Deep Learning and its Applications

3RSM Research Project

Tom Wright

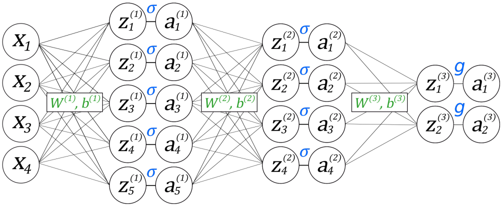
University of Birmingham

June 5, 2023

What is deep learning?

What is deep learning?

- Deep learning is a type of **machine learning**.
  Statistical technique for extracting knowledge from data (e.g. driving a car, translating text).
- Centres around the neural network (NN).
- Inspired by the human brain.
  Composition of many simple functions → complex behaviour.



History of deep learning

Deep learning has developed in three distinct waves.

- **Cybernetics** (1940s - 1960s)
  Simple linear networks, inspired by advances in psychology and neuroscience
- **Connectionism** (1980s - 1990s)
  Non-linearity, backpropagation, multi-layer networks
- **Deep Learning** (2000s - present)
  More powerful computers, larger datasets $\rightarrow$ *larger networks, increased complexity and accuracy*

Deep learning in recent times

Deep learning is now used widely throughout the modern world.

- Facial detection
- Medical diagnosis
- Speech recognition (e.g. Siri)
- Natural language processing (e.g. ChatGPT)
- Many other cases

Supervised Learning

Supervised Learning

$$\begin{pmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_d \end{pmatrix}^{\in \mathbb{R}^d} = X \xrightarrow{\substack{\text{"True" Relationship} \\ Y = f^*(X) + \epsilon}} Y \in \mathcal{Y}$$

Realisations

Realisations

$$\boxed{s = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^{n}}$$

Dataset

Goal of Supervised Learning

Given:

- Input $X \in \mathbb{R}^d$ and output $Y = f^*(X) + \epsilon$,
- Dataset $s = \left\{\left(x^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n}$,

determine a function

$$f_s : \mathbb{R}^d \to \mathcal{Y}$$

that makes an accurate **prediction**

$$\hat{y} = f_s(x)$$

of y on unseen observations.

Supervised Learning Task

A supervised learning task involves identifying the following.

1. A **loss function** $\mathcal{L}(f, (x, y))$ that measures the performance of $f : \mathbb{R}^d \to \mathcal{Y}$ on a single data point.
2. A **hypothesis set** $\mathcal{F}_\theta \subset \mathcal{M}(\mathbb{R}^d, \mathcal{Y})$ that defines a model for f_s with parameters θ (we will use neural network!).
3. A **training algorithm** $\mathcal{A}$ that outputs an optimised model $f_s = \mathcal{A}(s)$.

The algorithm $\mathcal{A}$ typically chooses θ by minimising **cost**, the average loss over the dataset:

$$C(f, s) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f, s^{(i)}).$$

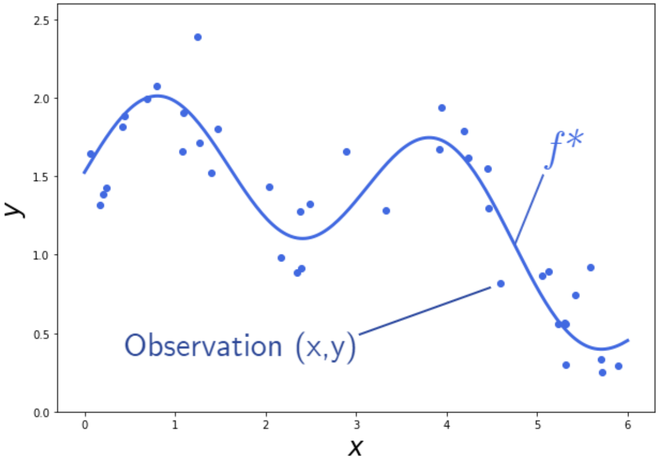
Regression vs Classification

Definition

In a **regression** task, the output Y is a vector of continuous real values with $Y \in \mathcal{Y} = \mathbb{R}^m$.

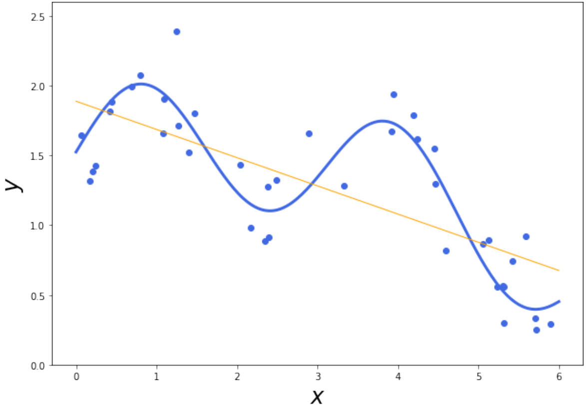
In a **classification** task, the output Y is one of K discrete classes, with $Y \in \mathcal{Y} = \{0, 1, 2, K - 1\}$.

Simple Regression



- $X, Y \in \mathbb{R}$
- Each point is an observation $(x^{(i)}, y^{(i)}) \in s$.

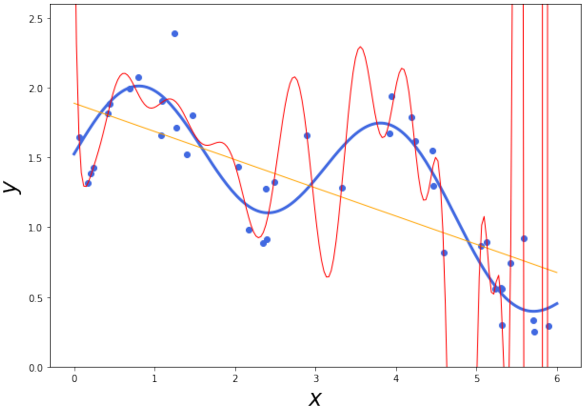
Fitting linear model



- Linear model $\mathcal{F}_{\{a,b\}} = \{f(x) = ax + b \mid a, b \in \mathbb{R}\}$,
- Train using squared residual loss:

$$\mathcal{L}(f, (x, y)) = (f(x) - y)^2.$$

Overfitting and Underfitting



- Linear model (orange) underfits.
- Polynomial (red) overfits.
- Neither is close to the true function f^*.

Overfitting neural networks

- Neural networks **generalise** remarkably well! [1]
  They tend to not overfit: if they fit the training data, they tend to be close to the true distribution
- We will focus on neural networks' ability to fit training data.

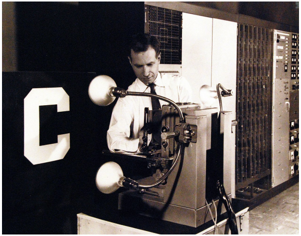
Neural Networks

Perceptron

- Simple linear network for binary classification i.e.

$$\mathcal{Y} = \{0, 1\}.$$

- Invented in 1943 by McCulloch and Pitts.
- Implemented as enormous eletronic machine by Frank Rosenblatt in 1958.



Source: Wikimedia Commons

Figure: Frank Rosenblatt with his Perceptron machine

Perceptron - model

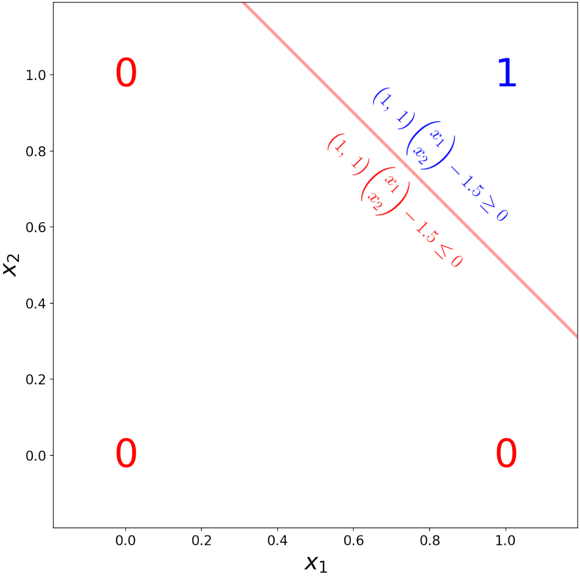
- Input: $x = (x_1, x_2, \dots, x_d)^T$
- Computes weighted sum

$$z(x) = w^T x + b = w_1 x_1 + \cdots + w_d x_d + b$$

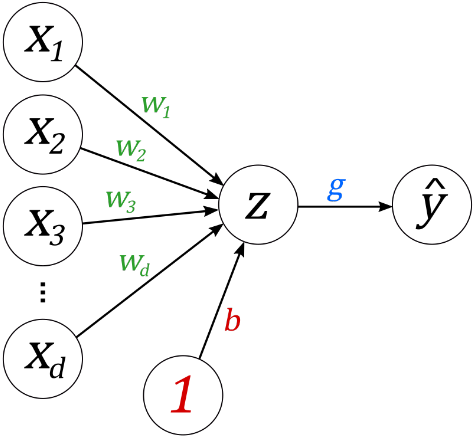
  for **weights** $w \in \mathbb{R}^d$ and **bias** $b \in \mathbb{R}$.
- Applies threshold $g(z) = \mathbb{1}(z > 0)$ to obtain prediction.
- Overall model is

$$\hat{y} = (g \circ z)(x) = \begin{cases} 1 & \text{if } w^T x + b > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Perceptron - example classifier

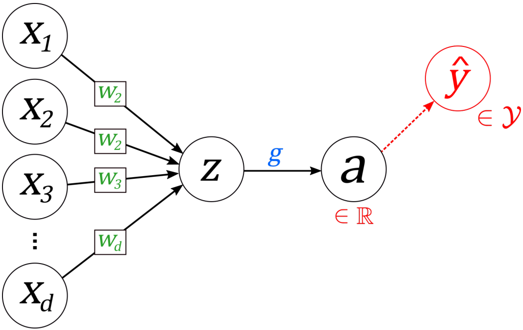


Perceptron - computational graph



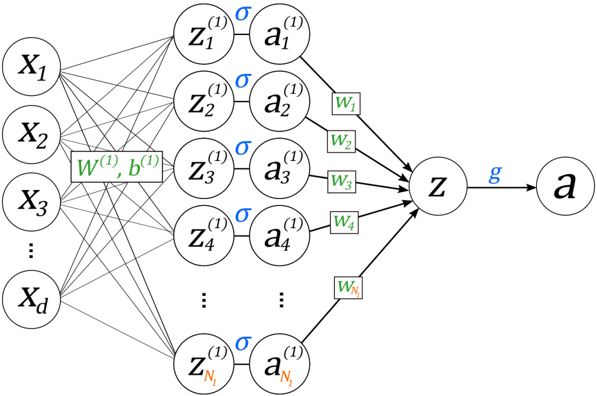
$$\hat{y} = (g \circ z)(x) = \begin{cases} 1 & \text{if } w^T x + b > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Extending perceptron



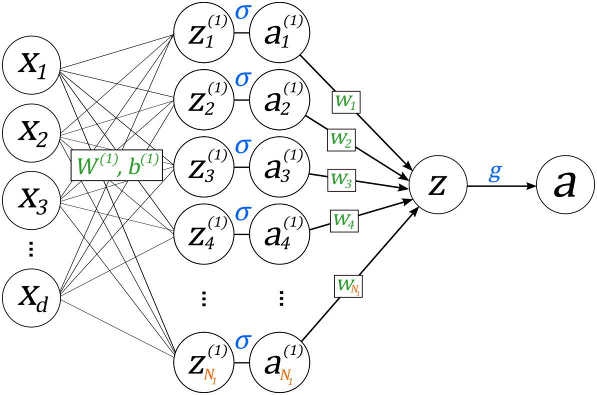
- Replace $\hat{y} \in \mathcal{Y}$ with $a \in \mathbb{R}$ (better for optimisation).
- Use a to predict $\hat{y}$ (depending on use case).
- Map $g : \mathbb{R} \to \mathbb{R}$ is any suitable output map.

Extending perceptron



- Introduce vector $z^{(1)} \in \mathbb{R}^{N_1}$; each entry is a distinct weighted sum of the inputs.
- Vector $a^{(1)} = \sigma(z^{(1)})$; σ is non-linear **activation function** applied element-wise.

Extending perceptron



- Entries $z_j^{(1)}$ are called **pre-activations**.
- Entries $a_j^{(1)}$ are called **activations**.

Matrix Notation

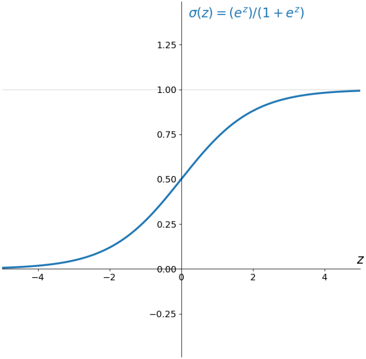
- Vector $z^{(1)}$ can be written

$$z^{(1)} = \begin{pmatrix} {w^1}^T x + b_1 \\ {w^2}^T x + b_2 \\ \vdots \\ {w^{N_1}}^T x + b_{N_1} \end{pmatrix} = W^{(1)} x + b^{(1)},$$

with

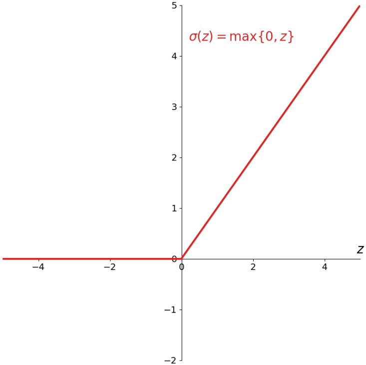
$$W^{(1)} = \begin{pmatrix} w_1^1 & w_2^1 & \cdots & w_d^1 \\ w_1^2 & w_2^2 & \cdots & w_d^2 \\ \vdots & \vdots & \ddots & \vdots \\ w_1^{N_1} & w_2^{N_1} & \cdots & w_d^{N_1} \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N_1} \end{pmatrix}.$$

Activation Functions



Sigmoid

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

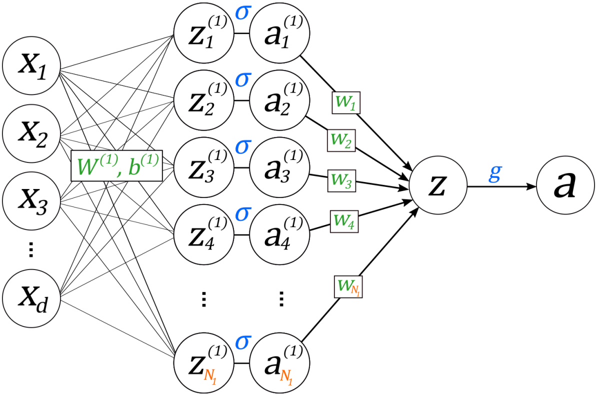


Rectified Linear Unit **(ReLU)**

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

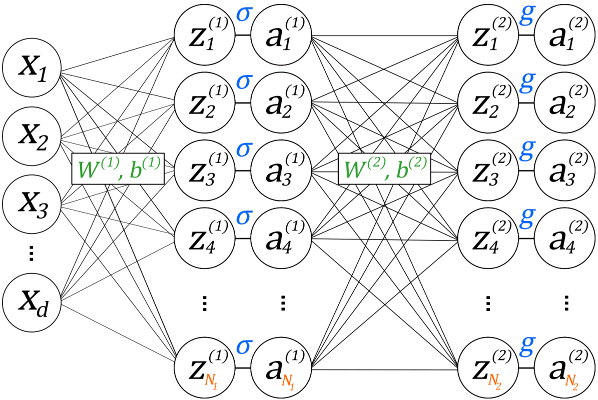
(More efficient for large networks on digital computers.)

Extending perceptron



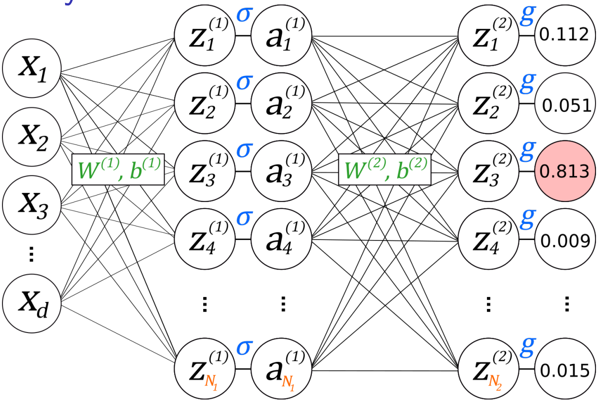
- We have $a = g(z) = g(w^T\sigma(W^{(1)}x + b^{(1)}) + b)$

Extending perceptron



- Replace z and a with vectors $z^{(2)} = W^{(2)}z^{(1)} + b^{(2)}$ and $a^{(2)} = g(z^{(2)})$

Output layer



- Different types of output layer are possible.
- For multi-class classification, may output probability for each class using a **softmax** output layer:

$$g_i(z) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}.$$

General Neural Network

A neural network with $L \in \mathbb{N}$ layers is defined by architecture $\alpha = (N, \sigma, g)$, where

- $N = (N_0, N_1, \dots, N_L) \in \mathbb{N}^{L+1}$ gives number of units in each layer,
- $\sigma : \mathbb{R} \to \mathbb{R}$ is activation function,
- $g : \mathbb{R} \to \mathbb{R}$ is mapping defining output layer.

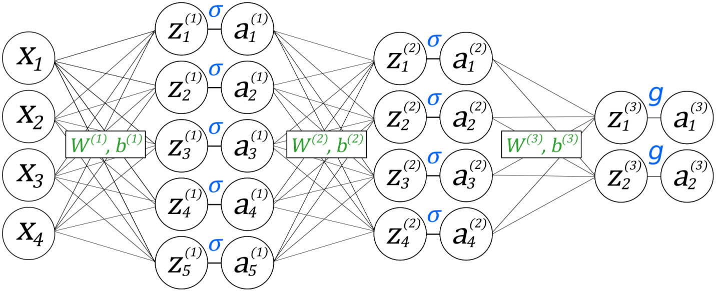
Activations

In layer $l = 1, 2, \ldots, L$ we have **activations** $a^{(l)} \in \mathbb{R}^{N_l}$ and **preactivations** $z^{(l)} \in \mathbb{R}^{N_l}$, related by the following.

$$z^{(1)}(x) = W^{(1)}x + b^{(1)},$$
$$z^{(l)}(x) = W^{(l)}a^{(l-1)} + b^{(l)} \quad \forall\ l = 2, 3, \ldots, L,$$

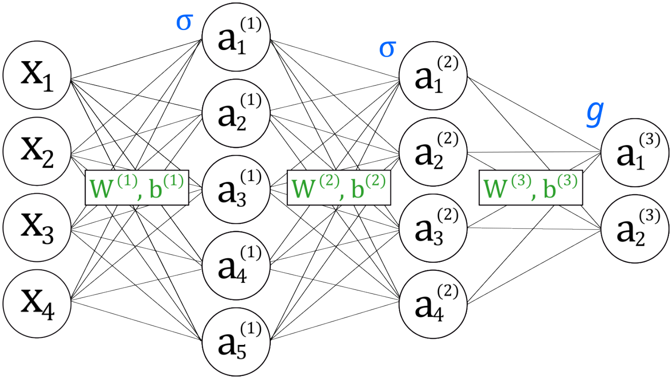
$$a^{(l)}(x) = \sigma(z^{(l)}) \qquad \forall\ l = 1, 2, \ldots, L-1,$$
$$\text{and} \quad a^{(L)}(x) = g(z^{(L)}).$$

Example: a deep network



Architecture $\alpha = ((4, 5, 4, 2), \sigma, g)$.

Example: a deep network

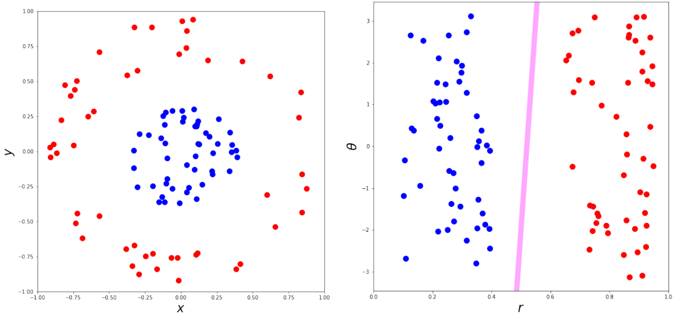


- Architecture $\alpha = ((4, 5, 4, 2), \sigma, g)$, preactivations omitted.
- These networks are also known as multi-level perceptrons (MLP).
- They are called **feed-forward** and **fully-connected**.

Representing XOR

Representations

- A **feature** is a measurable factor of variation.
- Traditional ML requires engineer to design good **representation**:
  A set of features that describe variation in the data well.

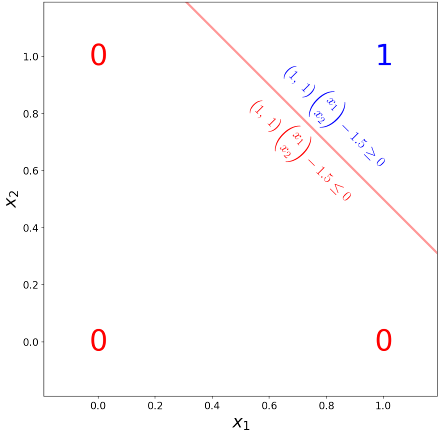


Representation learning in NNs

- Each layer of a neural network encodes an increasingly complex representation of the data.
  e.g. edges $\rightarrow$ *regions* $\rightarrow$ *facial features* $\rightarrow$ *identity*
- More layers, more units $\rightarrow$ more complex representations.
- By optimising network parameters to fit training data, a good representation is automatically selected.

AND Problem

. . . is linearly separable, so perceptron works!



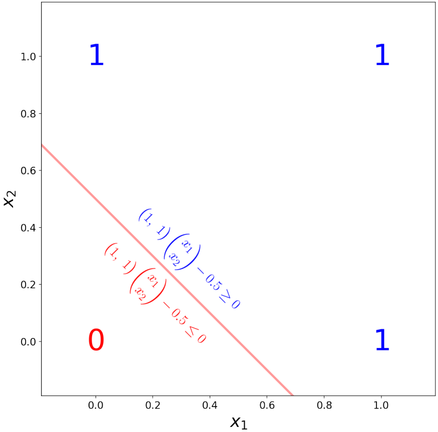
(x_1, x_2)	$\text{AND}(x_1, x_2)$
$(0,0)$	0
$(1,0)$	0
$(0,1)$	0
$(1,1)$	1

Predict $\hat{y} = 1$ if

$$x_1 + x_2 > 1.5$$

OR Problem

. . . is linearly separable, so perceptron works!



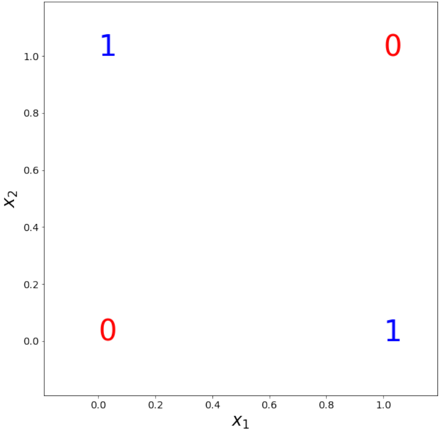
(x_1, x_2)	$\mathrm{OR}(x_1, x_2)$
$(0,0)$	0
$(1,0)$	1
$(0,1)$	1
$(1,1)$	1

Predict $\hat{y} = 1$ if

$$x_1 + x_2 > 0.5$$

XOR Problem

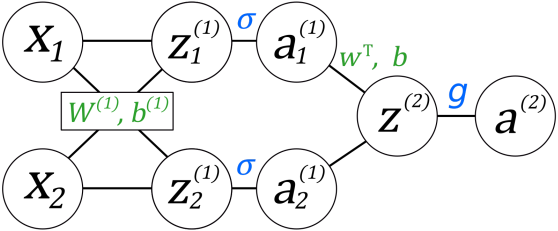
. . . is **NOT** linearly separable, so perceptron fails!



(x_1, x_2)	$\text{XOR}(x_1, x_2)$
$(0, 0)$	0
$(1, 0)$	1
$(0, 1)$	1
$(1, 1)$	0

No linear decision boundary! (Minksy and Papert, 1969).

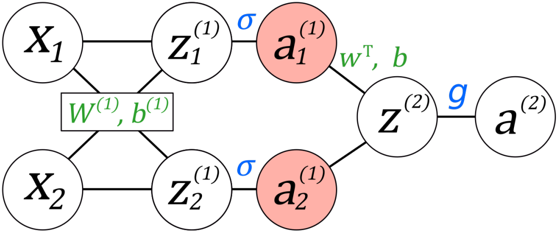
Shallow network for XOR



- ReLU activation $\sigma(z) = \max\{0, z\}$.
- Binary output $g(z) = \mathbb{1}(z^{(2)} > 0)$.
- Weights:

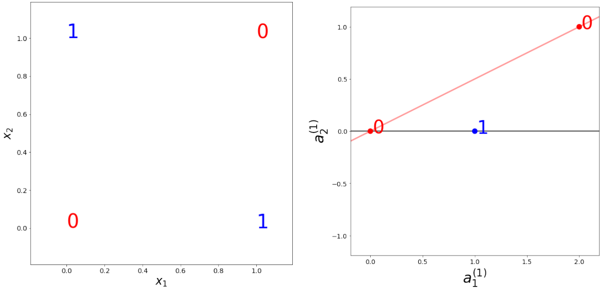
$$W^{(1)} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad b^{(1)} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}, \quad w = \begin{pmatrix} 1 \\ -2 \end{pmatrix}, \quad b = 0.$$

Shallow network for XOR



- With these weights, the **hidden layer** $a^{(1)}$ encodes a new representation of the data...

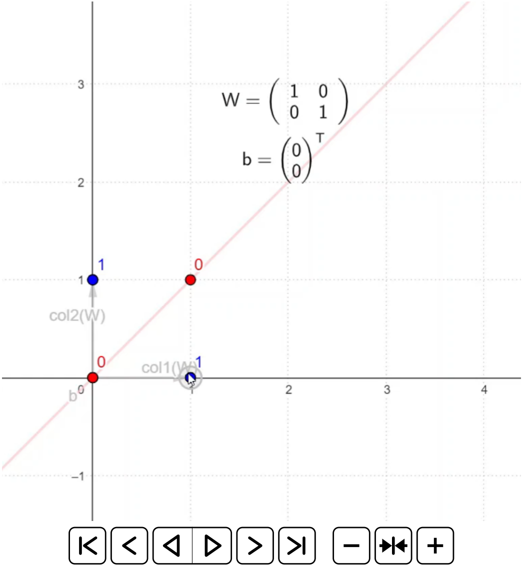
...solve XOR!



. . . in which the data is linearly separable! The weights $w = (1, -2)^T$ and $b = 0$ define a hyperplane: predict $\hat{y} = 1$ if

$$z^{(2)} = w^T a^{(1)} + b = a_1^{(1)} - 2a_2^{(1)} > 0.$$

Transformation of XOR problem



Universal Approximation

Theorem (Universal Approximation Theorem)

- *$\Phi_{(N,\sigma,g)}$ a neural network with $L \geq 2$ layers,*
- *Output $g(z) = z$ (identity),*
- *Finite feature space $\mathbb{R}^d$ and response space $\mathcal{Y}$,*
- *Weak conditions on activation σ*

Then, with enough hidden units, the network $\Phi_{(N,\sigma,g)}$ can approximate any Borel-measurable function $f : \mathbb{R}^d \to \mathcal{Y}$ with arbitrary precision [1].

Remark

Every continuous function $f : S \subset \mathbb{R}^d \to \mathbb{R}$ defined on compact set S is Borel-measureable.

Optimisation

Gradient Descent

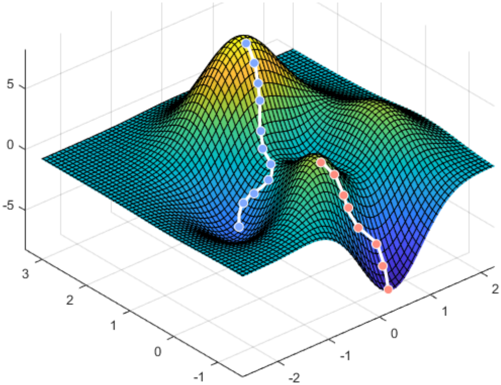
- Training data $s = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^{n}$.
- Parameters $\theta = \left(\left(W^{(l)}, b^{(l)} \right) \right)_{l=1}^{L}$.
- Minimise cost $C(\theta; s)$ over parameters θ,

$$C(f, s) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f, s^{(i)}).$$

Gradient descent

- Gradient descent - take small steps in direction of gradient, with learning rate $\eta > 0$:

$$\theta \leftarrow \theta - \eta \nabla_\theta C(\theta; s).$$



Backpropagation

- To calculate $\nabla_\theta C(\theta; s)$ we require

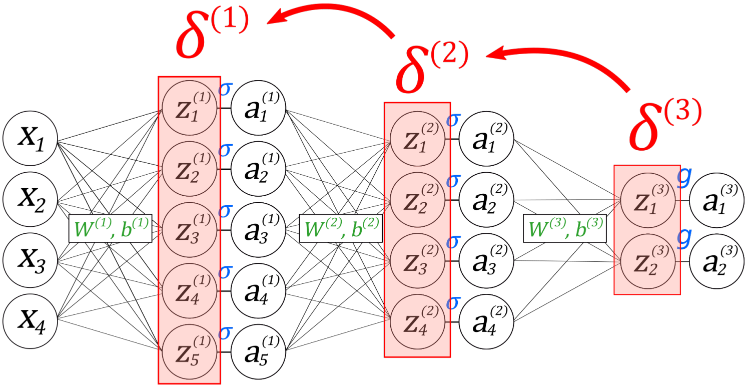
$$\frac{\partial C}{\partial w_{jk}^{(l)}} \quad \text{and} \quad \frac{\partial C}{\partial b_j^{(l)}}.$$

- To do this we define error vector $\delta^{(l)}$ for each layer with entries

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}}.$$

Errors

By a clever application of the chain rule, these errors can be propagated backwards.



Backpropagation: equations

This is done using the four equations of backpropagation.

$$\delta_j^{(L)} = \nabla_{a^{(L)}} \mathcal{L}(\theta) \;\odot\; g'(z^{(L)}), \tag{BP1}$$

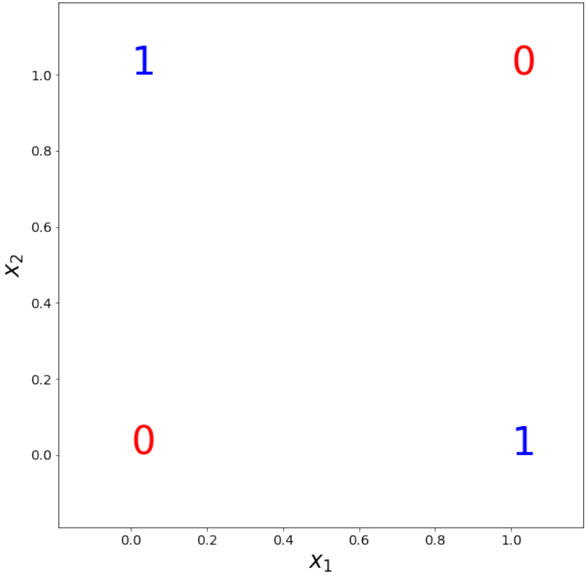
$$\delta^{(l)} = W^{(l+1)}{}^{T} \delta^{(l+1)} \odot \sigma'\left(z^{(l)}\right), \tag{BP2}$$

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \left[\delta^{(l)}\right]_j, \tag{BP3}$$

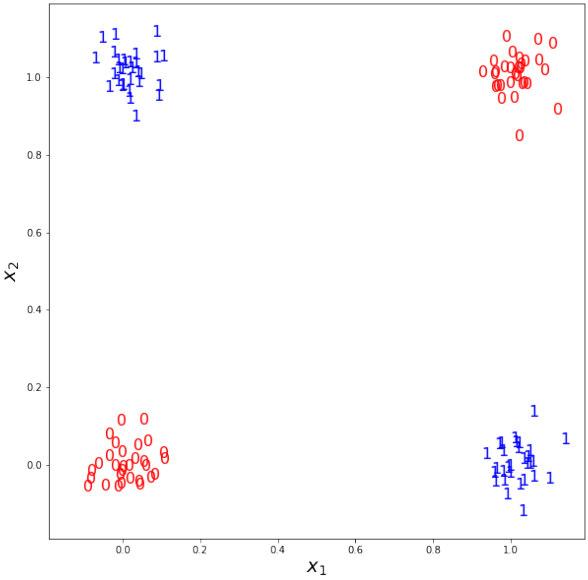
and $$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(l)}} = \left[\delta^{(l)} a^{(l-1)}{}^{T}\right]_{jk}. \tag{BP4}$$

Learning XOR

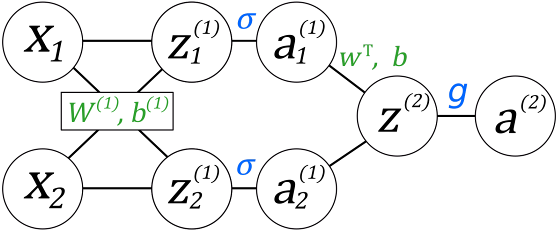
XOR without random noise



XOR with random noise



Network model

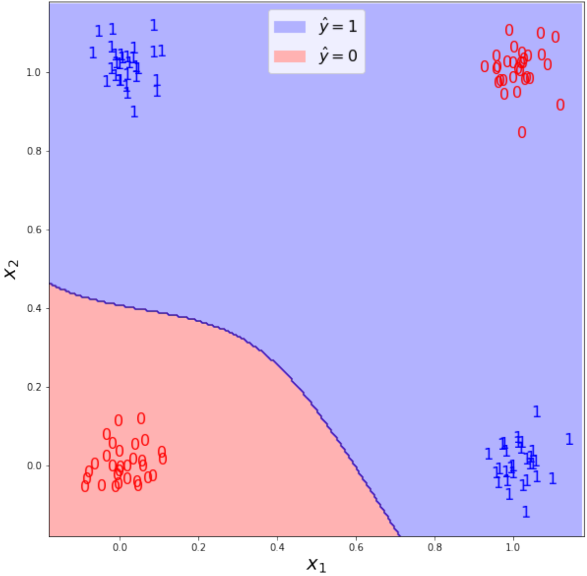


- Sigmoid activation σ and output g,
- Binary cross-entropy loss (from maximum-likelihood)

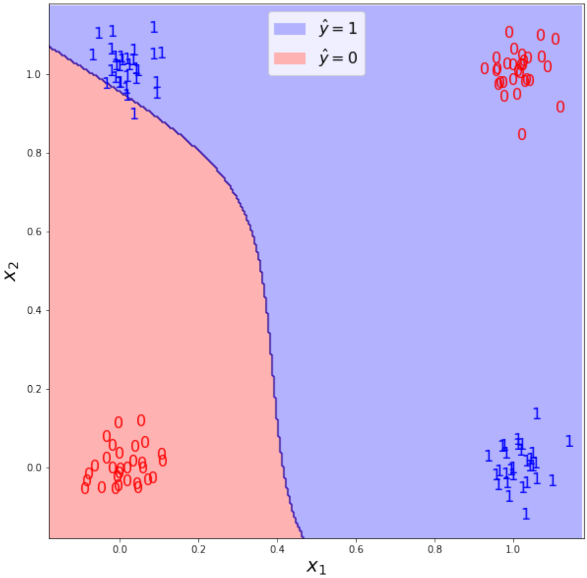
$$\mathcal{L}(\theta) = \zeta\left((1-2y)z^{(2)}\right),$$

- Stochastic gradient descent, $\eta = 0.1$, with backprop,
- Initial random weights and biases
  $W^{(1)} \in \mathbb{R}^{2\times 2},\ b^{(1)} \in \mathbb{R}^2,\ w \in \mathbb{R}^2,\ b \in \mathbb{R}.$

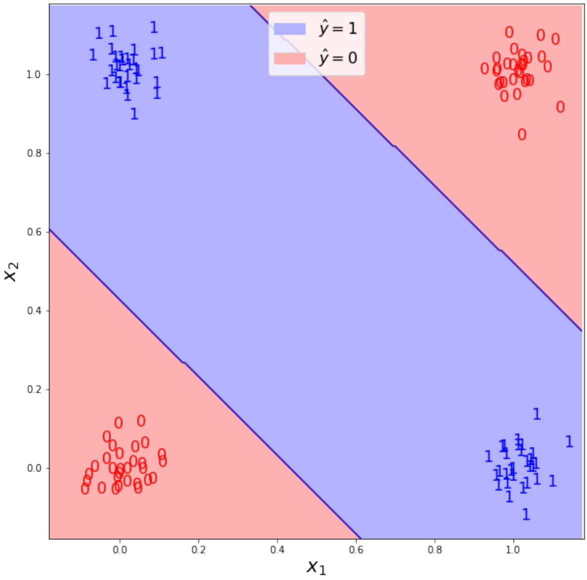
Bad Training



More Bad Training



Good Training

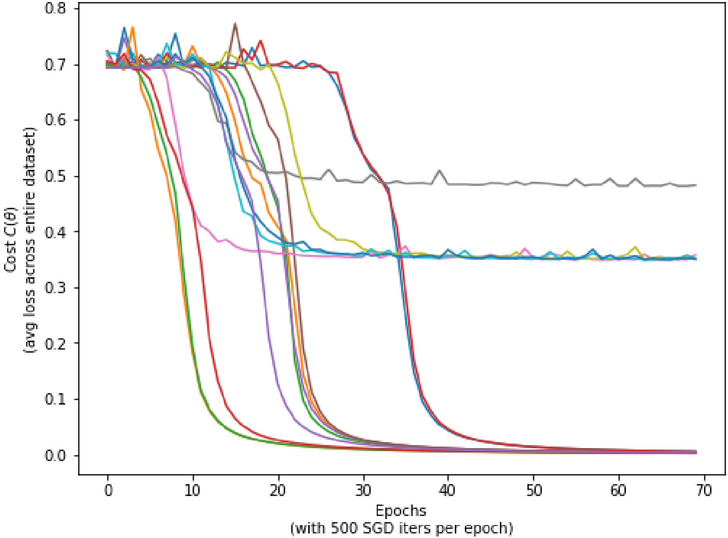


Weights

$$W^{(1)} = \begin{pmatrix} -5.63 & -5.48 \\ -7.27 & -7.26 \end{pmatrix}, \qquad b^{(1)} = \begin{pmatrix} 8.22 \\ 3.04 \end{pmatrix},$$

$$w = \begin{pmatrix} 12.07 \\ -12.43 \end{pmatrix}, \qquad \text{and } b = -5.71.$$

Evolution of cost with gradient descent iterations



Applications

Specialised Networks

We've studied the most general kind of neural network. In practice, specialised networks are used. These include:

- Convolutional Neural Networks (CNNs)
  Designed for data in grid form, e.g. medical diagnosis, facial recognition.
- Recurrent Neural Networks (RNNs)
  Designed for series data, e.g. weather prediction, natural language processing.

Implications

The rise of powerful AI will be either the best or the worst thing ever to happen to humanity. We do not yet know which. —Stephen Hawking

BBC News headlines from May 2023:

- "*'AI godfather' likens his emotions to atom bomb inventors*" (May 31st)
- "*New super-bug killing antibiotic discovered using AI*" (May 25th)
- "*BT to cut 55,000 jobs with up to a fifth replaced by AI*" (May 18th)
- "*Artificial intelligence could lead to extinction, experts warn*" (May 30th)

Further Reading

- **Mathematical Aspects of Deep Learning** (Grohs and Kutyniok, 2022) [1]
  Detailed introduction to mathematical analysis of neural networks.
- **Deep Learning** (Goodfellow et al., 2016) [2]
  Comprehensive textbook on deep learning.
- **Neural networks and deep learning** (Nielsen, 2015) [3]
  Excellent introduction to backpropagation.
- **An introduction to statistical learning** (Hastie et al., 2022) [4]
  Overview of all aspects of machine learning.

Bibliography

[1] Philipp Grohs and Gitta Kutyniok, eds. *Mathematical Aspects of Deep Learning*. Cambridge University Press, 2022. DOI: 10.1017/9781009025096.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[3] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[4] Gareth James et al. *An introduction to statistical learning: With applications in R*. Springer, 2022.

Thank you for listening!