributor

Manufacturing

I focus on the industrialization of additive manufacturing.

TWEET THIS

The two key applications of AI in manufacturing are pricing and manufacturability feedback

Researchers introduce a deep learning method that converts mono audio recordings into 3D sounds using video scenes

Complex of bacteria-infecting viral proteins modeled in CASP 13. The complex cont that were modeled individually. PROTEIN DATA BANK

Google's DeepMind aces protein folding

By Robert F. Service | Dec. 6, 2018, 12:05 PM

To create the final image in this set, the system generated 10 million revisions over 18 days.

y parent company Alphabet, is

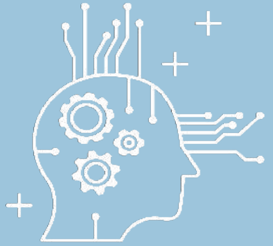
ascent self-driving technology

TensorFlow

What is Deep Learning?

Artificial Intelligence

Any technique that enables computers to mimic human behavior

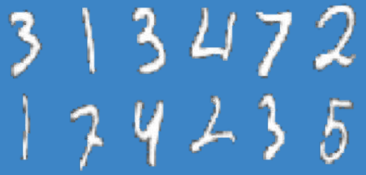


Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks

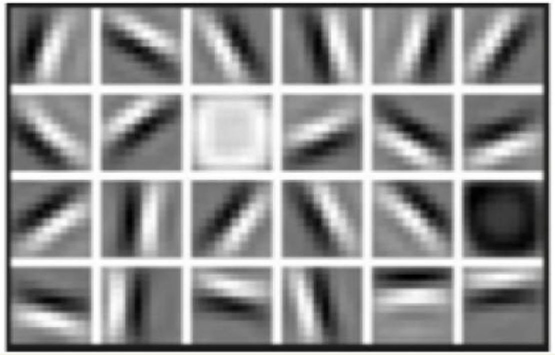


Why Deep Learning and Why Now?

Why Deep Learning?

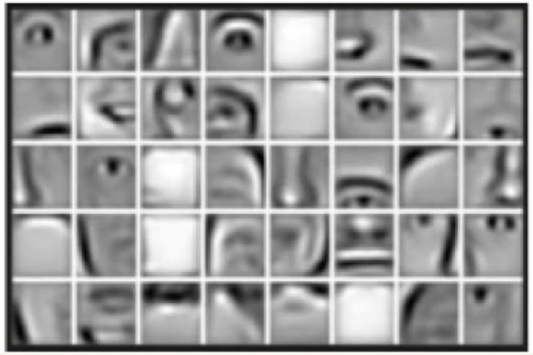
Hand engineered features are time consuming, brittle and not scalable
In practice Can we learn the **underlying features** directly from data?

Low Level Features



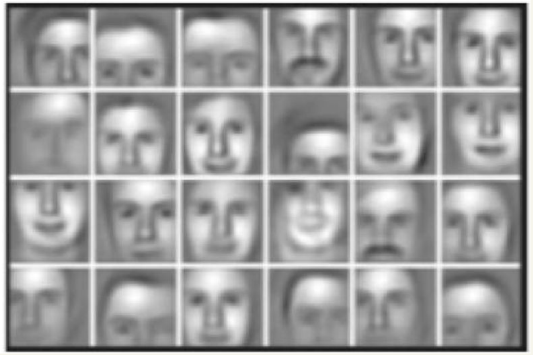
Lines & Edges

Mid Level Features



Eyes & Nose & Ears

High Level Features



Facial Structure

Why Now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

⋮

1. Big Data

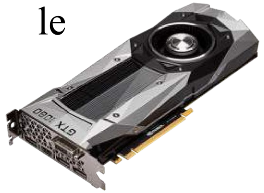
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

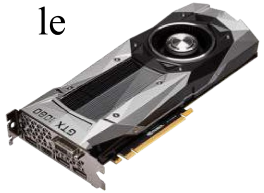
- Graphics Processing Units (GPUs)
- Massively Parallelizable



3. Software

- Improved Techniques
- New Models
- Toolboxes

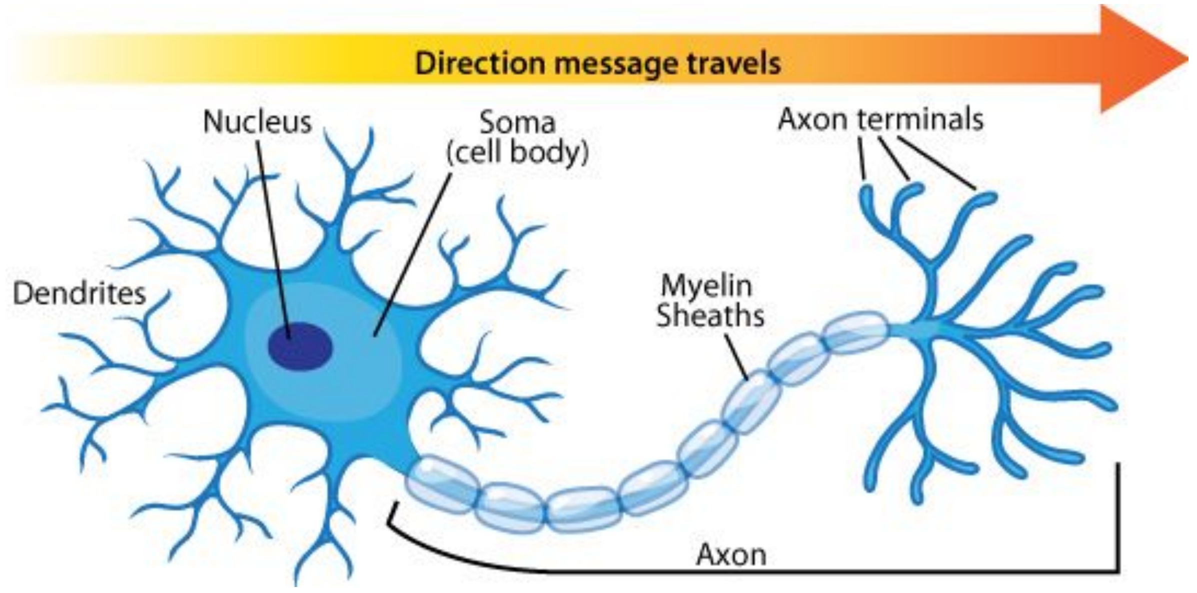
TensorFlow



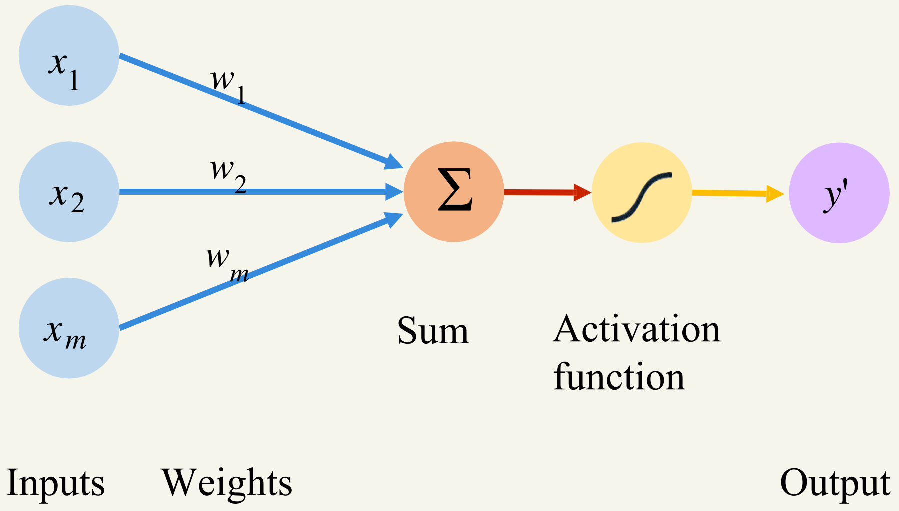
The Perceptron

The structural building block of deep learning

Neuron Structure



The Perceptron: Forward Propagation



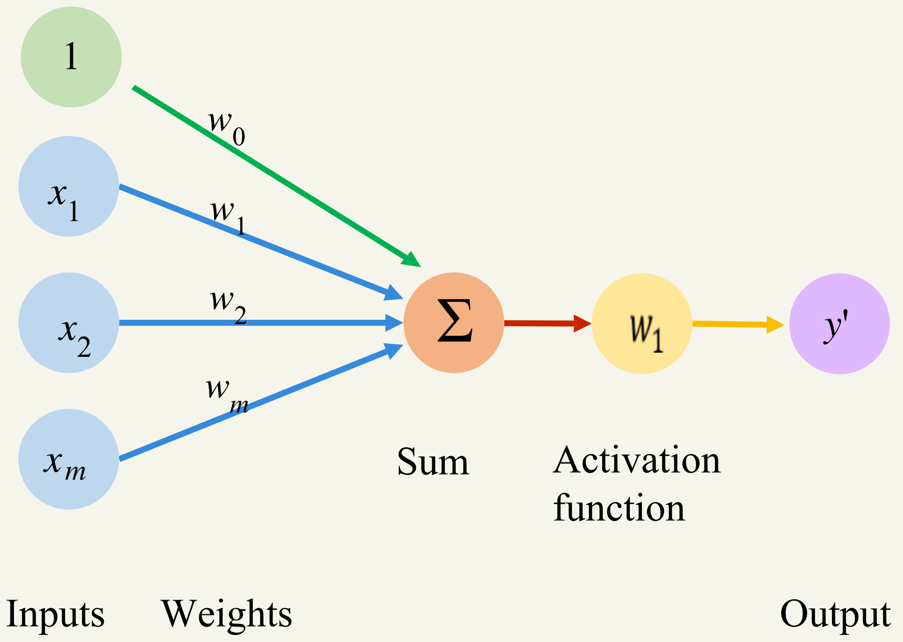
Output

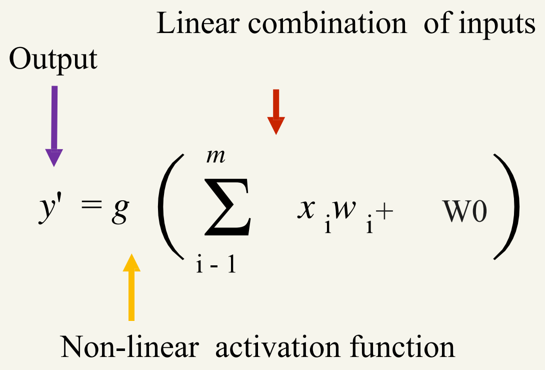
Linear combination of inputs

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

Non-linear activation function

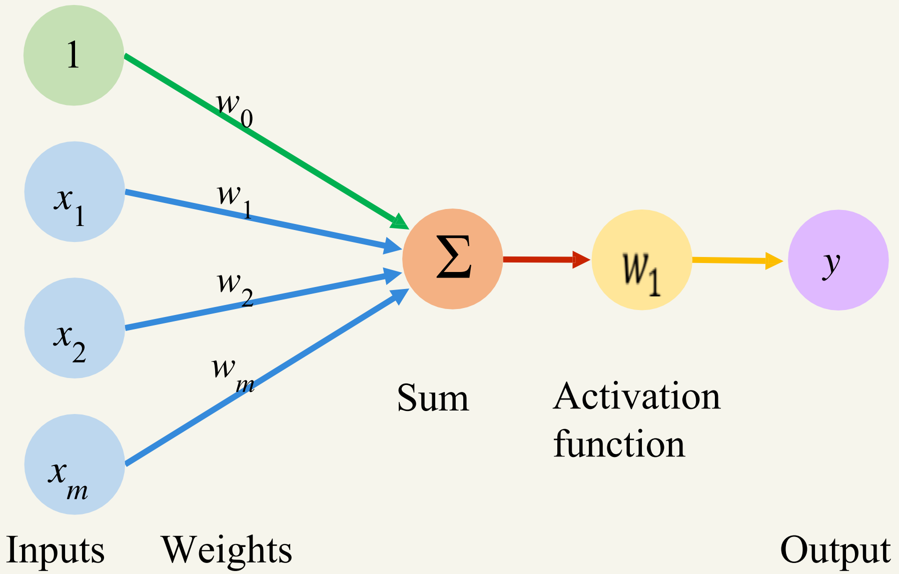
The Perceptron: Forward Propagation





$$y' = g\left(\sum_{i-1}^{m} x_i w_i + \mathrm{W0}\right)$$

The Perceptron: Forward Propagation

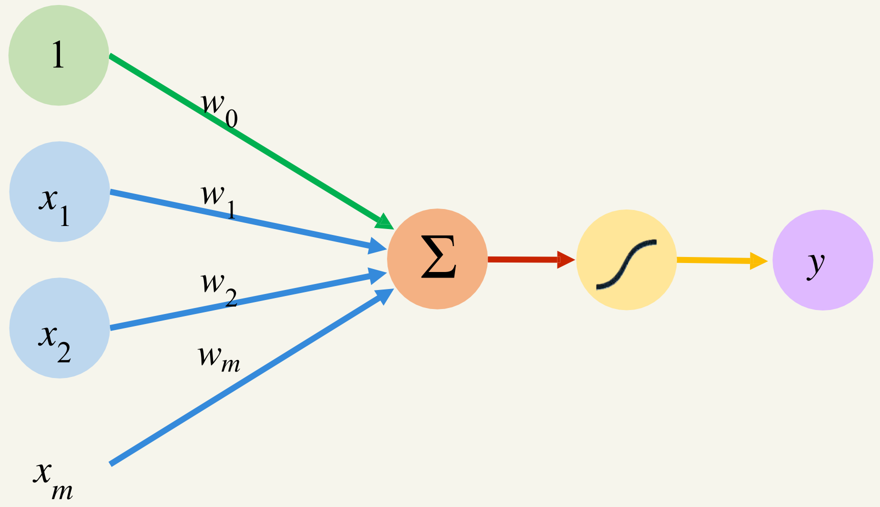


$$y' = g\left(\sum_{i-1}^{m} x_i w_i + \mathrm{W0}\right)$$

$$y = g\ \left(w_0 + X^T \mathrm{w}\ \right)$$

$$\text{where: } X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{and } 3\ = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The Perceptron: Forward Propagation

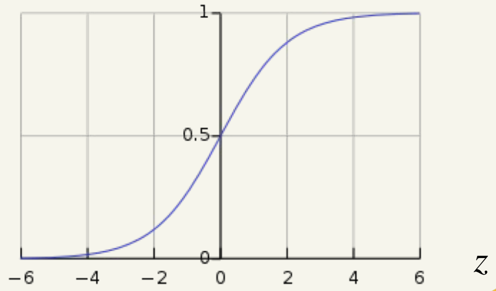


$$y = g\left(w_0 + X^T \mathrm{w}\right)$$

where: $X = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and 3 $= \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

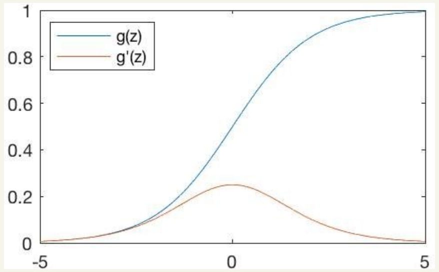
Example: sigmoid function

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



Common Activation Functions

Sigmoid Function

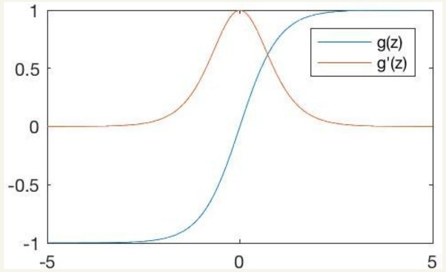


$$g(z) = \frac{1}{1 + e^{-z}}$$

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

tf.nn.sigmoid(z)

Hyperbolic Tangent

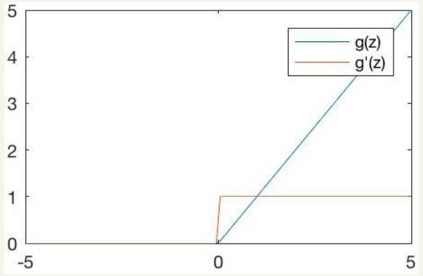


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

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

tf.nn.tanh(z)

Rectified Linear Unit (ReLU)



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

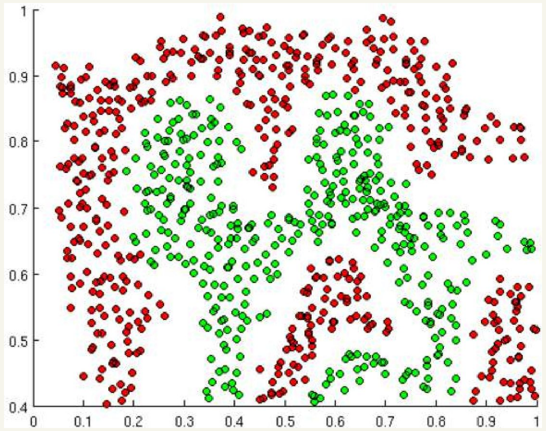
$$g'(z) = \begin{cases} 0 & z > 0 \\ 1 & \text{otherwise} \end{cases}$$

tf.nn.relu(z)

NOTE: All activation functions are non-linear

Importance of Activation Functions

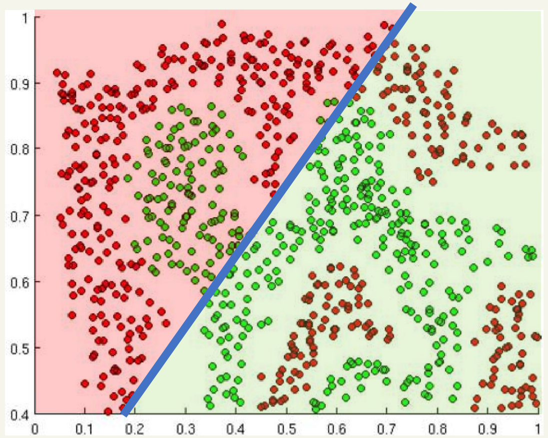
The purpose of activation functions is to introduce non-linearities into the network



What if we wanted to build a Neural Network to distinguish green vs red points?

Importance of Activation Functions

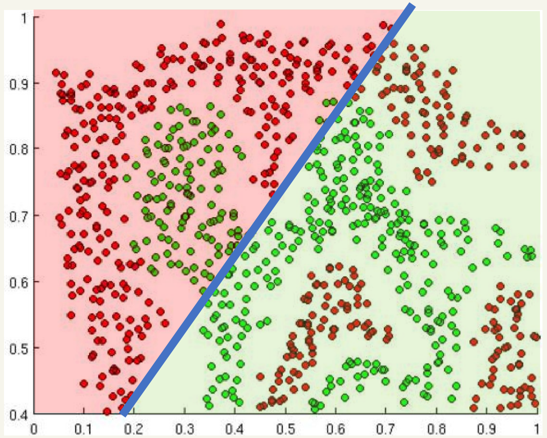
The purpose of activation functions is to introduce non-linearities into the network



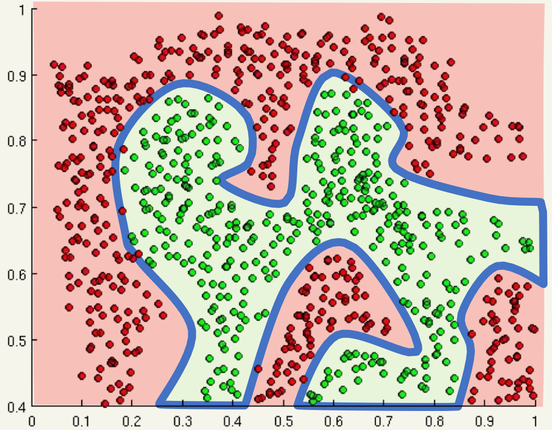
Linear Activation functions produce linear decisions no matter the network size

Importance of Activation Functions

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

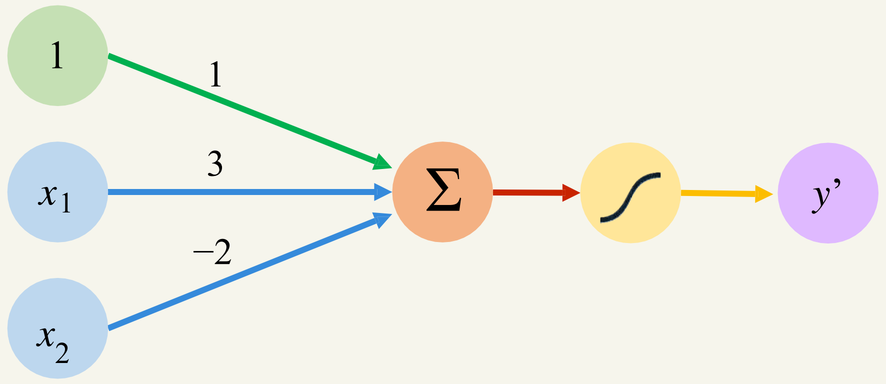


Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

The Perceptron: Example



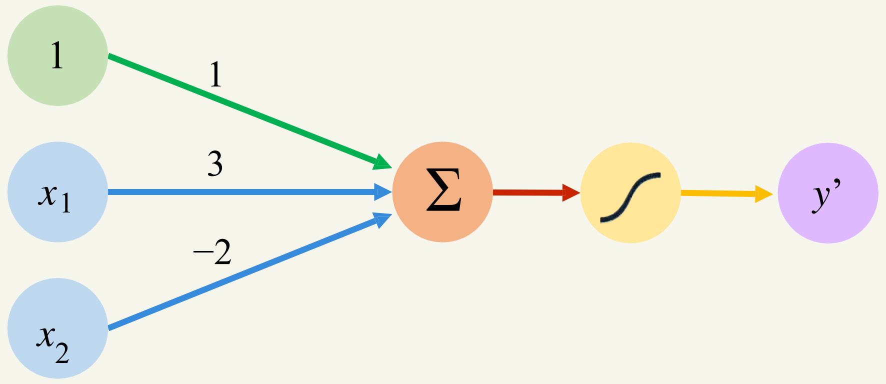
We have: $w_0 = 1$ and $W = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$y' = g\left(w_0 + X^T W\right)$$

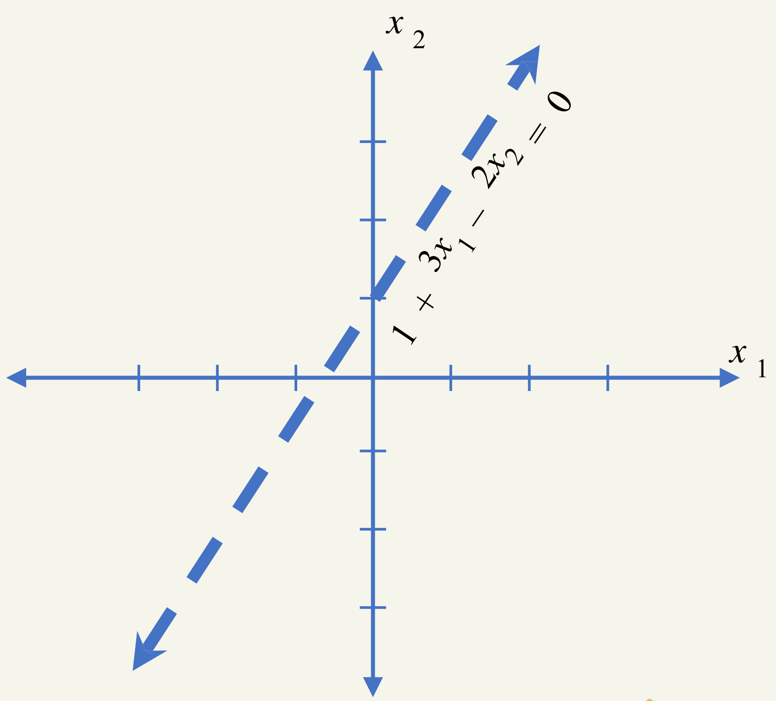
$$y' = g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$$

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

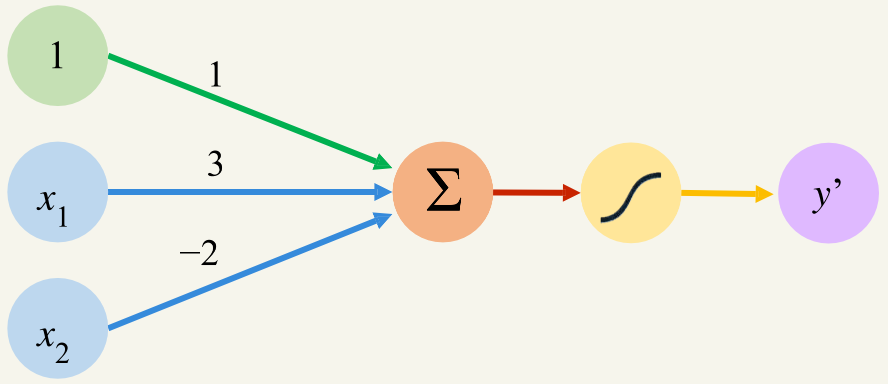
The Perceptron: Example



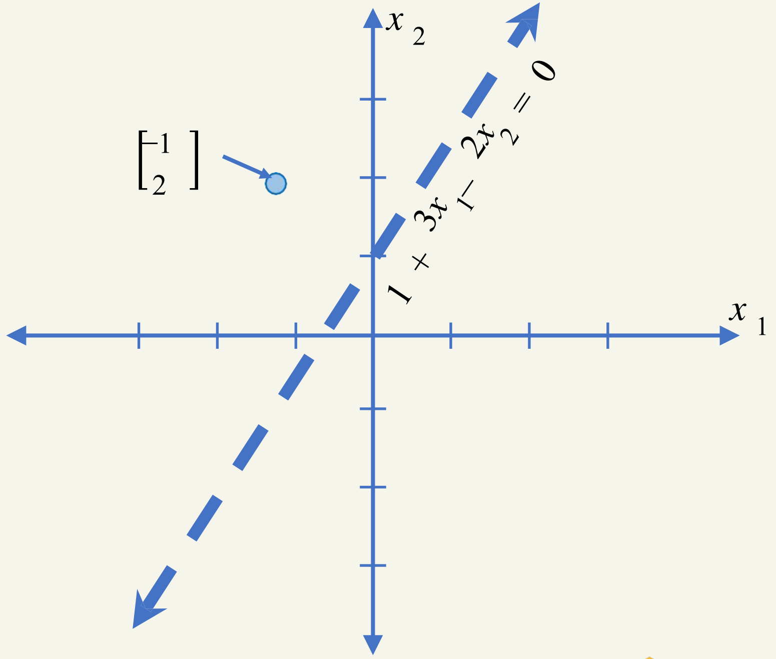
$$y' = g(1 + 3x_1 - 2x_2)$$



The Perceptron: Example



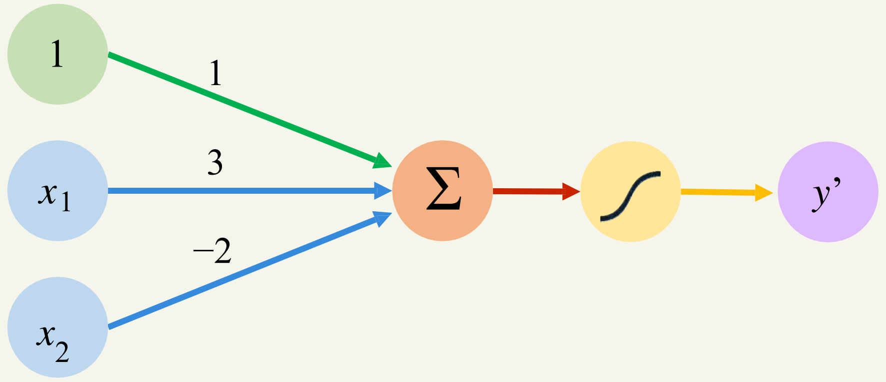
$$y' = g\,(1 + 3x_1 - 2x_2)$$



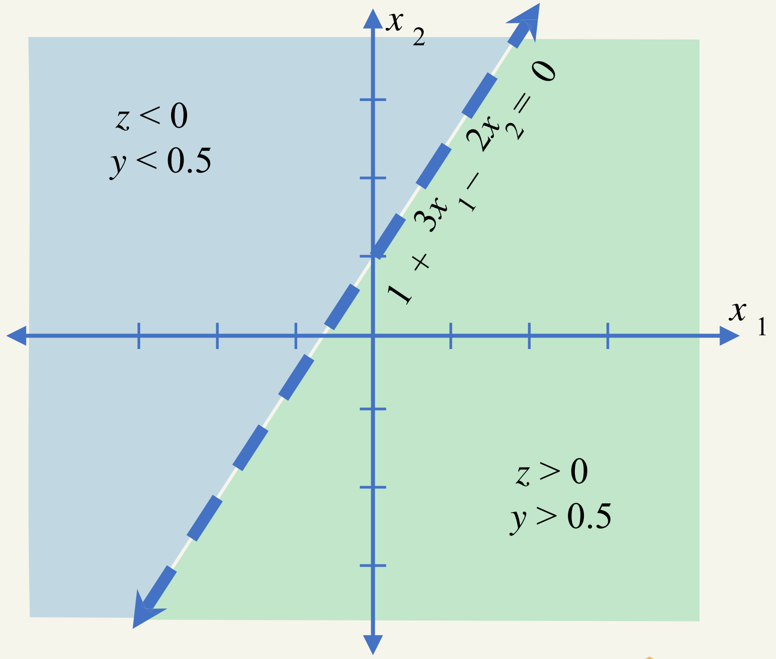
Assume we have input: $X = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$y' = g\,(1 + (3 * -1) - (2 * 2))$$
$$= g\,(-6) \approx 0.002$$

The Perceptron: Example

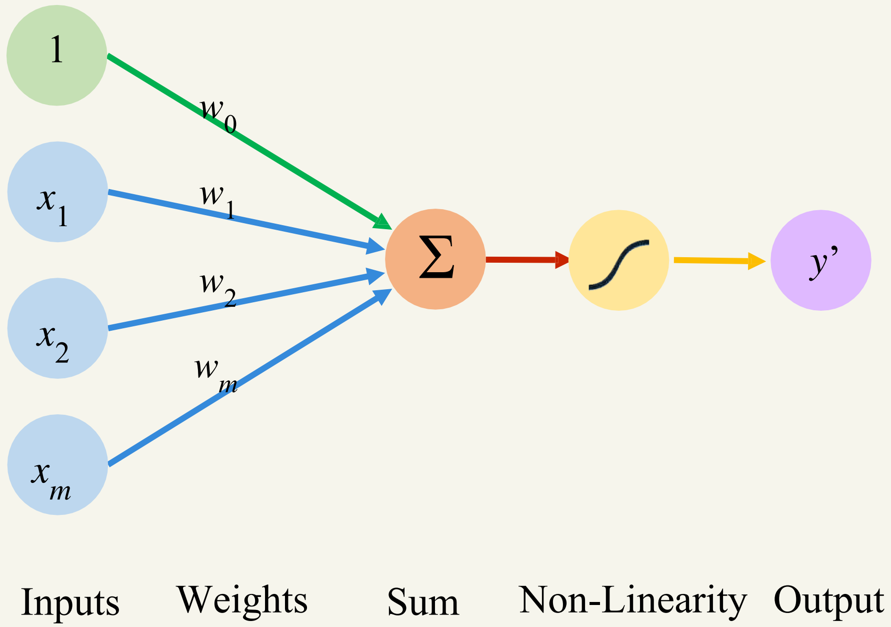


$y' = g(1 + 3x_1 - 2x_2)$

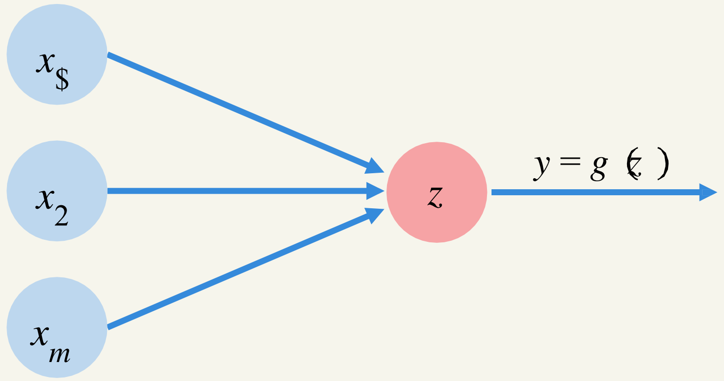


Building Neural Networks with Perceptrons

The Perceptron: Simplified

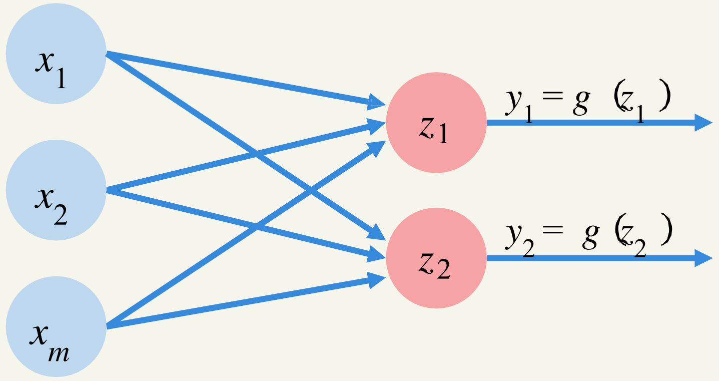


The Perceptron: Simplified



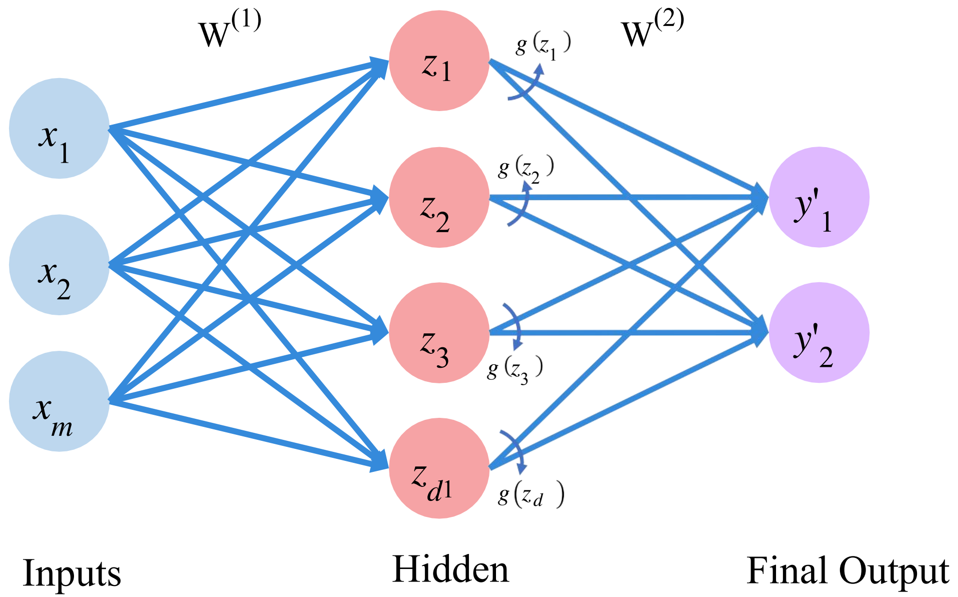
$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

Multi Output Perceptron



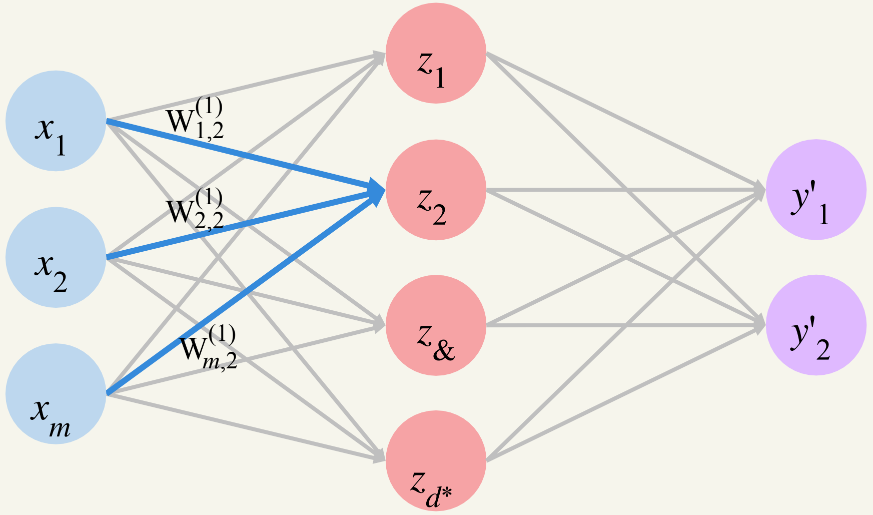
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

Single Layer Neural Network



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

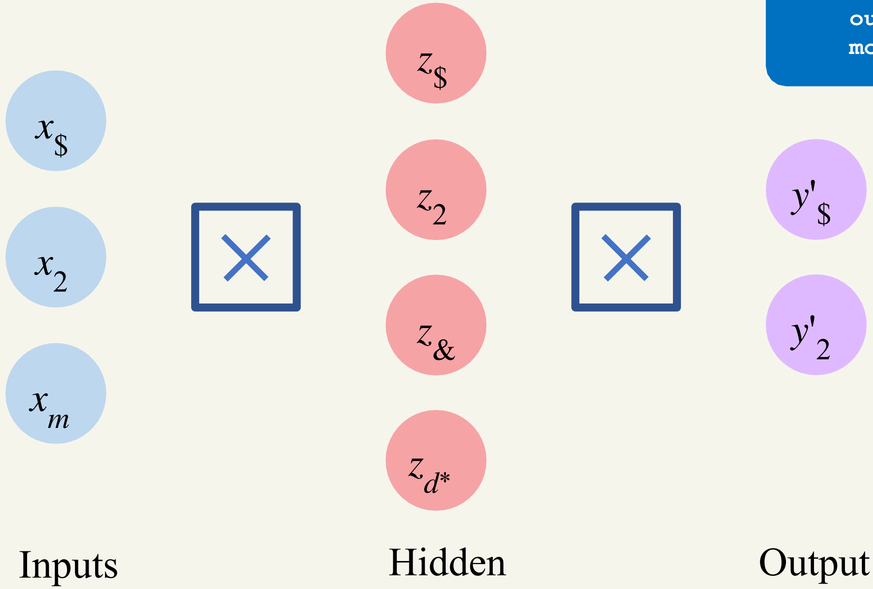
Single Layer Neural Network



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

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

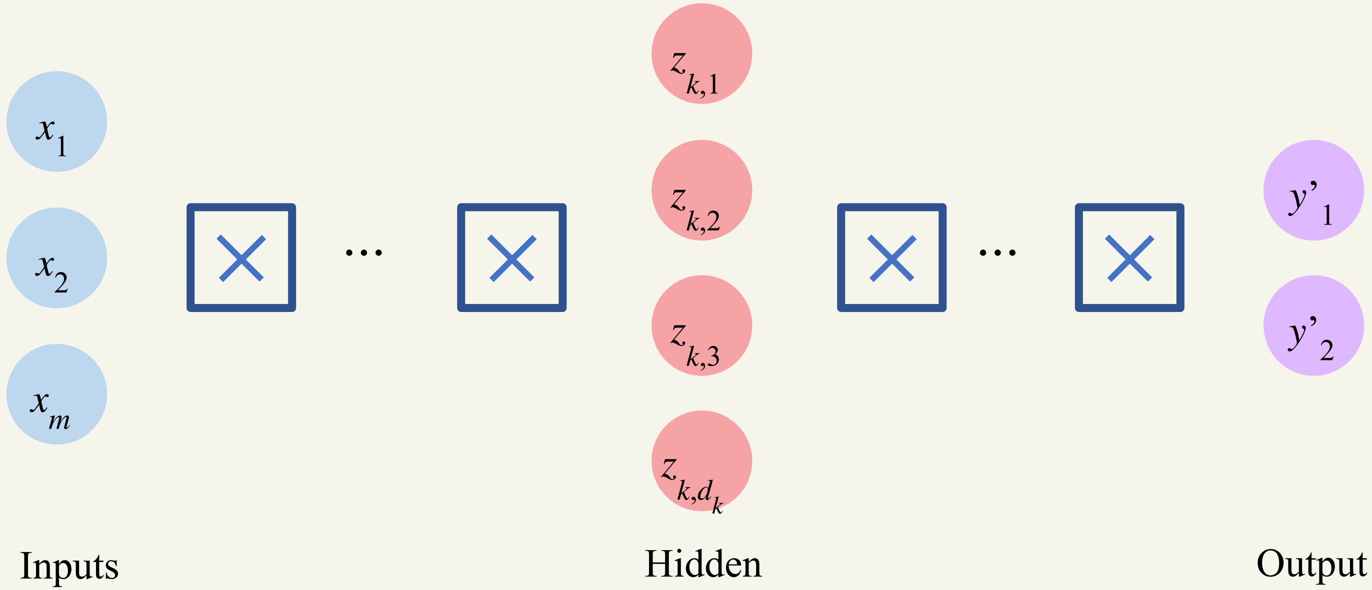
Multi Output Perceptron



```
from tf.keras.layers import *

inputs = Inputs(m)
hidden = Dense(d1)(inputs)
outputs = Dense(2)(hidden)
model = Model(inputs, outputs)
```

Deep Neural Network



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

Applying Neural Networks

Example Problem

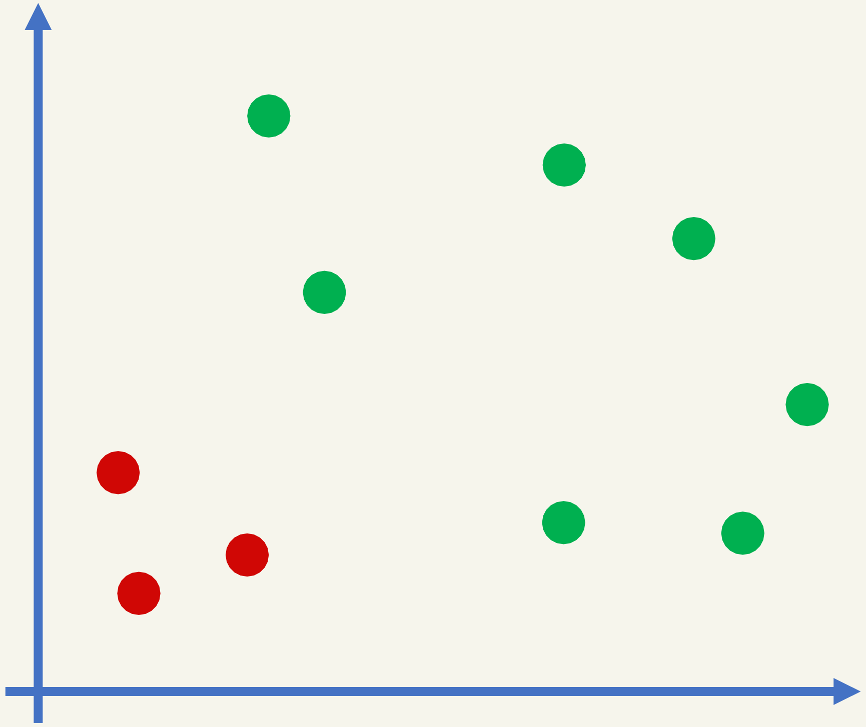
Will I pass this class?

Let's start with a simple two feature model

x_1 = Number of lectures you attend

x_2 = Hours spent on the final project

Example Problem: Will I pass this class?

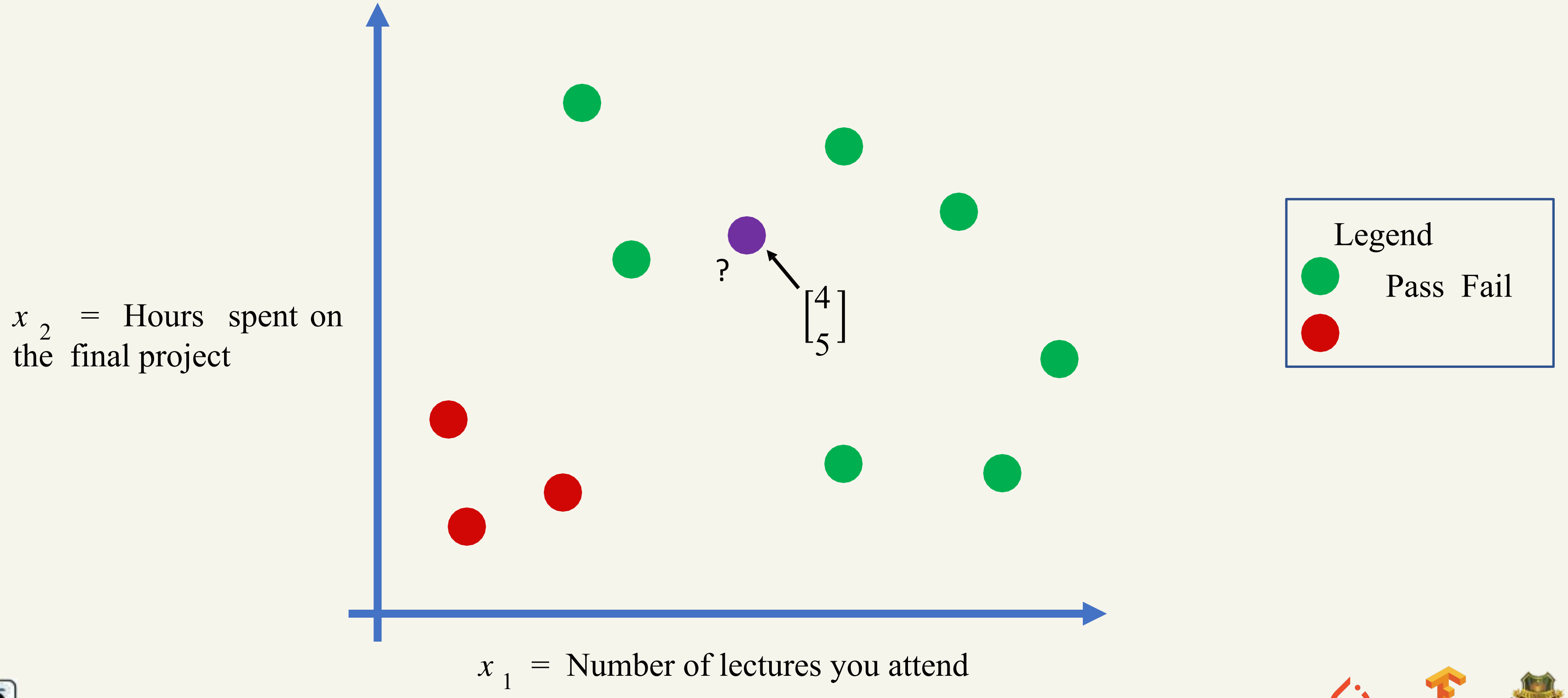


x_2 = Hours spent on the final project

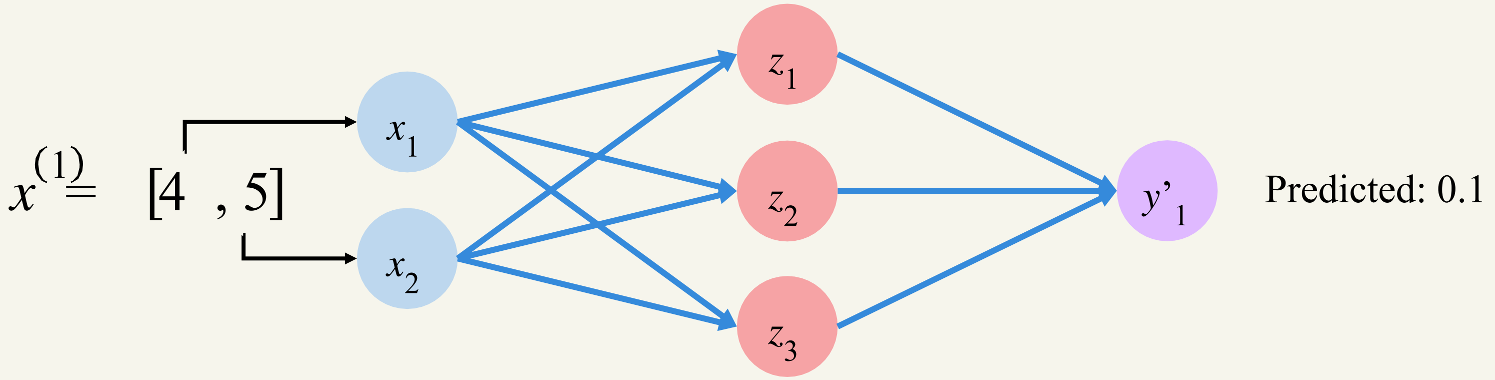
x_1 = Number of lectures you attend

Legend
- 🟢 Pass
- 🔴 Fail

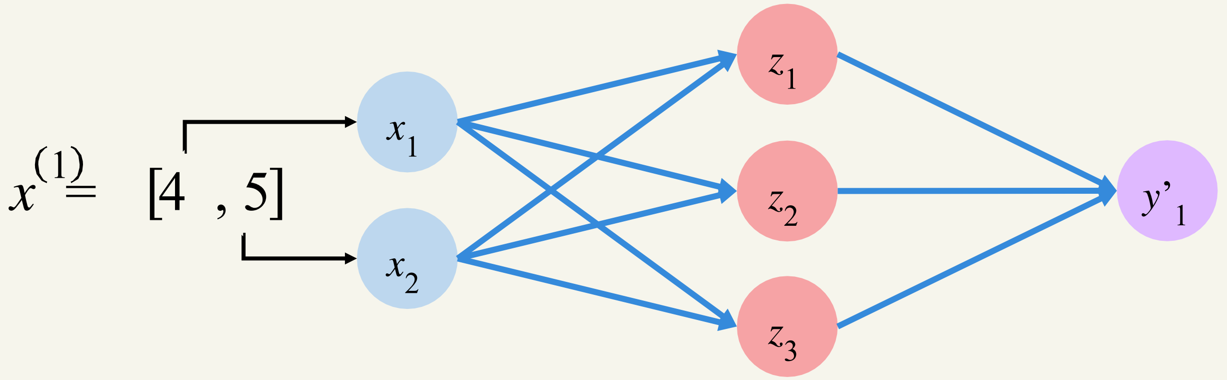
Example Problem: Will I pass this class?



Example Problem: Will I pass this class?



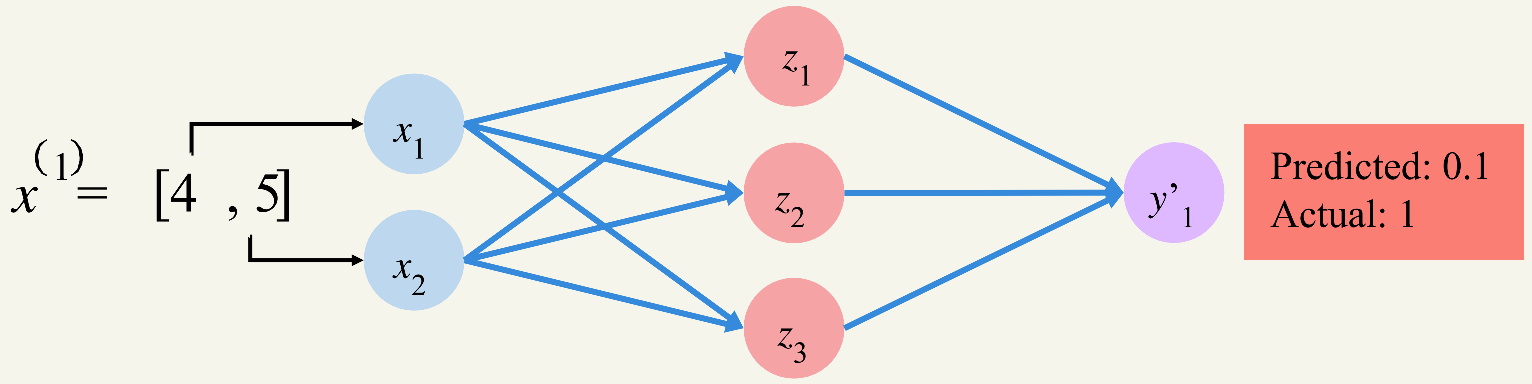
Example Problem: Will I pass this class?



Predicted: 0.1
Actual: 1

Quantifying Loss

The loss of our network measures the cost incurred from incorrect predictions

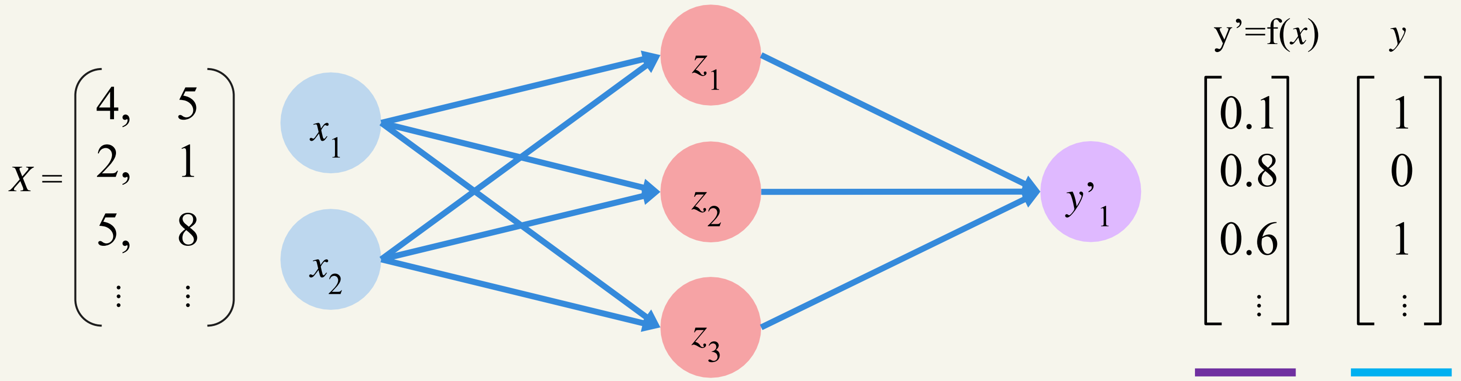


$$f\big(\underline{f(x^{(i)}; \mathrm{W})}, \underline{y^{(i)}}\big)$$

Actual

Empirical Loss

The empirical loss measures the total loss over our entire dataset



$X = \begin{pmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{pmatrix}$
x_1
x_2
z_1
z_2
z_3
y'_1
y'=f(x)
y
$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix}$
$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$

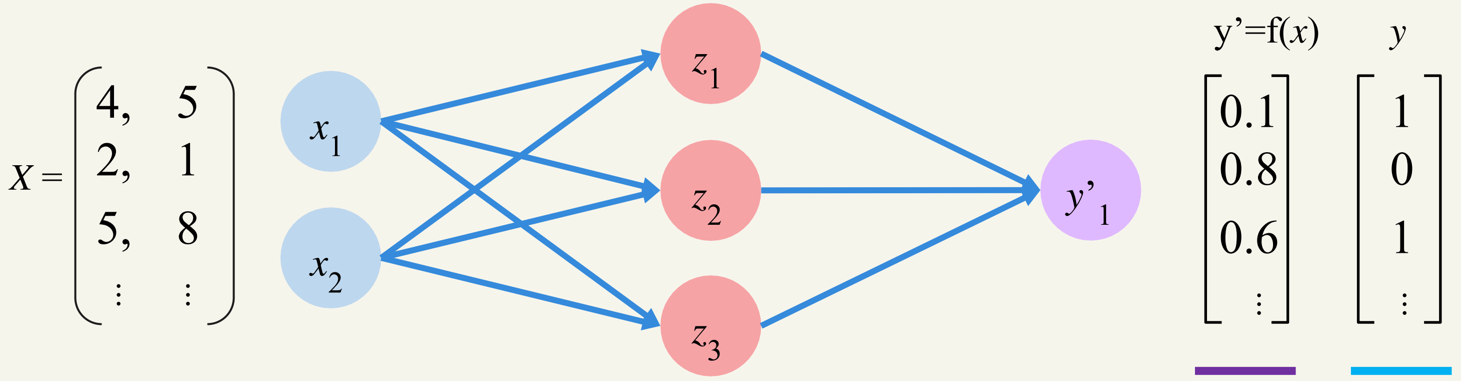
Also known as:

- Objective function
- Cost function
- Empirical Risk

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

Binary Cross Entropy Loss

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



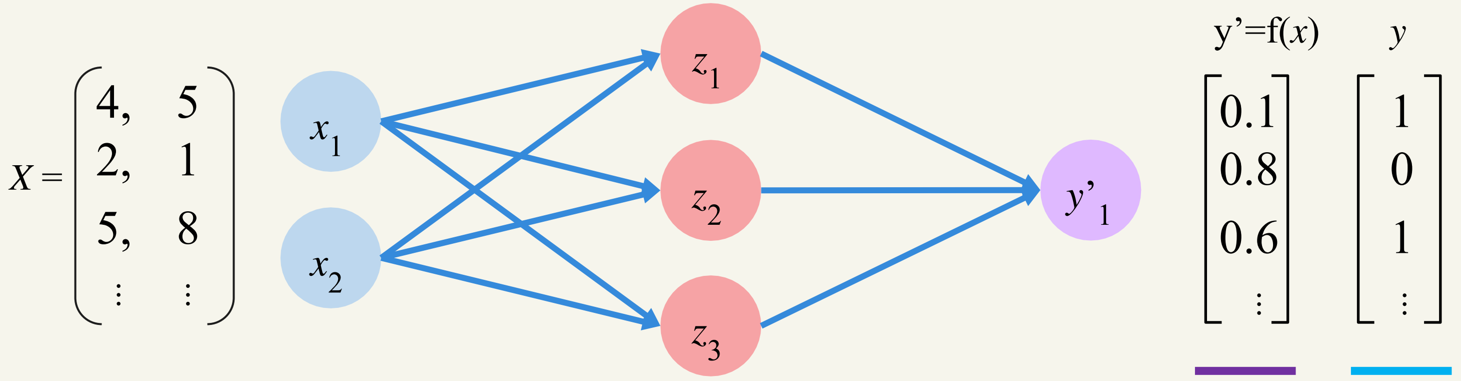
$X = \begin{pmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{pmatrix}$
x_1
x_2
z_1
z_2
z_3
y'_1
y'=f(x)
y
$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix}$
$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$

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

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

Mean Squared Error Loss

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



$X = \begin{pmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{pmatrix}$
x_1
x_2
z_1
z_2
z_3
y'_1
y'=f(x)
y
$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix}$
$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$

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

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