UWE Bristol | University of the West of England

Presentation by

Dr. Phil Legg

Associate Professor in Cyber Security

Date: Autumn 2019

Security Data Analytics and Visualisation

4: Machine Learning

Recap

- Data analysis is complex – there is no one solution for analysing a problem
- Feature Engineering is a fundamental challenge – how to express raw data as a value over time to make useful comparisons with future observations?
- Statistics can give excellent descriptive information about the data – but these are not always unique for a set of data however
- Machine learning techniques can help to identify trends and characteristics that may be present - important to know what the appropriate tool to use is
- Anomalies are probably the most commonly used characteristic in security investigations - however not all anomalies are outliers, it depends what you are looking for!
    - *"… if you look hard enough you'll always find something – even if there's nothing there …"* [*https://towardsdatascience.com/the-hidden-risk-of-ai-and-big-data-3332d77dfa6*](https://towardsdatascience.com/the-hidden-risk-of-ai-and-big-data-3332d77dfa6)

Analytical Algorithms

- We’ve started talking about how machine learning can help for analytics

- Input -> Process -> Output

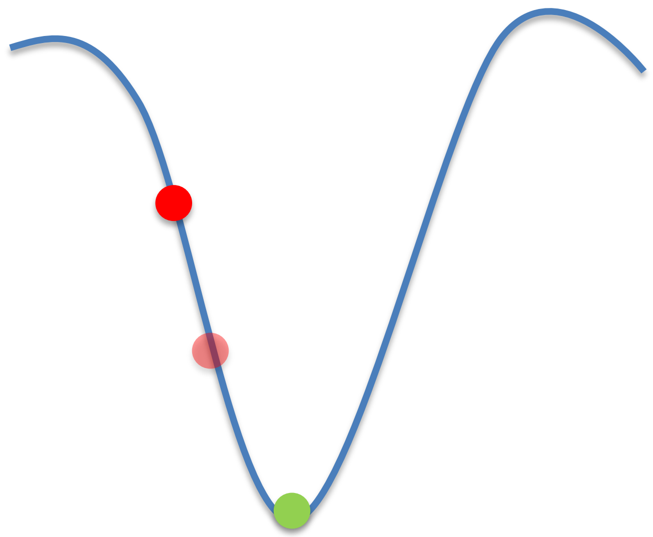
- To understand this further, we need to examine how these algorithms actually work.

- We will intentionally avoid complicated mathematics – but it is important to understand the concept of how the mathematics is being applied.

Analytical Algorithms

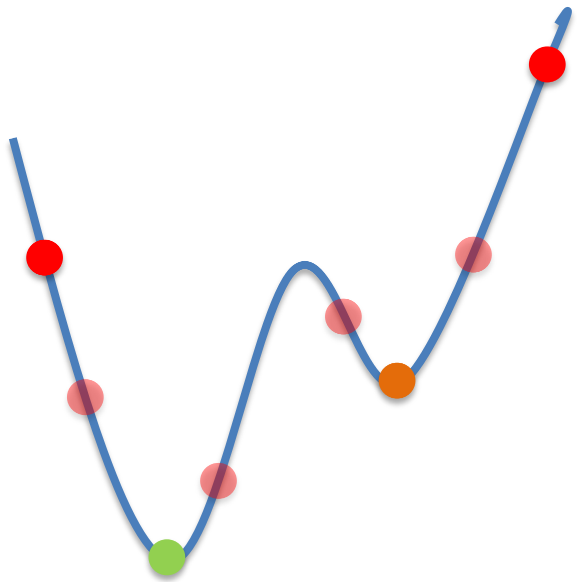
- Machine Learning is essentially about minimising an error measure (also known as maximising a “fitness” function).
- Machines work with numbers – so we need to be able to express our inputs and outputs numerically.
- The ”learning” is to learn a function that can map all possible inputs to all valid outputs, such that the error function can be minimised.
- We’ll work through some examples to help clarify this idea further.

Search Optimisation



- We've talked about "minimising" a function (e.g., minimising the mean squared error, or minimising the difference between our obtained result and ideal result)
- Think of it as a surface – at time ***t*** we are at the red ball. We need to find a way to get to the lowest point, much like rolling a ball down.
- The iterations / steps we take in each algorithm essentially get us closer to our solution in the same way.

Search Optimisation



- More challenging problems may not have an obvious minimum.
- The orange point is a local minima.
- We perform many different iterations to "get past" the local minima to obtain the correct solution.

Clustering

Unsupervised learning – identify clusters of similar instances

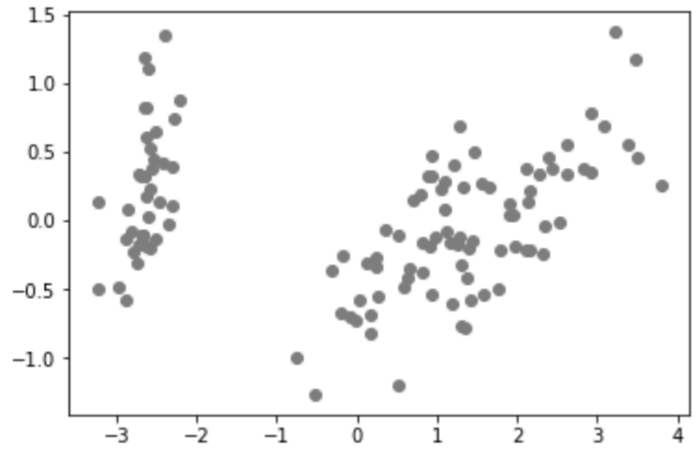
- e.g., k-Means Clustering

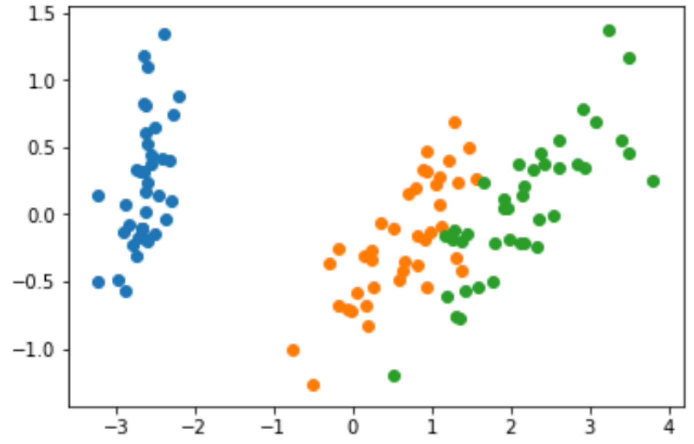
Data is typically in a high-dimensional space (i.e., we have many different attributes)

– how can we reduce this to something observable (i.e., 2D or 3D?)

Dimensionality Reduction

- Principle Component Analysis (PCA)
- T-Distributed Stochastic Neighbor Embedding (T-SNE)



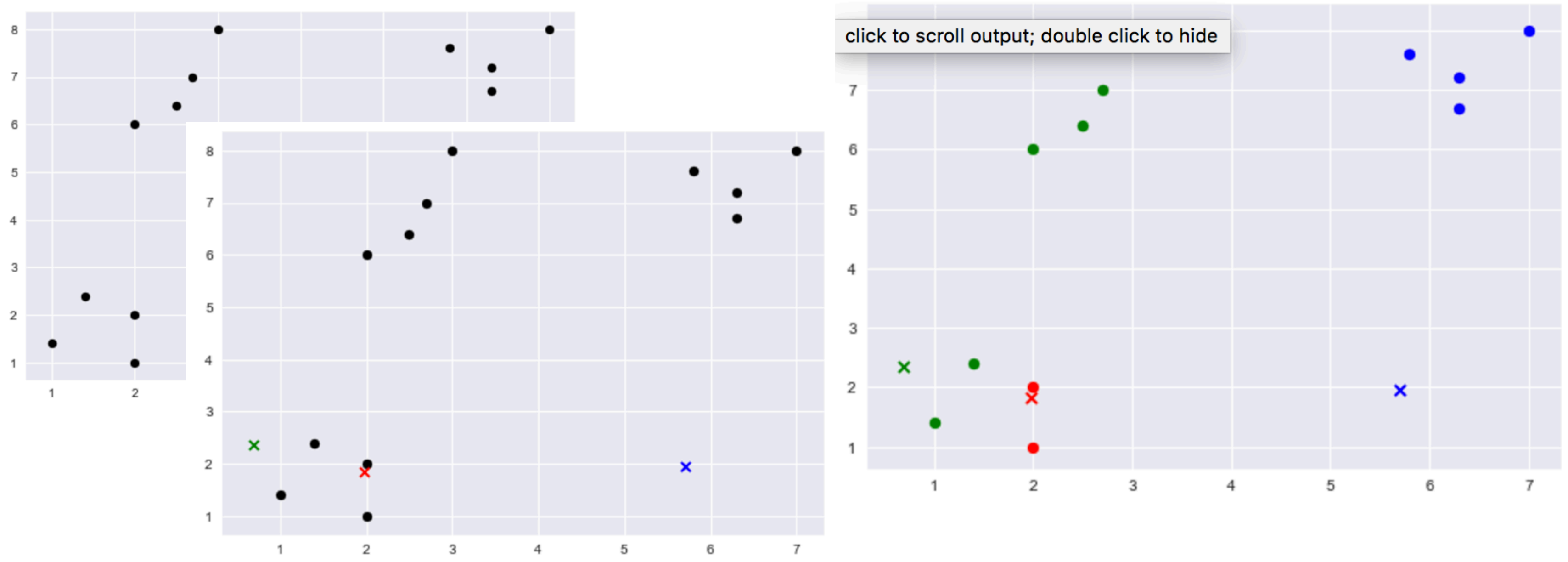


k-Means Clustering

- How does it work?

k-Means Clustering

- How does it work?



click to scroll output; double click to hide

Linear Regression

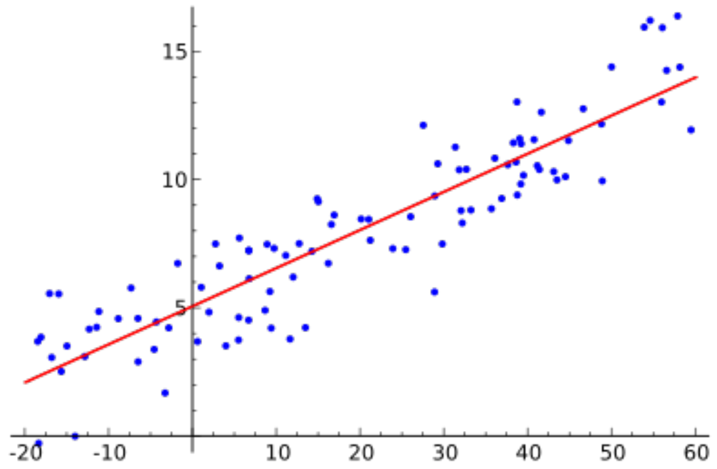
Learning a general representation of our data so that we can express as a function (e.g., line)

$$\mathbf{y} = m\,\mathbf{x} + c$$

If I observe a new x value, I can predict approximately what the corresponding y value would be.

e.g., If my system is running 20 processes (x), what would be the expected RAM usage for a uninfected system (y)?

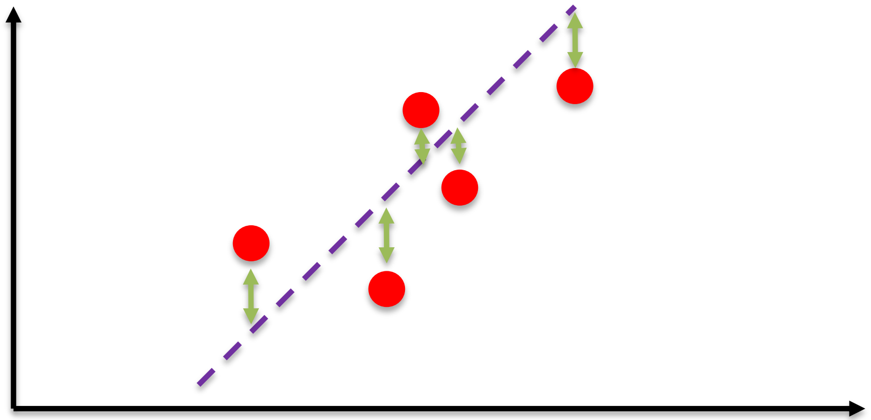
Mean Squared Error (MSE) can be used to measure how close the line fits the points



Linear Regression

- How does it work?

Linear Regression



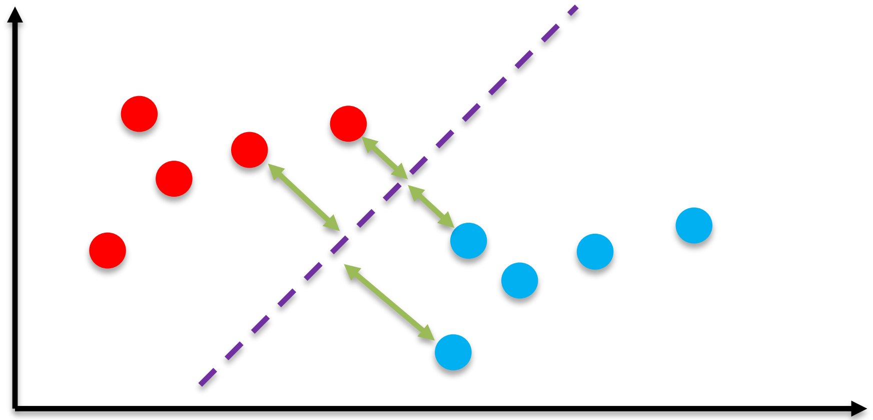
- Find the vector that best aligns with the data
- We want to minimise the vertical distance between each point and the vector
- This can be calculated by using the mean squared error

- `distances = [2, -2, 1, -1, -2]`
- `MSE = np.sum( distances ** 2 )`

Support Vector Machine

- How does it work?

Support Vector Machine



- Find the vector that separates our two classes
- Similar concept to linear regression, except we want to maximise distance from each class to the separating vector
- *Note we are now using euclidean distances – essentially meaning we can measure diagonally to find the shortest length (unlike regression where we worked using Manhattan distances where distance is vertical or horizontal only)*

What about high-dimensionality data?

- Many learning algorithms will work for higher dimensions (e.g., linear regression can take multiple attributes) – we just use 2D / 3D to visualise the process clearer

- How can we make sense of distances in high dimensional space?
- We can calculate in this space – however, this can be computationally expensive.

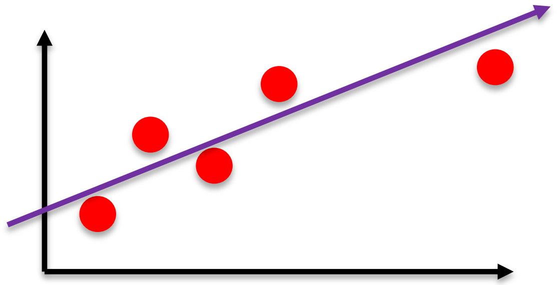
- One approach is to reduce the dimensionality to something manageable, whilst preserving characteristics of the original data.

- Principal Component Analysis / t-Distributed Stochastic Neighbourhood Embedding both work well for this!

Principal Component Analysis

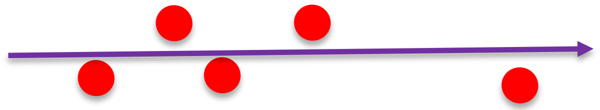
- How does it work?

Principal Component Analysis

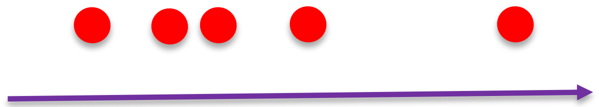


Find the direction with greatest variance

Use this as a “new” axis



Map our existing points to axis



- This example shows 2D to 1D
- Also works for getting from N-d to (N-1)-d
- Typically we reduce to 2D or 3D to then visualise

Neural Network

Popular technique that is suitable for learning how to map from inputs to outputs.

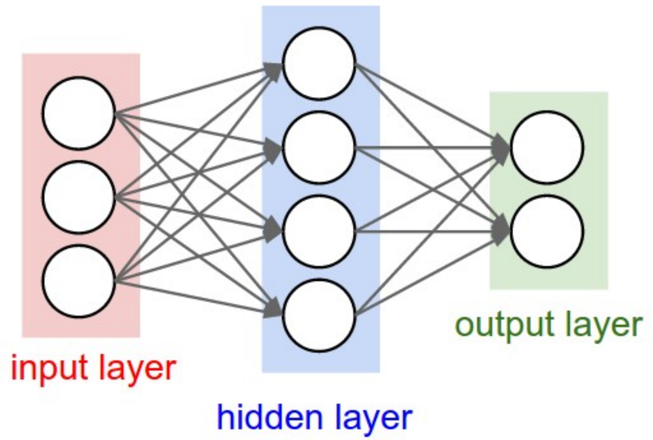
Network learns what should be at the hidden layer for transforming input to output

Image classification

- Input: Image pixels
- Output: Image type (e.g., person, cat, dog)

Anomaly detection

- Input: Activity features (e.g., counts)
- Output: Probability of anomaly

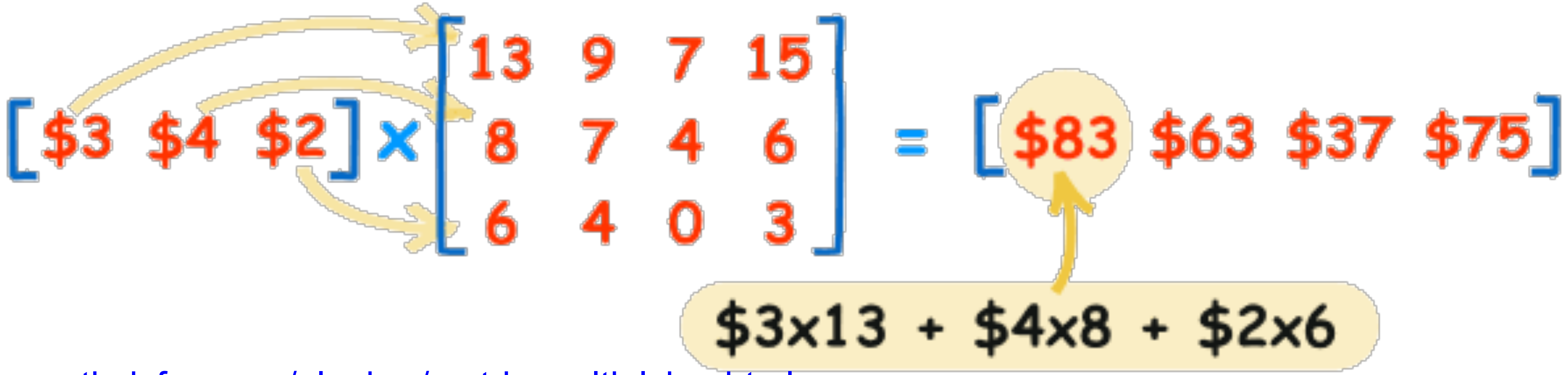


Neural Network

- How does it work?

Neural Network

- Matrix Multiplication – Dot Product
- Given two matrices A and B: **#columns in A == #rows in B.**
- New matrix will be of size: **[#rows in A x #columns in B]**
- For each position in new matrix, we multiple each row element in A with each column element in B, then sum together (see figure)



https://www.mathsisfun.com/algebra/matrix-multiplying.html

Neural Network

Input (***x***)

```
1
0
0
1
0
1
```

Weights (***w***)

```
[0.16316953,  0.35443148,  0.84137362,  0.45838469,  0.87889401, 0.3420689],
[0.61034289,  0.26322638,  0.72631579,  0.38975833,  0.67461149, 0.10938582],
[ 0.24693613,  0.61783659,  0.92494561,  0.71881251,  0.22810919, 0.66261227],
[ 0.54948748,  0.84506098,  0.96217536,  0.3804005 ,  0.18998701, 0.89607129],
[ 0.76759103,  0.18260298,  0.84298187,  0.44890229,  0.56532932,  0.95510022],
[ 0.64788966,  0.95962717,  0.93608893,  0.04686172,  0.3104502 , 0.29299659]]
```

Bias (***b***)

```
[ 0.35006693],
[ 0.04641672],
[ 0.68489831],
[ 0.35923794],
[ 0.36340899],
[ 0.08538542]]
```

Before sigmoid (***Y***)

```
1.31369005,
1.15590376,
2.31325923,
2.18519721,
2.53500253,
1.07313339
```

Output (***Y***)

```
0.78812998,
0.76058761,
0.90996923,
0.89891232,
0.92655949,
0.74519234
```

- **Y** = sigmoid (***wx*** + ***b***)
- `Y = (np.dot(w, x)) + b # this gets our output`
- `Y = 1 / (1 + np.exp(-Y)) # this applies the sigmoid activation`

Neural Network

$$\underbrace{\overset{(w)}{\begin{bmatrix} 0.16316953 & 0.35443148 & 0.84137362 & 0.45838469 & 0.87889401 & 0.3420689 \\ 0.61034289 & 0.26322638 & 0.72631579 & 0.38975833 & 0.67461149 & 0.10938582 \\ 0.24693613 & 0.61783659 & 0.92494561 & 0.71881251 & 0.22810919 & 0.66261227 \\ 0.54948748 & 0.84506098 & 0.96217536 & 0.3804005 & 0.18998701 & 0.89607129 \\ 0.76759103 & 0.18260298 & 0.84298187 & 0.44890229 & 0.56532932 & 0.95510022 \\ 0.64788966 & 0.95962717 & 0.93608893 & 0.04686172 & 0.3104502 & 0.29299659 \end{bmatrix}} * \overset{(x)}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}}} + \overset{(b)}{\begin{bmatrix} 0.35006693 \\ 0.04641672 \\ 0.68489831 \\ 0.35923794 \\ 0.36340899 \\ 0.08538542 \end{bmatrix}} = \overset{(Y)}{\begin{matrix} 1.31369005, \\ 1.15590376, \\ 2.31325923, \\ 2.18519721, \\ 2.53500253, \\ 1.07313339 \end{matrix}}$$

w * x becomes a matrix multiplication

#columns in w == #rows in x is TRUE

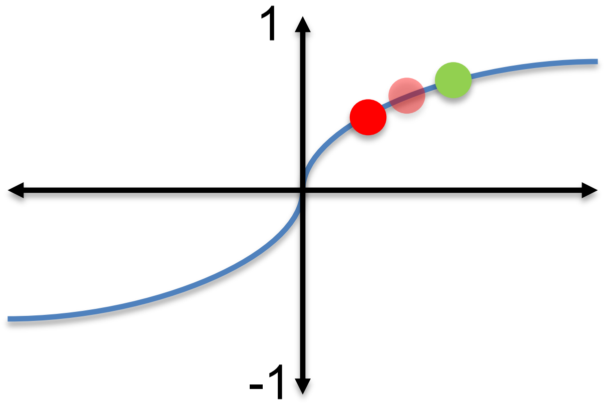
Output will be of size **(#rows in w, #columns in x) = (6,1)**

Neural Network

1.31369005,
1.15590376,
2.31325923,
2.18519721,
2.53500253,
1.07313339

$\rightarrow$

0.78812998,
0.76058761,
0.90996923,
0.89891232,
0.92655949,
0.74519234



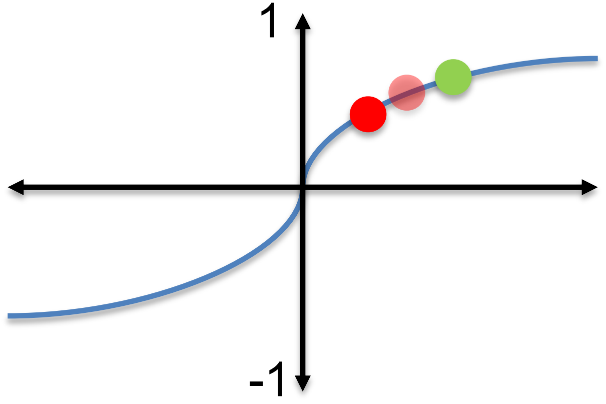
- After our matrix multiplication, we use a sigmoid to bound the resulting values between 0 and 1.
- The “result” of the network is the highest score (e.g., where each score may relate to a class).

Neural Network

- Neural Networks often use a “sigmoid” function to cap real values (X) between -1 and 1 (Y).

- Similar idea as we discussed earlier on search optimisation – the neural network essentially wants to find the optimal position on the sigmoid function such that inputs map to outputs correctly.

- “Backpropagation” is used to calculate this – called this because it propagates backwards starting from the error result, and finishing at the first layer, to update the weights of the network accordingly



Neural Network

- Suppose the orange column is the expected result (i.e., the input should map to the 3rd output).
- The classifier predicted the 5th output – which is incorrect.
- The error is the sum of the different (red)

0.78812998,	0	0.78812998,
0.76058761,	0	0.76058761,
0.90996923,	1	0.09003077,
0.89891232,	0	0.89891232,
0.92655949,	0	0.92655949,
0.74519234	0	0.74519234
		SUM = 4.2094

- We can update the weights based on how far we were off for each weight using the slope of the sigmoid (which is what backpropagation does)

Takeaway

- Learn by examples – examples of 3 common techniques available on Blackboard – *Clustering, Regression, Neural Network*
- Each method shows how we can take an input, learn a process, to map our data to a desired output
  - Output may be a group (clustering), may be a continuous value (regression), or maybe a more complex non-linear function that is either class or continuous (neural network)
- Often a function needs to be optimized (minimized) so that the machine can see when it is performing well or not
  - *Think of it as a scoring system – how do we express other security problems in terms of a score that can be minimized?*
  - *Game theory – multiple systems may compete to maximise own score and minimize opponent(s)*
  - *Could a score be manipulated or gamed by an attacker? Adversarial learning…*