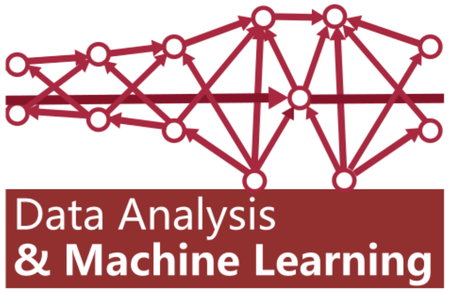


Data Analysis and Machine Learning 4 (DAML)

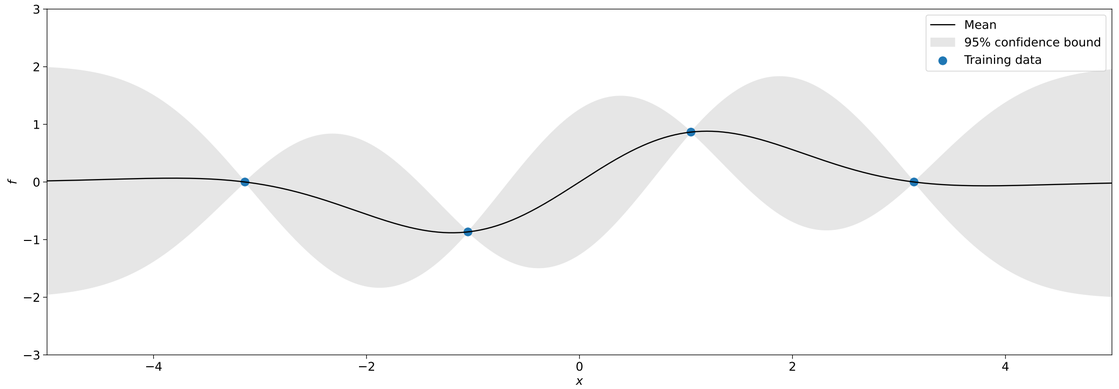
Week 10: Deep Neural Networks

Elliot J. Crowley, 25th March 2024

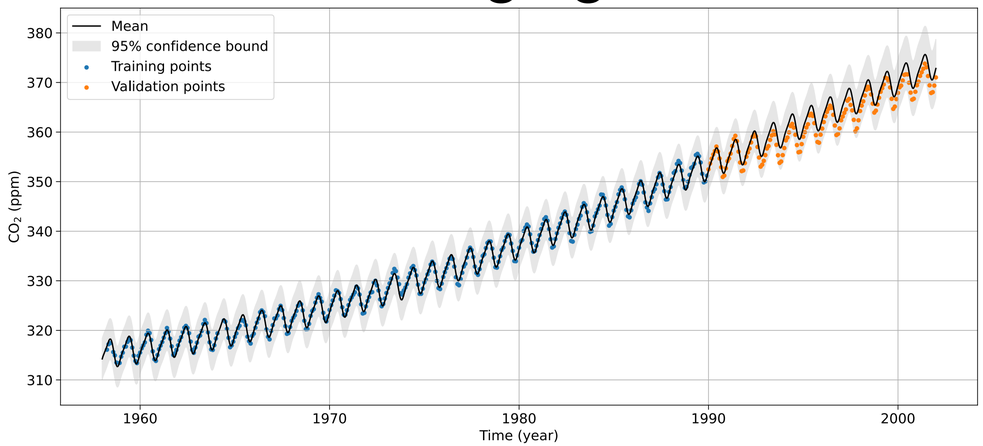
THE UNIVERSITY of EDINBURGH

Recap

- We looked at Gaussian processes for regression

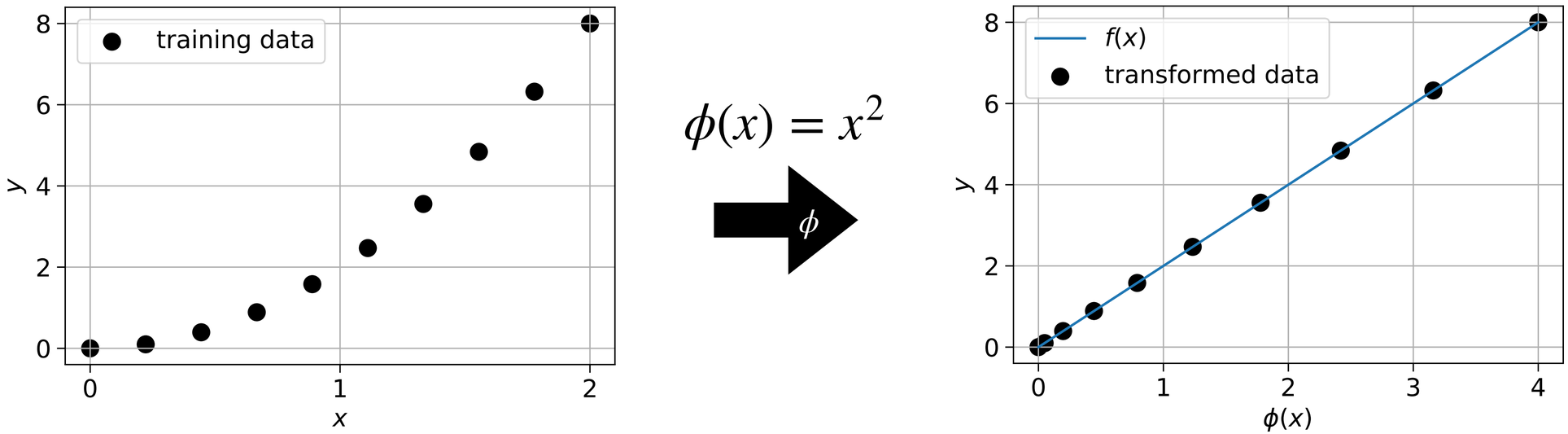


- We saw how changing the kernel affected the GP prior and posterior



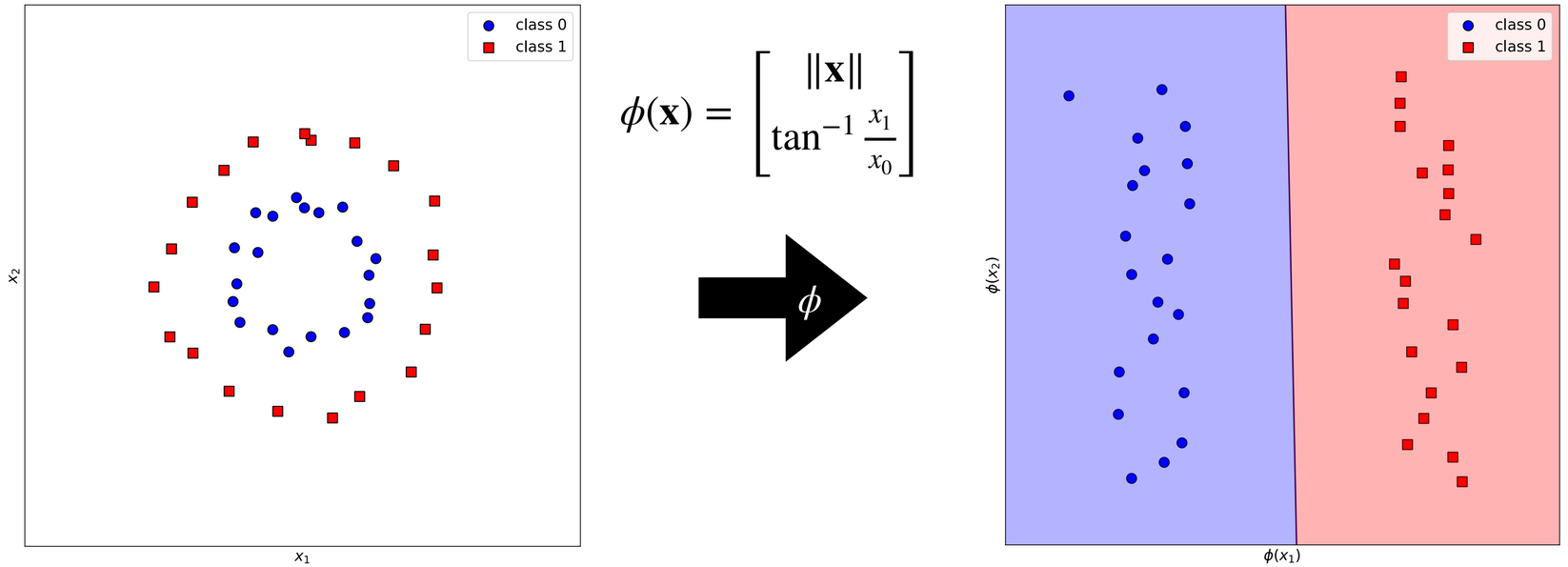
Linear regression

- Given training data $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$ ($\mathbf{x} \in \mathbb{R}^D$, $y \in \mathbb{R}$) we can learn a model:
  - $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ s.t. $y^{(n)} \approx f(\mathbf{x}^{(n)}) \forall n$
- We want ϕ to map the data to a space where we can fit a hyperplane to it



(Binary) linear classifiers

- Given training data $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$ ($\mathbf{x} \in \mathbb{R}^D$, $y \in \{0,1\}$) we can learn a model:
  - $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ s.t. the hyperplane $f(\mathbf{x}) = 0$ separates the classes
- We want ϕ to map the data to a space where classes can be separated by a hyperplane



Multi-dimensional output

- What if we want to perform multi-class classification or regress to a multi-dimensional output $f(\mathbf{x}) \in \mathbb{R}^K$?

  $f(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x}) + b$ with $\mathbf{w} \in \mathbb{R}^Z$ and $b \in \mathbb{R}$

  becomes

  $f(\mathbf{x}) = \mathbf{W}\phi(\mathbf{x}) + \mathbf{b}$ with with $\mathbf{W} \in \mathbb{R}^{Z \times K}$ and $\mathbf{b} \in \mathbb{R}^K$

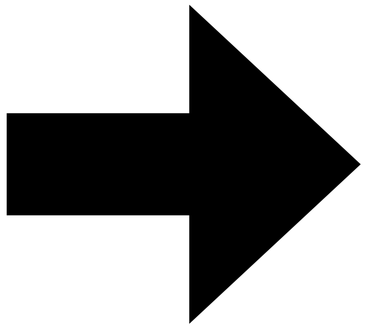
- We will assume this is the default output from now on as it is more general

Feature learning

- Trying to design ϕ for a new problem can be tedious or impossible!
- What if we could learn ϕ directly from our training data?
- This is what **deep learning** entails. It's **feature learning!**
- We represent ϕ as a parameterised function $\phi_{\boldsymbol{\theta}_f}(\mathbf{x})$ and learn $\boldsymbol{\theta}_f$ jointly with $\mathbf{W}$ and $\mathbf{b}$

$$f(\mathbf{x}) = \mathbf{W}\phi(\mathbf{x}) + \mathbf{b}$$

$$\underset{\mathbf{W},\mathbf{b}}{\text{minimise}}\, L$$



$$f(\mathbf{x}) = \mathbf{W}\phi_{\boldsymbol{\theta}_f}(\mathbf{x}) + \mathbf{b}$$

$$\underset{\boldsymbol{\theta}_f,\mathbf{W},\mathbf{b}}{\text{minimise}}\, L$$

Deep neural networks (DNNs)

- These are non-linear models (traditionally) consisting of $\mathscr{L}$ functional layers

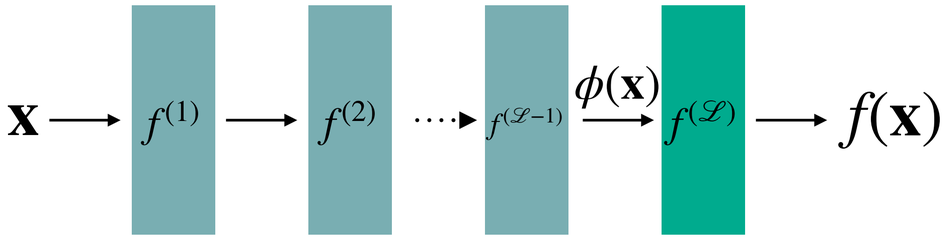
$$f(\mathbf{x}) = f^{(\mathscr{L})} \circ f^{(\mathscr{L}-1)} \circ \ldots \circ f^{(2)} \circ f^{(1)}(\mathbf{x})$$

- The first $\mathscr{L} - 1$ layers form a **learnable** feature map $\phi(\mathbf{x})$. These are known as **hidden layers**

$$\phi(\mathbf{x}) = f^{(\mathscr{L}-1)} \ldots f^{(2)} f^{(1)}(\mathbf{x})$$

- The last layer is (often) a linear transformation of the features

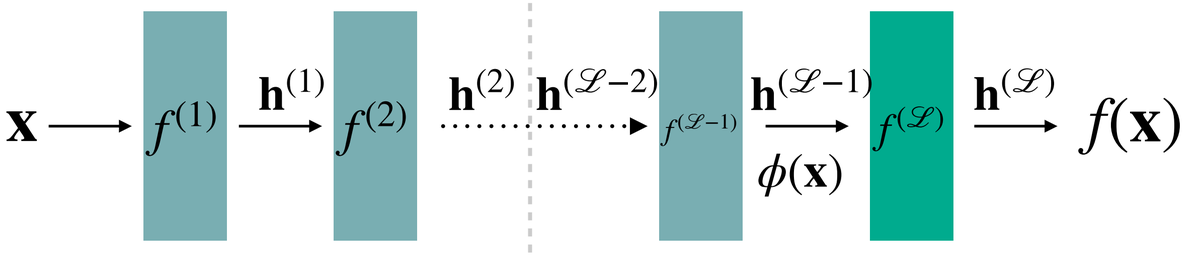
$$f(\mathbf{x}) = f^{(\mathscr{L})}(\phi(\mathbf{x})) = \mathbf{W}^{(\mathscr{L})}\phi(\mathbf{x}) + \mathbf{b}^{(\mathscr{L})}$$



The multilayer perceptron (MLP)

- A DNN takes the form

$$f(\mathbf{x}) = f^{(\mathscr{L})} \circ f^{(\mathscr{L}-1)} \circ \ldots \circ f^{(2)} \circ f^{(1)}(\mathbf{x})$$



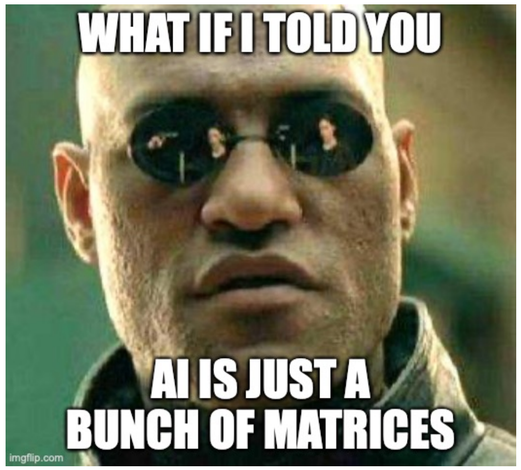
- An MLP is a network where each hidden layer output $\mathbf{h}^{(l)} \in \mathbb{R}^{H_l}$ is

$$\mathbf{h}^{(l)} = f^{(l)}(\mathbf{h}^{(l-1)}) = g(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \text{ for } l = 1,2,\ldots,\mathscr{L}-1$$

  - The layer input is the output of the previous layer $\mathbf{h}^{(l-1)} \in \mathbb{R}^{H_{l-1}}$
  - This undergoes a **linear transformation**
  - It then passes through a **non-linear activation function** g

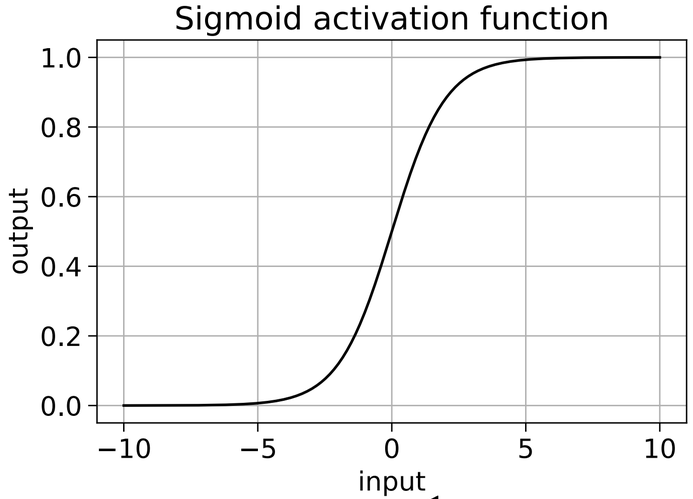
$\mathbf{a}^{(l)} = \mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}$ is known as the **pre-activation**

g is called an **activation function** and layer outputs $\mathbf{h}^{(l)}$ are called **activations**

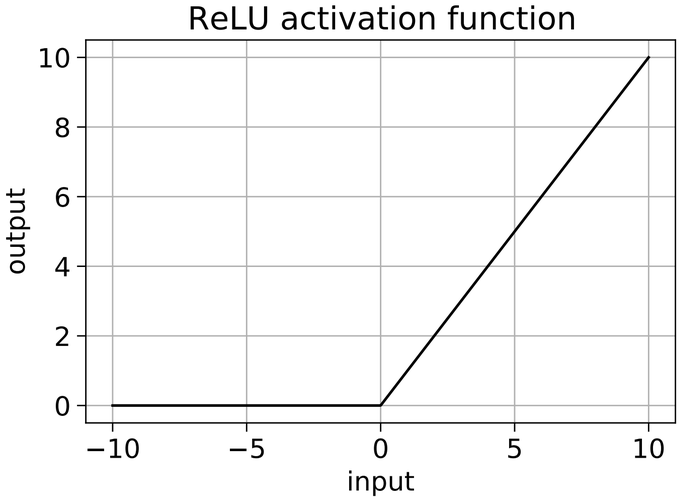


Activation functions

- These make our function non-linear. Without them an MLP collapses into a single linear transformation
- They are element-wise functions which means each element of the input vector is individually transformed



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



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

Two layer MLP

- For a 2 layer MLP with $\mathbf{x} \in \mathbb{R}^D$ and $f(\mathbf{x}) \in \mathbb{R}^K$ we have:

$$\mathbf{h}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

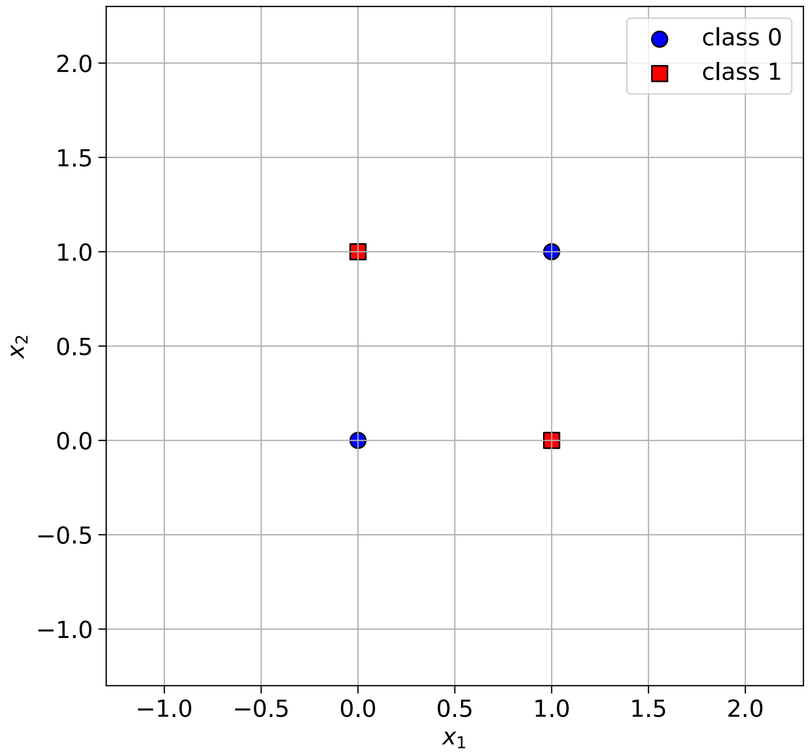
$$f(\mathbf{x}) = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}$$

- We can write the whole MLP as $f(\mathbf{x}) = \mathbf{W}^{(2)}g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}$

The form of g and the dimensionality (or width) of the hidden layer are design decisions

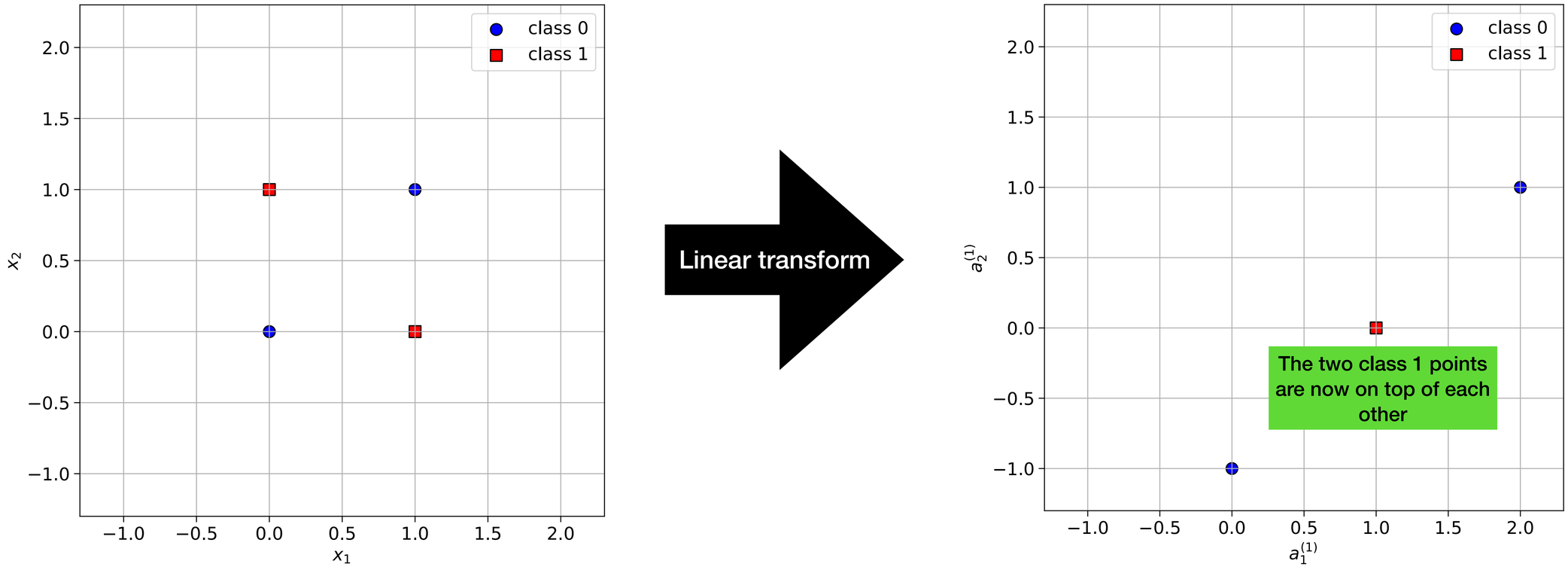
2 layer MLP (with pre-specified weights) for XOR

- We are going to walk through a 2 layer MLP solving a classification problem where the classes aren't linearly separable
- We will use a ReLU activation and a hidden layer with a width of 2



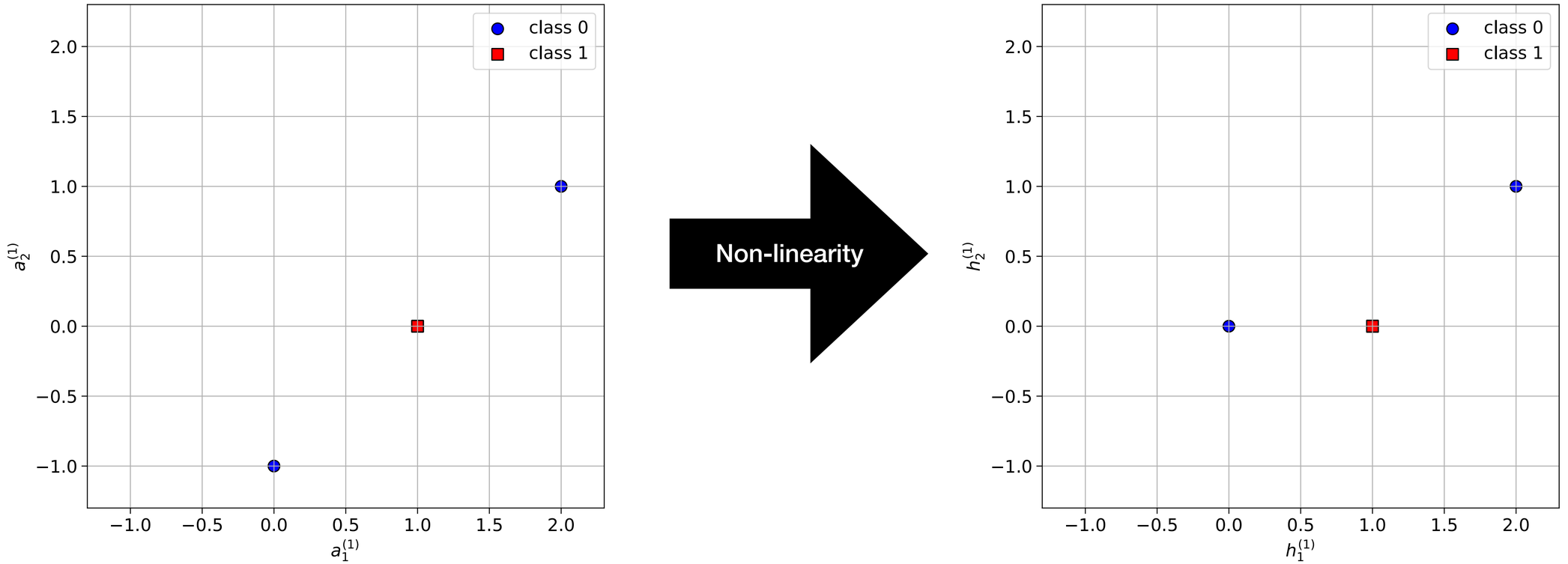
Layer 1: Compute the pre-activation

$$\mathbf{a}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \text{ where } \mathbf{W}_1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \text{ and } \mathbf{b}_1 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$



Layer 1: Apply the non-linearity

$\mathbf{h} = g(\mathbf{a})$. This is ReLU so $\begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \end{bmatrix} = \begin{bmatrix} \max(0, a_1^{(1)}) \\ \max(0, a_2^{(1)}) \end{bmatrix}$

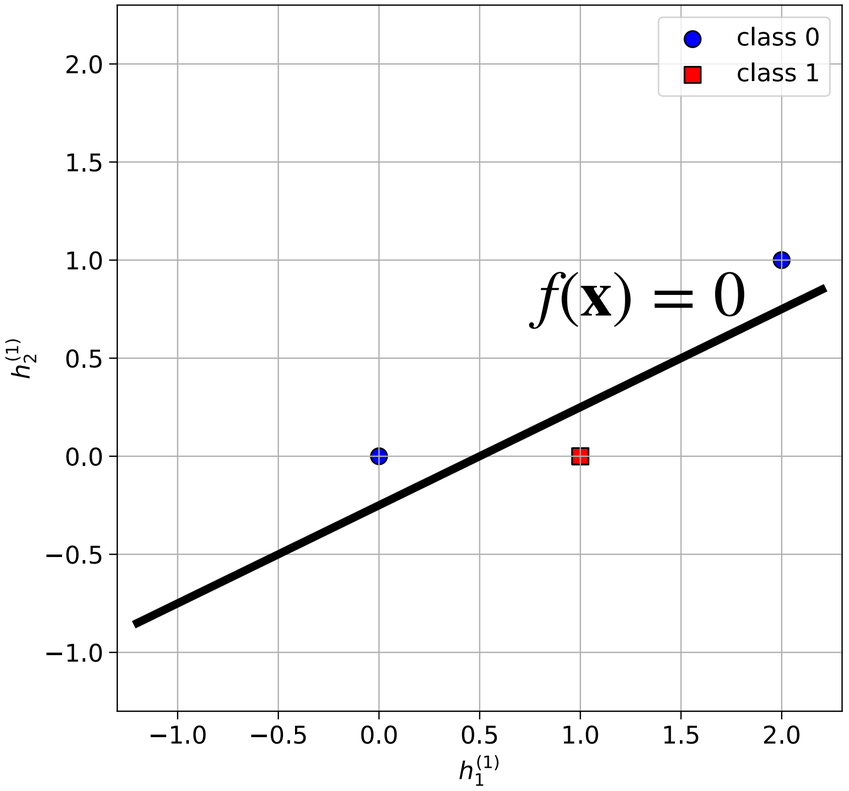


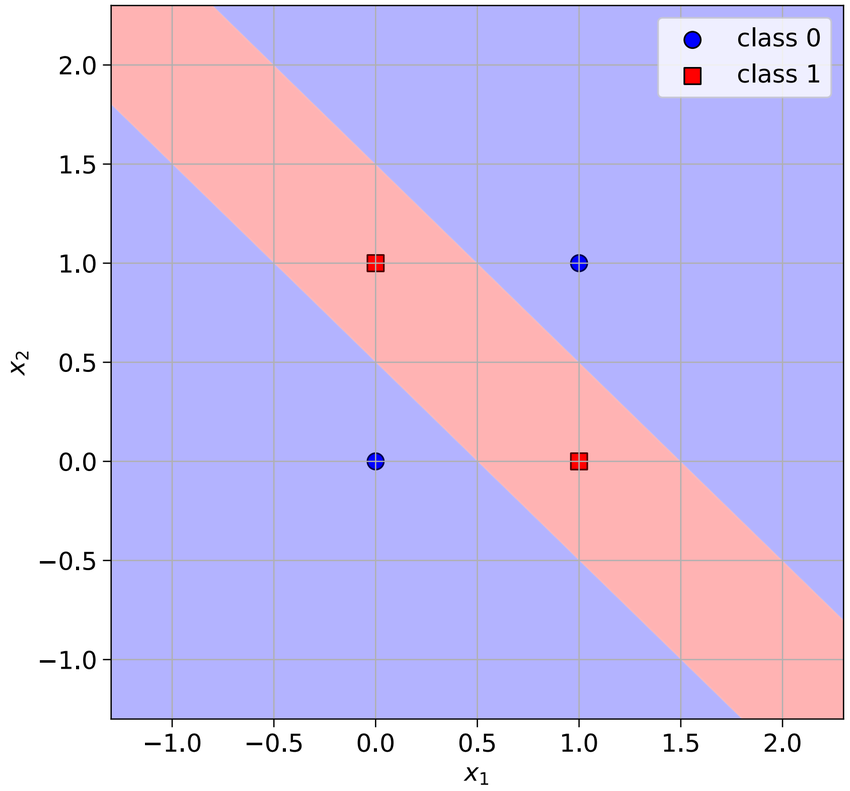
Layer 2: Just a linear classifier

Here, we have $\mathbf{W}_2 = [1 \quad -2]^\top$ (a vector)and $\mathbf{b}_2 = -0.5$ (a scalar) because its binary classification but I'm using the more general matrix/ vector notation anyway

$f(\mathbf{x}) = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}$. Let's just draw $f(\mathbf{x}) = 0$

This gives us a non-linear decision boundary in the original space



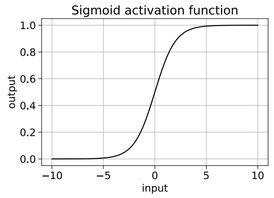


3 layer MLP

The form of g and the width of the hidden layers are design decisions!

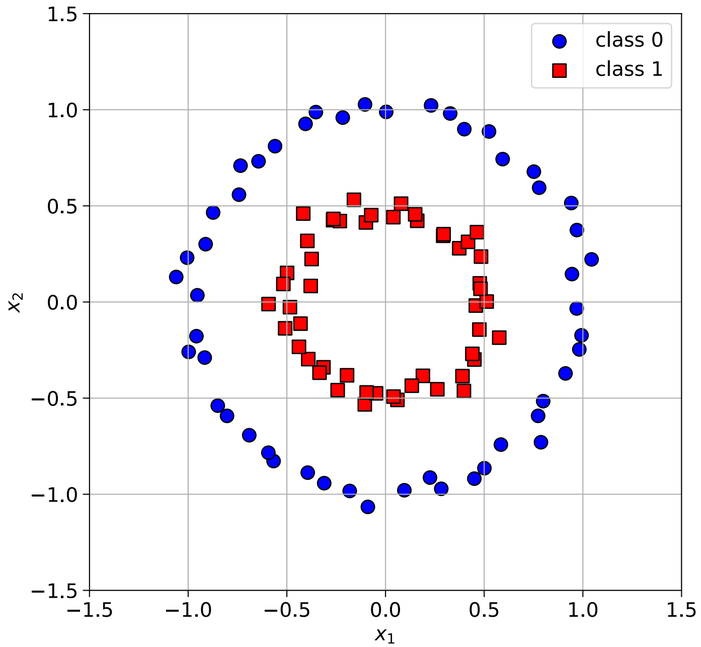
Credit to Oisin Mac Aodha for this example

- $\mathbf{h}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$
- $\mathbf{h}^{(2)} = g(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$
- $f(\mathbf{x}) = \mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}$

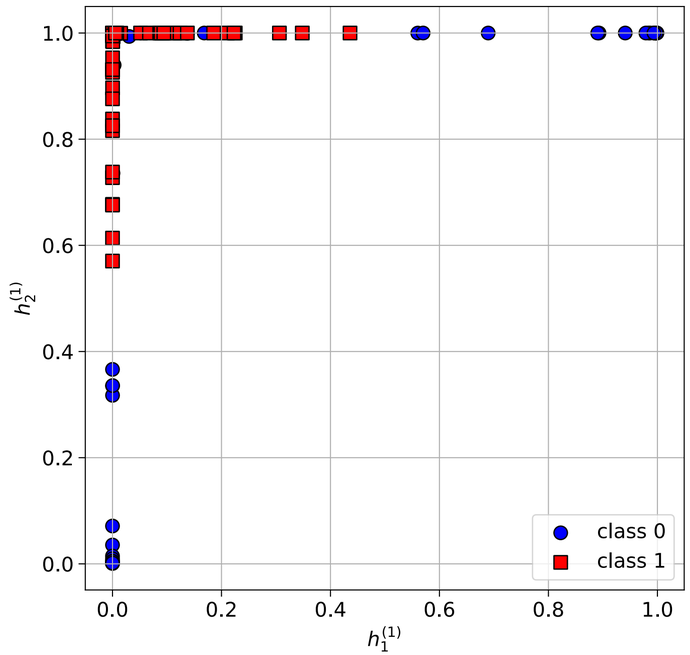


$$\mathbf{W}^{(1)} = \begin{bmatrix} 2.7 & 9.6 \\ 13.6 & 11.7 \end{bmatrix} \quad \mathbf{W}^{(2)} = \begin{bmatrix} -7.9 & 11.0 \\ 7.9 & -9.9 \end{bmatrix}$$

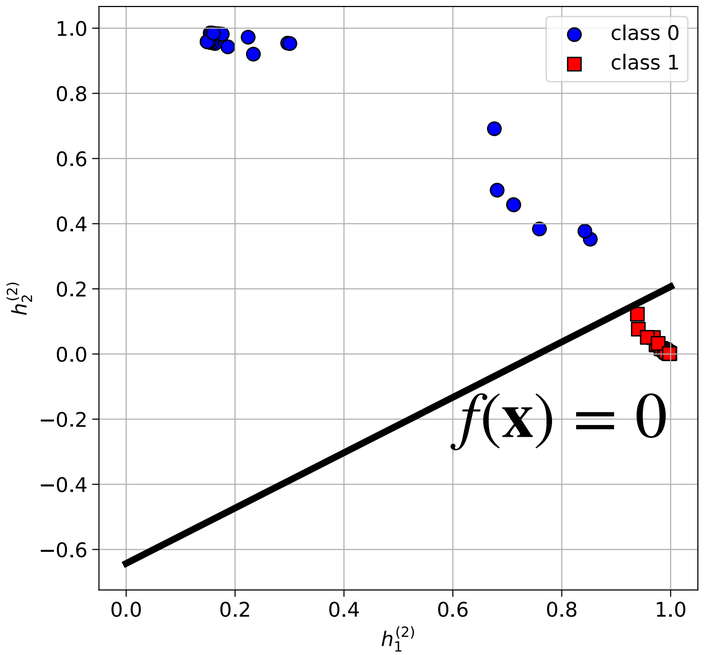
$$\mathbf{b}^{(1)} = \begin{bmatrix} -7.4 \\ 8.0 \end{bmatrix} \quad \mathbf{b}^{(2)} = \begin{bmatrix} 1.8 \\ 3.2 \end{bmatrix}$$



$\mathbf{x} \in \mathbb{R}^2$



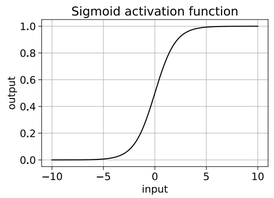
$\mathbf{h}^{(1)} \in \mathbb{R}^2$



$\mathbf{h}^{(2)} \in \mathbb{R}^2$

3 layer MLP

- $\mathbf{h}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$
- $\mathbf{h}^{(2)} = g(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$
- $f(\mathbf{x}) = \mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}$

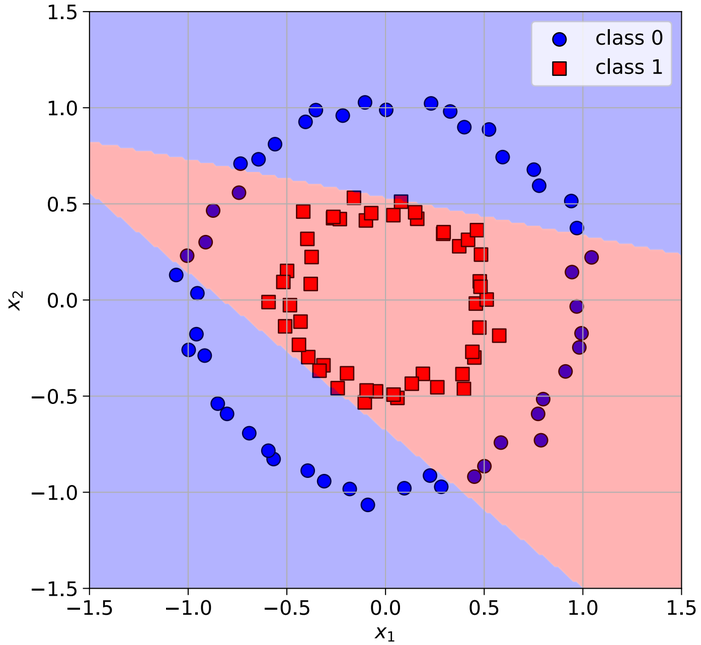


$$\mathbf{W}^{(1)} = \begin{bmatrix} 2.7 & 9.6 \\ 13.6 & 11.7 \end{bmatrix}$$

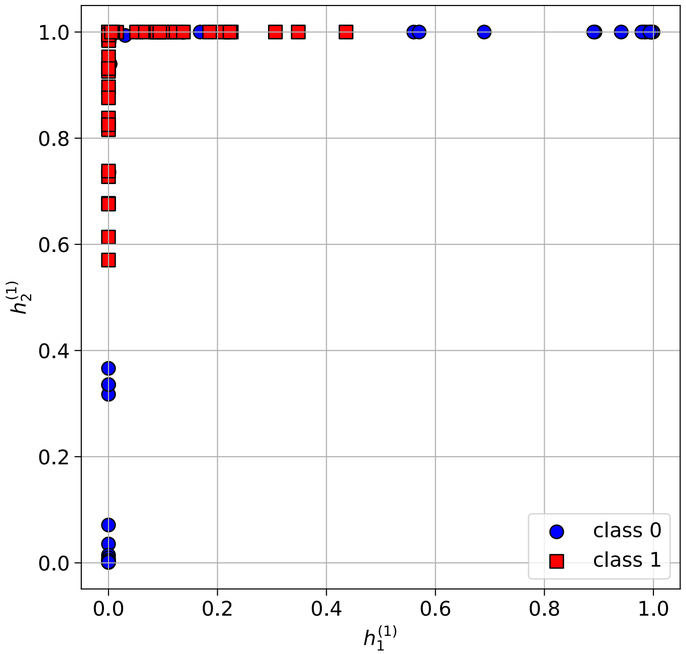
$$\mathbf{b}^{(1)} = \begin{bmatrix} -7.4 \\ 8.0 \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} -7.9 & 11.0 \\ 7.9 & -9.9 \end{bmatrix}$$

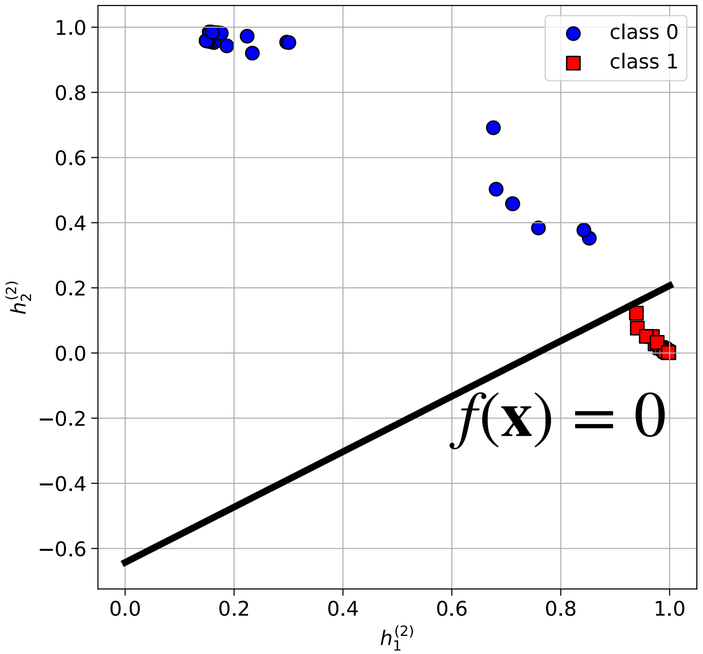
$$\mathbf{b}^{(2)} = \begin{bmatrix} 1.8 \\ 3.2 \end{bmatrix}$$



$\mathbf{x} \in \mathbb{R}^2$



$\mathbf{h}^{(1)} \in \mathbb{R}^2$

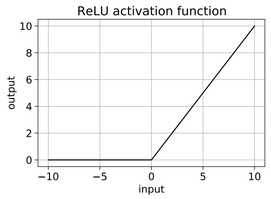


$\mathbf{h}^{(2)} \in \mathbb{R}^2$

Another 3 layer MLP

We are increasing the width of the 1st hidden layer significantly here

- $\mathbf{h}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$
- $\mathbf{h}^{(2)} = g(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$
- $f(\mathbf{x}) = \mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)}$

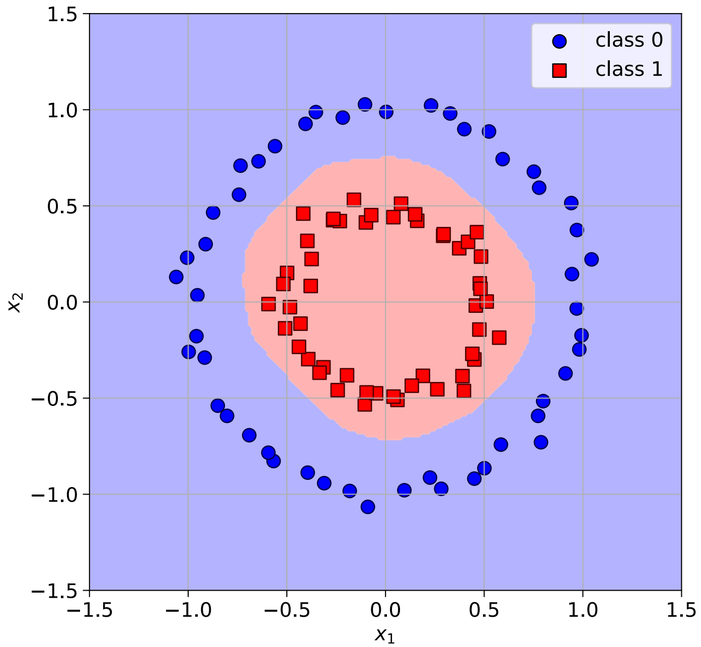


$\mathbf{W}^{(1)} \in \mathbb{R}^{100\times 2}$

$\mathbf{b}^{(1)} \in \mathbb{R}^{100}$

$\mathbf{W}^{(2)} \in \mathbb{R}^{2\times 100}$

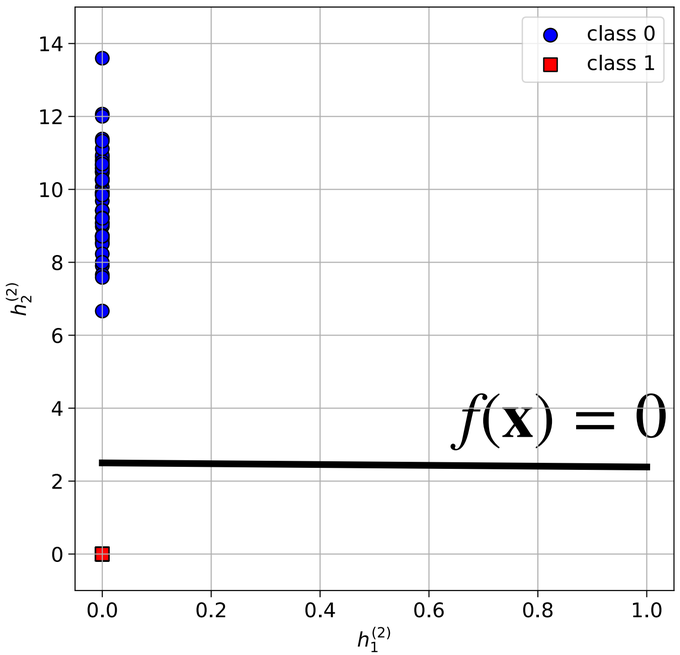
$\mathbf{b}^{(2)} \in \mathbb{R}^{2}$



$\mathbf{x} \in \mathbb{R}^2$

?

$\mathbf{h}^{(1)} \in \mathbb{R}^{100}$

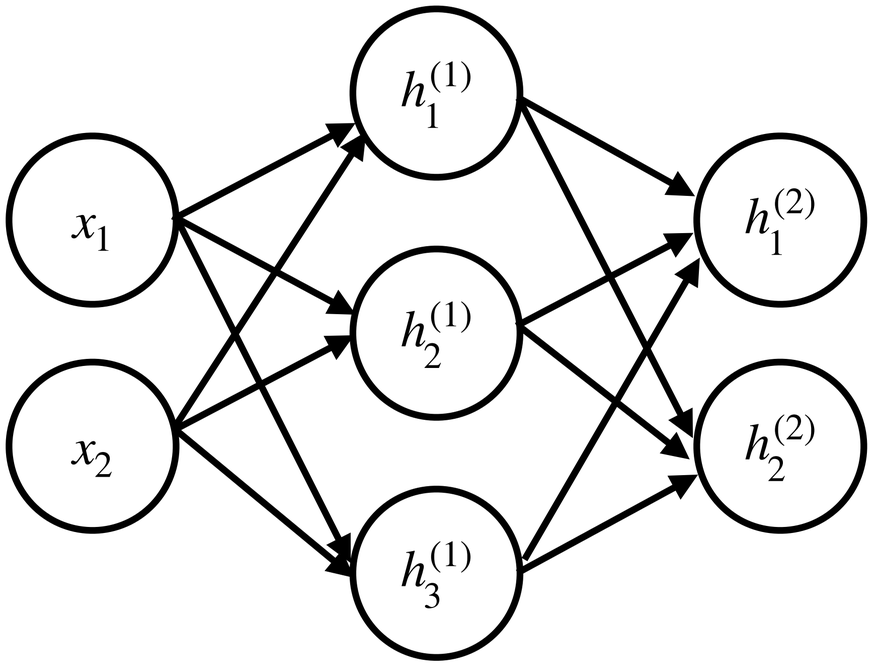


$\mathbf{h}^{(2)} \in \mathbb{R}^2$

Alternate view of a (2 layer) MLP

$$\mathbf{h}^{(1)} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$$

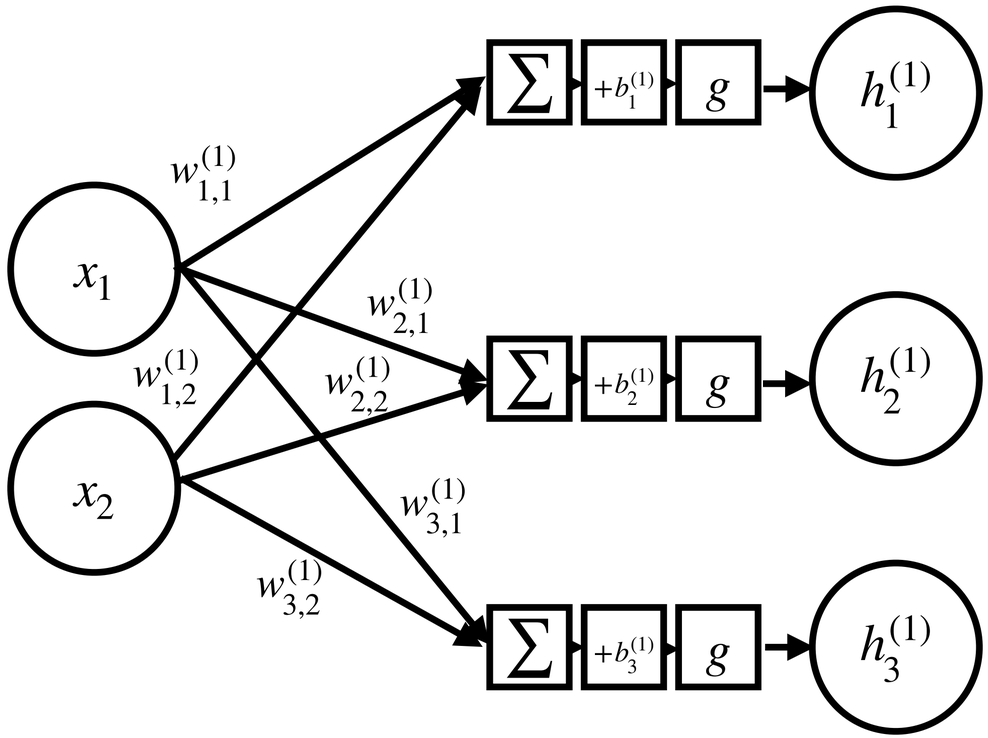
$$\mathbf{h}^{(2)} = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}$$



- Sometimes you see MLPs drawn as graphs
- Here, the elements of $\mathbf{x} \in \mathbb{R}^2, \mathbf{h}^{(1)} \in \mathbb{R}^3, \mathbf{h}^{(2)} \in \mathbb{R}^2$ are represented by nodes
- Stuff is happening at the node inputs!
- It follows that $\mathbf{W}^{(1)} \in \mathbb{R}^{3\times2}$, $\mathbf{b}^{(1)} \in \mathbb{R}^3$
- And also that $\mathbf{W}^{(2)} \in \mathbb{R}^{2\times3}$, $\mathbf{b}^{(1)} \in \mathbb{R}^2$
- Sometimes these nodes are referred to as *neurons*

MLP: Layer 1

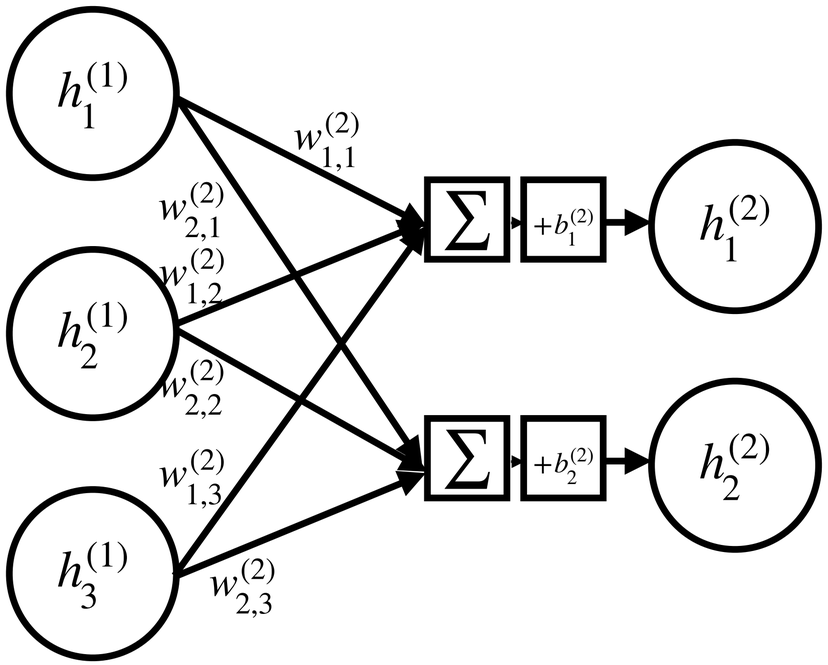
$$\mathbf{h}^{(1)} = \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ h_3^{(1)} \end{bmatrix} = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) = g\left(\begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} \\ w_{3,1}^{(1)} & w_{3,2}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix} \right)$$



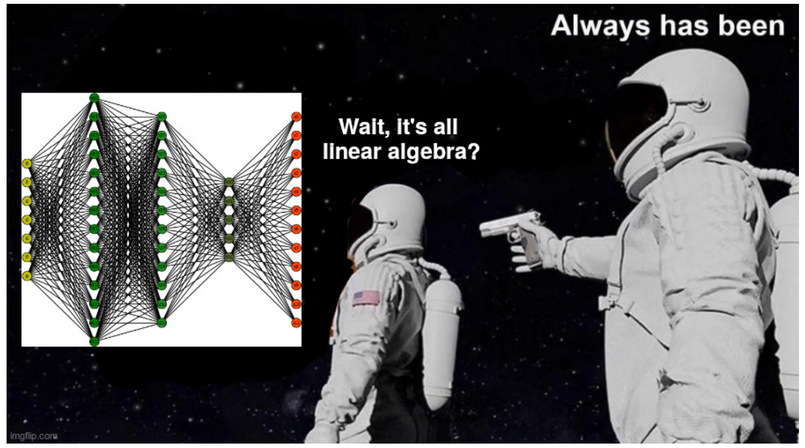
- Consider one of the *neurons of* $\mathbf{h}^{(1)}$
- It receives a weighted sum of the input neurons, to which a bias is added
- This *pre-activation* goes into an activation function g
- If we are using ReLU activations $g(z) = \max(0,z)$ then the pre-activation must be positive to pass through
- If this happens we say that the neuron has been **activated**

MLP: Layer 2

$$\mathbf{h}^{(2)} = \begin{bmatrix} h_1^{(2)} \\ h_2^{(2)} \end{bmatrix} = \mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} & w_{1,3}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} & w_{2,3}^{(2)} \end{bmatrix} \begin{bmatrix} h_1^{(1)} \\ h_2^{(1)} \\ h_3^{(1)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} \\ b_2^{(2)} \end{bmatrix}$$



- There is no activation function for the last layer in this example
- It's just a matrix multiplied by a vector plus another vector
- The previous layer was the same + a non-linearity



https://www.reddit.com/r/machinelearningmemes/comments/hst89w/always_has_been/

Why MLPs?

- We've gone from **learning your own features** to a bunch of linear transformations + activation functions
- There is a practical reason: apart from the activation function it's all just matrix multiplies which computers are very good at
- There is also theory in the form of a universal approximation theorem
- This basically tells us an MLP with at least 2 layers (and appropriate g) can represent a wide range of functions when they have the right weights

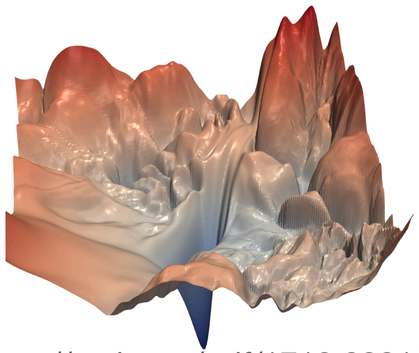
See https://www.deeplearningbook.org/contents/mlp.html 6.4.1 and https://cognitivemedium.com/magic_paper/assets/Hornik.pdf

Too good to be true?

~~Step 1: Use a 2 layer MLP to solve intelligence~~

~~Step 2: Use that to solve everything else~~

- The universal approximation theorem tells us an appropriate 2 layer MLP exists for lots of functions
- It doesn't tell us how wide the hidden layer should be or what weights to use!
- To make things worse, losses involving DNNs are generally **non-convex** :(
- (But this isn't actually that bad :)



https://arxiv.org/pdf/1712.09913.pdf

Going deeper

- Empirically, deeper networks (those with more layers) tend to work better up to a certain point

- Now is good time to mention that deep learning is **very empirical**
- There are rules of thumb for e.g. the number of layers, layer widths
- However, often you need to try stuff out (or use existing models)

https://knowyourmeme.com/photos/531557-we-need-to-go-deeper

Learning the parameters of a 2 layer MLP

- For $\mathbf{x} \in \mathbb{R}^D$ we can push a dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ through a 2 layer MLP using

$$\mathbf{H}^{(1)} = g(\mathbf{X}\mathbf{W}^{(1)\top} + \mathbf{1}\,\mathbf{b}^{(1)\top})$$

$$\mathbf{H}^{(2)} = \mathbf{H}^{(1)}\mathbf{W}^{(2)\top} + \mathbf{1}\,\mathbf{b}^{(2)\top}$$

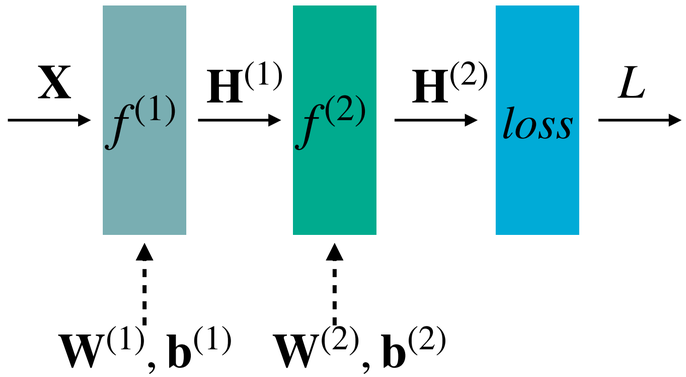
$\mathbf{1} \in \mathbb{R}^N$ is a vector of ones

- The learning process is very similar to that of linear models
- We pick an appropriate loss function L e.g. log loss for classification
- We then find the parameters that minimise the loss
- i.e. we solve $\underset{\boldsymbol{\theta}}{\text{minimise}}\, L$ where $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}\}$

The chain rule

- We can solve $\underset{\boldsymbol{\theta}}{\text{minimise}}\, L$ for $\boldsymbol{\theta} = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}\}$ using GD
- This involves computing gradients $\nabla_{\boldsymbol{\theta}} L = \left\{ \left(\frac{\partial L}{\partial \mathbf{W}^{(1)}}\right)^\top, \left(\frac{\partial L}{\partial \mathbf{b}^{(1)}}\right)^\top, \left(\frac{\partial L}{\partial \mathbf{W}^{(2)}}\right)^\top, \left(\frac{\partial L}{\partial \mathbf{b}^{(2)}}\right)^\top \right\}$
- We can obtain expressions for these using the chain rule

$$\mathbf{H}^{(1)} = g(\mathbf{X}\mathbf{W}^{(1)\top} + \mathbf{1}\,\mathbf{b}^{(1)\top})$$
$$\mathbf{H}^{(2)} = \mathbf{H}^{(1)}\mathbf{W}^{(2)\top} + \mathbf{1}\,\mathbf{b}^{(2)\top}$$



$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{W}^{(2)}} \qquad \frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{b}^{(2)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{H}^{(1)}} \frac{\partial \mathbf{H}^{(1)}}{\partial \mathbf{W}^{(1)}} \qquad \frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{H}^{(1)}} \frac{\partial \mathbf{H}^{(1)}}{\partial \mathbf{b}^{(1)}}$$

Warning! There be Jacobians. We aren't going to delve into what these terms actually look like on this course.

Automatic differentiation

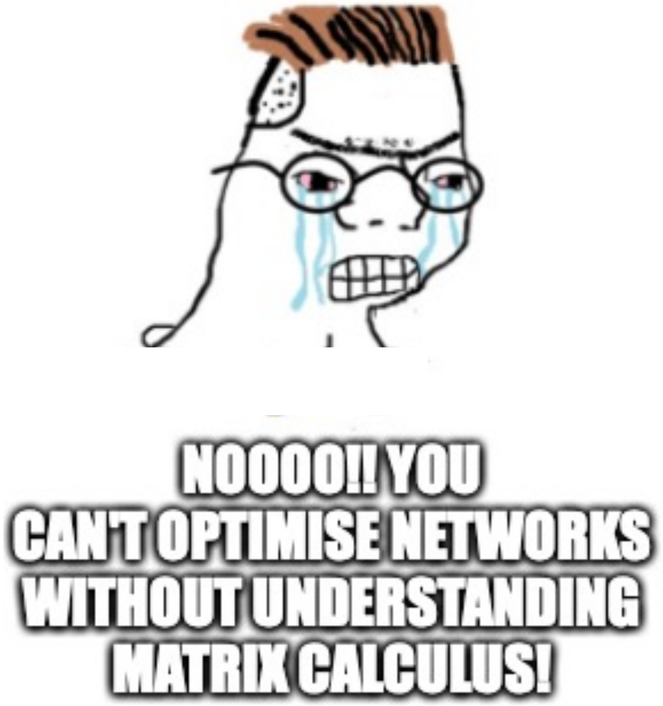
- Computers can perform automatic differentiation (/auto-diff/autograd/magic)
- We don’t need to work out closed form expressions for any derivatives!

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{W}^{(2)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(1)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{H}^{(1)}} \frac{\partial \mathbf{H}^{(1)}}{\partial \mathbf{W}^{(1)}}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{b}^{(2)}}$$

$$\frac{\partial L}{\partial \mathbf{b}^{(1)}} = \frac{\partial L}{\partial \mathbf{H}^{(2)}} \frac{\partial \mathbf{H}^{(2)}}{\partial \mathbf{H}^{(1)}} \frac{\partial \mathbf{H}^{(1)}}{\partial \mathbf{b}^{(1)}}$$



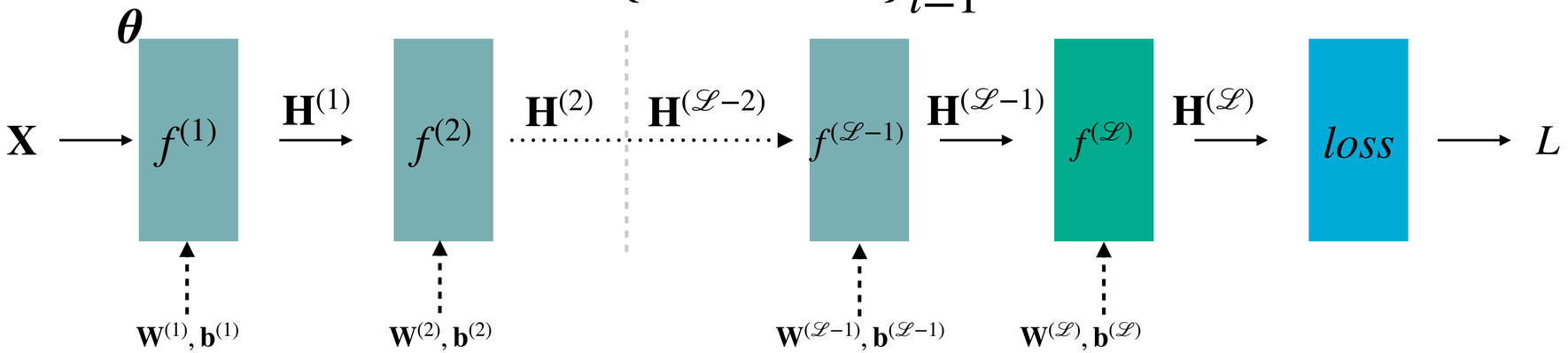
Learning the parameters of an $\mathscr{L}$ layer MLP

- For a dataset matrix $\mathbf{X}$ our $\mathscr{L}$ layer MLP is given by:

$$\mathbf{H}^{(l)} = g^{(l)}(\mathbf{H}^{(l-1)}\mathbf{W}^{(l)\top} + \mathbf{1}\,\mathbf{b}^{(l)\top}) \text{ for } l = 1,2,\ldots,\mathscr{L}$$

- $\mathbf{H}^{(0)} = \mathbf{X}$ and $g^{(l)}$ is a non-linear activation function e.g. ReLU for all layers but the last, which is typically the identity function

- The loss function takes in $\mathbf{H}^{(\mathscr{L})}$ (and some labels/targets) and we want to solve $\text{minimise}\, L$ where $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=1}^{\mathscr{L}}$



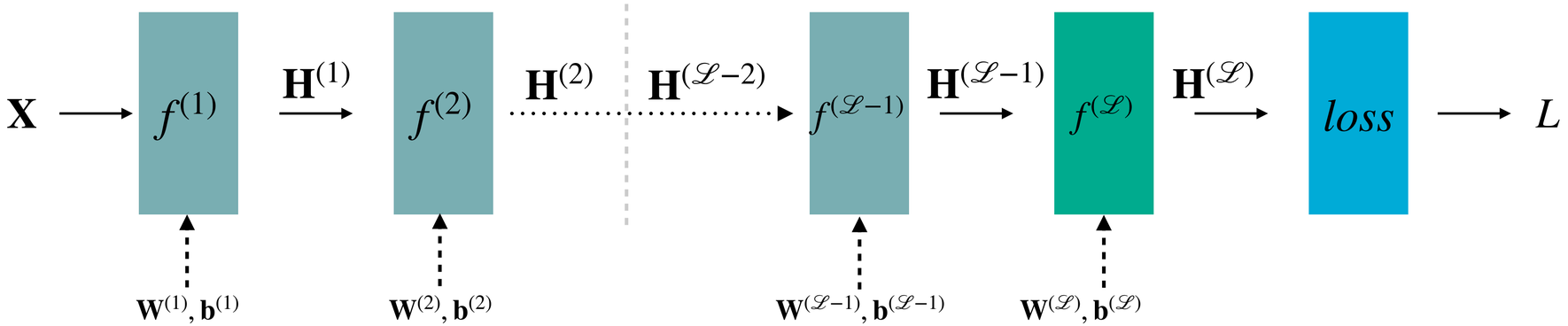
More chain rule!

See Murphy p435 if you're curious about the transposes. Otherwise, don't worry about them :)

- To use GD we need to compute $\nabla_{\boldsymbol{\theta}} L = \left\{ \left(\frac{\partial L}{\partial \mathbf{W}^{(l)}} \right)^\top, \left(\frac{\partial L}{\partial \mathbf{b}^{(l)}} \right)^\top \right\}_{l=1}^{\mathscr{L}}$
- We start with the last layer and can use the chain rule to write

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L})}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{W}^{(\mathscr{L})}} \qquad \frac{\partial L}{\partial \mathbf{b}^{(\mathscr{L})}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{b}^{(\mathscr{L})}}$$

- These expression are very similar so I'll just consider the $\mathbf{W}$ gradients for now, knowing we can obtain the $\mathbf{b}$ gradients in the same way



What do you notice?

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L})}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{W}^{(\mathscr{L})}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-1)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{W}^{(\mathscr{L}-1)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-2)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{H}^{(\mathscr{L}-2)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-2)}}{\partial \mathbf{W}^{(\mathscr{L}-2)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-3)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{H}^{(\mathscr{L}-2)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-2)}}{\partial \mathbf{H}^{(\mathscr{L}-3)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-3)}}{\partial \mathbf{W}^{(\mathscr{L}-3)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-4)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{H}^{(\mathscr{L}-2)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-2)}}{\partial \mathbf{H}^{(\mathscr{L}-3)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-3)}}{\partial \mathbf{H}^{(\mathscr{L}-4)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-4)}}{\partial \mathbf{W}^{(\mathscr{L}-4)}}$$

The same terms keep cropping up

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L})}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{W}^{(\mathscr{L})}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-1)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{W}^{(\mathscr{L}-1)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-2)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{H}^{(\mathscr{L}-2)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-2)}}{\partial \mathbf{W}^{(\mathscr{L}-2)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-3)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{H}^{(\mathscr{L}-2)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-2)}}{\partial \mathbf{H}^{(\mathscr{L}-3)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-3)}}{\partial \mathbf{W}^{(\mathscr{L}-3)}}$$

$$\frac{\partial L}{\partial \mathbf{W}^{(\mathscr{L}-4)}} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-1)}}{\partial \mathbf{H}^{(\mathscr{L}-2)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-2)}}{\partial \mathbf{H}^{(\mathscr{L}-3)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-3)}}{\partial \mathbf{H}^{(\mathscr{L}-4)}} \frac{\partial \mathbf{H}^{(\mathscr{L}-4)}}{\partial \mathbf{W}^{(\mathscr{L}-4)}}$$

- We can write $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \mathbf{G}^{(l)} \frac{\partial \mathbf{H}^{(l)}}{\partial \mathbf{W}^{(l)}}$ where $\mathbf{G}^{(l)} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}} \frac{\partial \mathbf{H}^{(\mathscr{L})}}{\partial \mathbf{H}^{(\mathscr{L}-1)}} \cdots \frac{\partial \mathbf{H}^{(l+1)}}{\partial \mathbf{H}^{(l)}}$
- We can iteratively compute $\mathbf{G}^{(l-1)} = \mathbf{G}^{(l)} \frac{\partial \mathbf{H}^{(l)}}{\partial \mathbf{H}^{(l-1)}}$ so we don't have to repeatedly calculate the same terms

The backpropagation algorithm

- Goal: Obtain gradients $\nabla_{\boldsymbol{\theta}} L = \left\{ \left(\frac{\partial L}{\partial \mathbf{W}^{(l)}} \right)^\top, \left(\frac{\partial L}{\partial \mathbf{b}^{(l)}} \right)^\top \right\}_{l=1}^{\mathscr{L}}$
- Compute $\mathbf{G}^{(\mathscr{L})} = \frac{\partial L}{\partial \mathbf{H}^{(\mathscr{L})}}$
- For l in $\mathscr{L}, \mathscr{L}-1, \ldots, 2, 1$:
  1. Compute $\frac{\partial L}{\partial \mathbf{W}^{(l)}} = \mathbf{G}^{(l)} \frac{\partial \mathbf{H}^{(l)}}{\partial \mathbf{W}^{(l)}}$ and $\frac{\partial L}{\partial \mathbf{b}^{(l)}} = \mathbf{G}^{(l)} \frac{\partial \mathbf{H}^{(l)}}{\partial \mathbf{b}^{(l)}}$
  2. Compute $\mathbf{G}^{(l-1)} = \mathbf{G}^{(l)} \frac{\partial \mathbf{H}^{(l)}}{\partial \mathbf{H}^{(l-1)}}$

See Murphy Section 13.3 for a more detailed and rigorous description!

SGD for neural network training

Storing lots of activations for a whole dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ can be expensive. Because of this SGD is typically used for DNN training. The procedure is:

- Initialise DNN weights at random e.g. from a normal distribution
- For e in range(E):
  - Split dataset into equal sized **mini-batches** $\{\mathbf{X}^{(b)}, \mathbf{y}^{(b)}\}_{b=1}^{B}$ at random
  - For b in range(B):
    1. Compute $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{X}^{(b)}, \mathbf{y}^{(b)})$ using backpropagation
    2. Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \mathbf{X}^{(b)}, \mathbf{y}^{(b)})$

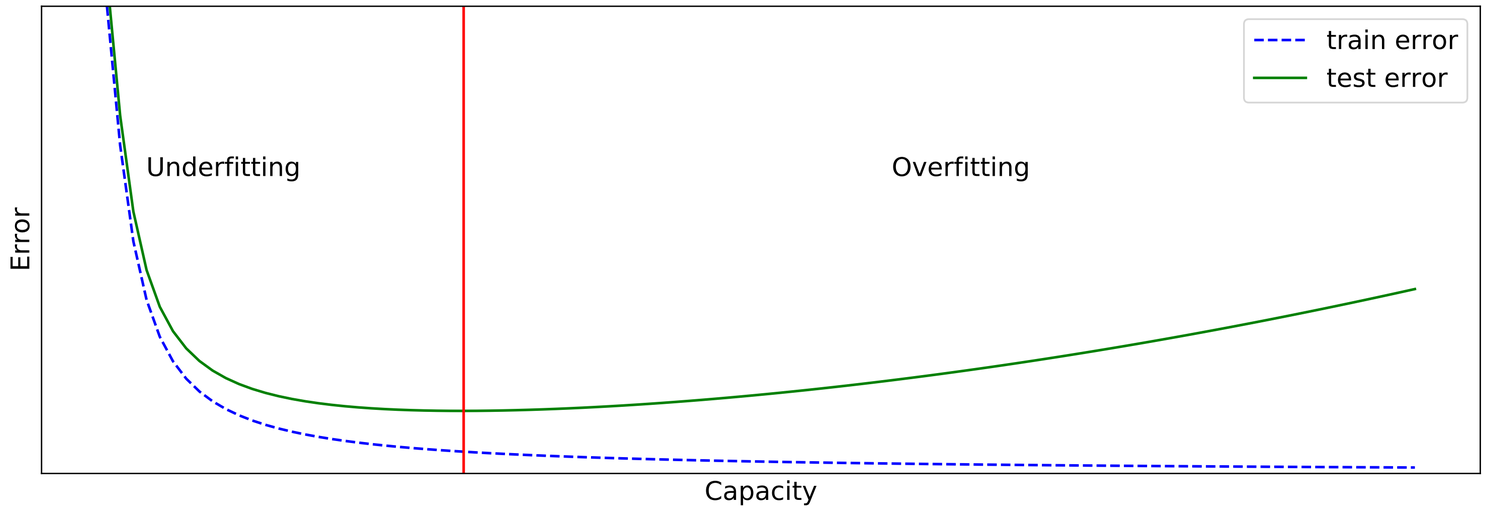
Each outer loop across the whole dataset is known as an *epoch*

Other optimisers are available

- e.g. the Adam optimiser (pictured right)
- Almost all take the gradients from backprop and do something with them
- You don’t need to know about any optimisers other than GD and SGD for this course

DNNs can overfit

- DNNs can represent lots of functions. They are high capacity models
- They are very susceptible to overfitting!
- Remember, we care about a model's ability to **generalise** to unseen data
- Regularisation is very important in DNNs!

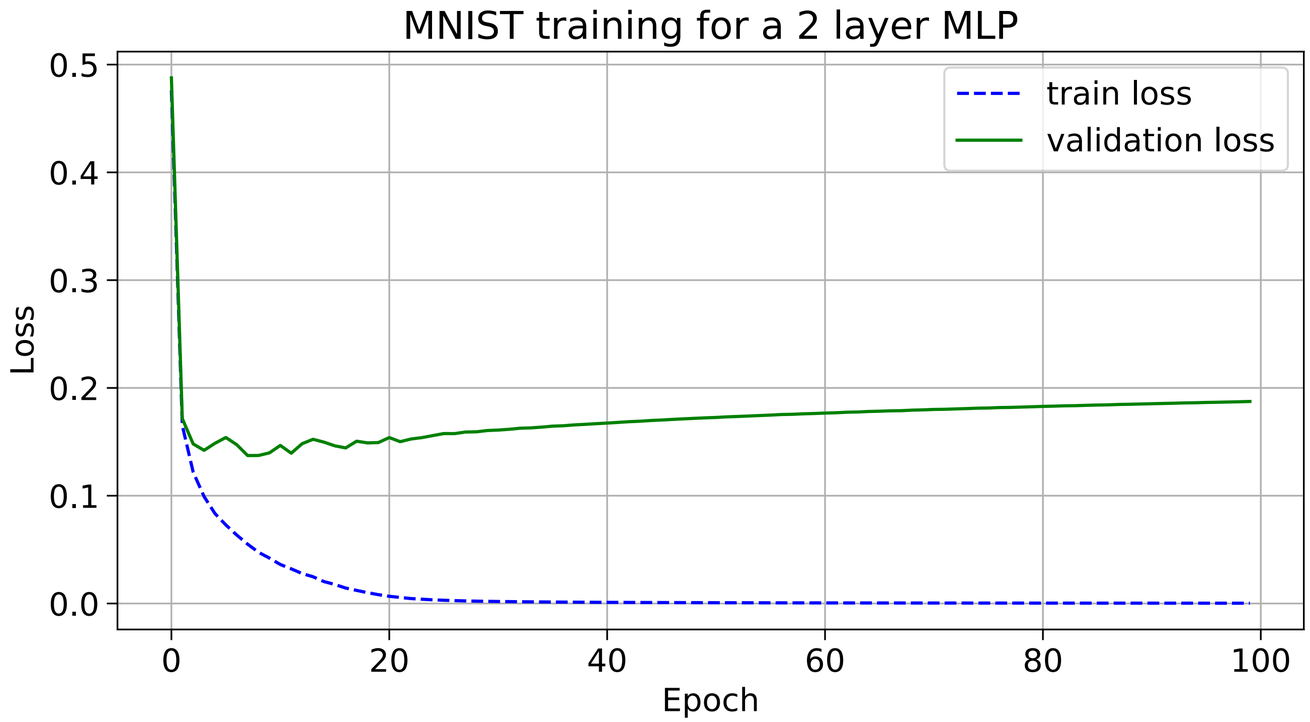


Early stopping

- Fitting to the test set is not allowed
- We can however look at the validation set throughout training as a proxy
- The model starts to overfit once validation loss stops decreasing with train loss
- We can stop training at this point

This looks very similar to the last figure!

Over training models tend to underfit and then overfit to the training data

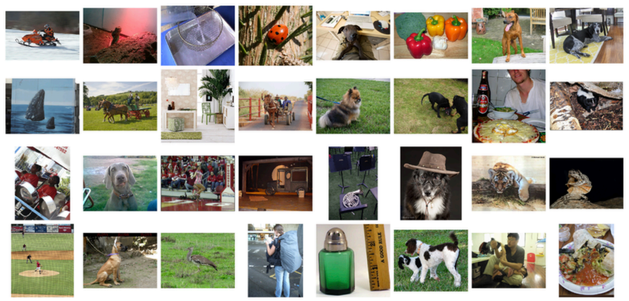


Weight decay

- Models that overfit tend to have large weights
- To mitigate this, we multiply all the weights by $1 - \lambda$ whenever we perform an update step in e.g. SGD
- λ is the amount of weight decay as is usually very small e.g. 10^{-4}
- This is basically equivalent to having L2 regularisation in the loss function

Why deep learning of all things?

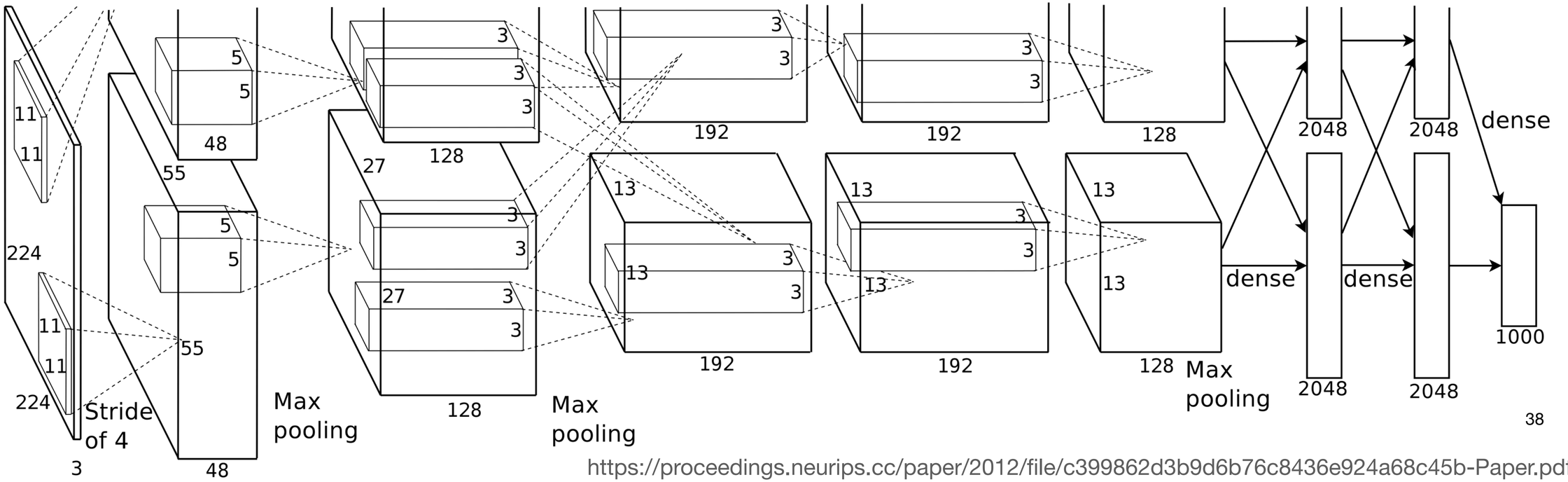
- A benchmark in computer vision is classification performance on ImageNet
- It is a 1000-way classification task with 1 million training images
- For the 2012 ImageNet challenge:
  - The 2nd place model used handcrafted features and got 26.2% top 5-error
  - The 1st place model used a deep neural network and got **15.3% top 5-error** (& 36.7% top 1-error)



https://arxiv.org/pdf/1409.0575.pdf

AlexNet (2012)

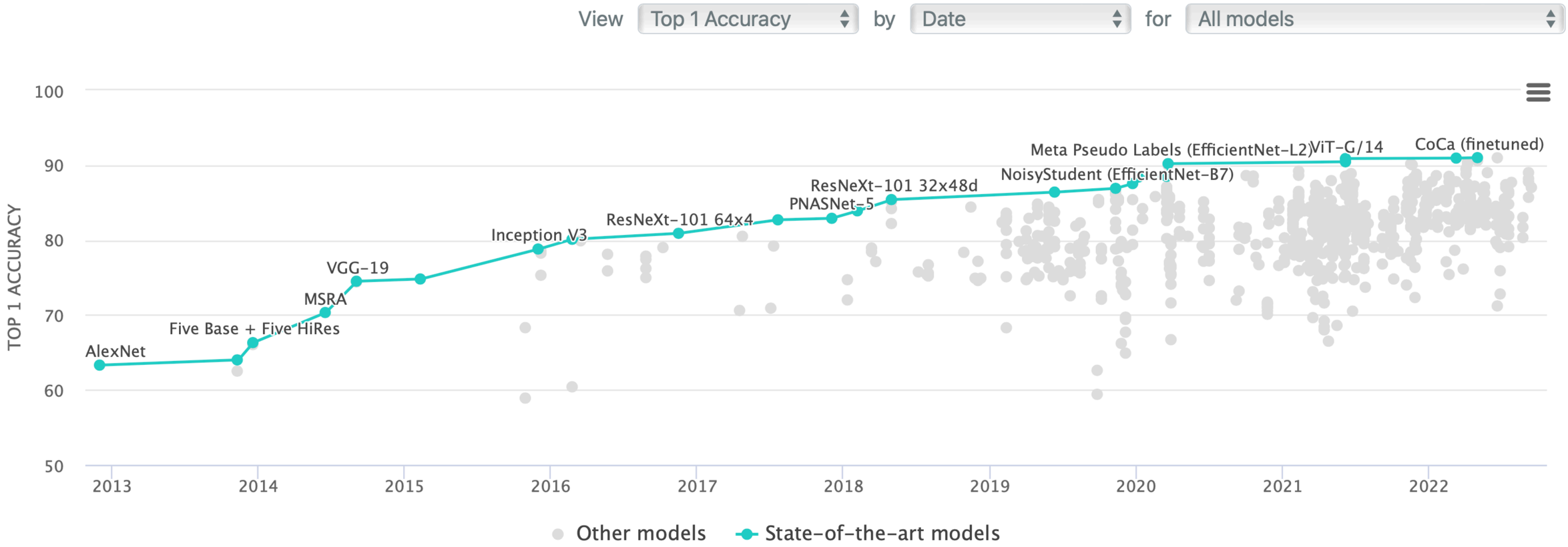
- The winning entry. It's split into two streams for 2 GPUs because of memory constraints (that no longer exist :))
- 5 convolutional layers, 3 max pools (interspersed), and 3 linear layers



https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

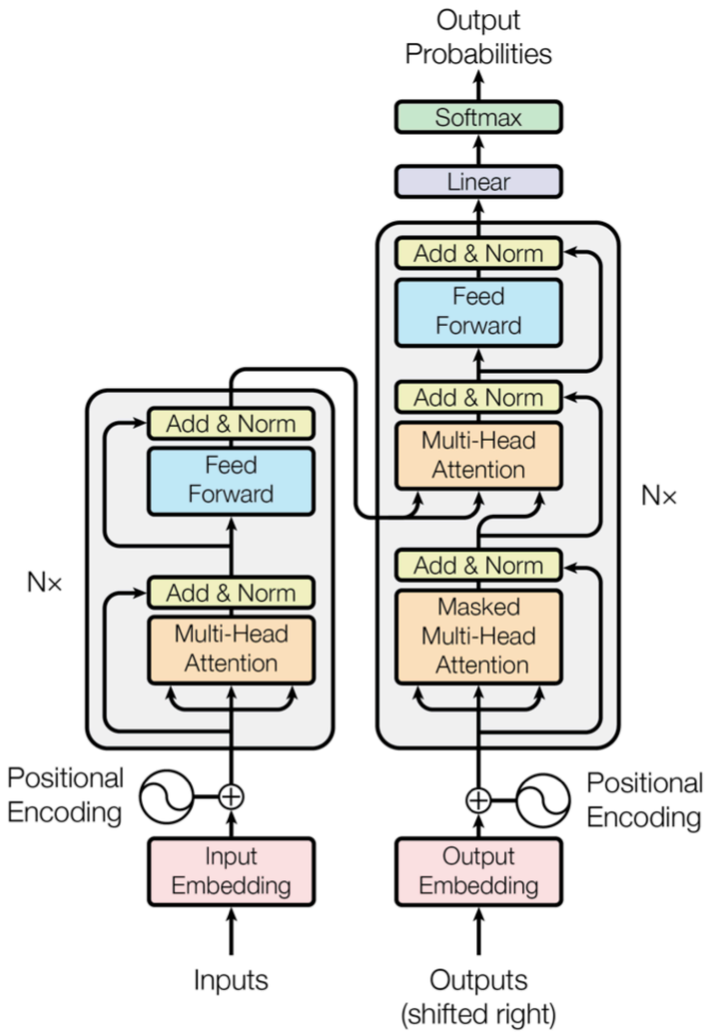
ImageNet top-1 accuracies

Leaderboard Dataset



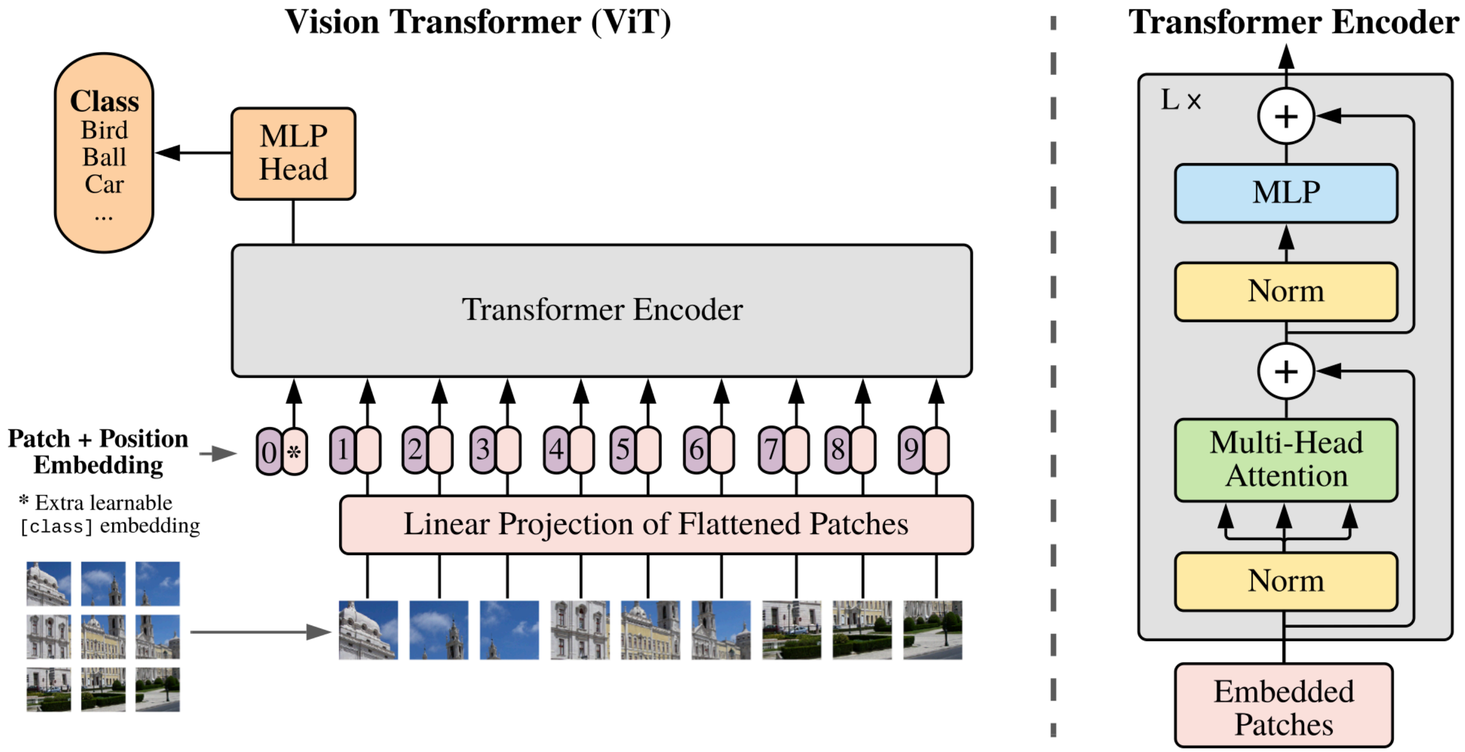
https://paperswithcode.com/sota/image-classification-on-imagenet

The transformer architecture (2017)



https://arxiv.org/abs/1706.03762

Vision transformers

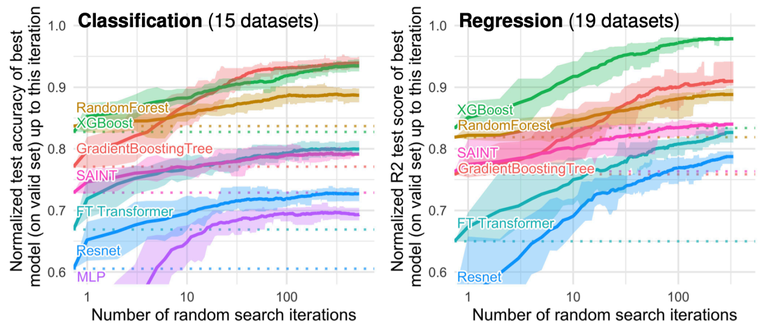


https://arxiv.org/pdf/2010.11929v2.pdf

https://upload.wikimedia.org/wikipedia/commons/thumb/0/04/ChatGPT_logo.svg/1200px-ChatGPT_logo.svg.png

Why not use deep learning for everything?

- With enough data, DNNs beat other ML approaches for learning on images, text, and audio data
- DNNs are often surpassed by decision tree-based models on tabular data
- DNN are near-impossible to interpret, so when this is required a linear model is preferable
- DNNs need lots of data to train from scratch which we may not have!
- DNNs are very expensive to train
- **We can however use their features for related tasks**



https://arxiv.org/pdf/2207.08815.pdf

Summary

- We have considered learning our features instead of using a pre-existing map
- We have seen how the structure of a DNN facilitates feature learning
- We have looked at the MLP architecture and worked through some examples
- We have found out how to train an MLP using backpropagation + SGD
- We looked at different ways to regularise DNNs

The end (of the lectures)

- You have visualised and analysed data
- You have considered the ethical implications of deploying ML in society
- You have learnt about linear models for classification and regression
- You have learnt about non-parametric and non-linear models
- You have written code to use these models

I hope you enjoyed it!