QBUS6850 Week 1

Neural Networks 1
Introduction

Dr Stephen Tierney

The University of Sydney Business School

Recommended Reading

- Chapter 11.1-11.3, The Elements of Statistical Learning, Hastie et al.
- Chapter 5.1, Pattern Recognition and Machine Learning, Christopher Bishop
- Chapter 6.1-6.4, Deep Learning, Goodfellow et al.

Machine Learning Fundamentals and Terminology

Types of Learning

- **Supervised Learning**
- Semi-supervised learning
- **Unsupervised Learning**
- Instance-based Learning
- **Reinforcement Learning**

Supervised Learning

The goal of supervised learning is to learn a mapping from an input $\mathbf{x}$ to an output $\mathbf{y}$.

In order to learn the relationship between $\mathbf{x}$ and $\mathbf{y}$ we require a set of labelled training pairs $(\mathbf{x}, \mathbf{y})$.

A "supervisor" has labelled the data i.e. they are prescribing which relationship to learn.

Examples:

- Regression
- Classification

Unsupervised Learning

In unsupervised learning there are no inputs or outputs. Rather a set of unlabelled data is provided.

This means there is no prescribed relationship to learn and the best we can do is to uncover hidden relationships in the data.

Examples:

- Clustering
- Embedding
- Dimension Reduction

Reinforcement Learning

In the reinforcement learning setting there is no pool of existing data to learn from.

Instead there is an agent, which we design, an environment and a set of possible actions to take.

These actions generate rewards and the goal is for the agent to maximise the total rewards received.

Inputs or Features

An object is represented by a feature vector $\mathbf{x}$.

Denote by $\mathbf{x} = (x_1, x_2, \ldots, x_d)^T$ where each component x_i is a feature value for the object.

Don't confuse x_i and $\mathbf{x}_i$. The former is an element of a feature vector, while the later is the feature vector for object i.

Each component x_i may be a quantitative value from $\mathbb{R}$ (the set of all real numbers) or one from a finite set of categorical values.

Outputs or Targets

The output generated for each object are denoted by y and are called targets.

For a given object i the corresponding target is denoted by y_i.

Each component y_i may be a quantitative value from $\mathbb{R}$ (the set of all real numbers) or one from a finite set of categorical values.

Dataset

A pair of observed $(\mathbf{x}, y)$ is called a training/testing datum.

In the unsupervised learning case, there is no target y.

All the available training data are collected together into a set denoted $\mathcal{D}$ with or without target observation $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ or $\{\mathbf{x}_n\}_{n=1}^N$.

Learner or Model

A learner or predictive model is the system that we design to predict the outcome for new unseen objects or characterise them if without targets.

A good learner is one that accurately predicts such an outcome or make a right characterisation.

Data Representation

When we have a data set with N observations $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, we will organise them into a matrix of size $N \times d$ such that each row corresponds to an observation (or a case).

If we have the target/output for N observation $\{y_1, y_2, \ldots, y_N\}$, we also organise them into a column (simulating a row corresponding to a case or an observation), which we denote as

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,d} \end{pmatrix} \in \mathbb{R}^{N \times d}, \ \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \in \mathbb{R}^N,$$

Neural Networks

The Plan

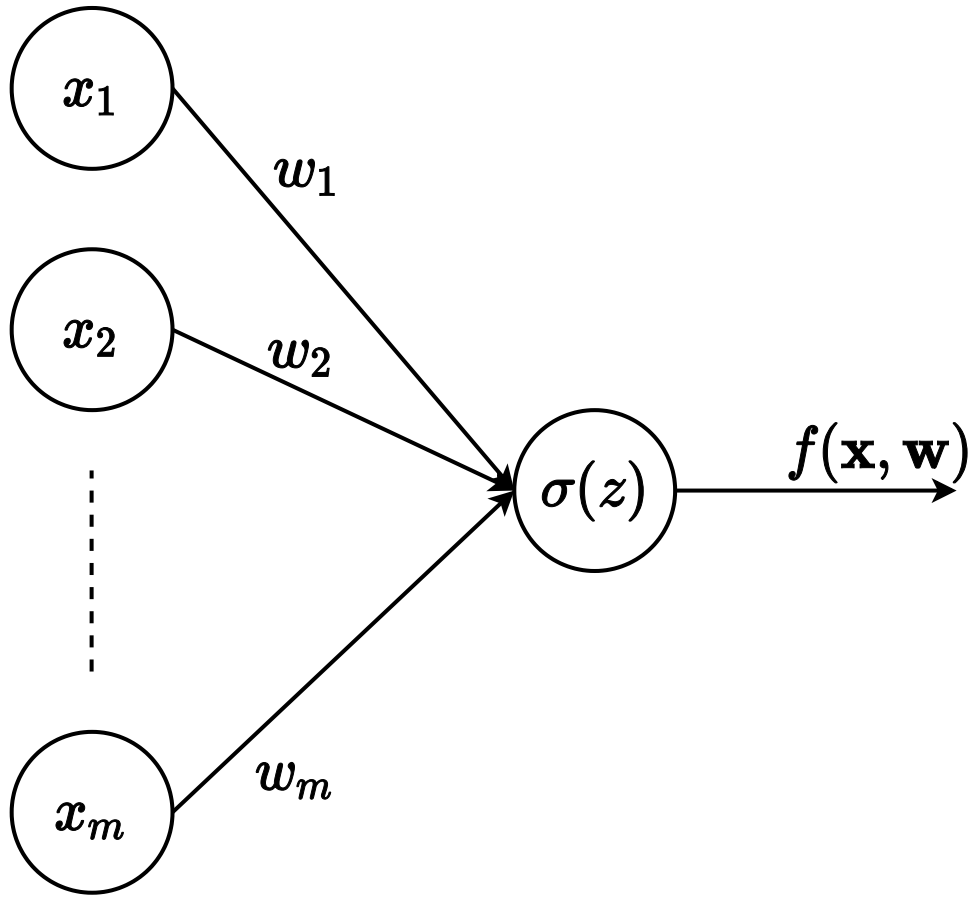
- **Week 1: Neural Network Introduction**
- Week 2: Training a Neural Network
- Week 3: Refining the Training Process
- Week 4: Network Design

Neural Networks

An Artificial Neural Network (ANN) is a collection of **artificial neurons**, which are arranged to perform a data processing task.

ANNs can be built to solve many problems such as regression, classification, representation learning and synthetic data generation.

Artificial Neurons



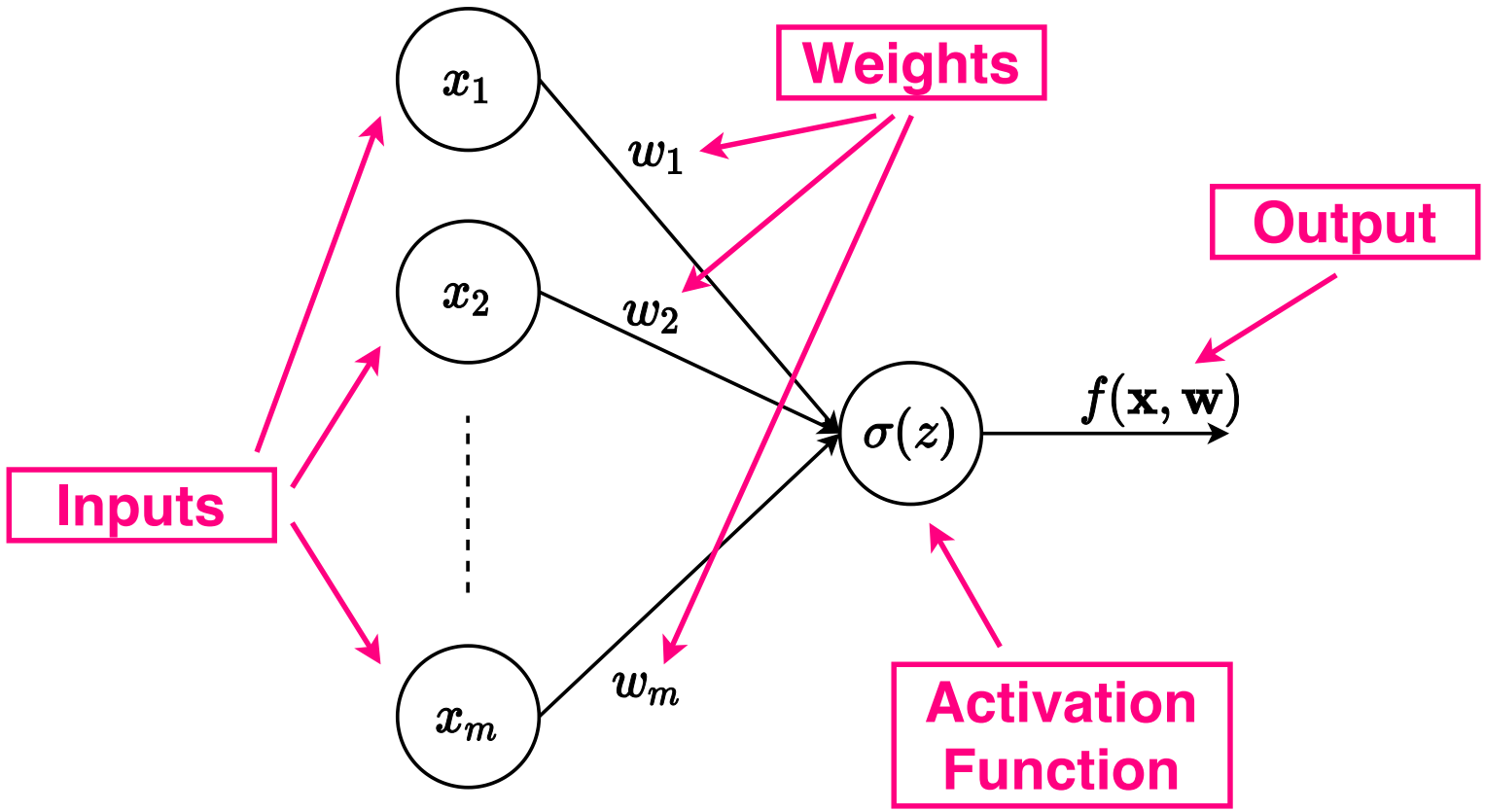
Artificial Neurons

An artificial neuron takes some inputs x_i with corresponding weights w_i, sums them and then applies an activation function σ.

We can write this as

$$\begin{aligned} f(\mathbf{x}, \mathbf{w}) &= \sigma\left(\sum_{i=1}^{m} x_i w_i\right) \\ &= \sigma(z) \end{aligned}$$

Artificial Neurons



Artificial Neurons - Exercise 1 (My Turn)

Using the activation function

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

and weights and data

$$\mathbf{w} = [0.2, -0.7, 0.5]^T$$
$$\mathbf{x} = [1.2, 3.4, -0.5]$$

Calculate the output $f(\mathbf{x}, \mathbf{w})$

$$f(\mathbf{x}, \mathbf{w}) =$$

Artificial Neurons - Exercises

You can check your answers the exercises on Ed!

https://edstem.org/au/courses/6467/lessons/13785/slides/99592

Artificial Neurons - Exercise 2 (Our Turn)

Using the same weights as before, calculate the output with data

$$\mathbf{x} = [1.6, -1.1, 0.5]$$

The answer is

$$f(\mathbf{x}, \mathbf{w}) =$$

Artificial Neurons - Exercise 3 (Your Turn)

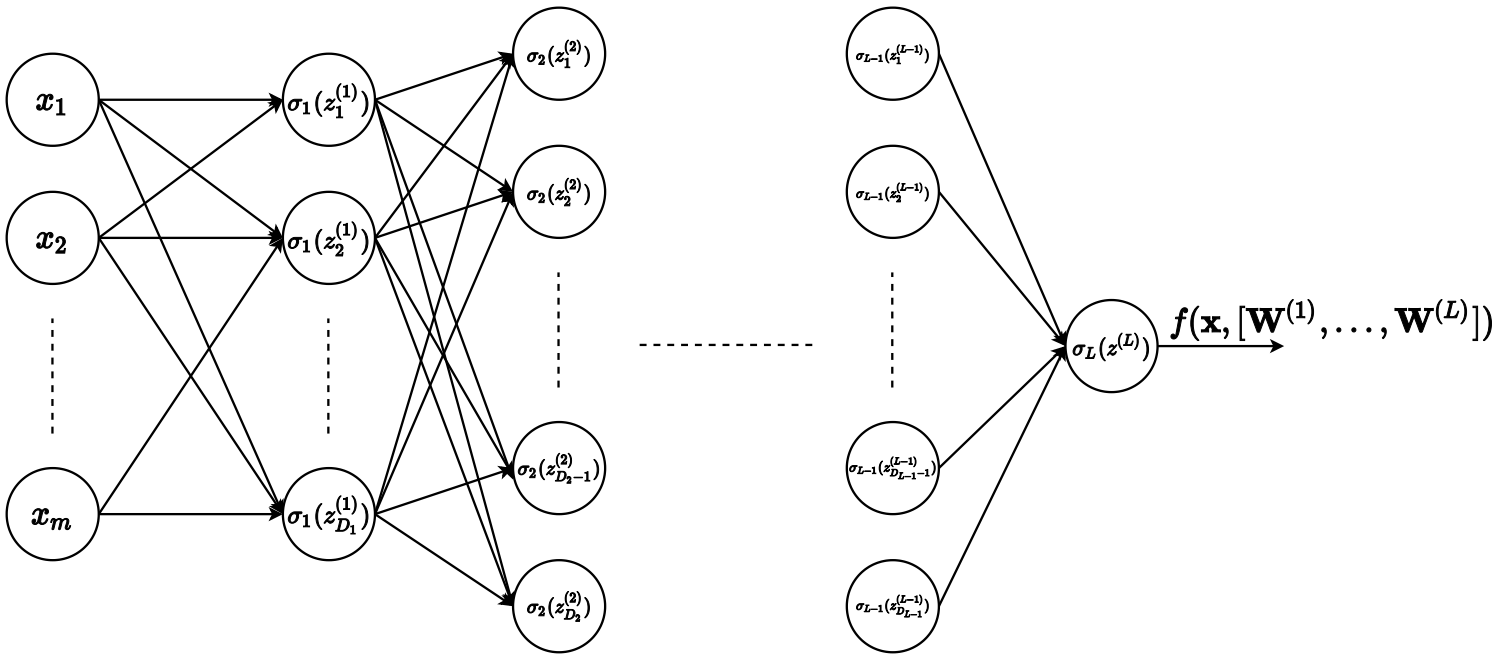
Using the following weights and data, calculate the output

$$\mathbf{w} = [-0.8, -0.1, 1.5]^T$$
$$\mathbf{x} = [2.1, 4.3, 0.5]$$

The answer is

$$f(\mathbf{x}, \mathbf{w}) =$$

Artificial Neural Networks



Artificial Neural Networks

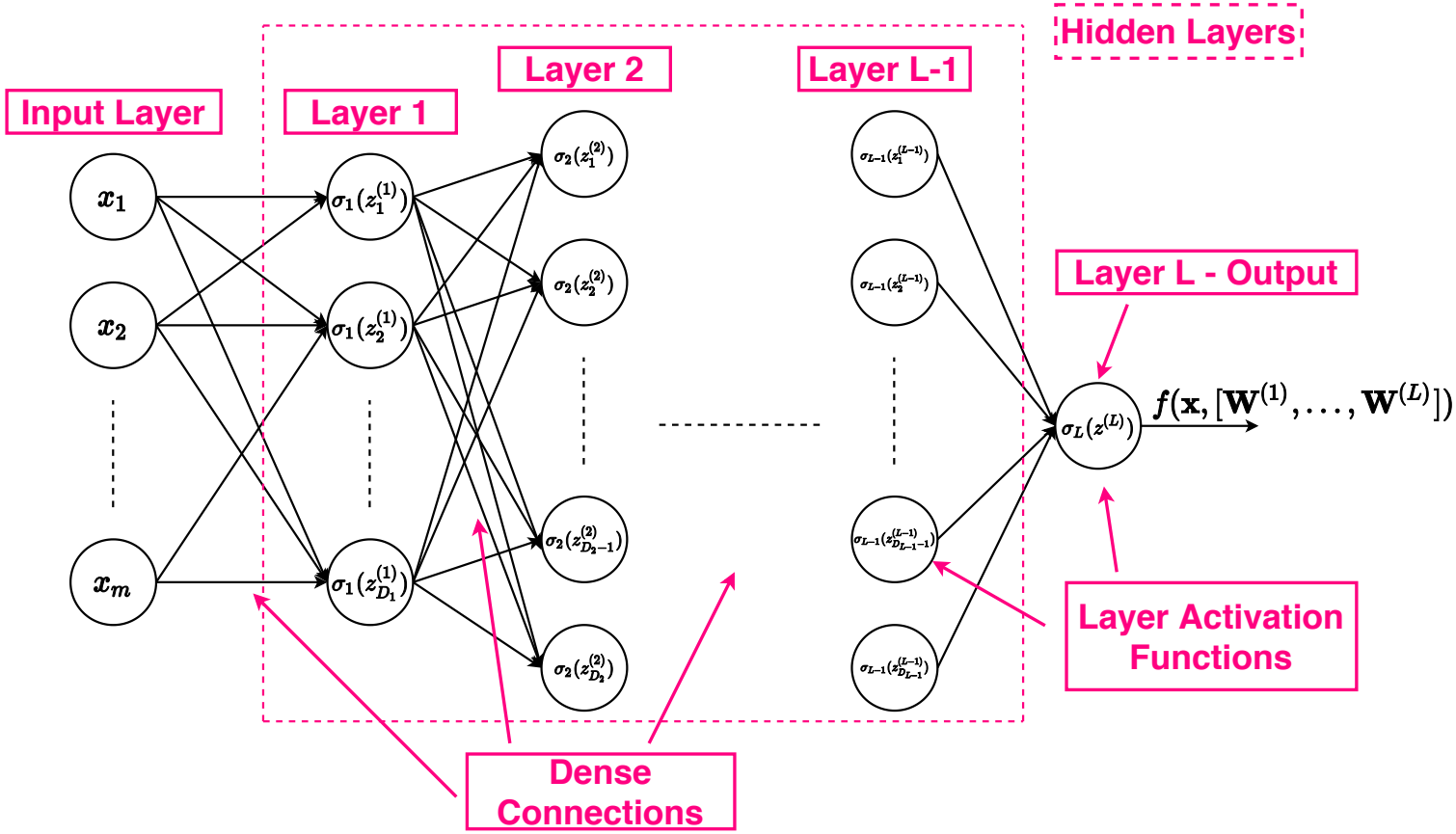
Networks are a collection of neurons arranged in L layers.

The internal layers are called **hidden layers**.

A typical network has the following characteristics:

- each layer has D_l neurons, which must be **greater than zero**
- every neuron in a layer uses the **same activation** function σ_l
- neurons from each layer are **densely** connected to the next
- each connection between neurons has a corresponding **weight** $w^{(l)}_{D_{l-1},D_l}$

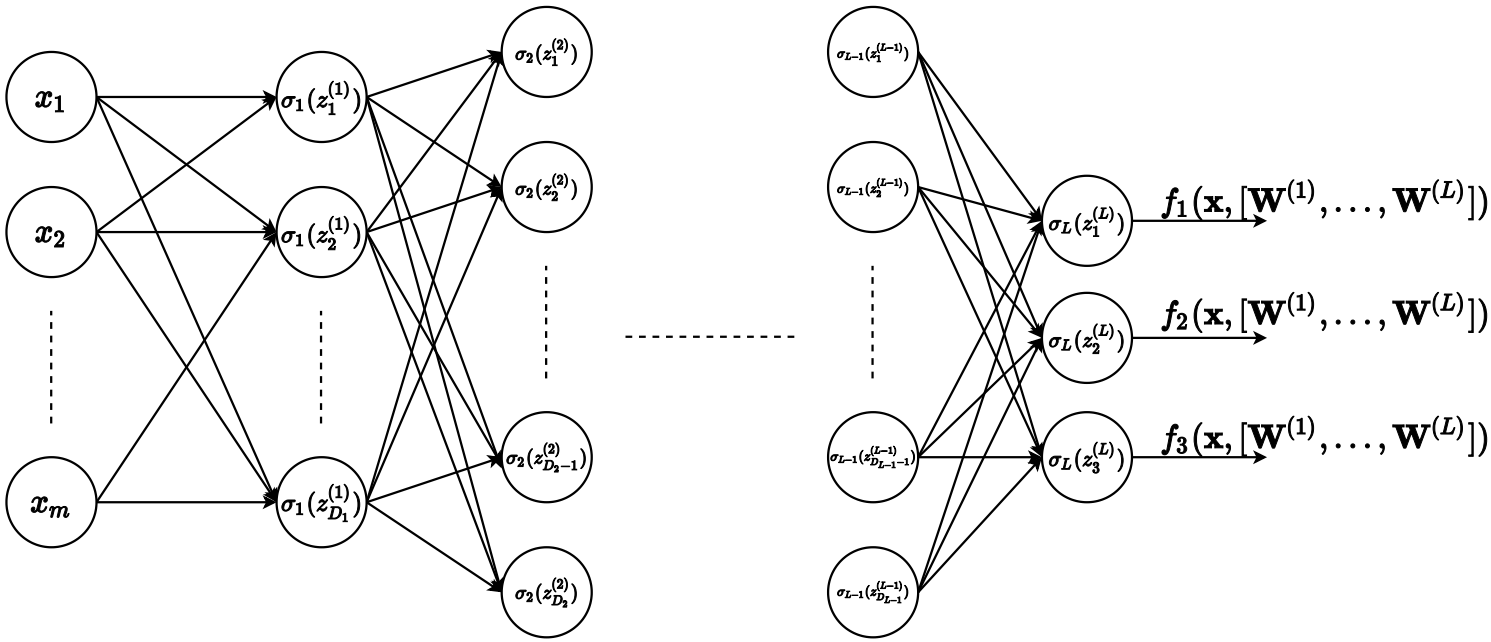
Artificial Neural Networks



Artificial Neural Networks

Optionally the network may have multiple outputs. In the example below the network produces three outputs simultaneously.

However for the remainder of this lecture we will focus on the single output case.



Generating a Prediction - Forward Propagation

To generate a prediction $\hat{y}$ from a network we input an observation's feature vector $\mathbf{x}$ into the network.

Mathematically we can write it as

$$\begin{aligned}\hat{y} &= f(\mathbf{x}, [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}]) \\ &= \sigma_L\left(\mathbf{z}^{(L)}\right)\end{aligned}$$

Where $\mathbf{z}^{(L)}$ is the weighted activations from layer $L-1$.

Computing this recursive function is called **Forward Propagation**.

Forward Propagation

We can simplify using matrix notation

$$\begin{aligned} f(\mathbf{x}, [\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}]) &= \mathbf{a}^{(L)} \\ &= \sigma_L\left(\mathbf{z}^{(L)}\right) \\ &= \sigma_L\left(\mathbf{a}^{(L-1)}\mathbf{W}^{(L)}\right) \end{aligned}$$

where $\mathbf{W}^{(L)}$ is a $D_{L-1} \times D_L$ matrix of weights

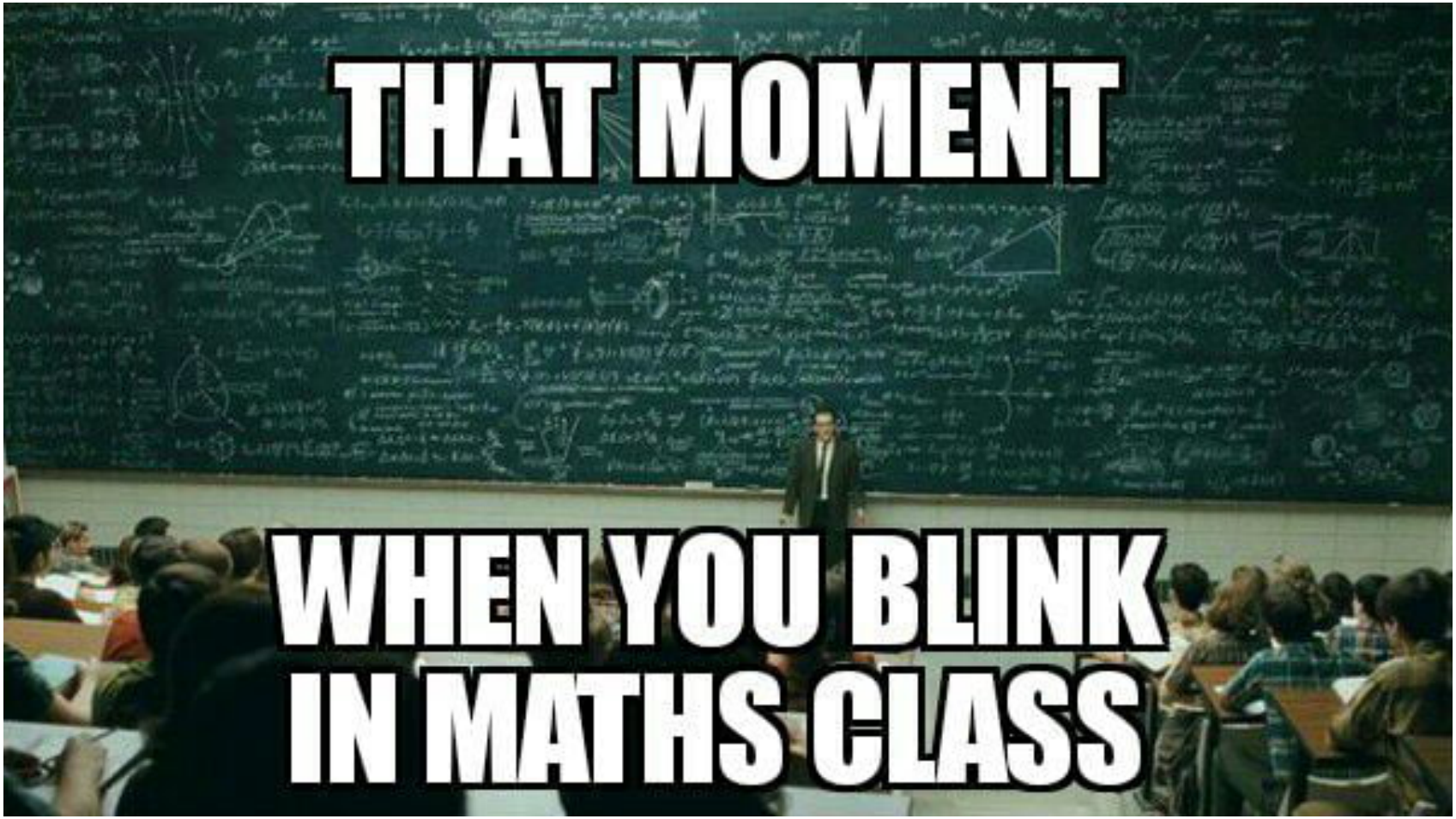
$$\mathbf{W}^{(L)} = \begin{bmatrix} w_{1,1}^{(L)} & w_{1,2}^{(L)} & \cdots & w_{1,D_L}^{(L)} \\ \vdots & \ddots & & \vdots \\ w_{D_{L-1},1}^{(L)} & & \cdots & w_{D_L-1,D_L}^{(L)} \end{bmatrix},$$

$\mathbf{a}^{(L-1)}$ is the D_{L-1} row vector of activation values

$$\mathbf{a}^{(L-1)} = \sigma_{L-1}(\mathbf{z}^{(L-1)}),$$

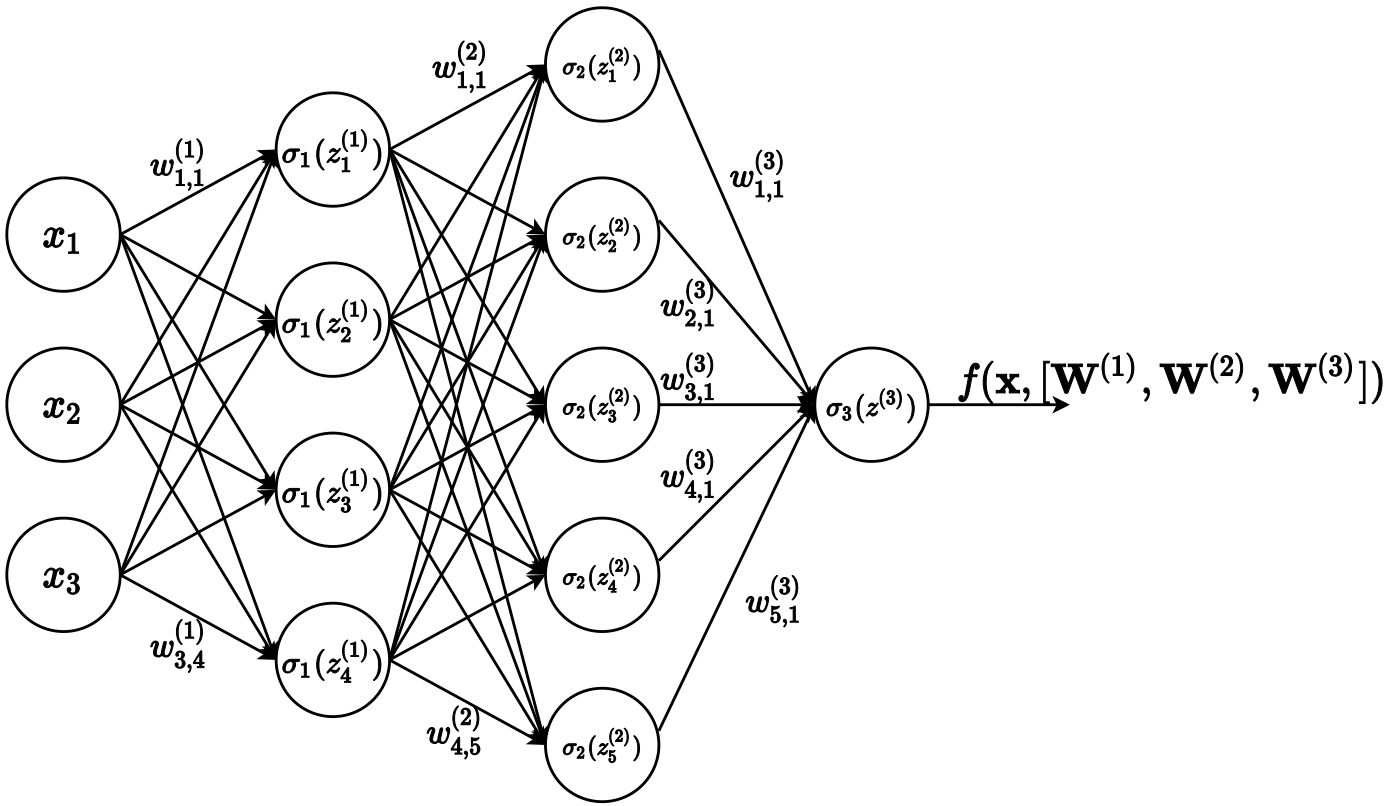
and $\mathbf{z}^{(L)}$ is the D_L row vector of weighted activations at layer L.

Forward Propagation



Forward Propagation - Example

Suppose we have the following network architecture



Forward Propagation - Example

Calculate the value of f for the two data points in $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} 1.2 & 3.4 & -0.5 \\ 5.2 & 0.3 & -0.2 \end{bmatrix},$$

using the following activation function at all hidden layers

$$\sigma(z)_H = \begin{cases} z, \text{ for } z > 0 \\ 0, \text{ for } z \leq 0 \end{cases},$$

and the sigmoid activation function at the output layer

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

Forward Propagation - Example

and the weight matrices

$$\mathbf{W}^{(1)} = \begin{bmatrix} 1.8 & 0.4 & 1. & 2.2 \\ 1.9 & -1. & 1. & -0.2 \\ -0.1 & 0.4 & 0.1 & 1.5 \end{bmatrix},$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} 0.8 & 0.1 & 0.4 & 0.3 & 1.5 \\ -0.2 & 0.3 & -0.9 & -2.6 & 0.7 \\ 0.9 & -0.7 & 2.3 & -1.5 & 0. \\ -0.2 & 1.5 & 1.5 & 0.2 & 0.4 \end{bmatrix},$$

$$\mathbf{w}^{(3)} = \begin{bmatrix} -0.9 \\ -2. \\ -0.3 \\ 0.2 \\ 1.2 \end{bmatrix},$$

Forward Propagation - Example

$$\mathbf{Z}^{(1)} = \mathbf{X}\mathbf{W}^{(1)}$$
$$= \begin{bmatrix} 8.67 & -3.12 & 4.55 & 1.21 \\ 9.95 & 1.7 & 5.48 & 11.08 \end{bmatrix}$$

Apply σ_H element-wise to $\mathbf{Z}^{(1)}$

$$\mathbf{A}^{(1)} = \sigma_H(\mathbf{Z}^{(1)})$$
$$= \begin{bmatrix} 8.67 & 0. & 4.55 & 1.21 \\ 9.95 & 1.7 & 5.48 & 11.08 \end{bmatrix}$$

Forward Propagation - Example

$$\mathbf{Z}^{(2)} = \mathbf{A}^{(1)}\mathbf{W}^{(2)}$$

$$= \begin{bmatrix} 10.789 & -0.503 & 15.748 & -3.982 & 13.489 \\ 10.336 & 14.289 & 31.674 & -7.439 & 20.547 \end{bmatrix}$$

$$\mathbf{A}^{(2)} = \sigma_H(\mathbf{Z}^{(2)})$$

$$= \begin{bmatrix} 10.789 & 0. & 15.748 & 0. & 13.489 \\ 10.336 & 14.289 & 31.674 & 0. & 20.547 \end{bmatrix}$$

Forward Propagation - Example

$$\mathbf{Z}^{(3)} = \mathbf{A}^{(2)}\mathbf{w}^{(3)}$$
$$= \begin{bmatrix} 1.7523 \\ -22.7262 \end{bmatrix}$$

$$\mathbf{A}^{(3)} = \sigma_O(\mathbf{Z}^{(3)})$$
$$= \begin{bmatrix} 0.852242664 \\ 1.34938769e-10 \end{bmatrix}$$
$$\approx \begin{bmatrix} 0.85 \\ 0 \end{bmatrix}$$

Forward Propagation - Example

Therefore the output (predictions) for the two data points are

$$f(\mathbf{X}, [\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{w}^{(3)}]) = \begin{bmatrix} 0.85 \\ 0 \end{bmatrix}$$

Suppose this network was built to be a binary classifier.

What class would each data point be assigned to?

Bias Units

In linear regression models, e.g.

$$f(\mathbf{x}) = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

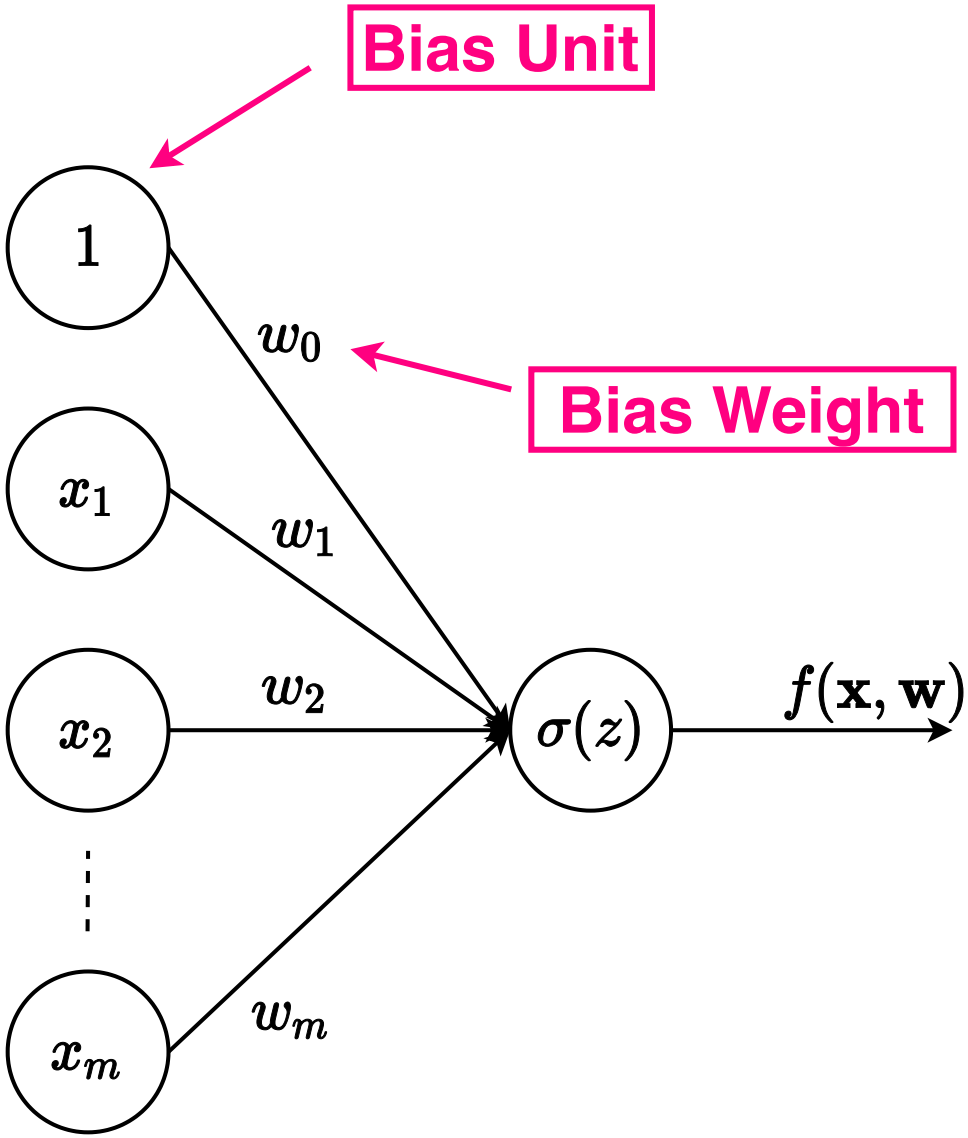
the intercept term β_0 plays an important role by shifting the mean of the prediction.

Notice that we can re-write this model as

$$f(\mathbf{x}) = 1 \times \beta_0 + \sum_{i=1}^{m} \beta_i x_i$$

In a neural network, the equivalent of an intercept is called a **bias unit**.

Bias Units


Bias Unit
1
w_0
Bias Weight
x_1
w_1
x_2
w_2
$\sigma(z)$
$f(\mathbf{x}, \mathbf{w})$
x_m
w_m

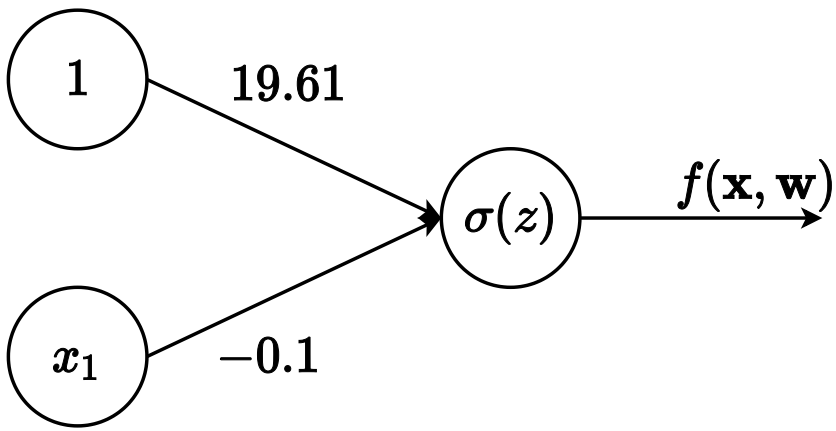
Bias Units

Suppose we have trained the following simple linear regression model

$$f(\mathbf{x}) = 19.61 - 0.1x_1$$

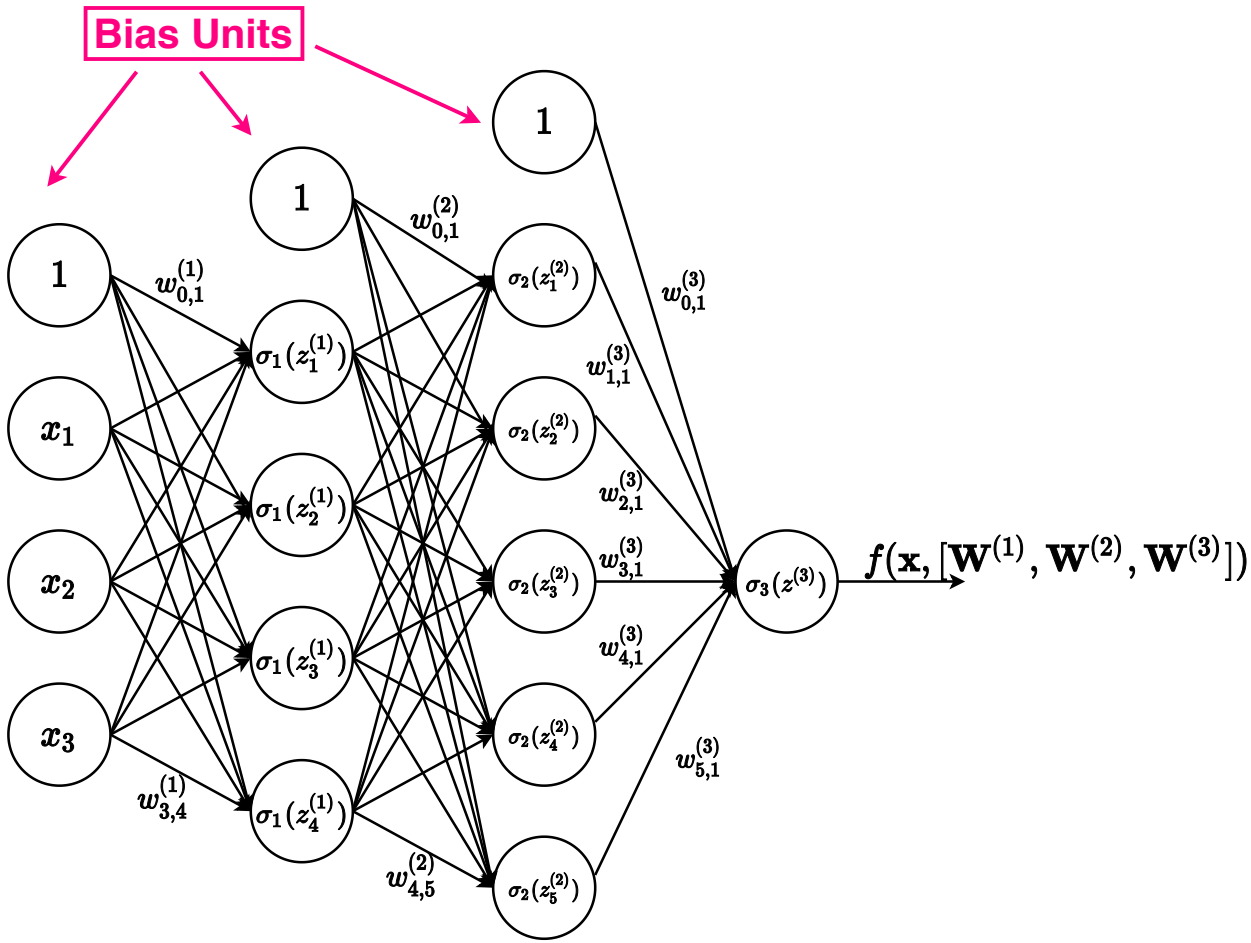
To replicate the regression model with a neural network we only need a single neuron with the identity activation function

$$\sigma(z) = z$$



Bias Units

We can add bias units at any layer in the network[1]. Notice that the bias units are not connected to the previous layer, they are **disconnected**.



[1] even the output layer, but this is not shown in the diagram

Bias Units

For a layer with densely connected Bias Units the forward propagation formula changes

$$\mathbf{z}_i^{(l)} = \mathbf{a}_i^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}$$

where $\mathbf{b}$ is a row vector of bias weights.

Therefore $\mathbf{z}_i^{(l)}$ is a row vector.

The Greatest Hits of Activation Functions

Common Hidden Layer Activations

Identity

$$\sigma(z) = z$$

ReLU (Rectified Linear Unit)

$$\sigma(z) = \begin{cases} z, & \text{for } z > 0 \\ 0, & \text{for } z \leq 0 \end{cases}$$

The Greatest Hits of Activation Functions

Common Output Layer Activations (Classification)

Sigmoid (Logistic)

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

TanH (Hyperbolic Tangent Function)

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

The Greatest Hits of Activation Functions

Common Output Layer Activations (Multi-Class Classification)

Softmax

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{d=1}^{D} e^{z_d}}$$

The Softmax function returns a vector, not a scalar. Each element is the exponent of the original element divided by the sum of exponents of every element.

Universal Function Approximation

It has been proven that it is possible to approximate many functions using a neural network.

Interestingly the network does not need to be particularly complicated to do so.

The usual theorem states that it is sufficient to use a single hidden layer!

However the theorem does not tell us how to design the network.

Universal Function Approximation

This paper rigorously establishes that standard multilayer feedforward networks **with as few as one hidden layer** *using arbitrary squashing functions are capable of* **approximating any Borel measurable function** *from one finite dimensional space to another to any desired degree of accuracy,* **provided sufficiently many hidden units are available**. *In this sense, multilayer feedforward networks are a class of universal approximators.*

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

Wrapping Up

Neural Networks Summary

The design of a Neural Network dictates the computation or function that it performs.

For each task that we want to solve we must carefully design the network.

It is theoretically possible to design a network to approximate any function, however determining the design or estimating the weights might be **extremely difficult**.

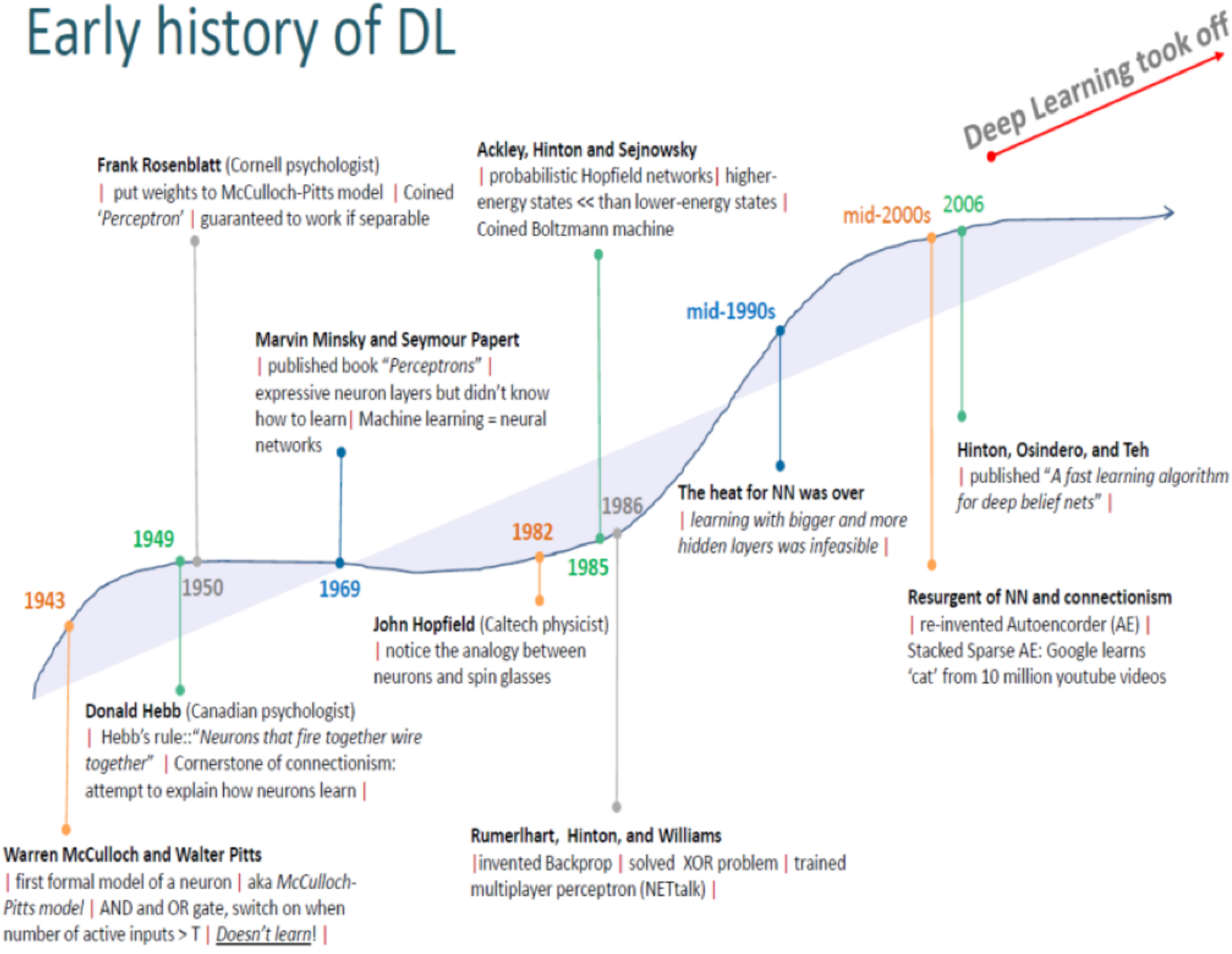
History

Artificial Neural Networks (ANNs) were originally researched in order to simulate how human and other animal brains worked.

Research on ANNs has been ongoing since the mid 20th century.

The practical application of ANNs was limited by computing power, which increased dramatically since the 1990s.

History


Early history of DL
Deep Learning took off
Frank Rosenblatt (Cornell psychologist)
| put weights to McCulloch-Pitts model | Coined 'Perceptron' | guaranteed to work if separable
Ackley, Hinton and Sejnowsky
| probabilistic Hopfield networks| higher-energy states << than lower-energy states | Coined Boltzmann machine
mid-2000s
2006
mid-1990s
Marvin Minsky and Seymour Papert
| published book "Perceptrons" | expressive neuron layers but didn't know how to learn| Machine learning = neural networks
Hinton, Osindero, and Teh
| published "A fast learning algorithm for deep belief nets" |
The heat for NN was over
| learning with bigger and more hidden layers was infeasible |
1986
1949
1982
1985
1943
1950
1969
Resurgent of NN and connectionism
| re-invented Autoencorder (AE) | Stacked Sparse AE: Google learns 'cat' from 10 million youtube videos
John Hopfield (Caltech physicist)
| notice the analogy between neurons and spin glasses
Donald Hebb (Canadian psychologist)
| Hebb's rule::"Neurons that fire together wire together" | Cornerstone of connectionism: attempt to explain how neurons learn |
Rumerlhart, Hinton, and Williams
|invented Backprop | solved XOR problem | trained multiplayer perceptron (NETtalk) |
Warren McCulloch and Walter Pitts
| first formal model of a neuron | aka McCulloch-Pitts model | AND and OR gate, switch on when number of active inputs > T | Doesn't learn! |

History

The use of and development of ANNs has exploded in the past 10 years.

ANNs in the form of "Deep Neural Networks" are now considered the best performing models in many application areas.

However there is a significant price to pay in terms of development and training time as well as the associated costs of doing so.

What's Next?

Next week we will learn how to estimate the weights using training data!

- Week 1: Neural Network Introduction
- **Week 2: Training a Neural Network**
- Week 3: Refining the Training Process
- Week 4: Network Design