Neural Networks

UNIVERSITY OF SUSSEX

Alejandra Carriero

Recap

Yesterday we introduced different models.

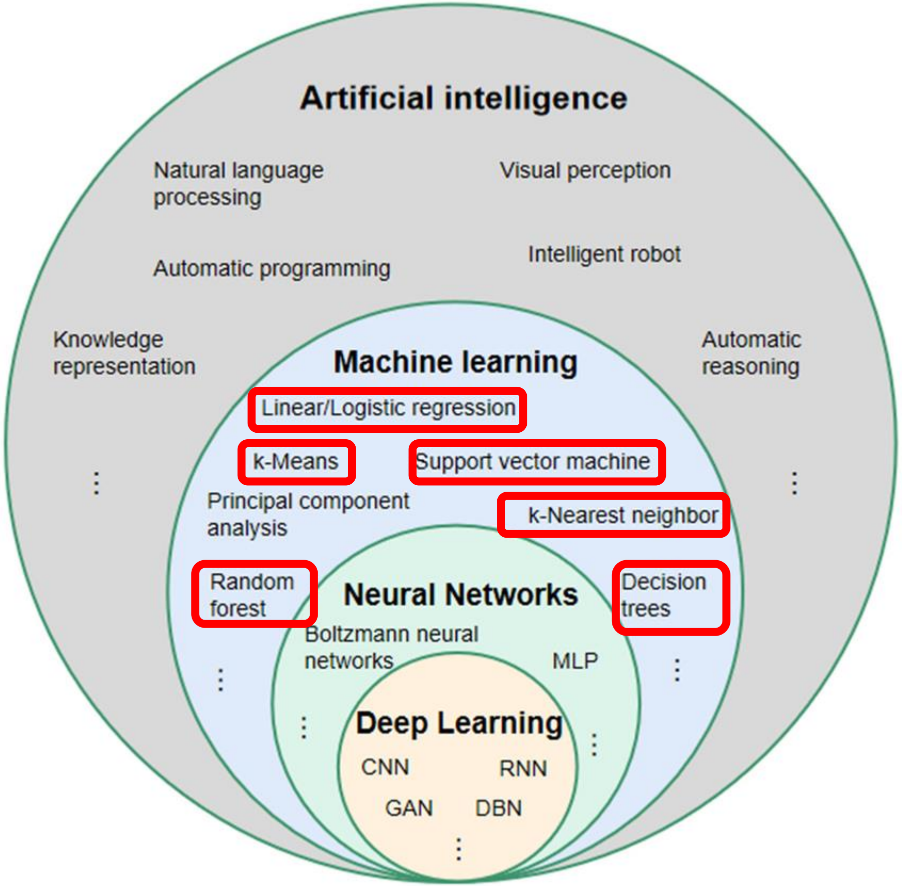
For **classification**:

- *SVM*
- *Random Forest*

For **clustering**:

- *K-means*

UNIVERSITY OF SUSSEX



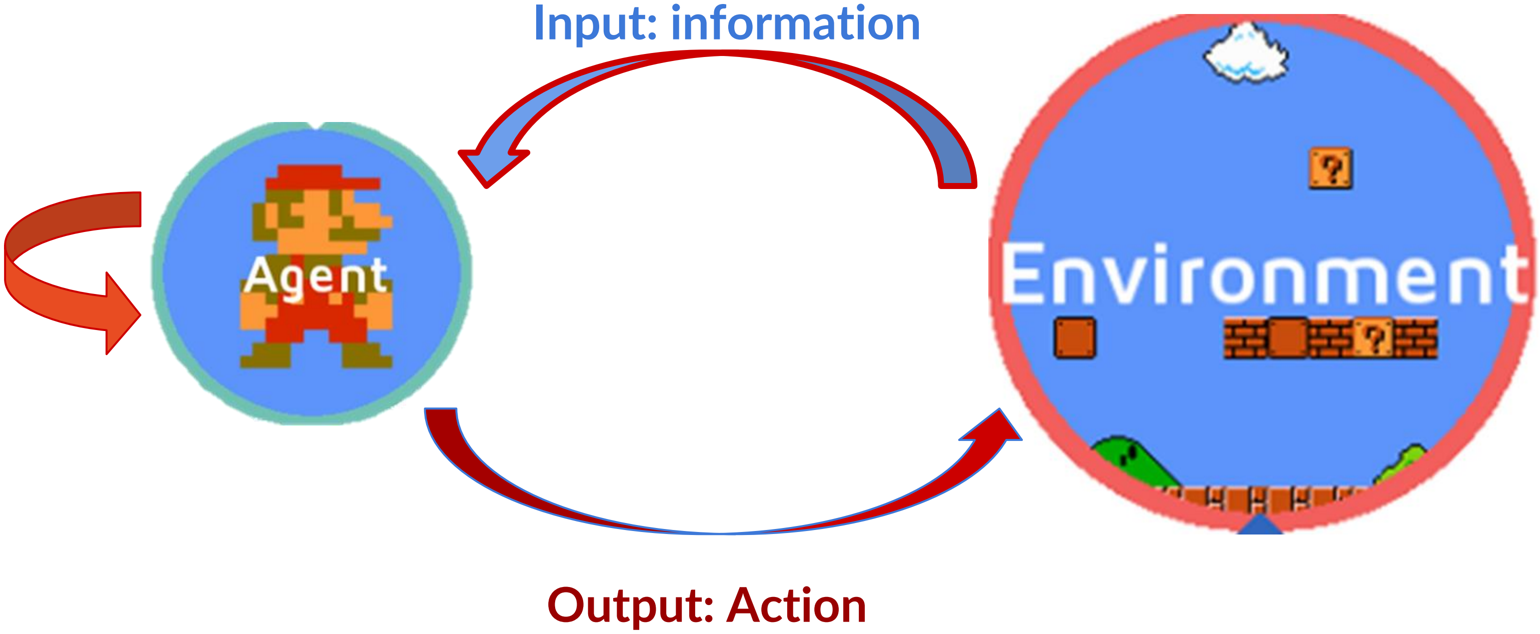
UNIVERSITY OF SUSSEX

Contents

- What are Neural Networks
- Artificial Neurons
- Training Neural Networks
- Some types of Neural Networks (CNNs, RNNs, LSTMs)

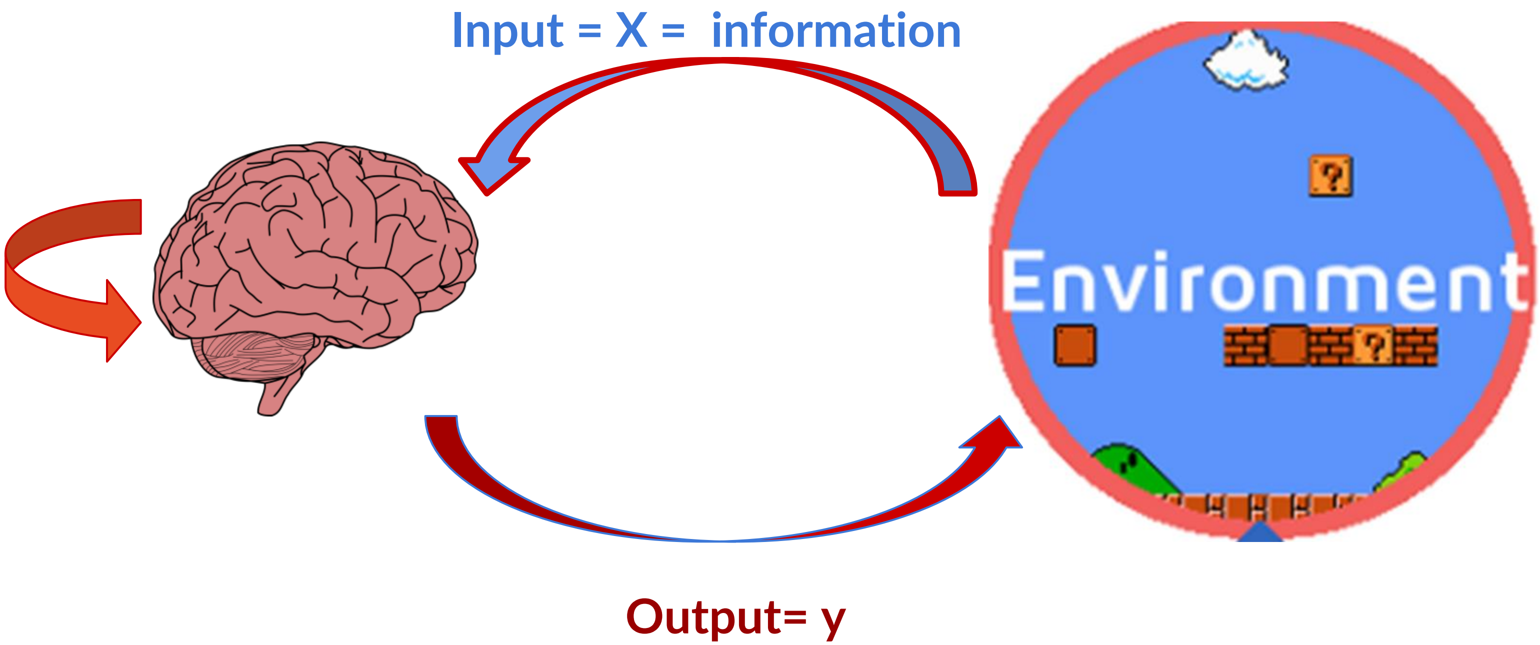
UNIVERSITY OF SUSSEX

Neurons and Artificial Neural Networks

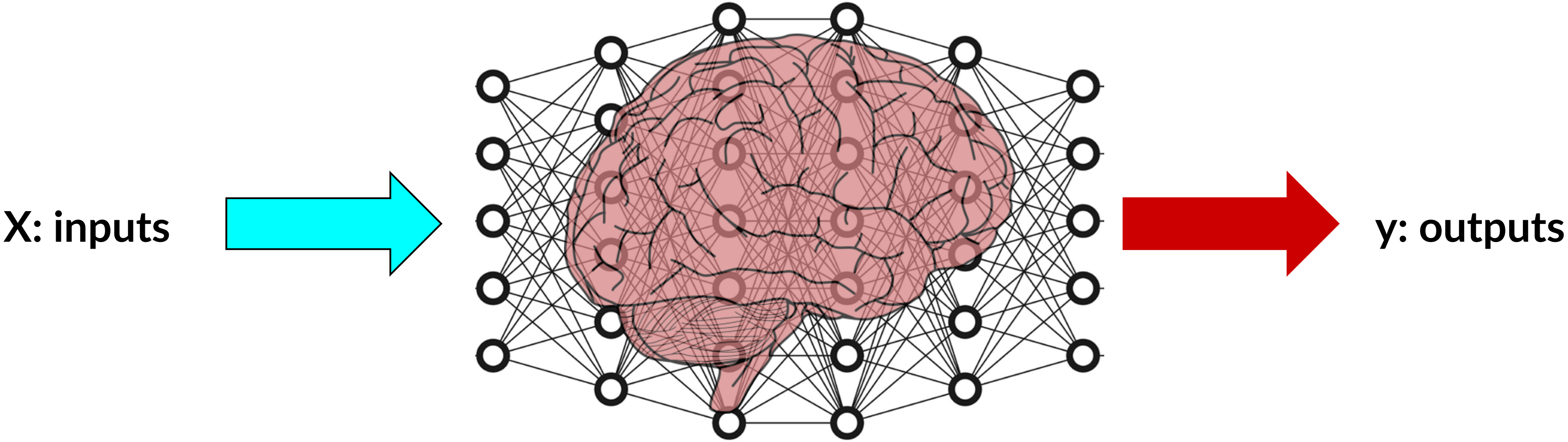


UNIVERSITY OF SUSSEX

Neurons and Artificial Neural Networks



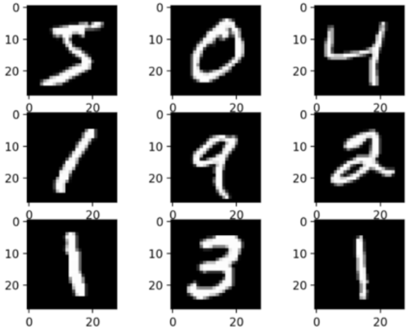
Neurons and Artificial Neural Networks



UNIVERSITY OF SUSSEX

Artificial Neural Networks can be classifiers

Dataset



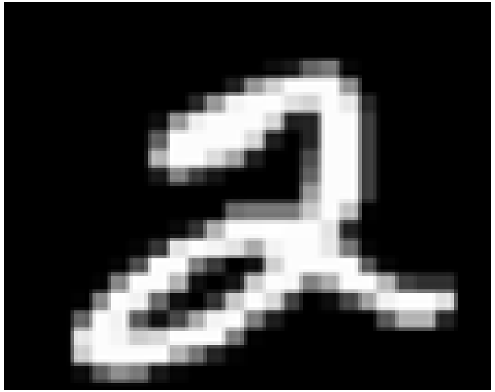
60,000 small square 28×28 pixel grayscale **images of handwritten single digits between 0 and 9**

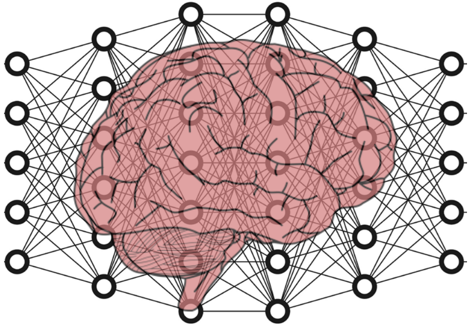
UNIVERSITY OF SUSSEX

Artificial Neural Networks can be classifiers

Image recognition

X:



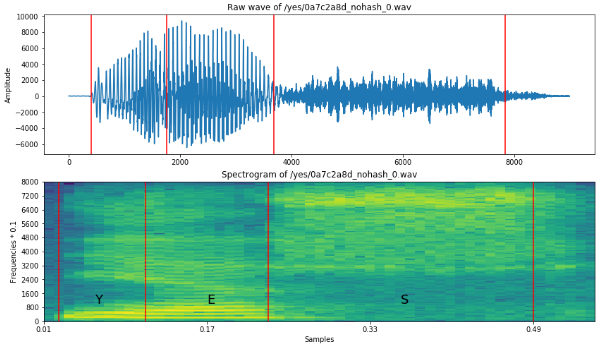


y: 2

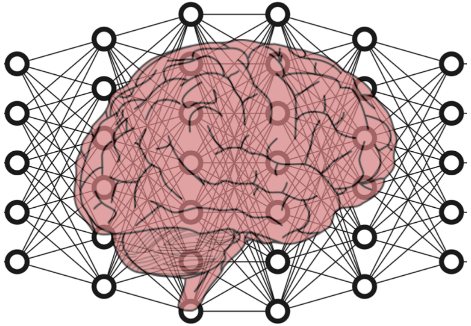
Artificial Neural Networks can be classifiers

Speech recognition

X:



Spectrogram of the spoken word "yes"



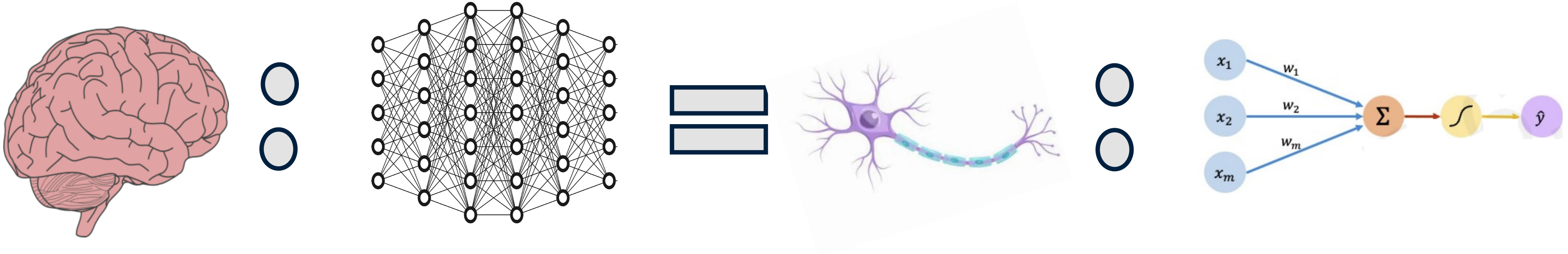
y: "yes"

Artificial Neural Networks

Can do countless things, including :

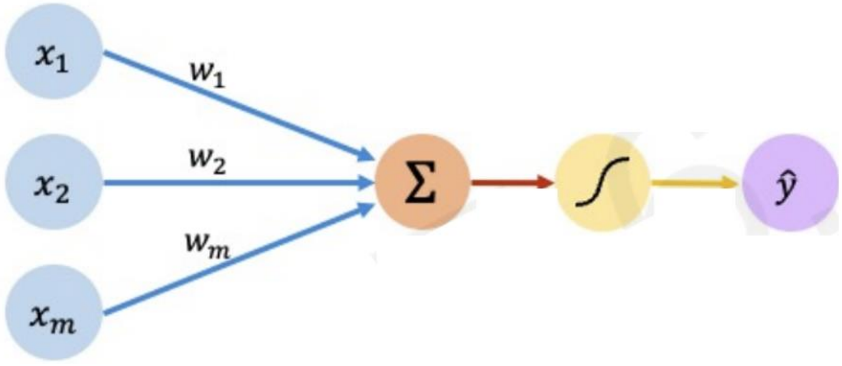
- **Natural Language Processing (NLP):**
  - **Input:** Text data (e.g., a sentence or document).
  - **Output:** Processed text (e.g., sentiment score, translated text, or named entities).
- **Medical Diagnosis:**
  - **Input:** Medical images (e.g., X-rays, MRIs) or patient data (e.g., age, symptoms).
  - **Output:** Diagnosis or probability of a condition (e.g., presence of a tumor).
- **Financial Services:**
  - **Input:** Historical financial data (e.g., stock prices, transaction records).
  - **Output**: Predictions (e.g., future stock prices, fraud detection alerts).
- **Autonomous Vehicles:**
  - **Input**: Sensor data (e.g., camera images, LIDAR scans).
  - **Output**: Driving actions (e.g., steering angle, acceleration, braking).
- **Recommendation Systems:**
  - **Input**: User behaviour data (e.g., past purchases, viewing history).
  - **Output:** Recommendations (e.g., movies, products

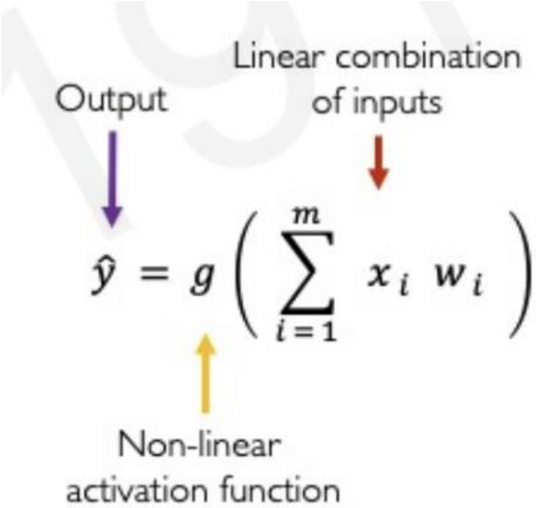
Perceptron: the singular neuron



Perceptron: the singular neuron

The building block of Artificial Neural Networks





Does this remind you of anything?

Inputs Weights Sum Non-Linearity Output

UNIVERSITY OF SUSSEX

Logistic regression

There are many important research topics for which the dependent variable is "limited."

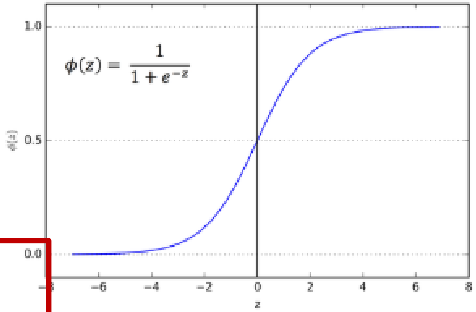
For example: voting, mortality, or number of accidents is not continuous or distributed normally.

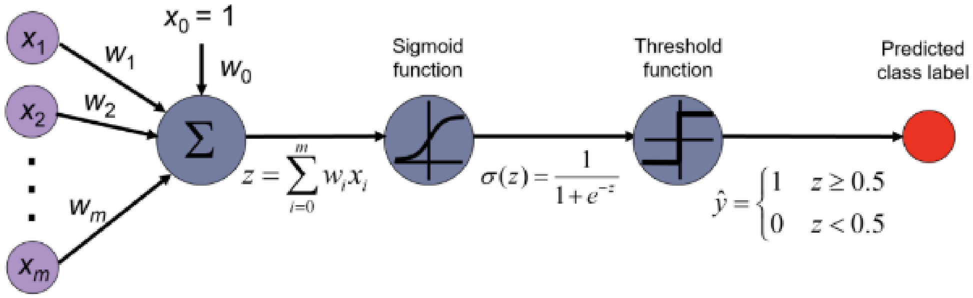
Z = X·W = w0 + w1·x1 + w2·x2 + … + wn·xn

Uses the sigmoid function to enforce the classification

You can also have a bias term

Number of weights = number of features + one bias

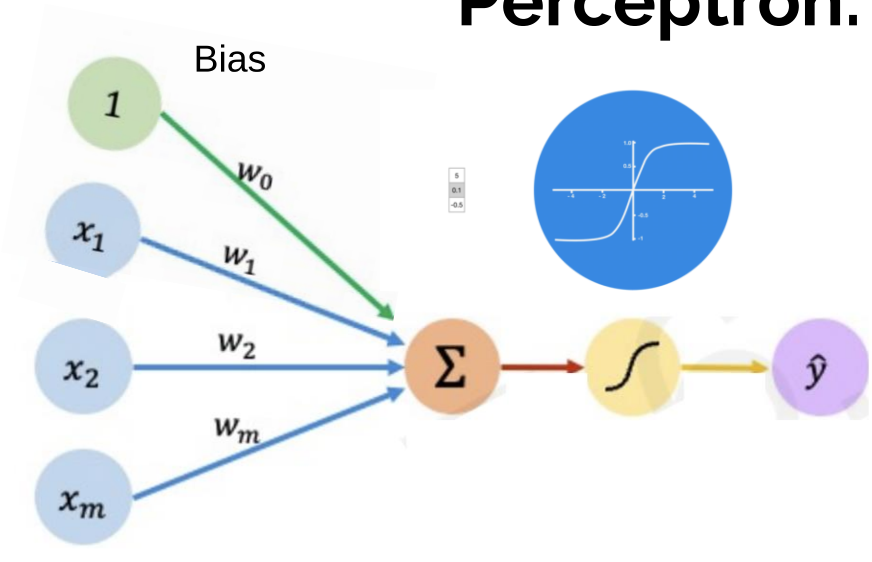


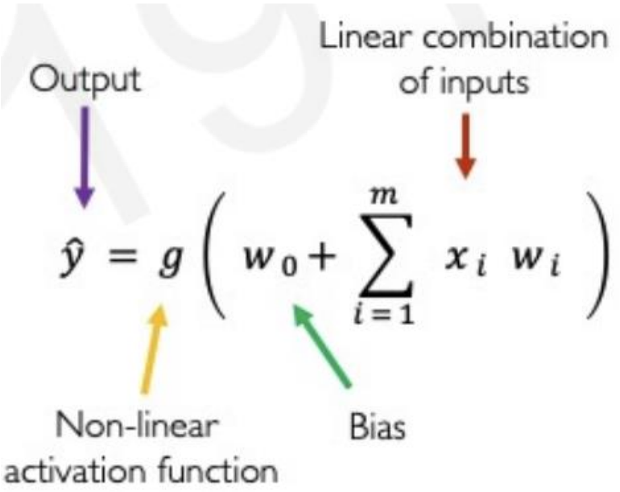


UNIVERSITY OF SUSSEX

Perceptron: the singular neuron

Bias

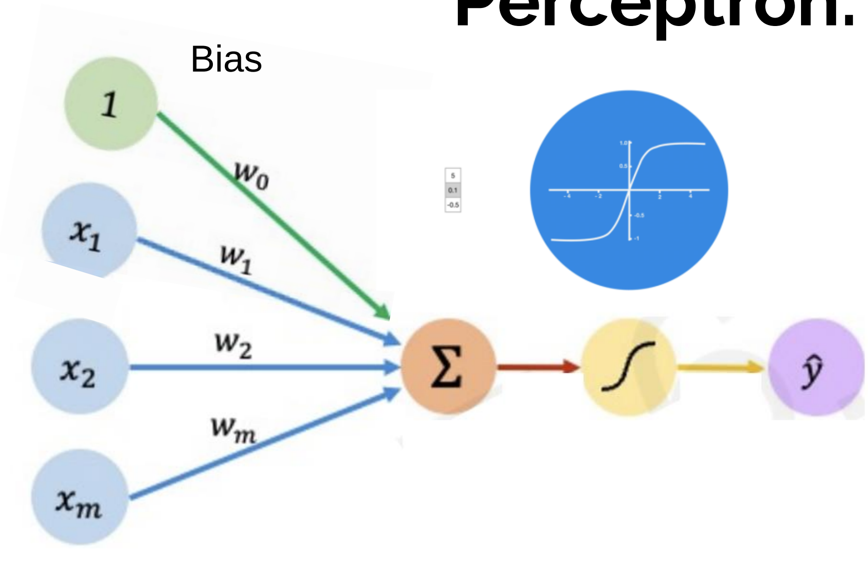




Inputs Weights Sum Non-Linearity Output

The bias term is a scalar that allows us to shift left or right along our activation function

Perceptron: the singular neuron



Inputs Weights Sum Non-Linearity Output

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

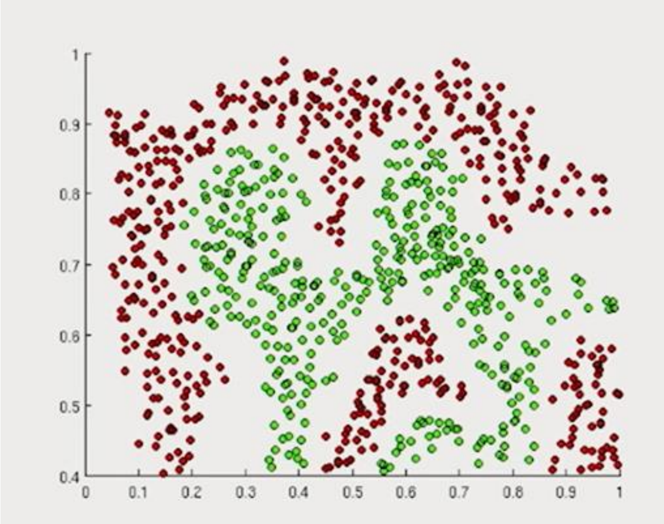
$$\hat{y} = g\,(\,w_0 + \boldsymbol{X}^T \boldsymbol{W}\,)$$

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

UNIVERSITY OF SUSSEX

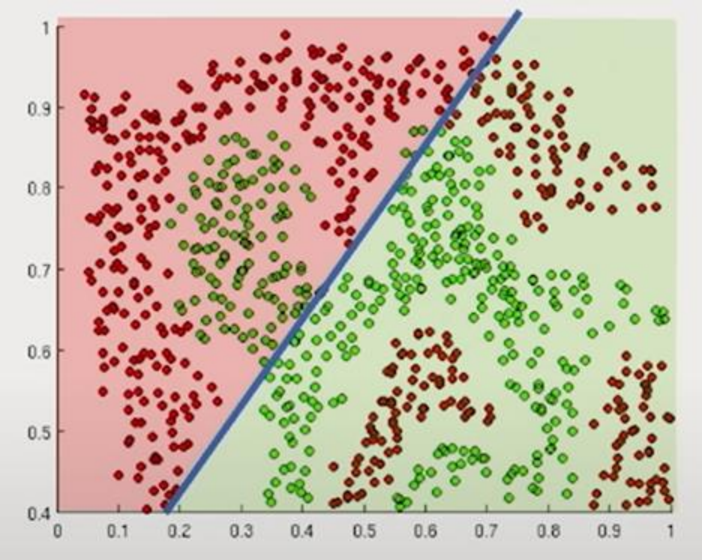
Why do we need Activation Functions?

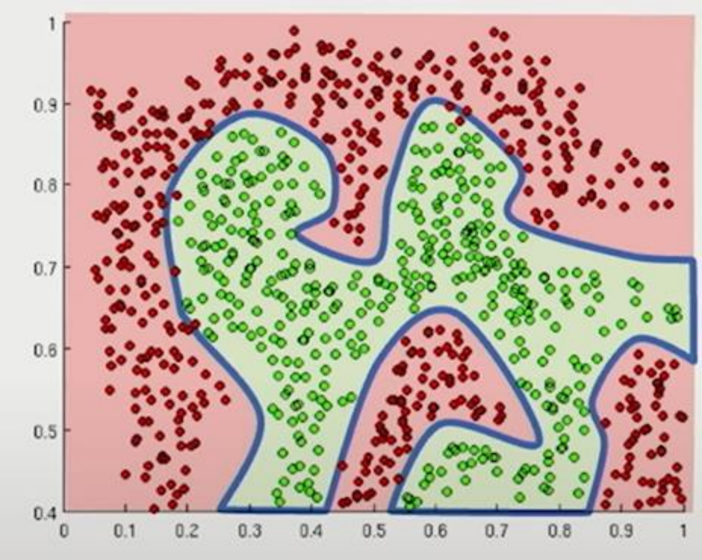
The purpose of activation functions is to introduce **non-linearities** into the network



UNIVERSITY OF SUSSEX

Why do we need non-linearity?





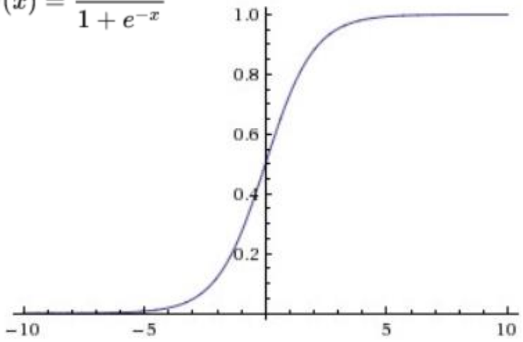
Linear activation functions produce **linear decisions** no matter the network size

Non-linearities allow us to **approximate** arbitrarily complex functions

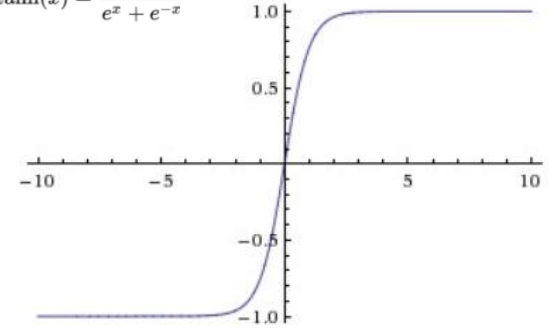
UNIVERSITY OF SUSSEX

What types of activation functions are there and how do we pick one?

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$$



$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



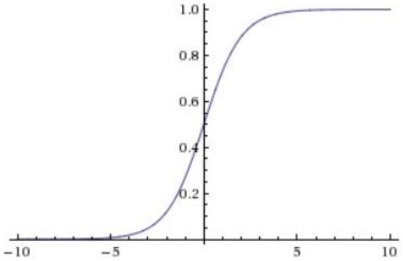
Sigmoid non-linearity squashes the numbers to range between [0,1].

Used for models where we have to predict the probability as an output

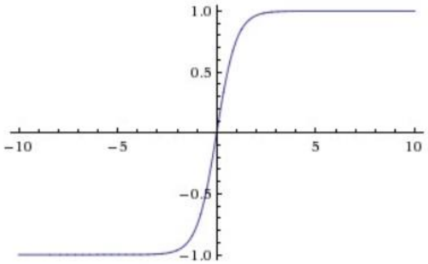
tanh non-linearity squashes the numbers to range between [-1,1]

Tanh tends to be preferred to sigmoid

What types of activation functions are there and how do we pick one?

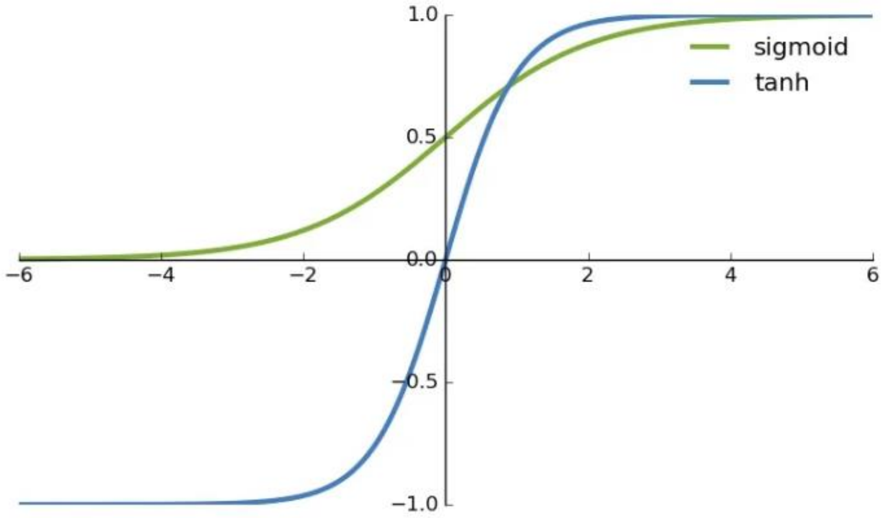



```python
from sklearn.neural_network import MLPClassifier

# Define the neural network with the sigmoid activation
function
clf = MLPClassifier(activation='logistic', hidden_layer_sizes=
(100,), max_iter=300)

# Train the neural network on the training data
clf.fit(X_train, y_train)
```




```python
from sklearn.neural_network import MLPClassifier

# Define the neural network with the tanh activation function
clf = MLPClassifier(activation='tanh', hidden_layer_sizes=
(100,), max_iter=300)

# Train the neural network on the training data
clf.fit(X_train, y_train)
```

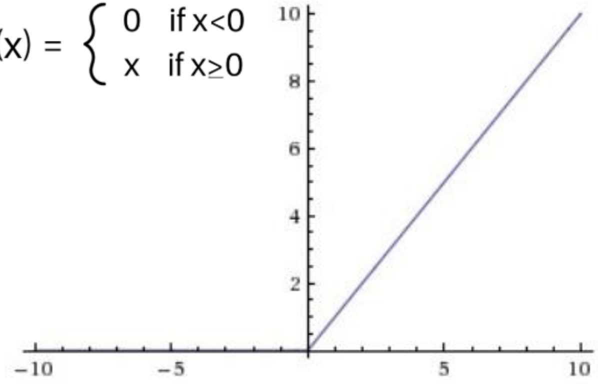
What types of activation functions are there and how do we pick one?



What types of activation functions are there and how do we pick one?

f(x)=max(0,x)

$$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$$



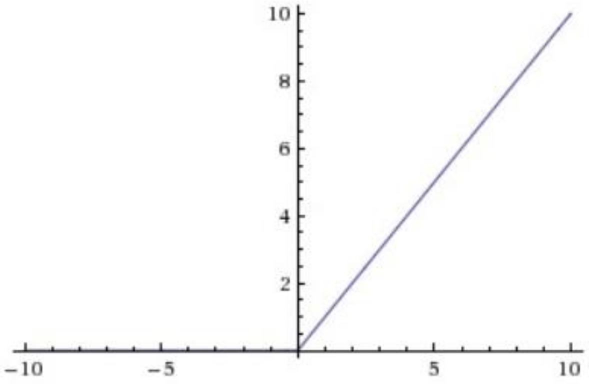
ReLu (Rectified Linear Unit) activation function is zero when x < 0 and then linear with slope 1 when x > 0

- **ReLu speeds up** the calculations up to 6x compared to **tanh**

- **ReLU units** can be fragile during training and can **“die”**. (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. **With a proper setting of the learning rate this is less frequently an issue.**

What types of activation functions are there and how do we pick one?

f(x)=max(0,x)



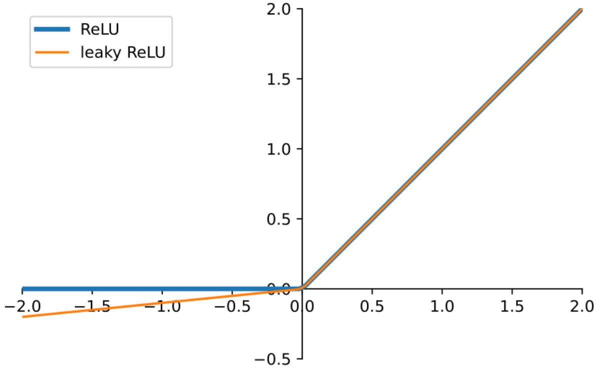
ReLu (Rectified Linear Unit) activation function is zero when x < 0 and then linear with slope 1 when x > 0

```
from sklearn.neural_network import MLPClassifier

# Define the neural network with the ReLU activation function
clf = MLPClassifier(activation='relu', hidden_layer_sizes=(100,), max_iter=300)

# Train the neural network on the training data
clf.fit(X_train, y_train)
```

UNIVERSITY OF SUSSEX

What types of activation functions are there and how do we pick one?

$$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

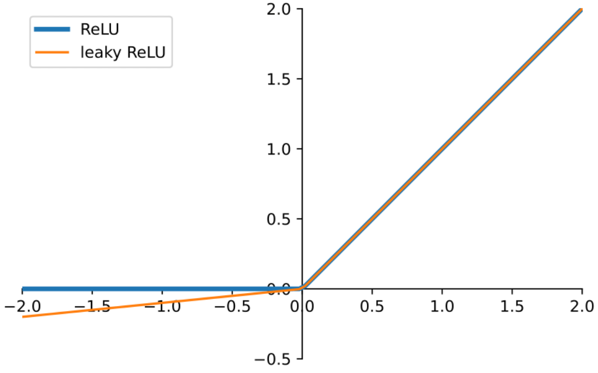


- **Leaky ReLUs** are one attempt to fix the "dying ReLU" problem**.**

- Instead of the function being zero when x < 0, a leaky ReLU will instead have a small positive slope (of 0.01, or so). That is, the function computes **f(x)=1(x<0)(αx)+1(x>=0)(x)** where **α** is a small constant**.**

Other solutions to the "dying ReLU" problem include ELUs (Exponential Linear Units) and PReLUs (Parametric ReLU).

What types of activation functions are there and how do we pick one?

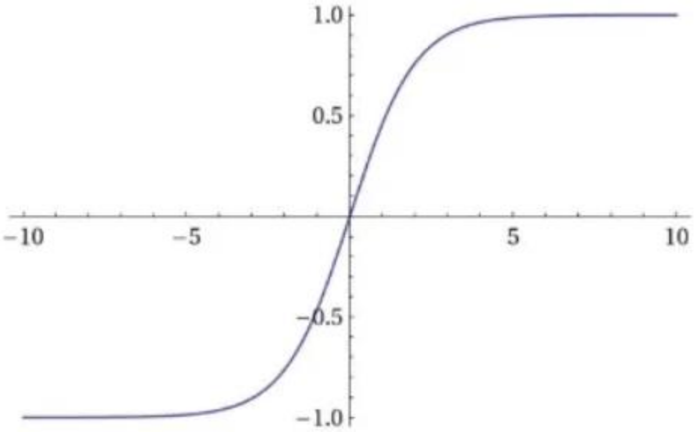


- **Leaky ReLUs** are one attempt to fix the “dying ReLU” problem.

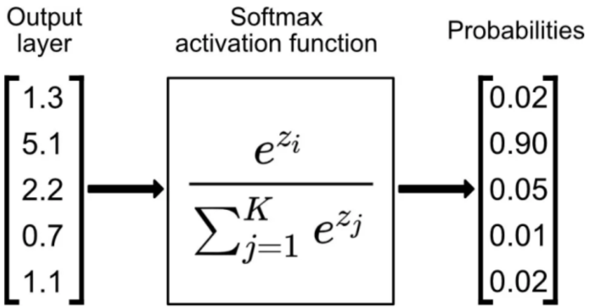
```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LeakyReLU

# Define the neural network model
model = Sequential([
    # Add a dense layer with 100 neurons
    Dense(100, input_dim=input_dim),
    # Apply Leaky ReLU activation function with alpha=0.01
    LeakyReLU(alpha=0.01),
    # Add the output layer with softmax activation for multi-class classification
    Dense(output_dim, activation='softmax')
])

# Compile the model with Adam optimizer and categorical crossentropy loss
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the neural network on the training data
model.fit(X_train, y_train, epochs=300)
```

UNIVERSITY OF SUSSEX

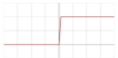
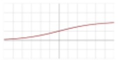
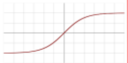
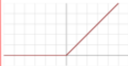
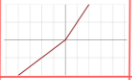
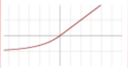
What types of activation functions are there and how do we pick one?



- **Softmax** is used when you need to classify data into more than two categories. For example, if you have a dataset with images of cats, dogs, and birds, Softmax can help determine the probability that a given image belongs to each of these classes.
- It is applied after the output.

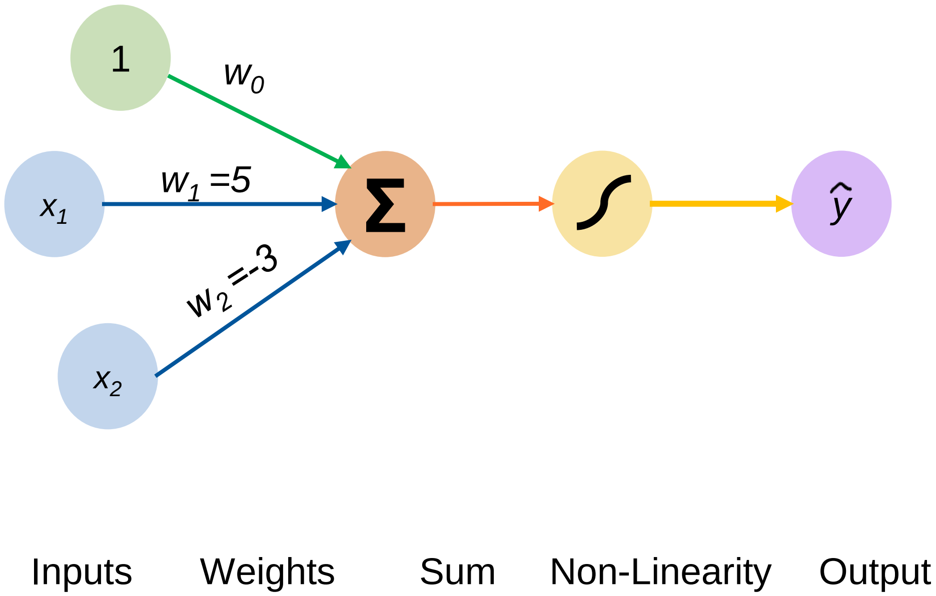


What types of activation functions are there and how do we pick one?

ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	$(-\infty, \infty)$
Binary Step		f(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		f(x) = $\sigma(x) = \frac{1}{1+e^{-x}}$	f'(x) = $f(x)(1-f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		f(x) = $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	f'(x) = $1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		f(x) = $\begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		f(x) = $\ln(1+e^x)$	f'(x) = $\frac{1}{1+e^{-x}}$	$(0, 1)$
Leaky ReLU		f(x) = $\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	f'(x) = $\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		f(x) = $\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	f'(x)= $\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

UNIVERSITY OF SUSSEX

Perceptron: an example



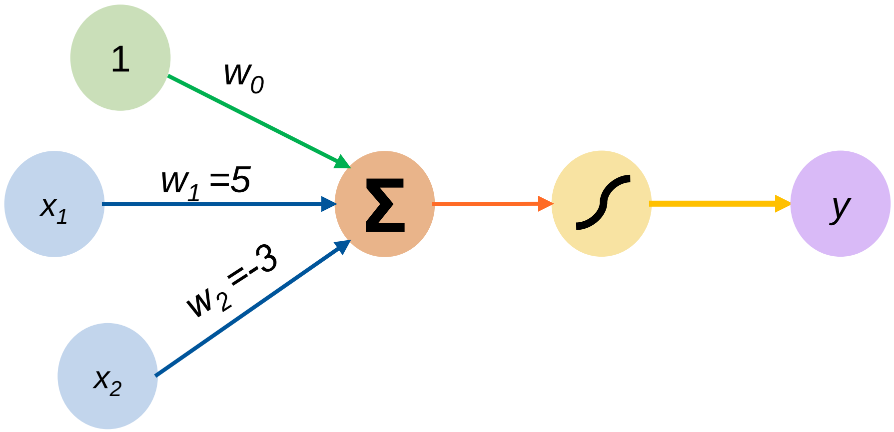
We have $w_0 = 1$ and $\boldsymbol{W} = \begin{bmatrix} \mathbf{5} \\ \mathbf{-3} \end{bmatrix}$

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

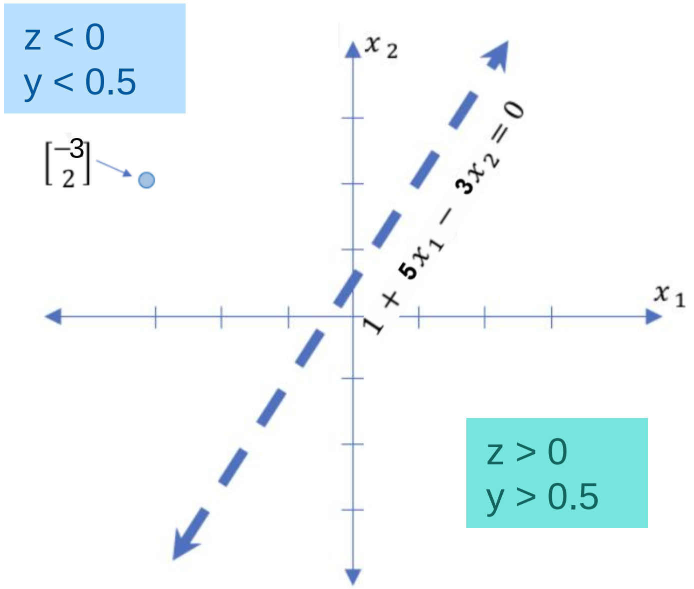
$$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} \mathbf{5} \\ \mathbf{-3} \end{bmatrix}\right)$$

$$\hat{y} = g(\underbrace{1 + \mathbf{5}x_1 - \mathbf{3}x_2}_{\text{This is just a line in 2D!}})$$

Perceptron: an example



$$\hat{y} = g(\underbrace{1 + \mathbf{5}x_1 - \mathbf{3}x_2}_{(z)})$$



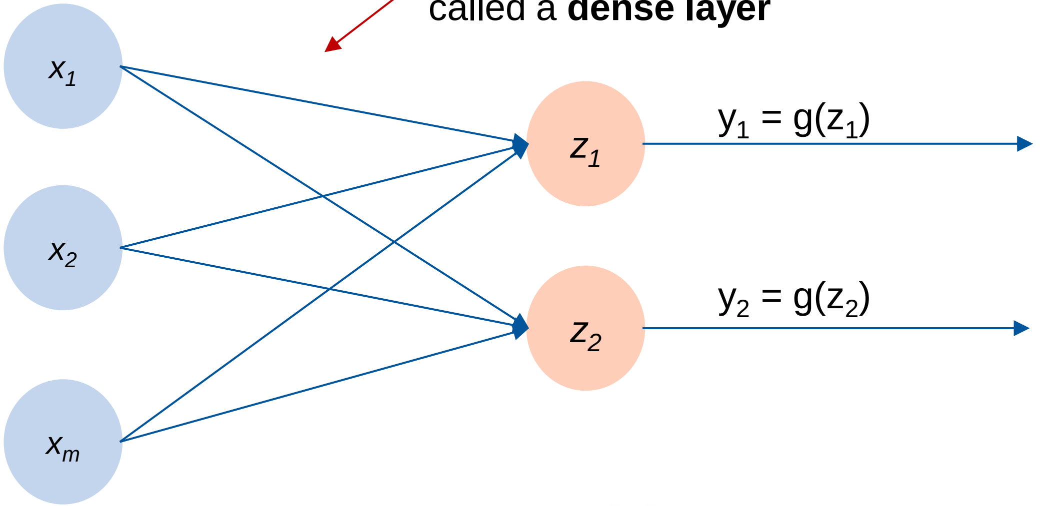
If we have an input $\boldsymbol{X} = \begin{bmatrix} -3 \\ 2 \end{bmatrix}$

$$\hat{y} = g\left(1 + (5 * -3) - (3 * 2)\right)$$
$$= \mathrm{g}(-20) = 2.06\ \mathrm{E}\text{-}9$$

UNIVERSITY OF SUSSEX

Multi-output perceptron

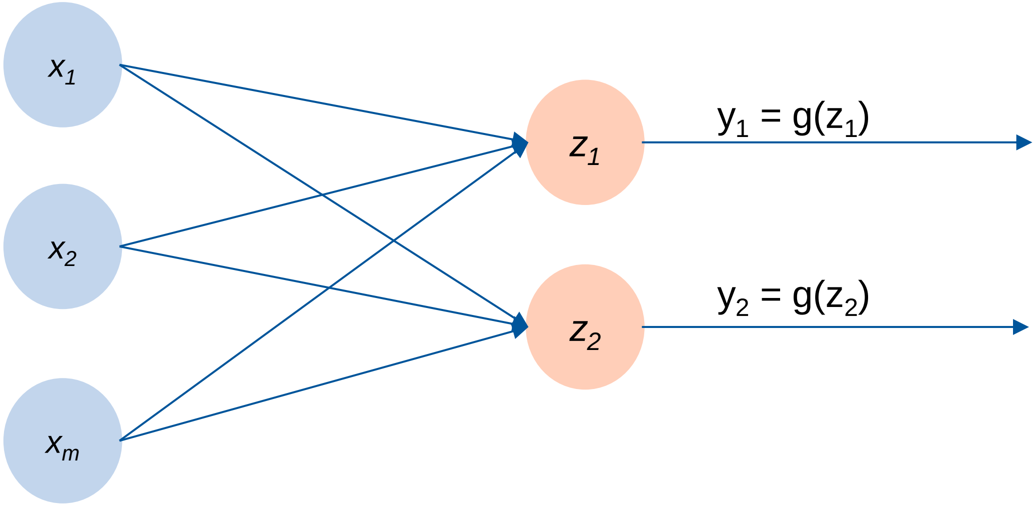
Because of the dense connections between inputs and outputs, this is called a **dense layer**



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

This is just an example to understand the intuition, but multi-output perceptrons are useful when you need to predict multiple target variables simultaneously. For example, predicting both temperature and humidity based on weather data.

Multi-output perceptron

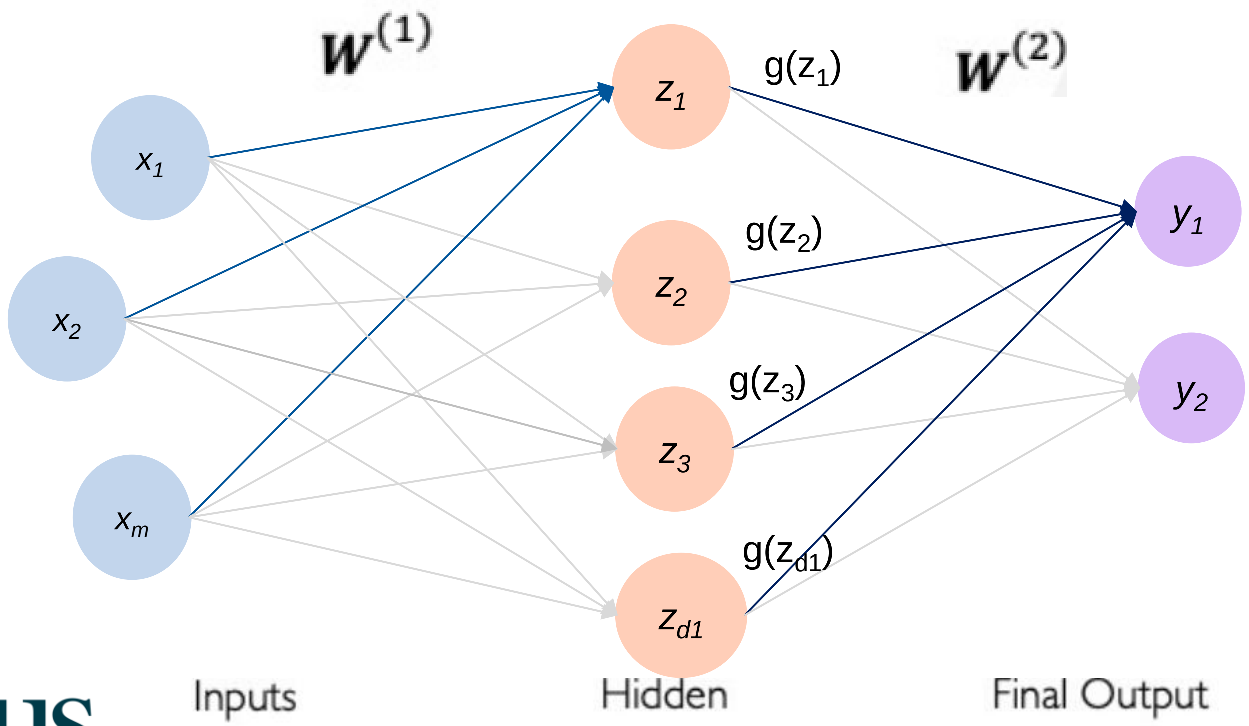



```python
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# Generate synthetic data with multiple outputs
X, y = make_regression(n_samples=1000, n_features=20,
n_targets=2, noise=0.1)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the multi-output perceptron
mlp = MLPRegressor(hidden_layer_sizes=(100,),
activation='relu', max_iter=300)

# Train the neural network on the training data
mlp.fit(X_train, y_train)

# Make predictions on the test data
predictions = mlp.predict(X_test)
```

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

UNIVERSITY OF SUSSEX

Single Layer Neural Network



In layered neural networks, the outputs from one layer become the inputs of the next layer

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

UNIVERSITY OF SUSSEX

Single Layer Neural Network

For classification:

```python
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic classification data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the single-layer neural network
clf = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
max_iter=300)

# Train the neural network on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
predictions = clf.predict(X_test)
```

For regression:

```python
from sklearn.neural_network import MLPRegressor
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split

# Generate synthetic regression data
X, y = make_regression(n_samples=1000, n_features=20, noise=0.1,
random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the single-layer neural network
reg = MLPRegressor(hidden_layer_sizes=(100,), activation='relu',
max_iter=300)

# Train the neural network on the training data
reg.fit(X_train, y_train)

# Make predictions on the test data
predictions = reg.predict(X_test)
```

Single Layer Neural Network

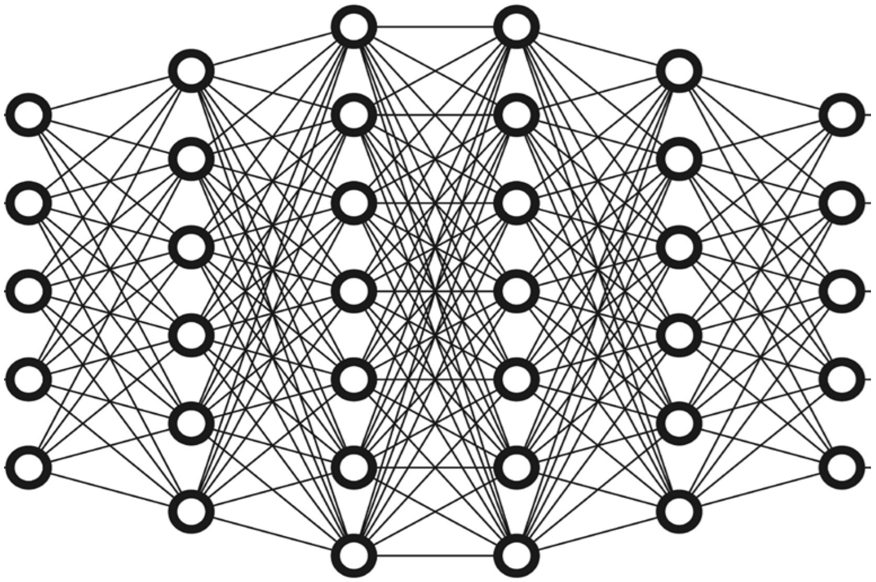
For classification:

```python
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic classification data
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2,
random_state=42)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Define the single-layer neural network
clf = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
max_iter=300)

# Train the neural network on the training data
clf.fit(X_train, y_train)

# Make predictions on the test data
predictions = clf.predict(X_test)
```

- **n_samples**= The number of samples (data points) in your dataset. 1000 creates 1000 data points.
- **n_features**=The number of features (input variables) in your dataset. 20 means each data point has 20 input variables.
- **n_classes**= It determines how many different categories the target variable (y) can take.2 means the target variable (y) will have three distinct class labels (e.g., 0, 1).
- **random_state**=A seed value for random number generation to ensure reproducibility. 42 ensures the data generation process is reproducible.

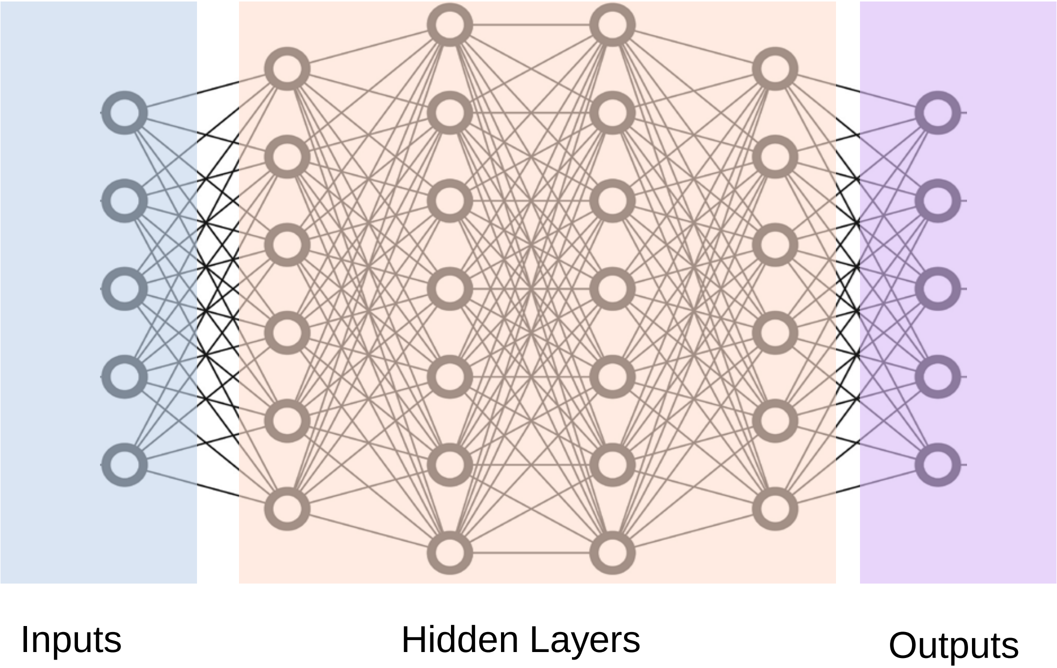
UNIVERSITY OF SUSSEX

Multi Layer Neural Network



In layered neural networks, the outputs from one layer become the inputs of the next layer

Multi Layer Neural Network



In layered neural networks, the outputs from one layer become the inputs of the next layer

Multi Layer Neural Network

For classification:

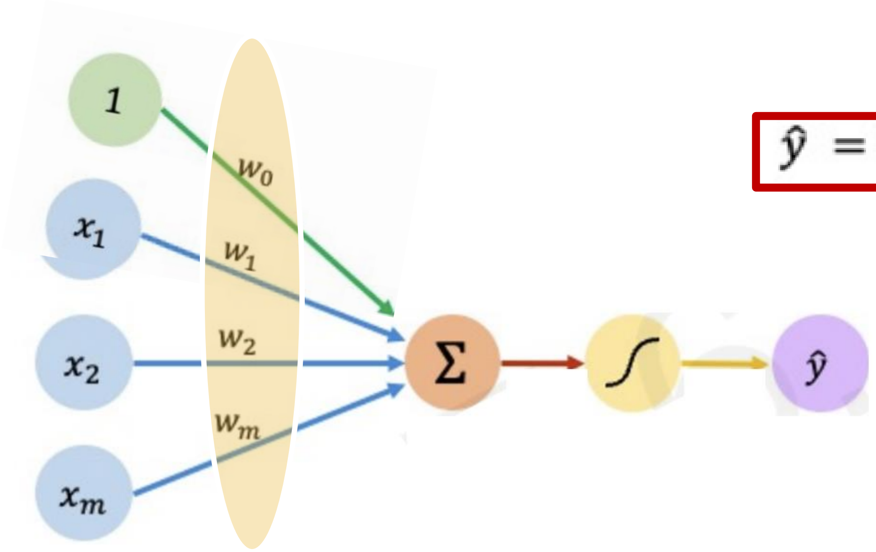
```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic classification data
# n_samples: number of samples
# n_features: number of features
# n_classes: number of classes
# random_state: seed for reproducibility
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Split the data into training and testing sets
# test_size: proportion of the dataset to include in the test split
# random_state: seed for reproducibility
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define the multi-layer neural network with two hidden layers
# hidden_layer_sizes: tuple specifying the number of neurons in each hidden layer
# activation: activation function for the hidden layers ('relu' in this case)
# max_iter: maximum number of iterations for training
clf = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', max_iter=300)

# Train the neural network on the training data
# X_train: training input data
# y_train: training target data
clf.fit(X_train, y_train)

# Make predictions on the test data
# X_test: testing input data
predictions = clf.predict(X_test)
```

- **hidden_layer_sizes**= in this case it specifies two hidden layers, the first with 100 neurons and the second with 50 neurons.

Training a neural network

The code does it for us. But what does it do exactly?

Think about your data and think about this model



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

Is there any information we don't have?

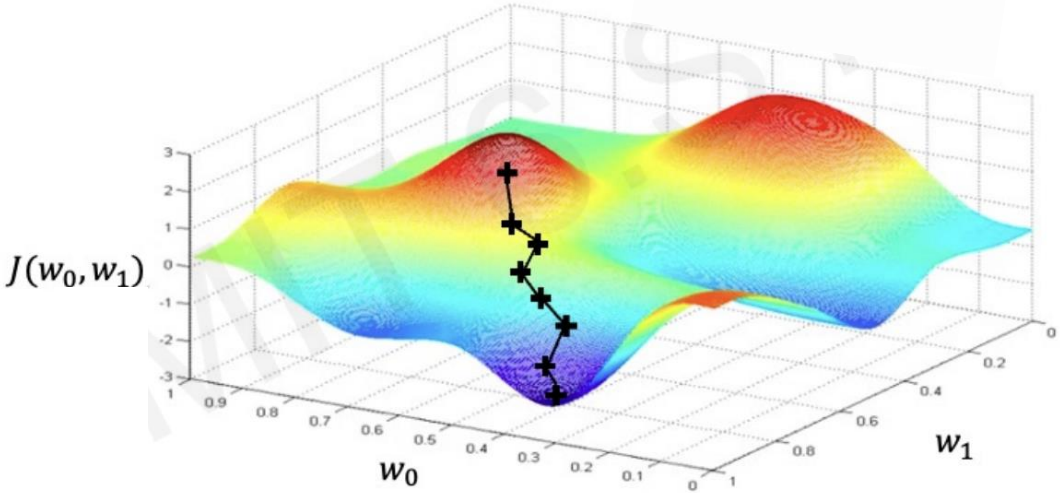
Loss optimisation

- Quantify the Loss (how wrong the outputs y are compared to our test data)
- Update the weights (through gradient descent and backpropagation) to minimise this loss

Loss optimisation

- Quantify the Loss (how wrong the outputs y are compared to our test data)

- Update the weights (through gradient descent and backpropagation) to minimise this loss



scikit-learn handles all the underlying details of backpropagation and gradient descent, allowing you to focus on defining and training your model

The code handles this for us

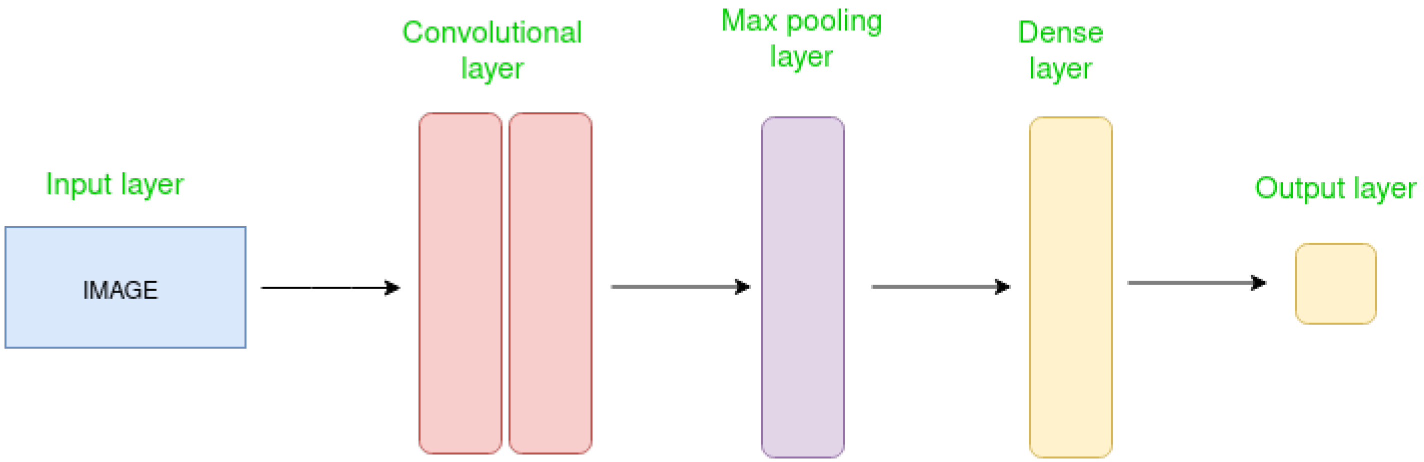
Types of ANNs

Convolutional Neural Networks (CNNs)

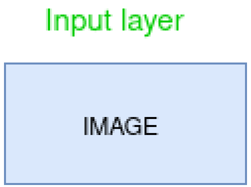
CNNs are designed for visual data processing.

UNIVERSITY OF SUSSEX

Convolutional neural networks



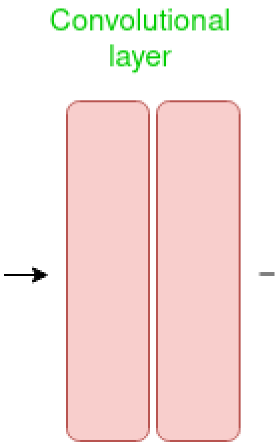
Convolutional neural networks



Let's take an example by running an image of dimension 32 x 32 x 3.

•**Input Layers:** It's the layer in which we give input to our model. In CNN, Generally, the input will be an image or a sequence of images. This layer holds the raw input of the image with width 32, height 32, and depth 3 (= the colour array).

Convolutional neural networks



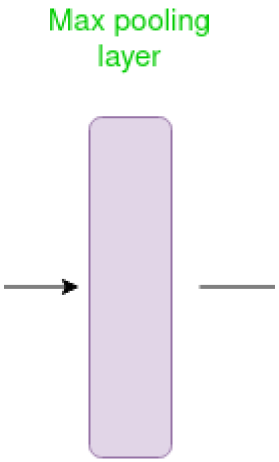
- **Filters/Kernels**: Small matrices that slide over the input image. Each filter is designed to detect specific features like edges, textures, or patterns. Imagine looking at an image through a small window that slides over the picture. Each window (filter) looks for specific things like edges or patterns.

- **Convolution Operation**: The filter is applied to the input image, producing a feature map. This operation involves multiplying the filter values with the corresponding pixel values and summing them up. The result is a new matrix that highlights the presence of the feature detected by the filter. As the window slides over the image, it creates a new picture (feature map) that highlights where it found those edges or patterns.

- **ReLU:** After the window slides over the image, ReLU changes all negative values to zero. This helps the network focus on important features and ignore less useful information.

UNIVERSITY
OF SUSSEX

Convolutional neural networks

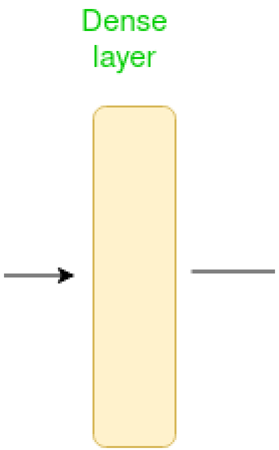


- **Max Pooling**: This layer reduces the dimensionality of the feature map while retaining important features. It does this by selecting the maximum value from a small region (e.g., 2x2) of the feature map. This helps in reducing computational load and preventing overfitting.

- Think of zooming out on a picture to make it smaller while keeping the important parts. Max pooling picks the biggest value from a small area, reducing the size of the feature map but keeping the key features.

UNIVERSITY OF SUSSEX

Convolutional neural networks

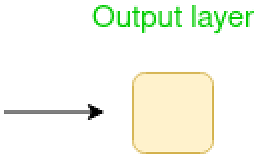


- **Flattening:** Converts the 2D feature maps into a 1D vector. This step prepares the data for the fully connected layers. Imagine taking all the important features from the picture and laying them out in a single line.

- **Dense Layers:** These layers connect every neuron from one layer to every neuron in the next layer. They combine the features learned by the convolutional and pooling layers to make final predictions. **This is where the network decides which class the input image belongs to.**

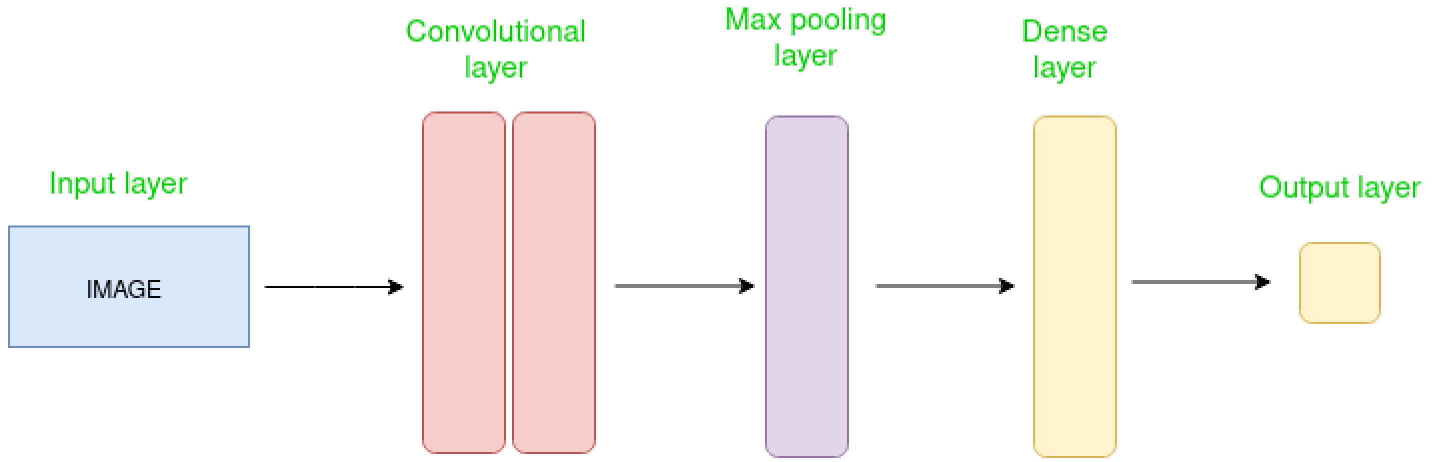
UNIVERSITY OF SUSSEX

Convolutional neural networks

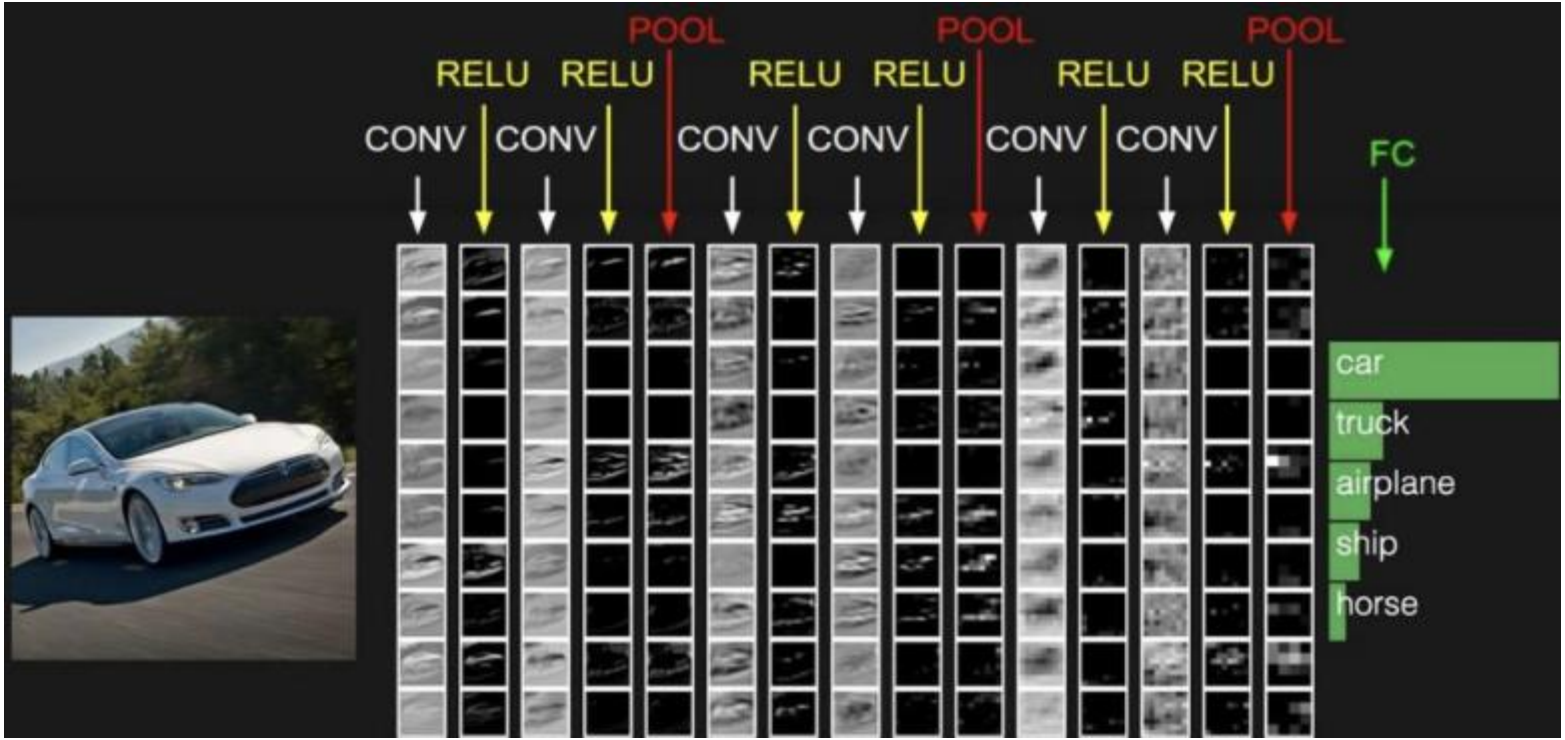


- **Output Layer:** The output from the fully connected layers is then fed into a logistic function for classification tasks like sigmoid or softmax which converts the output of each class into the probability score of each class.

Convolutional neural networks



Convolutional neural networks



UNIVERSITY OF SUSSEX

Convolutional neural networks

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define the CNN model
model = Sequential()

# Add a convolutional layer with 32 filters, a 3x3 kernel, and ReLU activation
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)))

# Add a max pooling layer with a 2x2 pool size
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add another convolutional layer with 64 filters
model.add(Conv2D(64, (3, 3), activation='relu'))

# Add another max pooling layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten the feature maps to a 1D vector
model.add(Flatten())

# Add a fully connected layer with 128 neurons and ReLU activation
model.add(Dense(128, activation='relu'))

# Add the output layer with softmax activation for multi-class classification
model.add(Dense(10, activation='softmax'))

# Compile the model with Adam optimizer and categorical crossentropy loss
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()"
```

1. **Convolutional Layers**:
    1. **Number of Filters**: The choice of 32 and 64 filters is typical for initial layers in a CNN. These numbers can be adjusted based on the complexity of the task and the size of the dataset.
    2. **Kernel Size**: A 3x3 kernel is a standard choice that balances computational efficiency and the ability to capture spatial features.
2. **Pooling Layers**:
    1. **Pool Size**: A 2x2 pool size is commonly used to reduce the spatial dimensions while retaining important features.
3. **Fully Connected Layers**:
    1. **Number of Neurons**: The choice of 128 neurons in the dense layer is a common practice, but it can be tuned based on the complexity of the task.
4. **Activation Functions**:
    1. **ReLU**: The ReLU activation function is widely used due to its ability to mitigate the vanishing gradient problem and speed up training.
    2. **Softmax**: Softmax is used in the output layer for multi-class classification tasks.
5. **Optimizer and Loss Function**:
    1. **Adam Optimizer**: Adam is a popular choice for its adaptive learning rate and efficiency.
    2. **Categorical Crossentropy**: This loss function is suitable for multi-class classification problems.

UNIVERSITY OF SUSSEX

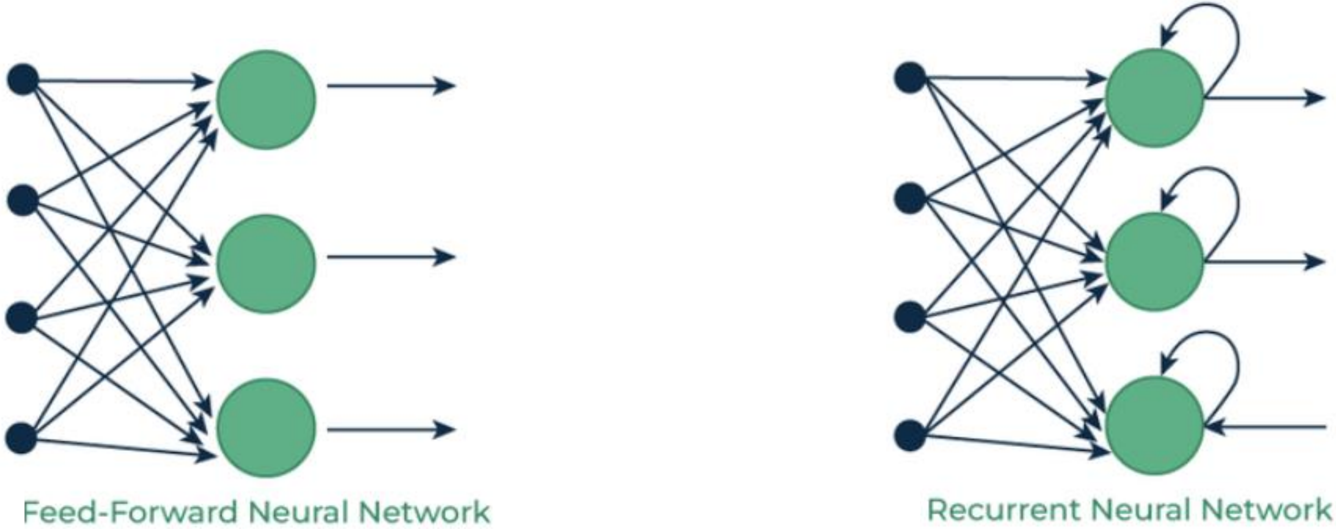
[Diving Deep into CNNs: Optimization and Regularization - Neural Nexus]()

Recursive Neural Networks (RNNs)

- Language Modeling and Text Generation
- Speech Recognition
- Sentiment Analysis
- Music Composition
- Machine Translation
- Time Series Prediction
- Video Analysis
- Healthcare

Recursive Neural Networks (RNNs)

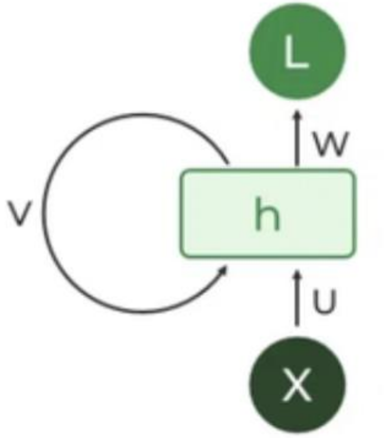
RNNs are unique because they have **loops in their architecture**, allowing information to persist. This makes them well-suited for **tasks where context or previous information is important**, such as language modeling or predicting stock prices.



UNIVERSITY OF SUSSEX

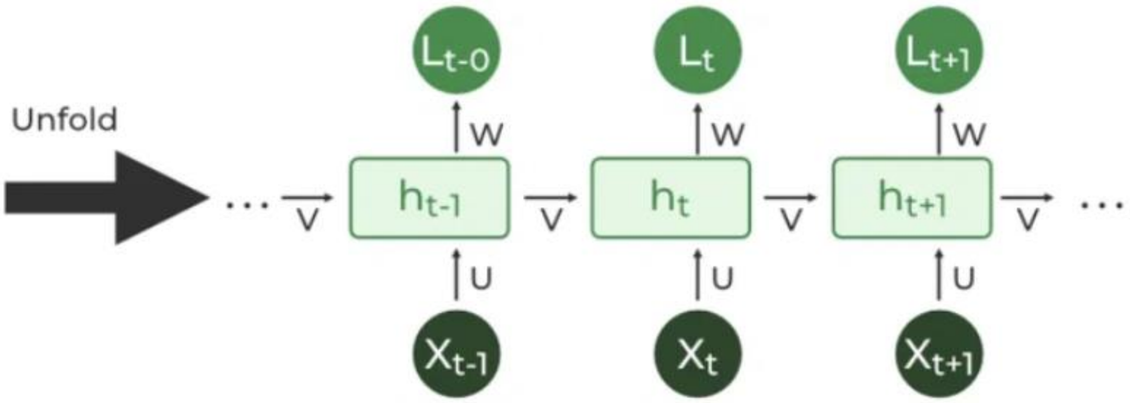
Recursive Neural Networks (RNNs)

The fundamental processing unit in RNN is a **Recurrent Unit.** .



- Recurrent units hold **a hidden state that maintains information about previous inputs in a sequence**. Recurrent units can “remember” information from prior steps by feeding back **their hidden state**

Recursive Neural Networks (RNNs)

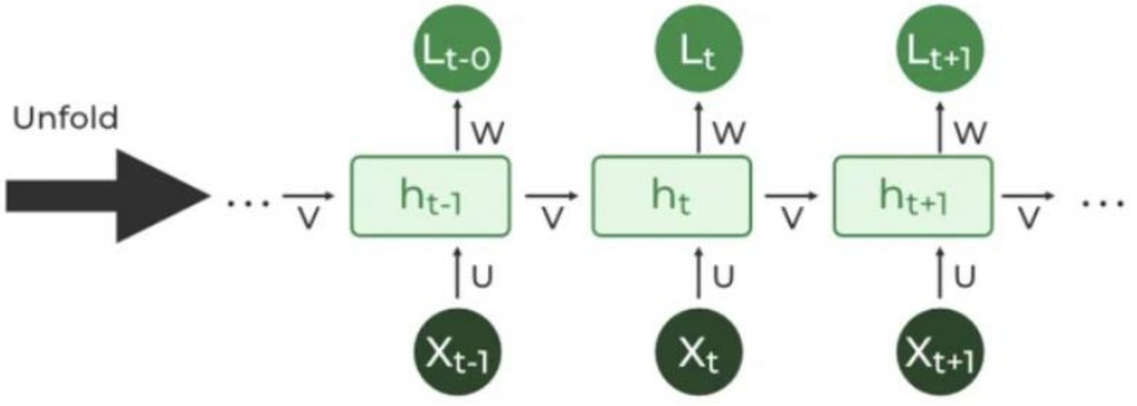


RNN Unfolding

- **Sequence Processing**: RNNs process input sequences one element at a time, updating the hidden state at each step.

UNIVERSITY OF SUSSEX

Recursive Neural Networks (RNNs)



RNN Unfolding

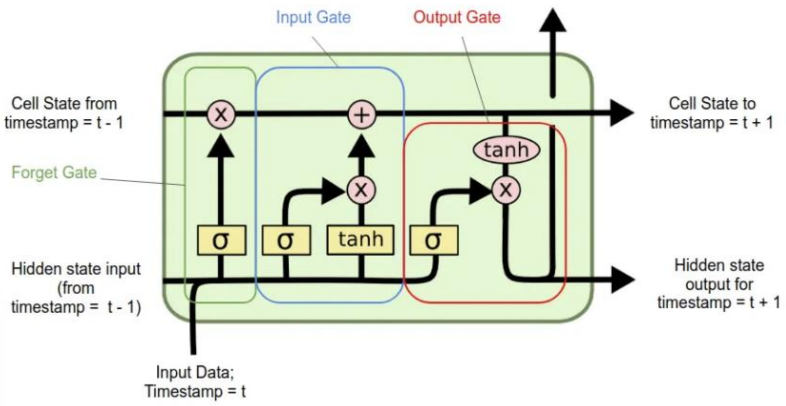
Since RNNs process sequential data **Backpropagation Through Time (BPTT)** is used to update the network's parameters.

But this is also handled by code, so we don't need to worry about this!

https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/?ref=lbp

Long Short-Term Memory (LSTM) networks

LSTMs are a type of Recurrent Neural Network (RNN) designed to address the limitations of traditional RNNs, particularly the issue of long-term dependencies



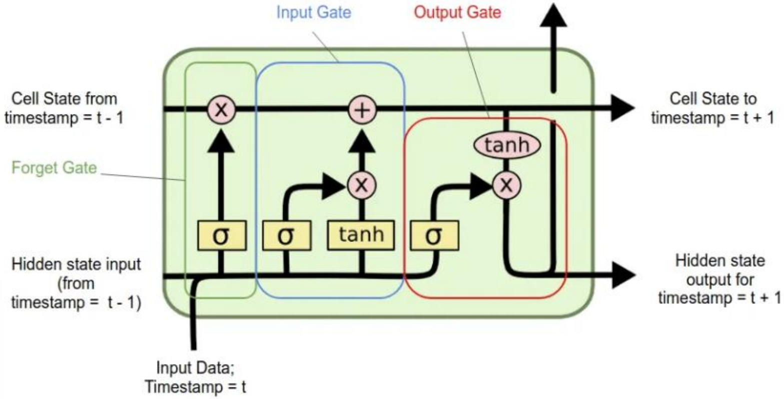
A single LSTM Cell

- **Cell State**: LSTMs have a cell state that runs through the entire sequence, allowing information to be carried over long distances.
- **Gates**: LSTMs use gates to control the flow of information. These gates are:
    - **Forget Gate**: Decides what information to discard from the cell state.
    - **Input Gate**: Decides what new information to add to the cell state.
    - **Output Gate**: Decides what information to output from the cell state.

UNIVERSITY OF SUSSEX

https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

Long Short-Term Memory (LSTM) networks



A single LSTM Cell

1. **Forget Gate**:
   - The forget gate determines which parts of the cell state to forget. It uses a sigmoid function to output values between 0 and 1, where 0 means "forget completely" and 1 means "keep completely."
2. **Input Gate**:
   - The input gate decides which new information to add to the cell state. It uses a sigmoid function to determine which values to update and a tanh function to create new candidate values.
3. **Cell State Update**:
   - The cell state is updated by combining the forget gate's output and the input gate's new candidate values.
4. **Output Gate**:
   - The output gate determines the final output of the LSTM cell. It uses a sigmoid function to decide which parts of the cell state to output and a tanh function to scale the output.

UNIVERSITY OF SUSSEX

https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

RNNs implementations

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

# Define the RNN model
model = Sequential()

# Add a SimpleRNN layer with 50 units
# 50 units is a common choice for capturing patterns without overfitting
model.add(SimpleRNN(50, input_shape=(None, 1)))

# Add a fully connected layer with 1 neuron and sigmoid activation
# Sigmoid activation is used for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model with Adam optimizer and binary crossentropy loss
# Adam optimizer is efficient and adaptive, binary crossentropy is suitable for binary classification
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

# Define the LSTM model
model = Sequential()

# Add an LSTM layer with 50 units
# LSTM units are optimal for capturing long-term dependencies
model.add(LSTM(50, input_shape=(None, 1)))

# Add a fully connected layer with 1 neuron and sigmoid activation
# Sigmoid activation is used for binary classification
model.add(Dense(1, activation='sigmoid'))

# Compile the model with Adam optimizer and binary crossentropy loss
# Adam optimizer is efficient and adaptive, binary crossentropy is suitable for binary classification
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

RNNs: Suitable for tasks with short-term dependencies, such as simple sequence prediction.

LSTMs: Suitable for tasks with long-term dependencies, such as language modeling, speech recognition, and time series forecasting.

Recap

- What are Neural Networks
- Artificial Neurons (perceptron, MLPs, Single and multi layered NNs)
- Training Neural Networks
- Some types of Neural Networks (CNNs, RNNs, LSTMs)

LAB

UNIVERSITY
OF SUSSEX

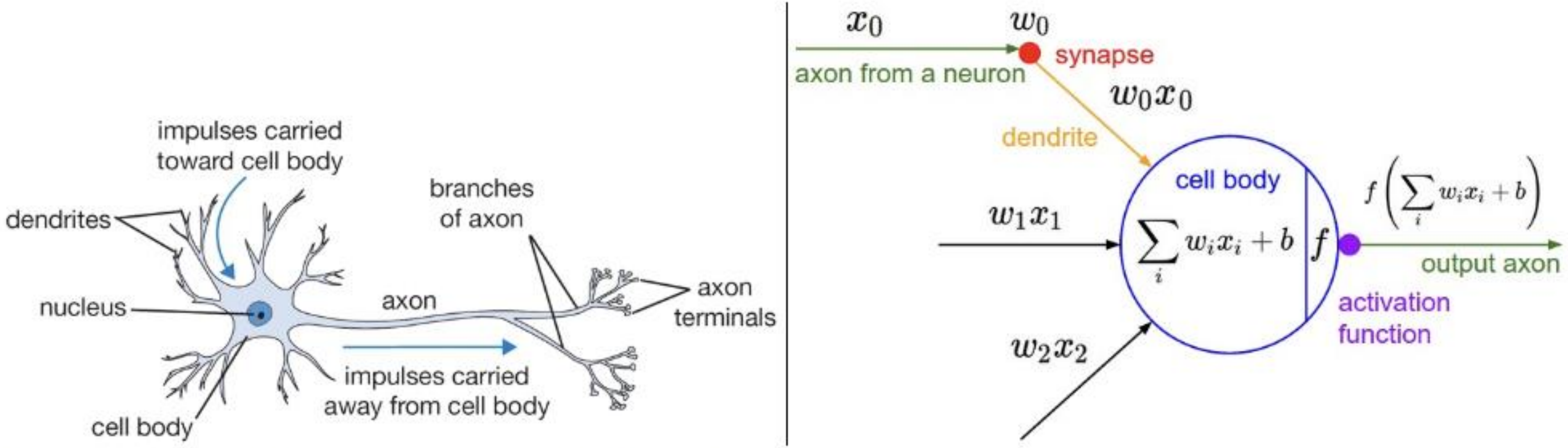
Neural Nets Lab

- Go to github for lab8.ipynb

Supplementary Material

UNIVERSITY
OF SUSSEX

Biological vs Artificial neuron



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Quantifying loss

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

- **Empirical loss:** measures the total loss over the entire dataset

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

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

- **Binary Cross Entropy Loss:** can be used with models that output a probability between 0 and 1

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

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

- **Mean Squared Error Loss:** can be used with regression models that output continuous numbers

$$\begin{array}{ccc} f(x) & & y \\ \begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} & \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} & \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix} \end{array}$$

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

UNIVERSITY OF SUSSEX

Loss optimisation

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

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

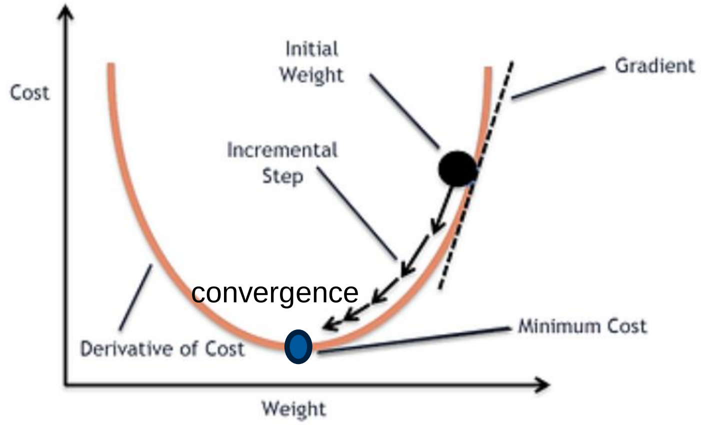
$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}}\, J(\boldsymbol{W})$$

Remember:
$\boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \cdots\}$

UNIVERSITY OF SUSSEX

Gradient descent

"A gradient measures how much the output of a function changes if you change the inputs a little bit." — Lex Fridman (MIT)

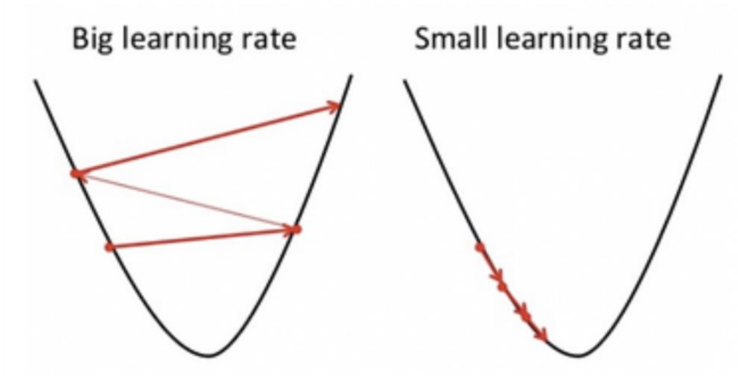


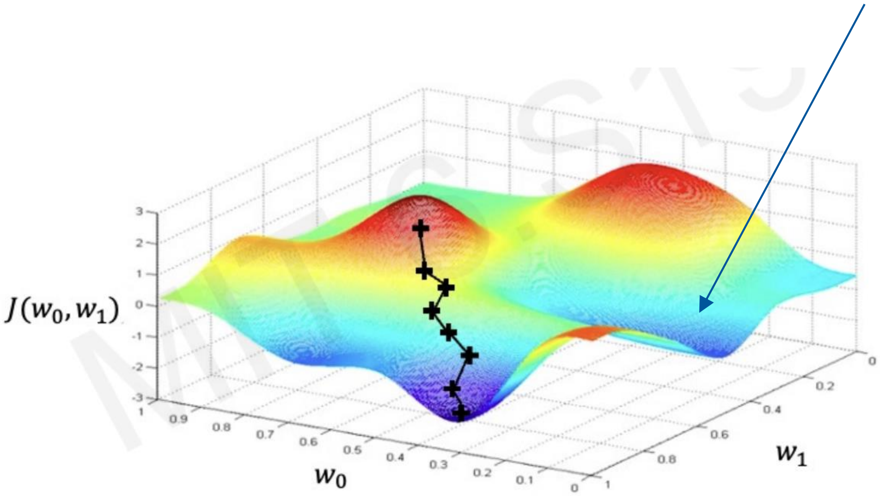
Algorithm

1. Initialize weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. 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

UNIVERSITY OF SUSSEX

Gradient Descent and learning rate





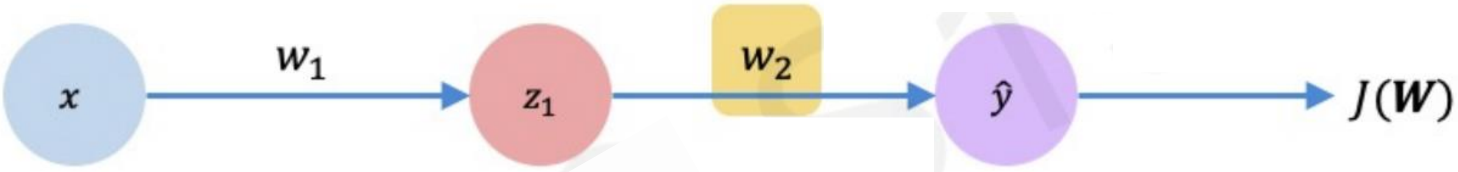
A big learning rate may not find a minimum, but a small learning rate might get stuck in a local minimum

Backpropagation

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

UNIVERSITY OF SUSSEX

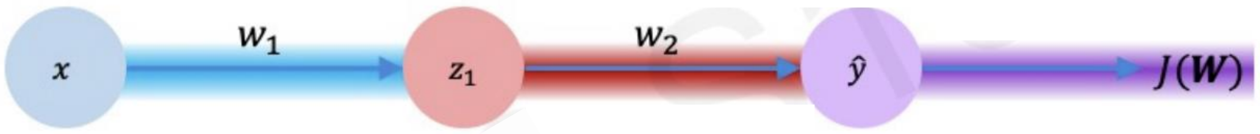
Backpropagation



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

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

Backpropagation



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

Applying the chain rule !

Recommended resources

- Stanford's *Convolutional neural networks for visual recognition* notes on neural networks https://cs231n.github.io/neural-networks-1/

- MIT's intro to deep learning course https://introtodeeplearning.com/

- A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks tutorial2.pdf