Visualising
Deep Neural Networks

— Sam Green —

— Imperial College London —

Supervisors: Dr William Knottenbelt and Mr Daniel 'Jack' Kelly [1]

1st September, 2015

[1]Submitted in partial fulfilment of the requirements for the MSc Degree in Computing Science

Abstract

Deep Neural Networks are quickly becoming the industry standard for many complex machine learning tasks. Their inner workings however are considered by many as a "black-box". This project explores visualisation as a method to shed light upon how these networks are behaving.

Through the application of quantitative data visualisation theory to the data collected from within training neural networks, this project develops a tool that reveals immediately observable patterns that may help guide academic researchers to make more effective tweaks of critically important network parameters.

This report explains important neural network and visualisation concepts, the development of a product specification, the use of dimensionality reduction as a visualisation tool as well as taking the reader through a number of experimental versions before providing an analysis of several observations made through using the tool.

The main contribution of this project is through it's demonstration that more sophisticated visualisation techniques are not just useful for explaining neural network research, but can be used to better understand the research undertaken by academics while it is performed.

Acknowledgments

I would like to thank my supervisors Dr. William Knottenbelt and Mr. Daniel 'Jack' Kelly for their constant optimism, support and advice, my parents for their love and support, and the Turing Lab team who kept me sane throughout this project.

Contents

1 Introduction **5**
1.1 Motivations . . . 5
1.2 Objectives . . . 5
1.3 Contribution . . . 6
1.4 Report Outline . . . 6

2 Identifying the Problem **8**
2.1 Understanding Neural Networks . . . 8
2.2 Black Box Problem . . . 13

3 Searching for a Solution **15**
3.1 Human & Computer Augmentation . . . 15
3.2 Visualisation Theory . . . 16
3.3 Existing NN Visualisations . . . 19

4 Project Goals **26**
4.1 User Survey & Interviews . . . 26
4.2 Goals . . . 28

5 Data Collection **29**
5.1 Neural Network Implementations . . . 29
5.2 Dataset . . . 31
5.3 Collecting Output Data . . . 32
5.4 Evaluation . . . 34

6 Dimensionality Reduction **35**
6.1 Visualising High-Dimensional Data . . . 36
6.2 Dimensionality Reduction . . . 37
6.3 t-Distributed Stochastic Neighbour Embedding . . . 39
6.4 Evaluation . . . 40

7 Iteration 1 - Animation **41**
7.1 Introduction . . . 41
7.2 Design . . . 42
7.3 Architecture . . . 42
7.4 Evaluation . . . 43

8 Iteration 2 - Online & interactive **45**
8.1 Introduction . . . 45
8.2 Design . . . 45
8.3 Architecture . . . 46
8.4 Evaluation . . . 49

9 Iteration 3 - Epochs & Layers **51**
9.1 Introduction 51
9.2 Design 51
9.3 Architecture 53
9.4 Evaluation 55

10 Iteration 4: metaSNE & Principle Component Analysis **59**
10.1 Introduction 59
10.2 Design 59
10.3 Architecture 60
10.4 Evaluation 66

11 Conclusions **69**
11.1 Final Observations 69
11.2 Future Work 73
11.3 Summary 77

A Classifying Academic Visualisations **84**

1 Introduction

1.1 Motivations

Deep Neural Networks, DNNs, are machine learning algorithms that enables incredibly accurate feature learning and hierarchical feature extraction. These algorithms were first employed decades ago, however made a strong comeback to the machine learning community in 2012 when, in the ImageNET competition the clear winner by an unusual margin was a DNN. Since 2012 they have seen a dramatic increase in popularity in communities as far ranging as medicine, finance and sports prediction, however are most famously used for complex challenges such as computer vision and speech recognition.

However unlike some other machine learning models that are widely understood, such as logistic regression techniques, no one fully understands Deep Neural Networks in their full complexity. This is a problem for novice users and experts alike, and a current trend in DNN research is to explore not only the power of what these networks can do, but how they do it.

As a slight side note, with the latest hopes of general artificial intelligence being pinned upon these networks, several very influential figures have called for *ethics* boards to monitor their progress. It's worth noting that it's difficult to police such subjects as they are so poorly understood. So, besides the obvious academic arguments for learning more about these neural networks, there is a moral one too.

While there have been many studies mathematically analysing these networks, often aiming to optimise elements at the heart of deep neural networks, such as optimising the 'gradient descent' algorithm, there has been little work that aims to solve some of the everyday issues focused by machine learning researchers. These problems include not fully understanding what their networks are doing, what they are learning, or why. This culminates in making the difficult task of optimising parameter exceedingly challenging.

The motivation for this thesis is to provide a means of improving this situation such that researcher have a way of better understanding their models.

1.2 Objectives

The main objective of this thesis is to develop a tool capable of visualising the internal changes occurring within a neural network. Should such a tool prove to be useful it will demonstrate that more sophisticated visualisation techniques are not just useful for explaining neural network research, but can be used to better understand ones research while it is being undertaken.

There are a number of challenges in developing such a tool:

- **Data Generation** The first objective is to build a neural network and have the ability to easily change and tweak parameters in a controlled manner. This should produce the data that will eventually be visualised.

- **Simple work flow integration** One of the key challenges identified in early research was to produce a tool that fits into a researchers existing work flow. The first iteration of the tool must aim to be both simple to use and provide useful feedback about the network.

- **A more advanced tool** Currently it's very difficult to spot patterns amongst the vast array of numbers output by a neural network using traditional methods. The third, and most important, objective of this project is to develop a tool that enables researchers to spot patterns within their neural networks data more readily.

1.3 Contribution

This project has two contributions. Firstly, a new visualisation tool for academic researchers that reveals immediately observable patterns that can guide academic researchers towards making more effective tweaks to their network models. Secondly, through the achievement of the first this project strengthens the argument that it is necessary to develop more sophisticated visualisation techniques for the purposes of understanding research, rather than the common purpose which is simply to display research.

1.4 Report Outline

- **Chapter 2: Identifying the Problem**

  Provides an introduction to the basics of neural networks and their design space. It also explores the problem that this resurging field is relatively poorly understood: the black-box problem.

- **Chapter 3: Searching for a Solutions**

  Introduces the concept of *Intelligence Augmentation* and with it some important concepts from the field of data visualisation. It also explores previous work performed in the area of *Visualising Neural Networks*.

- **Chapter 4: Project Goals**

  This chapter explains the introductory survey which led to a number of goals being defined for the project. It then goes into further detail about these goals.

- **Chapter 5: Data Collection**

  Introduces MNIST, the dataset used within this project, as well as the two Neural Networks used throughout the project. Finally this section explores the topic of collecting data output from these networks.

- **Chapter 6: Dimensionality Reduction**

  In the previous section it is explained that the data collected from Neural Networks is *Multi-Dimensional*. This section explores two methods for dealing with this: Visualising and Dimensionality Reduction. It also explains the choice of using tSNE as the major visualisation tool for this project.

- **Chapter 7: Iteration 1 - Animation**

  This section explores the use of MoviePy animation as a tool for exploring the data output by the neural networks.

- **Chapter 8: Iteration 2 - Online & Interactive**

  This section develops upon lessons learnt from the previous chapter, exploring online methods for interacting with neural network data outputs. The use of Node.js, tSNE.js and D3.js are all explained here.

- **Chapter 9: Iteration 3 - Epochs & Layers**

  Once again this chapter builds upon the preceding chapter, exploring a different method for the interactive exploration of activation data through tSNE. It also explains a shift from a JavaScript backend to create a distinct Python / JavaScript divide, and the adaptation of an existing animation code body.

- **Chapter 10: Iteration 4 - metaSNE & Principle Component Analysis**

  This final product development builds upon all three previous iterations to demonstrate a fully-functional and useful tool. It also explores the use of Principle Component Analysis and Varimax Rotation to aid the control of manipulating visualisations.

- **Chapter 11: Conclusions**

  This chapter concludes this report by exploring several observations made using the tool that are representative of how an academic researcher may interact with it. It also proposes some opportunities for future work, and finally makes some concluding comments.

2 Identifying the Problem

2.1 Understanding Neural Networks

2.1.1 Overview

In order to understand Neural Networks we must first consider the human brain; a highly advanced information processing machine composed of around ten billion neurons and their connections. Artificial Neural Networks (ANNs) are a class of machine learning algorithms that seek to adopt some of the components of this advanced machinery, using a combination of computational and statistical methods to automate information extraction from data and allow computers to learn in a way that mimics early stage human learning.

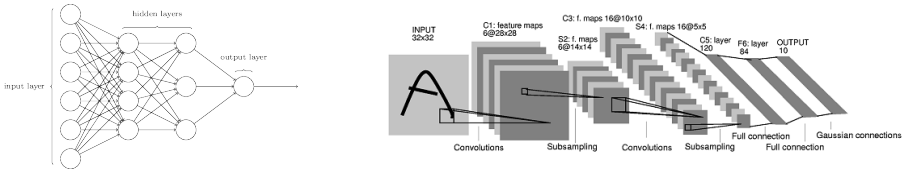
An ANN, in the case of supervised learning, is a collection of artificial neurons that are connected together in manor which allows them to successfully learn to process information to meet some previously defined end goal. The result of learning is that an ANN becomes a high-dimensional, non-linear, function that, while often taking vast amounts of time to train, is capable of performing a trained task within mere seconds when called upon. Provided with enough hidden units, it can approximate almost *any* function.

ANNs have been around for a long time, and had some early successes; such as when in 1989 Convolutional Networks (LeCun et al. 1989), or ConvNets, first demonstrated remarkable performance in tasks such as handwritten digit classification and face recognition. It was years later, in 2012, when they were finally put back on the machine learning map. The important leap forward came with the record breaking performance on the ImageNet classification benchmark, where the Krizhevsky ConvNet achieved an error rate of almost half that of the next best rival (16.4% in comparison to 26.1%) (Krizhevsky et al. 2012).

Several factors made the 2012 result possible where previously neural networks had been unsuccessful; the availability of vast training sets with millions of labelled examples, powerful GPU implementations speeding up training by great magnitudes thus enabling deeper models, and better model regularization strategies, such as Hinton's drop-out (Hinton et al. 2012).

Since the *Krizhevsky* success rapid advances in deep, or multi-layered, networks have produced significant outcomes in application areas such as vision (Russakovsky et al. 2015), speech (Sutskever et al. 2014), speech recognition (Sainath et al. 2015), NLP (Norouzi et al. 2014) and translation (Graves & Jaitly 2014). These developments brought deep learning into the heart of the current machine learning community, which for decades had dismissed them in favour of simpler models.

2.1.2 Network Structure



(a) Feed Forward (b) Convolutional

Figure 1: Two of the most common architectures used for DNNs

ANNs consist of a series of layers. These layers are composed of artificial 'neurons' that compute a function on the inputs provided by the previous layer. They then pass the results (activations, that are typically real-valued numbers in the range [0,1]) as outputs to deeper layers. Within any individual layer there exists only one type of neuron computing the same function: these neurons are differentiated by potentially distinct inputs, outputs and weight distributions. Layers themselves are defined by the number and pattern of connections between these neurons.

In order for a network to perform its task, a neural network must first be trained. This involves modifying the weights and biases of the network such that it produces the correct response for each of a number of training examples. The activations of the input units are set according to the feature values of the example, then these are propagated through the network to the output units, where the result is compared to the target output for that example and an error value calculated. This error signal is then back propagated through the network until the weights of the network have reduced the error at each node. The changes that occur are typically very small, and so large training sets are required to successfully converge the network on an optimal weight distribution.

The intuition behind back propagation, the algorithm that adjusts the weights with respect to the error value, is one of assigning 'blame'. The activations of the output nodes are determined by the activations of all the nodes below it, therefore error at the output is a result of the weights acting directly upon it from the preceding layer, and those recursively before it. In order to adjust the weights lower-down the error is backwardly propagated to the lowest hidden nodes that contributed an poor activation.

This process amounts to inductively learning how to solve a problem by exploiting regularities across a training set so that future similar examples may be classified in the same way. This is very similar to the way a human child learns, and again it's easy to see where these networks took some influence from.

2.1.3 Layers

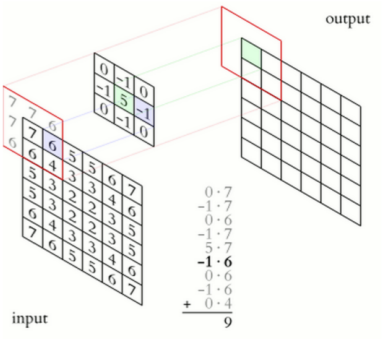
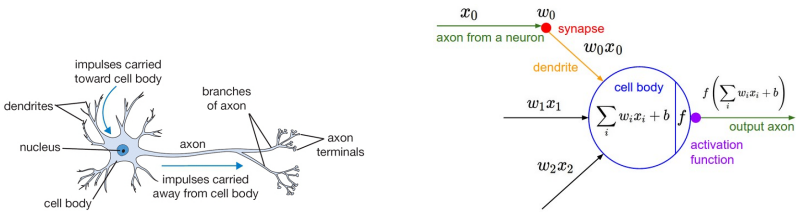


Figure 2: Convolutional Filters

There are a number of different types of layers that can be combined in a neural network: in a *fully connected layer* the neurons receive an input value from every neuron in the previous layer. In a *locally connected layer* the neurons are indexed spatially with inputs coming only from those nearby, and in a *convolutional layer* a number of filters are applied to create a convolution.

The convolution of an image is produced by applying a filter upon the input image. The filter is a kxk weight matrix such that k is an odd number to ensure the matrix has a true centre. The convolved image is produced pixel at a time by computing the dot product of the filter and the pixels below it, the central pixel of which is updated. A convolution is therefore produced by scanning the filter across the input pixel space until every pixel is replaced by a pixel that is some function of its filter bound neighbours. Deep successions of convolutions encode images in ways that make them invariable to translation and deformation. This is critical for classification (Bruna & Polytechnique 2012).

2.1.4 Neurons



(a) Multipolar Biological Neuron (b) Artificial Neuron Model

Figure 3

As mentioned previously, artificial neural networks are modelled on the human brain. They take influence from the *multipolar biological neuron*. The neuron receives multiple electric charges from its neighbours through the dendrites. This then triggers a single electric charge to a different set of neighbouring neurons through its axon terminals. Artificial neurons perform effectively

the same task and compute functions that take in multi-dimensional input but output a mono-dimensional result.

There are a number of different neurons used within the layers of an artificial neural network:

Binary Threshold Neuron

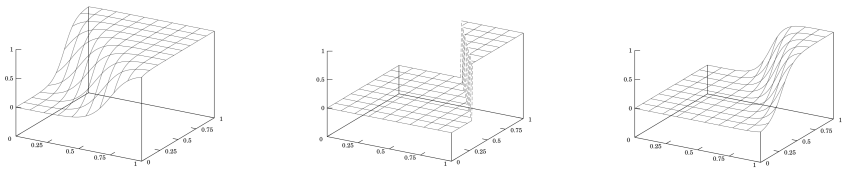
$$y = \begin{cases} 1 & \text{if } M \leq \sum_{i=1}^{k} x_i \cdot w_i + b \text{ where } M \text{ is a threshold parameter} \\ 0 & \text{otherwise.} \end{cases}$$

Here, y is the output of the neuron calculated by the weighted input acting upon it, and assessing this value against some threshold M. The threshold neuron works much like a biological neuron in that it either outputs a charge or it doesn't. This neuron however is rarely used due to the fact that it cannot be used in optimisation algorithms, such as gradient descent, which require a function to be differentiable.

Logistic Sigmoid Neuron

$$y = \frac{1}{1 + \exp(-z)} \text{ , where z} = \sum_{i=1}^{k} x_i \cdot w_i + b$$

A more commonly used transfer function is the sigmoid, which is an approximation of the threshold function above. Here the bias b performs a similar function to the threshold M in the previous example. The 'threshold' can be through of as the point at which the gradient of the *decision surface* is steepest. While in the threshold neuron this represents a hard boundary, the sigmoid represents a gradient of values. One disadvantage of the sigmoid is that is is more expensive to compute.



(a) Sigmoid A (b) Threshold (c) Sigmoid B

Figure 4

Rectified Linear Neuron (ReLU)

$$y = \max\{0, b + \sum_{i=1}^{k} x_i \cdot w_i\}$$

The rectified linear neuron is a hybrid function. It is more efficient to compute that the sigmoid neuron and is partially differentiable, thus making it suitable for gradient descent. The compromise here is the cost of sophistication of the result. The neuron introduces a non-linearity with its angular point, a smooth approximation of which is the softplus $f(x) = log(1 + e^x)$).

2.1.5 Design Space

In a typical machine learning workflow, including working with ANNs, practitioners iteratively develop algorithms by refining choices in areas such as feature selection, sub-algorithm selection, parameter tuning and more (Patel et al. 2008). This is usually done through a trial and error approach that is perhaps similar to hill-climbing in the model space and can lead to locally minimal results. This is generally considered to be unsatisfactory due to the small number of outputs that a researcher may be following as a guideline - such as error.

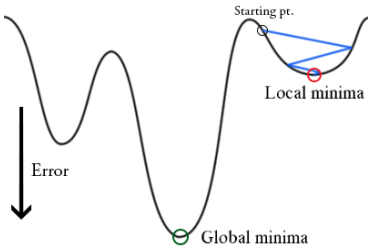


Figure 5: Hill Climbing in the parameter space (Gradient Descent)

The most challenging and time-consuming part of training a neural network lies in selecting the correct parameters, of which there are many, and each affects the network in an almost unknown capacity. Some examples are:

Size of Filters: if the filter is too small features will be too coarse, however if the filter is too large the complexity of a model increases significantly with little benefit.

Number of Layers: additional layers tend to improve performance, however they also increase a models complexity and thus its training time - this means that fewer model iterations are possible with a set time period. Back propagation issues with layers failing to train, can also arise.

Filters per Layer: additional filters likewise tend to improve performance, and again there is likely to be a cut-off point where diminishing returns are outweighed by increased model complexity and training time.

Layer Connectivity: variations in locally-connected and fully-connected layers can change performance dramatically, such as exhibited in the difference between convolutional layers, connected layers and those with dropout.

Input and Output Data Encodings: different vector encodings change the way the network learns. Images for example with a height, width and three colours per pixel are compressed into a one-dimensional vector as an effective input encoding.

Error Space, or Bound: changes how the network perceives error, and thus fundamentally effects what it learns during the back-propagation optimisation period.

Initialization of Weights: can also alter how a model learns. There are a number of different possible approaches to this: such as uniformly, randomly, as a Gaussian, unsupervised pre-training and more.

Auxiliary Layers: in ConvNets for example, pooling and normalization layers are often applied, however each has it's own set of additional parameters to tweak and a different effect on the model, thus requires complex tuning.

Non-linear functions: can make a large difference on model performance: the choice of which non-linearity you choose, for example choosing a 'Rectified Linear' neuron as opposed to a 'sigmoid'.

Optimization Parameters: such as step-size, or learning rate, regularisation, mini-batch sampling all need to be tuned for maximum accuracy and convergence speed. While there are common algorithms that help choose these parameters, such as AgaGrad (Duchi et al. 2011), manual tuning is often still required, and is difficult to get right.

Momentum Co-efficient: adds a fraction of the previous weight update to the current one, and is used to prevent the system from converging to a local minimum or saddle point, and increase the speed at which it converges. Too high and risk of overshooting the minimum, and too low the system might still hit a local minima.

2.2 Black Box Problem

2.2.1 Overview

While there have been a number of improvements to neural networks over the years (such as the development of drop-out, or deeper architecture) they remain to be considered by many as a black box algorithm, especially in comparison to some other better studied and less complex machine learning techniques such as support vector machines or logistic regression. Indeed many popular machine learning competitions are still won by those better understood algorithms (Adams et al. 2015).

There is still no clear understanding of why they perform so well or why certain combinations of internal weights and connections enable highly complex tasks, such as computer vision, to be performed. It is due to this lack of understanding that the development of new models falls largely upon a 'greedy' trail and error approach to tuning the network parameters. This is unsatisfactorily unscientific, using experience and intuition as the primary guiding factors - making insights hard to replicate.

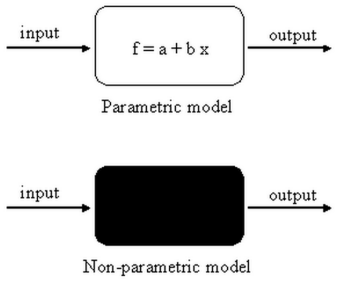


Figure 6: Black Box problem

2.2.2 The Challenges

There are a number of challenges that arise in attempting to change this way of working; firstly, these networks are composed of many functional components, the values of which as individuals and as a whole are not readily understood. In addition, each component of a network may have dozens of hyper-parameters linked to it, every one of which needs to be tuned to attain optimal performance. Finally, exacerbating these issues is that literature hasn't formalised methods for development or discussion, so even experts can only rely on others anecdotal results to guide network design.

In real terms, this means that designing and debugging deep neural networks is error-prone and time-intensive.

2.2.3 Possible Solutions

It is hoped that alternative work flows may provide some deeper insight. (Jarrett et al. 2009) for example uses a number of pre-evaluated models compared against number of datasets to make more informed decisions, this however doesn't leave room for new discovery. (Bergstra et al. 2013) uses a less human involved approach by using Bayesian statistics to automate the search of the parameter space, this is however computationally demanding and doesn't always provide an optimal solution.

A further area is to support decision making with visualisation allowing for the constant evaluation of networks to help researchers better understand the trajectory they are taking their models in as they go through the standard trail and of error tweaking different parameters. This is the approach that is being explored in this project.

2.2.4 Existing uses of visualisation

It's important to stress here that this is not a novel idea, and similar projects have been undertaken across a variety of areas within Machine Learning, in the visualisations of the naive-Bayesian network (Becker et al. 2001), decision trees (Ankerst et al. 1999), Support Vector Machines (Caragea et al. 2001) and Hidden Markov Models (Dai & Cheng 2008). Studies have shown that integrating such tools into the learning work flow can in fact produce better results than automated techniques alone (Ware et al. 2002).

3 Searching for a Solution

3.1 Human & Computer Augmentation

3.1.1 Solving Hard, Complex Problems in the Real World

When former world champion chess grandmaster Garry Kasparov was beaten by IBMs deep blue in February 1996, the headline was that Artificial Intelligence had finally surpassed human intellect. However following that loss Kasparov founded a competition known as freestyle, or advanced, chess - here human chess players use software to augment their play. The results were significant: humans who teamed up with machines could beat any of the autonomous machines. So while AI is often heralded, it's important to recognise that humans still bring important qualities to the intelligence scene.

$$\mathbb{IA} > \mathbb{AI}$$

Today far more sophisticated AI algorithms have been developed, and often included in the list of the best are Deep Neural Networks. Where companies like PayPal and Palantir use machines to process data and humans to analyse it - often through visualisations - to perform complex fraud detection tasks, perhaps by using the computer as a lever to analyse large datasets (the output of neural networks) it is possible to perform better algorithm design that either humans or computers can alone.

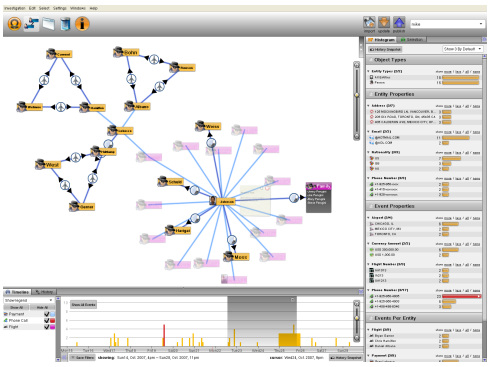


Figure 7: Palantir Visualisation tool: Humans augmented by computers

This idea was not developed by PayPal, indeed in 1999 Stuart Card commented in his book *Information visualisation* that *"The use of computer-supported, interactive, visual representations of abstract data can amplify cognition" (?).*

Visualisation can help us notice things that were previously hidden. Even when data volumes are vast, patterns can be identified quickly and with relative ease.

Visualisations convey information in a way that makes it simple to share ideas with others as well - it lets people say "Do you see what I see? And it can even help answer questions like "What would happen if we made an adjustment to that area?. This is perhaps a side benefit of visualising neural networks, that it not only enables researchers to understand their models better, but also allows them to share these learnings more effectively than simply their accuracy scores might.

3.1.2 Active Vision & Problem Solving

There has been a small revolution in our understanding of human perception, sometimes called 'active vision' (Ware 2010). Active vision means that we should think about graphic designs and visualisations as more than pretty images, but as cognitive tools that enhance and extend our brains. Diagrams, maps, web pages, information graphics, visual instructions, and more regularly help us to solve problems through a process of visual thinking, using the enormous proportion - almost half - of the human brain that is devoted to the visual sense.

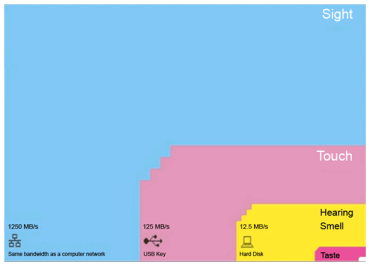


Figure 8: Tor Nrretranders Brain Bandwidth

Danish Physicist Tor Nrretranders discusses the "bandwidth of our senses in computer terminology to give an idea of the power of this visual system. In the diagram it's important to observe the comparison to the small white box at the corner which is *0.7%* of total power and is what we are aware off when all this processing is happening (Tufte & Sigma 2012).

"We are all cognitive cyborgs in this Internet age in the sense that we rely heavily on cognitive tools to amplify our mental abilities. Visual thinking tools are especially important because they harness the visual pattern finding part of the brain." (Ware 2010).

When producing data visualisations it is important to think about the particular details of design. What does it take to make a graphic symbol that can be found rapidly? How can something be highlighted? The problem for the designer is to ensure all visual queries can be effectively and rapidly served (Keim 2002).

3.2 Visualisation Theory

3.2.1 Overview

Visualising quantitative information, such as the data produced by neural networks, typically involves displaying measured quantities, or data, by means of the combined use of points, lines, a coordinate system, numbers, symbols, words, shading, and colour. These visual forms are more rapidly understood and are easier to critique than the information underlying them (DeFanti et al. 1989), (McCormick et al. 1987), (Tufte 2001).

In a numerical format vast quantities of data can be tedious to process, and often little understanding can be gained from such complex models. Visual data on the other hand communicates to the highly developed visual pattern-recognition capabilities of humans. Indeed, a majority of our brain's activity deals with the processing and analysis of visual images. Images

are pre-attentive and are processed before text in the human brain. Several empirical studies show that visual representations are superior to verbal or sequential representations across a number of different tasks; illustrate relations, identify patterns, to present overview and details, to support problem solving and to communicate different knowledge types (Burkhard 2004). As a species we are far better at recognising regularities, anomalies, and trends in images rather than in long lists of numbers (Ware 2010). Consider how difficult is may be to observe both global and local patterns in a list of numbers, in comparison to the relative ease when presented in a standard visualisation model such as a graph.

For data mining to be effective, it is important to include the human in the data exploration process and combine the flexibility, creativity and general knowledge of the human with the enormous storage capacity and computation power of computers. Visual data mining techniques have proven to be of high value in exploratory data analysis and they have high potential for exploring large datasets.

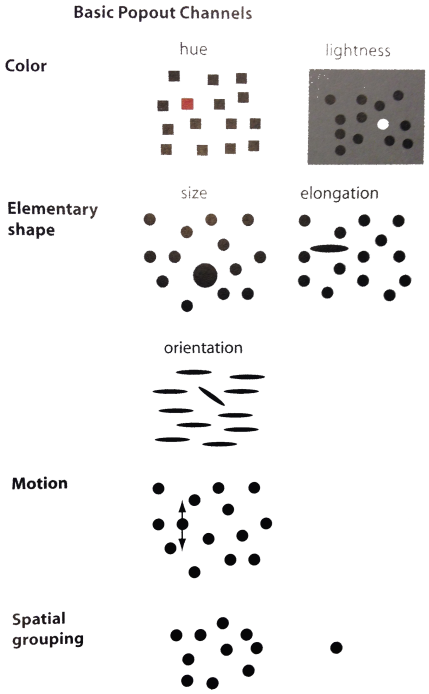


Figure 9: Ware's "Things that pop-out"

Visual data exploration is especially useful when little is known about the data and the exploration goals are vague - such as when attempting to understand the inner workings of a neural net. Since the user is directly involved in looking at the visualisation, shifting and adjusting the exploration goals of the human eye can be automatically (Keim 2002).

The canonical example of the usefulness of visualisation lies in the Anscombes quartet, where the four sets of numbers in the quartet have many identical summary statistics - mean of x values, mean of y values, variances, correlations and regression lines - but vary wildly when graphed (Shoresh & Wong 2011):

a

I		II		III		IV	
x	*y*	*x*	*y*	*x*	*y*	*x*	*y*
10	8.04	10	9.14	10	7.46	8	6.58
8	6.95	8	8.14	8	6.77	8	5.76
13	7.58	13	8.74	13	12.74	8	7.71
9	8.81	9	8.77	9	7.11	8	8.84
11	8.33	11	9.26	11	7.81	8	8.47
14	9.96	14	8.10	14	8.84	8	7.04
6	7.24	6	6.13	6	6.08	8	5.25
4	4.26	4	3.10	4	5.39	19	12.5
12	10.84	12	9.13	12	8.15	8	5.56
7	4.82	7	7.26	7	6.42	8	7.91
5	5.68	5	4.74	5	5.73	8	6.89

b

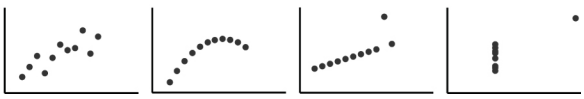
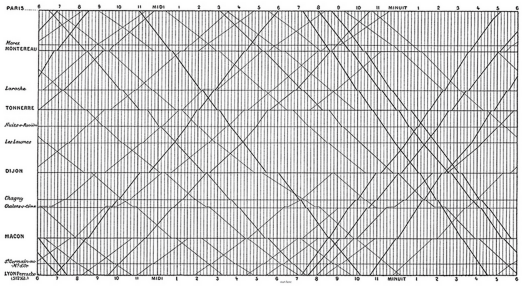


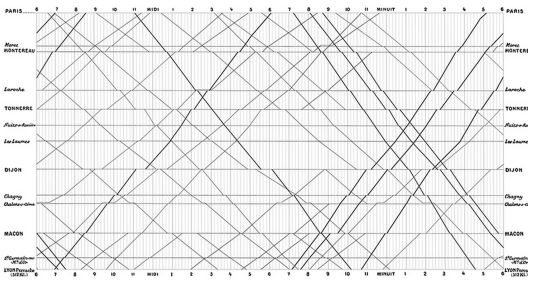
Figure 10: (a) The four sets of numbers that form Anscombe's quartet - (b) The highly distinctive graphs that result from plotting the data in a.

3.2.2 Tufte's Rules

Edward Tufte, a founding figure in laying out the core principles of data visualisation, provides us with a set of basic commandments (Tufte 2001):



(a) Poor Line Weights: unclear



(b) Better Line Weights: clear

Figure 11: Tufte's train line chart demonstrating excessive data-ink

- **Principle One:** *show only as much information as is required*

  This is Tufte's *data-ink* principle - irrelevant content is distracting, so should be removed. It is common place today to find charts and graphs with all sorts of three dimensional effects, unwanted background images and colours. The idea of having a data-ink ratio is to show only as much information as is required.

$$\text{Data-ink ratio} = \frac{\text{data-ink}}{\text{total ink used to print the graphic}}$$

- **Principle two:** *include visual differences only when required*

  The human brain has an amazing capability of spotting visual differences such as color, size and position. Often they look for the meaning to change depending on how these visual features and designed. If there is no difference, but embellishments are added, it often leads to confusion.

- **Principle tree:** *use visual encodings for quantitative values*

Successful examples are: length, for example the length of bar in a bar graph; 2-D location, for example the position of a data point in a scatter plot; size, for example the area in a pie chart; shape, orientation or hue, for example denoting different classes in any graph. All of these are automatically and immediate understood as they have natural properties that humans understand.

- **Principle four:** *differences in visual properties should correspond to actual differences in the data*

  Its important to encode differences consistently and not manipulate the visualisation to aid an argument. For example, ensuring that axes are consistent - from zero to some useful value without undergoing any form of distortion.

- **Principle five:** *do not visually connect values that are discrete*

  In a graph, when you draw lines between discrete values and connect them, people perceive those values as having a relationship to each other, and so this should be avoided.

- **Principle six:** *visually highlight the most important part of your message*

  All information on a chart might not be equal and it might be possible to direct a users attention to a particular part of the visualization by visually highlighting through use of color, position or another standard encoding.

- **Principle seven:** *augment short term memory through visual patterns*

  The human brain is limited to retaining around four pieces of information at any given time. By presenting quantitative information as visual patterns, more information can be simultaneously stored as one 'piece'.

- **Principle eight:** *Encourage the eye to compare different pieces of data*

  Information is not something that exists in isolation, and often by comparing pieces of information one is brought to new conclusions about that data.

- **Principle nine:** *Reveal the data at several levels of detail*

  Quantitative data often has several scales, with patterns appearing at both a global and local level. By enabling the data to be viewed at different levels of detail the data can be explored in all it's complexity.

- **Principle ten:** *Don't distort the data:*

  Often it is tempting to change the scale on a graph for it to 'fit' appropriately, or to crop the data hiding anomalies. With these elements of distortion the full picture is not revealed, and the purpose of visualisation compromised.

3.3 Existing NN Visualisations

Visualisation has been around helping researchers with neural networks for a long time, and techniques such as the *Hinton diagram* were first demonstrated as early as 1986. This section

provides a brief overview of similar techniques from around the nineties, where a number of the techniques are going to be visualisations of fig. **??**.

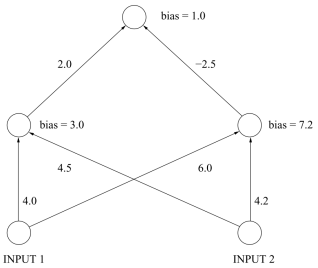


Figure 12: Simple Neural Network

3.3.1 Hinton Diagram

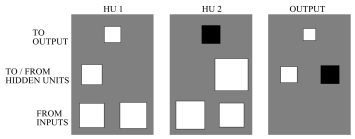


Figure 13: Hinton Diagram

One of the first practical visualisations of ANNs was the *Hinton Diagram* (Hinton 1986). It visualises the weights and biases related to a node within a network. Weights are represented as boxes, where its area represents the weights magnitude, and it's shade represents the sign on the weight - white is positive, black is negative. Biases are illustrated as weights from a node back to itself. There is a vague representation of the architecture as output nodes appear at the top of a diagram, hidden nodes are in the middle, and input nodes are at the bottom. However these diagrams are rather unclear, and lack of topological information is a problem. The advantage is they make it easy to see the signs and magnitudes of the weights that contribute to a neurons activation.

3.3.2 Bond Diagram

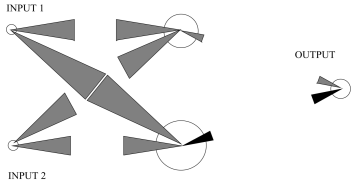


Figure 14: Bond Diagram

Similar to the *Hinton Diagrams*, the Bond diagram (Wejchert & Tesauro 1990) graphically depicts the values of the networks weights and biases. The bond diagram however attempts to

make the architecture of the network more clear; a neuron is depicted as a circle, where the diameter of the circle indicates the magnitude of the bias, and triangles connecting the circles represent the weights. The magnitude is indicated by the height of the triangle, and colour depicts the sign.

While it is perhaps easier to decipher the network structure from the Bond diagram, it is harder to gauge the relative importance of the weights and biases which have been depicted with different shapes. It makes the following question very difficult to answer: "which input units need to be active in order for the net input to exceed the threshold (bias) of the hidden units?" (Craven & Shavlik 1992), a useful question that Hinton diagrams are far better at answering.

3.3.3 Hyperplane Diagrams

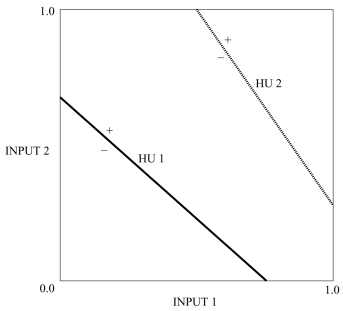


Figure 15: Hyperplane Diagram

A hyperplane depicts the 'threshold' of a decision surface. As this hyperplane moves throughout the training process, visualising the hyperplane as it moves can be a useful method to get an understanding of what a neuron is learning (Munro 1992). Neurons that appear in the same layer can have their hyperplanes shown in the same diagram due to a sharing of input space, making comparison easy.

One issue with this hyperplane representation is that while accurately representing a threshold function acting on a two-dimensional input space, the diagrams fall down when compared with most contemporary ANNs that require multiple dimensions (¿3) to be shown and more commonly use continuous transfer functions such as the sigmoid - which requires a gradual, rather than a sudden, division of the input space. That said, it can be assumed that the hyperplane is a close approximation of the gradual boundary and so can still provide useful observations.

3.3.4 Response-function plots

Figure 16: Response Function Plot

Response-function plots are very similar to hyperplane diagrams - they also display the decision surface. They differ in their solving of the issue of the gradual boundary. Instead of displaying

the space using a hyperplane, the space is displayed as a gradient of values to indicate the resulting activations.

Interestingly, both the Response-Function Plots and the hyperplane diagrams show the space between two successive layers of neurons. This provides only a fraction of information about the network, and problematically may lead to false assumptions about it. One way to address this is to describe the decision surface not just on the layer below, but across all previous layers of the input space.

3.3.5 Trajectory Diagrams

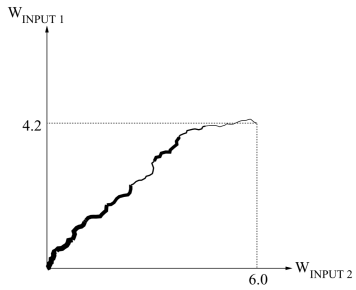


Figure 17: Trajectory Diagram

Trajectory Diagrams (Wejchert & Tesauro 1990) depict the change in weight space and in error over a neuron during training. These diagrams use the incoming weights of a neuron to create the axes of a plot. During training as the weights change they are visualised as a trajectory in the weight space. The error at a given time is indicated by the thickness of the trajectory line.

Again, along with many of these other early visualisation methods, the weakness of the trajectory diagram is its inability to display weight spaces of more than three dimensions. There have been efforts to combine dimensionality visualisation with trajectory diagrams - such as using radially projected axes, however this is fairly unsuccessful (Craven & Shavlik 1992).

3.3.6 Lascaux

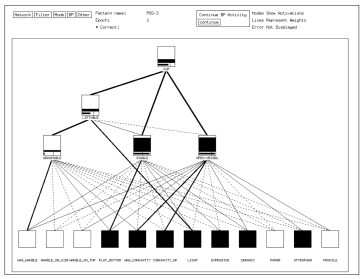


Figure 18: Lascaux Clip

Lascaux is a visualisation tool proposed by (Craven & Shavlik 1992) that aimed to clearly display the topology of a network. Here, each neuron is represented as a box and network weights are represented by interconnecting lines. A weights magnitude is visualised by the thickness of a line, and the positive or negative signs are visualised as solid and dashed lines respectively.

The tool depicts a range of information it one place. Activation of each neuron is show as a vertical bar within the neuron 'box'; a horizontal bar shows the net input relative to a threshold - shown as a line intersecting the bar; error is another vertical bar within the neuron box; a separate diagram shows the error propagating as connections between these boxes - where thickness describes magnitude.

The issue with *Lascaux* is that too much information is being displayed in a small space ineffectively. The approach uses standard two dimensional visualisation techniques, and simply squashes them into a neural network architecture. This makes the topology easier to understand, but at the sacrifice of more important elements.

3.3.7 Visualising Weights and Connections

When representing weights, it is important to consider the analytical impact of a visual decision. (Streeter et al. 2001) visualises the topology of the network but doesn't clearly show the weights themselves. This can lead to confusion when assessing the importance of a neuron. Consider for example a neuron that has appears to have a high value in one layer, however is subsequently cancelled out by low weights deeper within the network.

One problem here is that since the absolute values of the weights are used, the result does not provide the direction of the relationship.

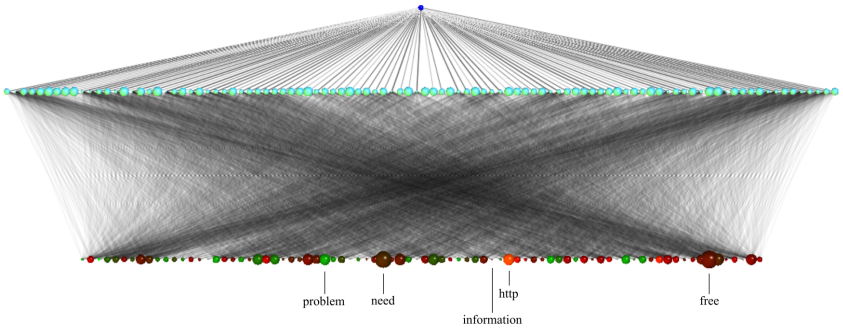


Figure 19: Tzeng Map

(Tzeng & Ma 2005) based on the work of (Garson 1991) and (a.T.C. Goh 1995) sought to solve this problem in a different way; by visualising the weights with line-thickness between nodes, thus making it easy to identify when a node is insignificant regardless of the magnitude of weights applied to it.

In addition (Tzeng & Ma 2005) propagate all of the layers influence through the network by multiplying each weight between the previous layers with those of the successive layers which connect to the same node. Here, they represent the contribution of a specific hidden node by adjusting the diameter of the circle visualising the neuron in their visualisation. The contribution of the input unit i to the output unit o through a hidden unit j is computed by multiplying the input-hidden weight strength and the hidden-output weight strength: $r_{ijo} = w_{ij} \times w_{jo}$, and the

relative contribution from each input node k to a hidden node j can be represented as:

$$r_{ijo} = \frac{|C_{ijo}|}{\sum_{k=1}^{m} |C_{kjo}|}$$

where the total contribution from an input node i is:

$$S_i = \sum_{j=1}^{n} r_{ijo}$$

and the relative importance of an input node is therefore:

$$RI_i = \frac{S_i}{\sum_{k=1}^{m} S_k}$$

This combination of statistical analysis and weight representation allows for a visualisation that demonstrates not only the raw data, but an abstraction that is more useful to the researcher given the relative importance of the nodes, and significance of the data - while still providing an architectural understanding of the network. This combination of mathematics and visualisation is one that continues across a number of other visualisation techniques for neural networks.

3.3.8 Features

Another popular part of visualising neural nets, is visualising the features of a CNN to gain an intuitive understanding about its internal behaviour is becoming commonplace, it is mostly limited to the simple visualisation of the 1st layer where projections to the pixel space are relatively easy to achieve. However there are exceptions, and a small number of researchers have developed methods for visualising deeper hidden layers.

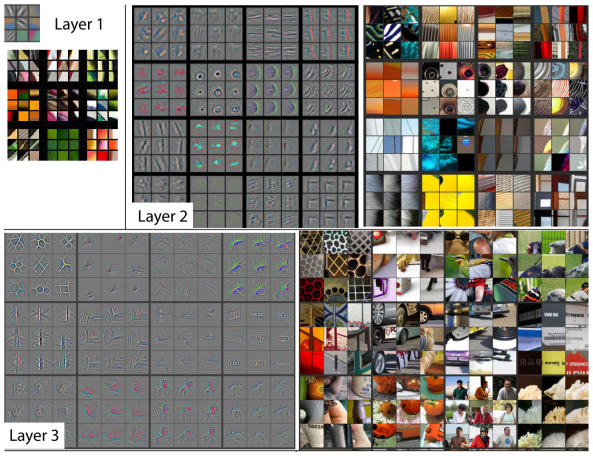


Figure 20: Zeiler Deconv

(Erhan et al. 2009) sought to find the optimal stimulation of a unit activations through gradient descent in the image space. This has been criticised as difficult to obtain due to the need for careful initialization, and the lack of information conveyed about a units invariance.

(Le et al. 2010) show how the Hessian of a given node may be computed numerically around an optimal response - thus fixing the formers shortcomings by providing a view of invariances. The issue with this approach is with the higher layers where invariances become increasingly complex and are thus poorly encoded in their quadratic approximations.

(Vondrick et al. 2013) use feature inversion algorithms, where an image is featurized and then recovered to a transformed but decipherable format - again to give intuitive access to abstract feature representations formed by the network. Using this technique they discovered single deep neurons that were trained to respond to faces and bodies, both human and animal.

(Zeiler & Fergus 2013) provide a technique called *Deconvolution* (Zeiler et al. 2011) which effectively reverses a convolutional network. Deconvolution is a type of feature inversion that renders re-weighted versions of inputs, highlighting areas, patterns and textures of an image deemed most important by a particular part of the network. It essentially approximates a reconstruction of the input of each layer from its output.

(Donahue et al. 2013) show visualisations identifying patches in a dataset that cause strong activations at higher layers in a network. However these have been criticized as only producing a cropped version of the input images, so are limited learning tools.

(Simonyan et al. 2013) describe a technique for visualising class models learnt by CNNs. Given a CNN and a class of interest, the visualisation method numerically generates an image that is representative of the class in terms of the CNN class scoring model.

Clearly with such a lot of attention placed on visualising featurizations, it's a significant opportunity to learn about the networks. It's important to realise however that one of the above is not necessarily better than the others: each show a different element of the featurisation, and as experts still know relatively little about the behaviour of ANNs it's important to not discard any of these visual aids rashly.

4 Project Goals

To help guide the development of a useful visualisation based research tool, a survey of existing researchers was performed. The results of which determined the initial direction for the project.

4.1 User Survey & Interviews

4.1.1 Survey Design

The intial user survey was developed in collaboration with this projects supervisor, Jack Kelly, and was distributed amongst a small number of Imperial College Staff and students known to be working with Neural Networks. The following areas were addressed.

- **Describe your working environment**

  This had specific subtopics asking about languages used, packages researchers were familiar with and time taken to develop working networks.

- **Describe your training methods**

  This asked specifically about how often neural network design / architecture parameters were tweaked, which of those were considered to be most important, which were the most frustrating changes made and how these challenges were currently solved.

- **Choose which visualisation technique you think is the most useful**

  A number of examples from the previous section were shown with weights, gradients, activation mapping, architecture graphing, classification distribution and filters all included.

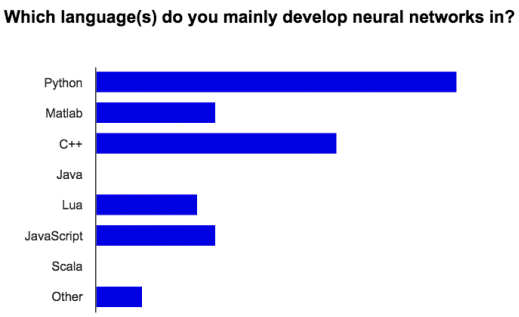
- **Choose which element you think would be most useful to visualise**

  A list of all the different parameters that could be tweaked during training were given.

- **Do you currently visualise neural networks, and if so - how?** The question asked researchers to mention specific packages that are known to be commonly used amongst research communities and also asked about preferred methods for interacting with the software, such as *.csv* upload or model upload.

4.1.2 Survey Results

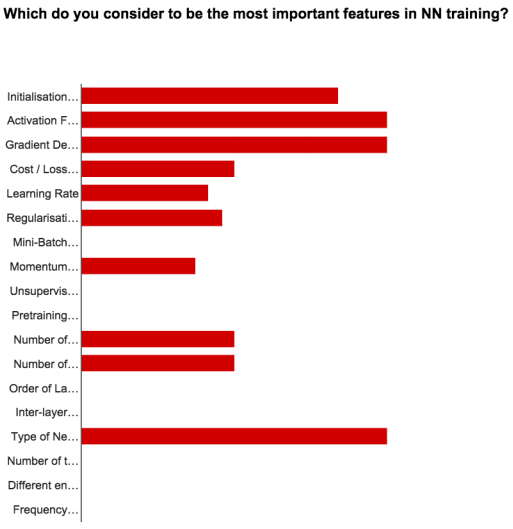
A relatively small number of researchers responded to the survey, however a second round of delivery was postponed in favour of working on product development and continued research into Neural Networks.



In response to working environment it was discovered that there was no package that researchers used more so than any other - and often single researchers would use multiple different neural network packages depending on the task at hand. For example: CUDA, Caffe, Lasagne, Torch and others.

For some researchers the tweaking of parameters and adjusting of network architectures was the subject of whole PhD's - suggesting tools which aimed to help analyse the success of these changes could be an invaluable addition to their research toolkit.

A wide array of methods were described for deducing the correct network parameters, or judging the quality of one set versus another. One example demonstrating the *'hacky'* nature of tools currently used goes as follows: the researcher would get up on the screen various weight matrices from different training epochs that corresponded to a layer deep within the network and would simply switch tabs as fast as possible to try and observe changing numbers or patterns in the data - signs that the network would be training. While this method appeared to work for this particular researcher, the functionality could certainly be improved by some simple visualisation implementations of the data.



With respect to commenting on existing visualisations, it was surprising to see that most hadn't thought much about visualisation as a serious tool beyond graphing the commonly used error rates or accuracy. The majority of the time, researchers would simply use visualisation as a way to demonstrate results.

Importantly, while current usage was limited, researchers generally appeared interested in the possibilities visualisation might hold.

4.2 Goals

A number of goals were set after the initial survey, review of neural networks, visualisation theory & existing visualisation software.

- Improve the visualisation tools currently available for neural network researchers
- Provide Visualisations that demonstrate visualisation is actually valuable as a method of research
- Visualise changing parts of the neural network: weights - bais matrices, activations etcetera
- Create a visualisation tool that adheres to the visualisation principles set out by Edward Tufte
- Investigate not just one visualisation method, but explore a range of options to demonstrate the value of any final product decided upon
- Create a tool that is easy for researchers to interact with
- Provide a visualisation tool that doubles up as a tool for collecting network data for other data-mining purposes
- Keep a full record of experiments taken, and their outcomes
- Real time visualisation updates to understand if a currently training network is developing as expected

5 Data Collection

Visualising neural networks is a complex task. In particular there are three major challenges to address: producing the data, collecting the data and visualising the data.

The final challenge is the reason for this thesis, so only this chapter will address the first two of these difficulties. In particular this section explores the Neural Network implementations used, the dataset implemented upon and some further considerations made when collecting the data.

5.1 Neural Network Implementations

Often research work pertaining to neural networks requires some new, advanced, model that solves a unique task or demonstrates significant efficiency gains.

The purpose here however is quite the opposite. Implementations should be very familiar to those looking into this study. This ensures that a complex network does not obfuscate the true goal of this report - which is to demonstrate the value of a visualisation tool for learning about these networks.

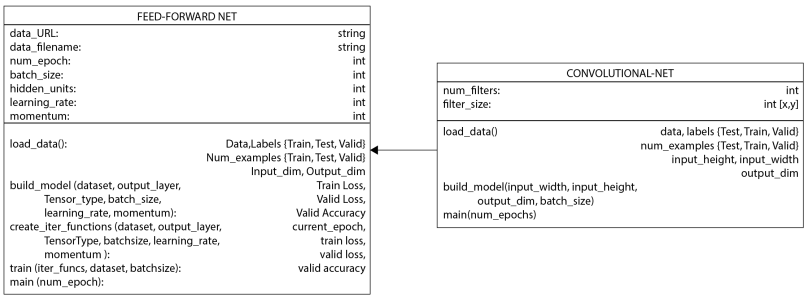
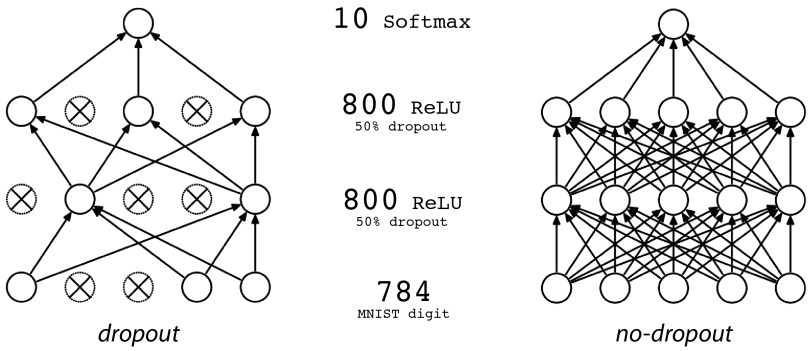


Figure 21

5.1.1 Feed Forward Net

Network models used were adapted from classic architectures: the feed forward network was an implementation of a network proposed in the 2012 Geoffrey Hinton paper in which he explains the concept of drop-out (Hinton et al. 2012) - an idea that addresses a significant problem in machine learning - over fitting. This network is well understood due to dropout being one of the major recent advancements within the neural network research scene.



The network itself takes 784-input values (the (28,28) MNIST images flattened into a single array) followed by two layers of 800 ReLU units and an output Softmax layer of 10 units - digits zero to nine.

Other parameters used from the Hinton paper include a 50% drop-out rate between hidden layers and 100-sized mini-batches. The implementation in this project differend in the number of epochs the experiments were run for. This was a trade off that enabled more visualisation tests to be performed, and is afforded by the minimal need to prove an accurate model.

5.1.2 Convolutional Net

The Convolutional architecture used throughout the project was a common adaptation of Yann LeCun's 1998 LeNet (**?**), the first network to successfully classify handwritten digits:

- Layer Input: (60000 [size of training set], 1, 28, 28 [dimensions of MNIST image])
- Layer 1: Convolutional Layer: 32 Filters, (3,3) Filter Size, (2,2) Pooling
- Layer 2: Convolutional Layer: 64 Filters, (2,2) Filter Size, (2,2) Pooling
- Layer 3: ReLU layer: 500 Units, 50
- Output: Softmax: 10 Units

5.1.3 Alteration

The above implementations are know to produce desirable results, or low classification error, however they are altered in a number of ways throughout this project.

In order to explore the ability of visualisation methods to capture interesting and important patterns within a networks output data such that it can influence researchers decisions, the network must be *broken* in across a number of different parameter settings.

With a variety of different parameter settings, it is possible to simulate the changes that a researcher might make when exploring the parameter space for optimal performance. It is hoped that a number of these flaws, imperfections, quirks and other visually identifiable qualities in the networks data can be isolated and resolved.

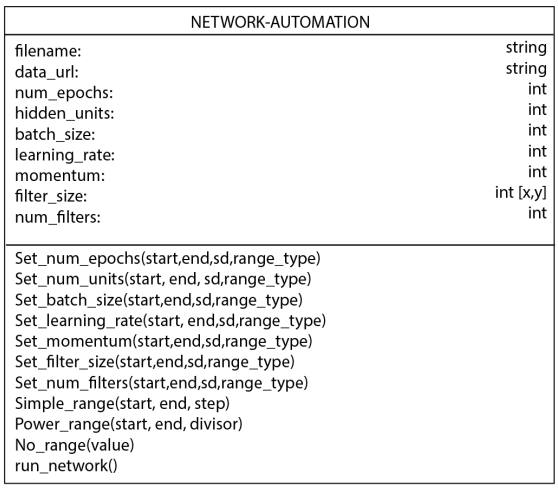


Figure 22: AutoNets

To successfully implement these changes, a power range was defined that would run fewer experiments as it neared the understood value - *[1, 2, 3, 5, 7, 11, 17, 25, 38, 57, 86, 129, 194, 291, 437, 656]* - these values are examples run for the number of hidden units used within the networks. However other parameters were automatically tweaked including number of epochs, learning rate and momentum. Other parameters which required tweaking in the neural network code itself were only occasionally adjusted - such as the number of hidden layers, or type of non-linearity used.

5.1.4 Practical Implementation

The neural networks described above were implemented in a library designed for neural networks called *Lasagne*.

Lasagne, is a neural network wrapper for the common machine learning python library *Theano* which uses symbolic functions that compile before runtime into *cython* to ensure efficient mathematical processing, often displaying efficiency gains of up to 10 times.

Lasagne was chosen as the network implementation package after the survey of researchers revealed that the largest minority of researchers currently used Python implementations of neural networks. Theano could have also been used, however was too detailed for the purposes of this project.

It's important to note that in this incredibly fast moving field that the frontrunning technology is continually changing and throughout the course of this project another library *Torch* has become increasingly popular due largely with it's ability to handle *Recurrent Neural Networks*. These haven't been mentioned in this report for simplicity reasons.

5.2 Dataset

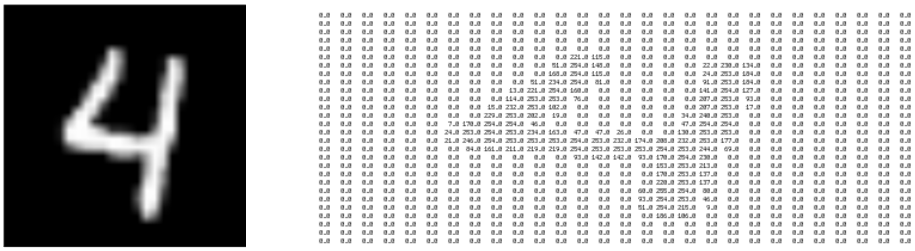
All experiments for this project were conducted using the MNIST dataset. The dataset is widely used as a benchmarking dataset not just within neural network community but in the wider machine learning community as a whole, making it appropriate as a test dataset for

visualisations. The dataset itself contains 60,000 training images and 10,000 test images. Each image is a 28 pixels by 28 pixels hand-written digit from one to nine.

Figure 23: 100 MNIST digits

In addition to being a natural benchmarking dataset, with it's complex structure across the images two dimensions the dataset is not only a useful dataset to test Feed Forward networks, but also ConvNets.

Both of these criteria could have also been satisfied by one of the CIFAR datasets, however MNIST was chosen due to it's representation being confined to one set of intensity values, rather than the more complex CIFAR set with Red, Green and Blue dimensions to consider. The below images demonstrate the simplicity of the dataset.



(a) MNIST digit (b) Intensities

Figure 24

5.3 Collecting Output Data

- Data used online must be stored in a compact format. This was achieved using a MongoDB database to store *JSON* objects in binary - *BSON*. In earlier iterations of the project this was interfaced with using a JavaScript backend, and in the later iteration of the project using the python library `PyMongo`.

- Data that researchers use to assess the quality of their models should also be stored in a common format that can be easily interrogated. This was achieved by using Pythons sophisticated `sys` and `os` packages to store the weights, biases, activations and other parameters in *csv* format.

- Data from the neural networks must retain it's shape for easy importing with the python library `numpy`. This was achieved by either using *numpy* to process the data, or by creating bespoke input-output functions that would be able to reconstruct the data's shape.
- Image data must be transformed in order to be processed with MongoDB. This is achieved by preprocessing image data with the python library `base64` which parses the images into a Float32 array in base64 notation - this then usefully becomes small enough to store in the *MongoDB* database.
- Coordinate data for tSNE plots (explained later) must be stored in a condensed format that enables easy interaction with the *d3.js* visualisations. Here, (500, 2) matrices are `numpy.reshaped` to (1000,) single dimension arrays which can be easily parsed by the d3.js client configuration.

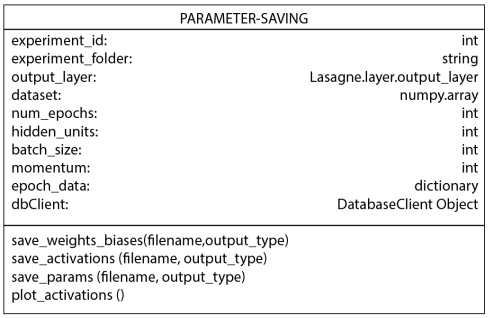


Figure 25: UML: Saving Functionality

The data collection methods described above interact with the complex machinery of a neural network. To effectively store the relevant data, it was important to fully understand how the discrete pieces of information interacted with one another. The following entity relationship diagram describes the majority of the features we are concerned with:

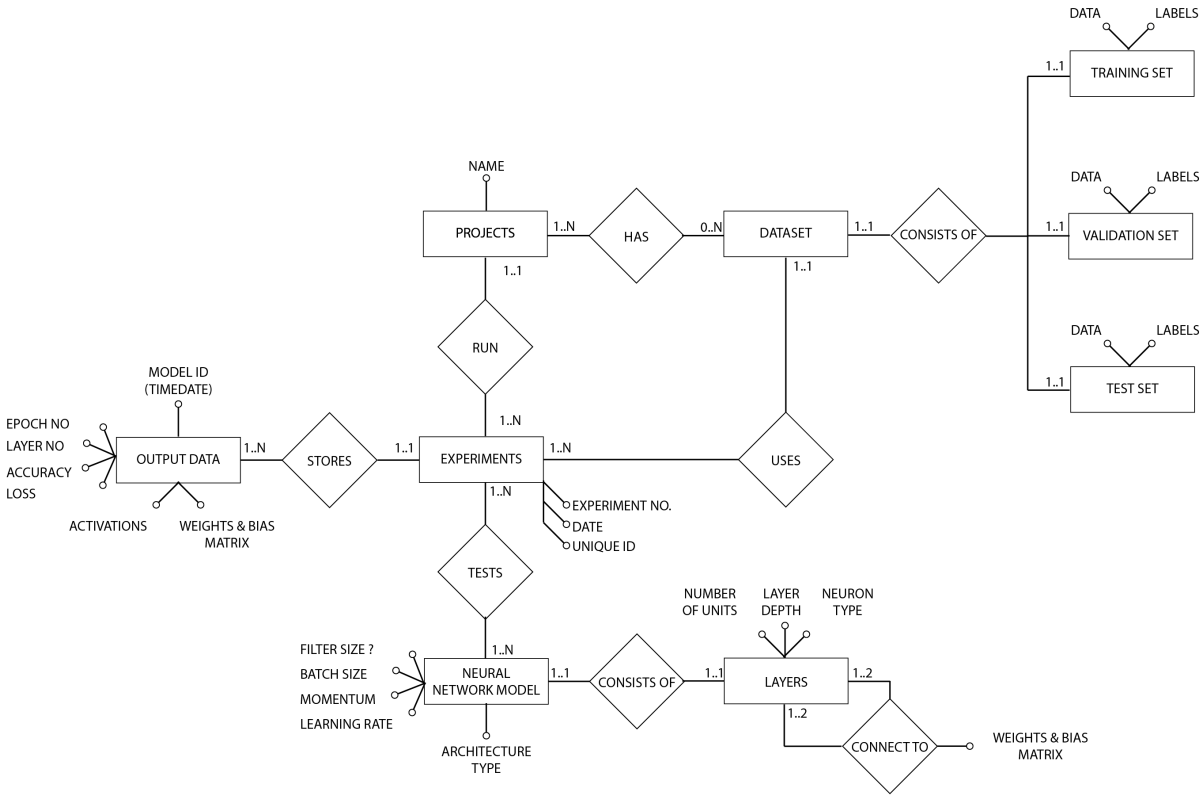


Figure 26: Initial ER diagram

5.4 Evaluation

The aim of this first section was to explain to the reader the foundation upon which this project was built. Namely, two well understood neural networks that get tweaked in various ways away from parameters that ensure optimal performance. This is done to better understand how poor networks may appear when visualised, and to be able to observe differences between those and the networks we know to be of industry standard. The section also explains the importance of gathering the data outputs of these networks in an appropriate fashion, and the reasons for choosing the MNIST dataset.

With a basic understanding of the neural networks and dataset used, and an understanding of the data collection methods, the remainder of the report will explore how this data can now be visualised.

6 Dimensionality Reduction

Neural Networks are famous for their ability to comprehend complex datasets such as in areas of vision or speech recognition. One of the principle complications in these fields results from complex high-dimensional datasets.

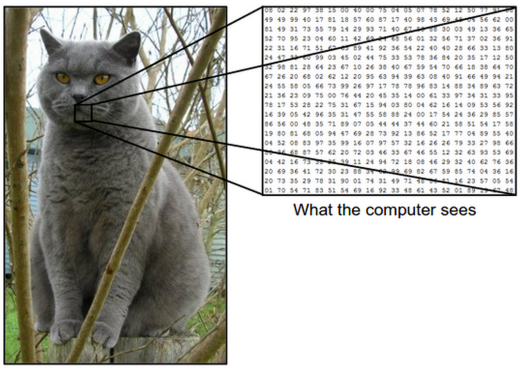


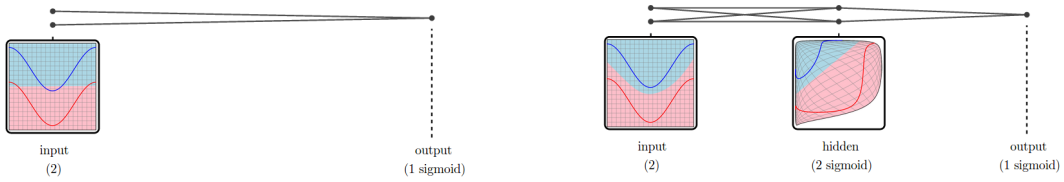
Figure 27: Cat

The image above is 248 pixels wide, 400 pixels tall, and has three colour channels Red, Green and Blue - which to the computer is stored in a multidimensional array of dimensions (248,400,3) or 297,600 numbers. When this image is passed into the computer it understands this data as 297,600 different data points - not as a cat.

As this image passes through a neural network, the number of dimensions (248,400,3) changes to make it easier for the network to classify as a cat - one of 10 classes in the CIFAR-10 dataset. So while a human can understand this image as a cat, when the data is transformed say to (512,20,2) dimensions the cat is no longer recognisable to to - but to a computer, may be more 'cat-like'.

It has been suggested that one reason for the success of neural networks is that they discover optimal representations of the data that allow for more accurate classification (Hinton 1986). These representations are captured in the later layers transformed data-space. Ultimately understanding these better should provide a method to guide the training process that is less situated in trial and error.

While the input space of network may require a relatively complex line to divide two curves on a plane, each new layer transforms the spatial data creating a new representation that is easier to classify with a simple hyperplane.



(a) No hidden layer (b) Hidden Layer

Figure 28: Chris Olah: Representations that warp the data

In order for the data to be transformed to this new representation, it must undergo a sequence

of manipulations. A tanh layer for example processing the function $tanh(Wx + B)$ consists of;

- a linear transformation by the weight matrix $\boldsymbol{W}$
- a translation by the bias vector $\boldsymbol{b}$
- and a point-wise application of the tanh activation function

Intuitively, what is occurring here is a stretching and warping of the space to make it easier to linearly divide - as can be seen above. It's important to note however that it does not cut, break or fold the space as it must retain it's 'topological' properties (Choi & Horowitz 2005).

Visualising the information contributing to these representations - the weight matrix, the bias vector and the activations - in a way that is understandable to the human brain, and therefore useful as a diagnostic tool, must map the vast number of dimensions present into the three spatial dimensions that we as humans understand with ease.

This section will explore the notion of both visualising high-dimensional data, where the aim is to retain data fidelity and show all of these high dimensional data points, and the notion of dimensionality reduction - where the aim is to reduce the number of dimensions mathematically.

6.1 Visualising High-Dimensional Data

Visualising high-dimensional data is a very important problem in several different domains that each deal with data of widely varying dimensionality. It is therefore a very well explored problem and a number of techniques for visualising high-dimensional data exist, a summary of which was composed by (Cristina et al. 2003).

This covers techniques by a number of different authors that could be useful for the visualising of neural network data;

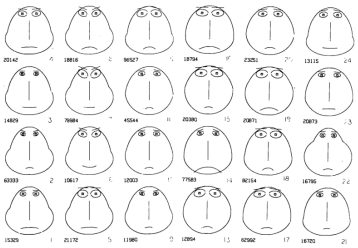


Figure 29: Chernoff Faces

Chernoff Faces are iconographic visualisations of faces by (Chernoff 1973); each point in k-dimensional space, $k < 18$, is represented by a cartoon of a face whose features, such as length of nose and curvature of mouth, correspond to points in the data. Thus every multivariate observation is visualized as a computer-drawn face. This presentation makes it easy for the human mind to grasp many of the essential regularities and irregularities present in the data. Looking at the faces it's easy to see which data points are similar and with which parameters - such as those faces with larger eyes would represent similarities across a common dimension. This technique is not useful for most neural networks which have greater that 18 dimensions.

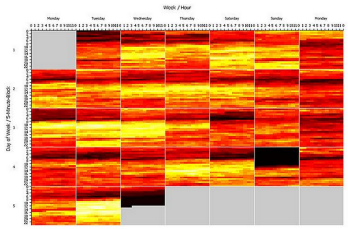


Figure 30: Pixel Based Techniques

Pixel Based; represent as many data points as possible on the screen at the same time by mapping each data value to a pixel of the screen and rearranging those pixels to suit the source (Keim 2000). One example is to use a gradient of colour to represent the value of a data-point, and multiple dimensions may be show in different as slices tiled together.

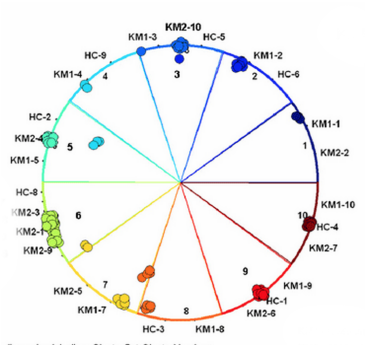


Figure 31: RadViz

Radial Coordinate Visualisation was designed by(Hoffman 1999); which for an n-dimensional visualisation, n lines emanate radially from the center of a circle and terminate at its perimeter, each line is associated with one attribute. The points that sit in amongst the radial portions represent the data described between the dimensions in a way that is similar to an x-y plot.

Evaluation

While these tools do have their uses as visualisation techniques, when it comes to exploring the high-dimensional data of neural networks they have been criticized (Maaten & Hinton 2008) as simply providing the tools to *display* more than two data dimensions, and leave a more difficult task of interpretation to the viewer. With the number of dimensions used in real-world neural networks often in the thousands, these techniques may provide limited insight, and so it's important to look in detail instead at Dimensionality Reduction which does some of the data interpretation for us.

6.2 Dimensionality Reduction

Dimension reduction differs from dimensionality visualisation, in that instead of visualising the multiple dimensions of a dataset in a format such as those already described, it actually converts the high-dimensional data set $X = \{x_1, x_2, ..., x_n\}$ into a low-dimensional data set that can then be displayed easily in a standard recognisable formats such as the scatter plot. Dimensionality

reduction aims to preserve as much of the significant structure of the data in higher-dimensions as possible while generating a low-dimensional representation that is easier for the researcher to interpret. This is fundamentally important for visualising neural nets where activations are often many thousands of dimensions.

It has been suggested by (Olah 2014*c*) that it is possible to draw a notion of how successful this dimensional reduction is by assuming that for any two data points, x_i and x_j there are two notions of distance between them that we can compare. First, is the distance between those points in the real world space, for example the L2 distance $d(x_{i,j}) = \sqrt{\sum_n (x_{i,n} - x_{j,n})^2}$, and the other is the distance between the points in the visualisation, $d_{viz}(x_{i,j})$, such that a cost function of the visualisations success can be defined.

If the cost C is high, then the distances are dissimilar to the original space, if low they are similar, and if zero the visualisation is a perfect representation. It's almost impossible however to get a perfect representation in all aspects, so different cost functions provide different compromises, and insights. Once the cost function is designed there simply exists an optimisation problem that can be tackled though a standard process such as gradient descent to ensure that points are optimally visualised with respect to the cost function. The cost function for standard Multi-dimensional Scaling (Torgerson 1952) is shown below:

$$C = \sum_{i \neq j} [d(x_{i,j}) - d_{viz}(x_{i,j})]^2$$

Another reduction method is Sammon's mapping (Sammon 1969), which aims harder to preserve the distances between nearby points than those further away. If the two points are twice as close in the original space than two others, it is twice as important to maintain the distance between them. This emphasises the local structure at the compromise of the global structure in the data:

$$C = \sum_{i \neq j} \frac{[d(x_{i,j}) - d_{viz}(x_{i,j})]^2}{d(x_{i,j})}$$

A number of other techniques were reviewed by (van der Maaten et al. 2009) who describes *Principle Components Analysis, PCA,* (Hotelling 1933) - which finds the angle that spreads out the points the most in order to capture the largest variance possible, and *Multidimensional Scaling* as seen above - as linear techniques that keep low-dimensional depictions of dissimilar points far away, but which fail to keep those data-points which are similar close together in the lower dimensional depiction.

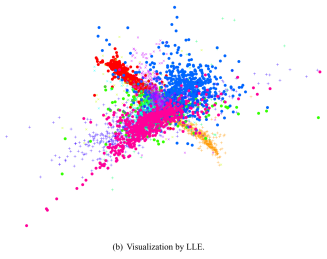


Figure 32: MNIST - a Locally Linear Embedding

In addition to Sammons mapping described above, (van der Maaten et al. 2009) also sites a number of other non-linear dimensionality reduction techniques that aim to preserve the local structure of data including; *Curvilinear Component Analysis* (Demartines & Herault 1995), *Stochastic Neighbour Embedding* (Hinton & Roweis 2002), *Isomap* (Tenenbaum et al. 2000), *Maximum Variance Unfolding* (Weinberger & Saul 2004), *Locally Linear Embedding* (Roweis & Saul 2000), *Laplacian Eigenmaps* (Belkin & Niyogi 2002).

These techniques all perform well with artificial datasets, however are criticised for not being capable of retaining both local and global structure in a single data map. Even semi-supervised variants are not capable of separating simple datasets such as MNIST into it's natural clusters (Song et al. 2007). Neural network data often requires the retention of both of these, and (Maaten & Hinton 2008) descibes a solution to this problem in the form of *t-Distributed Stochastic Neighbour Embedding.*

6.3 t-Distributed Stochastic Neighbour Embedding

t-Distributed Stochastic Neighbour Embedding (Maaten & Hinton 2008) has provided a successful and widely used alternative for neural network researchers. tSNE, as it is abbreviated, captures much of the local structure of high-dimensional data, while also revealing global structure such as the presence of clusters at several different scales.

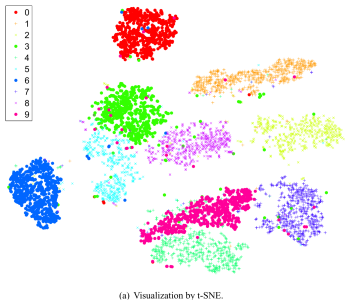


Figure 33: tSNE

tSNE can therefore be viewed as preserving the topology of the data, which as explained previously is incredibly important if we are to successfully capture the representations formed by these networks. tSNE constructs for every data point a notion of which other points are it's 'neighbours' and tries simultaneously to ensure that all points in the data have the same number of neighbours, in this sense is a lot like a nearest-neighbour graph, however instead having a set number of neighbours connected by edges, and non-neighbours for which there are no connections, data points in the tSNE reduction have a continuous spectrum of neighbours, for which they are neighbours to different, non-binary, extents. This makes tSNE very powerful in revealing global clusters and local sub-clusters within the data - which is ideal for working with complex neural network activations that should display a sophisticated understanding of both.

The one downside of tSNE is that it's prone to getting stuck at local minima, and due to it's increased complexity is more computationally expensive to run, such that changes cannot be made and visualised in real time on standard machines and can take any number of hours,

or days even, to produce.

6.4 Evaluation

Not one of the dimensionality reduction techniques mentioned appears to be superior. They are largely complimentary, and the choice of which to use intuitively depends on the needs of the data-set and the visualisation scenario.

Each has it's own trade off in order to preserve the most important properties for it's unique scenario. This is potentially obvious as there can be no exact mapping from high-dimensional space to low dimensional space, and so each situation must be evaluated separately.

PCA preserves linear structure, MDS preserves global geometry and tSNE tries to preserve a topological neighbourhood structure.

For the remainder of this project, the data produced by the neural networks will be reduced in dimensions using the tSNE algorithm, or a faster derivative Barnes-Hut-SNE (**?**).

Looking into neural networks it is unclear what exactly we are looking for, be it variance, local structure, global structure, or some unknown. tSNE preserves the overall topological structure and thus provides a good solution facing the wide array of unknowns. In addition, tSNE has been used incredibly successfully used in the past by some leading neural network researchers to visualise their data(Maaten & Hinton 2008), so it makes sense to follow their example.

Not only does tSNE satisfy several important criteria for enabling us to understand the high dimensionality of neural network data, but it also allows us to meet a number of Edward Tufte's theories of good visualisation:

- tSNE dimensionality reduction helps the visualisations meet Tufte's first principle *"show only as much information as is required"* in contrast to simple dimensionality visualisation. In the former, vast amounts of data is compressed to reveal just enough information to enable us to make useful judgements, where as in the latter we are likely showing far more data than is required, thus breaking Tufte's first rule.

- By transforming the non-visual information (numerical activation values) into two dimensional points through the tSNE algorithm, we have placed the data in a format that lends itself to classic visualisation in the x-y dimension - a scatter plot. This satisfies Tufte's third principle tapping into the human brains innate ability to understand spatial patterns.

- tSNE also explicitly follows Tufte's fourth principle of good data visualisation that, *differences in visual properties should correspond to actual difference in the data*, by in it's very aim which is to optimise a cost function that aims to preserve the actual differences in the data (here described using metrics such as the L2, or Euclidean, distance).

7 Iteration 1 - Animation

7.1 Introduction

Reducing the dimensionality of our data is in itself not enough. While it is possible to simply plot as much of the data as possible, the sheer number of tSNE plots would quickly put us back in the position of being unable to compare data due to information overload.

Instead, by looking back to Edward Tufte's principles of visualisation and observing those which we have not achieved, the solution becomes immediately obvious: animate the data.

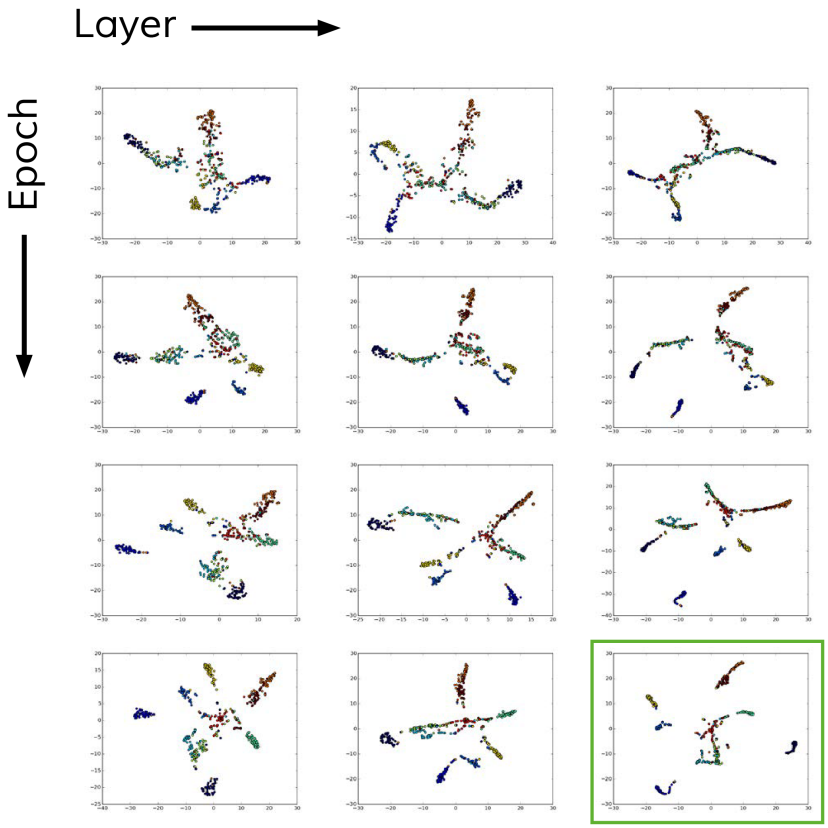


Figure 34: Already in the highlighted image it's clear to see the network is learning some distinction between the classes

Animating the data enables us to satisfy two more of Tufte's principles:

- Through the animation of tSNE plots across epochs or layers, Edward Tufte's eighth principle to *"encourage the eye to compare different pieces of data"* is now satisfied. The animation naturally encourages the eye to observe differences in the data as we see it transform from one shape to another.

- Not only does the animation enable us to compare data, but in doing so we also satisfy Tufte's seventh principle to *augment short term memory through visual patterns.* While it was possible previously to compare tSNE plots by flicking through several images - we are essentially automating this process with the animations which, as is often referenced in

visualisation theory, leaves an imprint on the retina of the previous image thus augmenting our memory with the visualisation.

While animation is definitely a great way to enhance our understanding of the neural network data by demonstrating changing patterns with our dataset of tSNE plots, it is not in itself a tool - it is simply a method of processing.

The aim of this project is to produce a tool for researchers to use to help them better understand neural networks and tweak parameters to ultimately ensure the successful training of their neural networks. To do this, animation must become part of a tool that ties together the previously discussed data collection process, and display the information in a format easily digestible by researchers.

7.2 Design

User interface components for the tool were sketched, and wire-frames evaluated, in response to the needs of neural network researchers and in relation to visualisation best practices.

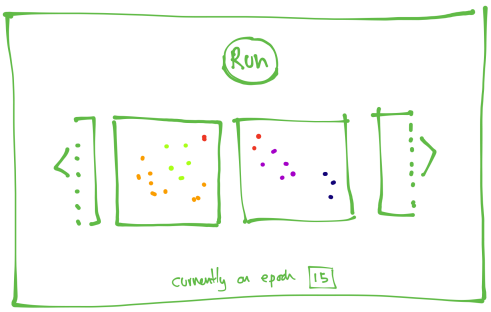


Figure 35: An initial sketch displaying animations moving across the screen as the net trains

This early sketch shows the animations running past the researcher on the screen after they have automated the testing of various models. With a number of animations sitting side-by-side the researcher can observe whether the networks improve across the parameter tweaking domain, or not.

7.3 Architecture

This early iteration was developed in a lightweight manor in keeping with the *lean product development* methodology (). This development style states that a *Minimal Viable Product (MVP)* should be built when testing ideas to enable fast testing and learning, which can then be reapplied to the product later without fear of completely rewriting the entire product.

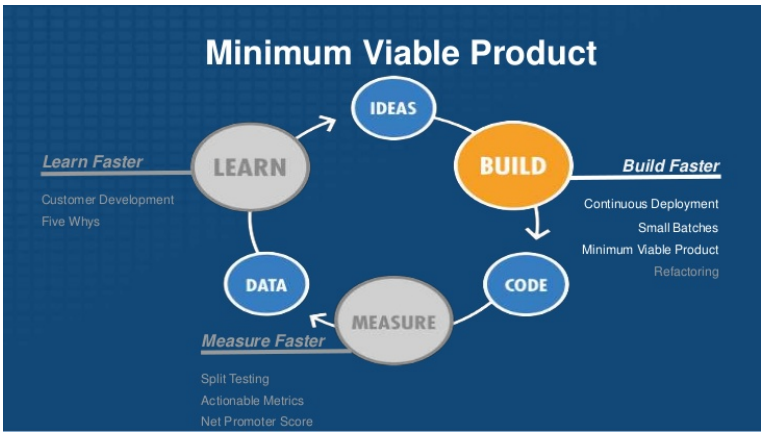


Figure 36: Lean Development Cycle

Here the MVP was a *Python* function that could be copied into the researchers neural network and would automatically run after the network had completed training. The implementation extracted the tSNE plot coordinates from the MongoDB database, processed them using the `numpy` library, into a format that could then be processed with another Python library designed for making films, `MoviePy` (), which transformed the `numpy array` of two dimensional tSNE coordinates into a chronologically ordered, by epoch, `.GIF` animation. These GIF's were stored locally and could be displayed as necessary.

TSNE-VIDEO	
filename:	string
vid_len:	int
no_points:	int
labels:	int [ ]
coords:	int [x1,y1..xn,yn]
dbClient:	DatabaseClient Object
Get_coords(filename) Reshape(coords, labels) Make_vid(coords, labels, no_points, vid_len, filename) Make_frame(time,coords,labels)	

Figure 37: Animation Processing

7.4 Evaluation

In order to effectively evaluate the success of this product, there are three perspectives that must be taken into account. The perspective of the neural network researcher and the ability of the tool in enabling them to discover more about their networks; the success of the product as assessed in accordance to Tufte's visualisation principles, and the success of the implementation in it's ability to provide the two previous goals.

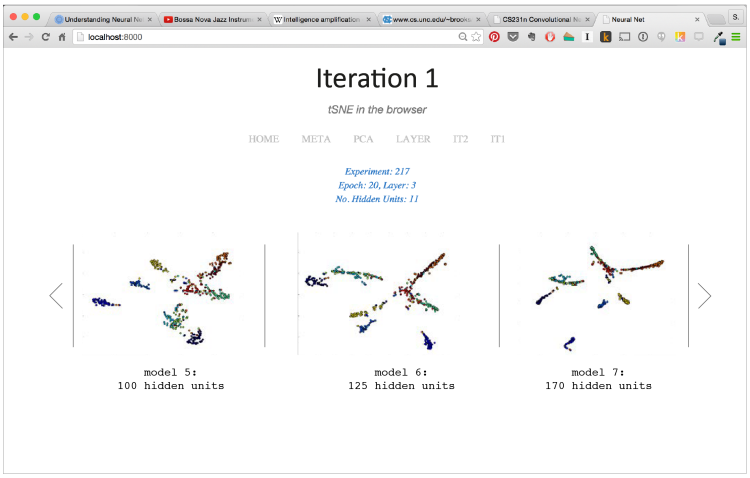


Figure 38: Screenshot iteration 1

7.4.1 Neural Network Perspective

Unfortunately it was not possible to get a large range of feedback on this iteration due to time constraints, however the little feedback that was attained from fellow students researching with neural networks suggested that it did little more than provide *"pretty"* images to look at. In particular the following reasons were mentioned.

- The first and most important consideration was the lack of ability to pause the animations to better understand stand-out abnormal tSNE plots. Now, while this was possible by looking back into the file directory of saved backup plots, this would provide too much friction for any reasonable use.
- Even if there was the ability to pause, the animated clips were not of high enough resolution, so where a researcher to stop the animation, not much could be learnt anyway.

These issues were addressed in iteration 2.

7.4.2 Visualisation Perspective

From a visualisation standpoint, as assessed in accordance with Tufte's principles, the animations were far more successful than the previous stand alone images. This is made clear in the introduction of this section.

In response for the call to interrogate the data more closely, it was decided that a key focus in iteration two should be the ability to zoom in and out of the data without loss of image quality.

7.4.3 Implementation Perspective

As mentioned previously, the implementation was a *rough and ready* solution that could enable a successful iterative process. There was however the realisation that everything should be easily accessible under one package, rather than requiring the researcher to grapple with lots of moving parts. Again, causing friction to the tool would ensure that it was never used.

These issues were addressed in iteration 2 as well.

8 Iteration 2 - Online & interactive

8.1 Introduction

In response to the points highlighted by the first iteration of the tool, the second iteration focussed on placing as much of the process online and therefore enable the much required interactive elements of the tool.

It's useful once again to to situate the project in the realm of Edward Tufte's visualisation principles so that we can assess the success of this new proposal. In addition to the principles held in iteration one, two more are added:

- Through the use of *tooltips*, an often used user interface element to provide more information about a topic upon hovering a computer mouse over the item of interest, we can satisfy Tufte's sixth principle to *visually highlight your message.*
- In addition, where previously with the animations there was no way to dig into the data without decreasing quality cause by poor pixel representations - the capability of the browser to handle *Scalable Vector Graphics, SVGs* enables the researcher to zoom in on the large data sets to observe the local structure captured by tSNE, and to zoom out observing captured global structure. This coincidently allows us to achieve another one of Tufte's principles of quality visualisation *"Reveal the data at several levels of detail"*.

8.2 Design

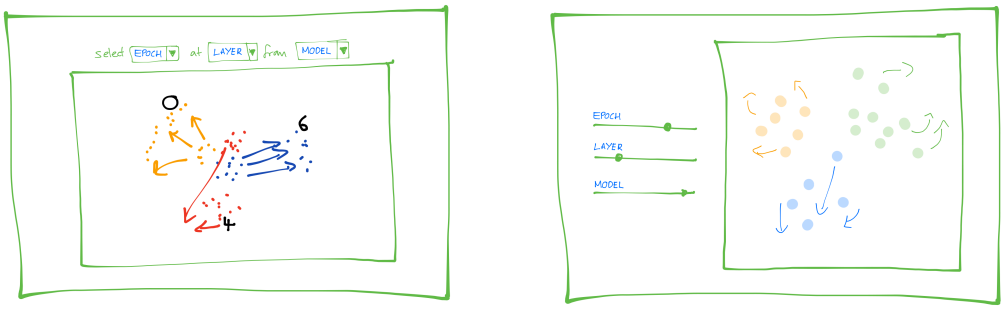
Where the previous iteration focused on inter-model differences, this iteration was influenced by Andrej Karparthy's tSNE visualisations where the model is visualised over a short duration of fitting. As the tSNE cost function is optimised, the visualisation changes - appearing as a series of *steps*. As shown below in the initial sketch designs below the aim was to show how tSNE functioned pulling apart the classes.

(a) (b) (c)

Figure 39: Screenshots of growth from iteration two

This granular view of a tSNE plot demonstrates how the the multidimensional data is transformed over the course of the cost function optimisation into a two dimensional representation displaying both local and global structures.

The two early sketches below demonstrate how the iteration would look if developed beyond the testing point reached. Here the researcher would be able to select the model, epoch and layer corresponding to the tSNE plot they wanted to interrogate.



(a) Using Angular.js Repeat with Dropdowns (b) Using Angular.js Sliders

Figure 40

8.3 Architecture

In order to bring the content online the *MEAN* development stack: Mongo (for the database), Express (for the routing), Angular (for the front-end interaction) and Node (the server), was used.

In addition, in order to produce the much called for SVG plots the *D3.js*, or *Data Driven Documents* library was used.

With the rest of the project written in JavaScript, it became clumsy to interact with the Python tSNE algorithm. For this reason `tSNE.js`, a JavaScript implementation of the tSNE algorithm produced by Andrej Karpathy, a Stanford PhD student, was used. This second implementation followed the example provided by Karpathy online in order to produce the tSNE plot that grows over time (**?**). Below is his example used with word-embeddings, where words from a vocabulary are mapped to vectors of real numbers in a low dimensional space.

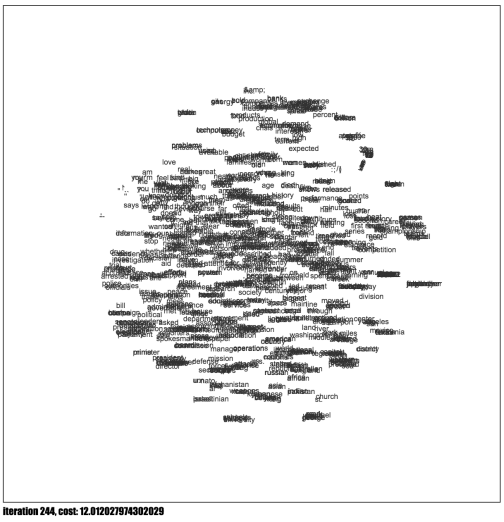


Figure 41: Andrej Karpathy Implementation

Below is the D3.js, tSNE.js class that implemented this transformation of data.

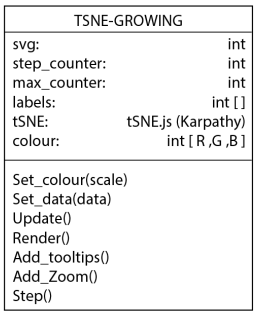


Figure 42: Visualisation

8.3.1 Node Server

Node.js was used in this iteration of the product as the backend server. It is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

In 2011, a package manager was introduced for Node.js library, called *npm*. The package manager allows publishing and sharing of open-source Node.js libraries by the community, and simplifies installation, updating and un-installation of libraries (Dahl 2009).

These features make Node.js an ideal option for developing a visualisation tool for, and were it developed further could lead to the easy development of a version for node package manager. Indeed, npm is already a common method of sharing proprietary DNN software within the deep learning community.

8.3.2 D3 Visualisation Library

D3.js, or Data Driven Documents, is a JavaScript library for producing dynamic, interactive data visualizations in web browsers.

D3 allows the binding of arbitrary data to a Document Object Model (DOM), and then apply data-driven transformations to the document. For example, you can use D3 to generate an HTML table from an array of numbers. Or, use the same data to create an interactive SVG bar chart with smooth transitions and interaction(Bostock et al. 2011).

D3 is extremely fast, even when using large datasets, making it ideal for working with the large output of the neural networks. The dynamic behaviours enabled for interaction and animation make it highly suited to the tasks in this project.

D3 uses a sophisticated method of joining data with the DOM. With three simple commands (Enter, Update, Exit) D3 enables the programmer to explain a relationship between data and the scalable vector graphic. For example, one might declare that circle elements should correspond to data, such as in a scatter plot. This contrasts with the more sequential process commonly used where one might create circles, collect all the circles and then finally assign each data point to a circle.

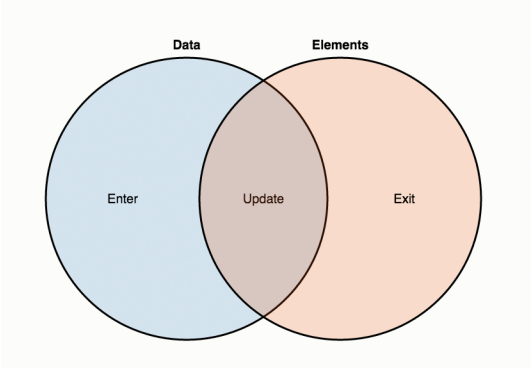


Figure 43: D3 Data Binding

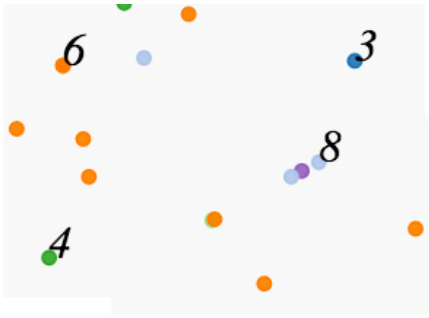
Data points, such as the coordinates in a tSNE plot, that are joined to existing circles produce the update selection. While unbound data (data for which there are no circles) produce the enter selection. Then, any remaining unbound circles produce the exit selection, where they are often removed. The significance of this is that a scatter plot can be created with not much more code that the following code:

```
var circle = svg.selectAll("circle")
    .data(data);

circle.exit().remove();

circle.enter().append("circle")
    .attr("r", 2.5);

circle
    .attr("cx", function(d) { return d.x; })
    .attr("cy", function(d) { return d.y; });
```

Figure 44: D3 managing data responsively

The simplicity of the D3 library is what makes it so powerful, and it was chosen in this project for that reason. Ideally with an implementation started in d3.js, other researchers can build upon the software with relative ease to continue to create extensions to this tool.

The following images show the tooltips used in the second iteration of the project and the relative ease at which they can be encoded.



(a) Text-tooltips

```
// Add tooltips
tooltip = g.append("text")
  .attr("dx", 0)
  .attr("dy", 0)
  .style("position", "absolute")
  .style("visibility", "hidden")
  .attr("text-anchor", "right")
  .style("font-size", "17px")
  .text(function(d) {
      return d.label;
  });
```

(b) Easily adaptable Tooltip d3.js Code

Figure 45

8.4 Evaluation

Below is a screen shot of the complete second iteration once the cost function has been optimised and the image has been focused upon. It demonstrate the high-fidelity of the d3.js SVG graphic, and shows a marked improvement upon the low fidelity pixelated animations seen in iteration one.

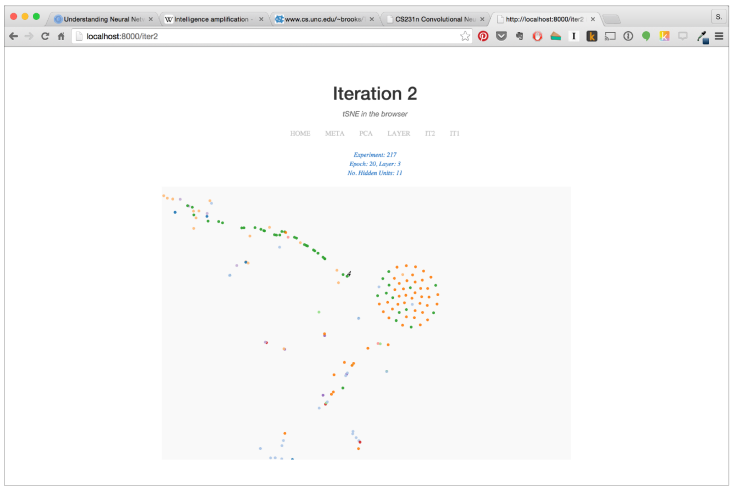


Figure 46: Zoomed in screenshot

8.4.1 Neural Network Perspective

It was widely agreed that this implementation was far superior to the GIF animations. The ability to zoom and interrogate data at different scales was welcomed, and the retention of quality in doing so was also a marked improvement.

Also notably the simple use of tooltips to display the actual value of the data, rather than just using colour, was a great addition as it allowed a researcher to first zoom in on some unlikely data samples and then see which exact data samples were causing the problem.

While the product was deemed to be a marked improvement on the previous iteration, there were still an number of problems to be addressed:

The version was criticised for taking too long to process each tSNE plot. This makes it hard for researchers to flick between layers or epochs in order to start identifying patterns - and violates Edwards Tufte's seventh principle that visualisation should augment short term memory through visual patterns. Here, the patterns emerged too slowly thus providing an ineffective means of comparison. The slow result is likely due to the slow performance of the client side optimising of the tSNE function used.

While the tooltips provided a useful way of understanding which points corresponded to which output classification in the range of one to nine, they were ineffective in demonstrating exactly which input values were causing this error. This is something that was then addressed in version three.

8.4.2 Visualisation Perspective

While from the perspective of a neural network researcher there are a number of visualisation improvements that need to be addressed, there were significant visualisation discoveries made through the use of the *data driven documents* library.

Most notably were the use of *D3.js* transitions to smooth over the difference between each step in the iterative refinement of the tSNE plot. These transitions are often used for added effect or embellishment, however in this instance they provide an important functional use - allowing the eye to easily follow specific points trajectories in space. This is useful in allowing the researcher to observe anomalies or peculiar changes in the data over time.

The introduction of transitions here is significant, and will be used in all future versions to enable easier pattern spotting within the changing data sets, be this between model, epoch or layer.

9 Iteration 3 - Epochs & Layers

9.1 Introduction

Where the previous iteration focussed on the tSNE optimisation of a single layer of activations output from a specific model, epoch and layer, the new version gives control from a model-level, over quickly switching between epoch and layer. First the researcher selects a model, then on the visualisation page uses a control-box with layer along the x-axis, and epoch across the y-axis to preview the tSNE plot corresponding to the layer-epoch selected.

Within the display box, the new tSNE plots are rendered using d3.js, retaining the ability to zoom in and out upon the data and interrogate individual points. In addition, the transitions are now being used to show how individual data-points are moving between layers and epochs rather than simply across the tSNE optimisation process.

This allows researchers to answer a number of questions very rapidly. For example, observing changes across layers may demonstrate if the back-propagation algorithm is successfully penetrating all layers and producing different representations. Observing changes across epochs demonstrates whether these layers are learning over time, or if they are simply arbitrary mapping the data, translating the data in ways that doesn't learn anything new.

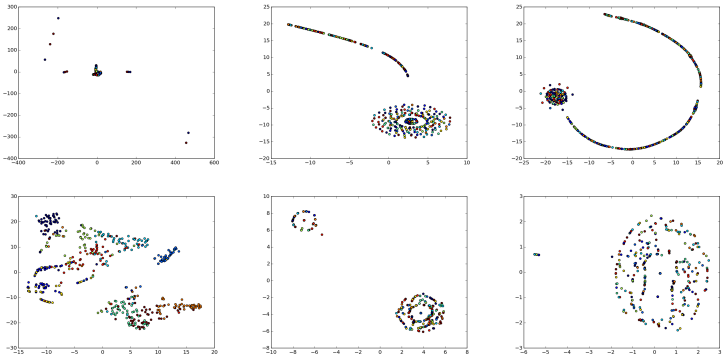


Figure 47: Epoch 2, Layer 1 from different models

In addition to the text tooltips created in the previous iteration that give information about the output classification, image tooltips have been added to map the data points exactly to the input data - the exact input image that this data point represents.

This is particularly useful in allowing researchers to make sense of commonly misclassified data points, such as sevens with ones, or eights with threes.

This new iteration therefore provides a substantial improvement upon the last.

9.2 Design

There were two important design elements within this iteration: the control scatter plot that the researcher uses to select the corresponding tSNE plot, and the display scatter plot, or tSNE plot, that is the item under consideration that the researcher can interrogate by zooming in to observe local structure, zooming out to observe global structure, hovering over points to see which input they correspond to.

A number of different methods were tried for the control unit, including using Angular.js sliders, check-boxes and drop down menus. However these all failed to unify the transition that was occurring, by which I mean they failed to highlight where exactly a layer-epoch combination sits with respect to any other layer-epoch combination - something that the user interface element decided upon did very well, a simple scatter plot.

The scatter plot enabled the product to have consistency in it's visual form, but also made it easy for users to simply hover over the points and observe as the display changed. This rapid loading of plots allows researchers to see at any given point how the networks was transforming it's data and indeed if it was learning anything new.

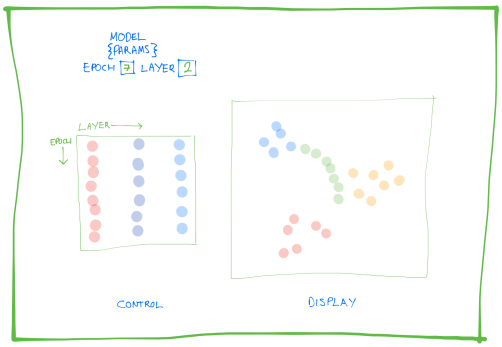
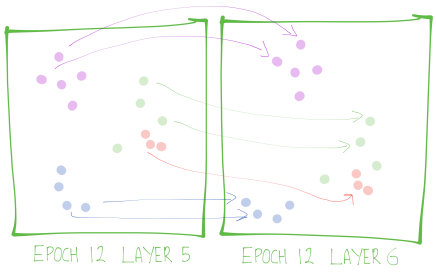


Figure 48: Wireframe Design

The new use of the D3.js transition was from one representation of a data point (or datum) to another via a number of rendered transition states. This allows the human eye to follow how specific points move with ease, and enables us to fulfil more of Tufte's guiding visualisation principles to *augment short-term memory loss with visual patterns* - here the transitions literally allow us to follow specific data points as they move across the screen.

```
d3.selectAll("circle")
  .transition()
  .delay(function(d, i) {
    return i * 100;
  })
  .duration(3000)
  .attr("cy", "50%")
```

(a) Barnes Hut SNE after applying to the imagified scatterplots



(b) Barnes Hut SNE after applying Principle Components Analysis

Figure 49

There is another important element to notice. Where the initial iteration used essentially static images, here when the image is no longer static and can be manipulated in various ways the axis are no longer relevant.

It's possible to argue that axis were never relevant due to the lack of units, but they did represent differing degrees of spatial distribution. Here we remove the axis entirely, however it's

still important the all of the images are presented in the same domain across the data so that patterns can be more easily spotted.

D3.js has yet another very useful tool `d3.scale.linear` which essentially normalises the data within the visual range on the screen. Mapping the input domain to the output range, and thus fitting the data so the user doesn't have to worry about distortion.

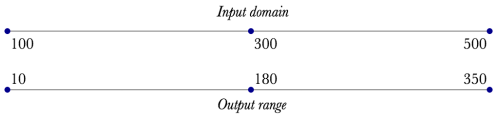


Figure 50: d3 scaling elements

9.3 Architecture

9.3.1 Entirely Python

Iteration one was implemented in a lightweight way with relatively little RESTful API use. Iteration two performed the interaction using a Node.js server to keep the entire toolkit in JavaScript.

In iteration three however, the node.js and express.js back-end components were replaced with Python's *Flask*, a *Sinatra-like* micro web application framework written entirely in Python. The framework is used by companies such as Pinterest and LinkedIn and serves as a far more appropriate product for the back-end.

The entire architecture has now been streamlined with a Python only back end; from the neural network, to the database, to the RESTful server and post-processing functionality. The entire front end is JavaScript using Angular.js for basic manipulation of the DOM and d3.js for the interactive visualisations. They speak to each other through a very basic RESTful API.

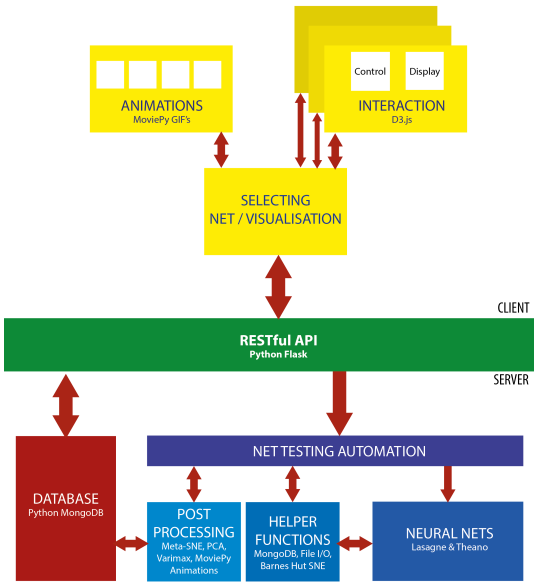


Figure 51: The architecture behind the whole system

9.3.2 Streamlined Database

In addition to the restructuring of the architecture to have a clearer divide between back-end Python and front-end JavaScript - the Mongo database is now interacted with using a python toolkit `PyMongo`.

This opportunity allowed a streamlining of the database structure as well, from the model represented in the ER schema at the beginning of the report to the following, perhaps slightly clumsier, however far easier to interact with JSON storage format - stored in Mongo's *BSON* format.

This new database schema stores the most important parameters for understanding the neural network, as well as the key data required for the visualisations.

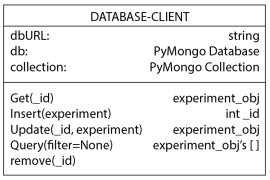


Figure 52: PyMongo Interaction


```
{
  "_ID": 010-1231-23423-4234-2,
  "HUMAN_NAME": "19-AUG-15, EX.2",
  "PARAMS": {
    "BATCH_SIZE": 600,
    "DATA_FILENAME": "mnist.pkl.gz",
    "EPOCH": 30,
    "INPUT_DIM": 784,
    "LEARNING_RATE": 0.01,
    "MOMENTUM": 0.9,
    "NUM_EPOCHS": 30,
    "NUM_HIDDEN_UNITS": 129,
    "OUTPUT_DIM": 10,
    "TRAIN_LOSS": 0.24068426056715722,
    "VALID_ACCURACY": 96.4375,
    "VALID_LOSS": 0.1277871899065497
  },
  "DATA": {
    "LAYER": [3, 1, 5],
    "EPOCH": [20,50,10],
    "TSNE_DATA": [
        [2,1,4,2,5,3,1,14,40,3],
        [7,1,10,2,14,4,20,23,40,41],
        [26,1,90,2,14,2,24,10,90,10]
    ],
    "TSNE_LABELS": [1,5,3,2,3],
    "IMG_LABELS": "AAAAAAEA9AABuPwAAfj8AAH4/AAB+PwAAfj8AAH4/"
  }
}
```


Figure 53: Simplified JSON scheme for PyMongo

9.3.3 Tooltip Adaptation

Another small change made, was to highlight the text tooltips by adding a fill to the background enabling the text to pop out - where previously the numbers where occasionally obscured.

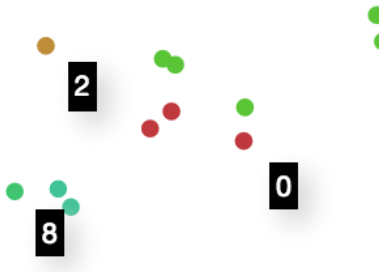


Figure 54: Tooltips

However a more major change was the addition of image-tooltips. Initially this involved trying a number of options uploading the `.png` images of the MNIST dataset to be rendered each time the researcher hovered over a point. This however was slow and contrasted to the fast approach of the d3.js application.

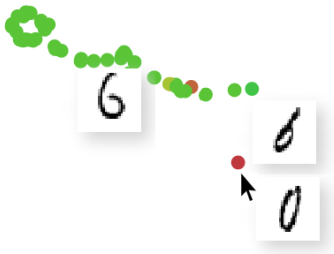


Figure 55: Tooltips

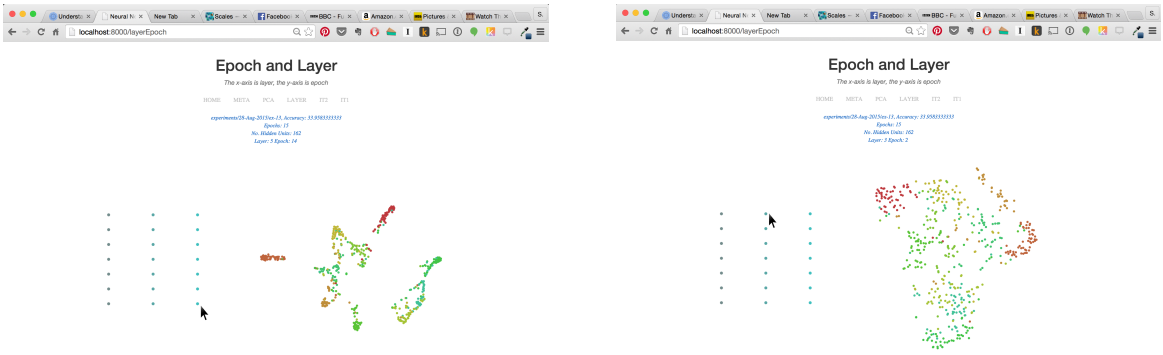
After some unsatisfactory attempts and exploration of other methods, the method used in the Chris Olah blog post (**?**) was discovered. It took a substantial amount of time to work out how the implementation worked, however it simply used the same `base64` arrays that had been used previously to store the MNIST dataset in the Mongo database.

Olahs method used *HTML5 Canvas* to render the points in the browser after uploading the entire dataset in one long *base64* array. The length of the input (784 pixels) was used to index the array and render it live using a combination of d3.js and HTML5 canvas.

Initially Olahs code was hacked onto the visualisation code from the previous iteration, however this was a poor implementation and prone to breaking. So fortunately Olah had also implemented a version of the scatter plot, all be it in a rather complex way, so it was decided to use the majority of his code governing the scatter plot and the tooltips, and build upon that to create the interface that was designed here. This saved time in contrast to the approach that would require a *reinvention of the wheel*.

9.4 Evaluation

Below are two final screen shots from this third iteration of the product.



(a) (b)

Figure 56

9.4.1 Neural Network Response

There are several advantages of this method in comparison to the previous. Where the previous helped researchers understand how the data is clustered using the tSNE algorithm, this new method is a vast improvement in analysing the data for the purpose of understanding what these networks are doing.

The ability for researchers to quickly compare changes across epoch allows them to identify data-points that are constantly failing to be categorised correctly.

After observing however that several of the tSNE plots appeared to simply be rotations of one another it was questioned whether researchers would accurately be able to diagnose which changes were the neural network actually learning, and which were simply transformations of the space in which no benefit actually occurs. For example, a network in epoch four could have a plot that appears to face north, then at epoch eight, south, and again at epoch twenty four, north. Here there is a full rotation of the data where nothing new is being learnt, however due to the implementation of scrolling by epoch and layer it's unlikely that the researcher would be able to pick this up.

For this reason, any future implementations should be able to quickly distinguish between true transformations, and transformations that are simply alterations in the Euclidean space - such as rotations and reflections.

Enabling researchers to process this information by epoch and layer however has clear advantages:

A researcher may compare an earlier epoch with a later one. If the loss values had started to platau, it would be expected that the network wasn't learning much. However with the ability to look into the data, the researcher might observe that the network is actually trying to classify one hard example, and in a few more hundred epoch could learn to classify this example that doesn't contribute much to the error but which is actually a very important example to classify. Consider Google's own algorithms that left them classifying people with Black skin as monkeys - a very sever mistake that could be identified as a single point that they will wait to be classified correctly.

A researcher may also look by layer to check that activations are indeed propagating from one layer to the next, and serving as a proxy to see that the network is actually adjusting the

weights later in the network. It's a common, and sometimes hard to diagnose problem, if weights are not changing sufficiently due to poor network initialisation or some other complication.

9.4.2 Visualisation Response

The key visual changes here were the re-implementation of transitions between layers and epochs enabling the researcher to observe patterns more accurately, and in addition ensuring that Edward Tufte's guide was still being followed.

The other major change was the introduction of the image tooltip. Previously the text-tooltip enabled the researcher to view the classification result to the accuracy of the output layer, however here each data point directly corresponds to one of the input values, each of which is visually rendered in the browser. This adaptation of Chris Olahs blog post provides a much needed extra dimension of exploration within the project.

9.4.3 Implementation Response

There were a number of significant implementation changes in iteration three; a more streamlined database, a properly separated Python back-end and JavaScript front-end, and the introduction of an adapted version of Chris Olahs (Olah 2014*a*) visualisation scatter plot and image tool tips.

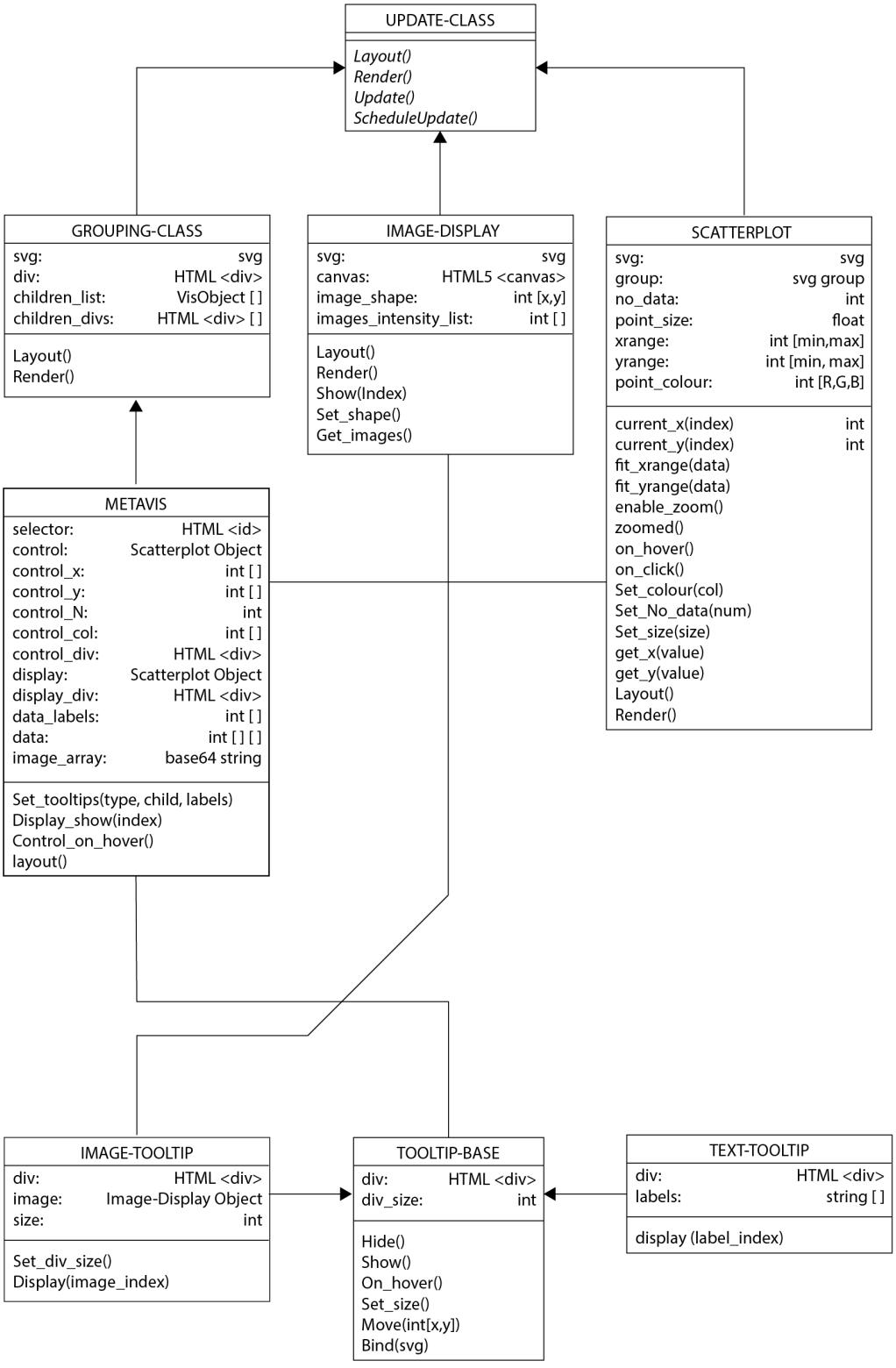


Figure 57: Adapted & Extended Olah D3.js interaction

The streamlined database ensures that researchers can easily interact with the product. The distinctive front and back-ends ensures that they can be used independently of one another. And the adapted and improved visualisation infrastructure ensures that it is easier for researchers to interact with.

10 Iteration 4: metaSNE & Principle Component Analysis

10.1 Introduction

Iteration four addresses the problem discovered through iteration three of rotational data. The main focus therefore of this section is an exploration into methods for mapping that rotational data to a space that enables researchers to better disipher it.

In particular this section looks at principle component analysis, PCA, and Varimax rotation.

10.1.1 Isometries in Representations

Many datasets have rotational duplicates appearing across them, and in data science & mathematics this is a common problem. Where a representation contains a transformations in the *Euclidean* space, these transformations are often called *Isometries* and are transformations such as rotation or flipping.

In the tSNE plots that visualise the representations being formed by the neural networks it's important to distinguish which plots demonstrate new pieces of information being learnt, and which simply appear to be learning however are just translations across the Euclidean space.

In his blog post on *Representations* (Olah 2014*d*) explores this notion of isometries stating that for any representation X there is an associated metric function, d_x, which gives us the distance between pairs of points within that representation. For another representation Y, $d_x = d_y$ if and only if X is isometric to Y. This is exactly the form required that can remove isometric data.

This transformation into metric space can be incredibly complex, so the approach taken instead approximates the method described by Olah, by using *Principle Component Analysis* and *Varimax Rotation* to subsequently orthogonalise the PCA mappings and create a dataset that has isometries which can be easily classified.

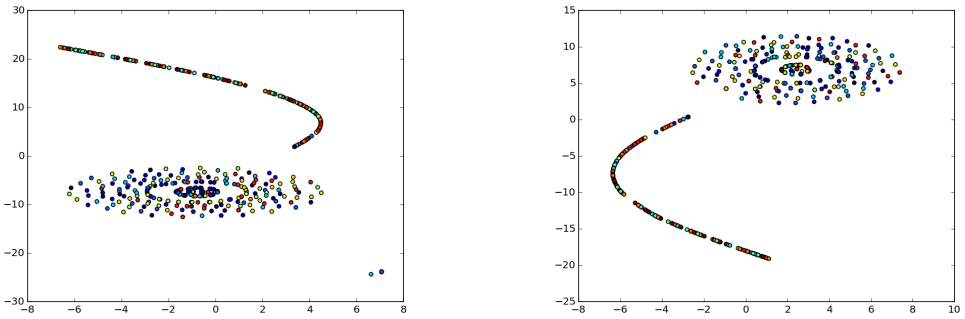


Figure 58: This experiment demonstrates two layers where the data has essentially just been flipped - demonstrating nothing particularly new has been learnt

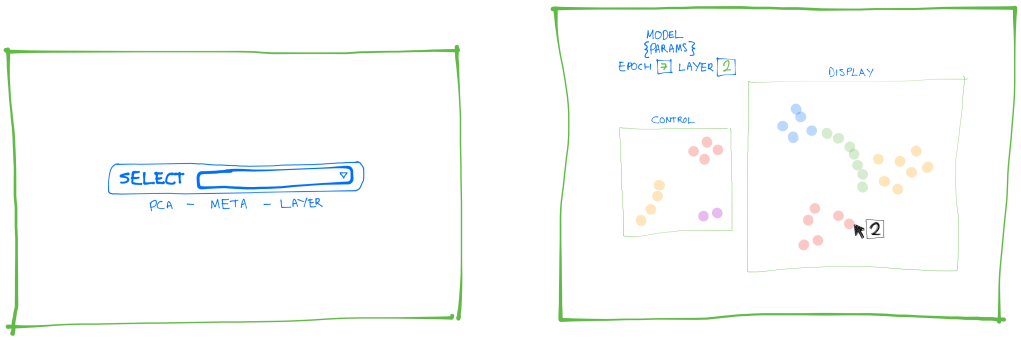
10.2 Design

The fourth iteration of this project is where the development ends. There are still improvements that can be made which can be made, and will be discussed in the concluding section, however by this point there is certainly a tool that can be utilised to make important decisions.

There are five key components to this final tool:

- First, the neural networks themselves: two neural networks are provided with the tool, a convolutional network and a feed forward network as described in chapter on data collection. These two classes are easily adaptable under the *Lasagne* guidelines.
- Second, the saving functionality afforded by *PyMongo* and the *MongoDB* database that saves key data in an easily accessible format to a local datastore, and all processing functions that allow this to integrate seamlessly with the *Lasagne* neural networks.
- Third, the post-processing functionality that provides the PCA-Varimax / meta-SNE plotting explained in this chapter.
- Fourth, the RESTful API providing data to the client.
- Fifth, the front end *Angular.js* and *D3.js* application consisting of an easy to use homepage where the researcher can select the model they want to explore, and the visualisation pages where the researcher can interrogate their data.

Below are two sketch diagrams that demonstrate the simplicity of the front end.



(a) Homepage (b) PCA interaction

Figure 59

10.3 Architecture

The final architecture has a number of components as were described in the previous design section; however the addition of the Principle Component Analysis and Varimax Rotation post-processing is explored below.

10.3.1 PCA-Varimax Post-Processing

In order to determine the success at which isometric data was removed by the post-processing function, a simplified dataset was created and functions tested upon.

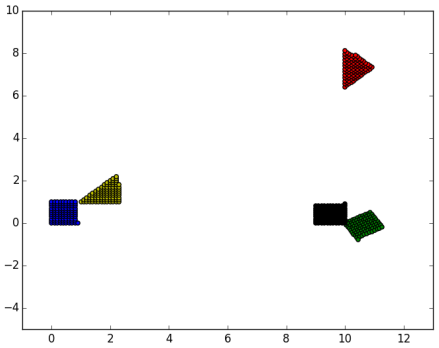


Figure 60: The test dataset

The test set, above, was a number of uniformly generated x and y points that when plotted produced a dataset that consists of two identical triangles that had been rotated by differing amounts, and three identical squares that had likewise been rotated by differing degrees with differing offsets. This creates an artificial dataset that makes it easy to determine the success of the post-processing functions.

The three different approaches were used in an attempt to generate an effective *metaSNE* plot, and the following methods tested were as follows:

Image Compression

The first method took influence from the tSNE plots created of the MNIST dataset where each of the 28 pixel by 28 pixel images had their dimensions collapsed into a single 784 pixel array before the tSNE algorithm is implemented upon it.

With the scatter plots, in order to do this, the (x, y) coordinates needed to first be transformed into an image with discrete dimensions. For example, a 500 pixel by 500 pixel image.

There were a number of different methods that could have been employed in order to do this, however the method adopted was to create a two dimensional histogram where, should a point to appear within a given coordinate bin, then that bin would be assigned the average of all points situated within it.

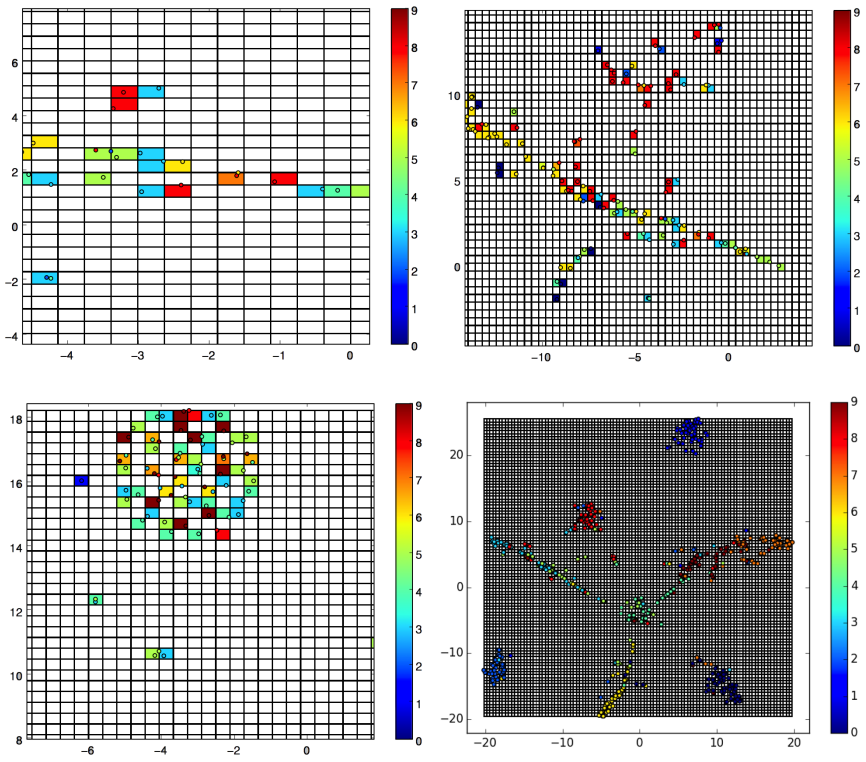


Figure 61: Imagifying Scatterplots

As can be seen from the figure above this works for the most part due to the relatively high-fidelity of the selected dimensions. However, where two or three points are situated very close together such that they enter the same bin, the transformation taking an average distorts the dataset somewhat representatively.

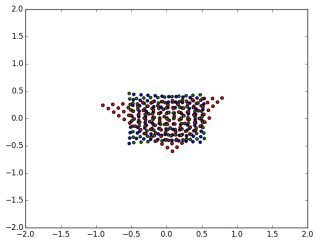
Principle Component Analysis

The second method attempted was to first apply Principle Component Analysis, PCA, to the scatter plot in an attempt to remove the majority of the rotational differences, and mean-centre the data.

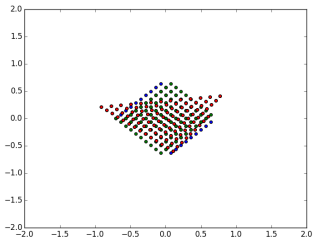
PCA is a statistical procedure that converts the set of points along a number of principle components. The largest principle component has the largest possible variability in the data, and each succeeding component has the highest variance possible under the constraint of those preceding it.

This mapping of plots along it's principle components should remove the majority of rotational data held within the plots, and clearly works as is demonstrated in the plot below on the left.

This second method was far more successful than the first in accurately distinguishing the differences and similarities in the original dataset once plotted, however upon application of tSNE the points were poorly classified.



(a) Application of Principle Component Analysis



(b) Application of Varimax Orthogonalisation

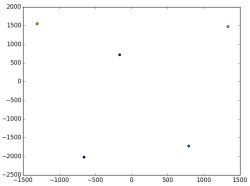
Figure 62

PCA & Varimax Rotation

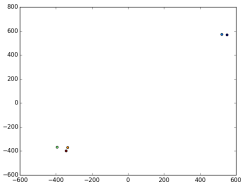
After the unsatisfactory results produced by the PCA process, it was decided that the datasets needed to be simpler for the tSNE algorithm to map effectively.

After much searching, a solution presented itself in the form of *Varimax Rotation* (**?**) which orthogonalises the data such that the actual coordinate system is unchanged, but the orthogonal basis that is being rotated is aligned with those coordinates. This produced the plot on the right which aligned the test set perfectly into it's original triangles and squares being centred directly on top of one another - thus removing the isometric data we wanted to remove.

The plot below demonstrates on the left the Barnes Hut SNE output when applied simply to the PCA rotated data, and the plot on the right after the data has occurred the Varimax Rotation as well - the former fails to accurately represent any of the data as similar, while the latter does exactly as desired - separating the squares from the triangles, and demonstrating that they are simply slight variations of one another.



(a) Barnes Hut SNE after applying Principle Components Analysis



(b) Barnes Hut SNE after applying Principle Components Analysis, and Varimax Orthogonalisation

Figure 63

It is this final version: PCA and Varimax Rotation that is used in the final tool to enable researchers to get a better grasp upon how similar these scatter plot really are.

PLOT-PLOTTER	
dbClient:	DatabaseClient
castPCA(np_array)	cast_array
castTSNE(np_array)	cast_array
Varimax(np_array, gamma,q,tol)	orthogonalised_array
meta_pca(experiment_id)	
meta_tsne(experiment_id)	

Figure 64: Post-Processing

10.3.2 A tool for visualising neural networks

The final architecture for the project remains very similar to that of the previous iteration.

The diagram below explains the interconnection between the various parts:

- **Client Side**

  Three components:

  - The network experiment selection, and visualisation selection
  - The browse by animation section
  - The interactive section where the researcher can explore by Epoch, Layer, PCA/Varimax centred, or pure meta-SNE (tSNE applied directly to tSNE).

- **RESTful API**

  The Flask Restful API enables the information on the server side to remain hidden until needed. It also ensures the site can load quickly without having to upload vast amounts of neural network data.

- **Server Side** The server side has two primary components: the database and the neural network processing elements. The neural network processing however is split into a further three parts:

  - The Neural Networks themselves which can be called from the network automation section.
  - The helper functions which facilitate the extraction of data from the neural network and store it appropriately
  - The post-processing unit which extracts from the database all stored tSNE plots, and processes them to produce a meta-SNE plot and PCA/Varimax plots.

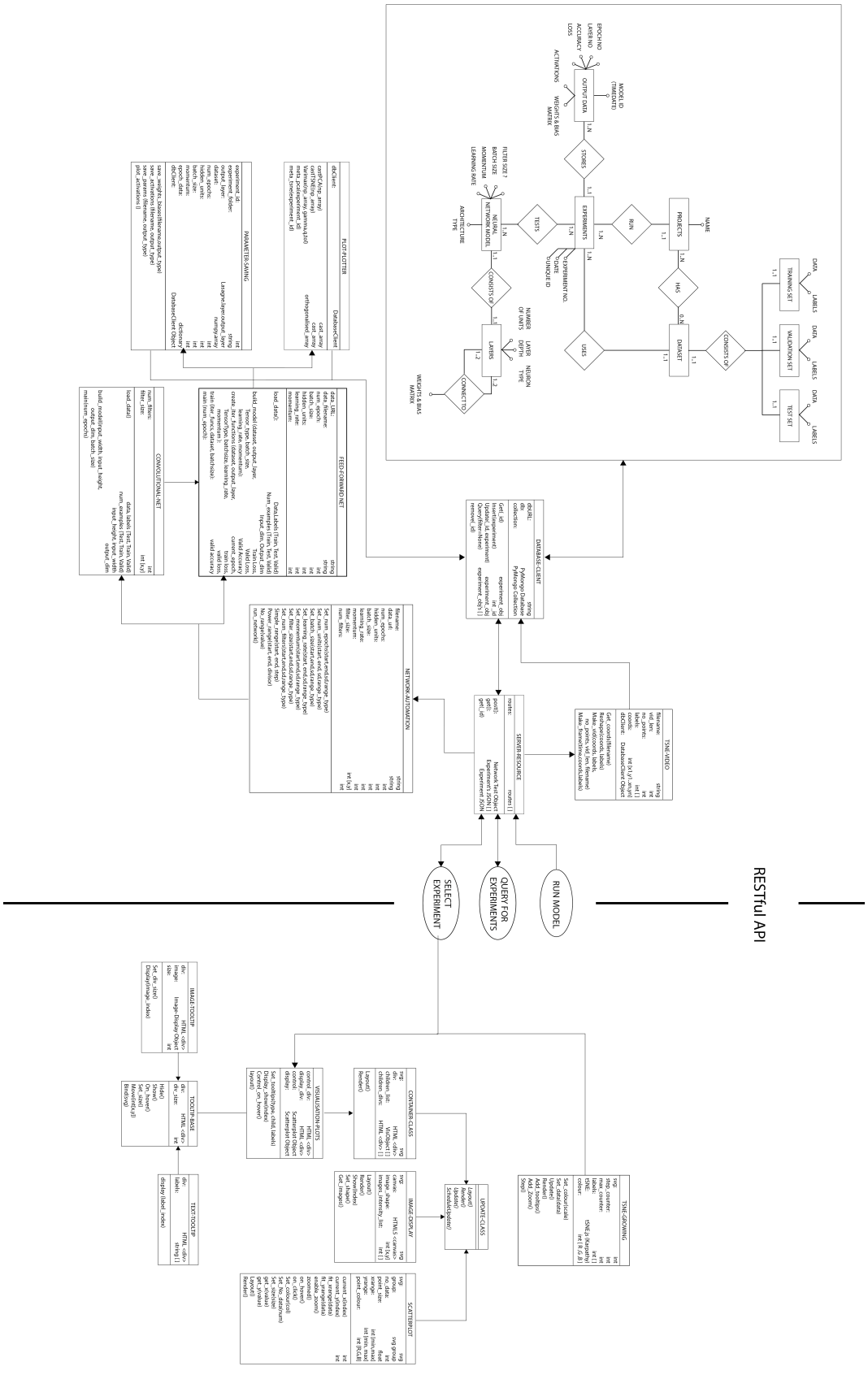
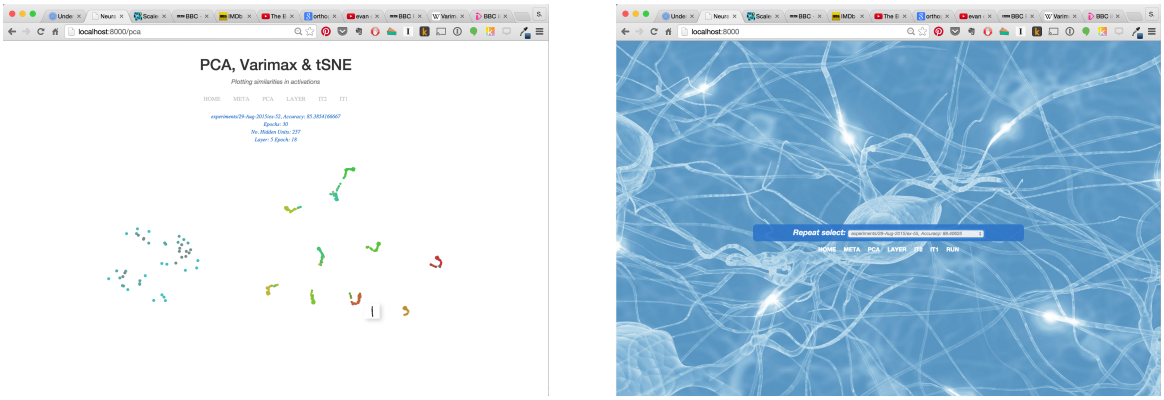


Figure 65: UML, ER, Scheme diagram hybrid

10.4 Evaluation

These two final screen shots demonstrate the control cluster side by side with the display cluster. The control has been created using the PCA-Varimax post processing such that points next to one another are similar or may be rotations of one another. The display shows the tSNE data once it has been transformed along the principle components and orthogonalised with Varimax.

The homepage is a simple place where the researcher can select a model from the database, which will then be passed up to the client side application and the researcher can browse by either: Epoch & Layer tSNE, PCA & Varimax tSNE, or, tSNE tSNE (meta-SNE).



(a) Final PCA exploration (b) Final Homepage

Figure 66

10.4.1 Neural Network Perspective

The response to this final iteration, that effectively ties together the lessons from the previous three, was overwhelmingly positive.

The tool enabled researchers to spot patterns emerging within datasets as networks trained. When using all of the elements together: the epoch-layer control as well as the PCA-varimax control to observe, zoom in on to get a more local view, or zoom out to get a more global view of the patterns emerging, and of course transitioning from one plot to another while following particular points - observing anomalies and rotational patterns in the tSNE mapped activation data.

Perhaps most interesting is the peculiar shapes that the data tends to be morphed into, however this will be discussed in the conclusion.

10.4.2 Visualisation Response

From a visualisation perspective this final tool really fulfils all of Edward Tufte's visualisation principles to a greater or lesser degree, as well as providing a useful tool for *Active Vision problem solving* through the use of sight and our natural abilities to spot patterns in spatially described data.

Assessing the tool against Tufte's principles:

- Principle 1, *show only as much information as required*, is fulfilled comprehensively through the use of the tSNE algorithm which compresses the multi-dimensional data down into the

2D plane.

- Principle 2, *include visual difference only when required*, is fulfilled by the PCA-varimax algorithms such that visual differences are not simply 'shown when required', but are intelligently organised to enable easier comparison.

- Principle 3, *use visual encodings for quantitative values*, in first transforming the multidimensional data down into two dimensions and then using the scatter plot diagram, which even without axis, uses the visual encoding of *spatial separation* to encode the differences in quantitative values.

- Principle 4, *differences in visual properties should correspond to actual differences in the data*, is explicitly implemented with the tSNE algorithm which optimises a cost function that aims to preserve the qualities in the data when mapping down to a number of dimensions that can be easily visualised.

- Principle 5, *do not connect values that are discrete*, is realised simply by not making any connections at all, and allowing the researcher to spot these connections of their own accord, helped along by the d3.js transitions.

- Principle 6, *visually highlight the most important part of your message*, occurs at two levels: firstly when the researcher scrolls over a point in the control box, the display box changes to highlight that item, and secondly as the researcher scrolls over the points in the display box, either the categorisation value is displayed as a string, or the input image is displayed as a HTML5 canvas rendering.

- Principle 7, *augment short term memory through visual patterns*, has been achieved by using transitions from one tSNE plot to another, thus supplementing the researchers short term memory of the previous plot by providing a trajectory for each point which can be followed through the transition.

- Principle 8, *Encourage the eye to compare different pieces of data*, is fulfilled by providing the control box which the researcher can scroll over to rapidly change the tSNE plot on display, thus being able to successfully compare the data.

- Principle 9, *Reveal the data at several levels of detail*, is enacted by simply enabling the researcher to zoom in and focus on the local structure within the tSNE plot, or to zoom out and focus on the global structure.

- Principle 10, *Don't distort the data*, has been fulfilled by using the d3.js linear scaling function that allows us to map data points directly to the visualisation with one constant scale without undergoing distortion. However it is clear that the data has undergone significant distortion by each of the algorithms that enable the visualisation to achieve earlier principles, and so this final principle is fulfilled to a lesser degree.

10.4.3 Implementation Response

The implementation of the final tool clearly separates the concerns of the client side and the server side. It partitions key elements in classes and folders within the file-structure, and maintains a clean database structure.

This implementation is deceptively simple and should make it easy for researcher to use the tool, or adapt it to their needs.

11 Conclusions

11.1 Final Observations

This section explores some of the observations that were made using the tool, exploring the tSNE data across both the epoch-layer control configuration as well as the PCA-varimax control configuration.

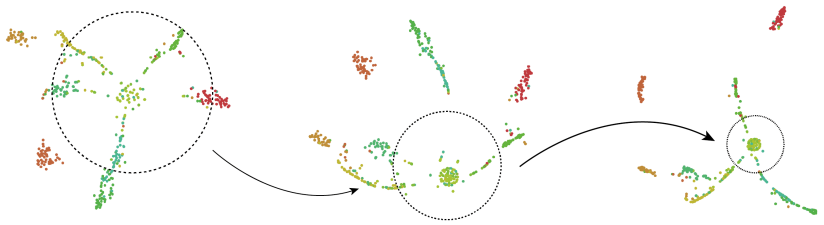


Figure 67: Exp.24, Hidden Units.7, Layers.1,3,5, Epoch.30

Layer Change

Above we can see that as the neural network gets deeper the clusters within the tSNE plots get gradually tighter. This indicates a transforming representation that has discovered a space that effectively separates the classes at an early layer, and the subsequent layers appear to be simply emphasising these early lessons.

This observation could lead the researcher to adjusting their model more substantially in the earlier layers - perhaps adding more hidden units, applying more effective drop-out to disassociate the relationships so directly between layers, avoiding what could be an early over fitting of the data.

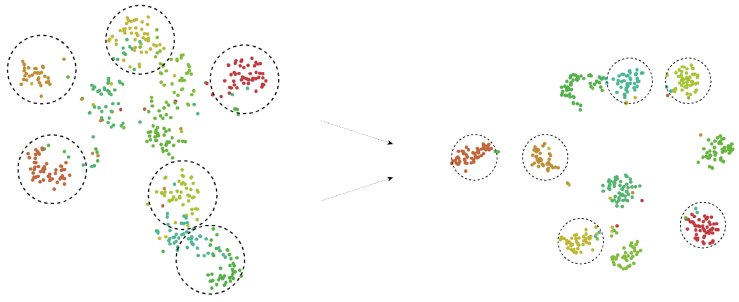


Figure 68: Exp.35, Hidden Units.512, Layer.3, Epochs.2-40

Epoch Change

Above we can see a successful clustering being developed across the epochs with readings taken across the same layer. The network has performed exactly as desired and separated the digits particularly well into different clusters.

One observation that could enable the researcher to make potential changes to the model are two clusters at the top, and two clusters at the bottom. Here the network is not creating a particularly distinct separation between the digits '7' and '9', and between the digits '3' and '5'. This could lead to the researcher including more examples of these digits in their training-set,

or in a convolutional network adjusting the filter and maxpooling parameters, as this could potentially help the network to distinguish between these admittedly similar shapes.

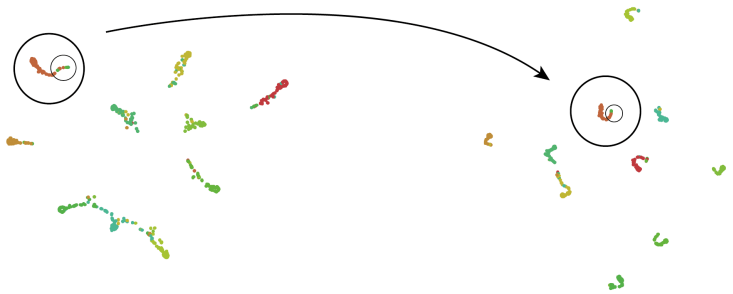


Figure 69: Exp.35, Hidden Units.512, Layer.5, Epochs.2-40

Epoch Change

In this example as the network trains the researcher begins to see a tightening of the clusters in a very distinct manor - forming little "c's". The highlighted portion is the retained misclassification of a small number of sevens within a cluster of ones .

These classification examples demonstrate why such a network often achieves very good accuracy, however not entire accuracy - the examples are indeed just very similar. In order for the researcher to improve such a network many more training examples such as this would be required - and this tool enables the researcher to discover exactly what these should be.

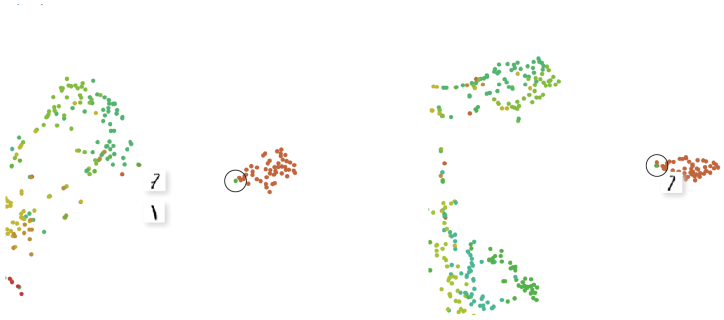


Figure 70: Exp.46, Hidden Units.16, Layers.1-3, Epochs.6-16

Layer change & Epoch change

This situation mirrors the above, where sevens are again misclassified as ones - however here the two examples are from different layers as well as distinctly separate epochs. This means that the network is really failing to learn much at all on either account.

Here the researcher would likely increase the number of hidden units in each layer.

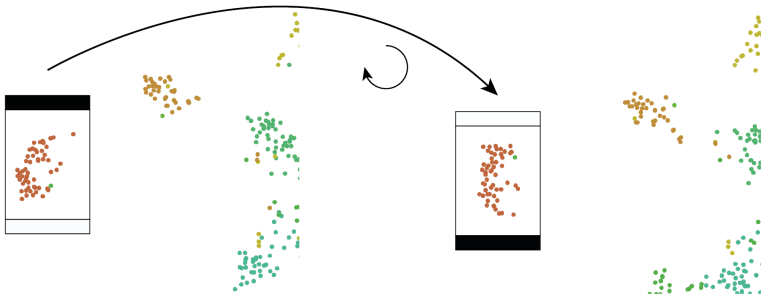


Figure 71: Exp.49, Hidden Units.64, Layers.1, Epochs.14-26

Epoch Change

Here are two very similar sets of data if we look at the patterns that emerge on the right hand side of each image. However, when we look more closely - and this is far easier to spot using the interactive tool with the transitions - then we observe the reflecting of the section highlighted almost directly along the x-axis.

The flipping of this section, and relative retention of the others could suggest that these units are proving a particular problem for the network. It could also prove that the network is happy with the global structure that it has identified within these digits - as they are primarily well classified, but not with the local structure, which it is still regularly tweaking.

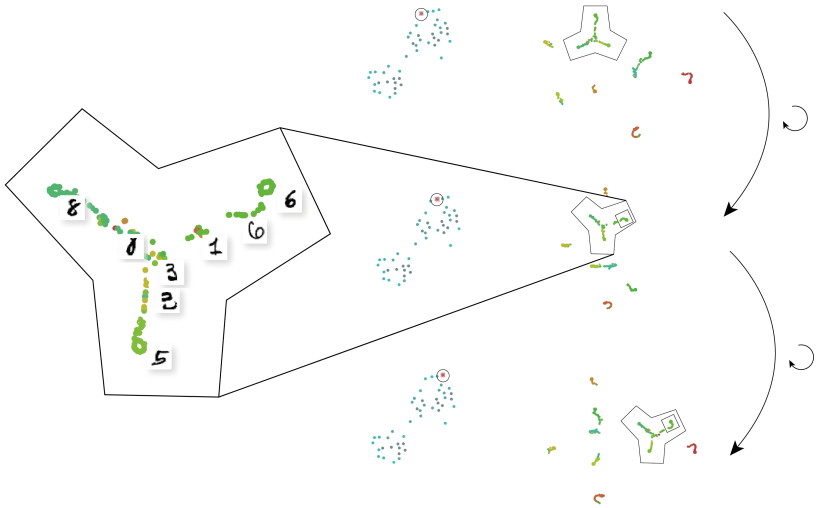


Figure 72: Exp.49, Hidden Units.64, Layers.5, Epochs.16-28

Epoch Change

In this example, the network appears to have learnt to distinctly classify digits 0,1,2,4,7 & 9 however is having difficulties with 3,5,6 & 8. Again this would tie into our intuition about these letters than one might describe as a whole are *curvy with distinct tops and bottoms.*

Firstly it's interesting that this section simply gets rotated, and not much is being learnt, but also the absence of the digits '9' from this triangular set which would in our minds also adhere to the above qualitative description given, suggesting perhaps that the network has learnt to

distinguish the upper half of the units, but not the lower half - a lesson that could perhaps encourage researchers to tweak the type of filters used in the convolutional layers to gain a more fine-grained understanding of the bottom half of the data-samples.

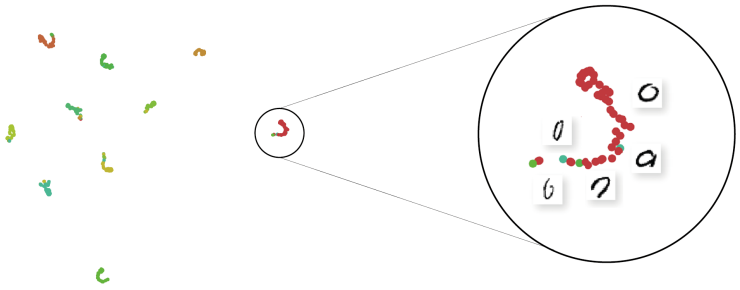


Figure 73: Exp.54, Hidden Units.649, Layers.5, Epochs.18

Zoom-in

This example demonstrates the value of the ability to zoom in upon the data to observe a local structure within the classifications, but also zoom out to observe the global structure.

The global structure identifies a network that has learnt very well to distinguish the digits, and has begun perhaps to pick up information about how the artificial dataset was configured - indicated by the localised 'c' shape that regularly appears.

The local, zoomed in, picture of the classification however demonstrates another story - that while the classifications are incredibly distinct, these distinct 'c' shapes contain some obviously misclassified points, with the digits 6,7 & 9 appearing within a cluster of 0's. This would suggest that the researcher should tweak the network such that before it started to exacerbate it's classifications, that it should focus more on getting the niche results right. Perhaps the researcher would look at adding in a layer which aimed to remove locally misclassified points.

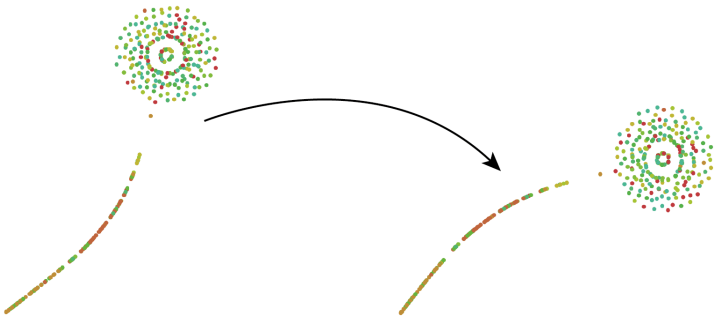


Figure 74: Exp.42, Hidden Units.2, Layers.3-5, Epoch.30

Layer change

In this example, across the layers the network is not gaining any more of a fine-grained representation of the data. It is simply producing a rotation.

This is probably expected give the relatively few numbers of neurons that exist here. The distributed representation that we understand to occur with larger networks, where each unit

may learn to encode some distinguishing feature of a dataset - such as gender in facial recognition - this much smaller network must encode all of the learnings within just two units at each layer, and it seems that this simply enables a rotation or reflection in the euclidean space. The researcher, rightly, would dismiss this as a bad network and increase the number of hidden units.

11.2 Future Work

While the project overall produced a tool that can most certainly highlight some important aspects of how neural networks train, this forms only one possible tool for probing, and many other probes exist that could be integrated or created.

This section explores a number of directions that this project could develop in the future.

11.2.1 Automatic Neural Network for Education

Currently, and in response to user feedback, the running of experiments is entirely within the control of the researcher. They decide the parameters, the architecture etcetera - the implementation is entirely within their control.

While this works for researchers, a possible alternate use of this project is as a tool to educate students about neural networks. A tool for education.

In this setting, it would be essential that all interaction was performed through the online system. A quick sketch below demonstrates that this could be achieved with relative simplicity, however admittedly a number of important parameters would be out with the control of the students here.

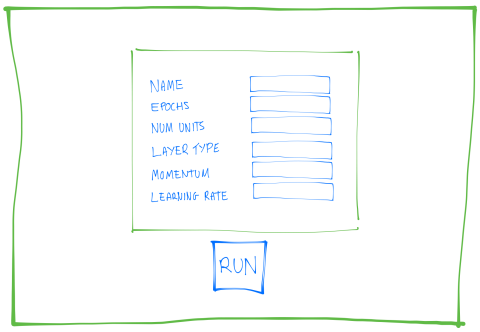


Figure 75: Wireframe Design

11.2.2 Application to other architectures

This project focussed on feed-forward neural networks and convolutional neural networks. However, the ability to interrogate the activation of networks through the tSNE, meta-SNE, epoch-layer, PCA-varimax interactions would be useful for all network architectures where the activations could successful be captured, and the inputs encoded textually or visually.

For example a fairly straightforward adaptation would be to implement the project upon recurrent neural networks, and map the recurring output and input.

11.2.3 Inter-Experiment Comparison

While in the final iteration it was possible to compare across layers and epochs, and to some extent by experiment, it would be good to make this explicit.

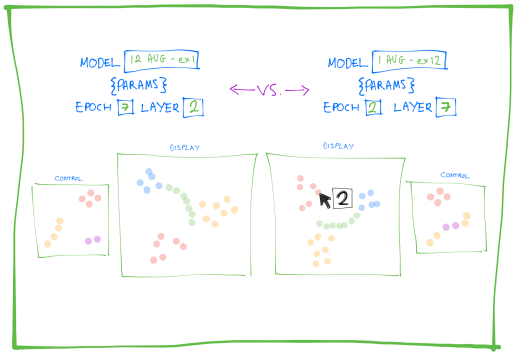


Figure 76: Wireframe Design: two models

One possibility, as shown above, is to place two experiments side by side. This means a direct comparison could be made between two experiments with ease.

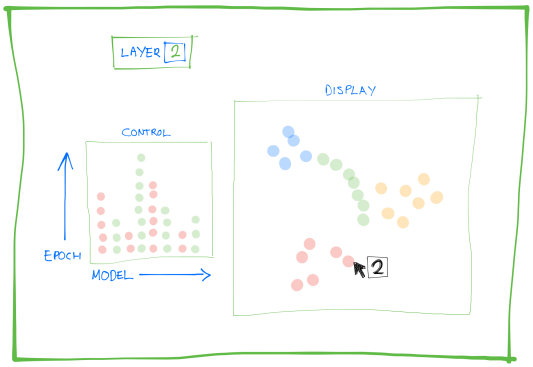


Figure 77: Wireframe Design: Epoch-Model

Another possibility would be to provide the opportunity to set the layer or epoch, and to map the resulting epochs or layers against different experiments rather than against each other.

11.2.4 Google's Inceptionism

The representations learnt at each layer of a neural network can directly correspond to learning distinct features within the training set. For example with images, the first layer might learn to identify edges and other layers overall components of images, until finally it could recognise whole objects such as a 'cat' or 'dog'.

The tool built with this project identifies patterns that have been learnt, but doesn't show us exactly how the data has been understood. Google, in cutting edge research released after the commencement of this project, unveiled project *Inceptionism* by (**?**).

This projects turns the network upside down and asks it to enhance an input image in such a way as to recreate the understanding of the image at any particular layer.

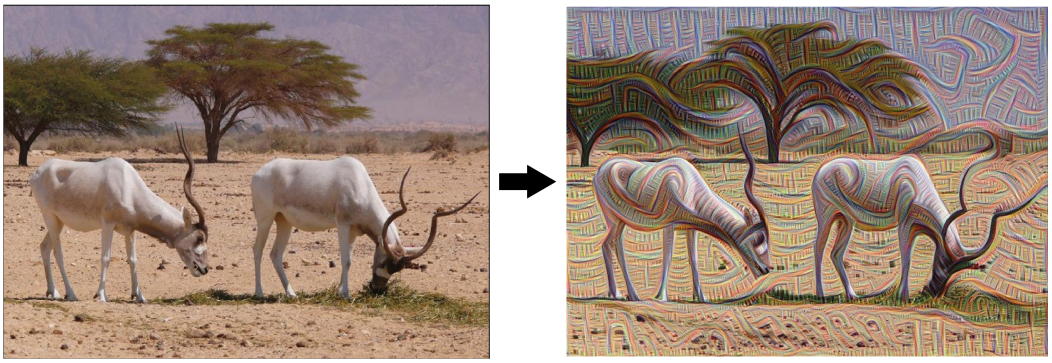


Figure 78: Google Inceptionism

The above image demonstrates a the projection of an early layer that captures edge detail back to the input image space. This can already give insights into how the network may be learning, and that tasks such as drawing boxes around particular features - trees, antelope, sky, ground - may prove to be difficult as they almost meld into one another.

Figure 79: Images that were understood to be 'lifting weights'

This image interestingly captures an understanding of lifting weights that may not be expected at first - that the weights are always correctly identified when they have a body-builders arm in the image as well. Thus identifying to the researchers that the training set probably needs to be modified to include more images of lifting weights where no arm is present.

A lot of the information needed to produce such images is already captured with this project, and a future development could be to attach the open-soured tool used to create these images to the back-end developed in this project to create an extra post-processing unit to help researchers visually understand their neural networks.

11.2.5 Architecture Mapping

As metioned in the literature review of this project, there are a number of different methods that have already been attempted to capture neural network data. One such representation that I believe showed promise was the *Tzeng* project that visualised the network architecture in a method that captured the relative influence of each unit.

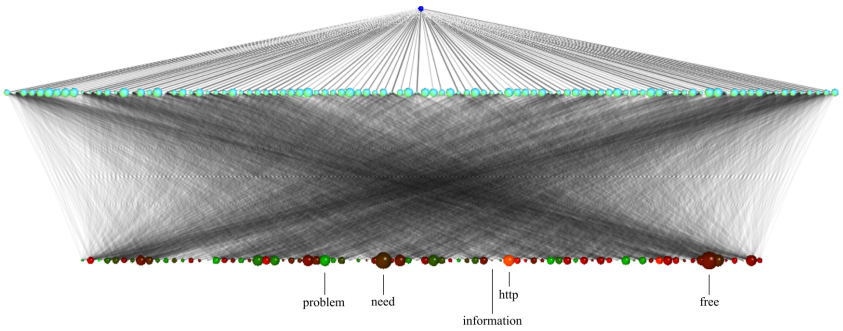


Figure 80: Tzeng Map

Another future development of this project could be to take the activations stored in the server and create a d3.js implementation that replicates the work started by Tzeng.

11.2.6 Alternate User Interfaces

While human beings are great an understanding two dimensional representation such as the scatter plots exhibited here, we live in a three dimensional world and these two dimensional representations are mostly compressions of the three dimensional data.

In order to fully appreciate the spatial representations of the tSNE plots, a further enhancement could be to map the multi-dimensional space to three dimensions instead of two. This could be displayed either online using an implementation such at *THREE's trackball controls*, or could be implemented such that it could be interacted with in virtual reality using means such as the *Occulus Rift*.

Below is a conceptual sketch of how this might appear analysing the MNIST dataset.

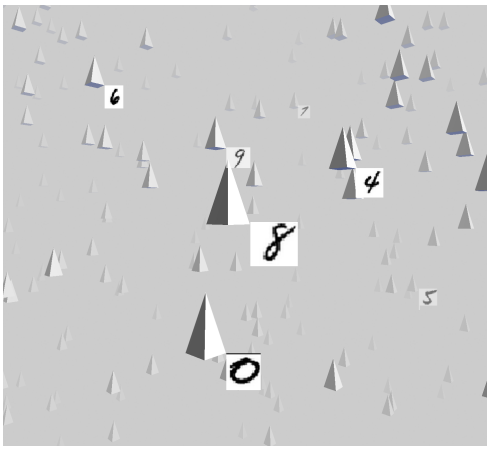


Figure 81: 3D Virtual Reality Exploration

11.2.7 Beyond Theano

A final future adaptation that could be implemented, and would really be taking this project far further in terms of its mass usability - would be to extend the project beyond the realms of Python and Theano.

While the initial research performed demonstrated a marginally bigger audience within those questioned, there are a wide array of tools and languages used when researcher neural networks.

A valuable extension to this project would be to enable the interaction with these many of other tools to extract their network outputs, store in the same database and visualise.

For an individual with knowledge of each of these other packages, this task shouldn't be tremendously challenging.

11.3 Summary

Deep Neural Networks are quickly becoming the industry standard for many complex machine learning tasks such as computer vision and speech recognition.

However unlike some other machine learning models that are widely understood, such as logistic regression techniques, no one fully understands Deep Neural Networks in their full complexity. This poses both practical and ethical problems for researchers and practitioners alike.

As these algorithms get closer to achieving general artificial intelligence, a thought spurred on by Deep Minds achievements in their Atari Games paper (**?**), there is a greater need to understand how exactly these networks work.

This thesis aimed, through the provision of a visualisation tool, to help researchers probe their neural networks while they train in order to better understand what they do, and how they represent their data.

This is a big, very current, area of research and while many methods being developed within the field are often mathematical - this thesis takes an alternative approach, one that has been proven to be successful across many other industries, using visualisation as a tool for understanding.

Having explored neural networks and visualisation theory, and having spoken to researchers about their everyday challenges, a tool began to develop centring around the tSNE dimensionality reduction algorithm.

tSNE retains important topological characteristics within datasets at both a global and local level. This makes it perfect for reducing the high dimensional data captured during a networks training to the two or three spatial dimensions understood intuitively by humans.

This report explains the development of a tool for probing the inner machinery of neural networks, looking at topics of visualisation, animation, interaction, data production, data collection, data manipulation and web development. All of which were necessary in the production of the final result.

Using an iterative development process the tool continually converged upon a set of specifications that made it easy to use, clear in its architecture, simple in its user interface and highly functional in enabling researchers to identify emerging patterns within their data that are ultimately indicative of the model characteristics that can modified through guided selection of network design parameters.

The main contribution of this project is a tool for researchers than can be used to better understand their neural networks, and the demonstration that more sophisticated visualisation techniques are not just useful for explaining neural network research, but can be used to better understand the research undertaken by academics while it is performed.

References

Adams, R., Gorman, K. & Perlich, C. (2015), 'Working With Data and Machine Learning in Advertising', *Talking Machines Podcasts* **13**.

Ankerst, M., Elsen, C., Ester, M. & Kriegel, H.-P. (1999), 'Visual classification: an interactive approach to decision tree construction', *Training* **1**, 392–396.
URL: *http://www.dbs.informatik.uni-muenchen.de/Publikationen/Papers/Kdd-99.final.pdf*

a.T.C. Goh (1995), 'Back-propagation neural networks for modeling complex systems', *Artificial Intelligence in Engineering* **9**(3), 143–151.

Becker, B., Kohavi, R. & Sommerfield, D. (2001), 'Visualizing the simple Bayesian classifier', *Information visualization in data mining and knowledge discovery* **18**, 237–249.
URL: *http://books.google.com/books?hl=en&lr=&id=2gSZv1fikJoC&oi=fnd&pg=PA237&sig=j1S-ESo9o_vC00W-4gV-Acay6Go*

Belkin, M. & Niyogi, P. (2002), 'Laplacian Eigenmaps and spectral techniques for embedding and clustering'.

Bengio, Y., Ducharme, R., Vincent, P. & Janvin, C. (2003), 'A Neural Probabilistic Language Model', *The Journal of Machine Learning Research* **3**, 1137–1155.

Bergstra, J., Yamins, D. & Cox, D. (2013), 'Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures', *Proceedings of the 30th International Conference on Machine Learning* pp. 115–123.
URL: *http://jmlr.org/proceedings/papers/v28/bergstra13.html*

Bostock, M., Heer, J. & Ogievetsky, V. (2011), 'D3.js - Data-Driven Documents'.
URL: *http://d3js.org/*

Bruckner, D., Rosen, J. & Sparks, E. R. (2013), 'DeepViz : Visualizing Convolutional Neural Networks for Image Classification'.

Bruna, J. & Polytechnique, E. (2012), 'Invariant Scattering Convolution Networks ', pp. 1–15.

Burkhard, R. (2004), 'Learning from architects: the difference between knowledge visualization and information visualization', *Proceedings. Eighth International Conference on Information Visualisation, 2004. IV 2004.* .

Caragea, D., Cook, D. & Honavar, V. (2001), 'Gaining Insights into Support Vector Machine Pattern Classifiers Using Projection-Based Tour Methods', *Proceedings of the KDD Conference* pp. 251–256.

Carlsson, G., Ishkhanov, T., De Silva, V. & Zomorodian, A. (2008), 'On the local behavior of spaces of natural images', *International Journal of Computer Vision* **76**(1), 1–12.

Chernoff, H. (1973), 'The use of faces to represent points in k-dimensional space graphically.', *Journal of the American Statistical Association* **68**(342), 361–368.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H. & Bengio, Y. (2014), 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation', *arXiv* .
URL: *http://arxiv.org/abs/1406.1078*

Choi, J. C. J. & Horowitz, R. (2005), 'Topology preserving neural networks that achieve a prescribed feature map probability density distribution', *Proceedings of the 2005, American Control Conference, 2005.* pp. 1343–1350.

Craven, M. W. & Shavlik, J. W. (1992), 'Visualizing Learning and Computation in Artificial Neural Networks', *International Journal on Artificial Intelligence Tools* **01**(03), 399–425.

Cristina, M., Oliveira, F. D. & Levkowitz, H. (2003), 'From Visual Data Exploration to Visual Data Mining : A Survey', **9**(3), 378–394.

Dahl, R. (2009), 'Node.js'.
URL: *https://nodejs.org/*

Dai, J. & Cheng, J. (2008), 'HMMEditor: a visual editing tool for profile hidden Markov model.', *BMC genomics* **9 Suppl 1**, S8.

DeFanti, T. a., Brown, M. D. & McCormick, B. H. (1989), 'Visualization: expanding scientific and engineering research opportunities', *Computer Graphics and Applications, IEEE* **22**(8), 12–16.

Demartines, P. & Herault, J. (1995), 'CCA : Curvilinear Component Analysis " 1 Introduction 2 Algorithm', *Kybernetik* pp. 1–4.

Dholakiya, J. H. & Kiran, R. (2015), 'Expresso : A user-friendly GUI for designing , training and using Convolutional Neural Networks'.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E. & Darrell, T. (2013), 'DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition', *International Conference on Machine Learning* pp. 647–655.
URL: *http://arxiv.org/abs/1310.1531*

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization', *Journal of Machine Learning Research* **12**, 2121–2159.
URL: *http://jmlr.org/papers/v12/duchi11a.html*

Erhan, D., Bengio, Y., Courville, A. & Vincent, P. (2009), 'Visualizing higher-layer features of a deep network', *Bernoulli* (1341), 1–13.
URL: *http://igva2012.wikispaces.asu.edu/file/view/Erhan+2009+Visualizing+higher+layer+features+of+a*

Garson, G. D. (1991), 'Interpreting Neural-network Connection Weights', *AI Expert* **6**(4), 46–51.
URL: *http://dl.acm.org/citation.cfm?id=129449.129452*

Graves, A. & Jaitly, N. (2014), 'Towards End-To-End Speech Recognition with Recurrent Neural Networks', *JMLR Workshop and Conference Proceedings* **32**(1), 1764–1772.
URL: *http://jmlr.org/proceedings/papers/v32/graves14.pdf*

Hinton, G. E. (1986), 'Learning distributed representations of concepts'.
URL: *http://www.cogsci.ucsd.edu/ ajyu/Teaching/Cogs202_sp13/Readings/hinton86.pdf*

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012), 'Improving neural networks by preventing co-adaptation of feature detectors', *arXiv: 1207.0580* pp. 1–18.
URL: *http://arxiv.org/abs/1207.0580*

Hinton, G. & Roweis, S. (2002), 'Stochastic Neighbor Embedding'.

Hoffman, P. (1999), 'Table Visualization: A formal model and its applications'.

Hotelling, H. (1933), 'Analysis of a complex of statistical variables into principal components', *J. Educ. Psych.* **24**.

Iliinsky, B. N. (2013), 'Choosing visual properties for successful visualizations', p. 12.

Jarrett, K., Kavukcuoglu, K., Ranzato, M. & LeCun, Y. (2009), 'What is the best multi-stage architecture for object recognition?', *Proceedings of the IEEE International Conference on Computer Vision* pp. 2146–2153.

Keim, D. A. (2000), 'Designing Pixel-Oriented Visualization Techniques : Theory and Applications', **6**(1), 1–20.

Keim, D. a. (2002), 'Information visualization and visual data mining', *IEEE Trans Vis Comput Graph* **8**(1), 1–8.

Krizhevsky, a., Sutskever, I. & Hinton, G. (2012), 'Imagenet classification with deep convolutional neural networks', *Advances in neural information processing systems* pp. 1097–1105.

Le, Q. V., Ngiam, J., Chen, Z., Chia, D., Koh, P. W. & Ng, A. Y. (2010), 'Tiled convolutional neural networks', *Advances in Neural Information Processing Systems 23* pp. 1279–1287.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989), 'Backpropagation Applied to Handwritten Zip Code Recognition'.

Maaten, L. V. D. & Hinton, G. (2008), 'Visualizing Data using t-SNE', *Journal of Machine Learning Research* **9**, 2579–2605.

McCormick, B. H., DeFanti, T. a. & Brown, M. D. (1987), 'Visualization in Scientific Computing'.
URL: *http://www.cogsci.ucsd.edu/ ajyu/Teaching/Cogs202_sp13/Readings/hinton86.pdf*

Meihoefer, H.-J. (1973), 'the Visual Perception of the Circle in Thematic Maps/Experimental Results', *Cartographica: The International Journal for Geographic Information and Geovisualization* **10**(1), 63–84.

Munro, P. (1992), 'Visualizations of 2-D hidden unit space', *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks* **3**, 468–473.

Norouzi, M., Mikolov, T., Bengio, S., Singer, Y. & Mar, L. G. (2014), 'Zero-Shot Learning by Convex Combination of Semantic Embeddings', *ArXiV* pp. 1–9.

Olah, C. (2014*a*), 'Deep Learning , NLP , and Representations', pp. 1–9.

Olah, C. (2014*b*), 'Neural Networks, Manifolds, and Topology'.
URL: *http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/*

Olah, C. (2014*c*), 'Visualizing MNIST: An Exploration of Dimensionality Reduction'.
URL: *http://colah.github.io/posts/2014-10-Visualizing-MNIST/*

Olah, C. (2014*d*), 'Visualizing Representations: Deep Learning and Human Beings'.
URL: *http://colah.github.io/posts/2015-01-Visualizing-Representations/#fn10*

Patel, K., Fogarty, J., Landay, J. a. & Harrison, B. (2008), 'Examining Difficulties Software Developers Encounter in the Adoption of Statistical Machine Learning', *Proc. AAAI 2008* (Hand 1998), 1998–2001.

Roweis, S. T. & Saul, L. K. (2000), 'Nonlinear dimensionality reduction by locally linear embedding.', *Science (New York, N.Y.)* **290**(5500), 2323–2326.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Jan, C. V., Krause, J. & Ma, S. (2015), 'ImageNet Large Scale Visual Recognition Challenge', *arXiv preprint arXiv:1409.0575* .

Sainath, T. N., Kingsbury, B., Saon, G., Soltau, H., Mohamed, A.-r., Dahl, G. & Ramabhadran, B. (2015), 'Deep Convolutional Neural Networks for Large-scale Speech Tasks', *Neural Networks* **64**, 39–48.
URL: *http://linkinghub.elsevier.com/retrieve/pii/S0893608014002007*

Sammon, J. W. (1969), 'A Nonlinear Mapping for Data Structure Analysis', *IEEE Trans. Comput.* **18**(5), 401–409.
URL: *http://dx.doi.org/10.1109/T-C.1969.222678*

Shoresh, N. & Wong, B. (2011), 'Points of view: Data exploration', *Nature Methods* **9**(1), 5–5.
URL: *http://dx.doi.org/10.1038/nmeth.1829*

Simonyan, K., Vedaldi, A. & Zisserman, A. (2013), 'Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps', *arXiv preprint arXiv:1312.6034* pp. 1–8.
URL: *http://arxiv.org/abs/1312.6034*

Song, L., Smola, A., Borgwardt, K. & Gretton, A. (2007), 'Colored Maximum Variance Unfolding', pp. 1–8.
URL: *http://eprints.pascal-network.org/archive/00003144/*

Streeter, M., Ward, M. & Alvarez, S. a. (2001), 'NVIS : an interactive visualization tool for neural networks'.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014), 'Sequence to Sequence Learning with Neural Networks', *Nips 2014* pp. 1–9.

Talbot, J., Lee, B., Kapoor, A. & Tan, D. S. (2009), 'EnsembleMatrix: interactive visualization to support machine learning with multiple classifiers', *Learning* pp. 1283–1292.
URL: *http://portal.acm.org/citation.cfm?id=1518895*

Tenenbaum, J. B., Silva, V. D. & Langford, J. C. (2000), 'Sci_Reprint', **290**(December), 2319–2323.

Torgerson, W. S. (1952), 'Multidimensional scaling: I. Theory and method'.

Tufte, E. R. (2001), *The visual display of quantitative information*, 2nd ed. edn, Graphics Press, Cheshire, Conn.

Tufte, E. & Sigma, M. (2012), 'Why Visualisation'.
URL: *http://www.mu-sigma.com/uvnewsletter/index.html*

Tzeng, F.-Y. & Ma, K.-L. (2005), 'Opening the Black Box - Data Driven Visualization of Neural Networks', *VIS 05. IEEE Visualization, 2005.* (x), 383–390.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1532820*

van der Maaten, L., Postma, E. & van den Herik, J. (2009), 'Dimensionality Reduction: A Comparative Review', *Journal of Machine Learning Research* **10**(January), 1–41.

Vondrick, C., Khosla, A., Malisiewicz, T. & Torralba, A. (2013), 'HOGgles: Visualizing object detection features', *Proceedings of the IEEE International Conference on Computer Vision* pp. 1–8.

Wang, R., Perez-Riverol, Y., Hermjakob, H. & Vizcaíno, J. A. (2015), 'Open source libraries and frameworks for biological data visualisation: A guide for developers', *Proteomics* **15**(8), 1356–1374.
URL: *http://doi.wiley.com/10.1002/pmic.201400377*

Ware, C. (2010), *Visual Thinking for Design : for Design*, Elsevier Science, Burlington.
URL: *http://ncl.eblib.com/patron/FullRecord.aspx?p=405649*

Ware, M., Frank, E., Holmes, G., Hall, M. & Witten, I. H. (2002), 'Interactive Machine Learning : Letting Users Build Classifiers', *Int. J. Hum.-Comput. Stud.* .

Weinberger, K. Q. & Saul, L. K. (2004), 'Learning a kernel matrix for nonlinear dimensionality reduction', (July).

Wejchert, J. & Tesauro, G. (1990), 'Neural Network Visualization', *Advances in Neural Information Processing Systems 2* pp. 465–472.
URL: *http://papers.nips.cc/paper/286-neural-network-visualization.pdf\nfiles/4502/Wejchert ? Tesauro - 1990 - Neural Network Visualization.pdf\nfiles/4503/286-neural-network-visualization.html*

Zeiler, M. D. & Fergus, R. (2013), 'Visualizing and Understanding Convolutional Networks', *arXiv preprint arXiv:1311.2901* .
URL: *http://arxiv.org/abs/1311.2901*

Zeiler, M. D., Taylor, G. W. & Fergus, R. (2011), 'Adaptive deconvolutional networks for mid and high level feature learning', *Proceedings of the IEEE International Conference on Computer Vision* pp. 2018–2025.

A Classifying Academic Visualisations

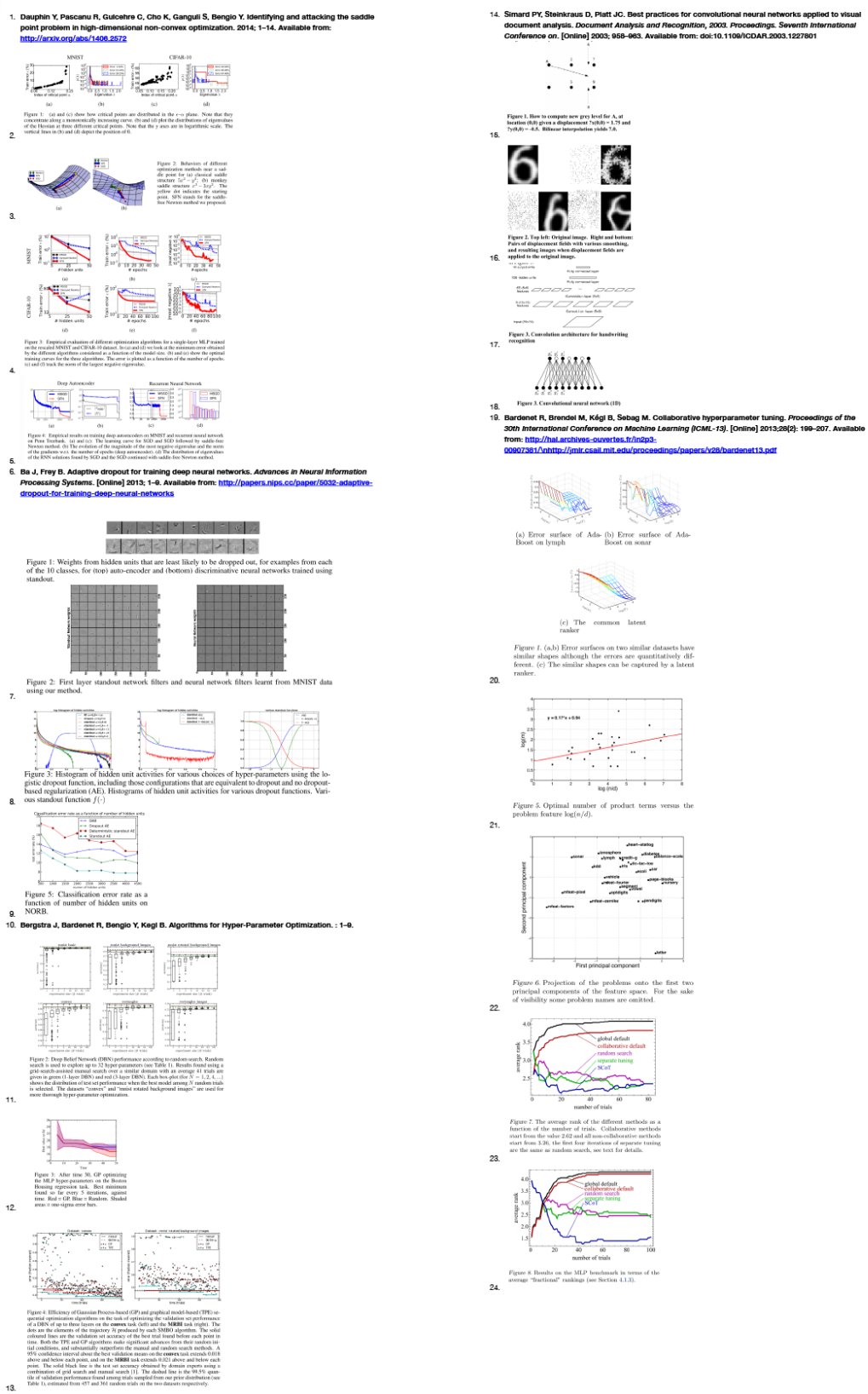


Figure 82: Sample of Classifying Image Data

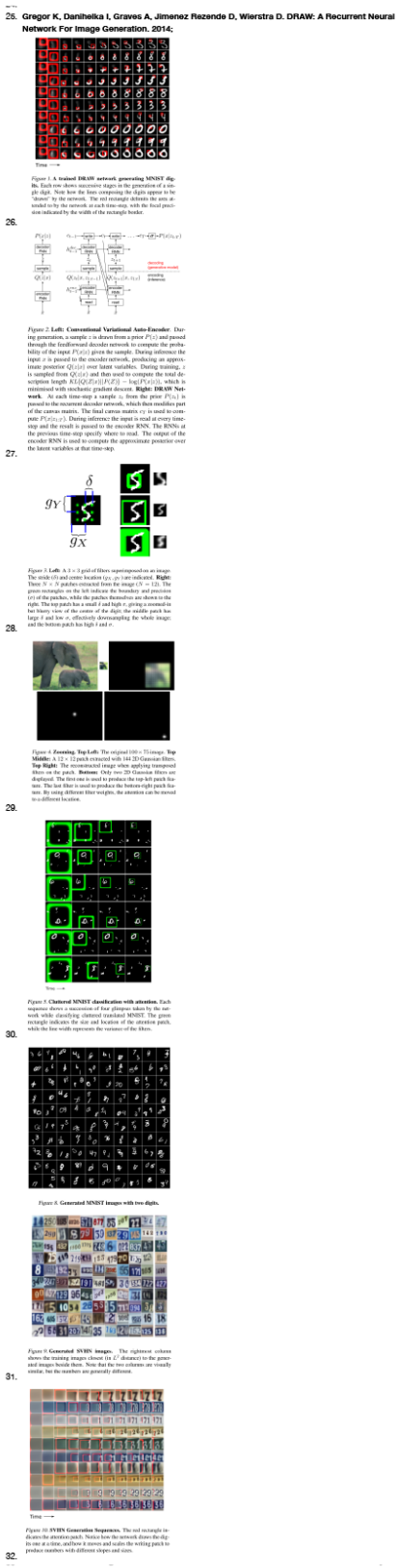


Figure 83: Sample of Classifying Image Data

No.	Type	Comparison	Scales & Visualisation Key
2	X-Y Line Graph	MNSIT -vs- CIFAR-10	(X) Index of Critical Point (Y) % Train Error
2	X-Y LIne Graph	Error weights (3)	(X) Eigenvalue
3	3D Surface Plot	Newton -vs- Saddle-Free Newton (SFN) -vs- Stochasic Gradient Descent (SGD)	Gradient Descent Paths on 3D plane (Monkey / Classical saddle structure)
4	X-Y Line Graph	MNSIT -vs- CIFAR-10 (Damped Newton, SFN & SGD)	(X) No Hidden Units (Y) % Train error
4	X-Y Line Graph	MNSIT -vs- CIFAR-10 (Damped Newton, SFN & SGD)	(X) No Epochs (Y) % Train error
4	X-Y Line Graph	MNSIT -vs- CIFAR-10 (Damped Newton, SFN & SGD)	(X) No epochs (Y) Largest Negative Eigenvalue
5	X-Y Line Graph	Deep Autoencoder -vs- RNN (SFN & SGD)	Learning Curve
5	X-Y Line Graph	Deep Autoencoder -vs- RNN (SFN & SGD)	Magnitude of Most Negative Eigenvalue & Normalised gradints (X) No. Epochs
5	Bar Chart	RNN (SFN -vs- SGD)	Distribution of Eigenvalues
7	Image (W x H)	Autoencoder -vs- Discriminative Neural Network	2 Row x 10 Col Images of 'Weights from hidden units least likely to be dropped out'
7	Image (W x H)	Standout Network Filters vs. Neural Network Filters (MNIST)	10 Row x 10 Col images of 'Network Weights'
8	Log Histogram	Dropout Functions & Standout Functions	Hidden unit activities for various choices of hyper-parameters
9	X-Y Line Graph	Different Algorithm Choices	(X) No Hidden Units (Y) % Test Classificiation Error rate
11	Box Plot	Performance testing 32-Hyperparameters & Different Network Depths	(Y) Accuracy (X) No. Trails / Experiment Size -> experimenting across different image sets
12	Area Chart	Gaussian Process (GP) optimized Multi-Layer Perceptron (MLP) Regression	(Y) Best value thus far (X) Time (Shaded Area) One-sigma error bars
13	Scatterplot	Manual -vs- random -vs- GP -vs- graphical-model based (TPE) Sequential Optimization Al	(Y) Error, as fraction of incorrect (X) Time (Trails) --> Convex -vs-MNIST Rotated Background Images
16	Image (W x H)	Pairs of displacement fields & Resulting Images when applied	2 Row x 4 Col Images
17	Diagram	Convolutional Architecture	Layer by Layer: Rectilinear Planes
18	Network Diagram	Convolutional Architecture	Nodes and Edge Connections
20	X-Y-Z Graph (3D)	Error Surface of AdaBoost Algorithm on two different but similar datasets	(Z) Error (Y) Log M [number of terms] (X) Log T [number of boosting operations]
21	Scatterplot	Case Study Example	(Y) Log M [numbr of product terms] (X) Log (n/d) [number of training instances / number of attributes]
22	Annotated Scatterplot	Projection of the problems onto the first two principle compontents	(Y) Second Principle Component (X) First Principle Component
23	X-Y Line Graph	Separate Tuning -vs- Random Search -vs- Global Default	(Y) Average Rank (X) Number of trails
26	Image (W x H)	Successive stages in generation of a single MNIST digit	11 Row x 11 Col -> Col's increase by over time. Red rectangle incating focus of generation.
27	Process Diagram	DRAW network architecture -vs- Convolutional network architecture	Rectilinear Boxes
28	Annotated Image (W x H)	3 x 3 grid of filters superimposed onto an image.	Image and Scaled up Comparison
30	Annotated Image (W x H)	Cluttered MNIST classification, with attention to area being classfied	(Green highlighting box) indicates size and location of attention patch (Line Width) Variance of the filters
32	Annotated Image (W x H)	SVHN Generation Sequences	(Y) Different Image Samples (X) Time
34	Network Diagram	Network Architecture (Standard Neural Net -vs- After Applying Dropout)	Nodes and Edge Connections. Dropped units have crosses filled in, and no edges connecting them.
36	X-Y Line Graph	With dropout -vs- without dropout (Different architecture comparison of error)	(Y) Classification Error % (X) Number of weight updates
37	Image and Bar Graph	ImageNet test cases, and corresponding 4 most probable labels	(Bar graph) Normalised Probabilities (Colour) The ground Truth
38	Image (W x H)	Learned Features on MNIST: Dropout -vs- without dropout	Two lots of (15 Row x 15 Col Images)
39	Histogram	With dropout -vs- without dropout (Mean Activation and Activation Graphs)	(Y) Occurances of Specific Activation (X) Mean Activation (or Activation)

Figure 84: Sample of Analysis of Image Data

Name	Language	Platform	Type	Description	Supported Categories	URL
D3.js	JavaScript	Web	Open source	JavaScript visualisation framework designed to utilise the capabilities of CSS3, HTML5 and SVG	Charts/Hierarchies	http://d3js.org/
JFreeChart Orson Charts	Java	Desktop Web	Free	Java well-known library to generate charts in 2D and 3D	Charts	http://www.jfree.org/jfreechart/
Google Charts	JavaScript	Web	Free	Google project to embed many different kinds of charts and maps in web pages	Charts/Hierarchies	http://code.google.com/apis/charttools
matplotlib	Python	Desktop	Open source	Desktop plotting library written in Python for creating quality plots (primarily in 2D)	Charts/Hierarchies	http://matplotlib.org/
MPLD3	Python	Web	Open source	Python package that provides a *D3.js* based viewer for *matplotlib*	Charts	http://mpld3.github.io/
GRAL	Java	Desktop	Open source	Java library for displaying plots	Charts	http://trac.erichseifert.de/gral/
Jzy3d	Java	Desktop	Open source	Library to easily draw 3D scientific data	Charts	http://www.jzy3d.org/
XChart	Java	Desktop	Open source	Lightweight Java library for plotting data	Charts	http://www.xeiam.com/xchart
Flot	JavaScript	Web	Open source	Plotting library for jQuery	Charts	http://www.flotcharts.org/
Bokeh	Python	Web	Open source	Python interactive visualization library that targets modern web browsers	Charts	http://bokeh.pydata.org/
Cytoscape	Java	Desktop	Open source	Framework for integrating, visualising and analysing data in the context of biological networks	Networks/Hierarchies	http://www.cytoscape.org/
Cytoscape.js	JavaScript	Web	Open source	Framework for integrating, visualising and analysing data in the context of biological networks	Networks/Hierarchies	http://cytoscape.github.io/cytoscape.js
Gephi	Java	Desktop	Open source	Platform written in Java for visualising and manipulating large graphs	Networks/Hierarchies	http://gephi.github.io/
Graphviz	DOT, Java, Python, C, C++	Desktop/Web	Open source	Graph Visualization Framework for drawing graphs specified in DOT language scripts	Networks/Hierarchies	http://www.graphviz.org/
Sigma.js	JavaScript	Web	Open source	JavaScript library dedicated to graph drawing, using either the HTML canvas element or WebGL	Networks	http://sigmajs.org/
mxGraph	JavaScript	Web	Commercial	Commercial JavaScript library	Networks/Hierarchies	http://www.jgraph.com/mxgraph.html
JGraphX	Java	Desktop	Open source	Open-source Java Swing graph visualization	Networks/Hierarchies	https://github.com/jgraph/jgraphx
JUNG	Java	Desktop	Open source	A library that provides a common and extendible language for graph visualization	Networks/Hierarchies	http://jung.sourceforge.net/

Figure 85: Overview of visualisation software by Rui Wang