Introduction to Artificial Neural Networks (ANN) – Part A

Dr Maria Anastasiadi
(m.anastasiadi@cranfield.ac.uk)

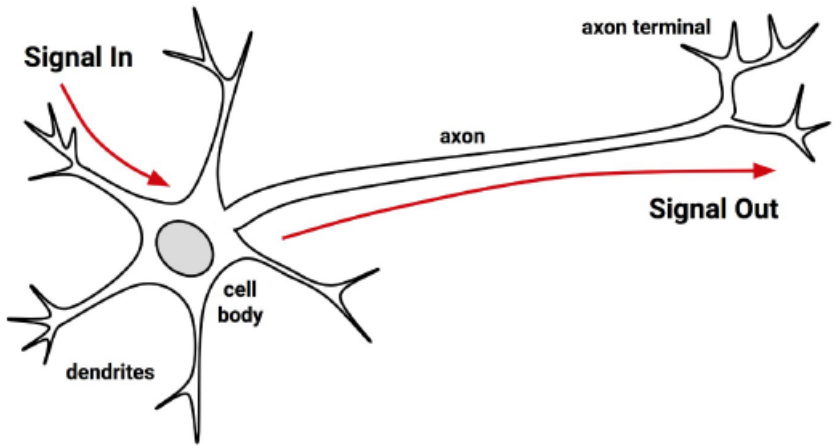
15th January 2025

www.cranfield.ac.uk

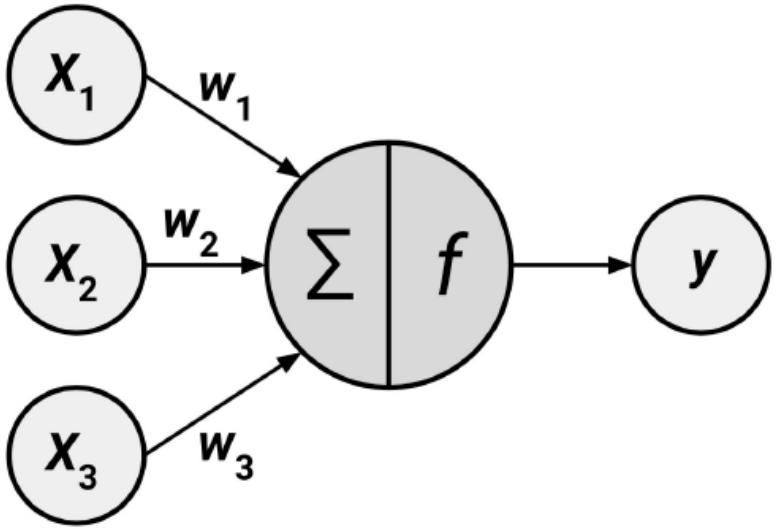
ML: Black Box methods

Neural Networks

- **Artificial Neural Networks** (**ANNs**) model the relationship between input signals and an output signal using a model derived from our understanding of how the brain responds to sensory stimuli.
- ANN uses a network of artificial neurons or **nodes** to solve learning problems in a similar way as a brain uses a network of interconnected **neurons** to create a massive parallel processor.



Biological neuron



Artificial neuron

Neural Networks

Examples of ANNs applications:

- Speech and handwriting recognition programs like voicemail transcription services.
- The automation of smart devices like self-driving cars.
- Models predicting weather and climate patterns.
- Medical imaging and diagnosis.
- Prediction and financial analysis.
- Character recognition (handwriting analysis).

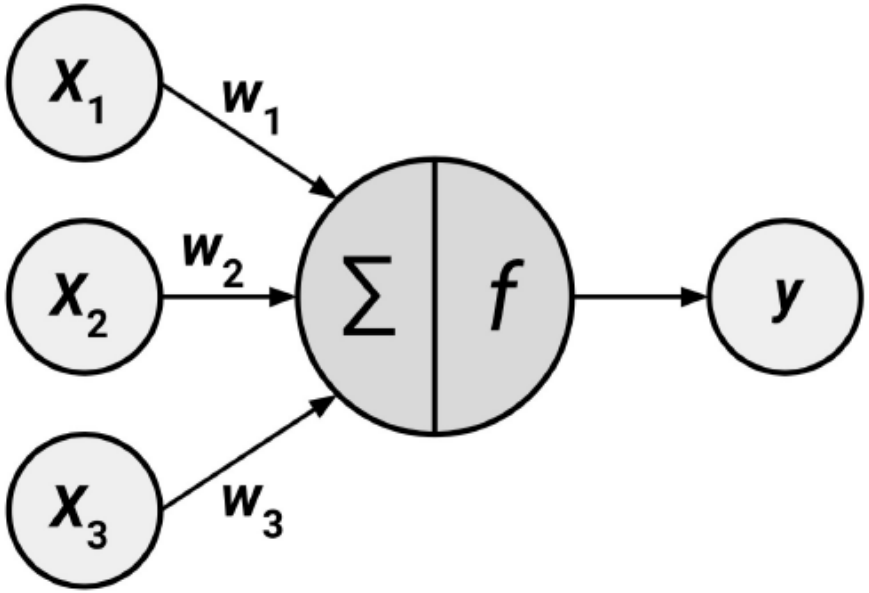
Neural Networks – Defining Characteristics

All ANN are defined by the following characteristics:

- An **activation function f();**
- A **network topology** (or architecture);
- The **training algorithm.**

Neural Networks – Defining Characteristics

Activation function f() transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network.



Artificial neuron

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Where: y(x) = the resulting signal

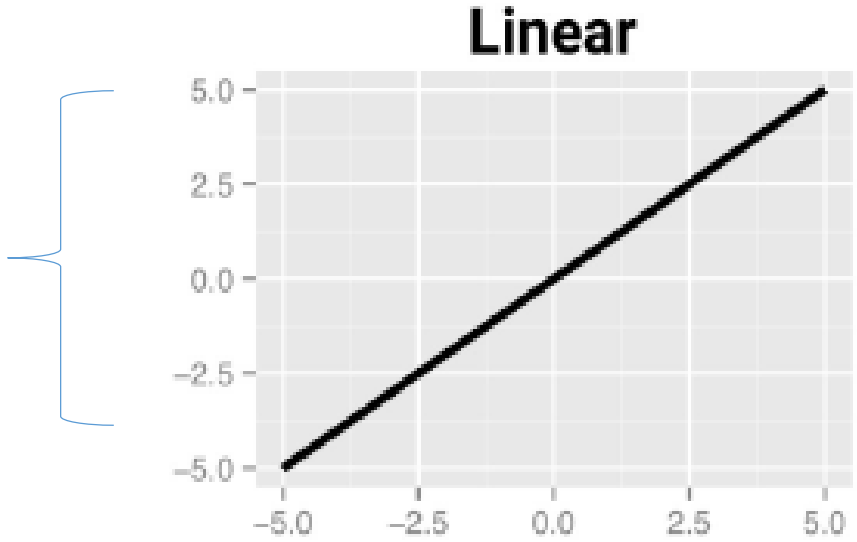
f() = the activation function

w_i = the weight of each input x_i

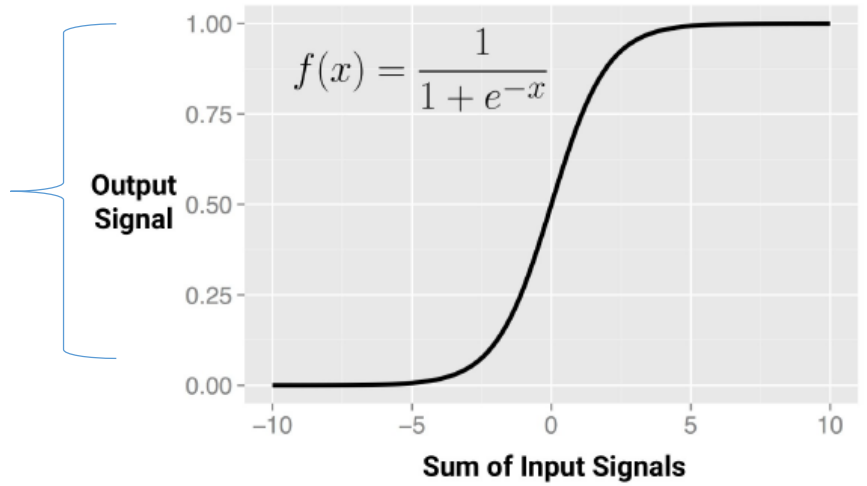
b = the bias vector

Neural Networks- Activation functions

The linear activation function: results in a neural network very similar to a linear regression model. Used for regression problems



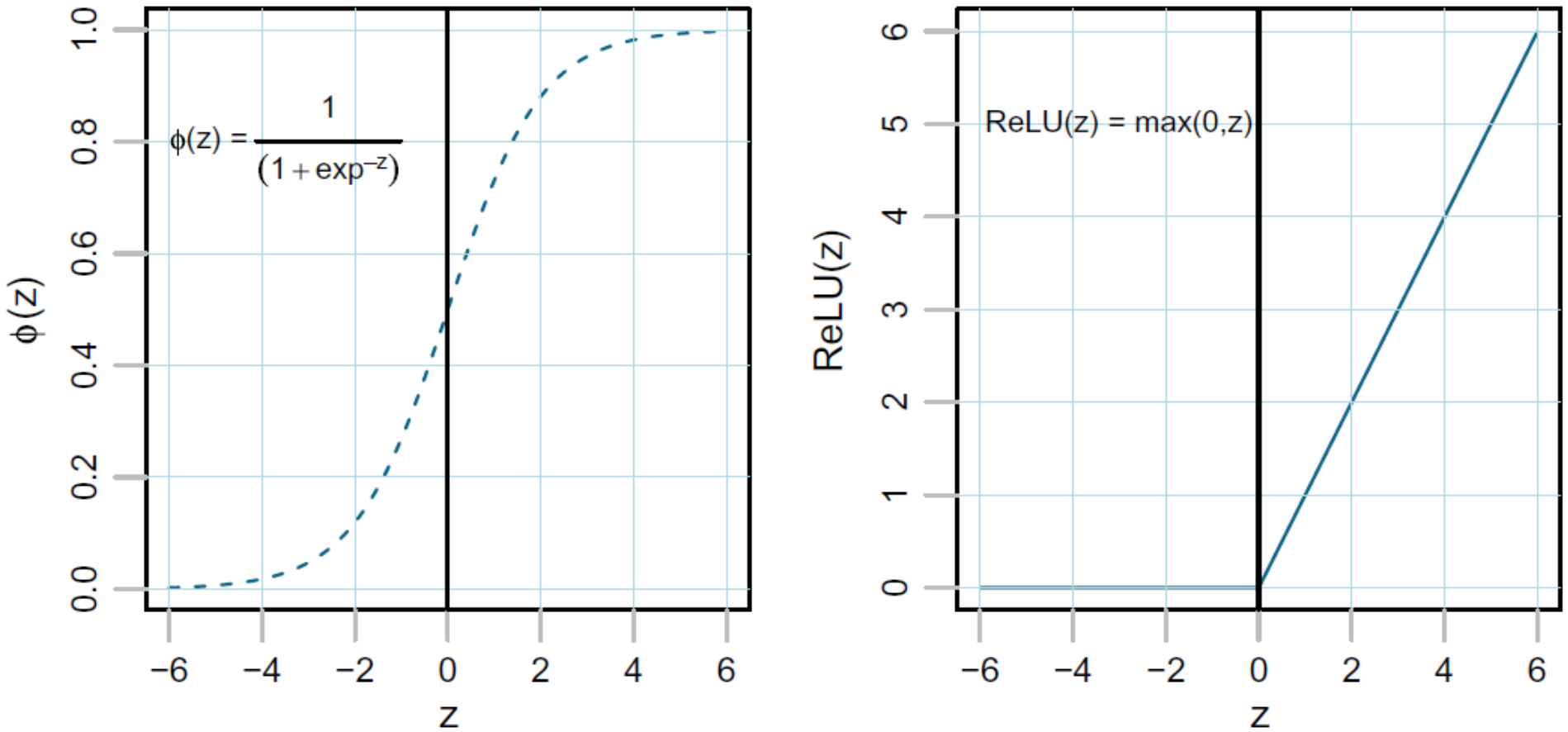
The sigmoid activation function: the output signal can fall anywhere in the range from 0-1. Most commonly used in ANN for binary classification.



For multiclass problems we use **softmax** activation function

Neural Networks- Activation functions

The rectified linear unit **ReLU** is the most used activation function for the hidden layers.



Sigmoid function vs ReLU

Neural Networks - Network topology

Network topology describes the number of neurons in the model and the number of layers and manner in which they are connected.

The ability of a neural network to learn is rooted in its **topology.**

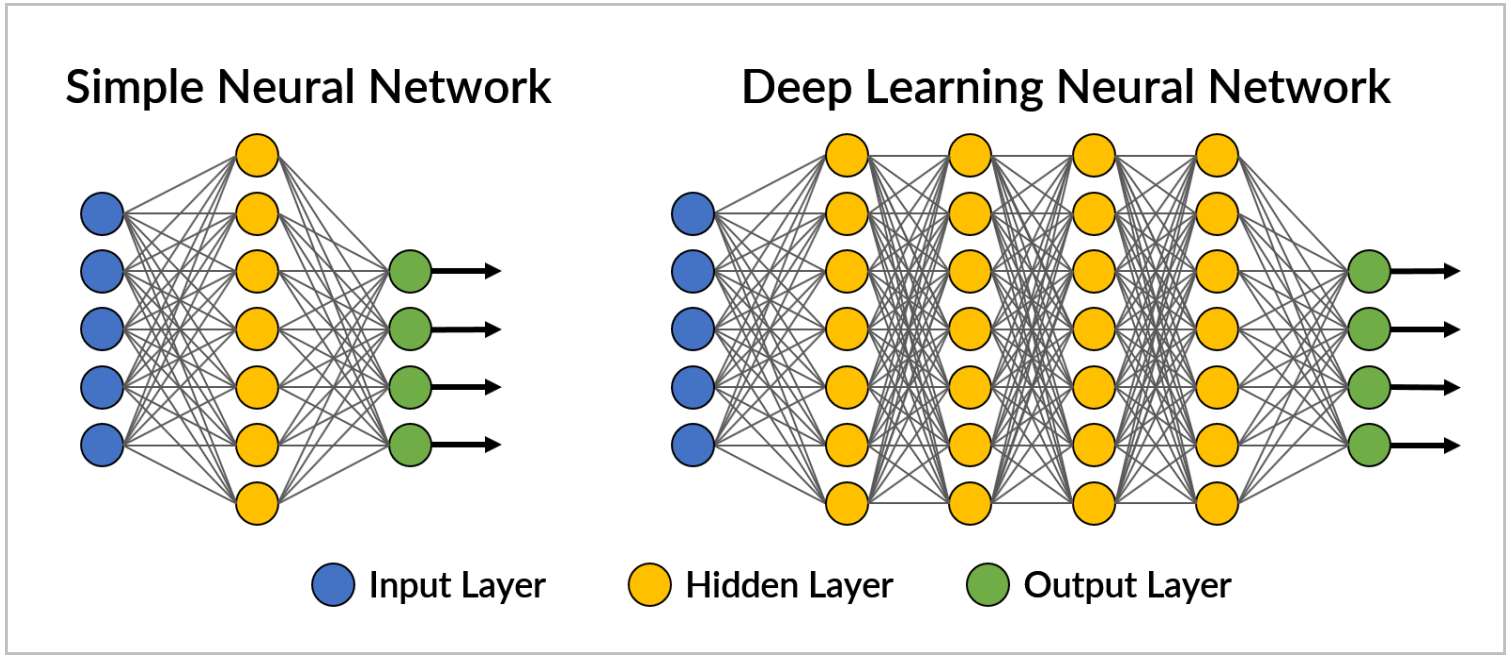
Defining characteristics:

- The number of layers;
- Whether information in the network is allowed to travel backward;
- The number of nodes within each layer of the network.

The power of a network is not only a function of the network size, but also the way units are arranged.

Neural Networks - Network topology

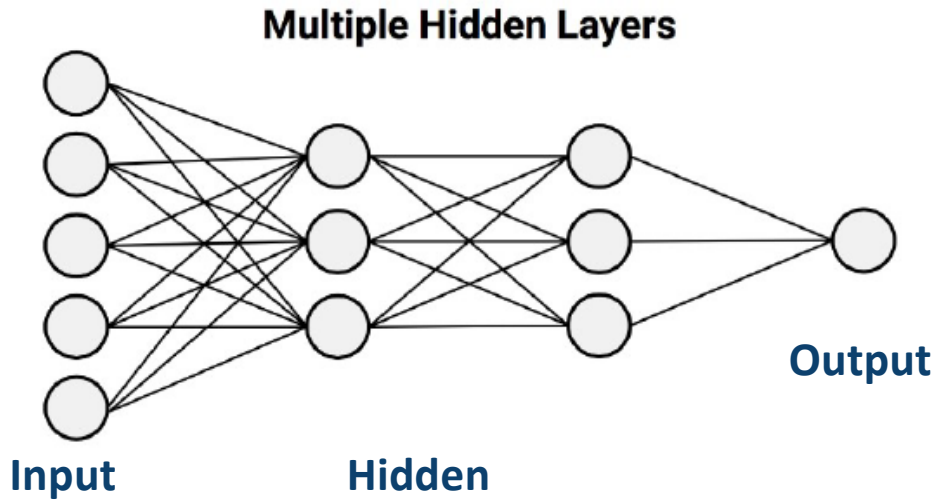
- Single-layer networks can be used for basic pattern recognition.
- A deep neural network (DNN) is a neural network with one input layer, one output layer and multiple hidden layers.
- Deep learning is enabled through computation power and big data developments.

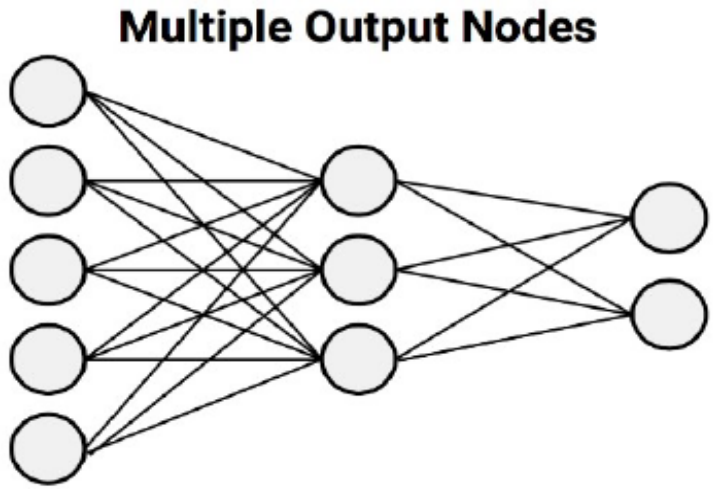


Neural Networks - Network topology

Number of nodes in a network

- The number of input nodes is predetermined by the number of features in the input data.
- The number of output nodes is predetermined by the number of outcomes to be modelled or the number of class levels in the outcome.
- The number of hidden nodes is left to the user to decide prior to training the model.





Neural Networks – Feedforward NN

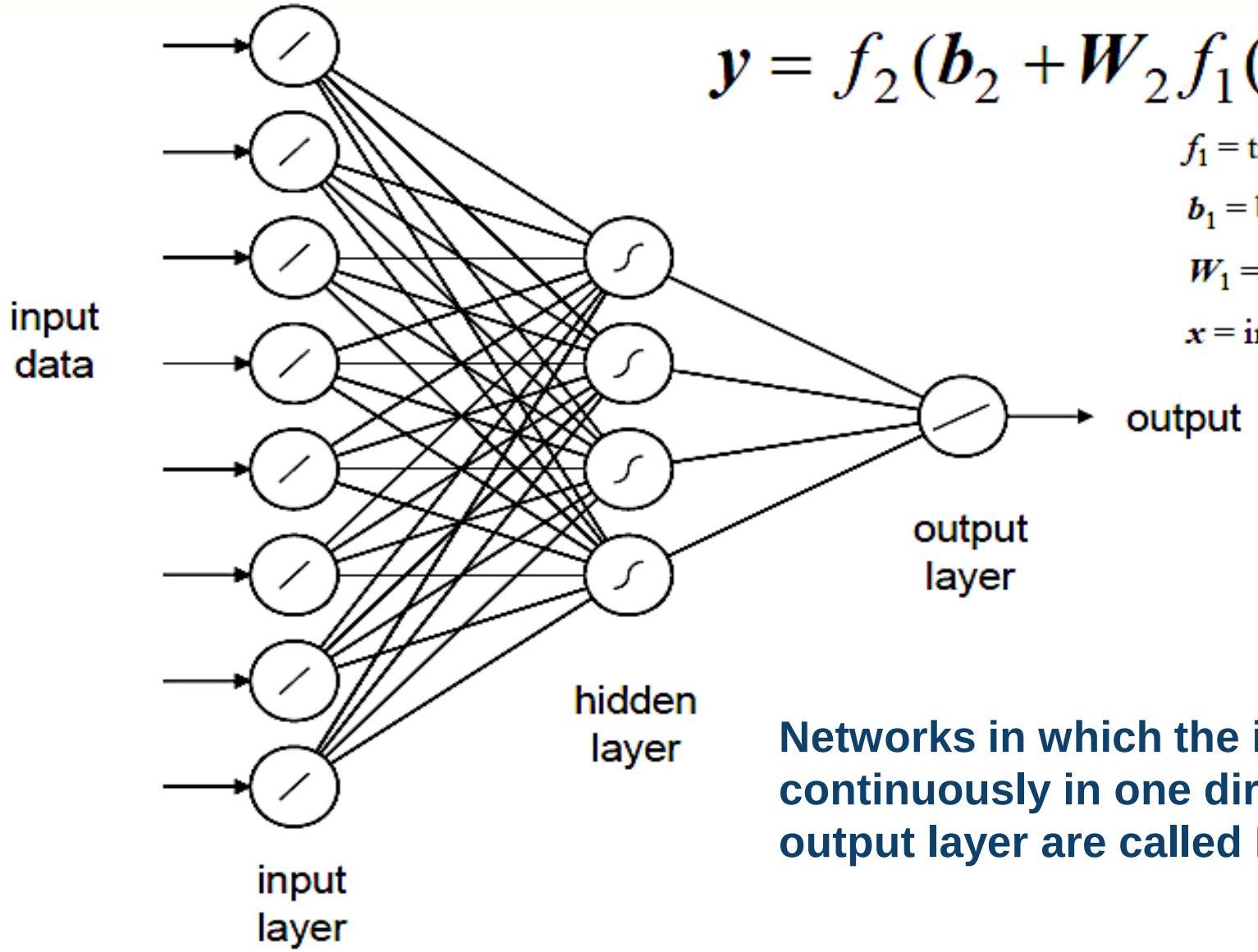
$$\boldsymbol{y} = f_2(\boldsymbol{b}_2 + \boldsymbol{W}_2 f_1(\boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{x}))$$

f_1 = transfer function of layer 1

$\boldsymbol{b}_1$ = bias vector of layer 1

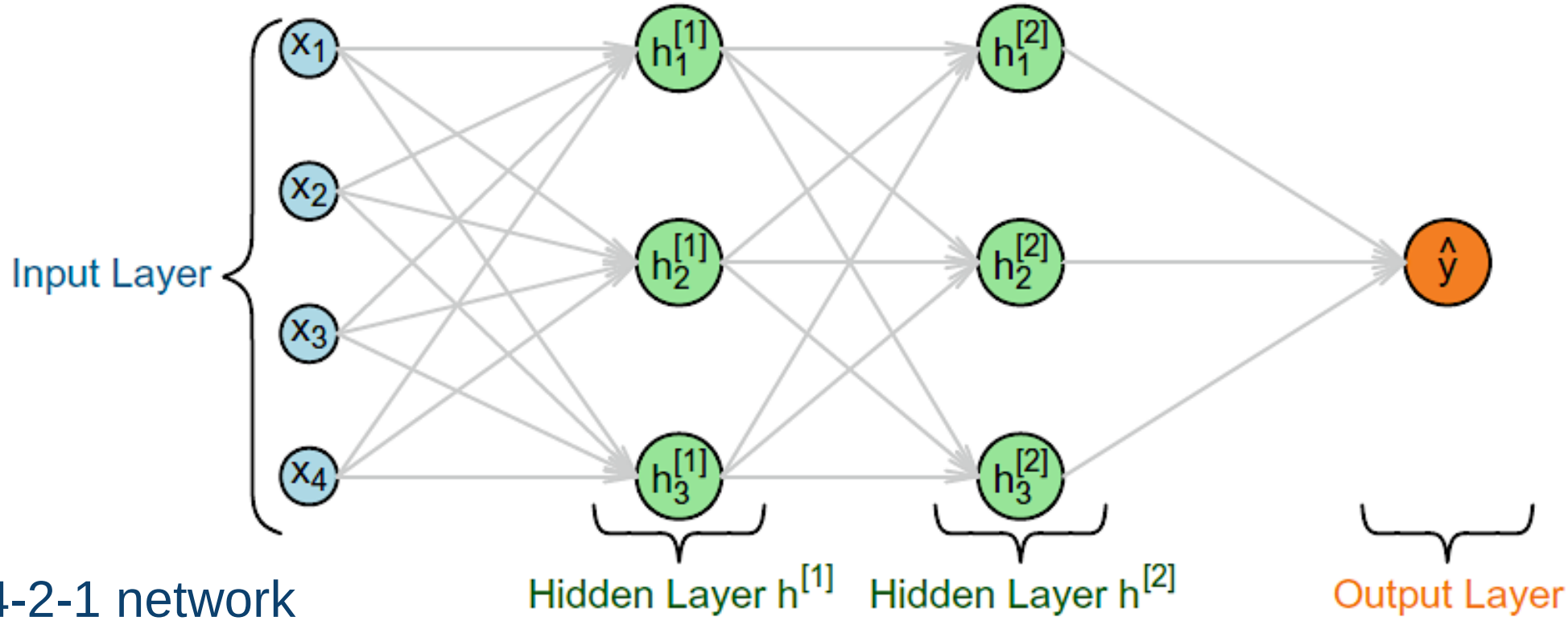
$\boldsymbol{W}_1$ = weight matrix of layer 1

$\boldsymbol{x}$ = input vector



Networks in which the input signal is fed continuously in one direction until it reaches the output layer are called Feedforward networks.

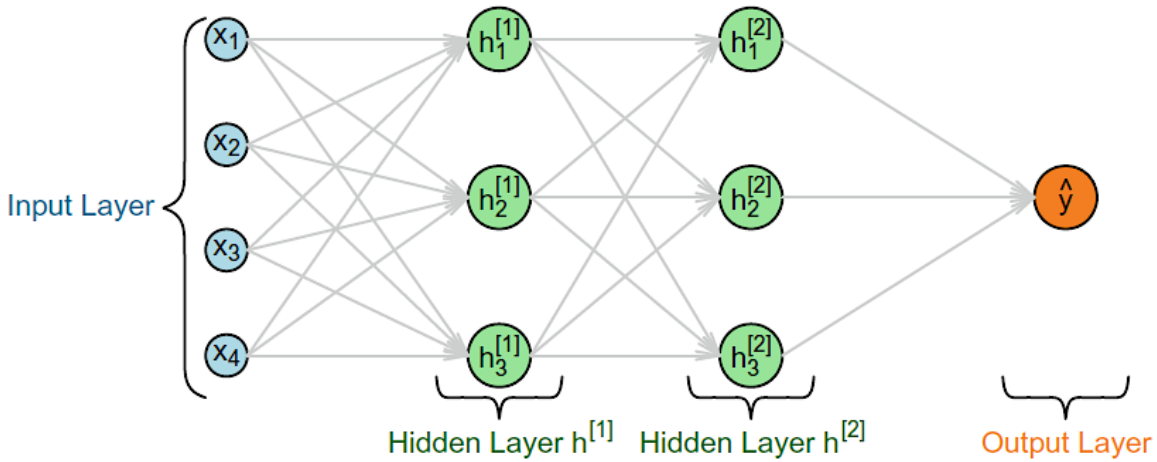
Neural Networks- Feedforward NN



4-2-1 network

For each layer the number of weights is the number of features of the previous layer multiplied by the number of nodes.

Neural Networks- Hidden Layers



$$W^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

For hidden layer 1 (h1) we need 4X3 =12 weights $W^{[1]}$

$$W^{[1]T} \cdot X = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}^T \begin{bmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \\ x_3^{(1)} & x_3^{(2)} \\ x_4^{(1)} & x_4^{(2)} \end{bmatrix}$$

X= Data matrix with 2 samples matrix and 4 features each

Neural Networks- Hidden Layers

After adding the Bias vector we get:

$$\boldsymbol{Z}^{[1]} = \boldsymbol{W}^{[1]T}\boldsymbol{X} + b^{[1]}$$

$$\boldsymbol{Z}^{[1]} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}^{\boldsymbol{T}} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} \\ x_2^{(1)} & x_2^{(2)} \\ x_3^{(1)} & x_3^{(2)} \\ x_4^{(1)} & x_4^{(2)} \end{bmatrix} + \begin{bmatrix} r \\ s \\ t \end{bmatrix}$$

$$= \begin{bmatrix} z_{1,1}^{[1]} & z_{1,2}^{[1]} \\ z_{2,1}^{[1]} & z_{2,2}^{[1]} \\ z_{3,1}^{[1]} & z_{3,2}^{[1]} \end{bmatrix}$$

Neural Networks- Hidden Layers

Finally we apply the activation function g (e.g. sigmoid) to the $Z^{[1]}$ matrix

$$A^{[1]} = \begin{bmatrix} g(z_{1,1}^{[1]}) & g(z_{1,2}^{[1]}) \\ g(z_{2,1}^{[1]}) & g(z_{2,2}^{[1]}) \\ g(z_{3,1}^{[1]}) & g(z_{3,2}^{[1]}) \end{bmatrix}$$

The output from the first hidden layer $A^{[1]}$ are the inputs to the second hidden layer.

The weights for h2 are 3 x 3 = 9

Following the same logic we calculate the activation function for Layer 2.

$$A^{[2]} = \begin{bmatrix} g(z_{1,1}^{[2]}) & g(z_{1,2}^{[2]}) \\ g(z_{2,1}^{[2]}) & g(z_{2,2}^{[2]}) \\ g(z_{3,1}^{[2]}) & g(z_{3,2}^{[2]}) \end{bmatrix}$$

Neural Networks- Output Layer

Consider a binary classification problem (e.g. Class A vs Class B) → 2 output nodes.

- The input to the output layer will be $A^{[2]}$ a 3x2 matrix
- We apply a sigmoid activation function (binary problem) to the $Z^{[3]}$ final matrix.

$$A^{[3]} = \begin{bmatrix} \mathbf{0.66} & 0.46 \\ 0.33 & \mathbf{0.64} \end{bmatrix}$$

- Sample 1 has a 66% probability of belonging to Class A
- Sample 2 has a 64% probability of belonging to Class B

Training Neural Networks - Optimisation

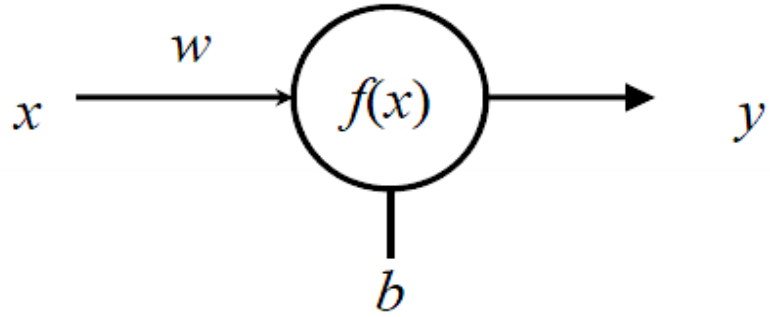
- **Training** of ANNs involves modifying the weights of the connections between network layers such that the **minimum output error** is achieved for the training data set.

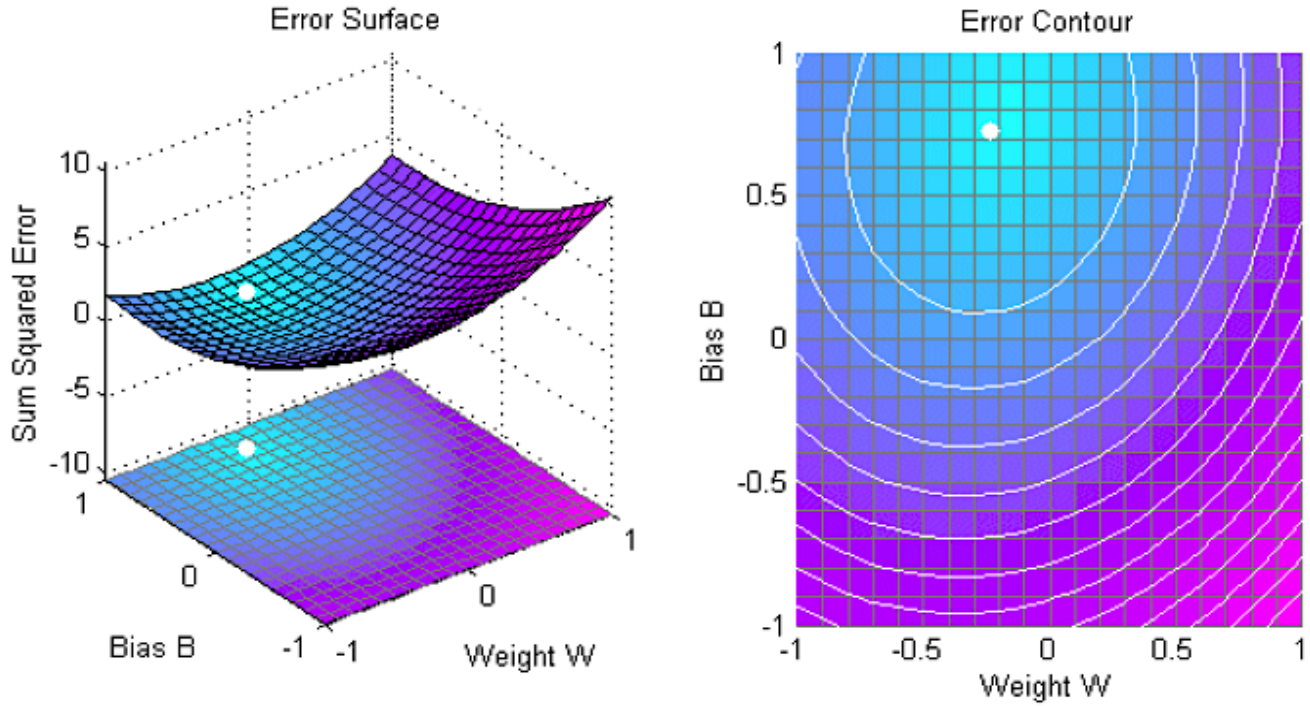
$$\boldsymbol{y} = f_2(\boldsymbol{b}_2 + \boldsymbol{W}_2 f_1(\boldsymbol{b}_1 + \boldsymbol{W}_1 \boldsymbol{x}))$$

- The most common training method is '**back propagation of errors**' whereby weights of interconnections are iteratively adjusted to minimise the error between the expected outputs and the actual outputs.
- Just like with the statistical techniques, a set of training data from known samples is needed, and a set of validation samples to test the network's ability to generalise.

`Keras` library in python or R

Training Neural Networks - Optimisation



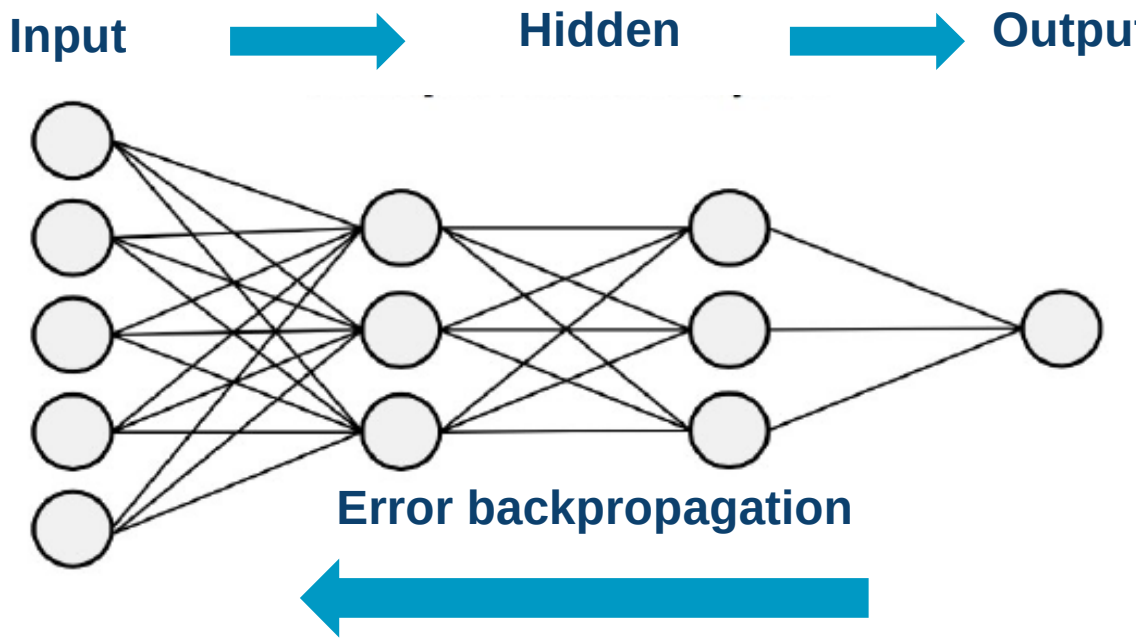


Same principle as in multivariate regression

Training Neural Networks - Optimisation

Backpropagation Algorithm

- In its most general form, the backpropagation algorithm iterates through many cycles known as **epochs**.
- Because the network contains no *a priori* knowledge, the starting weights are typically set at random.

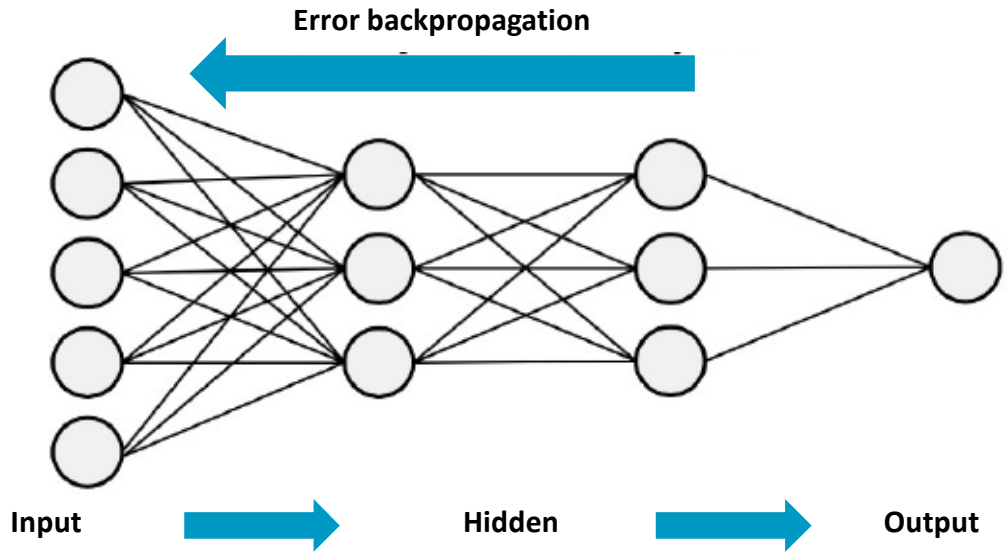


Training Neural Networks - Optimisation

Backpropagation Algorithm

Each epoch in the backpropagation algorithm includes:

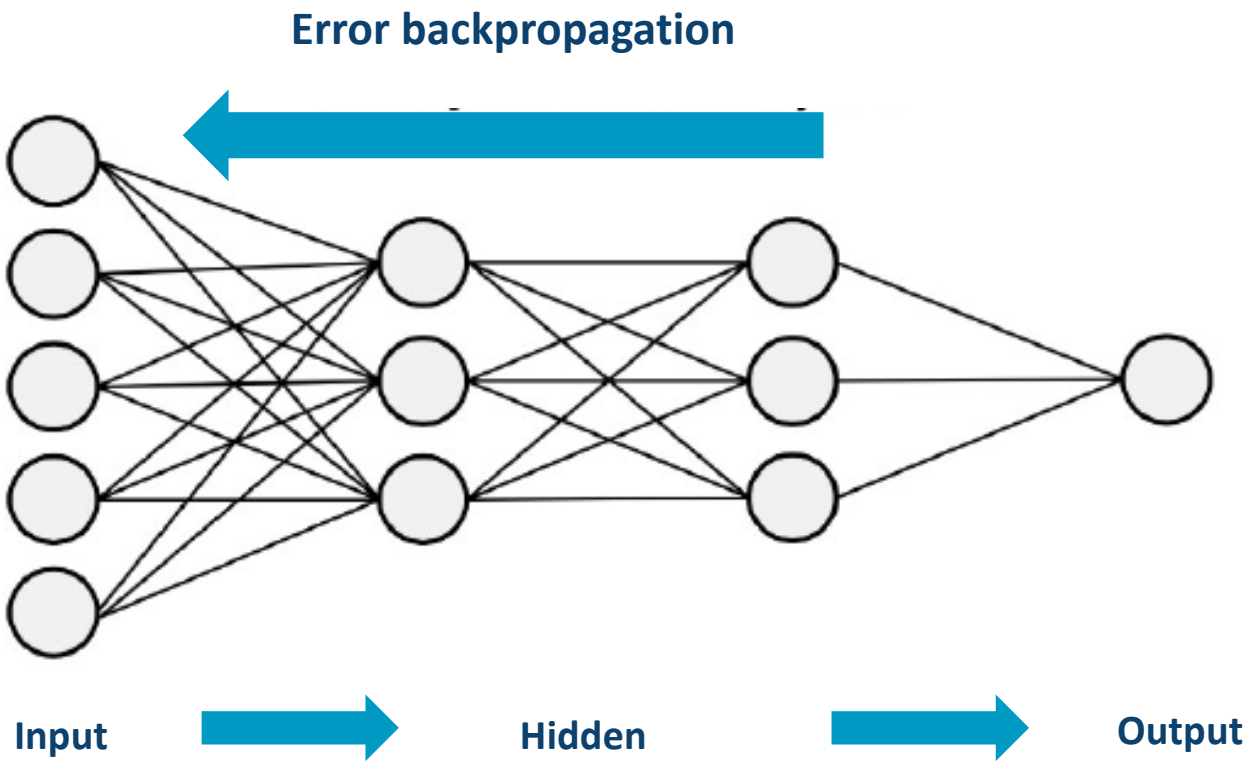
- A **forward** phase in which each neuron in the network processes the input data in sequence from the input layer to the output layer until an output signal is produced.
- A **backward** phase in which the network's output signal resulting from the forward phase is compared to the true target value in the training data. This results in an *error* value which is propagated backwards in the network to modify the connection weights between neurons.



Training Neural Networks - Optimisation

Backpropagation Algorithm

- Over time, the network uses the information sent backward to reduce the total error of the network and produce the correct output. i.e. the network *learns*.

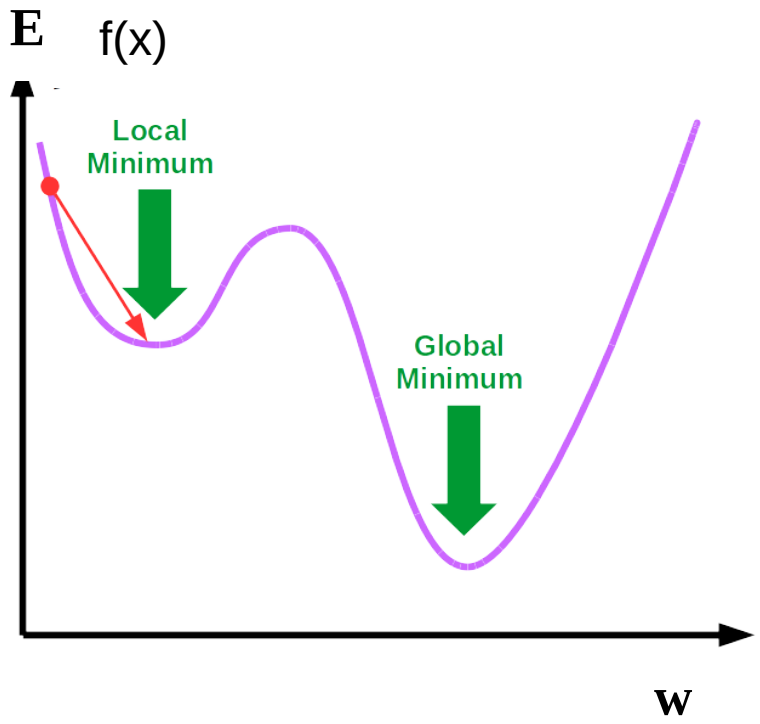


Optimising Neural Networks – Gradient Descent

Loss (Cost) Function Optimisation

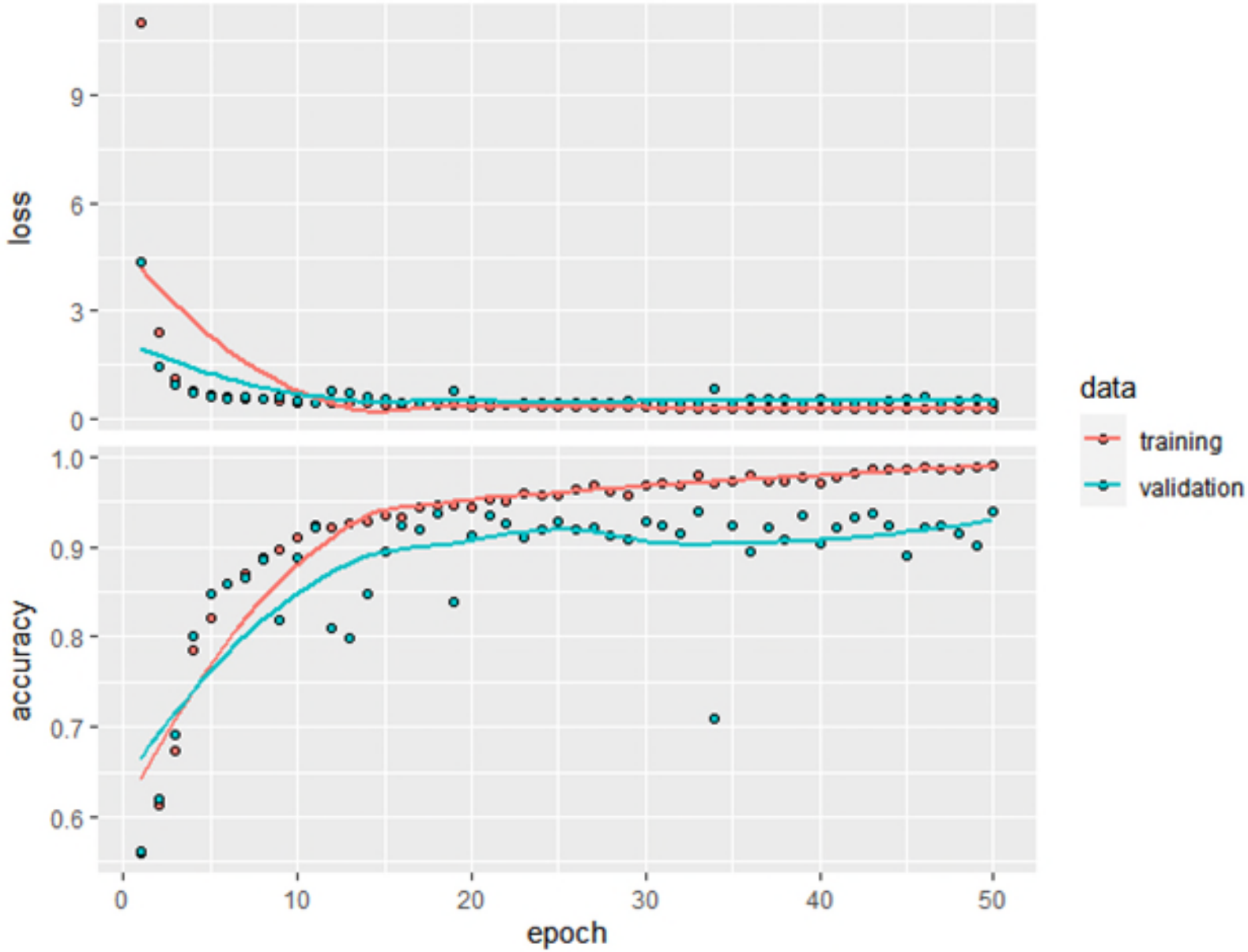
But how does the algorithm determine how much a weight should be changed?

- This is done with a technique called **gradient descent**.
- The backpropagation algorithm uses the derivative of each neuron's activation function to identify the gradient in the direction of each of the incoming weights.
- The gradient suggests how steeply the error will be reduced or increased for a change in the weight.
- The algorithm will attempt to change the weights that result in the greatest reduction in error by an amount known as the **learning rate**.



Training Neural Networks

Plot of accuracy and loss with respect to epochs using a keras model for binary classification



Optimising Neural Networks – Regularisation

- Weight inflation can lead to overfitting of a NN.
- Regularization is a method by which we impose constraints on our neural network to prevent overfitting.
- **Dropout Regularisation**: Forces a percentage of neurons in a layer to be deactivated.
- ℓ_2 regularization applies a regularisation term to the loss function to force all the weights to smaller values by making it costly to have large weights.

$$J = \frac{1}{n}\sum_i (y_i - f(x_i))^2 + R(f)$$

$$R(f) = \frac{1}{2}\lambda \sum w^2$$

Small weights minimise loss function

Training Neural Networks

Neural Networks	
Strengths	Weaknesses
Can be used for classification or numeric prediction problems	Computationally intensive and slow to train
Capable of modelling very complex patterns	Very prone to overfitting
Makes few assumptions about the underlying relationships in a dataset	Model output very difficult even impossible to interpret

Other NN algorithms:

- Probabilistic Neural Networks (PNNs)
- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNN)

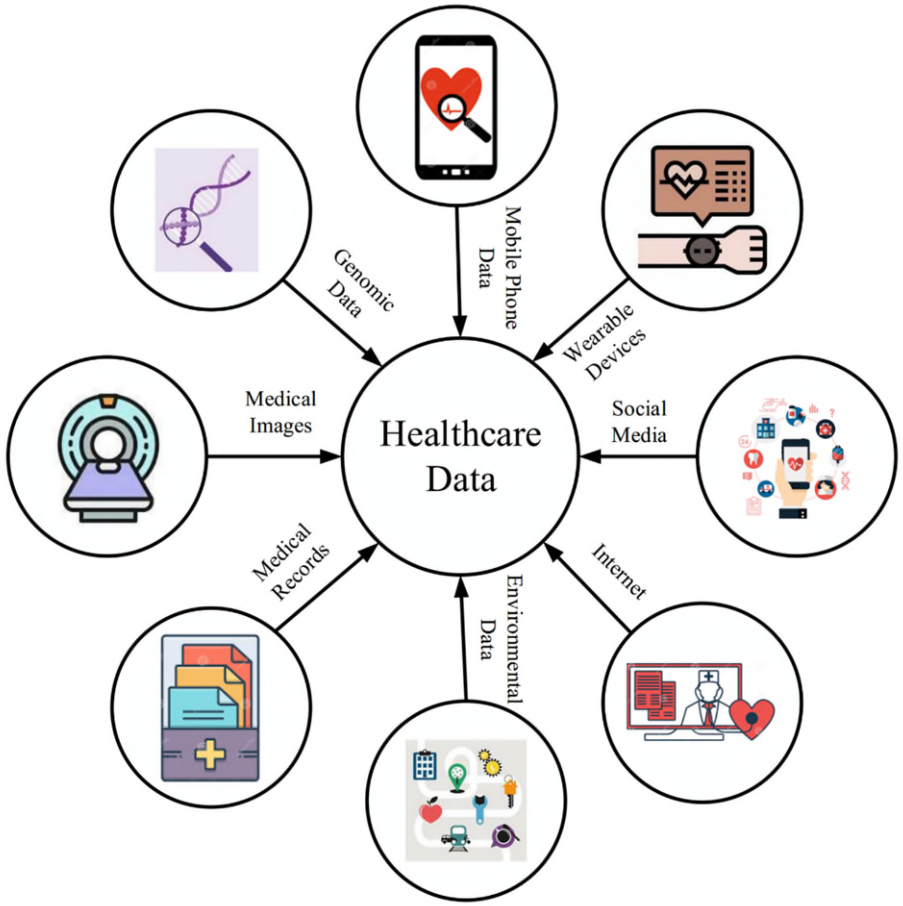
Neural Networks

Examples of ANNs applications:

- Speech and handwriting recognition programs like voicemail transcription services.
- The automation of smart devices like self-driving cars.
- Models predicting weather and climate patterns.
- Medical imaging and diagnosis.
- Prediction and financial analysis.
- Character recognition (handwriting analysis).

Applications of ANN in Healthcare

- Huge number of personal data can be generated through a variety of sources.
- We can use this information to devise personalized health plans for disease prevention and treatment.



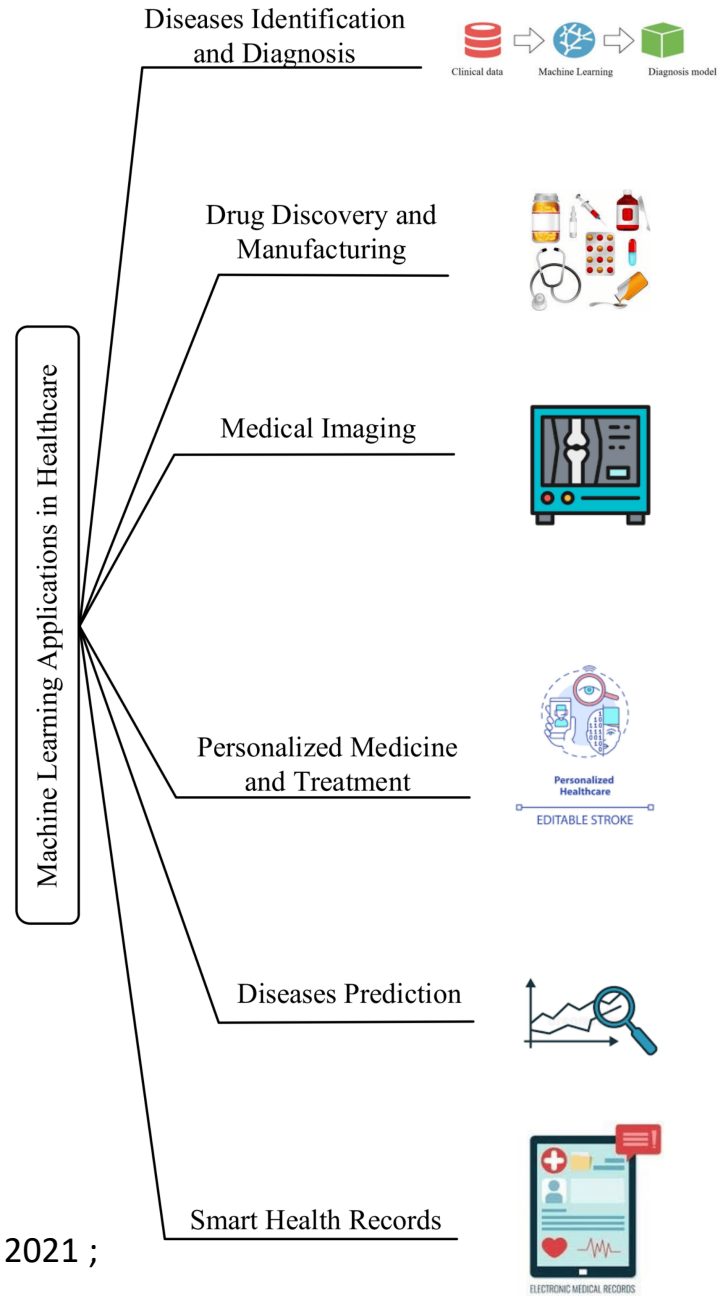
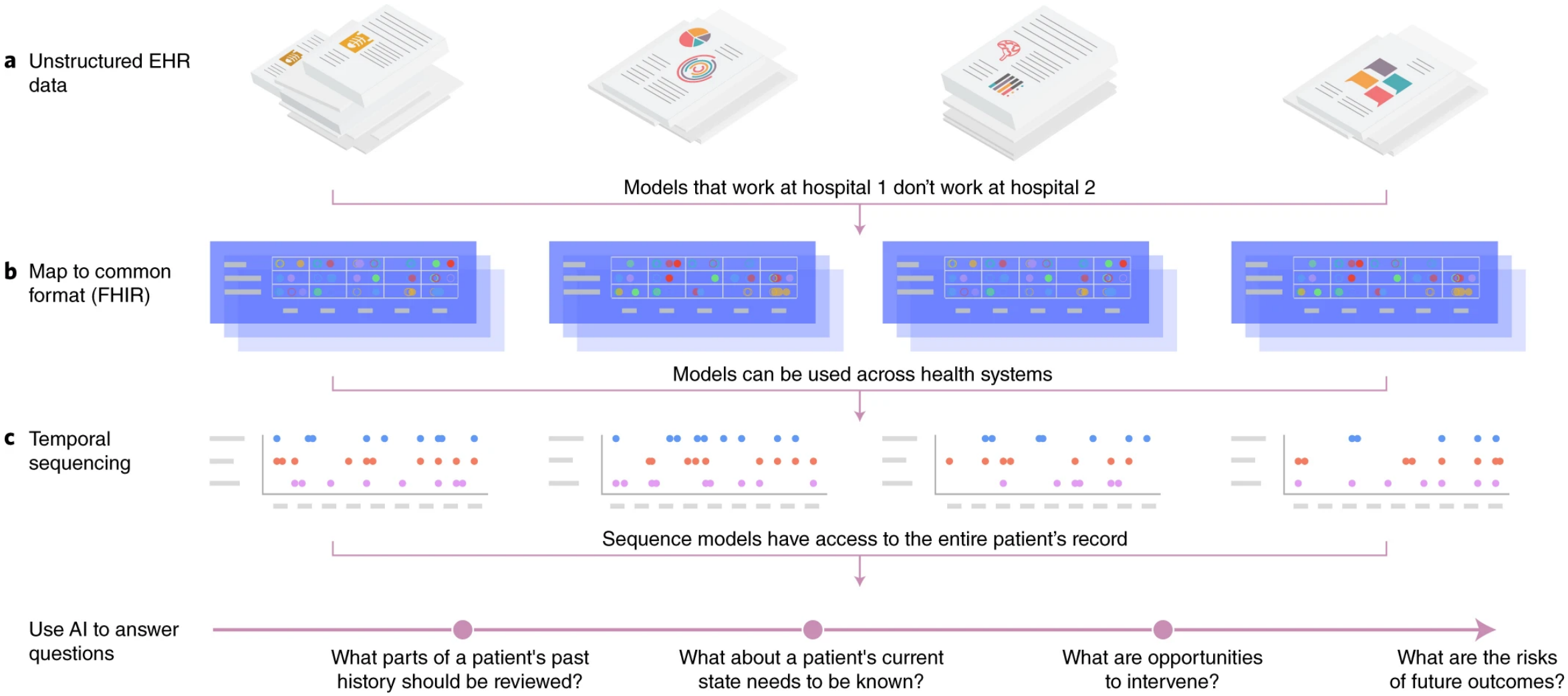


Image adapted from: Rahmani A. M. et al., *Mathematics* 2021 ; https://doi.org/10.3390/math9222970

Application of ANN in Electronic Health Records (HER)



(Figure ref: Esteva A. et al., *Nat Med* (2019). https://doi.org/10.1038/s41591-018-0316-z)

Summary

- Introduction to Black Box Methods in ML: ANN
- ANN characteristics
- ANN optimisation
- Examples of ANN applications

www.cranfield.ac.uk

T: +44 (0)1234 750111

@cranfielduni

@cranfielduni

/cranfielduni