SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

Image Classification

Chapter Goals

After completing this chapter, you should be able to understand :

- What is image classification
- How to build features for image classification
- Limitations of traditional approaches for image classification
- How to implement an image classification pipeline in python

Image classification

Classification

Image Classification: A core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Classification

Label=F(features)

Class

Depending on the problem:
Features of image
Features of object

Classification: Important for other tasks

Example: Object Detection

Classification: Important for other tasks

Example: Object Detection

Background
Donut
Coffee
Person
Car

Classification: Important for other tasks

Example: Object Detection

Background
Donut
Coffee
Person
Car

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

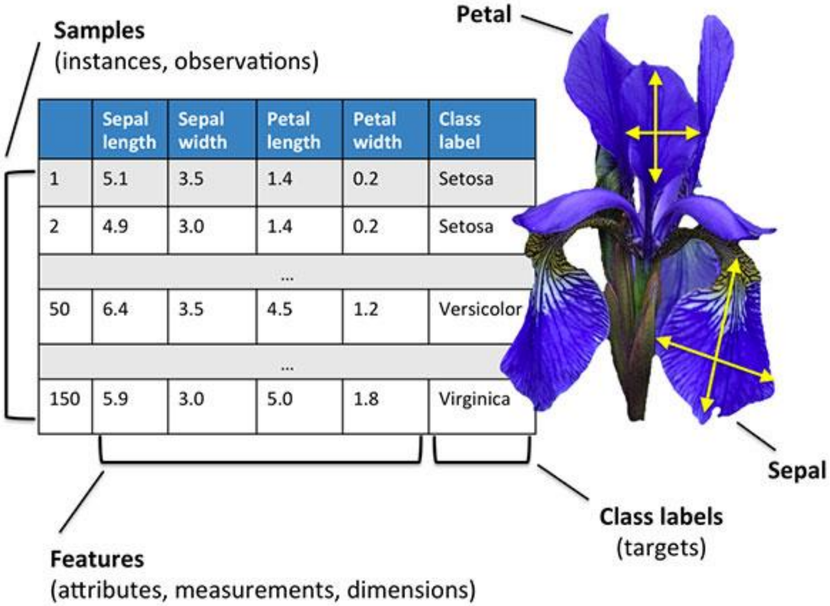
Data Classification

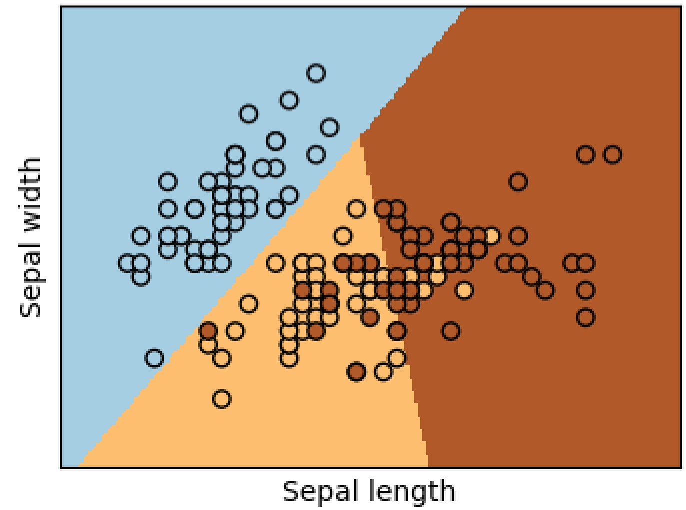
Label=F(features)

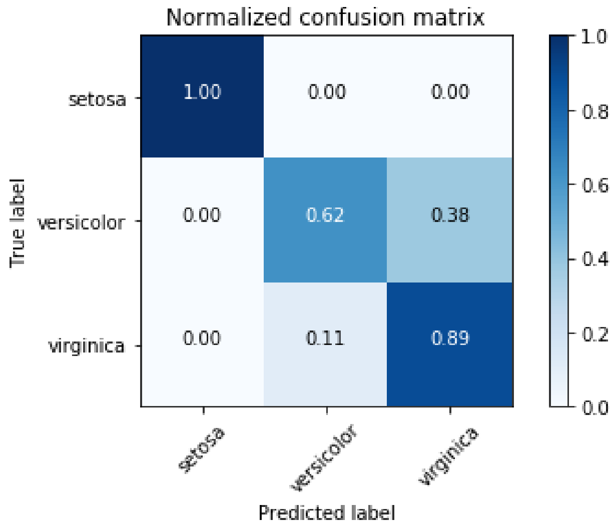
Versicolor (0)
Setosa (1)
Virginica (2)

-Sepal length
-Sepal width
-Petal length
-Petal width

Data Classification







Classical image classification

255	255	127	125	255
255	255	127	125	255
127	127	124	255	255
127	127	65	127	255
255	255	127	127	255
255	127	127	266	266
266	127	255	64	255
255	255	255	64	64
255	64	64	64	255
255	255	64	64	255
255	255	64	255	255

Train

[[255,255,127,255,255,255,255,127,125,255...] [0
[255,255,123,225,207,255,205,127,120,255...] 1
[251,225,127,255,255,255,255,130,135,255...] 0
[255,255,127,255,235,255,255,127,125,255...] 2
[255,205,127,255,255,255,222,127,125,255...]] 0]

Validate

[[255,255,124,255,251,255,255,127,125,255...] [1
[255,205,127,255,255,255,255,137,125,255...] 0
[255,255,127,235,255,255,255,129,125,255...] 2
[255,255,125,255,255,255,255,127,125,255...]] 0]

Classical image classification

Trained model

Label: Virginica

Classical image classification: Features

Trained model

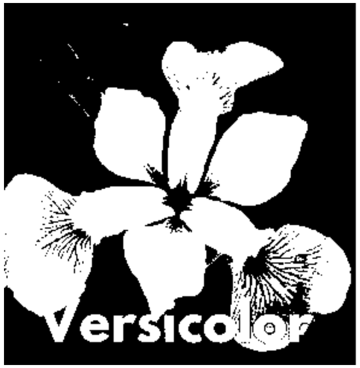
Label: Virginica

Label=F(features)

No obvious way to choose features

Classical image classification: Features

We could try:

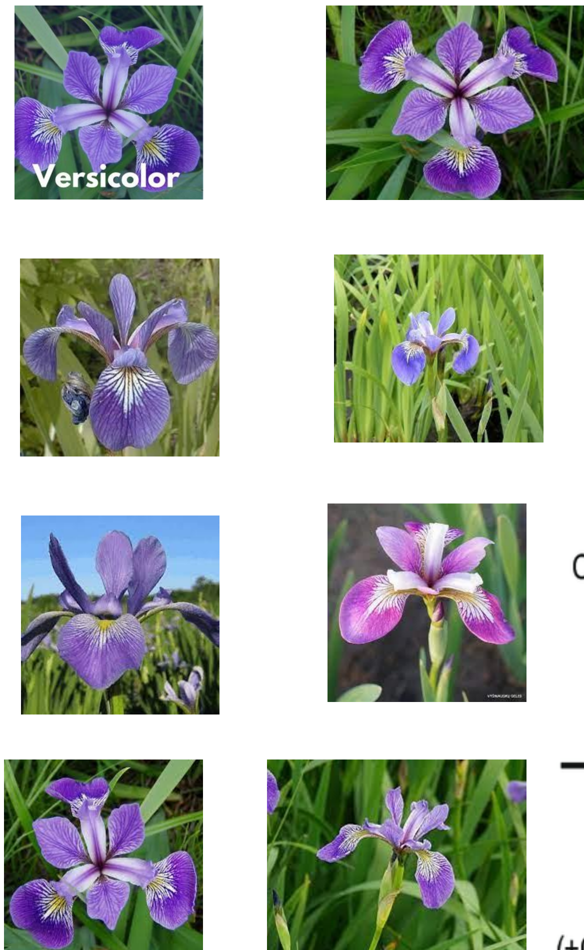


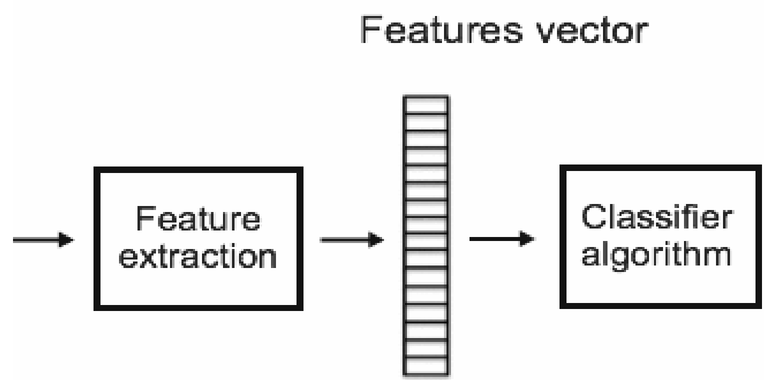
or

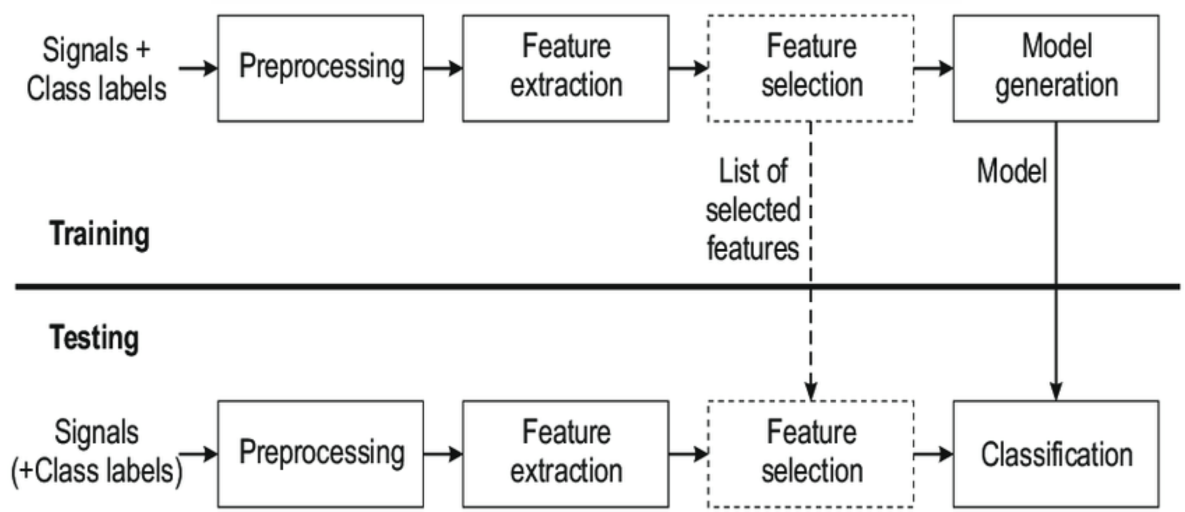
Or…

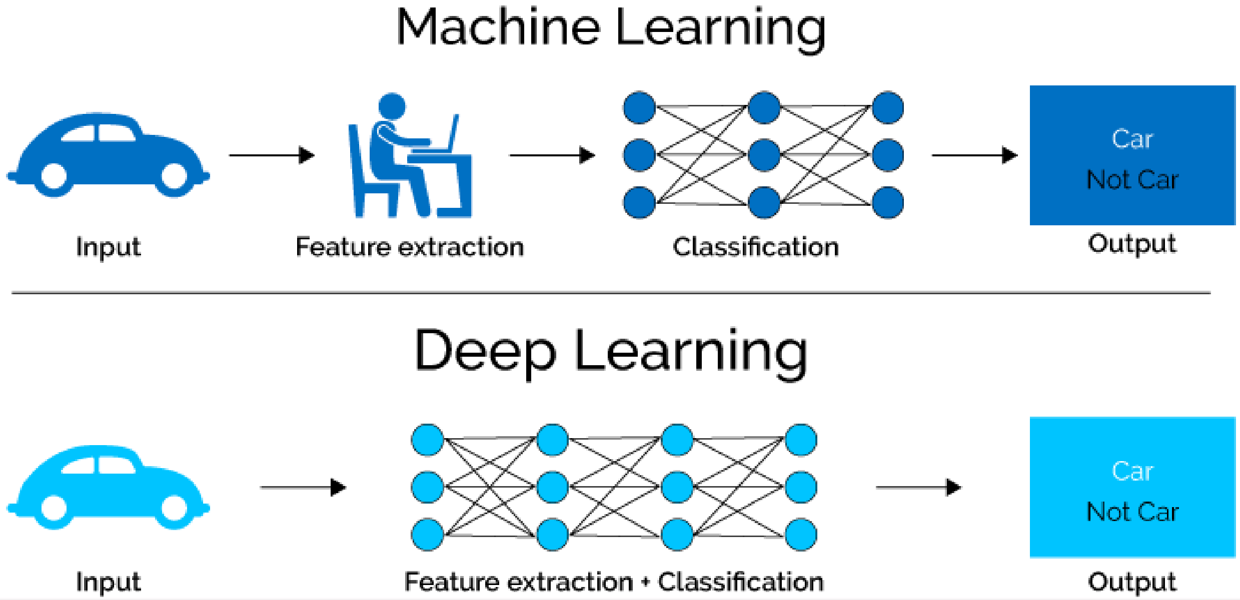
Label: Versicolor

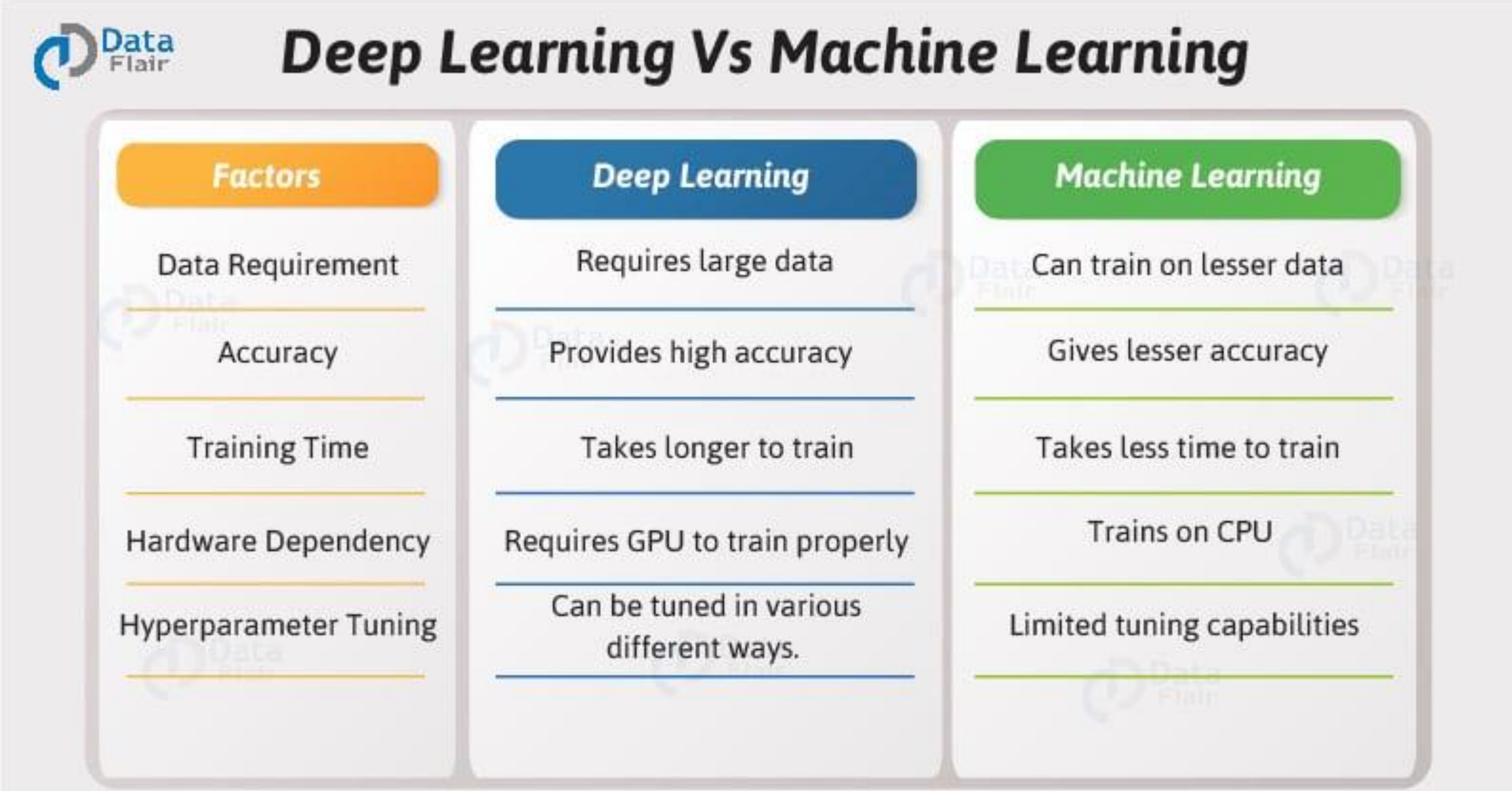
Classical image classification: Features











Deep Learning Vs Machine Learning

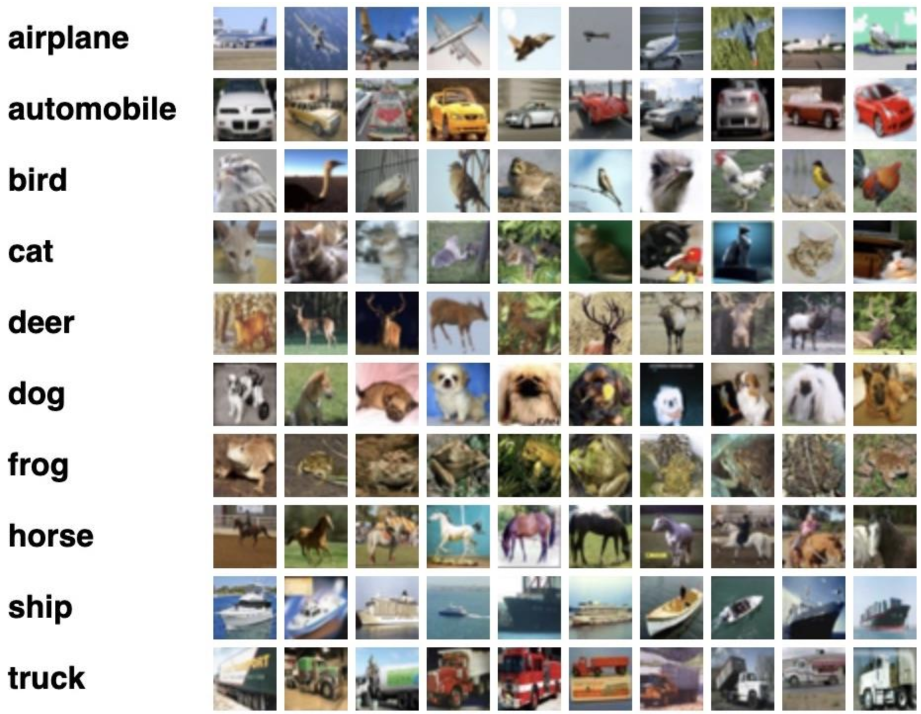
Factors	Deep Learning	Machine Learning
Data Requirement	Requires large data	Can train on lesser data
Accuracy	Provides high accuracy	Gives lesser accuracy
Training Time	Takes longer to train	Takes less time to train
Hardware Dependency	Requires GPU to train properly	Trains on CPU
Hyperparameter Tuning	Can be tuned in various different ways.	Limited tuning capabilities

Image Classification Datasets: MNIST

10 classes: Digits 0 to 9
28x28 grayscale images
50k training images
10k test images

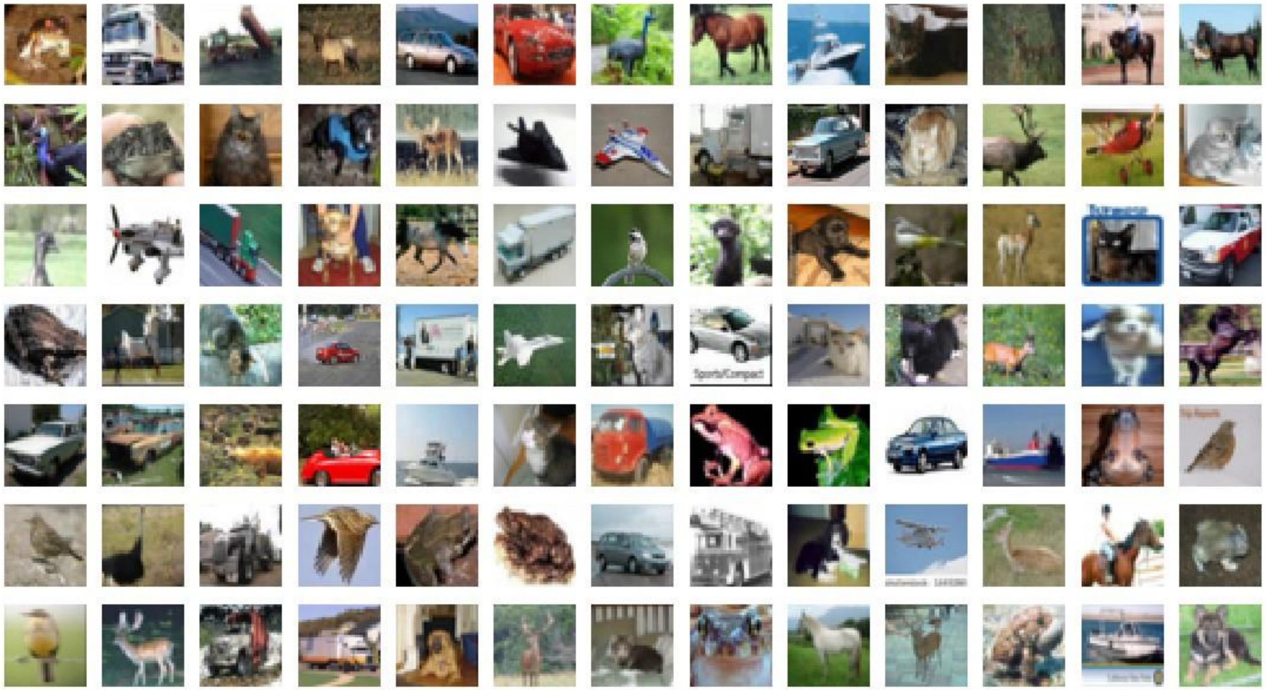
Results from MNIST often do not hold on more complex datasets

Image Classification Datasets: CIFAR10



10 classes
50k training images (5k per class)
10k testing images (1k per class)
32x32 RGB images

Image Classification Datasets: CIFAR100



100 classes
50k training images (500 per class)
10k testing images (100 per class)
32x32 RGB images

20 superclasses with 5 classes each:
Aquatic mammals: beaver, dolphin, otter, seal, whale
Trees: Maple, oak, palm, pine, willow

Image Classification Datasets: ImageNet

1000 classes
~1.3M training images (~1.3K per class)
50K validation images (50 per class)
100K test images (100 per class)
Performance metric: Top 5 accuracy
Algorithm predicts 5 labels for each image; one of them needs to be right

Images have variable size, but often resized to 256x256 for training

Deng et al, “ImageNet: A Large-Scale Hierarchical Image Database”, CVPR 2009 Russakovsky et al, “ImageNet Large Scale Visual Recognition Challenge”, IJCV 2015

Model performance

Accuracy Ratio of correctly predicted observation to the total observations. Accuracy is a good measure but only when values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model.

Accuracy = TP+TN/TP+FP+FN+TN

Precision Ratio of correctly predicted positive observations to the total predicted positive observations. High Precision relates to the low false positive rate.

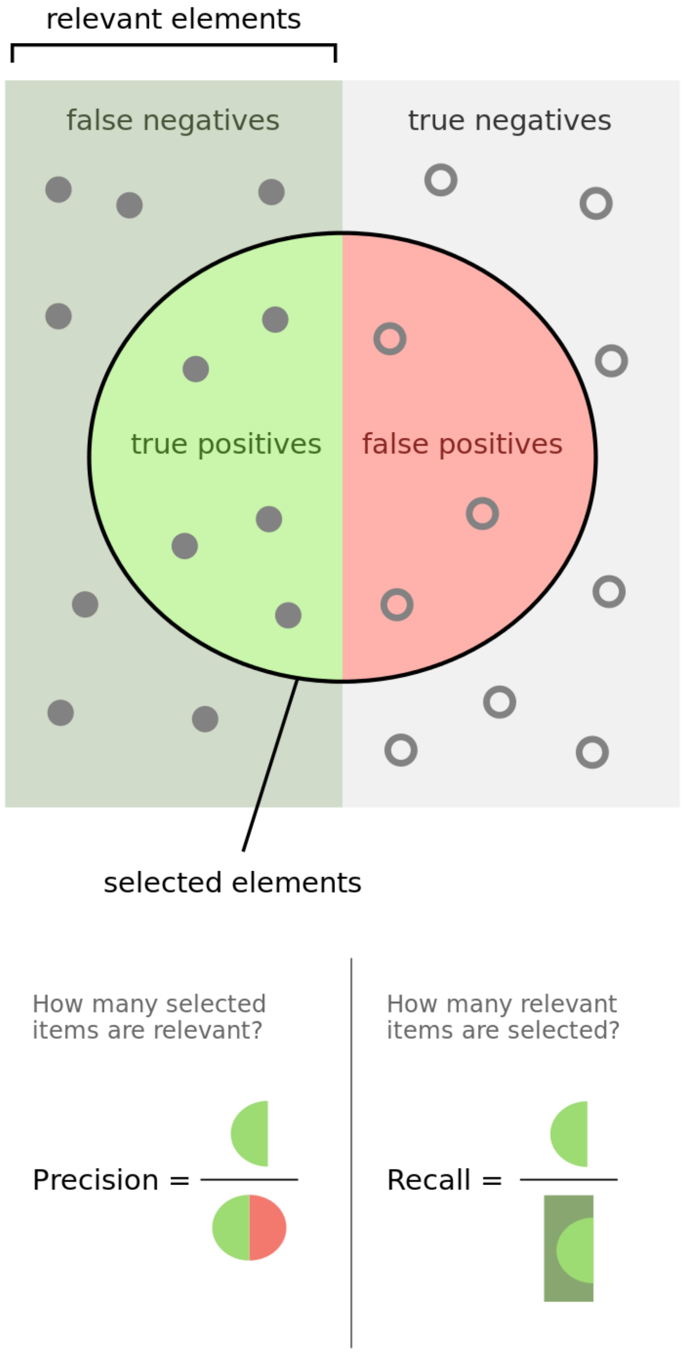
Precision = TP/TP+FP

Recall Ratio of correctly predicted positive observations to the all observations in actual class

Recall = TP/TP+FN

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)



Limitations to classical image classification

- Algorithms are susceptible to variations in color, contrast, lighting, size, angle…
- This can be mitigated by image preprocessing, but can require a lot of work
- Techniques apply to entire array of pixels, but only some pixels describe the object you are trying to classify
- Can be heavily influenced by background pixels that have little or no effect on the class of object

Limitations to classical image classification

Intraclass Variation

Limitations to classical image classification

Fine-Grained Categories

Maine Coon

Ragdoll

American Shorthair

Limitations to classical image classification

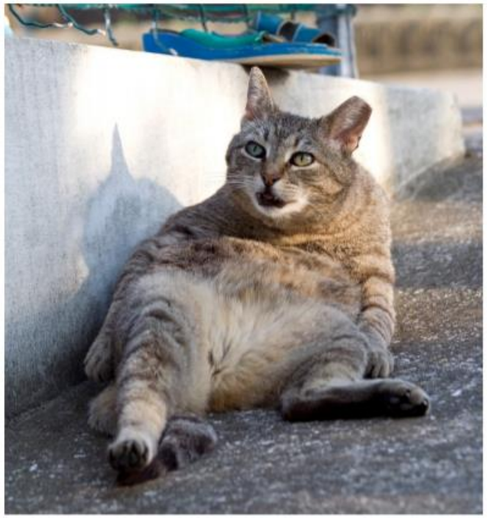
Background Clutter

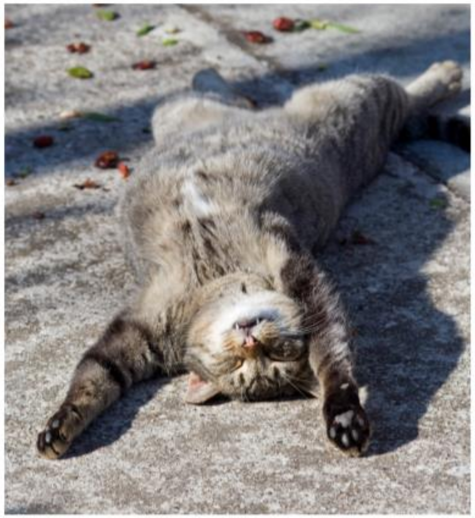
Limitations to classical image classification

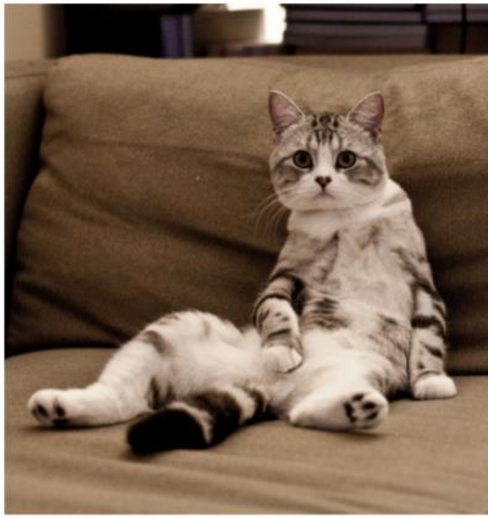
Illumination Changes

Limitations to classical image classification

Deformation







Limitations to classical image classification

Occlusion

Classical image classification in Python

Classical image classification in Python

```python
def prep_data (folder):
    # iterate through folders, assembling feature, label, and classname data objects
    import os
    import numpy as np
    import matplotlib.pyplot as plt

    class_id = 0
    features = []
    labels = np.array([])
    classnames = []
    for root, dirs, filenames in os.walk(folder):
        for d in sorted(dirs):
            print("Reading data from", d)
            # use the folder name as the class name for this label
            classnames.append(d)
            files = os.listdir(os.path.join(root,d))
            for f in files:
                # Load the image file
                imgFile = os.path.join(root,d, f)
                img = plt.imread(imgFile)
                # The image array is a multidimensional numpy array
                # - flatten it to a single array of pixel values for scikit-learn
                # - and add it to the list of features
                features.append(img.ravel())

                # Add it to the numpy array of labels
                labels = np.append(labels, class_id )
            class_id  += 1

    # Convert the list of features into a numpy array
    features = np.array(features)

    return features, labels, classnames


# The images are in a folder named 'shapes/training'
training_folder_name = '../data/shapes/training'

# Prepare the image data
features, labels, classnames = prep_data(training_folder_name)
print(len(features), 'features')
print(len(labels), 'labels')
print(len(classnames), 'classes:', classnames)
```

numpy.ravel

numpy.ravel(*a*, *order='C'*) [source]

Return a contiguous flattened array.

A 1-D array, containing the elements of the input, is returned. A copy is made only if needed.

example

```
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> np.ravel(x)
array([1, 2, 3, 4, 5, 6])
```

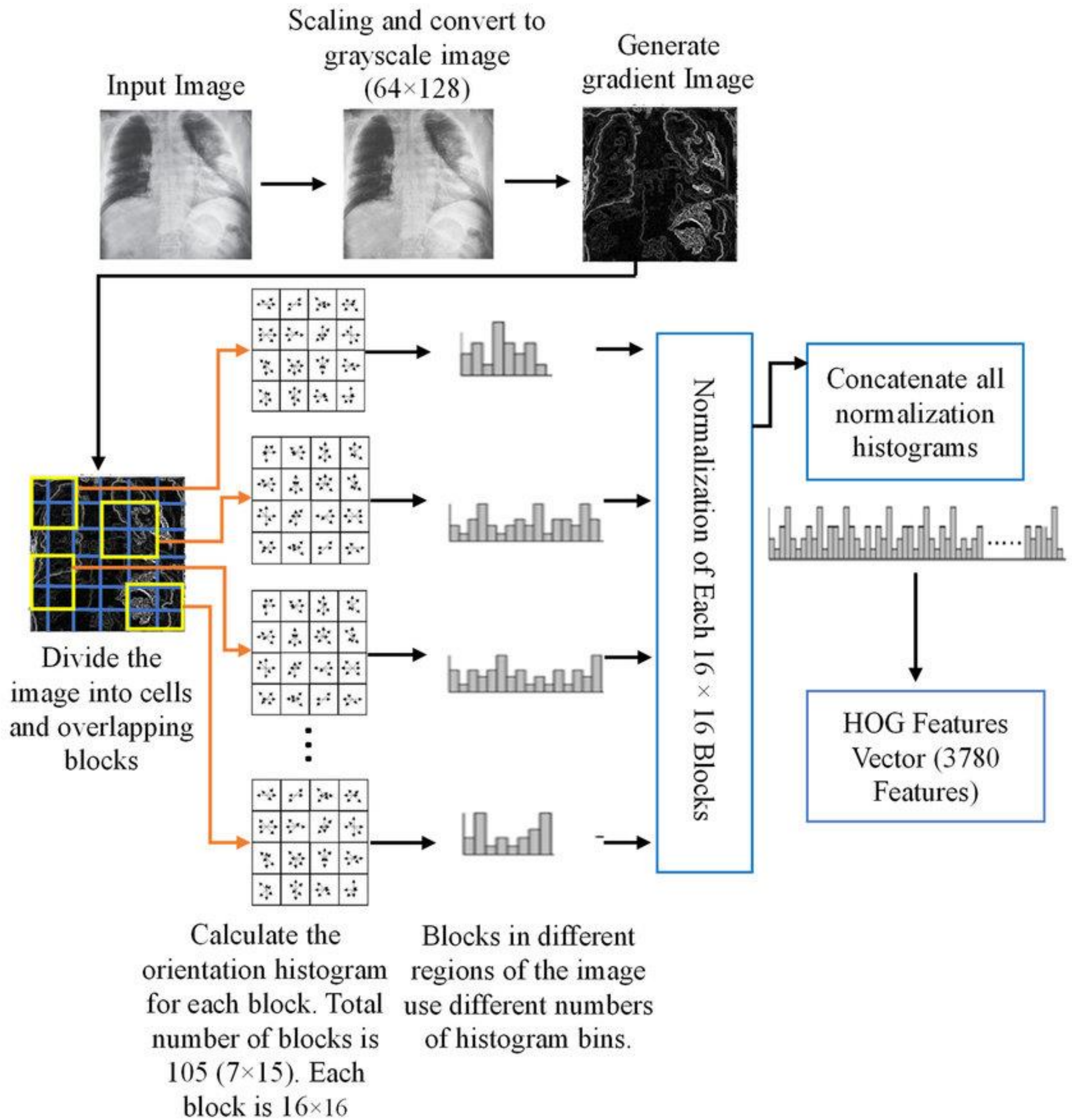
Chapter Summary

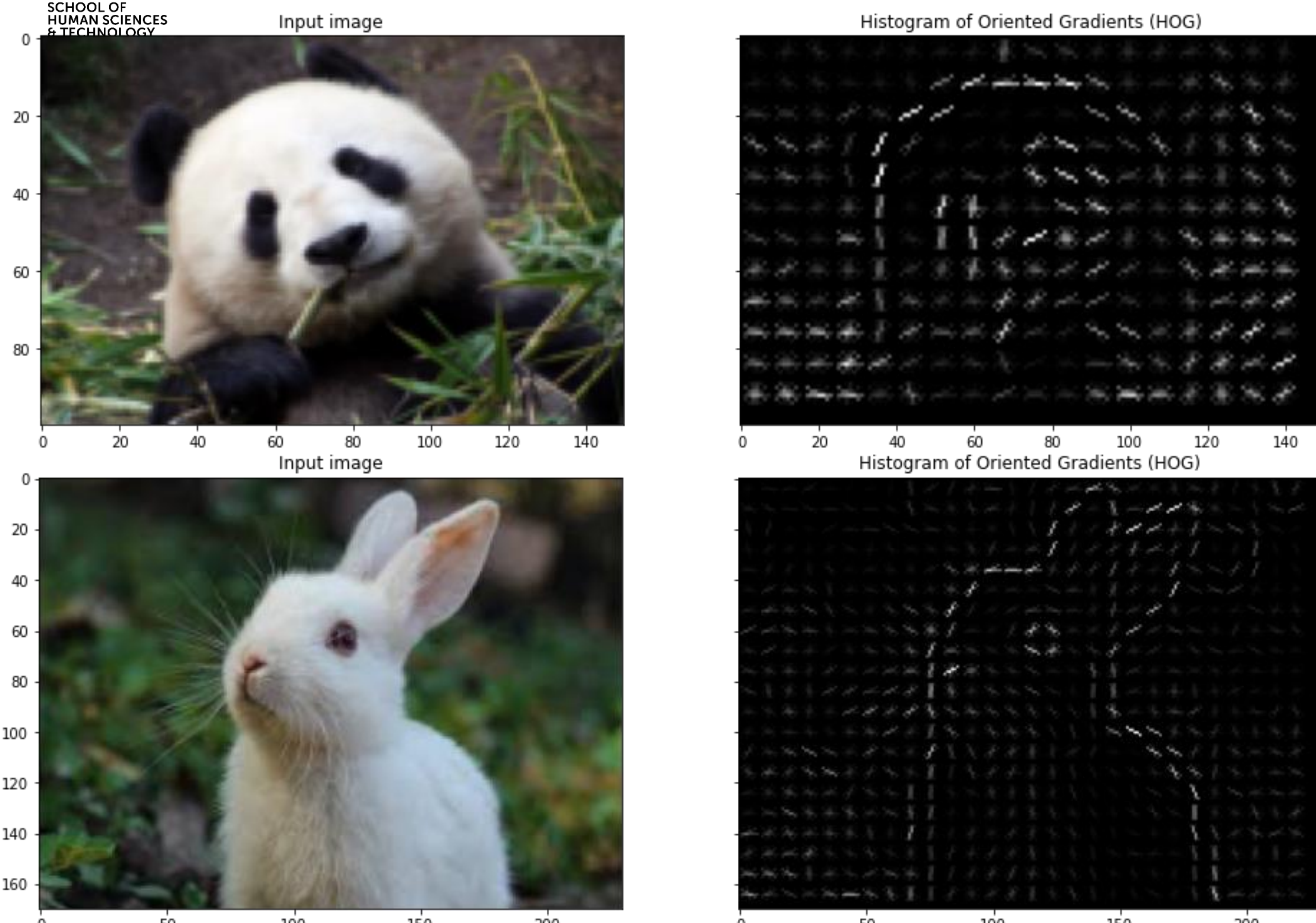
- We reviewed the concept of image classification
- We saw some example commonly used image datasets
- We saw how to build an image classification pipeline in python
- We experimented with the concept of feature engineering
- We applied concepts learned in previous session for feature engineering task

HOG Features: Histogram of Oriented Gradients

Definition

The **histogram of oriented gradients (HOG)** is a feature descriptor used in computer vision and image processing for the purpos of object detection. The technique counts occurrences of gradient orientation in localized portions of an image.

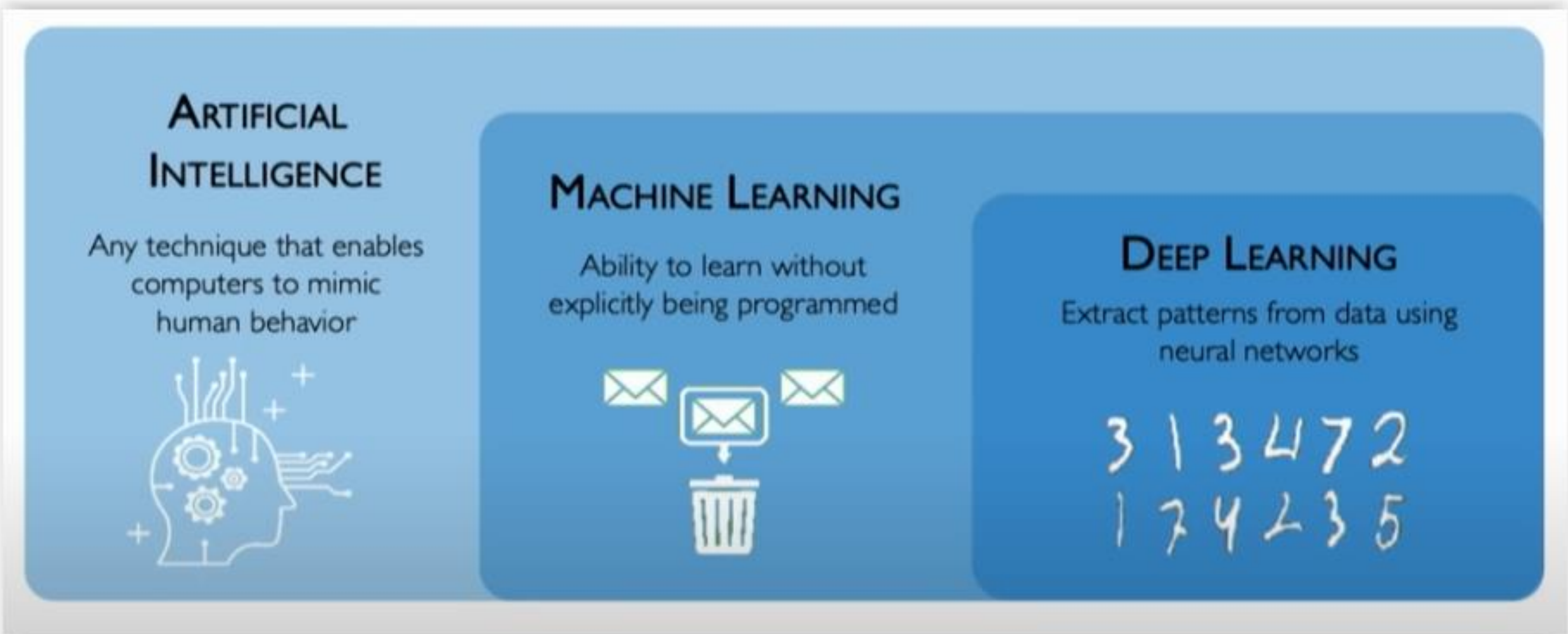



Input image
Histogram of Oriented Gradients (HOG)
0
20
40
60
80
0 20 40 60 80 100 120 140
0 20 40 60 80 100 120 140
Input image
Histogram of Oriented Gradients (HOG)
0
20
40
60
80
100
120
140
160

SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

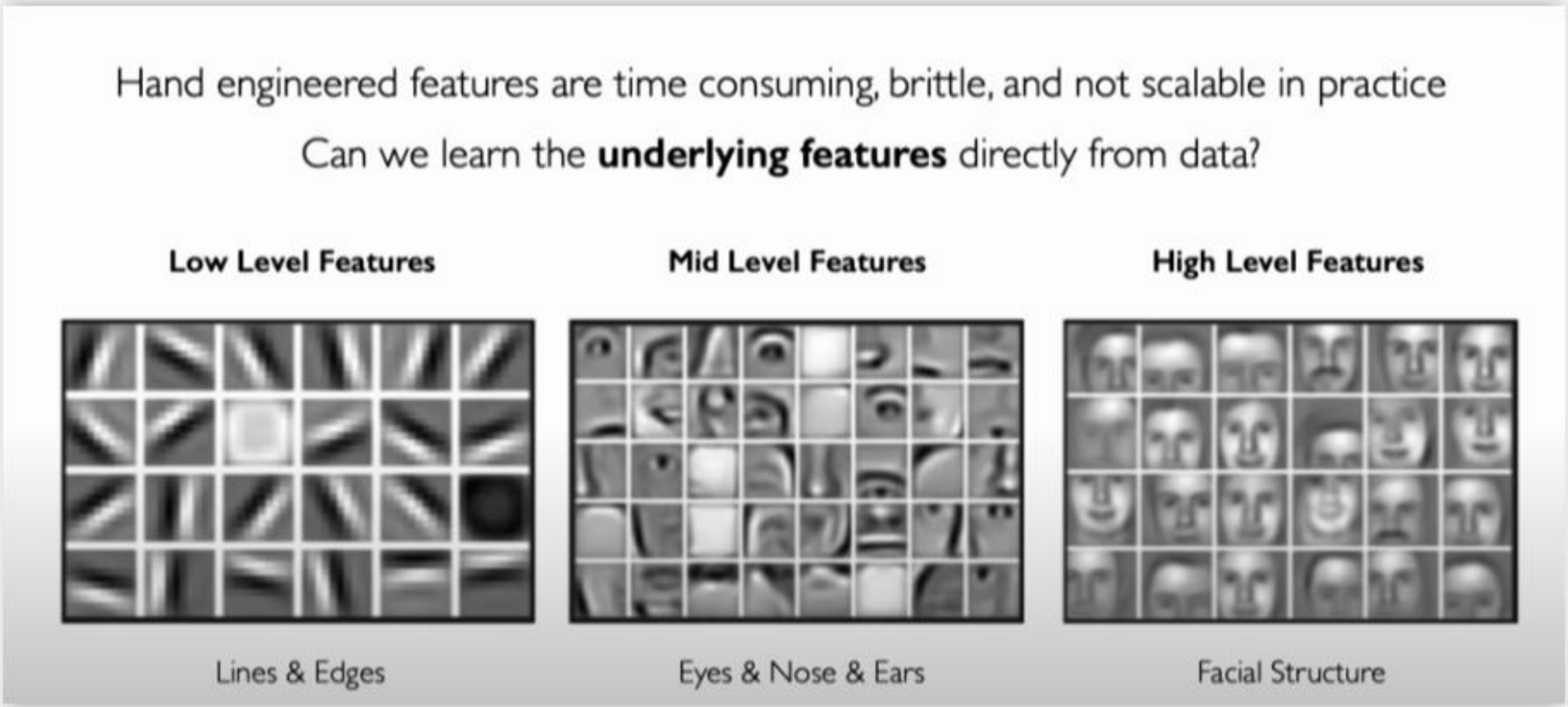
The basic concepts of Neural Networks

What is Deep Learning?

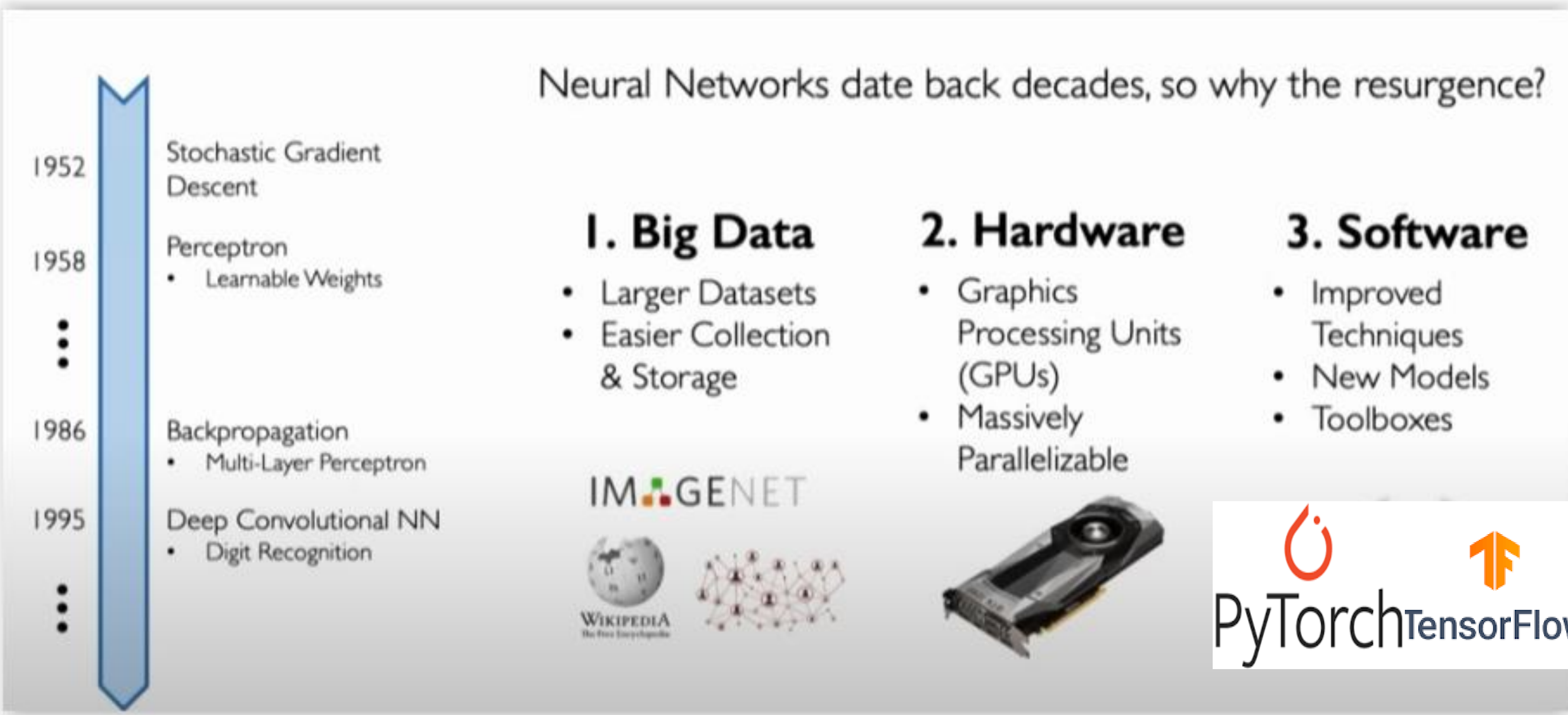


Teaching computers how to **learn a task** directly from **raw data**

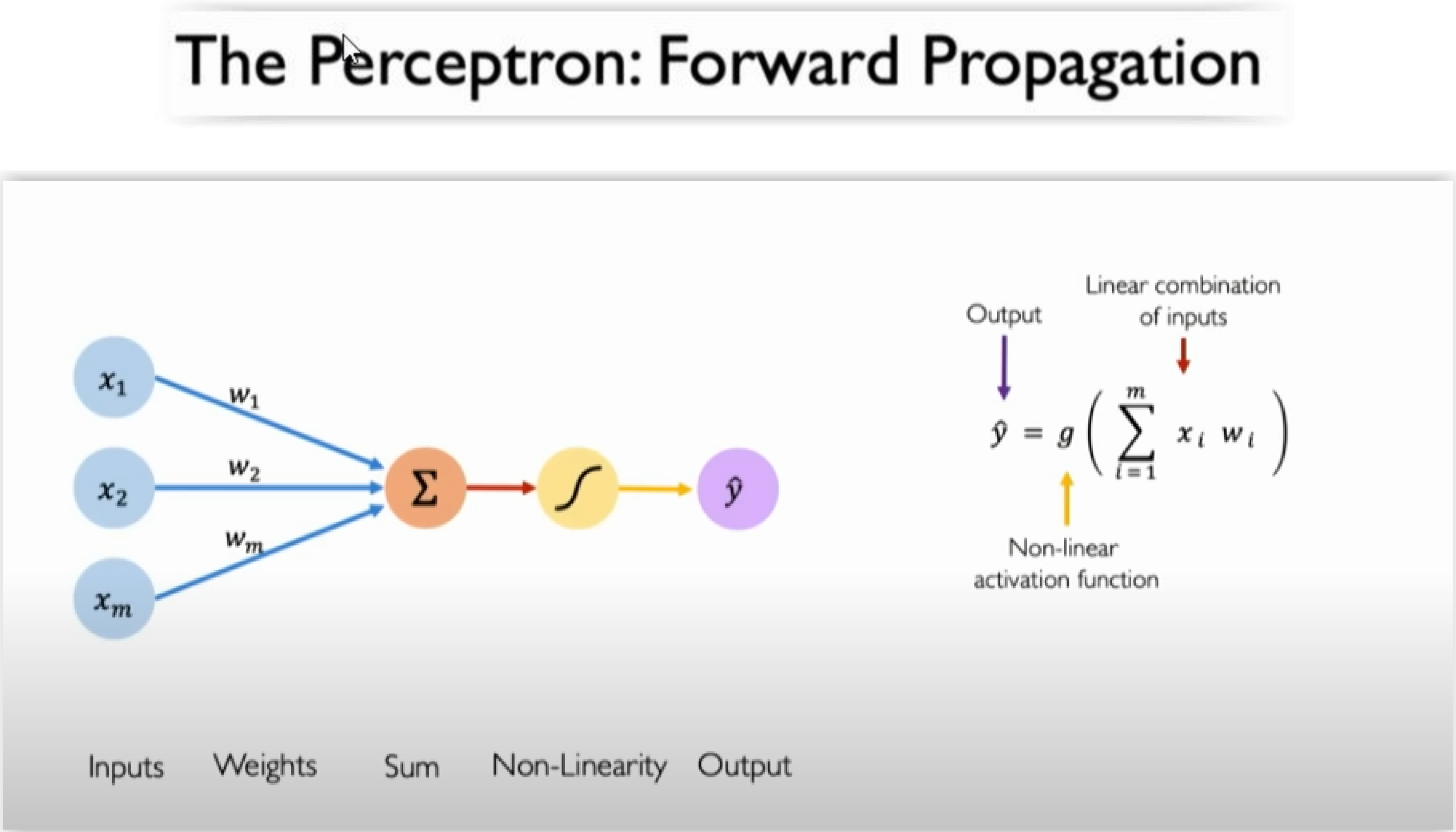
Why Deep Learning, and Why Now?



Why Now?

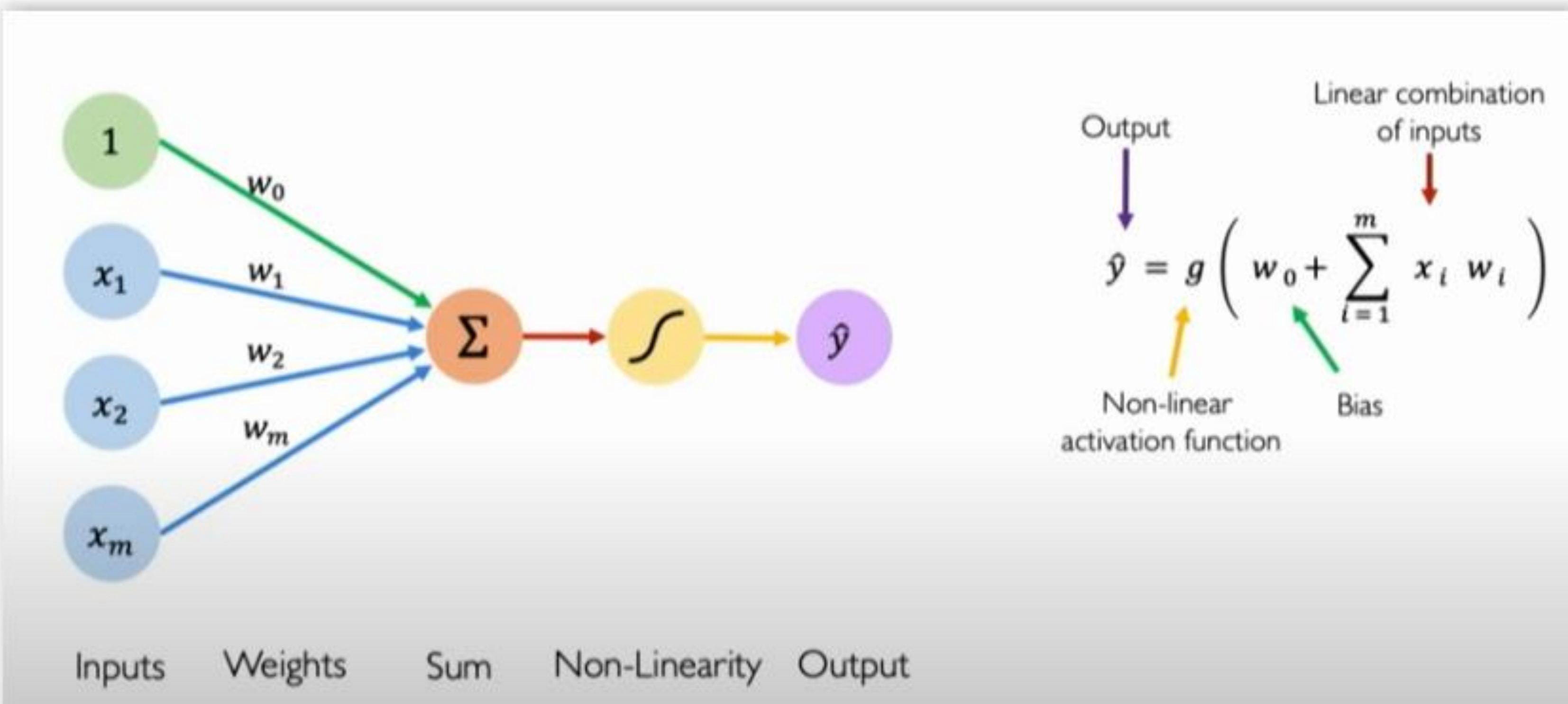

Neural Networks date back decades, so why the resurgence?
1952
Stochastic Gradient Descent
1958
Perceptron
• Learnable Weights
1986
Backpropagation
• Multi-Layer Perceptron
1995
Deep Convolutional NN
• Digit Recognition
1. Big Data
• Larger Datasets
• Easier Collection & Storage
2. Hardware
• Graphics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

The perceptron. The structural building block of deep learning

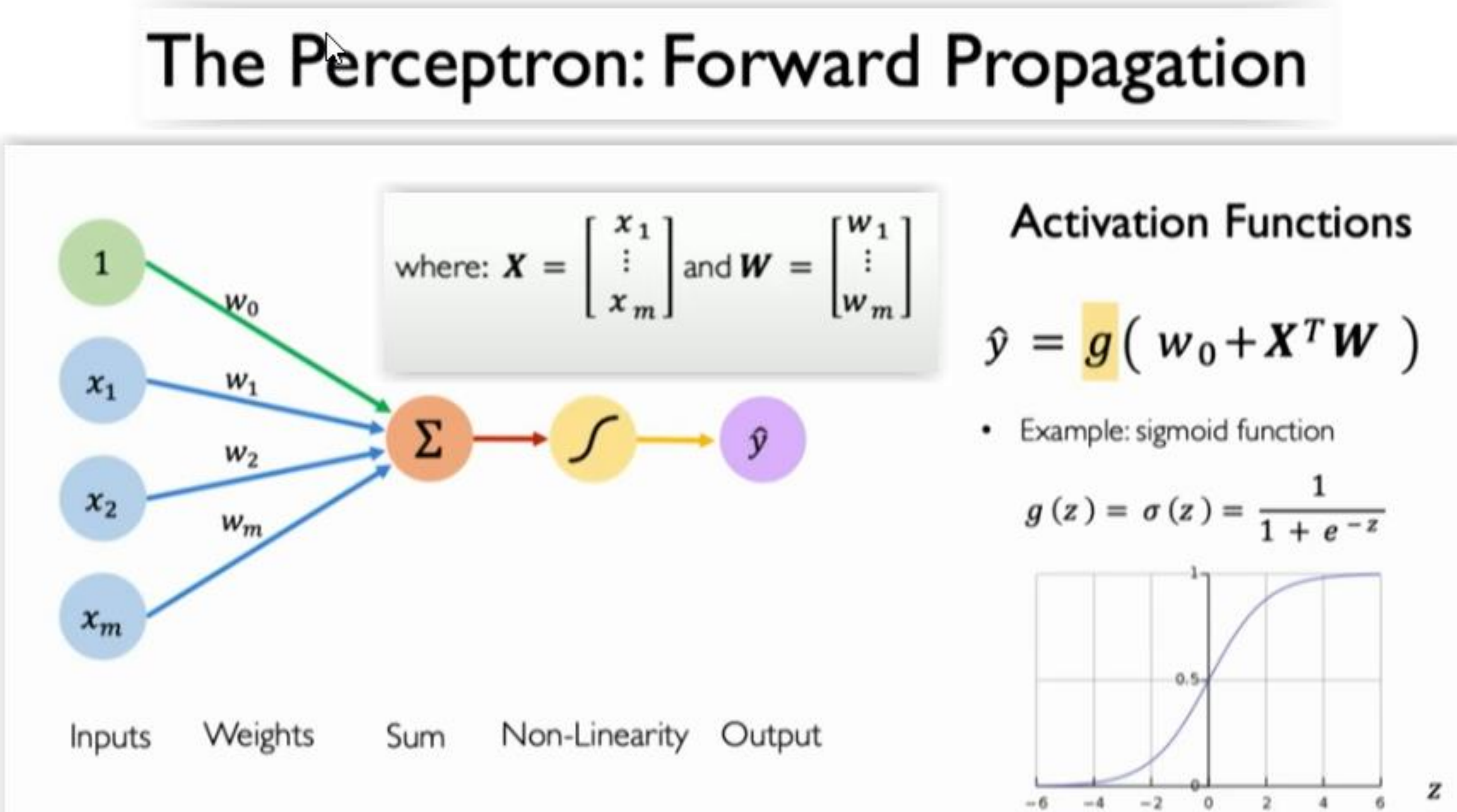


The perceptron. The structural building block of deep learning

The Perceptron: Forward Propagation



The perceptron. The structural building block of deep learning

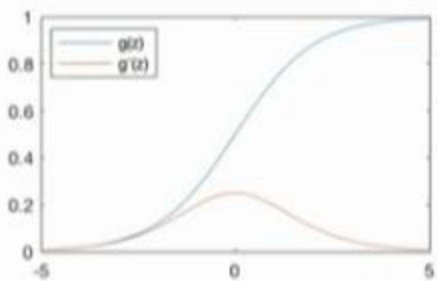


ie
SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

The perceptron. The structural building block of deep learning

Common Activation Functions

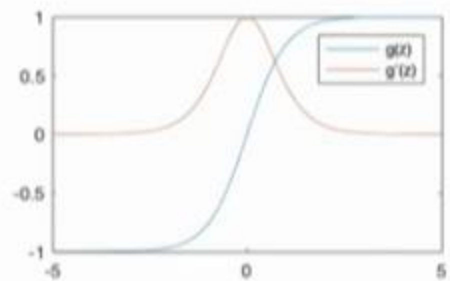
Sigmoid Function



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

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

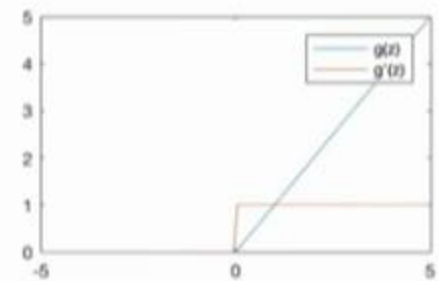
Hyperbolic Tangent



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

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

Rectified Linear Unit (ReLU)



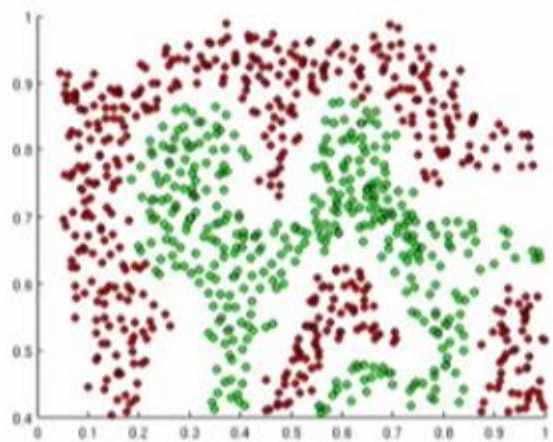
$$g(z) = \max(0, z)$$

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

The perceptron. The structural building block of deep learning

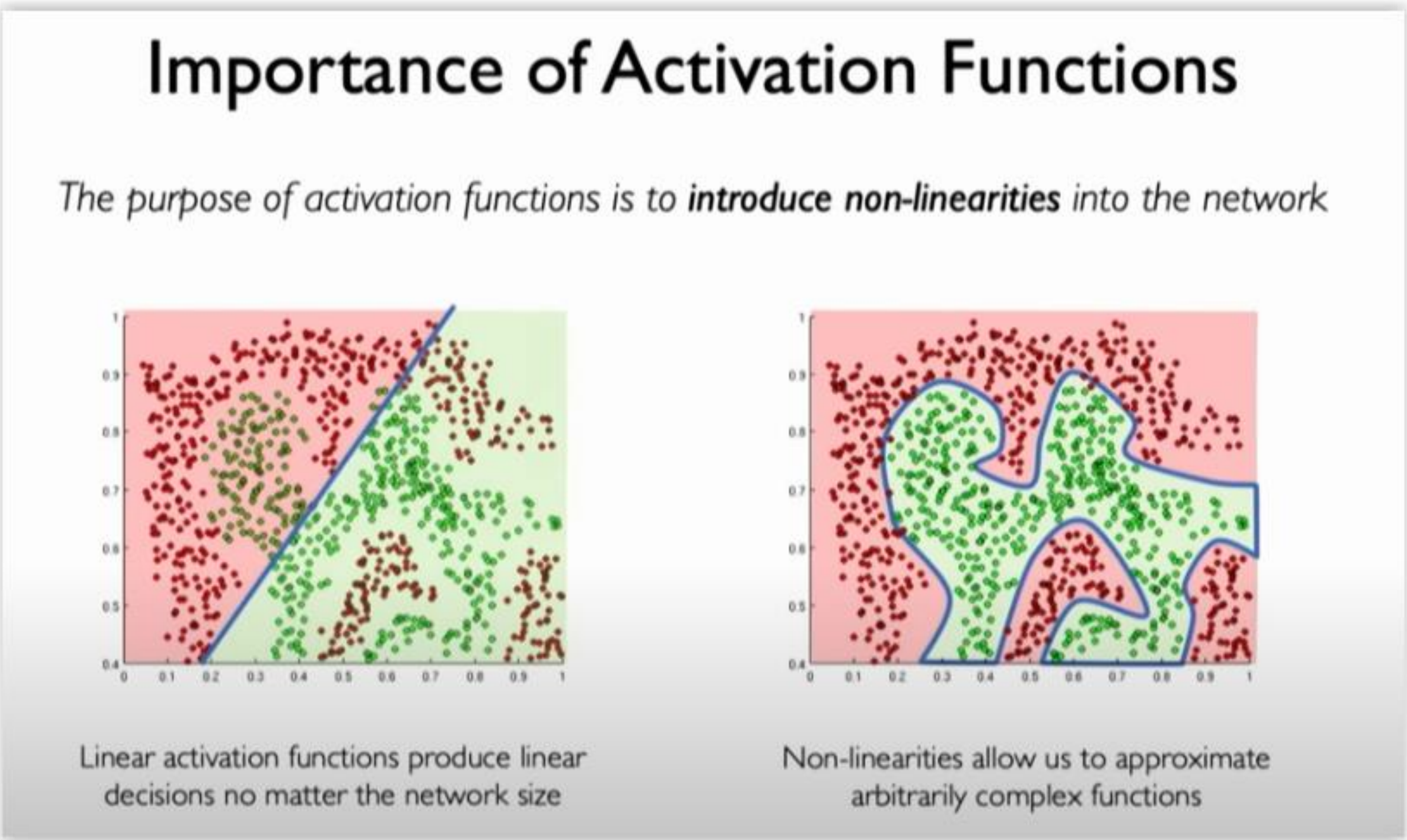
Importance of Activation Functions

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



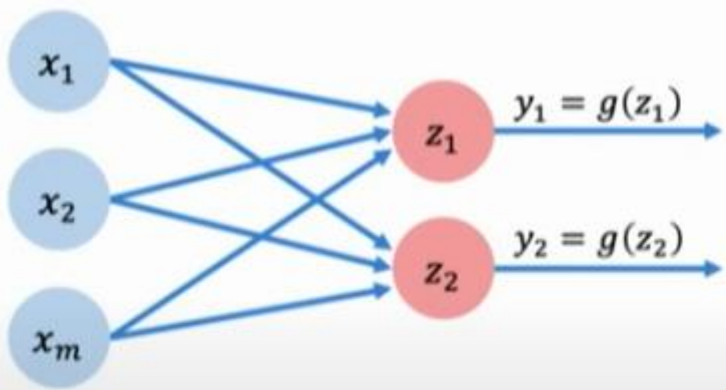
What if we wanted to build a neural network to distinguish green vs red points?

The perceptron. The structural building block of deep learning



Building Neural Networks With Perceptrons

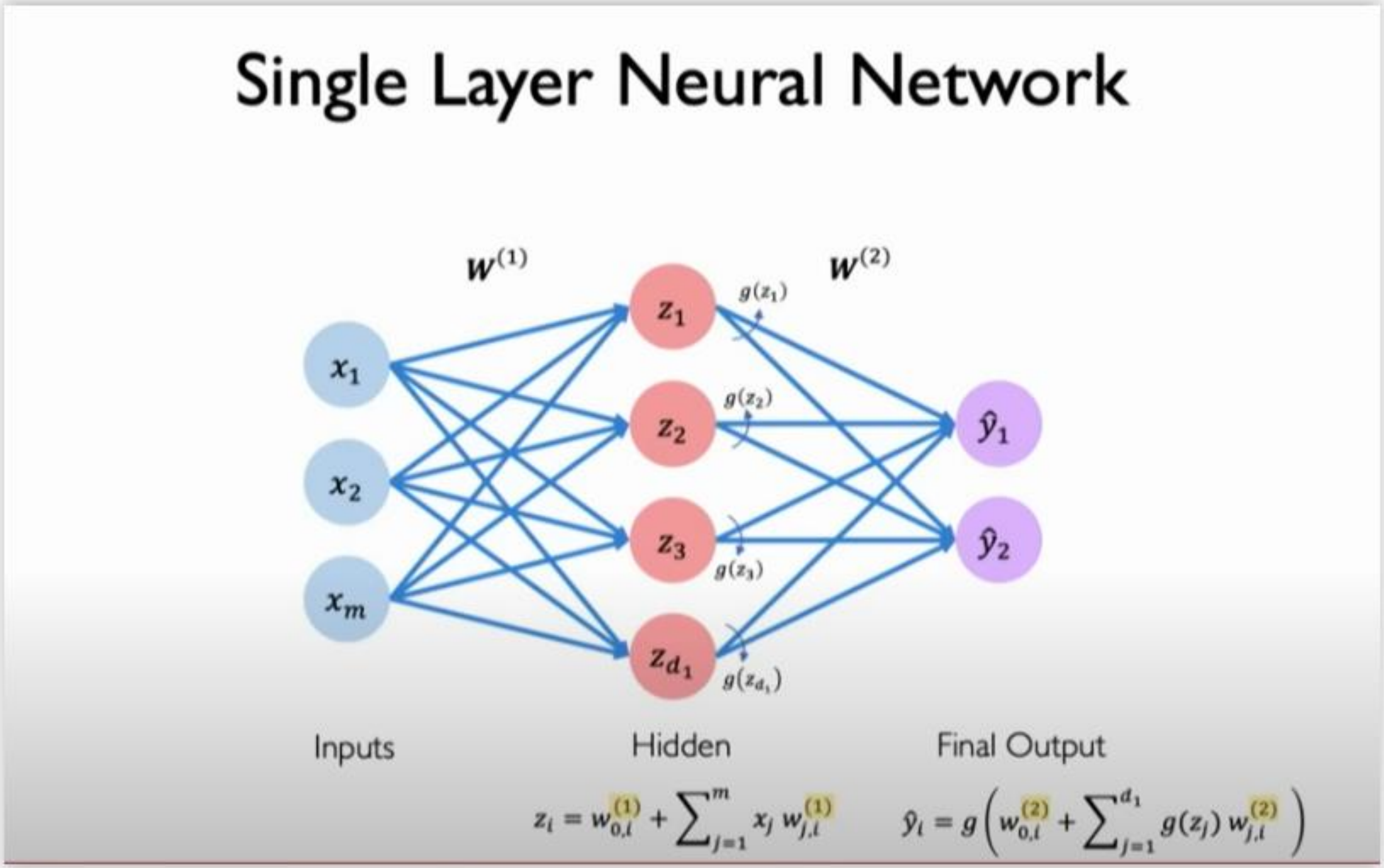
Multi Output Perceptron



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

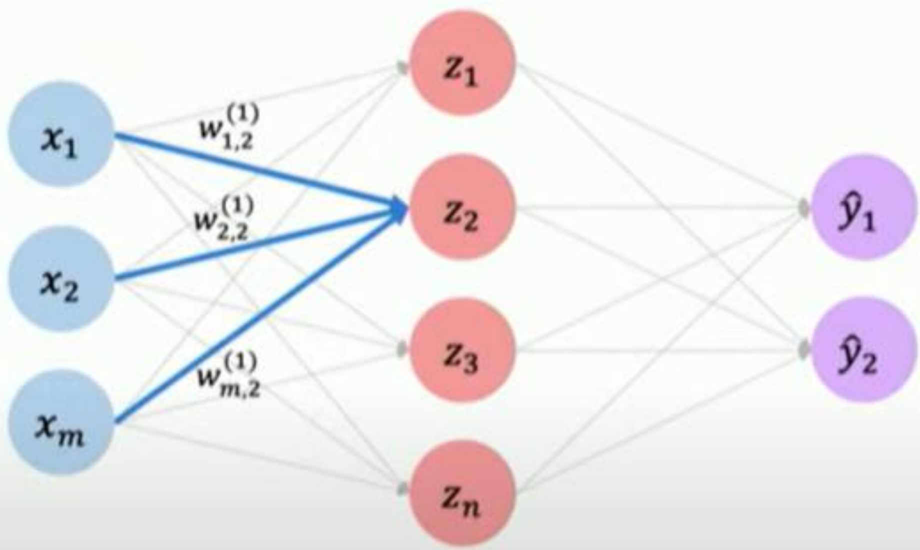
SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Building Neural Networks With Perceptrons



Building Neural Networks With Perceptrons

Single Layer Neural Network

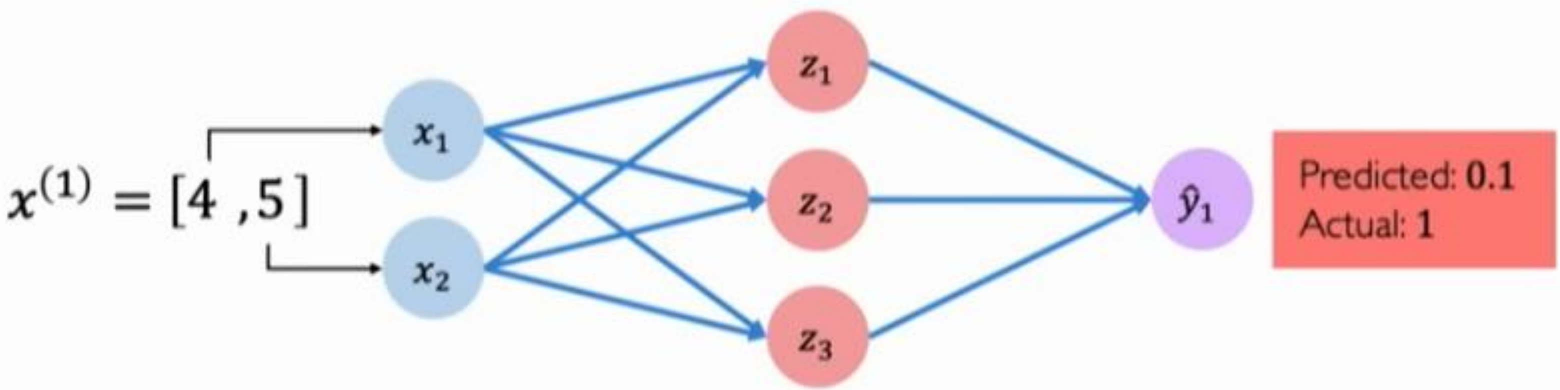


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

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

Quantifying Loss

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

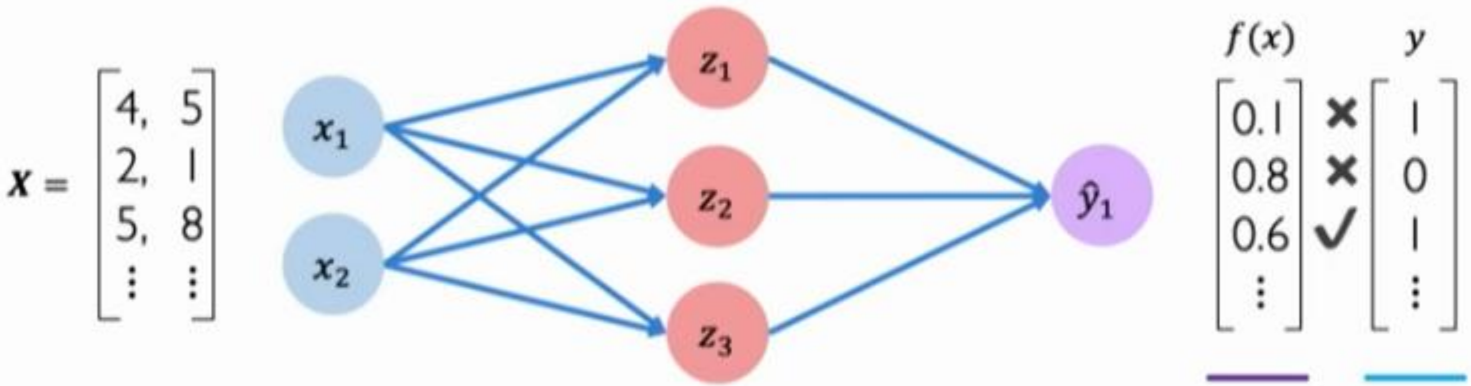


$$\mathcal{L}\big(\underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}}\big)$$

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Empirical Loss

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



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

Also known as:

- Objective function
- Cost function
- Empirical Risk

Training A Neural Network

Loss Optimization

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

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

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

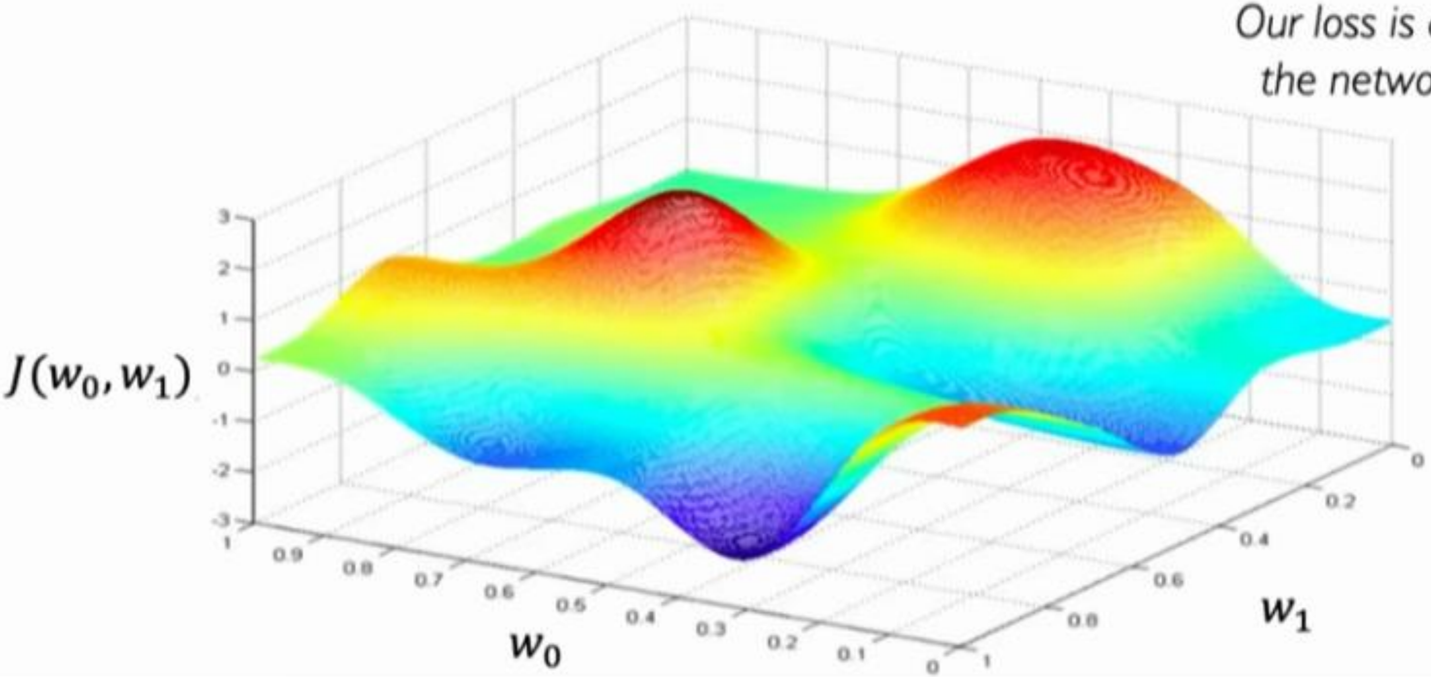
Remember:

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

Loss Optimization

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

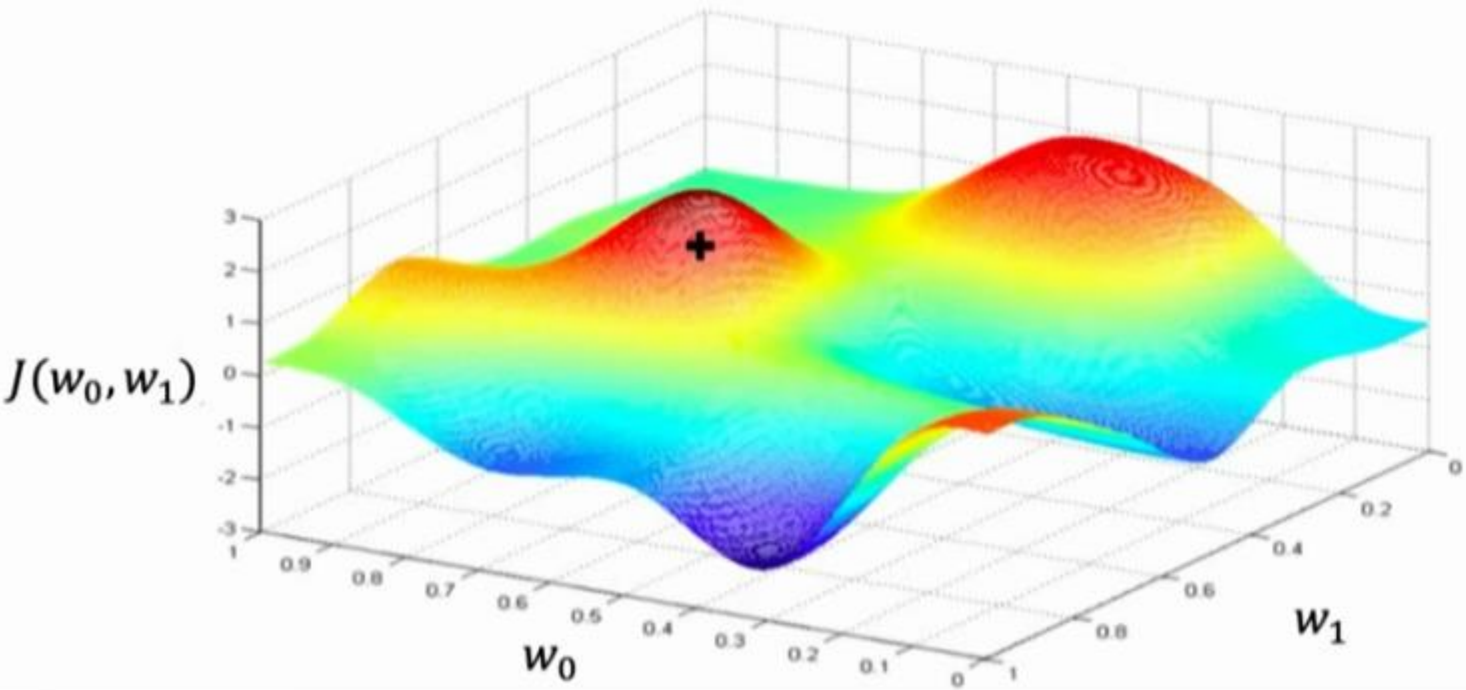
Remember:
Our loss is a function of the network weights!



SCHOOL OF
HUMAN SCIENCES
& TECHNOLOGY

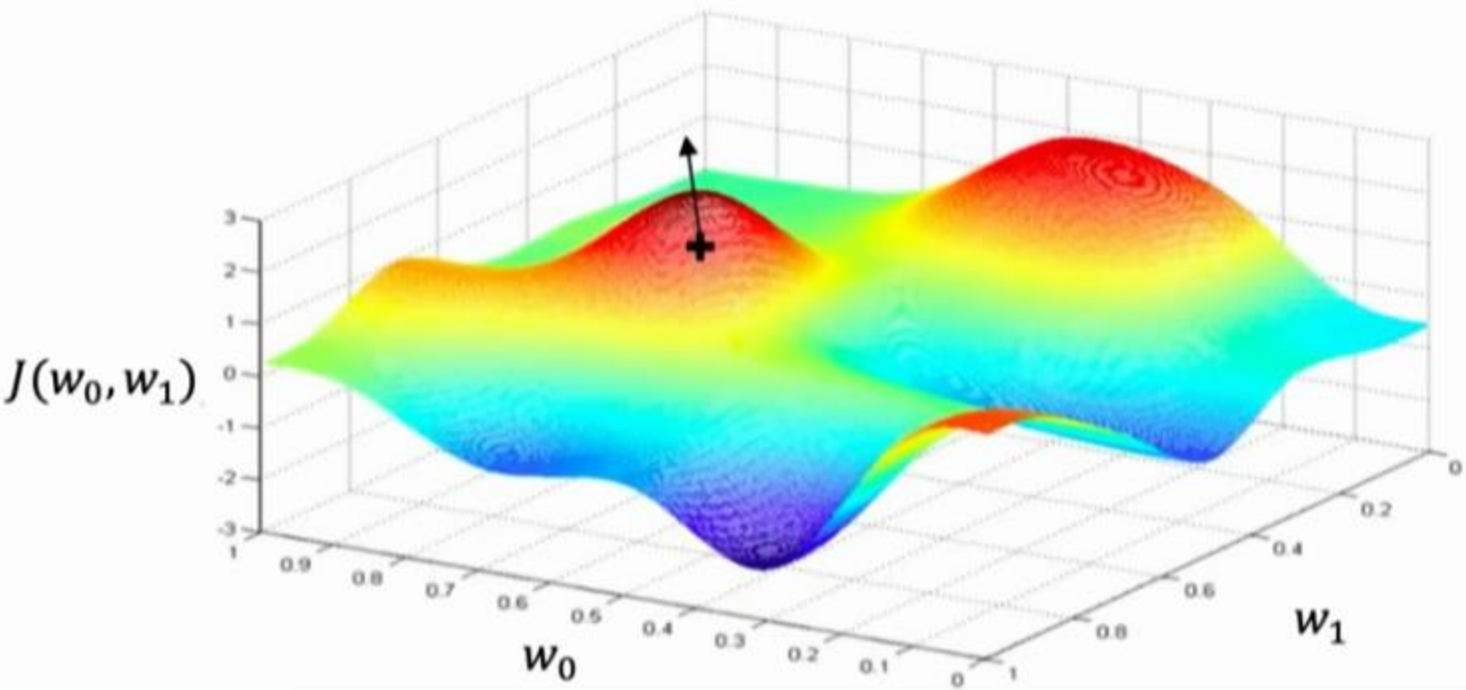
Loss Optimization

Randomly pick an initial (w_0, w_1)



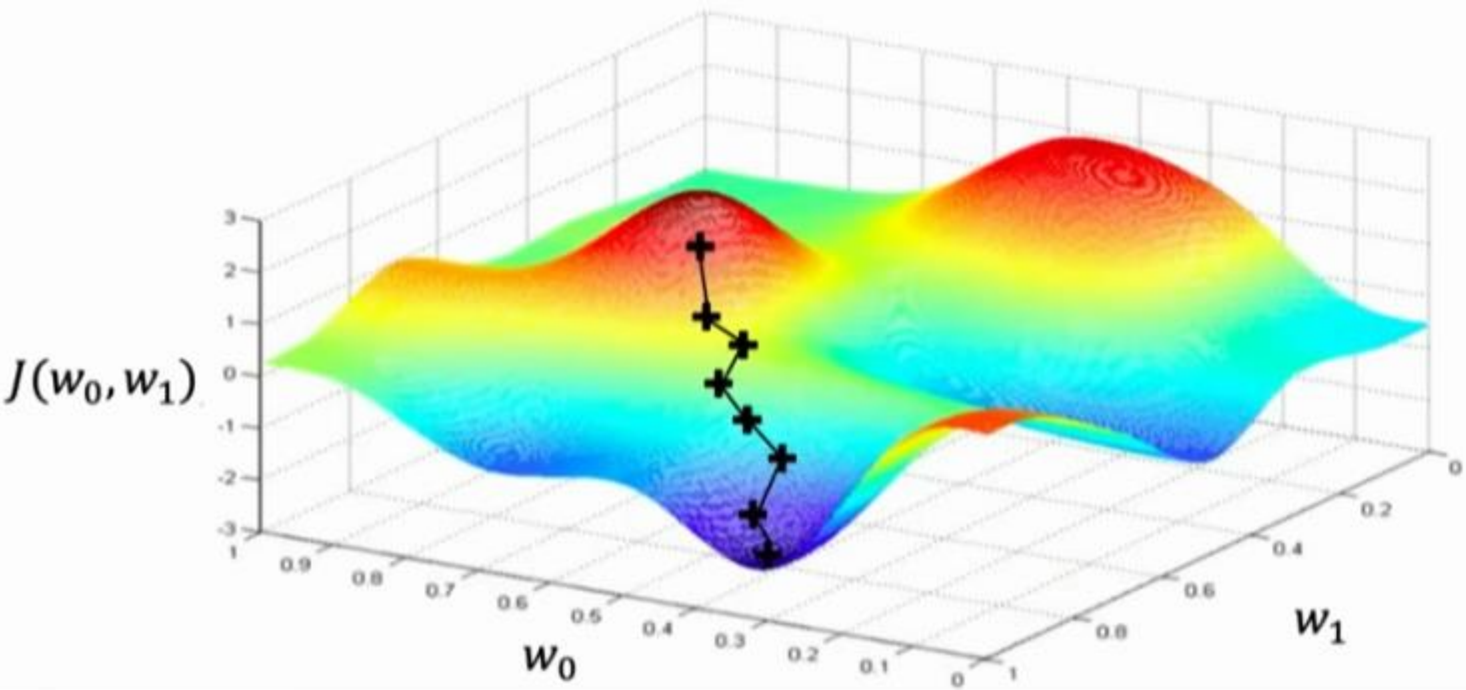
Loss Optimization

Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$



Gradient Descent

Repeat until convergence



Gradient Descent

Algorithm

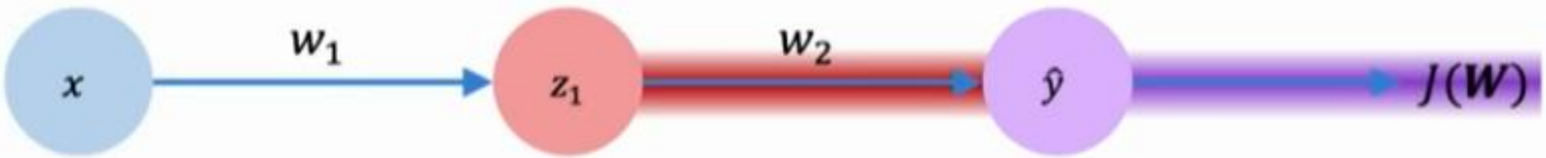
1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\boxed{\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$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

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Computing Gradients: Backpropagation

How does a small change in one weight (ex. $\mathbf{w_2}$*) affect the final loss* $J(\boldsymbol{W})$*?*

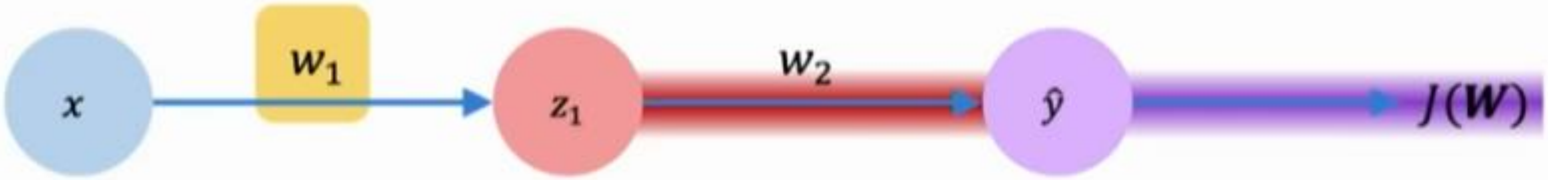
Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

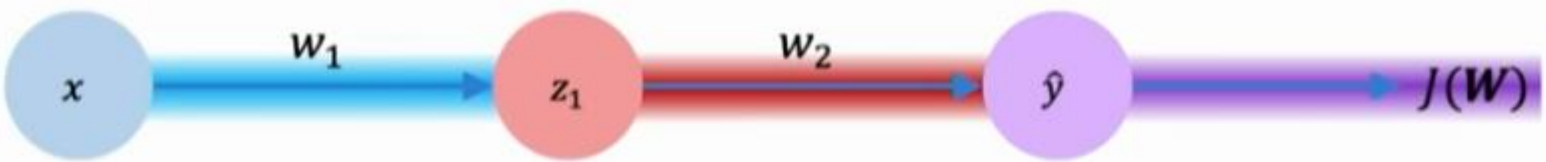
Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule! Apply chain rule!

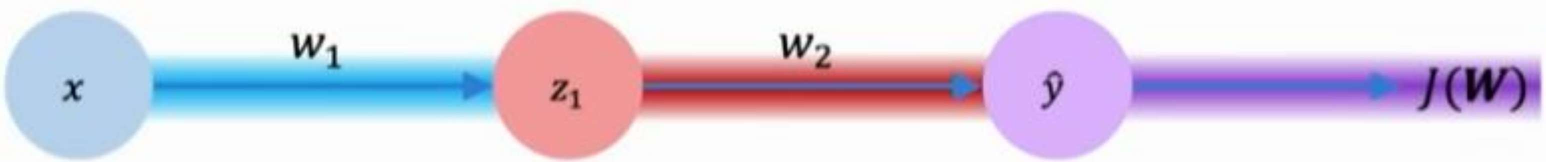
Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

SCHOOL OF HUMAN SCIENCES & TECHNOLOGY

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*