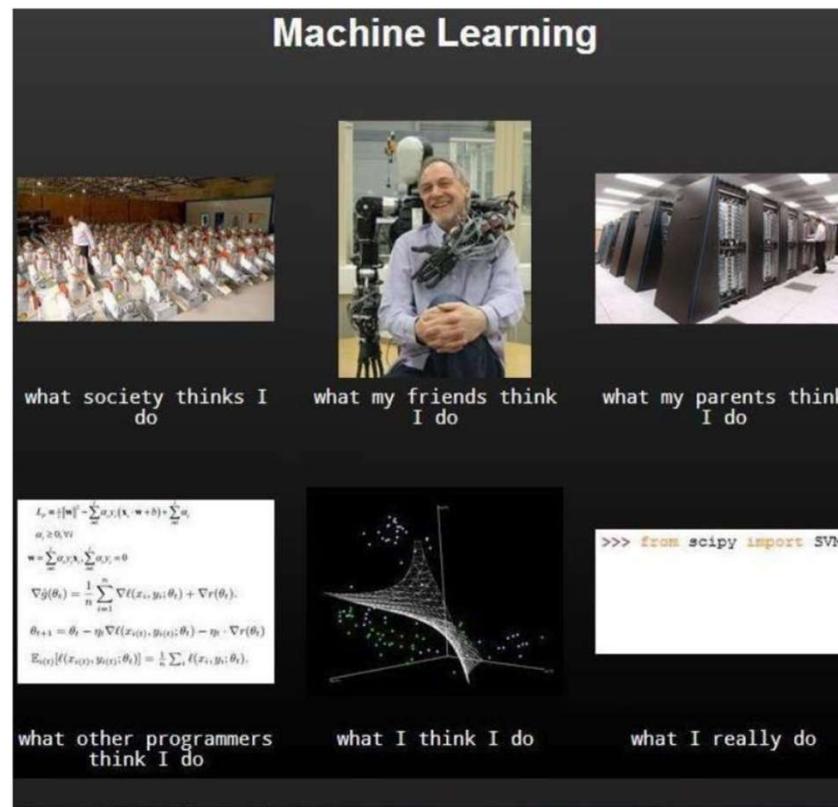


CAB420: Machine Learning Basics

WITH PICTURES AND VIDEOS

What is Machine Learning?

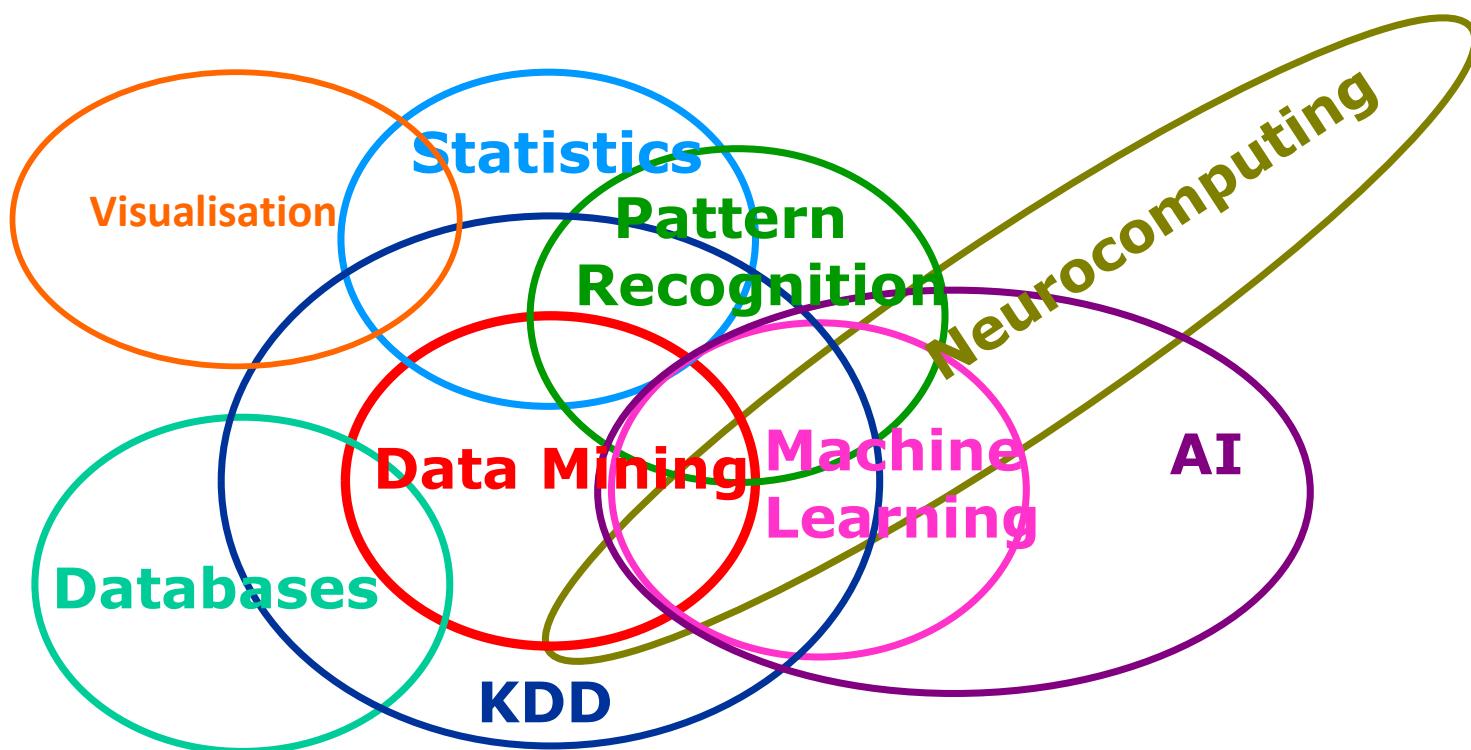


What is Machine Learning?

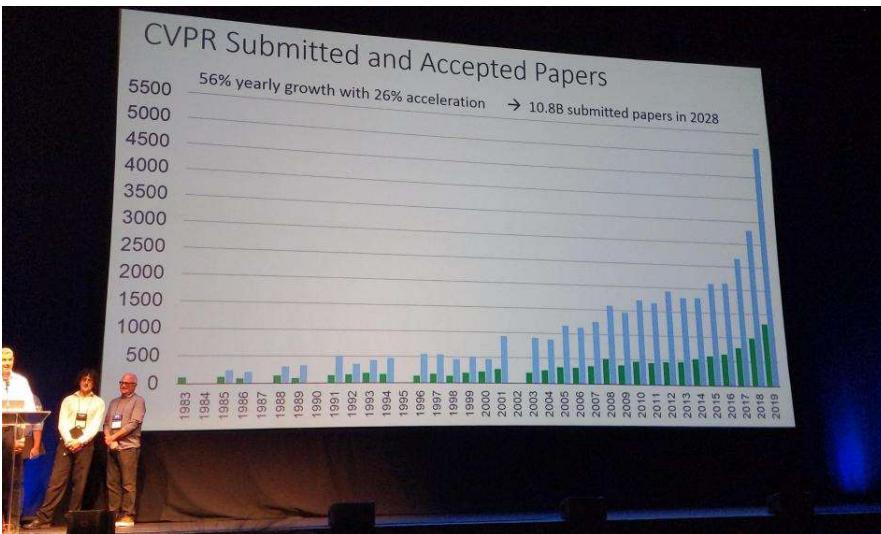


From XKCD.

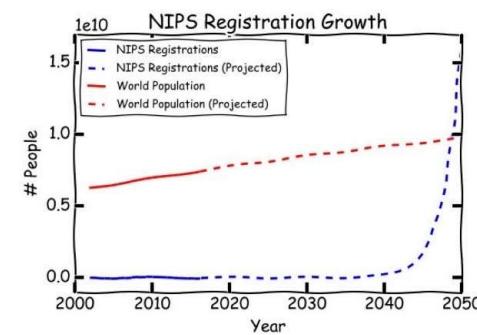
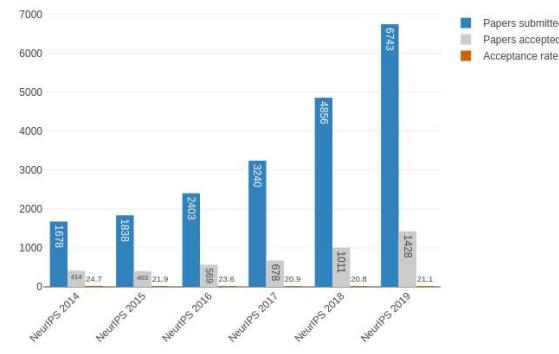
A Multidisciplinary Field



A Growing Field

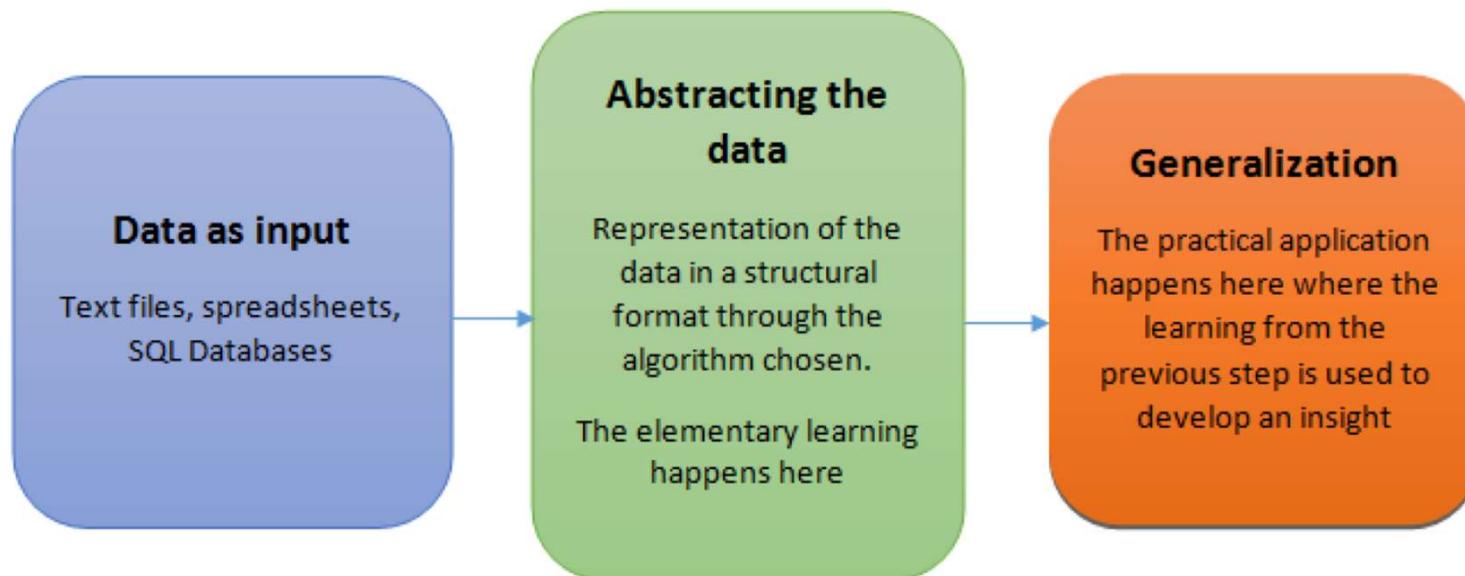


Statistics of acceptance rate NeurIPS

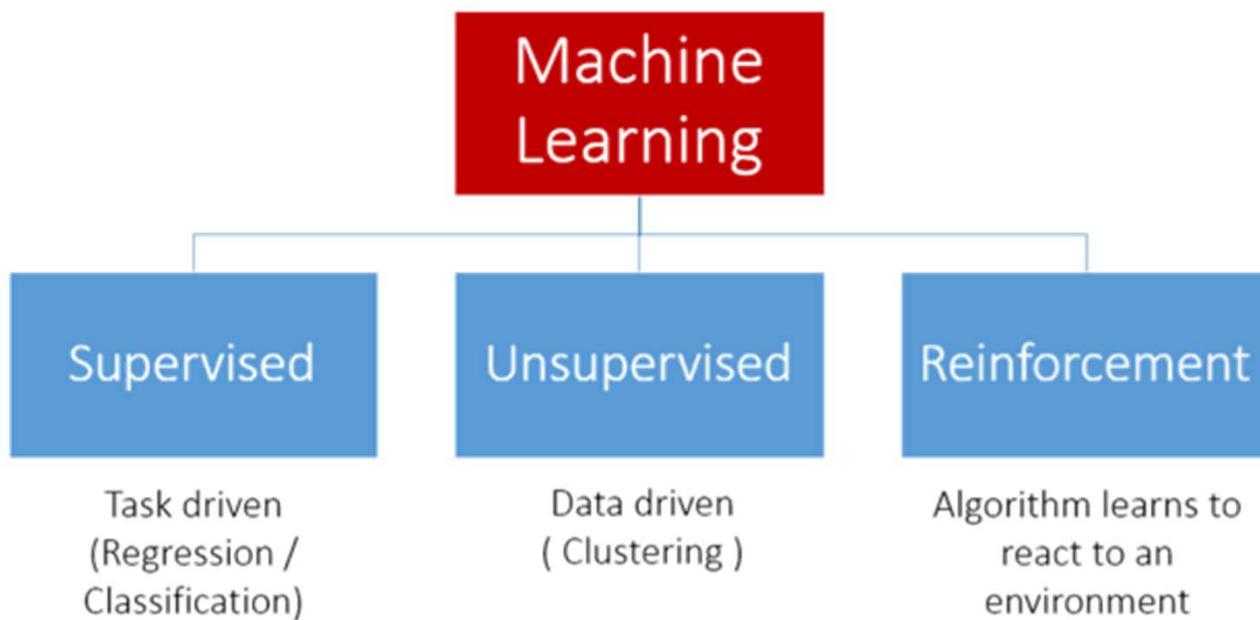


How do we teach machines?

Broadly, there are three steps



Types of Machine Learning

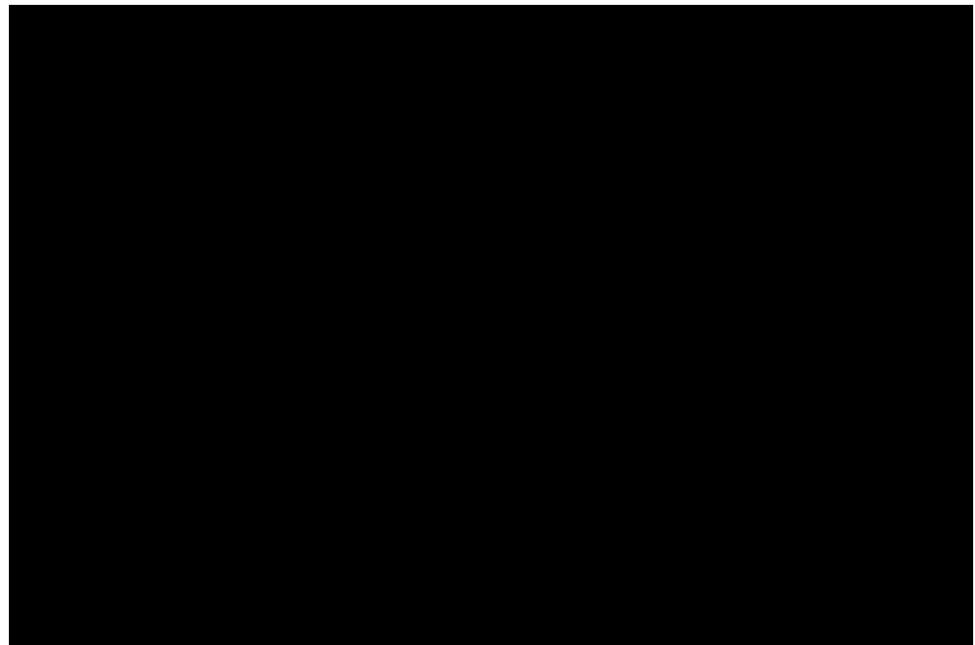


An Example: Regression

Regress from image features

- Size, texture, shape

to a person count



An Example: Clustering

Group pedestrians by the way that they move

- Using information extracted that describes movement



GVEII Dataset

An Example: Supervised Learning

Tracking people with supervised models to:

- Detect people
- Learn how people move (path prediction)

Optimisation task

- Assignment of detections to tracks
- Hungarian Algorithm



3D MOT 2015 BENCHMARK
PETS09-S2L2

An Example: Unsupervised Learning

Detect abnormal behaviours

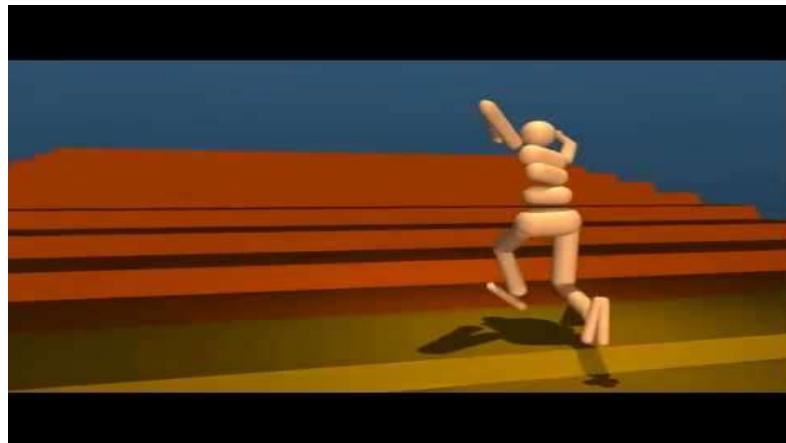
- Learn from all data, data points that are rare, are abnormal



An Example: Reinforcement Learning

Learning to walk and run

- Learns from experience
- We're not covering reinforcement learning in this subject



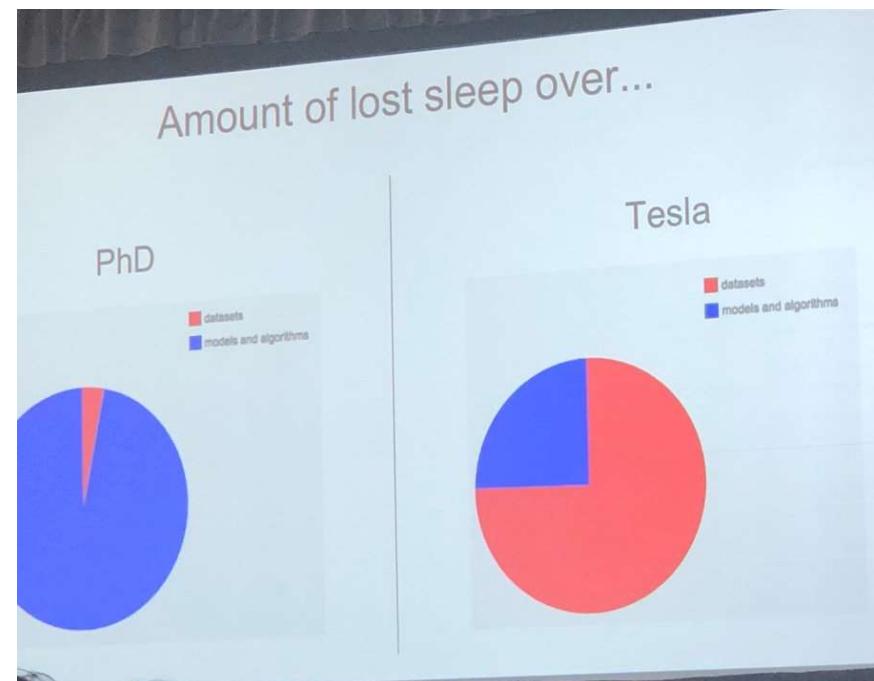
Video from Google DeepMind

Types of Machine Learning

- Supervised
 - We have labelled data
 - For each example, we know what the answer should be
- Unsupervised
 - No labelling (though we may have prior knowledge)
 - Knowledge discovery
- Semi-Supervised
 - Only some (usually a small amount) of our data is labelled
 - Half-way between supervised and unsupervised
 - Learn from both labelled and unlabelled data
- Reinforcement
 - Learn through interactions with the environment
 - Not covered in CAB420

Data is Important

- In research we care more about the algorithm
- In the real world, a well prepared dataset is often more important



Slide from a presentation given by Andrey Karpathy,
Director of AI at Tesla

Garbage In -> Garbage Out

- Machine learning needs data
 - But that data also needs to be collected properly
- Poor data will lead to poor models
 - No amount of fancy modelling will compensate for errors in annotation, or a poorly collected dataset

Bias and Imbalance in Data

- Models can only learn from what we provide them
 - If we provide biased data, models will learn those same biases
 - Becoming an increasing problem as we place more trust in machine learning to make decisions
- Google Photos example
 - Classified African Americans as Gorillas
 - <https://www.forbes.com/sites/mzhang/2015/07/01/google-photos-tags-two-african-americans-as-gorillas-through-facial-recognition-software/#1834204d713d>
- Amazon Recruitment tool example
 - Biased against women
 - <https://www.theguardian.com/technology/2018/oct/10/amazon-hiring-ai-gender-bias-recruiting-engine>

Data splitting for machine learning



Data splitting for machine learning

- Training
 - Data that we use to train the model
- Validation
 - Data that we use to tune model parameters. Separate to the training data.
- Testing
 - Unseen data. Emulates the “real-world” data we want to apply our model to.
- Some fields may switch the role of validation and testing

Holdout Validation

- Data is “held out” for validation and/or testing
- This is usually the ideal approach
 - Training, validation and testing are all totally separate
 - Requires a large enough amount of data to be able to split it

What if we don't have enough data?

- Cross-Fold Validation
 - Don't create a separate validation dataset
 - Dynamically split the training dataset into training and validation (say 80/20)
 - Repeat with 5 “folds”, so that each 20% ends up being in the validation set
 - The best model on average over all folds is the one we choose

What if we don't have enough data?

- And how much data is enough?
- Machine Learning may not be possible if we don't have enough data
 - Our data needs to contain whatever patterns, relationship, etc, we seek to model
 - For some problems, this may be a couple of hundred samples, for others it may be millions
- Extrapolating beyond our data can be problematic
 - Needs lots of data to build a good model to allow for extrapolation
 - Hard to extrapolate from very few samples



IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Cartoon from XKCD

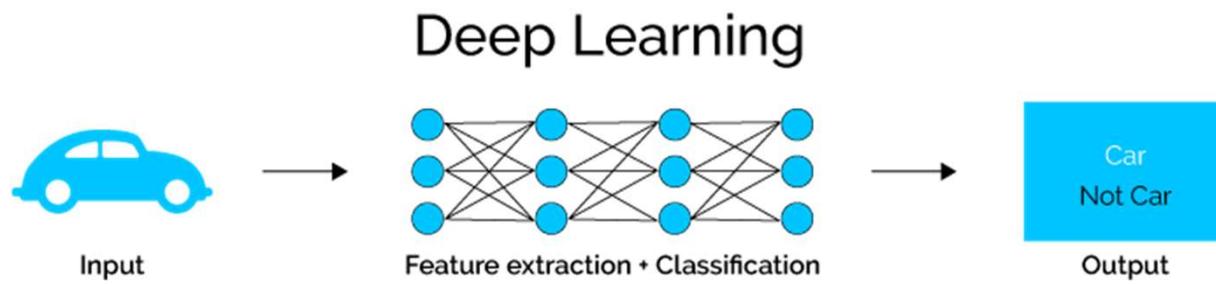
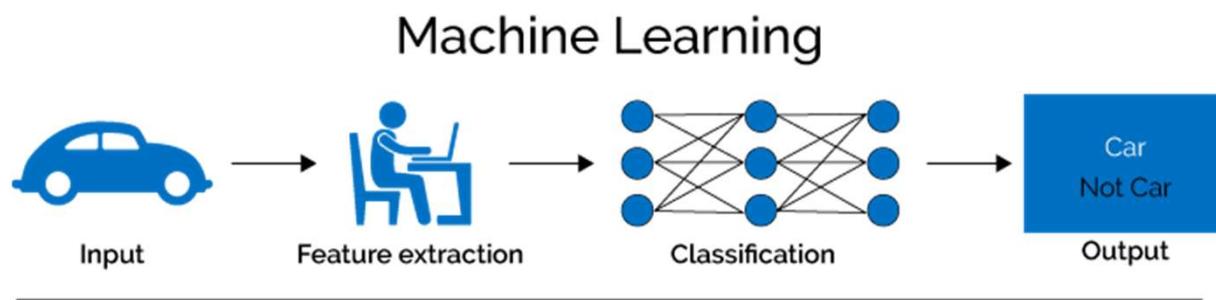
The Curse of Dimensionality

- Datasets will often have
 - Lots of dimensions (i.e. lots of columns, observed variables)
 - Not that many samples (i.e. few rows)
- Using all dimensions in this case may not be possible
 - Too many variables and too few samples will lead to overfitting
 - Feature space becomes sparse
 - All points are far away from each other in the N-dimensional space
- For every variable you add, you need more data
 - How much data depends on the type of classifier or learning algorithm being used.
 - Keep your models simple

CAB420: Traditional ML vs Deep ML

THE SAME, BUT DIFFERENT

Traditional vs Deep



“Traditional” Machine Learning

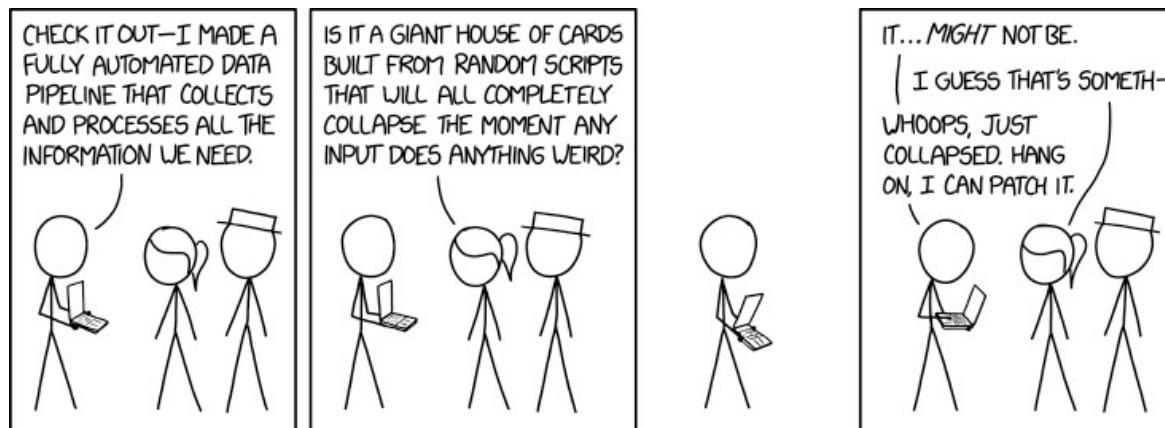
- Basically, this just refers to a time before deep learning
 - About 2012/13 and earlier
- Common Pipeline:
 - Pre-processing
 - Resizing, filtering, noise reduction
 - Feature extraction
 - MFCC (audio), HOG (image/video), Bag-of-Words (text)
 - Informed by the task
 - Do need to recognise shapes in images? If yes, select a technique that captures edge/gradient information
 - Am I using audio? Perhaps something that pulls our short-term frequency based features?
 - Machine Learning
 - Pass extracted features to the chosen learning technique

“Traditional” Machine Learning

- Feature extraction based on domain knowledge
 - Leads to “Feature engineering”: finding the optimal features for a problem/dataset
 - Can be a tedious, iterative process
 - Requires domain knowledge to extract the “best” features
- Different tasks require quite different formulations
 - Audio and Image tasks have totally different pipelines, even if the machine learning model is the same

“Traditional” Machine Learning

- Multiple steps within the pipeline for a single problem
 - May need to extract multiple sets of features
 - Machine learning component is separate
 - Lots of places for things to go wrong



Cartoon from XKCD

“Traditional” Machine Learning in CAB420

- We will look at some "traditional" methods
 - Regression, SVMs, Random Forests, etc.
- We will take a very limited look at feature extraction
 - This is a highly domain and data specific process
 - We will briefly cover some methods as they relate to specific data and examples

Neural Networks

AND HOW WE GOT TO DEEP LEARNING

Neural Networks

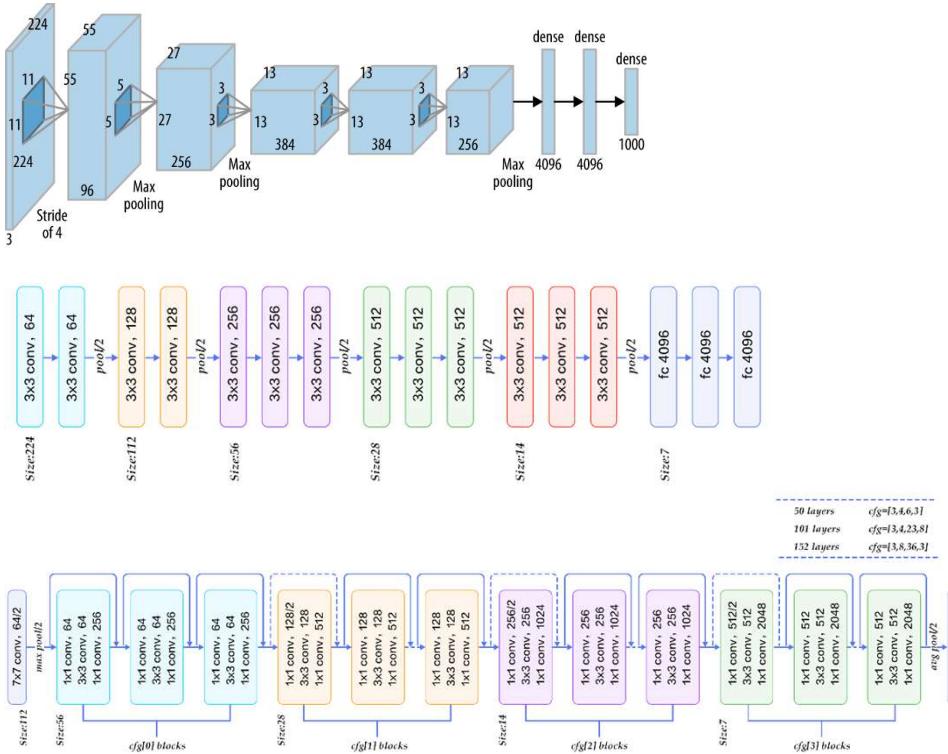
- A computer system modelled on the human brain and nervous system
 - Built on simple, stacked operations
- Typically have
 - High numbers of parameters
 - Lots of interconnections
- Data propagates through the networks
 - Input -> Intermediate Layers -> Output
 - Broadly similar to how we think the brain works

History

- Initial research started in the 1940's and 50's
 - Perceptron developed in 1958
 - First learnable networks soon after
- Ongoing development through the 1980's and 90's
 - Backpropagation proposed in 1986
 - Allowed learning of complex, multi-layer, networks relatively easily
 - Recurrent Networks
 - LSTM proposed in 1997, now a mainstay of deep learning methods
 - Methods on the periphery of machine learning and optimisation
- Deep Learning
 - Emerged in the late 2000's
 - Gained widespread attention in 2012 with "Alexnet"
 - Now the dominant machine learning method across most domains

How Much Depth?

- A few years back the focus was making things as deep as possible
 - AlexNet, 8 layers
 - VGG19, 19 layers
 - ResNet, 50 to 152 layers (depending on which variant you used)
- The obsession with depth has somewhat passed now
 - Benefits diminish as we get deeper, i.e. going from 3 to 15 layers gives a huge gain; going from 100 to 500 doesn't add much.



More Deep is More Better?

- More depth gives
 - Ability to learn higher level features, so a richer representation
 - Typically more parameters, so more representative power
- But we also get
 - More parameters, which is harder to learn, and more likely to overfit
 - More layers, further for gradients to propagate
 - Harder to fit networks in memory
 - Need specialist hardware, and very long times to train
 - Not a problem for Google, a pain for the rest of us

Deep Learning Pipeline

- Common Pipeline:
 - Pre-processing
 - Resizing, filtering, noise reduction
 - Input images (usually) need to be the same size
 - Deep Learning
 - Pass raw data to network
 - Let the network learn it's own representation and make the decision in one pass

Deep Learning: Pros

- Easy to extend model to multiple tasks, or adapt to new tasks
 - Change the loss function; or
 - Add an extra loss
 - Fine tune models
- No feature engineering
 - Let the model learn the features
 - Can lead to more robust representations
- State of the art performance for most (all?) tasks

Deep Learning: Cons

- Models are huge compared to “traditional” approaches
 - Millions vs Hundreds of parameters
 - Requires more data, runtime, memory
 - Models are harder to interpret
 - Explainable AI. Why did my model make this decision?
- No feature engineering, instead we have network engineering
 - What layers, how many, what parameters, etc.
 - Can be very slow to iterate over these decisions, or simply may not be possible to determine a truly “optimal” solution

CAB420: Linear Regression

OUR FIRST ML APPROACH

Simple Linear Regression

- Often, we want to find a relationship between two variables to:
 - Predict behaviour
 - Explore relationships
- The simplest way to do this is to assume a linear relationship

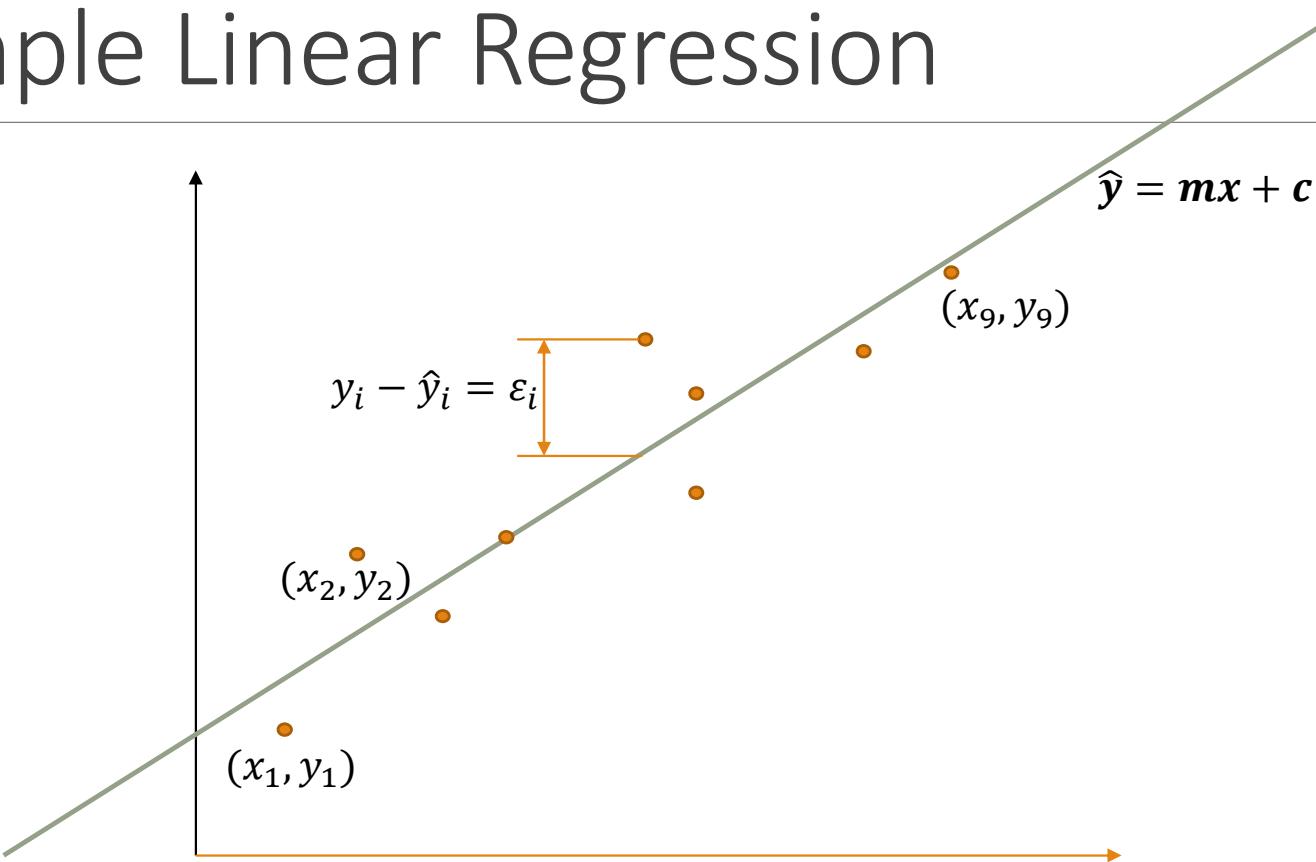
$$y = \beta_0 + \beta_1 x$$

- Often, this assumption is reasonable and a good starting point.

Simple Linear Regression

- For points (x, y) on a line $y = mx + c$ we have the relationship
 - $y_1 = mx_1 + c$
 - $y_2 = mx_2 + c$
 - ...
 - $y_n = mx_n + c$
- In practice, this is rarely (if ever) a perfect fit to our data
- But what if our data aren't all on a straight line together?
- We propose that our points should be on a line.
 - $\hat{y} = mx + c$

Simple Linear Regression



Simple Linear Regression

- Point i in our regression line is given by

$$y_i = mx_i + c + \varepsilon_i$$

- The (vertical) distance to where a point *should* be, \hat{y}_i , according to the straight line model, and where it *is*, y_i , is called the **error**, ε_i . We assume $\varepsilon_i \sim N(0, \sigma^2)$
- We want to minimise the distance from all the observed y_i to all the \hat{y}_i
- Find the values of c and m that minimise the sum of the square of the errors,

$$\sum_{i=1}^n \varepsilon_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - (mx_i + c))^2$$

Simple Linear Regression

- **Simple** Linear Regression
 - We have only one \mathbf{x}

$$y = \beta_0 + \beta_1 x_1$$

- Need to estimate β_0 and β_1 given a set of observations, $\mathbf{y} = (y_1, \dots, y_n)$ and $\mathbf{x} = (x_1, \dots, x_n)$

Simple Linear Regression

- From this, we can write a set of equations:

$$y_1 = \beta_0 + \beta_1 x_1$$
$$y_2 = \beta_0 + \beta_1 x_2$$

$$\cdots$$
$$y_n = \beta_0 + \beta_1 x_n$$

- Note that our number of samples should be much larger than the number of terms we are trying to predict.

Simple Linear Regression

- We expect each prediction to have a small error (i.e. our model won't be perfect)

$$y_i - \beta_0 - \beta_1 x_i = \varepsilon_i$$

- ε_i is the residual

- We assume:

- Residuals are independent, identically distributed random variables
- Residuals follow a Gaussian distributed around 0

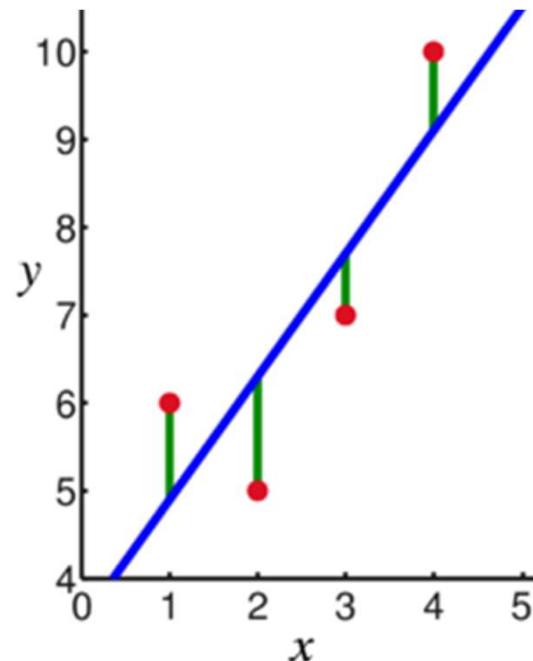
- From this, we can solve directly for β_0 and β_1

How is this done?

- Least Squares Fitting:
 - We try to find the line that minimises

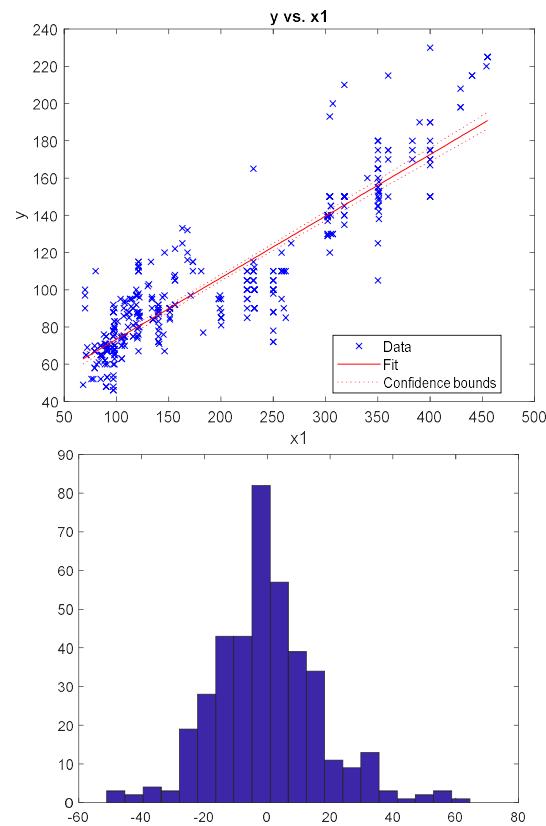
$$S = \sum_{i=1}^n y_i - y'_i$$

- y_i are the true values
- y'_i are the predicted values



Least Squares Fitting

- Given this we expect
 - Our fitted line to go “through the middle” of the points
 - Our errors (the residuals) to be symmetric
- Note that this has a closed form solution
 - i.e. can be solved directly



Correlation

AND HOW IT CAN HELP US WITH REGRESSION

A Definition: Correlation

noun: a mutual relationship or connection between two things

Or

- How much two things go up or down together

Measures a linear relationship between two data series

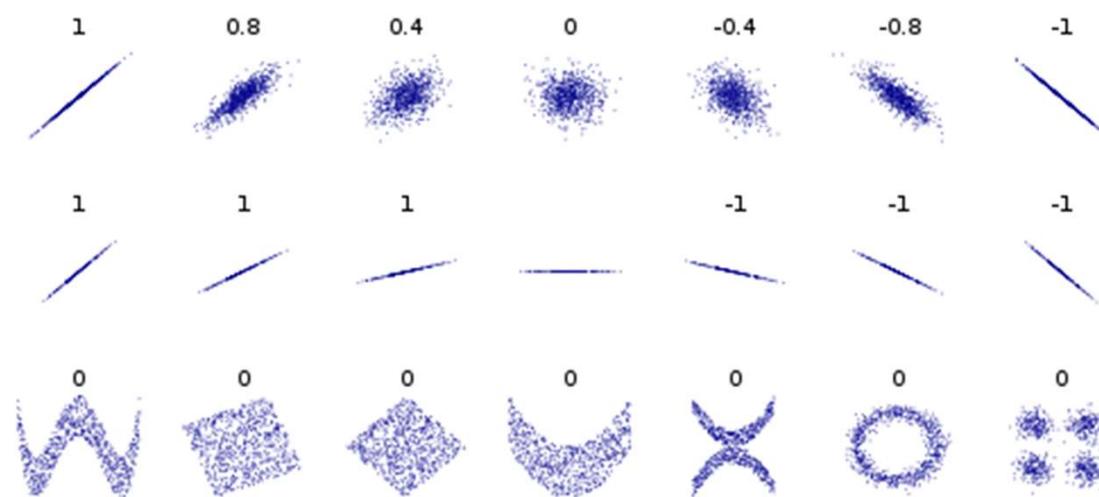
- If things are correlated, we can use regression to predict one from the other

Correlation - Formulation

- We're considering Pearson's Correlation Coefficient
- We first need to know our co-variance
 - $\sigma_{xy} = \frac{1}{N} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y)$
- From which we can derive the correlation
 - $\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$
- We can also get the correlation for a sample (rather than the population)
 - $s_{xy} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$
 - $r_{xy} = \frac{s_{xy}}{s_x s_y}$

Correlation Coefficient

- Ranges from -1 to +1
 - -1, as one variable increases the other decreases
 - 0, statistically independent
 - +1, increase and decrease together



Anscombe's quartet

- Four sets of (x, y) data
 - Means of $x = 9$
 - Means of $y = 7.5$
 - Standard deviations of $x = 3.3$
 - Standard deviations of $y = 2.0$
 - Correlations = 0.81

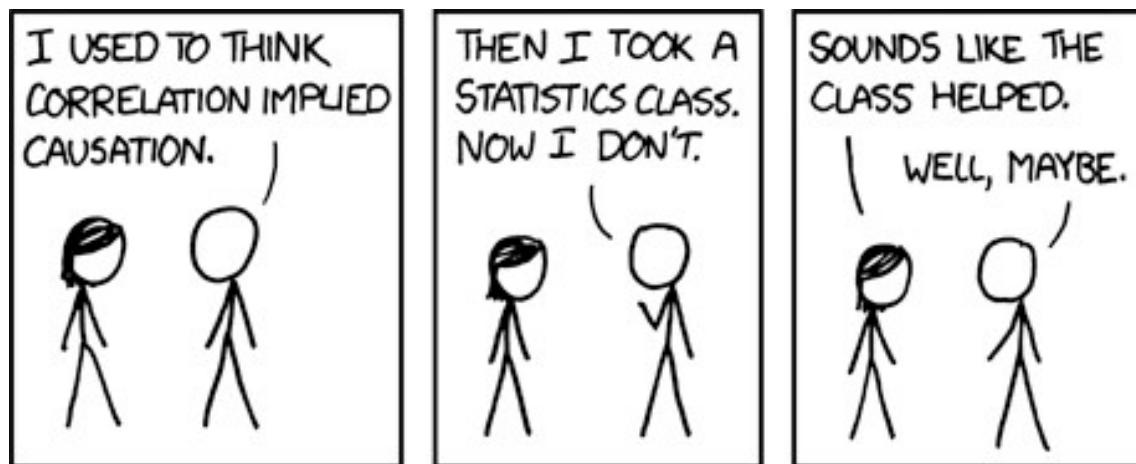
Anscombe's quartet

y

x

Correlation vs Causation

From XKCD



Correlation: Why do we care?

- Ideally, we want
 - Our predictors to be correlated with the response
 - i.e. there is a linear relationship there to be modelled
 - Our predictors to be uncorrelated with each other
 - i.e. each predictor models a different aspect of the overall relationship
- If we have correlated predictors, we can end up with redundancy in the model and unexpected p-values

Multiple Linear Regression

PREDICTING ONE THING FROM SEVERAL THINGS

Multiple Linear Regression

- Still fitting a line to some data
 - Just in multiple dimensions
- Our line is now:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

or

$$y = \beta_0 + \sum_{i=1}^p \beta_i x_i$$

- We have many predictors (x_1, x_2, x_3, \dots)
- We have one response (y)
- We need to find $\beta_0, \beta_1, \beta_2, \dots, \beta_p$
- We also want our number of samples, n , to be much larger than p
 - This becomes harder as we add more terms

Multiple Linear Regression: Matrix Formulation

- Linear regression can be seen as a single equation involving matrices and vectors. For example, with two explanatory variables and n points of data:

$$\mathbf{y} = \mathbf{x}\beta + \varepsilon$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \mathbf{x} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} \\ 1 & x_{2,1} & x_{2,2} \\ \vdots & \vdots & \vdots \\ 1 & x_{n,1} & x_{n,2} \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}, \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

- Error vector here is $\varepsilon = \mathbf{y} - \mathbf{x}\beta$, which is a function of β .

Multiple Linear Regression: Matrix Formulation

- We want to minimise the **sum of squared errors**:

$$SSE(\beta) = \sum_{i=1}^n \varepsilon_i^2 = \varepsilon' \varepsilon = (\mathbf{y}' \mathbf{y} - 2\beta' \mathbf{x}' \mathbf{y} + \beta' \mathbf{x}' \mathbf{x} \beta)$$

- The way a function changes with respect to certain variables can be calculated using the function's **derivative**
- Once the derivative is calculated, set the equation to be equal to zero and solve for the same variable in order to find maximum(s) and/or minimum(s) of the function.

Multiple Linear Regression: Matrix Formulation

- Sum of squared errors term:

$$SSE(\beta) = (\mathbf{y}'\mathbf{y} - 2\beta'\mathbf{x}'\mathbf{y} + \beta'\mathbf{x}'\mathbf{x}\beta)$$

- Derivative of SSE with respect to β :

$$\nabla SSE(\beta) = 2(\mathbf{x}'\mathbf{x}\beta - \mathbf{x}'\mathbf{y})$$

- Setting to 0 and solving for β gives the optimal vector, $\hat{\beta}$:

$$\hat{\beta} = (\mathbf{x}'\mathbf{x})^{-1}\mathbf{x}'\mathbf{y}$$

- Linear regression is also known as **ordinary least squares**

Multiple Linear Regression: Demo

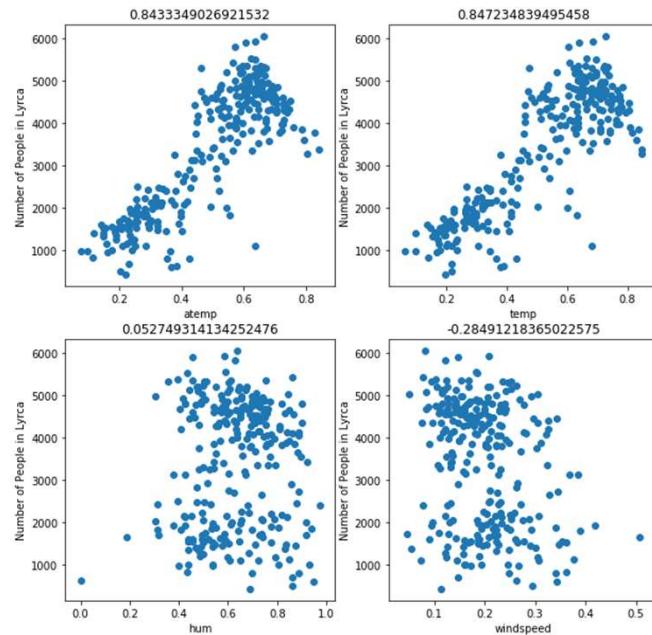
- See ***CAB420_Regression_Example_1_Linear_Regression.ipynb***
- Task:
 - Fit a linear regression model to predict cyclist numbers from weather data
 - Temperature, apparent temperature, humidity, windspeed
 - Evaluate the model, and explore the impact of each variable and how they contribute the to the model
- This demo will be covered in more detail in the interactive lecture session

Regression Demo: High Level Process

- Load and visualise data
 - Looking for missing data, errors, exploring relationships (i.e. correlation) in the data
- Split data
 - Train and test sets
- Extract predictors (inputs) and response (output)
- Fit model
- Explore model performance
 - Quality of fit, validity of assumptions
- Refine model
 - And explore performance, refine again, etc.

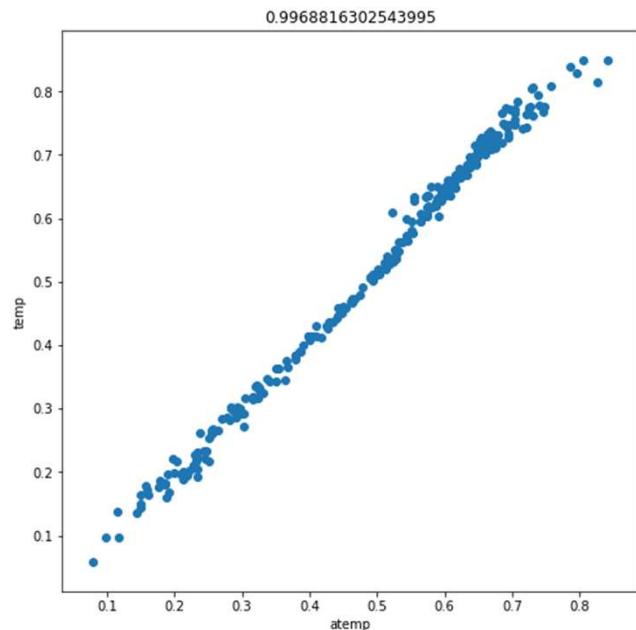
Correlation between predictors and response

- temp and atemp highly correlated with our response (cnt)
- hum and windspeed are less correlated with the response (cnt)



Correlation between predictors

- temp and atemp are very highly correlated



A Simple Model

Fitting this:

- $\text{cnt} \sim \text{atemp} + \text{temp} + \text{hum} + \text{windspeed}$

We get this output:

```
OLS Regression Results
=====
Dep. Variable:          cnt    R-squared:       0.748
Model:                 OLS    Adj. R-squared:   0.744
Method:                Least Squares F-statistic:     198.4
Date: Mon, 11 Jan 2021 Prob (F-statistic): 8.06e-79
Time: 11:36:23           Log-Likelihood: -2190.2
No. Observations:      273    AIC:             4390.
Df Residuals:          268    BIC:             4408.
Df Model:                  4
Covariance Type:        nonrobust
=====
            coef    std err        t      P>|t|      [0.025      0.975]
-----
Intercept  1708.8451  296.956      5.755      0.000    1124.182    2293.509
atemp     -3132.5570  3164.040     -0.990     0.323    -9362.093    3096.979
temp       8823.6644  2823.617      3.125     0.002    3264.372  1.44e+04
hum       -1134.9410  302.778     -3.748     0.000    -1731.067    -538.815
windspeed -3052.9184  642.368     -4.753     0.000    -4317.647   -1788.190
=====
Omnibus:            15.311 Durbin-Watson:      0.963
Prob(Omnibus):      0.000 Jarque-Bera (JB):  22.768
Skew:              -0.383 Prob(JB):        1.14e-05
Kurtosis:           4.190 Cond. No.         132.
=====
```

Regression Diagnostics

- How do we know if the model is valid? We need to look at:
 - Are our assumptions on the residuals correct?
 - Are there outliers that distort the model?
 - Are all the terms significant?
 - Are there correlated variables?
 - Does the model do a good job of explaining variation in y ?

Analysing Model Terms

- Typical regression output on right
 - coef: the actual coefficient value, i.e. $\beta_0, \beta_1, \beta_2, \dots, \beta_p$
 - std err: standard error, a measure of how much the coefficient changes by if we resample the data and recompute the regression
 - $t = \frac{\text{Estimate}}{\text{SE}}$, we want this to be a long way from 0, as it indicates that the term is less likely to be the result of noise
 - $P>|t|$: p-value for the null hypothesis that the coefficient is equal to 0. Low p-value means that you can reject the null hypothesis, i.e. that the term is significant.

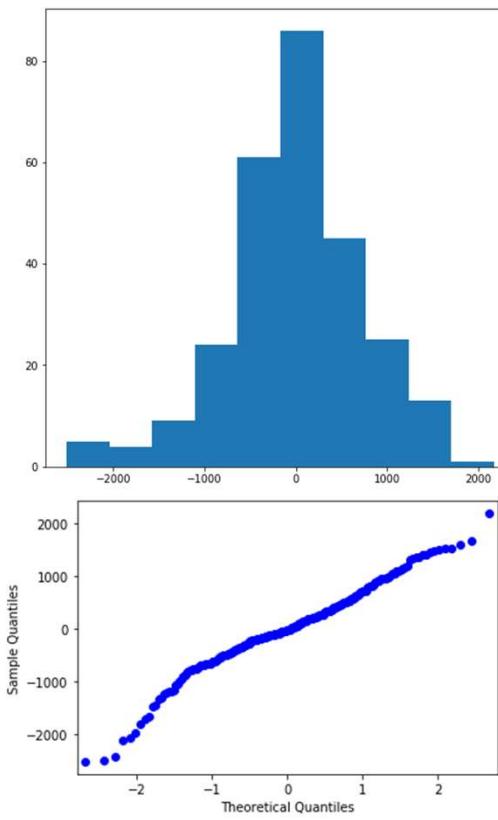
	coef	std err	t	P> t
Intercept	1708.8451	296.956	5.755	0.000
atemp	-3132.5570	3164.040	-0.990	0.323
temp	8823.6644	2823.617	3.125	0.002
hum	-1134.9410	302.778	-3.748	0.000
windspeed	-3052.9184	642.368	-4.753	0.000

Analysis Model Terms

- Focus on the p-values
 - High p-value indicates a term that is not significant
 - Does not contribute to the model
 - Why does it not contribute?
 - There could be no actual relationship
 - Could have correlated variables
 - Two or more predictor variables capture the same relationship with the response
 - Remove poor terms one by one until all terms are significant

Analysing Residuals

- Our residuals should be Gaussian distributed
 - We can use a histogram of residuals and look for a normal distribution
 - We can use a qqplot and see how many points are away from the normal distribution line



Whole of Model Analysis

- R-Squared: Proportion of observed variance explained by the model. A value of 1 means the model explains everything, a value of 0 means it explains nothing.
Defined as follows:
 - $R^2 = 1 - \frac{SSE}{SSY}$
 - $SSE = \sum_{i=0}^n (y_i - \hat{y}_i)$
 - $SSY = \sum_{i=0}^n (y_i - \bar{y})$; \bar{y} = mean of the data
- Adjusted R-Squared: Considers the model complexity (number of terms) alongside how much variance it explains.
 - $R^2_{Adj} = 1 - \frac{n-1}{n-p-1} \frac{SSE}{SSY}$
- F-statistic: Test statistic for the whole model. A p-value is also provided to indicate if the entire model is significant, much the same as we have for the individual components.

Whole of Model Analysis

- Number of observations: how many samples we have, in general, more is better (to a point, things get trickier when we can't fit the data in memory)
- Error degrees of freedom: Number of observations minus the number of terms. We want this to be big.
- Root Mean Squared Error: A measure of error for the model. Smaller is better (not in Python's StatsModels output, though easy to calculate).

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=0}^n (y_i - \hat{y}_i)^2}$$

Whole of Model Analysis

- RMSE is only really useful to compare models trained on the same data
 - The range of RMSE depends on the data
- R-squared and adjusted R-squared are good indicators of predictive power
 - Can only calculate this data on the training set
 - Based around the assumptions made for residuals and their distribution
 - R-squared is ALWAYS calculated on training data
 - Blindly maximising this can lead to overfitting

Improving the Model

- atemp and temp are very similar variables
 - High correlation
 - Impacts fitted terms and p-values
- Remove one of the terms
 - See example script for detailed results

```
=====
      coef    std err        t     P>|t|      [0.025      0.975]
-----
Intercept  1708.8451   296.956     5.755    0.000   1124.182   2293.509
atemp     -3132.5570  3164.040    -0.990    0.323   -9362.093   3096.979
temp       8823.6644  2823.617     3.125    0.002   3264.372   1.44e+04
hum        -1134.9410  302.778    -3.748    0.000   -1731.067   -538.815
windspeed  -3052.9184  642.368    -4.753    0.000   -4317.647   -1788.190
=====
```

```
=====
      coef    std err        t     P>|t|      [0.025      0.975]
-----
Intercept  1598.9598   275.424     5.805    0.000   1056.698   2141.222
temp       6037.2059   227.171    26.576    0.000   5589.946   6484.466
hum        -1144.5128  302.612    -3.782    0.000   -1740.303   -548.723
windspeed -2966.6479  636.407    -4.662    0.000   -4219.619   -1713.677
=====
```

Other Considerations

- Data splits
 - Random vs Sequential
- Categorical Variables
 - Specified and handled differently by the model
 - Each category has an "offset" associated with it that is included when that category is "on"
 - Can rapidly increase model size
 - Category with 20 options will add 19 new terms
 - Can cause problems when some categories are rare

Other Considerations

- Higher order terms
 - Product of two predictors, square of a predictor, higher order polynomials relating predictors
 - Can greatly improve model performance
 - i.e. can now capture quadratic relationships
 - Can cause overfitting
 - Lots of additional terms can be added very quickly
 - Care needs to be taken to ensure all terms are meaningful and data is sufficient

CAB420: Overfitting and Linear Regression

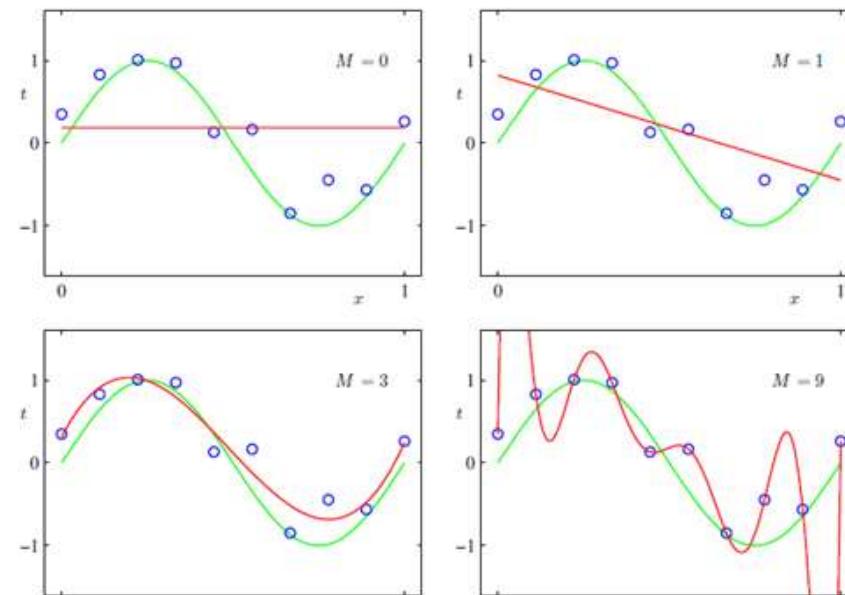
WHAT IS IT? AND WHY DO I CARE?

Overfitting and Regression

- Consider a multi-variate linear regression task
- We can (usually) make the model more accurate on the test set by adding more terms
 - Additional variables
 - Higher order terms

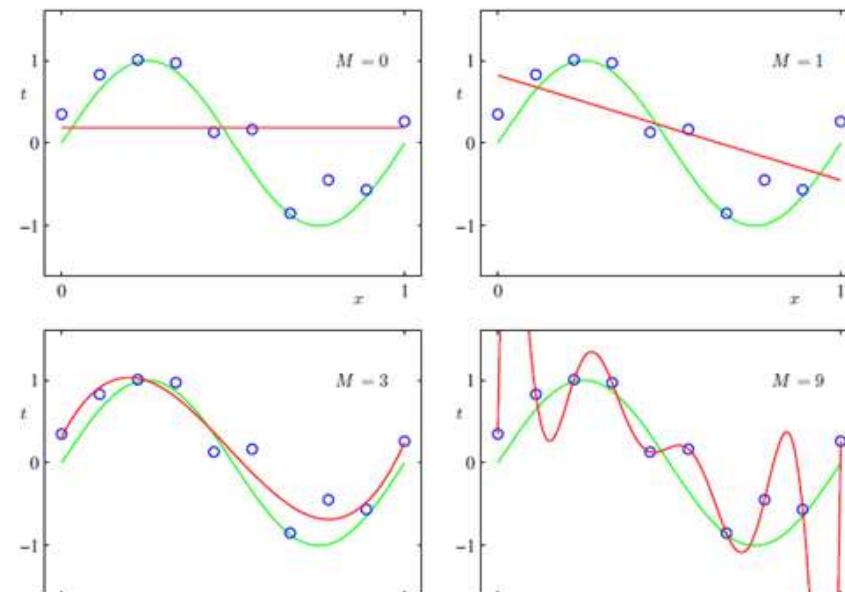
Overfitting and Regression

- On the right we have:
 - A sine wave in green, which has been sampled
 - Samples have been offset by noise
 - We seek to fit a curve (in red) to the sampled data



Overfitting and Regression

- M=9 (9th order polynomial) offers the best fit to the data
 - Hits all the points almost perfectly
- M=3 actually captures the function the best
 - Some error in predictions
 - Overall shape correct however
- Consider, how would M=9 and M=3 perform on a new set of points?
 - Which one would look more correct?



Detecting Overfitting

- We cannot observe overfitting using the training set alone
 - Validation and testing sets are required
- Performance will likely always increase on the training set
 - Need to evaluate performance on other data held out of training
 - Validation data, Testing data
- Often referred to as testing if a model **generalises to unseen data**

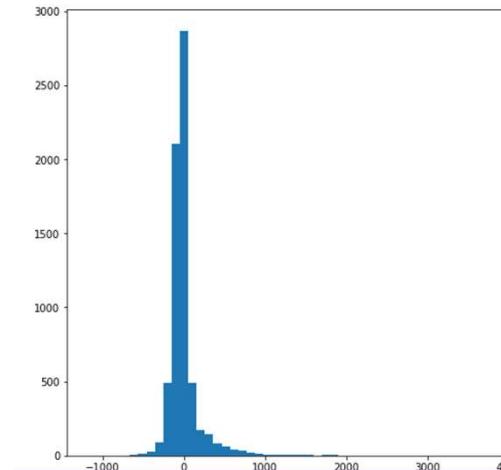
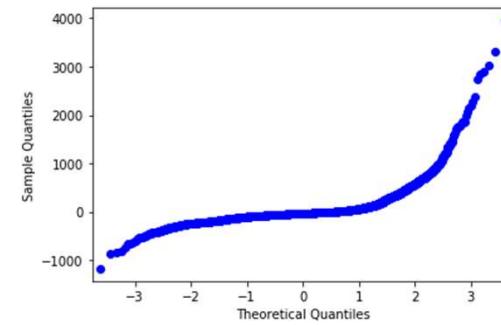
Overfitting in Practice

- See ***CAB420_Regression_Example_2-Regularised_Regression.ipynb***
- Demo Overview
 - Load traffic data from Brisbane which contains average travel times between key points on the road network
 - We'll consider the first 9 data series and time of day
 - First 8 series as predictors, with the hour as a categorical
 - 9th series is the response
 - Apply linear regression to data, increase complexity and observe results

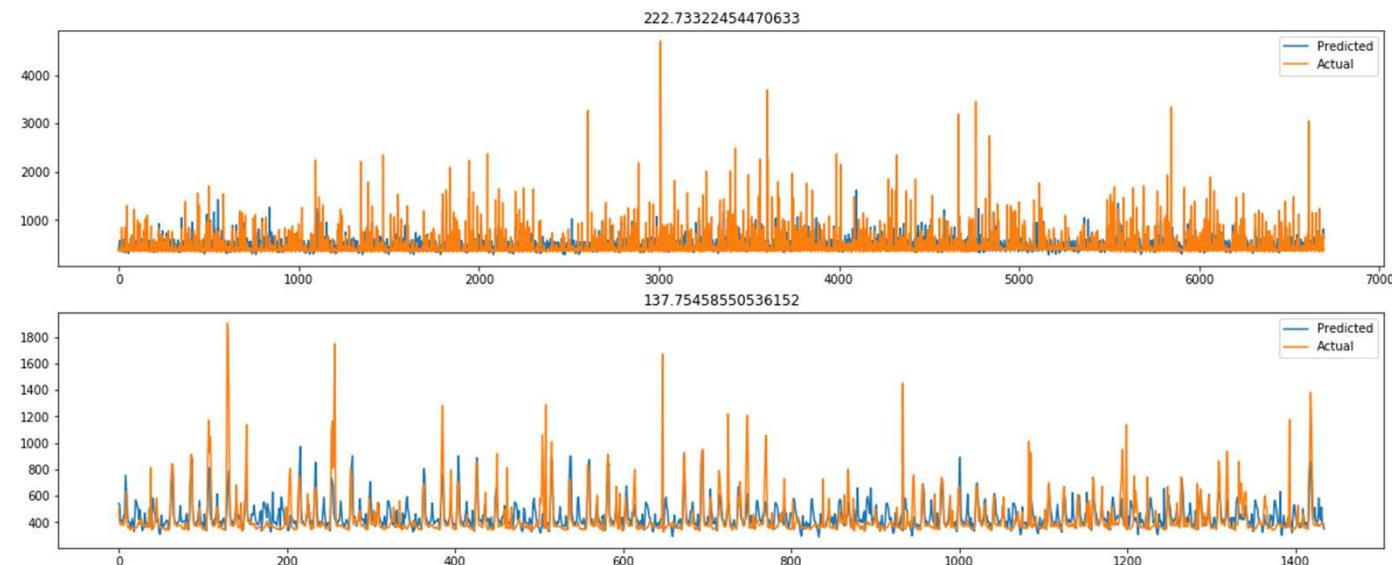
Simple Linear Model

(linear terms with hour of day categorical term)

```
OLS Regression Results
=====
Dep. Variable: x_1260_1261 R-squared: 0.256
Model: OLS Adj. R-squared: 0.253
Method: Least Squares F-statistic: 73.95
Date: Wed, 13 Jan 2021 Prob (F-statistic): 0.00
Time: 20:03:40 Log-Likelihood: -45686.
No. Observations: 6694 AIC: 9.144e+04
Df Residuals: 6662 BIC: 9.165e+04
Df Model: 31
Covariance Type: nonrobust
=====
            coef    std err      t      P>|t|      [0.025      0.975]
-----
const    119.9389   26.334     4.555      0.000     68.316     171.562
x_1098_1056_  0.0462   0.057     0.804      0.422     -0.066     0.159
x_1058_1059_  0.2443   0.082     2.980      0.003     0.084     0.405
x_1057_1056_  2.7346   0.167    16.356      0.000     2.407     3.062
x_1017_1007_  0.2636   0.048     5.492      0.000     0.169     0.358
x_1115_1015_  1.3343   0.161     8.268      0.000     1.018     1.651
x_1015_1115_  0.1983   0.272     0.730      0.466     -0.334     0.731
x_1103_1061_ -0.1058   0.134    -0.792      0.429     -0.368     0.156
x_1135_1231_  0.7734   0.112     6.891      0.000     0.553     0.993
1       -50.3136  38.054    -1.322      0.186    -124.912    24.284
2       33.6517  43.325     0.777      0.437    -51.279    118.583
3      -41.9974  27.007    -1.555      0.120    -94.940    10.945
4      -70.6742  24.455    -2.890      0.004   -118.614   -22.734
5      -5.7150  24.002    -0.238      0.812    -52.767    41.337
6      128.0683  24.261     5.279      0.000    80.510    175.627
7      115.4191  24.745     4.664      0.000    66.912    163.927
8      52.9131  24.839     2.130      0.033    4.221    101.605
9       2.7002  24.065     0.112      0.911   -44.475    49.876
10     -86.9350  24.414    -3.561      0.000   -134.793   -39.077
11     -94.3841  24.791    -3.807      0.000   -142.982   -45.786
12     -121.3032  25.006    -4.851      0.000   -170.323   -72.283
13     -126.5093  25.019    -5.052      0.000   -175.593   -77.425
14     -105.4192  25.107    -4.588      0.000   -214.638   -116.201
15     -188.6012  26.228    -7.191      0.000   -240.016   -137.186
16     -153.1932  24.073    -4.870      0.000   -204.309   -102.086
```



Simple Linear Model

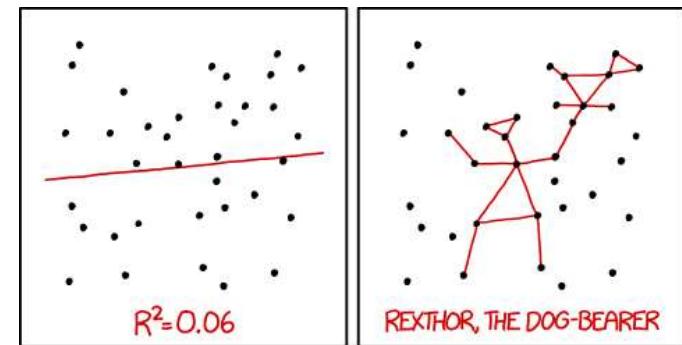


Simple Linear Model

- R-squared quite low
- Lots of data
- Most terms significant
 - 3 of our other predictors have poor p-values
 - Could investigate co-linearity here
 - May also be predictors that are unrelated to the response
 - Hour of day significant
 - Note that if one of the categorical terms is significant, we consider the whole model significant
- Residuals not normally distributed
- Predictions not great
- Higher accuracy on the test set
 - Not overfitting

Simple Linear Model: Is it any good?

- Sort of
 - No overfitting, simple model
 - Some poor terms, but most are meaningful
 - Predictive power is limited, but model seems to capture the main trends
 - End use needs to be kept in mind – is the model fit for purpose? How accurate does it need to be?
- Improving the model
 - Investigate higher order terms



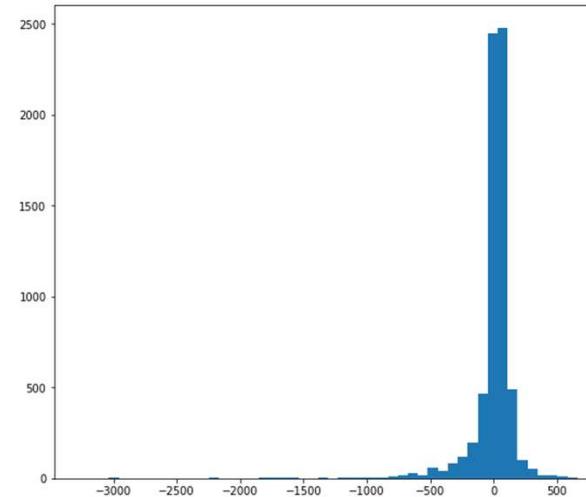
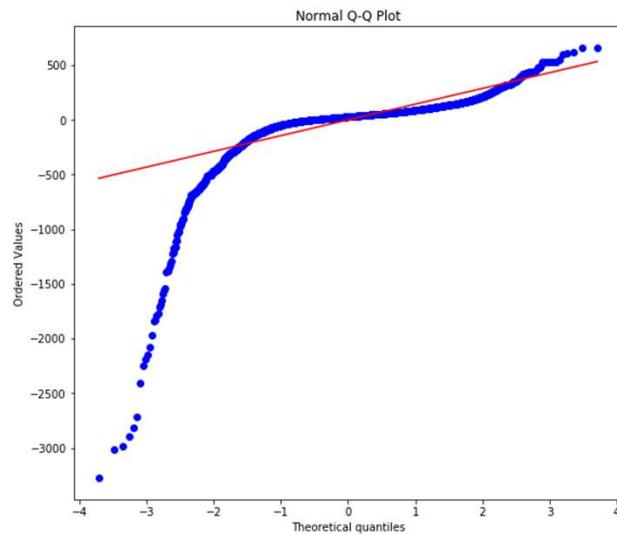
I DON'T TRUST LINEAR REGRESSIONS WHEN IT'S HARDER TO GUESS THE DIRECTION OF THE CORRELATION FROM THE SCATTER PLOT THAN TO FIND NEW CONSTELLATIONS ON IT.

Cartoon from XKCD

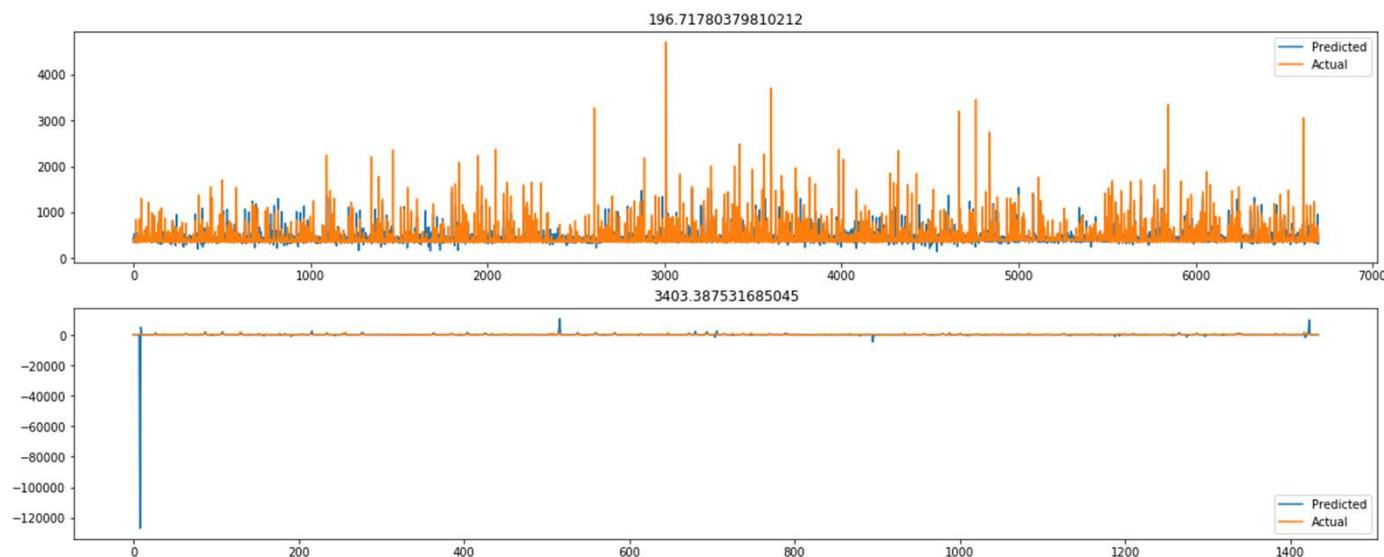
A More Complex Model

(quartic terms with interactions, and hour of day categorical term)

- ~500 model parameters
 - Too many terms to reasonably consider p-values, etc
 - R-squared of 0.412
 - A big improvement over what we had



A More Complex Model



A More Complex Model

- Improved R-squared (though with room for further improvement)
- Improved accuracy on training set
- Residuals not normally distributed
- Massive errors on the testing set
 - Model is overfitting

Complex Linear Model: Is it any good?

- Not really
 - Unpredictable performance on test data
 - Very high number of parameters
 - Difficult to inspect or tune due to size
 - Likely large amounts of redundancy, though difficult to assess due to model size
- Improving the model
 - Removing terms:
 - Reverting to lower order (i.e. quadratic rather than quartic) would reduce complexity, but may discard useful terms
 - Manual investigation is difficult given model size

CAB420: Regularisation

MAKING MODELS REGULAR?

Bias and Variance

- Bias and Variance are two factors in regression which we try to manipulate in order to find the "best" model.
- The **variance** of a model is the error from sensitivity to small changes in the training data. High variance can lead to overfitting.
 - Somewhat indicated by the R^2
- The **bias** of a model is the error from erroneous assumptions in the model. High bias can lead to underfitting.
 - Somewhat indicated by the RMSE
- As more terms are added to a model (i.e., it becomes more complex), the coefficients more accurately fit the given data (i.e., *bias decreases*).
- However, as more terms are added the model will become worse at predicting new data (i.e., *variance increases*) due to **over-fitting**

Bias and Variance

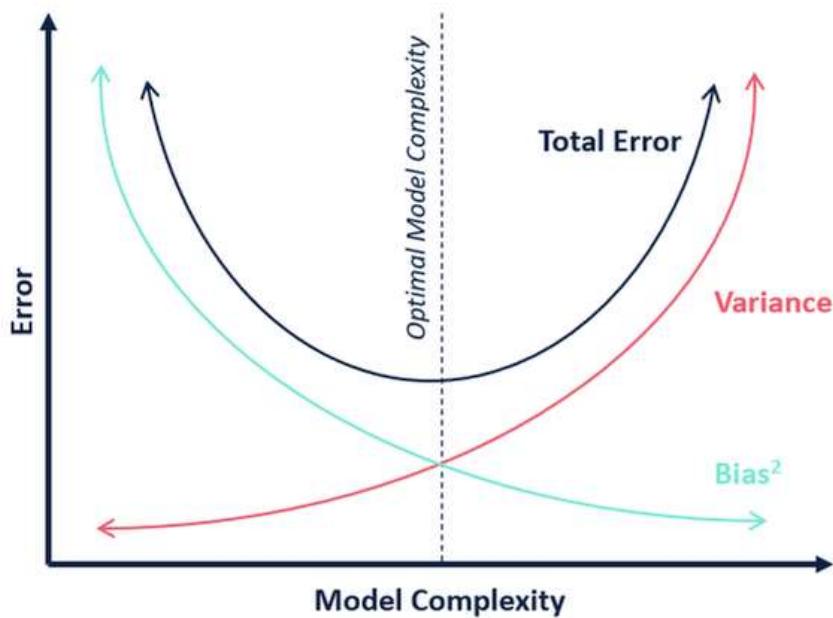


Image taken from blog on bias vs variance, found at:
<https://community.alteryx.com/t5/Data-Science-Blog/Bias-Versus-Variance/ba-p/351862>

Regularises

- Reduce the **magnitude** and/or **number** of parameters in order to reduce model complexity.
- Reduction in model complexity → reduced variance and increased bias.
- Useful when applied to models with many parameters.
- Regularisation seeks to penalise complex models
 - We have an intuition that a small change in input value to a model should lead to a small change in output value
 - Model complexity often leads to overfitting, reducing parameters (complexity) makes overfitting less likely

Regularisation and Regression

- Regularises are applied by penalising slope terms, β .
- There are two types of regularization we look at in CAB420:
 - L1 regularisation (Lasso regression), and
 - L2 regularisation (ridge regression).
- Both L1 and L2 seek to
 - Penalise big coefficients
 - Favour models with small slopes for individual data points
- Why?
 - A large slope means a small change in the data gives a large change in the estimate
 - Seek to reduce the model's variance, and make estimates more stable

Regularisation and Regression

- With linear regression we aim to find values for β that minimises

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2$$

- Regularisation applies a penalty term

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda P$$

where λ is a weight that controls the influence of our penalty

Regularisation and Regression

- Adds extra term(s) to the objective function
 - Terms don't operate over data or errors, but rather the model parameters
 - Regularisation terms are usually weighted
 - We can control how strong the regularisation is
 - How do we select the weight?
- Regularisation can also help when we have more dimensions than samples
 - Though in such situations we need to use an optimisation algorithm to find parameters

CAB420: Ridge Regression

L2 REGULARISATION

Ridge Regression

Linear Regression with L2 regularisation

- Add to our loss term the sum of the coefficients squared

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_2$$

- We don't add the intercept
- Very big slopes are penalised heavily
 - Favour smaller slopes for all terms
 - Weight the L2 term by a factor, lambda
 - The ridge term

Regression Formulation: Revision

- Recall that for OLS regression:

- Sum of squared errors term:

$$SSE(\beta) = (\mathbf{y}'\mathbf{y} - 2\beta'\mathbf{x}'\mathbf{y} + \beta'\mathbf{x}'\mathbf{x}\beta)$$

- Derivative of SSE with respect to β :

$$\nabla SSE(\beta) = 2(\mathbf{x}'\mathbf{x}\beta - \mathbf{x}'\mathbf{y})$$

- Setting to 0 and solving for β gives the optimal vector, $\hat{\beta}$:

$$\hat{\beta} = (\mathbf{x}'\mathbf{x})^{-1}\mathbf{x}'\mathbf{y}$$

Ridge Regression Formulation

- We want to minimize

$$(\mathbf{y}'\mathbf{y} - 2\beta'\mathbf{x}'\mathbf{y} + \beta'\mathbf{x}'\mathbf{x}\beta) + \lambda\beta'\beta$$

- Derivative with respect to β :

$$2(\mathbf{x}'\mathbf{x}\beta - \mathbf{x}'\mathbf{y} + \lambda\beta)$$

- Setting to 0 and solving for β gives the optimal vector, $\hat{\beta}$:

$$\begin{aligned} 0 &= \beta(\mathbf{x}'\mathbf{x} + \lambda I) - \mathbf{x}'\mathbf{y} \\ \hat{\beta} &= (\mathbf{x}'\mathbf{x} + \lambda I)^{-1}\mathbf{x}'\mathbf{y} \end{aligned}$$

- Known as **ridge** regression because the slope penalty term is added along the diagonal of $\mathbf{x}'\mathbf{x}$ like a ridge.

Demo

- See ***CAB420_Regression_Example_2-Regularised_Regression.ipynb***
- Same setup as our overfitting example from before
- Fit to data using Ridge Regression

Using Ridge Regression

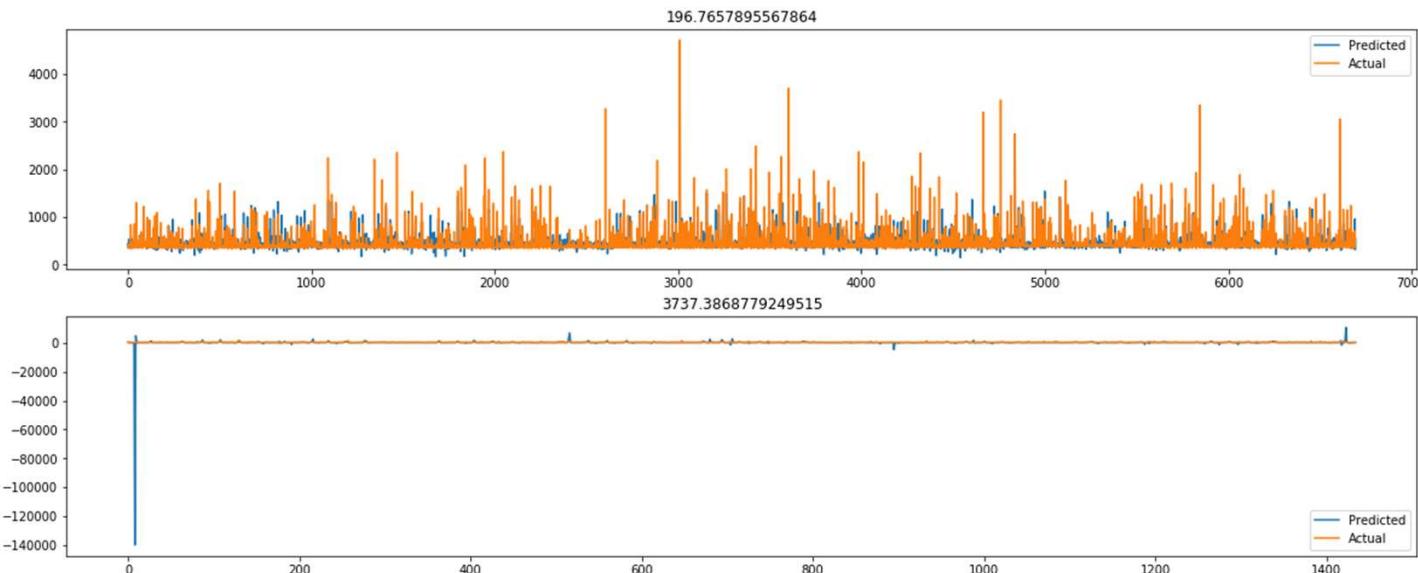
- Formula:

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_2$$

- We need to choose λ
- What should λ be?
 - What happens if it's 0?
 - Let's try 1

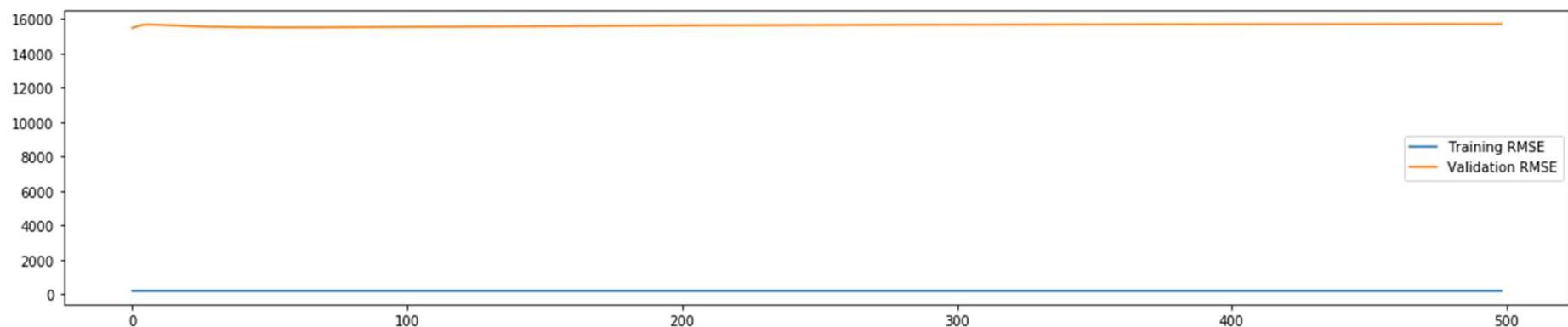
Ridge Regression: Results

- λ perhaps should not be 1
- Instead, try a range of values
 - 0, 2, 4, 6,, 498, 500



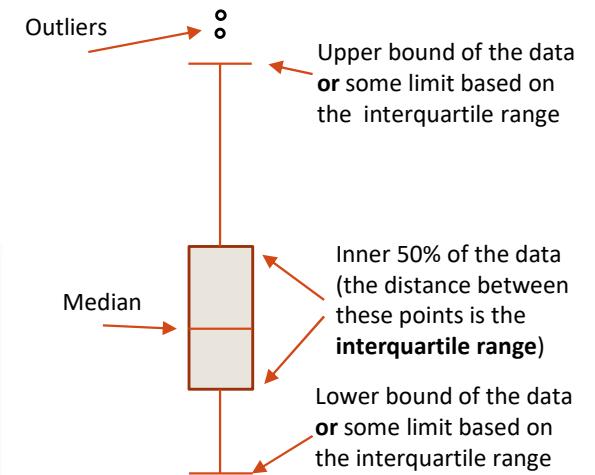
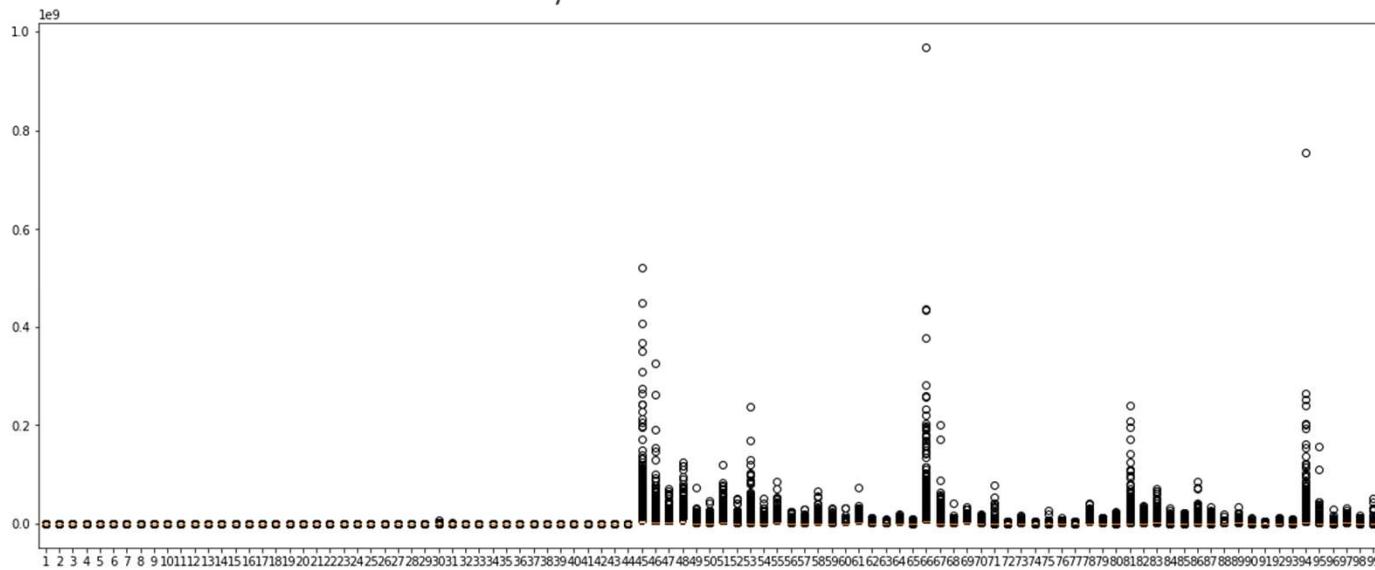
Ridge Regression: Results

- Plotting RMSE as λ changes
- We see a very small change as λ increases
 - Clearly λ needs to be much bigger with the data as it is



An Aside: Standardisation

- Let's visualise our data using a box plot
- We can see that different variables have very different ranges
 - First 100 dimensions only shown



Standardisation – Why?

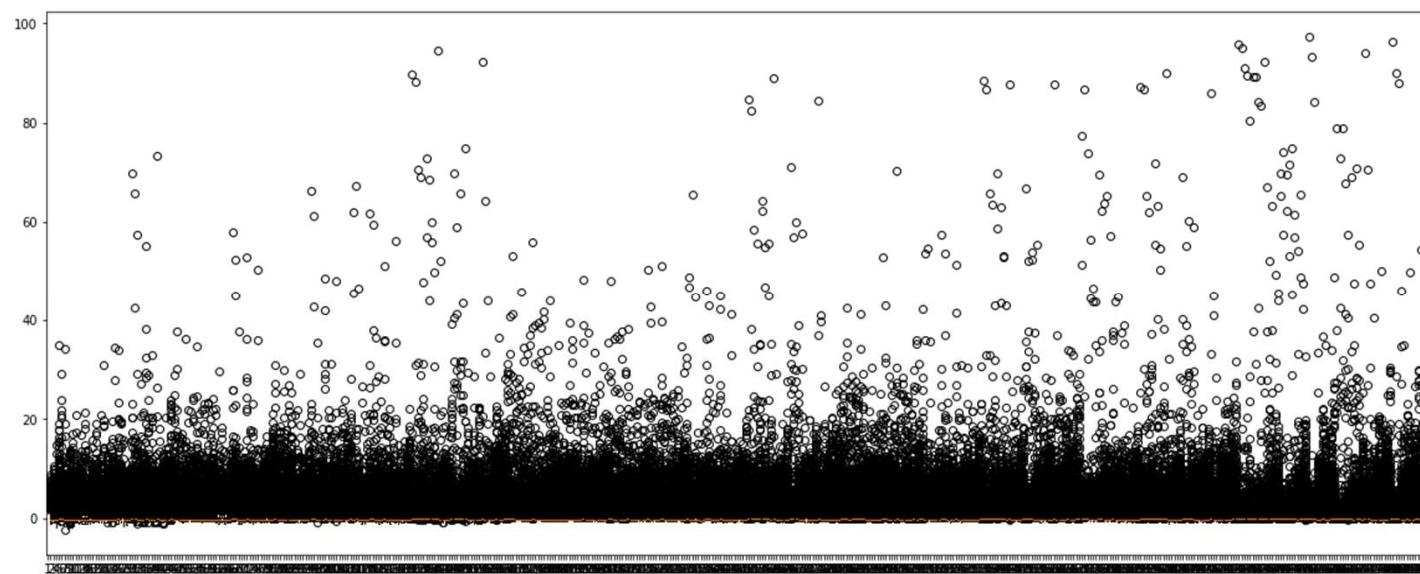
- For a given dataset, dimensions are usually in different scales
 - i.e. Dimension 1 may range from [0..1], Dimension 2 may range from [100...100000]
 - With a regularisation penalty, Dimension 1 may be penalised much more than Dimension 2 due to its scale
- We seek to scale all dimensions equally, so that they are all considered equally when fitting a model

Standardisation – What?

- For each dimension
 - Get the mean and standard deviation
 - For that dimension, subtract the mean, divide by the standard deviation
- End result:
 - All dimensions have mean 0, standard deviation 1
 - i.e. they are all scaled to the same range
 - Outliers are preserved
 - A point that is 10 standard deviations away in the original set, is still 10 standard deviations away
- Also
 - It usually makes the model easier to visualise

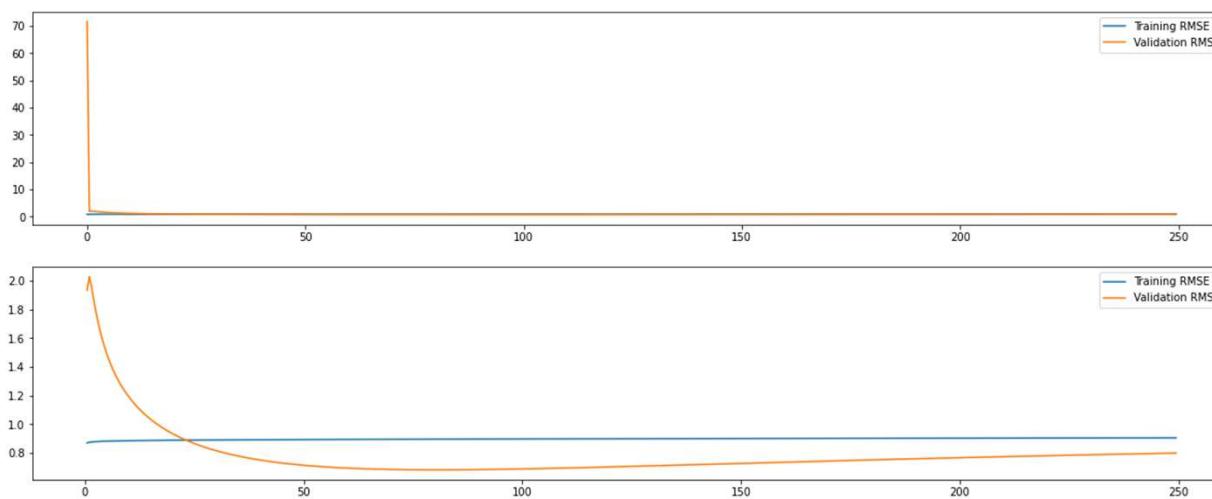
Standardised Data

- All data now has a similar range
 - First 100 dimensions shown
 - Lots of outliers still visible



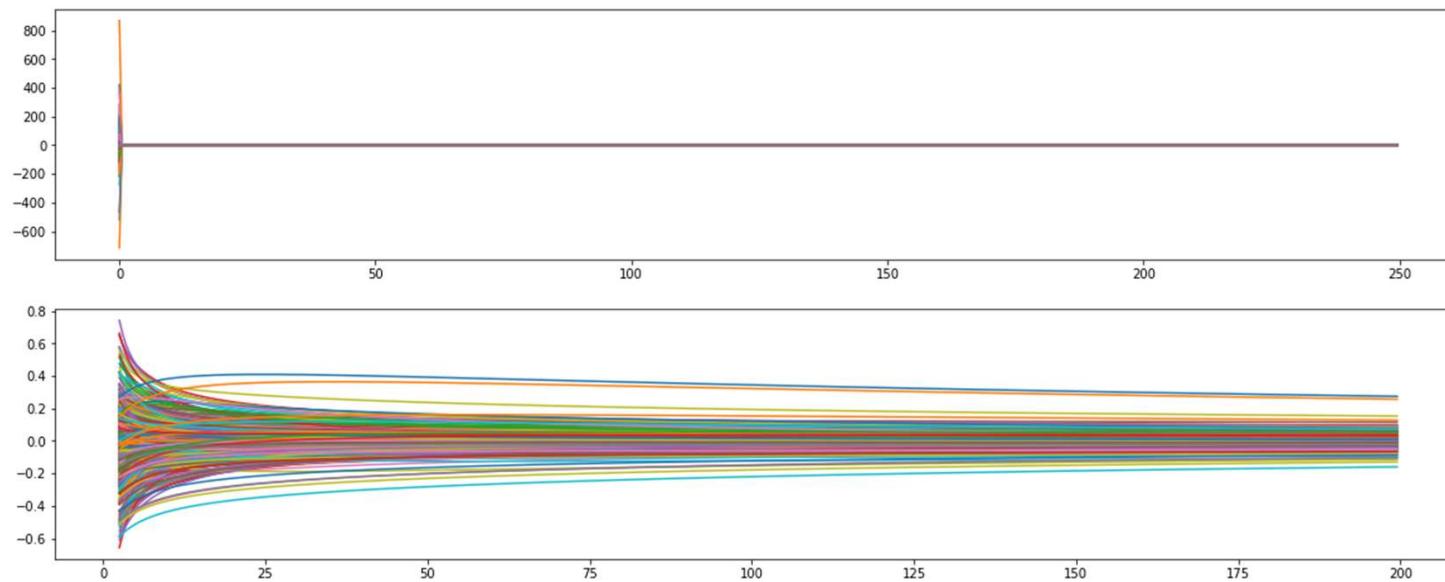
Ridge Regression with Standardised Data

- RMSE vs λ
 - We see an immediate drop as we increase λ
 - Remember, $\lambda = 0$ is least squared regression
 - Value which minimises the Validation RMSE is our best λ
 - For us, this is 79.5
 - Training RMSE will gradually increase with λ
 - Variance vs Bias



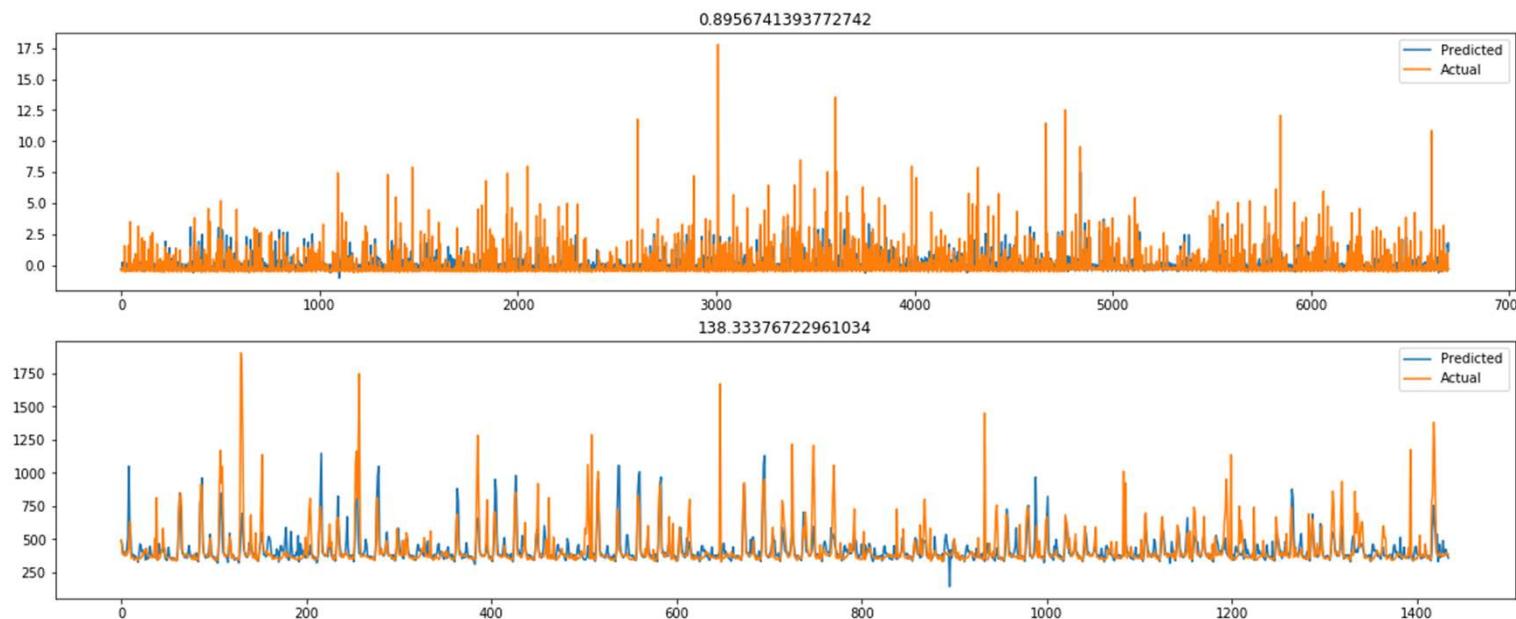
Ridge Trace Plot

- Individual Coefficients vs λ
 - Increases in λ lead to smaller coefficients overall
 - Note the distorted scale when $\lambda = 0$ is included
 - Coefficients gradually decrease and slowly approach 0



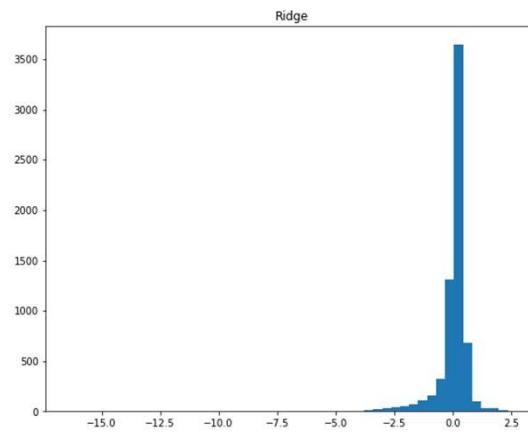
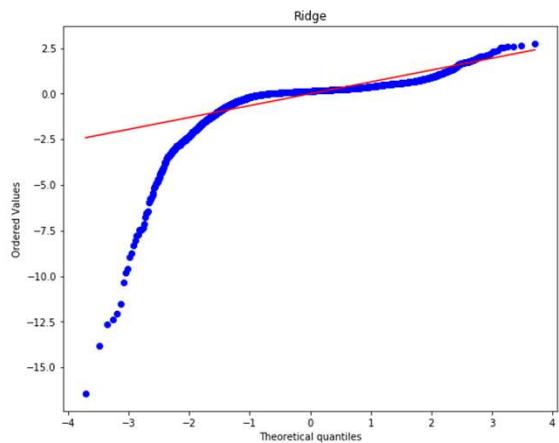
Ridge Results

- Final Model, $\lambda = 79.5$
 - Similar performance to original Linear model



Ridge Results

- Final Model, $\lambda = 79.5$
- $R^2 = 0.244$
 - Much lower R^2 than our higher order linear model, yet better performance on validation data
 - Variance vs Bias
- Similar looking residual plots to previously



CAB420: LASSO Regression

L1 REGULARISATION

LASSO Regression

Linear Regression with L1 regularisation

- Add to our loss the sum of absolute values of coefficients

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_1$$

- Again, we don't add the intercept
- Compared to Ridge Regression

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_2 \text{ vs } \sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_1$$

- Only difference is the type of norm being used
 - L1 (LASSO) vs L2 (Ridge)
- Big coefficients aren't penalised quite as badly
- Coefficients can go to 0
 - We can eliminate poor terms
- L1 norm still controlled by a scaling factor

Lasso Regression Formulation

- We want to minimize

$$(\mathbf{y}'\mathbf{y} - 2\beta'\mathbf{x}'\mathbf{y} + \beta'\mathbf{x}'\mathbf{x}\beta) + \lambda\beta$$

- The following is the derivative with respect to β :

$$2\mathbf{x}'\mathbf{x}\beta - 2\mathbf{x}'\mathbf{y} + \lambda I$$

- Setting to 0 and solving for β gives the optimal vector, $\hat{\beta}$:

$$\hat{\beta} = (2\mathbf{x}'\mathbf{x})^{-1}(2\mathbf{x}'\mathbf{y} - \lambda I)$$

- Where does the name come from?

- Acronym: Least Absolute Selection and Shrinkage Operator

- Not completely straight-forward, as the term in the first line should be $\lambda|\beta|$

- This actually makes it a lot more complex

Demo

- See ***CAB420_Regression_Example_2-Regularised_Regression.ipynb***
- Same setup as our overfitting and ridge regression
- Fit to data using LASSO Regression

Using Lasso Regression

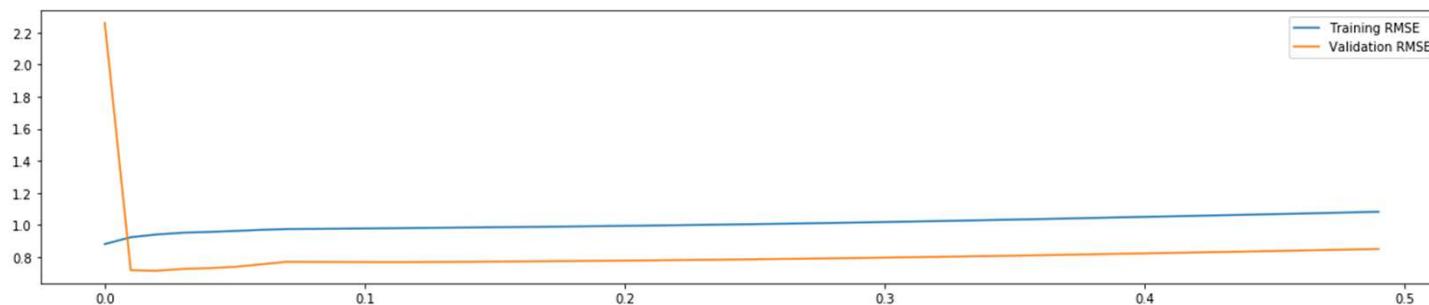
- Formula:

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_1$$

- We need to choose λ
- As per Ridge, we'll use a range
 - 0 to 0.5 in steps of 0.01
 - Lasso typically uses a smaller λ than ridge
- We'll use standarised data from the start

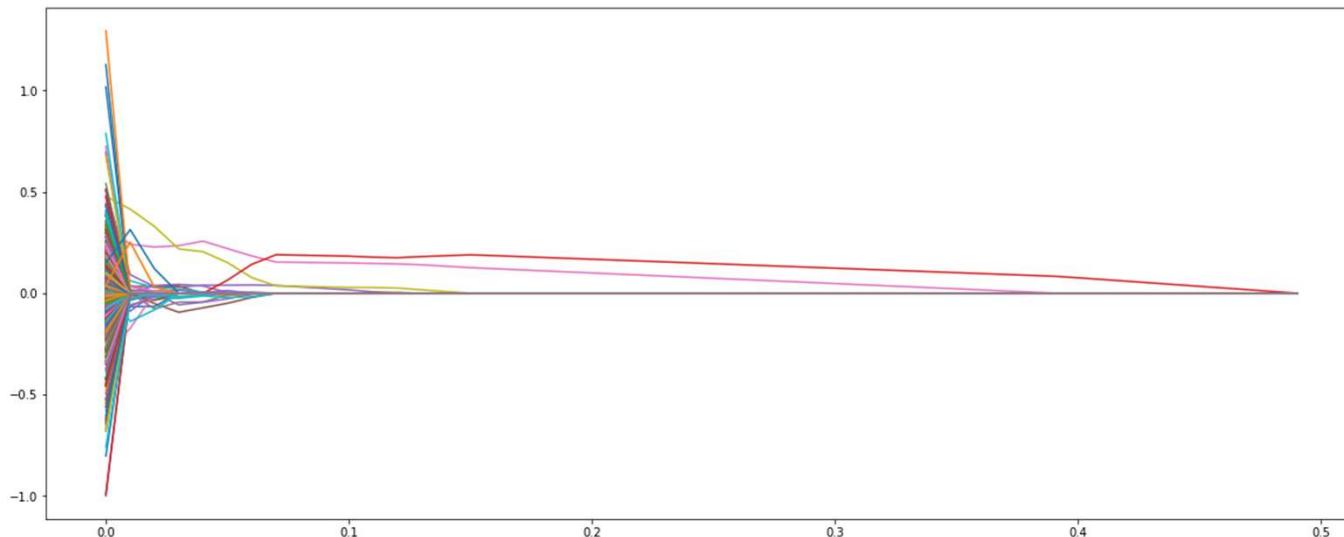
Lasso: Selecting Lambda

- Best $\lambda = 0.02$
- Same trend as ridge
 - Training data always increases with λ
 - Validation data decreases to a minimum, then increases



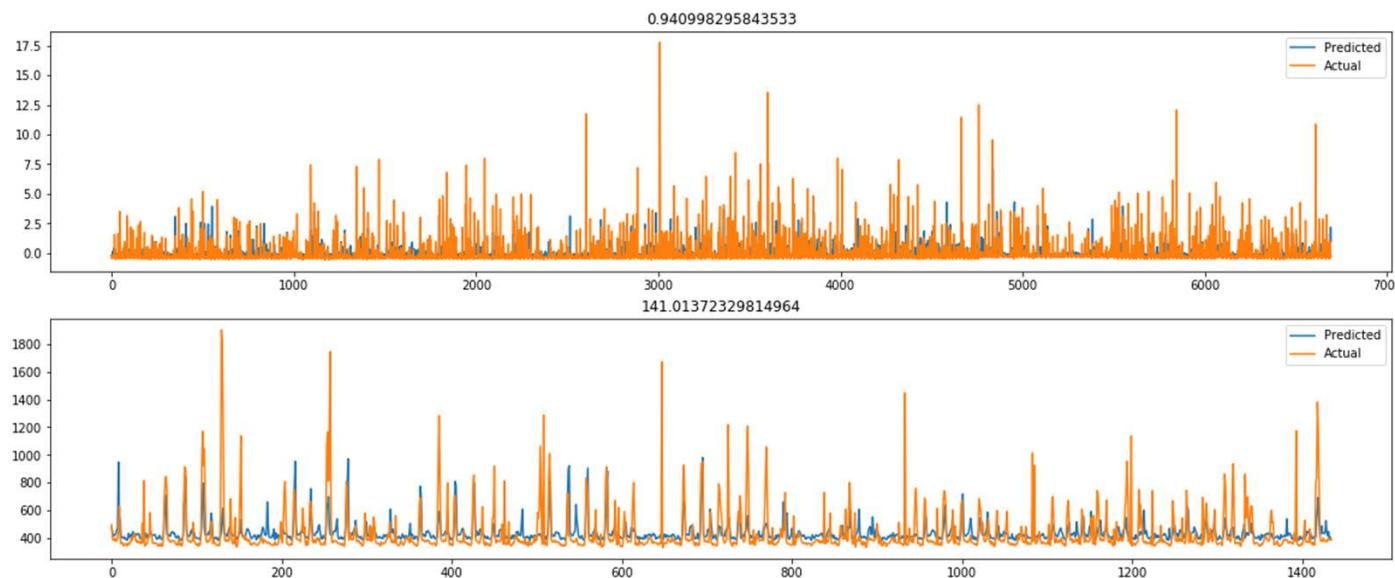
Lasso Trace Plot

- Terms decrease in value as λ increases
 - Terms can go to 0 and be eliminated
 - At the far end of the plot, all terms are 0 (constant model)



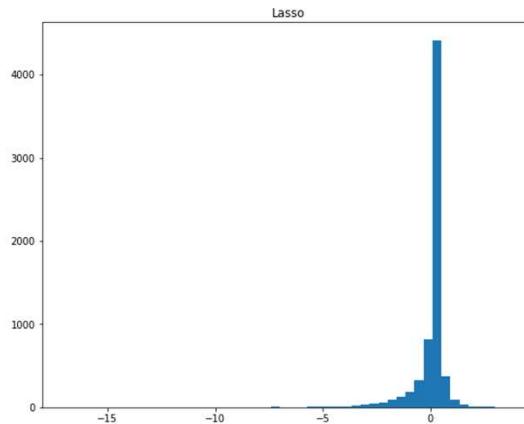
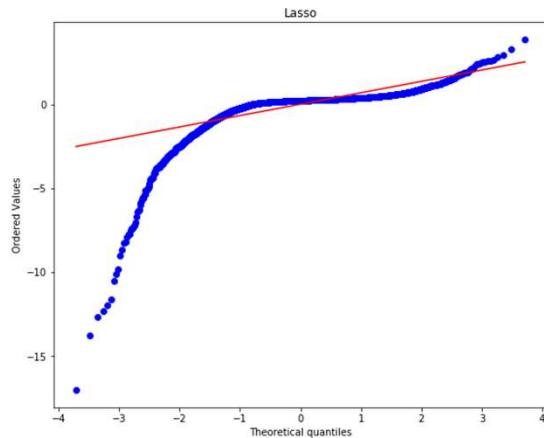
Lasso Results

- Final Model, $\lambda = 0.02$
 - Similar to Ridge and Linear Model
 - Final model contains 26 terms (all others are 0)



Lasso Results

- Final Model, $\lambda = 0.02$
- $R^2 = 0.315$
 - Between higher order linear model and ridge model
- Model less accurate than ridge on training data, more accurate than higher order linear model, Variance vs Bias again
- Similar looking residual plots to previously



ElasticNet Regression

- Bonus Regression Method!
- StatsModels regression implementation also does ElasticNet Regression
 - L1 and L2 terms added to the least squares loss
 - By default the function does pure Lasso
- Does this mean it's twice as good?
 - Not really, though it's not bad either
 - It does mean that we now have another hyper-parameter to tune
 - We need to select the relative weight of the two terms

A Note on Comparing Models

- We're only comparing our data on
 - Training data: which the model is trained on
 - Validation data: which is used to select lambda
- Ideally, we want a third dataset
 - Testing data: totally unseen, used to confirm that our model generalises to unseen data

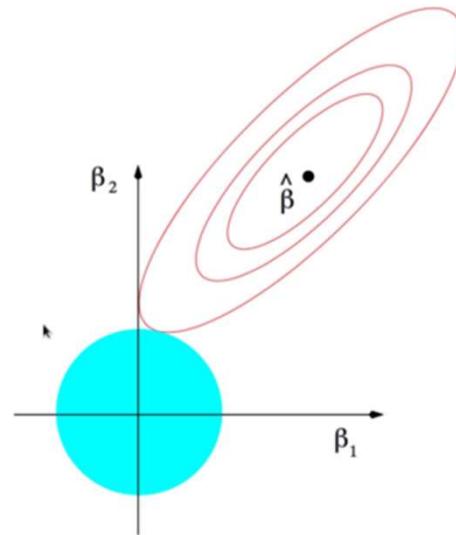
CAB420: Ridge vs LASSO

WHICH ONE?

Ridge vs Lasso

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_2$$

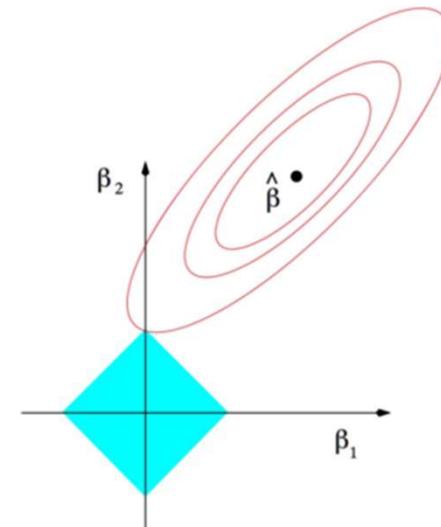
- We have two coefficients
 - The “best solution” according to least squares is $\hat{\beta}$
 - The blue area is the constraint region for a given λ
- Ridge uses an L_2 norm
 - Circular constraint region
 - Closest point on the constraint region to $\hat{\beta}$ is our ridge solution



Ridge vs Lasso

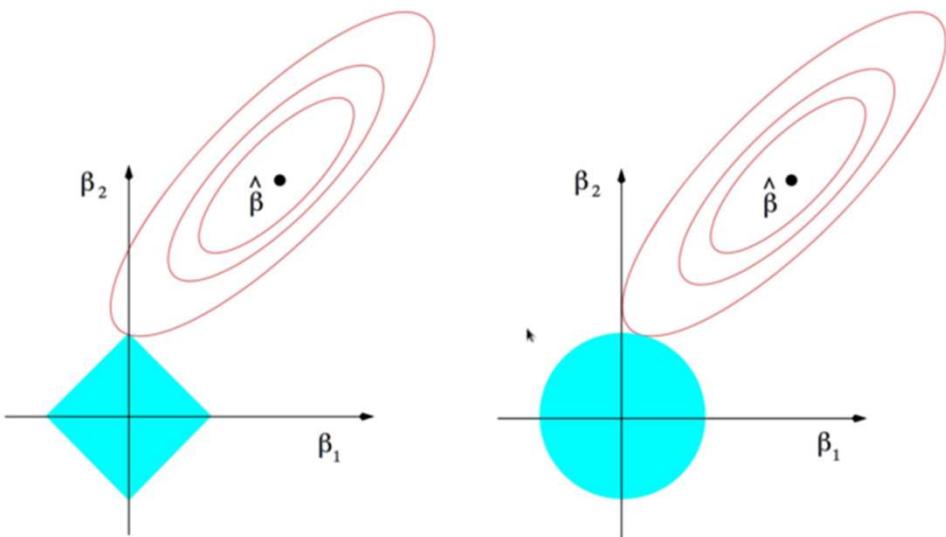
$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_1$$

- We have two coefficients
 - The “best solution” according to least squares is $\hat{\beta}$
 - The blue area is the constraint region for a given λ
- Lasso uses an L_1 norm
 - Diamond shaped constraint region
 - Closest point on the constraint region to $\hat{\beta}$ is our ridge solution



Ridge vs Lasso

- Due to the shape of the constraint region
 - Lasso can pull terms to 0
 - Ridge can make terms very small, but not 0



Impact of λ

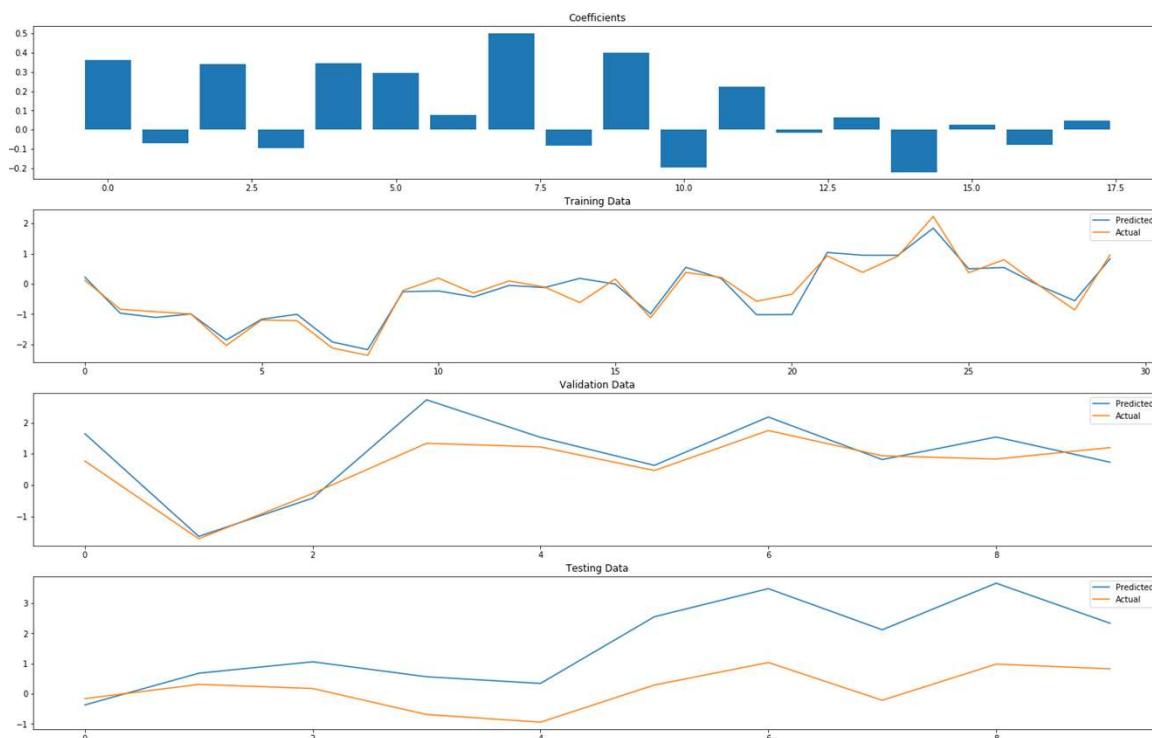
ANOTHER LOOK AT WHAT IT DOES

A Simple Example

- See ***CAB420_Regression_Additional_Example-Regularisation_Impact.ipynb***
- Predict traffic times again
 - Standardised data
 - 18 predictors
 - Linear, Ridge and Lasso models
 - Training, validation and testing set all taken from different time periods
 - Split in chronological order

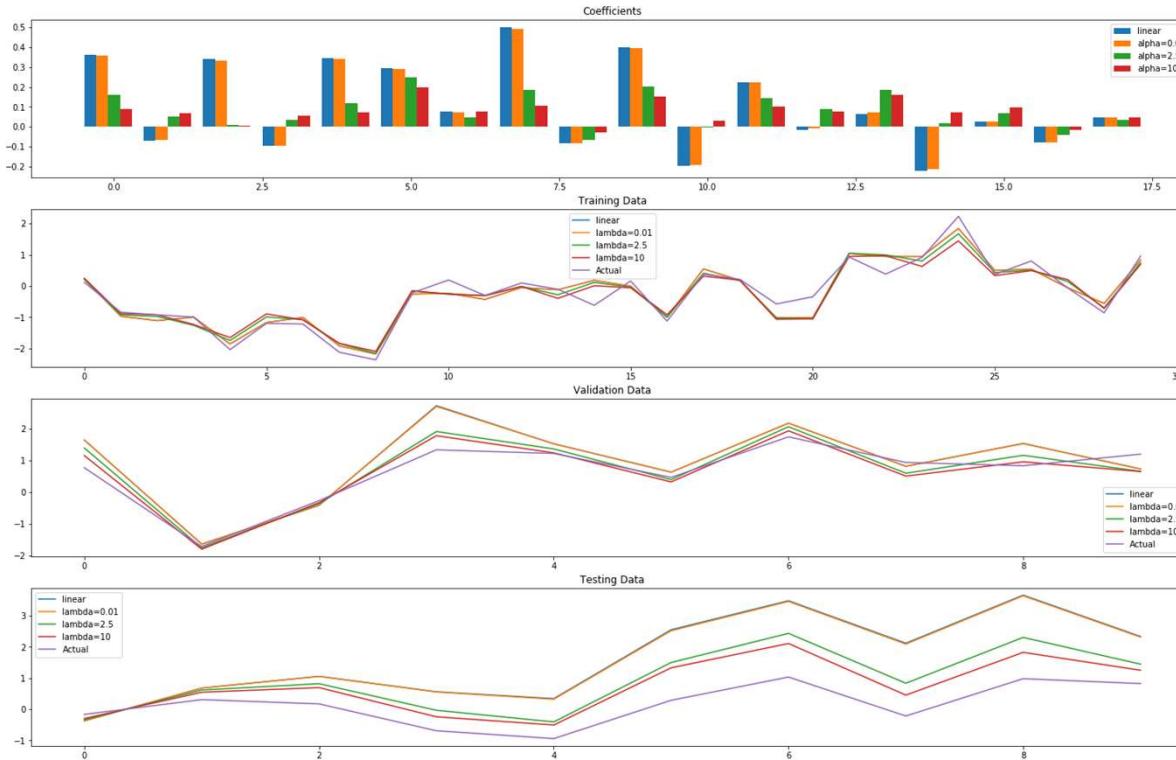
Linear Model

- Excellent fit to training data
- Fit gets worse for validation and testing data
- Coefficients vary in value



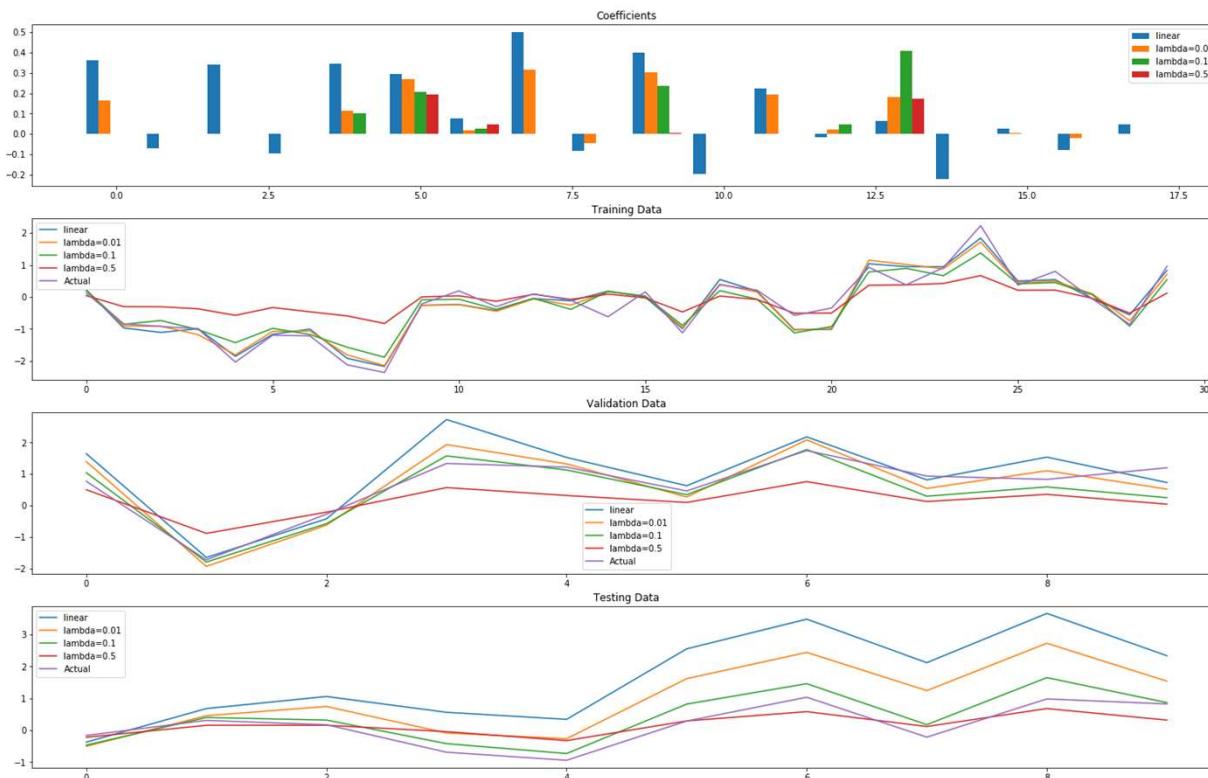
Ridge Model

- Larger λ leads to
 - Smaller coefficients
 - Flatter prediction curves
 - Coefficients can change sign
- Largest λ is least accurate on training data, most accurate on testing data



Lasso Model

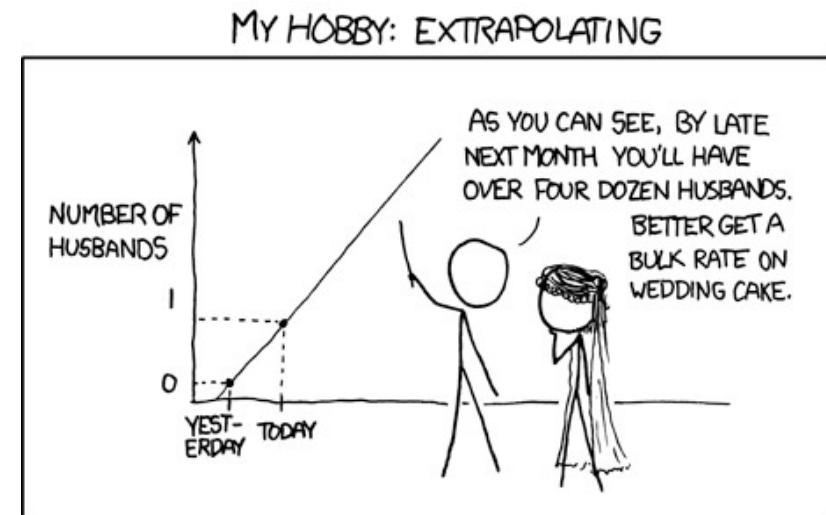
- Larger λ leads to
 - Smaller coefficients
 - Flatter prediction curves
 - Coefficients can change sign
- Coefficients can go to 0
 - Can happen at very small lambda
- Large λ will push all coefficients to 0



Regularised Regression and Small Datasets

Regression Data Requirements

- Usually, we would like to have more data points than parameters
- If we don't have this, direct solutions to fit a regression function will fail
- However, gradient descent can be used to find a solution
 - Allows us to fit high dimensional models to small datasets
 - Increases the danger of overfitting
- In general, extrapolation with linear regression can be risky



Cartoon from XKCD

Demo

- See ***CAB420_Regression_Example_3_Regression_with_Less_Data.ipynb***
- Traffic time prediction again, but with very limited data
 - 50 samples total
 - 30 training, 10 validation, 10 testing
 - ~150 variables
- Linear model will overfit
- Lasso and Ridge can be used to get a better fit to the data
- Review this example in your own time
 - Covered in more detail in the interactive session

CAB420: Classification

WHAT IS IT?

Classification

noun: the action or process of classifying something.

Or

- Supervised learning method
- Learn a model to separate two (or more) classes of data, and then given a new sample, output the class to which it belongs
 - Possibly with some sort of measure of confidence

Classification

- Binary Classification
 - Two class classification
 - Diseased / not diseased
 - Faulty / not faulty
 - ..
- Multi-Class Classification
 - N classes
 - Object classification (chair, desk, ferret, otter)
 - Types of faults
 -

Classification

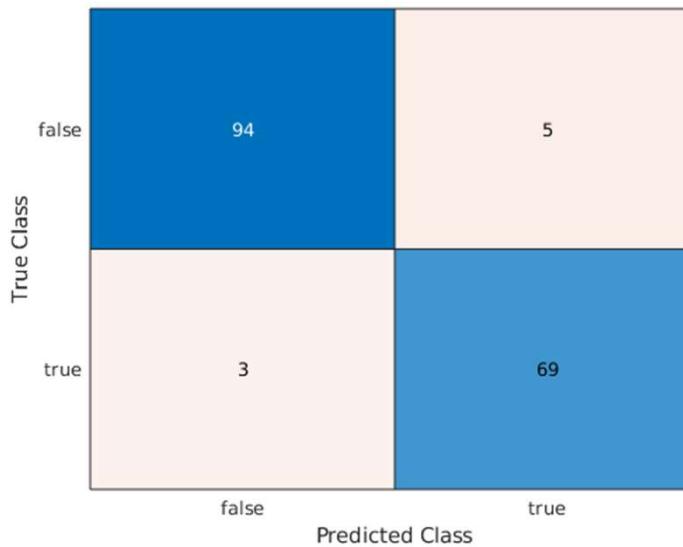
- Lots of methods for both binary and multi-class
 - Logistic Regression
 - Clustering
 - SVMs
 - Deep Networks
 - Trees
 - Ensembles of models
 - ...
- We'll focus on
 - SVMs
 - K-Nearest Neighbours Classification (CKNN)
 - Random Forests

Measuring Classification Accuracy

DOES MY CLASSIFIER WORK?

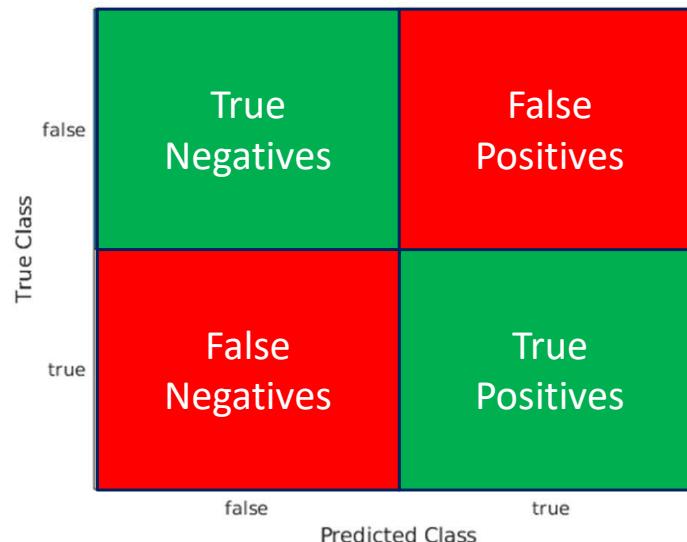
Confusion Charts (or Confusion Matrices)

- Often used to show classification accuracy
 - Vertical axis shows the true class
 - Horizontal axis shows the predicted class
- The diagonal represents correct results
 - We want as much of the data as possible on the diagonal
- Points off the diagonal indicate errors
 - And which class is confused with which other class



Confusion Charts and Types of Errors

- True Positives
 - Samples that are correctly predicted as True (1)
- True Negatives
 - Samples that are correctly predicted as False (0)
- False Positives
 - Samples that are predicted as True (1), but are actually False
- False Negatives
 - Samples that are predicted as False (0), but are actually True



Classification Accuracy

- Simplest way to measure classification performance
 - How often the classifier is correct
- Can be expressed as:

$$A = \frac{TN + TP}{TN + TP + FN + FP}$$

where TN, TP, FN, and FP are the number of true negatives, true positives, false negatives, and false positives

- Can also be viewed as the sum of the diagonal of the confusion matrix, divided by the sum of the entire matrix
- Provides a single number to indicate performance
 - But lacks nuance

Precision, Recall and F1 Score

- Precision

$$P = \frac{TP}{TP + FP}$$

- Recall

$$R = \frac{TP}{TP + FN}$$

- F1 Score

- Harmonic mean of precision and recall

$$F1 = \frac{2PR}{P + R}$$

Precision, Recall and F1 Score

- Compared to accuracy, provide a more detailed analysis of how a model is performing
- But, three values to analyse rather than one
- Need to consider what is most important
 - Minimising false negatives?
 - Use recall
 - Minimising false positives?
 - Use precision
 - Overall performance
 - Use F1 score

Metrics and Multi-Class Classification

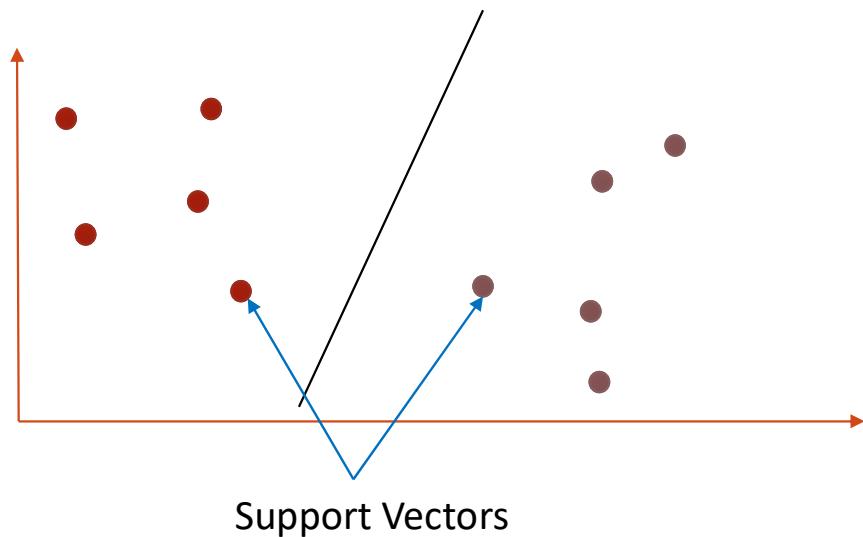
- Metrics are broadly similar when dealing with multi-class classification
 - Accuracy can be computed using the sum of the diagonal of the confusion matrix, divided by the sum of the entire matrix
 - Precision, Recall and F1 can be computed per-class
 - Allows for analysis of performance across individual classes
 - Need to be mindful of the number of samples in each class when considering overall performance
- See ***CAB420_Classification_Example_3_Classification_Metrics.ipynb*** for an example

CAB420: Support Vector Machines (SVMs)

WHAT'S A "SUPPORT VECTOR" ANYWAY?

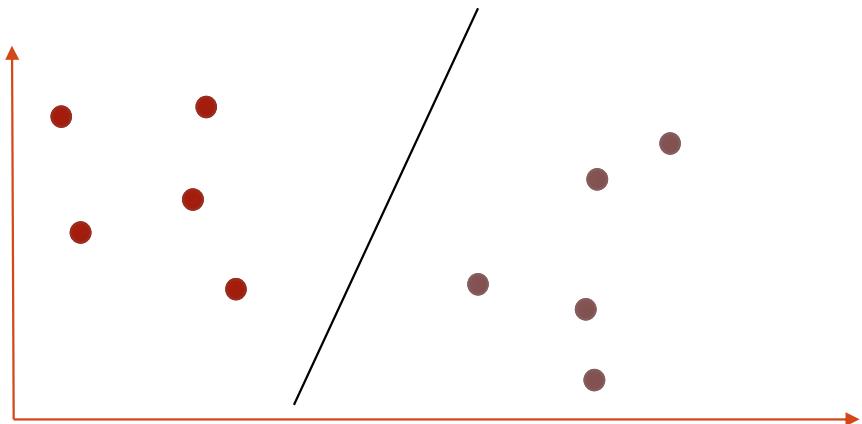
Support Vector Machines

- Supervised machine learning method
 - Finds a hyperplane that best divides a dataset into two classes



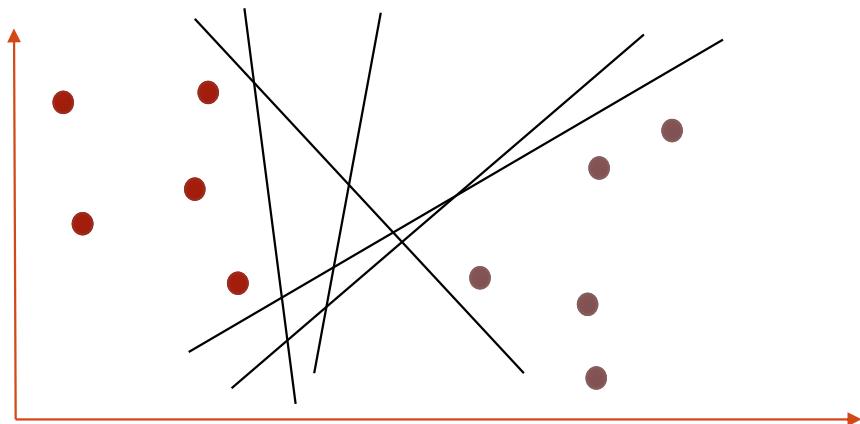
The Hyperplane

- Separates the two classes
- Defined by the support vectors
 - Remove the support vectors, change the hyperplane
- We are more confident about points far from the hyperplane



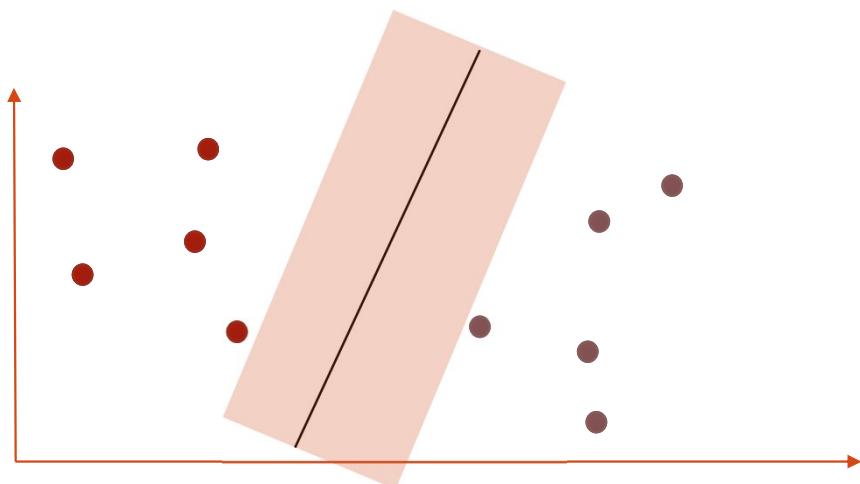
Which Hyperplane?

- What is there are multiple possible planes?
 - All work
 - Which is best?



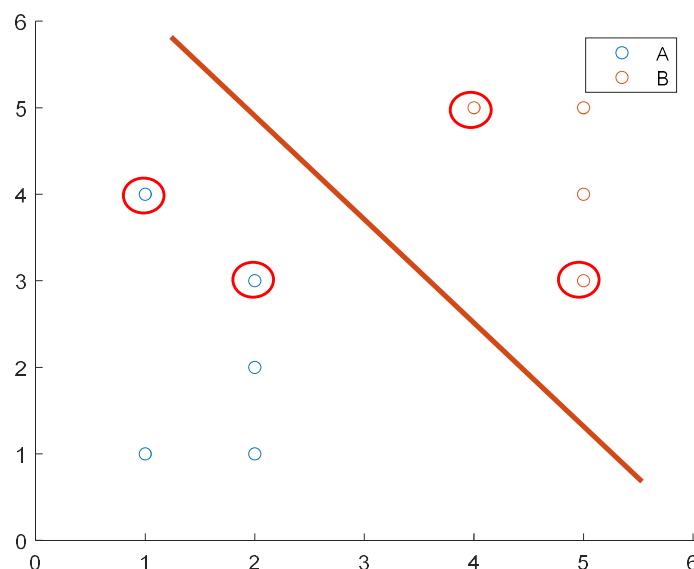
Finding the Hyperplane

- We seek to maximise the margin between the hyperplane and the points
 - Greater margin, greater confidence in prediction



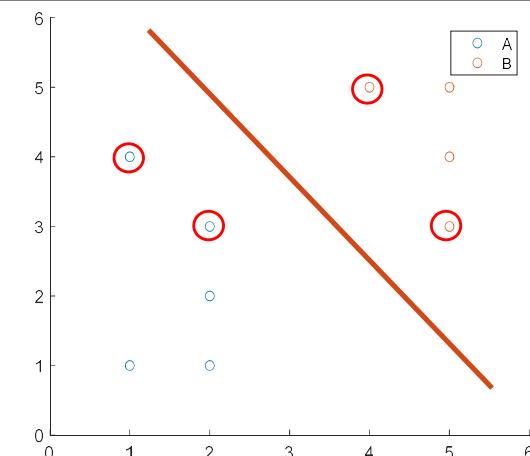
Finding the Hyperplane

- Consider the below problem
 - Two classes, clear separation between them



Support Vector Machines

- Consider an overlay where all class one values are negative and all class two values are positive.
- We can use vectors to show this mathematically.
- Consider some vector, w , which will represent a line **perpendicular** to the hyperplane.
- We define the hyperplane by the equation
$$w \cdot x + b = 0$$
- where w is the unknown vector perpendicular to our hyperplane, x is a set of dimension values and b is the error term.



Support Vector Machines

- Not enough constraints to find w or b . In order to help, we add two artificial constraints:

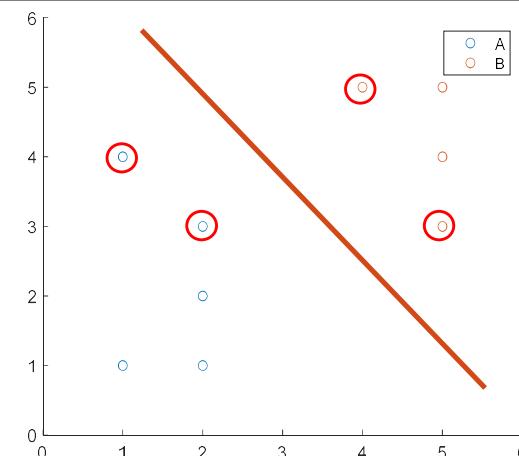
$$1. \quad w \cdot x_+ + b \geq 1$$

$$2. \quad w \cdot x_- + b \leq -1$$

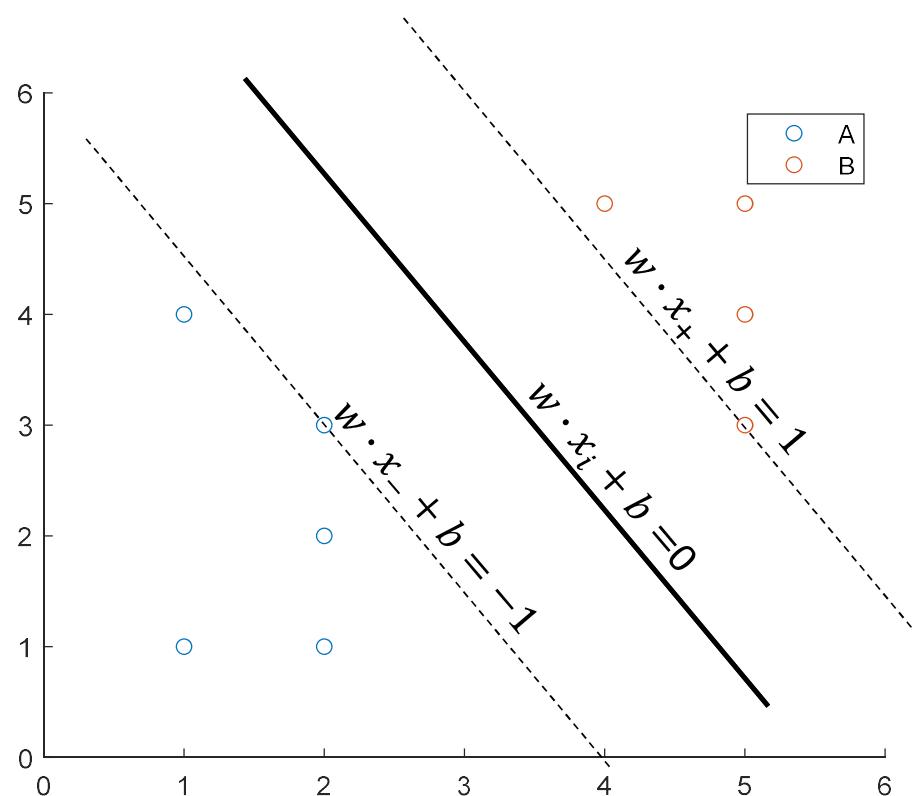
- We can then create a hard decision variable which tells us whether a value is classified as **positive** or **negative**:

$$y_i = \begin{cases} 1 & \text{for } w \cdot x_+ + b \geq 1 \\ -1 & \text{for } w \cdot x_- + b \leq -1 \end{cases}$$

- Then $y_i \times (w \cdot x_i + b) \geq 1$.
- When $y_i \times (w \cdot x_i + b) = 1$, x_i is a **margin point**



Support Vector Machines

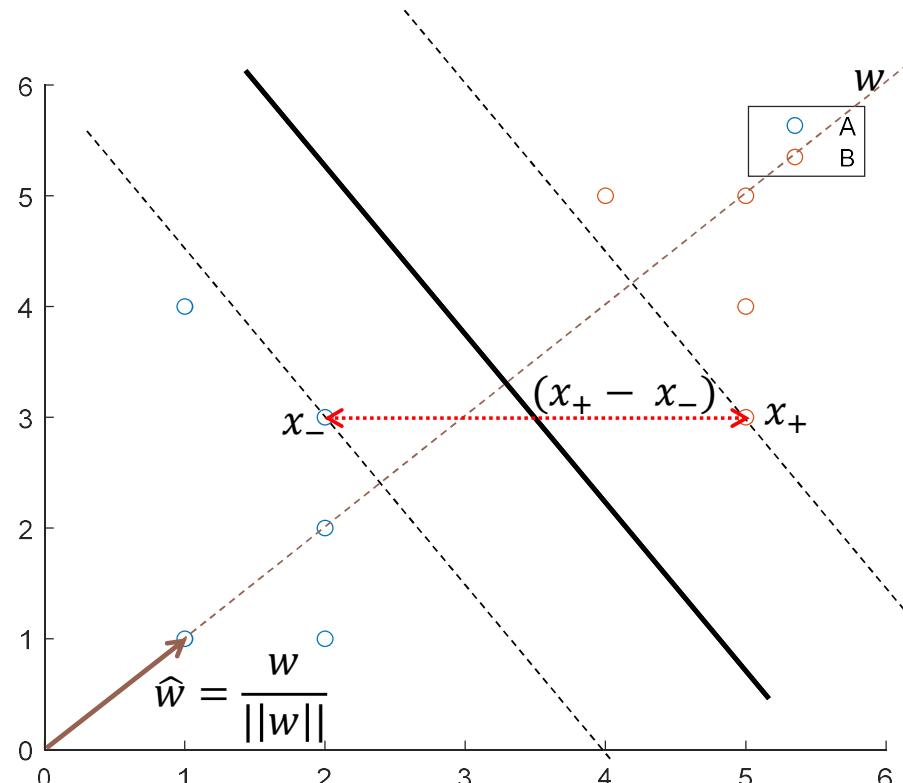


Support Vector Machines

- Remember, the optimal hyperplane is found by **maximising** the margin between two classes of data.
- We can find the width mathematically by looking at a single margin point from each class
- Consider two margin points, x_+ and x_- . The magnitude of the difference between them can be used to find the **width** of the margin by taking the dot-product of the result with the unit vector, \hat{w}
 - \hat{w} is the unit vector normal to the hyperplane
- The width of the margin is therefore $\frac{2}{\|w\|}$.
- Maximising $\frac{2}{\|w\|}$ is the same as minimising $\frac{\|w\|}{2}$.

Support Vector Machines

- We want to determine $(x_+ - x_-)$, but in the direction perpendicular to the hyper-plane
- w is a vector perpendicular to the hyper-plane, $\hat{w} = \frac{w}{\|w\|}$
- Width = $(x_+ - x_-) \frac{w}{\|w\|}$



Support Vector Machines

- Width =

$$(x_+ - x_-) \frac{w}{\|w\|}$$

- Remember

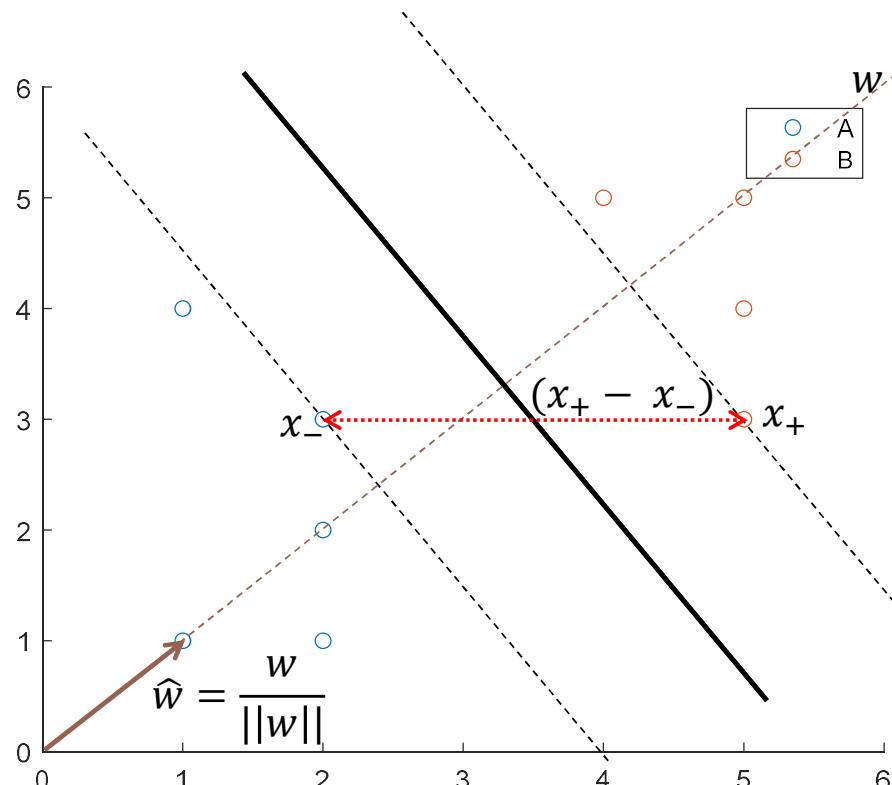
$$y_i = \begin{cases} 1 & \text{for } w \cdot x_+ + b \geq 1 \\ -1 & \text{for } w \cdot x_- + b \leq -1 \end{cases}$$

- And that all our x_+ and x_- are on the margins, therefore

$$w \cdot x_+ = 1 - b$$

$$w \cdot x_- = -1 - b$$

- Width = $(1 - b + 1 + b) \frac{1}{\|w\|} = \frac{2}{\|w\|}$



Maximising the Margin

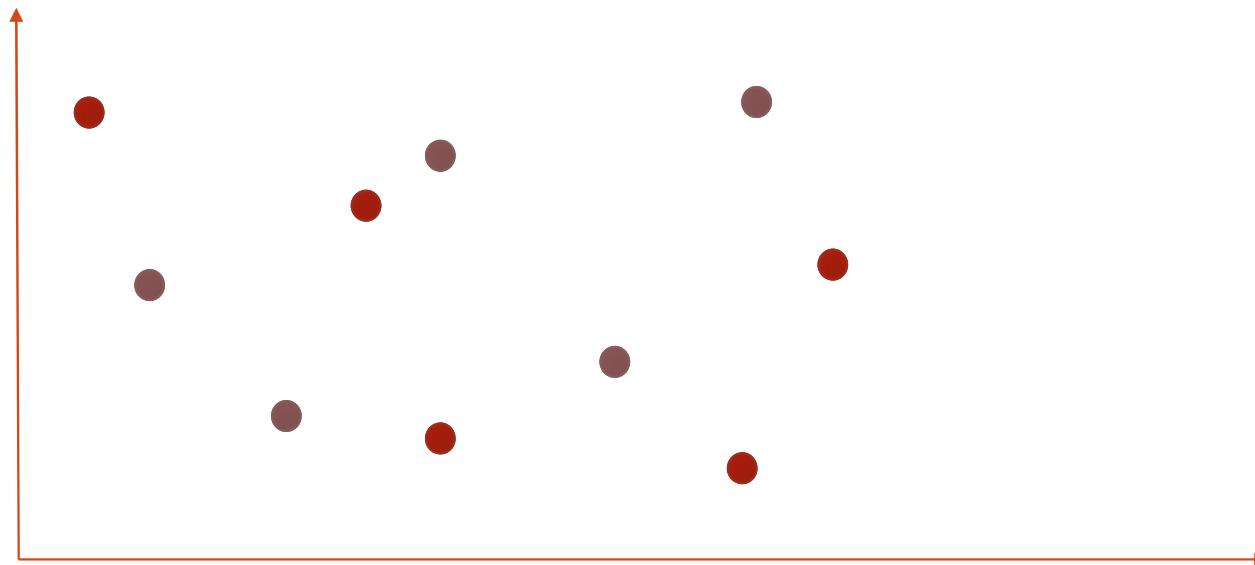
$$\begin{aligned} & \min_{w,b} \frac{\|w\|}{2} \\ \text{st. } & y_i(w^T x_i + b) \geq 1 \end{aligned}$$

Meaning

- Maximise the distance, while ensuring that points for different classes fall either side of line
- Hard-Margin SVM
 - Class separability is not optional; if this is not satisfied, the model will not converge
 - Solution involves Lagrange multipliers and is convex, i.e.; we will never hit a local minimum (if we do solve, we will always find the absolute best solution).
- You may also see this written as
 - $\min_w \frac{\|w\|}{2} + \sum_i \max\{0, 1 - y_i w^T x_i\}$
 - $\sum_i \max\{0, 1 - y_i w^T x_i\}$ is often referred to as the “Hinge Loss”

But...

What if our data is not linearly separable?



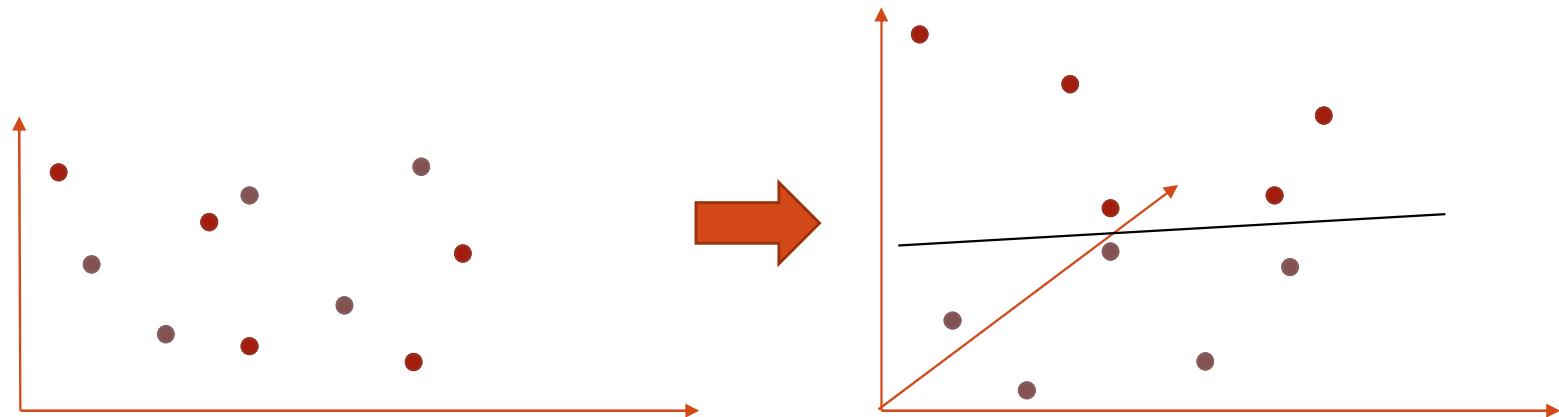
Option 1: Slack Variables

$$\begin{aligned} & \min_{w,b} \frac{\|w\|}{2} + C \sum_{i=1}^N \xi_i \\ \text{st. } & y_i(w^T x_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned}$$

- For each sample, we have a slack variable ξ_i
 - Influence of ξ_i controlled by C
- Basically, we allow the optimisation to misclassify some points
 - “Soft Margin” SVM
 - $C = \infty$ gives a hard margin SVM

Option 2: Transforming the Data

- Map the data to a higher dimensional space
 - Larger space, greater chance of the data being separable
- Data transform achieved via the “kernel trick”
 - Does not actually transform the raw data
 - Transformation occurs during the optimisation
 - Can be performed very efficiently



Kernel Tricks

- The term kernel refers to the method of using a linear classifier to solve a non-linear problem.
- The solution to our optimisation problem contains a dot product of two points.
 - Not shown on previous slides, comes a few steps later
- No need to transform the data as well the line we're looking for and hence the optimisation. We can instead change only the data points being multiplied.
- This works because the dot product produces a scalar only - no transform of dimensions.
- The simplest kernel is a linear kernel, which takes our dot product $x_i \cdot x_j$ and replaces it with...

$$K(x_i, x_j) = x_i \cdot x_j$$

- This is our standard solution.

Kernels: RBF

- Known as both the radial basis function kernel or the Gaussian kernel.
- The RBF kernel takes our dot product $x_i \cdot x_j$ and replaces it with

$$K(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

- The $\frac{1}{2\sigma^2}$ is often replaced by a separate variable, γ , and the kernel becomes:

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$

Kernels: Polynomial

- The polynomial kernel is the sum of polynomial terms in ascending order up to order n .
- The polynomial kernel takes our dot product $x_i \cdot x_j$ and replaces it with

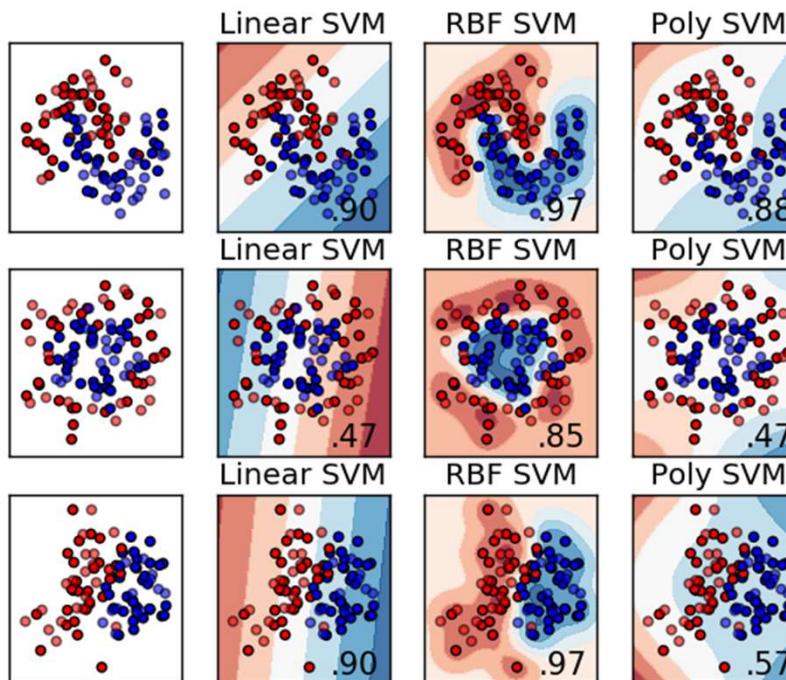
$$K(x_i, x_j) = (x_i \cdot x_j + 1)^n$$

Commonly Used Kernels

- Linear
 - Or, “no kernel”, leave the data as it is
 - Often the “default” kernel
 - With limited data risks overfitting
- Radial Basis Function (RBF)
 - One parameter: variance (or depending on formulation γ)
 - Defines how far a points influence extends
- Polynomial
 - Need to define the order
 - Generally harder to tune than the RBF

Commonly Used Kernels

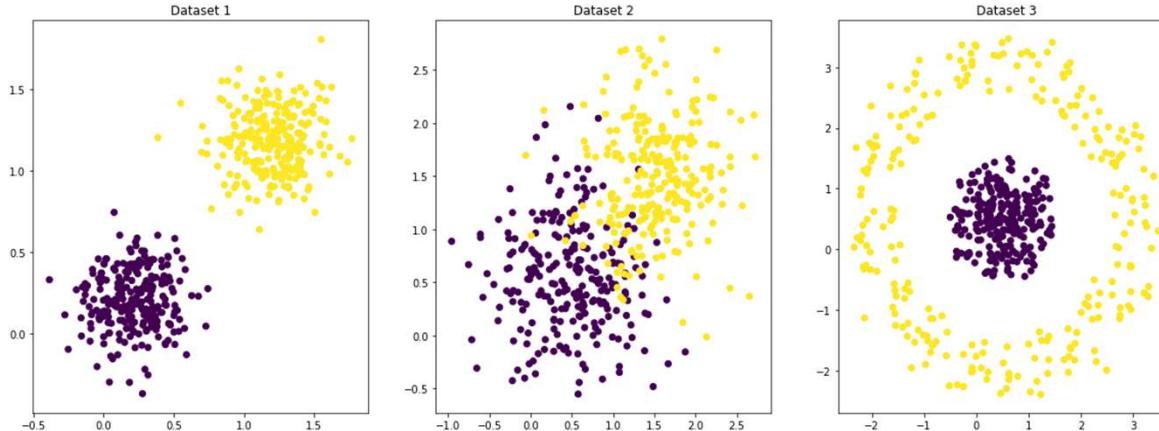
- There is no “best kernel”
 - Optimal kernel choice depends on the data
 - Expect to explore the data and/or experiment to find the best choice
- But...
 - RBF is usually a good starting point



From <https://gist.github.com/WittmannF/60680723ed8dd0cb993051a7448f7805>

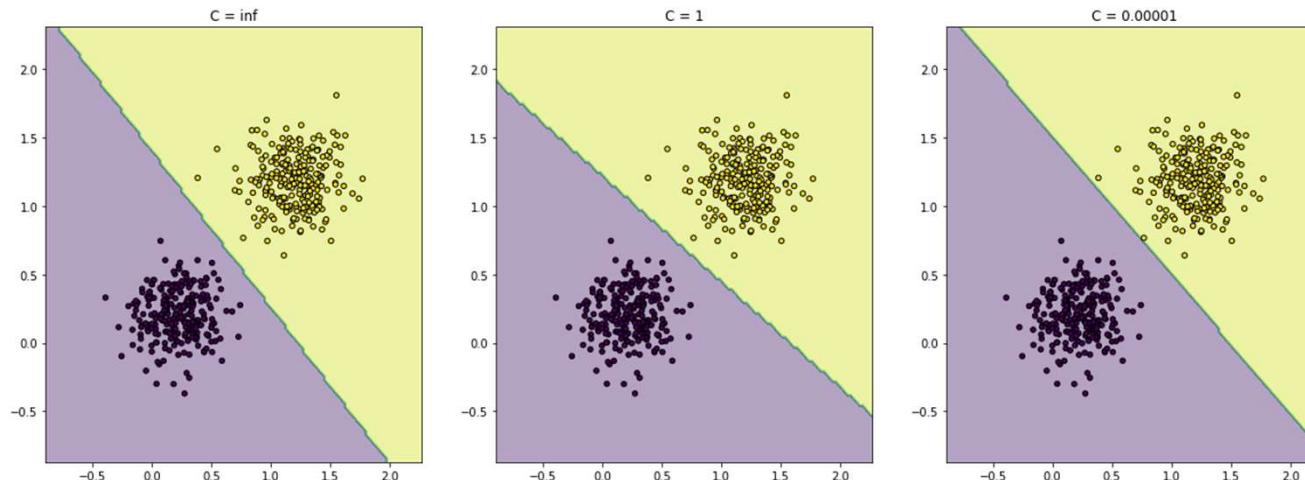
SVMs in Practice

- Visuals taken from ***CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb***
- We'll consider 3 sample datasets and explore SVM performance on these



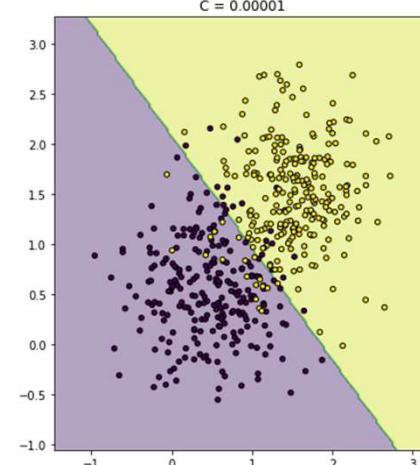
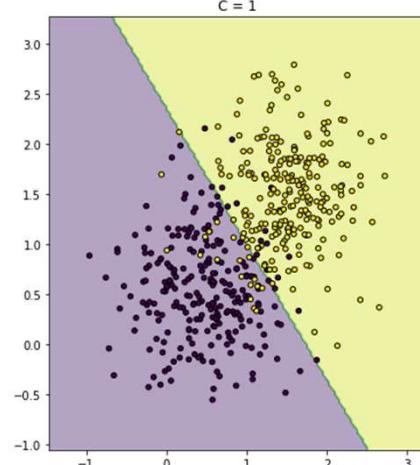
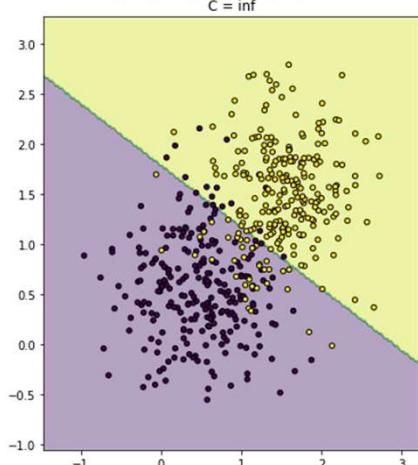
Dataset 1: Linear Kernel

- Varying C
 - Infinite C: optimal separation
 - C=1 and C=0.00001: good separation, but the line doesn't go straight through the middle (but goes close)



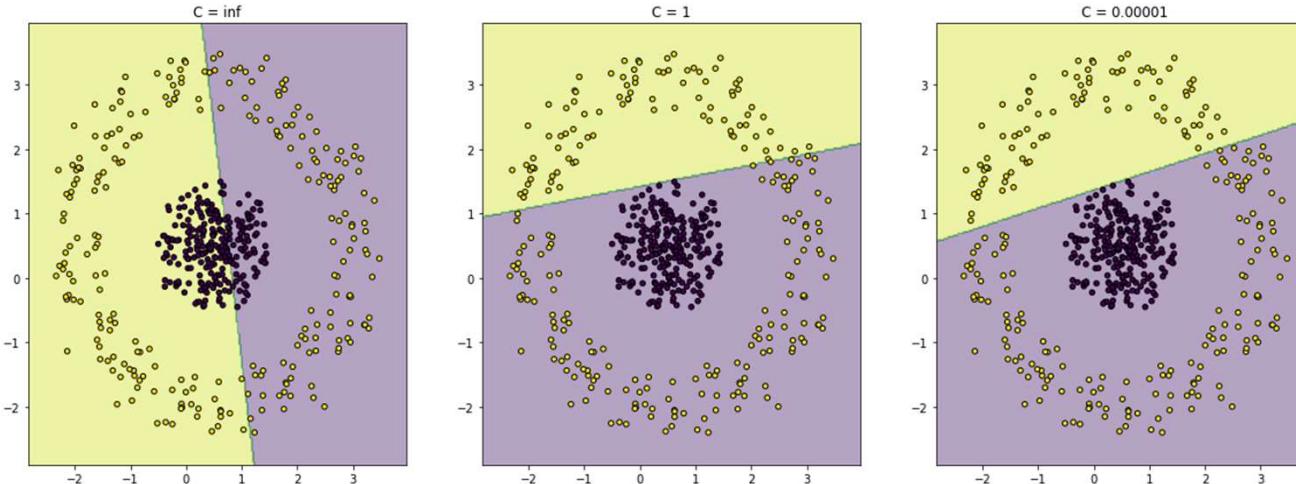
Dataset 2: Linear Kernel

- Varying C
 - $C = \infty$: Model fails to converge
 - **Data is not linearly separable**
 - $C = 1$ and $C = 0.00001$: model fits and finds a line though the middle. Roughly equal error on either side.



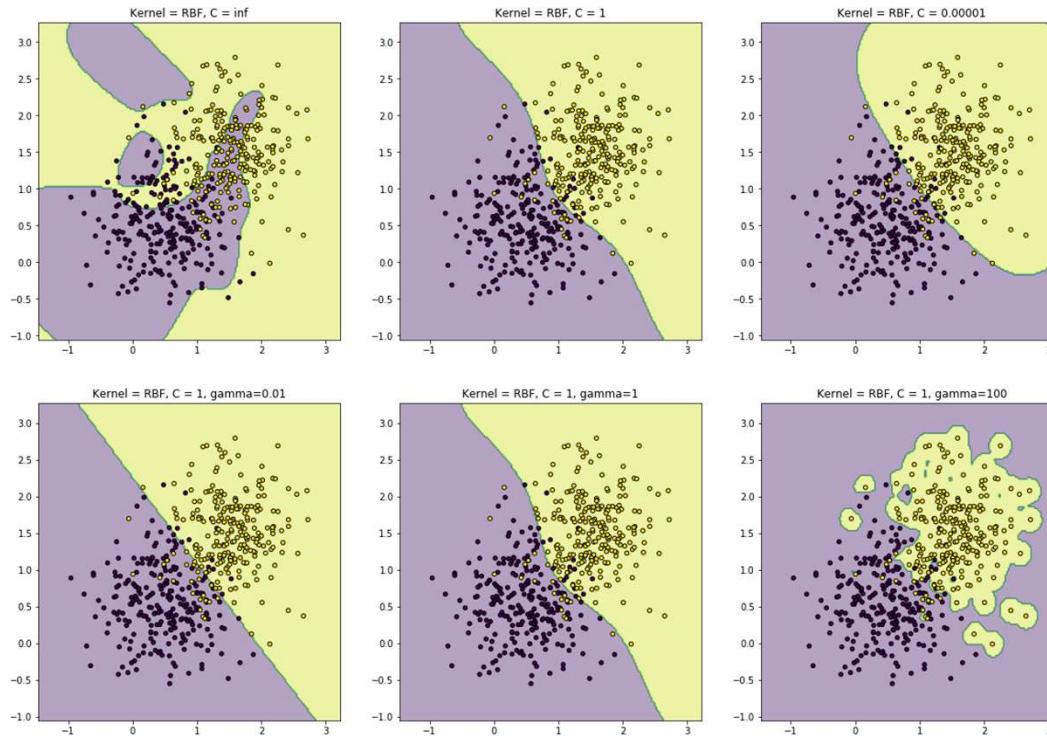
Dataset 3: Linear Kernel

- Varying C
 - All values of C do poorly, $C = \infty$ fails to converge
 - **Data is not linearly separable**



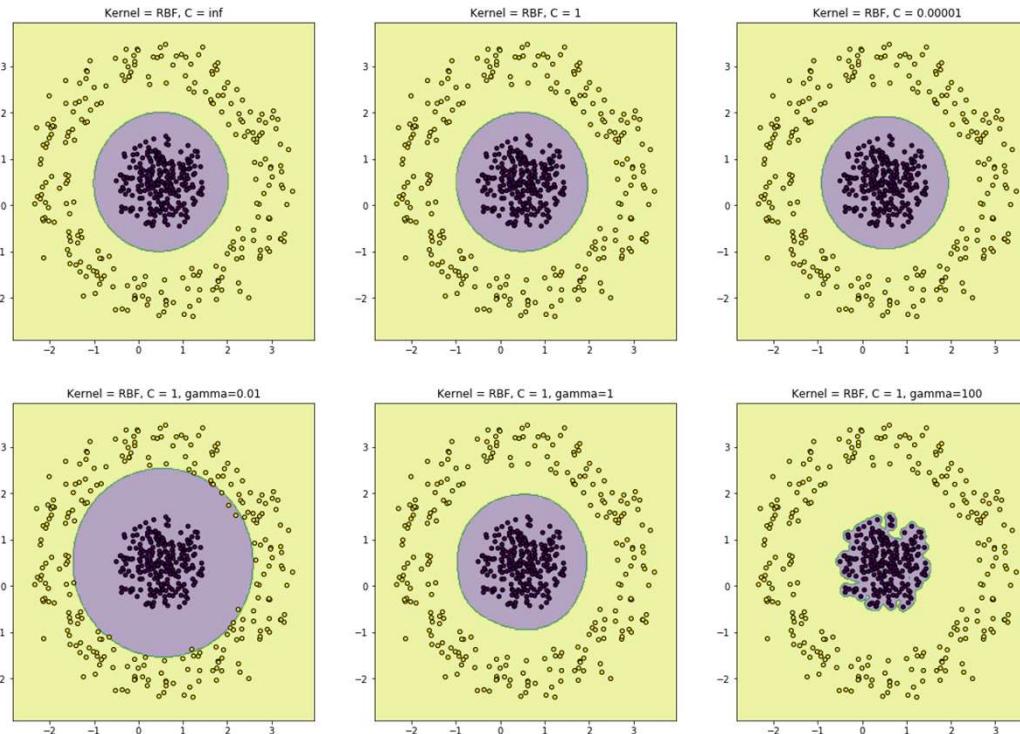
Dataset 2: RBF Kernel

- $C = \infty$ fails to converge
- γ is a measure of how far a point's influence extends. Small γ means influence extends further
- As γ increases, shapes become more complex
 - Small γ cannot capture complex shapes
 - Larger values of γ may lead to overfitting



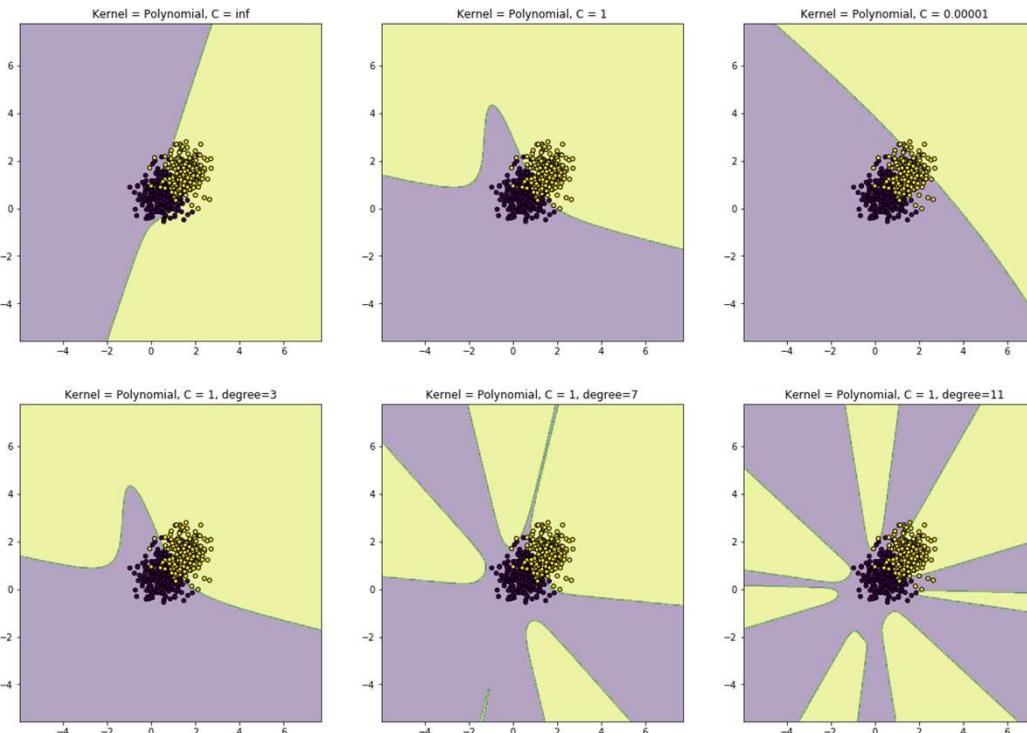
Dataset 3: RBF Kernel

- All configurations achieve a good boundary
 - $\gamma = 0.01$ makes some errors
- RBF kernel can exploit separation between the classes, even if that separation is not a straight line



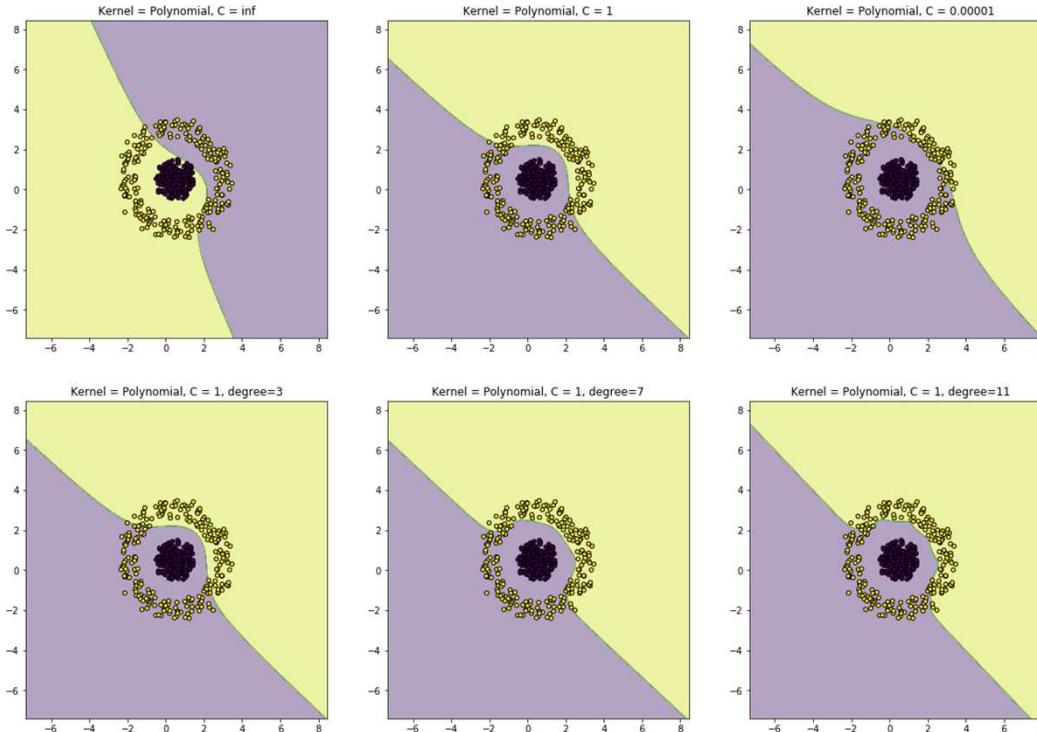
Dataset 2: Polynomial Kernel

- $C = \infty$ fails to converge
- $C = 0.00001$ is underfitting
- Boundaries are polynomial shapes (i.e. cubics, quartics, etc)
 - Degree defines polynomial order, and thus shape complexity
- 3rd degree kernel learns a reasonable boundary
 - But boundary behaves strangely outside of data range



Dataset 3: Polynomial Kernel

- $C = \infty$ fails to converge
- Higher degree kernels learn a better semi-circle, but can't properly separate the classes



SVMs and Hyperparameters

- C (also known as the box constraint)
 - C = infinite: hard margin, needs to be clear space between the classes
 - If C is too big (or infinite), model may fail to converge
 - If C is too small, the model can underfit (learn a poor decision boundary)
- Kernels:
 - Linear: default, good first choice
 - RBF: can fit to complex distributions, gamma used to tune
 - Polynomial: controlled by the degree (or order) of the line, can be hard to tune

K-Nearest Neighbours Classification

SIMILAR POINTS SHOULD BE THE SAME CLASS

K-Nearest Neighbours Classification

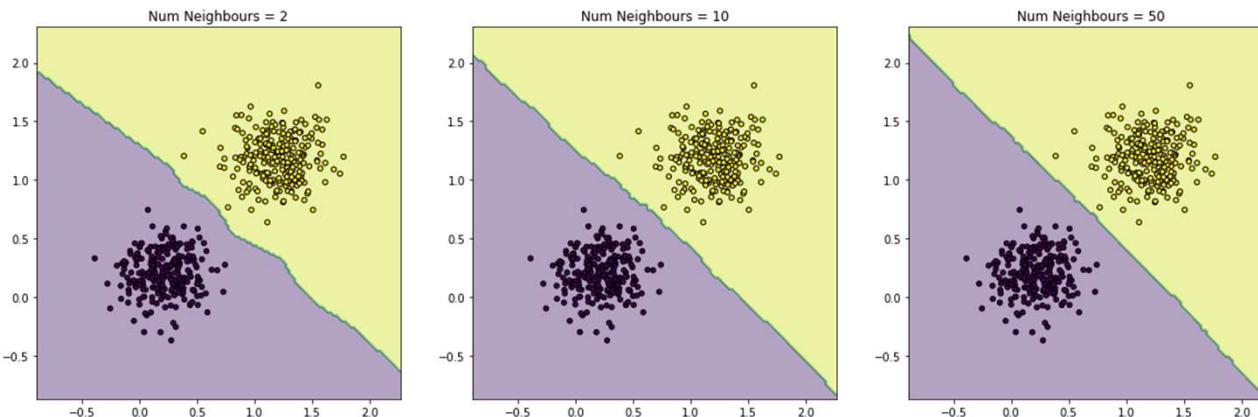
- Based on a simple intuition
 - Points of the same class should be near each other in some space
- What does this mean for classification?
 - Classify based on similar looking points
 - Requires our training data to contain diverse examples
 - Model won't adjust to unusual examples

KNN Classifier

- Input
 - Labelled data, i.e. our points, and the class they belong to
- At test time
 - New point comes in
 - Compare this point to all others
 - Find it's closest K neighbours, and the dominant class of those is the class of our input point

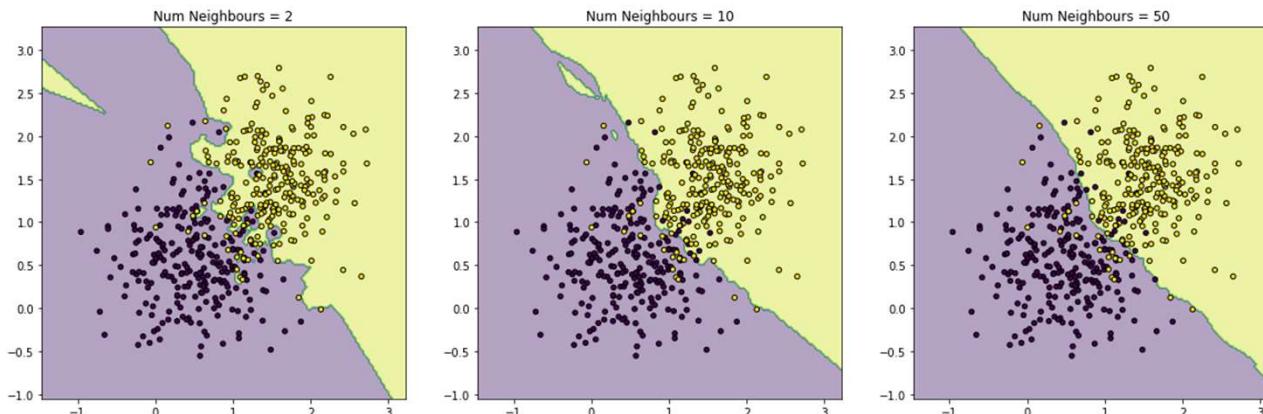
CKNNs in Practice

- See *CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb*
- Smaller K means more precise separation
 - Greater sensitivity to noise
- Larger K leads to a smoother boundary
 - Can cause problems when one or more classes are scarce



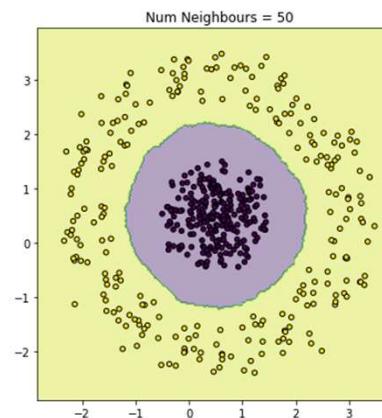
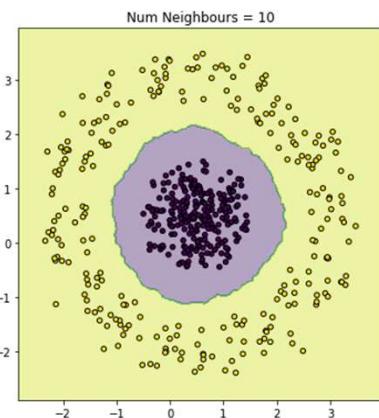
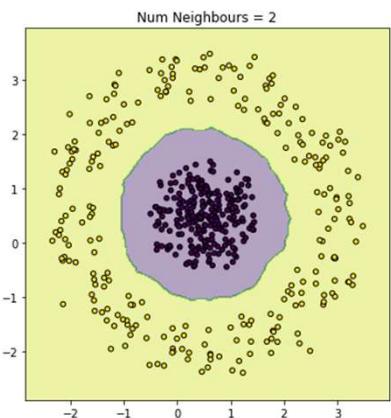
CKNNs in Practice

- See *CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb*
- Smaller K means more precise separation
 - Greater sensitivity to noise
- Larger K leads to a smoother boundary
 - Can cause problems when one or more classes are scarce



CKNNs in Practice

- See *CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb*
- Smaller K means more precise separation
 - Greater sensitivity to noise
- Larger K leads to a smoother boundary
 - Can cause problems when one or more classes are scarce



Distance Metrics

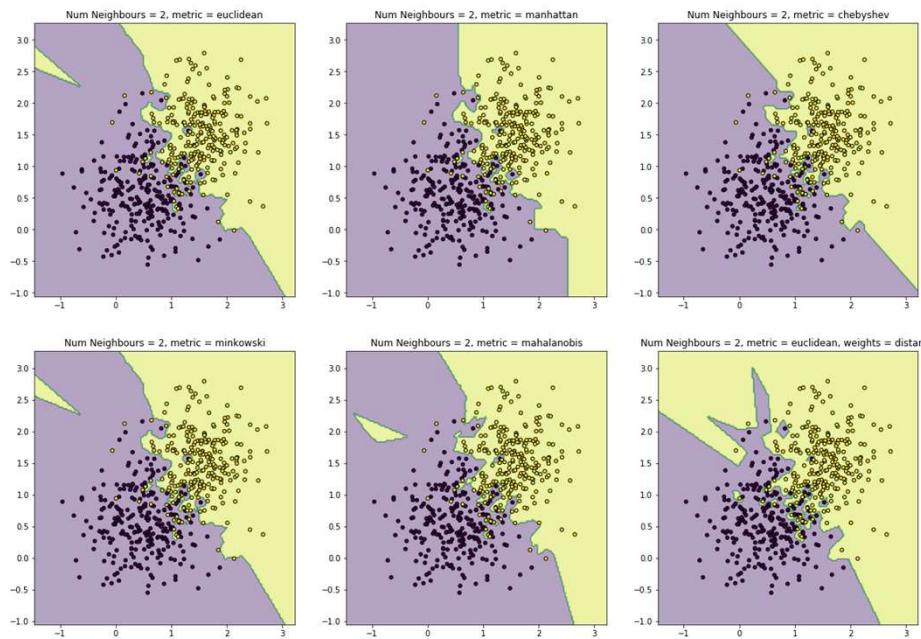
- CKNN is based around measuring how far apart two points are
- We need a way to measure distance
- Many possible options:
 - Magnitude based
 - Euclidean: $dist_E = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$
 - Manhattan: $dist_M = \sum_{i=1}^n |p_i - q_i|$
 - Cosine: $dist_C = 1 - \frac{p \cdot q}{\sqrt{\sum_{i=1}^n p_i^2} \sqrt{\sum_{i=1}^n q_i^2}}$
 - Angle between vectors
 - Useful when magnitude is not important

Distance Metrics

- More options:
 - Mahalanobis Distance: $dist_M = \sqrt{(x - y)^T S^{-1} (x - y)}$
 - S is the covariance matrix
 - Can be seen as a dissimilarity measure
 - Minkowski Distance: $dist_{min} = \left(\sum_{i=1}^n |x_i - y_i|^P \right)^{1/P}$
 - Hyper parameter P
 - Generalisation of Manhattan and Euclidean distances
 - Hamming
 - Binary distance, equivalent to Manhattan distance for binary vectors

Distance Metrics

- For all examples, K=2
 - Limited variation on this data
- Distance weighting method also important
 - Default is all points weighted equally
 - "distance" weights points inversely proportional to their distance
 - Closer points are more important



Random Forests

LOTS OF CLASSIFIERS, WORKING TOGETHER

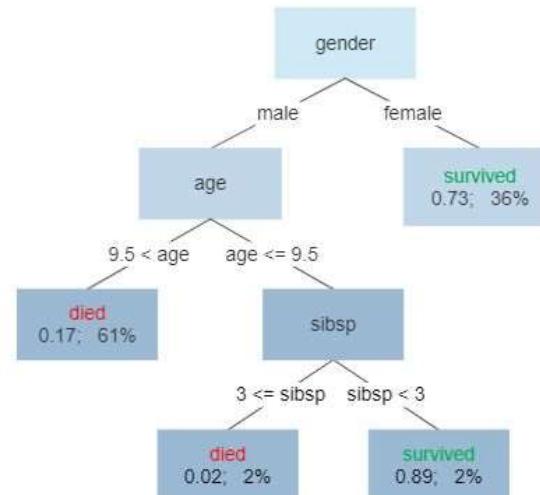
Random Forests

- An ensemble classifier
 - Combine many decision trees to achieve improved performance
 - Reduce variance with only a small increase in bias
 - Number of trees included in the forest is a hyper parameter
- Aim to have many uncorrelated classifiers
 - Uncorrelated classifiers can be averaged to improve performance and robustness to noise
 - Obtain uncorrelated classifiers by using
 - bootstrap aggregation
 - feature bagging

Decision Trees

- Classical Machine Learning/Data Science method
- Iteratively splits data based on a feature to obtain a classification result at the leaves of the tree
- Typically built up to a supplied maximum depth
- Deeper trees lead to
 - Greater accuracy on training data
 - Higher likelihood of overfitting

Survival of passengers on the Titanic



Decision Trees

- At each branch, a variable and threshold are selected to split the data, based on some criteria
- Common criteria
 - Gini impurity (python's default): how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. 0 when the subset contains a single label.
 - Information gain: related to entropy and KL-divergance

Bootstrap Aggregation (Bagging)

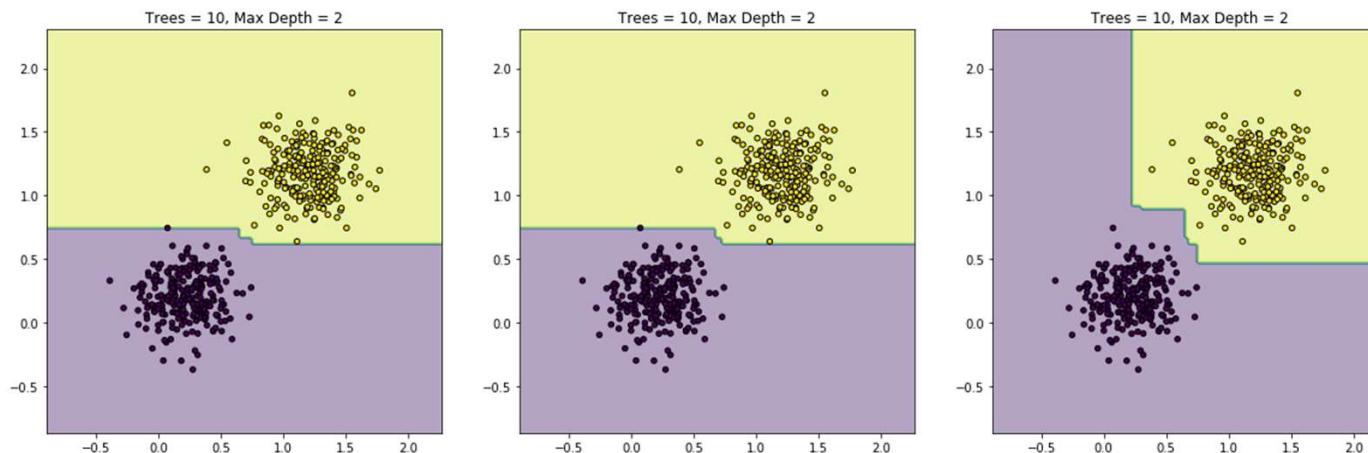
- We seek to build a set of **uncorrelated** trees
- If we use the same data each time however, we will get the same trees
- Bootstrap aggregation
 - Given a set of N predictors, $X = \{x_1, x_2, \dots, x_n\}$, and $Y = \{y_1, y_2, \dots, y_n\}$
 - Sample with replacement N points, X_b, Y_b Points can be repeated in the dataset
 - Repeat this B times, obtaining B sets, each of which trains a different tree

Feature Bagging

- If we have highly descriptive features, even with feature bagging we may end up with very similar trees as the same features are always selected to split the tree
- Feature Bagging:
 - At each branch, select a random subset of features and compute the split from that feature set
 - Size of random feature subset is a hyper parameter

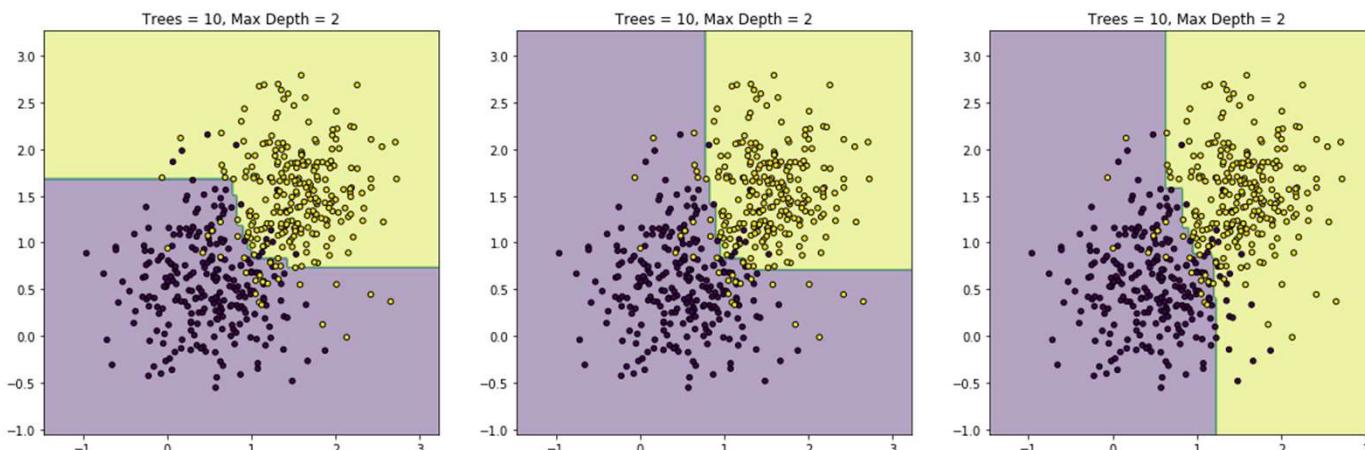
Random Forests in Practice

- See [*CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb*](#)
- Each row contains classifiers trained using the same settings and a different random seed
 - The word **random** is in the name for a reason
- As the number of trees increases, this variation will decrease



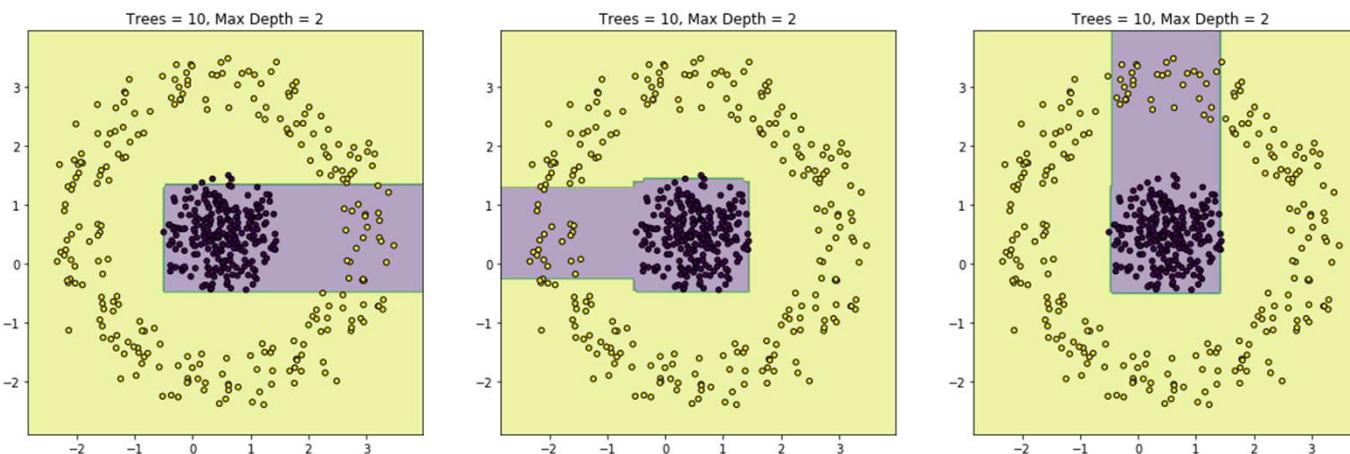
Random Forests in Practice

- See [*CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb*](#)
- Each row contains classifiers trained using the same settings and a different random seed
 - The word **random** is in the name for a reason
- As the number of trees increases, this variation will decrease



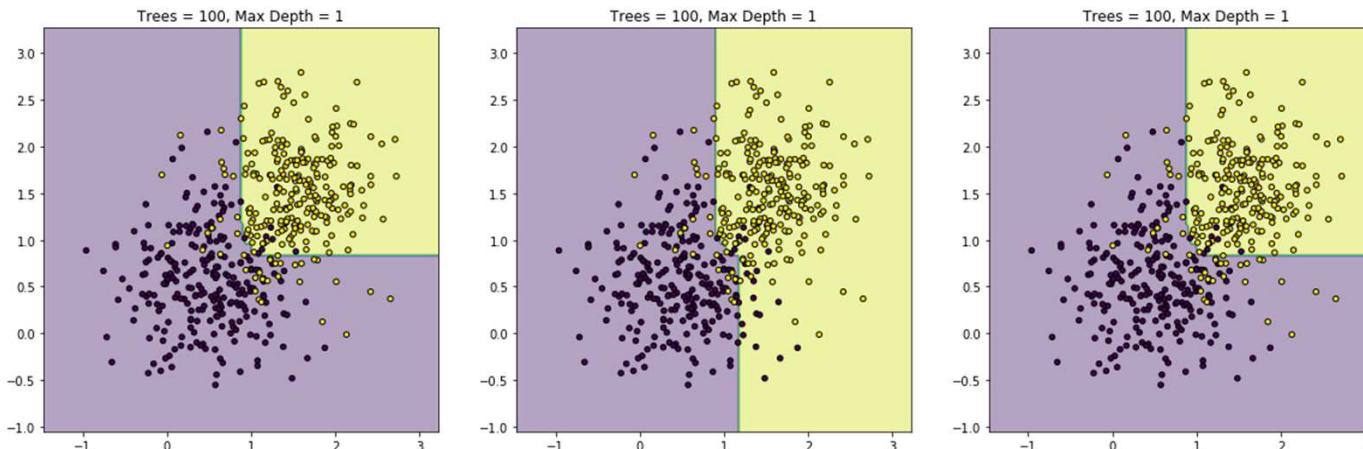
Random Forests in Practice

- See [*CAB420_Classification_Additional_Example_Classifier_Parameters_and_Decision_Boundaries.ipynb*](#)
- Each row contains classifiers trained using the same settings and a different random seed
 - The word **random** is in the name for a reason
- As the number of trees increases, this variation will decrease



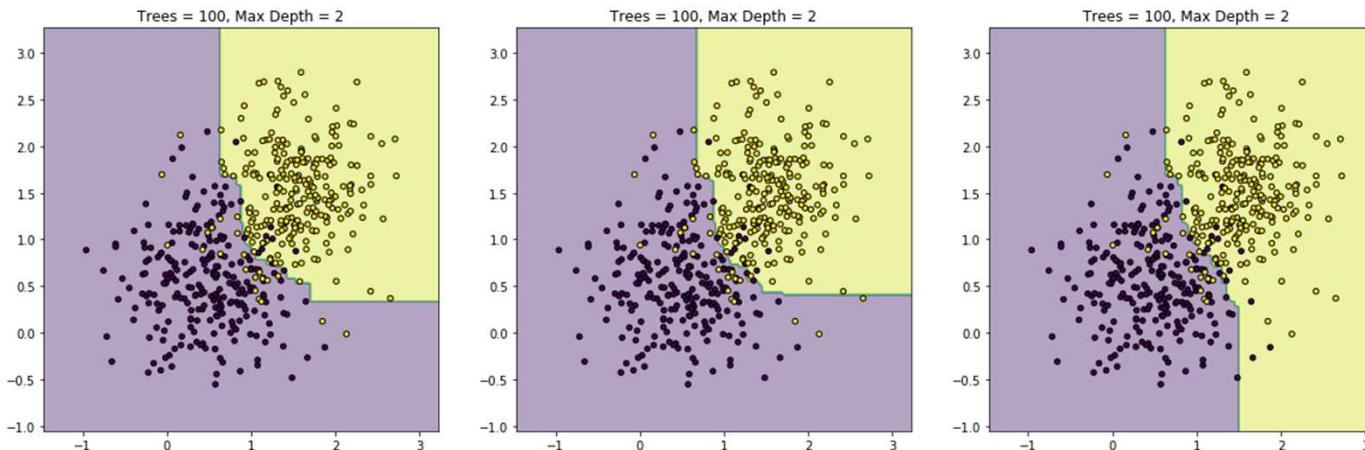
Tree Depth

- As we increase tree depth
 - Boundary complexity increases
 - Risk of overfitting increases
- Need sufficient tree depth to consider all classes/variables
 - Though not necessarily within a single tree



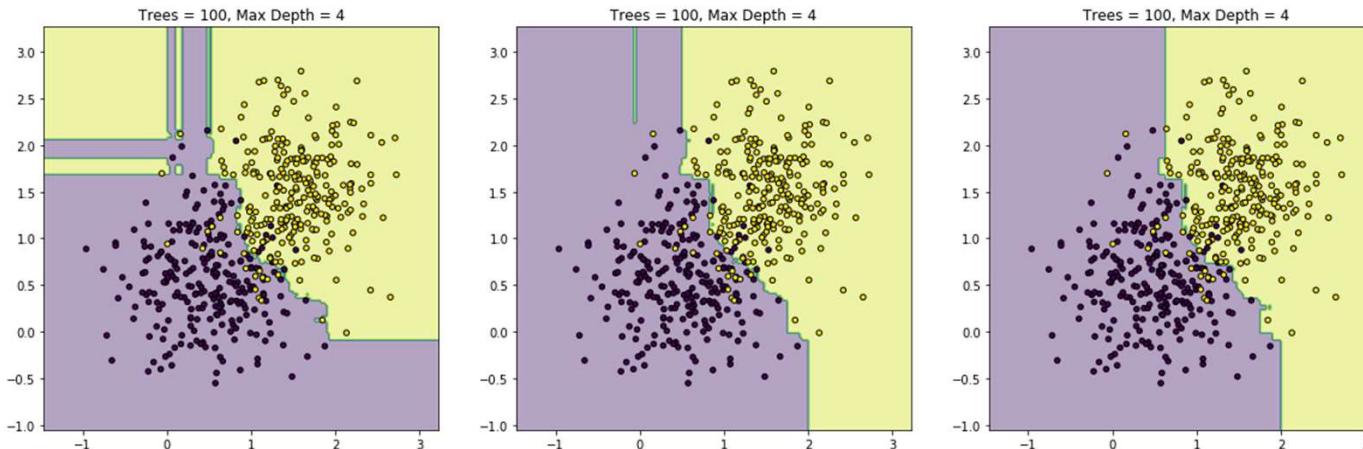
Tree Depth

- As we increase tree depth
 - Boundary complexity increases
 - Risk of overfitting increases
- Need sufficient tree depth to consider all classes/variables
 - Though not necessarily within a single tree



Tree Depth

- As we increase tree depth
 - Boundary complexity increases
 - Risk of overfitting increases
- Need sufficient tree depth to consider all classes/variables
 - Though not necessarily within a single tree



CAB420: Comparing Classifiers

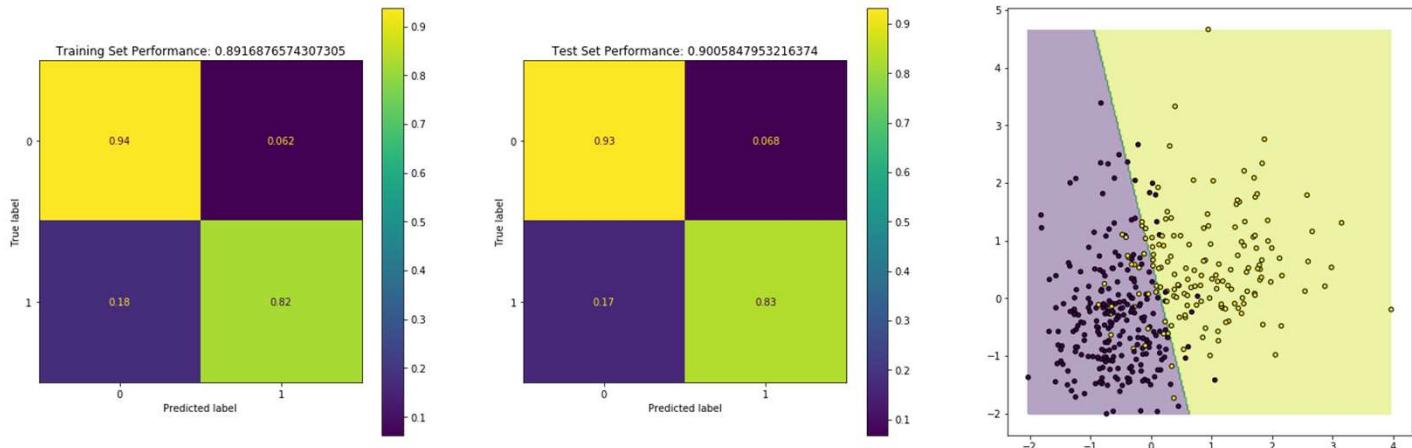
COMPARE THE PAIR (OR TRIO)

Binary Classification Example

- See ***CAB420_Classification_Example_1_Classification_Three_Ways.ipynb***
- Data:
 - Cancer cell data
 - Various measures of cells, and a label to indicate if the sample is malignant or benign
 - For visualisation simplicity, we'll use just two of ~30 dimensions
 - Data is standardised
 - Often helpful when distance measures are involved
 - Helps visualisations

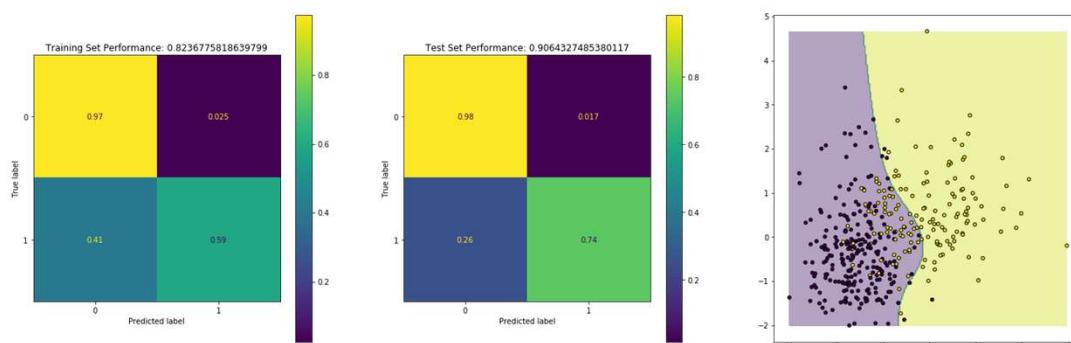
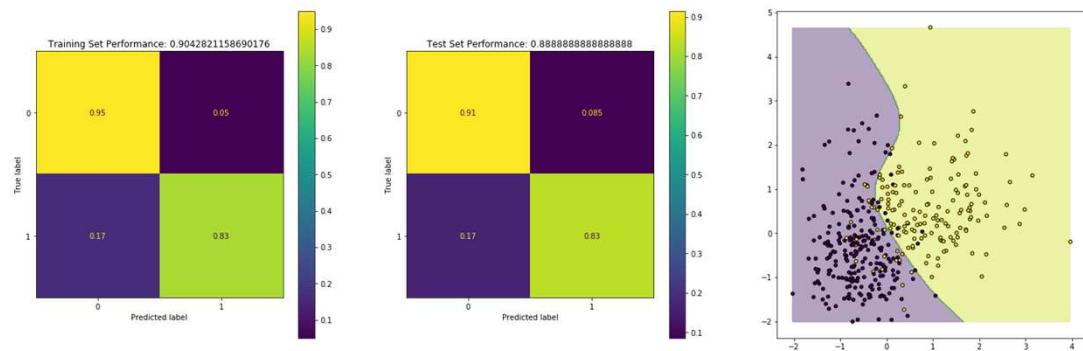
Linear SVM

- $C = 1$, Kernel = Linear
 - ~90% accuracy, no overfitting
 - Mild class accuracy imbalance



Polynomial and RBF SVM

- $C = 1$, default parameters
 - Top Right: RBF
 - Bottom Left: Polynomial
- Very similar to linear SVM
- How do we select the best parameters?

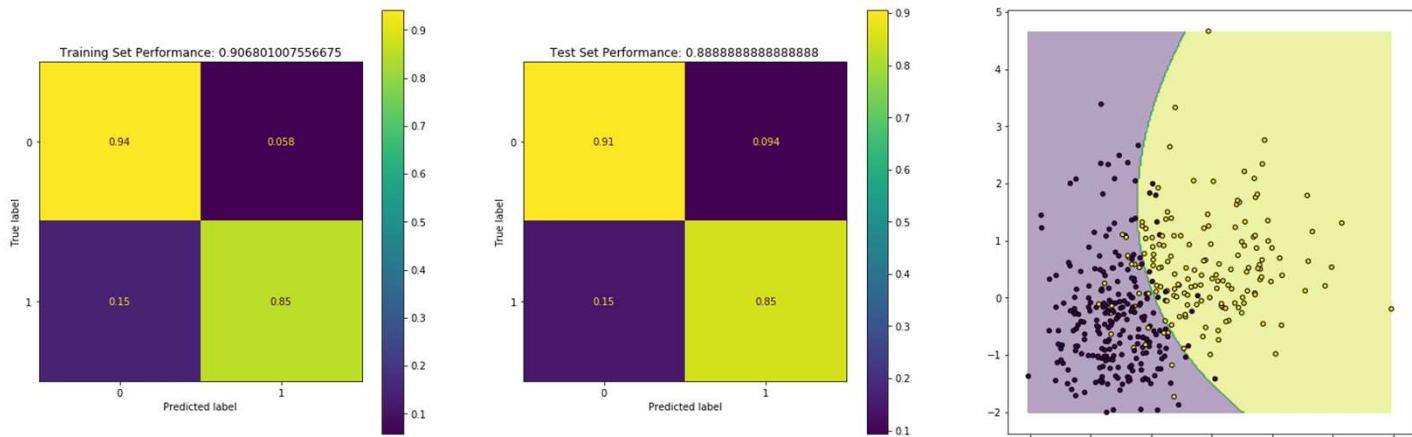


Grid Search

- Exhaustive search over a specified set of parameters
- We'll search over 3 grids, one per kernel:
 - `{'C': [0.1, 1, 10, 100, 1000], 'kernel': ['linear']}`
 - `{'C': [0.1, 1, 10, 100, 1000], 'gamma': [0.1, 0.01, 0.001, 0.0001], 'kernel': ['rbf']}`
 - `{'C': [0.1, 1, 10, 100, 1000], 'degree': [3, 4, 5, 6], 'kernel': ['poly']}`
- Can be very slow depending on grid size
 - Often performed in a coarse-to-fine manner

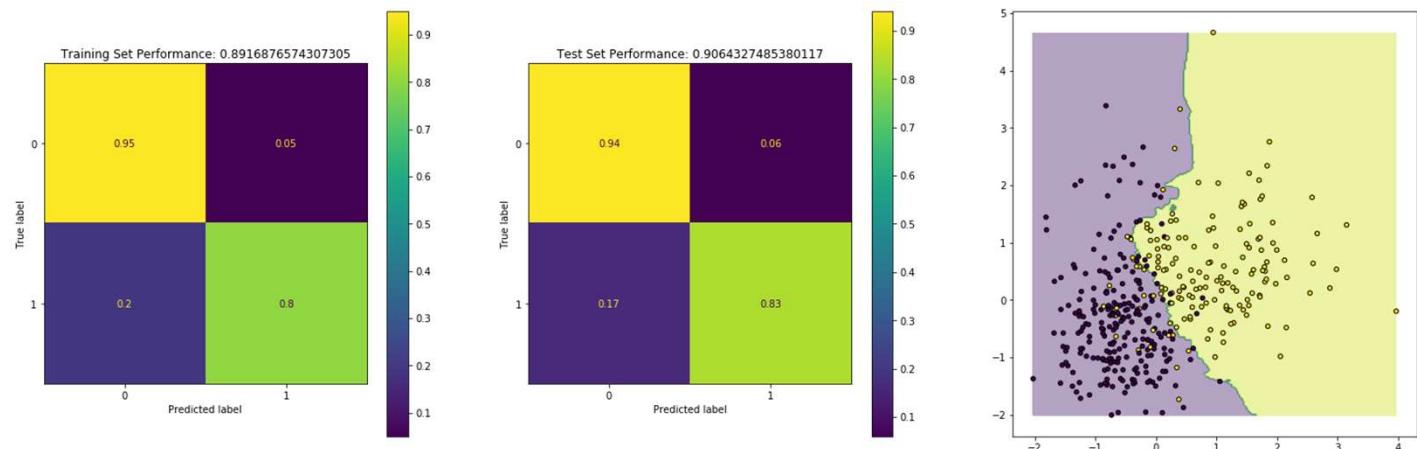
Optimal SVM

- RBF Kernel, $C = 100$, $\gamma = 0.01$
- Is this truly optimal?
 - Probably not
 - Could conduct a finer grained grid search around this point
 - Less accurate on the testing set than our original linear model



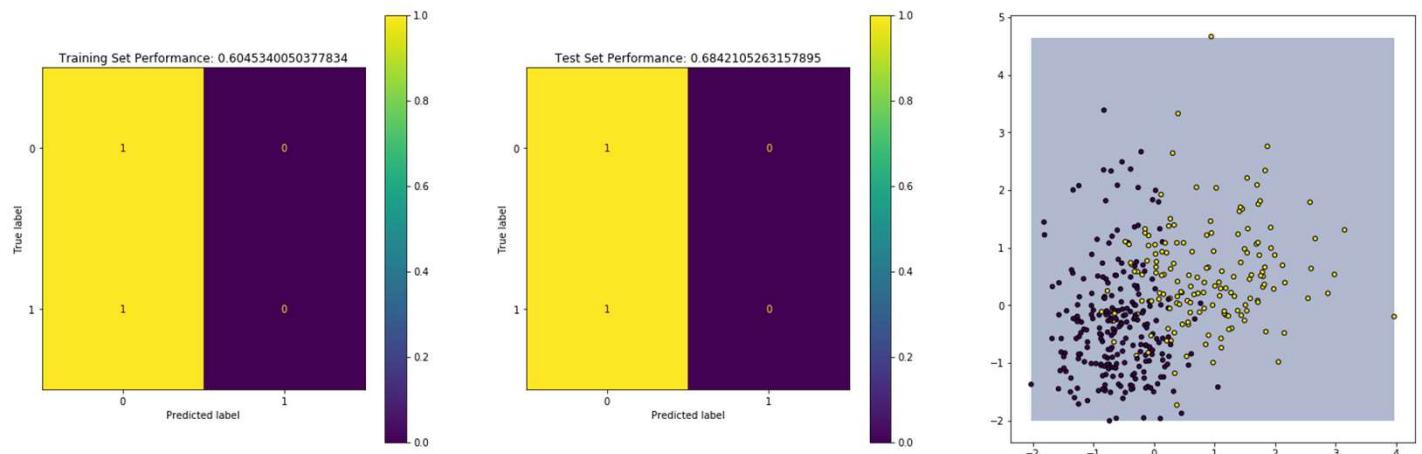
CKNN

- 10 Neighbours, uniform weights
 - Good performance, very similar to SVM
 - Different looking decision boundary compared to SVM



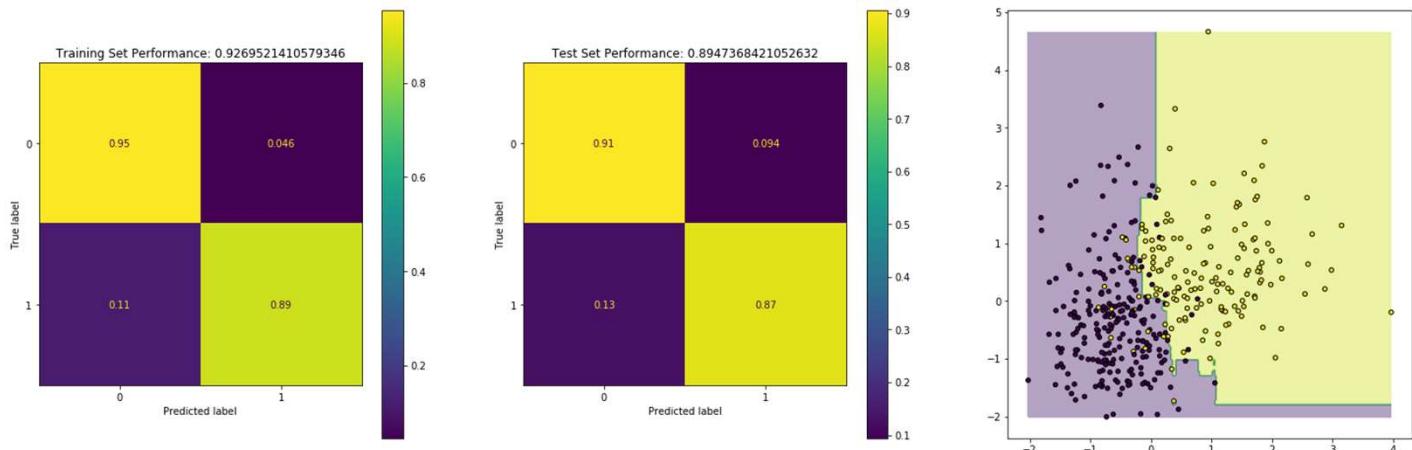
CKNN Dangers

- Need to be careful with selection of K
- K=350
 - Can't classify anything as class 0
 - This is an extreme example, but when you have class imbalance this can be a problem



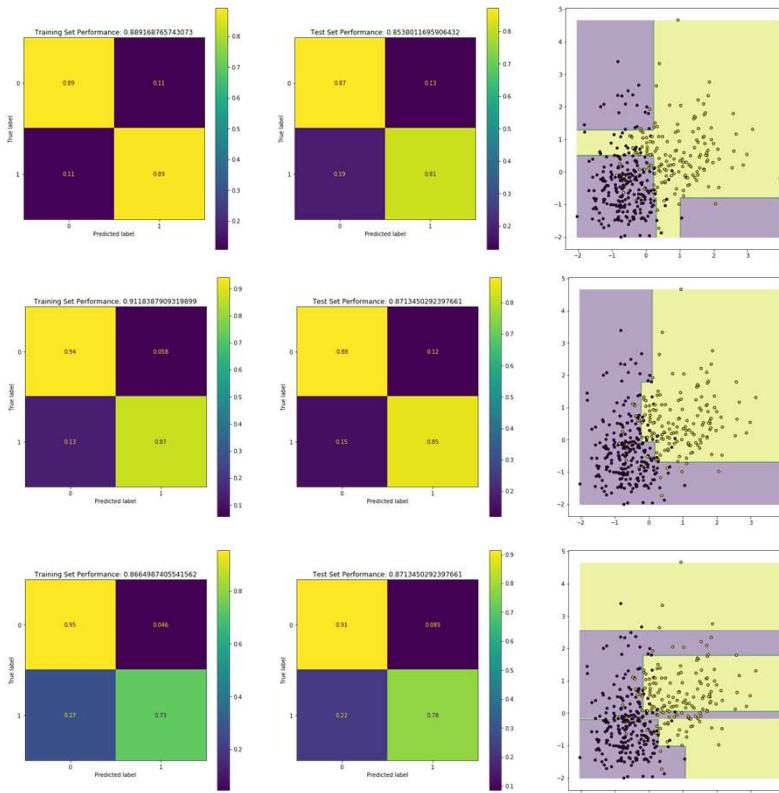
Random Forests

- 100 trees, max depth of 4
 - Similar performance to other models
 - Similar looking decision boundary



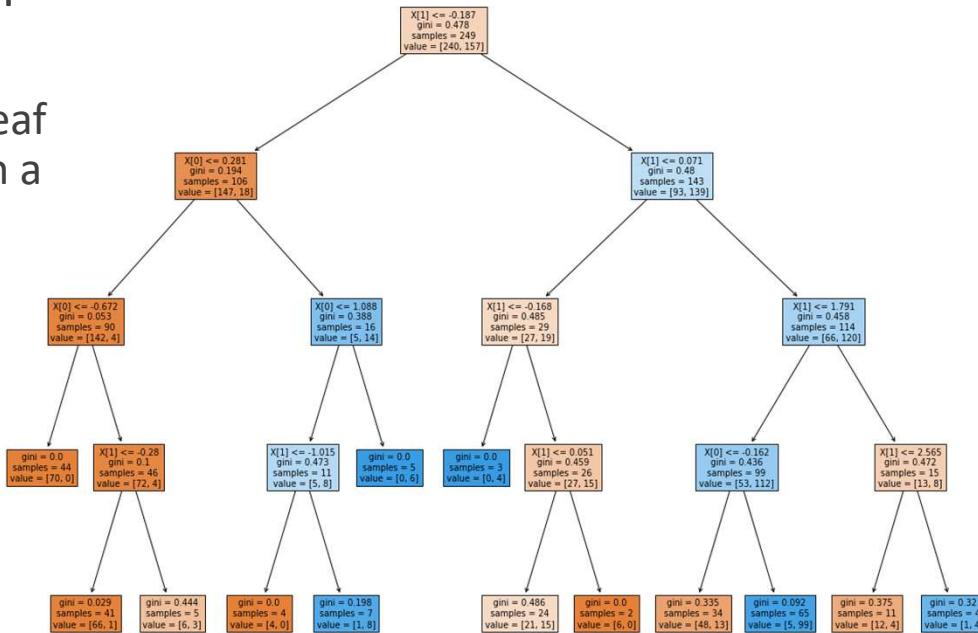
Variation in the Forest

- Three trees from the same ensemble
 - Very different decision boundaries



Visualising the Trees

- Decisions can be visualised
- ~Half of the leaf nodes contain a single class
- Gini=0



Hyper Parameter Optimisation

- We've already discussed grid search
- Randomized Search
 - Randomly sample parameter combinations, attempting to maximise overall accuracy (parameter score)
 - Typically allows for a quicker search than grid search
 - Though not as exhaustive
 - Can use as an initial search followed by a more fine-grained search
- Halving Grid Search
 - Like grid search but operates over several iterations
 - First iteration trains all models with a small amount of data
 - Top half of models are kept, trained on more data
 - Repeats until best two models are trained on all data
 - More efficient than Grid Search, but can be problematic on small/unbalanced datasets
- There are a number of other methods too, see https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

Which method is best?

- On this data, they all work about as well as each other
 - This data is small and simple though – most data is not
- Which is best is determined by
 - Data: how much do you have, is it linearly separable? Does it fit in memory?
 - Classes: how many? And how bad is the class imbalance?
 - Dimensions: how many? Are they all useful? Are samples sparse (contain a lot of 0's)?

CAB420: Multi-Class Classification

BECAUSE MOST TASKS AREN'T BINARY

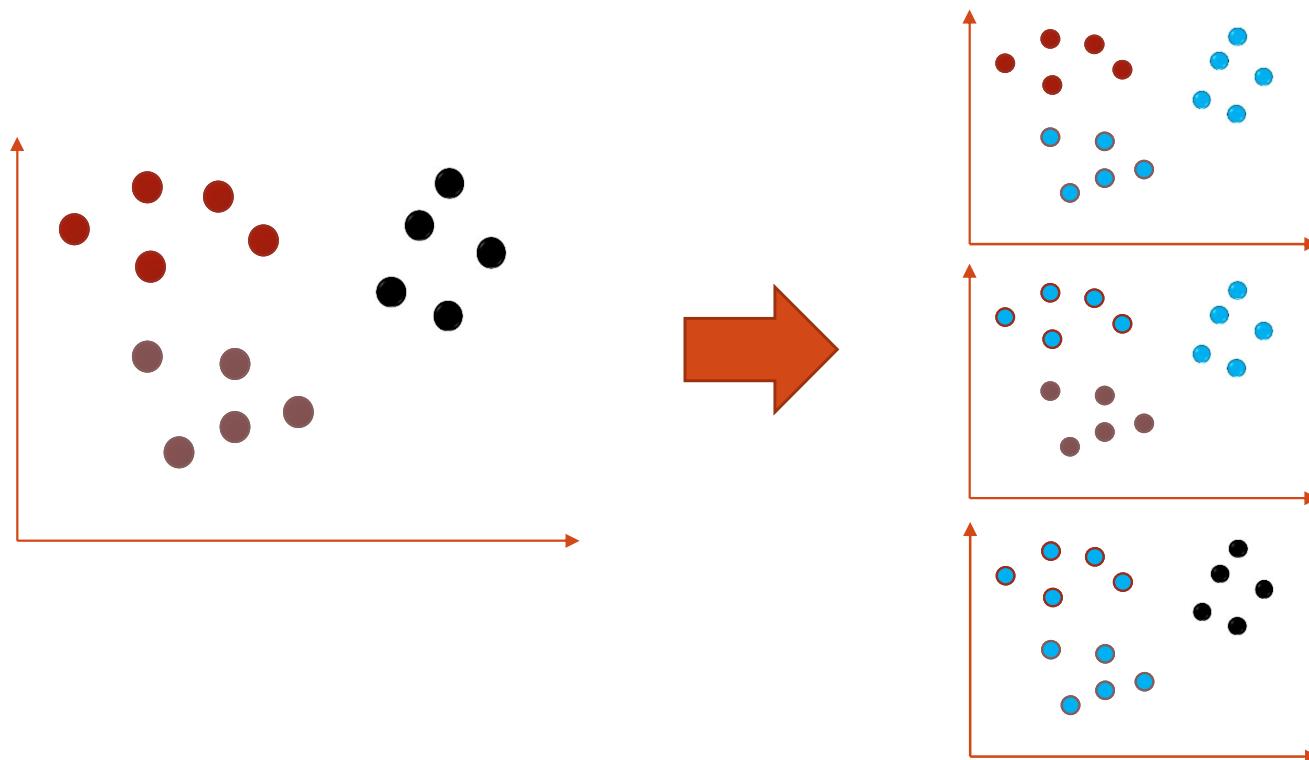
Multi-class Classification

- Multi-Class C-KNN
 - We've already covered it
 - By having more than two labels in the training data, we have a multi-class classifier
 - Need to be mindful of K and class imbalance
- Multi-Class Random Forest
 - We've already covered it too
 - Scales directly to multiple classes
 - Need to consider tree depth and class imbalance

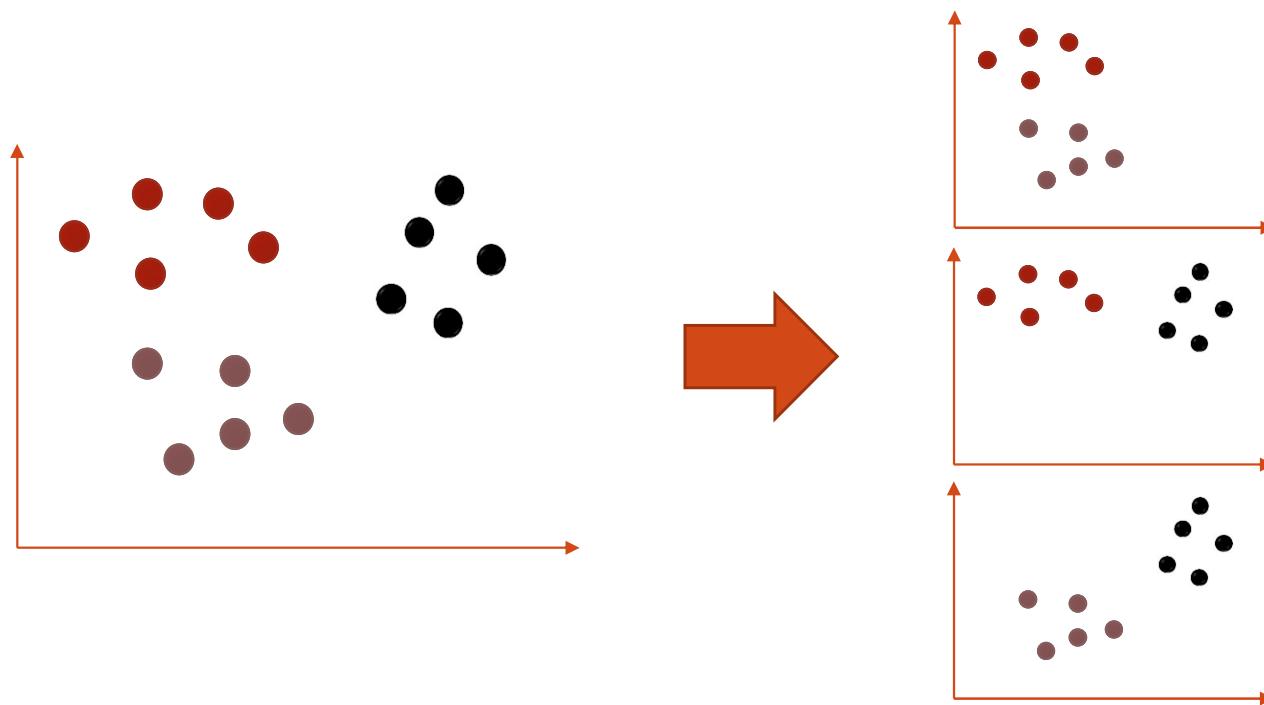
Multi-class Classification

- Consider a dataset with three classes
- Problem
 - How can we use a binary classifier (i.e. SVM, logistic regression) to separate an input into one of three classes?
- Solution
 - Use lots of Binary Classifiers!
 - We'll do this using SVMs, but you can use other binary classifiers just as easily

One vs All



One vs One



Limitations

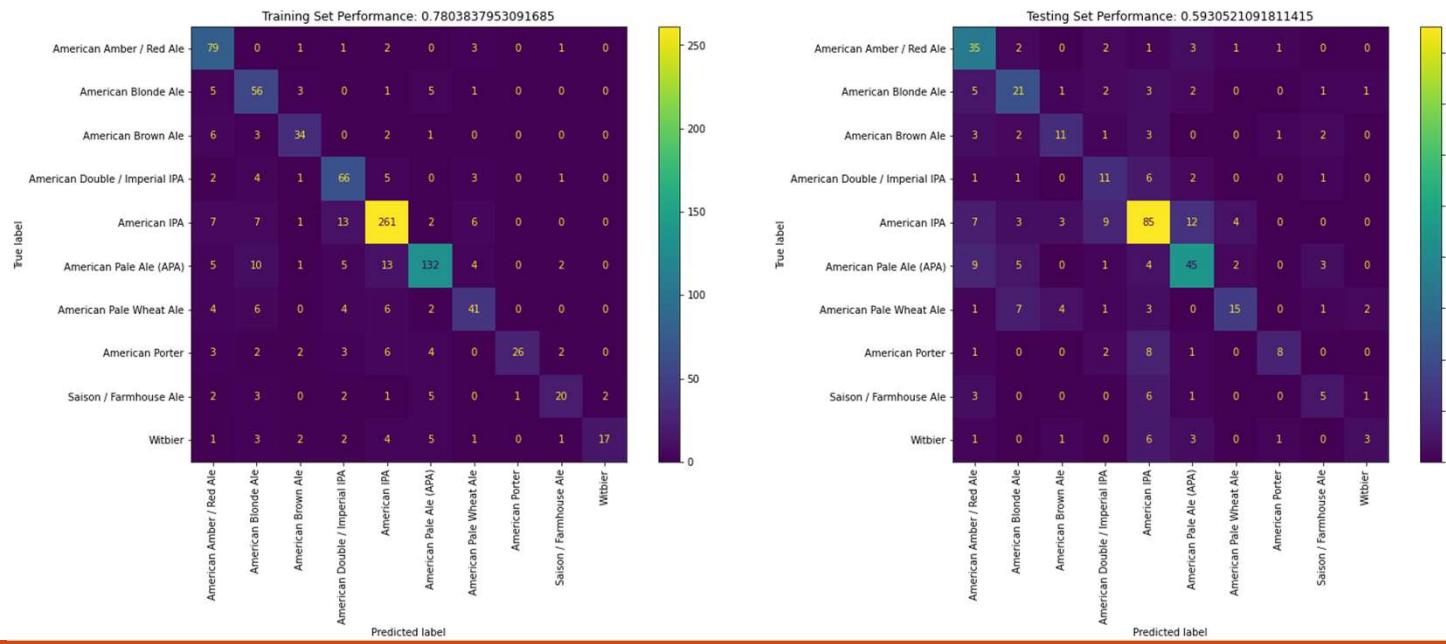
- One vs All
 - More likely to encounter class imbalance issues
- One vs One
 - As number of classes increases, number of classifiers gets big
- Neither one is efficient for very large numbers of classes
 - But One vs All will be much better. Consider a 10-class problem:
 - One vs All : 10 classifiers
 - One vs One: 45 classifiers ($9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$)

Multi-Class Classification Example

- See ***CAB420_Classification_Example_2_Multi_Class_Classification.ipynb***
- Data:
 - Table of American Craft Beers
 - The Task:
 - Classify the type of beer given the beer name
 - We'll tokenise the text and obtain a numeric representation to use in classification
 - More details in the scripts and the interaction session
 - Data is very high dimensional (300 dimensions, and that's after keeping it small)
 - We'll take just the 10 most common classes
 - Avoid very small classes, and very high class numbers

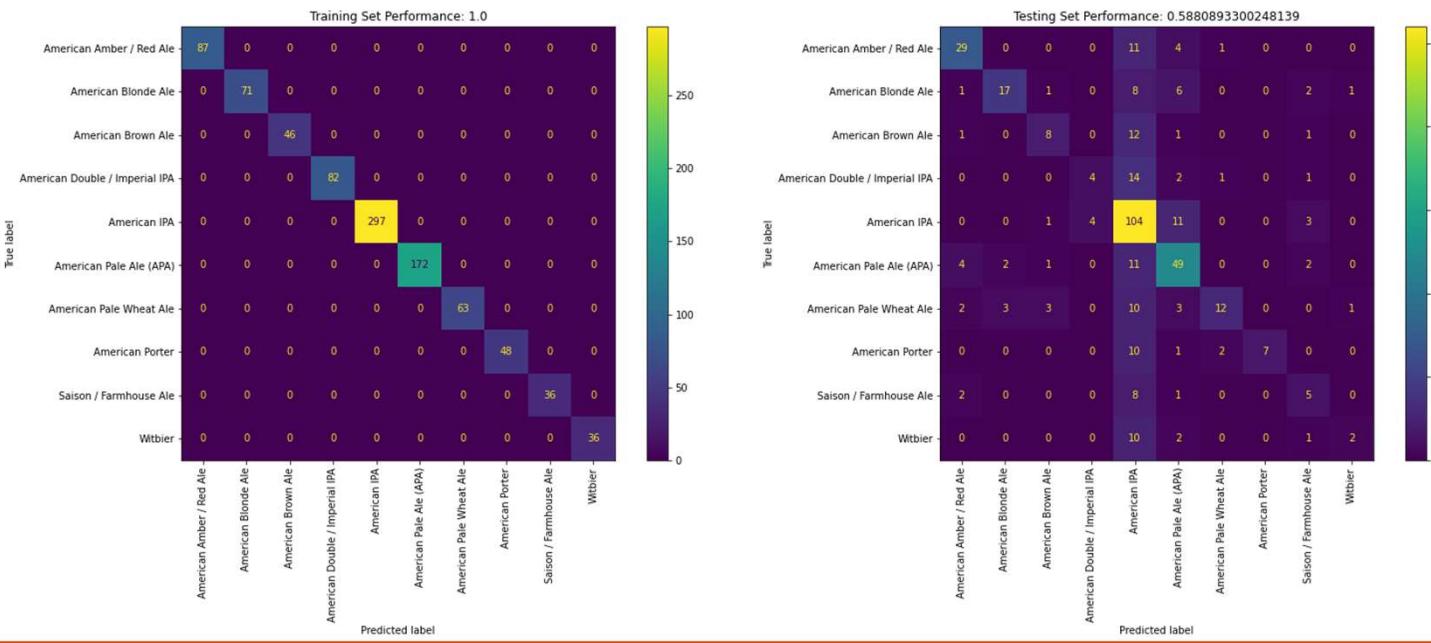
CKNN

- $K = 3$, weights = 'uniform'
 - Fair performance, some struggles on the sparse data



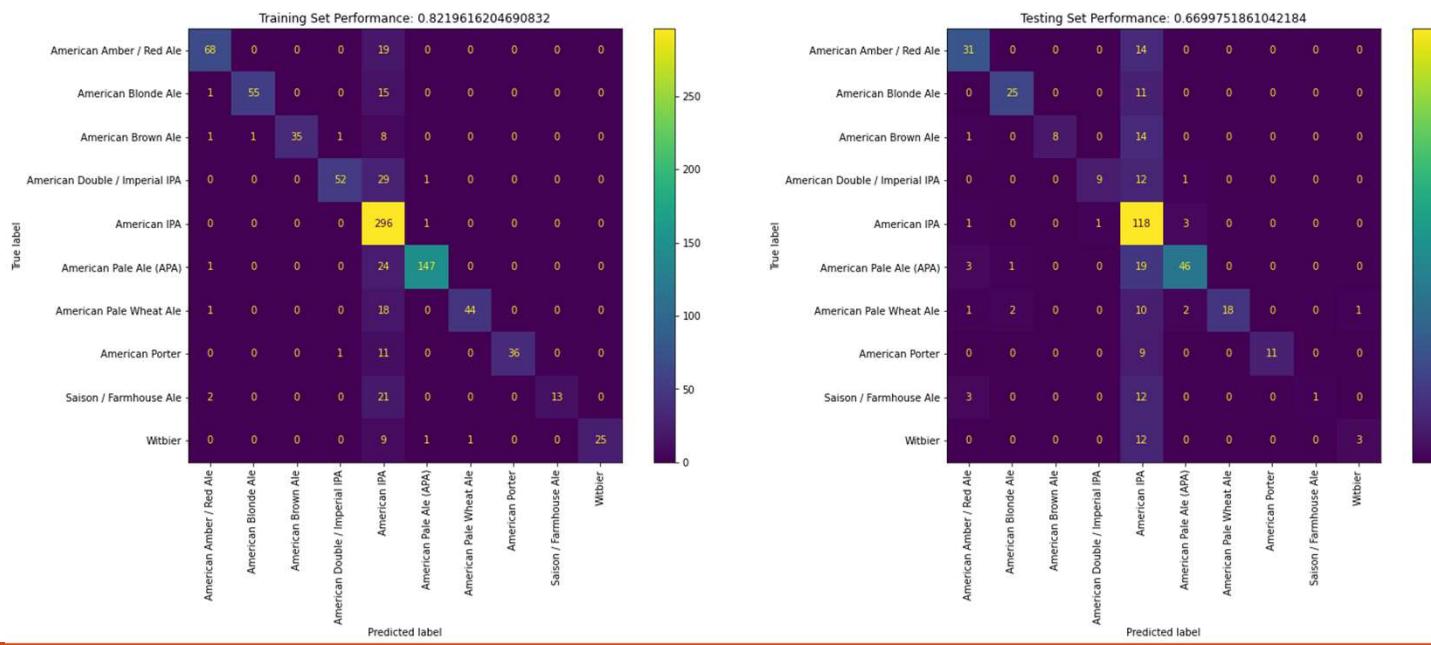
CKNN

- K = 20, weights = 'distance'
- Training results perfect due to 'distance' weights
- Testing results about the same



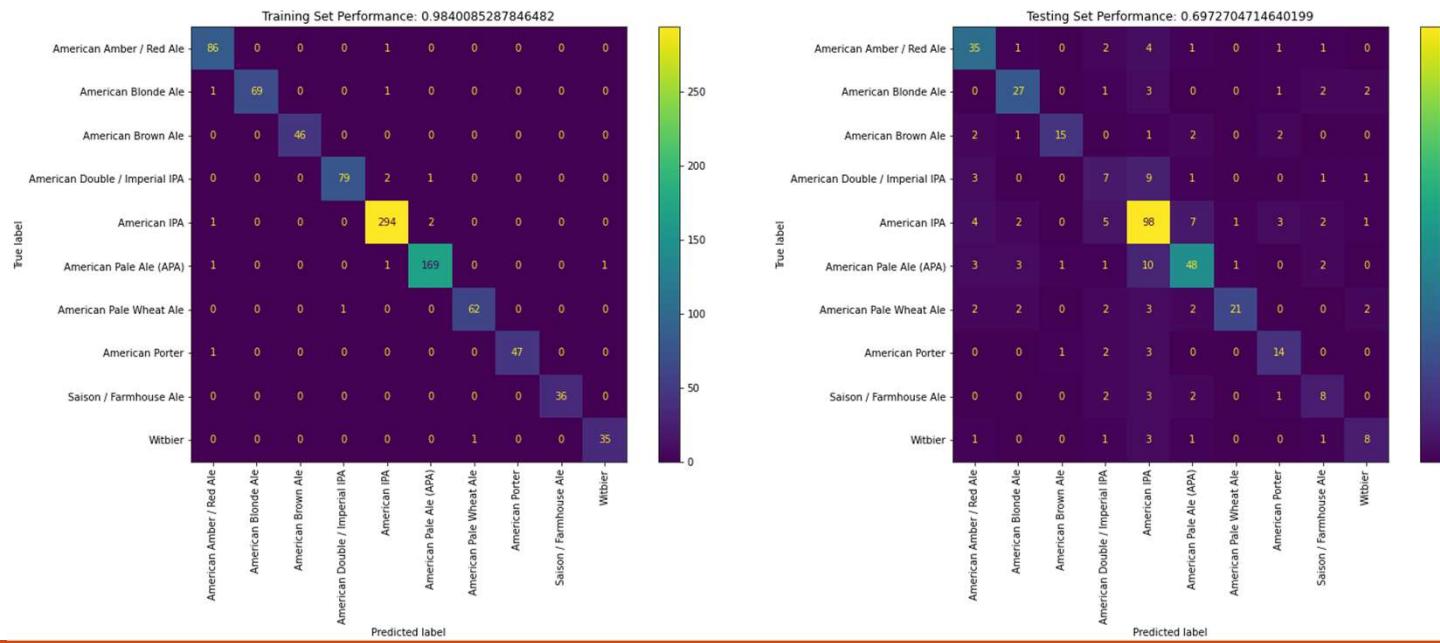
SVM – One vs One

- Linear Kernel, C = 1
 - Outperforms our CKNN
 - Struggles with small classes



SVM – One vs All

- Linear Kernel, C = 1
 - Oddly, outperforms the One vs One including on small classes
 - This is atypical



Class Imbalance

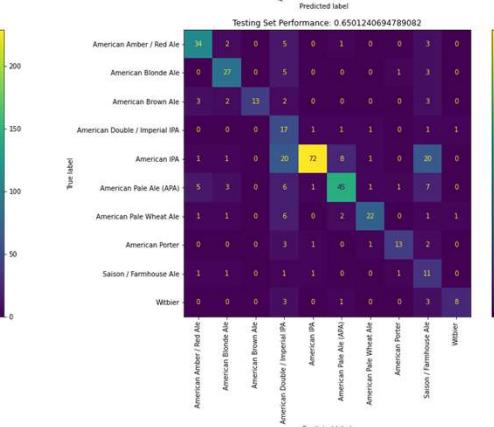
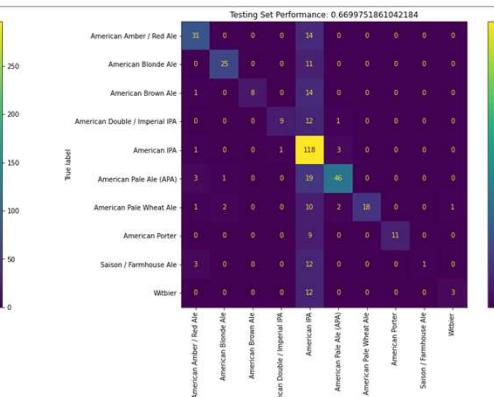
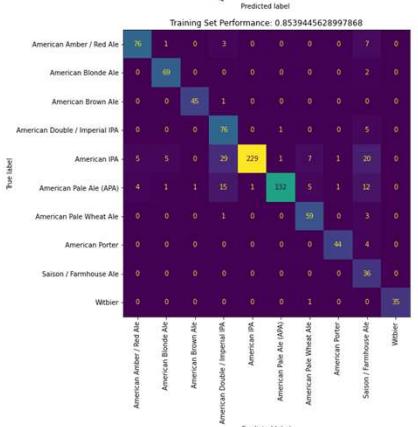
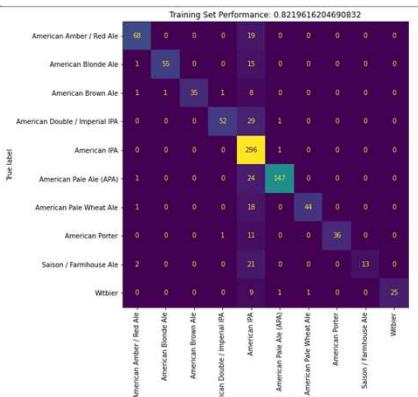
- By default, our models care about all points and classes equally
- This generally makes sense, but consider a situation where:
 - You have 1000 samples
 - 990 are from class 1
 - 10 are from class 2
- In this situation, the model will be 99% accurate if it totally ignored class 2
- Models and optimisation routines will often favour majority classes as being accurate on the majority class helps maximise overall accuracy

Class Weights

- Class weights can be used to counteract class imbalance
- Apply a weight to errors from different classes
 - Weight is inversely proportional to the amount of data available for the class
 - Errors on rare classes "cost" more
- Need to mindful of classifier setup when constructing class weights
 - One vs One and One vs All have different weight matrices due to different numbers of samples when training the model

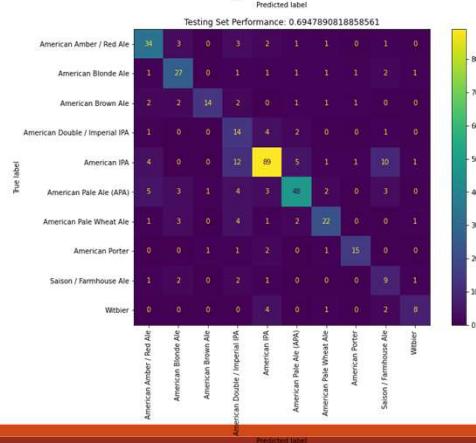
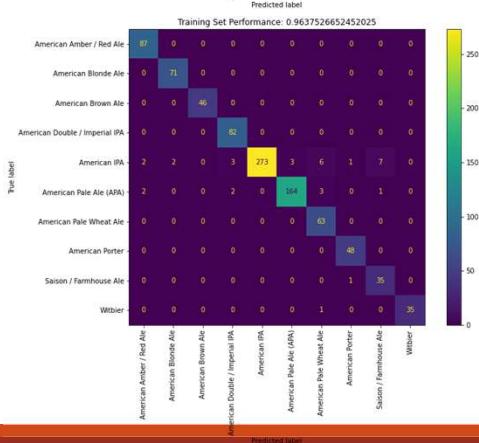
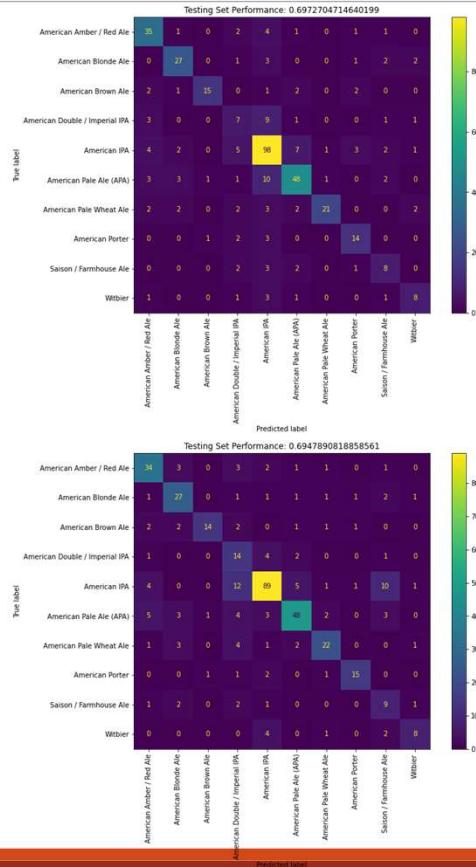
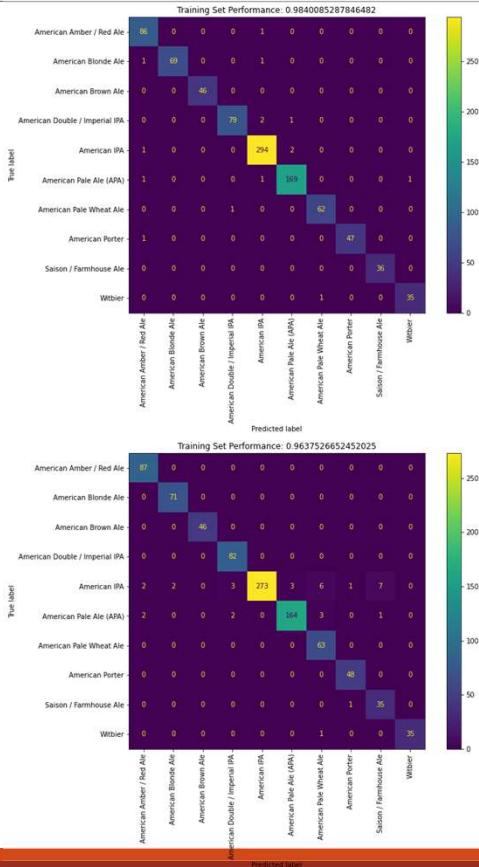
SVM – One vs One with Class Weights

- With class weights (below) we have
 - Slightly lower accuracy overall
 - Better performance on rare classes
 - Worse performance on common classes
 - This causes the overall accuracy drop



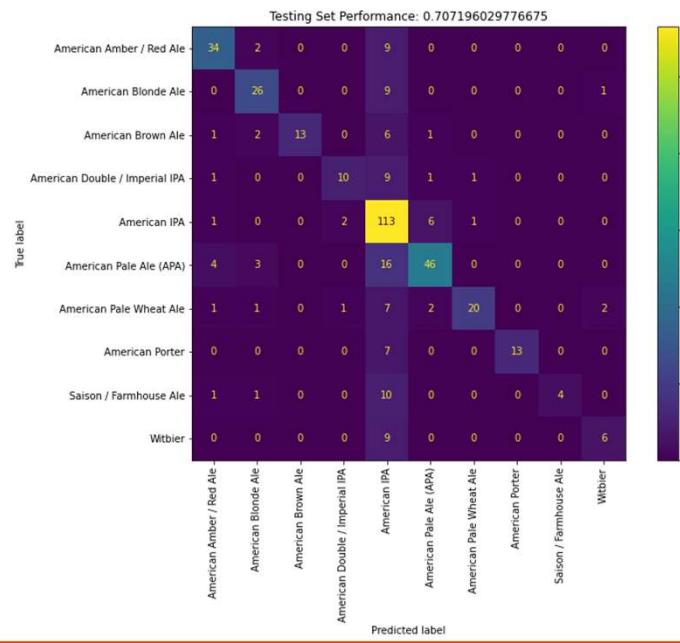
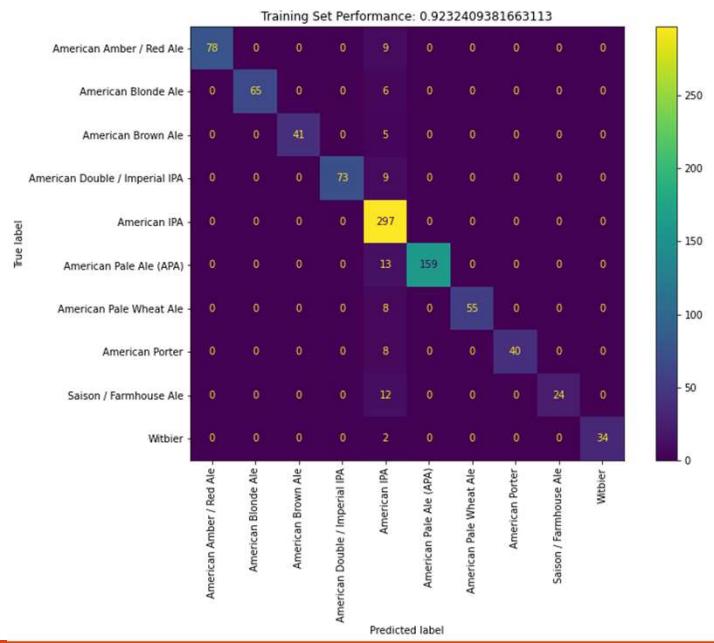
SVM – One vs All with Class Weights

- With classes weights (below) we have
 - Almost the same overall accuracy
 - Better performance on rare classes
 - Slight drop on common classes



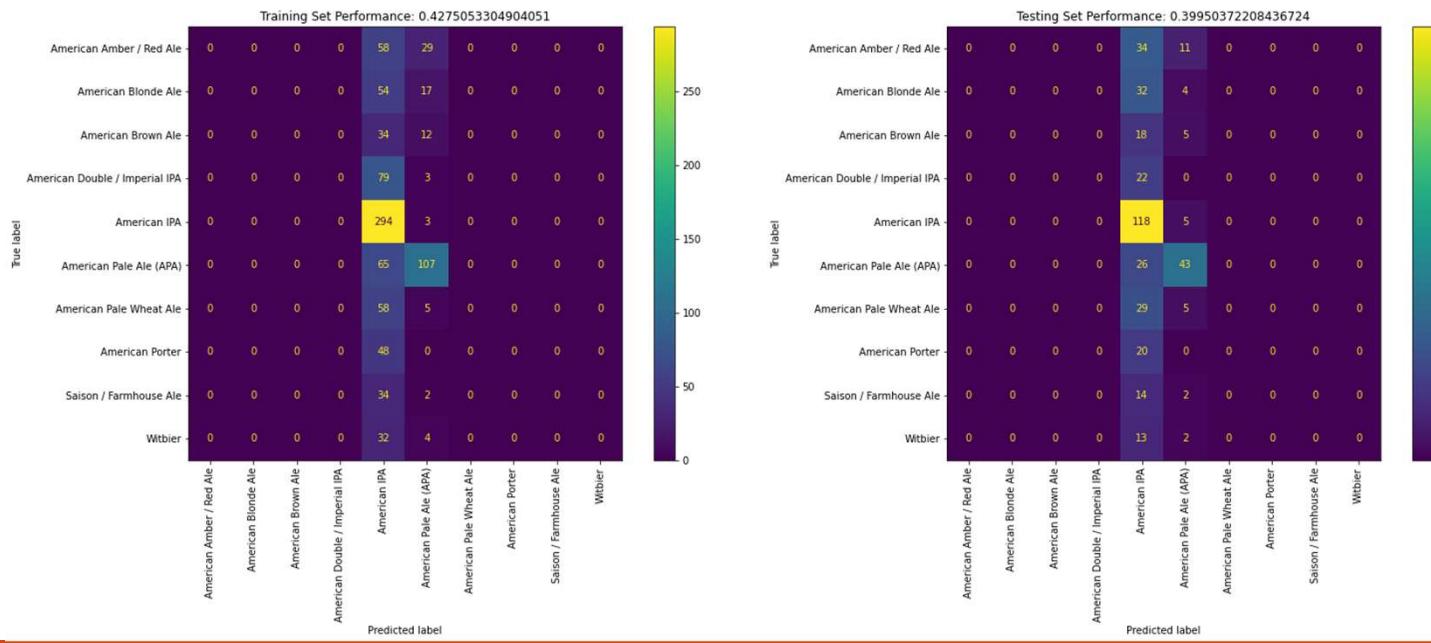
Random Forest

- 250 trees, max depth of 15
 - Best performance so far
 - Some struggles with rare classes



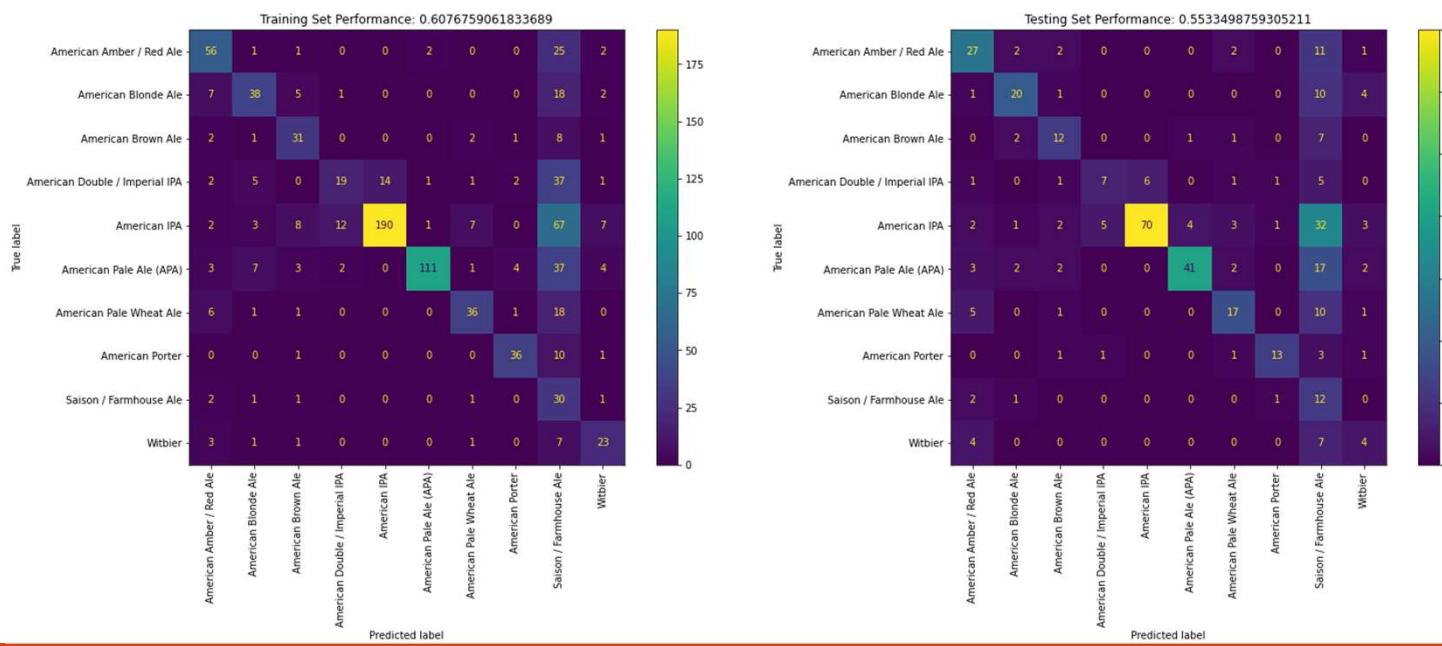
Breaking the Random Forest

- 250 trees, max tree depth of 2
 - Everything classified into majority classes
 - Not enough depth to separate data into all classes



Random Forests and Class Weights

- 250 trees, max tree depth of 2
 - Middling overall performance (but very shallow trees)
 - Forced the model to consider all classes



Random Forests and Class Weights

- Each tree sees a random fold of the data
- Class weights can be:
 - Determined across the whole dataset
 - Each tree has the same weights
 - Determined for each bag
 - Each tree has weights specific to the dataset it has
 - Either is valid, which is more appropriate depends on the data and the level of imbalance

Are these classifiers optimal?

- No
- We've done minimal hyper-parameter tuning, this could include
 - CKNN
 - Number of neighbours
 - Distance metric
 - SVM
 - Kernel and C
 - Encoding (one vs one or one vs all)
 - Random Forest
 - Tree parameters
 - Forest size
- In your own time
 - Apply grid search (or other hyper parameter search) to improve performance

CAB420: Classification Summary

BECAUSE IT'S EASY TO SKIP OVER THINGS

Support Vector Machines

- Learn a decision boundary between two **linearly separable** classes
 - But, if you need machine learning, it's probably not linearly separable
- So we can
 - Relax our constraints via a slack variable
 - Map to a higher dimensional space where things might be linearly separable via kernels
 - Care must be taken with kernel selection and parameters
 - Or do both
- In general
 - Good with high-dimensional and/or sparse data
 - Can be difficult to train with large (10,000 +) datasets
 - Binary in nature (though model ensembles can enable use on multi-class problems)

C-KNN

- K-Nearest Neighbours
 - Classification based on finding similar points
 - Sensitive to dataset size, and the number of neighbours
 - Compared to SVMs
 - Can more easily capture non-linear relationships via use of neighbours
 - Can be sensitive to outliers via use of neighbours
 - Extends to multi-class classification trivially
 - No actual learning
 - Decision boundary is based directly off provided input points

Random Forests

- Ensemble of decision trees
 - Relies on classifiers being uncorrelated
 - Each classifier uses a random selection of training points
 - Each branch in each tree uses a random set of features
 - Inherently multi-class
 - Typically, very efficient
- Depth of tree leads to more accurate decisions on training data
 - Can lead to overfitting
- Class weights can be included in the same manner as an SVM
- Can obtain a likelihood from results across all trees
 - i.e. 50% of trees said class 1, 25% said class 2, etc

Hyper Parameters

- Changing hyper parameters can lead to a big change in performance
- If in doubt
 - Start with default values (they're defaults for a reason)
 - Run small scale tests to explore performance
 - Select 20% of the data, train some models
 - Use hyper parameter selection
 - Explore further resources on relevant hyper parameters

Performance Metrics

- Confusion matrix
 - Shows number and type of correct and incorrect decisions made
- Accuracy
 - Simple, single, overall measure
 - How often the model is correct
 - Can be misleading with data imbalance
- Precision, Recall and F1 score
 - Provide information on the type of errors occurring
 - False negatives and false positives
 - F1 provides an overall measure

Data Considerations and Model Selection

- The nature of the data should be considered when deciding what model to use
 - Multi-class or single class?
 - SVMs don't scale well to large numbers of classes
 - Class distribution? Even or imbalanced?
 - Some parameters are sensitive to small classes. Some models can incorporate class weights
 - How much data do you have? Does this make some models hard to train due to memory constraints?
 - This can be a problem for SVMs with 10,000+ points
 - How many dimensions? Are all meaningful? Does the data need to be standardised?
 - SVMs can benefit from high dimensional data sets
 - CKNNs can be impacted by data with large range variations between dimensions

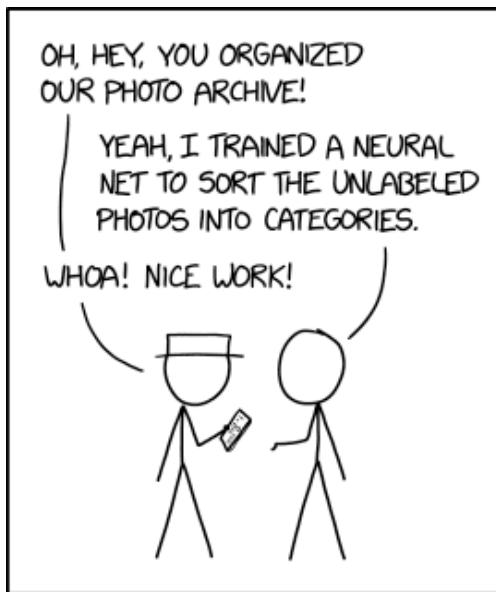
CAB420: Neural Networks and their Components

A BIT LIKE LEGO

Neural Networks

- Neural networks are a collection of layers
- In a simple network, layers connect to each other sequentially
 - Data flows from one layer to the next
- Overall structure inspired by the human brain
 - Activations cascading through the brain is mimicked by data propagating through the neural network

Cartoon from XKCD



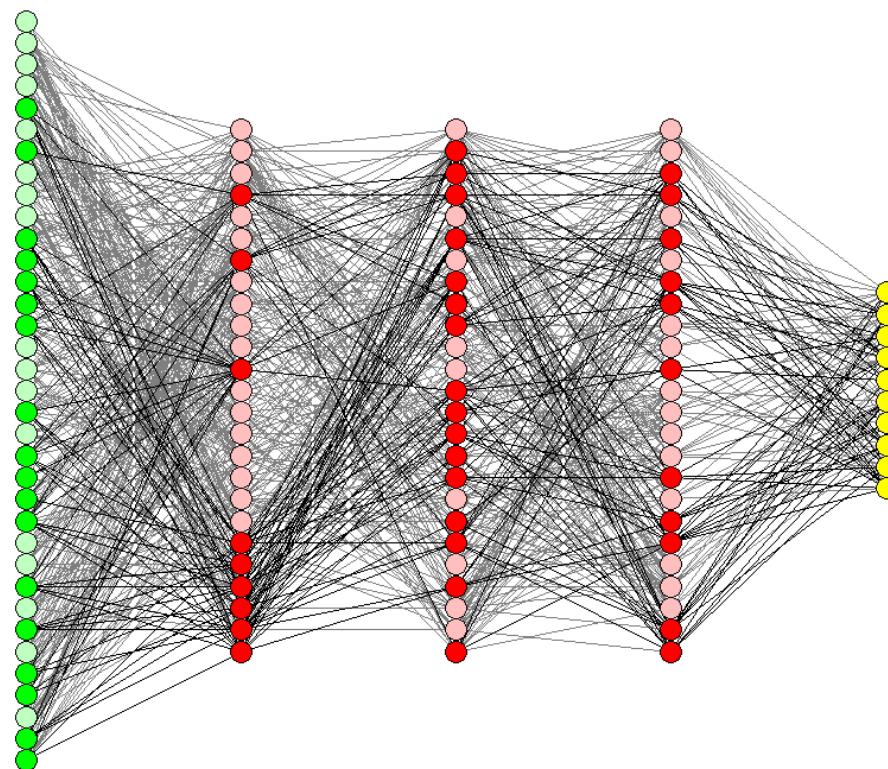
ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND,
YOU CAN TECHNICALLY SAY YOU
TRAINED A NEURAL NET TO DO IT.

Neural Network Components

- Networks are built from a collection of layers, separated by non-linearities
 - Lots of possible layers
- We'll start with
 - Fully Connected Layer
 - Convolutional Layer
 - Pooling
 - Activation

Fully Connected Layers

- Every neuron in one layer is connected to every neuron in the next
 - Doesn't really capture spatial relationships in the data
 - i.e. Spatially adjacent neurons do not necessarily give similar results
 - Essentially a matrix multiplication



Fully Connected Layers

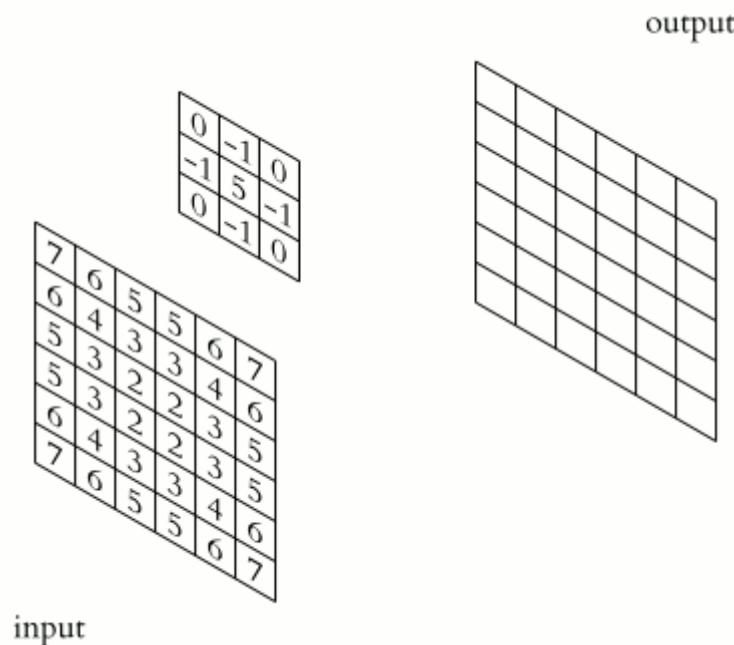
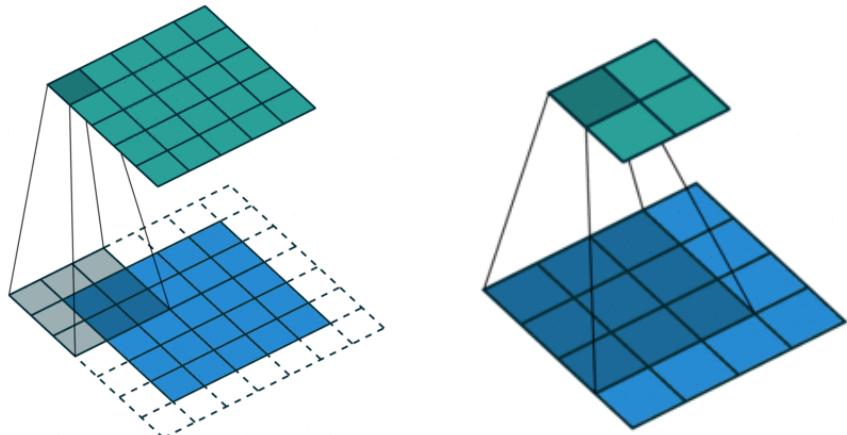
- Input: a vector of size $[1 \times M]$
- Output: a vector of length $[1 \times N]$
- Parameters:
 - Weight matrix, $[M \times N]$ in size
 - Bias vector, $[1 \times N]$ in size
- Computation:
 - $\text{Output} = \text{input} * W + B$
 - For large M and/or N , W becomes very large

Convolutional Layer

- Convolution

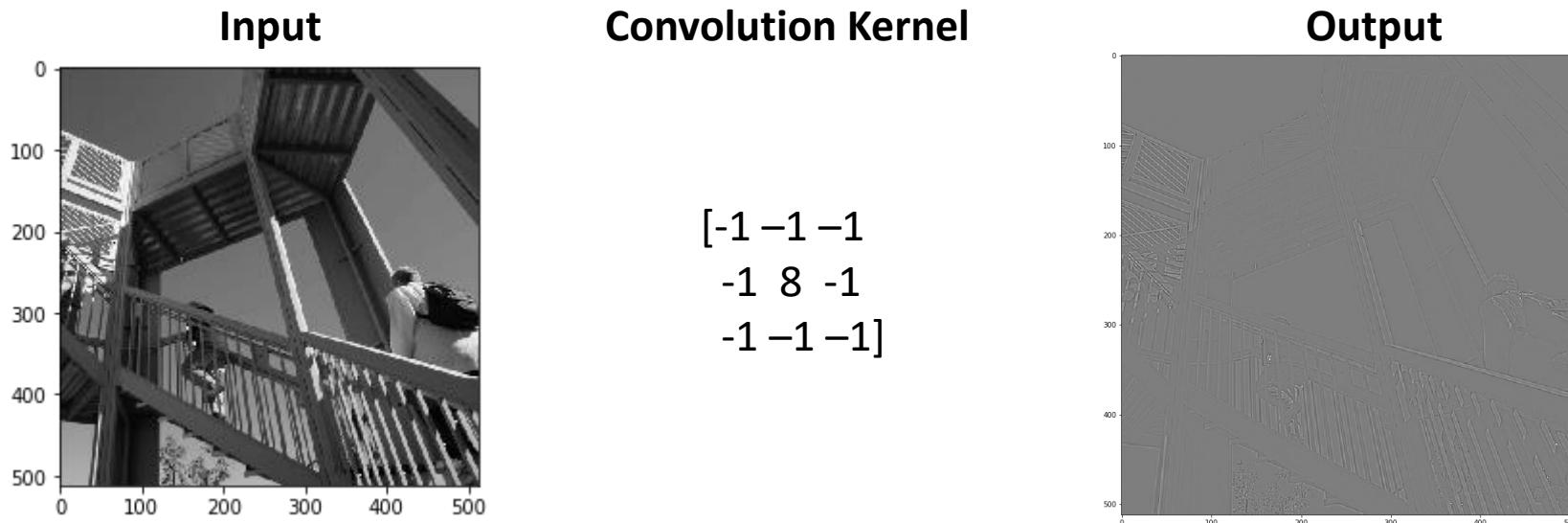
$$(f * g)(t) \triangleq \int_{-\infty}^{+\infty} f(\tau)g(\tau - r)dr$$

- The integral of the product of the two functions after one is reversed and shifted
- Performs a weighted sum of one input according to the second
- Typically viewed as a filtering process



Convolution

- With images, can be thought of as applying a filter



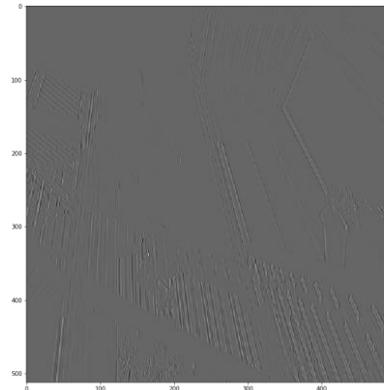
- Example kernel is an edge detection kernel
 - Detects all edges by finding pixels where there is a local change in contrast

Convolution

- Different Kernels will give different outputs

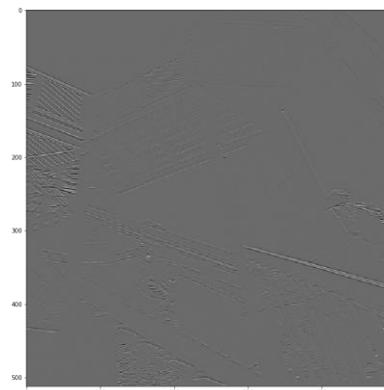
- Vertical Edges

- $$\begin{bmatrix} -4 & 8 & -4 \\ -4 & 8 & -4 \\ -4 & 8 & -4 \end{bmatrix}$$



- Horizontal Edges

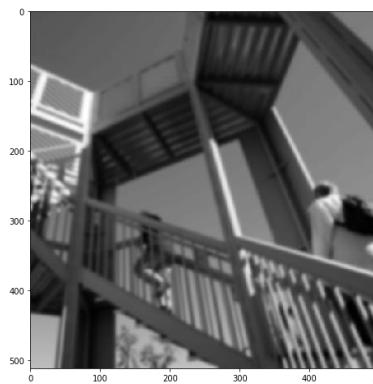
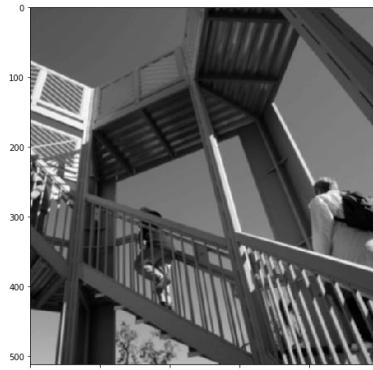
- $$\begin{bmatrix} -4 & -4 & -4 \\ 8 & 8 & 8 \\ -4 & -4 & -4 \end{bmatrix}$$



Convolution

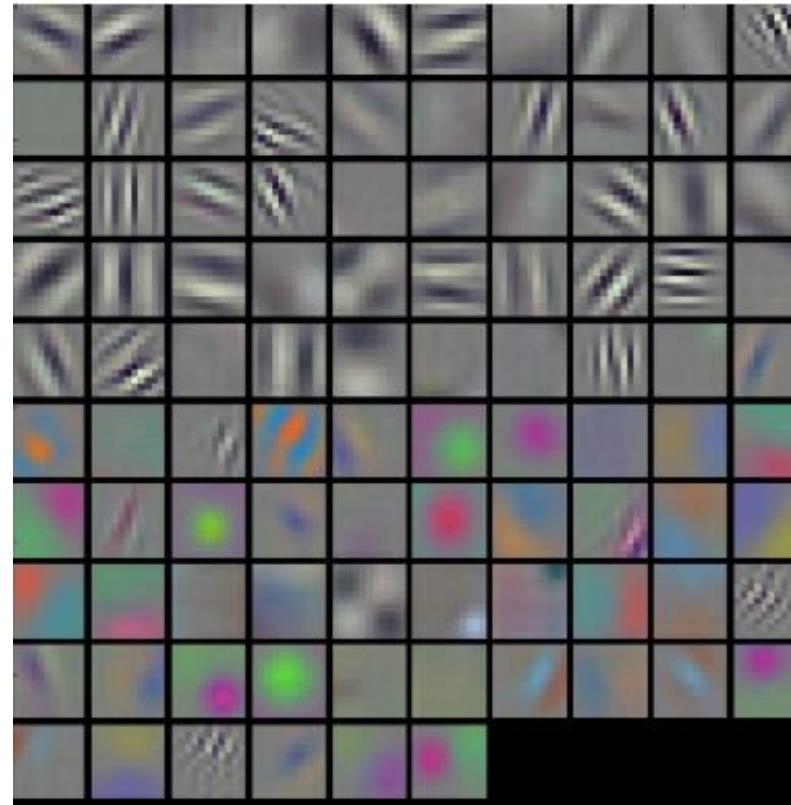
- The output of one convolution operation can be used as the input to another
 - Stacking filters
 - Blurring kernel:
 - [0.111 0.111 0.111
0.111 0.111 0.111
0.111 0.111 0.111]
 - Top image: after 1 blur
 - Bottom image: after 7 blurs

Example output taken
from *CAB420_DCNNs_Additional_Example_2_Convolutions.ipynb*



Convolutional Layer

- In neural networks
 - We learn the filters
 - Typically learn lots of filters at once
- Learned filters can
 - Represent simple shapes, edges, textures in early layers
 - Represent more complex structures in later layers
- Filters operate over all channels in the input
 - If we have a colour input, we have a colour filter

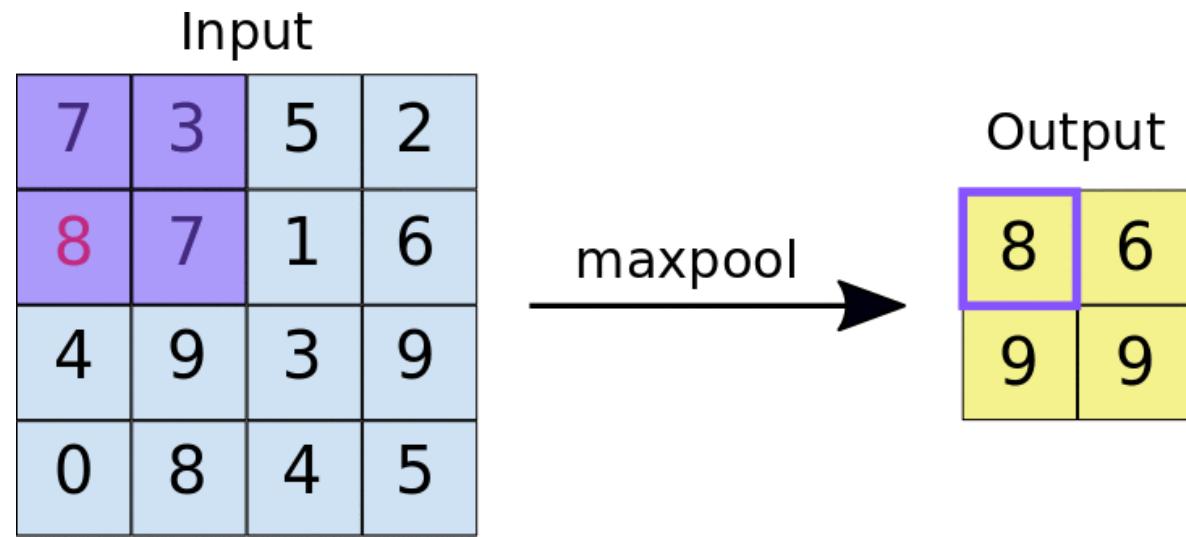


Convolution Layer

- Input: $[W \times H \times C]$ image
- Output: $[W \times H \times N]$ image
 - N is number of learned filters
- Parameters:
 - Each filter is $[X \times Y]$ in size, and has $[X \times Y \times C]$ weights
 - 1 bias value per filter
- Computation:
 - Each filter applied at each location in target image, bias is added per filter
 - Operates over all image channels at once
 - Each filter results in one output channel in the output
- Other configuration options:
 - Stride: do we apply the filter at every pixel, or skip some?
 - Behavior at borders: do we pad such that all pixels can be used?

Pooling

- Used to aggregate features
 - Reduces dimensionality
- Multiple types of pooling
 - Min, Max, Average
 - We almost always use Max Pooling
 - Take the maximum value in a region as output
 - Typically placed after a convolution layer



Pooling

- Input: $[W \times H \times C]$ image
- Output: $[W' \times H' \times C]$ image
 - Output is reduced spatially, but number of channels is unchanged
- Parameters:
 - No learned parameters
 - Size and type of pooling operation is fixed when network is created

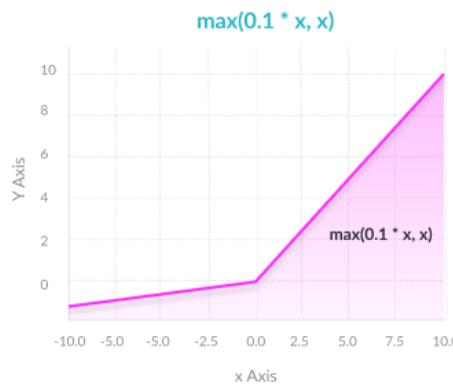
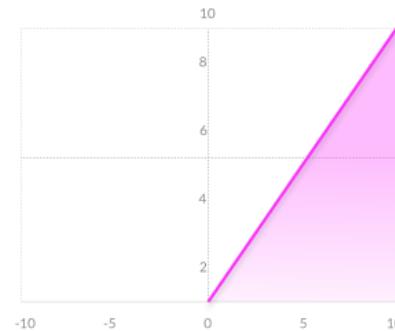
Activations

- Neural networks connect outputs of one layer to the inputs of the next
- However, we don't just feed them straight in, we pass them through an “activation function”
 - Can be seen to turn some neurons on, and others off
 - Introduces non-linearities, helpful for learning complex functions
 - Different activations let different amounts of information flow
 - Can have impacts on learning

Common Activations

ReLU

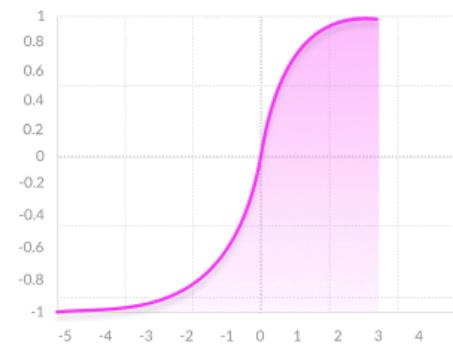
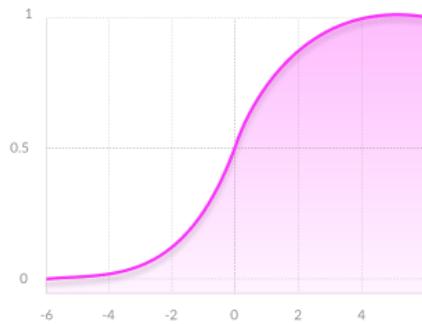
- Rectified Linear Unit
 - Linear for values greater than 0
 - 0 for negative values
- Leaky ReLU
 - Like ReLU, but doesn't totally attenuate the values less than 0
 - Can help learning by allowing gradients to propagate with negative values



Common Activations

- Sigmoid
 - Maps input to $[0, +1]$
 - Acts to normalise the outputs (within fixed bounds)
 - Effectively learns a classifier

- TanH
 - Maps input $[-1, +1]$
 - Otherwise like the Sigmoid



Common Activations

- SoftMax Activation
 - Normalise the output such that it sums to one
 - Typically used as the output of a classification network
 - Highlight the highest response, suppress all others
- There are lots of other activations
 - Exponential Linear Unit (ELU)
 - Clipped ReLu
 - SoftPlus
 - Swish

Network Layers and Computational Efficiency

- Neural Networks have a (well deserved) reputation for being computationally demanding. But, operations can be implemented very efficiently.
- Consider convolution:
 - Each pixel in the output can be computed independently
 - Massive potential for parallelisation
 - Hence, rapid performance gains possible via GPUs
 - Huge numbers of very simple processing cores
- Many other operations can be similarly broken down
 - Fully connected layers are a matrix multiplication. Each output element can be computed independently of all others

CAB420: Building A Network for Classification

A SMALL ONE TO START WITH

A Network

- A collection of layers
 - Computation layers
 - Fully connected, convolution
 - Essentially can be expressed as $y=wx+b$, where all variables are matrices
 - Activation layers
 - Non-linearities between computations, regulate the flow of data
 - Pooling layers
 - Reduce dimensionality, combine activations
- Output of one layer is input to the next
 - Data propagates through the network

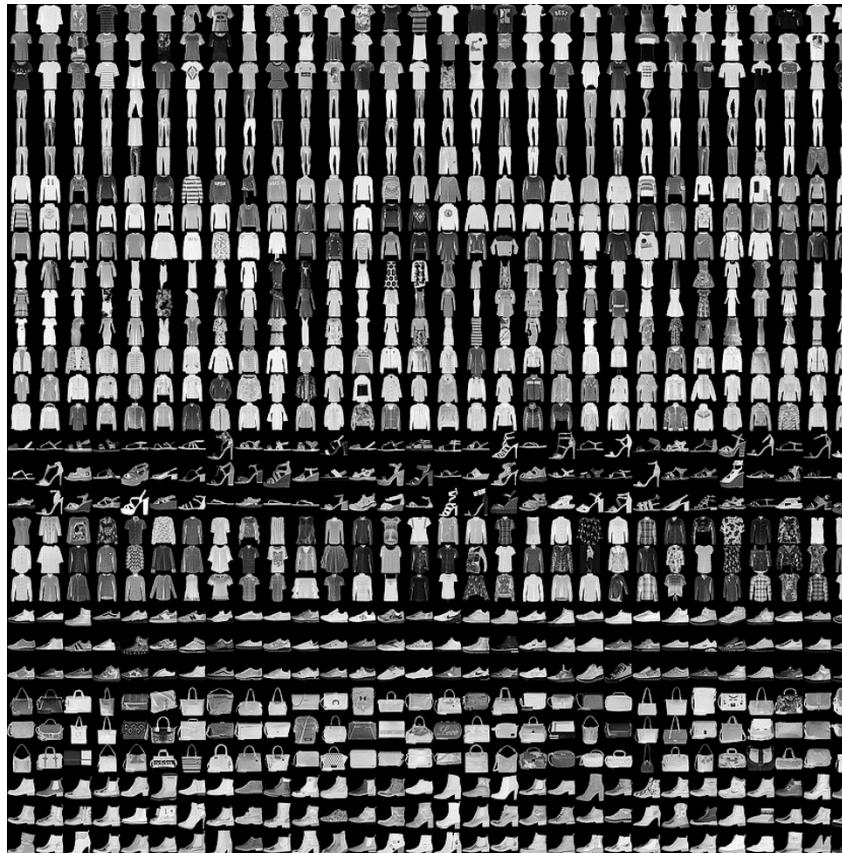
Network Structure

- Networks can have
 - Multiple branches
 - Multiple inputs and/or outputs
 - Skip connections
 - i.e. some layers are skipped and features are concatenated elsewhere
- We'll stick to simple networks (for now)
 - One input
 - One output
 - Feed-forward structure

A Classification Problem

- Fashion MNIST
 - 60,000 28x28 pixel greyscale images of clothing
 - 10 types of clothes
- The task
 - Classify images into the type of clothes they show

We're using images here. If you're uncertain about images as a data type please have a look at
CAB420_DCNs_Additional_Example_1_Images_Introduction.ipynb



An Approach

- Start simple
 - A couple of fully connected layers
 - This won't work that well
- Then add complexity
 - Convolutions!
- See ***CAB420_DCNNs_Example_1_Classification_with_Deep_Learning.ipynb***

Network Output

- We have a classification task, so our network needs to tell us which class something is. How?
 - Using a “one-hot” vector
- Consider our 10 class classification task
- We can represent this as a vector of length 10, where one element is 1 and the rest are 0’, i.e.:
 - 0001000000, would be class 4
 - 1000000000, would be class 1

Network Losses

- We need a way for our network to know when it's right or wrong
 - Enter, the loss function
- Loss functions
 - Are 0 when the network get's it right
 - Usually return increasingly large values as a network becomes more wrong
- These provide the errors that are back-propagated to train the network

Binary and Categorical Cross Entropy

- Use to measure the loss for classification tasks

$$CE = - \sum_i^N y'_i \log(y_i)$$

- where
 - y'_i is the true class probability, [0..1]
 - y_i is the predicted probability, [0..1]
 - N is the total number of classes
- Measures mismatch between observed and expected distributions

Cross Entropy Explored

$$CE = - \sum_i^N y'_i \log(y_i)$$

- $y_i = [001], y'_i = [001]$
- $CE = -(0x\log(0) + 0x\log(0) + 1x\log(1)) = \text{inf} + \text{inf} + 0$
 - Problem, $\log(0)$ is undefined
- In practice
 - Our estimates are almost never 0, activation functions see to that

Cross Entropy Explored

$$CE = - \sum_i^N y'_i \log(y_i)$$

- $y_i = [010], y'_i = [001]$
 - Note, we'll treat the 0's in y_i as very small positive numbers
- $CE = -(0 \times \log(0.000001) + 0 \times \log(1) + 1 \times \log(0.000001)) = -(0 + 0 + -6) = 6$
 - We record a high loss as our classifier was totally wrong

Cross Entropy Explored

$$CE = - \sum_i^N y'_i \log(y_i)$$

- $y_i = [0.2 \ 0.4 \ 0.4], y'_i = [001]$
- $CE = -(0 \times \log(0.2) + 0 \times \log(0.4) + 1 \times \log(0.4)) = -(0 + 0 + -0.39) = 0.39$
 - Our loss is not as high as we had some likelihood in the correct result

Binary vs Categorical Cross Entropy

- Categorical Cross Entropy (CCE)
 - You have N exclusive classes
- Binary Cross Entropy (BCE)
 - You have two exclusive classes
 - 2-class case of CCE
- Multi-Class Classification
 - A sample can belong to more than 1 class
 - Use BCE, effectively treat membership of class as a binary classifier

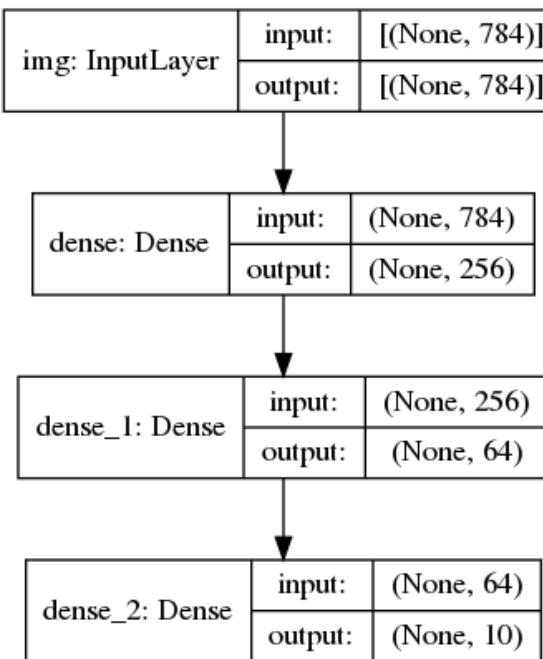
A Simple Network

- Vectorise input
 - [28 x 28] image becomes a [1 x 784] vector
 - Destroys spatial information
 - 3 dense layers
 - Intermediate 1: 256 neurons
 - Intermediate 2: 64 neurons
 - Output: 10 neurons
 - 10 classes

```
Model: "fashion_mnist_model"
-----  
Layer (type)          Output Shape       Param #  
=====  
img  (InputLayer)      [(None, 784)]       0  
dense (Dense)          (None, 256)        200960  
dense_1 (Dense)        (None, 64)         16448  
dense_2 (Dense)        (None, 10)         650  
-----  
Total params: 218,058  
Trainable params: 218,058  
Non-trainable params: 0
```

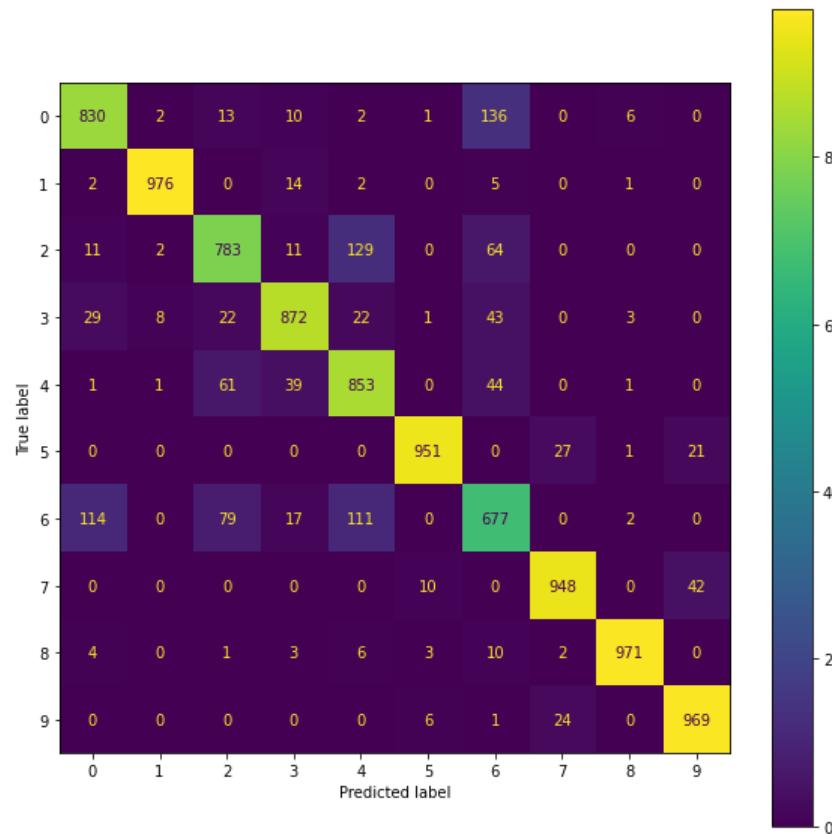
A Simple Network

- 200,000 + parameters
 - And this is a simple network
- Dense layers become smaller as we go deeper
 - Seek to discover most salient information for the task at hand
 - What is salient (important) is determined by the training



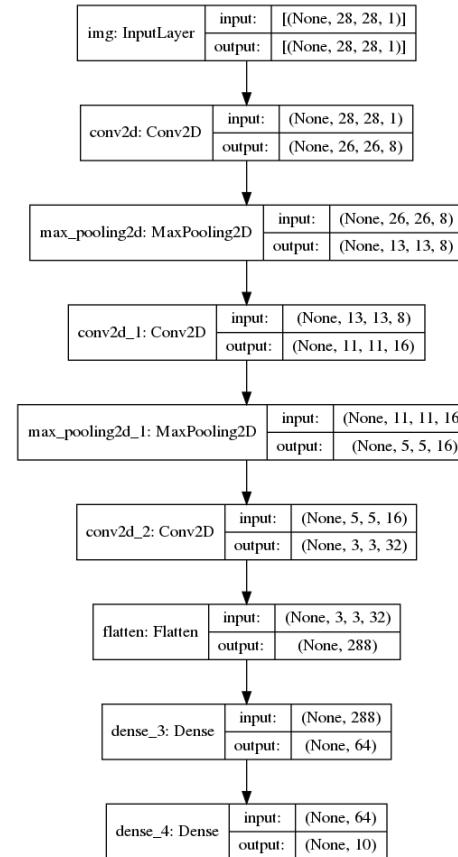
Simple Network Performance

- 88.3% accuracy on test set
- We can improve accuracy by including spatial information



My First CNN

- Image shaped input
 - $28 \times 28 \times 1$
- Three 2D convolution layers
 - Max-pooling after first two
 - Reduce size of representation
 - Keep only the most important features
 - More filters as we go deeper
 - Learn simple filters on the raw image
 - Learn more complex filters over earlier outputs
 - Convolution output width and height are **not** the same as our input width and height
 - Boundary effects, we have no padding, so can't convolve at the image edges
 - Can change this as a layer parameter if we wish
- Two dense layers
 - Final layer for classification
 - Same final output structure as earlier network



My First CNN

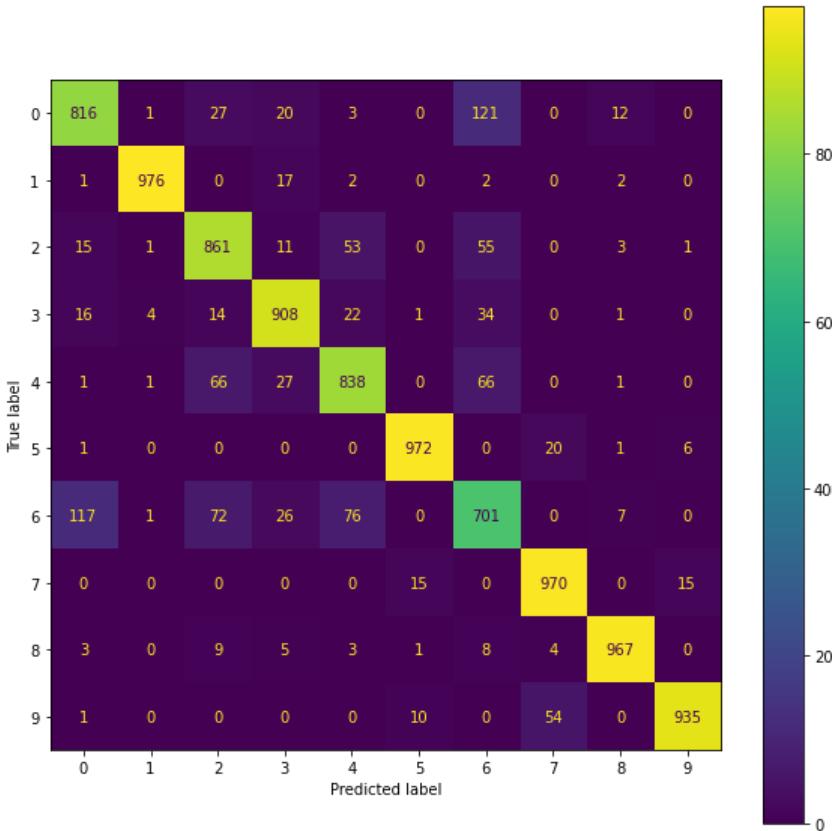
- 25,000 parameters
 - ~1/8th our first network
- Convolution layers are usually more efficient in terms of parameters than fully connected layers
 - 75% of our parameters are in the dense layers
- But convolution layers are more computationally intensive
 - This network will be slower to train

```
Model: "fashion_mnist_cnn_model"

Layer (type)          Output Shape         Param #
=====              ======              =====
img (InputLayer)      [(None, 28, 28, 1)]   0
conv2d (Conv2D)        (None, 26, 26, 8)    80
max_pooling2d (MaxPooling2D) (None, 13, 13, 8) 0
conv2d_1 (Conv2D)       (None, 11, 11, 16)   1168
max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 16) 0
conv2d_2 (Conv2D)       (None, 3, 3, 32)    4640
flatten (Flatten)      (None, 288)           0
dense_3 (Dense)        (None, 64)            18496
dense_4 (Dense)        (None, 10)            650
=====
Total params: 25,034
Trainable params: 25,034
Non-trainable params: 0
```

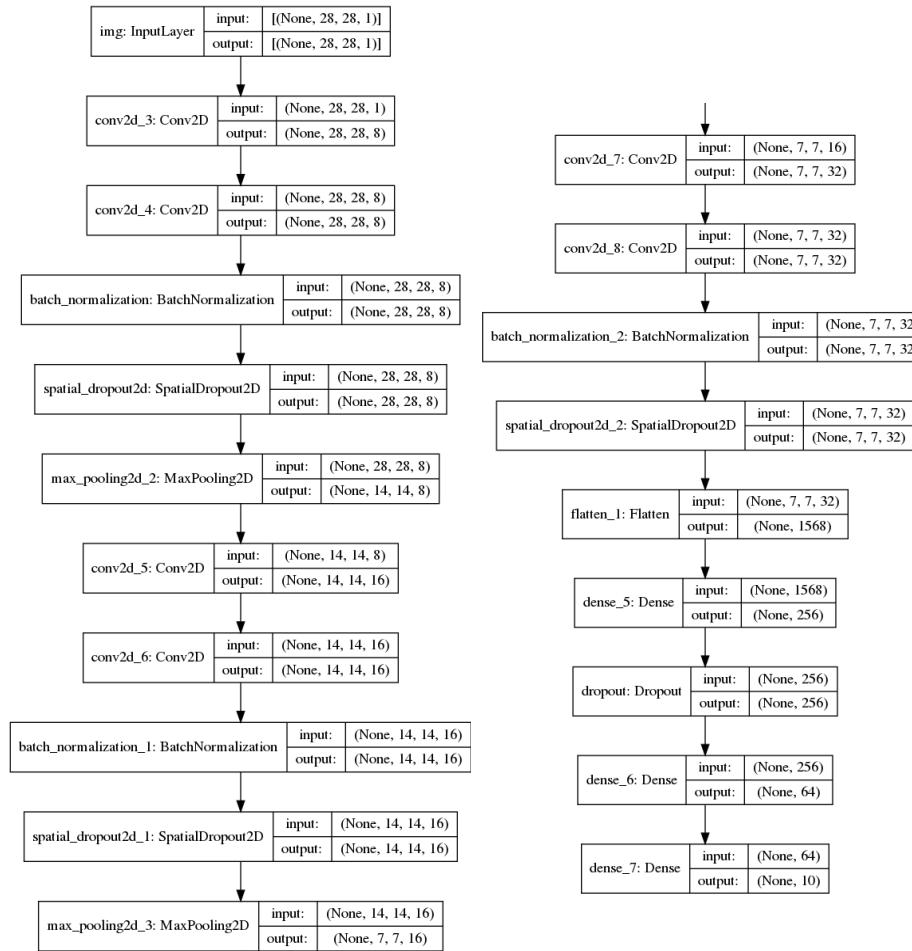
CNN Performance

- 89.4 % accuracy on testing set
 - Small improvement over dense network
 - The dense network was already quite good
 - As we get closer to perfect, it becomes harder and harder to find gains



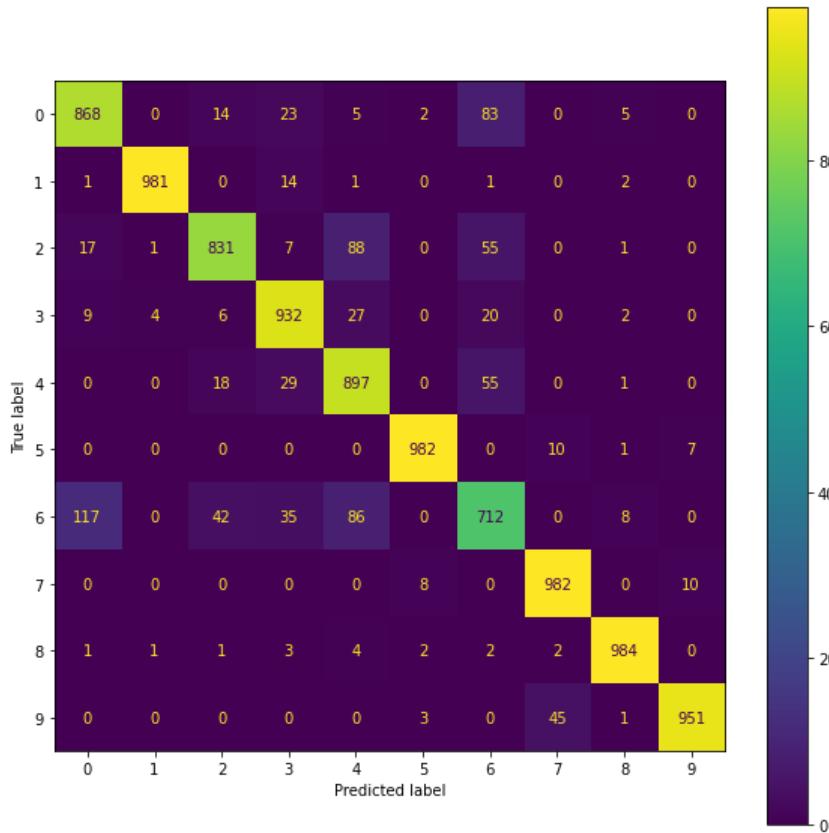
Making it Bigger

- 6 Convolution layers
 - Stacked in pairs
- 3 Dense layers
- 437,026 parameters
- Same input and output as earlier CNN
 - Just more stuff in the middle



Bigger CNN Performance

- 91.2% Accuracy
- Training time has greatly increased
 - 3-4 times our earlier CNN



Network Size and Accuracy and CAB420

- To a point, larger more complex networks will give better performance, however
 - Gains decrease as networks grow
 - Larger network take longer to train, and require more memory
 - Larger networks need more data and are more likely to overfit
 - We can go too deep and break things
- In CAB420, we are not expecting you to train models for hours at a time
- When playing with networks
 - Start small
 - Accept that you will not get state of the art performance
 - Consider using services such as the QUT hosted Jupyter Notebook, or Google Colab to access GPUs if you don't have one

CAB420: Regression with Deep Nets

A LOT LIKE CLASSIFICATION WITH DEEP NETS

Regression with Deep Nets

- As simple as changing our output layer
 - For classification, we have a “softmax” output
 - 0 or 1 (or somewhere in between) to indicate classification certainty
 - For regression, we want a continuous output (usually)
 - ReLu (or similar) activation
 - And a regression target to learn against
 - Can regress to multiple outputs
- Other than that, the networks are pretty similar

Regression Losses

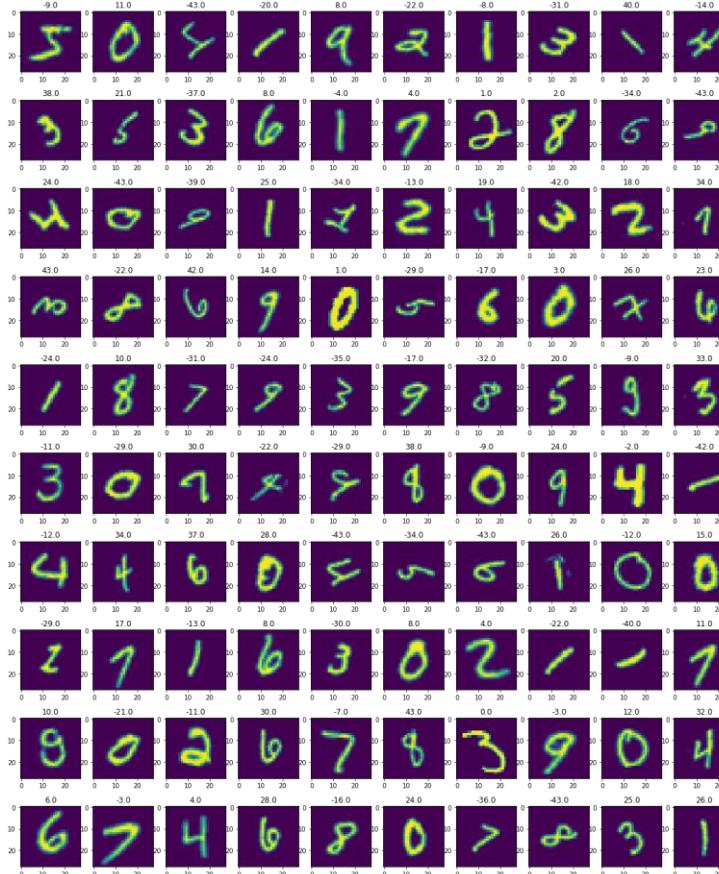
- Usually, we'll use something like Mean Squared Error

$$\text{MSE} = \sum_i^N (y'_i - y_i)^2$$

- Other times we may wish to use Mean Absolute Error, or other distance measures depending on the problem and data
 - You can see a list of existing losses within tensorflow/keras here: <https://keras.io/api/losses/>

Regression Example

- See ***CAB420_DCNNs_Example_2_Regression_with_Deep_Learning.ipynb***
- Data
 - Rotated digits, digits have been randomly rotated by [-45 ... +45] degrees
- Task
 - Estimate the amount of rotation a digit has undergone
 - Single output, regressing from an input to one number



The Network

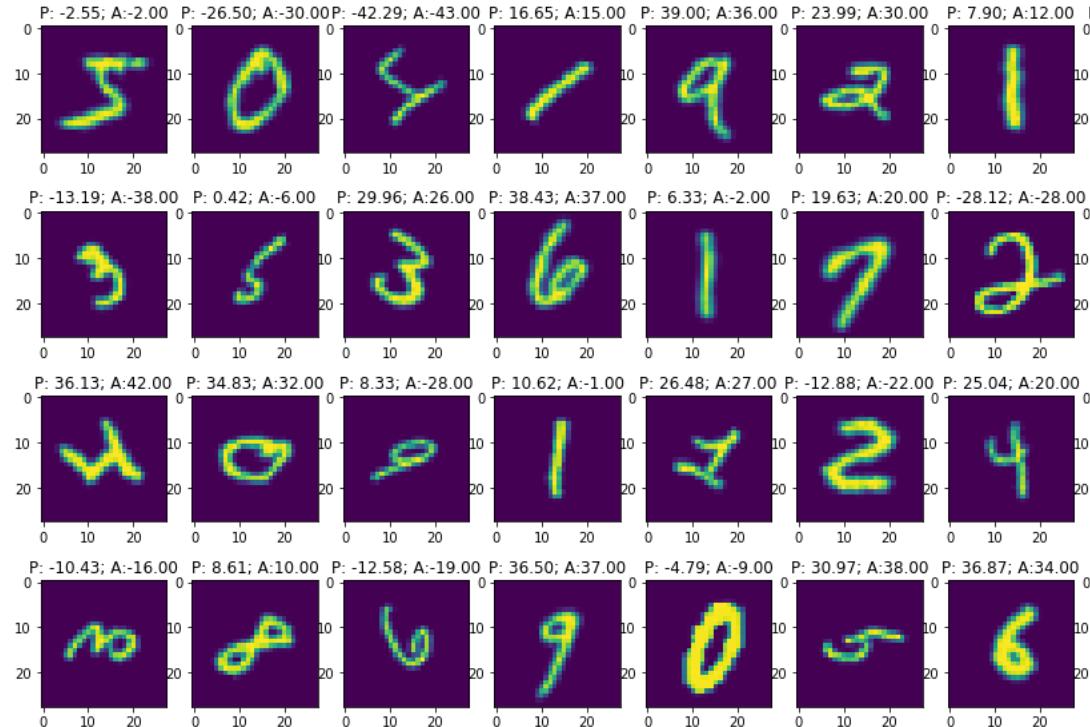
- Almost identical to "My First CNN"
 - CNN architectures are very adaptable
- One change
 - Our final dense layer is now size 1
- We also change our loss
 - MSE rather than categorical cross entropy
 - Could also use MAE (mean absolute error), or other regression loss

Model: "mnist_angles_cnn_model"		
Layer (type)	Output Shape	Param #
img (InputLayer)	[(None, 28, 28, 1)]	0
conv2d (Conv2D)	(None, 26, 26, 8)	80
max_pooling2d (MaxPooling2D)	(None, 13, 13, 8)	0
conv2d_1 (Conv2D)	(None, 11, 11, 16)	1168
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 16)	0
conv2d_2 (Conv2D)	(None, 3, 3, 32)	4640
flatten (Flatten)	(None, 288)	0
dense (Dense)	(None, 64)	18496
dense_1 (Dense)	(None, 1)	65

Total params: 24,449
Trainable params: 24,449
Non-trainable params: 0

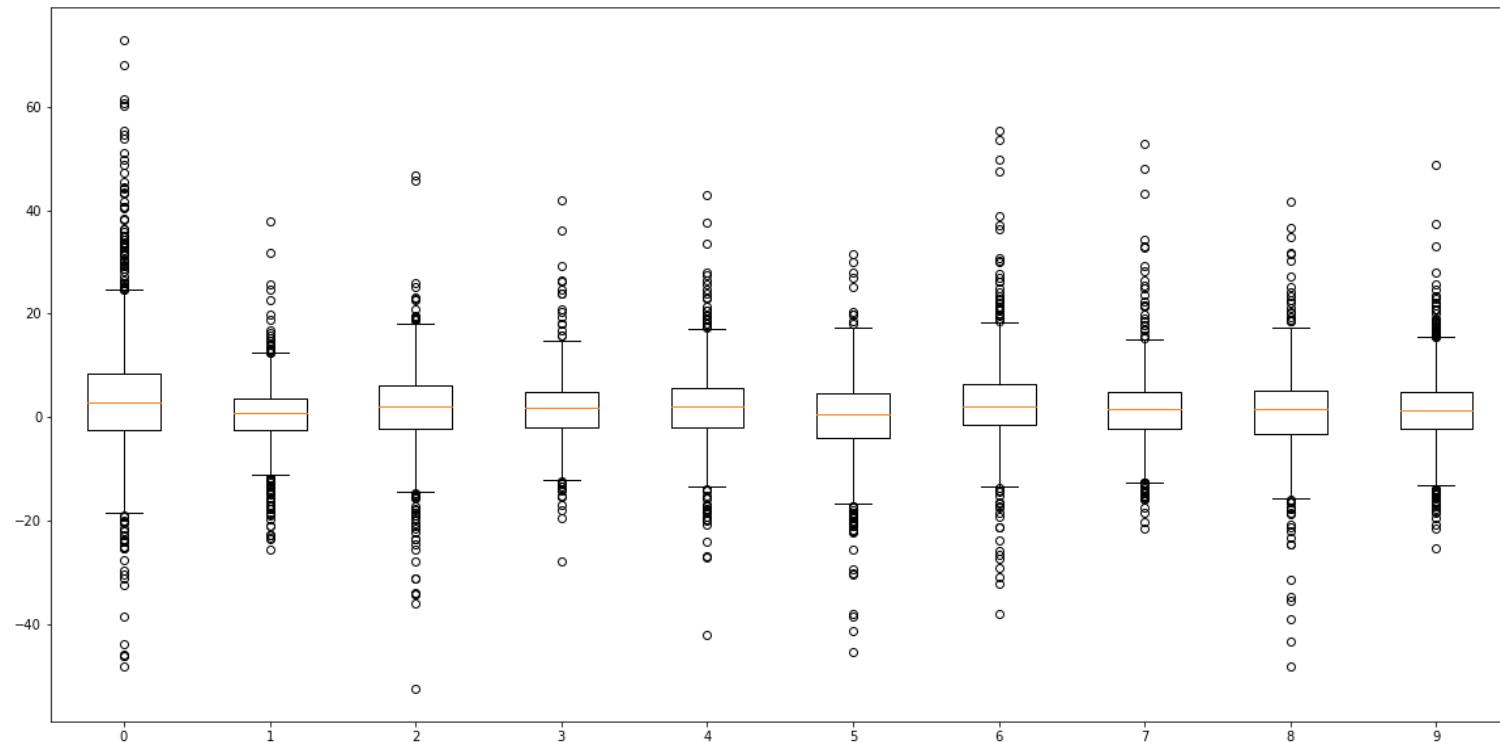
Results

- Model is fairly accurate
 - Can estimate rotation for all digits
- Network has no explicit knowledge of the digits



Results

- Network finds 0 the hardest to correct
 - Performance broadly similar for all digits though



CAB420: What is Learned?

PARAMETERS, LOTS OF PARAMETERS

DCNNs as a “Black Box”

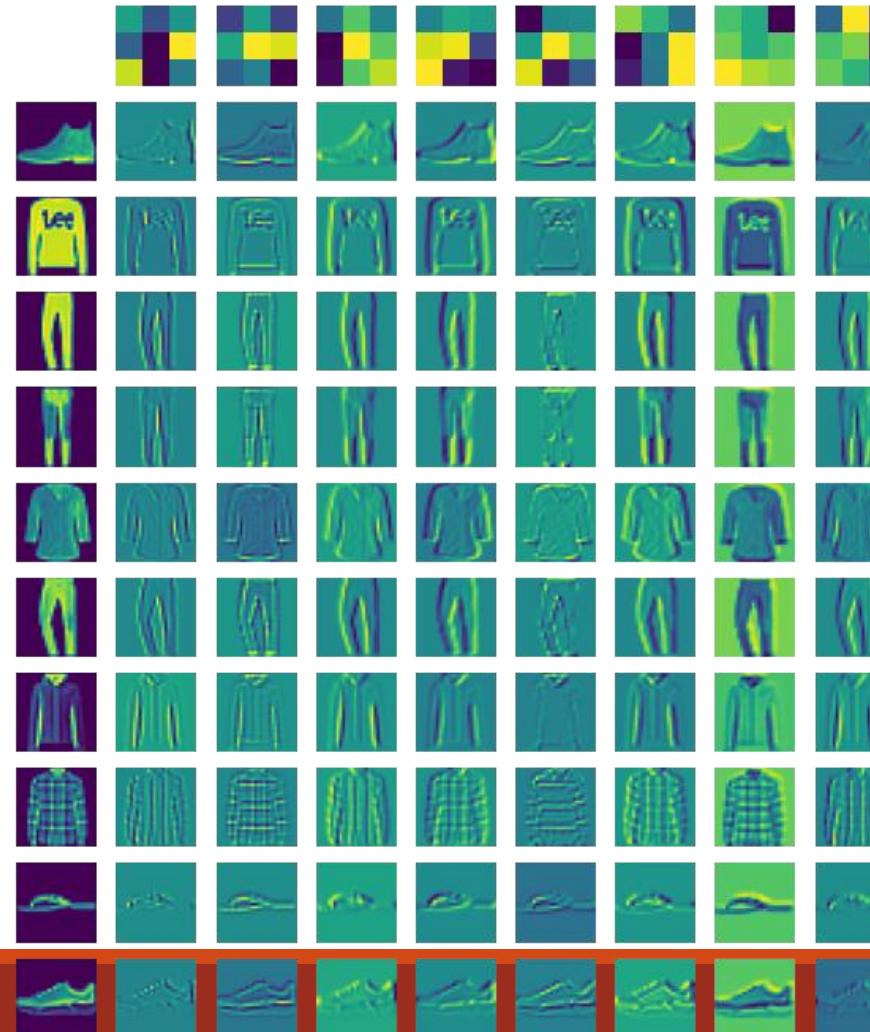
- You will often see DCNNs and Deep Learning models referred to as a “Black Box”
 - Based on the idea that such models are hard to (or even impossible to) understand or interpret
- Consider a linear regressor
 - Learned parameters are the values of β
 - β define the importance and direction of influence for each variable
- For a DCNN the “meaning” of the learned parameters is less obvious
 - There are also many, many more of them, which is really the problem
 - But we can access and visualize parameters if we wish
 - See ***CAB420_DCNNs_Example_3_What_Does_the_Network_Learn.ipynb***

Dense Layers

- For a dense layer we have:
 - An input vector, x , of length M
 - An output vector, y , of length N
- The dense layer learns
 - $y = wx + b$
 - where w is a matrix of size $M \times N$, and b is a bias vector of size N
- The dense layer operation can be decomposed as follows:
 - $y[0] = w[:, 0]x + b[0]$
 - $y[1] = w[:, 1]x + b[1]$
 - ...
 - $y[N - 1] = w[:, N - 1]x + b[N - 1]$
- Each of the above lines is a single linear regressor
 - The values in w simply indicate the strength and direction of influence

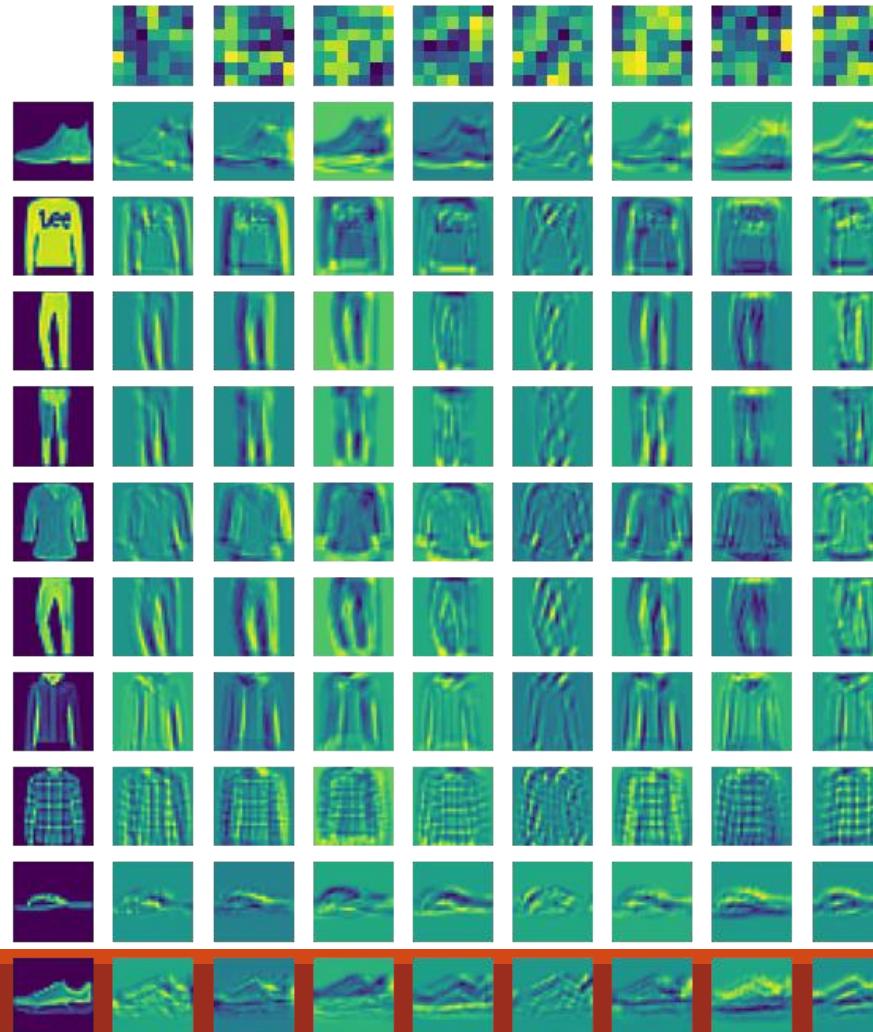
Convolution Layers

- Convolution layers learn a set of filters
 - We can visualise the filters, and their output (response)
- Left column: Input images
- Top row: Learned 3x3 filters
 - First convolutional layer
- The rest: Responses to filters
 - Notes that each image has been scaled independently
- Filters focus on edges
 - Different filters capture different edges
- Filters are unique
 - But limited 3x3 patterns are possible



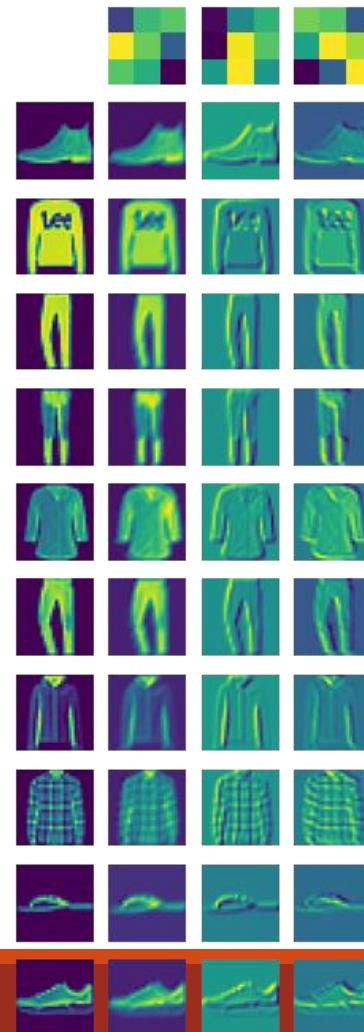
Larger Convolution Filters

- Same setup as before, but with 7x7 filters
 - More complex filters leads to more complex response maps
 - Larger filters consider a larger area of the input image
- Many more possible 7x7 patterns than 3x3



Stacking Convolution Filters

- Two convolution layers
- First layer has three 3x3 filters
 - Response maps similar to what we saw before, just less of them
 - Detecting edges in different orientations

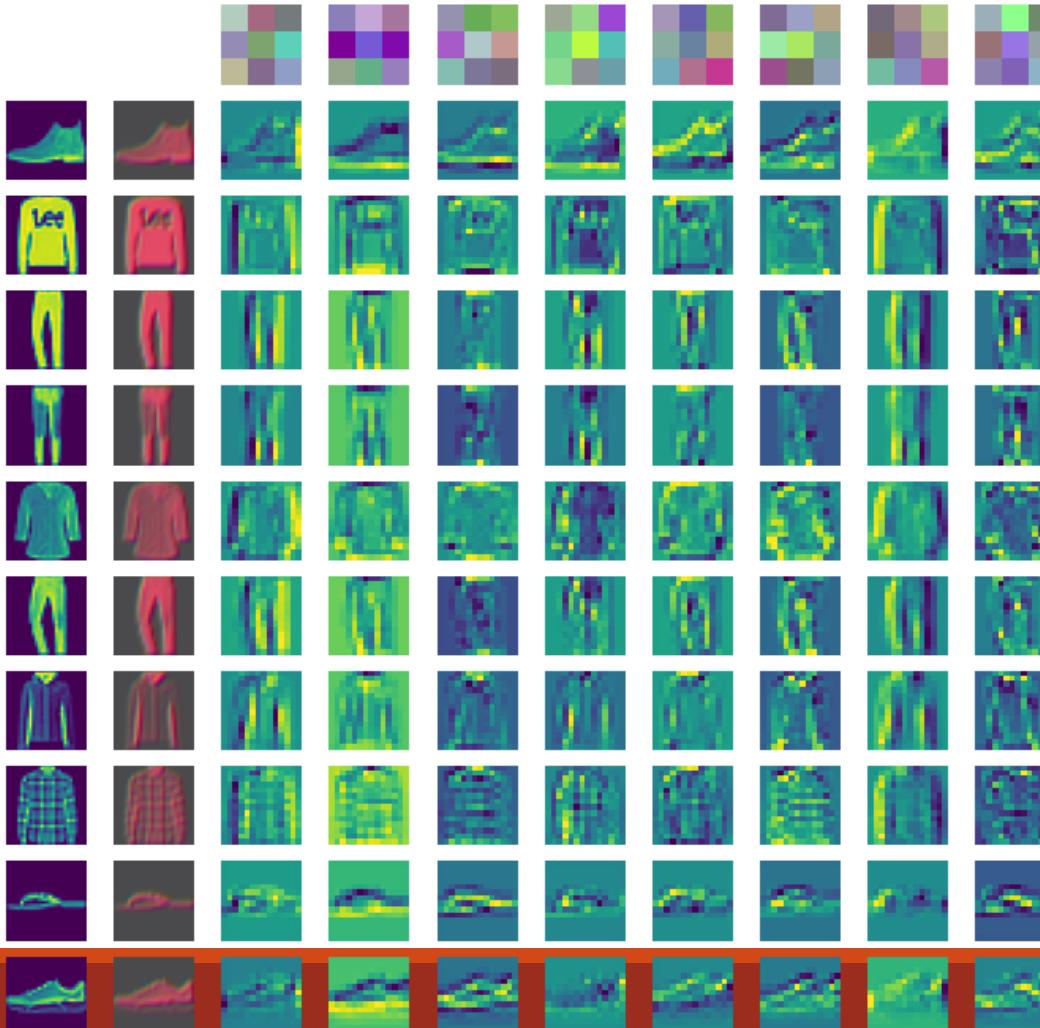


Stacking Convolution Filters

- Left column: Input Images
- 2nd column from left: Output of first layer

- Second layer has 8 3x3 filters
- Second layer's input is the stacked responses from the first layer
 - First layer has three filters, first layer output is a three channel image
 - Second layer learns 3x3x3 filters
 - There is also a max-pooling between the two layers

- Filter responses much more complex than first layer
 - Similar in complexity to 7x7 filters
 - Filters are looking for interactions between outputs of the first layer



Interpreting DCNNs

- The challenge is not that we can't get to the parameters, it's that there's so many of them
- In practice
 - We will rarely, if ever, visualise learned weights (i.e. the filter kernels)
 - We will look at intermediate outputs at times
 - Visualise filter responses to understand what the network is looking at
 - We use other techniques to understand what a network is looking at
 - Class Activation Maps (CAM) and Gradient-weighted Class Activation Maps (Grad-CAM)
 - Indicate what regions of image contribute to the score for a class
 - Shapely Values
 - Indicate the contribution of each input dimension to a decision
 - Neural Conductance
 - Captures information flow through the neural network
 - And many, many, more
- Interpreting DCNNs is outside the scope of CAB420, but it's important to be aware that these methods exist
 - But if you are interested, see the bonus examples

CAB420: Training Your Network

CAUSE WE KIND OF IGNORED THAT BEFORE

Back Propagation and Gradient Descent

- Neural Networks are trained using Back Propagation and Gradient Descent
 - Change the weight and bias terms using gradient descent
 - Can have problems when
 - Gradients becomes very small (vanishing gradients)
 - Gradients become very big (exploding gradients)
 - Partial derivatives are used to update parameters
 - Becomes very complex, for large networks
 - Occurs behind the scenes in CAB420
 - Back propagation is a crucial component of neural networks that allows for optimisation of the cost function.
 - Gradient descent allows us to approach an optimal solution over a number of iterations

Gradient Descent

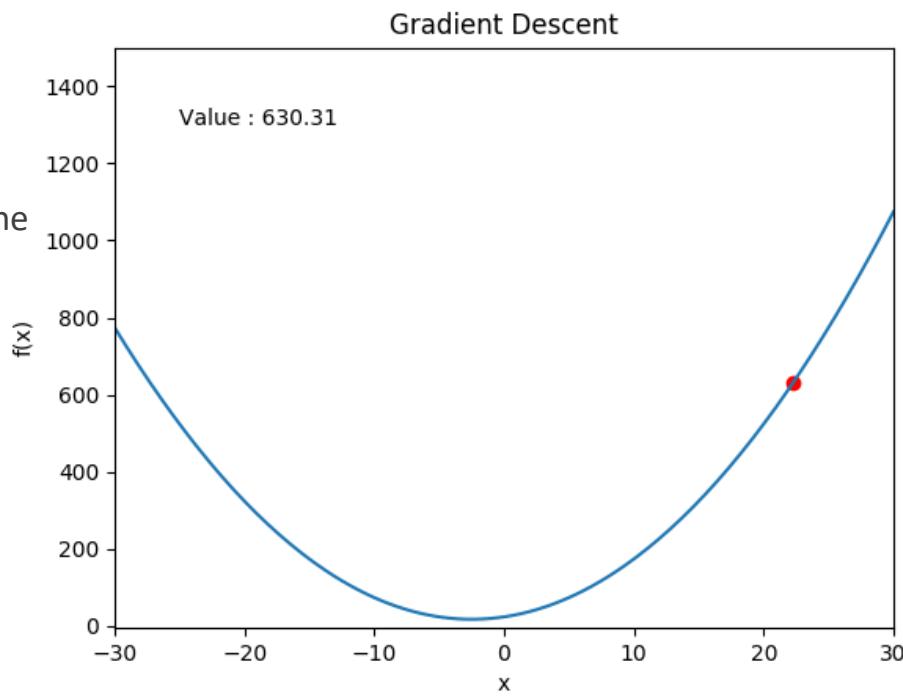
ROLLING BALLS DOWN HILLS

Optimisation

- Usually in machine learning we can't directly determine a model's parameters
 - i.e. we can't directly determine model coefficients
- In such cases we can use an iterative approach:
 - Make an initial estimate
 - Evaluate that estimate
 - Update the estimate and evaluate again
 - Repeat until either
 - The estimate stops changing (or only changes slightly)
 - A maximum number of steps is reached

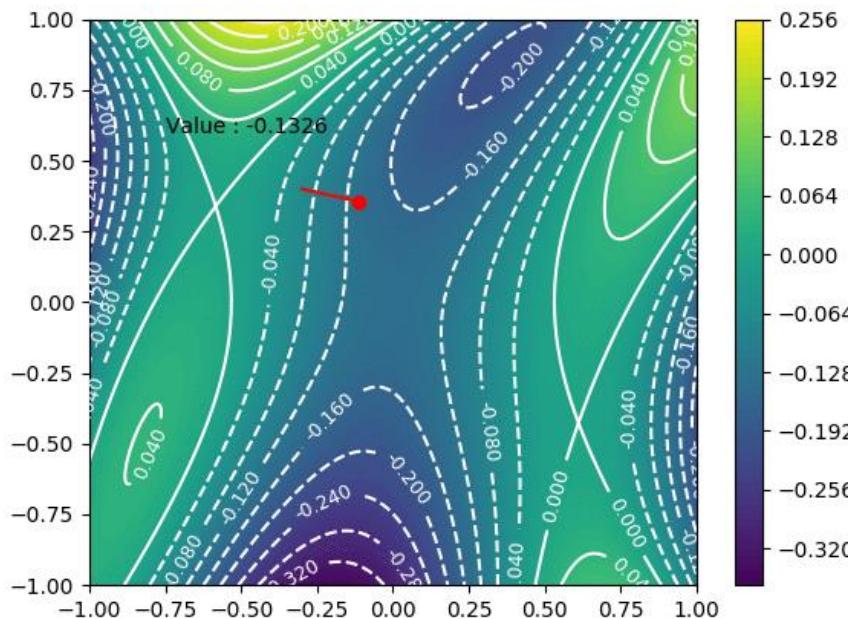
Gradient Descent

- One of the most popular optimisers is Gradient Descent
 - Start at some estimate
 - Evaluate the gradient
 - Move in a direction that minimises the gradient



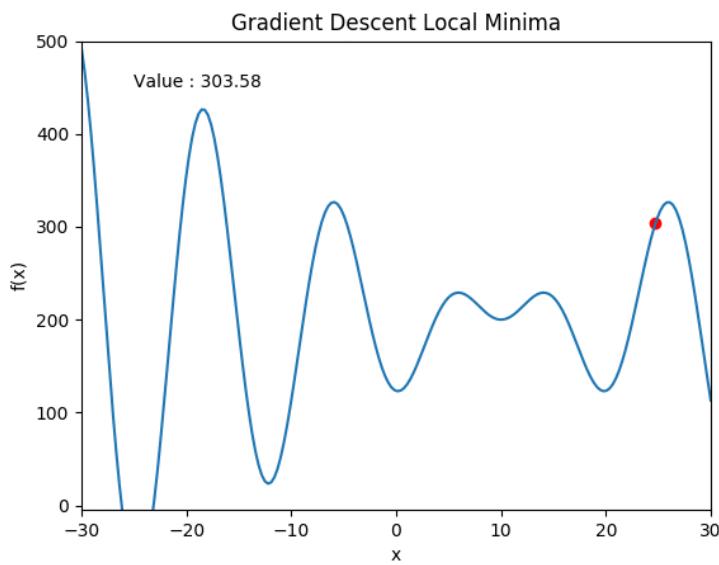
Gradient Descent

- Scales to an arbitrary number of dimensions
 - Uses partial derivatives to determine gradients



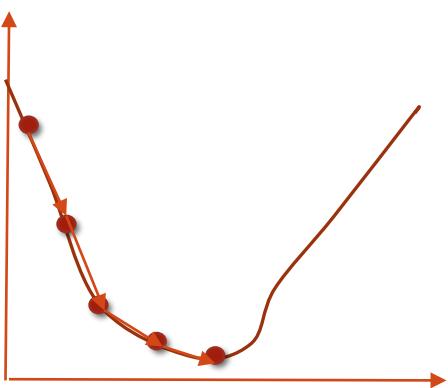
Gradient Descent

- Sensitive to starting conditions
 - Can get stuck in local minima rather than finding global minima

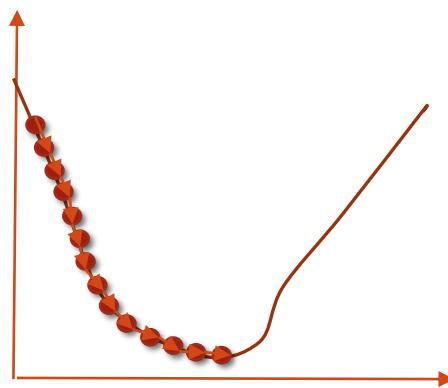


Gradient Descent

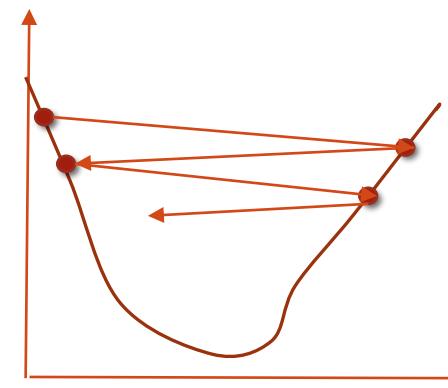
- Learning rate is important
 - How much do you change the model each step?
- Too slow
 - Takes a long time to get to a solution
 - More prone to getting stuck in local minima
- Too big
 - Can “jump over” the best solution



Good Learning Rate



Slow Learning Rate



Fast Learning Rate

Training DCNNs

BY ROLLING BALLS DOWN HILLS

Terminology

- Epoch
 - One complete pass through the data
 - After one epoch, the network has seen all examples
- Batch
 - One update of the networks, based on a small sample of data
- Optimiser
 - Gradient descent approach that we use to train
- Learning Rate
 - How fast we allow the model parameters to update

Why not train on all data at once?

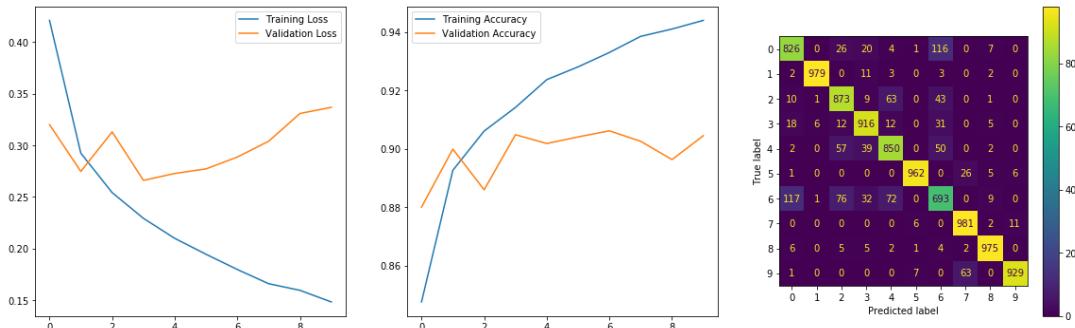
- Consider Fashion MNIST, we have
 - $28 \times 28 \times 50,000 = 39,200,000$ pixels
 - That's a lot to process at once
 - And this is a "toy" dataset
- For most tasks, parsing all data at once is not practical
 - Hence, batches
- A batch is a smallish collection of inputs
 - Usually somewhere between 1-256 depending on
 - How much data you have
 - How big the network is
 - How much money you spent on hardware (or how much you can fit in memory)

Batch Size vs Epochs

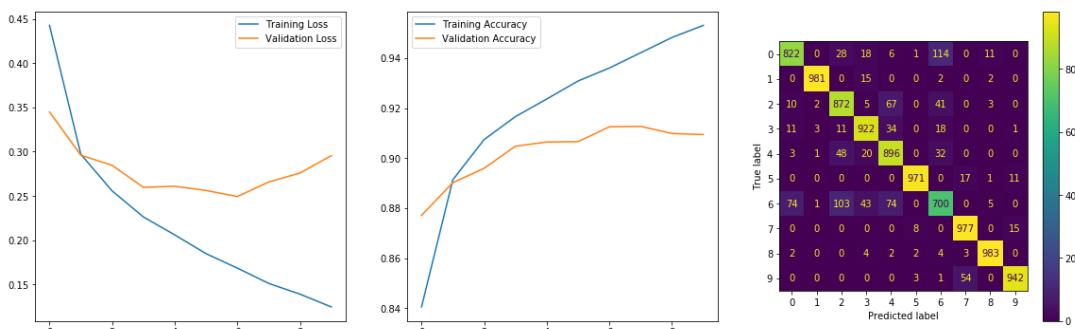
- A small batch size means
 - More updates per epoch
 - Can train the network in fewer epochs because you have more updates
 - But....
 - Each batch is less representative of the overall data shape
 - Can lead to a poor fit depending on how imbalanced data is

Impacts of Batch Size

- See **CAB420_DCNNs_Additional_Example_3_Training_Parameters.ipynb**
- Batch size = 4

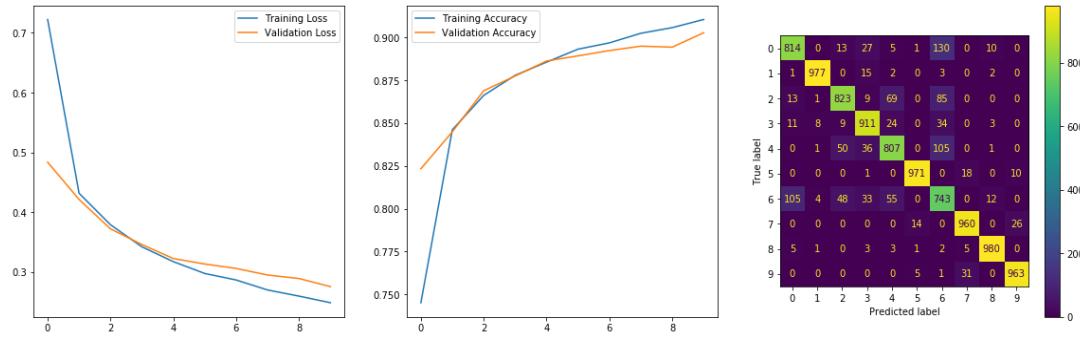


- Batch size = 16

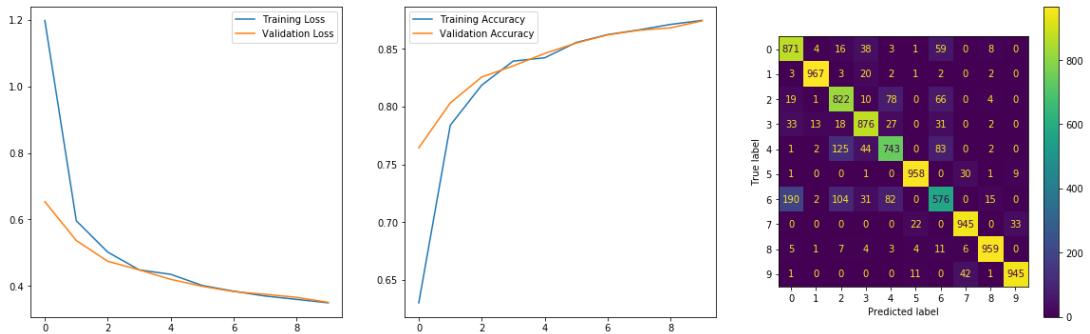


Impacts of Batch Size

- Batch size = 256



- Batch size = 1024



- Larger batch size leads to

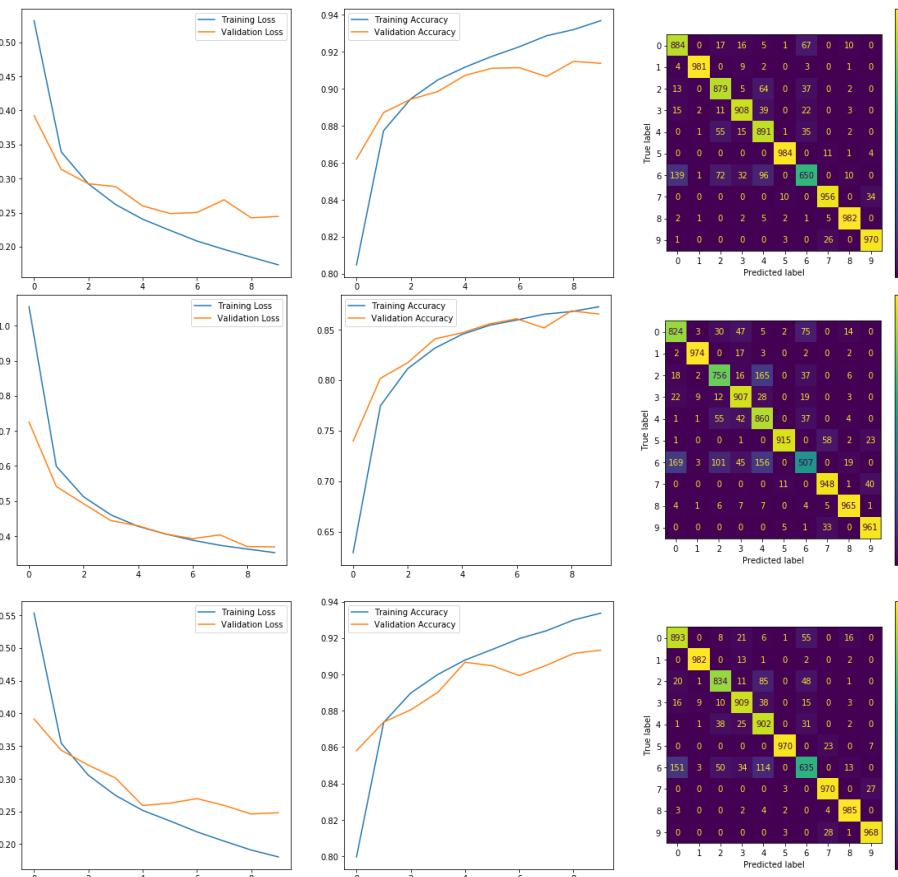
- Smoothened training
- Slower convergence (in terms of epochs)
- Higher memory requirements

Optimisers

- All based on gradient descent to train via backpropagation
 - Propagate gradients back up the network to adjust weights
- Many options exist
 - There is no real “standard” optimiser
 - Adam is the closest thing to a default
 - Differences between optimisers are often small (see <https://arxiv.org/abs/2007.01547>), and the variation in performance for a single optimiser is often larger than the difference between optimisers
 - Some tasks or networks will work better a given optimiser
 - This is not consistent however

Optimisers

- Three training runs
 - RMSProp (Top)
 - SGD (Middle)
 - Adam (Bottom)
- SGD slower to learn
- All achieve similar performance
 - Our model and data are not complex and so similar performance is achieved for all
 - If you re-run this, you will see some variation. Maybe SGD will be a bit quicker next time.
 - You will likely see minimal variation in CAB420 with regards to optimiser choice
 - If in doubt, use Adam

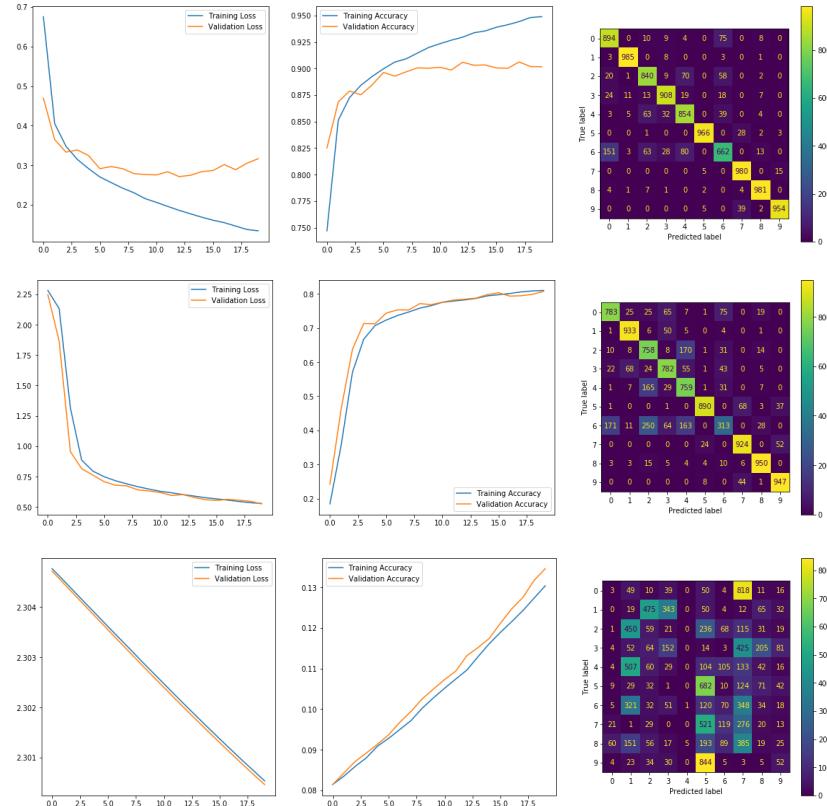


Learning Rate

- Bigger number -> Faster Learning
- Faster learning can be good early
 - You're a long way from a solution
- Slower learning is better once you have a good estimate
 - With fast learning, the danger is you “overshoot” the solution
- A good practice is to use a learning rate schedule
 - For example, start fast, drop by a factor of 10 every 10 epochs

Learning Rate

- Fast (0.1)
 - Converged after ~8 epochs
- Slow (0.001)
 - Almost converged after 20 epochs
- Glacial (0.00001)
 - Nowhere near convergence after 20 epochs



Avoiding Overfitting

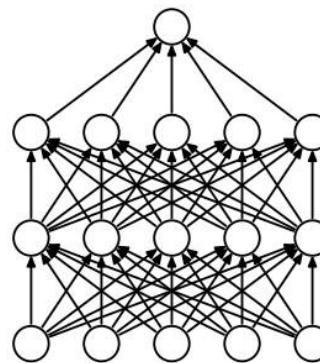
AND OTHER TRICKS AND HACKS

Overfitting with DeepNets

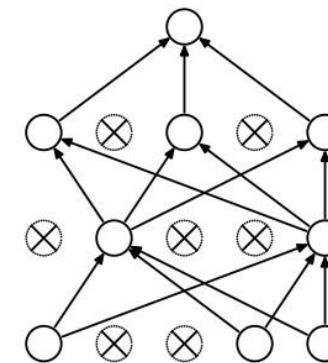
- It's really easy to overfit
 - Potentially millions of parameters
 - Almost always have more parameters than samples
- Two possible approaches
 - Modify the network to reduce overfitting chance
 - Get more data
 - Or make it up

Drop Out

- Randomly disconnect a portion of neurons each pass through the network
- Why?
 - Means we never learn on the whole network at once
 - Reduces overfitting
 - But slows training
- Can be applied at different levels
 - At neurons
 - At Convolutional Filters
 - Spatial Dropout, drops a percentage of whole filters



(a) Standard Neural Net



(b) After applying dropout.

Batch Normalisation

- Neural networks propagate information from one layer to the next
 - Layer N takes results of Layer N-1 as input
 - And N-1 takes results of N-2, and so on
 - What if the range of values coming from N-1 keeps changing?
 - This tends to happen a lot during the early stages of training
- Batch Normalisation helps address this
 - Improves training speed
 - Reduces overfitting

Batch Normalisation

- Batch normalisation normalises the output of a batch at a designated point in the network
 - By default, 0 mean and unit std.dev
 - But can learn a different mean and std.dev
- Why?
 - If we perform batch norm after layer N-1, we now know that the input to Layer N will have 0 mean and unit std.dev
 - Makes it easier to learn layer N as the layer will always get data in the same range
 - Essentially provides a model checkpoint

Batch Normalisation

- Layer placement impacts performance
 - Generally, place before an activation
 - BatchNorm will standardise outputs around a learned mean
 - If placed after an activation, outputs have been altered by the activation
 - Placing before an activation makes it easier to consider the impact of the proceeding layer
- Not needed after every layer
 - Consider adding after repeating blocks
 - i.e. after pairs of convolutions
 - Experiment with placement

Weight Regularisation

- Neural networks have lots of weights
 - Big weights can indicate overfitting
 - Much like Ridge regression, we'd prefer smaller weights
- Weight Regularisation applies a penalty to the network based on the total sum of the weights
 - Can be L2 (like Ridge regression)
 - Or L1 (like Lasso)
 - Or a combination
- In Keras/Tensorflow
 - Specified per layer on any (or all) of
 - Weights (kernal)
 - Bias
 - Activation
 - Flexible in terms of regulariser (L1, L2, L1 and L2, custom) used
 - Off on all layers by default

Demo Script

- See ***CAB420_DCNNs_Additional_Example_4_Layer_Order_and_Overfitting.ipynb***
 - Explore in your own time
 - Don't feel you need to understand everything immediately
 - Play with the options in here over time
 - Feel free to ask questions, and try things out in different examples

DCNNs and Variation

YMMV

Network Initialisation, Training and Randomness

- There is lots of randomisation in a neural network training process
 - Network parameters (weights and biases) are randomly initialised
 - Data is randomly shuffled after each epoch
 - Training and validation splits may be random
- Unless controlled for, no two training runs are exactly the same
 - If you're training to convergence (or close to), they will be similar

An Experiment

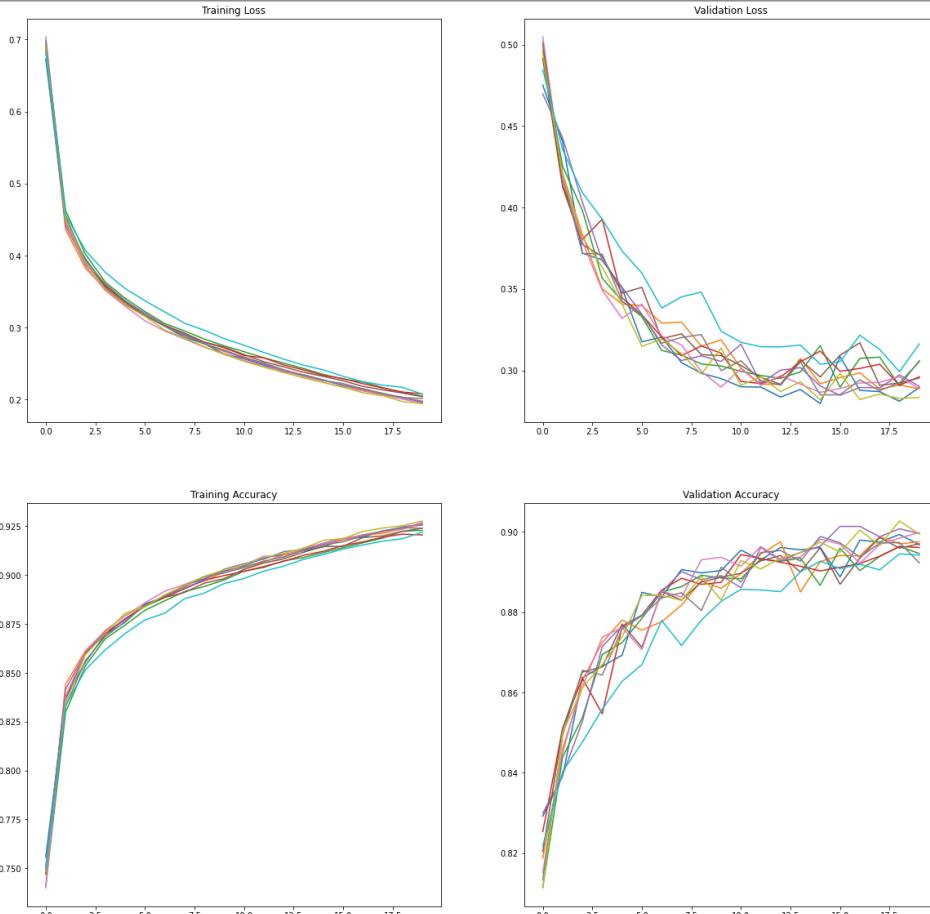
- See ***CAB420_DCNNs_Additional_Example_5_Variation.ipynb***

- 10 identical simple CNNs
- All trained on Fashion MNIST for the same length of time
- Same batch size, same optimiser
- Seek to see what sort of variation is observed in the models

10 Models

- Results are all similar
 - But there is variation
- More variation in validation performance than testing performance

- A note on convergence
 - Validation loss and accuracy curves have flattened out in these plots
 - Training much beyond the 20 epochs leads to overfitting
 - Training accuracy will continue to improve towards 100% if training continues



Differences in Predictions

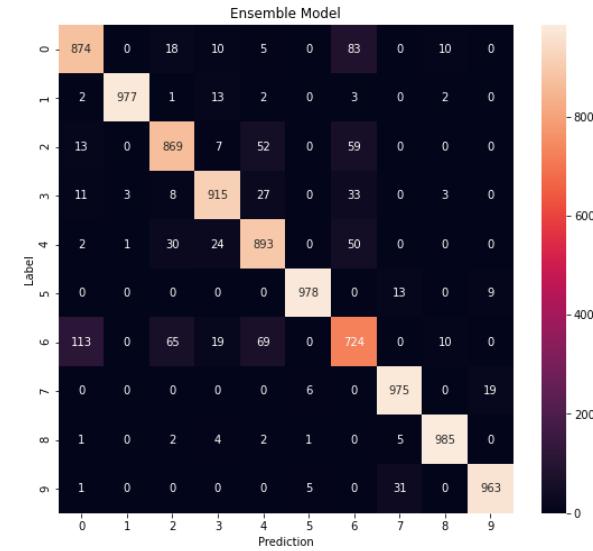
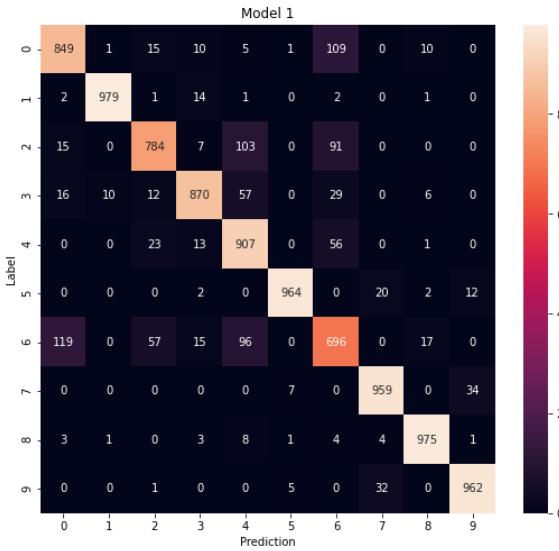
- Let's consider some misclassified examples
 - Examples are misclassified by the first model
 - Report results for all models and the ground truth
- Different models (sometimes) make different decisions
 - Models can vary in that some may be right and wrong
 - Others may have different wrong answers
 - Even when all models make the same prediction, the SoftMax value varies

```
Index 17; True Class: 4
Model 1, Predicted class 6 (0.708114)
Model 2, Predicted class 4 (0.970968)
Model 3, Predicted class 2 (0.649878)
Model 4, Predicted class 2 (0.569351)
Model 5, Predicted class 6 (0.894694)
Model 6, Predicted class 4 (0.583097)
Model 7, Predicted class 4 (0.917198)
Model 8, Predicted class 4 (0.984307)
Model 9, Predicted class 4 (0.945560)
Model 10, Predicted class 2 (0.624152)
Average Model, Predicted class 4 (0.506741)

Index 23; True Class: 9
Model 1, Predicted class 5 (0.999711)
Model 2, Predicted class 5 (0.999991)
Model 3, Predicted class 5 (0.999992)
Model 4, Predicted class 5 (0.998271)
Model 5, Predicted class 5 (1.000000)
Model 6, Predicted class 5 (1.000000)
Model 7, Predicted class 5 (0.999940)
Model 8, Predicted class 5 (0.840722)
Model 9, Predicted class 5 (0.999996)
Model 10, Predicted class 5 (0.999162)
Average Model, Predicted class 5 (0.983778)
```

Averaging Models

- We can create an ***ensemble*** of models
 - Average the results of a set of identical models
 - “The wisdom of the crowds” for deep nets
 - Similar to Random Forests
 - Though each “tree” is a bit more complex
- Original models all perform at around 89% accuracy
 - Model 1 achieves 89.45%
- Ensemble achieves 91.53%
 - Small, but noticeable gain
- Is this worth it?
 - 2% performance gain for 10x the compute
 - However we see diminishing returns as we increase complexity anyway. Is this much worse?
 - This has the added benefit of giving us a way to measure confidence
 - This is otherwise difficult with deep networks

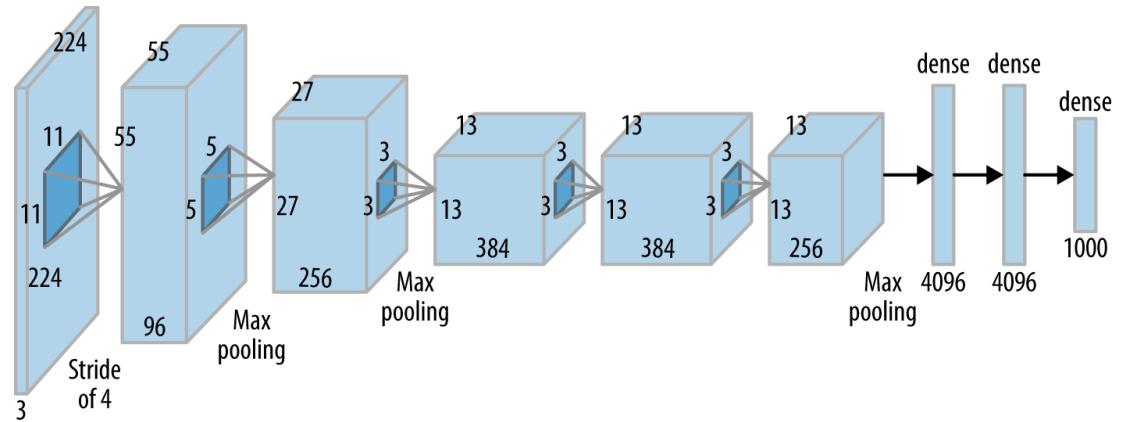


CAB420: Network Depth

AND IT'S ADVANTAGES AND PITFALLS

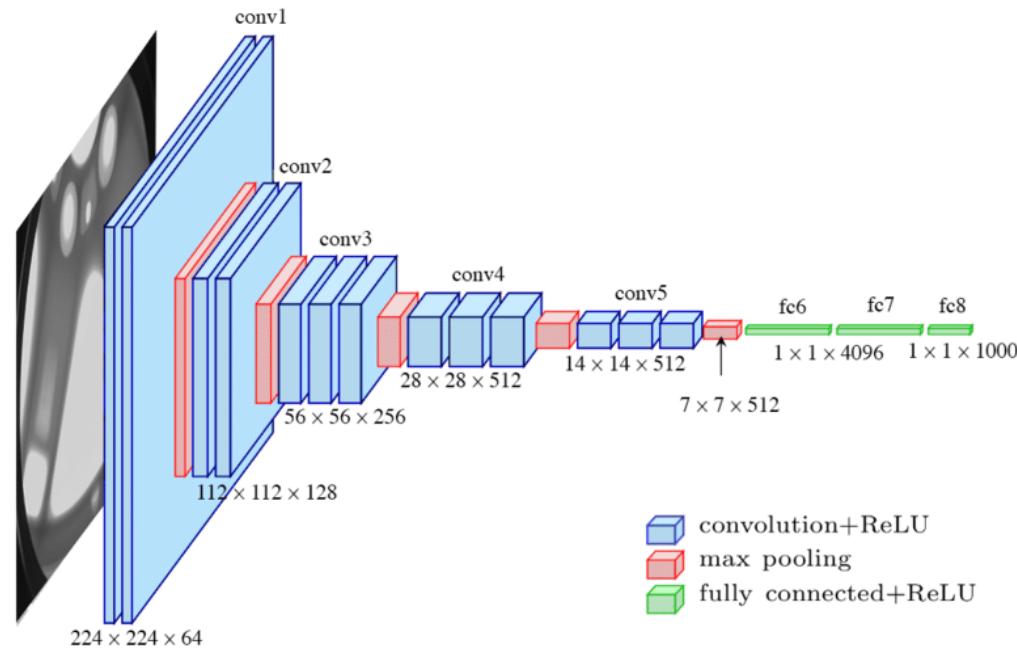
AlexNet

- AlexNet is where DCNNs really got started
 - 8 computational layers (really not that deep)
 - 5 convolutional layers
 - 3 fully connected layers
- Large filters in early convolution layers
 - 11x11 filters in first layer (96 filters)
 - 5x5 filters in second layer (256 filters)



VGG Style Networks

- So far, we've played (mostly) with VGG-style networks
 - Two (or more) small (3x3) convolutions, followed by a max-pool
 - Rinse and repeat
 - End with FC layers



Convolution Layers and Filters

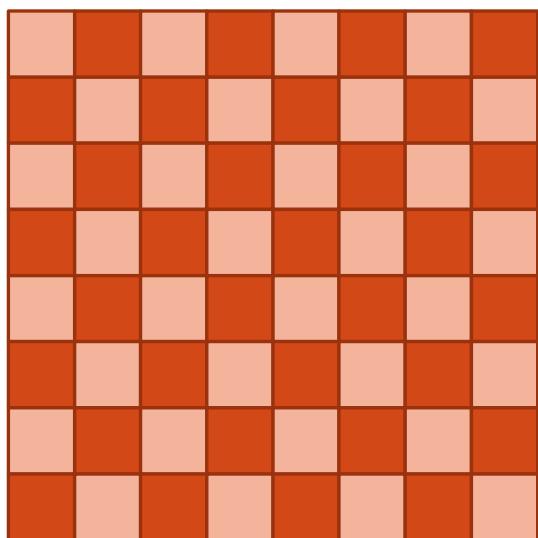
- Filters are of a fixed size
 - Set during network design
 - Usually, odd numbered square size
- Size controls
 - How much of an image a filter operates over
 - Bigger filters see more of an image at a time
 - How many parameters are in the filter
 - Bigger filters mean more parameters

Why Small Convolutions?

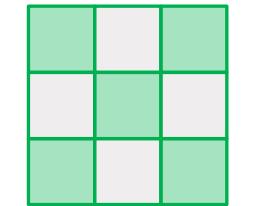
- Why do we use 3x3 convolutions rather than bigger filter sizes?
- Intuitively
 - Smaller filters see smaller patches of the image
 - They extract more localised features
 - They can't extract or "see" large patches of the image, and thus should miss large features
- But
 - If we stack them deep enough, we overcome this

Receptive Fields

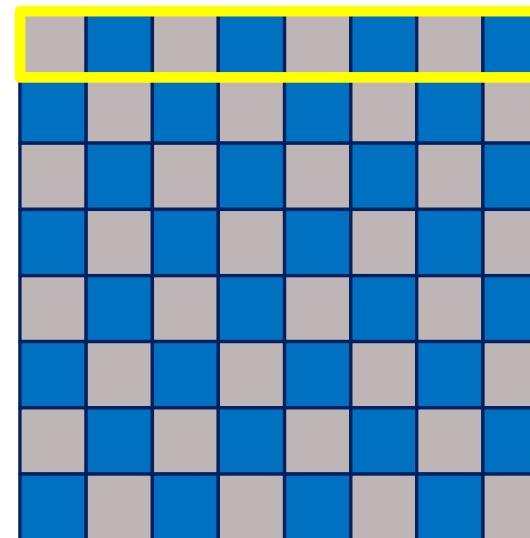
- How much of an image a filter can effectively see
- Consider the following
 - Input image, followed by
 - 3x3 Conv2D layer



Input Image



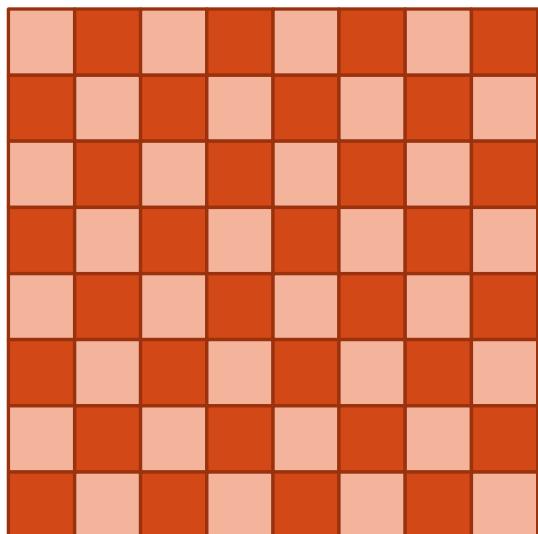
3x3 Conv Filter



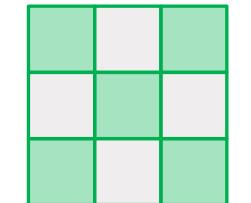
Output Features

Receptive Fields

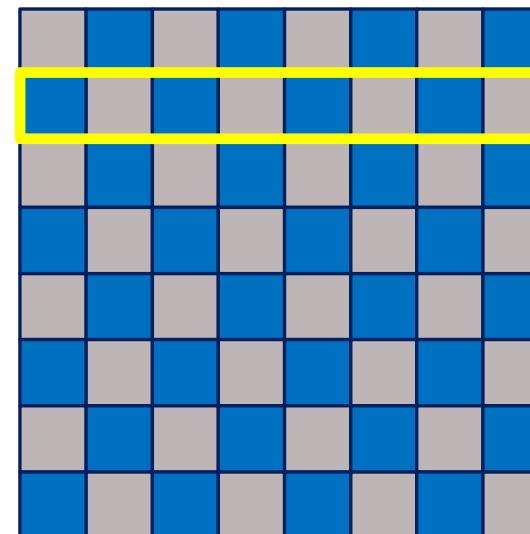
- How much of an image a filter can effectively see
- Consider the following
 - Input image, followed by
 - 3x3 Conv2D layer



Input Image



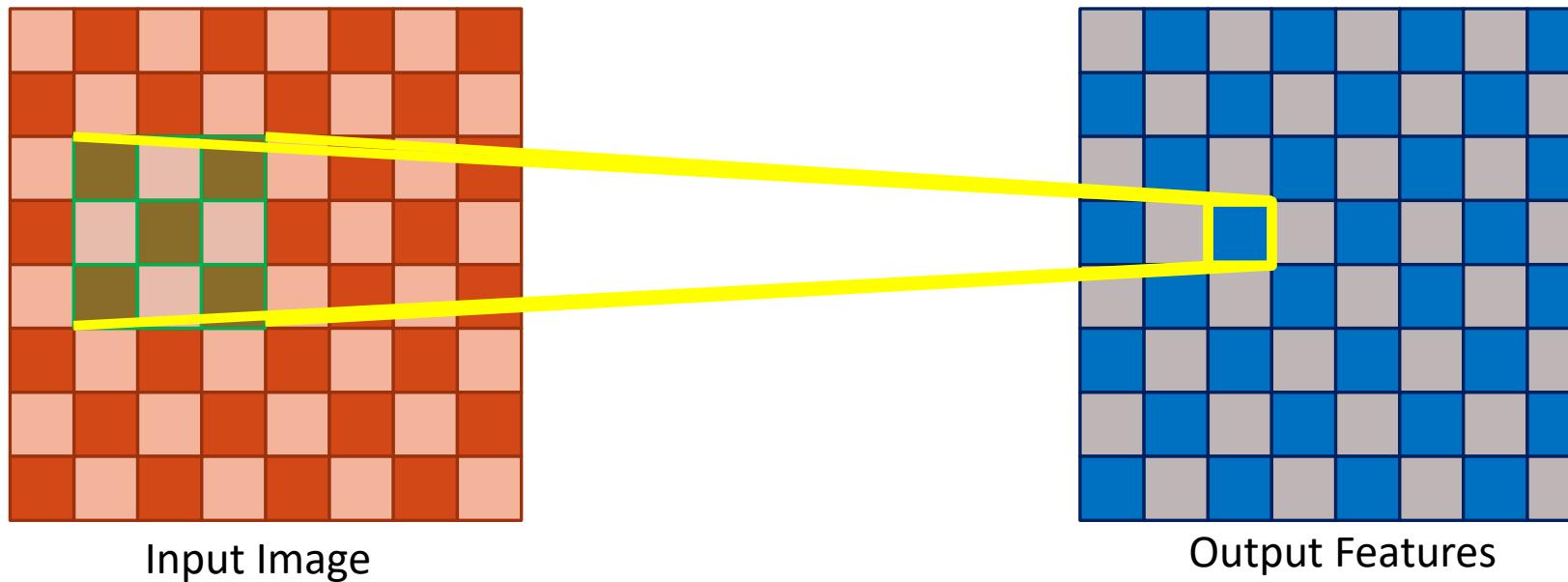
3x3 Conv Filter



Output Features

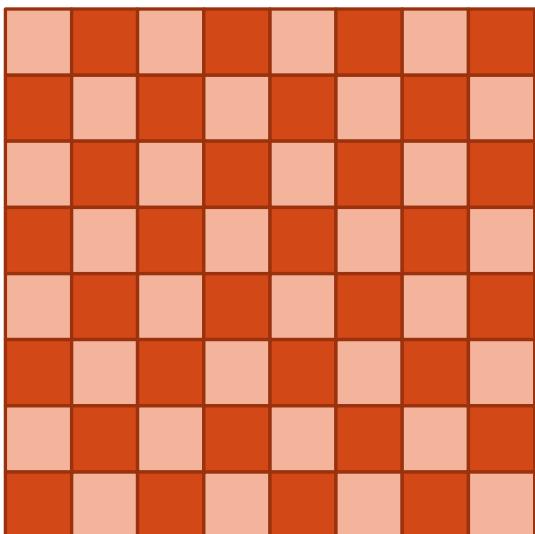
Receptive Fields

- How much of an image a filter can effectively see
- Consider the following
 - Input image, followed by
 - 3x3 Conv2D layer

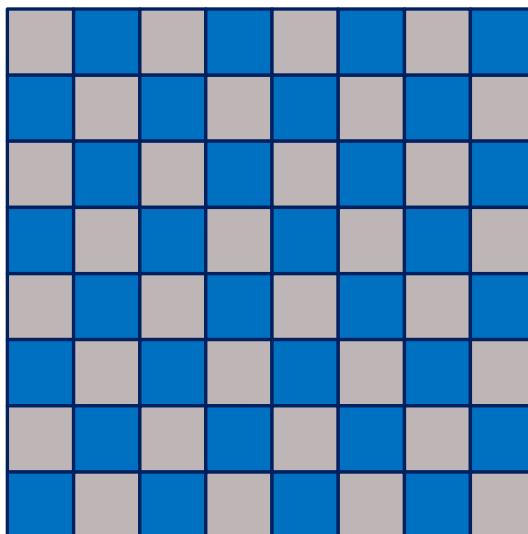


Receptive Fields

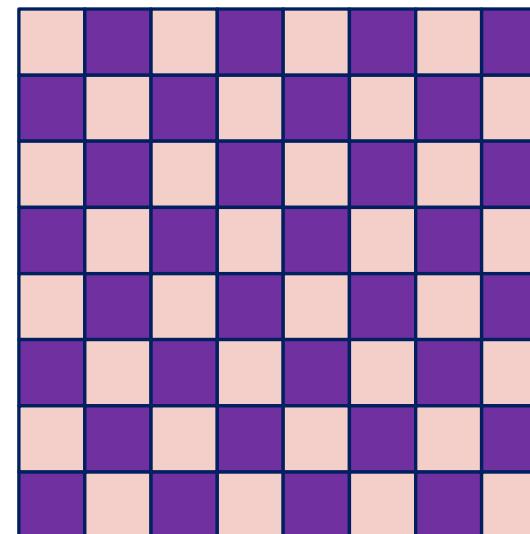
- Now consider
 - Input image, followed by
 - 3x3 Conv2D layer, followed by
 - 3x3 Conv2D layer



Input Image



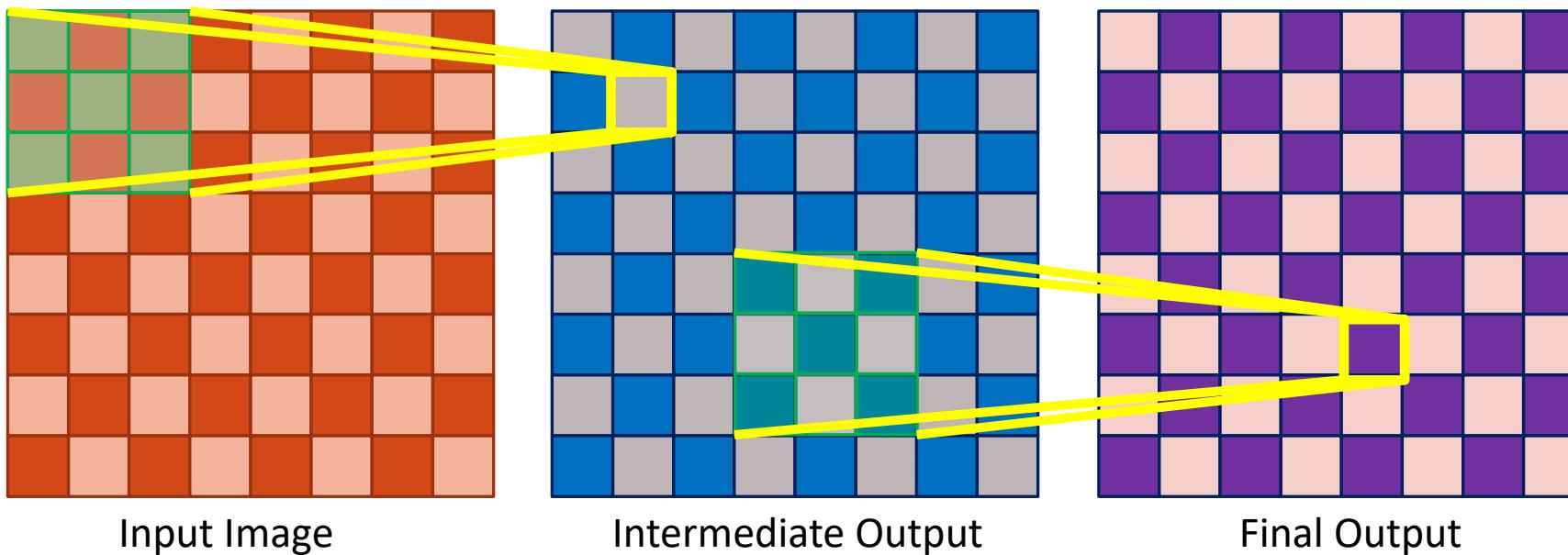
Intermediate Output



Final Output

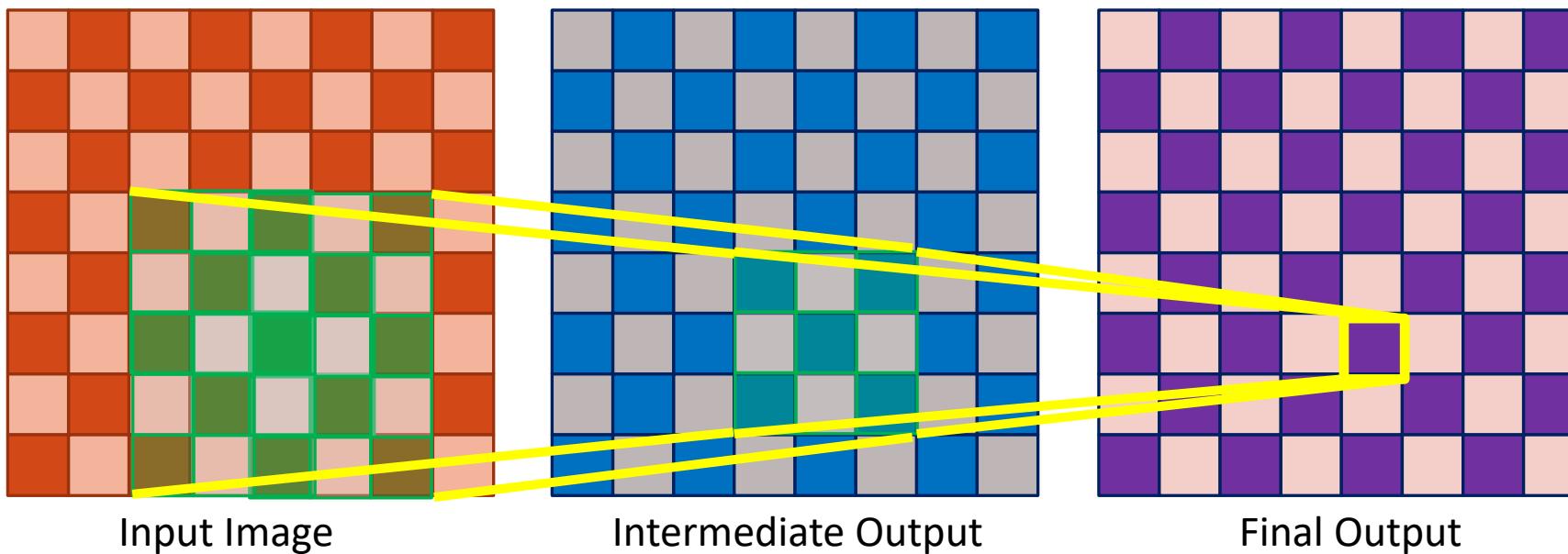
Receptive Fields

- Now consider
 - Input image, followed by
 - 3x3 Conv2D layer, followed by
 - 3x3 Conv2D layer



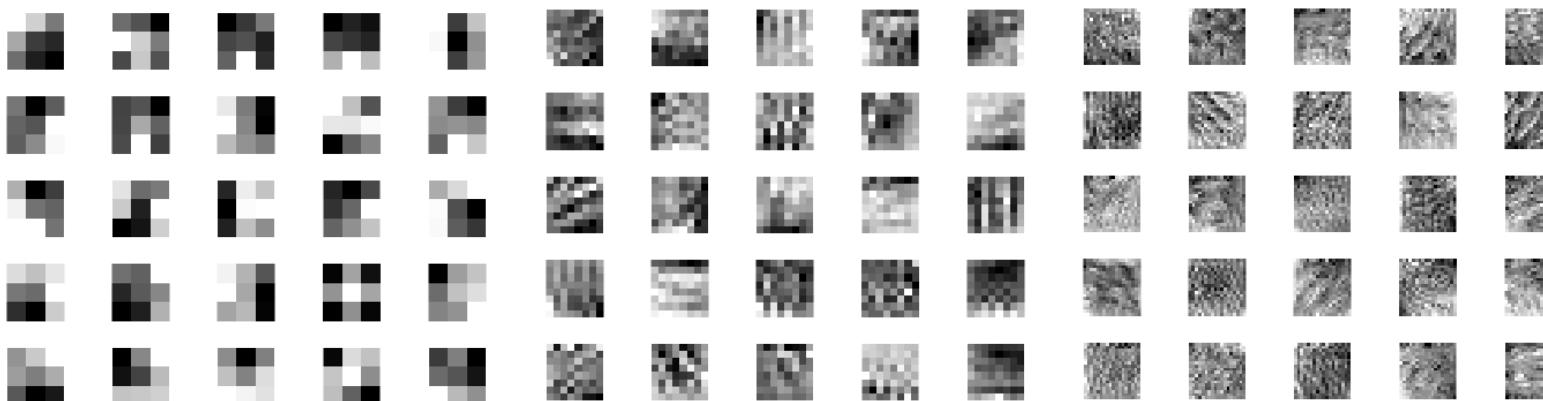
Receptive Fields

- Now consider
 - Input image, followed by
 - 3x3 Conv2D layer, followed by
 - 3x3 Conv2D layer



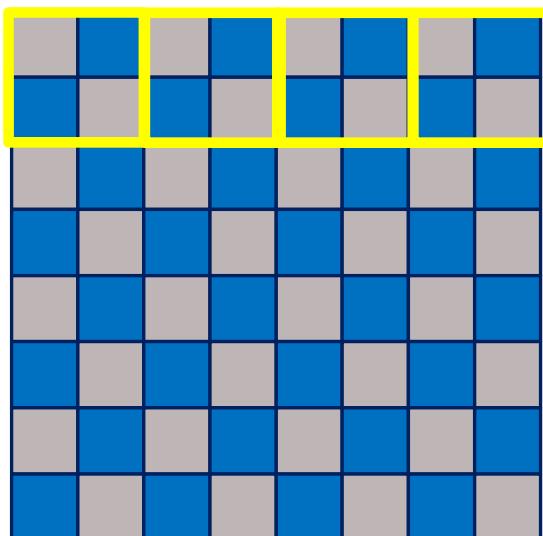
Stacking Convolutions

- As we stack convolutions, we get bigger receptive fields
- But, why use two 3x3 layers over one 5x5 layer?
 - With multiple filters we learn more complex representations
 - Filters “build” on each other
 - Below (left to right)
 - conv1, conv3 and conv5 from a simple VGG trained on Fashion MNIST

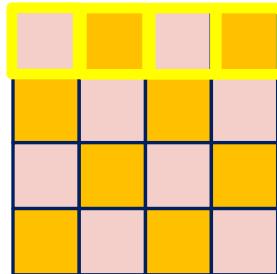


Convolutions with Max-Pooling

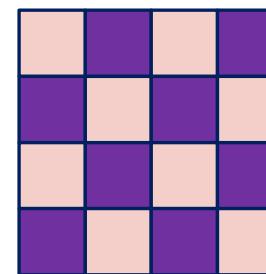
- Now consider
 - 3x3 Convolution Layer
 - 2x2 Max-Pooling
 - 3x3 Convolution Layer



Input Features



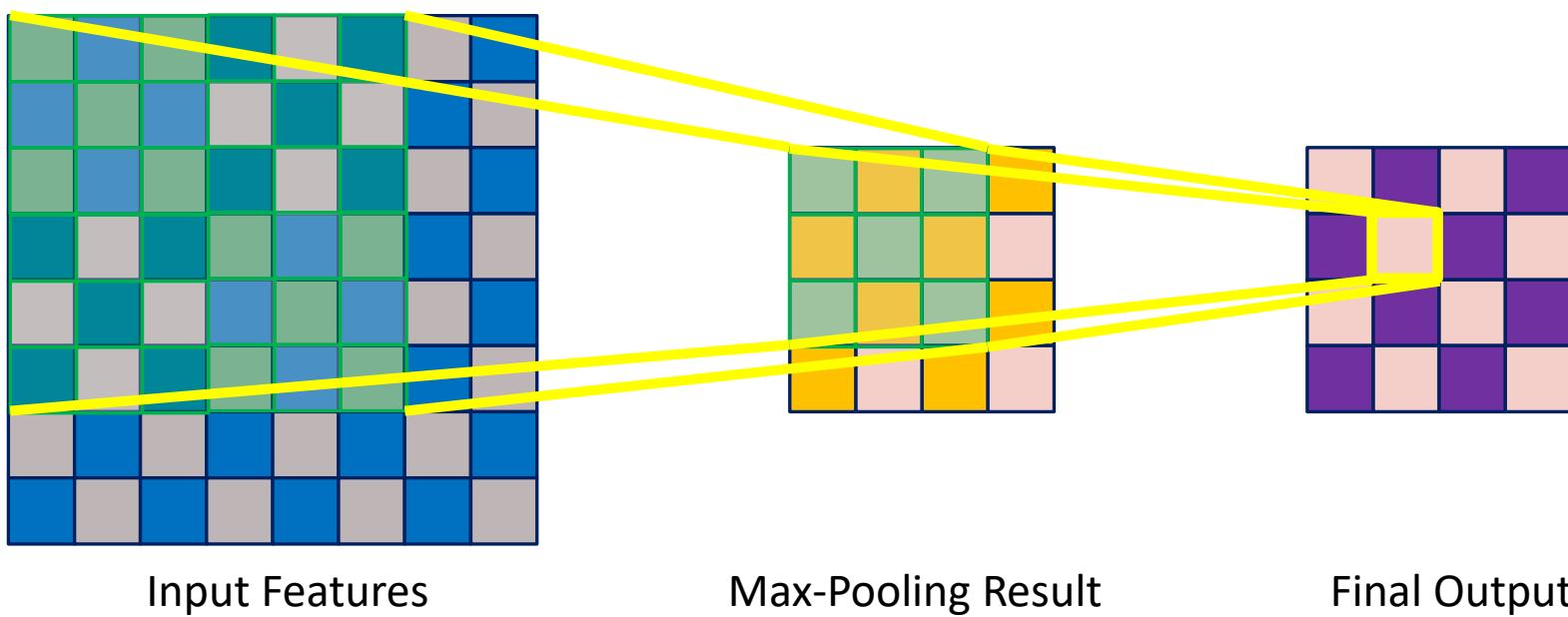
Max-Pooling Result



Final Output

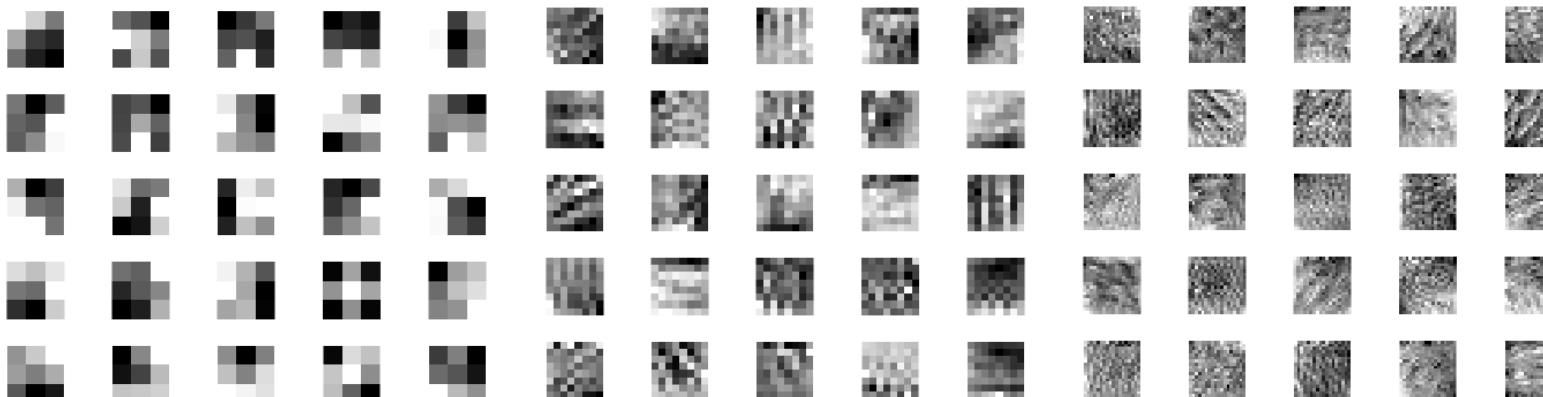
Convolutions with Max-Pooling

- Now consider
 - 3x3 Convolution Layer
 - 2x2 Max-Pooling
 - 3x3 Convolution Layer



3x3 Convolutions For Life?

- Mostly...
 - In, general, multiple layers of smaller filters is better
- But, sometimes...
 - You don't have data to train that many layers
 - Few layers of larger filters may be better given the constraints
 - Your data may have no fine/small details
 - Initial filters at least should be bigger to capture relevant details
 - You may have particular considerations around filter stride, padding, output size, etc.
 - Filter parameters become a function of other constraints



Problems with VGG-Style Nets

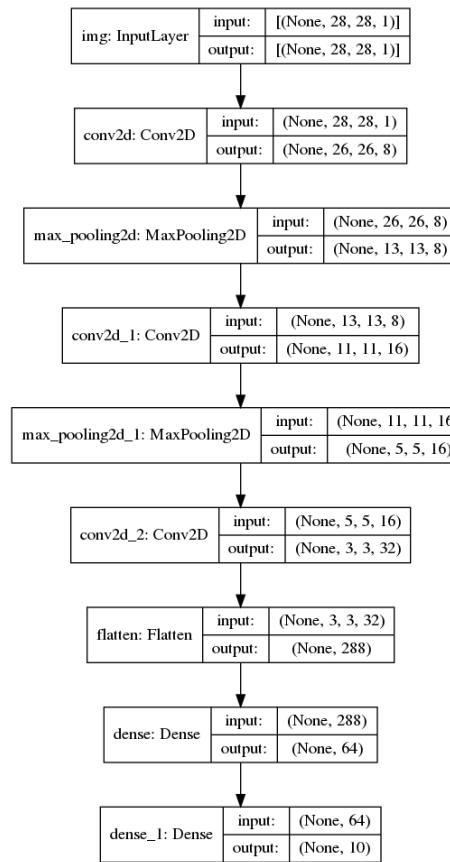
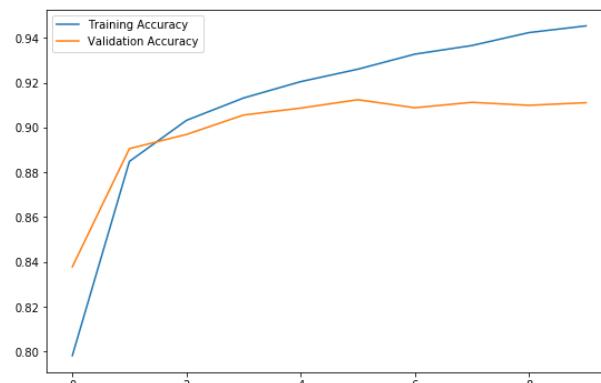
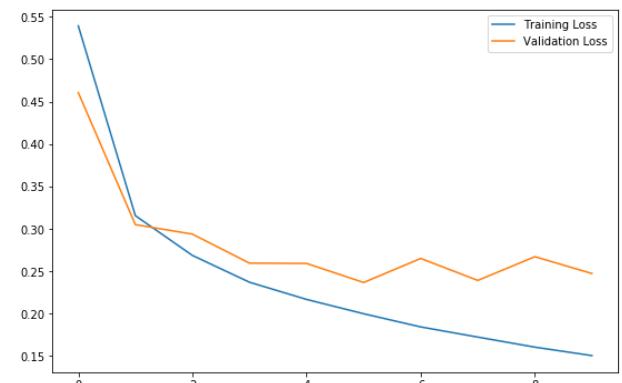
- Depth
 - We want more depth to learn higher level representations
 - Extract more complex concepts as we go deeper
 - But as we go deeper, it's harder to train
 - Data has to go through a lot of layers
 - Changes in early layers impact later layers
 - Training can collapse
 - BatchNorm can help, but won't totally solve the problem
 - Performance tends to max out (and go backwards) somewhere between 20-50 layers
 - Exact point depends on data, etc.

Breaking VGG

- See ***CAB420_DCNNs_Example_4_Breaking_VGG***
- Data
 - Fashion MNIST, of course
- Task
 - Standard classification, though with increasingly deep networks

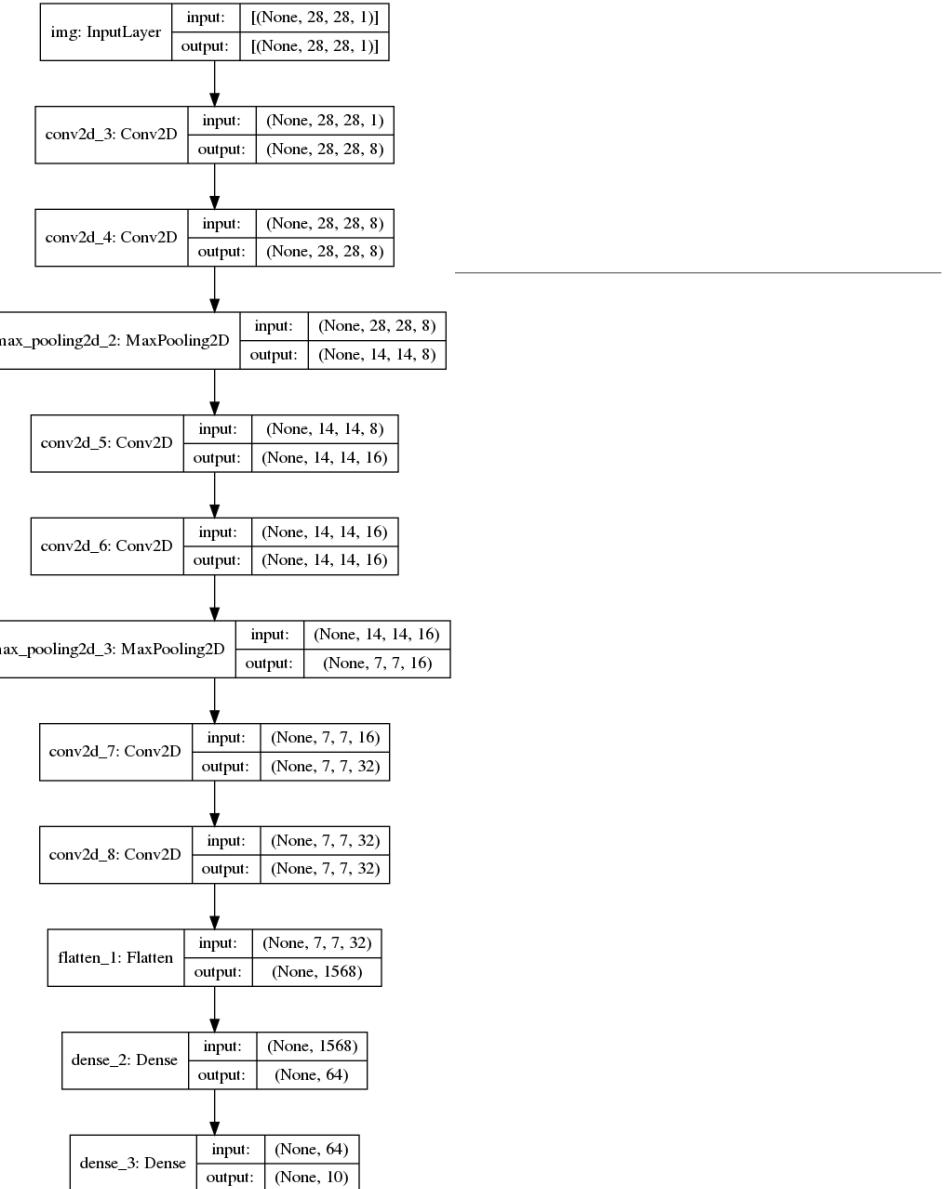
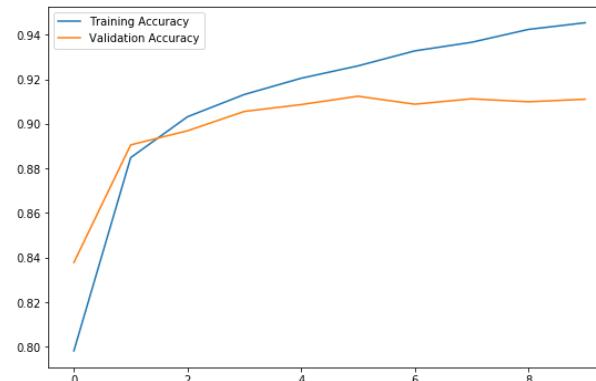
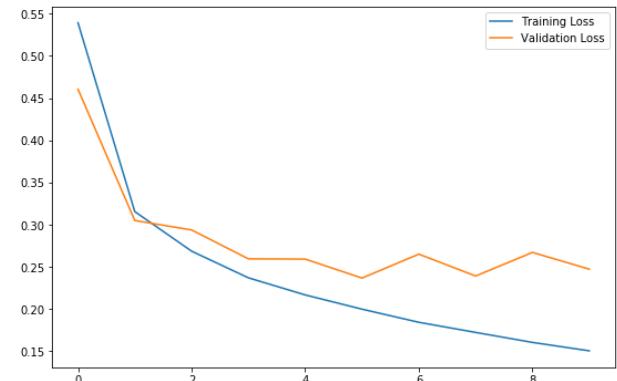
A Normal Network

- 3 Convolutions
 - Max pooling after each
- Model trains and works as expected



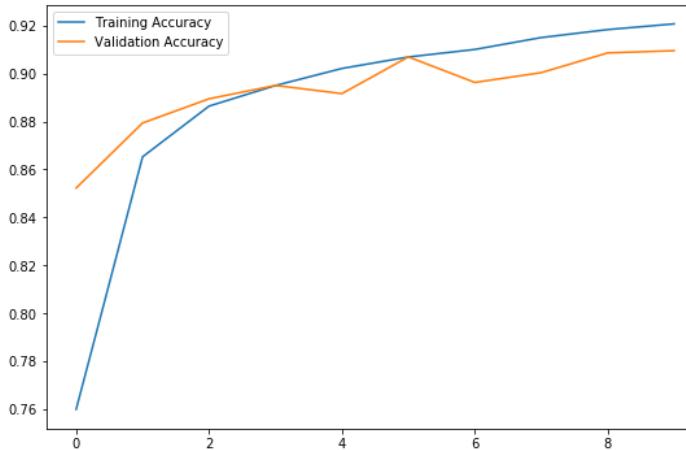
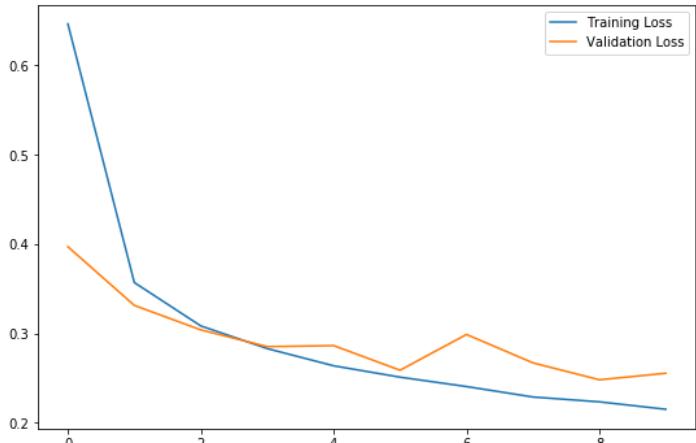
VGG Style Network

- 3 pairs of convolutions
 - Max pooling after each pair
- Model trains and works as expected



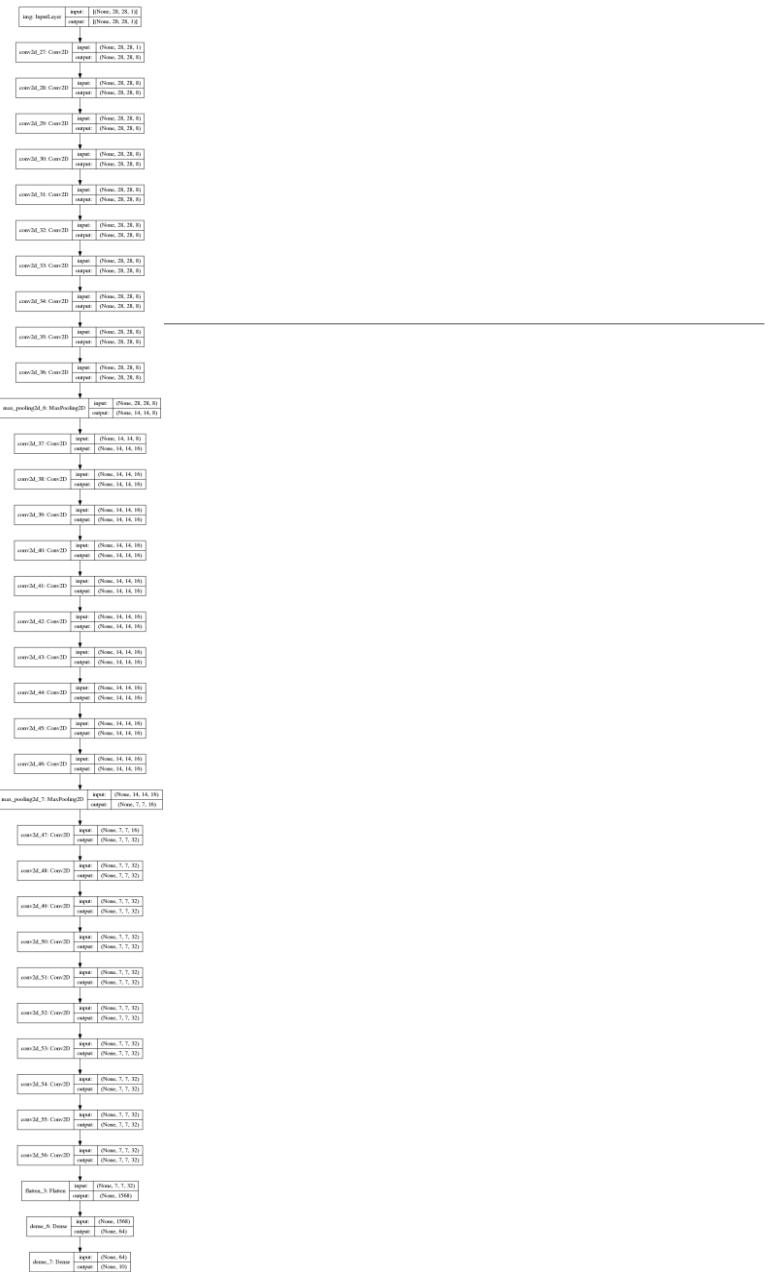
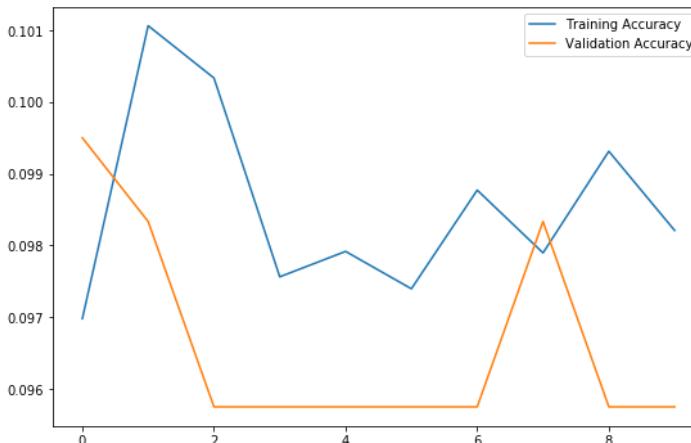
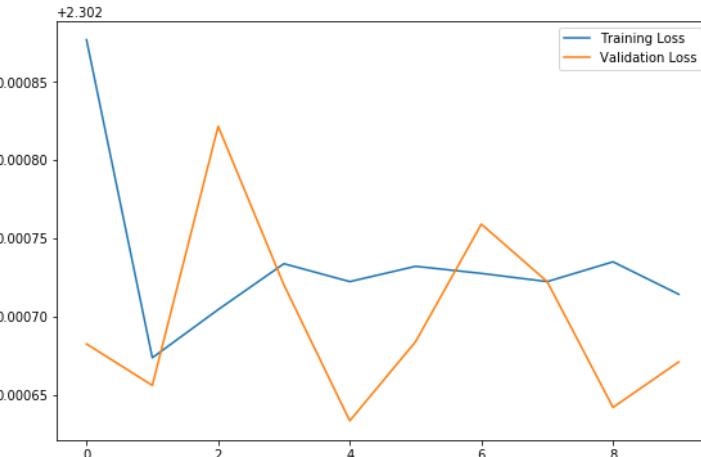
Getting Silly

- 3 sets of 6 stacked convolution layers
 - Max pooling after each set
- Still works
 - Training takes a while though



Getting Very Silly

- 3 sets of 10 convolution layers
 - Max pooling after each set
- Training fails
 - Classifier never gets better than chance



Stop that, it's silly

- Networks train via back-propagation
 - The error at the output is propagated back towards the input
 - If our network gets too deep, the gradients we use to adapt the network vanish
 - i.e. there is not enough information to update any parameters, and thus the network stops learning
- We can increase the viable depth slightly though batch normalisation, but this won't get us too much deeper
 - We need a way to get a more direct connection between the network output and input

CAB420: ResNet

LESS SILLY, MORE DEEP

ResNet

- Residual Networks
 - Introduces the idea of skip connections
 - Makes it easier to push data though the network, in particular for deep networks earlier in training
 - Skip connection may need to adjust the dimensionality in the identity block for the addition to be valid

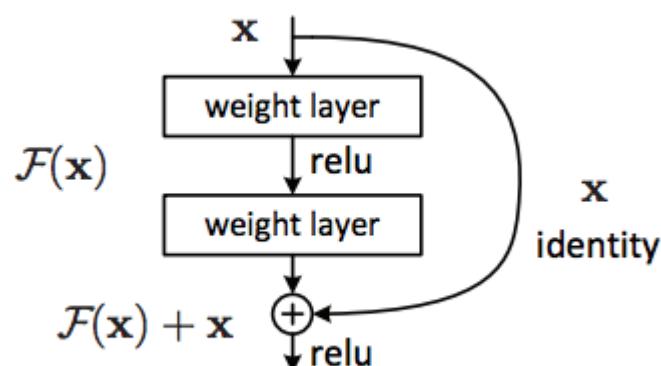
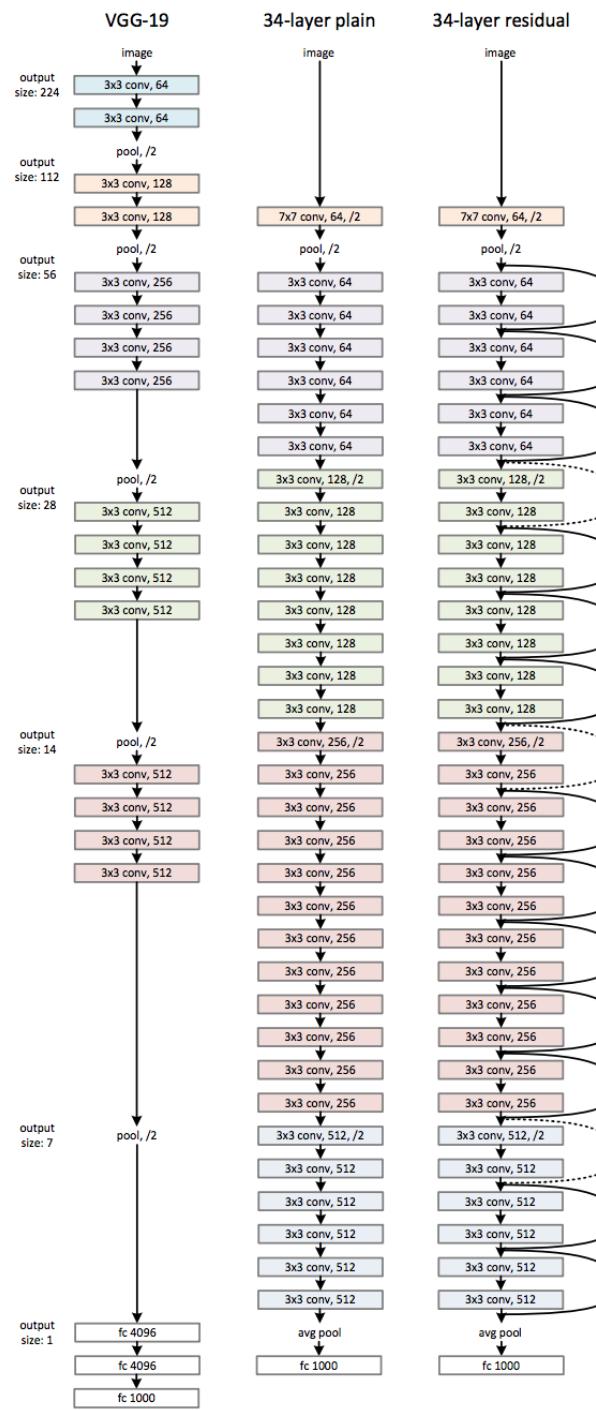


Figure 2. Residual learning: a building block.

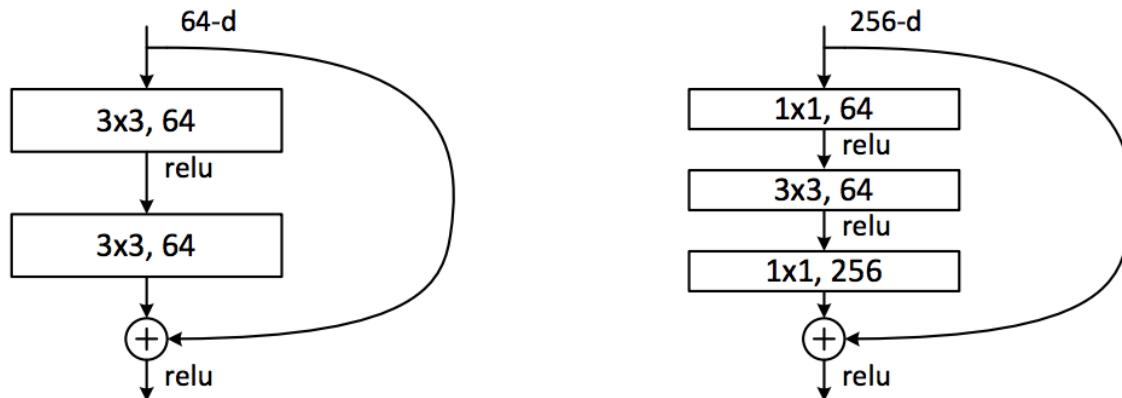
ResNet vs VGG

- Many more convolution layers
 - “Direct” path back to the input
 - Gradients propagate on two paths, the “main” path and the “skip” connections
- Even early in training, early layers can receive meaningful feedback
 - Leads to faster and more stable training



Bottleneck Blocks

- ResNet let's us get some very deep networks
 - But having huge numbers of 3x3 convolutions becomes expensive in terms of memory
- Bottleneck blocks help overcome this by
 - Down sampling via a 1x1 convolution
 - Computing a 3x3 convolution
 - Up sampling with a 1x1 convolution



1x1 Convolutions

- We normally think of convolutions as spatial filters
 - Look for patterns in a local region
- Remember, filters operate across channels
 - For example, we have a [14 x 14 x 8] representation and a [3 x 3] convolution layer
 - Each filter will have 3 x 3 x 8 parameters
 - Each filter will operate over the whole 8 channel volume
 - i.e. patterns across the channels are considered

1x1 Convolution

- Considers patterns across the channels only
 - No consideration of spatial information
- Allows us to
 - Increase the number of channels
 - Decrease the number of channels
- Effectively learns a set of channels that are weighted combinations of existing channels
- Programmatically
 - The same as our regular 2D convolution
 - Just with a kernel of size [1, 1]

1x1 Convolution Visualisation

- See **CAB420_DCNNs_Additional_Example_6_1x1_Convolutions.ipynb**

- We have the network to the right trained on Fashion MNIST

- **1x1 Convolutions:**

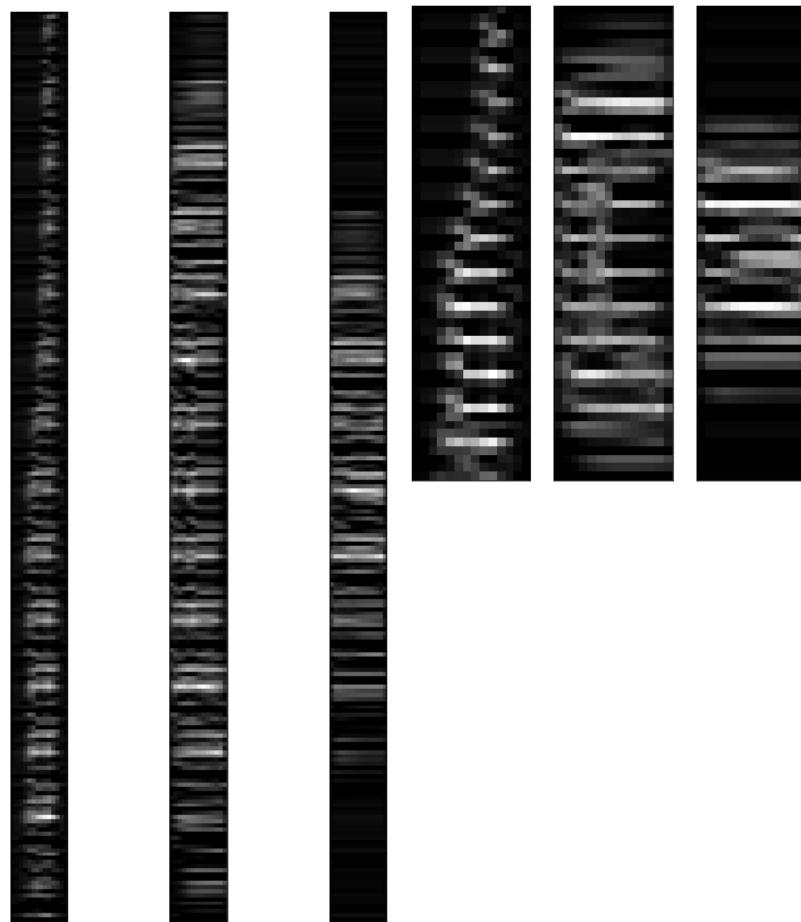
- To decrease channels from 16 to 4 (before1 -> after1)
- To increase channels from 16 to 32 (before2 -> after2)

Note: This example is just to illustrate 1x1 convolutions. This is not a great network design

Layer (type)	Output Shape	Param #
<hr/>		
img (InputLayer)	[(None, 28, 28, 1)]	0
conv2d_15 (Conv2D)	(None, 28, 28, 8)	80
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 8)	0
before1 (Conv2D)	(None, 14, 14, 16)	1168
after1 (Conv2D)	(None, 14, 14, 4)	68
max_pooling2d_9 (MaxPooling2D)	(None, 7, 7, 4)	0
before2 (Conv2D)	(None, 7, 7, 16)	592
after2 (Conv2D)	(None, 7, 7, 32)	544
flatten_4 (Flatten)	(None, 1568)	0
dense_8 (Dense)	(None, 64)	100416
dense_9 (Dense)	(None, 10)	650
<hr/>		
Total params: 103,518		
Trainable params: 103,518		
Non-trainable params: 0		

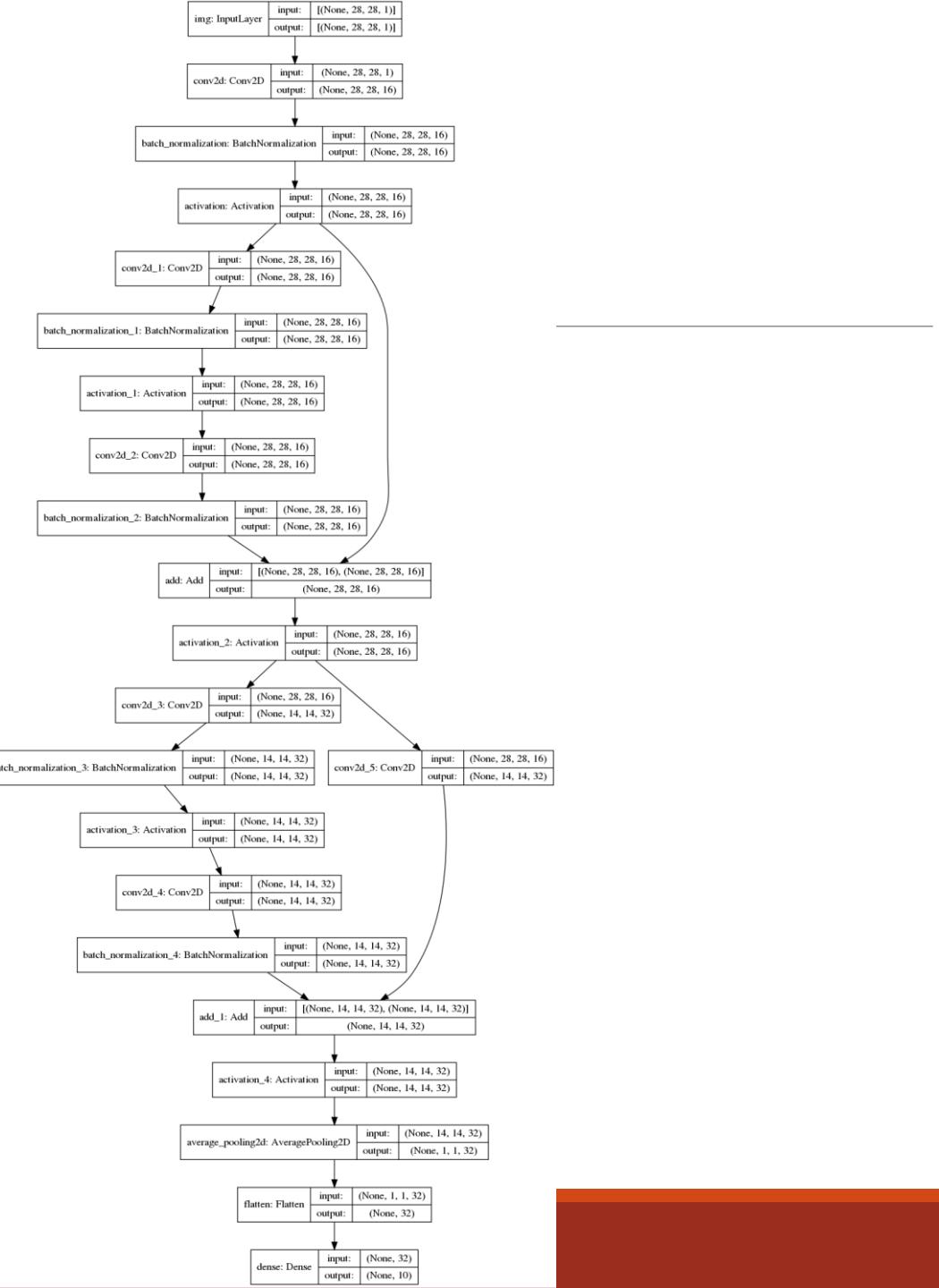
1x1 Convolution Visualisation

- Each column is the unrolled activations from all channels for a single image
 - Left: before 1x1
 - Right: after 1x1
- Right images are compressed versions of the left
 - Same patterns are present, just more compact
- Can use the same approach to upsample
 - See [*CAB420_DCNNs_Additional_Example_6_1x1_Convolutions.ipynb*](#) for an example



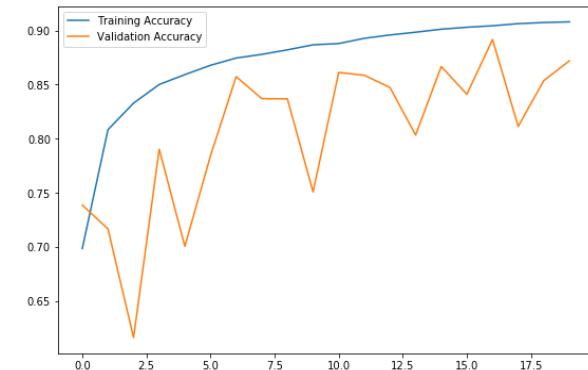
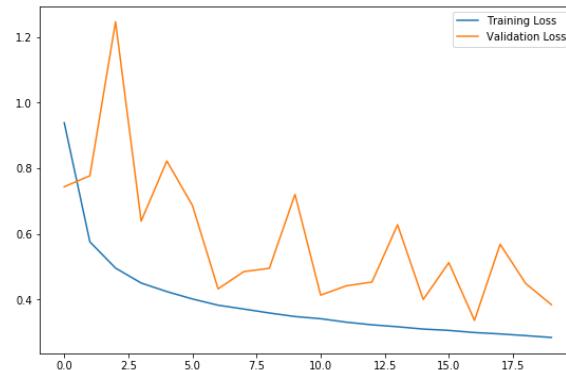
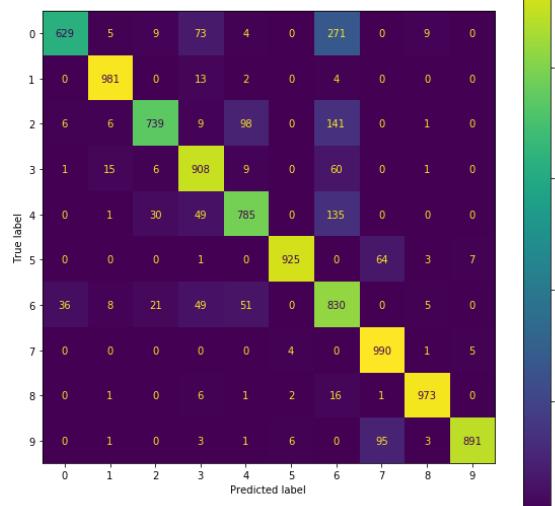
ResNets in Action

- See **CAB420_DCNNs_Example_5_ResNet.ipynb**
 - Example uses function to build ResNets of varying sizes
- Simple ResNet
 - No bottleneck layers
 - ~20,000 params
 - Average pooling greatly reduces size of dense layer



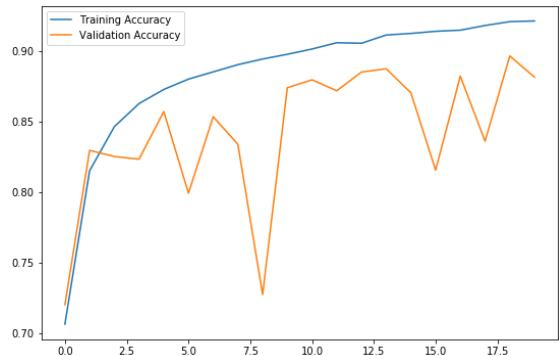
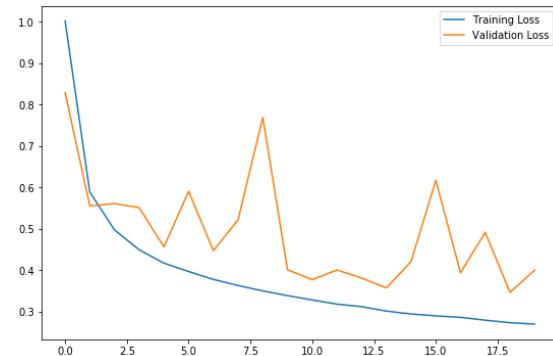
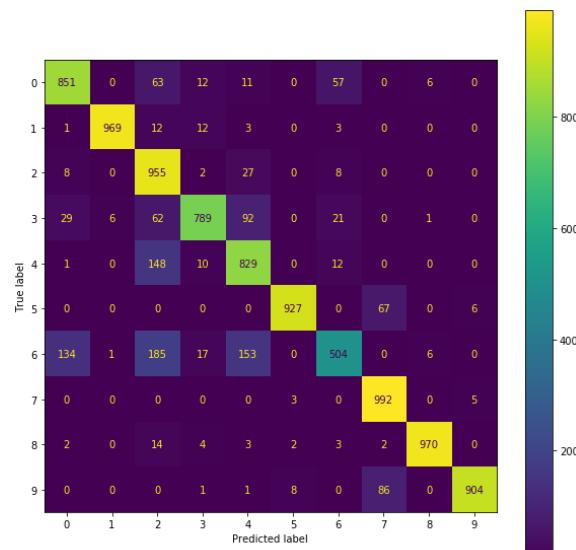
Simple ResNet: Performance

- It's not suddenly magically better
 - ~86% test accuracy on FashionMNIST
 - Could perhaps train a little longer though
- The residual structure helps most with network depth
 - Allows us to go much deeper
 - For complex problems, this is important
 - FashionMNIST is not that complex



ResNet with Bottleneck

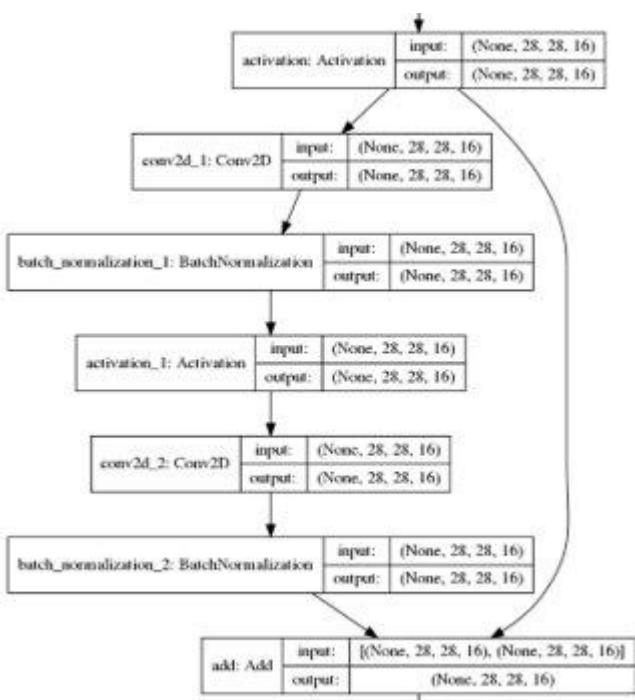
- See [**CAB420_DCNNs_Example_5_ResNet.ipynb**](#)
- This time with bottleneck layers
 - Slightly deeper than our previous network
 - ~24,000 params
 - Performance very similar
 - ~87% test accuracy



Residual Component (pt 1)

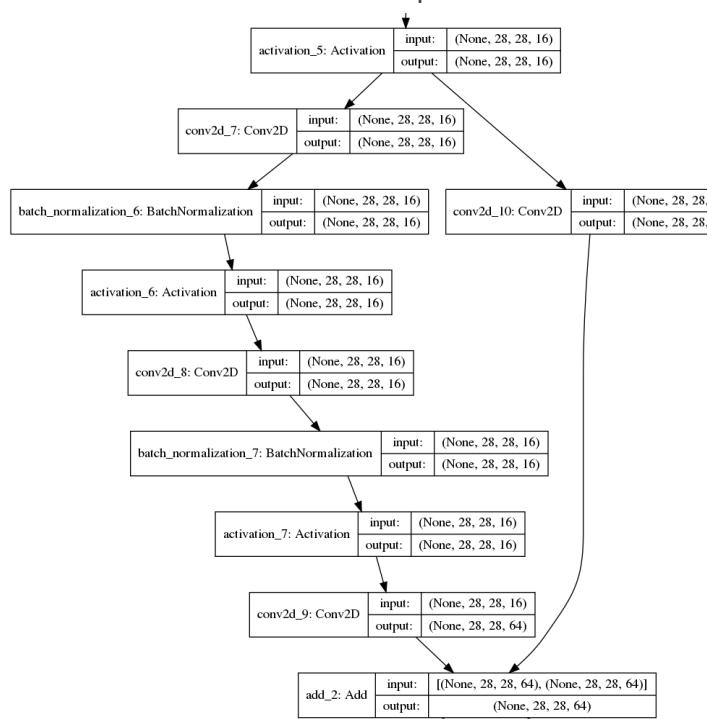
- V1 (no bottleneck)

- Two convolutions in "main" branch
- Nothing in the skip branch



- V2 (with bottleneck)

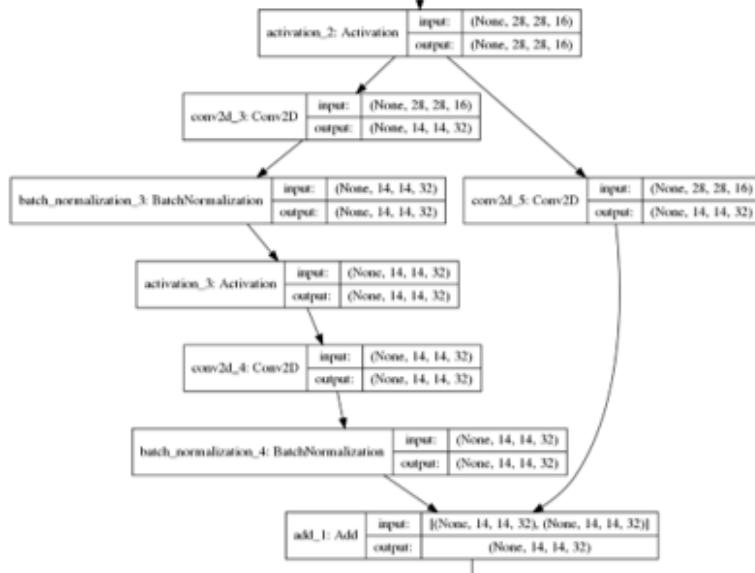
- Three convolutions in "main" branch
- Last one is 1x1, to upsample channels
- One 1x1 convolution in "skip" branch



Residual Component (pt 2)

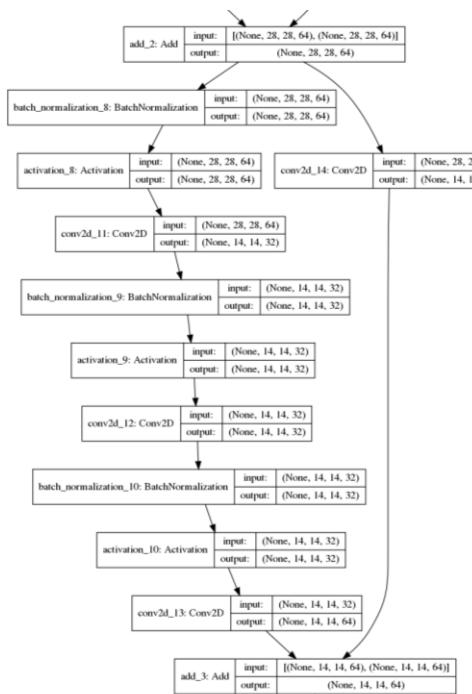
- V1 (no bottleneck)

- Two convolutions in "main" branch
- One convolution in "skip" branch
- Same number of filters (32) in all



- V2 (with bottleneck)

- Different number of filters in both branches
- Larger numbers in filters in places than the V1 network
- 1x1 convolutions at the start and end of the main branch



ResNet V1 vs V2

- V2 networks use bottleneck layers to keep internal representations small
 - Allows deeper networks while keeping memory use manageable
 - Need to have much larger networks to see the full benefit
- Both work well
 - Both improve over VGG and address the issue of vanishing gradients
 - Either one is a good choice
 - Parameters of the network (number of filters, number of residual blocks, depth, etc) more important for performance than which of V1 or V2 you choose

Beyond ResNet

- Several other architectures have been proposed since ResNet
 - ResNeXt
 - DenseNet
 -
- All of these retain the idea of skip connections
 - And often use a lot more of them
- I encourage you to explore these in your own time
 - See <https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035> as a starting point

CAB420: DCNNs with Less Data

BECAUSE SOMETIMES YOU DON'T HAVE THE
TIME TO COLLECT MILLIONS OF SAMPLES

DCNNs and Data

- So far, DCNNs have worked well. But
 - We've had large datasets (50,000 samples)
 - We've had small images (28x28)
- What if we have samples with more dimensions (i.e. bigger images), and fewer samples?
 - Overfitting
 - Training instability
- We need a way to work with smaller datasets. We'll look at two (which can be used together):
 - Fine tuning
 - Data augmentation

Fine Tuning

RECYCLING MODELS

Why?

- So far, we've always trained from scratch
 - i.e. start from random weights, learn everything
- Ideally, we'd always do this, but
 - It requires a lot of data
 - For large models, it requires a lot of time
 - ImageNet models can take weeks to train
- For many tasks, networks learn similar things
- Consider an image recognition task
 - Edges and primitive shapes will always be of interest
 - This is what the early layers learn
 - Why not re-use this?

How?

- Usually when we train a network, we
 - Start from a set of random weights and biases
 - Progressively update those over time
- Instead we can
 - Start from a set of weights learnt on a related task
 - Progressively update those over time
 - We may change some layers
 - Different output type
 - We may set some layers to be fixed and never change
 - i.e. leave early convolution layers as they are

Fine Tuning

- See ***CAB420_DCNNs_Example_6_Fine_Tuning_and_Data_Augmentation.ipynb***
- Basic approach is
 - Load a model from a related domain
 - Change output and other layers if needed. We may do this if:
 - If we're changing from a 10-class classification problem to a 40-class classification problem
 - We're changing from a classification problem to a regression problem
 - Change the input size if needed
 - This may require the removal and re-creation of all dense layers
 - Generally, if you can avoid doing this, it's best to
 - Decide if some layers should have their weights frozen
 - Most likely early convolutional layers
 - Compile and train the network as you normally would

Advantages of Fine-Tuning

- Can train with much less data
 - Does require some commonality between the tasks
 - Usually, “image based” is enough commonality
- Can train much quicker
 - Given we already have a good initialisation, convergence is much faster
- Can use much more complex than otherwise possible
 - It’s very hard to train ResNet-152 yourself
 - But you can fine-tune it

Data Augmentation

MAKING THINGS UP, SORT OF

Data and DCNNs

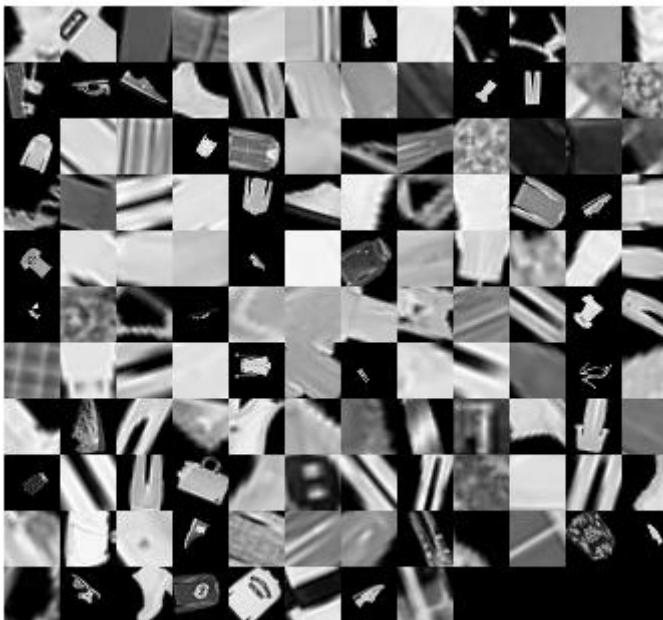
- DCNNs need lots of data. Sometimes this is hard
 - Data capture can be expensive
 - Annotation is also expensive
 - Or very boring
- Data variety is key to avoiding overfitting
- Yet often
 - Different data samples can appear very similar

Data Augmentation

- Creates a “new” dataset by applying simple transforms to what data you have
- Simple transforms may include:
 - Scale changes
 - Rotations
 - Horizontal and/or vertical flips
 - Adding noise or small colour shifts
 - Translations

Using Data Augmentation

- Consider your data and what changes make sense
- On the right, we have augmented Fashion MNIST with
 - Vertical and Horizontal flips
 - Large rotations
 - Large scale changes
- This has gone too far



Using Data Augmentation

- Fashion MNIST, take 2:
 - Horizontal flips
 - Small rotations
 - Small translations
- Data looks subtly different from the original
 - Still recognisable as being from the same domain



Using Data Augmentation

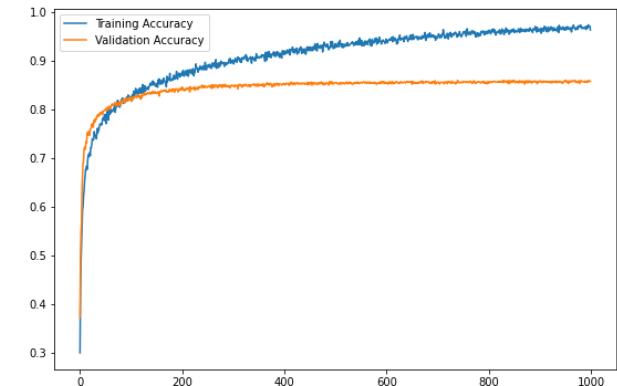
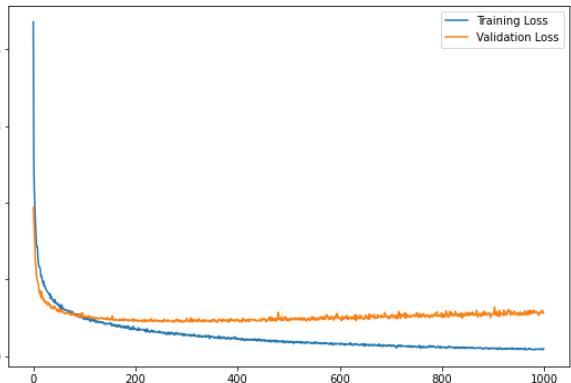
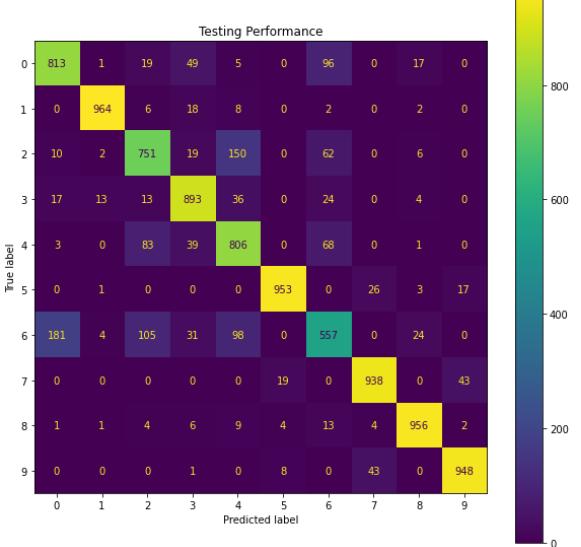
- Less can be more
 - Augmentation should not change the meaning of an image
 - Characters for example may change meaning if flipped
 - Consider n vs u
- Inspect the augmentation results before proceeding
 - If they make sense to you, that's a good start
 - If they've been changed so much that you can't tell things apart, you've gone to far

An Example

- From ***CAB420_DCNNs_Example_6_Fine_Tuning_and_Data_Augmentation.ipynb***
- Our task
 - Fine tune a simple VGG network, trained on MNIST, on FashionMNIST
 - All layers trained, no layers removed/changed
 - Datasets are the same size images, same number of classes, etc.
- The catch
 - We're going to remove 95% of the data
 - 3000 samples, total
- Compare
 - Fine-tuned network with no data augmentation
 - Fine-tuned network with augmentation

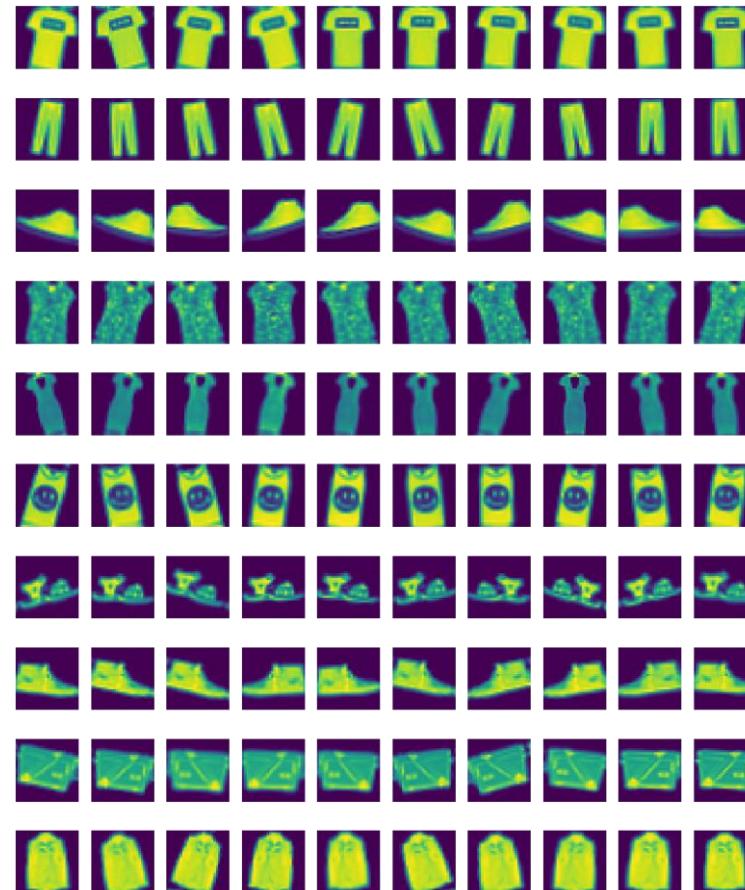
No Augmentation

- Overfit central
 - Very quickly we see validation performance flatline and the model begins to overfit
- Trained for 1000 epochs
 - In part because we have such a small dataset



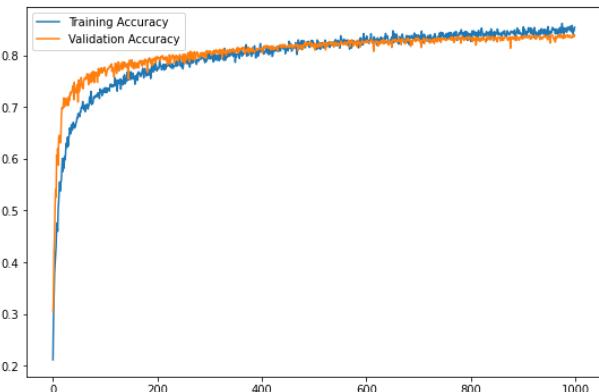
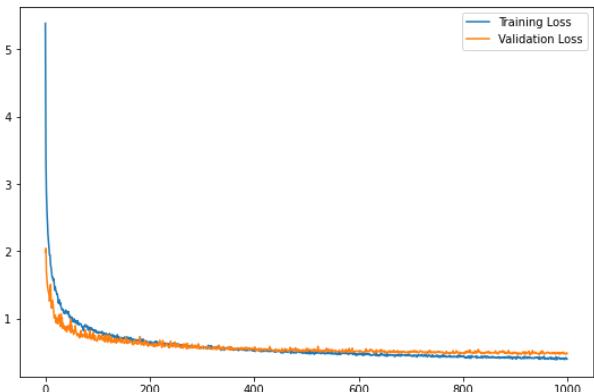
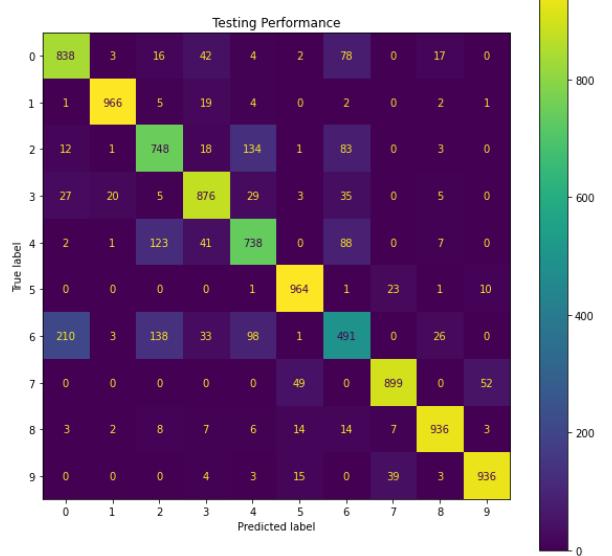
With Augmentation

- Augmentation using
 - Rotations
 - Zoom
 - Translation
 - Horizontal Flip
- All samples still recognisable as their true class



With Augmentation

- Solid performance
- Validation data is not augmented
 - Thus, we see validation data working better for a long time
- Could keep training for longer still at this point
 - Model not fully converged



Things to Tune

SOME MODELS

Models for Fine-Tuning

- Two scripts are on blackboard for both python and matlab
 - *CAB420_DCNN_Models_Additional_Script_Lots_of_ResNet_Models.ipynb*
 - *CAB420_DCNN_Models_Additional_Script_Lots_of_VGG_Like_Models.ipynb*
 - These train a bunch of models of different sizes on
 - MNIST
 - FashionMNIST
 - CIFAR-10
 - Exported models are on blackboard too
 - So you don't have to train them yourselves
- You can use these models as a basis for fine-tuning
 - Select a model based on size, source data, etc
 - Play with different models as a base, see what happens

Models for Fine-Tuning

- But what if I want something else to fine-tune?
 - Send an email to cab420query@qut.edu.au
 - If I can, (and I think the request is reasonable) I'll train it and post it

CAB420: DCNNs vs Everything Else

FIGHT!

DCNNs vs Other Classifiers

WHO WINS?

Comparing Classifiers

- DCNNs work best with specific data types
 - Images, Audio, other signals
 - Not well suited to tabular data
- DCNNs also need large amounts of data
 - Though this can be reduced through fine-tuning and augmentation
- We'll compare SVMs, CKNNs, Random Forests and DCNNs using Fashion MNIST
 - Vectorise data for SVMs, CKNNs and Random Forest
 - Compare performance, training time, and evaluation time using different sized datasets
 - We're using non-optimal feature extraction for non-DL methods
 - Our DCNN is very simple, and also not optimal
 - Evaluation will still show broad trends with respect to classifiers however
 - See ***CAB420_DCNNs_Additional_Example_7_Runtimes.ipynb***

Training Runtime

- DCNN and Random Forest increase linearly with training dataset size
 - DCNN training for a fixed number of epochs
- SVM increase exponentially
 - Does not scale well to large dataset sizes
- CKNN very efficient
 - No real learning performed



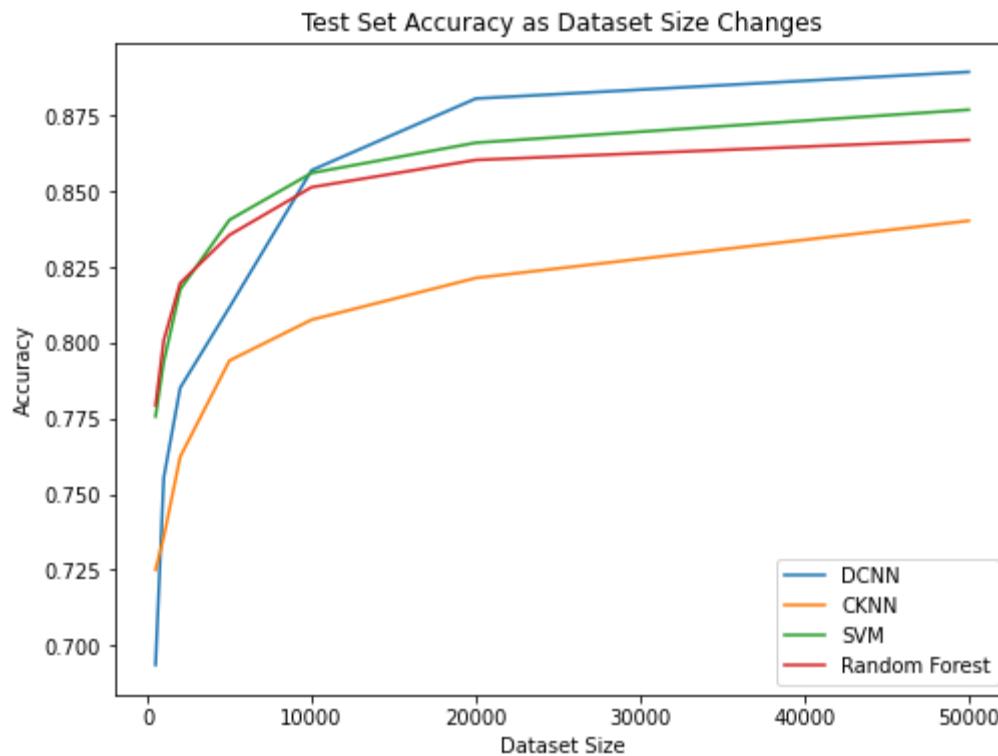
Testing Runtime

- Time to evaluate 10,000 test samples
- DCNN and Random Forest don't change
 - Evaluation time dependant on network complexity (DCNN); number and depth of trees (Random Forest)
- CKNN and SVM get more expensive as training set size increases
 - More points to search/compare to



Accuracy

- Once datasets size reaches 10,000 samples, DCNN wins
- For all classifiers, performance gains slow with increasing dataset size



DCNNs vs the Human Brain

BRAAAAINS!

Biological Motivations

- Neural networks as a whole are directly inspired by the brain, and various components also take inspiration from nature
 - Convolutional networks are inspired by the visual cortex
 - Mechanisms have been developed to model attention and memory
- While these are inspired by nature, they are not mimicking nature
 - We don't know enough about the brain (human or otherwise) to simulate one
 - Often we seek to mimic the output of a process (as we understand it), rather than the underlying mechanism itself

Neural Network Properties

- Neural networks have become increasingly large in recent years
 - Recent networks for image processing can have 100's of millions of parameters
 - GPT-3 (a natural language model) has about 175 billion parameters
- Neural networks generally have simple topologies
 - Networks may have simple branches, skip connections, etc.
 - Networks can use recurrent structures to simulate loops, feedback processes, etc.
- Networks propagate data through a known path
 - Data arrives at the input, propagates through the network, arrives at the output
- Limited fault tolerance, or ability to extrapolate beyond training bounds
- Networks learnt through gradient descent and back propagation
 - Typically trained once, and then used for inference with no on-going updates
 - Though there is work in this area to allow continuous learning

Properties of the Brain

- The human brain is estimated to have about 86 billion neurons
 - But over 100 trillion synapse (connections)
 - High level of interconnectedness, complex topologies
 - Neurons can fire asynchronously
- Biological networks are (to a point) fault tolerant
 - Both through redundancy, and an ability to heal
- Learning is not well understood
 - Learning is continuous
 - Neural plasticity allows connections to change over time
 - Learning is robust to rare or unseen examples

Summary

BECAUSE I TEND TO RAMBLE

Neural Networks

- Now state of the art for most machine learning tasks, but
 - Require lots of data
 - Or lots of tricks to work with limited data
 - Can be very resource intensive to train
- In general
 - Deep networks lead to greater representative power
 - But at very high depth training becomes difficult and architectural tweaks are needed

Learning with Neural Networks

- Classification
 - Achieved via a “softmax” output
 - Attempts to create a “one-hot vector”, i.e. a vector where one element is 1 and the rest are 0
- Regression
 - Very much like classification
 - Just change the output and the loss function
- Network architectures are flexible
 - Almost identical networks were used for
 - Classification of pieces of clothing
 - Estimation of how much a digit had been rotated by
 - But the networks will have learned very different thing

Making Networks Better

- Can use various layers to improve model fitting
 - Dropout
 - Though be careful when applying to convolutional layers
 - BatchNorm
 - Makes training easier by ensuring that values in the middle of the network are in a known range
 - Weight Regularisation
 - Implementation varies according to platform
 - Explore demo example
 - *CAB420_DCNNs_Additional_Example_4_Layer_Order_and_Overfitting.ipynb*

ResNet

- Simple feed forward models can only get us so far
 - More depth makes things harder to train
 - Ultimately, adding layers makes things worse
- ResNet uses skip connections to overcome this
 - Allows multiple paths through a network
 - Helps gradients propagate to the early layers
 - Improves training speed
- Skip connections are a standard features of many current architectures
 - DenseNet
 - U-Net
 - And many, many, more

Making do with Less Data

- Fine Tuning
 - Take an existing network, and adapt it
 - Can be seen as a type of transfer learning
- Data Augmentation
 - Create additional data by applying simple transforms to what you have
 - Don't go overboard
 - Look at what your augmentations do to the data
- Both techniques can make use of DCNNs with small datasets possible

CAB420: Ethics and ML

WITH GREAT POWER, COMES GREAT RESPONSIBILITY

Machine Learning in Society

- We are seeing rapid adoption of ML in society for several applications
- Some of these applications are fairly innocuous
 - Predictive text when sending messages
 - Advertising recommendations
- Some, are a bit more serious
 - Border control
 - Job application pre-screening
 - Surveillance and monitoring technologies
- For many we don't quite know what the impacts are
 - Chat-GPT, DALL-E
- Consider the implications of these systems
 - All can be used “for good”
 - All can have negative impacts, though some more severe than others

Machine Learning in Society

- What makes a machine learning system suitable for use?
 - What level of accuracy is needed?
 - Does the level of accuracy change depend on the application?
 - If so, how?
 - What happens when a machine learning system “gets it wrong”?
 - Who’s at fault when a system makes an error?
 - Is the system biased?
 - Should the impacts of a system be considered before it’s released upon the world?
- At present, there is
 - No single set of standards or guidelines to help with the above questions
 - Little to no external oversight regarding any deployed ML model
 - Little, if any, consideration to the impacts of systems on others

Machine Learning and Errors

- Errors can be caused by many factors
 - Issues with training data
 - Not enough diversity, incorrect labels, too little data, too little curation
 - Issues with model design
 - Too simplistic, too complex, invalid assumptions
 - There may also be genuine outliers that lead to an error
 - Or is this just another issue with a lack of diversity?
 - Can an ML system ever cover all possibilities?

Dealing with Errors in Practice

- Ideally, we'd review decisions made by an ML system (human in the loop), but
 - Is this practical?
 - Will the human be more accurate?
- Could only “unsure” samples be reviewed?
 - Reporting model confidence is not straight forward
 - Particularly for deep learning
 - Model decisions can still be confident and be totally wrong
- Mechanisms to deal with incorrect decisions are limited, if present at all
- Within chat/text systems, “blacklisting” is often used
 - Simple rules to avoid certain topics, but...
 - Siri blacklisted topics for healthcare, yet “she could tell you where to hide a body”
 - Chat-GPTs blacklisted topics can be bypassed quite easily
 - <https://www.lesswrong.com/posts/7fYxxtZqjuYXhBA2D/testing-ways-to-bypass-chatgpt-safety-features>

ML and Bias

- Bias is a large concern in ML models that are deployed
- Data Bias
 - Caused by using limited or flawed data in a model
 - Very hard to avoid totally, all data sets are a sample
 - Amazon's hiring tool that was biased against women (<https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scaps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G>)
 - Hate speech detection biased against black people (<https://futurism.com/the-byte/google-hate-speech-ai-biased>)
- Technical and contextual biases
 - Design choices/flaws that disadvantage one group over another
 - Omitting information that provides context for the data and decision

ML and Data

- ML depends on data, but data needs to be sourced and used ethically
- Ever, a cloud storage photo app, illegally used user photos to train face recognition models, and was ordered to delete the resultant models
 - <https://techcrunch.com/2021/01/12/ftc-settlement-with-ever-orders-data-and-ais-deleted-after-facial-recognition-pivot/>
- Duke MTMC dataset, a large multi-target multi-camera object tracking and person re-id dataset, was heavily used by the Chinese military and Chinese surveillance companies, and was subsequently removed.
 - https://exposing.ai/duke_mtmc/
- Training on data collected and annotated in an automated way is common
 - GPT-3 is trained by data pulled from the internet
 - Including copywritten material, and posts on Reddit
 - DALL-E is trained in a similar way, but with images
 - 80 Million Tiny Images dataset uses labels automatically generated from WordNet
 - Contains harmful, racist and derogatory terms as labels
 - <https://openreview.net/pdf?id=s-e2zaAIG3I>
- Within academia, it is common to mine social media for data
 - Youtube, flickr, twitter, etc.
 - Commonly seen as “fair use”, though sharing such data repositories is murkier

ML and Ethics

- Increasing awareness of ethical issues relating to ML
 - More coverage in academic literature, including dedicated events (<https://dl.acm.org/doi/proceedings/10.1145/3531146>)
- Companies are beginning to enact their own policies
 - Microsoft (<https://www.microsoft.com/en-us/ai/responsible-ai?activetab=pivot1%3aprimaryr6>)
- But often only when convenient
 - Google firing their Ethics team leader after they pointed out various issues with machine learning systems
 - <https://www.theverge.com/22309962/timnit-gebru-google-harassment-campaign-jeff-dean>
- Several guidelines/principals for ethical use of AI have been developed by governments and public bodies in the past couple of years
 - Australian Government (<https://www.industry.gov.au/data-and-publications/building-australias-artificial-intelligence-capability/ai-ethics-framework/ai-ethics-principles>)
 - Institute for Ethical AI and Machine Learning (<https://ethical.institute/principles.html>)

AI Ethics Principals (Australian Government)

- Human, social and environmental wellbeing
 - Throughout their lifecycle, AI systems should benefit individuals, society and the environment.
- Human-centred values
 - Throughout their lifecycle, AI systems should respect human rights, diversity, and the autonomy of individuals.
- Fairness
 - Throughout their lifecycle, AI systems should be inclusive and accessible, and should not involve or result in unfair discrimination against individuals, communities or groups.
- Privacy protection and security
 - Throughout their lifecycle, AI systems should respect and uphold privacy rights and data protection, and ensure the security of data.

AI Ethics Principals (Australian Government)

- Reliability and safety
 - Throughout their lifecycle, AI systems should reliably operate in accordance with their intended purpose.
- Transparency and explainability
 - There should be transparency and responsible disclosure to ensure people know when they are being significantly impacted by an AI system, and can find out when an AI system is engaging with them.
- Contestability
 - When an AI system significantly impacts a person, community, group or environment, there should be a timely process to allow people to challenge the use or output of the AI system.
- Accountability
 - Those responsible for the different phases of the AI system lifecycle should be identifiable and accountable for the outcomes of the AI systems, and human oversight of AI systems should be enabled.

Related AI/ML Ethics Issues

- There are other questions around ethics and AI/ML
- Rapidly emerging issues
 - Intellectual property and copywrite challenges
 - Training models on copywrite data, and to mimic the artistic style of others
 - Job loss and wealth inequality
 - What happens when people's jobs are automated?
 - Military applications of AI/ML
 - Ongoing rollout of surveillance systems
- Potential Future issues
 - The singularity and maintaining control over AIs
 - How should we treat AIs? What rights would they have?

Ethics and ML/AI

- Ethical and equitable development and deployment of ML systems is challenging and complex issue, and is **not just a tech problem**
- Greater consideration is needed about how we deploy and use models, and their broader implications
- Eliminating data bias totally may never be possible
 - But we can be clear about limitations
- We can strive to include uncertainty estimates in models
 - Though this is on-going research, and we're not there yet
- We can provide a “human in the loop” to help review and improve models
- Ethics in AI/ML is an evolving discussion
 - If you are keen to continue in ML, I'd encourage you to read further about these issues

CAB420: Regression Summary

WHAT WAS REGRESSION AGAIN?

Regression

- Predicting a continuous output from one or more inputs
 - There are methods to deal with non-continuous outputs, but these are outside of CAB420’s scope
- We’ve considered
 - Linear Regression
 - Regularised Linear Regression
 - Ridge (L2) Regularisation
 - LASSO (L1) Regularisation
 - Neural Networks

Linear Regression

- Learning a “line of best fit”
- Models a linear relationship between the inputs (predictors) and output (response)

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_j x_j$$

- To find the “line of best fit”, we seek to minimise the sum of squared differences between the actual points and the predicted points (least squares regression)

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2$$

- Several assumptions
 - Samples are independent
 - This may not be the case if we have time-series data
 - Predictors are independent
 - Often, we’ll have some correlation between predictors and violate this assumption
 - Residuals follow a Gaussian distribution
 - Can test by looking at a histogram of residuals, or a qq-plot
- Ideally, we have more samples than predictors

Regularised Regression

- Adds a penalty (regularisation) term to our objective function

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda P$$

- The form of the penalty term determines the type of regularisation
- L2 penalty is Ridge regression

- Penalty term is the sum of the model coefficients squared

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_2$$

- L1 penalty is LASSO regression
- Penalty term is the sum of the absolute values of the model coefficients

$$\sum_{i=1}^n \left(y_i - \sum_j x_{ij} \beta_j \right)^2 + \lambda \sum_{j=1}^p \|\beta_j\|_1$$

Regularised Regression

- Regularisation imposes additional constraints on the model
 - Constraints are not based on accuracy, but complexity
 - Constraints have a weight, λ , that we need to set
- Constraints encourage smaller model coefficients
 - Smaller coefficients mean that the model is less sensitive to small changes in the input
 - Can help curb overfitting, but will reduce accuracy on the training set
- Use the validation set to find the optimal value of λ
 - Train the model on the training set
 - Evaluate the model on the validation set
 - Find the value of λ that leads to best accuracy on the validation set
- Ridge and LASSO will both lead to smaller coefficients as λ increases, but
 - Ridge will bring coefficients increasingly close to 0, but they will never reach 0
 - LASSO will eliminate terms (set coefficients to 0), and produce a constant model (all terms 0) once a sufficiently large λ is reached
- Standardisation often used with regularised regression
 - Helps avoid issues caused by scale variation in the data
 - Eliminates the constant term, β_0 , from the model

Regression with Deep Neural Networks

- Deep networks can be used for regression
- For regression, the network should
 - Use an output activation appropriate for the task
 - If the regression output is unbounded (i.e. can range from $[-\infty \dots +\infty]$), then no activation is appropriate
 - If it output is positive (i.e. $[0 \dots \infty]$), then a ReLu makes sense
 - Use an appropriate loss
 - Mean Squared Error is the sum of squared difference, the same thing that is minimised in regular least-squares regression
 - Mean Absolute Error may also be appropriate depending on the problem

An Example

- See ***CAB420_Summary_1_Regression.ipynb***
- Two regression tasks
 - Predict the price of diamonds given some properties of the diamonds
 - Predict the year a song was released given some properties of the songs
- For each task, apply
 - Linear regression
 - Ridge and LASSO regularised regression
 - A Deep Neural Network
 - Not a DCNN, so no convolutions
 - Data is tabular data, no spatial relationships for convolution to exploit
- Review this example in your own time
 - Ask questions in class or via email/slack

CAB420: Classification Summary

SOMETHING ABOUT CLASSES?

Classification

- Assigning an input to one of a predefined set of categories (or classes)
 - All classes need to be known in advance, and be present during training
 - Classifiers will tend to favour classes for which there is more data
- We've considered
 - Support Vector Machines (SVMs)
 - K Nearest Neighbours Classifiers (CKNN)
 - Random Forests
 - Deep Neural Networks

Non-Deep Learning Methods

OLD SCHOOL MACHINE LEARNING

Support Vector Machines

- Binary Classifier
- Learns a decision boundary between two **linearly separable** classes
 - But, if you need machine learning, it's probably not linearly separable
- So, we can
 - Relax our constraints via a slack variable
 - Map to a higher dimensional space where things might be linearly separable via kernels
 - Care must be taken with kernel selection and parameters
 - Or do both
- In general
 - Good with high-dimensional and/or sparse data
 - Can be difficult to train with large (10,000 +) datasets
 - Binary in nature (though model ensembles can enable use on multi-class problems)

Support Vector Machines Parameters

- C (sometimes referred to as the box-constraint), which essentially provides regularisation
 - C = infinite: hard margin, needs to be clear space between the classes
 - If C is too big (or infinite), model may fail to converge
 - If C is too small, the model can underfit (learn a poor decision boundary)
- The kernel (which has it's own parameters), which can be used to project the data into a higher dimensional space where it may be easier to separate
 - Linear: default, good first choice
 - RBF: can fit to complex distributions, gamma used to tune
 - Polynomial: controlled by the degree (or order) of the line, can be hard to tune
- Can use class weights to deal with class imbalance
 - Weight incorrect decisions in inverse proportion to the number of samples

K-Nearest Neighbours Classification (CKNN)

- K-Nearest Neighbours
 - Classification based on finding similar points
 - Sensitive to dataset size, and the number of neighbours
 - Compared to SVMs
 - Can more easily capture non-linear relationships via use of neighbours
 - Can be sensitive to outliers via use of neighbours
 - Extends to multi-class classification trivially
 - No actual learning
 - Decision boundary is based directly off provided input points

K-Nearest Neighbours Classification (CKNN) Parameters

- K, the number of neighbours
 - Bigger values of K will learn smoother boundaries, be less sensitive to noise, but will make rare classes harder to classify
- The distance metric
 - CKNN relies on finding the closest N points, changing the distance metric changes what we define as “close”
 - Euclidean is a good default if in doubt
- Distance weighting scheme
 - Default is uniform, all K neighbours are equal
 - Can use an inverse scheme, where closer neighbours are given a higher weight relative to their proximity
 - Inverse distance weighting can help when using a large K with rare classes present

Random Forests

- Ensemble of decision trees
 - Relies on classifiers being uncorrelated
 - Each classifier uses a random selection of training points
 - Each branch in each tree uses a random set of features
 - Inherently multi-class
 - Typically, very efficient
- Depth of tree leads to more accurate decisions on training data
 - If trees get too deep, can lead to overfitting
- Class weights can be included in the same manner as an SVM
- Can obtain a likelihood from results across all trees
 - i.e. 50% of trees said class 1, 25% said class 2, etc

Random Forest Hyperparameters

- Tree depth
 - Deeper trees allow for more fine-grained decisions, but increase overfitting
 - Generally, more classes and/or more dimensions will require deeper trees
- Number of trees
 - More trees means a larger cohort of models that are averaged
 - Small numbers of trees will lead to a greater sensitivity to noise
 - Run-time and memory will increase linearly with the number of trees, huge forests can become impractical

Selecting Hyper Parameters

- Changing hyper parameters can lead to a big change in performance
- If in doubt
 - Start with default values (they're defaults for a reason)
 - Use a grid-search
 - If run-time is a consideration
 - Possibly start with a coarse grid search, then refine around that
 - Run small scale tests to explore performance
 - Perhaps select 20% of the data, train some models, use this to inform decisions
 - Consider what performance metric you should be optimising for
 - F1? Accuracy?

Deep Learning

A SOMEWHAT POPULAR MACHINE LEARNING APPROACH

Deep Neural Networks

- Now state of the art for most machine learning tasks, but
 - Require lots of data
 - Or lots of tricks to work with limited data
 - Can be very resource intensive to train
- In general
 - Deep networks lead to greater representative power
 - But at very high depth training becomes difficult and architectural tweaks are needed
- Network architectures are flexible
 - The same network structure can be used for multiple problems
 - Tweak the input or output shapes, losses, etc as needed

Learning with Neural Networks

- Classification with deep neural networks
 - For a multi-class classification task
 - Output layer with N neurons, where N is the number of classes
 - Softmax activation, to emphasise 1 output above all others
 - Categorical cross entropy loss
 - For a binary classification task
 - Output layer with 1 neuron
 - Sigmoid activation, to push output towards either 0 or 1
 - Binary cross entropy loss
- Can use class weights to counter issues of class imbalance

Making Networks Better

- Overfitting can be an issue
- To address overfitting we may consider
 - Dropout
 - Though be careful when applying to convolutional layers
 - BatchNorm
 - Makes training easier by ensuring that values in the middle of the network are in a known range
 - Weight Regularisation
 - Implementation varies according to platform
 - Fine Tuning
 - Take an existing network, and adapt it to a new task
 - Data Augmentation
 - Create additional data by applying simple transforms to what you have
 - Fine-tuning and data augmentation can be used to help train networks with small datasets

Feature Representations

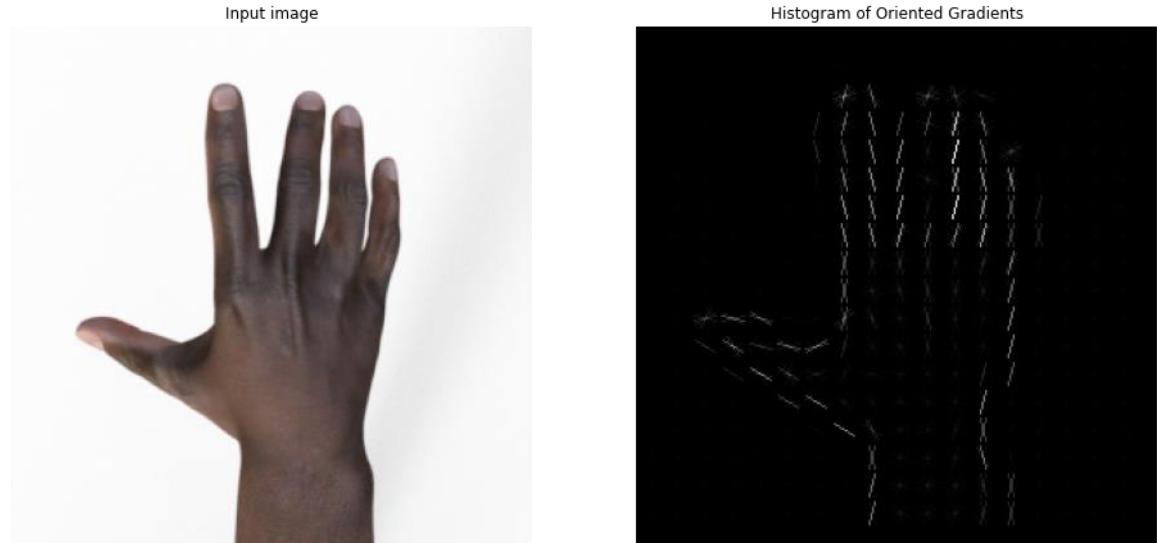
SOMETHING ELSE TO COMPLICATE THINGS

Feature Extraction

- So far, we've (mostly) used data "as is"
 - This is not always best
- Raw data may:
 - Contain useless or redundant information, or noise that hinders classification
 - Be very high dimensional
 - Not highlight the most important information for our task
- Feature extraction can be used to help
- Feature extraction will transform the data to another representation that is better suited to our task
 - Generally very domain specific, different methods exist for text, images, audio, etc
 - Methods typically have several parameters, which may need to be tuned
- A complete coverage of feature extraction is not possible in CAB420
 - We'll look a couple of methods here for images and text

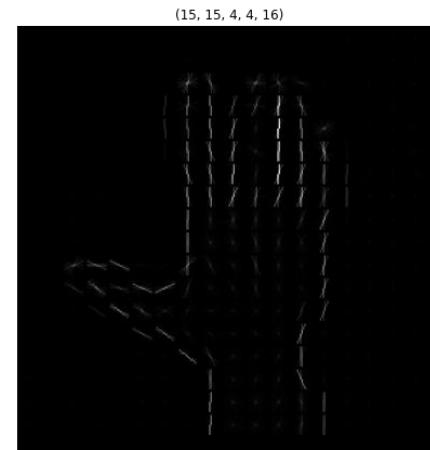
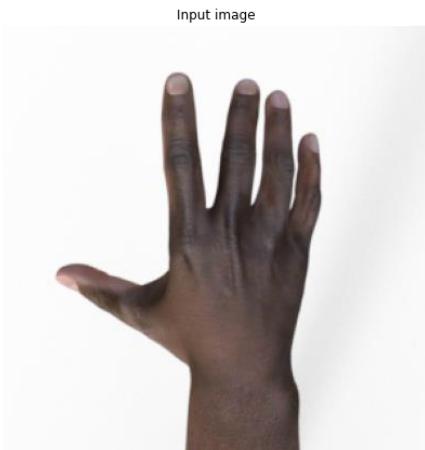
Histogram of Orientated Gradients (HOG)

- Extract features based on local texture
- The process is as follows:
 - Break image into patches
 - Compute gradients for each pixel in each patch
 - Compute magnitude and direction of gradient
 - High magnitude gradients will occur at the edge of the objects/regions
 - Direction of gradient indicates the direction of the edge
 - Quantise directions into a number of predefined directions
 - Build histograms for each region
 - Concatenate features for regions to obtain an overall image feature
- Image taken from
CAB420_Classification_Bonus_Example_HOG.ipynb



Histogram of Orientated Gradients (HOG)

- When extracting HOG you can select
 - Different numbers of regions/region sizes
 - Different numbers of gradient orientations
 - To aggregate features at a local region level
- Can lead to fine-grained or coarse features
- Image taken from ***CAB420_Classification_Bonus_Example_HOG.ipynb***

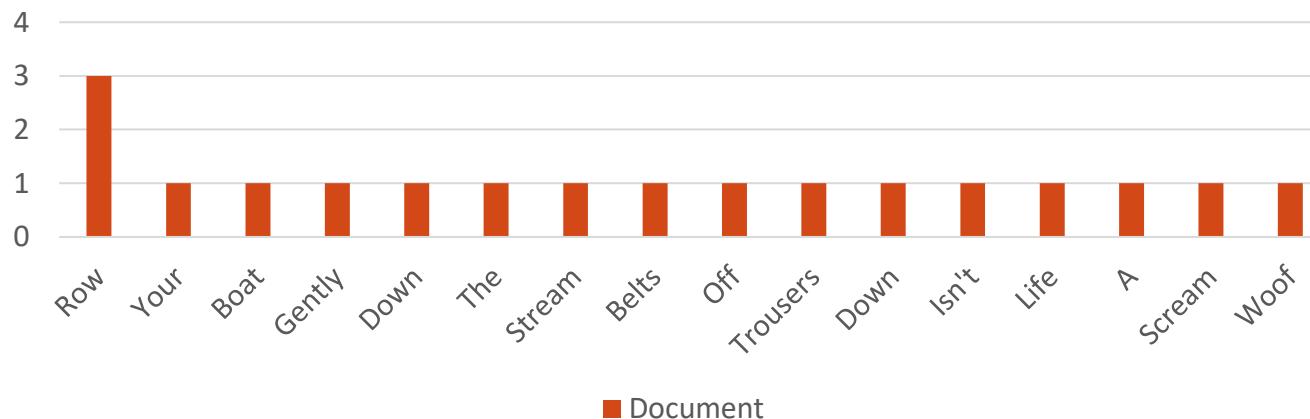


HOG vs Raw Pixels

- HOG is typically more compact (this does depend on HOG parameters)
 - Beneficial when dealing with small datasets
- HOG less sensitive to lighting changes
 - If the brightness of an image changes, the gradients don't
- HOG throws away colour information
 - This may or may not be important for the problem
- HOG offers some small invariance to translation and rotation
 - How much depends on the size of regions being considered, and the number of gradient bins
- HOG parameters do need to be set
 - These are essentially extra hyper-parameters that need to be trained
- Multiple feature extractors can be used in combination
 - HOG + ???
 - Increase feature dimension
 - Increases feature discriminability

Bag of Words (BoW)

- Method to encode text data
 - Text samples are likely to vary in length, BoW encodes a sample into a fixed length
- Encode a document as a histogram that measures word occurrences
- Example document and histogram
 - “Row, row, row your boat, gently down the stream, belts off, trousers down, isn’t life a scream, woof!”



Bag of Words

- Transforms variable length data into a fixed length representation
- Destroys the order of that data
 - Hence the “bag”
- Histograms can be very large and sparse
 - There might be thousands of words, but documents may be quite small
 - Most documents will only contain a subset of all words
- Can tune the size of the dictionary
 - Can remove rare words
 - But rare words can be very informative
 - What’s rare?
 - Can remove really common words
 - If they’re really common, they’re probably not that informative

Bag of Words: Preparation

- We usually do some preprocessing before applying Bag of Words
 - Typically, very similar to what we do prior to learning embeddings
 - Convert to lowercase
 - Remove punctuation
 - Possibly remove plurals, or specific words
 - Tokenise document
 - Extract out each word on its own
- Can also build models based on combinations of words
 - You might use all words, and all pairs of consecutive words
 - Captures some order which is otherwise lost by BoW
 - Dictionaries get very big, very fast

Bag of Words

- Methods can be sensitive to the size of the document
 - Small documents will have small bags
 - Big documents will have big bags
- Can normalize based on document frequency
 - Often also use inverse frequencies
 - Term Frequency-Inverse Document Frequency (tf-idf) is a common weighting scheme

Feature Representations and Deep Learning

- Deep learning uses multiple layers to learn its own representations
 - Feature extraction is generally less common, or simpler, with deep learning
- We can think of early layers of a network as performing feature extraction
 - But the feature extraction is learnt on the data itself, so it's tailored to the problem
- However, feature representations may still be used with deep learning
 - For 1D data (audio, biomedical signals) frequency based transforms are common
 - For text data, text needs to be converted to a numeric form (typically word embeddings)
 - Data is often resized or resampled to a fixed size

Some Code Examples

FOR CLASSIFICATION

An Example

- See ***CAB420_Summary_2_Classification.ipynb***
- Classification of images (CIFAR-10), but using raw pixel values as features are not ideal
 - Pixel features are sensitive to changes in lighting, position, rotation, etc
 - Pixel features can be huge, a 200x200 colour image has $200 \times 200 \times 3 = 120,000$ pixels, this is a lot of features
 - Feature extraction can be used to help
 - Histogram of Orientated Gradients
- We'll classify images using
 - SVM
 - CKNN
 - Random Forest
 - DCNN
 - We have spatial relationships in the data, so convolutions (2D being image data) make sense
- Review this example in your own time
 - Ask questions in class or via email/slack

An Example

- See ***CAB420_Summary_3_Text_Classification.ipynb***
- Classification of tweets into positive or negative sentiment
 - We'll use BoW features, though we could also use word embeddings
 - Word embeddings become less practical with longer sequences, though this is not such an issue with twitter data
- We'll classify tweets using
 - SVM
 - CKNN
 - Random Forest
 - DNN
 - Just a regular Deep Neural Network, no convolutions
 - Input data is a histogram, bins are not in any sort of order that makes sense for convolution operations
- Review this example in your own time
 - Ask questions in class or via email/slack

CAB420: Dimension Reduction

DIMENSIONS ARE BAD, MMMKAY

Why Reduce Dimensions?

- True vs Observed Dimensionality
 - Consider 20x20 binary bitmaps of handwritten digits
 - We have $\{0,1\}^{400}$ possible patterns
 - Most of these will never be seen
 - True dimensionality
 - Possible variations of the pen stroke
 - The set of examples we can actually see



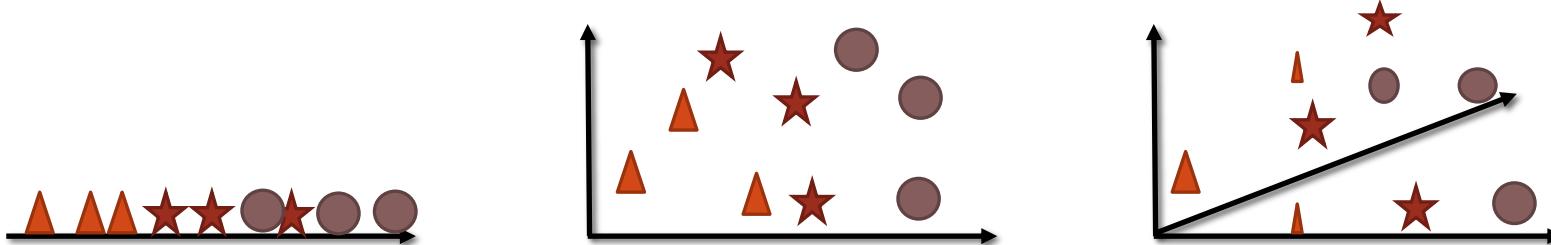
Curse of Dimensionality

Machine learning methods are based on statistics

- More observations means more reliable statistics

But

- More dimensions (without more examples) means that our examples are sparser in our feature space
- Less reliable models



Why Reduce Dimensions?

- Simplify or reduce the data
 - Help better represent the data given a limited number of samples
- Not all dimensions are equal
 - Some are high informative, others are not so useful, it's good to get rid of the useless ones
- Can make other machine learning tasks easier
 - And faster

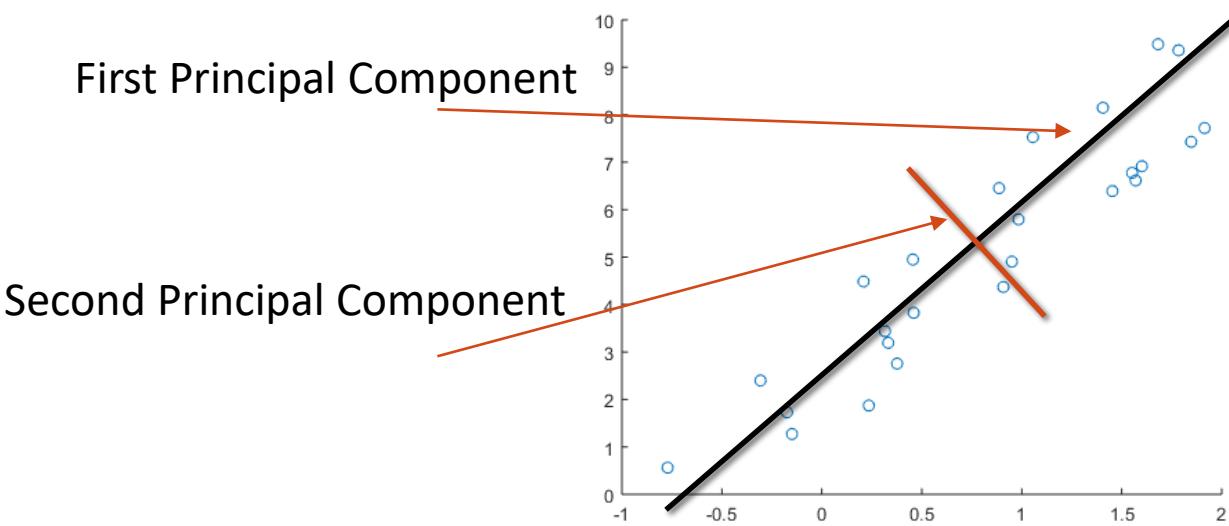
Principal Component Analysis

PCA

Principal Component Analysis

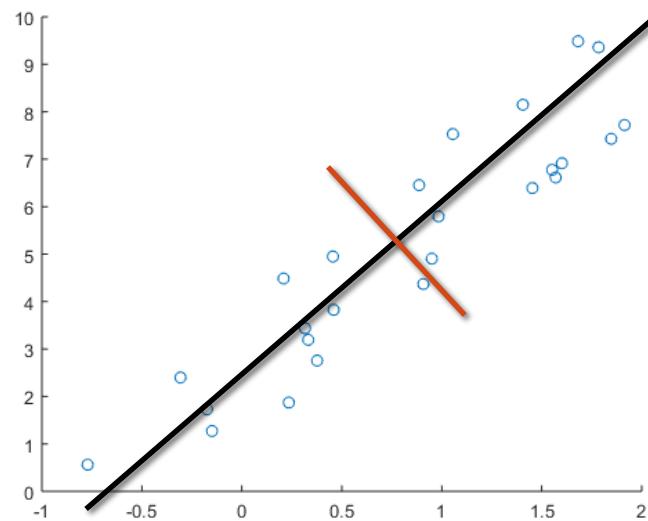
Principal components are:

- The directions where there is the most variance
- Can be seen as the underlying structure of the data



Eigenvectors and values

- Exist in pairs
 - Eigenvectors are the directions of variance
 - Eigenvalues specify the amount of variance in for the eigenvector
- We have as many pairs as we do dimensions in the source data



Finding Eigenvalues and Eigenvectors

- A covariance matrix can be formed between all dimensions in a dataset
- This matrix will be the same whether or not our data is centered (i.e., every entry in a dimensions is reduced by its mean)
- Principal components (PCs) are the directions in which variance is changing the most, not typically along an axis line
- PCs are represented by both eigenvalues and eigenvectors
 - Eigenvalues are related to the amount of variance in a principal component
 - Eigenvectors determine the direction of the PCs variance
 - These eigenvectors must have a magnitude of 1, as they will become unit measures.
- Recovery of Eigenvalues and Eigenvectors is done via linear algebra
 - Outside the scope of this subject
- Can also be found using Singular Value Decomposition
 - Typically, more numerically stable

Principal Component Analysis

- As the output of our PCA process, we get

$$T = XW$$

- X is the input data, containing N samples and P dimensions
- T is the transformed data
 - Same size as X
- W is the learned transformation
 - $P \times P$ matrix
 - Each column describes how the P dimensions are combined to create a new dimension
- To compute PCA, we need $N > P$
 - More samples than dimensions

Principal Component Analysis

- PCA will also offer a measure of variance for each new dimension
 - Derived from the Eigenvalues
 - New dimensions are returned in order of variance
 - The first few dimensions can be seen as the most important
 - Contain the most variance, thus the most information
 - Dimensions that contain limited variance are closer to being a constant, thus contain less information
- Using data that is not standardised can distort PCA results

Principal Component Analysis

Doesn't

- Add new dimensions
- Remove dimensions
- Change the data

It does

- Re-project the data along a new set of axes such that
 - The first dimension has the most variance
 - The second dimension has the second most variance
 - And so on

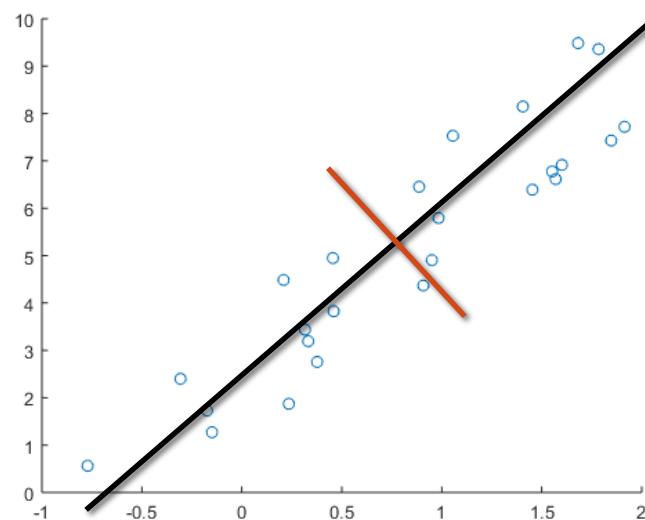
Projecting the Data

- When projecting it also centres the data
 - Translations of the data have no impact on the PCA results
- Projected axes are orthogonal to each other
 - i.e. at right angles
 - Ensures that the new axes can cover the data space as efficiently as possible
- We can also transform from the PCA space back to the original space

$$\hat{X} = TW'$$

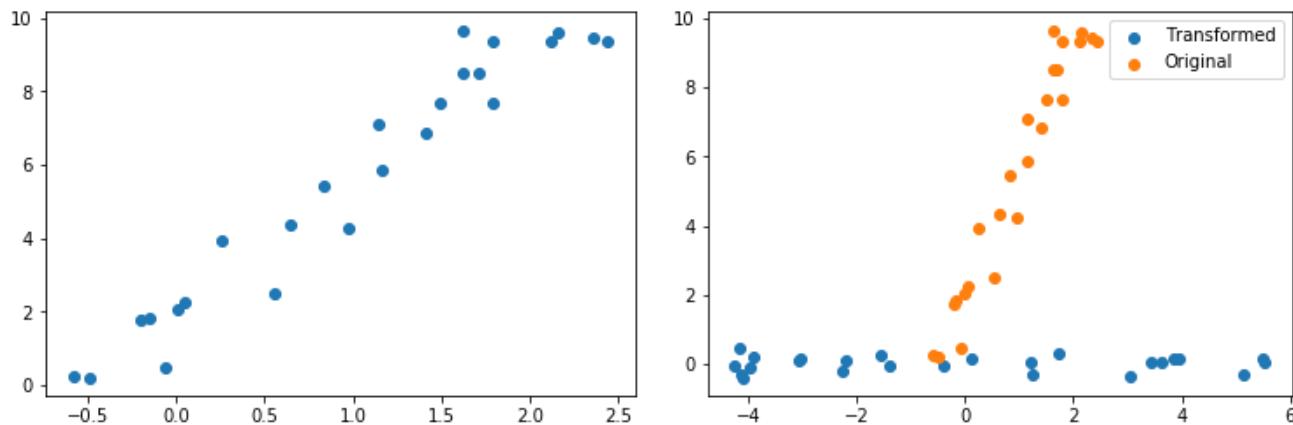
What are the new Dimensions?

- The new dimensions can be seen as weighted sums of the existing ones
 - Our first PC is (roughly)
 - $0.5x + 0.5y$
 - Our second PC is (roughly)
 - $-0.5x + 0.5y$
- PCA can be seen as computing a rotation matrix and rotating the data



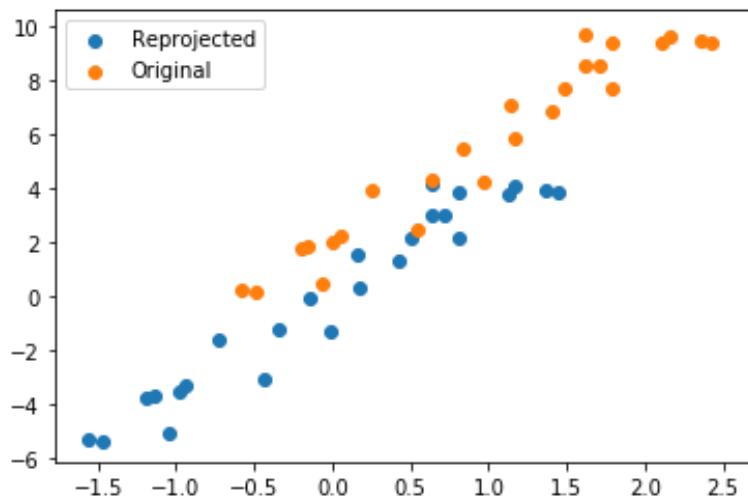
Simple Example

- See ***CAB420_Dimension_Reduction_Example_1_Principal_Component_Analysis.ipynb***
- Our data
 - Some random points on a line
- After applying PCA
 - Points are "lined up" on X and Y dimension
 - Centred
 - The way the points are distributed (their shape) is the same



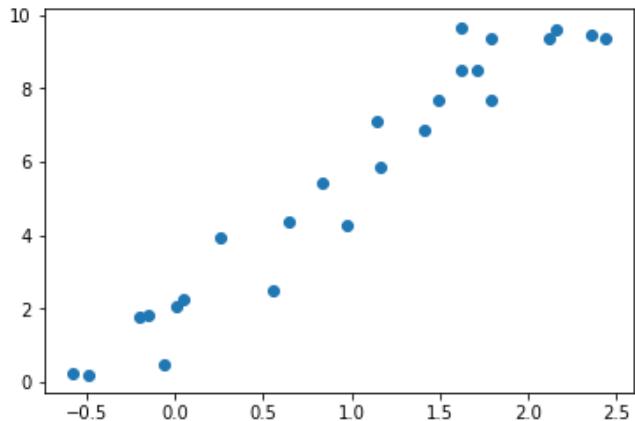
Simple Example: Reprojection

- When we reproject the data, we get a shifted version
 - PCA centers the data for us
 - Set the mean to be 0
 - First step of standardisation
 - Our reprojected version is thus centered at the origin
 - We need to save the mean of the original data to totally recover the original
 - Python will save the mean for us



Simple Example

- Projection matrix, W , shows how new dimensions are created
 - New dimensions are a weighted combination of original dimensions
- Eigenvalues show variance in each new dimension
 - First dimension contains 99.5% of the variance
 - This makes sense, our original data is spread out along one narrow line



$$W = \begin{bmatrix} [-0.260 & -0.966] \\ [-0.966 & 0.260] \end{bmatrix}$$

$$\text{Eigenvalues} = [12.144 \\ 0.050]$$

CAB420: PCA for Dimension Reduction

WE'VE GOT SOME NEW AXES, NOW WHAT?

Reducing Dimensions

PCA gives us

- A new coordinate frame
- Knowledge about how much of the total variance is captured by each new dimension

So we can just select the top N dimensions

- Retain a given amount of variance
- Remove dimensions that contribute very little

Reducing Dimensions

- We can view this as being

$$T_L = XW_L$$

- L is the number of dimensions we wish to keep
 - We select the first L columns of W , W_L
 - Our output now has only L dimensions

- We can reproject from the reduced space using

$$\hat{X} = T_L W_L'$$

- Information will be lost in such a reprojection

An Example

- See ***CAB420_Dimension_Reduction_Example_2_PCA_and_Dimension_Reduction.ipynb***
- Our data:
 - The Iris dataset, measurements of Iris petals, 4 dimensions
- Approach:
 - Apply PCA and transform data
 - Reconstruct the original using only some of the dimensions

Computed Transform

- When we compute PCA, we get
 - A transform
 - We have 4 dimensions in our input, so our transform matrix is 4x4
 - A measure of variance in each new dimension
 - The explained variance ratio is shown below (rather than the raw Eigenvalues)

```
[ [ 0.361 -0.085  0.857  0.358 ]
  [ 0.657  0.730 -0.173 -0.075]
  [-0.582  0.598  0.076  0.546]
  [-0.315  0.320  0.480 -0.754] ]
```

```
[ 0.92461872
  0.05306648
  0.01710261
  0.00521218]
```

1 Dimension

- Reconstructed and actual values for first 5 samples
 - Reconstructed samples are close to the original in values
 - We had ~92.5% of variance in the first principal component

Reconstructed:

```
[[4.86247892 3.28673941 1.4328746 0.22688569]
 [4.79929088 3.30151808 1.2830867 0.16423922]
 [4.85120324 3.28937661 1.40614547 0.21570665]
 [4.85721176 3.28797132 1.42038875 0.22166368]
 [5.01906124 3.25011732 1.8040546 0.38212597]]
```

Actual:

```
[[4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]]
```

Error: 0.085604

2 Dimensions

- Reconstructed and actual values for first 5 samples
 - Slightly more accurate than 1 dimension

Reconstructed:

```
[[4.7462619 3.15749994 1.46356177 0.24024592]
 [4.70411871 3.1956816 1.30821697 0.17518015]
 [4.6422117 3.05696697 1.46132981 0.23973218]
 [5.07175511 3.52655486 1.36373845 0.19699991]
 [5.50581049 3.79140823 1.67552816 0.32616959]]
```

Actual:

```
[[4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]]
```

Error: 0.025341

3 Dimensions

- Reconstructed and actual values for first 5 samples
 - Minimal error with three dimensions included

Reconstructed:

```
[[4.86875839 3.03166108 1.4475168 0.12536791]
 [4.69370023 3.20638436 1.30958161 0.18495067]
 [4.6238432 3.07583667 1.46373578 0.25695828]
 [5.0193263 3.58041421 1.37060574 0.24616799]
 [5.40763506 3.89226243 1.68838749 0.41823916]]
```

Actual:

```
[[4.9 3. 1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5. 3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]]
```

Error: 0.005919

Breaking PCA

BREAKING STUFF IS FUN

PCA

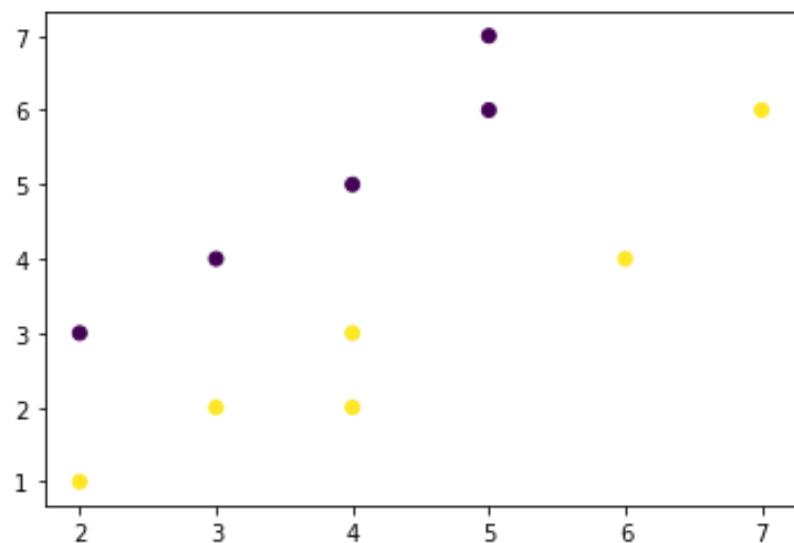
- Dimension reduction technique
 - Project the data into a set of new dimensions such that the
 - First new dimension captures the most variance
 - The second captures the second most variance
 -
- Can reduce dimensionality by electing to only retain a percentage of total variance
 - Typically most (90% or more) variance is in a small number of dimensions (10% or less)

But...

- Is the amount of variance captured the best criteria for determining what to keep?
- Are the most important variables the ones that make different types of object distinct?

Breaking PCA

- See ***CAB420_Dimension_Reduction_Example_3_Linear_Discriminant_Analysis.ipynb*** (the first part)
- Simple dataset
 - Two classes
 - Well separated



Breaking PCA

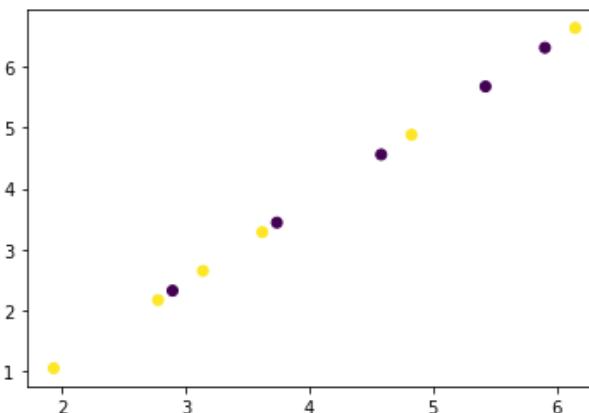
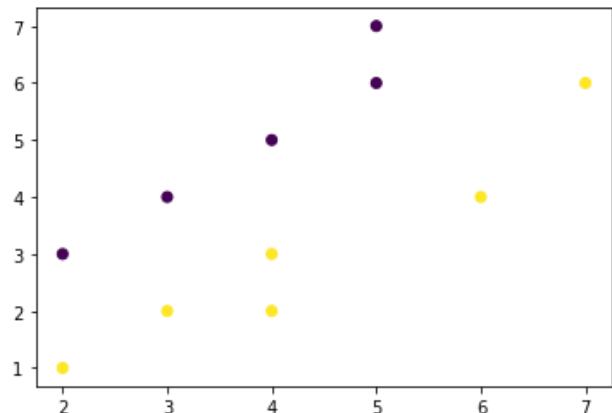
- Computing PCA on our dataset, we get:
 - Explained variance: [0.85471369 0.14528631]
 - Most variation is in the first dimension
- If we transform the data using just the first PC, and reproject we get:

Actual:	Reconstructed:
[3 4]	[3.73842482 3.44234505]
[4 5]	[4.58253002 4.56007607]
[5 6]	[5.42663522 5.67780709]
[5 7]	[5.90755333 6.31462]
[2 1]	[1.9324834 1.05098821]
[3 2]	[2.7765886 2.16871923]
[4 2]	[3.13977569 2.64963734]
[4 3]	[3.6206938 3.28645025]
[6 4]	[4.82798609 4.88509938]]

- Average reconstruction error = 0.408243

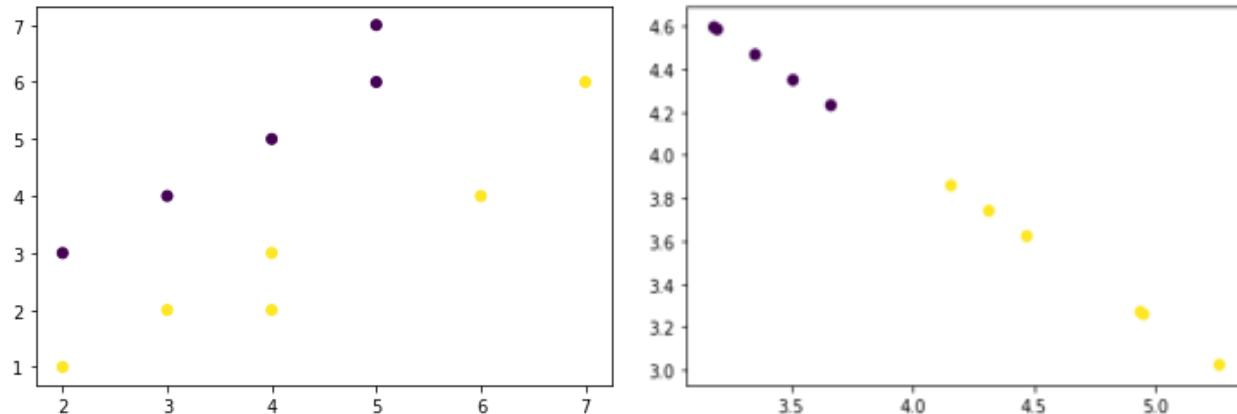
Breaking PCA

- Left: original data
- Right: reconstructed data (from first PC)
- We've lost any ability to separate the classes



Breaking PCA

- If we used only the second principal component
 - Our reconstruction looks nothing like the original data
 - Class separation is preserved



CAB420: Linear Discriminant Analysis

A BIT LIKE PCA, BUT NOT

Linear Discriminant Analysis

- Also often called Fisher Discriminant Analysis
- LDA seeks to
 - Find a projection that maximises the ratio between the inter class and between class scatter matrices
 - Meaning:
 - Samples in the same class get closer together
 - Samples in different classes get further away
 - Find a projection that best **discriminates** between the classes

LDA Overview

- PCA considers the variance between existing dimensions of the data
 - Finds a new set of dimensions such that the first dimension contains the most variance, etc.
- In Linear Discriminant Analysis, we instead focus on the "variance" relating to classes of the data.
- In order to effectively separate clusters during dimension reduction, we must take into consideration:
 1. The "variance" between classes, and
 2. The "variance" within each class.

LDA Overview

- The difference between each point and its cluster's mean is the crucial part of LDA.
 - We don't need the "divided by $n - 1$ " that we normally associate with variance. In fact, this will only scale our new dimensions.
 - The matrix result without the division is instead called a **scatter matrix**.
- One important note is that we should ensure a sufficient number of points in each class in order for LDA to succeed. This is because LDA is very sensitive to outliers.
- The number of data points in the smallest class should be larger than the number of explanatory variables.

LDA –Scatter Matrices

- LDA is based on scatter matrices

- Within class

- $S_w = \sum_{i=1}^n (x_i - \mu_{y_i})(x_i - \mu_{y_i})^T$

- where

- n is the number of points

- μ_{y_i} is the mean of the i th class

- Between class

- $S_B = \sum_{k=1}^m n_k (\mu_k - \mu)(\mu_k - \mu)^T$

- where

- m is the number of classes

- n_k is the number of points in the k th class

- μ_k is the mean of the k th class

- μ is the mean of the data

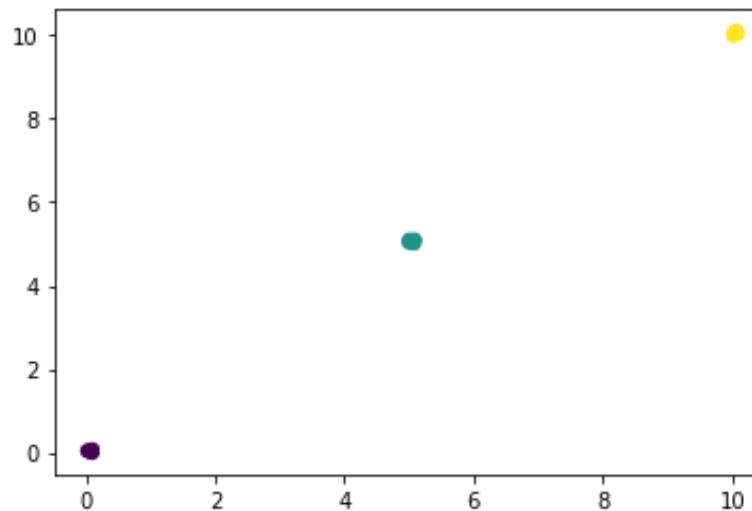
Scatter Matrices Visualised

- See [*CAB420_Dimension_Reduction_Example_3_Linear_Discriminant_Analysis.ipynb*](#) (the second part)
- 2D Data, 3 Classes
 - Within class scatter

```
[ [0.01331477  0.0015448 ]  
[ 0.0015448   0.01423041]]
```

- Between class scatter

```
[ [248.86709419  249.70285217]  
[ 249.70285217  250.54892153]]
```



Scatter Matrices Visualised

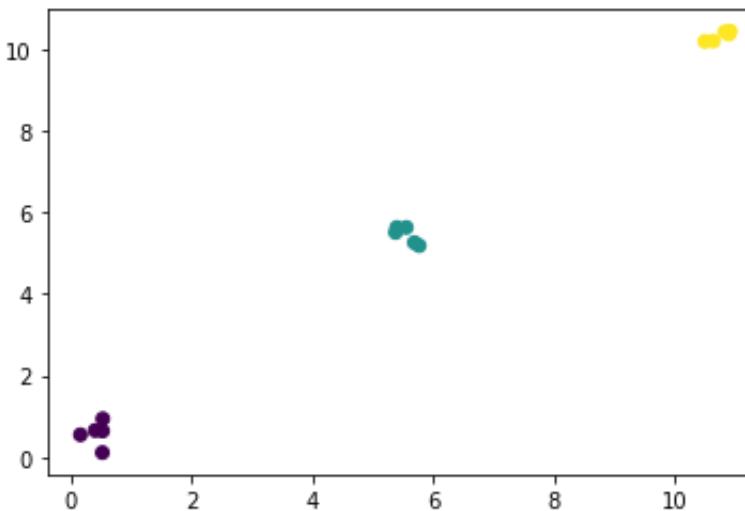
- 2D Data, 3 Classes

- Within class scatter

```
[ [ 0.35319307 -0.03366045]  
[ -0.03366045 0.59361245] ]
```

- Between class scatter

```
[ [266.73188433 251.55160505]  
[251.55160505 237.23710806] ]
```



Scatter Matrices Visualised

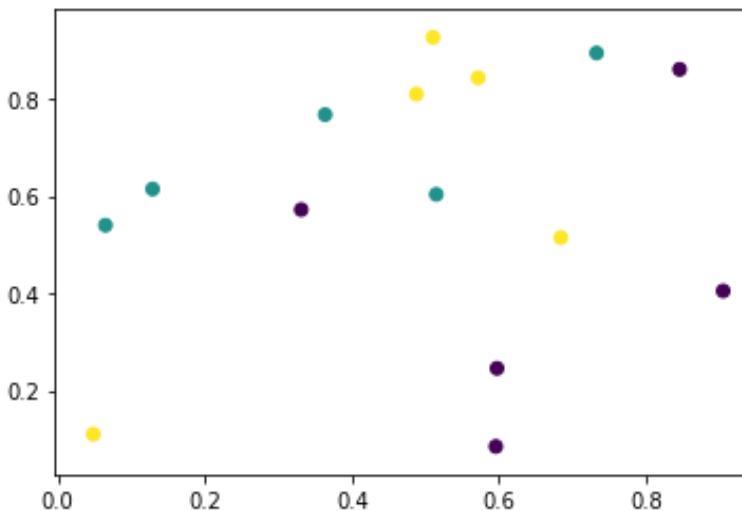
- 2D Data, 3 Classes

- Within class scatter

```
[ [ 0.74859984  0.41766819]  
[ 0.41766819  0.89271018] ]
```

- Between class scatter

```
[ [ 0.22332289 -0.19700407]  
[-0.19700407  0.17936888] ]
```



Computing LDA

- LDA seeks to maximise ratio of between class to within class scatter

$$\hat{w} = \operatorname{argmax}_w \frac{w^T S_B w}{w^T S_W w}$$

- This can be solved by computing the eigenvectors for $\frac{S_B}{S_W}$.
- LDA will only return $C - 1$ meaningful components
 - C is the number of classes in the data

Computing LDA

- Same simple sample data we broke PCA with (on the left)

- Within class scatter matrix

```
[ [24.13333333 24.          ]
  [24.          26.          ] ]
```

- Between class scatter matrix

```
[ [ 0.77575758 -2.90909091]
  [-2.90909091 10.90909091] ]
```

- Transformed Data (on the right)

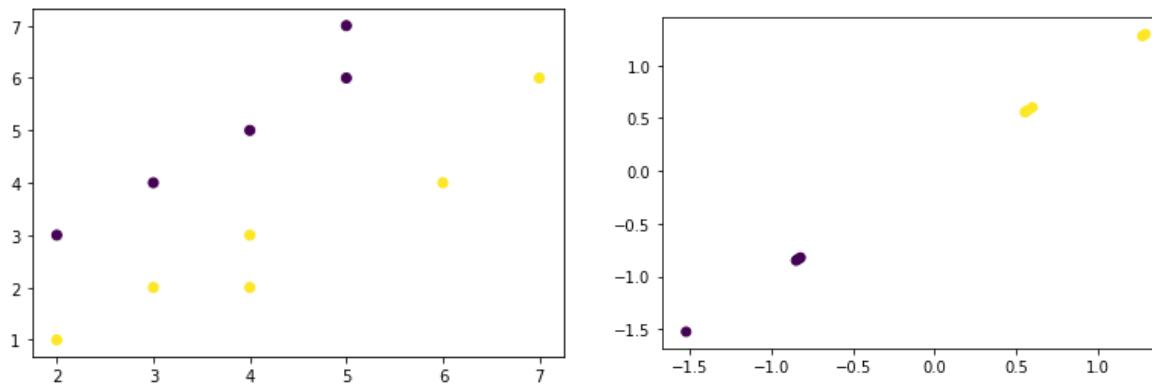
- Eigenvectors

```
[ [-0.96623494  0.71169327]
  [-0.25766265 -0.70249034] ]
```

- Eigenvalues

```
[0.          8.22044277]
```

- Only 1 non-zero Eigenvalue

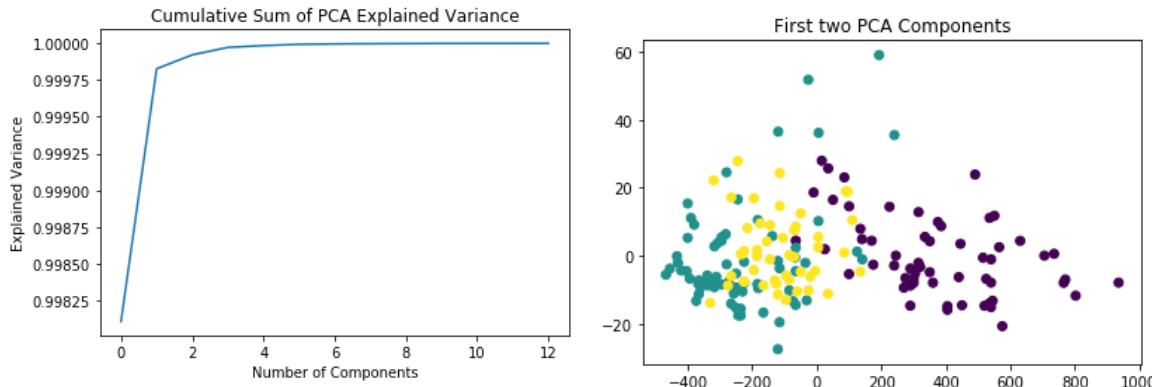


An Example – PCA vs LDA

- See ***CAB420_Dimension_Reduction_Example_4_Linear_Discriminant_Analysis_II_Action_Time.ipynb***
- Data
 - Chemical properties of wine from three different cultivars in Italy
 - 12 variables
 - This is ostensibly a classification task, though we'll just analyse the data and visually look at class separation

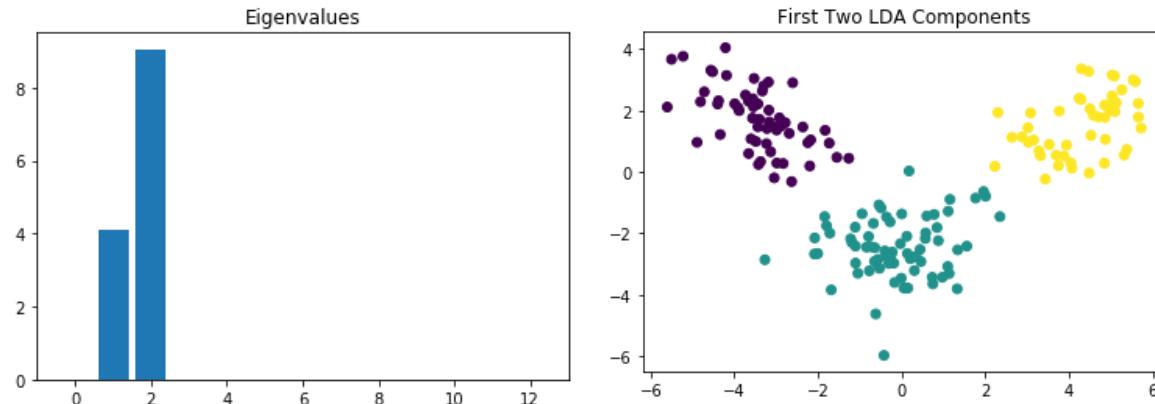
PCA

- Almost all the information is in the first dimension
 - 99.8% of the variance in one dimension
- Plotting the top two dimensions (~99.975% of the variance) we see that we have some class separation
 - Purple is somewhat separated, the other classes not so much



LDA

- Note that I only have two meaningful dimensions
 - LDA returns $C - 1$ components
 - I have 3 classes, thus 2 components
- Two dimensions result in very good separation of classes

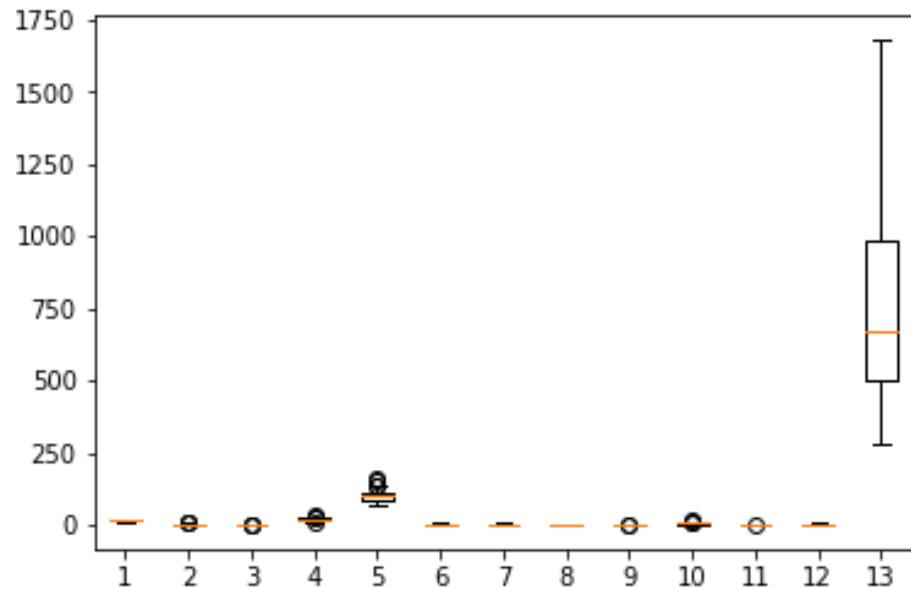


Breaking LDA

- LDA is sensitive to the number of samples per class
- If we have very few samples per class
 - Scatter matrices become hard to predict
 - LDA solution can become unstable
- This problem becomes more pronounced with different solvers
 - SVD (Python's default) is generally more stable and robust in these situations

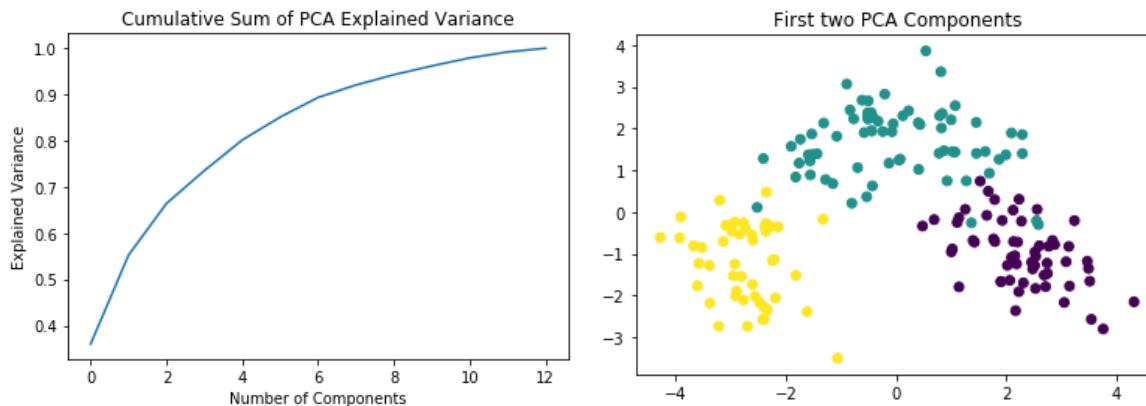
PCA – what went wrong?

- The boxplot of the data explains our problem
 - The variation in the data is dominated by a single dimension
 - This distorts our PCA, and hurts it's performance



Improving PCA

- Standardised data
- We no longer have all the variation in one dimension
- Class separation much better on first two dimensions
 - Not as good as LDA, but close



LDA and Acronym Overuse

Warning

- There is another LDA: latent Dirichlet allocation
- Some toolkits/text refer to latent Dirichlet allocation as LDA exclusively
 - If you search for LDA in MATLAB you will get taken to info on latent Dirichlet allocation
 - Google will give you results for both
- “Fisher Discriminant Analysis” is another name for Linear Discriminant Analysis

Incidentally...

- Latent Dirichlet allocation is a really cool method for modelling topic distributions
- We'll look at that when we look at sequences

CAB420: t-SNE

YET MORE DIMENSION REDUCTION,
YET ANOTHER ACRONYM

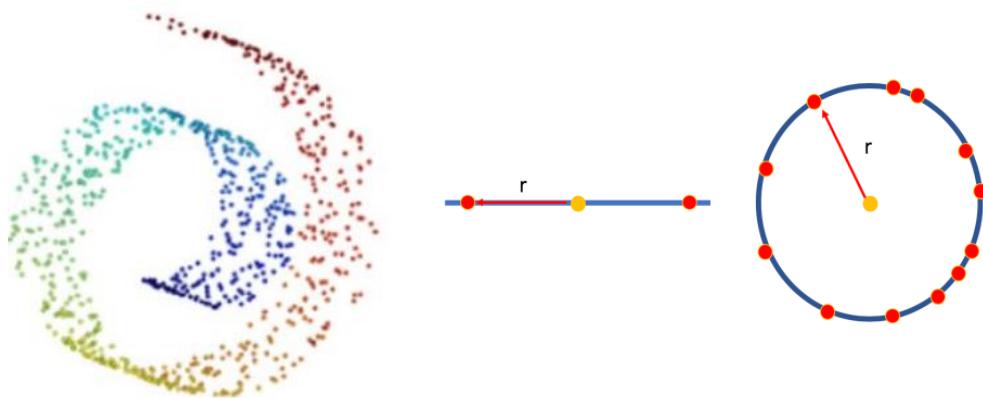
t-distributed Stochastic Neighbour Embeddings

t-SNE is a bit different

- Fairly recent method (2008) developed for visualisation
- Not a linear projection
- Projects data into two dimensions
- Uses local relationships to create a low dimensional mapping that can capture non-linear structure

Why t-SNE?

- Lots of things aren't linear (see left), and PCA (and LDA) struggle to capture such a relationship
- Overcomes the “crowding problem”, when projecting from a high to low dimensional space points can become tightly clustered and “crowded”



Why t-SNE?

- Intended for visualisation
 - Often we use dimension reduction to visually analyse data, t-SNE is designed for this
- Becoming widely used
 - Used a lot when visualising neural network intermediate states or outputs
 - We will use it for this and similar purposes
 - Important to understand how it differs from other dimension reduction techniques

t-SNE Overview

Two main steps:

- Step 1: In high dimensional space, create a probability distribution that captures the relationship between points
- Step 2: Create a low dimensional space that follows the created distribution as best as possible
- More details: <https://mlexplained.com/2018/09/14/paper-dissected-visualizing-data-using-t-sne-explained/>
- <https://distill.pub/2016/misread-tsne/>

Limitations

- Non-convex optimisation
 - Not guaranteed to reach a global minima (i.e. a constant best solution)
 - Non-deterministic
 - If you run it again, you'll get different results
- Assumes Locally Linear Manifolds
 - Uses Euclidean distance to compare points (which is generally an ok assumption)
 - If local relationships are highly non-linear, may struggle to make sense of things

CAB420: Eigenfaces

FACE REC WITH PCA

Face Recognition: Eigenfaces

- Classic face recognition approach that uses PCA
- See ***CAB420_Dimension_Reduction_Example_5_Eigenfaces.ipynb***
- Problem
 - Identify a person from an image, given an input image and a database of images with known identities

Face Recognition: Naïve Solution

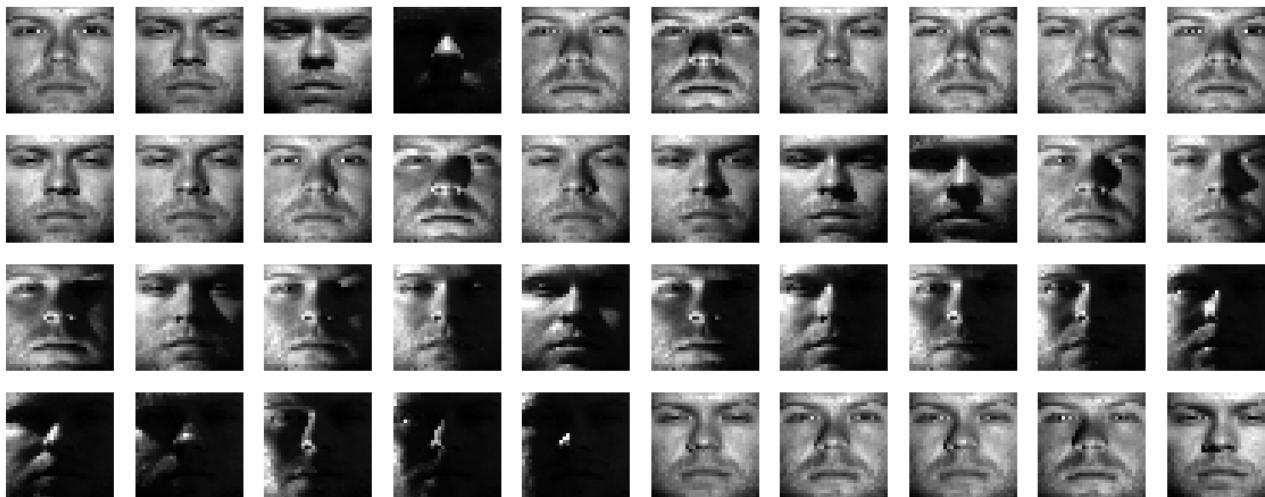
- For input image
 - Align/crop image such that it's at a consistent pose/scale
 - Compare, pixel-wise, to all images in the dataset
 - Select identity of closest image as best match
- Problems with this approach
 - Image are large, even small images have very high dimensionality
 - What happens in the lighting changes, the background changes, etc?

Eigenfaces

- For our database
 - Compute PCA over database, return top N dimensions
- For input image
 - Map to PCA space
 - Compare to database in PCA space and select best match as target ID
- Still sensitive to lighting, background, etc
 - Though less so than a raw pixel approach
- Much quicker due to fewer retained dimensions

Input Data

- Face images
 - 32 x 32 pixels, greyscale
 - Varied lighting
 - Consistent pose
 - Images normalised based on eye locations



PCA and Faces

- The mean face
 - Fairly average looking
 - Perhaps more male than female
 - Nature of the dataset, more males than females in the data
 - All lighting variation removed



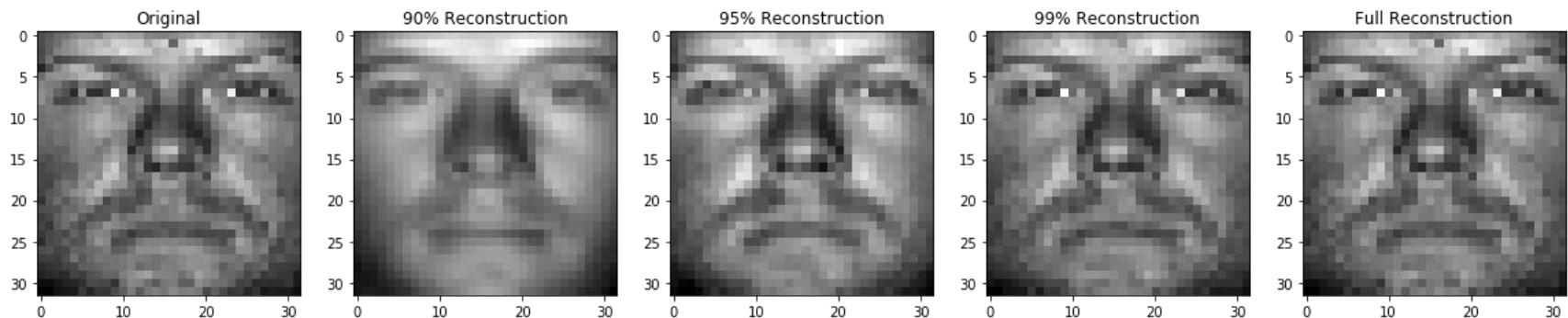
Eigenfaces

- Our data are images
 - Thus, our eigenvectors (principal components) are images
- Our eigenvectors (eigenfaces) capture the major variations in faces
 - Early eigenvectors capture the drastic lighting changes



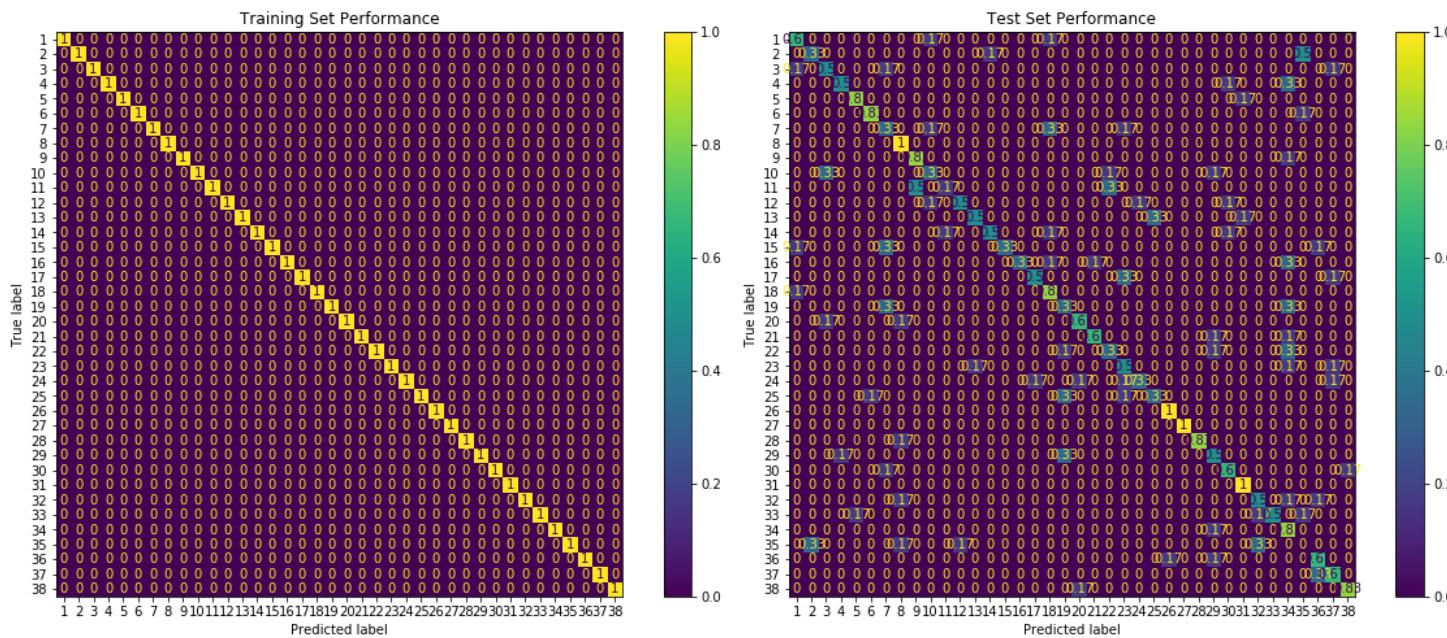
Low Dimensional Faces

- Reconstructions as retained PCs are increased
 - 90% is 23 PCs
 - 95% is 60 PCs
 - 99% is 238 PCs
- More PCs means more fine-grained details
 - Fewer PCs can be (sort-of) seen as a low pass filter



Recognising Low-Res Faces

- Train a classifier
 - CKNN
 - Trained on dimension reduced data



Combining PCA and LDA

LIVING TOGETHER, IN PERFECT HARMONY
(OR SOMETHING LIKE THAT)

Why?

Don't they do similar things?

- Reduce dimensions, etc?

Sort of, but not really

- PCA is focussed on reconstruction
- LDA focussed on class separation
 - Throws data away

LDA can have problems when you have

- Lots of dimensions and/or classes
- Few sample points per class

When LDA Goes Bad

When we perform LDA with

- Lots of dimensions and/or classes; and/or
- Few sample per class

The estimate of the within scatter class matrix in particular become poor

- Can lead to overfitting
- Can lead to a singular or close to singular matrix
 - Singular -> non invertible
 - We need to invert the within class scatter matrix
- Leads to bad results

PCA as Pre-Processing

A Solution

- Reduce the size of the data space
- i.e. PCA

The process

- Apply PCA, select top N components, removing uninformative dimensions
- Apply LDA to reduced space
- When transforming a new sample point:
 - $x_{transformed} = x * PCA * LDA$
 - i.e. transform to PCA space, then to LDA space

This will be explored further in this weeks tutorial

Final Thoughts

SUMMING UP AND ALL THAT

LDA vs PCA

- PCA
 - Unsupervised
 - Aim to preserve as much information as possible
 - Can be inverted
- LDA
 - Supervised
 - We need to know class labels
 - Aim to preserve as much discriminative power as possible
 - Cannot be inverted

PCA Requirements

- PCA is unsupervised
 - No labels needed, just the raw data
- We need to have more samples than we do dimensions
 - If we don't have this, many PCA methods will transpose the data
 - Or be unstable
- Standardisation can help (a lot)

PCA Requirements

- Ideally, we want many more samples than dimensions
 - PCA may become unstable if we only have slightly more samples than dimensions
- We can do robustness tests by computing multiple PCA transforms on datasets sampled from our overall data
 - If the PCA transform is consistent, we can claim that we have enough data
 - Similar to the idea of the standard error measured when fitting a regression model

LDA Requirements

- Class labels
 - LDA is supervised
- Sufficient examples per class to estimate scatter matrices
 - Ideally, more samples per class than dimensions
- A problem that doesn't require reconstruction
 - We cannot reconstruct the original signal from LDA
- Standardisation not needed
 - The scatter matrices capture this
 - Standardisation may result in an axis being flipped, but that's it

Which one to use?

As the previous slides suggests, it depends on

- What are you trying to do?
 - Classification?
 - Visualisation?
 - Compression?
- What data do you have?
 - Class labels or not?
- Methods can be used in combination
 - FisherFaces: apply PCA, then apply LDA

Lots of other dimension reduction techniques too

The methods that we've look at are not always best. Other methods include:

- Independent Component Analysis
- Factor Analysis (and it's many variants)
- Probabilistic PCA and Probabilistic LDA
- Deep network-based methods
 - Metric learning
 - Auto-encoders
- And many, many, more ...

CAB420: Comparing Things with Deep Neural Networks

THING A VS THING B

Comparing Things with Machine Learning

- Often in machine learning we want to know how similar two things are
- Two main uses for comparisons
 - Verification
 - Identification
- The face recognition examples from our look at PCA and LDA are good examples
 - Are these two people the same?

Verification

- Verification, seeks to answer the question “are these two things the same?”
 - Example: SmartGate at Australian Airports
 - You claim an identity via your passport
 - The system then attempts to verify that it’s you
 - Usual approach
 - Compute distance between claimed identity and observed identity
 - Apply a threshold to decide if samples are the same

Identification

- Identification, seeks to answer the question “which thing is this?”
 - Based on having a gallery of previously seen “things”, and comparing the new “thing” to all these other “things”
 - Typically returns a ranked list, i.e. the N most similar “things”
- Usual approach
 - Compute distance between new “thing” and all other previously seen “things”
 - Return a ranked list based on distance

A Simple Approach

- Telling if two things are the same can be viewed as a classification task
 - Requires us to know all classes in advance
 - Then we train a classifier to split them
 - This is what we've done for face rec with PCA and/or LDA and a CKNN
- However
 - What happens when a new class of things appears?
 - Open world scenario: we don't know what all the "things" are when we train the network
 - In this case, we need to retrain the lot
 - That gets impractical quickly

Open World Approaches

- What we want to do is
 - Computes a subspace where
 - Things of the same class are close
 - Things of different classes are far away
 - To compare two things, we
 - Project them into the subspace
 - Compute the distance between them
- This method scales and allows us to add new “classes” after training
 - Assuming the data is of the same broad type/format, etc

Replicating this with DCNNs

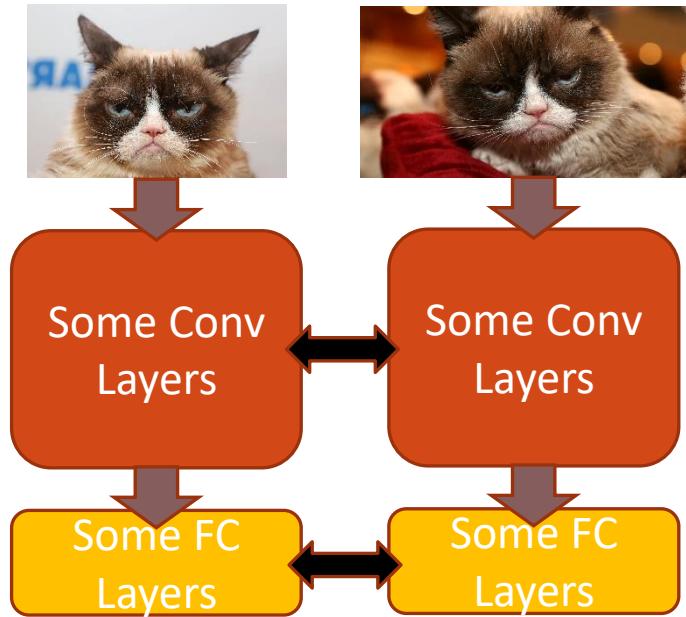
- We need to
 - Train a model that can learn a subspace where we can compare things
 - This should, given a pair of images extract a compact representation that
 - Has things of the same class close to each other
 - Has things of a different class far away from each other
 - The distance between should, ideally, reflect similarity

CAB420: Siamese Networks

COMPUTER VISION AND CATS

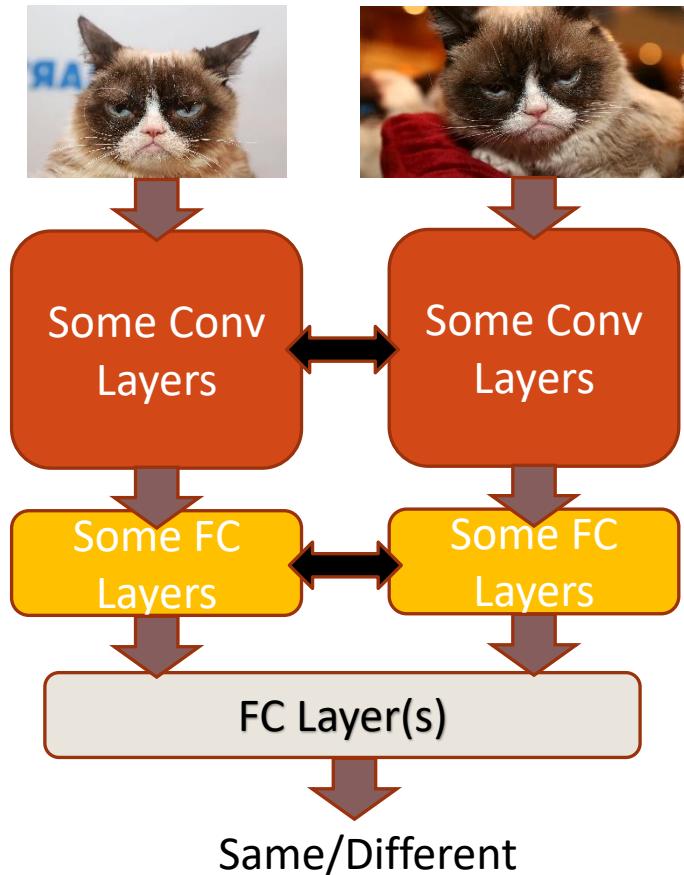
Siamese Networks

- We have two streams
 - Each is identical
 - Same structure, weights, etc.
 - For the same image, each stream will extract the same features
 - For images of the same class we'll get similar features
 - Hopefully



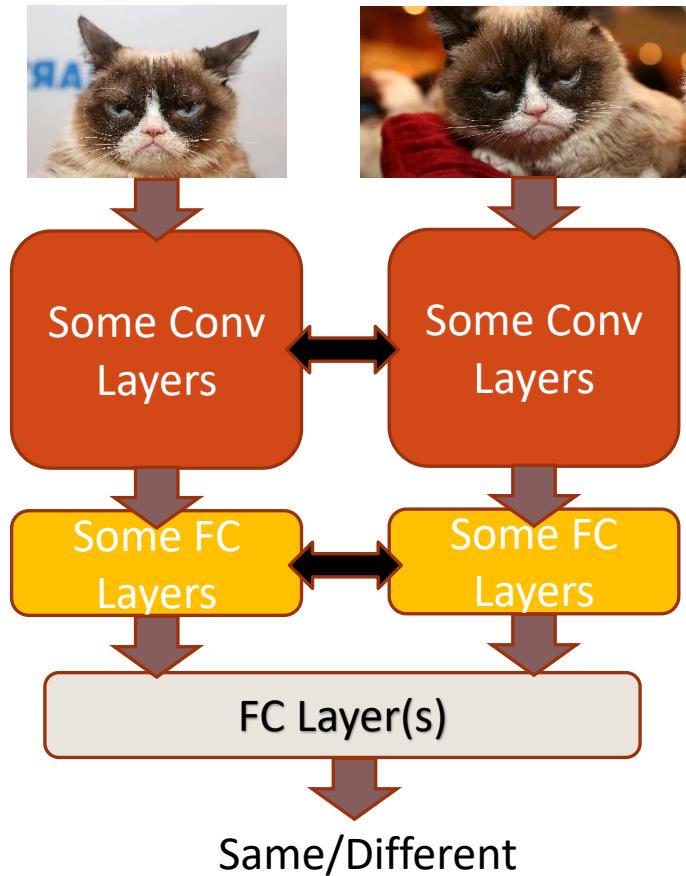
Siamese Networks

- The output of these streams is a compact representation
 - We'll call this an embedding
 - We can compare these embeddings to tell if things are the same, or different
 - May be compared by another small network



Siamese Networks

- The main stream of the network
 - Can be anything
 - VGG
 - ResNet
 - Something else
 - The embedding
 - Can be whatever size you want
 - Larger embeddings are potentially more descriptive



Comparing Things with Siamese Networks

NEURAL NETWORKS AND CATS – A MATCH
MADE IN INTERNET HEAVEN

Comparing Embeddings - Naïve Solution

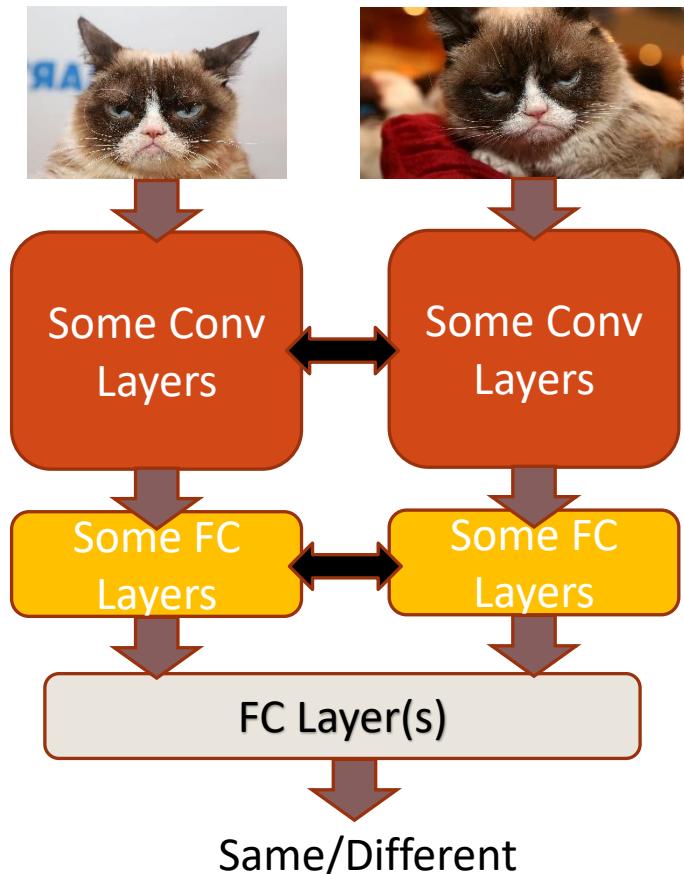
- We can view this as a binary classification task
 - A pair of images is either
 - Of the same type
 - Of a different type
 - Therefore
 - Binary Cross Entropy

$$L = -\frac{1}{N} \sum_{i=1}^N y_i \log(\bar{y}_i) + (1 - y_i) \log(1 - \bar{y}_i)$$

- y_i = true probability, 0 or 1
- \bar{y}_i = estimated probability
- N = batch size

Comparing Embeddings - Naïve Solution

- What this means is
 - We take our two embeddings
 - We input them into another network with a single output
 - An output of 1 indicates the same class
 - An output of 0 indicates different classes

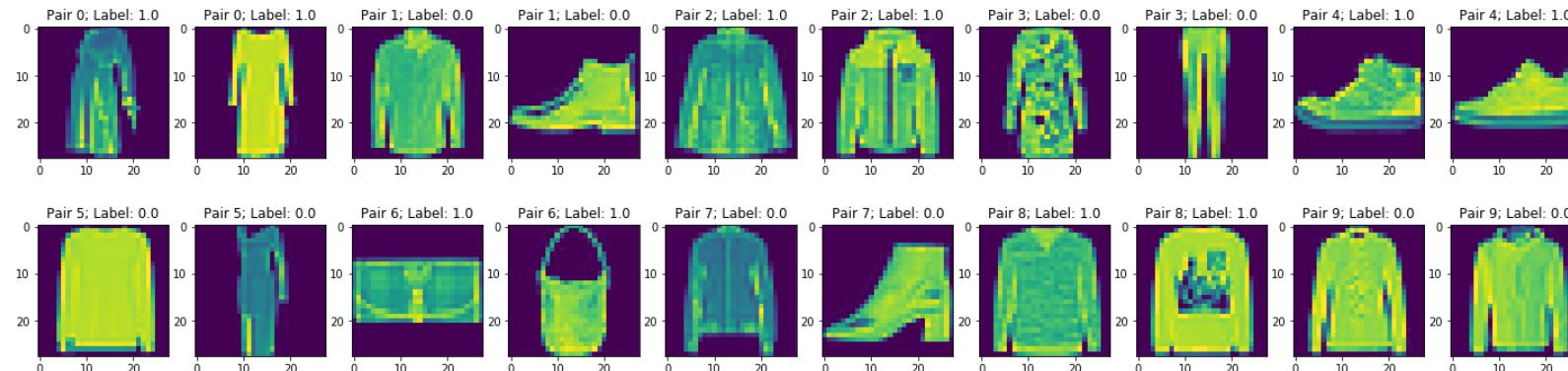


Data

- We need pairs of images
- We can generate this from a labelled dataset
 - i.e. we have class identities
- Randomly select a sample
 - For a positive pair, randomly select another sample of the same class
 - For a negative pair, randomly select another sample of a different class

An Example

- See ***CAB420_Metric_Learning_Example_1_Siamese_Networks.ipynb***
- Data
 - Fashion MNIST
 - Random pairs create, 50% positive (same class), 50% negative (different class)



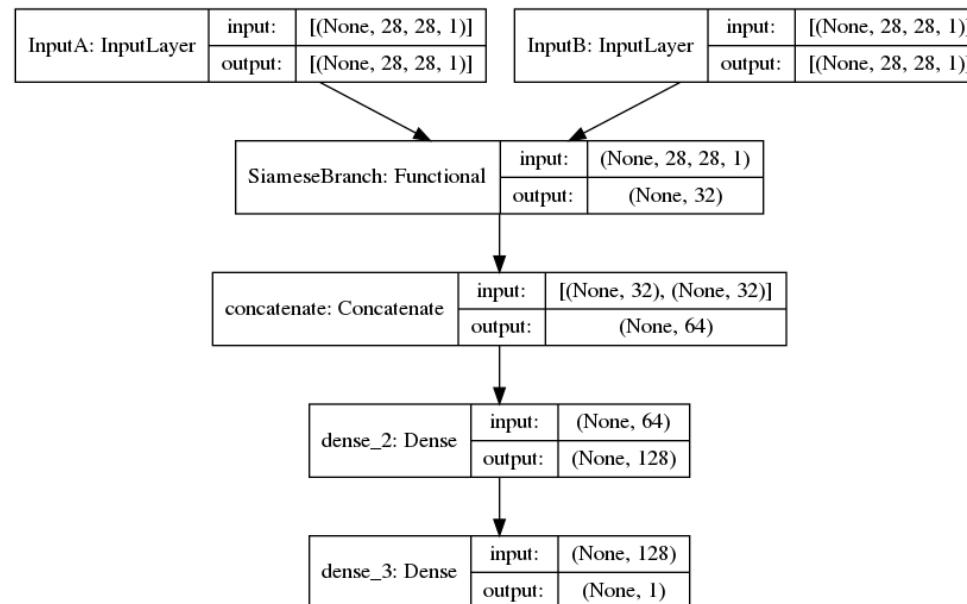
Backbone Network

- The backbone is the network that processes the two input images prior to comparing them
 - VGG Style Network
 - 3 pairs of two 2D Convolution layers
- Final dense layer of size 32
 - The embedding
- We could use a pre-trained network here and fine-tune

Model: "SiameseBranch"		
Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[None, 28, 28, 1]	0
conv2d_24 (Conv2D)	(None, 28, 28, 8)	80
conv2d_25 (Conv2D)	(None, 28, 28, 8)	584
batch_normalization_13 (BatchNormalization)	(None, 28, 28, 8)	32
activation_4 (Activation)	(None, 28, 28, 8)	0
spatial_dropout2d_12 (SpatialDropout2D)	(None, 28, 28, 8)	0
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 8)	0
conv2d_26 (Conv2D)	(None, 14, 14, 16)	1168
conv2d_27 (Conv2D)	(None, 14, 14, 16)	2320
batch_normalization_14 (BatchNormalization)	(None, 14, 14, 16)	64
activation_5 (Activation)	(None, 14, 14, 16)	0
spatial_dropout2d_13 (SpatialDropout2D)	(None, 14, 14, 16)	0
max_pooling2d_9 (MaxPooling2D)	(None, 7, 7, 16)	0
conv2d_28 (Conv2D)	(None, 7, 7, 32)	4640
conv2d_29 (Conv2D)	(None, 7, 7, 32)	9248
batch_normalization_15 (BatchNormalization)	(None, 7, 7, 32)	128
activation_6 (Activation)	(None, 7, 7, 32)	0
spatial_dropout2d_14 (SpatialDropout2D)	(None, 7, 7, 32)	0
flatten_4 (Flatten)	(None, 1568)	0
dense_11 (Dense)	(None, 256)	401664
batch_normalization_16 (BatchNormalization)	(None, 256)	1024
activation_7 (Activation)	(None, 256)	0
dense_12 (Dense)	(None, 32)	8224
=====		
Total params: 429,176		
Trainable params: 428,552		
Non-trainable params: 624		

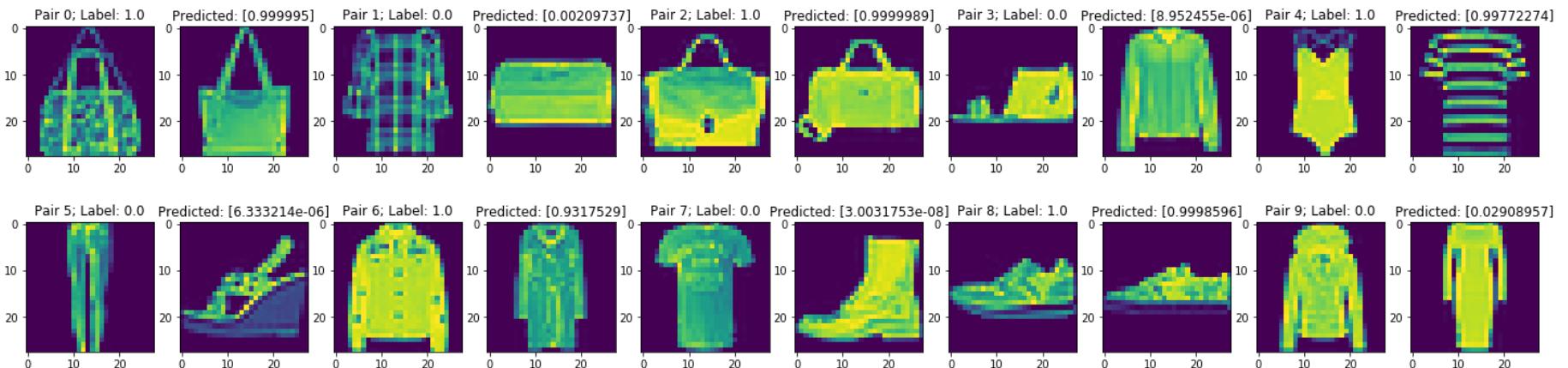
Siamese Network

- Pass two images through the same backbone
- Concatenate the output
- Pass the concatenated result through one or more addition dense layers
- Final layer of size 1:
 - True/False to indicate if images are the same or different classes



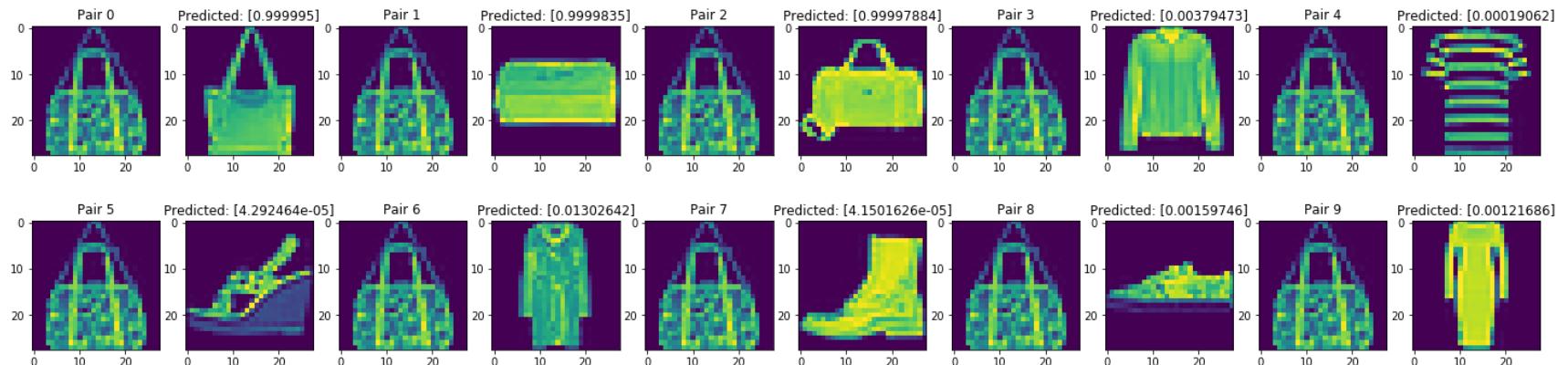
Some Predictions

- All show correct results
 - Values close to 1 for positive (same) pairs
 - Values close to 0 for negative (different) pairs



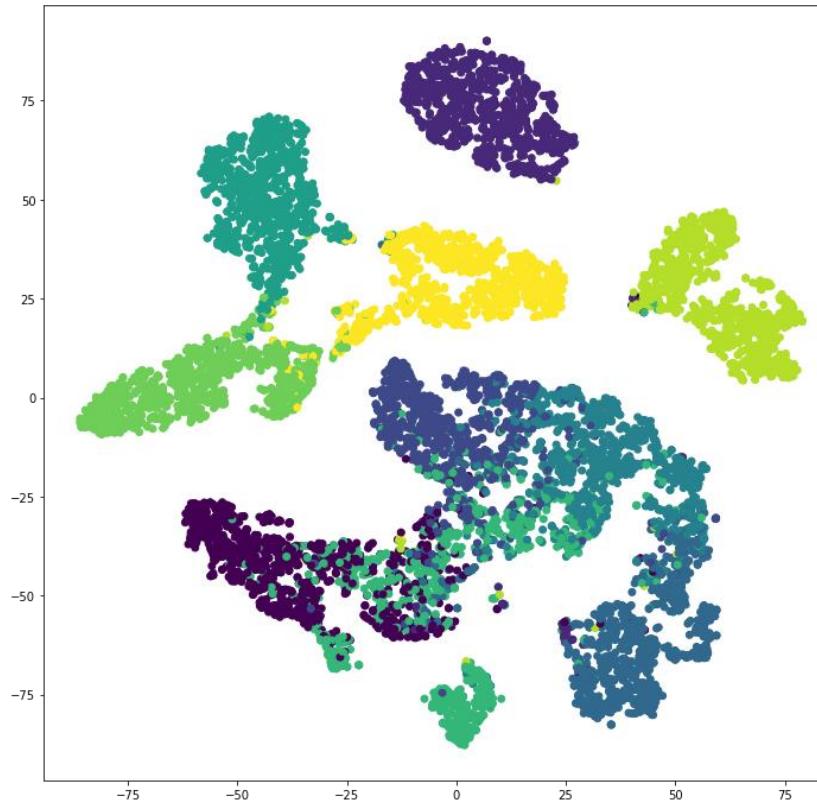
Some Predictions

- Same query image in all cases
 - Correct labels are predicted in all cases
 - Little, if any, meaning in the actual scores beyond same/different



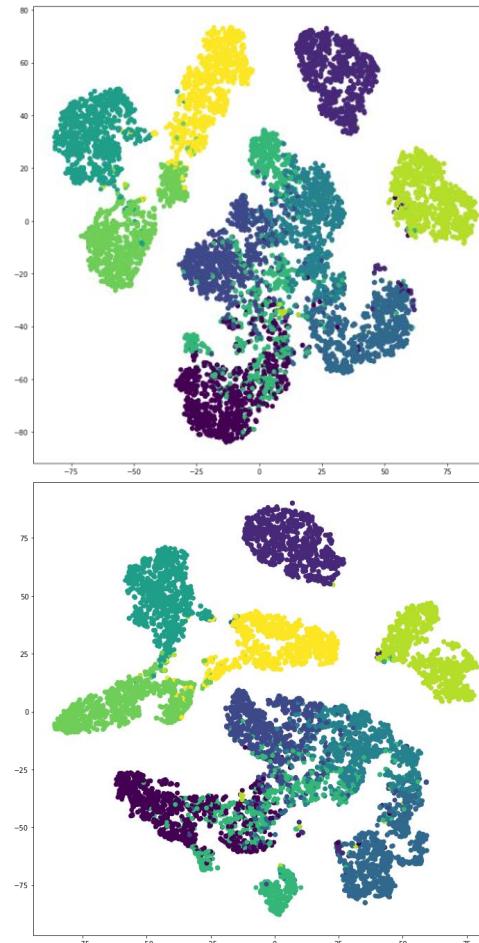
What did the network learn?

- Visualise embeddings using t-SNE
- Pass entire test set through the backbone network
 - Extract embeddings
- Plot embeddings in 2D using t-SNE
 - Some classes are well separated
 - Others, not so much



Comparing with a regular classification network

- Take our same backbone, train for classification using categorical cross entropy
 - Extract embeddings (size 32) and plot with t-SNE
- Top: DCNN + Categorical Cross Entropy
- Bottom: Siamese
 - Both networks perform similarly
 - Slightly cleaner class boundaries with the Siamese network
 - To be expected, both networks have the same structure and a very similar loss



CAB420: A Better Loss (Contrastive Loss)

POOR PERFORMANCE MAKES ME GRUMPY

What's Wrong with Binary Cross Entropy?

- With our first approach we can determine if two items are the same or not by applying a threshold
- But...
 - The loss (BCE) does not force similar images to have similar features
 - We rely on another network to do this
 - This limits our ability to “rank” things

Contrastive Loss

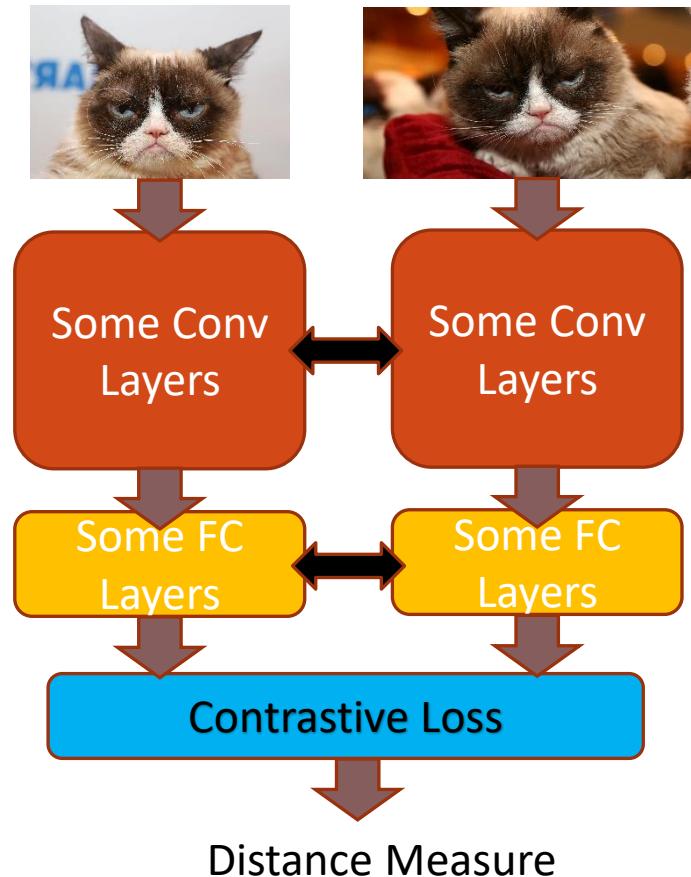
- What we'd like is to try to replicate LDA
 - Have examples of the same class close to each other
 - Have examples of different classes far from each other
- Contrastive Loss

$$L = \frac{1}{N} \sum_{i=1}^N y_i d_i + (1 - y_i) \max(\text{margin} - d_i, 0)$$
$$d_i = |\bar{y}_1 - \bar{y}_2|_2$$

- note that we can use a different distance metric here, such as cosine or L1
- y_i is the pair label; 1 indicates the pair is the same class; 0 indicates they are different
- Contrastive loss aims to separate positive and negative pairs of embeddings by a margin

Contrastive Loss Network

- What this means is
 - We take our two embeddings
 - We compute the loss directly on them
 - No subnetwork needed



Contrastive Loss Margin

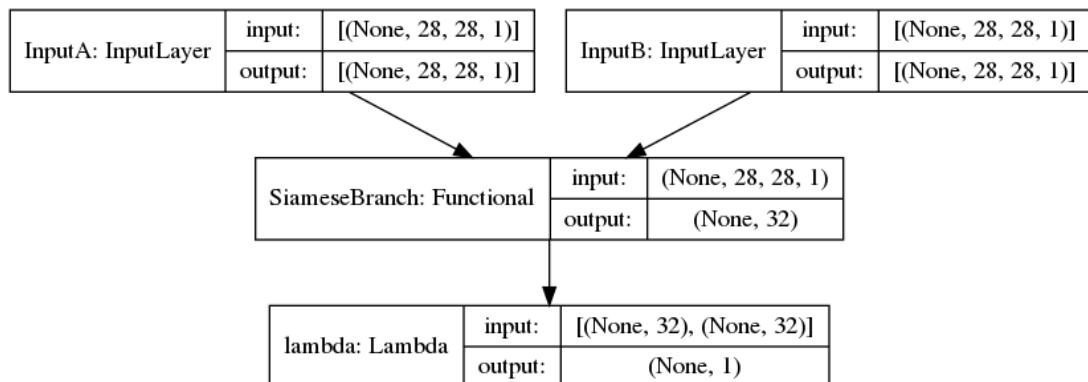
- What should it be?
 - Depends on the range of the magnitude of the embedding
 - Bigger embedding, potentially bigger margin
 - Could change it dynamically
 - As the network learns, make it bigger
 - Or, normalise the embedding
 - Embeddings now have a constant magnitude, regardless of size
 - Can then set the margin to 1 and regardless of embedding size, this will work
 - We'll do this

An Example

- See ***CAB420_Metric_Learning_Example_2_Contrastive_Loss.ipynb***
- Data
 - Same as Example 1

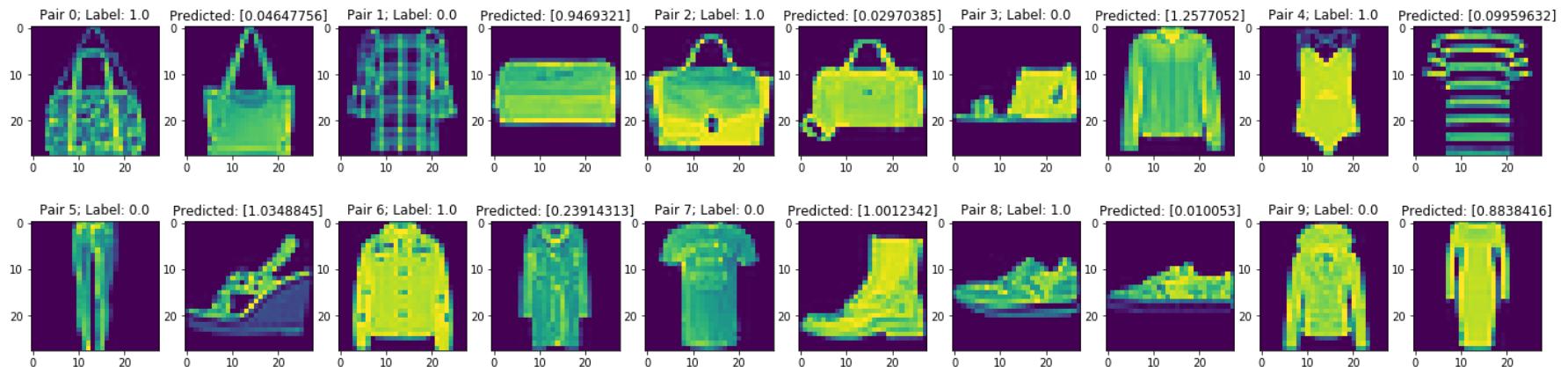
The Network

- Very similar to Example 1
 - Same backbone
 - No feature concatenation and dense layer(s)
 - Loss is computed directly on the two embeddings from the backbone
 - Simpler network on the whole



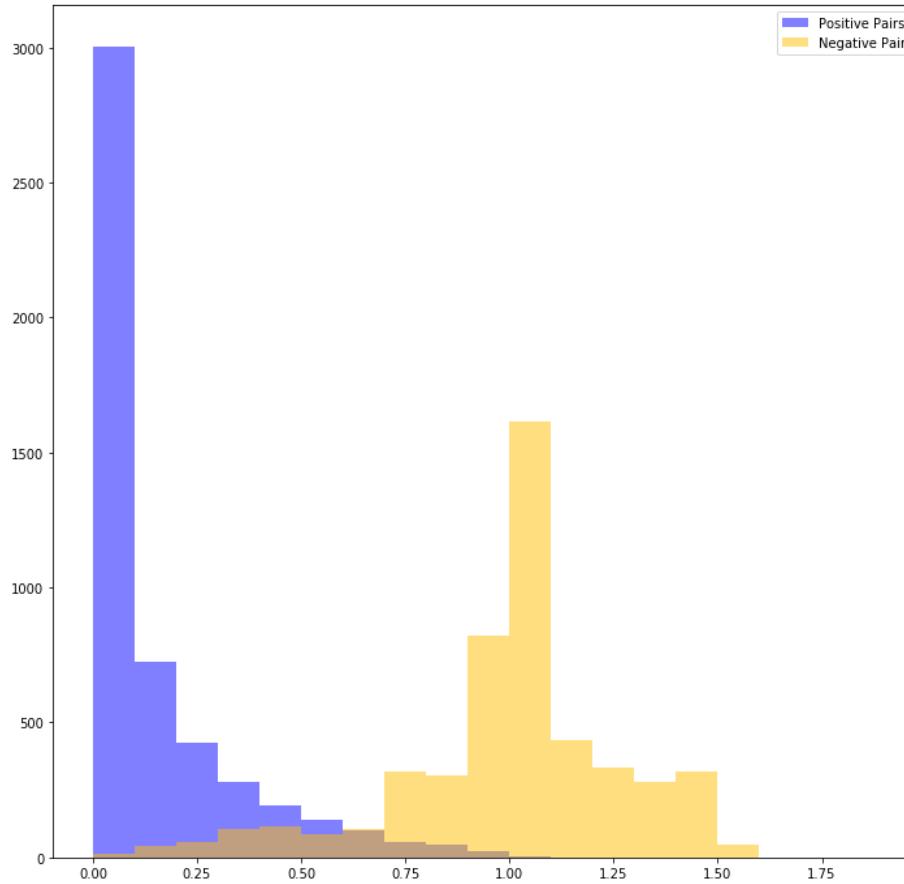
Some Predictions

- Items that are the same (label 1) should have a small distance
- Similar items have distances less than 0.5
- Dissimilar items have distances greater than 0.5



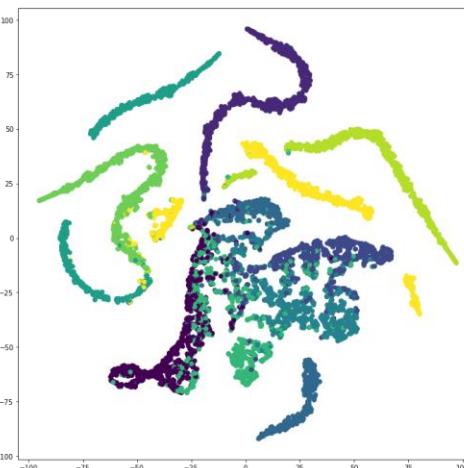
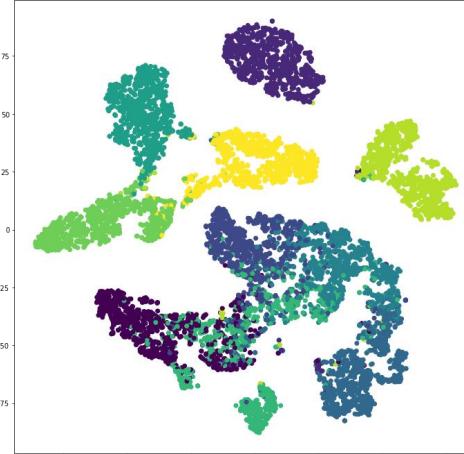
Distribution of Distances

- This plot shows
 - A histogram of distances for matched pairs
 - A histogram of distances for mismatched pairs
- We have trained the network to have items of the same class closer together in feature space
 - This has, mostly, been achieved
 - Some overlap in distributions
 - Would lead to errors when making decisions



Embedding Space

- t-SNE plots of the learned embedding
 - Top: Binary Cross Entropy
 - Bottom: Contrastive loss
- Contrastive loss leads to much clearer class boundaries
 - Still some confusion
 - Some classes are broken into multiple clusters
 - Possibly capturing different semantic details
 - Different types of shoes perhaps?

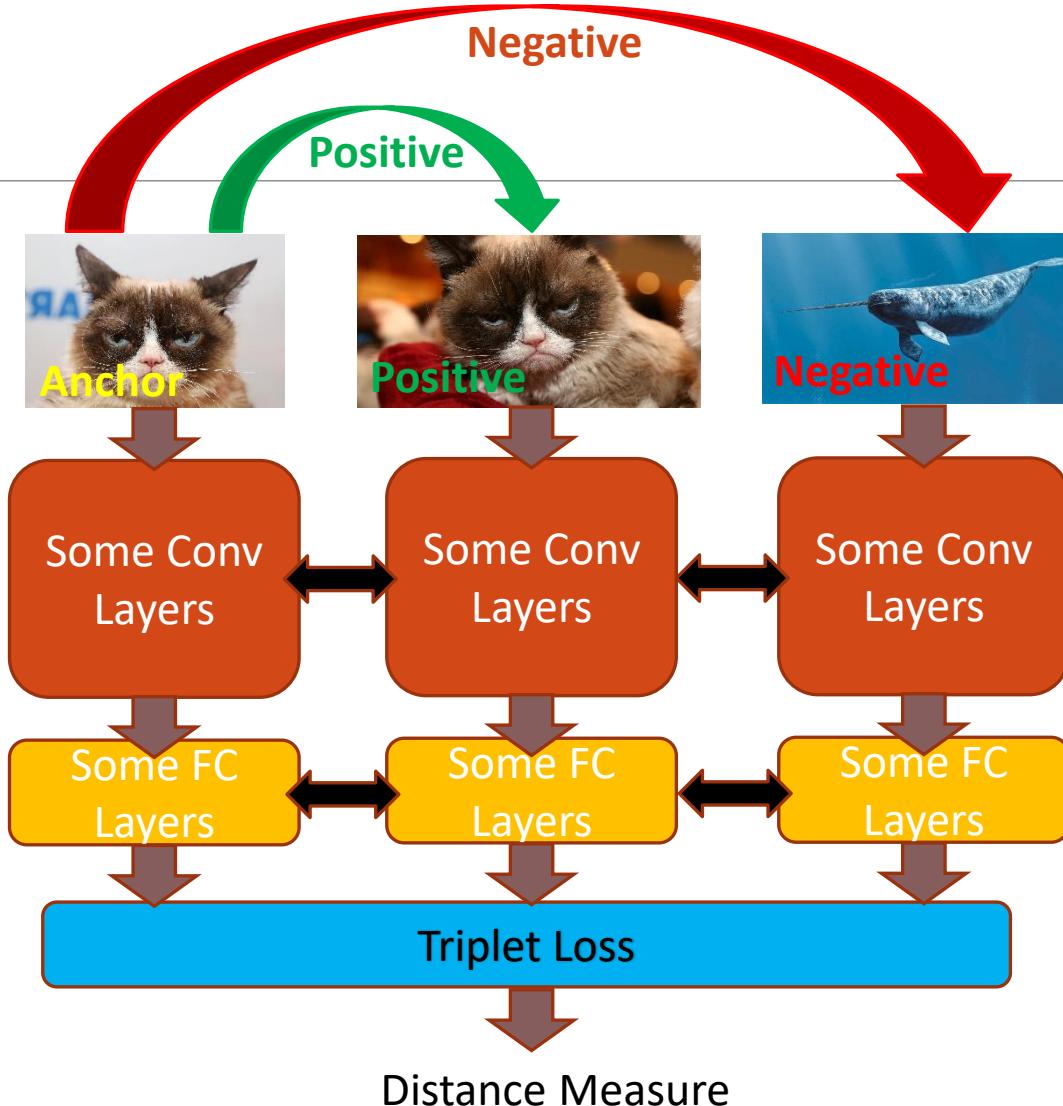


CAB420: Because Two's Not Enough

TRIPLET NETWORKS

Triplets

- Three images
 - Anchor
 - Positive
 - Negative
- Two pairs
 - Positive Pair
 - Negative Pair
- Pull the positive pair embeddings close
- Push the negative pair embeddings far away



Triplet Loss

$$L = \max(d(a, p) - d(a, n) + \text{margin}, 0)$$

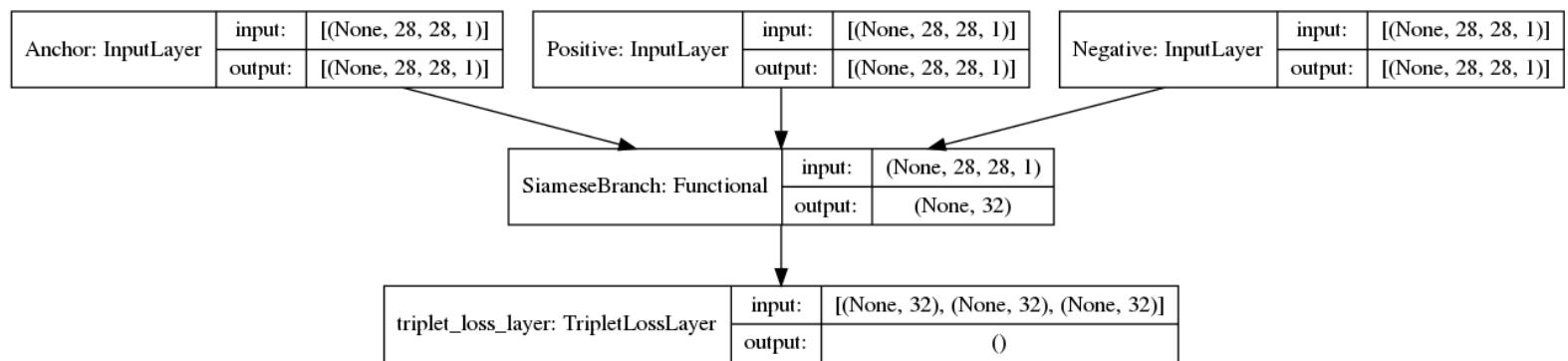
- $d(a, p)$, distance between anchor and positive
- $d(a, n)$, distance between anchor and negative
- As usual we can use whatever distance function we like
- **The margin**
 - Serves much the same role as it does in the contrastive loss
 - Again, if we normalise vectors we can fix this at 1

An Example

- See ***CAB420_Metric_Learning_Example_3_Triplet_Loss.ipynb***
- Data
 - Same as first two example, except we now have triplets
 - Each sample has
 - A random **anchor** image
 - A random **positive** image, the same class as the anchor
 - A random **negative** image, of a different class to the anchor

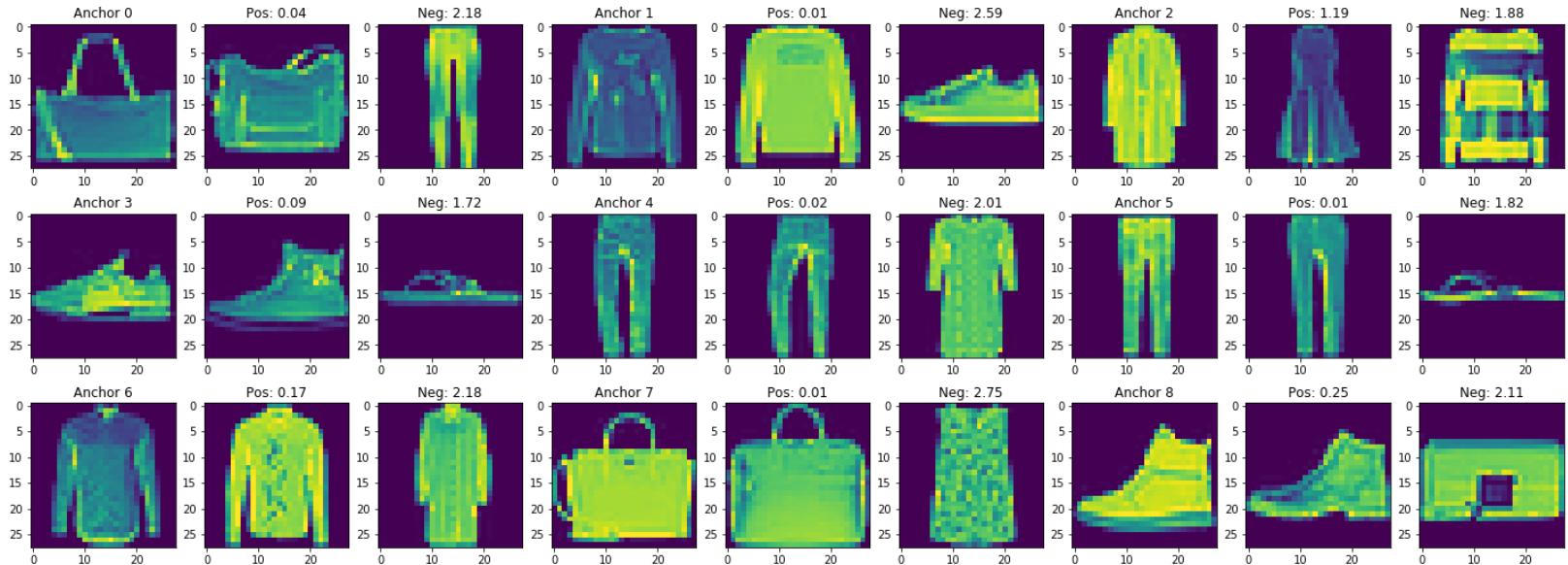
The Network

- Same backbone as Examples 1 and 2
- Three inputs
- Triplet loss
 - As per contrastive loss, no additional layers beyond the backbone
 - Loss computed directly on the embeddings



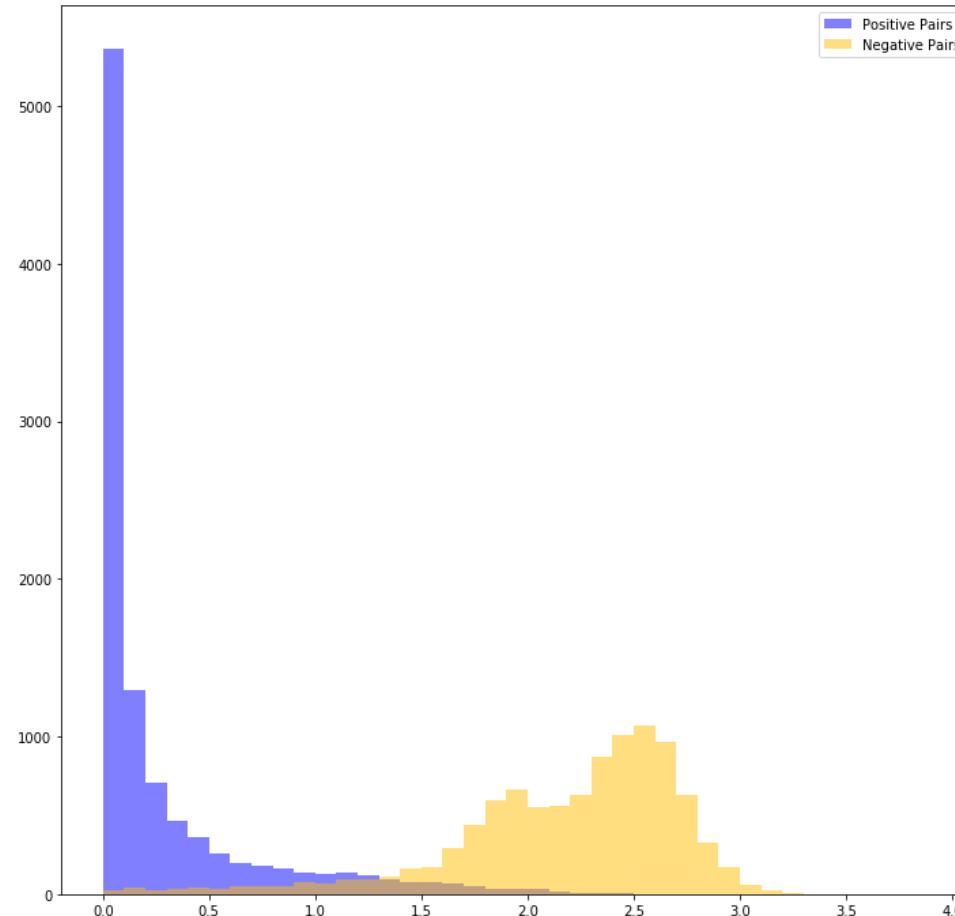
Some Predictions

- Visualising results for triplets
 - Negative images much further from the anchor than positive images



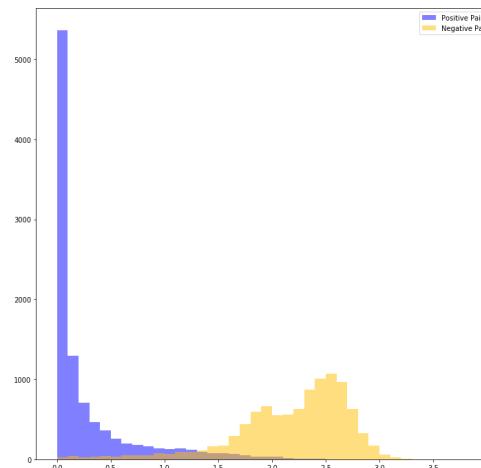
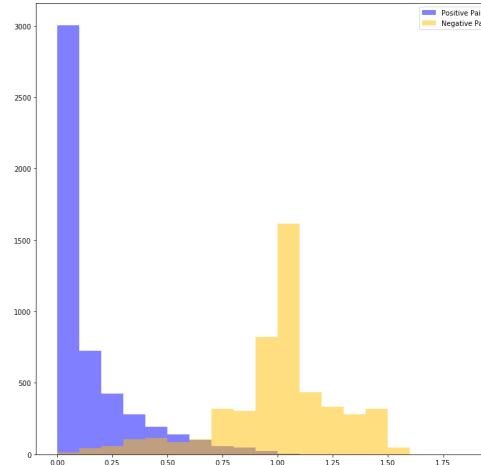
Distribution of Distances

- This plot shows
 - A histogram of distances for matched pairs
 - A histogram of distances for mismatched pairs
- Positive and Negative distributions well separated
 - Though with some overlap
- Negative distribution has a greater spread



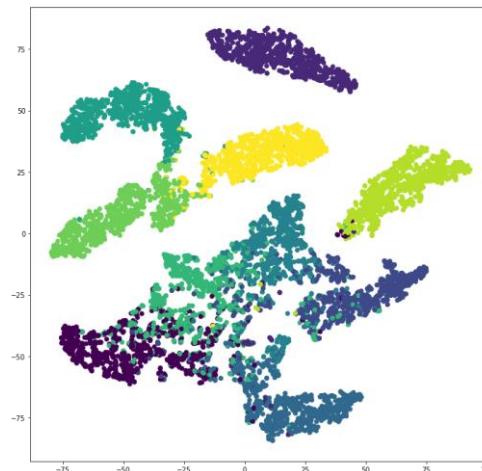
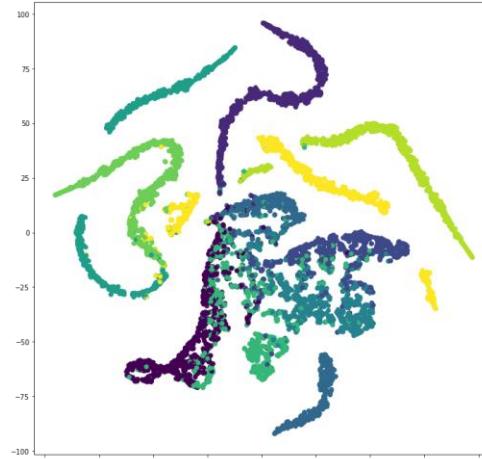
Comparing Distributions

- Top: Contrastive Loss
- Bottom: Triplet Loss
- Triplet loss leads to clearer separation
 - Mean of negative pair distances further from the mean of positive pair distances
 - Negative pair spread larger in both cases



Embedding Space

- t-SNE plots of the learned embedding
 - Top: Contrastive loss
 - Bottom: Triplet loss
- Similar embedding spaces
 - Slightly better class separation for triplet loss



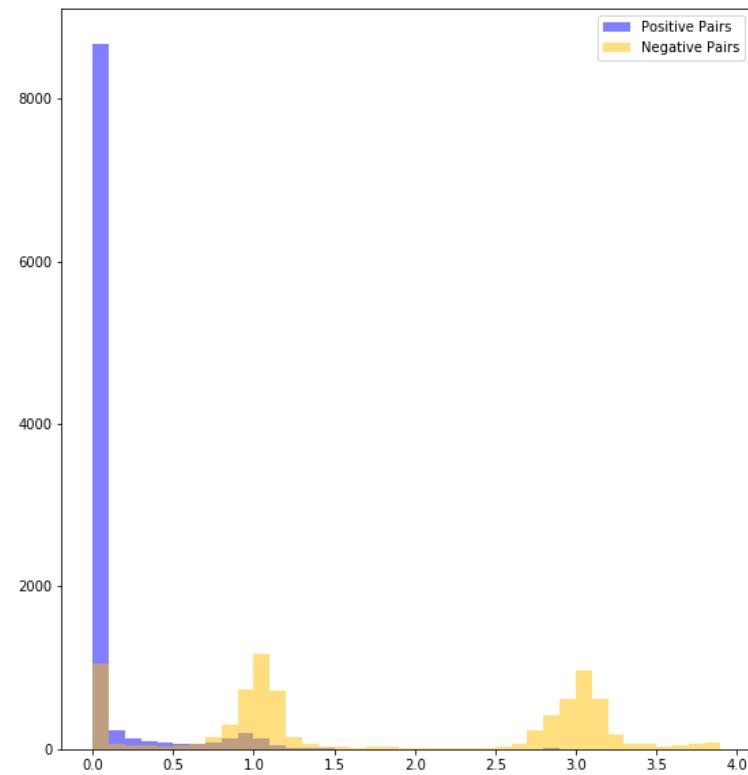
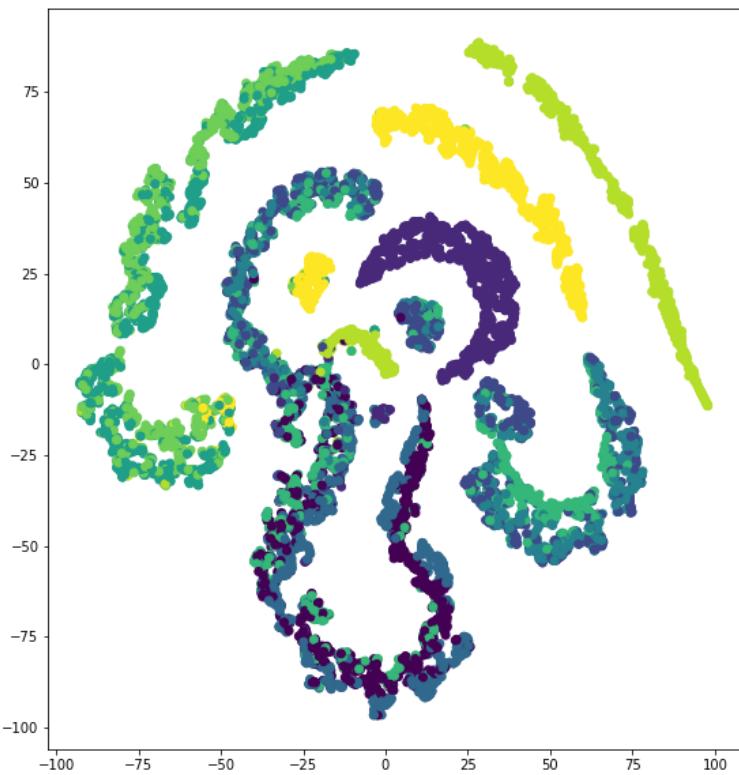
CAB420: Embedding Size

AND WHY 32 MAY NOT BE THE BEST CHOICE

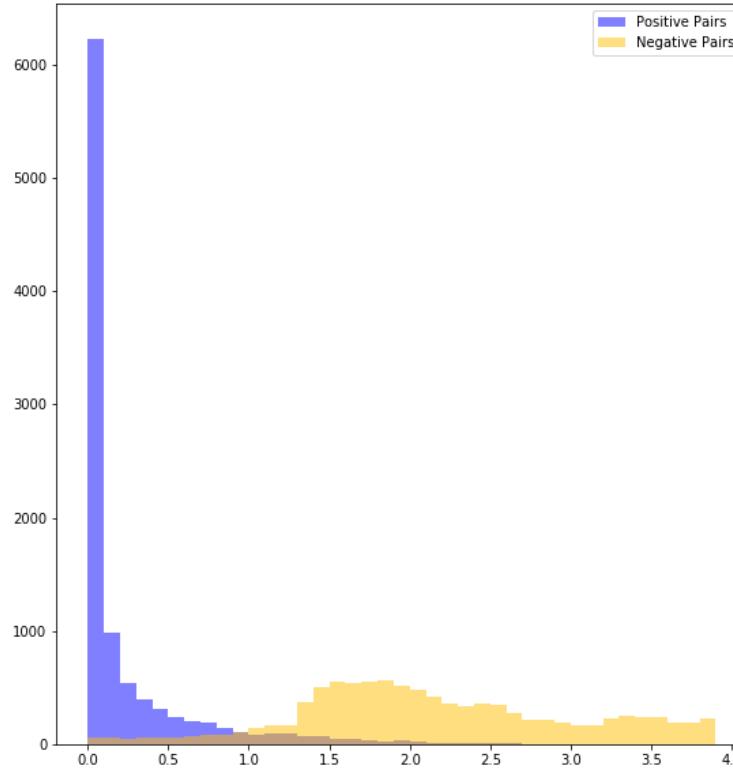
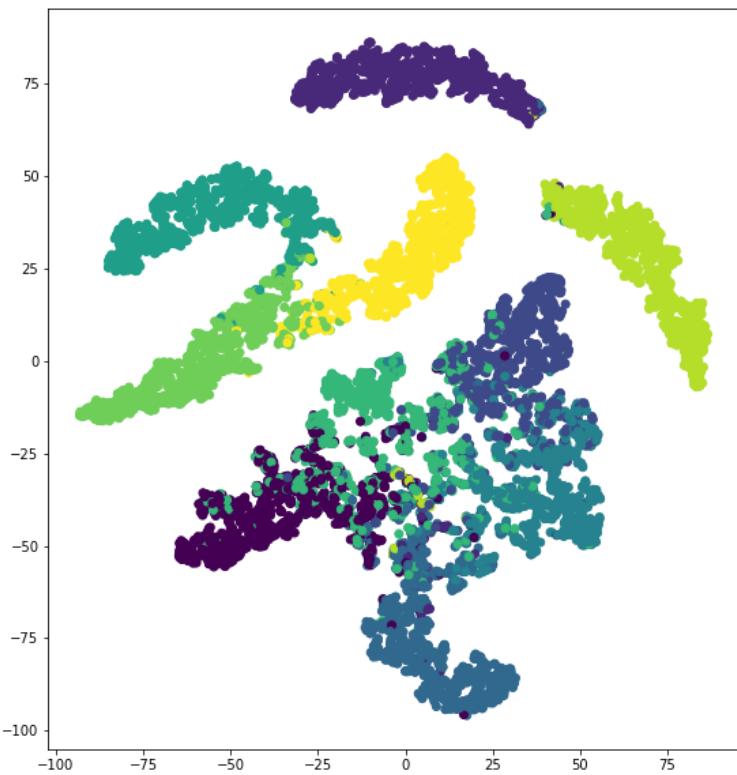
Embedding Size

- In all examples we have used an embedding size of 32
- What happens when this changes? We'll try
 - 2
 - 4
 - 8
 - 32
 - 128
- All using triplet loss, and the same backbone with Fashion MNIST
 - See ***CAB420_Metric_Learning_Additional_Example_EMBEDDING_SIZE.ipynb*** for code

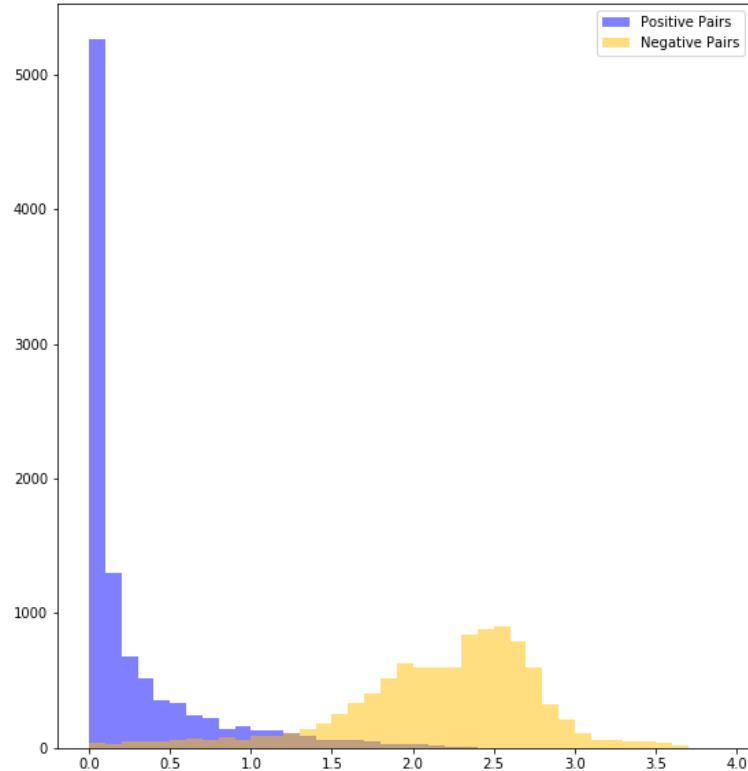
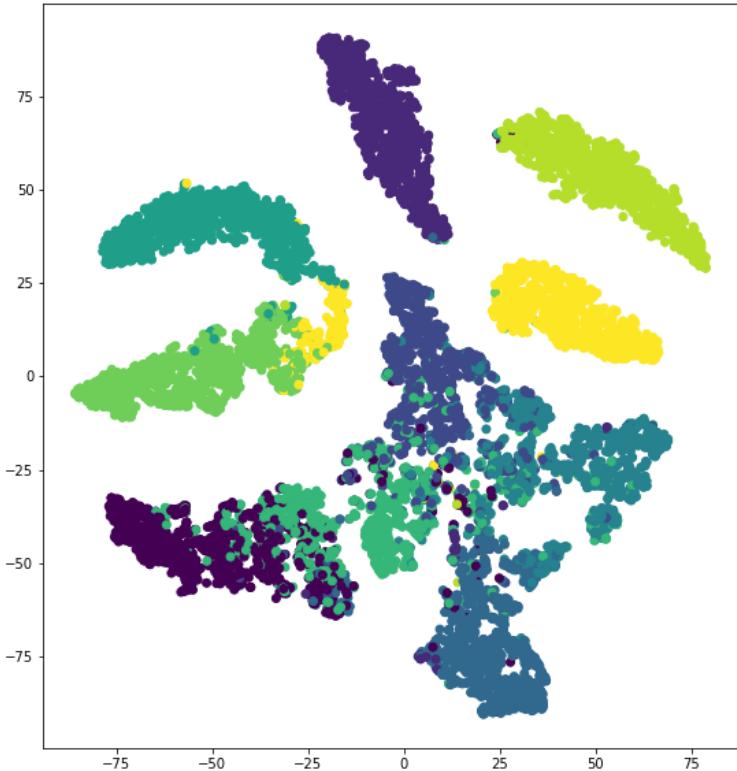
Embedding Size = 2



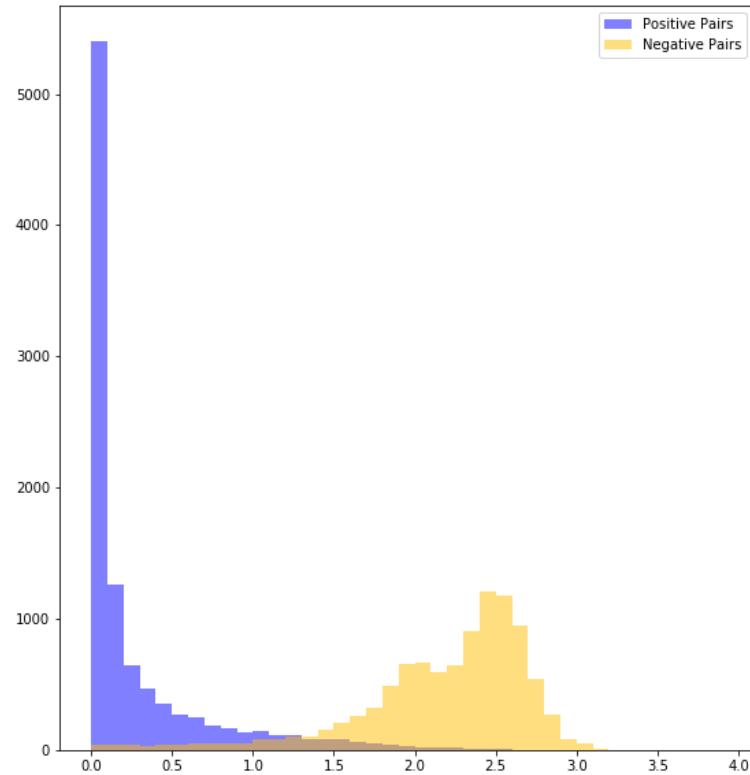
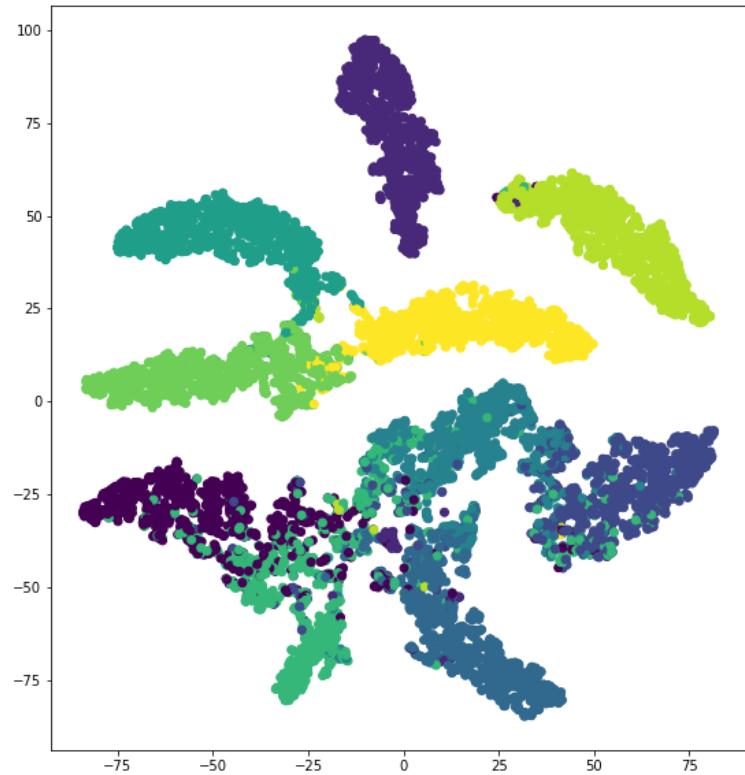
Embedding Size = 4



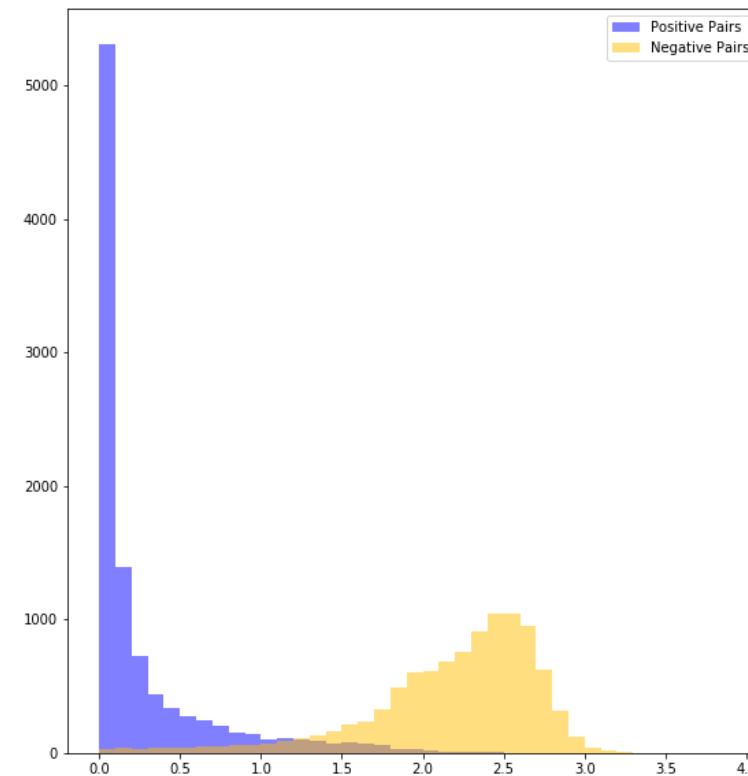
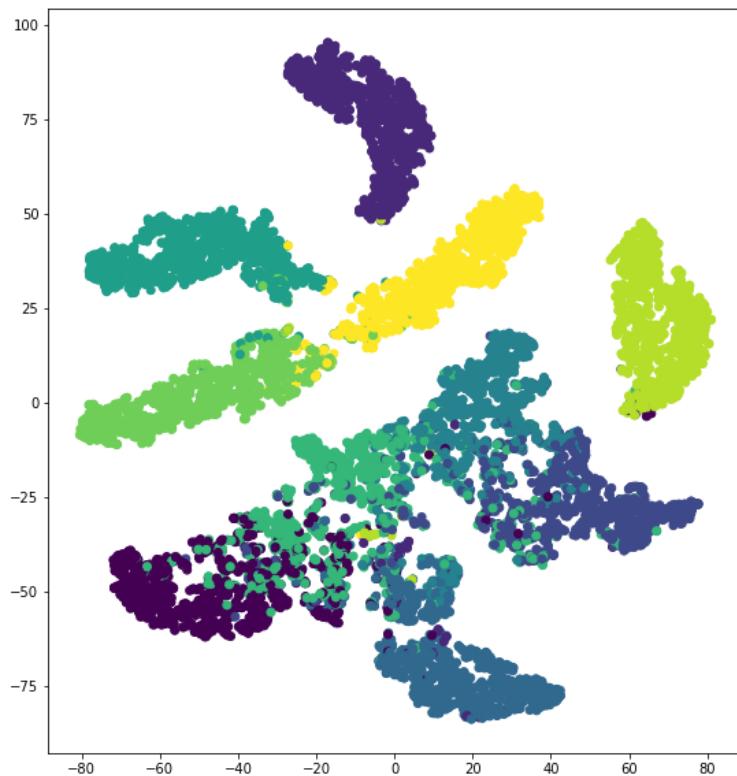
Embedding Size = 8



Embedding Size = 32



Embedding Size = 128



Embedding Size

- Smaller embeddings perform worse
 - 2 and 4 are noticeably worse than others
- Continued improvement as embedding size increases
 - Improvement slows as embedding gets larger
- Improvement gains for larger embeddings limited by
 - Backbone, we have the same backbone for all networks, and this can only capture so much detail
 - Data, we have 10 classes. Do we need 128 dimensions to separate these?
- Embedding size should be considered for each problem

Siamese Network Applications

WHAT DO I ACTUALLY DO WITH THIS?

Applications of Siamese Networks

- Biometric systems
 - Solve the problem of "are these two people the same"
 - Compare sets of embeddings to determine if a person is the same
- Content Based Retrieval
 - Find content like a provided sample, i.e. finding similar images
 - Compare the embedding for a given sample to a database of other content, returning the most relevant
 - Can order results by distance to return the most relevant results first

Why Siamese Networks?

- Extend better to unseen classes
- Consider a biometric system. We seek to enrol a new person:
 - For a Siamese network, we
 - Compute an embedding for their enrollment data
 - Add this to the existing database
 - For a classification DNN, where the network is trained to classify the identify, we
 - Modify the network to contain an additional output class
 - Re-train (possibly just fine-tune) the network

Summary Time

CATS, NARWHALS, AND LOSS FUNCTIONS

Comparing things with DCNNs

- We can replicate the idea of LDA with a DCNN
 - Supervised data required
 - No ability to map from learned feature back to the input space
- Use a DCNN as a feature extractor
 - Requires us to formulate a loss function to encourage
 - Samples from the same class to be close together
 - Samples from different classes to be far apart
- A naïve approach of binary cross entropy can do a fair job of verification, but
 - Does not provide great embeddings;
 - Won't provide a good ranking of similarity.

Comparing things with DCNNs

- Contrastive loss considers pairs of images
 - Tries to force similar pairs to be closer together than negative pairs by a margin
- Triplet loss uses three images and forms two pairs
 - Tries to make the positive pair closer together than the negative pair by a margin
- For both, we can use normalisation to simplify setting the margin

Comparing things with DCNNs

- What we ultimately compare is a learned embedding
 - Compact representation of the input feature
- We can control the size of the embedding
 - Hyper-parameter chosen at design time
 - Bigger embedding means a richer description
 - May take longer to train
 - May have severe impacts on run-time for some tasks
 - More complex problems (larger number of classes, greater variation) will require a larger embedding

CAB420: Clustering and K-Means

WHAT, WHY AND (ONE APPROACH FOR) HOW

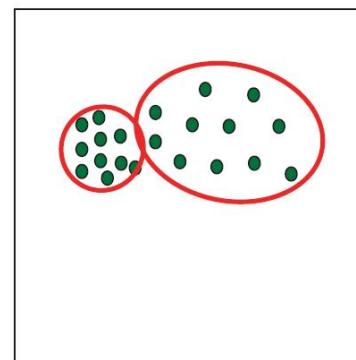
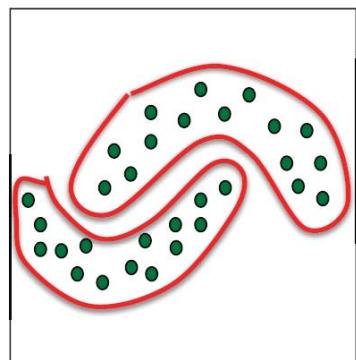
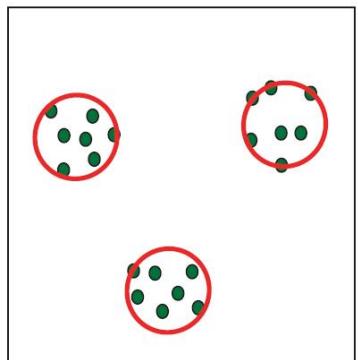
What is Clustering?

An unsupervised learning method

- Discover patterns in the data
- Group related points into groups

But

- What makes points related?



Clustering Data

Why cluster data?

- To simplify it
 - Go from thousands or millions of points to a small set of cluster centres
- To find patterns or relationships
 - Find points that are similar
- To find things that are abnormal
 - i.e. points that don't fit with the rest of the data

There are lots of clustering methods

- We'll look at K-means and GMMs first

K-Means

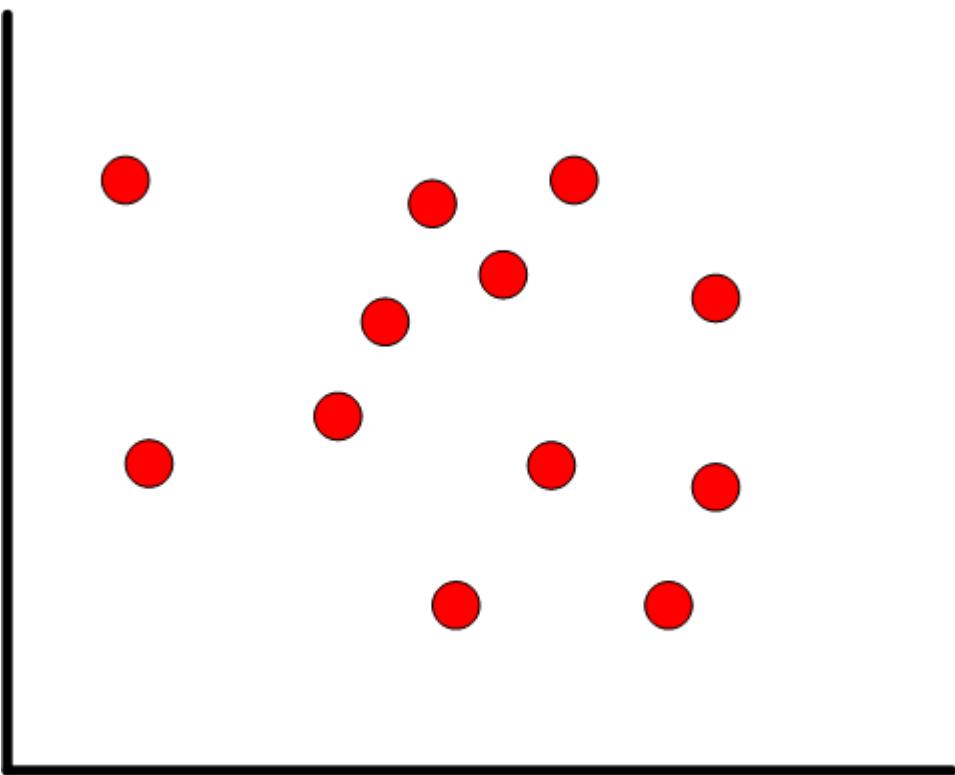
EXPLAINED

K-Means Clustering

- Starts with a dataset and a target number of clusters, K
- Aims to find a set of K clusters that minimises the distance between each point and its cluster centre

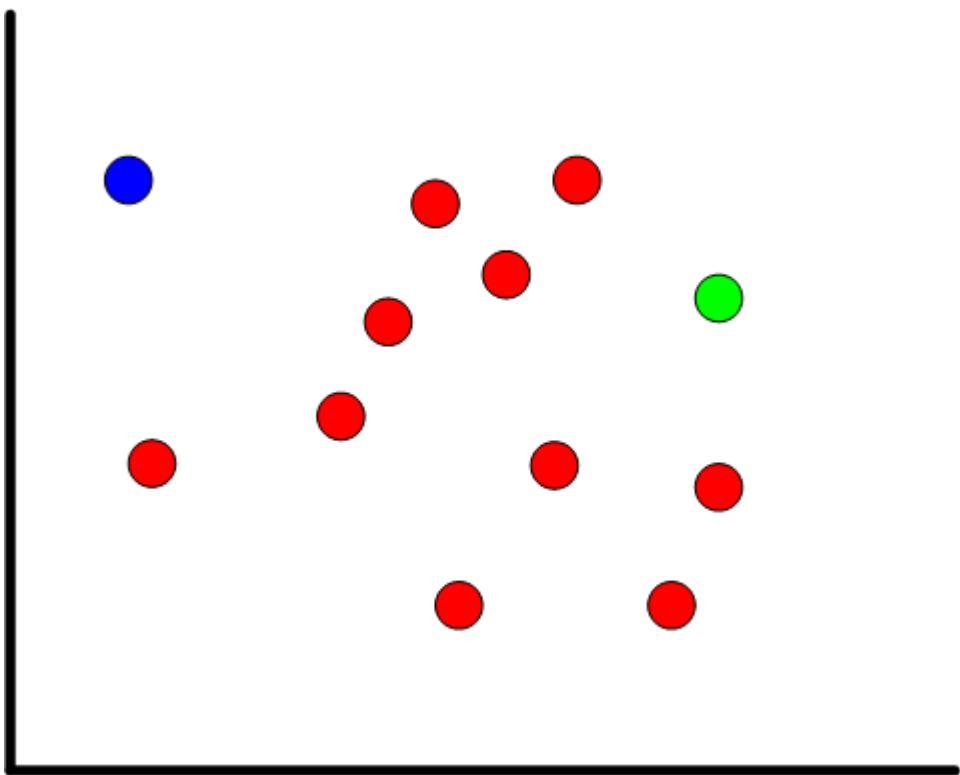
K-Means Clustering

- Start with a set of points, and a target number of clusters, K
 - $K = 2$



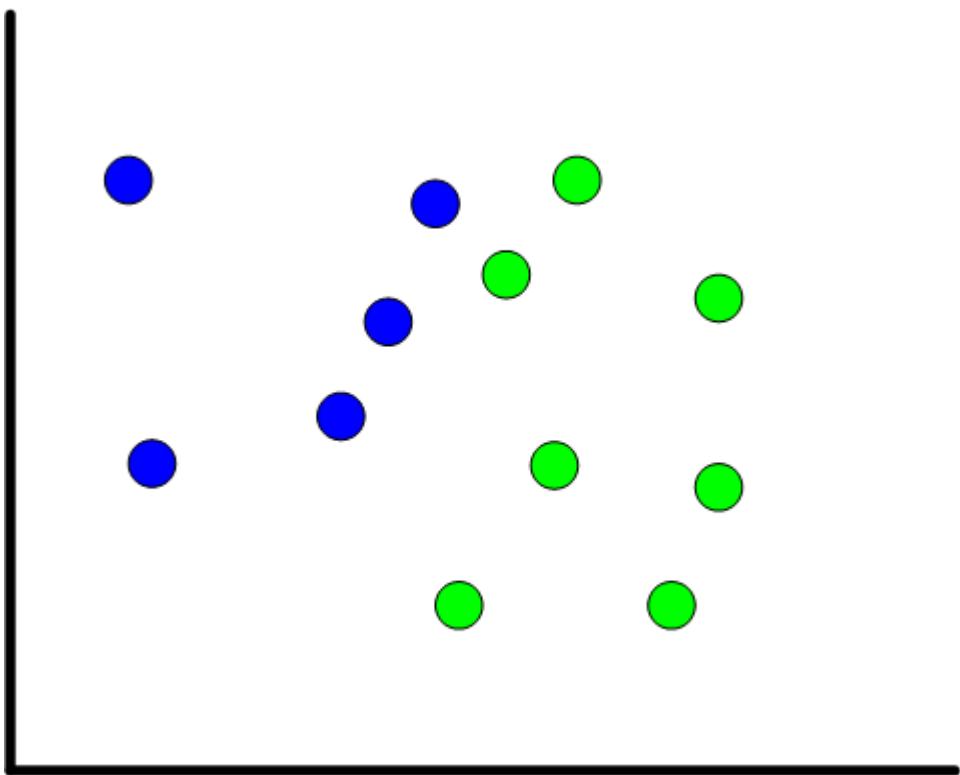
K-Means Clustering

- Pick two points at random to be our initial cluster centres



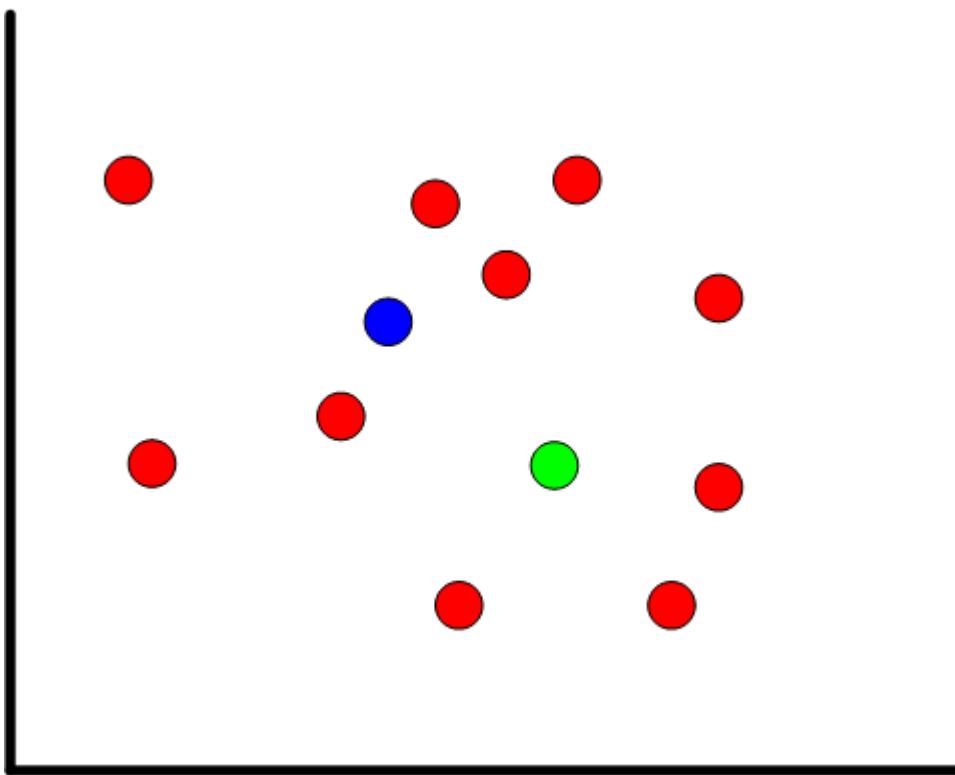
K-Means Clustering

- Assign every point to its nearest cluster centre



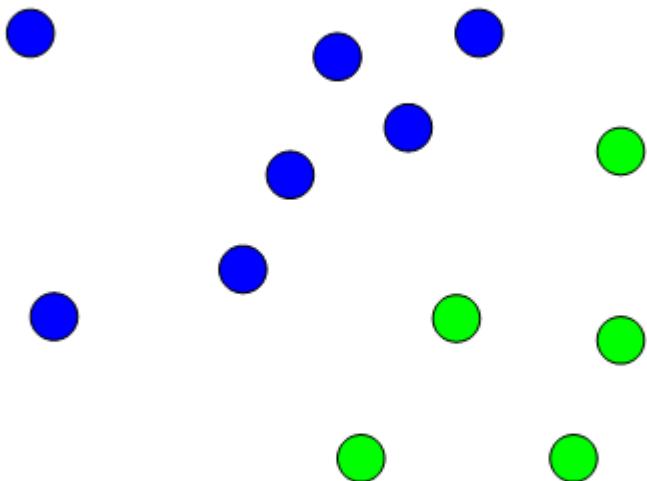
K-Means Clustering

- Calculate new centre points based on the initial clustering, and run again



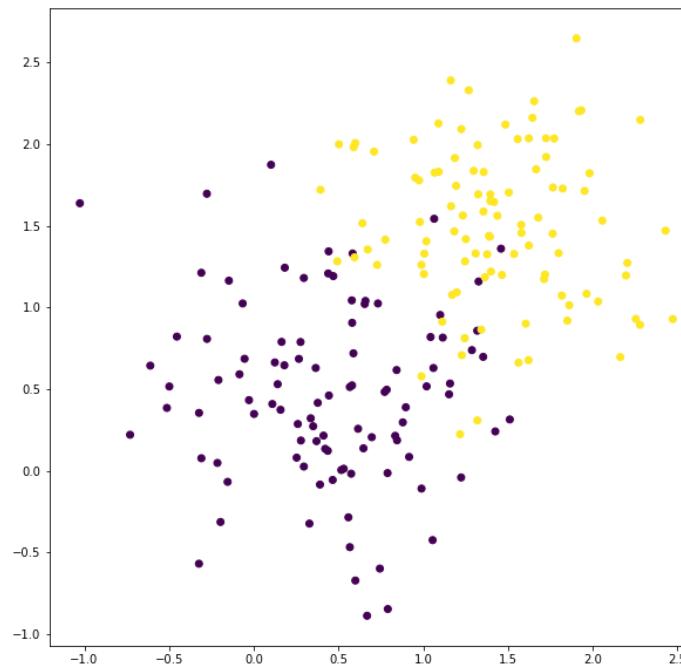
K-Means Clustering

- We keep doing this until
- The result has converged, i.e. it's stopped changing, or is only changing by a very small amount
- We reach a total number of iterations



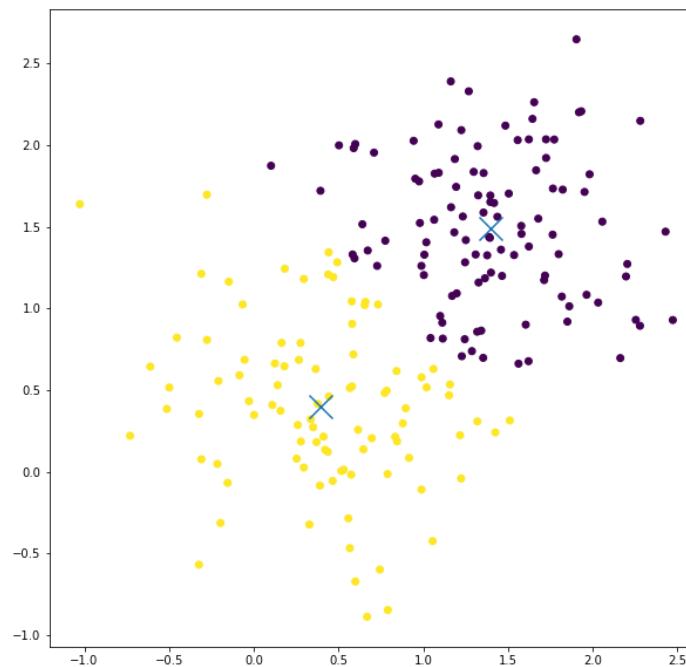
An Example

- See ***CAB420_Clustering_Example_1_KMeans_Clustering.ipynb***
- We'll cluster some random data
 - Data contains two actual clusters
 - True cluster centres are
 - $(0.5, 0.5)$, the first 100 points are in this cluster
 - $(1.5, 1.5)$, the second 100 points are in this cluster
 - Clusters have some overlap



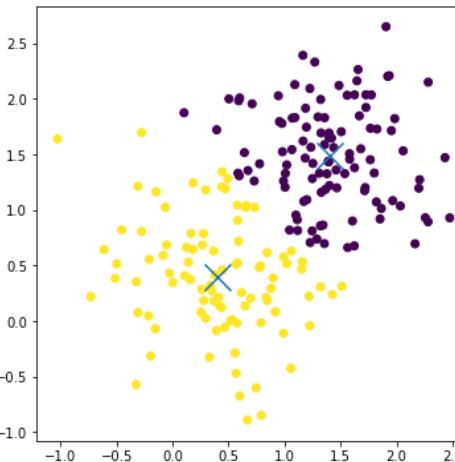
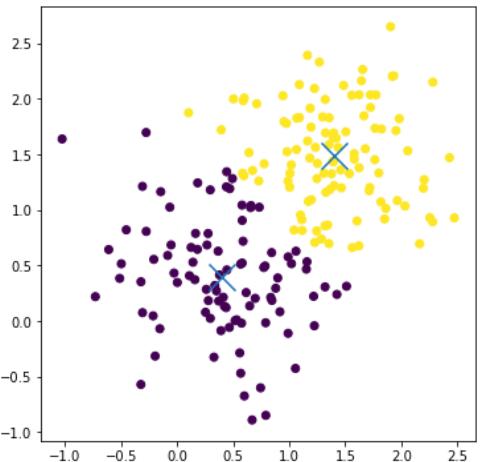
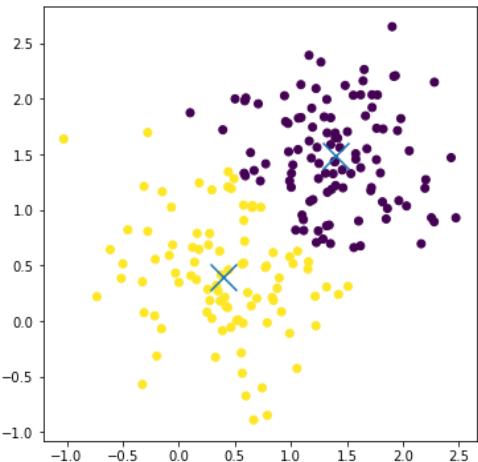
Clustering Results

- Cluster centres estimated as:
 - (1.4003926, 1.49235897)
 - (0.39658001, 0.39824299)
- Cluster assignment is fairly accurate
 - Estimated centres are close to true centres
 - Some points at the boundary are grouped into the "other" cluster
 - This is not necessarily an error or problem, and is expected in this case



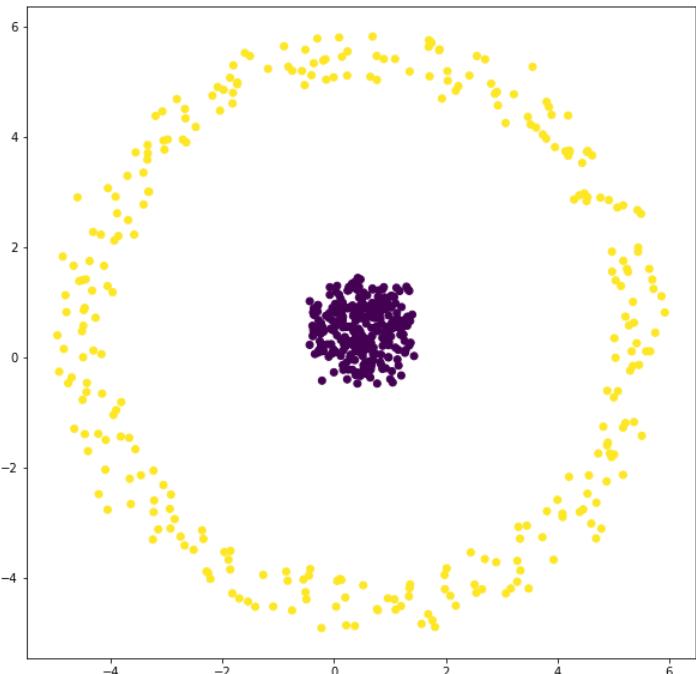
K-Means and Randomness

- Recall, K-Means starts from random initial cluster centres
 - Different starting points lead to different results
 - Differences are small in this case due to fairly simple data
 - Differences will be more pronounced for larger values of K and more complex data



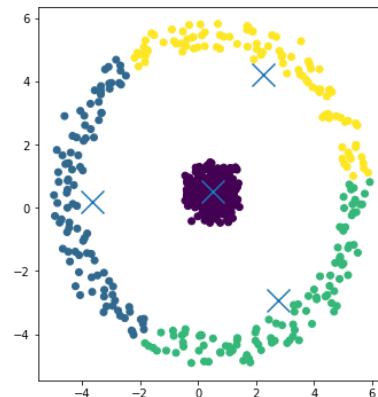
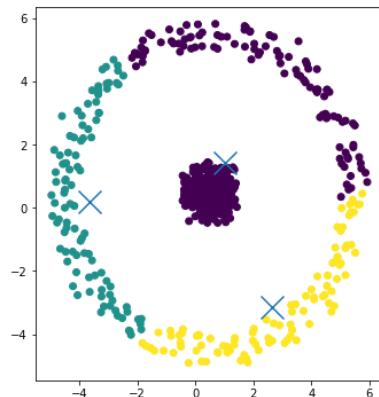
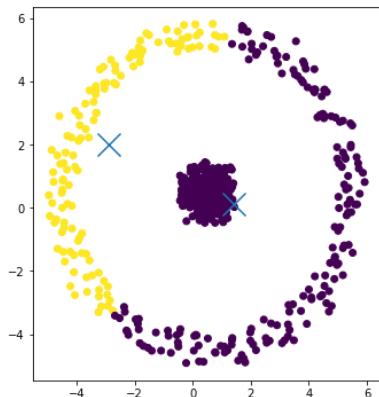
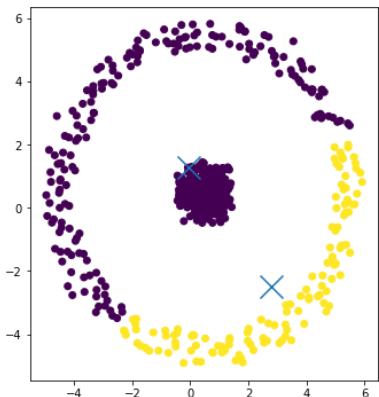
A Second Test Case

- Two true clusters again
 - Both with a true centre of $(0, 0)$
 - Clear physical separation between clusters



Clustering Results

- For K=2, we cannot recover the true clusters at all
- For K=4, we can recover the true centre cluster, but the outer ring is broken into three clusters
 - Over-clustering



Things to Consider

WITH K-MEANS IN PARTICULAR, NOT IN GENERAL

K-Means and Randomness

The initial cluster centres are random

- Or semi-random in the case of K-Means++

This means:

- Different runs may give us different results
- Differences will become more pronounced with
 - Bigger, more complex datasets
 - Fewer iterations
 - More clusters

K-Means and Cluster Shapes

- K-Means extracts spherical clusters
 - Or circular in a 2D case
- Clusters will typically all have similar shapes and sizes
- Clusters cannot overlap
 - K-Means uses hard assignment, a point either belongs to a cluster, or does not

Distance Metrics

- We can cluster any type of data
 - Just need to define a way to calculate the distance between points
- Common Metrics
 - Euclidean distance (L2)
 - Manhattan distance (L1)
 - Cosine distance (angle between points)
 - Hamming Distance (used for binary data)
- Distance metrics can have a big influence on data
 - Use the right metric for your data
- Python (sklearn) is very limited in what distance metric you can select
 - pyclustering may be worth considering if you need to change metrics

k-Means Distance Scaling and Standardisation

- The distance metrics have an important note attached to them; comparable distance.
- If a dimension of our data has a large scale, the results of our clustering may be completely dominated by that scale.
- In cases where this is apparent, the data must be scaled in certain situations to ensure that no one variable controls the clustering.
- This is done through normalisation of data:
 - Scaling: Changing the minimum data point to 0 and maximum to 1, or
 - Standardisation: Changing the mean to zero and standard deviation to 1.

k-Means Drawbacks

- Remember that no clustering method is the best by default.
- k-means clustering is restricted to roughly **spherical** clusters.
- Consider a 'ring' of data around a small cluster of data. Another method can deal with clusters like this.
- k-means clustering restricts objects to **one specific cluster**.
- If a point is on the border between two clusters, there is a chance it could belong to either. Another method can deal with cases like this.

Gaussian Mixture Models (GMMs)

ALSO EXPLAINED

K-Means and ‘Hard Decisions’

With K-Means, each point is assigned to one cluster

- What if the point is at the boundary of two clusters?
- Wouldn’t it be better to say the point is 51% cluster 1 and 49% cluster 2?

Gaussian Mixture Models

- A Gaussian mixture model assumes that each cluster has its **own** normal (or Gaussian) distribution with parameters μ_c and σ_c .
- Each cluster has a **weight**, π_c , included to prioritise certain clusters
 - Clusters with a higher weight have more points in them, or are more likely
- We can instead calculate the probability that a point belongs to a certain cluster.
- This probability is based on several parameters
 - The mean of each cluster, μ_c
 - The covariance between variables, Σ_c
 - The weight term, π_c
- We can find the optimal parameters for this model using **maximum log likelihood estimation**.

Log Likelihood Function

- From your regression knowledge, it is clear that different values of **parameters** lead to different data **predictions**.
- In statistics, and in our case machine learning, we would like to know the distribution that most likely gave the data points we are observing.
- These distributions rely on parameters, such as mean and variance, much like regression models rely on slopes and intercepts.
- Maximum likelihood estimation is a method that will allow us to calculate distribution parameters that dictate the shape of the distribution.

Log Likelihood Functions

- If each data point is generated independently of the others, then the total probability of observing our data is the product of the probability of observing each single data point separately.
- Thus, the likelihood function is given by:

$$P(x|\theta) = \prod_{i=1}^n \Pr(x_i)$$

where θ is the set of parameters in a distribution and $\Pr(x_i)$ is the **probability mass function** of the distribution which gives our data points.

- These expressions can be difficult to differentiate (which is necessary to optimise), so we take the **log** of the function to simplify the product:

$$L(P(x|\theta)) = \sum_{i=1}^n \log(\Pr(x_i))$$

Log Likelihood Functions

- Once we simplify the log likelihood function, we **differentiate it** with respect to each of the parameters.
- In order to find the **maximum** likelihood estimate for each parameter, we set the derivative to 0 and solve.
- In machine learning, typically optimisation requires finding minimum values (i.e., minimising errors).
- We use **negative log likelihoods** (NLLs);

$$NL(P(x|\theta)) = -L(P(x|\theta))$$

as a way to ensure that this process will always result in a minimum.

- If we set out to **minimise** the likelihood estimator, we will now find the **actual maximum**, rather than some local minimum likelihood estimate which would be the opposite of our goal!

Gaussian Mixture Models

- The probability mass function for a data point in a Gaussian mixture model is given by:

$$\Pr(x) = \sum_{c=1}^K \pi_c N(x|\mu_c, \Sigma_c)$$

where $\sum_{c=1}^K \pi_c = 1$, $0 \leq \pi_c \leq 1$, and x has multiple dimensions based on the terms in the clustering

- The negative log likelihood that we must minimise for optimal parameters is

$$-\log(\Pr(x|\pi, \mu, \Sigma)) = -\sum_{i=1}^N \log\left(\sum_{c=1}^K \pi_c N(x|\mu_c, \Sigma_c)\right)$$

- Difficult to solve due to the sum inside the logarithm.
- Optimisation is instead achieved using an iterative technique: the expectation maximisation (EM) algorithm

Learning a GMM

Expectation-Maximisation (EM) Algorithm

- Iterative, two step process
- Expectation Step
 - Determine likelihoods for each sample for current model
- Maximisation Step
 - Update model parameters

Learning a GMM – Starting Conditions

We need an initial set of clusters

- Use K-means to create an initial clustering result
 - Mean μ_c
 - Covariance Σ_c
 - Weight (or size) π_c

Good initialisation is important

- EM will converge to a maximum
- Need to a good initialisation to avoid getting stuck in a local maximum

Learning a GMM - Expectation

For each data point

- Compute r_{ic} , the probability that it belongs to cluster c
- Normalise to sum to 1

$$r_{ic} = \frac{\pi_c N(x_i; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} N(x_i; \mu_{c'}, \Sigma_{c'})}$$

- Points with a good fit to a mode will have a high weight

Learning a GMM - Maximisation

Start from the assignment probabilities, r_{ic}

- Update model parameters: μ_c, Σ_c, π_c

For each Gaussian (cluster):

- Update parameters

$$m_c = \sum_i r_{ic}; \pi_c = \frac{m_c}{m}$$

$$\mu_c = \frac{1}{m_c} \sum_i r_{ic} x^i$$

$$\Sigma_c = \frac{1}{m_c} \sum_i r_{ic} (x^i - \mu_c)^T (x^i - \mu_c)$$

Learning a GMM

Each EM step increases the accuracy of the model

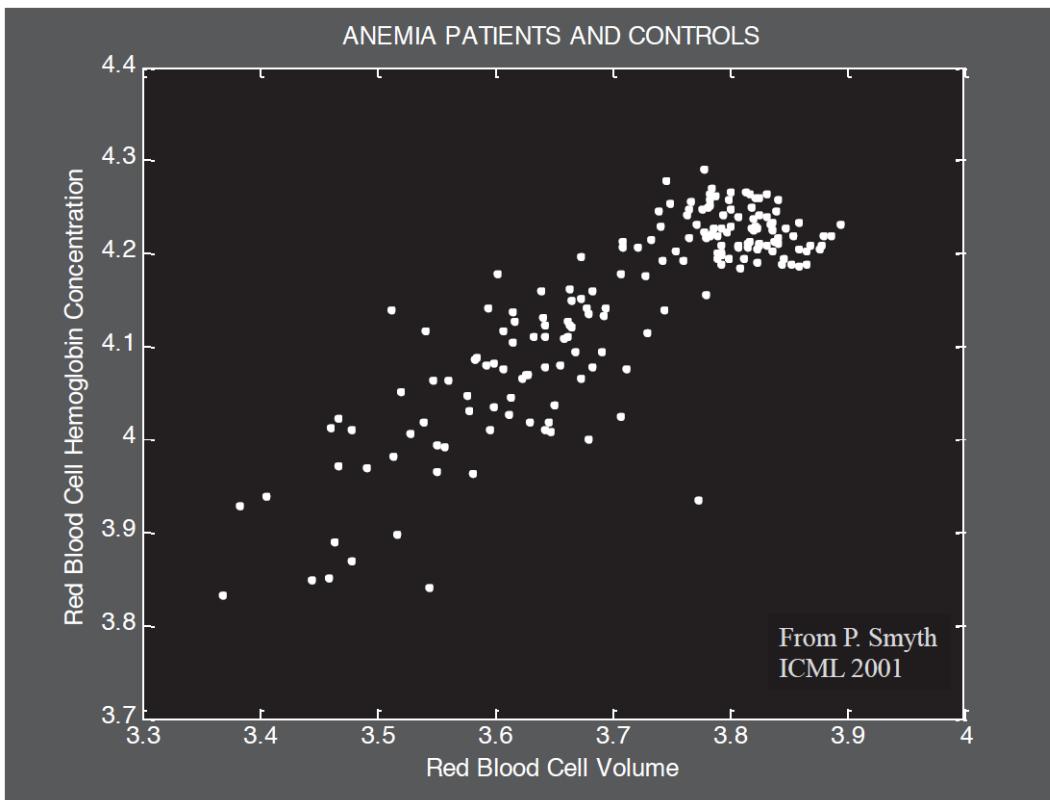
- Increases the log-likelihood

Iterate until convergence

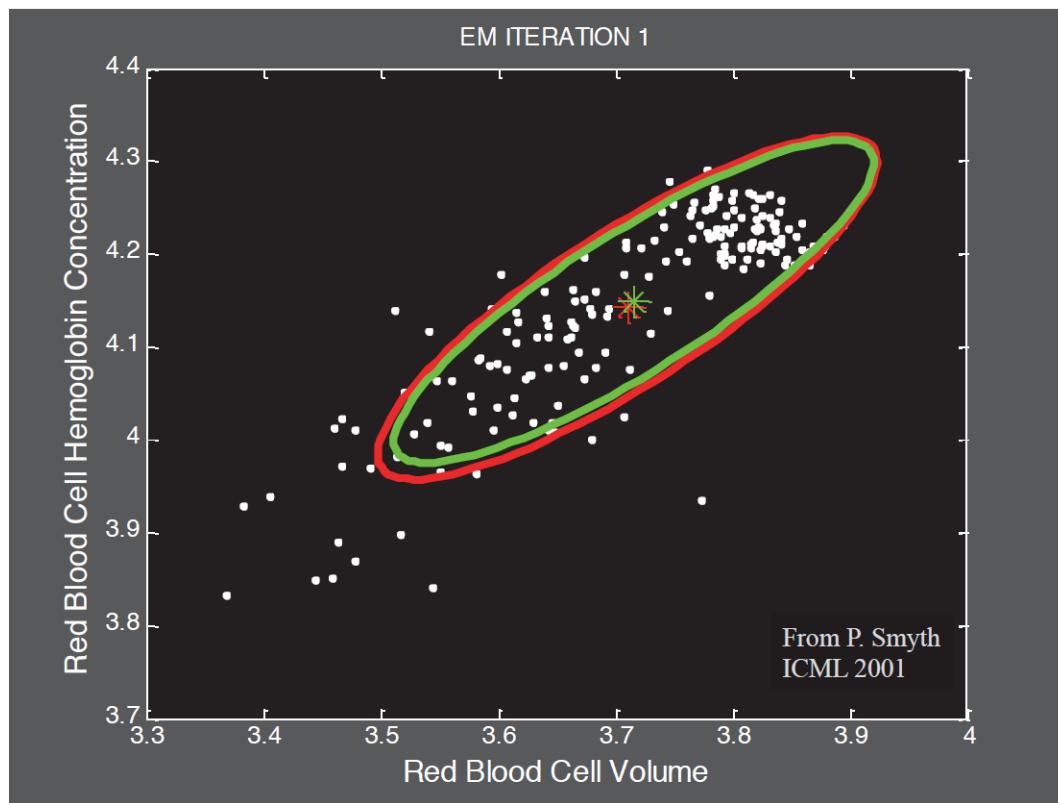
- It will converge

EM - Example

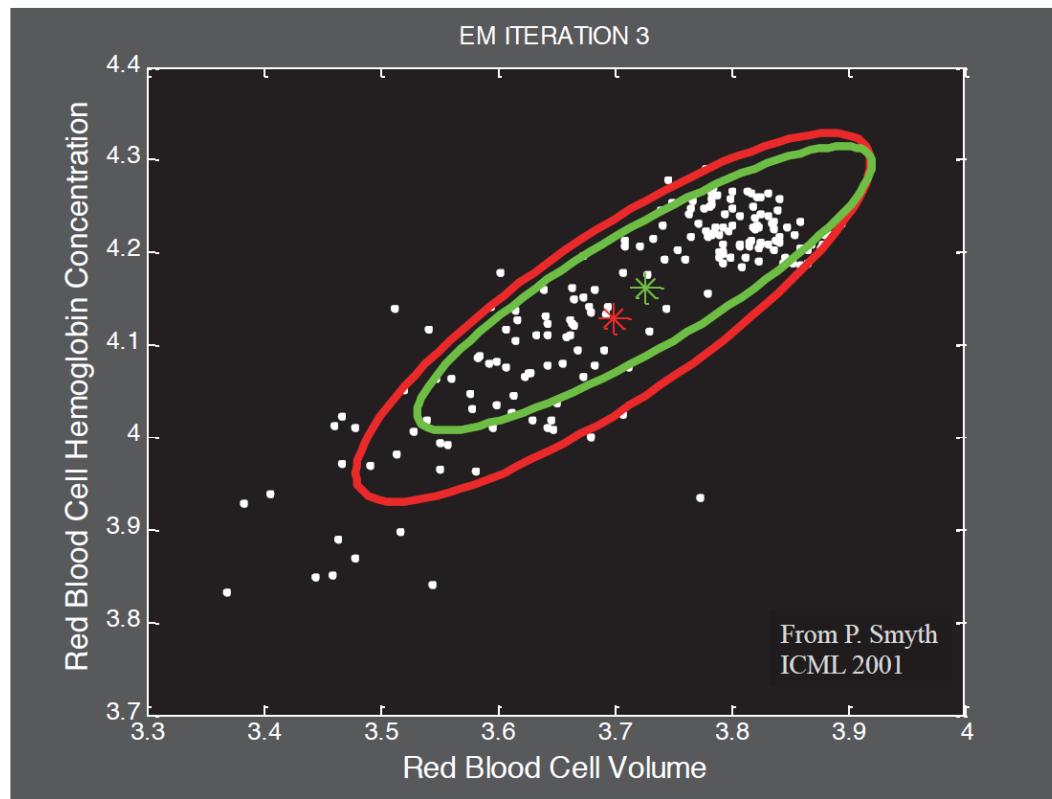
- Fit a GMM with two mixtures to this data



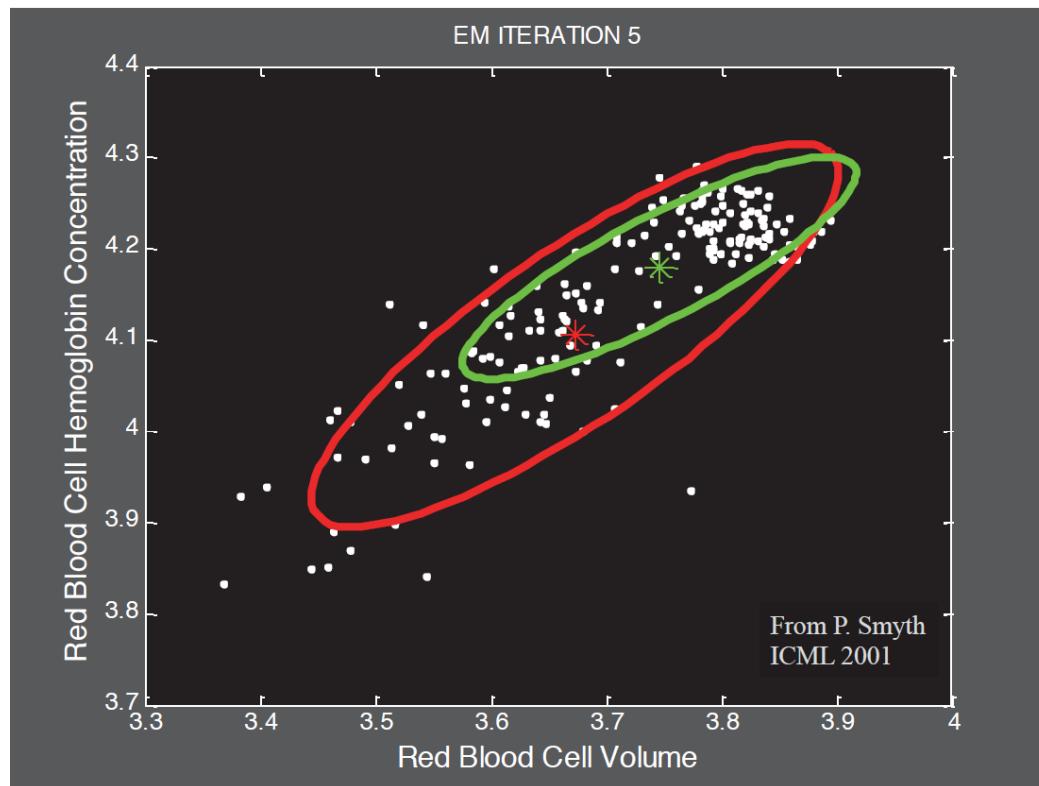
EM - Example



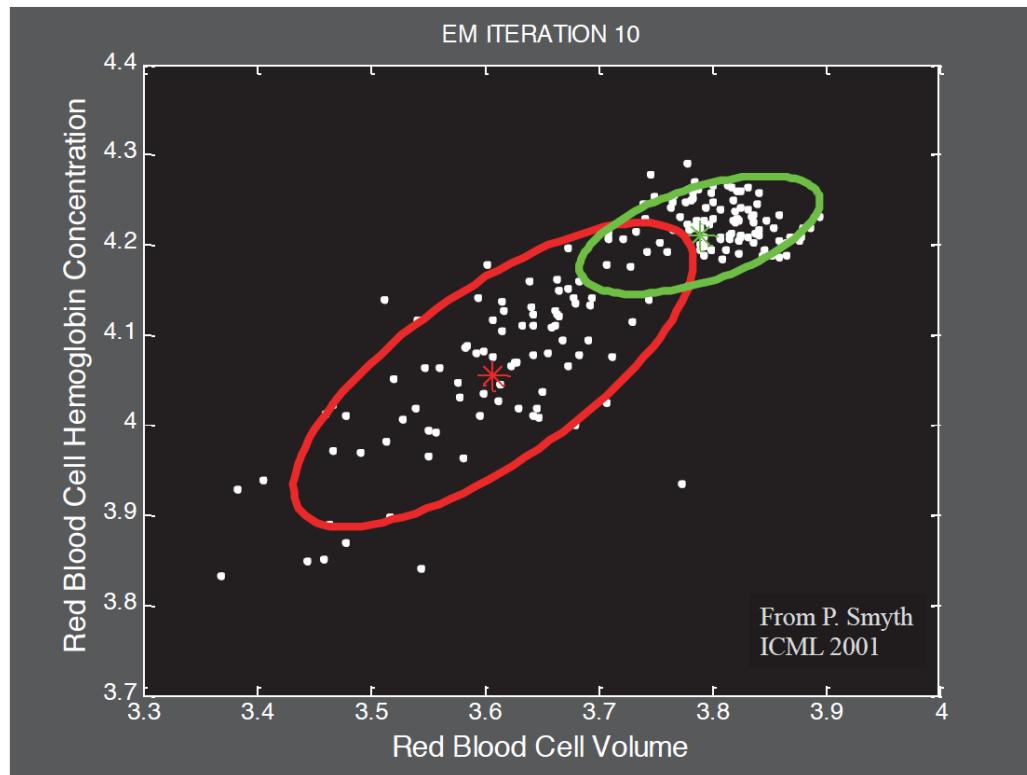
EM - Example



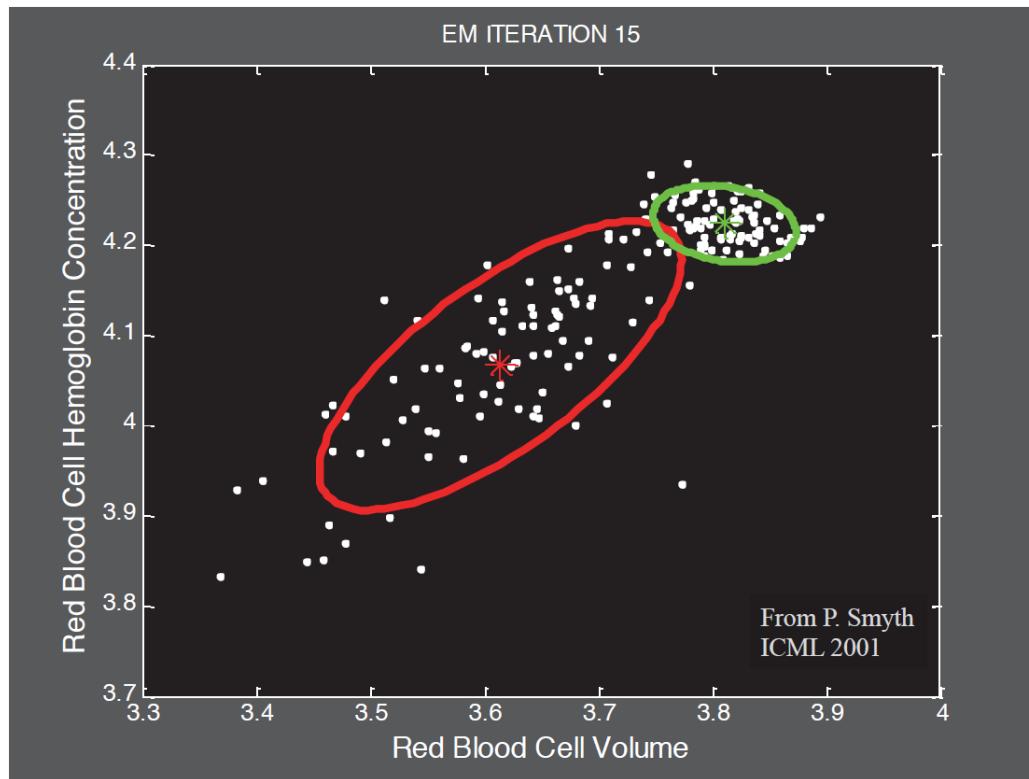
EM - Example



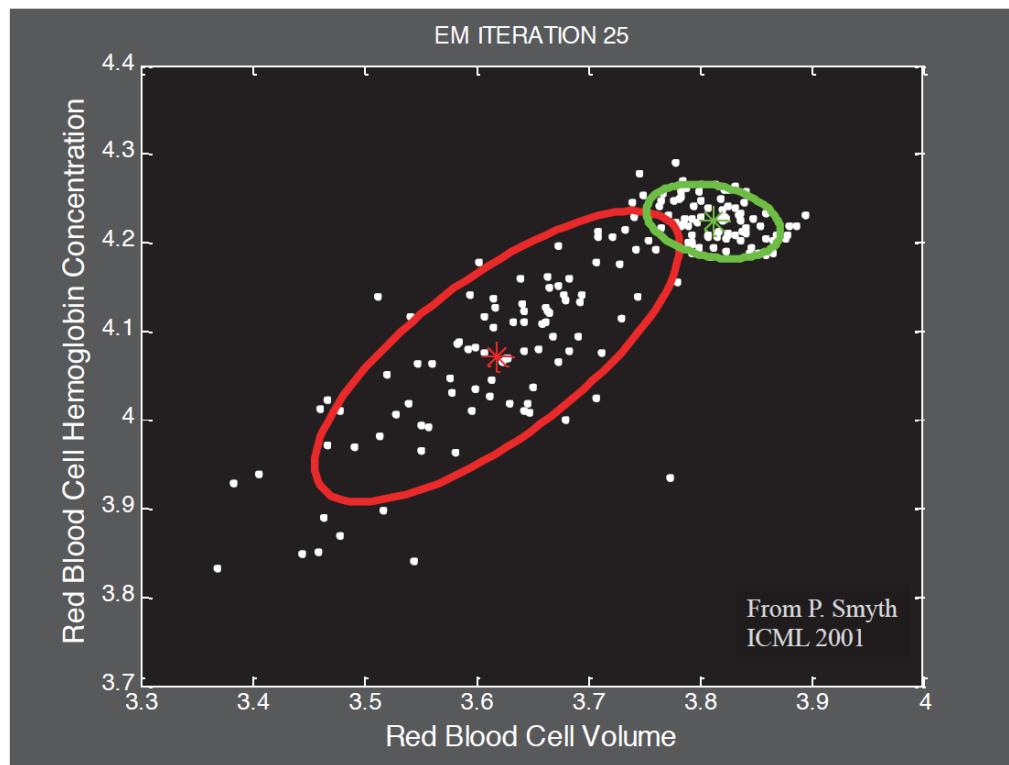
EM - Example



EM - Example



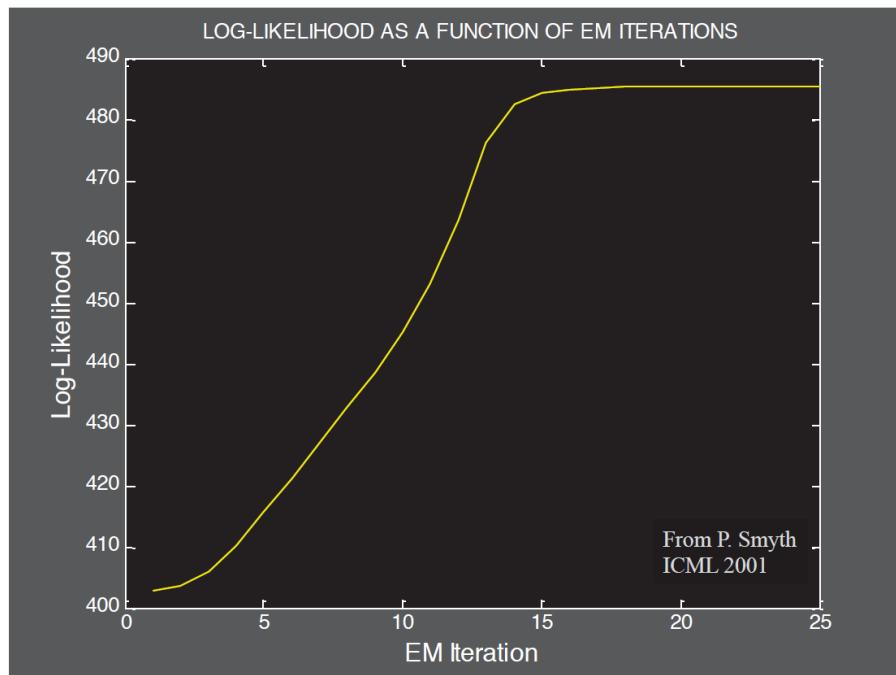
EM - Example



EM - Example

Fit has converged

- Changes fast early
- Changes slower as the model nears the optimum

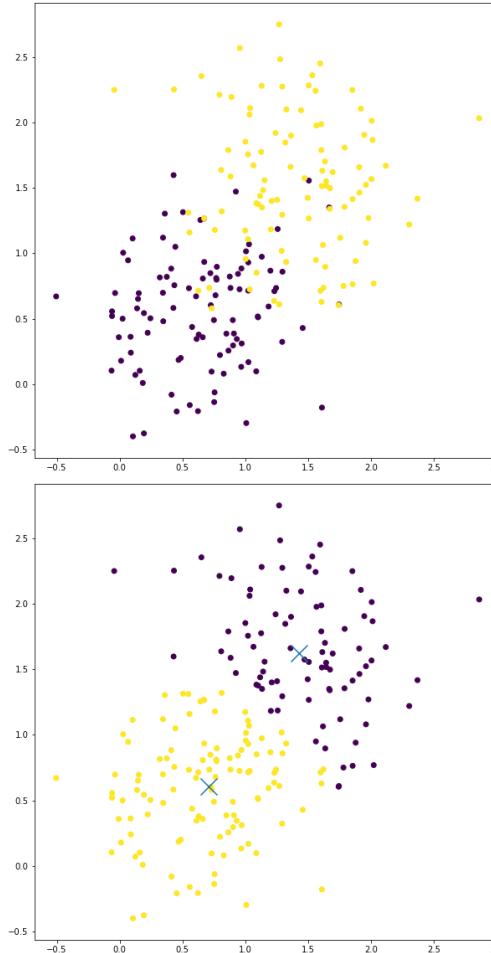


An Example

- See ***CAB420_Clustering_Example_2_Gaussian_Mixture_Models.ipynb***
- Same data setup as K-Means example

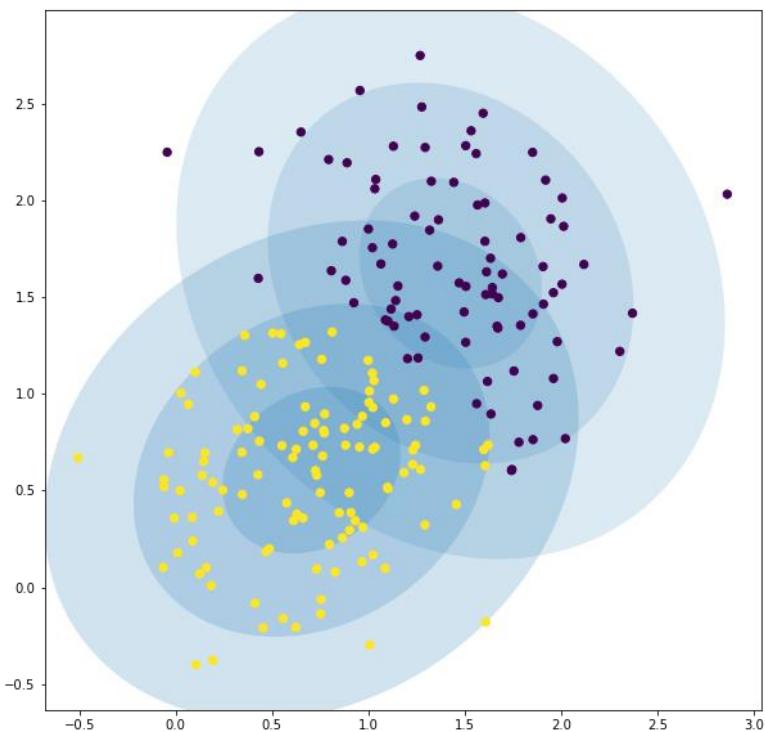
Clustering Results

- Top: original data
- Bottom: GMM Results
- Cluster centres
 - $(1.42573618, 1.62444572)$
 - $(0.70451705, 0.60575063)$
- Cluster weights:
 - 0.43276892
 - 0.56723108
- Centres are close to true centres and weights are fairly even
 - This makes sense, we have 100 points in each true cluster



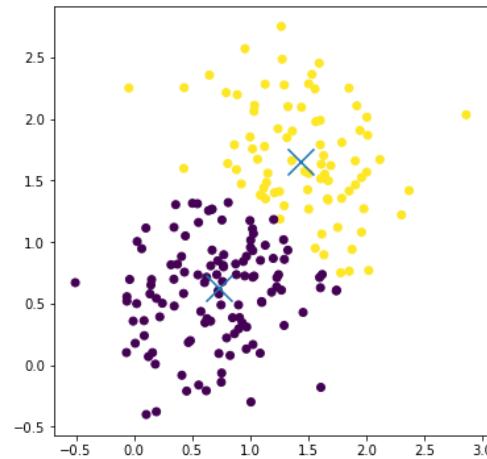
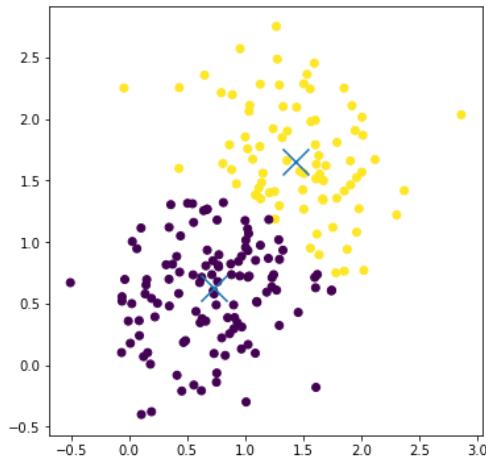
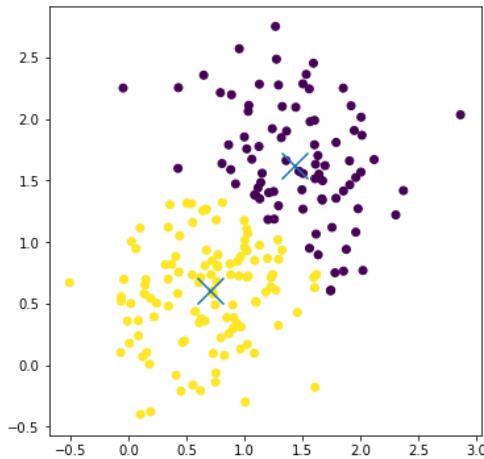
GMMs and Soft Decisions

- GMMs don't have hard boundaries between clusters
 - Clusters can overlap
 - Points can belong partially to both clusters



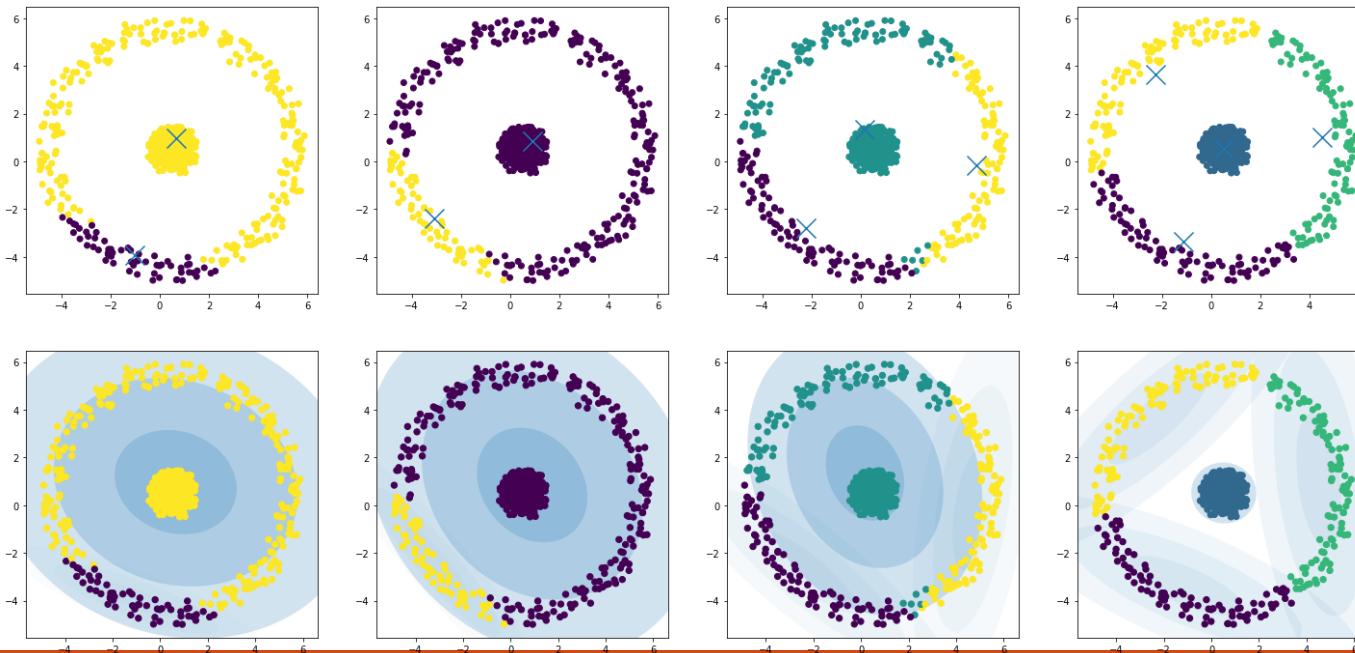
GMMs and Randomness

- GMMs need an initial set of cluster centres
- Initial centres come from K-Means, which is impacted by randomness
 - Has a knock-on effect for the GMM
 - Variation typically less severe than with K-Means



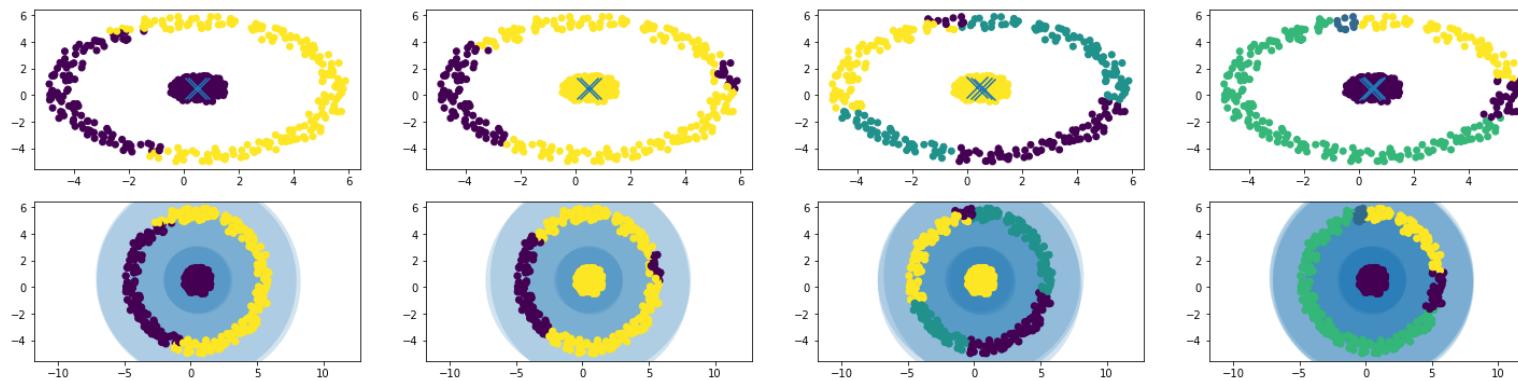
A Second Test Case

- Overlapping clusters
 - Using K-Means for initialisation
 - Cannot separate data with K=2
 - Similar results to K-Means overall



Random Initialisation

- We can initialise the GMM differently
 - Using other clustering results, our own estimates, or randomly
- We get several clusters with very similar centres
 - But different shapes, sizes and densities
- With careful initialisation, it would be possible to separate these clusters



CAB420: How Many Clusters?

AND THE DARK ARTS OF MODEL SELECTION

How do we select the number of clusters?

Is more clusters always better?

- Depends on our error measures

K-means cluster assignment cost

- $C(\underline{z}, \underline{\mu}) = \sum_i \|x_i - \mu_i\|^2$
- Will decrease as clusters increase
 - More cluster centres, so on average points will be closer to a centre
- Need to add a penalty for model size

Bayesian Information Criterion

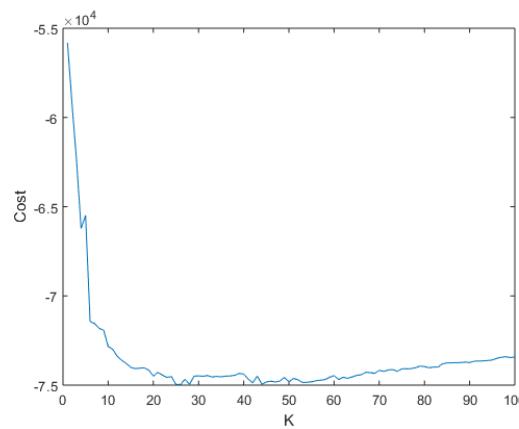
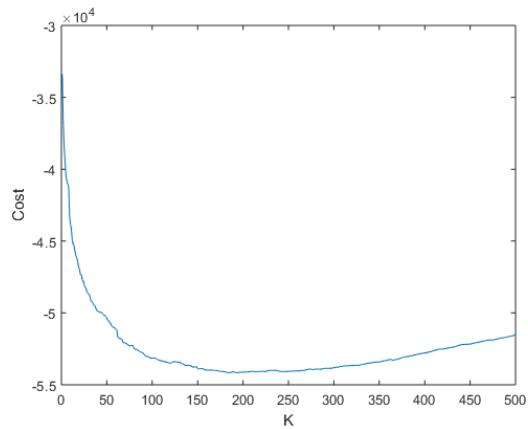
BIC

- Captures how informative a model is while also considering complexity
- Approximate form needed for K-means
- $J(\underline{z}, \underline{\mu}) = m \log \left(\frac{1}{m} \sum_i \|x_i - \mu_i\|^2 \right) + k \log m$
 - m = number of samples
 - k = size of the model (number of parameters)
- $k \log m$ will increase with model complexity, first term must decrease by enough to make the extra parameters “worth it”

Optimum Number of Clusters

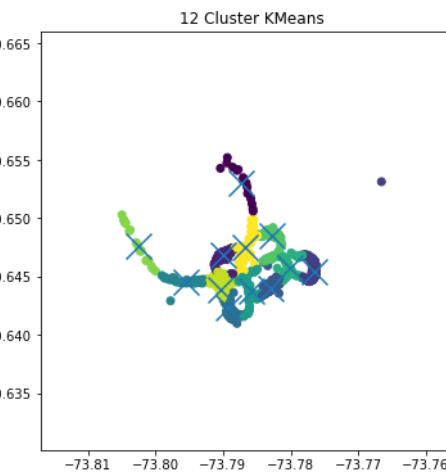
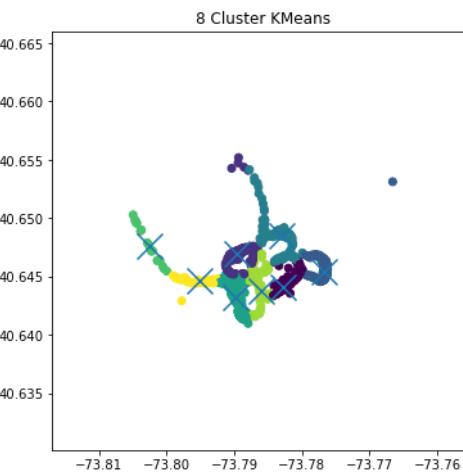
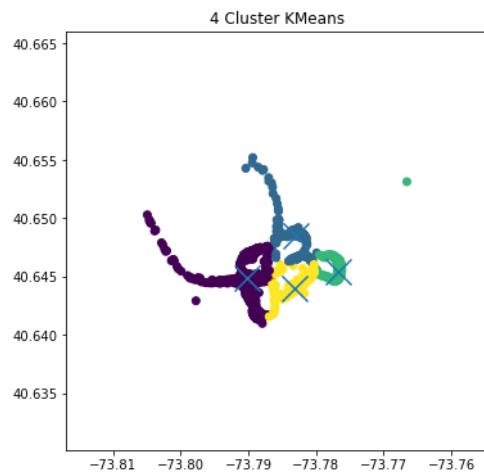
Not the same for K-means and a GMM. Why?

- Different number of parameters
 - GMM has many more parameters
- Approximations in K-means BIC formulation
 - Impacts accuracy



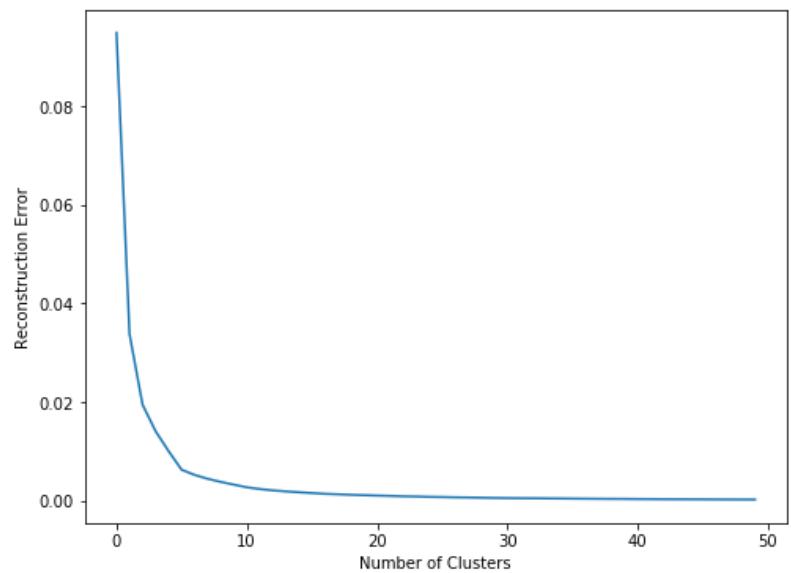
An Example

- See ***CAB420_Clustering_Example_3_How_Many_Clusters.ipynb***
- Our data
 - New York taxi data
 - Focus on drop-off locations around JFK airport



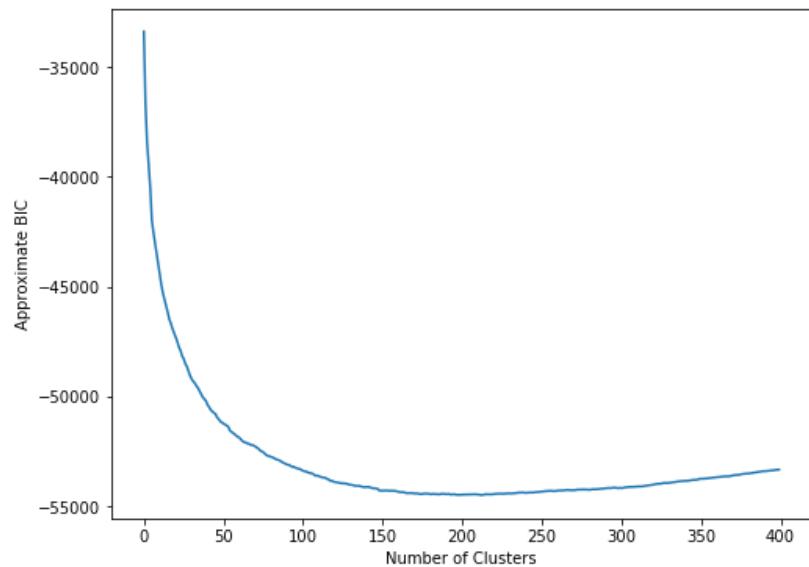
K-Means: Selection of K

- Reconstruction error
 - Average distance to assigned cluster centre
- As K increases, error decreases
- Can use the "elbow" point of the curve as a metric to choose K
 - ~5 in this case
 - Very much a heuristic, but not a bad one all the same



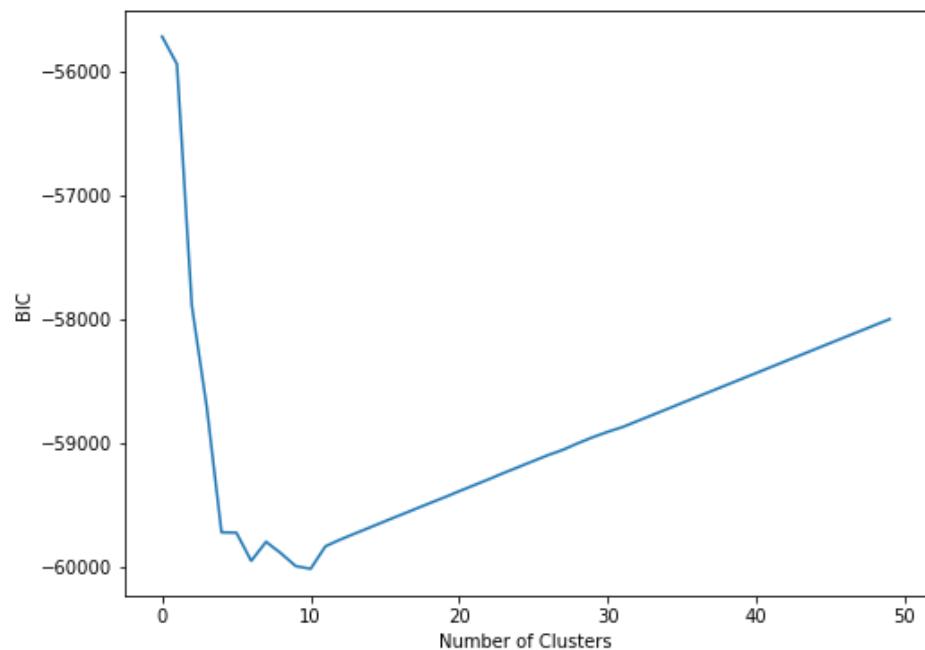
K-Means: Selection of K

- Approximate BIC
 - Reconstruction error plus a term for model complexity
- Minimum of curve is best value of K
 - ~200 in this case
 - Approach somewhat sensitive to scale of data (this impacts reconstruction error)
- Very different value of K to before



GMM: Selection of K

- BIC
 - Combination of model complexity and error
- Minimum of curve is best value of K
 - ~10



Why is K different each time?

- K-Means vs GMMs
 - GMMs have more parameters, so complexity penalties are larger for the same K
- Reconstruction cost is dependent on data scale
 - Data that has a very small range will have smaller reconstruction costs
 - Can lead to big differences between looking at reconstruction curve "elbow" and approximate BIC minimum

Selecting K

- Methods offer a suggested value
 - There may be reasons to use a different value
- Judgement of problem/data/solution requirements is important
 - Does a high K make analysing results too hard?
 - Does a small K risk grouping things together that are (or should be) distinct?
- You may have prior knowledge to help inform selection of K
 - You may know that there are 10 actual things to cluster
 - If you have prior knowledge, use it
- There are other methods to select K
 - Silhouette score for example

What happens with K is wrong?

- Two possible errors:
 - Over-clustering
 - True clusters are split into multiple sub-clusters, i.e. we have too many clusters
 - Under-clustering
 - True clusters are merged into a single cluster, i.e. not enough clusters
- Hard to work out what's happening when the true clusters aren't known.

CAB420: Clustering Applications

CLUSTERING ACTUAL DATA

Knowledge Discovery

- Given a dataset, try to extract some useful information to make sense of the data
 - Very broad and vague, what is "useful"?
- Clustering is one approach to help
 - Identify a small set of typical samples
 - Cluster centres
 - Clusters may have semantic meanings
 - Determine distribution of samples based on clustering
 - Which clusters are most common?
 - Do cluster occurrence rates change over time?

An Example

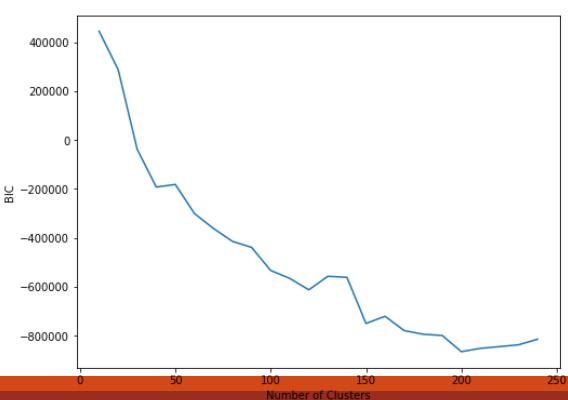
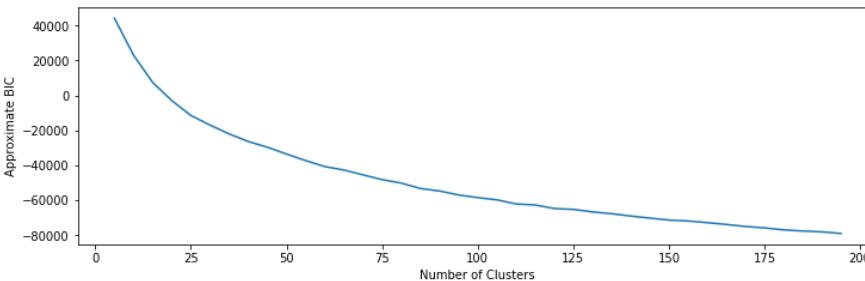
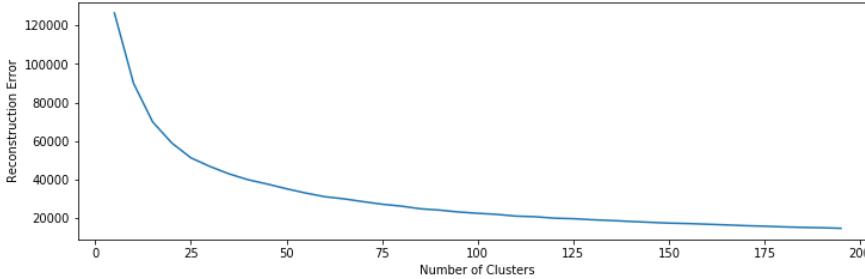
- See ***CAB420_Clustering_Example_4_Clustering Applications.ipynb***
- The Data
 - Bike share data from NY
 - Three months: July and December 2019, and July 2020
- Our Task
 - Compare usage between the three months

Data setup and pre-processing

- Use five dimensions
 - Trip duration (in seconds)
 - Start location (lat, lon)
 - End location (lat, lon)
- Dimensions have very different scales
 - Standardise data
- Some trips are very long (days or more)
 - Remove trips over 2 hours in length
 - Somewhat arbitrary choice

Selecting K

- K-Means (top)
 - Elbow of reconstruction curve (left) is at ~20
 - Minimum of approximate BIC is >200
- GMM (bottom)
 - Minimum of BIC at ~200

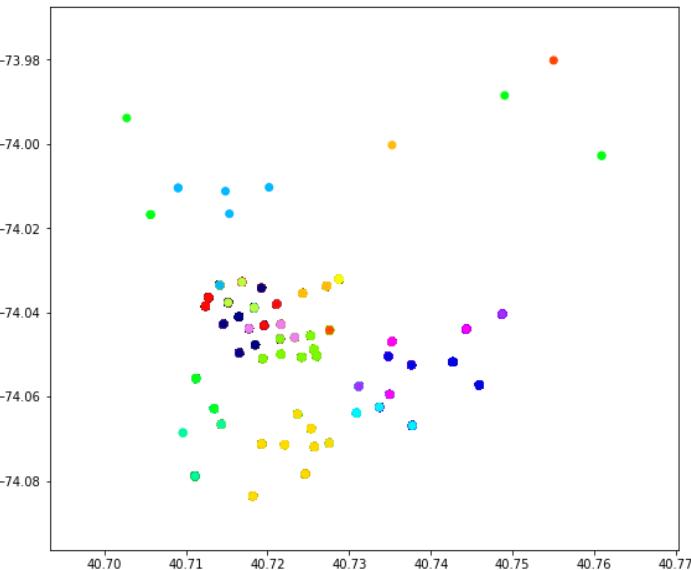
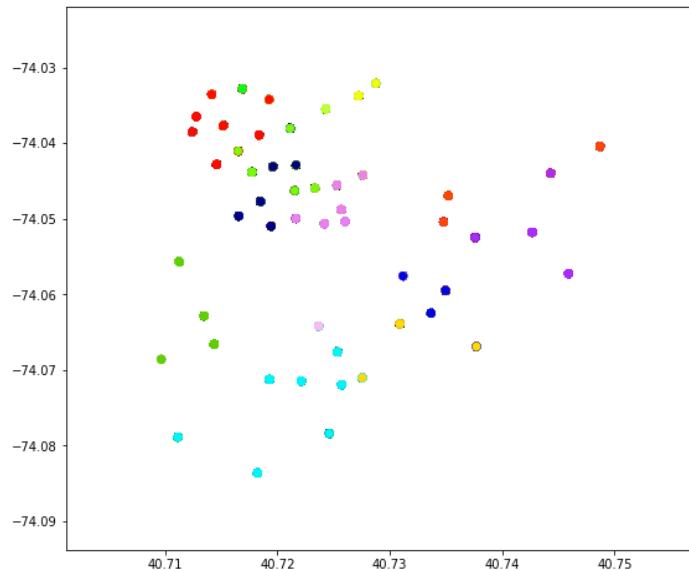


Selecting K

- Very different values of K for each plot
- Consider the application and data
 - Knowledge Discovery: we're seeking to use clustering to find a set of representative points (bike trips) which we can use to analyse the data
 - How many "types" of trip are possible?
 - How many can we reasonably analyse/compare?
 - Can we assign semantic meaning to clusters?
 - If so, how many and at what granularity?
- We'll stick to a smaller value of K to simplify analysis
 - K=20
 - Large enough to group very different behaviours
 - Small enough to keep analysis simple
 - Likely leading to under-clustering, in that true clusters are being merged

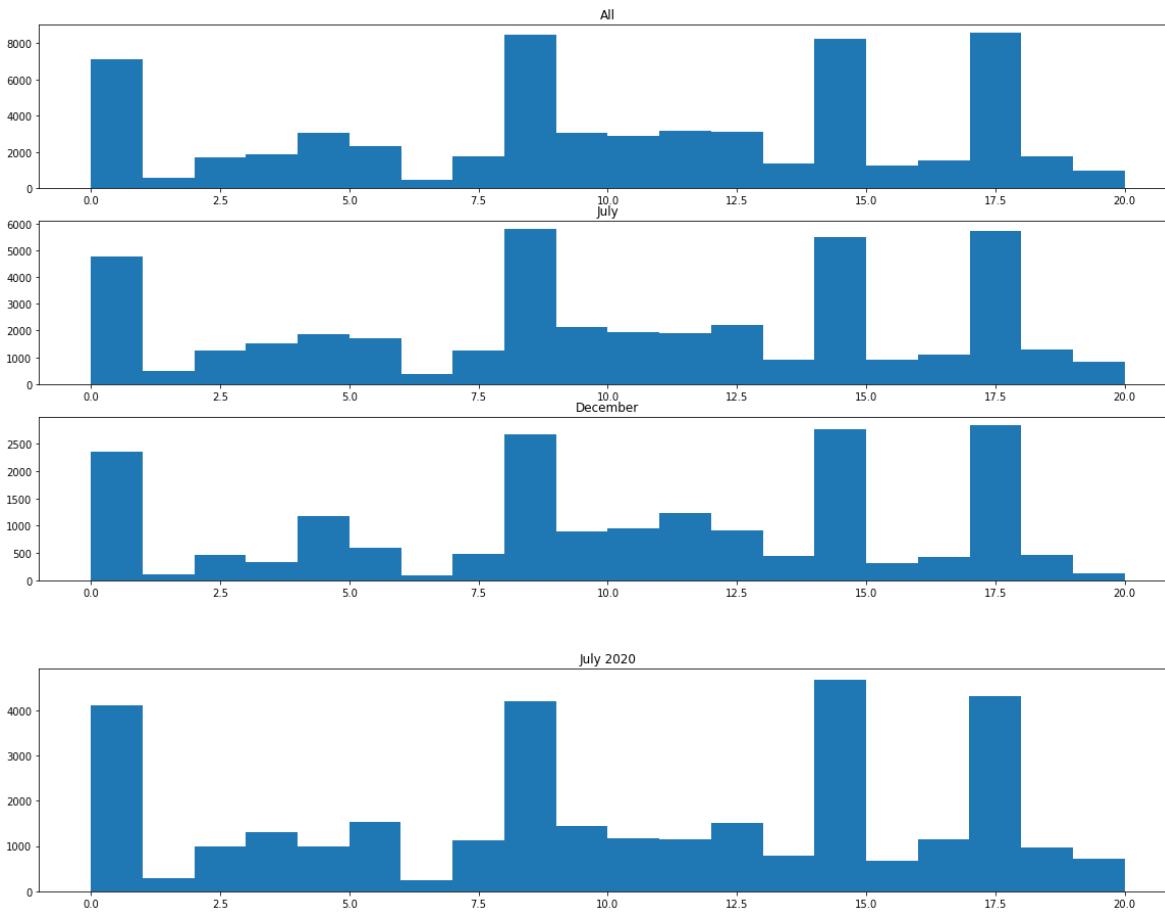
K-Means Clusters

- Start location (left) and end location (right) shown
 - Trip Duration not shown
 - Hard to plot 5D data
- Inspection of cluster centres (see example) shows that a couple of clusters capture long trips



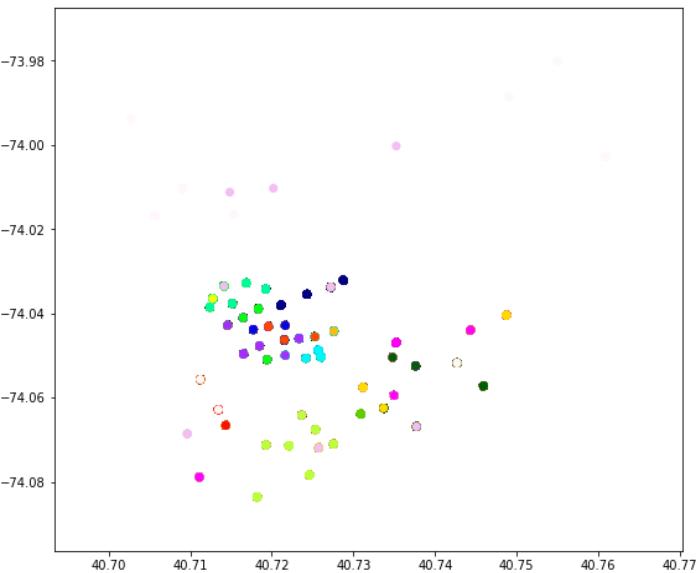
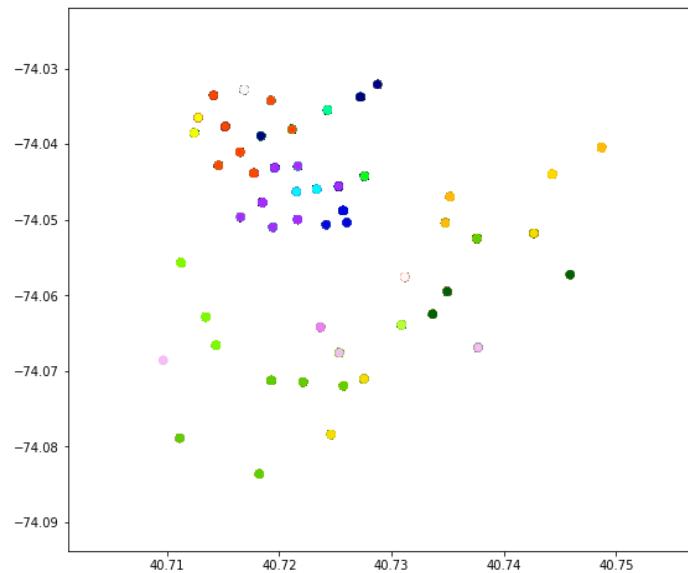
K-Means Analysis

- Patterns of use visualised by looking at how often points in each cluster occur
- Overall patterns similar across all considered time periods



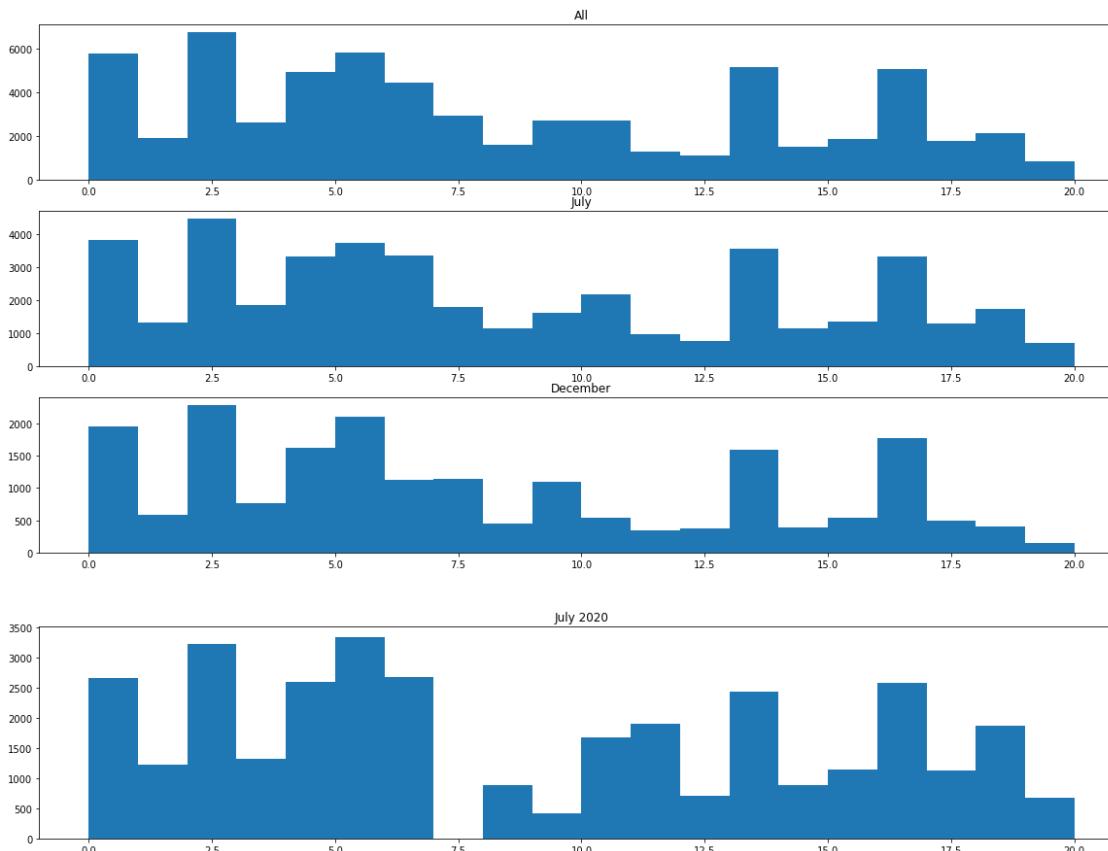
GMM Clusters

- Similar to K-Means, but clusters less uniform in size
 - GMMs allow cluster size and shape to vary
 - K-means limited to spherical clusters



GMM Analysis

- Same visualisation as K-Means
- Both 2019 time periods appear very similar
- More pronounced differences between 2019 and 2020



Further Thoughts

- Distance metrics are important
 - Is the way points are being compared valid?
 - We have locations (lat, lon) and durations? Is Euclidean distance appropriate?
- Cluster distributions can be compared numerically
 - Histogram intersection, Bhattacharya distance
 - See example
- Number of clusters will impact analysis
 - More clusters will tend to better highlight differences
 - Too many clusters will show differences where they're actually aren't any

Anomaly Detection

- Given a set of data, find points that are unusual or abnormal
 - Also referred to as outlier detection
- Typical approach is
 - Train a model on normal data
 - Evaluate the model on a new set of data
 - Any point that is a sufficiently poor fit to the data is an outlier
 - Requires a threshold to define what "sufficiently poor" is
- Value of K can impact performance
 - Larger K means more clusters, which means points overall will fit the model better

An Example

- See ***CAB420_Clustering_Example_4_Clustering Applications.ipynb***
- The Data
 - Same as before (Bike share data from NY)
 - July and December 2019 from the training set
 - July 2020 is the test set
- Our Task
 - Find abnormal trips in the test set

Data setup and pre-processing

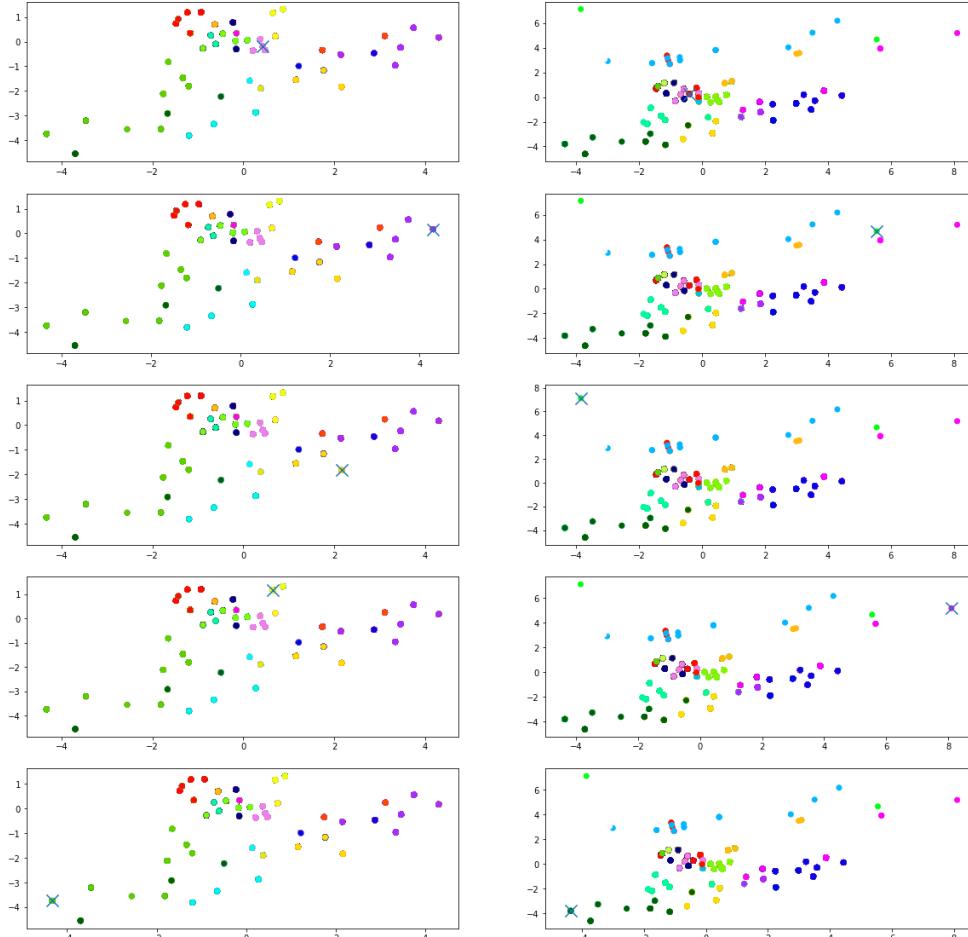
- Same as for previous application
 - Same dimensions
 - Use the same value of K for clustering approaches
- Only concerned with anomalies in July 2020
- Rather than set a threshold, we'll find the set of the most abnormal points
 - Ideally, to set a threshold we'd have a dataset with known anomalies, and use this to tune a threshold to reach the desired detection sensitivity

Anomalies and K-Means

- We can use distance to assigned cluster centre as a proxy for how unusual a point is
 - Limited in that it can identify points that lie at the boundary of two clusters
 - Can be misleading as it does not consider cluster spread or density

Anomalies and K-Means

- Abnormal points are dominated by long trips
- Longer trips lead to a larger distance, even with standardised data

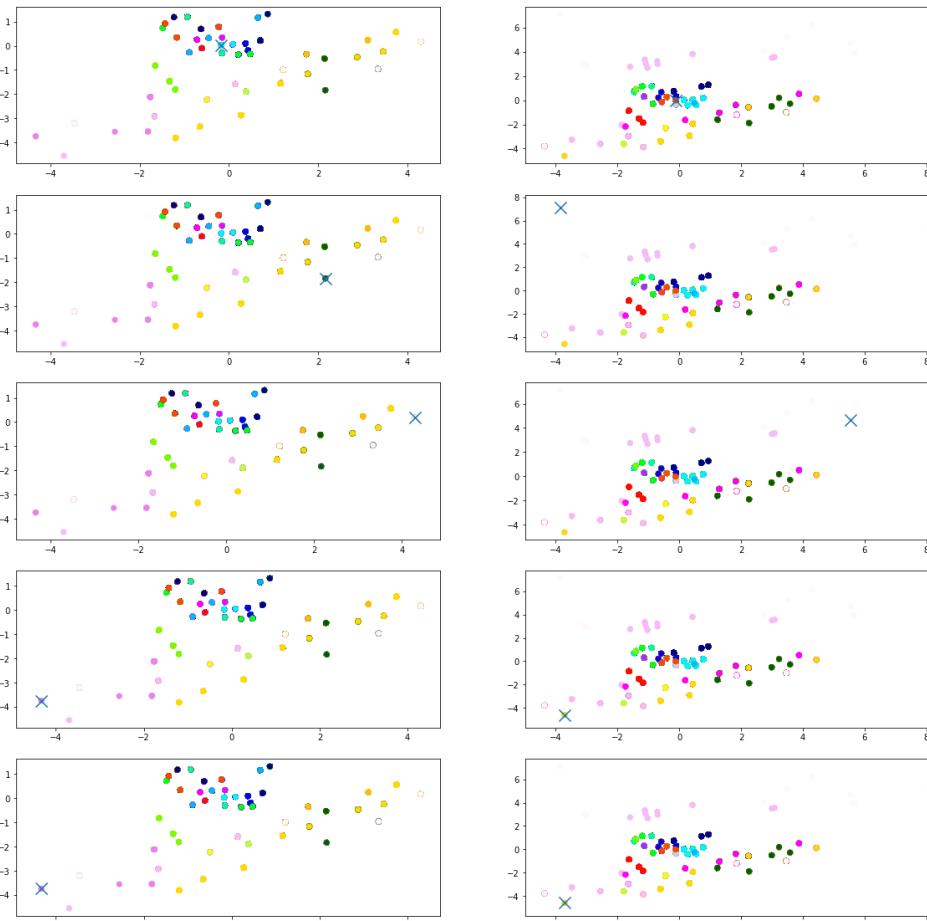


Abnormalities and GMMs

- GMMs allow us to determine the likelihood of a point
 - How likely is it that this point belongs to this distribution?
- Allows us to identify highly unlikely (abnormal) points

Abnormalities and GMMs

- Abnormal trips are a mix of long and short trips
 - Seems more realistic than the K-Means results
- All abnormal points belong to cluster 19
 - Seems odd, suggests that the clustering results need further investigation
 - Could be under-clustering and have several behaviours grouped together



CAB420: Auto-Encoders

ENCODE YOURSELF

Encoders and Decoders

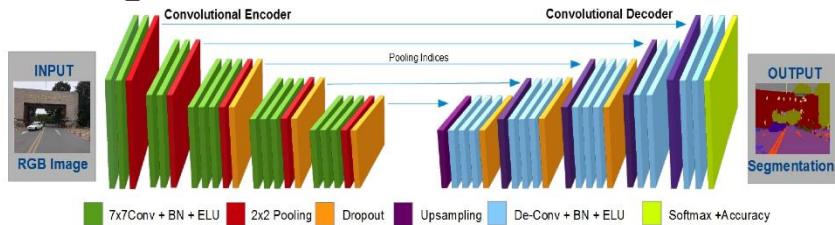
- Common components in deep learning
- Typically operate on signal-based input
 - Images, videos, audio clips
- Often used in a pair
 - Image -> Encoder -> Decoder -> Output (often another image)
- We already seen lots of “encoders”
 - Siamese networks encode the input into an embedding

Encoders and Decoders

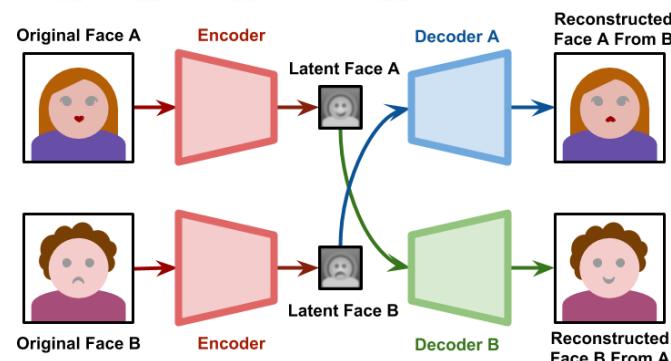
- Encoders
 - Take an input signal, aim to extract a compressed representation
 - Compressed representation may be good for different things depending on how the network is setup
- Decoders
 - Takes the compressed representation
 - Outputs a synthesised signal
 - Can be the original signal (auto-encoder)
 - Can be something else

Encoder-Decoder Applications

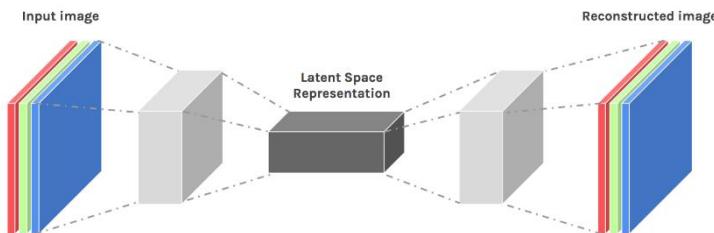
- Image to Image translation



- DeepFakes



- AutoEncoders



Auto-Encoders

- Given an input
 - Encode it into a compact representation
 - Then decode it, getting the original back
- Deep Learning for Dimension Reduction
 - Unlike PCA or LDA, can learn a highly non-linear representation
 - Like PCA, the compressed representation can be used to reconstruct the original signal
- Typically seen as unsupervised learning
 - No explicit ground truth signal or label
 - Target label is the same as the input

Learning Objective

- Auto-encoders try to reconstruct the original signal

$$L_{recon} = \sum_i^N (x_i - \hat{x}_i)^2$$

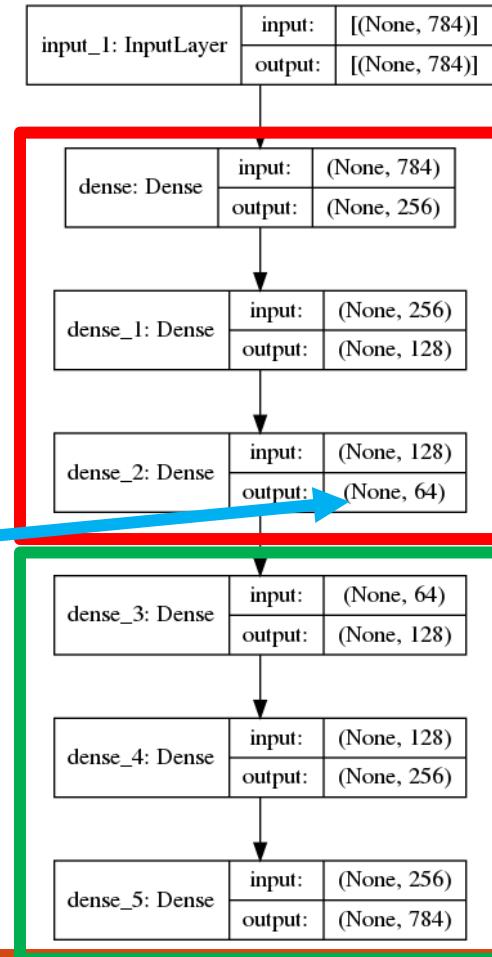
- x_i is the input signal
- \hat{x}_i is the reconstructed signal
- N is the size of the signal
- You may also often see an L1 distance used
- Can be seen as a many-to-many regression problem
 - Regress N outputs from N inputs

An Example

- See ***CAB420_Encoders_and_Decoders_Example_1_AutoEncoders.ipynb***
- Our data
 - Fashion MNIST
- Our Task
 - Encode Fashion MNIST into a compact representation
 - Decode it to reconstruct the original data

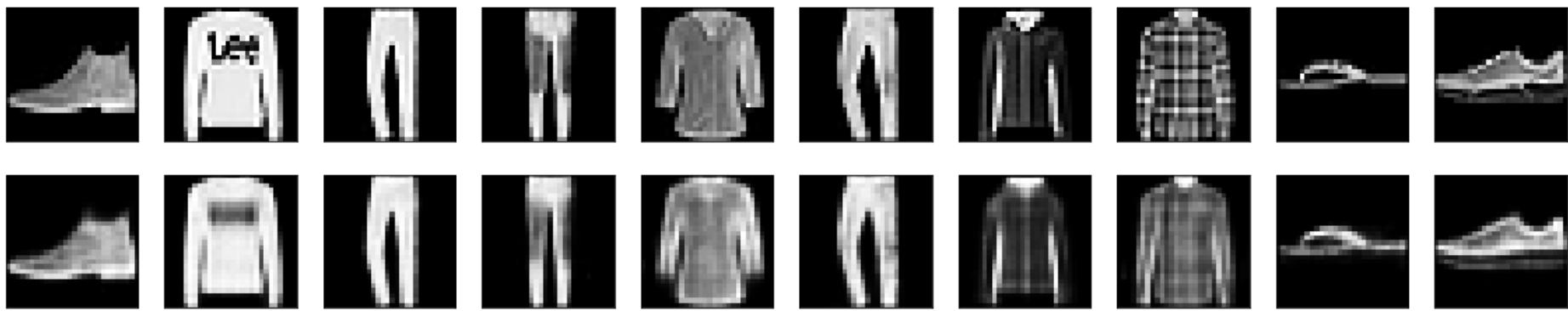
My First Auto Encoder

- Vectorised input
 - 28x28 image to 1x784 vector
- Encoder
 - Three dense layers
 - Final shape is 1x64
(compressed representation)
- Decoder
 - Three dense layers
 - Mirrors the encoder
 - Upsample back to 1x784 **Bottleneck**
 - Reconstruct the original signal



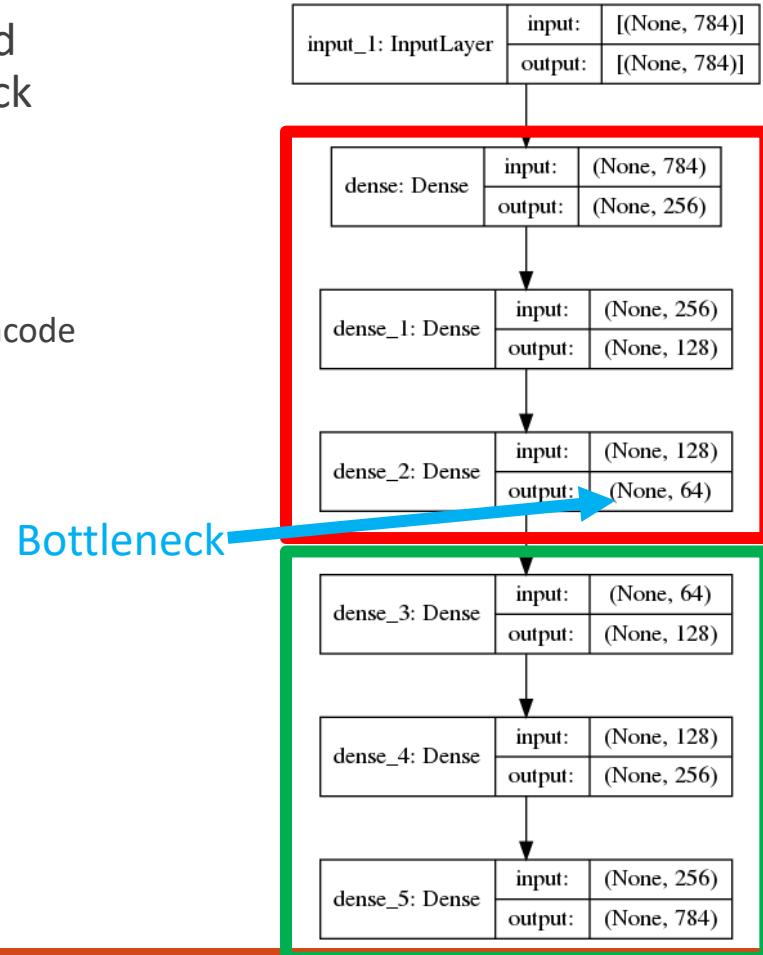
Auto Encoder Output

- Reconstructions contain major details
 - Edges are blurred
 - Textures (text, checked patterns) are somewhat lost
 - Broad shape/structure preserved



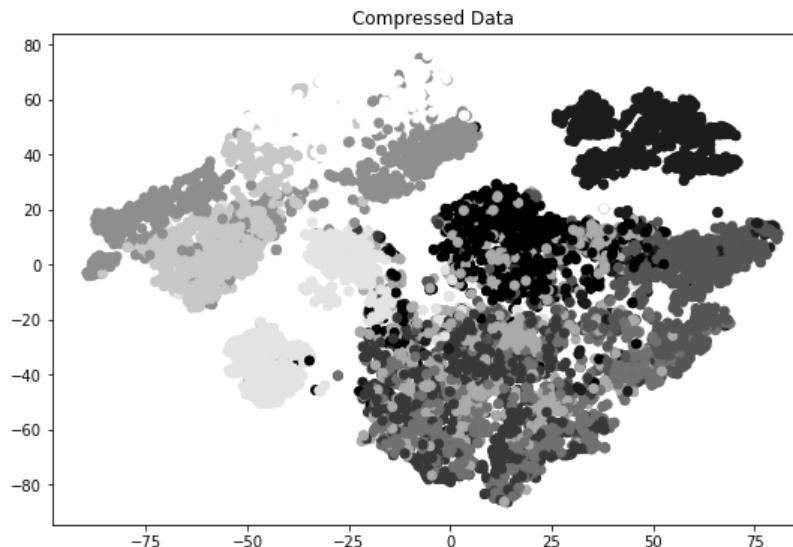
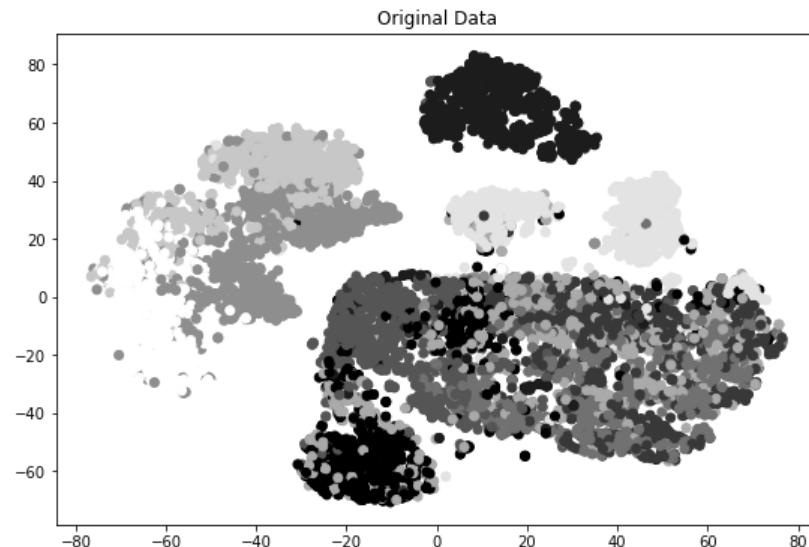
Sparse Autoencoders

- We may also wish to add sparsity to our bottleneck layer
- L1 penalty
 - Similar to Lasso regression
 - Force network to learn to encode the input with fewer active nodes
 - Hope to promote a more meaningful representation



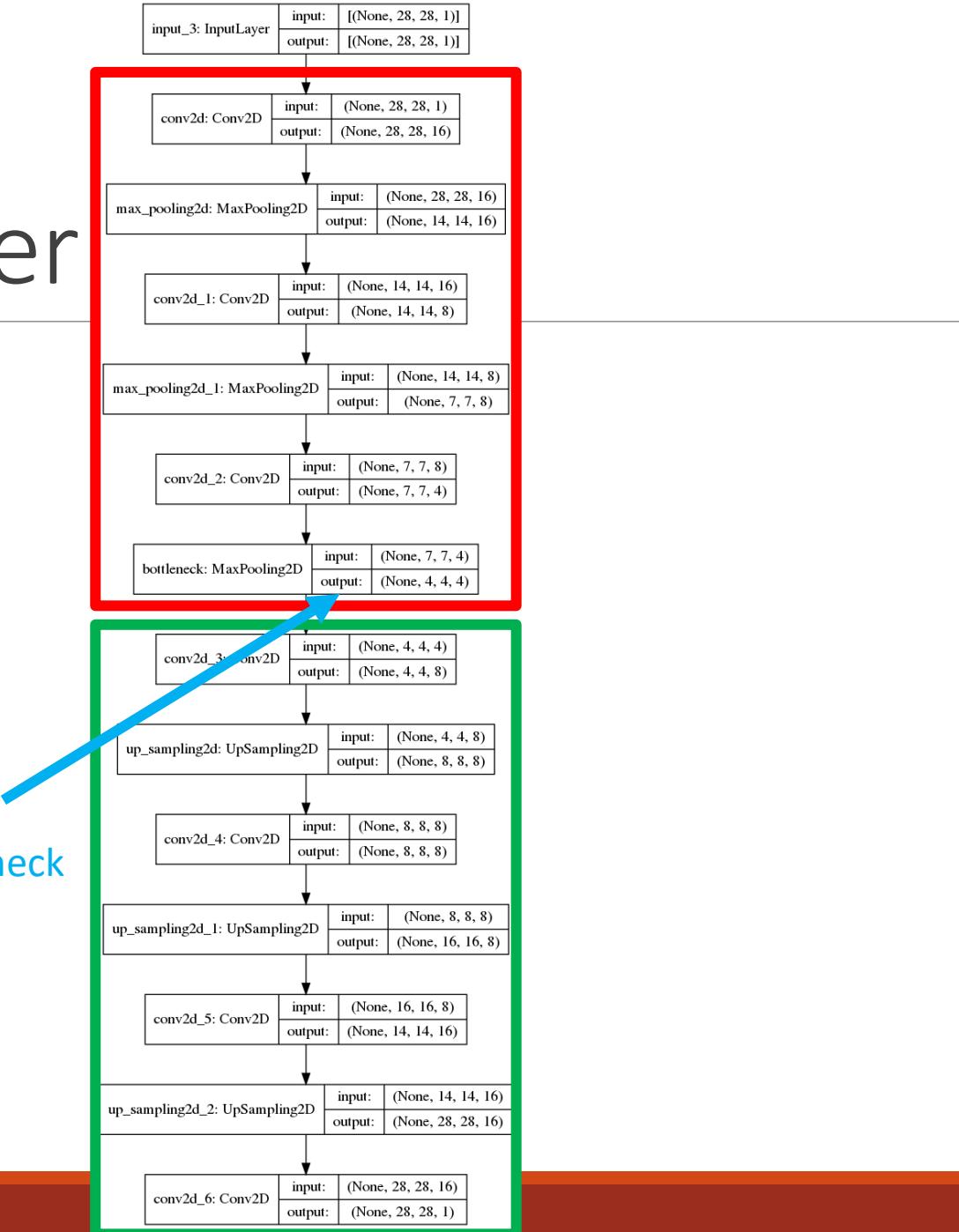
What is Learnt?

- t-SNE plots of the original data and bottleneck output
- Broad shape similar
- Auto-encoder is unsupervised
 - No class labels, cannot learn a strong boundary between classes



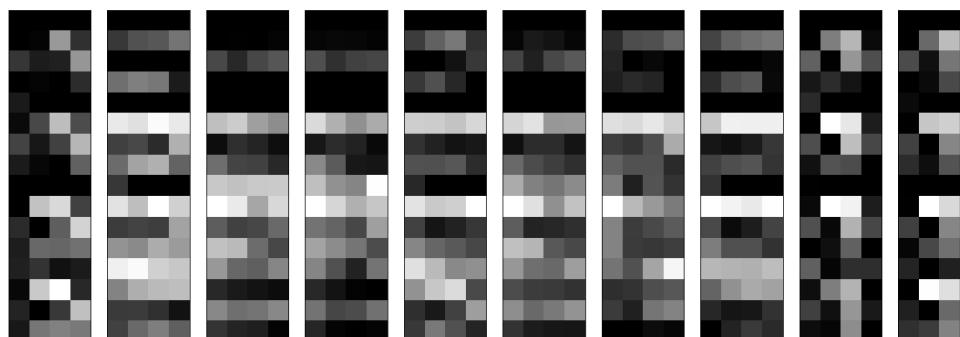
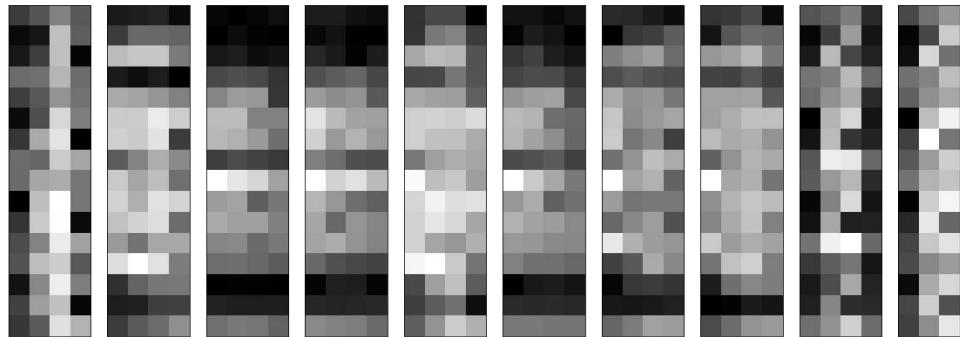
A Better Autoencoder

- Encoder
 - 3 Conv2D layers
 - Number of filters decreasing as we go deeper
- Decoder
 - Reverse of encoder
 - Upsample layers in place of MaxPooling
- Bottleneck
 - 4x4x4 tensor (64 units)



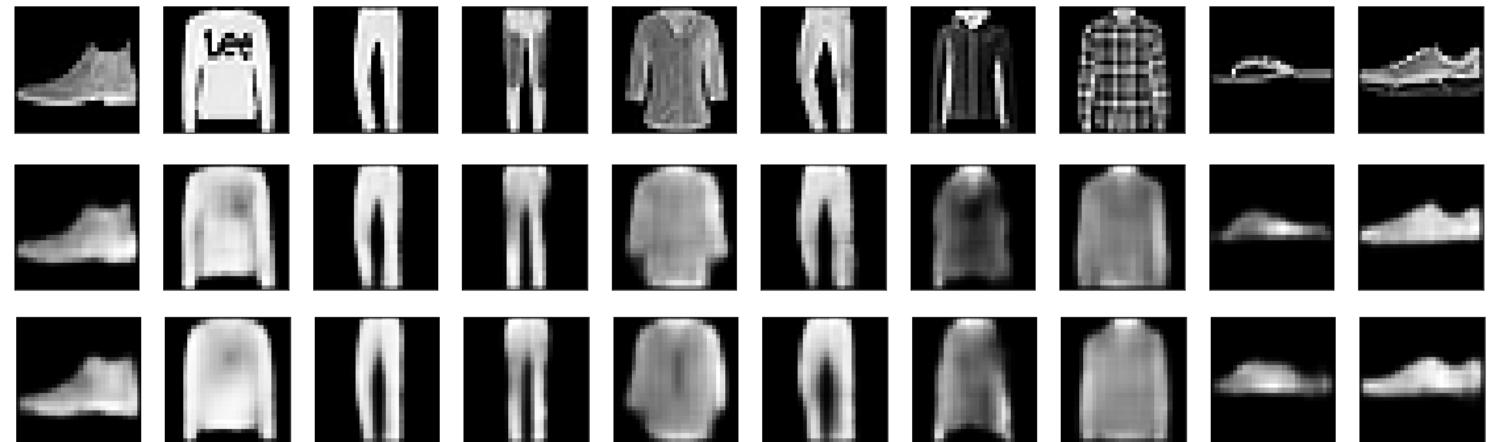
Impact of Sparsity

- Activations for the same input for network
 - Each column are the feature maps for one input sample
 - Without sparsity penalty (top)
 - With sparsity penalty (bottom)
 - Many fewer neurons active



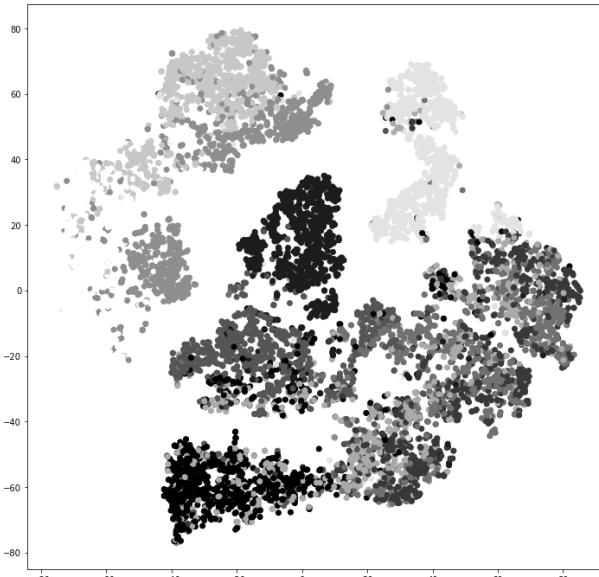
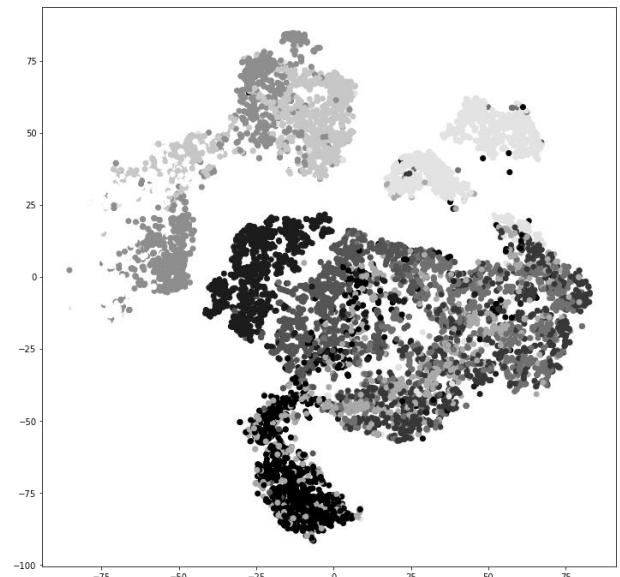
Impact of Sparsity

- Top: Original data
- Middle: Reconstruction without sparse constraint
 - Perhaps contains slightly more fine detail
- Bottom: Reconstruction with sparse constraint



Impact of Sparsity

- t-SNE plots of bottleneck features
- Left: without sparsity constraint
- Right: with sparsity constraint
 - Does a slightly better job disentangling classes
 - Sparsity helps to model to associate specific neurons with specific classes



Why Auto-Encoders?

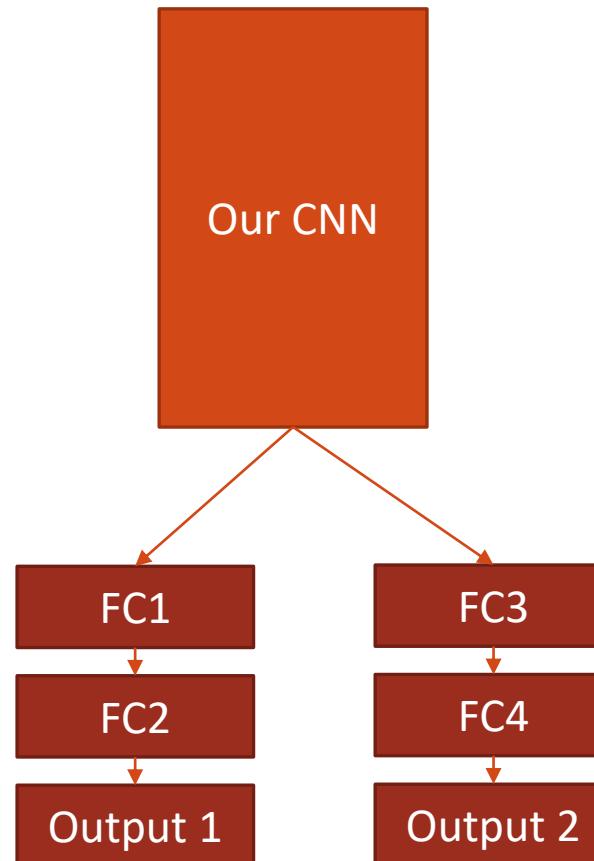
- It does have applications
 - Non-linear data compression/dimension reduction
 - Can stack them to get more compression
 - Anomaly detection
 - Given an input, compress and reconstruct
 - Something normal will be reconstructed well, something abnormal will have a high error
 - As pre-training for a network
 - Reuse the encoder in a classification network

CAB420: Multi-Task Learning

MULTI-TASKING FOR DEEP NETS

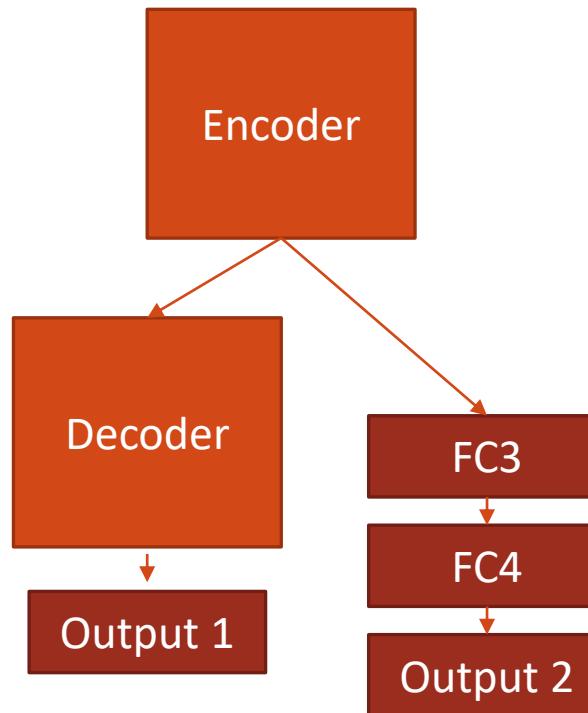
Multi-Task Learning

- We've seen with deep networks that
 - The same network can usually do different things, we just need to change the output shape and/or loss
- Why not then have multiple outputs?
 - Multiple output layers
 - One loss per layer
 - Can have a different loss function for each output
 - Overall loss is just the sum of the losses
 - Can be weighted such that some outputs are more important than others



Multi-Task Learning

- Outputs can come from different parts of the network
 - Outputs can have wildly different shapes
 - Image outputs
 - Numeric Outputs



Multi-Task Learning Objective

- Multiple outputs means multiple losses
- Output 1 (auto-encoder): reconstruction loss

$$L_{recon} = \sum_i^N (x_i - \hat{x}_i)^2$$

- Output 2 (classifier): categorical cross entropy:

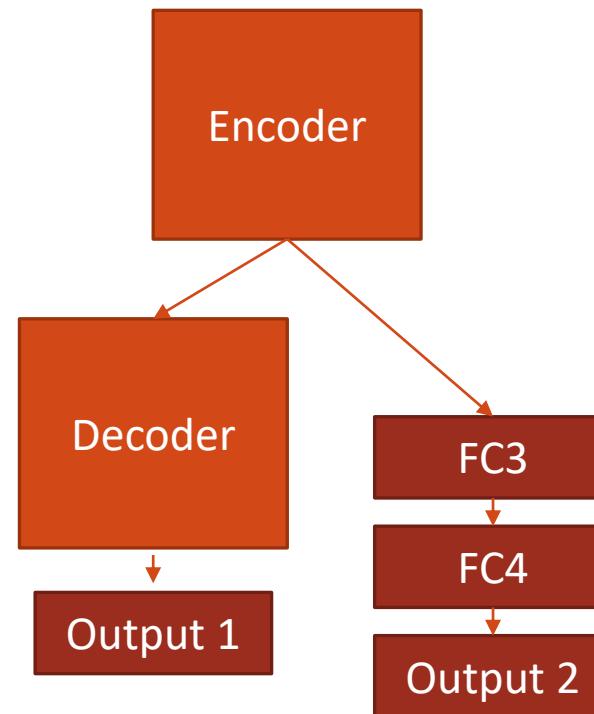
$$L_{CE} = - \sum_i^N y'_i \log(y_i)$$

- Overall loss combines the two

$$L_{Overall} = \lambda_1 L_{recon} + \lambda_2 L_{CE}$$

- We can set λ_1 and λ_2 as we see fit.

- If in doubt, $\lambda_1 = \lambda_2 = 1$



Multi-Task Learning

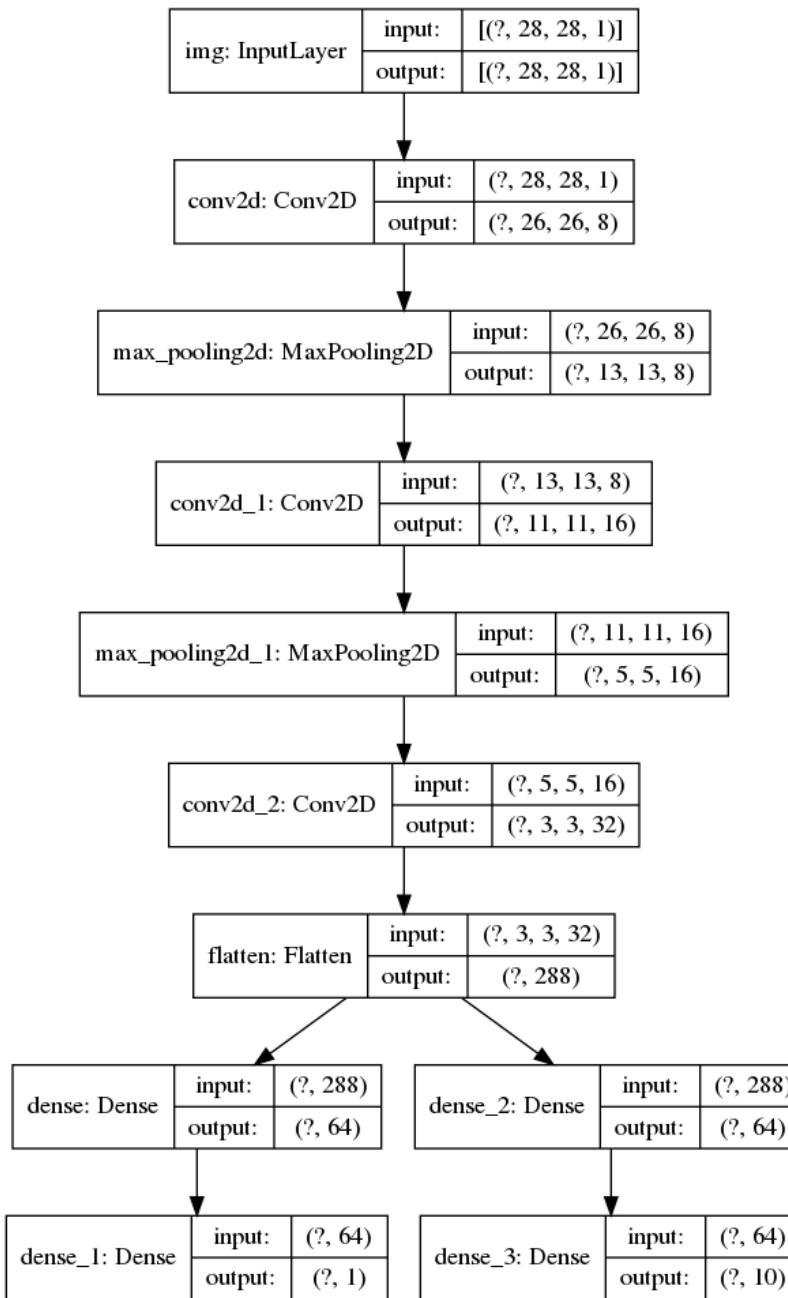
- Pros
 - Usually has a low overhead
 - Can just append a couple of extra layers to get another output
 - Cheaper than having a network for each task
 - Usually helps learning
 - Particularly if tasks are related
 - One task helps regularise the other
- Cons
 - We now need two sets of labels
 - It's hard enough to annotate one set

An Example

- See ***CAB420_Encoders_and_Decoders_Example_2_Multiple_Outputs.ipynb***
- Our Task
 - Rotated Digits Datasets
 - Simultaneously estimate
 - The digit (0, 1, 2, ...)
 - How much the digit has been rotated by

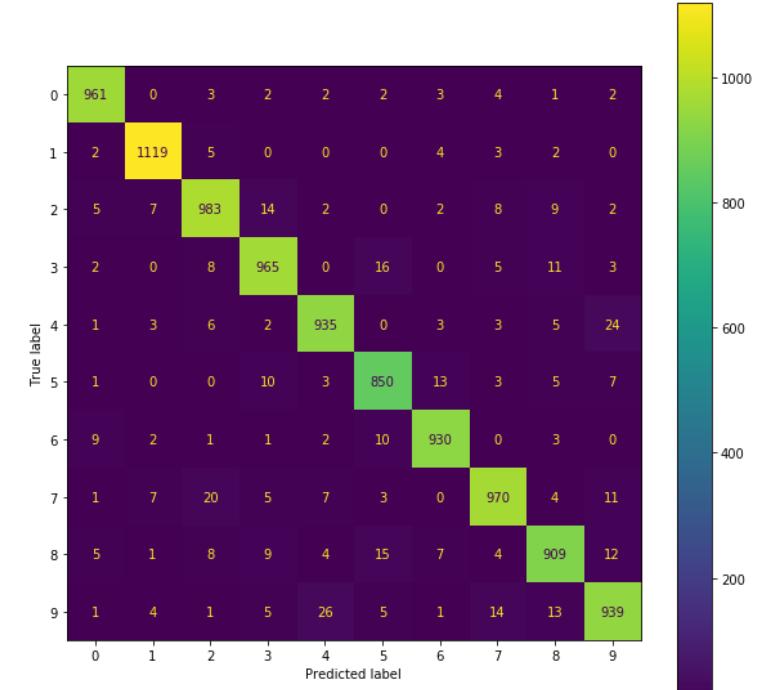
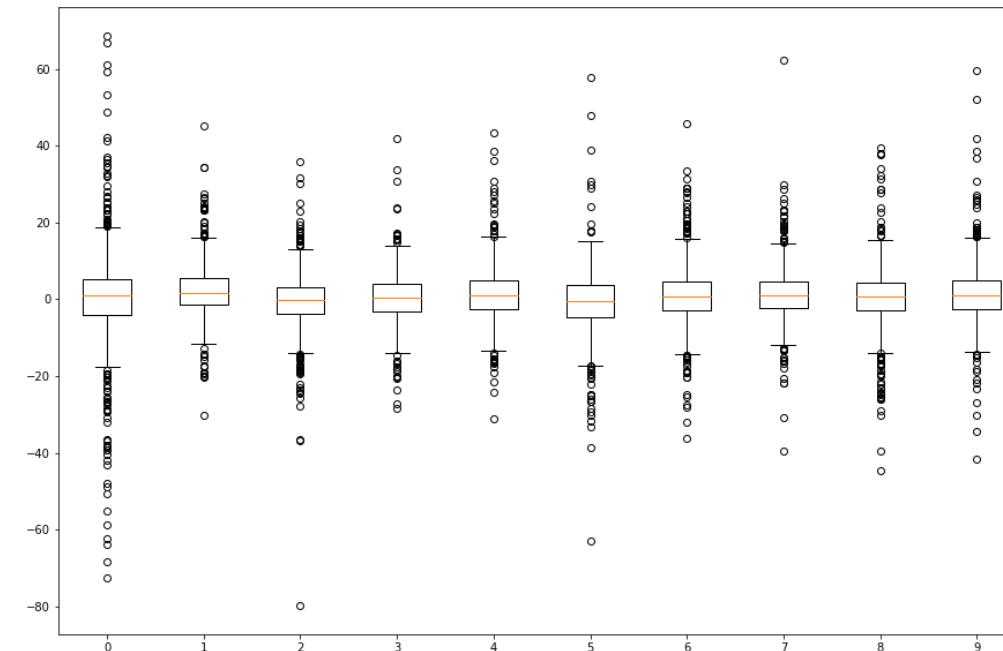
Our Network

- Simple CNN
 - 3 Convolution layers
- Network branches after last convolution
 - Both branches are two dense layers
 - Different output sizes in each
 - `dense_1`, size $(?, 1)$ is the angle output
 - Mean Squared Error loss
 - `dense_3`, size $(?, 10)$ is the digit classification output
 - Categorical Cross Entropy loss



Network Performance

- Overall, both tasks are performed well
- Similar performance to when performing either task individually
 - Unsurprising, tasks are closely related, should help each other



Training Performance

- One loss (MSE) dominates
 - Scale of MSE is much larger than Cross Entropy
 - Means this loss may have more of an impact on learning – bigger values equals bigger gradients
 - Has limited impact here due to very complementary nature of the tasks



Tweaking Loss Weights

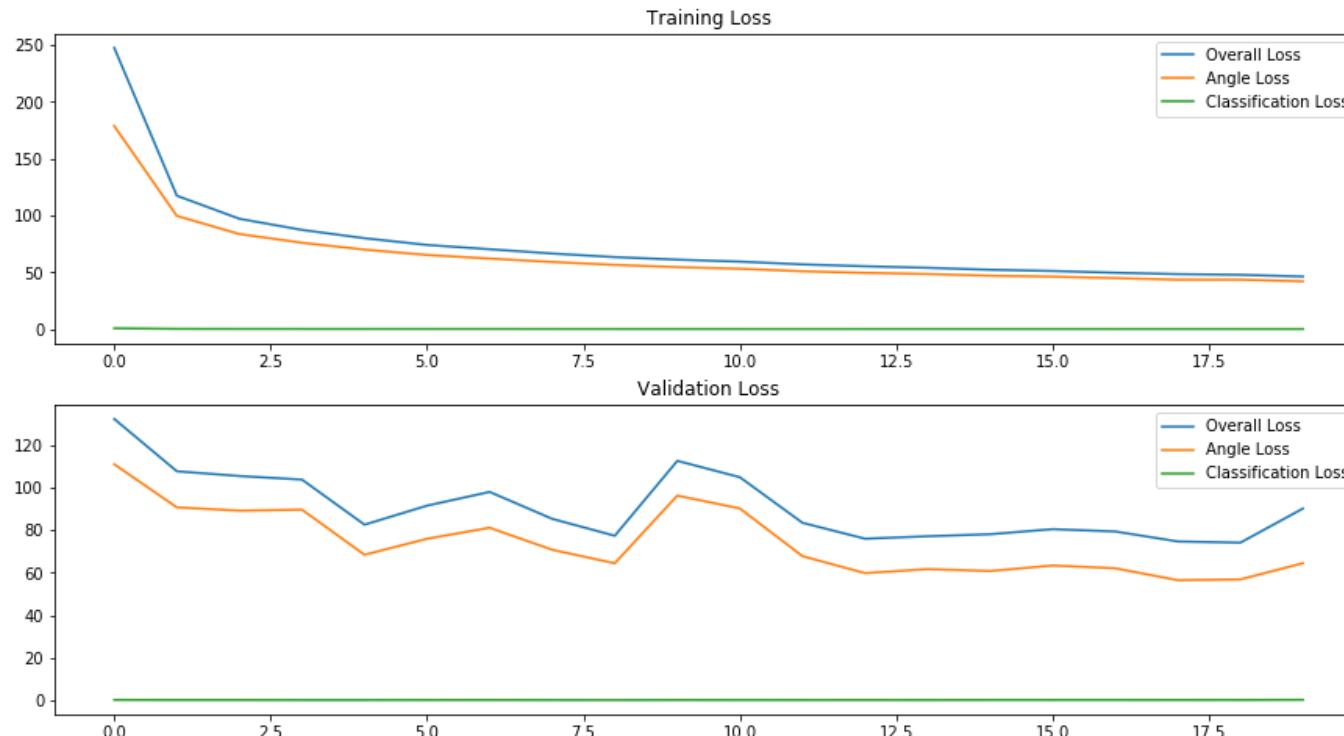
- Our loss can be expressed as

$$L_{Overall} = \lambda_1 L_{MSE} + \lambda_2 L_{CE}$$

- Our first approach set $\lambda_1 = \lambda_2 = 1$
- This time, we'll use $\lambda_1 = 1; \lambda_2 = 100$

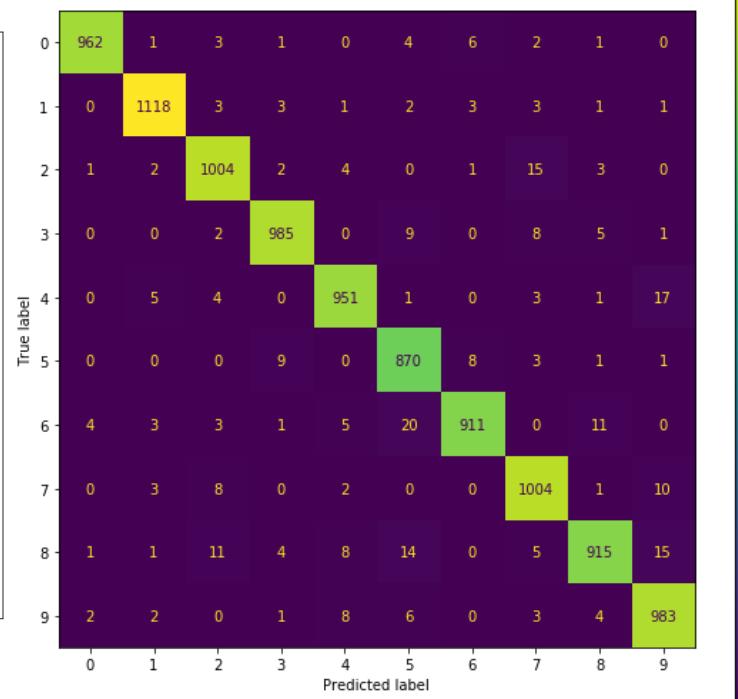
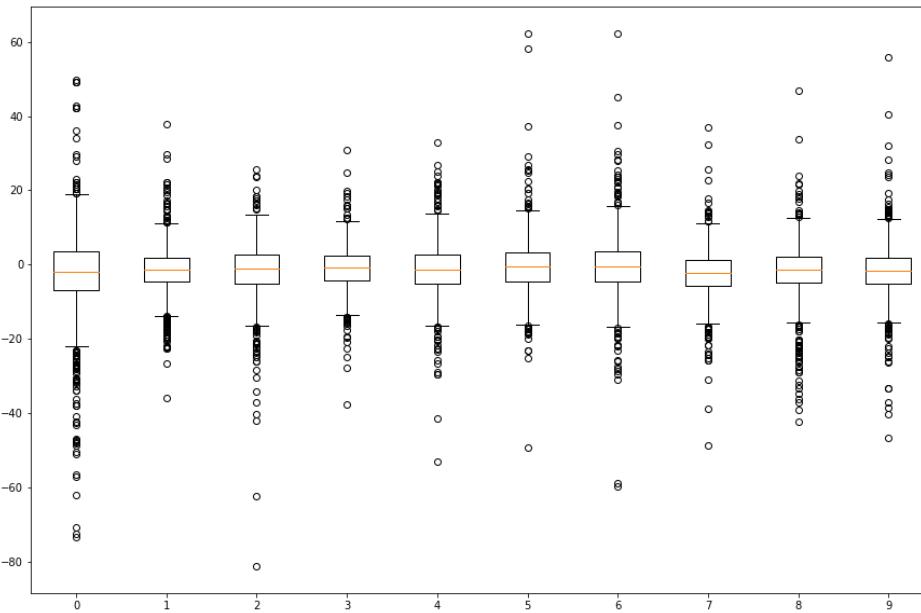
Tweaking Loss Weights

- Change visible in overall loss
 - Classification loss is not scaled when plotting



Network Performance

- Very similar to without loss weights
 - Highly complementary tasks, so little was lost by the imbalanced loss scale



CAB420: Semi-Supervised Learning

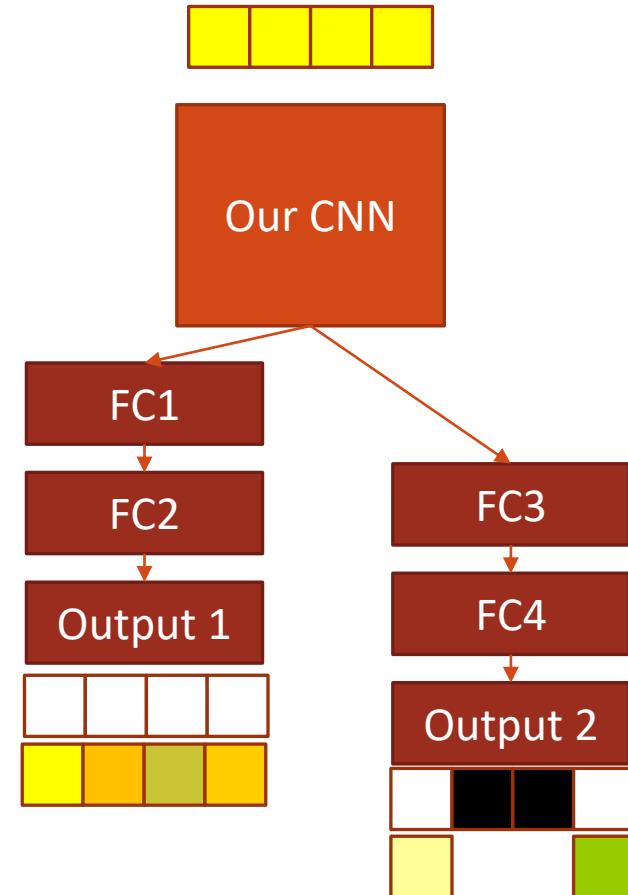
WHEN YOU CAN'T BE BOthered LABELLING
ALL YOUR DATA

Semi-Supervised Learning

- Pretend we have a dataset which has annotation for two tasks
 - One has full annotation
 - One has data only for some samples
- We wish to learn both tasks
 - We don't wish to do any further annotation

Semi-Supervised Learning

- Modify our loss functions to avoid missing samples
 - Given 4 input samples
 - Output 1 has data for all 4
 - Output for this batch will be the sum of the loss for all four samples
 - Output 2 has data for only 2
 - Output for this batch will be the sum of the loss for the two samples for which data exists



Semi-Supervised Learning

- For each sample, we should have some ground truth signal
 - Need some annotation
 - Can workaround this with
 - Auto-encoders
 - Input becomes an output
 - GANs
- May wish to adjust output weights to reflect what data we have
 - Outputs with limited data may be given a higher weight
 - Encourage learning from whatever data we have

Semi-Supervised Learning Objective

- Output 1, classification objective

$$L_1 = - \sum_i^N y'_{1,i} \log(y_{1,i})$$

- Output 2, semi-supervised classification objective

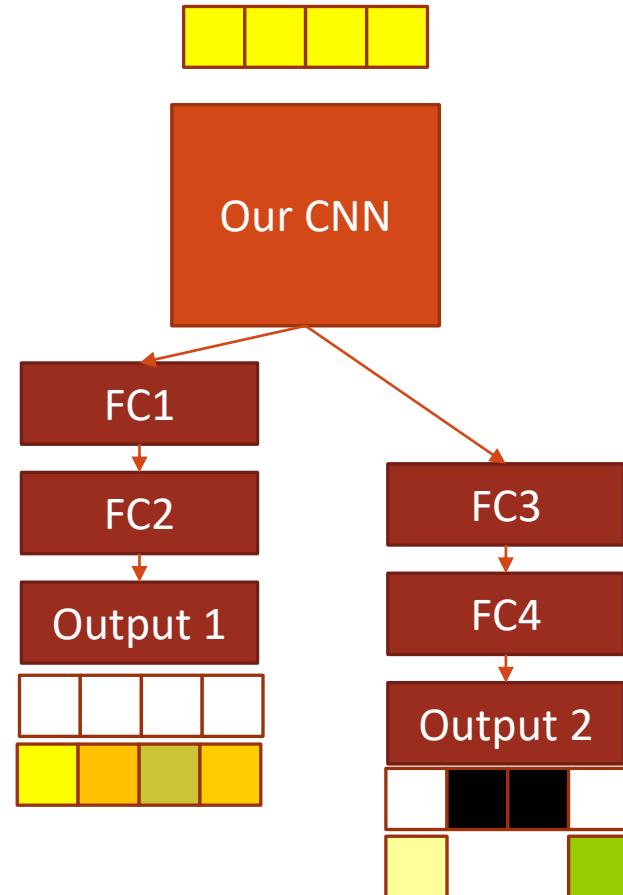
$$L_2 = - \sum_i^N M_i y'_{2,i} \log(y_{2,i})$$

- M_i is a mask variable, equals 1 if we have ground truth, 0 if not

- Overall Loss

$$L_{Overall} = \lambda_1 L_1 + \lambda_2 L_2$$

- If M_i is often 0, L_2 will be small. May need to increase λ_2 to compensate

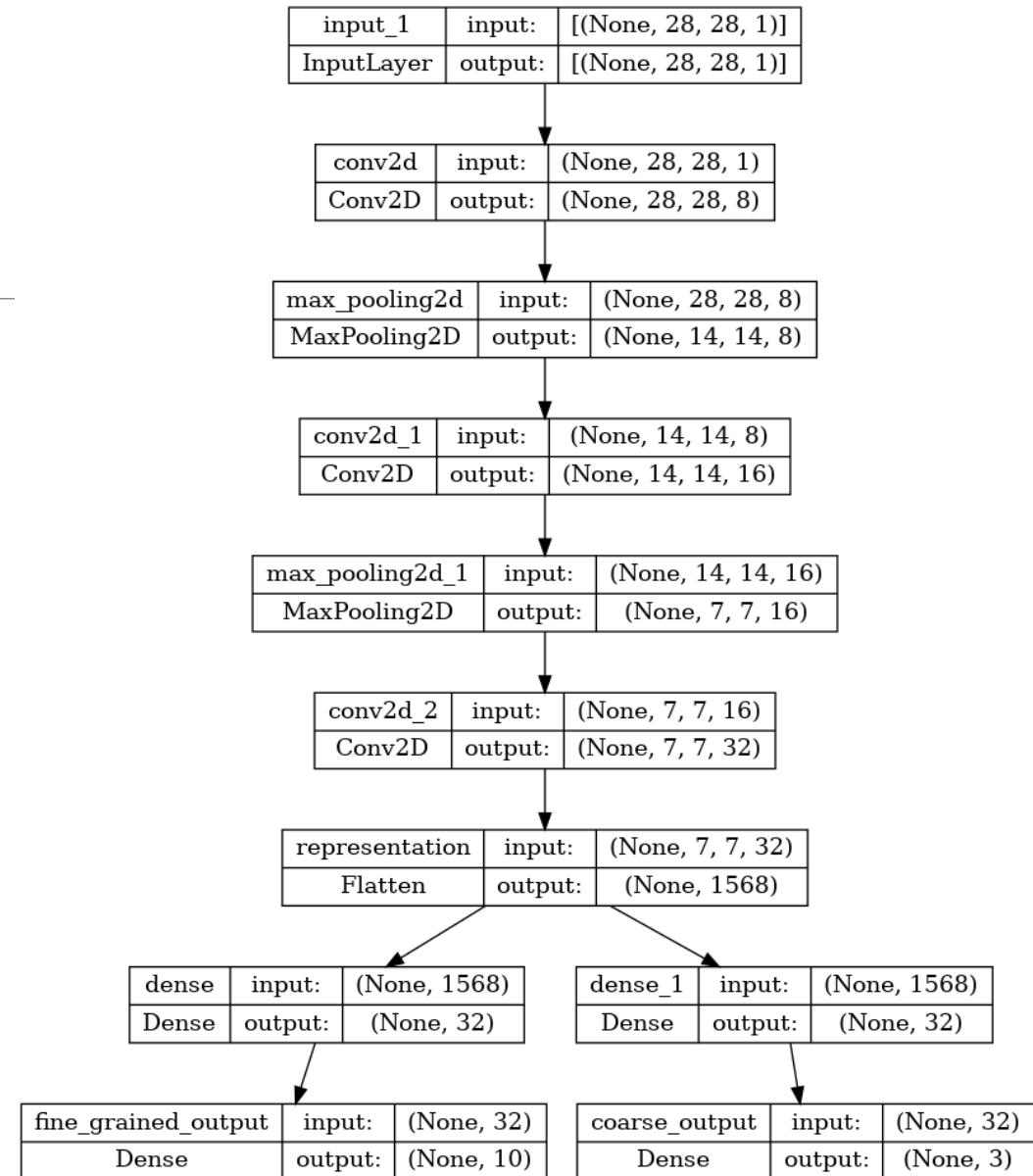


An Example

- See ***CAB420_Encoders_and_Decoders_Example_3_Semi_Supervised_Learning.ipynb***
- Our data
 - Fashion MNIST
- Our task
 - Coarse and fine-grained clothing classification
 - Coarse task
 - 3 classes (tops, bottoms, other)
 - Data for all samples
 - Fine-grained task
 - Usual 10-class problems, but with limited data

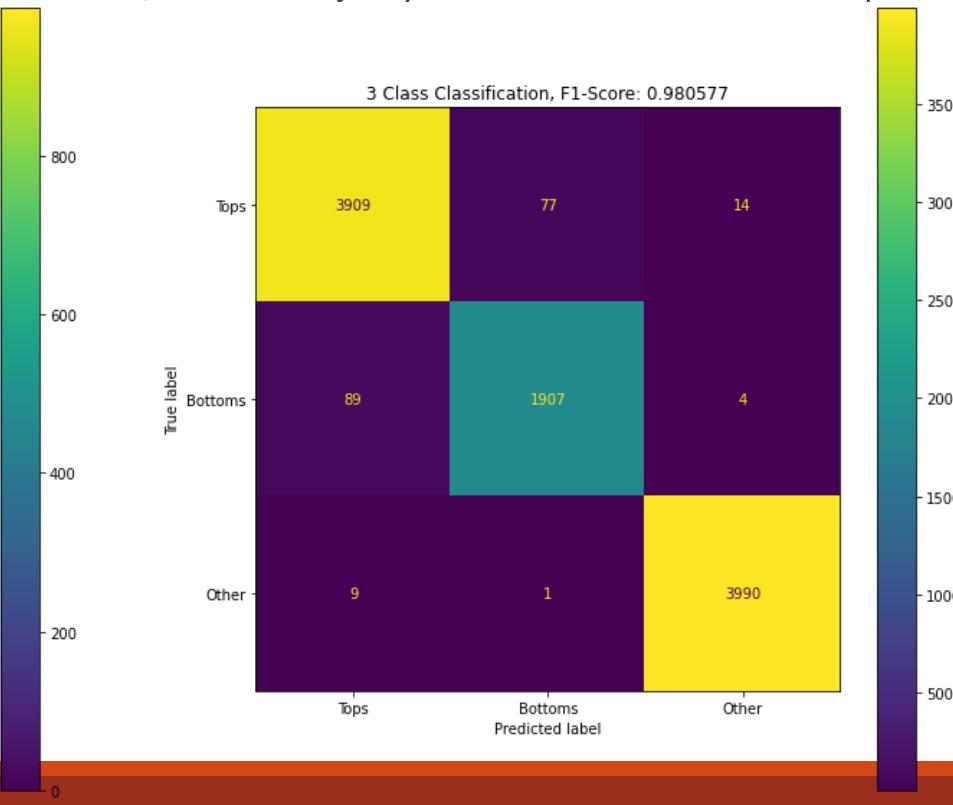
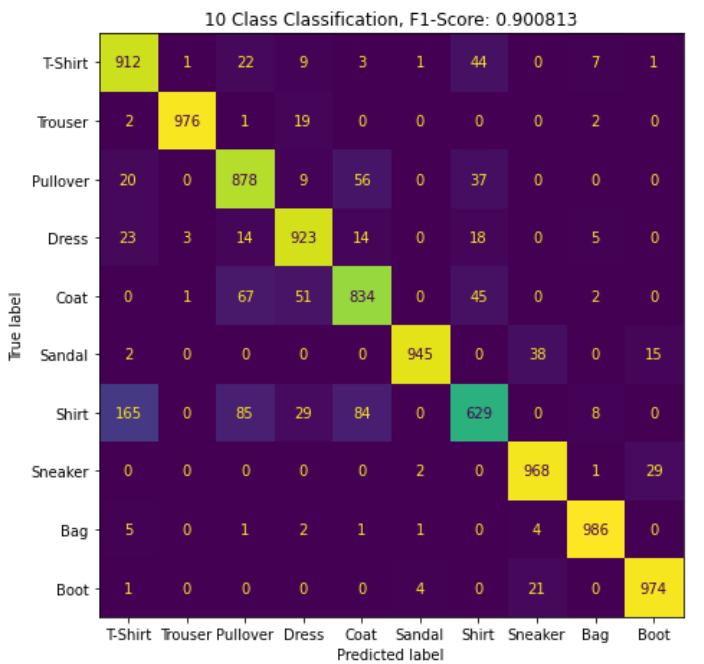
Our Network

- Modified classification network
- Standard convolution backbone
- Branch at flattened representation
 - One coarse classifier
 - 3 tasks
 - One fine-grained classifier
 - 10 tasks



Training with all the data

- Good performance for both tasks
- Coarse task clearly supporting fine-grained task
 - Note errors in 10-class confusion matrix, the vast majority are confusion between examples within a coarse class



Removing Data

- Remove 75% of the labels
- Labels contain a one-hot representation
- Masks contain all -1 when the sample is removed
- Use a modified loss function that takes the masks as an extra input
 - Exclude masked samples from calculations

Labels

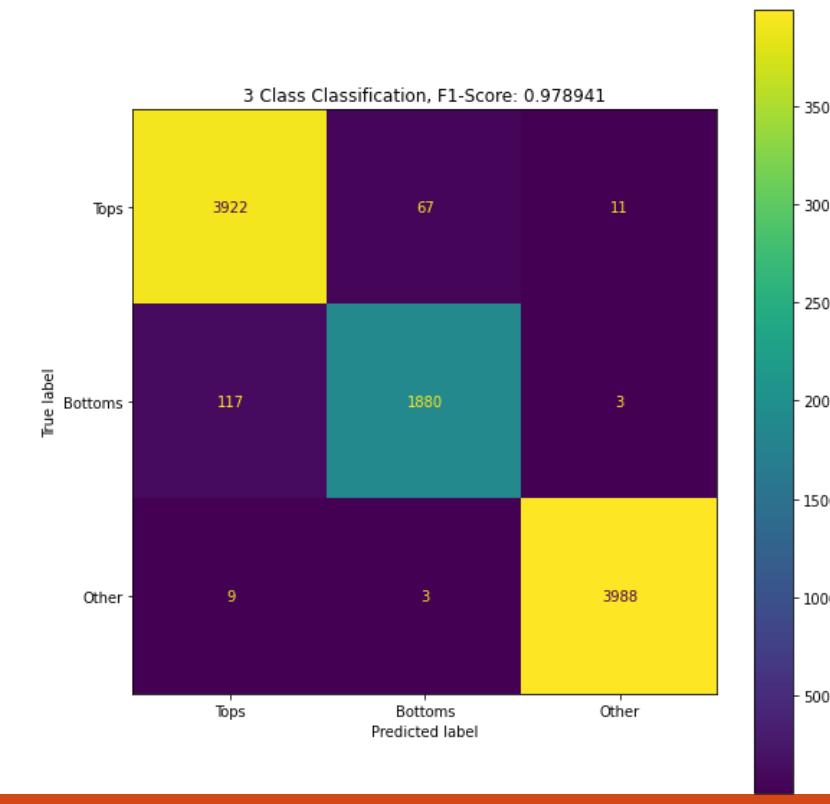
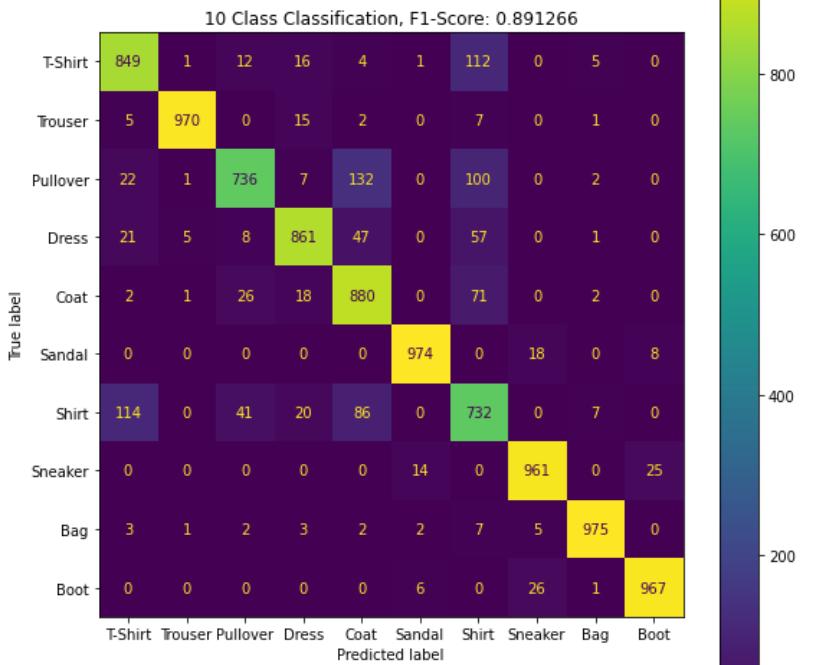
```
[[0. 0. 0. ... 0. 0. 1.]  
[1. 0. 0. ... 0. 0. 0.]  
[1. 0. 0. ... 0. 0. 0.]  
...  
[0. 0. 0. ... 0. 0. 0.]  
[1. 0. 0. ... 0. 0. 0.]  
[0. 0. 0. ... 0. 0. 0.]]
```

Masks

```
[[ 0.  0.  0. ... 0.  0.  1.]  
[-1. -1. -1. ... -1. -1. -1.]  
[-1. -1. -1. ... -1. -1. -1.]  
...  
[ 0.  0.  0. ... 0.  0.  0.]  
[-1. -1. -1. ... -1. -1. -1.]  
[ 0.  0.  0. ... 0.  0.  0.]]]
```

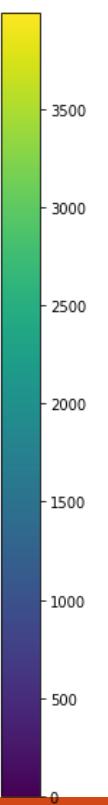
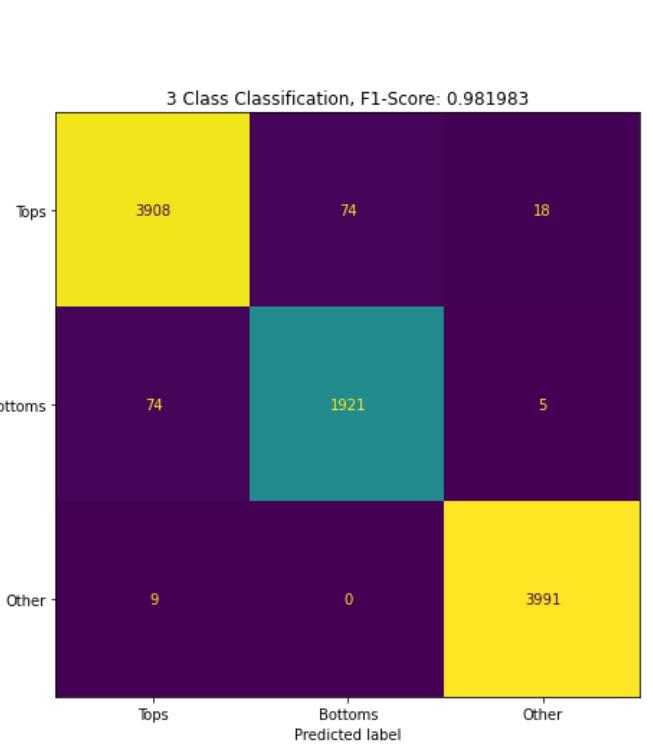
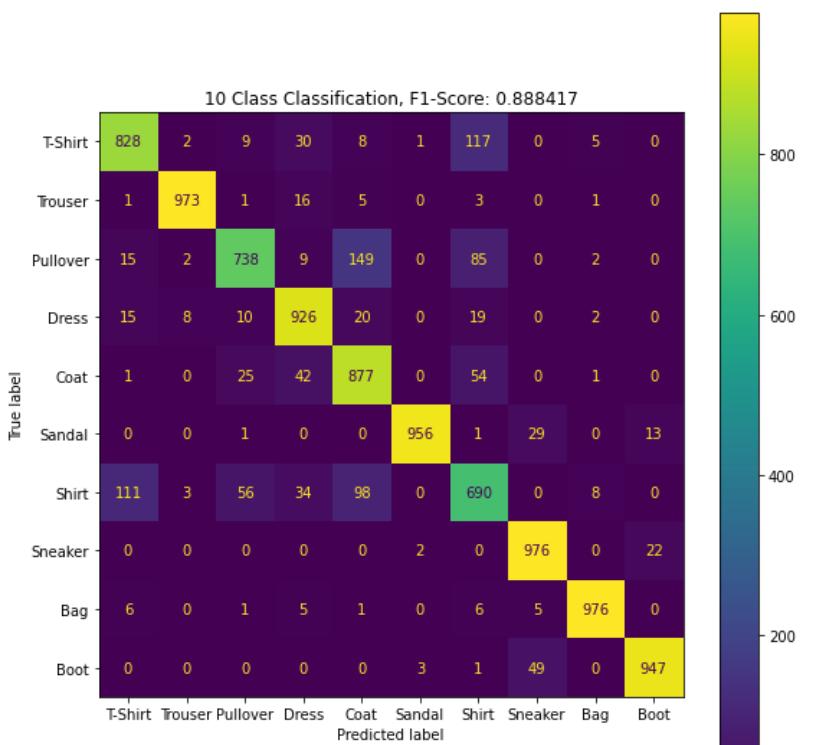
Training with 25% of the data

- Network still works well
- Small performance drop is within what we'd expect from simple sample variation when training the network



Training with 5% of the data

- Again, performance very similar
 - Perhaps a slight drop in the fine-grained task now



Other Considerations

- We can add class weights
 - May need to increase class weights in relation to the number of labels to improve training
 - May also wish to do this to prioritise one task over the other
 - In our example, the autoencoder really exists as a dummy task to support the classification
 - Thus, classification is far more important and could be weighted more
- We can have multiple tasks with partial data
 - And different tasks may have annotations for different samples

How Realistic is this Setup?

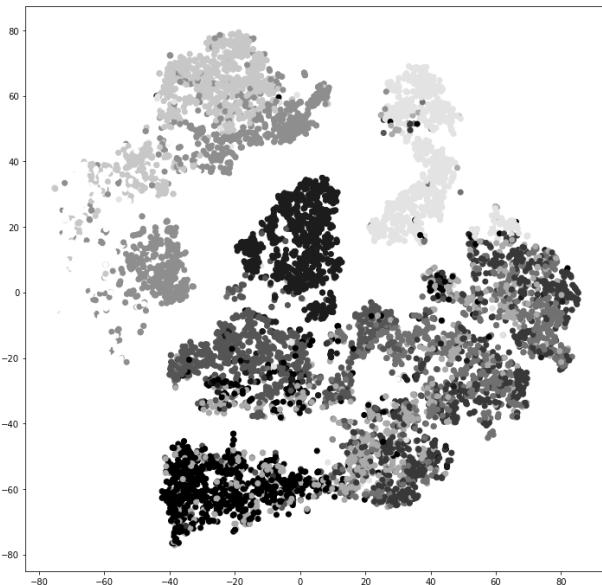
- Note that in this example, we've been greatly helped by the nature of the tasks
 - The “coarse” task is a simplified version of the main task - this helps a lot
- However, this sort of setup is not uncommon
 - Coarse annotation is much easier than fine-grained
 - Such coarse labels can be produced in an automated (or semi-automated) manner
- Unsupervised tasks are also very common in a semi-supervised setup
 - An auto-encoder using all data
 - A classifier using only the available labels
 - Such a network would be geared towards the classification, i.e. no tiny bottleneck in the auto-encoder

CAB420: Variational Auto- Encoders

LEARNING DISTRIBUTIONS

Auto-Encoders

- Learn a compact representation of data by learning how to map from the input to itself via a bottleneck layer
 - Structure of representation is decided by the network (mostly). Though we can influence it using
 - Secondary losses
 - Sparsity constraints
 - Size and shape of the bottleneck

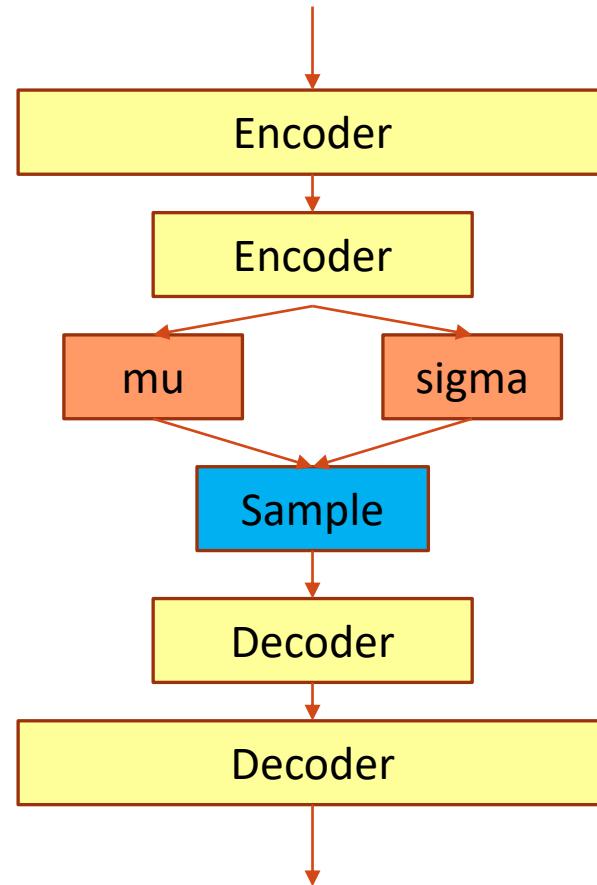


Variational Auto-Encoders

- Generative model
 - i.e. we can sample from it to “create” new data
- Learn a continuous latent space which we can sample from
 - Standard auto-encoders learn a discrete space
 - There can be large gaps in the latent space where no samples can exist

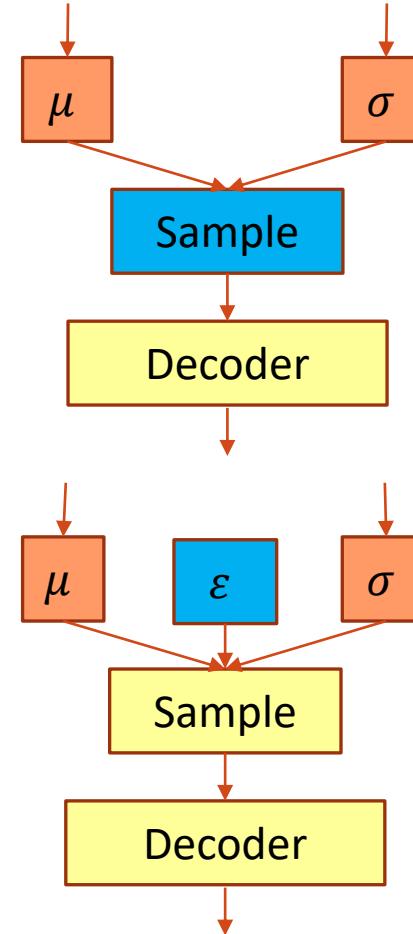
Variational Auto-Encoders

- For an input
 - Compute a mean and std.dev
- The decoder
 - Samples from the distribution described by the mean and std.dev
 - Decodes the sample to try to reconstruct the input
- By sampling we
 - Ensure that for the input, we don't necessarily get the same output
 - Help the decoder to learn the relationship between similar points/inputs



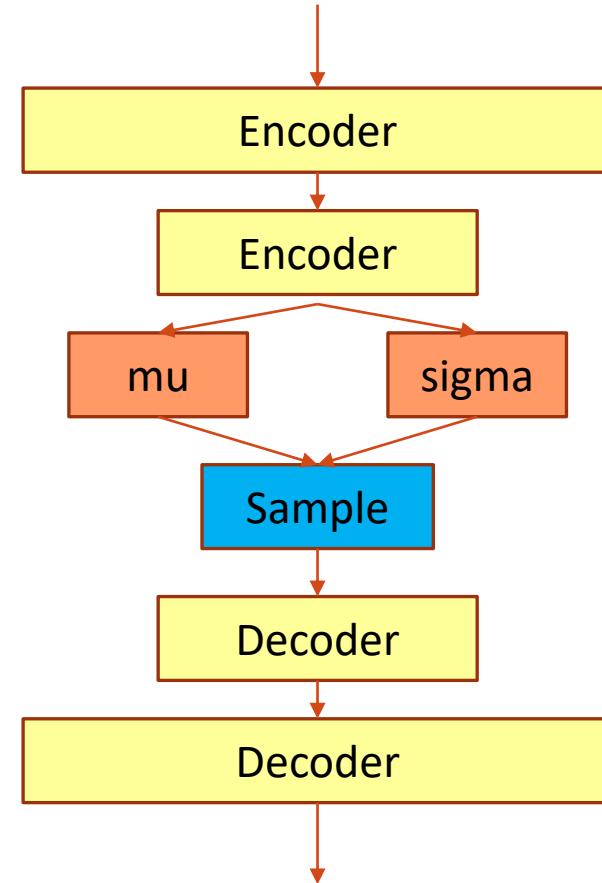
Variational Auto-Encoders

- Sampling and Backpropagation
 - Sampling directly from μ and σ makes backpropagation difficult
- Re-parameterization Trick
 - Leave μ and σ alone
 - Introduce ε
 - Sample becomes
$$z = \mu + \varepsilon\sigma$$
 - Moved the random node to an input
 - No longer in the main back-prop pathway



Variational Auto-Encoders

- By default, we will learn a discontinuous space
- Different classes will be in different regions of the latent space
- No smooth transition from one class to the next
- Place constraints on the learned distributions
- Use KL Divergence
- Seek to make our distribution look like a standard normal distribution
- Ensure that samples are distributed across the latent space
- Prevent the VAE from “cheating” and packing things into separate corners of the space



Variational Auto-Encoders Objective

- Reconstruction Loss

$$L_{recon} = \sum_i^N (x_i - \hat{x}_i)^2$$

- KL-Divergence Loss

- Measures the similarity between two distributions

$$\begin{aligned} D_{KL}[N(\mu(X), \Sigma(X)) || N(0, 1)] &= \\ \frac{1}{2} \sum_k (e^{\Sigma(X)} + \mu^2(X) - 1 - \Sigma(X)) \end{aligned}$$

- We are comparing the learned distribution, $N(\mu(X), \Sigma(X))$, with a unit normal distribution, $N(0, 1)$

- Combined Loss

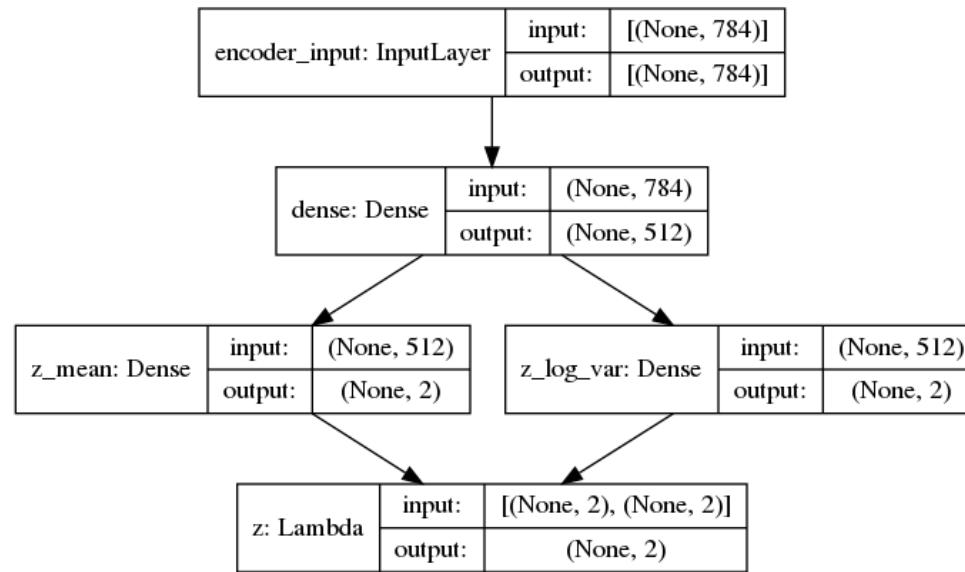
$$L = L_{recon} + D_{KL}$$

An Example

- See ***CAB420_Encoders_and_Decoders_Example_4_VAE.ipynb***
- Our data
 - MNIST
- Our task
 - An autoencoder, reconstruct the original sample

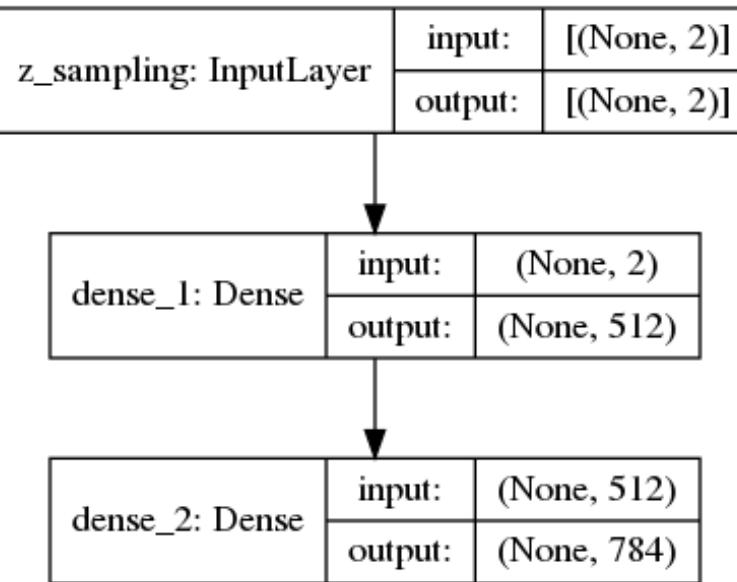
Encoder

- Very simple
 - Vectorised input
 - One common dense layer
 - Branches to learn
 - Mean
 - Variance
 - Sampling layer
 - Take the mean and variance and add a random value to "sample" from the learned distribution



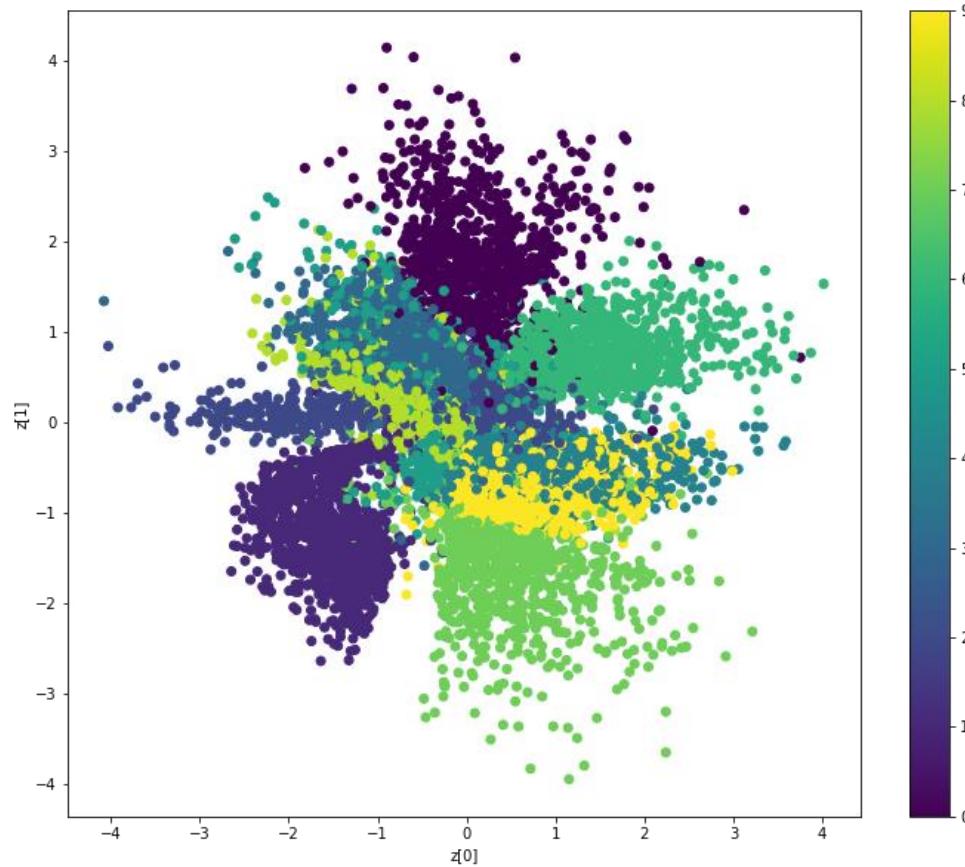
Decoder

- Also, very simple
 - Two dense layers to reconstruct original sample
 - Reconstructing from the "sampled" value
- Encoder and Decoder trained end-to-end
 - Like a regular autoencoder



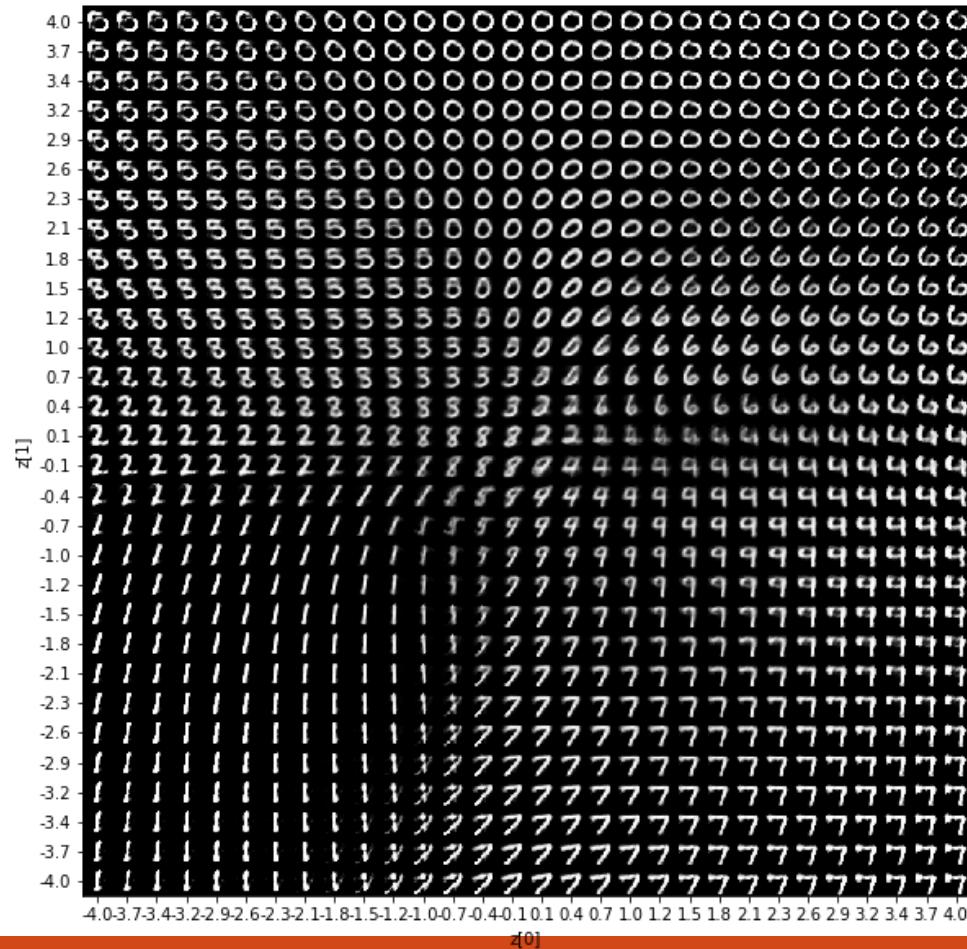
Embedding Space

- Our target distribution is a unit normal distribution
 - Mean of 0
 - Std.dev of 1
- Classes are separated
 - But with no space between them
 - One class blends into the next
 - Model is using the entire feature space
- Other embedding plots we've seen usually contain large spaces
 - Often this is our aim, to separate the classes

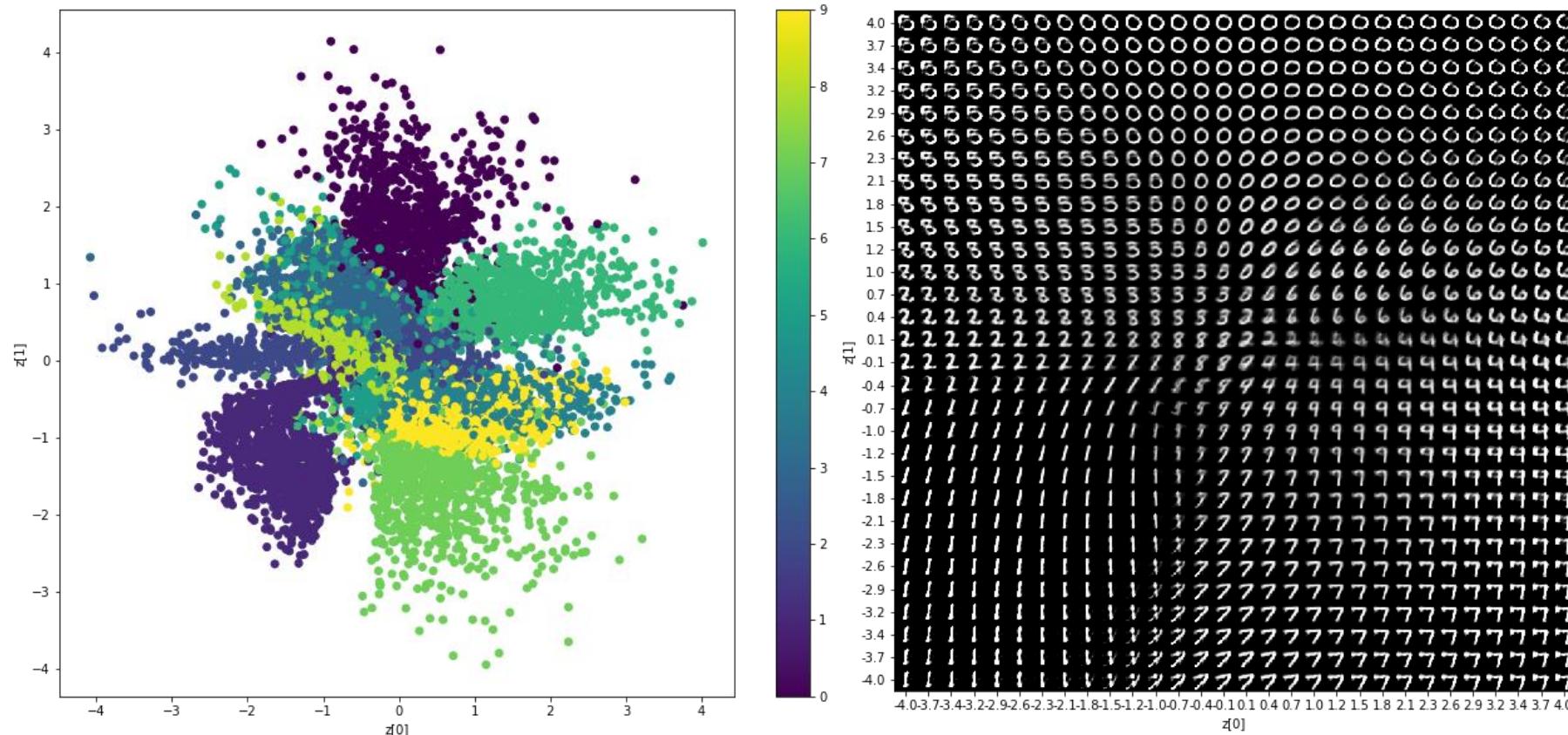


Sampling from the latent space

- We can sample from the learned distribution to create new data samples
- We can also sample in a uniform grid to see how our classes are distributed about the feature space
- At class boundaries, one number warps into another



Sampling from the latent space



Considerations

- We've learnt a 2D VAE
 - We've done this for visualisation
 - Richer representations, and better reconstructions are possible with larger networks and bigger representations
- Could use convolution layers to further improve the network
 - Similar to our other autoencoders, but our representation is captured by a mean and a variance

CAB420: Encoder and Decoder Summary

AND OTHER MUSINGS

Encoders and Decoders

- Encoders: given some data
 - Compute a compact representation of that data
- Decoders: given a compact representation
 - Expand out to a data sample
- Auto-encoders aim to reproduce their input at the output via a compressed representation
 - Bottleneck layer
- Encoder-Decoders can be used to do other things
 - Pixel to pixel transforms
- Variational Auto-Encoders (VAE) extend autoencoders by learning a continuous latent space
 - Generative model, i.e. we can sample from the model to create new data

Encoders and Decoders

- Autoencoders get more compact as we go towards the middle
 - Smaller layers, fewer filters, etc
 - This is important for compression, but if that's not our aim, we don't need to do this

Multi-Task and Semi-Supervised Learning

- Neural networks are very adaptable
 - Can do multiple things at the same time
 - Works best if they're related
- Don't need data for all tasks all the time
 - Semi-Supervised Learning
 - Need to track which samples we have which data for and mask the loss as needed
- Can adjust loss weights
 - More important tasks can be given higher weights
 - Can use to compensate for missing data

Multiple Inputs

- Just like we can have multiple outputs, we can have multiple inputs
 - Have an encoding stage for each
 - Merge some intermediate features
 - Have another learning stage on the combined representation

Other uses of Encoders and Decoders

- Many other forms of encoder-decoder in machine learning
 - Semantic Segmentation (see additional example)
 - Given an input, produce a segmented output that labels each sample with its class
 - Land use classification, classify an aerial image into building, road, grass, etc.
- Can encode multiple inputs to one output
 - For land use classification, encode input RGB and elevation map
- Can decode to multiple outputs
 - For land use classification, estimate land use and elevation from a single RGB

Generative Models

- Very large topic in Machine Learning
- Generative Adversarial Networks (GANs) have demonstrated great performance for a multitude tasks
 - Generator takes noise and synthesises an input
 - Discriminator receives a real or fake (from the generator) input and tries to determine if it's real
 - Networks compete with each other
 - Generator trying to fool the discriminator
 - Discriminator trying to correctly determine what's real and what's fake
 - The conditional GAN (cGAN) provides the generator with extra stimulus
 - Generate data conditioned on some other thing

CAB420: Representation Learning

TRANSFORMING DATA WITH MACHINE LEARNING

Representing Data

- Early in semester we mostly used the raw data “as is”
 - When we didn’t , we used a “feature extraction” method to obtain an alternate representation
- We can use Machine Learning to obtain a different representation
- This may allow for
 - A more compact representation
 - A more discriminative representation
- We’ve considered
 - PCA, unsupervised non-deep learning method
 - LDA, supervised non-deep learning method
 - t-SNE, unsupervised, non-deep learning method used for visualisation
 - Metric Learning, supervised deep-learning method
 - Auto-encoders, unsupervised non-deep learning method

Principal Component Analysis

PCA FOR SHORT

PCA

- PCA is unsupervised
 - No labels needed, just the raw data
- Returns a rotated version of the data
 - Rotated such that the first “new” dimension contains the most variation, second “new” dimension contains the second most variation, etc
 - “Most variation” does not mean most discriminative, or most useful for the task at hand
- Can reduce dimensions by selecting the top N dimensions
 - N may be dictated by a fixed requirement
 - N may be set to capture X% of the total variance (often 90%, 95%, or 99%)
- Standardisation can help (a lot)

PCA

- Can transfer in and out of PCA space, i.e.
 - Start in a high dimensional space
 - Use PCA, reduce dimensions, do some other ML
 - Reconstruct an approximation of the original sample from the compressed version if needed
- Reconstruction accuracy will depend on how many dimensions are retained
 - Fine-grained or high frequency details are the first to be lost
- Ideally, we want many more samples than dimensions
 - PCA may become unstable if we only have slightly more samples than dimensions
 - If we don't have this, many PCA methods will transpose the data
- We can do robustness tests by computing multiple PCA transforms on datasets sampled from our overall data
 - If the PCA transform is consistent, we can claim that we have enough data
 - Similar to the idea of the standard error measured when fitting a regression model

PCA and Regression

- PCA returns a new set of dimensions ordered by the amount of variance
 - Most variance first
- The new dimensions are also orthogonal to one another
 - This means that the returned dimensions are uncorrelated
- This has implications for linear regression which assumes independent predictors
 - Avoid co-linearity
 - Does not guarantee improved performance
- See ***CAB420_Summary_4_PCA.ipynb***
- Review this example in your own time
 - Ask questions in class or via email/slack

Linear Discriminant Analysis

LDA FOR SHORT, AND NOT TO BE CONFUSED WITH THE OTHER LDA,
LATENT DIRICHLET ALLOCATION

LDA

- Supervised method
 - Requires class labels
- Learns a transform that brings samples from the same class closer together, while pushing samples from different classes further apart
 - Ideal for classification
 - Transform computed from within-class and between-class scatter matrices
- Transform will return $C-1$ dimensions
 - C is the number of classes
 - Generally, all $C-1$ dimensions are used
- Transform is not invertible
 - We cannot reconstruct the original signal from LDA

LDA

- Requires sufficient examples per class to estimate scatter matrices
 - Ideally, more samples per class than dimensions
 - Can use PCA followed by LDA to help with data requirements
- Standardisation not needed
 - The scatter matrices capture and compensate for scale
 - Standardisation may result in an axis being flipped, but that's it

Metric Learning

KIND-OF-LIKE LDA, BUT WITH DEEP LEARNING

Metric Learning

- Can be thought of as replicating the idea of LDA with a DCNN
 - Supervised data required
 - No ability to map from learned feature back to the input space
- Use a DCNN as a feature extractor with a loss function to encourage
 - Samples from the same class to be close together
 - Samples from different classes to be far apart
- Contrastive loss considers pairs of images
 - Pass two images through the same encoder
 - Tries to force similar pairs to be closer together than negative pairs by a margin
- Triplet loss uses three images and forms two pairs
 - Pass three images through the same encoder
 - Tries to make the positive pair closer together than the negative pair by a margin
- For both formulations, we can use normalisation to simplify setting the margin

Metric Learning

- What we ultimately compare is a learned embedding
 - Compact representation of the input feature
- We can control the size of the embedding
 - Hyper-parameter chosen at design time
 - Bigger embedding means a richer description
 - May take longer to train
 - May have severe impacts on run-time for some tasks
 - More complex problems (larger number of classes, greater variation) will require a larger embedding
 - More complex problems will also require a larger base (or backbone) network
- Can use fine-tuning, taking an existing network and applying contrastive or triplet loss

Auto-Encoders

KIND-OF-LIKE PCA, BUT WITH DEEP LEARNING

Auto-Encoders

- Encoder-Decoder network
 - Often symmetric in design
 - Input is passed through an encoder to obtain a compressed representation
 - Compressed representation is decoded to reconstruct the input
 - Typically trained use MSE or MAE loss
- Network output is the same as the input
 - No annotation needed
 - Can be seen as unsupervised, or self-supervised
- Often used as a means of pre-training a network
 - Learn an initial representation using un-labelled data
 - Truncate network and fine-tune for desired task
- Can be used in a multi-task setting or semi-supervised setting

What Method When?

SO MANY CHOICES

Things to Consider

- What annotation do you have?
 - Metric learning and LDA require labels
- How much data do you have?
 - Deep learning methods are data hungry, but can also benefit from fine-tuning and augmentation
 - PCA needs more samples than dimensions
 - LDA needs sufficient samples per class
- Do runtime and memory need to be considered?
 - Metric learning and auto-encoders will generally be more demanding
- Do you need to reconstruct the data?
 - PCA and auto-encoders will allow you to reconstruct an approximation of the original data
- Do you need to separate classes?
 - Metric learning and LDA will try to separate classes

An Example

- See ***CAB420_Summary_5_Learning_Representations.ipynb***
- Learn representations of CIFAR-10 and evaluate their performance with a simple classifier
- Considers the following:
 - Raw pixel and HOG representations
 - PCA with raw pixel data and HOG data
 - LDA with raw pixel data and HOG data
 - An autoencoder
 - Metric Learning (with triplet loss)
 - A multi-task network (auto-encoder with semi-supervised classification loss)
- In all cases, a compact representation is learned, which is fed to a simple, single layer neural network to evaluate the representation
- Review this example in your own time
 - Ask questions in class or via email/slack

CAB420: Clustering Summary

GROUPING SIMILAR POINTS

Clustering

- Unsupervised learning method
- Groups related points into clusters
- “Related” points are determined based on a distance metric
 - Nearby points are similar
- Two methods considered
 - K-Means
 - GMMs
- Both methods require us to select K, the number of clusters
 - Selecting K can be tricky
- Clustering may be used for
 - Analysis and exploration, trying to understand a dataset
 - To identify patterns
 - To find anomalies

K-Means

- Splits data into K clusters
- You need to select K. You can use
 - Prior information, maybe you know how many clusters there are?
 - A heuristic method, such as the “elbow” of the reconstruction error plot, or the minimum of the approximate BIC
- Resultant clusters will be “spherical”
 - Approximately uniform in size and shape
 - All clusters have the same covariance (spread)
- Performs hard assignment
 - A point belongs completely to one cluster
- Can be sensitive to
 - Outliers
 - Scale (can resolve with standardisation)
- Starts from a random set of initial clusters
 - Different runs (with different random seeds) may give different results

GMMs

- Like K-means, it
 - Splits data into K clusters
 - You need to select K. You can use
 - Prior information, maybe you know how many clusters there are?
 - A heuristic method, such as the minimum of the BIC
 - Starts from a random set of initial clusters
 - The initial set is actually found by running K-means
 - Sensitive to scale (can resolve with standardisation)
- Unlike K-means
 - Clusters can have different sizes, shapes and densities
 - Each cluster has its own covariance
 - Leads to many more parameters per cluster
 - Less sensitive to outliers
 - Can perform soft assignment
 - A point can be 25% in one cluster and 75% in another
 - Can measure how likely a point is overall given the learned distribution

Which Clustering Method?

Depends on:

- Your task
 - Anomaly detection
 - Segmentation
 - General Analysis
- Data characteristics
 - Consistent Density
 - Sparseness
 - Noise
- Computational Considerations
 - Memory, runtime or both
- Selection can at times be a judgement call
 - Sometimes there is a clear “best choice”, sometimes there is not

An Example

- See ***CAB420_Summary_6_Diarisation.ipynb***
- Diarisation of audio of meetings
 - Determining who spoken when
 - Train a triplet network to extract embeddings of speakers
 - Cluster embeddings to group audio into individual speakers
- Example considers
 - K-means
 - GMM
 - Uses prior knowledge (true number of speakers) to select K
- Review this example in your own time
 - Ask questions in class or via email/slack

CAB420: Multi-Task and Semi- Supervised Learning Summary

DOING MULTIPLE THINGS AT ONCE

Multi-Task Learning

- Neural networks can learn multiple tasks at once
- Each task has
 - Its own output
 - Its own loss function
 - Its own set of ground truth data
- The overall network loss is just the sum of individual losses
 - Can apply weights to the losses to give some tasks more emphasis than others
- A network can branch at any point to allow for multiple losses, but generally
 - All tasks have a common “backbone”, which is most of the network
 - Individual task “heads” are more compact
 - Individual tasks branch at the same point
 - But they don’t have to
- Multi-task learning works best when tasks are related
 - One task provides regularization for the second

Semi-Supervised Learning

- Deep networks learn by comparing an expected value (ground truth) with a value predicted by the network
 - The gradient of the error is computed, and backpropagated to update the network
 - Gradients are averaged over all samples in a batch
- What if we don't have labels for all samples in the batch?
 - Just get the error for those that we do, ignore the rest
 - This is semi-supervised learning
- Semi-supervised learning is commonly used in a multi-task setting and with self-supervised/unsupervised tasks
 - Learn the “backbone” from all data using an auto-encoder
 - Learn a supervised task “head” from the small amount of available labels
 - Self-supervised/unsupervised tasks are often used to facilitate learning from all data and improve robustness
 - These tasks may then be ignored or layers removed once the network is trained

Semi-Supervised Learning

- Need to modify the loss function to be aware of missing data
 - An easy approach is to embed a mask in the ground truth. Consider a classification task:
 - Ground truth should be a one-hot encoding, all values are 0 or 1
 - Values of -1 in the ground truth can indicate missing data
 - Loss function can be modified to filter samples first, then compute the loss on the available data
 - Can have multiple semi-supervised tasks
 - With different data available for each

An Example

- See ***CAB420_Summary_5_Learning_Representations.ipynb***
- Learn representations of CIFAR-10 and evaluate their performance with a simple classifier
- Considers the following:
 - Raw pixel and HOG representations
 - PCA with raw pixel data and HOG data
 - LDA with raw pixel data and HOG data
 - An autoencoder
 - Metric Learning (with triplet loss)
 - **A multi-task network (auto-encoder with semi-supervised classification loss)**
- In all cases, a compact representation is learned, which is fed to a simple, single layer neural network to evaluate the representation
- Review this example in your own time
 - Ask questions in class or via email/slack

CAB420: Sequences and ML

AND WHY WE NEED TO TREAT
THEM DIFFERENTLY

Sequential Data

- A lot of our data is actually sequential
 - Text
 - Audio
 - Video
 - Biomedical Signals
- Sequences impose an order on our data
 - The same data, in a different order, can mean different things
 - Consider a video of a door opening and a video of a door closing
 - One's a reverse of the other
 - Same data, different order

Challenges with Sequences

- How do we capture order
 - Or should we just ignore it?
- Sequences can be different lengths, but all of the models we've explored so far require a fixed dimensional input
 - Do we resample data?
 - How do you resample text?
 - Do we pad data?
 - Can make data large and unwieldy
 - Do we truncate data?
 - Which bits do we get rid of?

Sequences in CAB420

- We've already covered a few examples of sequences in CAB420:
 - Classifying beer names
 - CAB420_Classification_Example_2_Multi_Class_Classification.ipynb
 - Convert text to word embeddings, and concatenate embeddings to obtain a fixed size representation
 - Pad short sequences, omit long sequences, to obtain a standard length representation
 - Classifying twitter sentiment
 - CAB420_Summary_3_Text_Classification.ipynb
 - Use bag-of-words to obtain a fixed length representation
 - Classifying topics in text
 - CAB420_Classification_Bonus_Example_BOW.ipynb
 - Use bag-of-words to obtain a fixed length representation
- We'll revisit these methods briefly

Classifying Twitter Data

MODERATION CAN'T BE THAT HARD

Classifying Tweets

- Our task
 - Classify tweets into positive or negative sentiment
 - Supervised Learning
- Our data
 - The text that someone felt compelled to share
 - A label as to whether the tweet was positive or negative

Considerations

- Our input data is text
 - Need a numeric representation for text
- Different numbers of words/characters per sample
 - Need to standardise input length

Two Approaches

- Both will use similar pre-processing
 - Convert to lowercase
 - Remove punctuation
 - Possibly remove plurals, or specific words
 - Tokenise document
 - Extract out each word on its own
- Word embeddings
 - Convert tokenised words to embeddings, and concatenate embeddings
- Bag-of-Words
 - Build dictionary and convert set of token into a histogram

A Naïve Approach

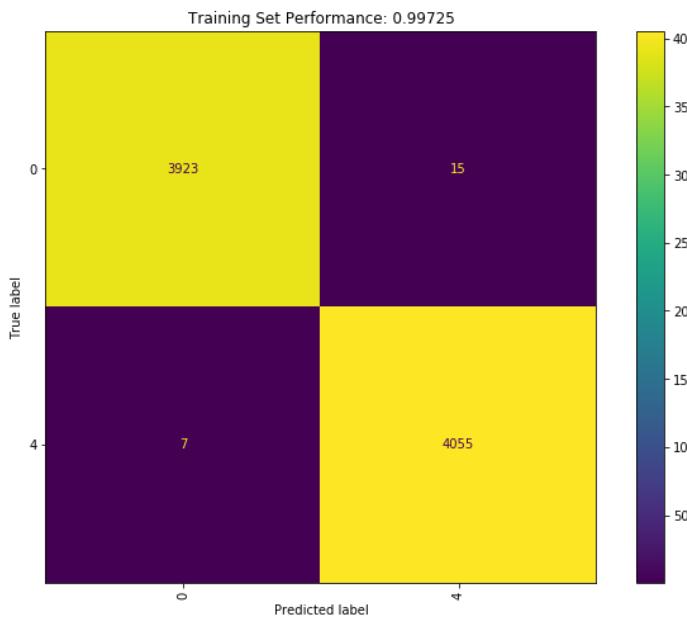
- We've already played with sequential data once
 - Multi-class classification of beer types
 - Let's do that again
- The process:
 - Transform words into an embedding
 - Feature for a document becomes a list of embeddings
 - All documents are the same length, padded with 0's when there aren't enough words
 - Train a classifier

An Example

- See ***CAB420_Sequences_Additional
Example_1_Sequence_Classification.ipynb***
- Our approach follows what we did with beer names earlier in semester
 - Convert text into a vector representation
 - 100 dimensions per word
 - Maximum of 38 words per tweet
 - Features of size 3,800
 - We take the first 10,000 samples (8,000 training, 2,000 testing) for our models
 - You can use more, but things like SVMs will get very slow
 - Try different classifiers
 - CKNN, SVM, Random Forest

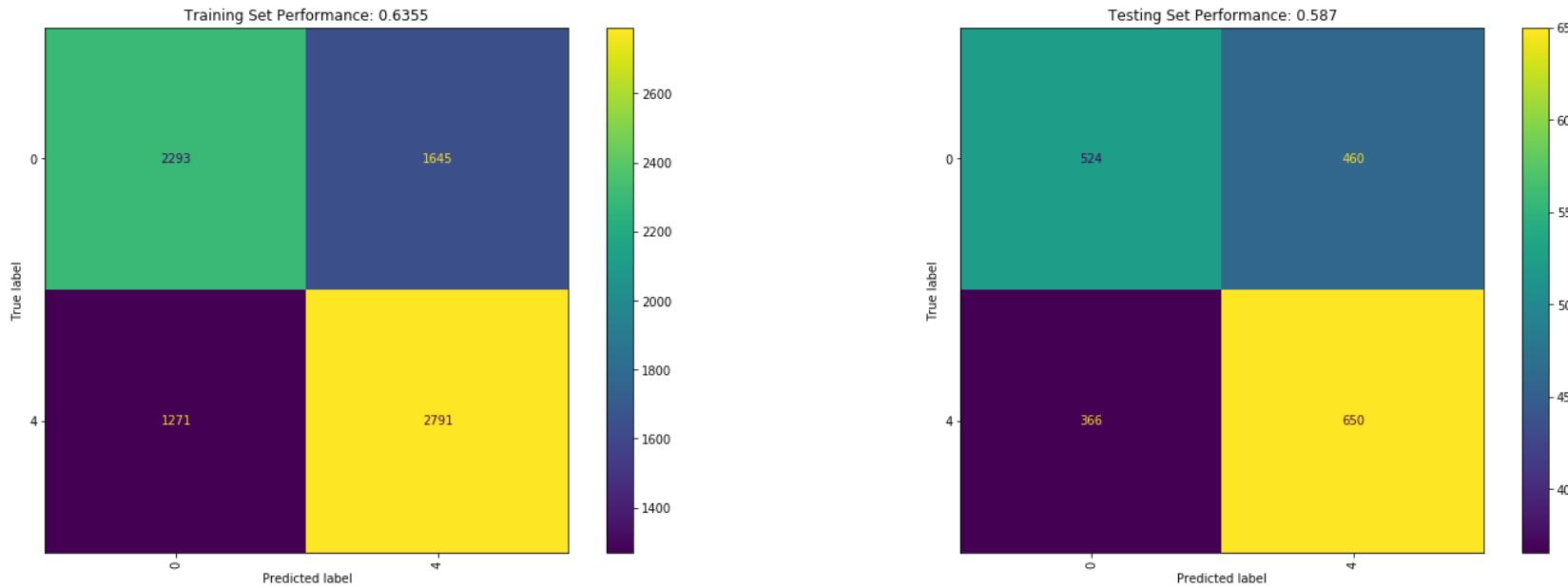
CKNN Performance

- 100 neighbours, inverse weighting, Euclidean distance
 - Performance is little better than chance



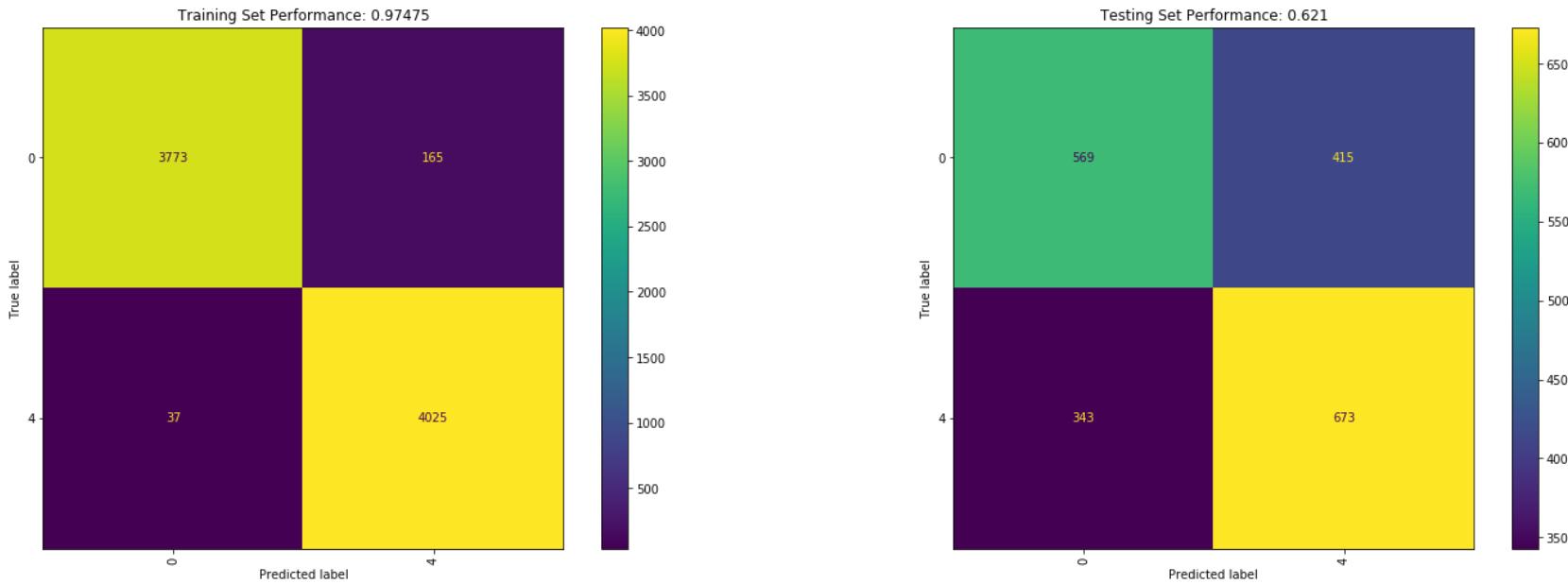
SVM Performance

- Linear kernel, C = 1
 - Slightly better than CKNN
 - Still ordinary



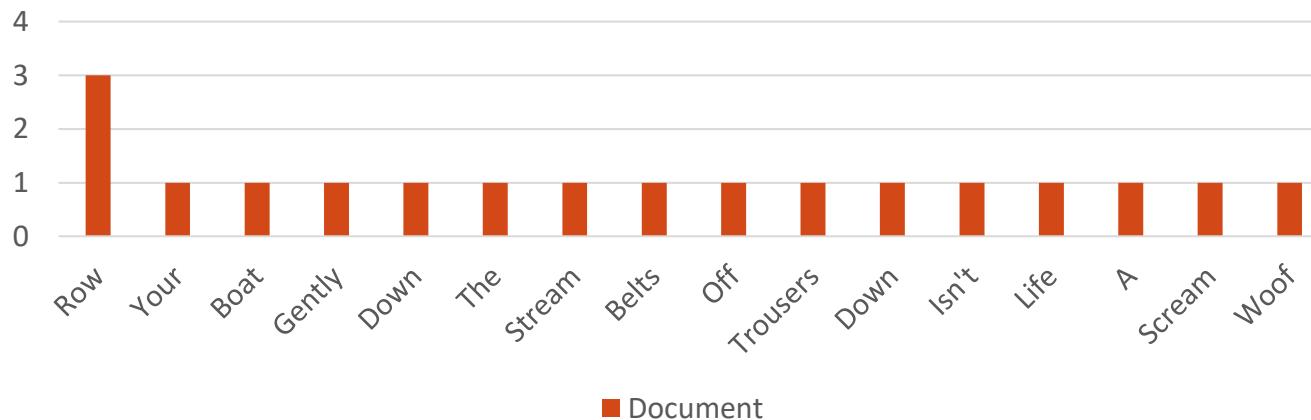
Random Forest

- 250 trees, max depth of 15
 - Improving performance, though still not great



Bag of Words (BoW)

- Encode a document as a histogram that measures word occurrences
- Example document
 - “Row, row, row your boat, gently down the stream, belts off, trousers down, isn’t life a scream, woof!”



Bag of Words

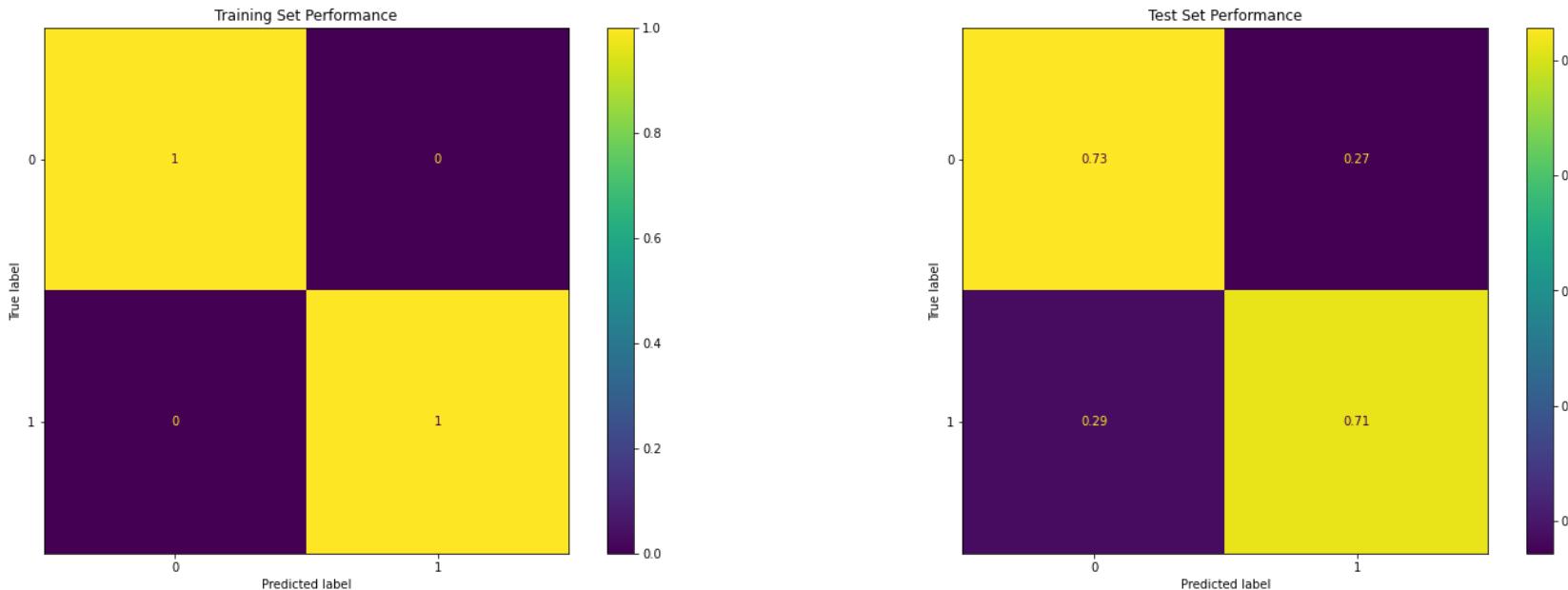
- Transform variable length data into a fixed length representation
- Destroy order of that data
 - Hence the “bag”
- Histograms can be very large and sparse
 - There are lots of words
- Can tune the size of the dictionary
 - Can remove rare words
 - But rare words can be very informative
 - What’s rare?
 - Can remove really common words
 - If they’re really common, they’re probably not that informative
- Methods can be sensitive to the size of the document
 - Small documents will have small bags, big documents will have big bags
- Can normalize based on document frequency
 - Often also use inverse frequencies
 - Term Frequency-Inverse Document Frequency (tf-idf) is a common weighting scheme

An Example

- See ***CAB420_Summary_3_Text_Classification.ipynb***
- Using the following settings
 - Minimum document frequency = 0.01%
 - Maximum document frequency = 25%
- Leads to a dictionary of size 11,290
 - Each tweet, which is 140 characters long, becomes a histogram of size 11,290
 - Features are very sparse
 - Contain lots of 0's

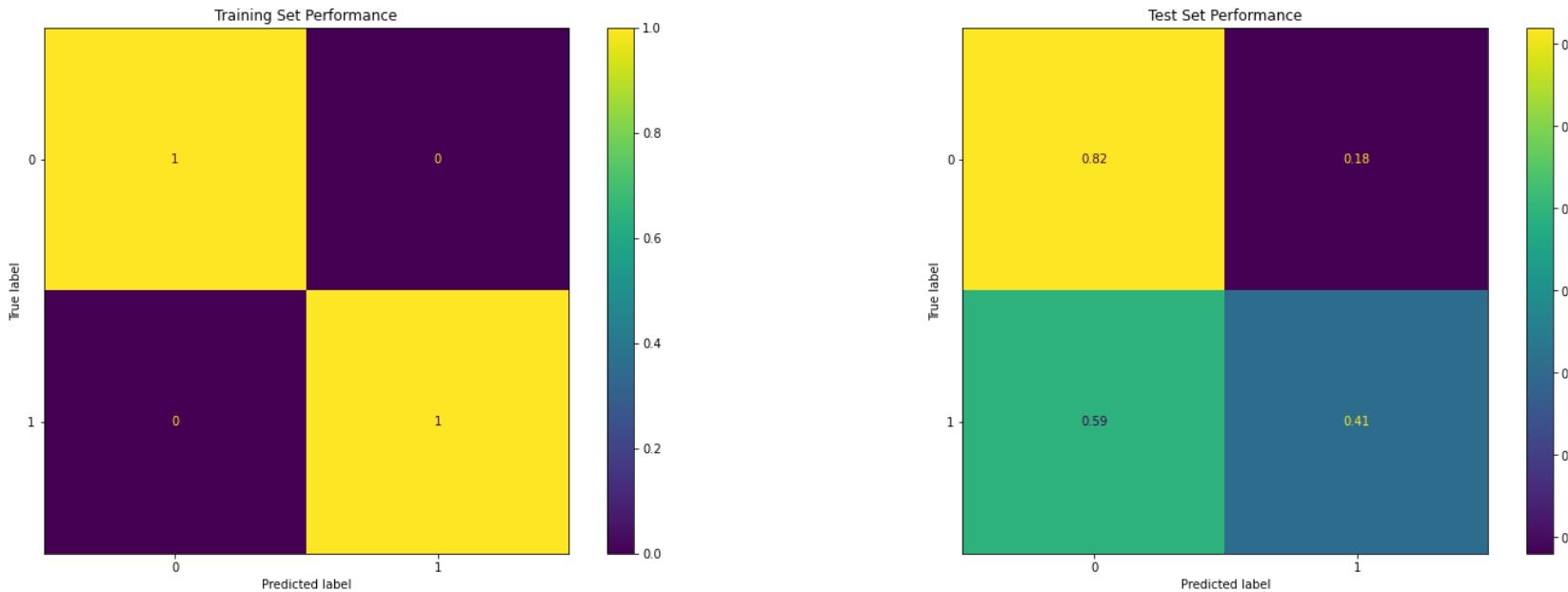
SVM Performance

- F1 of 0.72 on the test set



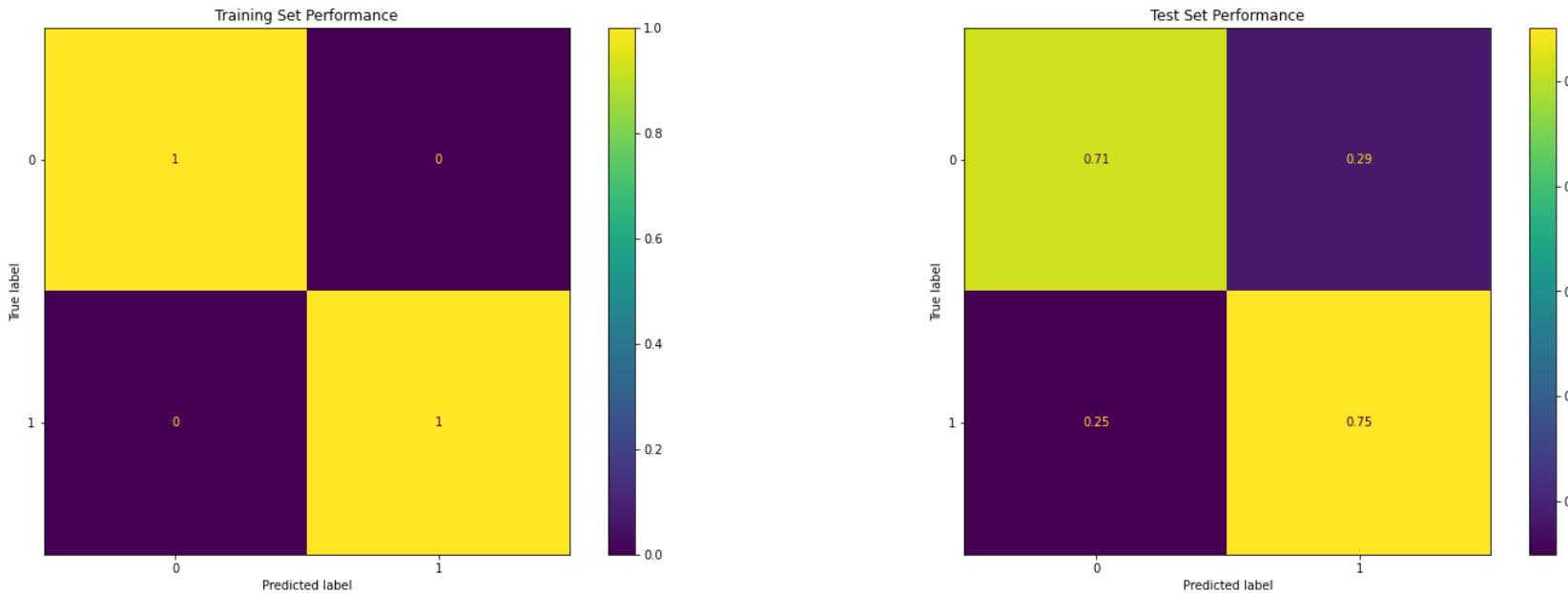
CKNN Performance

- F1 of 0.60 on the test set



Random Forest

- F1 score on test set of 0.73



Problems with our Approaches

- Our features are huge
 - 3,800 dimensions for our first approach
 - And that's with a small word embedding
 - 11,290 for the bag of words model
 - Could be more selective with word frequency
 - May start to eliminated useful information
- A lot of the features are 0's
 - "padded" features in the first approach
 - Sparse histograms with bag of words

Problems with our Approach

- Order of words is over-emphasised in our first approach
 - Consider these tweets, encoded using a simple embedding size 1
 - “This is a positive tweet” -> [1, 2, 3, 4]
 - note, “a” is ignored as a small word
 - “Positive tweet, this is” -> [3, 4, 2, 1]
 - “Negative tweet, this is” -> [5, 4, 2, 1]
 - The order of the words means that “Positive tweet, this is” and “Negative tweet, this is” are very similar when we compare feature vectors
- Order of words is totally ignored in the second
 - **BAG** of words, order is destroyed by design to transform to a fixed length representation

Problems with our Approach

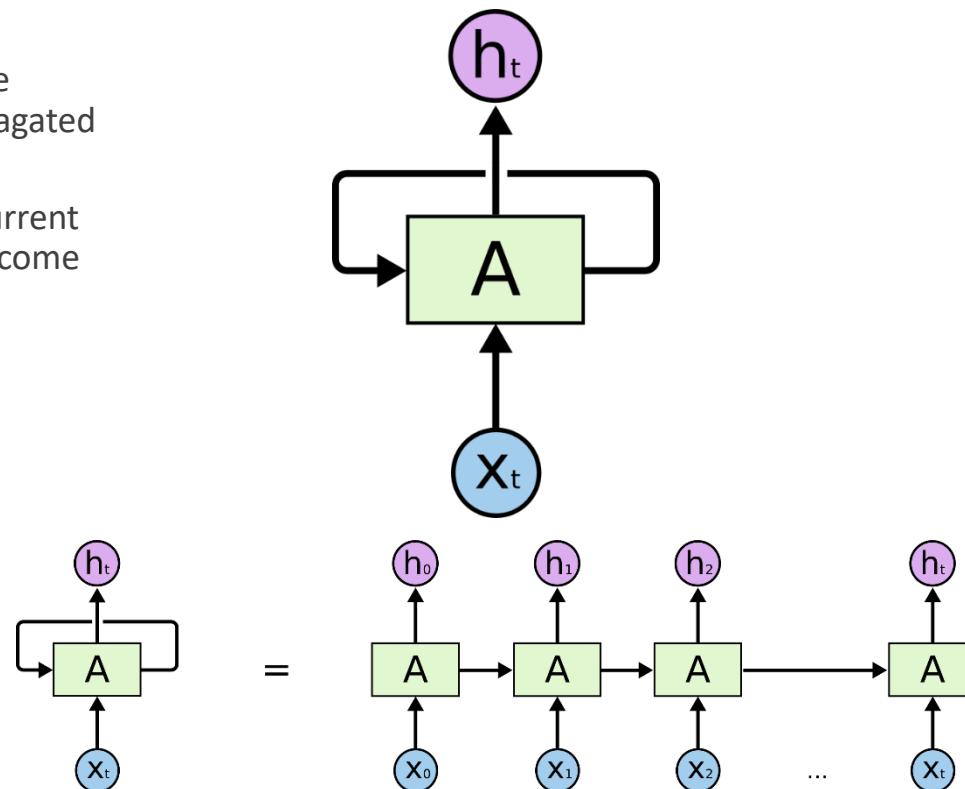
- Learning is slow
 - Any machine learning model can get slow when dealing with very large features
 - SVM training is not quick, even for a very small subset of data
 - The full dataset takes many hours to train
 - Doesn't benefit from GPU acceleration either
 - This makes optimising models a bit more painful
 - Note that a grid search is not any harder to do, just slower

CAB420: Recurrent Neural Networks

DEEP LEARNING STRIKES AGAIN

Recurrent Neural Networks

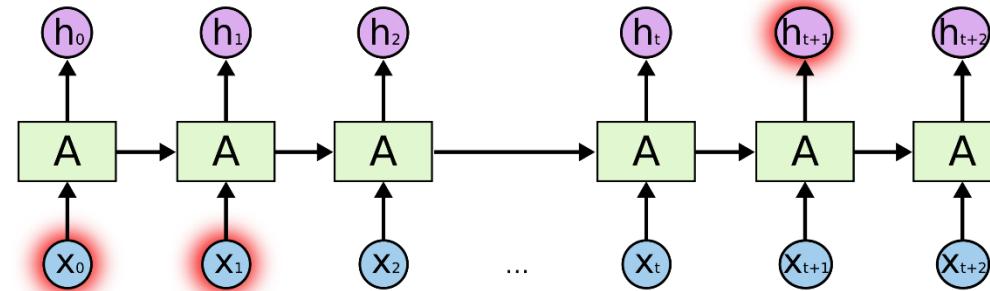
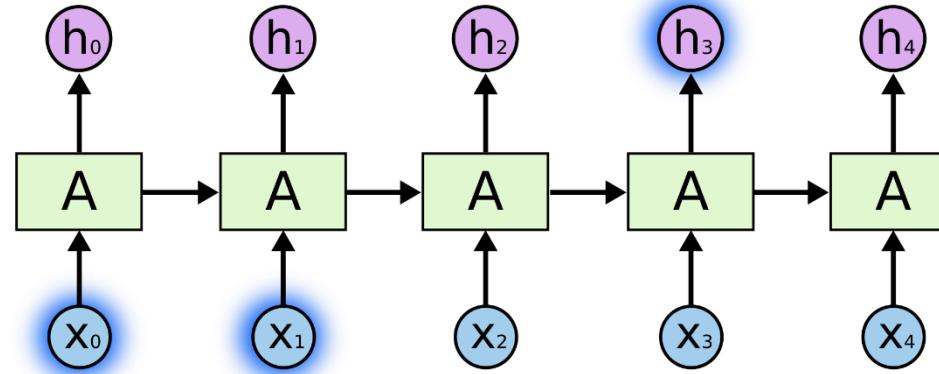
- Networks with loops
 - Some information from the previous time-step is propagated to the next
 - Output is dependant on current information and what has come before



Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Recurrent Neural Networks

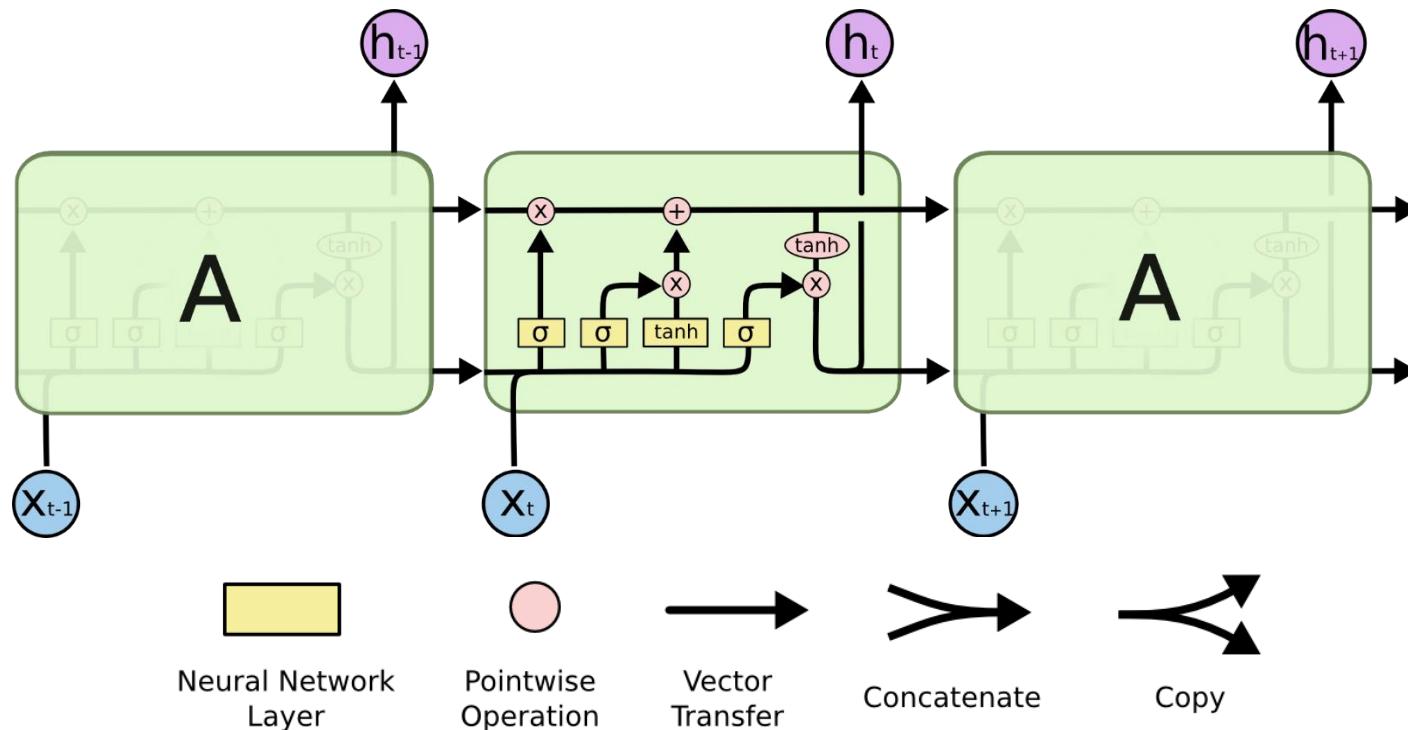
- Work well when things that we need to associate are close
- Gets harder as information we need to associate becomes further apart



Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTMs

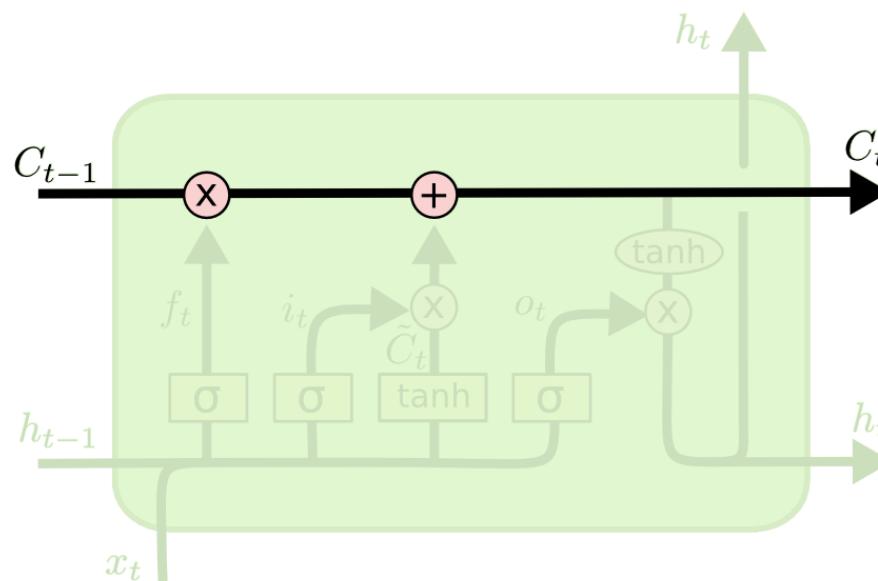
- Long Short Term Memory



Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTMs

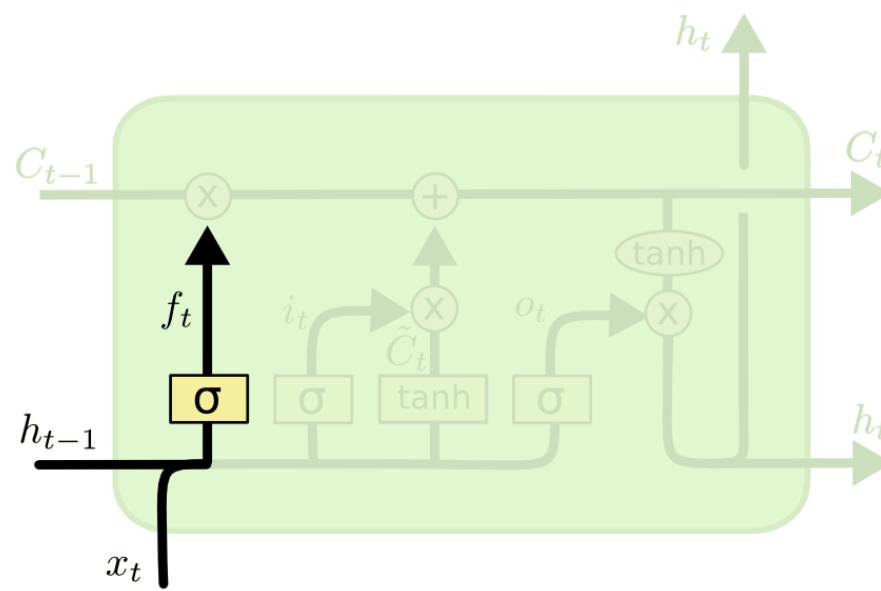
- The cell state
 - Information that is passed from previous time steps
 - Is updated by the cell
 - The cell learns how to update itself



Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTMs

- The forget gate
 - Determine what to keep and what to forget from the previous cell state

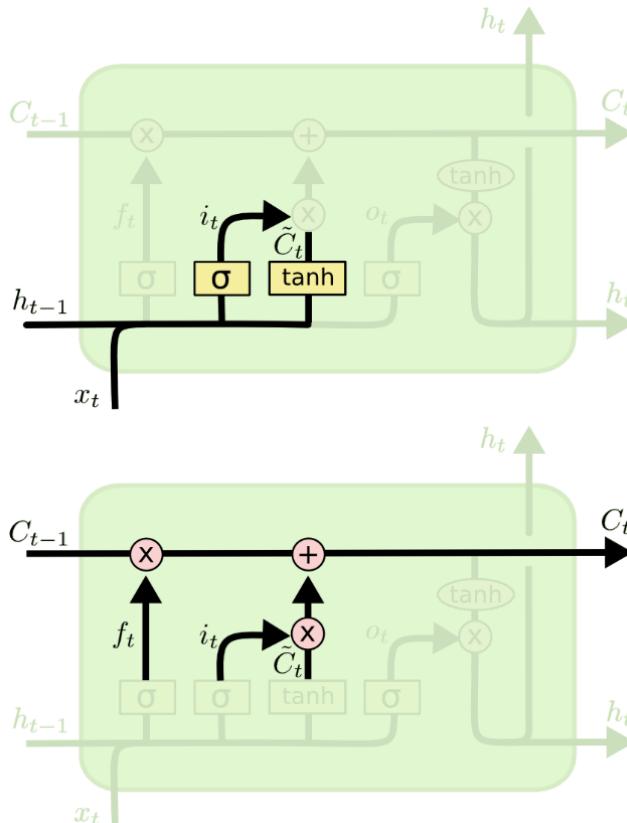


Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTMs

- Input update gate

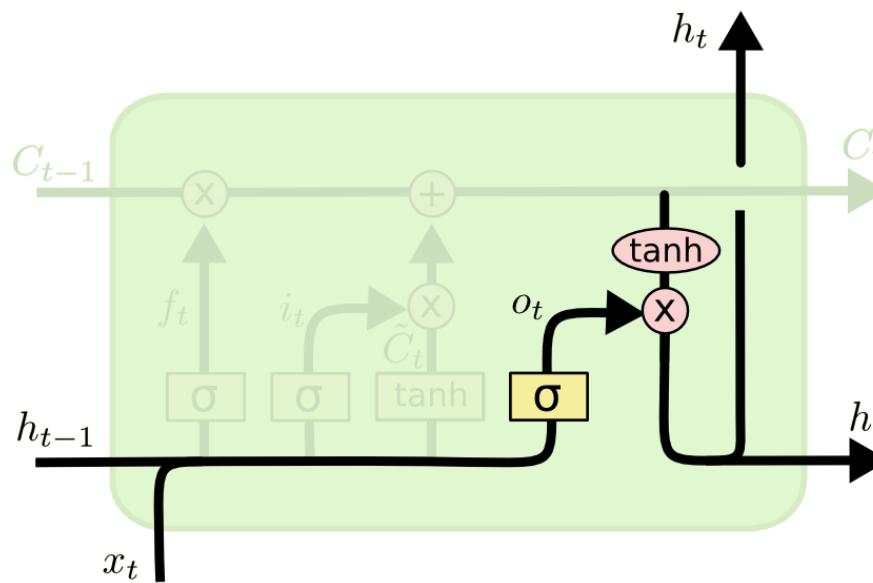
- Look at the current input and decide what new information to incorporate in the cell state



Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTMs

- Output Gate
 - Determine what to output
 - Filtered version of the cell state
 - Note that output is propagated to the next time step as well



Figures taken from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

An Example

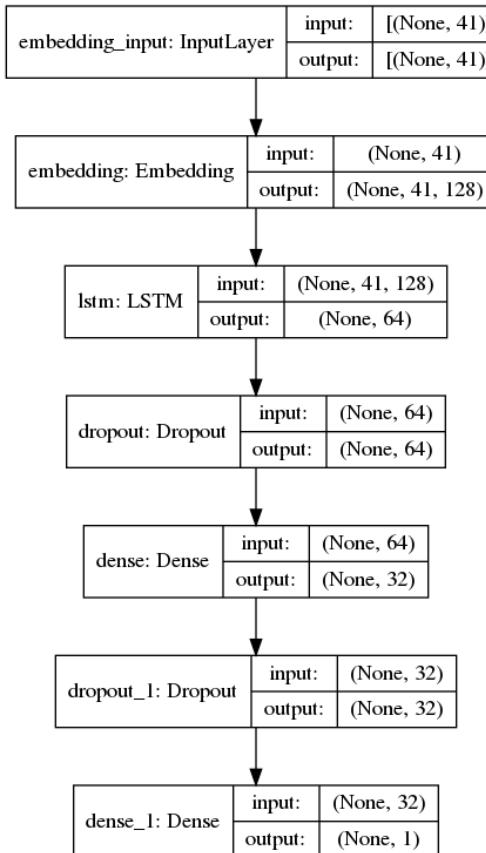
- See ***CAB420_Sequences_Example_1_Recurrent_Neural_Networks.ipynb***
- We're still using twitter data
 - 60,000 samples now
 - 50,000 for training, 10,000 for testing

Building Our Model

- Our input
 - Text data from twitter
 - Need to convert this to something numeric
 - Word Embeddings
- Our output
 - Positive or Negative sentiment
 - Binary Cross Entropy
- Map from a sequence to a single output

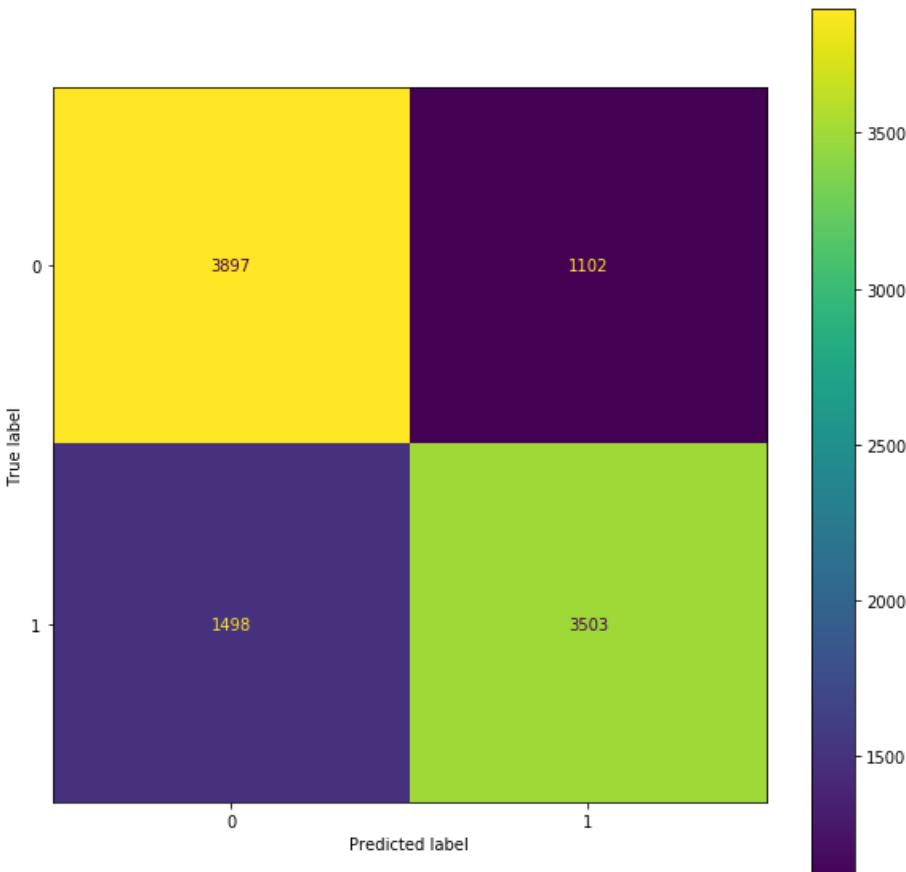
Our Network

- Raw words in
 - Padded to maximum sequence length of 41
- Embedding learnt by the network
 - Convert words to numbers
- LSTM to map sequence of words to 64-dimensional vector
- Dense layers for classification follows
 - Binary Cross Entropy Loss
 - Binary output: positive or negative sentiment



Performance

- Trained for 10 epochs
 - 74% testing accuracy
 - 98% training accuracy
 - Overfitting somewhat

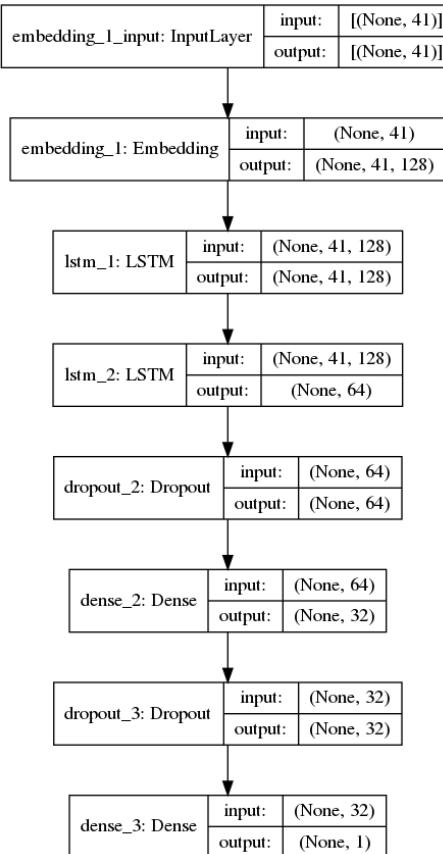


Stacking LSTMs

- We can stack LSTMs like we do other layers
 - Typically, we don't go too deep
 - Networks get harder to train with very deep LSTMs
 - Gradients get harder to propagate and can vanish easily
 - BatchNorm generally not used
 - Dropout can be though (and can be used internally within the LSTM)

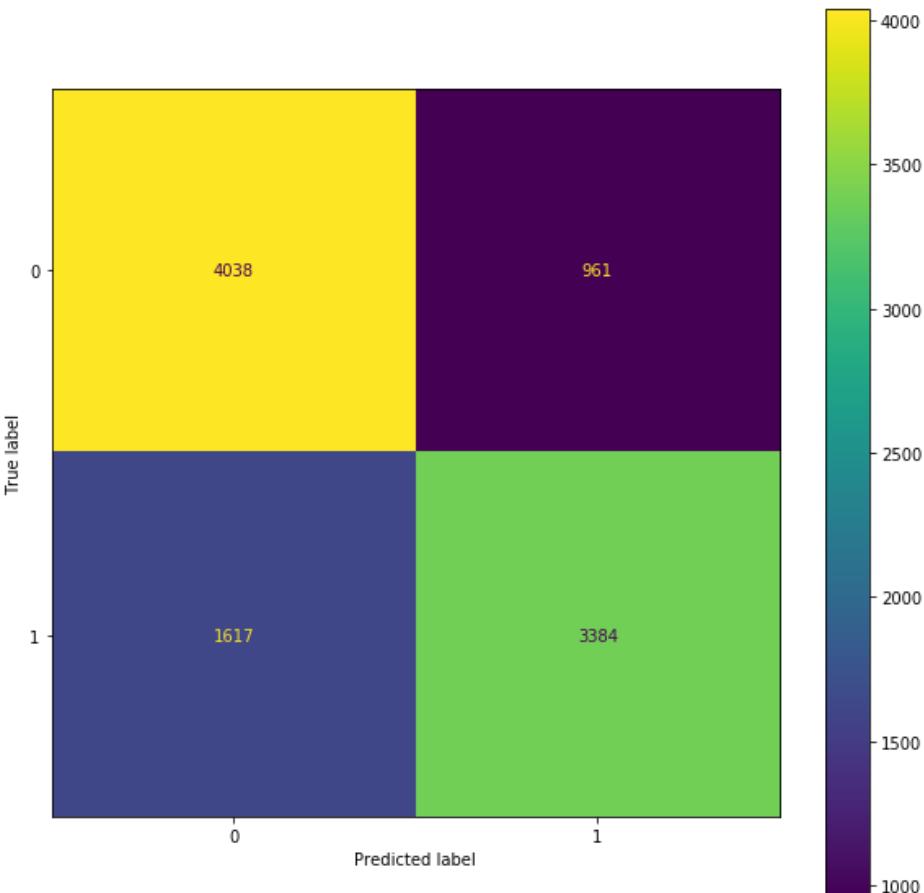
Our Network

- Two LSTMs
 - First LSTM returns a sequence
 - Second LSTM returns a vector
- Simple classifier follows the LSTMs
 - Binary Cross Entropy Loss



Performance

- Trained for 10 epochs
 - 74.2% testing accuracy
 - 98% training accuracy
 - About the same as before
 - Dataset is not big enough/complex enough to warrant larger network and further complexity



CAB420: Predicting Sequences

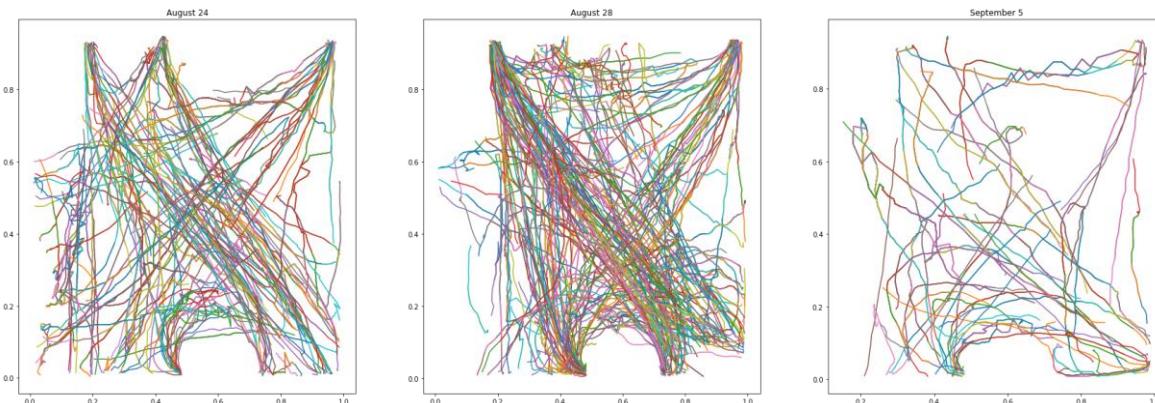
SEQUENCE IN, SEQUENCE OUT

Predicting Sequences

- LSTMs can predict:
 - A single output for the whole sequence
 - An output for each element in the sequence
 - i.e. they can predict a sequence
- Sequence to Sequence networks
 - Map from one sequence to another
 - Can predict an output for each element in a sequence
 - We saw this with our stacked LSTMs
 - Output sequence can be
 - Learned via regression, we'll do this for trajectory prediction
 - Learned via classification, frame-by-frame event classification would be an example

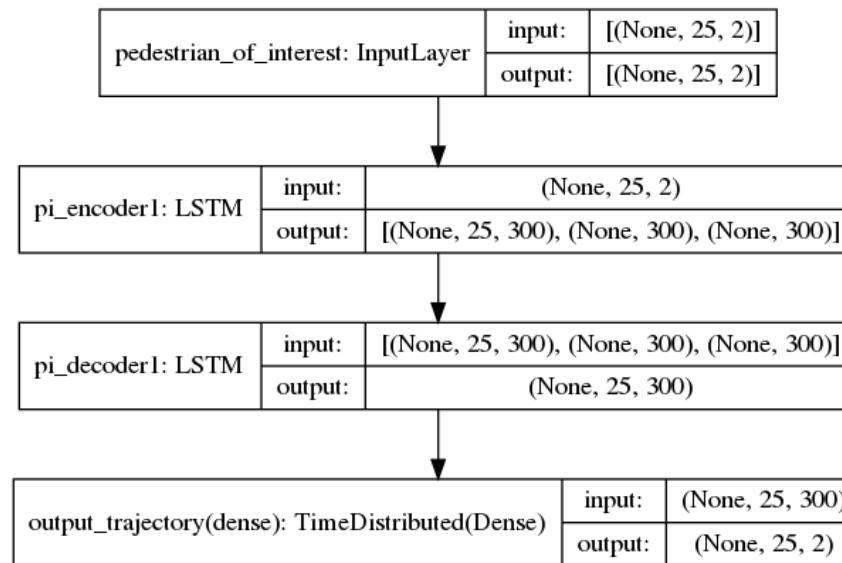
An Example

- See ***CAB420_Sequences_Example_2_Sequence_to_Sequence_Prediction.ipynb***
- Our data
 - Person trajectories
- Our task
 - Predict motion



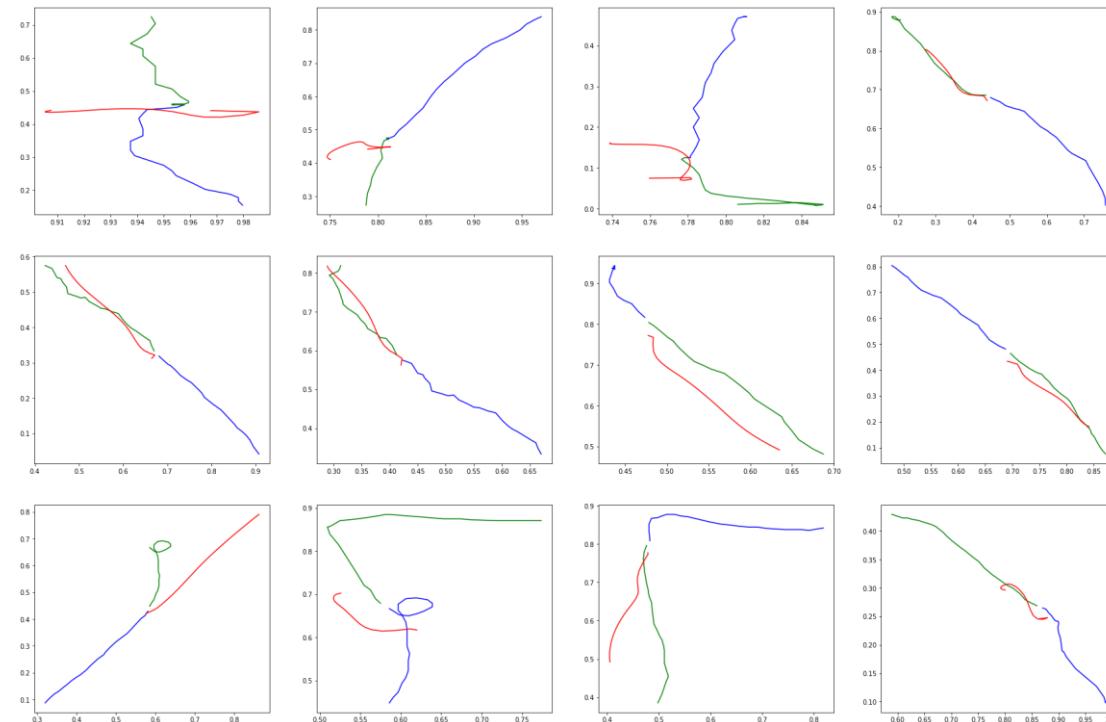
Our Network

- Stacked LSTMs
 - Encoder
 - Decoder
 - Output of both is a sequence
- Output layer is a dense layer that maps from LSTM embedding to position
 - Same dense layer is used at all locations
 - Achieved by TimeDistributed wrapper
- MSE loss
 - We're essentially doing regression from an input to an output trajectory



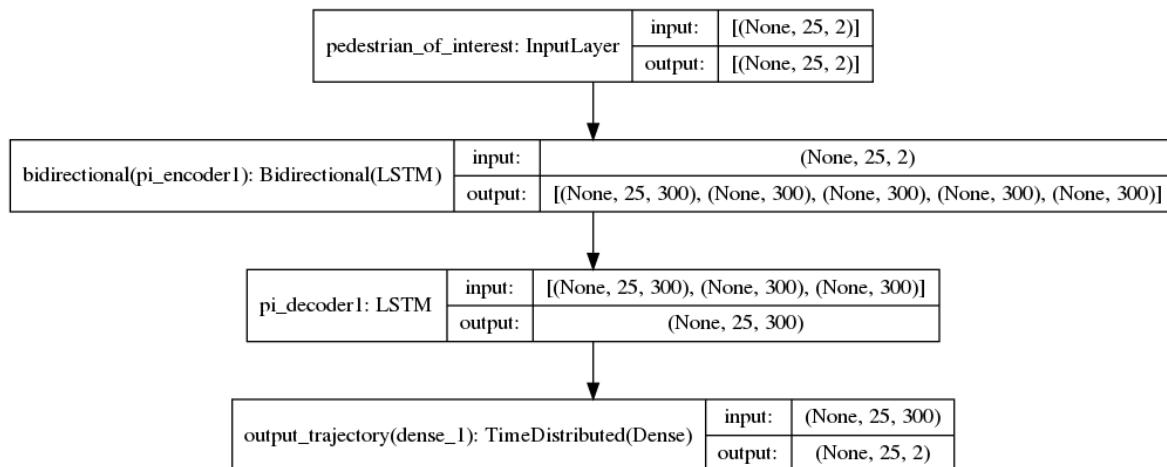
Performance

- Blue: Input path
- Green: Actual future path
- Red: Predicted future path
- Mixed results
 - Discontinuities from input to output
 - Some wildly inaccurate



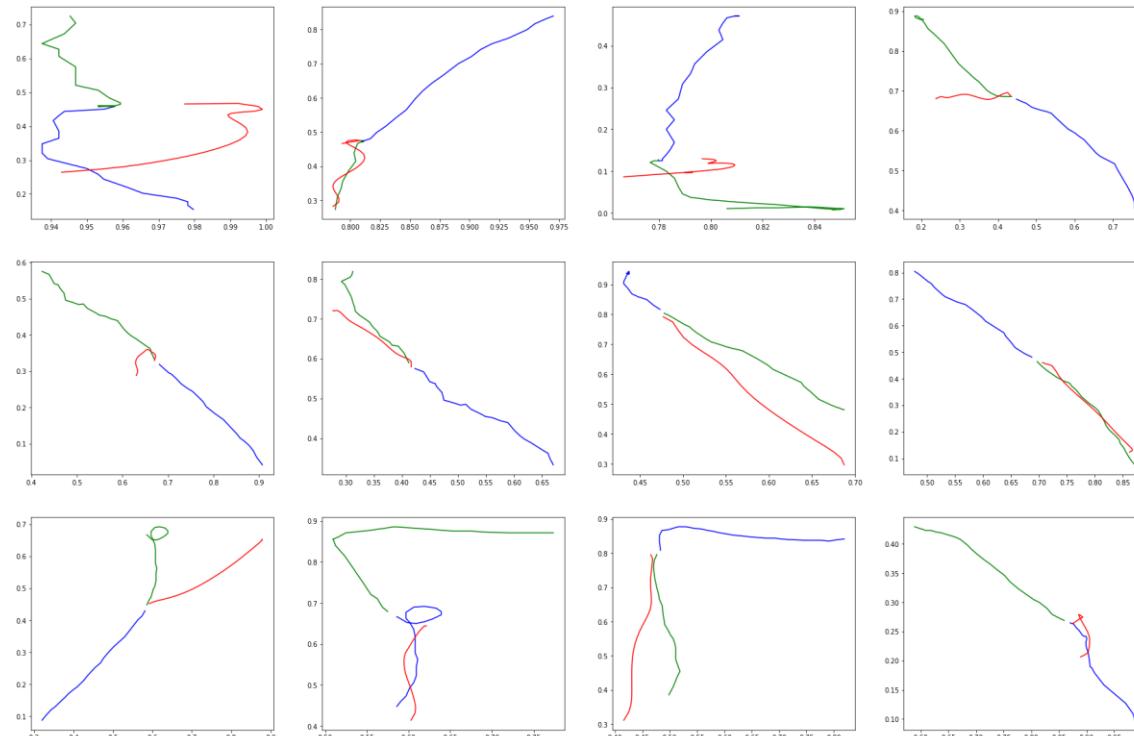
Bi-Directional LSTMs

- Run the sequence forwards and backwards through the network
 - Combine results
 - Better capture relationships within the sequence
 - More computationally demanding



Performance

- Blue: Input path
- Green: Actual future path
- Red: Predicted future path
- Generally better performance than single direction LSTM
 - Still have some errors such as discontinuities and very wrong trajectories



CAB420: Neural Attention

TRAINED TO THE POINT OF DISTRACTION

Neural Networks and Attention

- Neural networks have a lot of parameters
 - Not all are good, and some can get in the way of the task at hand
- Ideally, the networks themselves learn to filter important content
 - This doesn't always work due the large amount of content
 - Learned attention can also enable the network to incorporate more complexity and learn richer relationships
- Inspired by human attention process
 - Focus on the most important details within a signal

Hand-Crafted Attention

- Sometimes referred to as “hard” attention
 - Not always
- Attention signal is external to the network
- Determined based on rules or prior knowledge and provided to the system
- Examples:
 - When estimating a pedestrian’s trajectory based on their and their neighbour’s movement, weight observations with a fixed weight based on proximity
 - When performing text removal from a scene, provide a mask showing where text is as a secondary input
 - Can work well, but requires careful crafting

Learned Attention

- Attention is learned by the network with everything else
 - i.e. the network learns how to identify important things
- Typical approach:
 - Learn a layer to identify salient information
 - Output of this layer is combined with the input to the layer
 - No external stimulus to simulate learning
 - The network should learn on its own what types of features maximise performance

Learning Attention

LOOK OVER THERE!

Attention Implementation

- A simple implementation might look like this

$$x_1 = f_{Layer\ 1}(x)$$

$$\alpha = f_{Attn}(x_1)$$

$$\hat{x}_1 = x_1 \alpha$$

$$x_2 = f_{Layer2}(\hat{x}_1)$$

- Definition of layers is flexible

- Dense layers, recurrent layers, convolution layers can all be used

- α however is normally passed through either

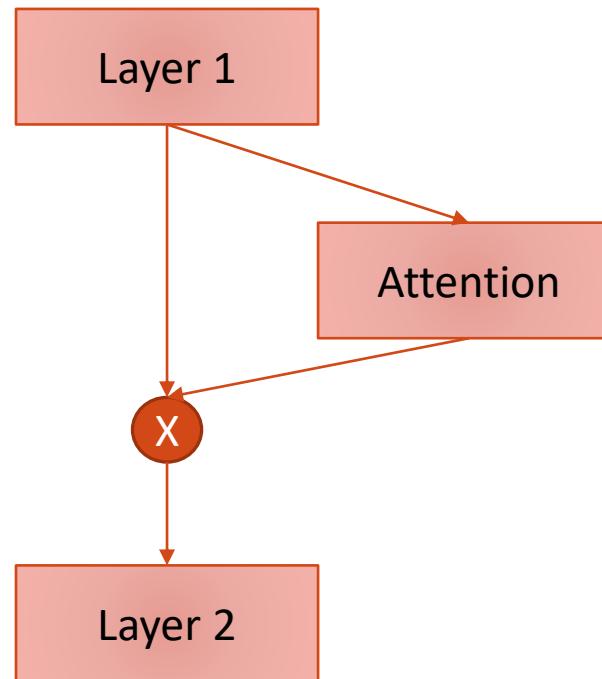
- a

- Softmax activation

- Find the most important details, suppress all others

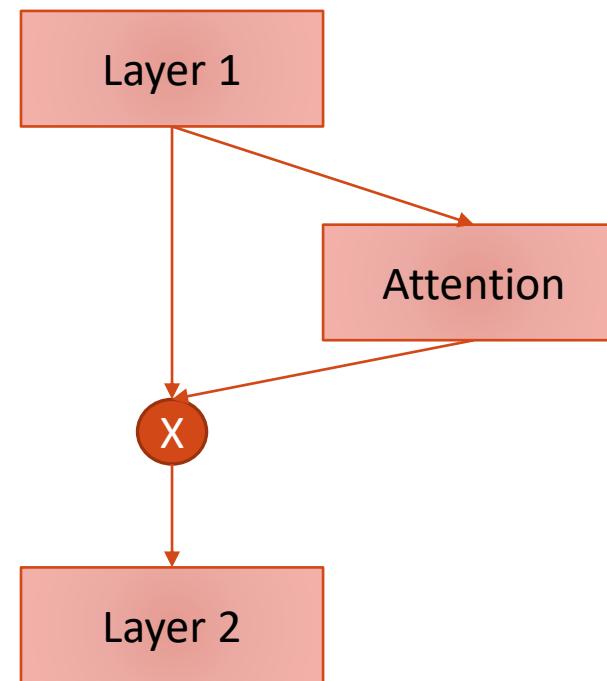
- Sigmoid activation

- Emphasise or suppress individual features



Additive and Multiplicative Attention

- We can also vary the operation that we use to incorporate our attention
- Many variations on the theme
 - Additive or multiplicative attention
 - Self-attention
 - Determines attention by looking at itself
 - Cross-attention
 - Uses information from one stream to direct attention in a second

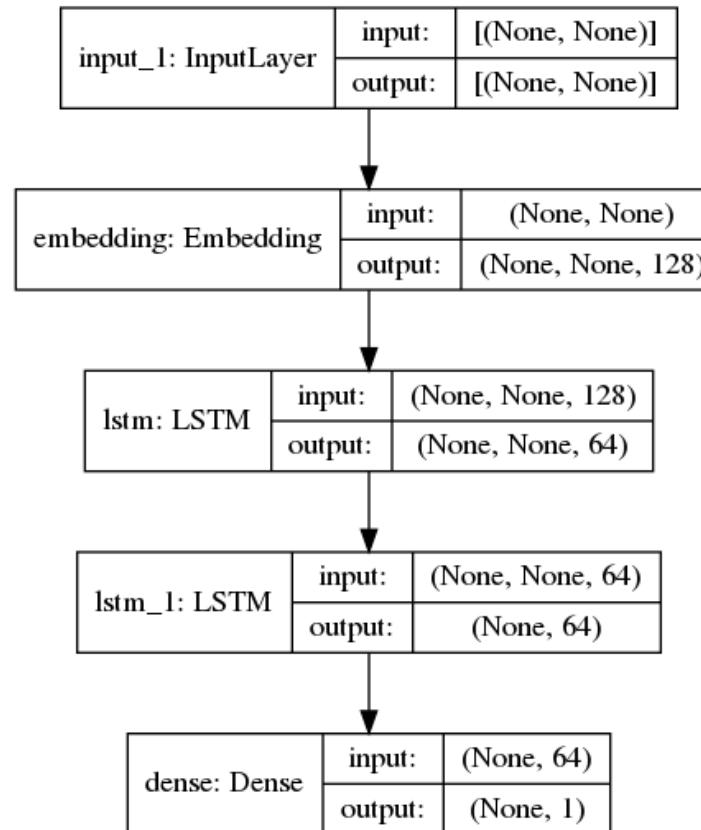


An Example

- See ***CAB420_Neural_Attention_Example_1_Attention_and_RNNs.ipynb***
- Our task
 - Classify tweets into positive or negative sentiment
 - Supervised Learning
- Our data
 - The text that someone felt compelled to share
 - A label as to whether the tweet was positive or negative

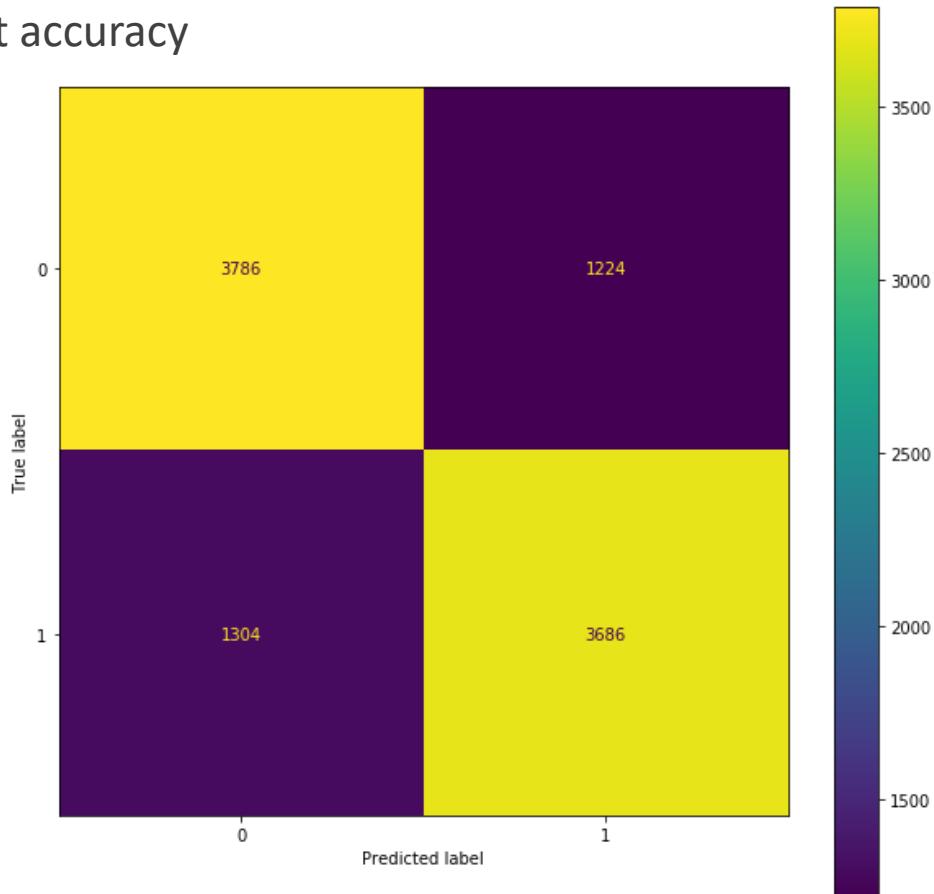
Our Baseline

- Very similar network to what we had when looking at sequences
- Two LSTMs
 - First outputs a sequence
 - Second outputs a vector
- Simple dense layer and Binary Cross Entropy loss follow to determine sentiment



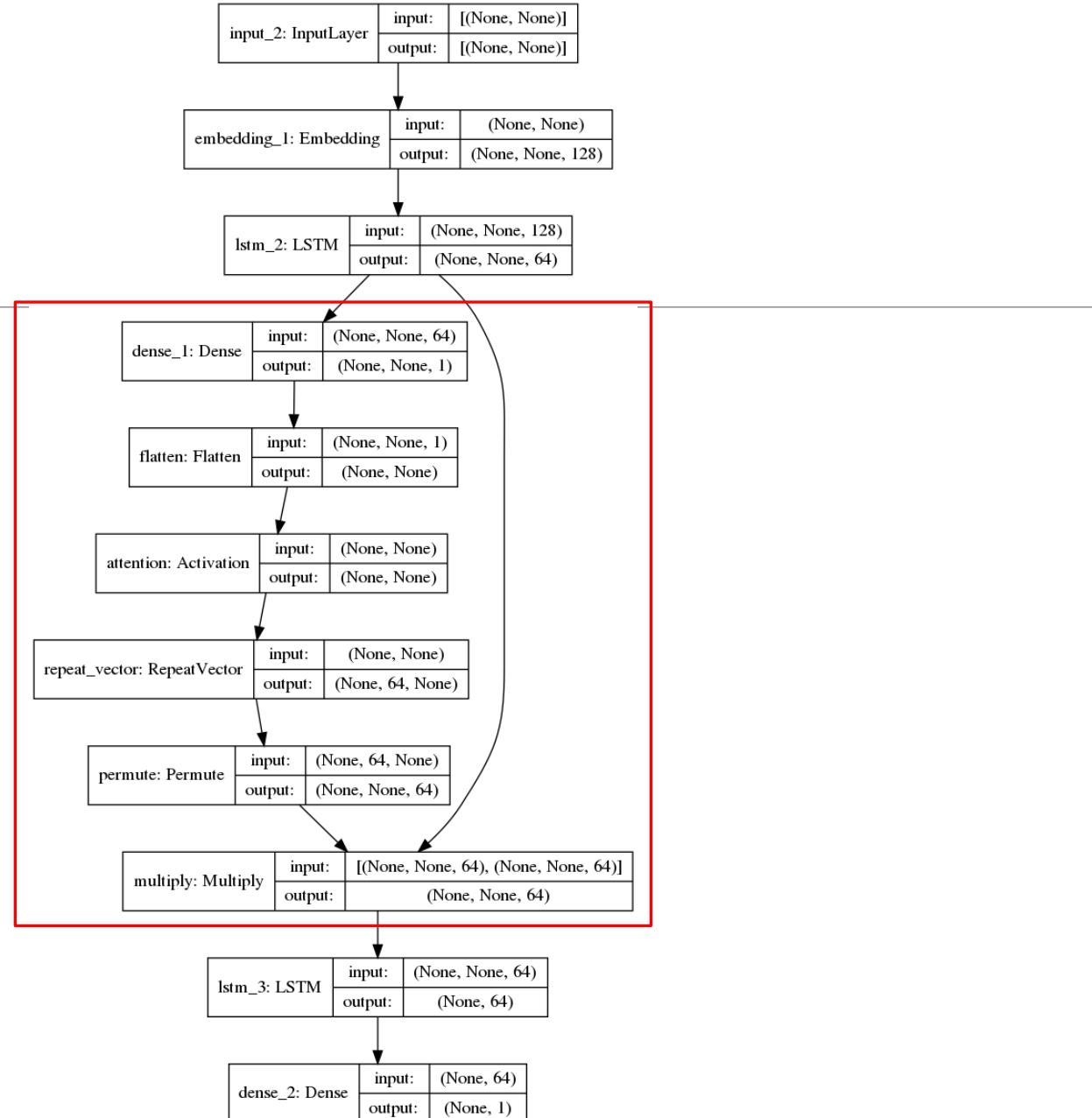
Performance

- ~74% test accuracy



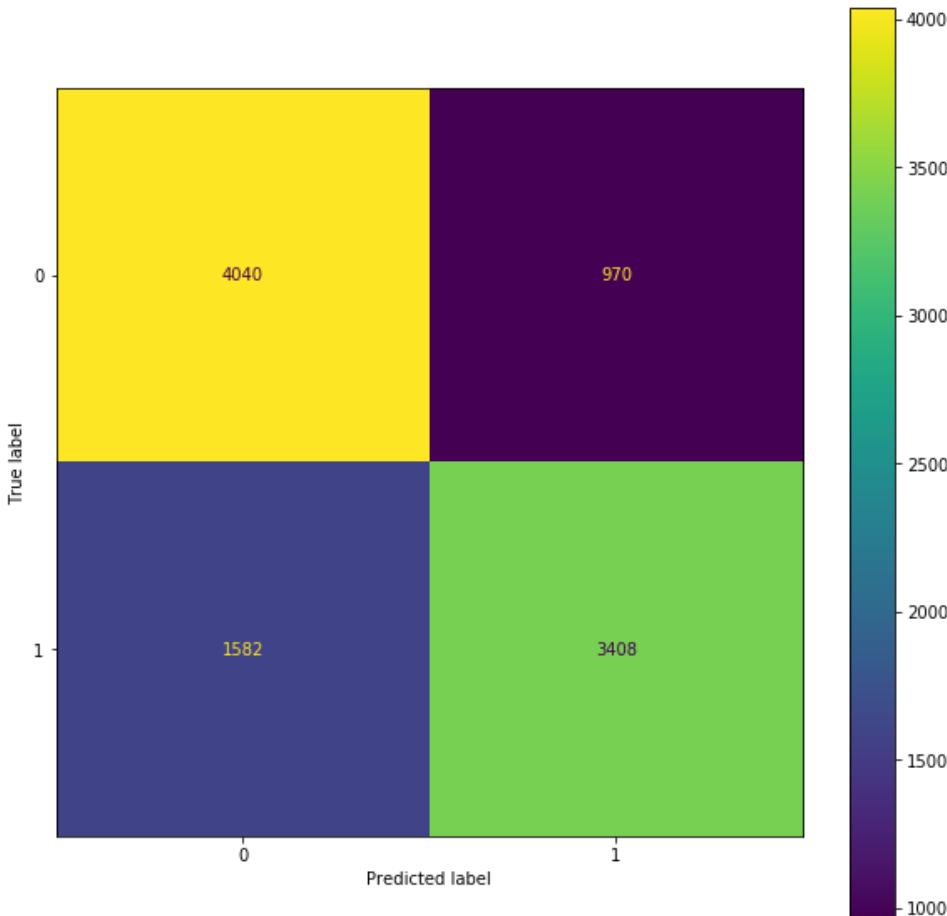
Adding Attention

- Attention Mechanism
 - Dense layer that operates on each element in the sequence from the LSTM
 - Outputs one value per item in the sequence
- Pass resultant vector through a softmax (or a sigmoid)
- Need to get attention vector into the same shape as the LSTM output
 - RepeatVector and Permute layers to stack 64 copies of the attention vector
- Multiply resultant matrix with the input



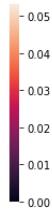
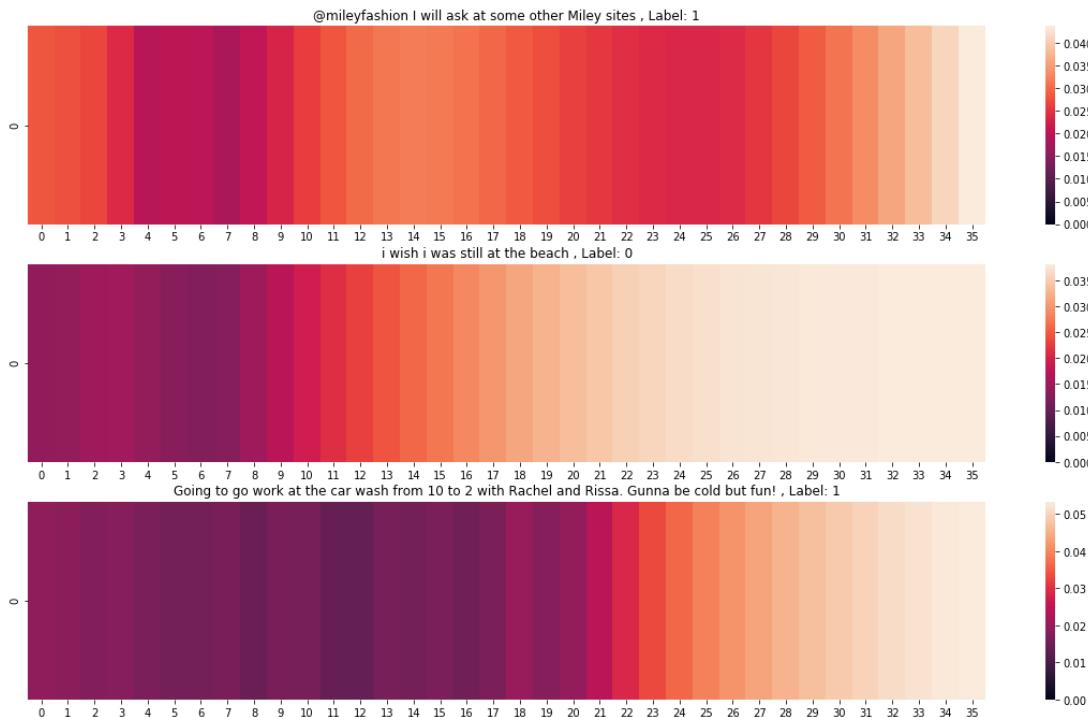
Performance

- ~74.5% accuracy
 - About the same as before
- Sequences aren't long enough to leverage full advantages
 - LSTMs can keep track of important features over short sequences



Visualisation

- Each column corresponds to an element in the sequence output
 - Does not align to the input on a word-to-word level
 - Each output in the sequence is a function of the word at that position and all words before it



Extending Attention in RNNs

- We have lots of options to play with here. We can
 - Play with different activations
 - Have attention be a function of all the words
 - Weight one word in relation to all others
- Other formulations are also possible with RNNs
 - Use cell state output from LSTM to weight activations
 - A form of cross attention, where one result is used to weight another
- Transformers
 - Networks built entirely on the idea of attention

CAB420: Transformers

MICHAEL BAY NOT INCLUDED

The Other Sequence Learning Approach

- Transformers
 - Proposed in the paper: “Attention is all you need”
 - Learning on sequences without a recurrent network
 - State of the art for sequence and language tasks
 - Emerging as a powerful techniques for images too
 - Make very heavy use of attention
 - Multi-headed Self Attention
 - Measure the importance of each element in relation to all others
 - Propagate this through multiple layers to model deep relationships in the sequence

Multi-Headed Self Attention

- Let's start with the "Self Attention" part
 - We start from a sequence of embeddings
 - Like we've been using
 - From our input sequence, we compute
 - Query, given by $x_i \times W_Q$
 - Key, given by $x_i \times W_K$
 - Value, given by $x_i \times W_V$
 - x is an input; W_Q , W_K and W_V are learned weights
 - Trained with the network
 - Result in vectors, usually of similar size to the embeddings (though we can make these whatever size we want)
 - From these we determine the importance of each word

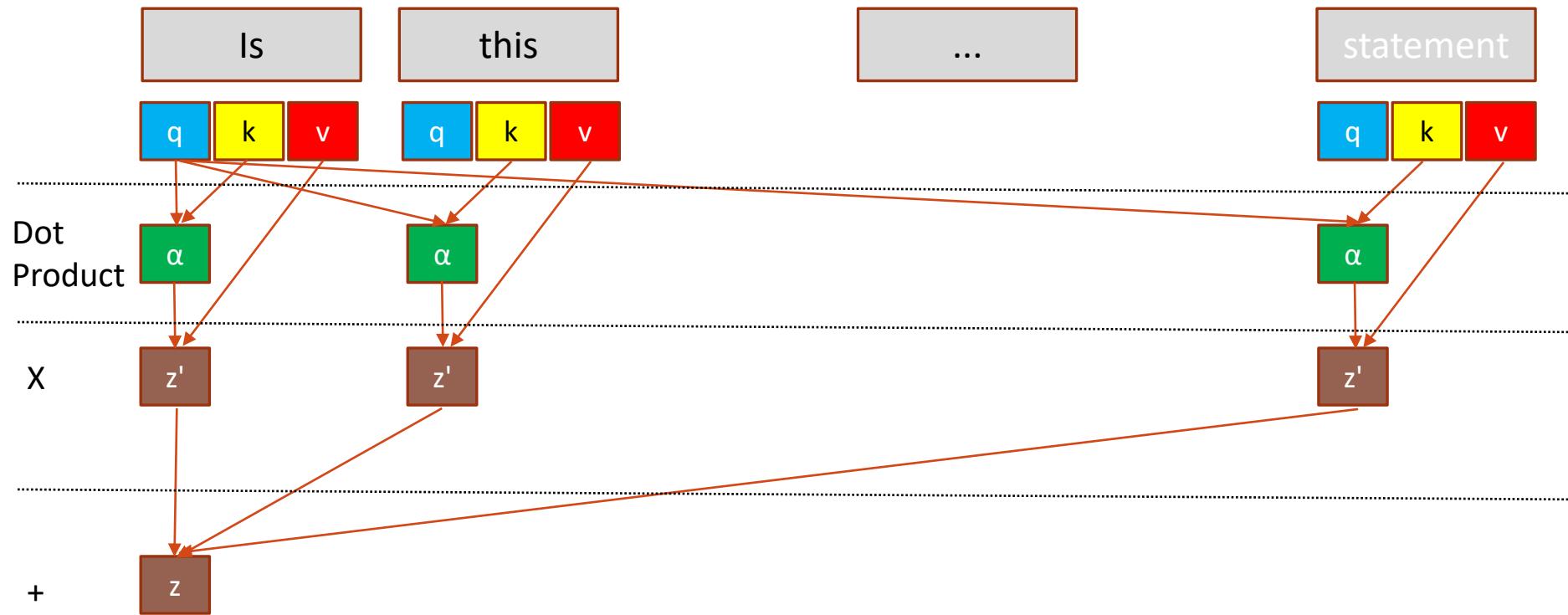
Multi-Headed Self Attention

- Consider the sequence “Is this a positive statement?”
- We use W_Q , W_k and W_V to obtain
 - $q_{IS}, k_{IS}, v_{IS}, q_{this}, k_{this}, v_{this}, \dots, q_{statement}, k_{statement}, v_{statement}$
- We need to measure the importance of each word with respect to each other, we do this by taking the dot product of query and key vectors
 - $q_{IS} \cdot k_{IS} = \alpha_{IS,IS}$
 - $q_{IS} \cdot k_{this} = \alpha_{IS,this}$
 - $q_{IS} \cdot k_{statement} = \alpha_{IS,statement}$

Multi-Headed Self Attention

- We now have a vector that tells us how important each word is to some target word:
 - $[\alpha_{Is,Is}, \alpha_{Is,this}, \dots, \alpha_{Is,statement}]$
- We apply a softmax to normalise these values
 - $[\alpha'_{Is,Is}, \alpha'_{Is,this}, \dots, \alpha'_{Is,statement}]$
- Then multiply these by the value vectors and sum the results
 - $z_{Is} = \alpha'_{Is,Is}v_{Is} + \alpha'_{Is,this}v_{this} + \dots + \alpha'_{Is,statement}v_{statement}$
 - z_{Is} is the output for the word “Is”
- We do this for each word in turn, and the output is a new sequence
 - $[z_{Is}, z_{this}, z_a, z_{positive}, z_{statement}]$

Multi-Headed Self Attention



Multi-Headed Self Attention

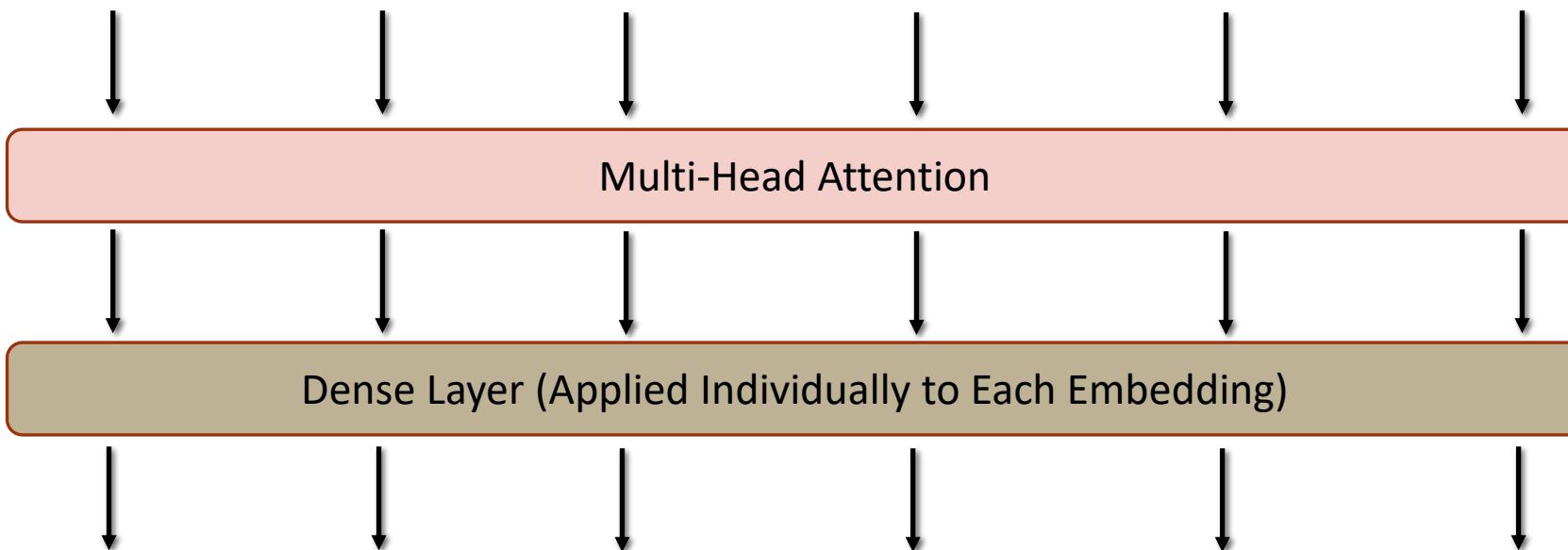
- Now for the multi-headed part, this simply means that we have multiple versions of W_Q , W_k and W_V
- If we have two heads, this means we get
 - $[z_{Is}^0, z_{this}^0, z_a^0, z_{positive}^0, z_{statement}^0]$
 - $[z_{Is}^1, z_{this}^1, z_a^1, z_{positive}^1, z_{statement}^1]$
- We then learn a separate weight matrix, W_O , to combine this into our final output
- Why multi-headed?
 - Each head can learn a different aspect of the mapping
 - All leaned directly from the data

Multi-Head Self Attention

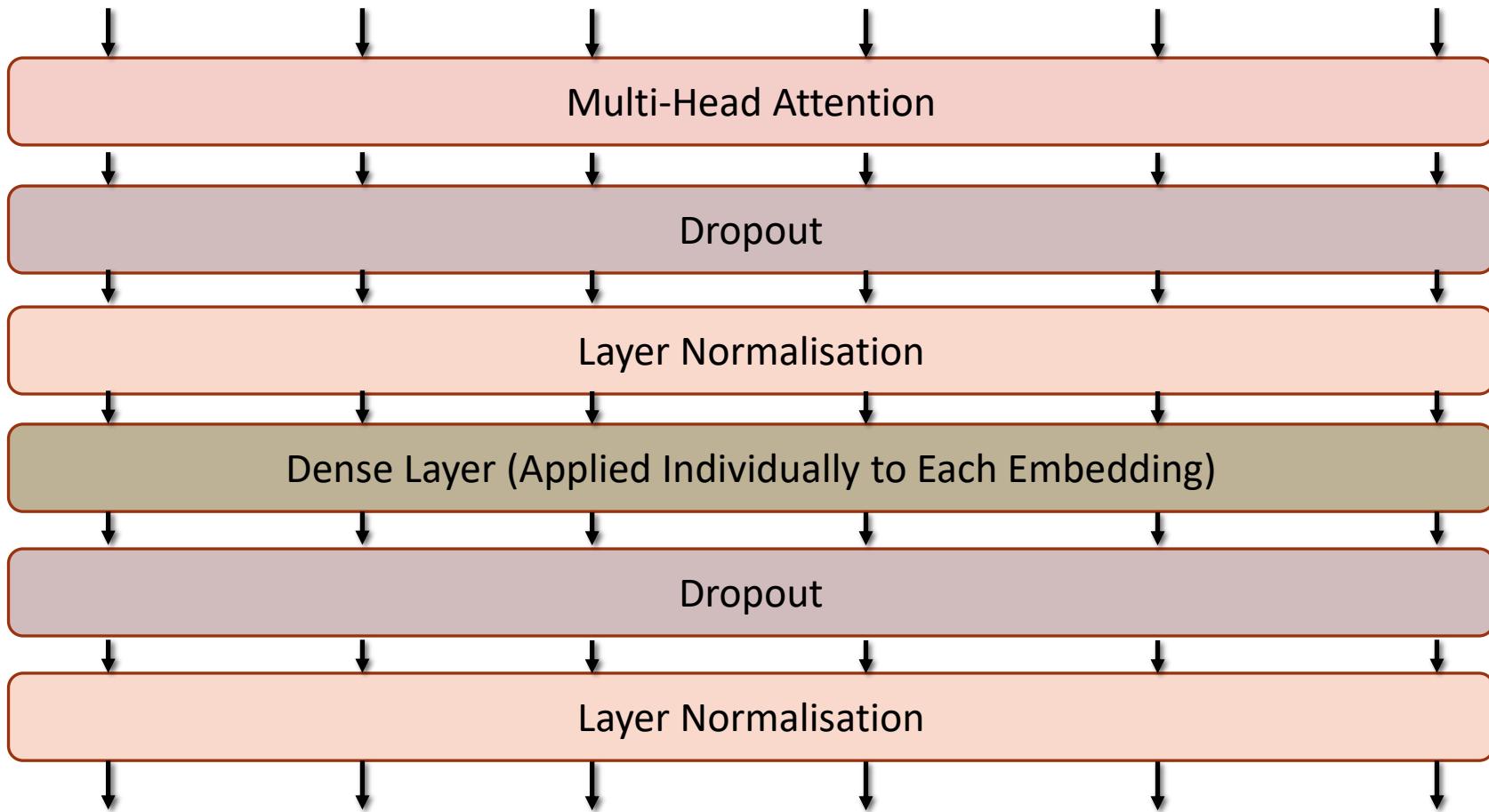
- Compared to what we had previously
 - We considered each element independently of each other
 - Relied on the LSTM embeddings to capture relationships
- The multi-head self attention approach scales better to long sequences or large input spaces
 - Still very computationally demanding for large input spaces
 - Easier to parallelise

Transformer

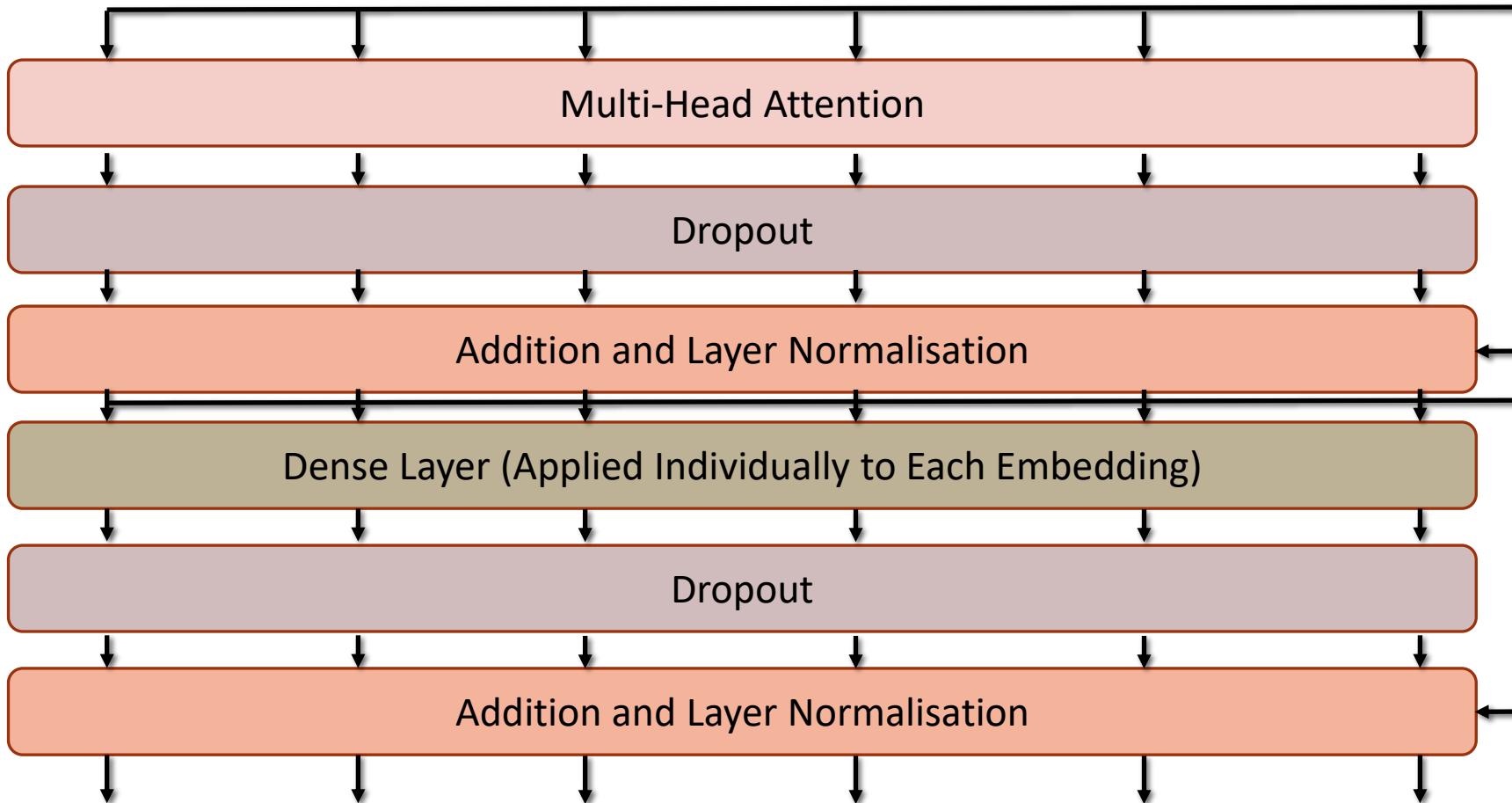
- Combine multi-head attention with a dense layer
 - Multi-Head attention inputs and outputs a sequence
 - Dense layer operates on each sequence item independently



Transformer



Transformer



Transformer

- Dropout and Layer normalisation between components
 - Layer is normalisation standardising each embedding
- Residual connections between
 - Input and layer normalisation prior to the feed forward network
 - Input to dense layer and layer normalisation prior to the output

Transformer Input

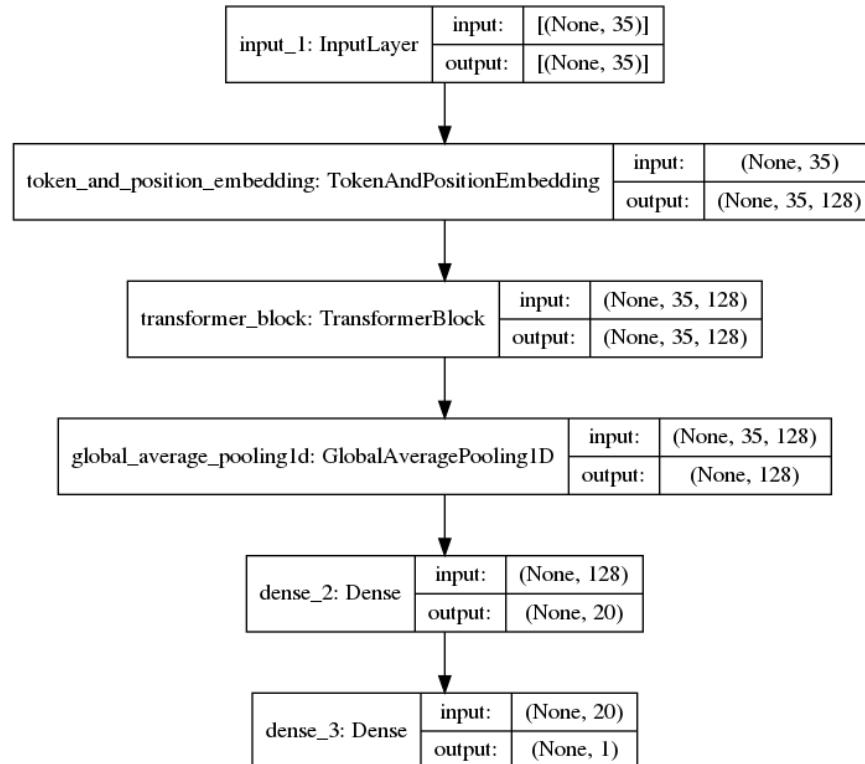
- Again, we're using an embedding, but we also encode position information in the embedding
- We have two embedding generators
 - Word embedding
 - Unique embedding for each word
 - Same as we've been using elsewhere
 - Position embedding
 - Unique embedding for each position in the sequence
 - Captures the position of a word in the sequence
- Final input is the sum of the two embeddings
 - Embedding thus captures the word and it's location

An Example

- See ***CAB420_Neural_Attention_Example_2_Transformers.ipynb***
- Our task
 - Classify tweets into positive or negative sentiment
 - Supervised Learning
- Our data
 - The text that someone felt compelled to share
 - A label as to whether the tweet was positive or negative

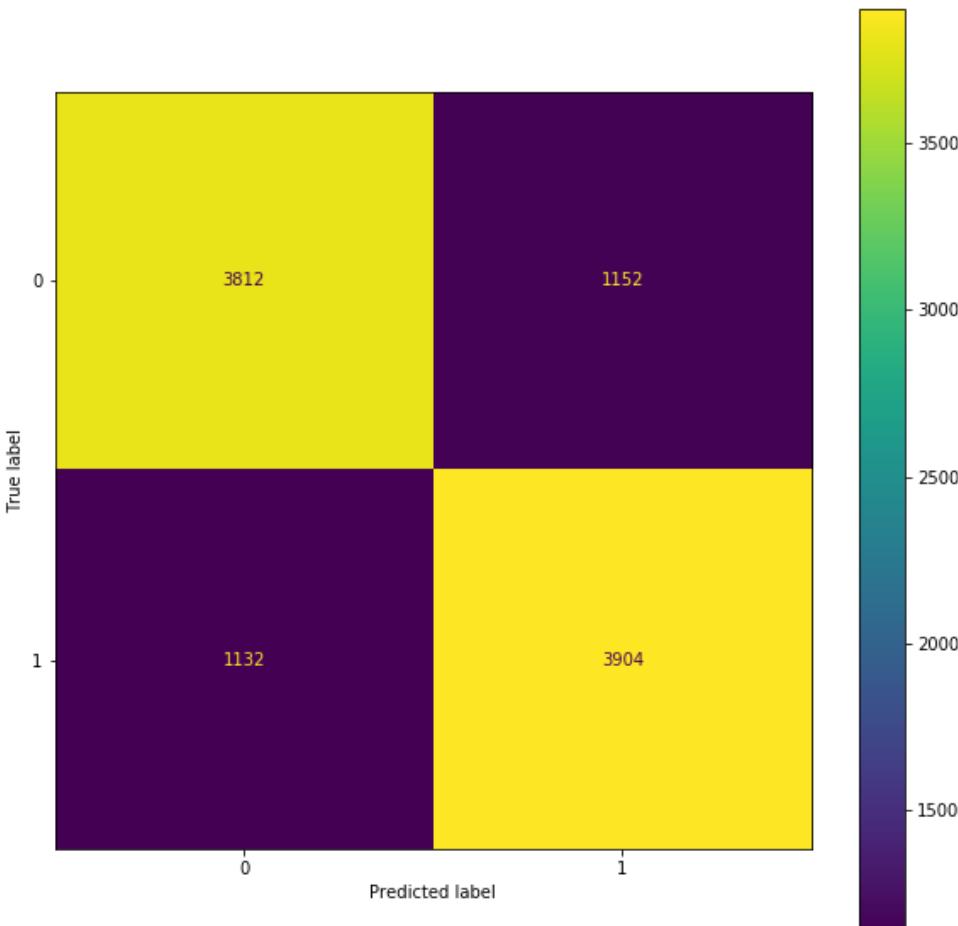
Classifying Tweets

- Our network
 - 128-D embeddings
 - 1 transformer block
 - 2 attention heads
 - Simple classifier after transformer
 - Binary Cross Entropy loss



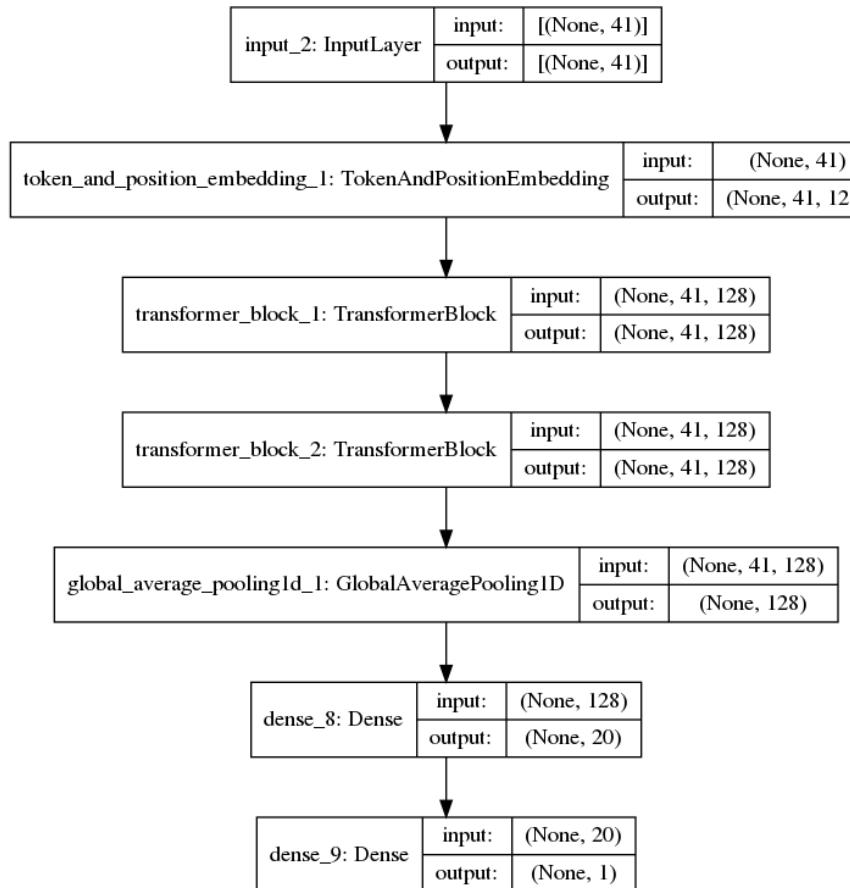
Performance

- 77% accuracy
 - Best accuracy we've achieved on this data
- Only trained for 2 epochs
 - Network begins to overfit quickly



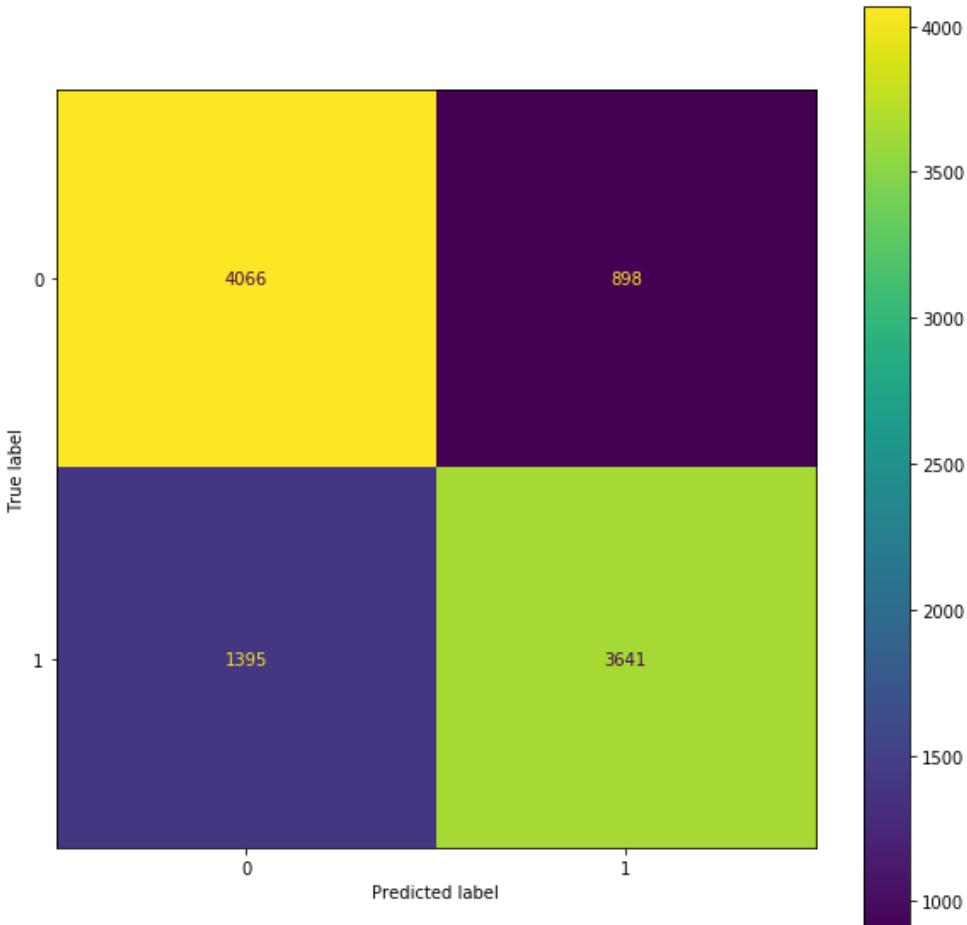
Stacked Transformers

- We can stack transformers like we do LSTMs
- Our network
 - 128-D embedding
 - 2 transformers
 - 4 attention heads each
 - Simple classifier after transformer
 - Binary Cross Entropy loss



Performance

- 77% test accuracy
 - Very similar to our first transformer
 - Data is not complex/diverse enough to warrant a larger network



Compared to LSTMs

- Transformers are much slower
 - Though on a GPU are easier to parallelise as all elements of the sequence are processed at once
- Lead to increased accuracy
 - And tend to overfit quicker, on this data at least
- Have attention built in
- Always output a sequence
 - Though pooling, or a dense layer can be used to obtain a vector output

CAB420: Attention Final Thoughts

SQUIRREL!

Attention

- Not all information in an input is equally important
 - The same applies to the intermediate layers on a neural network
- Attention seeks to emphasise important things, and mask irrelevant things
- Attention is learnt by the network
 - A separate layer (or layers) learns what's of interest
 - Can be formulated in many different ways

Transformers

- Apply attention en-masse to sequences
 - Multi-head attention, essentially just multiple different instances of attention that are combined
- Process sequences concurrently
 - Each element is processed at the same time
 - Self attention is used to capture relationships in the data
- Very powerful and current state of the art for NLP and sequence learning tasks
- Can be applied to images
 - See ***CAB420_Neural_Attention_Bonus_Example_Visual_Transformer.ipynb***