# Machine Learning

## SICCS-Oxford 2021

Dr Thomas Robinson, Durham University

June 2021

## Hello!

Today's workshop:

- 1 hr 15 min lecture
- 15 minute break/Q&A
- 1.5 hour coding walkthrough on constructing neural networks in `R`

**Caveat**: three hours is not a lot of time!

- Introduce where I think ML is most useful in social sciences
- Equip you with some fundamental tools that can be applied across:
  - Contexts
  - Data sources
  - Algorithms

## Lecture content

Goals are threefold:

1. Brief overview of machine learning
   - What is ML?
   - Prediction problems
   - Bias-variance tradeoff
2. Building basic neural networks
   - Highly flexible, “engineering-grade" ML method
   - Now easily implementable in `R`

## What is machine learning?

## (Machine) learning and statistics

ML is a vague term:

> *"Machine learning is a subfield of* ***computer science*** *that is concerned with building* ***algorithms*** *which, to be useful, rely on a collection of examples of some phenomenon. . . the process of solving a practical problem by 1) gathering a dataset, and 2) algorithmically building a statistical model based on that dataset." – Burkov 2019*

To me, ML is defined by:

1. "Computationally-intensive" methods
2. Where researchers underspecify the relationship between variables
3. And allow the computer to search for (or learn) these relationships

## Machine learning

Expectation: I need a $1m super computer

Reality: It runs in minutes on a personal computer

Figure 1: Google: Tensor Processing Unit server rack

## Why machine learning?

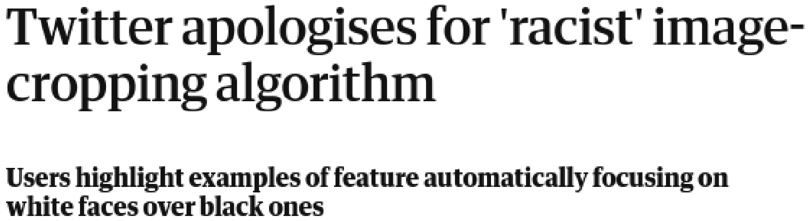
Machine learning can be:

- Powerful
- Flexible
- Reduce the burden on the researcher

It helps solve lots of **prediction problems** and can assist in **inference problems** too

But ML is not a panacea!

- ML cannot solve problems of poor research design
- And can introduce its own issues



Twitter apologises for 'racist' image-cropping algorithm

**Users highlight examples of feature automatically focusing on white faces over black ones**

## Prediction and inference

Consider the following linear model:

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{1i}$$

- Inference is concerned with estimating the size/direction of the relationship between variables ($\hat{\boldsymbol{\beta}}$ problems)
- Prediction is concerned with estimating some outcome, using the relationships between variables ($\hat{\boldsymbol{y}}$ problems)

These two facets are clearly connected:

- If we know the size/direction of the relationships, we can predict the outcome
- But we rarely know (or even pretend to know) the true model
- Sometimes we can get good at $\hat{\boldsymbol{y}}$ problems without knowing $\hat{\beta}$

## There are $\hat{X}$ problems too

We can also think about where the prediction problem lies:

- $\hat{\boldsymbol{y}}$ problems are about the dependent variable
  - To predict an election winner…
  - … or the probability of revolution…
  - … or the weather tomorrow
  - These are not necessarily inferential problems
- $\hat{\boldsymbol{X}}$ problems are about independent variables
  - Dimensions of interest that may be important to our theory…
  - … but which are not directly observable (i.e. latent)
  - We want to make predictions over $\boldsymbol{X}$ so we can test an inferential theory about the relationship between $X$ and $y$

## Bias and variance

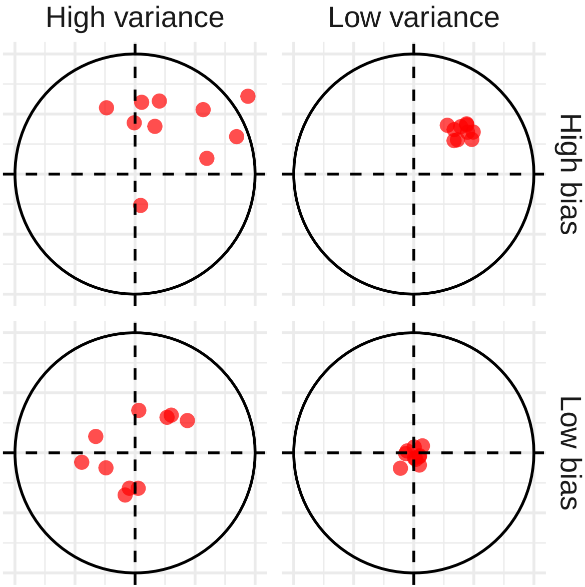
Bias is a feature of the estimator:

- $\text{Bias}_{\beta} = (\mathbb{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta})$
- With OLS under Gauss Markov assumptions, $(\mathbb{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta}) = 0$

Variance occurs due to resampling from the population:

- Parameter estimates change (slightly) as we re-estimate the model with new data
- $\mathbb{V}_{\hat{\boldsymbol{\beta}}} = \mathbb{E}\big[(\mathbb{E}[\hat{\boldsymbol{\beta}}] - \hat{\boldsymbol{\beta}})^2\big]$
- The average distance between a particular parameter estimate and the mean of parameter estimates over multiple samples

## Visualising bias and variance



## Bias-variance trade off

So can't we just choose a low-variance, low-bias modeling strategy? Not quite!

Assume we calculate the mean squared error of some new data $\boldsymbol{X'}$ given a trained model $\hat{f}$:

$$\mathrm{MSE} = \mathbb{E}[(\hat{f}(\boldsymbol{X'}) - y)^2].$$

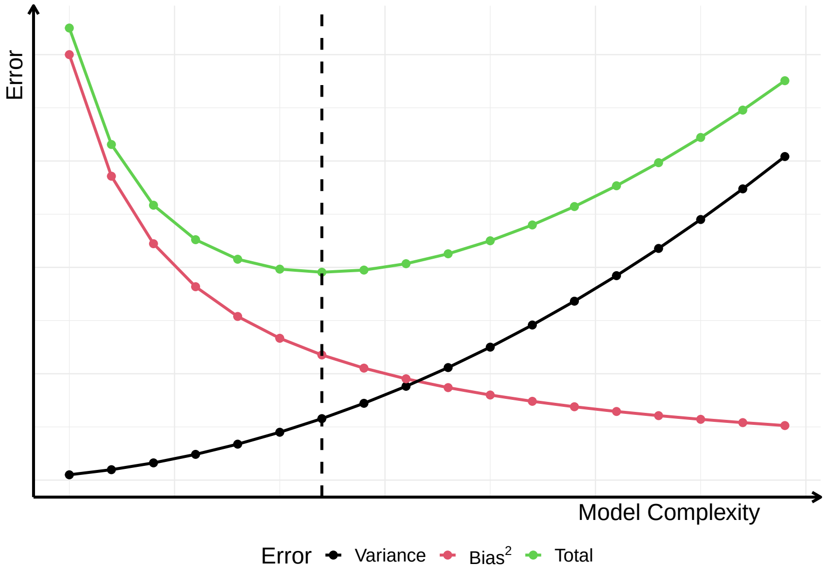
We can decompose this further:

$$MSE = \underbrace{\mathbb{E}\left[(\hat{f}(\boldsymbol{X'}) - \mathbb{E}[\hat{y}])^2\right]}_{\text{Variance}} + \underbrace{\left(\mathbb{E}[\boldsymbol{\hat{y}}] - \boldsymbol{y}\right)^2}_{\text{Bias}^2}$$

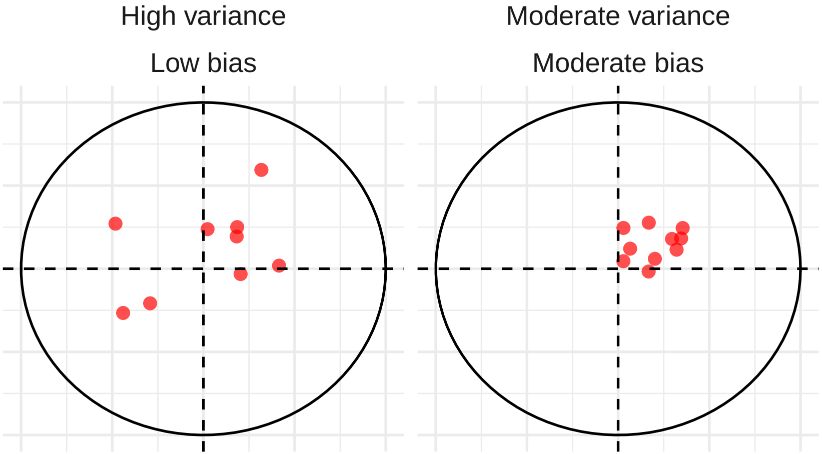
So holding the MSE fixed, if we reduce the variance we must increase the bias

- I.e. there is a **bias-variance trade-off**

## Visualising the trade-off



## A bit of bias can be useful



## Bias in ML

ML methods are typically powerful because they allow a tradeoff between variance and bias:

- We do this by “regularizing” our estimator
- Good for prediction
- Bad for inference (in simple applications)

A nice introduction to bias, regularisation and ML is provided in:
*Kleinberg et al (2015). Prediction Policy Problems, AER.*

## Treatment effect estimation and neural networks

## Effect heterogeneity

Suppose we have 8 observations of an outcome, treatment assignment and two covariates:

| y | d | Gender | Education |
|---|---|---|---|
| 12 | 1 | Female | High |
| 13 | 1 | Female | Low |
| 5 | 0 | Female | High |
| 6 | 0 | Female | Low |
| 7 | 1 | Male | High |
| 8 | 1 | Male | Low |
| 7 | 0 | Male | High |
| 6 | 0 | Male | Low |

Table 1: Observed

| y | d | Gender | Education |
|---|---|---|---|
| ? | 0 | Female | High |
| ? | 0 | Female | Low |
| ? | 1 | Female | High |
| ? | 1 | Female | Low |
| ? | 0 | Male | High |
| ? | 0 | Male | Low |
| ? | 1 | Male | High |
| ? | 1 | Male | Low |

Table 2: Unobserved counterfactual

$$\text{ATE}_{\text{Observed}} = 10 - 6 = 4$$

*The ATE may mask considerable heterogeneity*

## Conditional Average Treatment Effect

We can break down our ATE into conditional estimates:

$$\text{CATE} = \mathbb{E}[Y|d = 1, X = a] - \mathbb{E}[Y|d = 0, X = a]$$

In particular, we can think about estimating the individual level effect, i.e.

$$\text{ITE} = [Y_i|d = 1] - [Y_i|d = 0]$$

- This is not an average
- Conventionally impossible to estimate given **fundamental problem of causal inference**

Prediction problem:

- For those treated (control) units, what would they have done under control (treatment)?

## Enter the neural network

What we need is a flexible way of modelling the potential relationship between $\mathbf{y}$, $\mathbf{d}$ and $\mathbf{X}$:

- Neural networks offer one such approach
- Very popular in industry
- Recently a lot more accessible in `R`
- Other methods available:
  - Random forests
  - BART
  - Best subset modelling

## Simple perceptron model

Suppose we have a single vector of input data $\boldsymbol{x}$, and we want to predict the output $\boldsymbol{y}$

A simple perceptron model looks like the following:

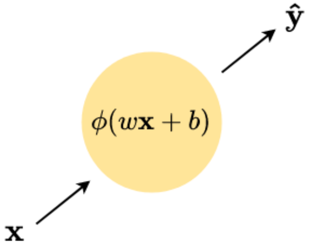


Figure 2: Single node, single layer perceptron model

$w$ is the weight term and $b$ is the bias term – in this simple case, both are scalar.

## Activation functions $\phi$

The activation function is simply a function applied to the result of $w\boldsymbol{x} + b$, that controls the range of the output vector

$\phi$ may simply be the **identity function**:

- I.e. $\phi(\boldsymbol{x}) = \boldsymbol{x}$

**Sigmoid function**:

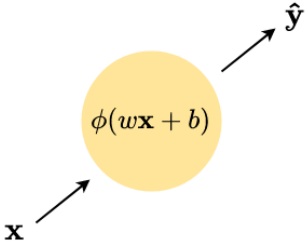
- $\phi(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{x}}}$

**Rectified Linear Unit (ReLU)**:

- $\phi(\boldsymbol{x}) = \max(0, x)$

These functions (and others) are particularly useful because they have known derivatives – which we'll return to later!

## Gaining a prediction from our simple model



Suppose:

- $\phi$ is the ReLU function
- $\boldsymbol{w} = \mathbf{2}$, $b = 1$

And we observe the following input vector $\boldsymbol{x}$:

$$\begin{bmatrix} 5 \\ 1 \\ -1 \end{bmatrix}$$

What is $\hat{\boldsymbol{y}}$?

## Multiple inputs

The first model is very basic, so we can adapt it to accept **multiple** inputs:

- Let $k$ index input variables, i.e. $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$
- Let $\boldsymbol{w}$ be a vector of weights, i.e. $\boldsymbol{w} = \{w_1, \ldots, w_k\}$

Inside our activation function we replace $w\boldsymbol{x} + b$ with

$$w_1\boldsymbol{x}_1 + \ldots + w_k\boldsymbol{x}_k + b \equiv \sum_k w_k\boldsymbol{x}_k + b$$

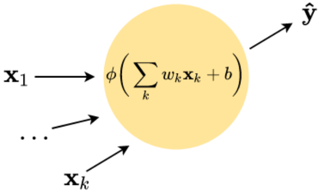


Figure 3: Single node, multiple input perceptron model

## Initialisation and training

We need to set up a network structure, prior to feeding in our data.

For a single-node perceptron model with $k$ inputs, that means instantiating the weights and biases

- A naive option sets $\boldsymbol{w} = \mathbf{0}$
  - This is rarely optimal – it can lead to significantly slower convergence (and can even disrupt convergence entirely)

A now standard approach is to use **Xavier initialisation** where:

$$w_k \sim \mathcal{N}(0, \frac{1}{k})$$

- where $k$ is the number of inputs to the node
- Typically used when $\phi$ is tanh or sigmoidal
- Bias terms are instantiated at zero

## Loss functions

The goal of the perceptron is to minimise the predictive error between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}} = \phi(\sum_k w_k \boldsymbol{x}_k + b)$

Depending on the type of prediction problem, we want to use a different function:

**Continuous $\boldsymbol{y}$**

- $\phi$ will be linear or ReLU
- Minimise the mean squared error
- I.e. $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

**Binary $\boldsymbol{y}$**

- $\phi$ will be sigmoid
- Minimise using **cross-entropy** loss function
- I.e. $= -\frac{1}{N}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{ic}\log(\hat{y_{ic}})$
  - where $c$ indexes the classes within the binary/categorical variable

## OLS/Logistic regression as a single-layer perceptron

We can construe OLS as a single-node perceptron model,

$$\boldsymbol{y} = \phi(b + w_1 \boldsymbol{x_1} + ... + w_k \boldsymbol{x_k}),$$

when:

- $\phi$ is the identity function
- $\boldsymbol{w}$ is the regression coefficient vector
- $b$ is the intercept

and solved via MLE.

Similarly logistic regression is where $\phi$ is the sigmoid activation function.

## Limitations and extensions

A single-node perceptron model is not particularly exciting:

- With identity/sigmoid activation functions we get conventional estimators
- The model is linear in inputs

To complicate our models we need to think about creating a **network** of nodes

- Increase the number of computational units
- Determine the flow of information along the network

## Deep learning

## Complicating the network

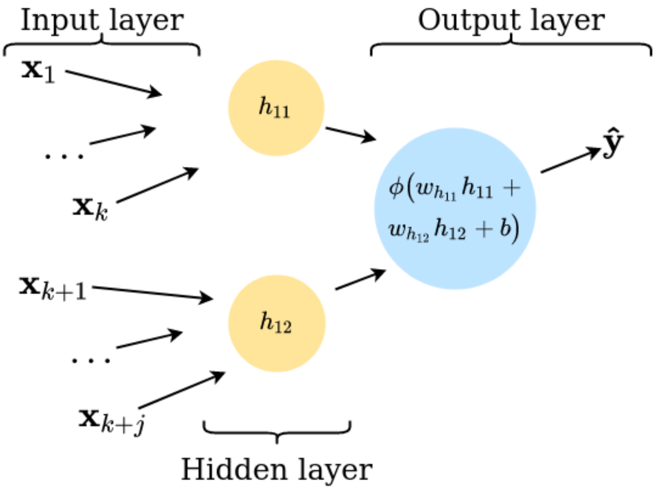


Figure 4: Multi-layer (but not deep) network

## Deep neural network

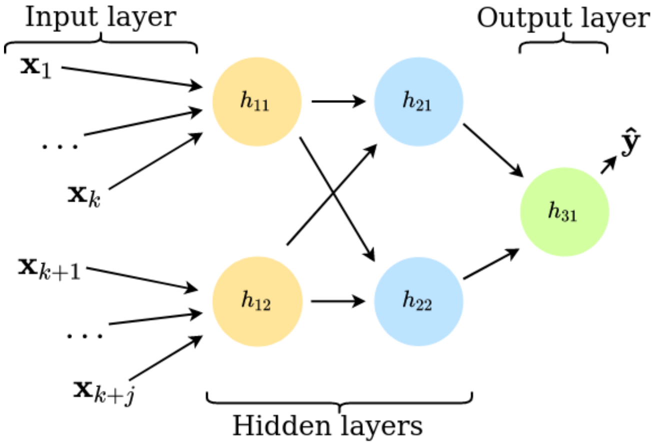


Figure 5: Multi-layer **deep** network

## Multi-layer network notation

The computation of outputs through layer $h$ of a neural network is:

$$\mathbf{y}^{(h)} = \sigma(\mathbf{W}^{(h)}\mathbf{y}^{(h-1)} + \mathbf{b}^{(h)}),$$

where:

- $\mathbf{y}^{(h)}$ is a vector of outputs from layer $h$
- $\mathbf{W}^{(h)}$ is a matrix of weights for layer $h$
- $\mathbf{b}$ is a vector of biases for layer $h$
- $\sigma$ is an activation function

This model can be generalized to an arbitrary number of hidden layers $H$:

$$\mathbf{y} = \Phi(\mathbf{W}^{(H)}[...[\sigma(\mathbf{W}^{(2)}[\sigma(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})] + \mathbf{b}^{(2)})]...] + \mathbf{b}^{(H)}),$$

where $\mathbf{x}$ is a vector of inputs and $\Phi$ is a final-layer activation function.

## Fully-connected networks

In a fully connected network:

- Every output from layer *h* is an input to every node in layer $h + 1$

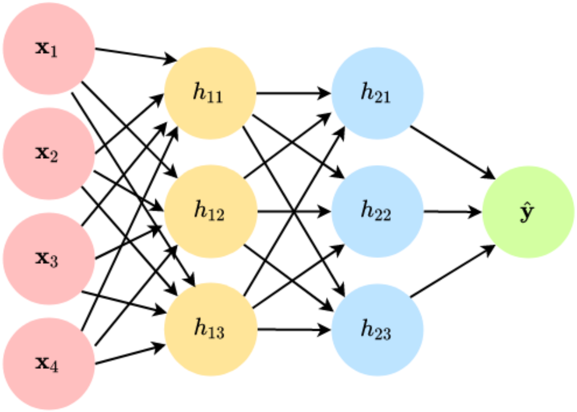


Figure 6: Fully-connected neural network

## Feed-forward training

We initialise a multi-layer model like a single-layer model:

- Set weight terms for each node within each layer via (Xavier) initialisation

During training, an **epoch** consists of:

1. Feeding every observation through the model
    - When there are no cycles in the network, this is called "feed-forward"
2. Calculate the loss associated with the prediction
3. Adjust weights and biases based on the **gradient** of the loss
    - This is complicated with multiple layers
    - Adjusting the weights and bias affects the output of a node
    - . . . and the input of the nodes (plural!) that it feeds into!
    - This process is called **backpropagation**

## Estimating treatment effect heterogeneity

1. Train a neural network model on experimental data
2. Use trained model to predict counterfactual outcomes
   - Invert treatment assignment
   - Keep all covariates the same
3. Estimate ITE

$$\begin{pmatrix} \widetilde{\mathbf{y}}_{\mathbf{i,d=1}} \\ 14 \\ 12 \\ 12 \\ 13 \\ 7 \\ 7 \\ 6 \\ 7 \end{pmatrix} - \begin{pmatrix} \widetilde{\mathbf{y}}_{\mathbf{i,d=0}} \\ 7 \\ 7 \\ 4 \\ 6 \\ 8 \\ 6 \\ 8 \\ 6 \end{pmatrix} = \left(\begin{array}{c|cc} \textbf{ITE} & \textbf{Gender} & \textbf{Education} \\ 7 & \textit{Female} & \textit{High} \\ 5 & \textit{Female} & \textit{Low} \\ 8 & \textit{Female} & \textit{High} \\ 7 & \textit{Female} & \textit{Low} \\ -1 & \textit{Male} & \textit{High} \\ 1 & \textit{Male} & \textit{Low} \\ -2 & \textit{Male} & \textit{High} \\ 1 & \textit{Male} & \textit{Low} \end{array}\right)$$

4. Examine how ITE varies across covariates