MONASH University

ETC3250/5250: Introduction to Machine Learning

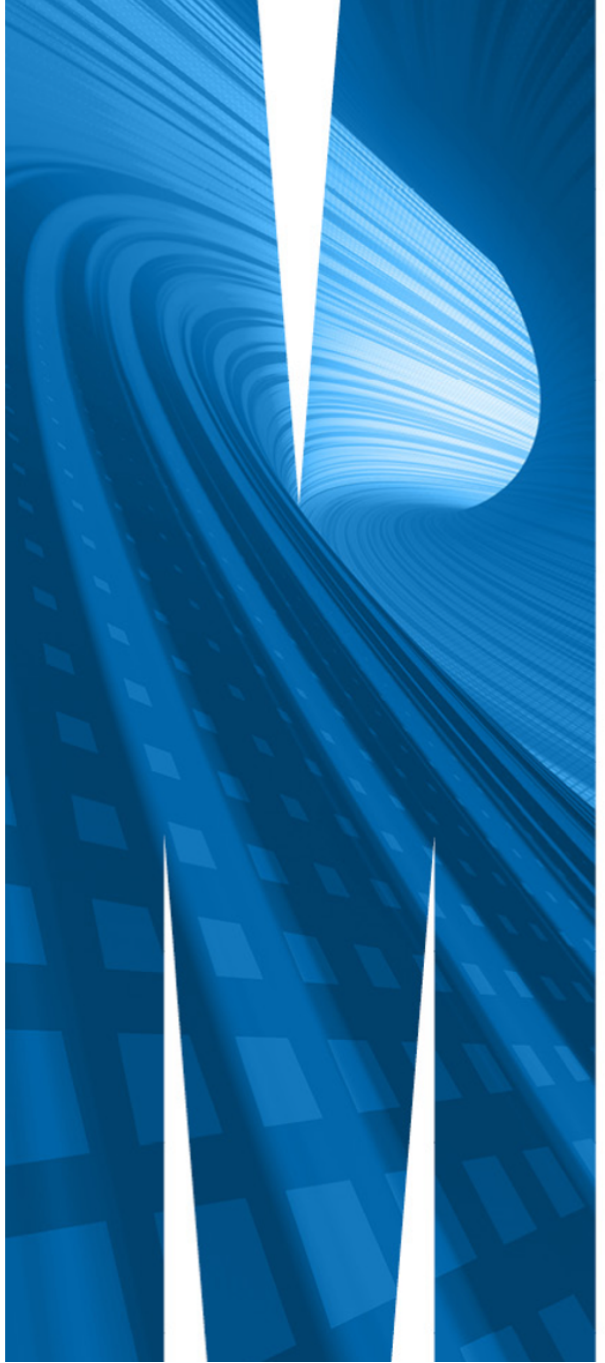
Neural networks and deep learning

Lecturer: *Professor Di Cook*

Department of Econometrics and Business Statistics

ETC3250.Clayton-x@monash.edu

Week 8a



What number is this?

This is a three, right?

What number is this?

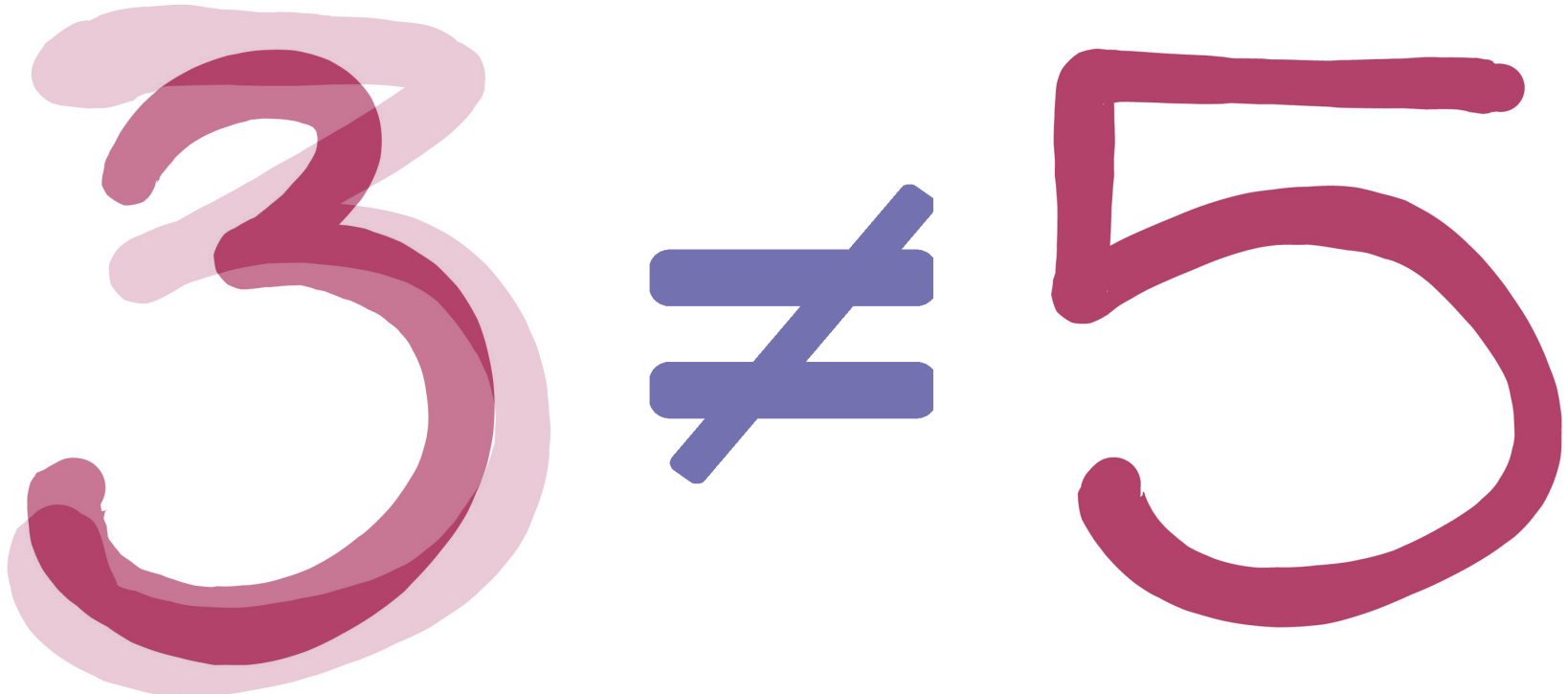
Is this also a three?

What number is this?

But what about this number? Not a three?

The human brain

The human brain can efficiently recognise that although the images of the two threes are different, they are the same number, and are both distinct from the five.



MNIST data

The MNIST data was presented to AT&T Bell Lab's to build automatic mail sorting machines.

Goal: Analyse handwritten digits and predict numbers written, given a grid of 28×28 pixels for each of the 60000 training images. Digits range from 0-9.

Sample images from MNIST test dataset .

MNIST data

How do we do this?

Humans are good at detecting different features about the images, such as thickness of line, angles, edges, completeness of circles, etc.

It is evident a complex relationship is presented in the images. Neural networks can help us automatically capture these complexities.

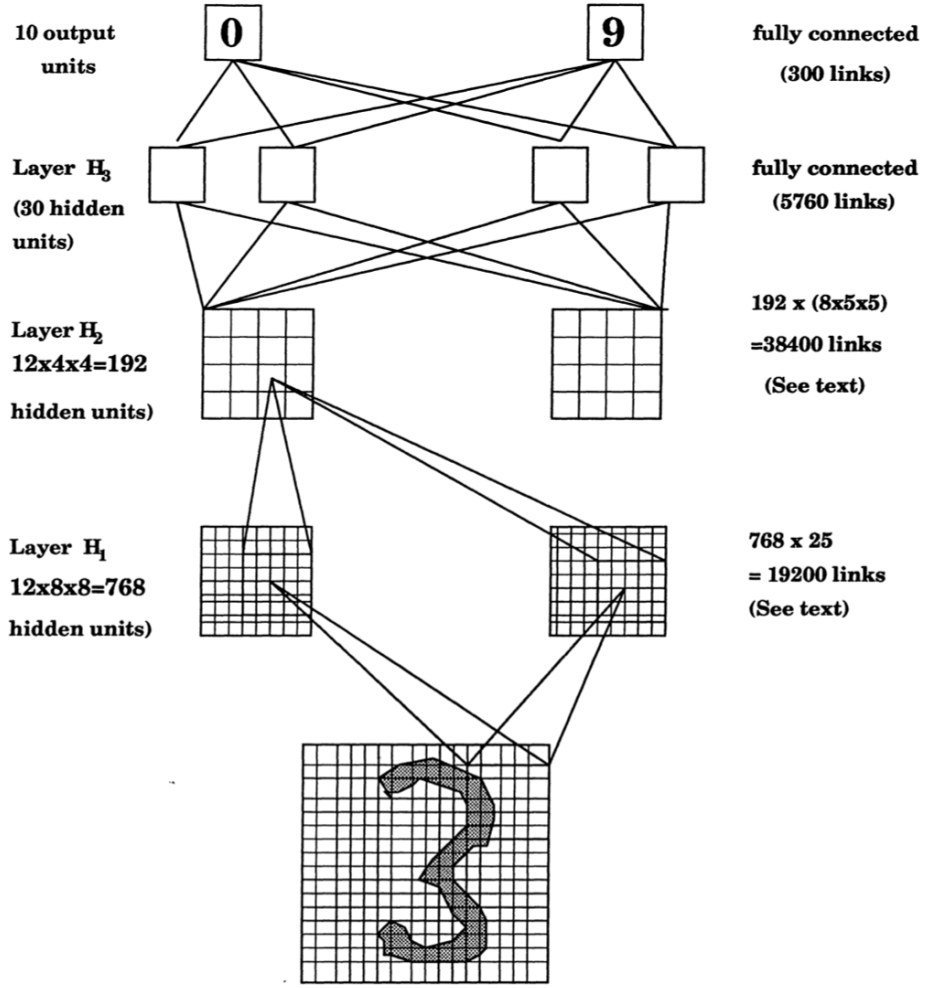


Image from Neural Networks: A Review from a Statistical Perspective

So, what are neural networks?

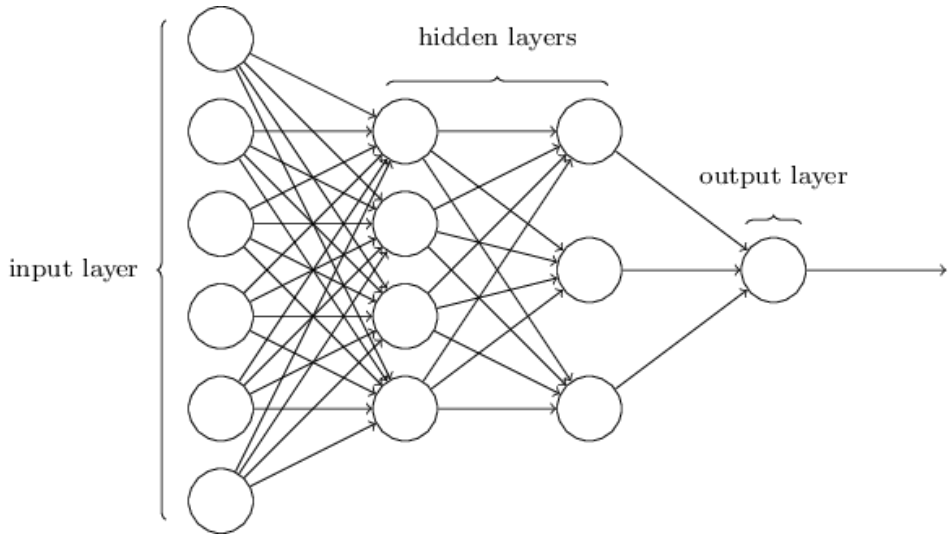
Idea: Capture a complex relationship between outputs and inputs by creating layers of derived variables.

y = output variable

x = original input variable

$f(x)$ = derived variable input

$$y = f_1(f_2(\ldots(f_d(x))))$$



Source: Hands on Machine Learning with R - Deep Learning

How do we build a neural network?

To build a feedforward neural network, we need four key components:

1. Input data (*in this case, the MNIST data*)
2. A pre-defined network architecture;
3. A feedback mechanism (optimisation) to enable the network to learn; and
4. A model training approach.

Following instructions at Hands on Machine Learning in R Chapter 13.

1. Preparing the data

Data preparation

There are some data cleaning steps we need to keep in mind before we use neural networks.

- Data needs input to be *numeric*. This means if our data has categorical variables, we will need to represent these as *dummy variables* (revise, Week 2!). This is also called one-hot encoding in ML literature.
- Neural nets are sensitive to scale of the feature values - hence they should be *standardised* first (have mean zero and unit variance).
- If response is categorical (such as "0" through "9" response in MNIST data) - needs to be recoded as binary matrix.

Data preparation

This arose when AT&T Bell Lab's was asked to help build automatic mail-sorting machines for the USPS aroound 1990. Data available at http://yann.lecun.com/exdb/mnist/ but owner has blocked automatic download with `dslabs::read_mnist()`.

To get mnist data follow the instructions at https://tensorflow.rstudio.com/guide/tfestimators/examples/mnist/

```r
library(tidyverse)
library(keras)

load(here::here("data/MNIST_data/mnist.rda"))
mnist$train$x <- mnist$train$x / 255
mnist$test$x <- mnist$test$x / 255
mnist_x <- mnist$train$x
mnist_y <- mnist$train$y

# Rename columns and standardize feature values
colnames(mnist_x) <- paste0("V", 1:ncol(mnist_x))
p <- ncol(mnist_x)

# One-hot encode response
# mnist_y <- to_categorical(mnist_y, 10) FAILURE
mnist_y_mat <- matrix(0, length(mnist_y), 10)
for (i in 1:nrow(mnist_y_mat))
  mnist_y_mat[i,mnist_y[i]] <- 1
mnist_y <- mnist_y_mat
```

2. Network Architecture

Network architecture

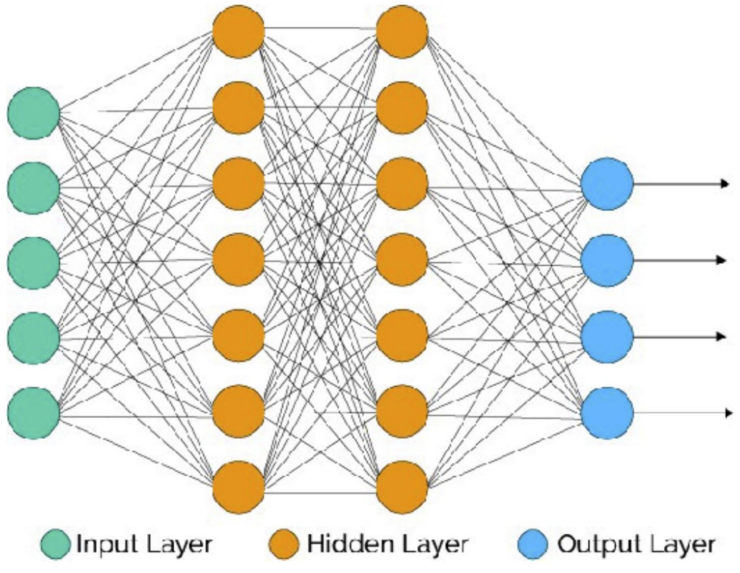
When building architecture for the neural network, we are concerned about two key features:

- The number of layers and nodes, and
- How signal is activated throughout the network.

Layers and nodes

Our complex relationships are captured using layers and nodes. There are two different types of layers, namely,

- Input and output layers, and
- Hidden layers.
  - No well-defined approach for selecting the number of hidden layers - this is just one of many hyperparameters we will need to tune! .monash-orange2[2-5 layers works well most of the time for regular tabular data].
  - The more hidden layers - the longer the model will take to train (as we are adding more parameters!)



Output layers

Choice of nodes for the output layer is determined by the ML task.

- If you are doing regression - a single node.
- Classification - a node for each class if multiclass.
- If binary, single node for probability of predicting success.

Think! How many output nodes will MNIST data neural network contain? 🤔

Building network structure in R

We use the `keras` package to build neural networks in R. This is *very different* to other forms of ML algorithms in R. In `keras`, we first define the network structure as a standalone from our data.

```
library(keras)
model <- keras_model_sequential() %>%
  layer_dense(units = 16,
              input_shape = p) %>%
  layer_dense(units = 16) %>%
  layer_dense(units = 10)
```

Activation - how do the layers speak?

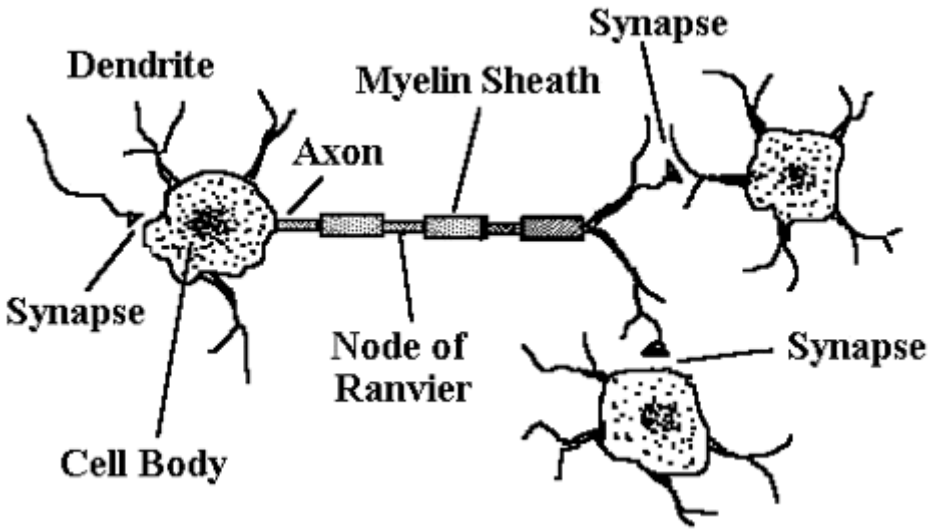
Now that we have our structure in place, we need to determine how to pass signal throughout the network.

It uses methods we already know: logistic and linear regression.

But first, some history

"A logical calculus of the ideas immanent in nervous activity" (1943) Warren S. McCulloch & Walter Pitts

Mathematical model for a neuron.



Logistic regression

Remember the logistic function:

$$y = \frac{e^{\beta_0 + \sum_{j=1}^{p} \beta_j x_j}}{1 + e^{\beta_0 + \sum_{j=1}^{p} \beta_j x_j}} = \frac{1}{1 + e^{-(\beta_0 + \sum_{j=1}^{p} \beta_j x_j)}}$$

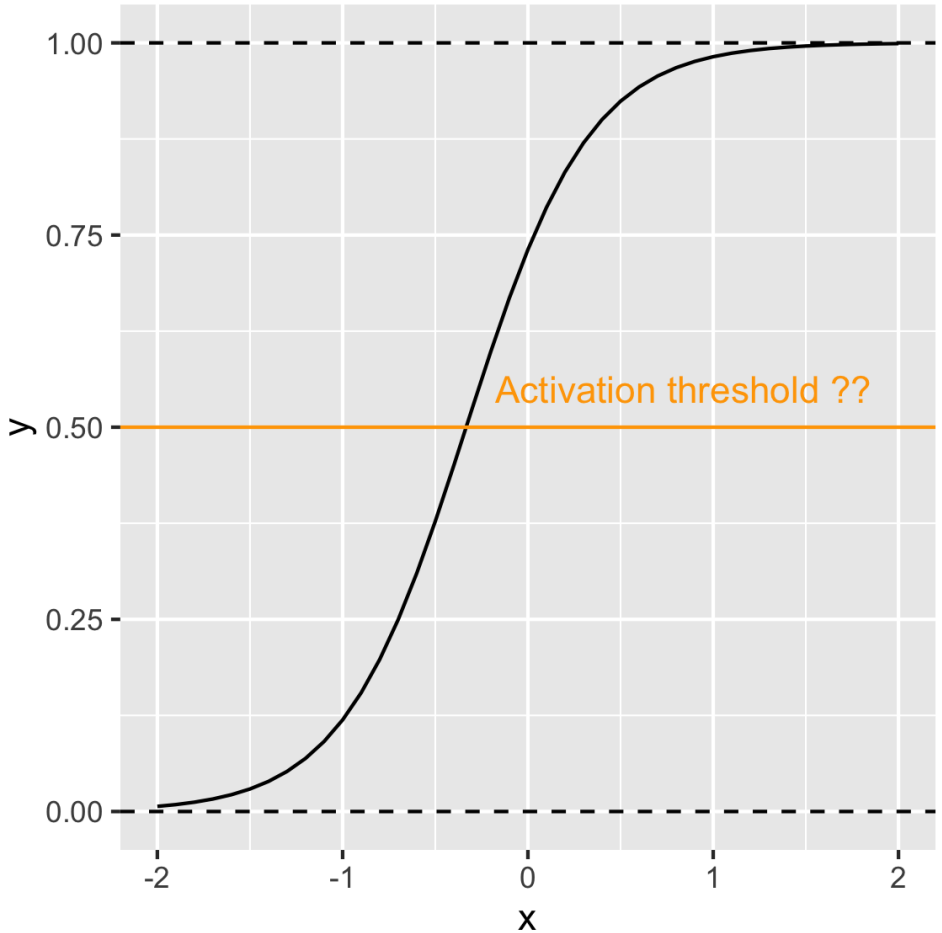
Alternatively,

$$\log_e \frac{y}{1-y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

Logistic regression

What the logistic function looks like:

$$y = \frac{1}{1 + e^{-(\beta_0 + \sum_{j=1}^{p} \beta_j x_j)}}$$



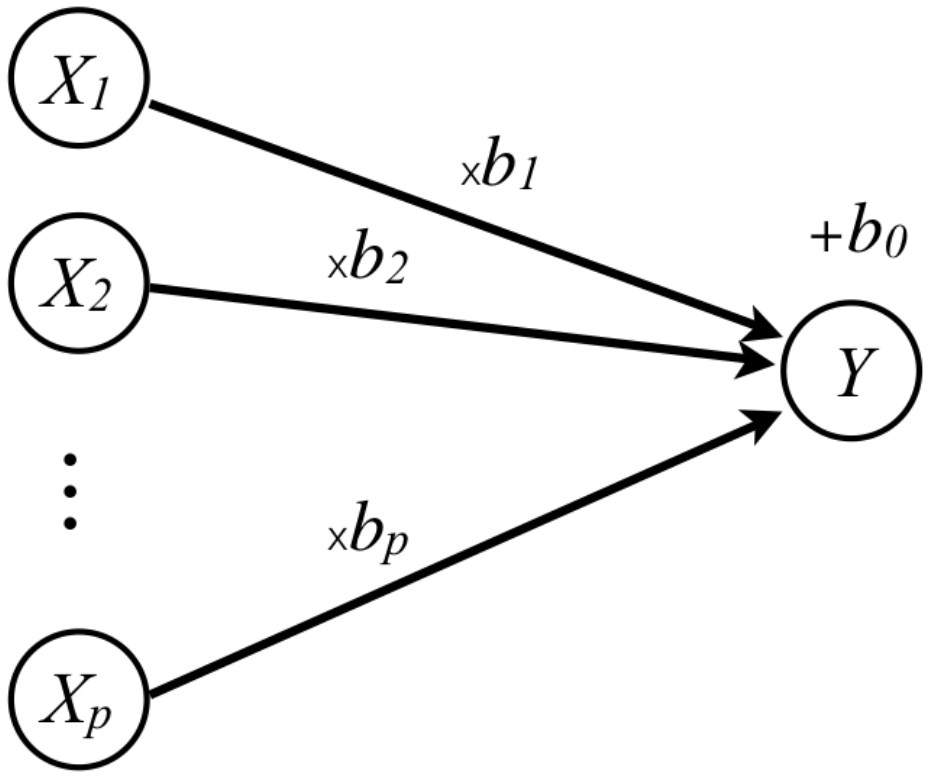
Hang on to this idea....

Linear regression as a network

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

Drawing as a network model:

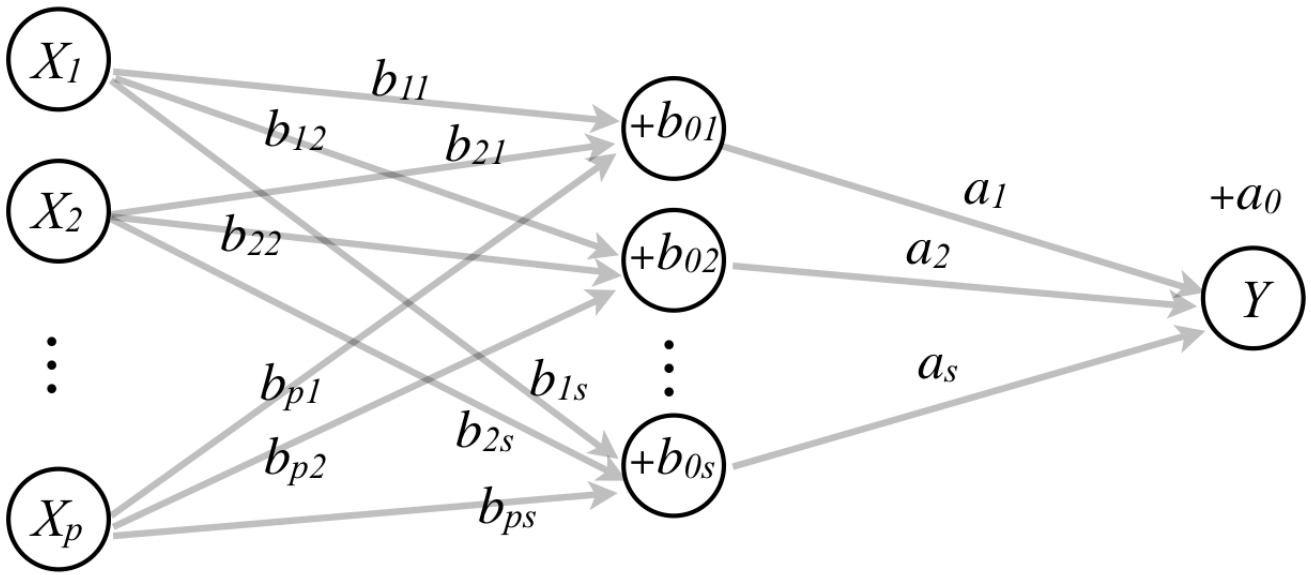
p inputs (predictors), multiplied by weights (coefficients), summed, add a constant, predicts output (response)



Network explanation - hidden layers

$$\hat{y} = \alpha_0 + \sum_{k=1}^{s}(\alpha_k(\beta_{j0} + \sum_{j=1}^{p}\beta_{jk}x_j))$$

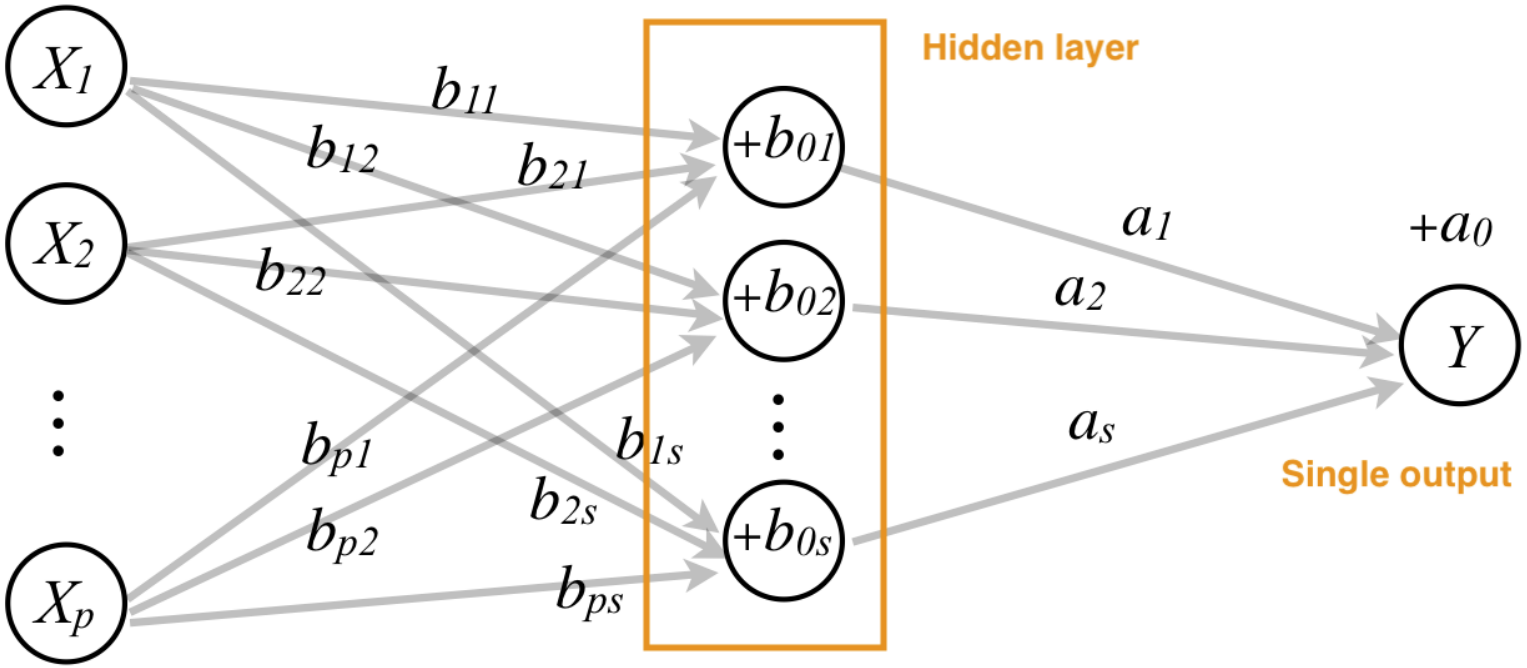
A linear regression model nested within a linear regression model allows for intrinsic dimension reduction, or expansion.



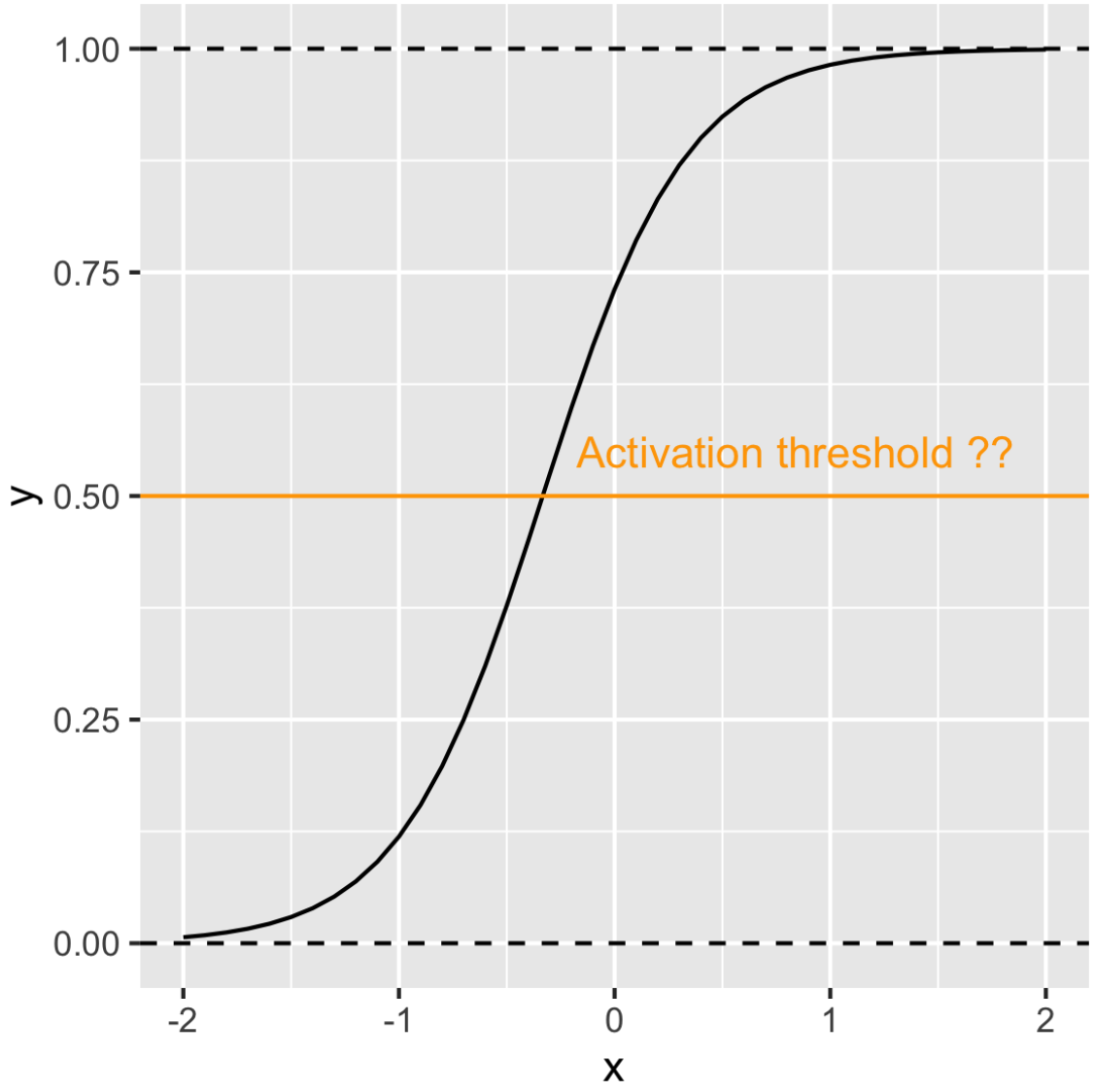
Two layer perceptron

This is a single output, 2 layer, perceptron (neural network), with a linear threshold.

$$\hat{y} = \alpha_0 + \sum_{k=1}^{s} (\alpha_k (\beta_{j0} + \sum_{j=1}^{p} \beta_{jk} x_j))$$



Back to logistic regression: When the proportion gets to threshold, it activates an event to happen $(Y = 1)$



Activation functions

$$\hat{y} = g(\alpha_0 + \sum_{k=1}^{s}(\alpha_k f(\beta_{0k} + \sum_{j=1}^{p}\beta_{jk}x_j)))$$

Let $u = \beta_0 + \sum_{j=1}^{p}\beta_j x_j$

- Logistic: $\frac{1}{1+e^{-u}}$
- Gaussian radial: $\frac{1}{\sqrt{2\pi}}e^{-u^2/2}$
- Hyperbolic tangent: $\frac{e^u - e^{-u}}{e^u + e^{-u}}$

Example

Consider the wiggly data

Overall fit

This is the best fit from many, many random starts, with different parameters. It's a beautiful fit.

Individual layers

The best fit used four nodes in the hidden layer, each fitting a logistic regression. Node 2 does the most work, catching the main linear divide, and node 1 catches the first wiggle, node 4 the second wiggle and node 3 the third wiggle.

Adding in activation

Activations determine whether the node has enough information to send a signal to the next layer. For example, the `relu` activation function is takes summed weighted inputs and transforming them to 0 (not fire) or > 0 (fire).

To add in activation information to our model in `keras`, we simply adjust our model structure previously by adding in the activation functions we would like to use for each layer.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 16,
              activation = "relu",
              input_shape = p) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
```

3. Feedback Mechanism

Compiling the model

Now that we have a model architecture in place - how will the model *learn* from the data? To do this, we need to specify a **loss function** and **optimiser** to use during training.

- The *loss function* (also called objective function) helps measure performance. For example, in regression use the MSE, and for classification you may use cross entropy.
- The *optimiser* controls which optimisation algorithm is implemented in our NN.

```
model %>%   compile(
    loss = 'categorical_crossentropy',
    optimizer = "rmsprop",
    metrics = c('accuracy')
  )
```

4. Model Training

Model training

Now that we have created the model specification, we are ready to give it some data! We can use the `fit` function in `keras` to achieve this.

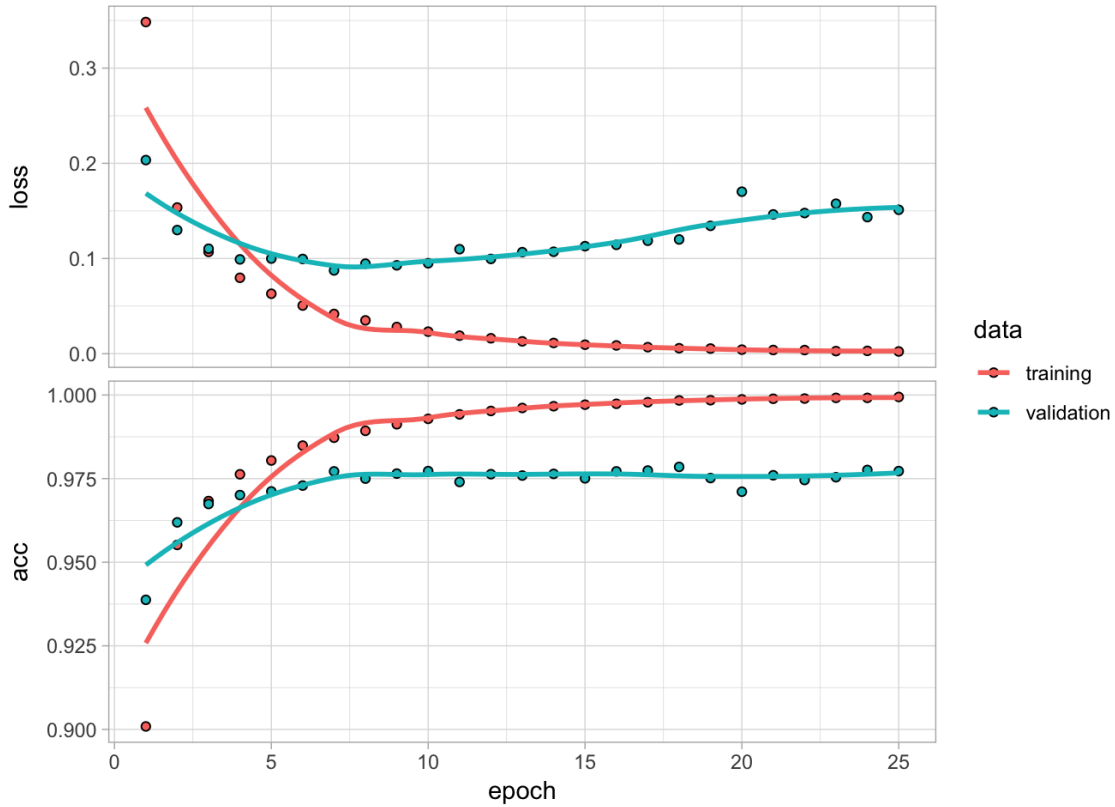
```
fit <- model %>% fit(
  x = mnist_x,
  y = mnist_y,
  batch_size = 512,
  epochs = 10,
  validation_split = 0.2
)
```

- `batch_size` refers to the number of samples used to estimate the error gradient for the optimisation at any time, manages computational load, and model stability.
- `epoch` refers to how many iterations through the entire training data, it is typically large.
- `validation_split` sets a hold-out proportion to avoid over-fitting.

Model training

We can plot the accuracy and loss of the neural network using the `plot` function.

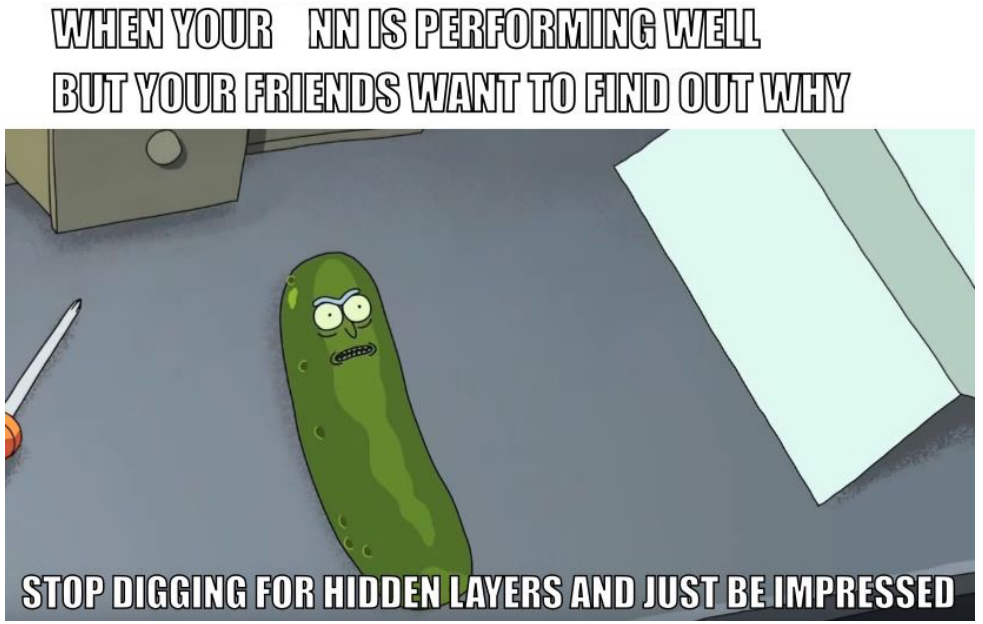
```
plot(fit)
```



So why don't we use neural networks for all machine learning problems?

Minimal interpretability

- Core concept of prediction vs inference.
- Neural networks are seen as a black box type of model, with limited information provided to as how the neural net is making decisions. (*Contrast this to trees, or logistic regression, say*)



Source: Machine Learning Memes for Convolutional Teens

Data intensive

- Deep learning algorithms don't work well when the number of features is larger than the number of observations (highly over-parameterised).
- If we only have a limited number of training data points, the model can potentially overfit and fit very closely to the training data whilst lacking predictive performance for new data.

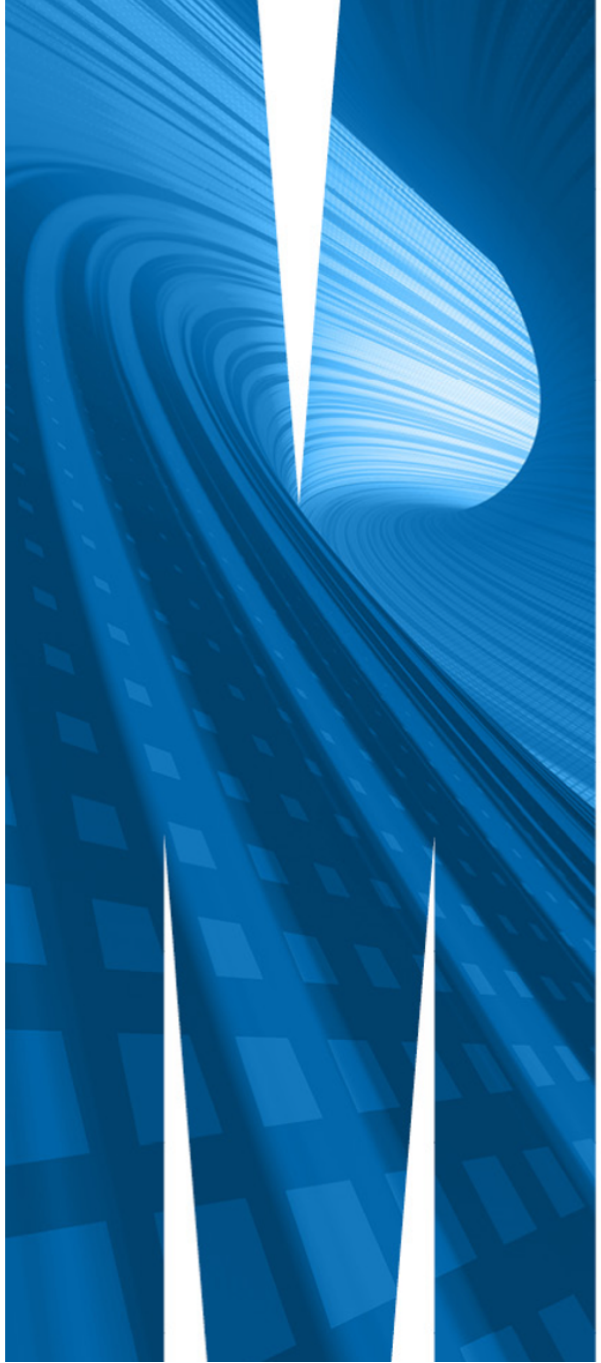
Source: Machine Learning Memes for Convolutional Teens

Computationally intensive

- Many calculations are required to estimate all of the parameters in many neural networks (the one we have shown today is quite basic).
- Deep learning involves huge amounts of matrix multiplications and other operations.
- Often used in conjuction with GPUs to paralellise computations.

Source: Machine Learning Memes for Convolutional Teens

MONASH University



Lecturer: *Professor Di Cook*

Department of Econometrics and Business Statistics

ETC3250.Clayton-x@monash.edu

Week 8a