01/05/16

DA : NN

Differences

- In LR the target variable must be categorical or a continuous target variable with values in the range of 0-1. NN do not restrict the value of the output variable
- The scale of the input variables in NN can affect calculation of the weights. The input variables must therefore be standardised to have mean 0 sd=1
- The input variable for logistic R can be of any type. The logistic formula contains each continuous predictor variable and a dummy variable for every category of a predictor.
- The Beta parameters in LR are found using MLE. With a typical NN, you may have many local minima if error is plotted as a function of weights. Therefore, optimisation methods must be independent with a number of random starting points to find the best values for the weights
- In NN variables must be pre-selected and if there are too many, PCA or another dimension reduction technique can be used to reduce the number of variables without losing too much information (do not necessarily omit variables)
- In LR, important predictor variables are assigned significant p-values in the output. The technique automatically chooses the important variables, and it can deal with a large number of variables
- LR provides standard errors or confidence intervals for its output, unlike NN

NN versus MULTIPLE LINEAR REGRESSION

Similarities

- A MLR can be modelled by a neural network with no activation function or with activation function equal to the identity function
- Both models take input variables and produce an output variable, NN can produce multiple output variables,
- Neither model produces graphical output.
- Both models are affected by outliers - preliminary analysis is important

Differences

- MLR can only model linear relationships

- NN can model non-linear relationship with the use of the relevant activation function.
- In NN cases with missing data are ommitted, in MLR cases can be imputed or ommitted
- The scale of the data in the predictor variable can affect the NN and ∴ variation of standardising with mean 0 SD=1
- MLR is not susceptible to the same restriction as explanatory variables
- MLR provides estimates of the standard errors and co-efficient whereas NN doesn't have this capacity
- MLR can be implemented in a simple excel spread sheet, whereas NN requires more complicated and possibly more expensive software

01/05/16

DA

NEURAL NETS OVERVIEW

- NN are learning models that attempt to map a series of inputs to an output
- They are typically comprised of an input layer, a number of hidden layers and an output layer
- Input layer contains the input variables that are being passed to the NN
- At each node in the network, two functions are applied. The first is a combination function which is usually some form of weighted summation of the multiple inputs. The second is an activation function

- The activation function defines the output from the node. This is where complex non-linearity can be introduced to the model because this function can be non-linear. Threshold, piecewise and hyperbolic tan functions are all non-linear
- This fundamental ability of NN makes them dissimilar from linear regression. When the identity function is used as the activation function and there are no hidden layers, the NN is like a linear regression

- The nodes interact by passing their values sequentially through the hidden layers via a series of weights and biases.
- They are often considered a "black box" method of classifying or prediction

Neural Nets versus Classification trees.

Similarities
- Both modelling techniques for prediction
- Both require specialist software packages
- Both very accurate for a wide range of cases
- Both punish for over complexity
- Neither provide standard errors or confidence intervals
- Both have graphical output
- Both model non-linear data

Differences

- NN can model linear structures when an activation function other than the identity function is applied in the code. CART can detect a linear structure but cannot represent it effectively
- CART detects interaction automatically. When X2 is split in two different ways following a split on X1, then the interaction is present. Interaction must be built in manually in NN
- CART deals automatically with missing values by the use of surrogate splitters. NN deletes/omits a case if it contains missing values
- CART deals automatically with outliers and they do not affect the modelling of the data structure, unlike NN who are affected by outliers.
- NN requires an expert or adequate statistical knowledge to develop and interpret a model. CART requires only moderate supervision by the analyst and produces easily interpretable output in graphical form
- The scale of input variables can affect NN. The variables are therefore standardised with mean 0 and SD 1. The input variables for CART can be of any datatype and scale
- CART automatically separates relevant from irrelevant predictors. In NN the variables must be pre-selected. This could mean a delay in preparation of data for NNs.

NN versus LOGISTIC REGRESSION

Similarities

- Both predictive modelling techniques.
- Both methods require a significant amount of knowledge and expertise to create and understand a hand-coded model
- Both models can model non-linear and linear relationships. LR can reasonably approximate many non-linear structures with a linear structure. NN introduces non-linearity through the use of activation functions in the nodes.
- Outliers affect both models and must be dealt with before building the models. Transformation of the [illegible] data so can be taken into both models
- In LR missing values are imputed or case is left out. In NN case is left out
- Interactions can be built into both models but are not recognised automatically

3/04/16 DA

Neural Nets

- Used for classification, prediction and clustering problems
- Black boxes with mysterious internal workings ~~as~~ input → box → output
- Objective : map a series of inputs onto an output : $y = f(x_i)$

LR

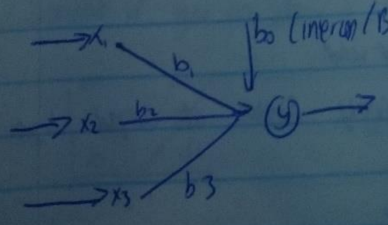
- Estimate values using LS or max likelihood to obtain β-weight estimate
- Other methods ridge or lasso regression
- Interested in size of β-weights
- Wanted model to predict for future

Types of NN

- Feed forward multilayer perception
- Radial basis function network
- Bayesian neural N
- Kohonnen self organising maps - NN version for clustering

Representation

- Multi layer perception - feed forward
- 3 independent variables and 1 output variable y



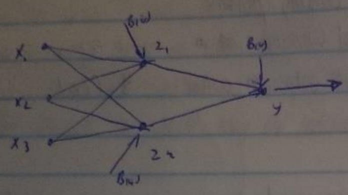
Origins

- late 1930's -40's
- Model brain activity mathematically

[illegible] network of neurons

16/ai/16

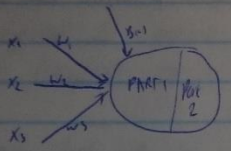
More Complicate NN



3 input variables x_1 x_2 x_3

1 hidden layer with two nodes: z_1, z_2

1 output variable y



Part 1 - Combination function - usually weighted summation

$a = \beta_{(0)} + w_1x_1 + w_2x_2 + w_3x_3$

Part 2: Transfer or activation function

- identity function $g(a) = a$
- linear function $g(a) = \alpha + \beta x$ for some α, β
- Threshold function $g(a) = \begin{cases} 1 & a \geq 0 \\ 0 & a < 0 \end{cases}$

Introducing a non linear transform of the data

- Sigmoid function $g(x) = \frac{1}{1-e^{-x}}$
- Hyperbolic tan $g(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$

Example: Feeding Data through a network

- CASE 1 $x_1 = 1$, $x_2 = 0$, $x_3 = -1$ $y = 1$ (target value)
- Hidden layer - use Sigmoid function and identity function at output layer

$a_1 = .086 + .86(1) - .27(0) + .33(-1) = -1.35$

$g(a_1) = 1/1+e^{-a_1} = 1/1+e^{-1.35} = 0.79$

$a_2 = .91 + .23(1) - .13(0) - .031(-1) = 1.41$

$g(a_2) = 1/1+e^{-a_2} = 1/1+e^{-1.41} = .81$

$y = .31 + 0.76(.79) - .47(.81) = -0.04$

$g(y) = y = -0.04$

16/04/16 DA

- Neural network predicts a value of -0.09 for these inputs and weights.
- Error = 1 - (-0.09) = 1.09
  → just a linear combination of the xs with lots of parameters

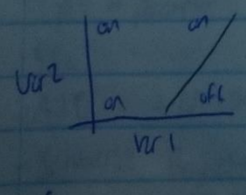
What makes them more powerful?
- Use of a non-linear activation function
- Model can handle more complicated situations
- Turns out to be a non-linear ~~method~~ model with lots of parameters ~ weights

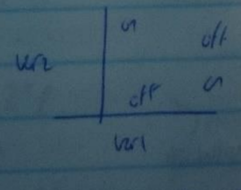
Non-Linear Functions

$$g = a_4 + \frac{e^{a_5 + 0.6 * V_1 - 0.7 * V_2}}{e^{a_4 + a_8 * V_5}} + a_1 * V_6 + a_2 * V_8 + a_3 * V_7 \quad \text{eg (made up)}$$

- Specify the form of the function and estimate parameters
- In neural nets you do not have to specify the functional form
- Useful when you have a series of inputs and don't care about process behind it and don't have to explain it

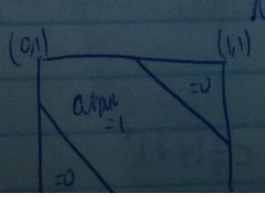
Linear Seperability
- Two input variables - either on or off.
- Seperable if you can draw a line which seperates the two outputs

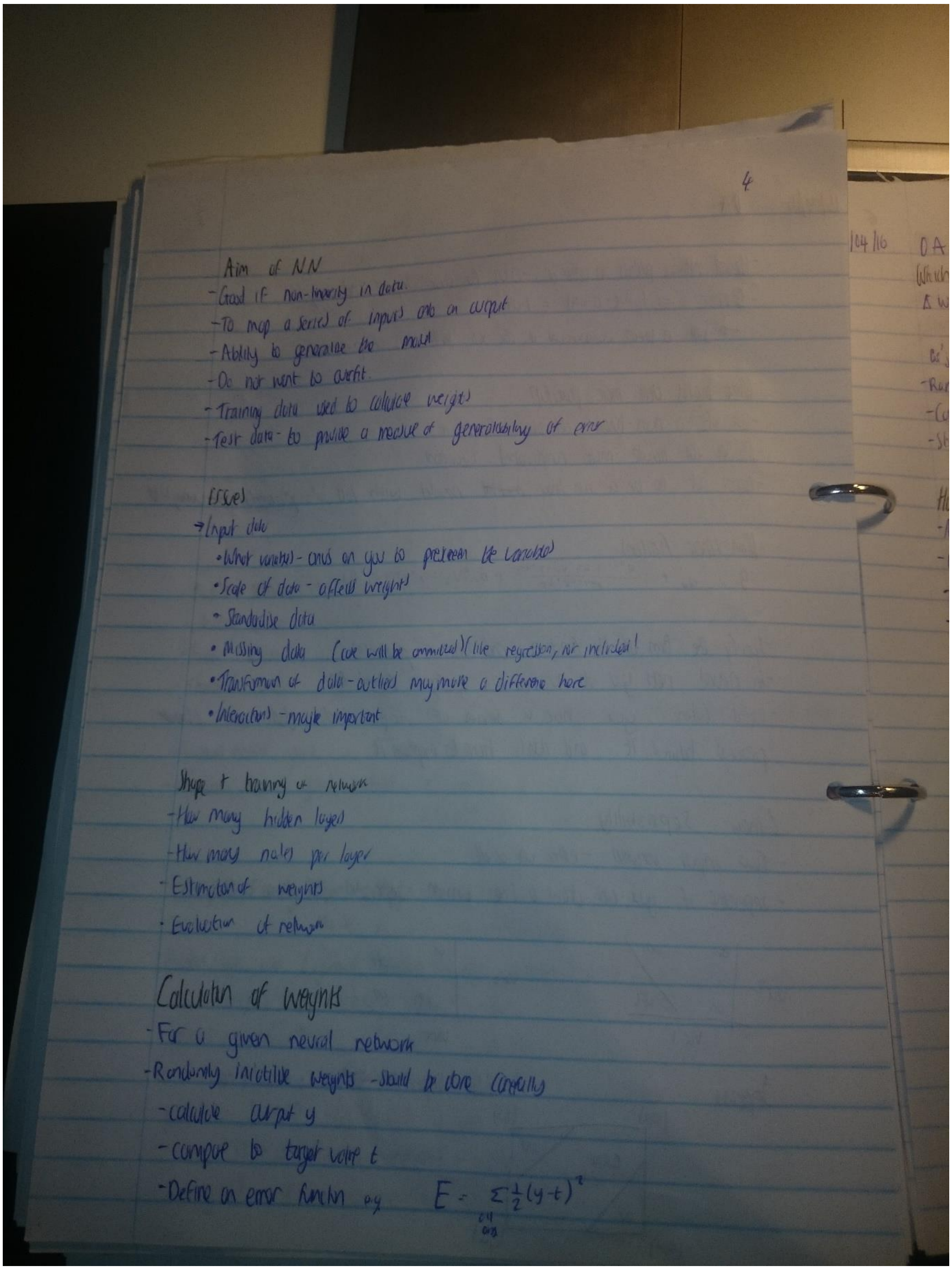




Seperable

Not Seperable → Draw a neural net.





Aim of NN

- Good if non-linearity in data.
- To map a series of inputs onto an output
- Ability to generalise the model
- Do not want to overfit.
- Training data used to calculate weights
- Test data - to provide a measure of generalisability of error

Issues

→ Input data

- • What variables - onus on you to prescreen the variables
- • Scale of data - affects weights
- • Standardise data
- • Missing data (case will be ommitted)(like regression, not included!
- • Transformation of data - outliers may make a difference here
- • Interactions - maybe important

Shape + training of network

- How many hidden layers
- How many nodes per layer
- Estimation of weights
- Evaluation of network

Calculation of weights

- For a given neural network
- Randomly initialise weights - should be done carefully
  - calculate output y
  - compare to target value t
  - Define an error function e.g. $E = \sum_{\text{all obs}} \frac{1}{2}(y-t)^2$

/16 DA -NN

Which weights are responsible?

$\Delta w_{ji} = -\frac{\delta E}{d w_{ji}}$ difference error function wrt weight

$w'_{ji} = w_{ji} + \Delta w_{ji}$

-Run another record (or all records) through and update again

-Calculate E using training set.

-Stop when E changes only a little

Huge Area

-Many ways for determining weights / optimising

-Grave danger of finding a local minima

-Numerical analysis problem

-Goal is to calculate a set of weights close to the optimum solution that works in the future

Hessian Matrix

-Matrix that contain $d^2E / dw_i dw_j$

-Eigenvalues should all >0 for global maximum

-Can look at these for each solution

Evolution of Network

-How can I do this for quantitative y? -Standardised data helps

$p = \#$ parameters $N = \#$ of cases

-SSE

$$\left.\begin{array}{l} \text{-AIC} = N' \ln(SSE/N) + 2^{*} p \\ \text{- Schwartz Bayesian Criterion} \\ \quad SBC = N^{*} \ln(SSE/N) + p^{*} \ln(N) \end{array}\right\} \text{minimise}$$

Tweaking a NN

- Start with a simple model - i.e like a SLR to find baseline for error
- Major numerical analysis problem
- Try to avoid local minima
- Run it a few times - may return different values for Squared error and weights
- look at eigenvalues of Hessian Matrix and weights - one of weights could be way off
- Do I really need a complicated model?

What can I do?

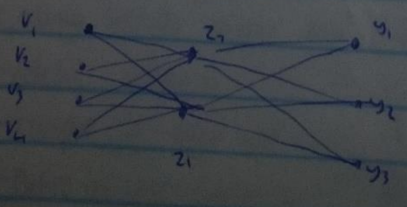
- Scale data beforehand: mean = 0 SD = 1
- Start with SLR and run a few times.
- Use a for loop to see variation in error
- Cannot compare models with unscaled and scaled data

Selection of activation function

- Hidden layer activation function, no choice in R has to be logistic function
- May want to use specific activation function to use outcomes as probabilities.
- Consider the interpretation and range of the target variable

Classification Problems

- K classes, K output variables y_n $\sum_{i=1}^{K} y_i = 1$
- Use class.ind function in nnet
- Range of the y_n $[0,1]$
- Like to interpret the output as probability of belonging to class k
- Use softmax transformation



16/04/16 DA

- $y_1 = 1$ if class 1, 0 otherwise etc.
- Define $T1 = w1 * z1 + w2 * z2 + bias$ - corresponds to y_1
- Similarly for y_2 and y_3

Softmax function to map Ts into $[0,1]$ range

- $y_i = \frac{e^{T_i}}{\sum_{i=1}^{k} e^{T_i}}$ ensure $\sum_{i=1}^{k} y_i = 1$

$\exists$ interpret y_i's as probabilities

Weight Decay

- Objective function = Error function + penalty
- Penalty = $d * \sum_i w_i^2$
- Keeps weights from flying off to ∞
- Values for d are usually in the region of 0.001
- May be too strong a penalty
- Try various values
- Ridge/lasso/elastic regression ways of eliminating errors to zero

Network Architecture

- Input variables, hidden nodes, hidden layers

Hidden layers

- No hidden layers - linear separability
- 1-hidden layer with X nodes should approximate most functions
- 2 hidden layers will introduce more complexity usually at a [illegible]

# of weights to estimate

n-inputs, m hidden nodes in first layer, p hidden nodes in second layer.

~~#~~ 1 hidden layers $m(n) + m = m^*(n+1)$ weights
2 hidden layers $p^*[m^*(n+1)+1]$

Two Approaches
Growing - Start with simple network and keep adding nodes + layers. look at AIC
Pruning - complicated model and delete paths
Radial base approach

Growing
- Start with no hidden nodes or layers
- Put 1 hidden layer with 1 hidden node and look at BIC/AIC in training set.
- At each stage use weights already determined as initial weights - can do this in R
- Add another layer with 1,2,3 hidden nodes - (can't do this with nnet)
- See what hidden layers are doing, what are they capturing?

Pruning
- Complicated network is trained
- Which weights/paths can be deleted
- What are least important weights
- remove smallest weights?
- little theoretical motivation
- Performs poorly in practice

Radial bases func - function whose value depends on the distance from the origin

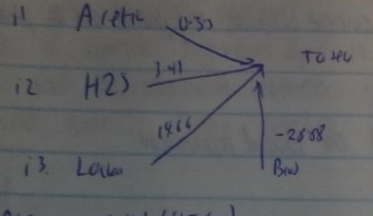
NN - R Output Studied
n=30 4 variables - Predicting taste of cheese

- Run SLR first as a baseline model

16/04/16 DA - NN

-Run NN about 10 times to see if it is stable

Result 3-0-1 network (3 input, 0 hidden nodes, 1 output node.
b → o (intercept) i1 → o ^{input} i2 → o $^{input 2}$ i3 → o $^{input 3}$ with weight

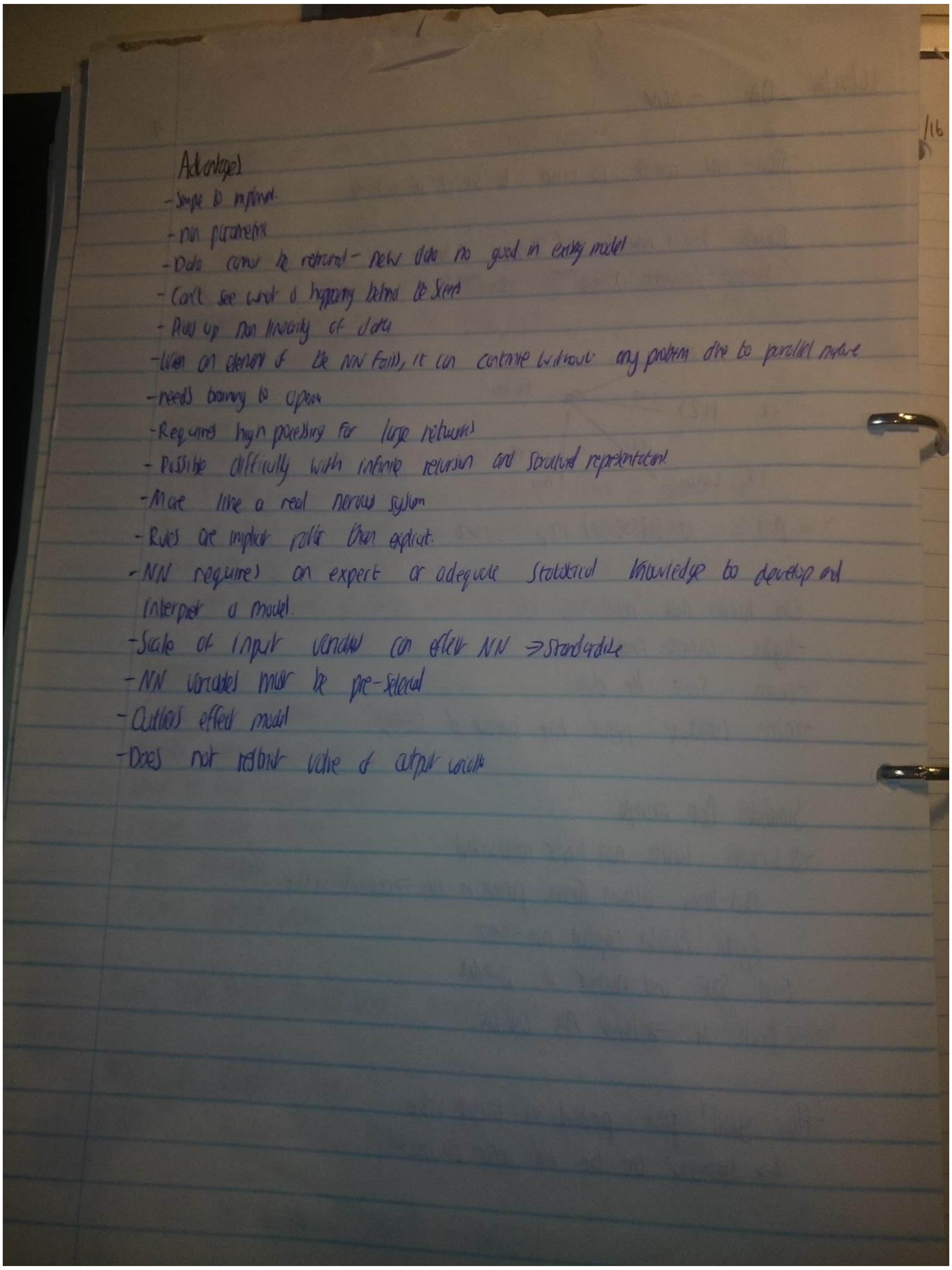


$AIC = N * \ln(SSE/N) + 2p = 142$

One hidden node result:
-logistic activation function
-could scale the data
-more consistent result here because of scaling

Simulated Data example
-8 variables with non linear relationship
non-linear activation function present in nn ⇒ reduction in SSE
logistic function captures non-linearity
Gives SSE and number of parameters.
→ Pick 4 ⇒ lowest AIC and SSE

-How good? plot predicted vs target value
↳ straighter the line the better the model



Advantages

- Simple to implement
- no parameters
- Data cannot be retrained - new data no good in existing model
- Can't see what is happening behind the scenes
- Picks up non linearity of data
- When an element of the NN fails, it can continue without any problem due to parallel nature
- needs training to operate
- Requires high processing for large networks
- Possible difficulty with infinite recursion and structured representation
- More like a real nervous system
- Rules are implicit rather than explicit.
- NN requires an expert or adequate statistical knowledge to develop and interpret a model.
- Scale of input variables can effect NN ⇒ Standardise
- NN variables must be pre-selected
- Outliers effect model
- Does not restrict value of output variable