Visualisation for Information: Deep Neural Networks

— Background and Progress Report —

Sam Green
sg5414@imperial.ac.uk

Supervisors: Dr William Knottenbelt and Mr Daniel 'Jack' Kelly

28th August, 2015

Abstract

A package, or platform designed to help experts in gaining deeper understanding of their artificial neural network models to diagnose potentially problematic issues with their model structure. This should allow for a more rapid iteration process as the researcher seeks to converge upon a well performing model.

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

1.1 Motivations

Deep Neural Networks 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 unlike some 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.

There have been many studies mathematically analysing these networks, several aiming to optimise the 'gradient descent' algorithms that are at the heart of Deep Neural Networks. However this paper is concerned less with the theoretical underpinnings of the networks, but how, in a field where there is little to base complex decisions such as parameter tuning on, does a research decide how to train their models.

This paper seeks to explore the usefulness of visualisation as a research tool. The hope is that visualisation may prove to be an effective means of exploring ones network - providing key and potentially novel insights for the researchers and practitioners currently working in the field of *deep learning*.

Data visualisation can be defined as the graphical display of abstract information for two purposes: data analysis and communication. Data visualisation has long been an integral tool for scientific research, constituting a powerful means to discover and understand the information available in the data and to present them to others. As we currently are in the 'Big Data' era, it becomes more important to expand our capacity to process this information for analysis and communication. The main goal of visualising data is to benefit from the natural human pattern recognition ability, and apply this through interactive software for efficient exploration and communication.

1.2 Objectives

The main objective is to develop a tool capable of visualising the internal changes occurring within a neural network. As with any tool there are different use cases and so the objectives of this report will be to explore a number of components: Generating the Training Data (Running a neural net) Produce an easy to integrate package to the workflow Produce a more advanced system that allows for interrogation

- **Data Generation** With the majority of papers in the neural network field publishing result figures, or basic network structures that can be particular to any one set of data. The first objective to explore visualisation as a tool for understanding these networks, is to build a neural network and have the ability to easily change and tweak parameters in a controlled manner. This should produce the required data for using data visualisation upon, a collection of different models and their outputs.
- **Data Visualisation: 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 as simple to use as possible, and provide useful feedback upon a networks ability to classify and train.
- **Data Visualisation: A more advance tool for exploration** Having used the simple workflow tool it's likely that the researcher will begin to spot patterns in their networks output, and unforunately with simple methods it's difficult to interrogate these outputs in a particularly effecitve manner. This requires the use of an interactive tool, and so the third objective of this project is to develop an interactive web-app that allows researchers to not only visualise their data, but to interact with it as well.

1.3 Contribution

This project contributes a new tool for academic researchers that enables them to explore the changes occurring within their neural networks at every stage of training. The first tool provides a rough-and-ready approach to *looking inside the black box* and quickly making assessments whether your network is learning or not. The second tool allows researchers who want to have a closer look into their data the ability to interact with the outputs of their neural network by plotting the activations of the network using dimensionality reduction techniques and correlation mapping. This allows for a more meaningful understanding of the matrix data that is output, showing how certain input data-points get misclassified, what types of representations the networks are learning, and whether the network is learning anything useful, or simply just rotating the data (a common problem).

1.4 Report Outline

Chapter 2: Presents a short introduction to Neural Networks and Visualisation before exploring in further depth how the two fields have been combined and looks at some important issues to consider.
Chapter 3: Outlines
Chapter 4
Chapter 5
Chapter 6

2 Identifying the Problem

2.1 Understanding Neural Networks

2.1.1 Overview

In order to understand Neural Networks lets 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 patterns within 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 human learning.

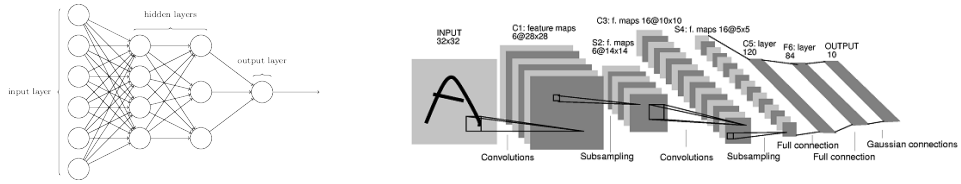
An ANN is a collection of artificial neurons that are connected together in manor which allows them to successfully learn to process information to meet some previously defined end goal. The result of learning is that an ANN becomes a high-dimensional, non-linear, function that is capable of performing a trained task quickly when called upon. Provided with enough hidden units, it can approximate *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 in 2012 however when they were 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 dropout (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 Architecture (b) Convolutional Architecture

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

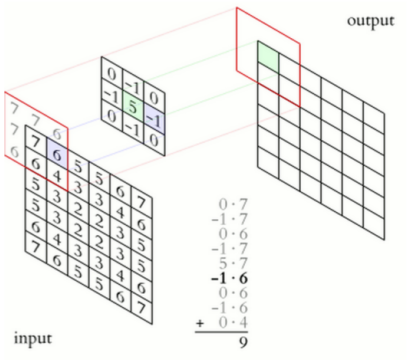


Figure 2: Convolutional Filters

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

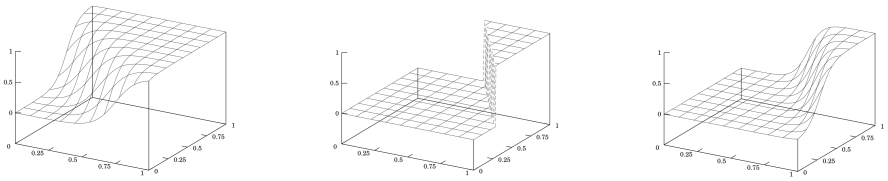
$$y = \begin{cases} 1 & \text{if } M \leq \sum_{i=1}^{k} x_i \cdot w_i + b \text{ where } M \text{ is a threshold parameter} \\ 0 & \text{otherwise.} \end{cases}$$

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.

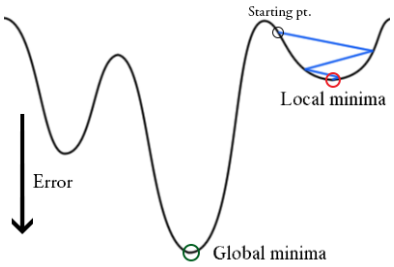


Figure 5: Hill Climbing in the parameter space (Gradient Descent)

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 backpropogation 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 dropout, 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.

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 best are Deep Neural Networks. However, as mentioned earlier there is a problem - to design the networks to they perform as expected is incredibly difficult and there is a great challenge in understanding what these networks are actually doing.

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)

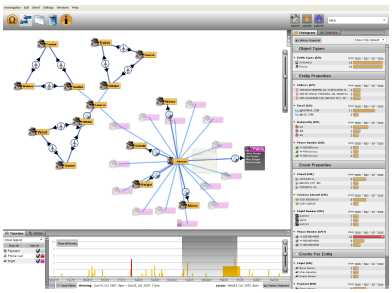


Figure 6: Palantir Screenshot Visualisation

"The use of computer-supported, interactive, visual representations of abstract data to amplify cognition" (**?**)

we can develop visualisations that allow us to work with the computers data handling capabilities and human pattern recognition and understanding to understand neural networks better.

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?

3.1.2 Active Vision Problemsolving

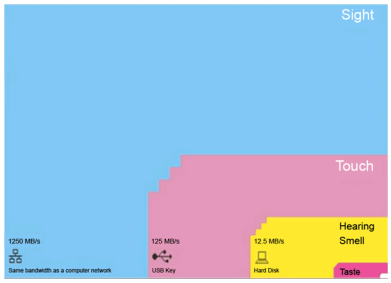


Figure 7: Tor Nrretranders Brain Bandwidth

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 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.

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.

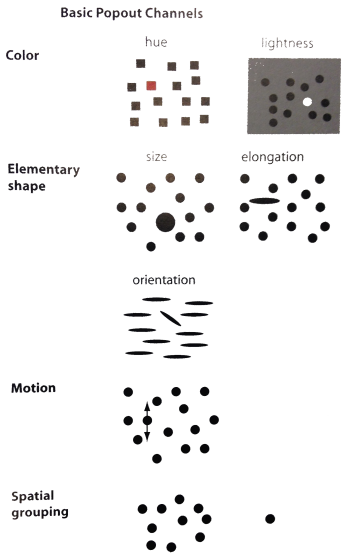


Figure 8: Ware's "Things that pop-out"

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.

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

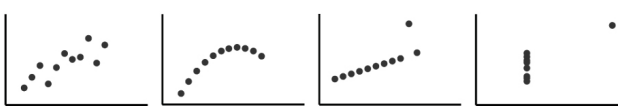
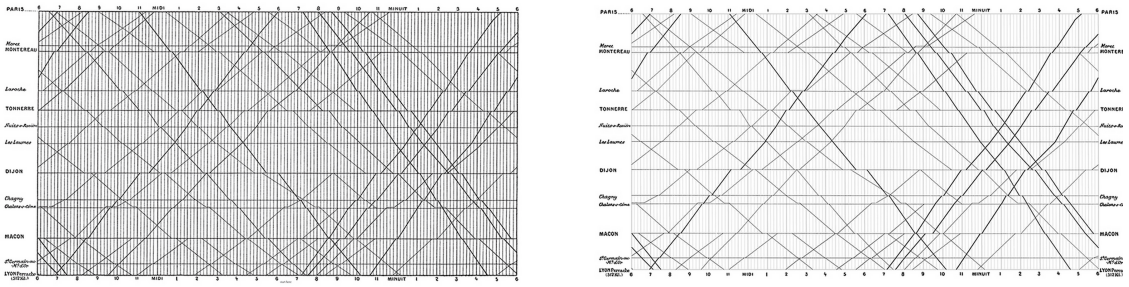


Figure 9: (a) The four sets of numbers that form Anscombe's quartet - (b) The highly distinctive graphs that result from plotting the data in a.

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 10: 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 3D 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. **??**.

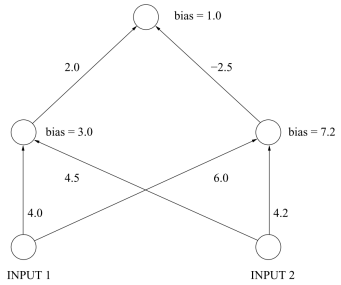


Figure 11: Simple Neural Network

3.3.1 Hinton Diagram

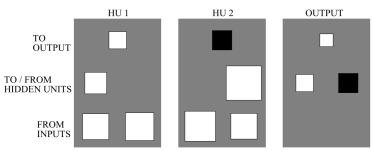


Figure 12: Hinton Diagram

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

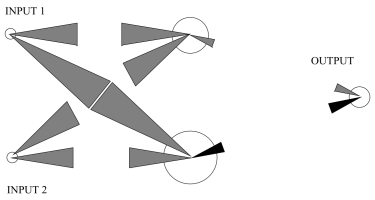


Figure 13: Bond Diagram

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

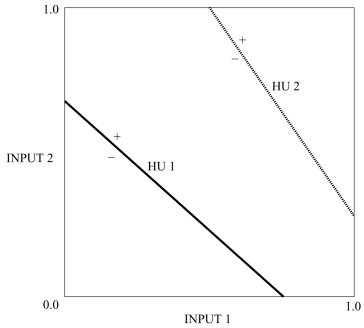


Figure 14: Hyperplane Diagram

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

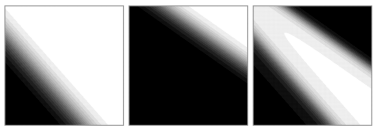


Figure 15: Response Function Plot

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

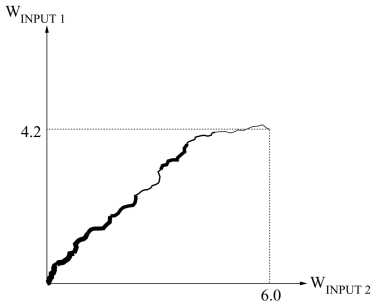


Figure 16: Trajectory Diagram

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).

Lascaux

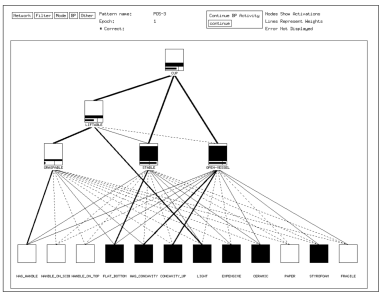


Figure 17: Lascaux Clip

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.

3.4 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.

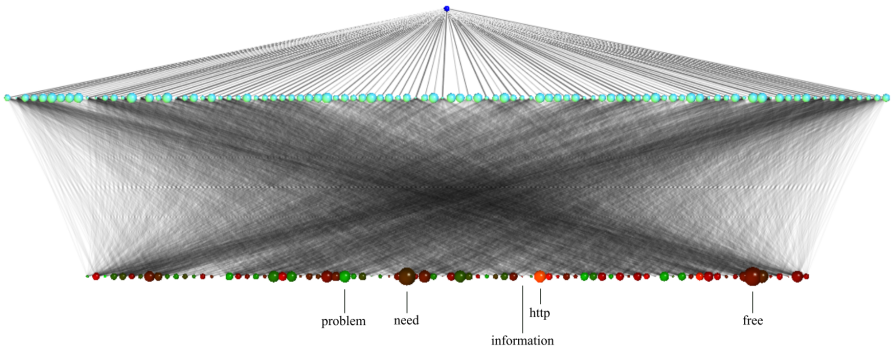


Figure 18: Tzeng Map

(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.

3.4.1 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.

(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.

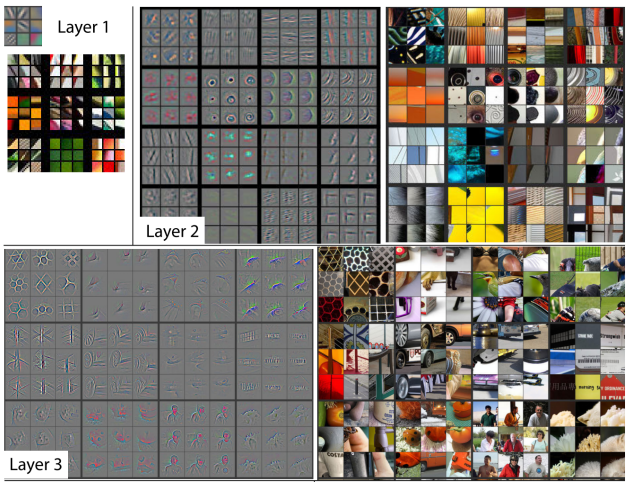


Figure 19: Zeiler Deconv

(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 featurization, and as experts still know relatively little about the behaviour of ANNs it's important to not discard any of these visual aids rashly.

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 20: Overview of visualisation software by Rui Wang

4 Project Goals

To help guide development during the project a number of goals were identified. Over the course of the project the goals have been refined, new goals have been added and some goals have been dropped.

4.1 User Survey & Interviews

4.1.1 Survey Design

In order to help develop a set of refined goals for what was a broad goal - "Visualising Neural Networks" a survey was developed in collaboration with Jack Kelly to distribute amongst the Imperial College Staff known to be working with Neural Networks.

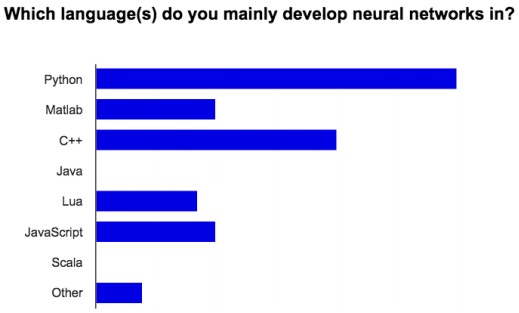
The survey was created and distributed to a small portion of those working closely with Neural Networks in relevant areas. The survey had a number of parts:

- **Describe your working environment** This had specific subtopics on languages used, packages familiar with and time taken to develop
- **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 needed to be made and how they were currently solved.
- **Choose which visualisation technique you think is the most useful** The examples were shown of weights, gradients, activation mapping, architecture graphing, classification distribution and filters.

- **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 mentioned specific packages that are known to be commonly used amongst research communities and also asked about preferred method of interacting with the software.

4.1.2 Survey Results

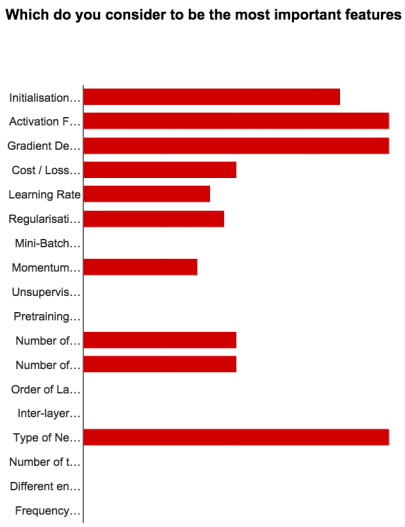
The results from the initial sample of researchers was very conclusive, and tied into my intuitions and my supervisors thoughts on the topic, and the fact that the number of researchers across Imperial was still relatively small for which this early survey would appeal - a second round of delivery was postponed in favour of product development and research into Neural Networks.



In response to working environment it was discovered that there was no package that researchers used any more than any other - and indeed 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 the tweaking of parameters and adjusting neural network architectures was the subject of whole PhD's - suggesting that tools that could help analyse the success of these changes could be invaluable to these researchers.

Numerous methods were described for deducing the correct network parameters, or judging the quality of on set versus another. One researcher would get up on the screen various weight matrices from different training epochs upon 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.



With respect to commenting on existing visualisations, it was surprising to see that most hadn't thought much about visualisation as a tool beyond graphing the commonly used error rates or accuracy. However were interested in the possibilities it might hold, and did respond when asked about which areas would be most useful.

4.2 Goals

Through research into the problem of understanding neural nets, a number of goals were identified. After performing a review of existing software and distributing the survey, the process of constructing and refining a neural network and the network structure themselves the goals were refined. Some of the goals were achieved, however others arose once development had begun and as a result a number of goals were left incomplete.
The initial goals were:

- **Improve existing capabilities** The purpose of this project is to improve the visualisation tools that are available to neural network researchers.
- **Provide Visualisations that take into account the domain** This mean to create a more bespoke tool that can be used during the development of a functional neural network.
- **Visualise changing parts of the neural network** Neural Networks, as explained earlier, have a fairly complex architecture. Individual neurons of different types form layers at different depths and with different numbers of units. Between these layers are weight matrices that change depending on the activations fired from neurons. These activations in the final sigmoid layer enable us to calculate a Validation Loss and Accuracy value. A visualisation tool would hope to visualise all of these changing parts to some degree or another. Not every element was implemented, however the majority are captured indirectly through the activations - which result in the other changes.
- **Animation of this data** Having investigated the foundational theories of visualisation as proposed by Edward Tufte. Another goal of the tool is to enable the animation of the data, to show how the data changes over time. This way helping resaerchers to see how their models are learning - either in classifying the data better, worse or simply differently. This has been implemented in Iteration 1, and researchers can see how their networks classify data differently over time.
- **Investigate which visualisation combinations work best** The purpose of this goal was to try and discover if any visualisations would perform better than others when placed in the context of learning about neural networks. This goal has been achieved iteratively throughout the duration of the project, as each interaction develops on the combinations of visual elements that are needed to effectively help neural network researchers.

- **The ability to look at data from a number of different scales** This goal is crucial in being able to understand both local and global patterns that emerge within the data. In iterations 2,3 & 4 this goal has been satisfied.
- **Experiment data-mining** The purpose of this goal was to have some level of automatic data-collection from their network that would not interfere with their workflow. This goal has been implemented: researchers simply have call a function (API) within their neural network and the tool extracts the relevant information from the training network and stores it in Binary JSON (BSON) format using PyMongo. This was only implmeneted after iteration 3 due to last minute server changes.
- **Provide full record of the network to revert back to successful networks** This goal has been implemented, and while the visualisation content is stored in a database. The crucial parameters of the current network are stored in both JSON and CSV format - a format that researchers are more comfortable working with. This enables researchers to reload models that were previously successful having analysed them, and to then make small adjustments.
- **making the data accessible** All data stored is very accessible, and while visualisation data (coordinates and their corresponding parameters) are avaliable in the pymongo database, easily accessed through a database interaction class. Regular Weights, Biases activations and other parameters are stored on file in formats that are easily accessible.
- **Realtime visualisation update** This goal is clear, that as the neural network is training, the researcher can run the visualisation tool at any time to explore it's progress. The webserver automatically uploads new information when the user selects a project.

4.3 Adjusting Expectations

While the majority of the goals stated in the previous session were achieved by the end of the project. Some of these were left partially complete - such as the "Visualise changing parts of the neural network" which while the final product would visualise a neurons activations and these directly influence the weights, biases, error values and accuracy - the product did not directly visualise anything other than those activations, which proved to be sufficiently useful and complex for the time of this project.

5 Data Collection

The task of visualising neural networks primarily has three challenges to address; producing the data, collecting the data and visualising it.

This chapter addresses the first two of those difficulties. In particular looking at the Neural Network implementations used, explaining the dataset and some considerations when collecting the data.

5.1 Neural Network Implementations

The purpose of creating a neural network in this project is not to create a unique implementation for a dataset that hasn't fully been explored yet. In fact quite the opposite - the implementations should be familiar enough to those looking into this study to enable them to partially ignore the networks producing the data, and instead understand the value of the visualisations that arise because of the network. Added complications in the network would only obfuscate the value of the visualisations.

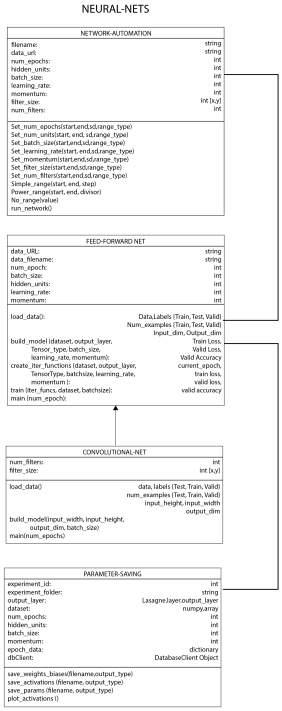
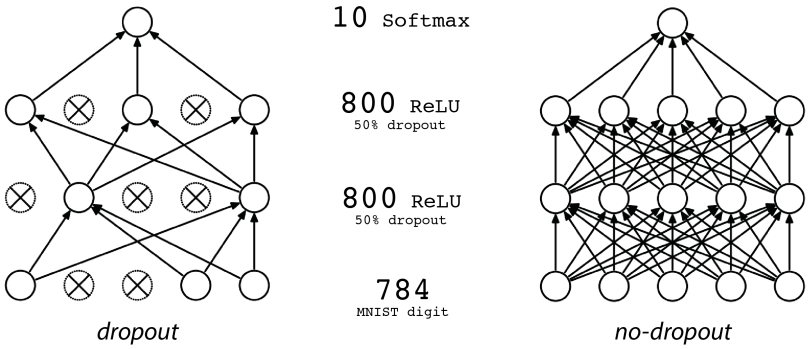


Figure 21

5.1.1 Feed Forward Net

In order to make the experimental network as close to industry standard as possible, the first neural network architecture is the same as that used in the classic Geoffrey Hinton 2012 paper in which he explains the concept of dropout (Hinton et al. 2012) - an idea that addresses a significant problem in machine learning - overfitting.



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

The other parameters used from the Hinton paper were a was 50% dropout between hidden layers and 100-sized minibatches. However training did not follow the recommended 3000 epochs which, where the focus here is on visualisation and not on classification accuracy, was reduced to 100 for most tests, however some were experimented with larger samples and others with far fewer.

5.1.2 Convolutional Net

Similarly to the FeedForward Net, the aim was to use a well understood network.

The convolutional architecture used had the following structure at the beginning, which is a common adaptation of Yann LeCun's 1998 LeNet (**?**):

- 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 networks that are know to produce desirable results, or low classification error, this again does not fully satisfy the needs of this project.

One aim of visualising the networks is to understand when the networks have flaws, imperfections, quirks and other hopefully visually identifiable qualities. In order for us to demonstrate the value of the visualisation, the networks were broken in strategic ways.

For example 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, 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 Neural Network 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 at before runtime to ensure efficient mathematical processing.

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

It's important to note that in this incredibly fast moving field, 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* which haven't been mentioned in this report for simplicity reasons.

5.2 Dataset

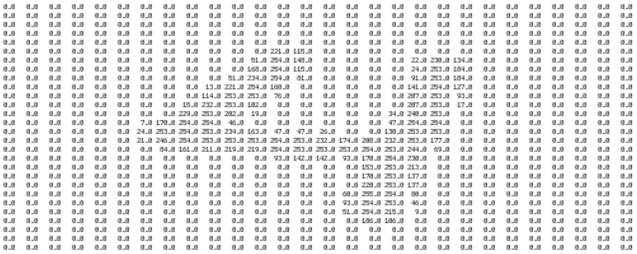
All experiments for this project were conducted using the MNIST dataset. There are a number of reasons behind this: firstly, the dataset is widely used as a benchmarking dataset not just within neural network community but in the wider machine learning community as a whole; secondly, the dataset provides a useful base to test both a Feed Forward Network as well as a Convolutional Network - which should prove the value of the visualisation tool more so than attempting this on any one model architecture.

Figure 22: 100 MNIST digits

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.

It's important to recognise that for the purposes of testing this tool for research the MNIST dataset provides a quick set to train upon. While the CIFAR datasets would provide more interesting analysis and represent a more realistic task - such as image recognition - the MNIST dataset is simply one list of intensity values in comparison the the three Red, Green and Blue of the CIFAR set. The below images demonstrate the simplicity of the dataset.

(a) MNIST digit



(b) Intensities

Figure 23

5.3 Collecting Output Data

5.3.1 Data Collection Goals

There were a number of requirements to fullfill when collecting the data:

- Data that will be used online must be stored in a compact format. This was achieved using a MongoDB database which stores JSON objects in binary, 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.
- Data from the neural networks must retain it's shape for easy importing with the python library `numpy`.
- Image data must be stored in a more compact format that integers of intensity or RGB value due to the vast size. This is achieved by preprocessing image data with the python library `base64`

which compresses the images into a Float32 array in base64 notation - which then usefully becomes small enough to store in the *Mongo* database.

- Coordinate data for the scatter plots must be stored in a condensed form that allows easy itteration over within *d3.js*. (500, 2) matrices are therefore `numpy.reshaped` to (1000,) single dimension arrays.

5.3.2 Structuring the data

Within neural networks there are several moving parts, and with those tremendous amounts of data that can be collected during pretraining, training and during the testing phases.

It was important to fully understand these components in order to effectively store the data outputted from the neural networks in a suitable database. The following entity relationship diagram demonstrate the majority of the moving parts that this project is concerned with, and infact covers the majority of the neural network parameters and data that can be collected.

5.3.3 Collecting the data

In order to process the data in accordance with the above goals, a library of python functions was developed that could be imported with ease into the neural network itself.

These functions effectively create an easy to use API for connecting the neural network to two main methods of storage - the file system storage that enables researchers to examine their data first hand, and the database storage that stores only those pieces of information important for the visualsiations.

The functionality behind this data collection was implemented at first using pythons `os` and `sys` libraries that enable the user to save directly into their file directory. While early on in the process a complex database implementation was created following the below entity relationship schema for the *Node.js* backend, and *MongoDB* database - the *mongodb* collection was later simplified to make it easier for researchers to interact with the database. In addition the database was moved from *JavaScript* to *Python* using a *mongodb* wrapper *PyMongo* to ensure that the backend processing was entirely in Python.

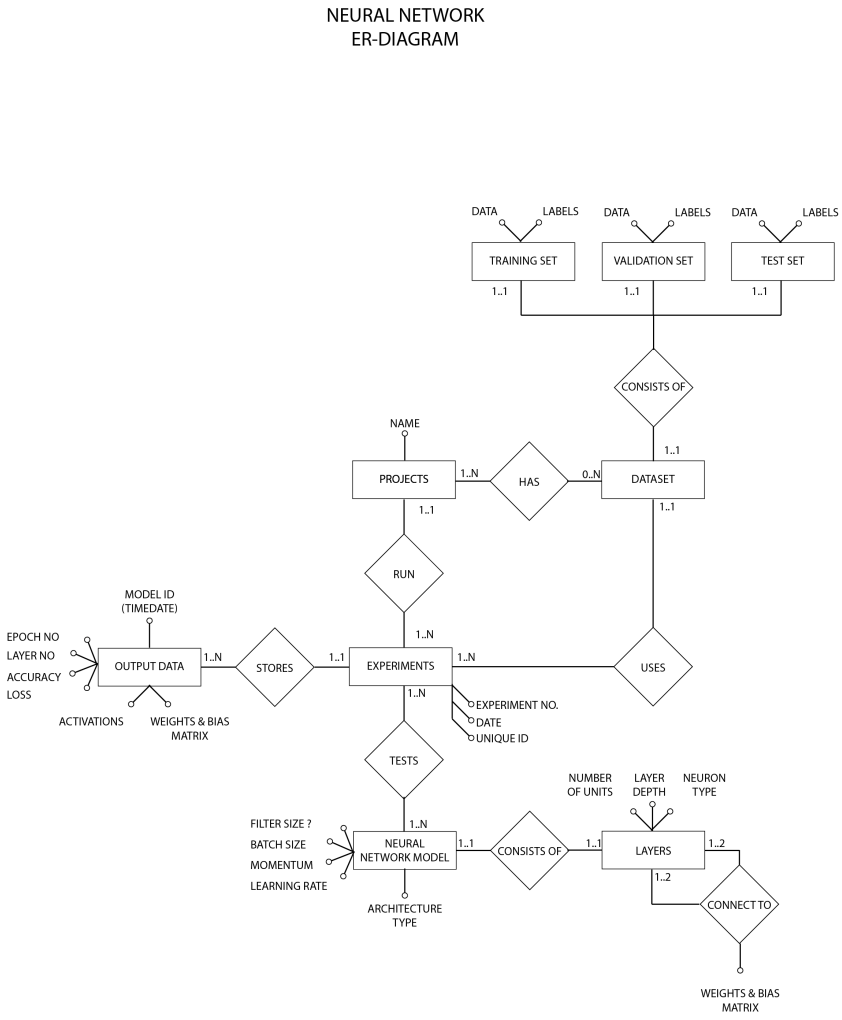


Figure 24: Initial ER diagram

5.4 Evaluation

The aims 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 in order to try to understand better how poor networks may appear when visualised. The section also explains the importance of gathering the data outputs of these networks in an appropriate fashion, and the importance of 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 that data can be visualised and what is significant about this.

6 Dimensionality Reduction

Neural Networks are famous for ability to comprehend complex datasets such as vision or speech. These datasets are often complex for one primary reason - the data contains very high dimensionality.

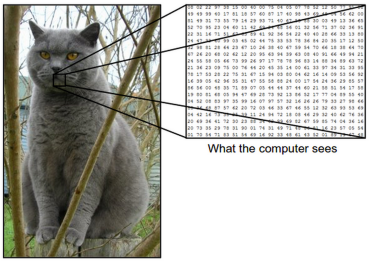


Figure 25: Cat

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 datapoints - 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 this image 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 recognisatble to to - but to a computer, may be more 'cat-like'.

These later dimensions are defined by the activations that a layer of neurons produces in response to being passed the weight-transformed cat image. It is these dimensions that we are interested in, and give us an insight into how successfully our neural network is training.

In order for use to visualise this information in a way that is understandable to the human brain (and therefore useful as a diagnostic tool), we cannot simply show the numbers - we must map these numbers to a smaller number of dimensions that we as human beings are incredibly good at understanding - our 3 spatial dimensions.

This section will explore the notion of both visualising high-dimensional data, where the aim is to retain data fidelity and show all the datapoints, and the notion of dimensionality reduction - where we 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;

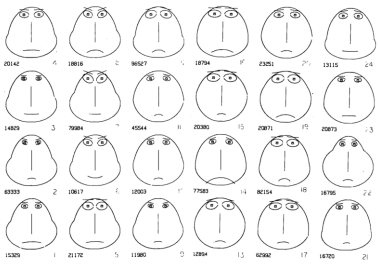


Figure 26: Chernoff Faces

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 datapoints 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.

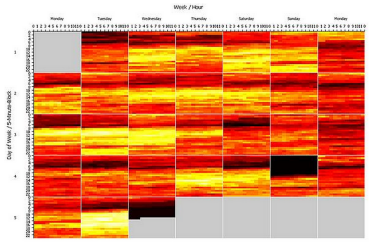


Figure 27: Pixel Based Techniques

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.

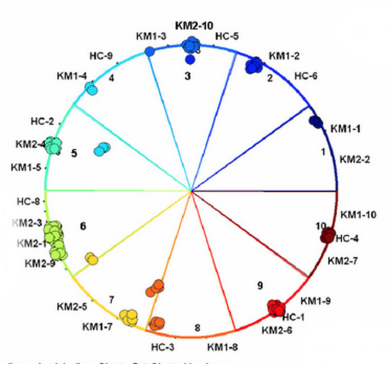


Figure 28: RadViz

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.

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 using 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 user 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.

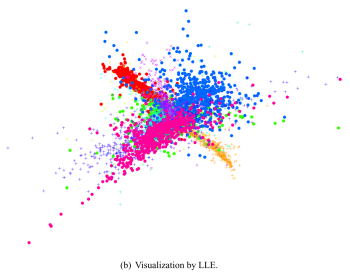


Figure 29: MNIST - a Locally Linear Embedding

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).

6.3 t-Distributed Stochastic Neighbour Embedding

More recently, and in direct challenge to those mentioned earlier, *t-Distributed Stochastic Neighbour Embedding* (Maaten & Hinton 2008) has provided a successful and widely used alternative for neural network researchers. t-SNE, 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.

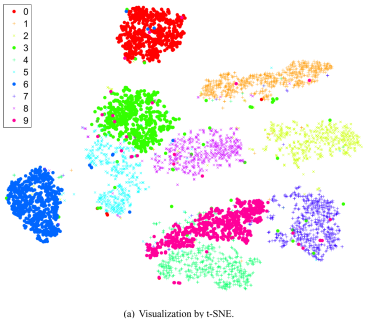


Figure 30: tSNE

t-SNE can therefore be viewed as preserving the topology of the data. t-SNE 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. t-SNE 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 t-SNE reduction have a continuous spectrum of neighbours, for which they are neighbours to different, non-binary, extents. This makes t-SNE very powerful in revealing global clusters and local subclusters 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 t-SNE 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 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.

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

For the remainder of this project data produced by the neural networks will be reduced in dimensions using the t-SNE algorithm, or a faster derivative thereof. The reasons behind this are that when looking into neural networks it is unclear what exactly we are looking for, be it variance, local structure, global structure, or some unknown. t-SNE preserves the overall topological structure and thus provides a good solution when looking into the data generated by the neural nets. In addition, t-SNE has been used incredibly successfully used in the past by some leading neural network researchers to visualise their data(Maaten & Hinton 2008).

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 comparison to the 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 scatterplot. 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 preseve the actual differences in the data (here described using metrics such as the L2, or Euclidean, distance).

7 Iteration 1

7.1 Animation

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 quantities of tSNE plots would quickly put us back in the first positions - being unable to compare data due to information overload, potentially requiring us to reapply tSNE.

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.

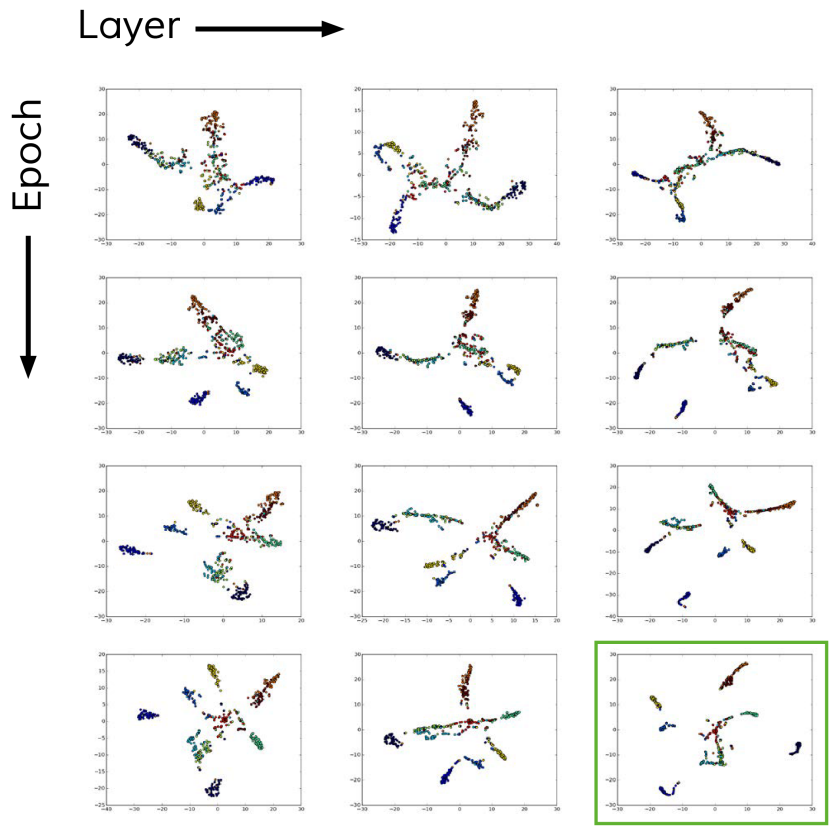


Figure 31: Already in the highlighted image it's clear to see the network is learning some distinction between the classes

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 the data 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 image - 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 bring patterns across the tSNE plots to the fore of our mind, it is not in itself a stand alone too. It is simply a method of processing.

The aim of this project however 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.

7.2 Design

Before setting out to build the tool, a number of options for the best way to display animations were considered. User interface components were continually sketched, and wireframes evaluated in response to the needs of neural network researchers and in relation to visualisation best practices.

Unfortunately it was not possible to get a large range of feedback on the designs due to time constraints, however the little feedback that was attained from fellow students researching with neural networks enabled the project to progress from iterations two, three and four.

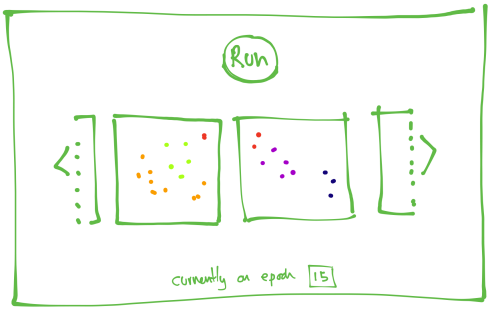


Figure 32: An initial sketch displaying animations moving across the screen as the net trains

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.

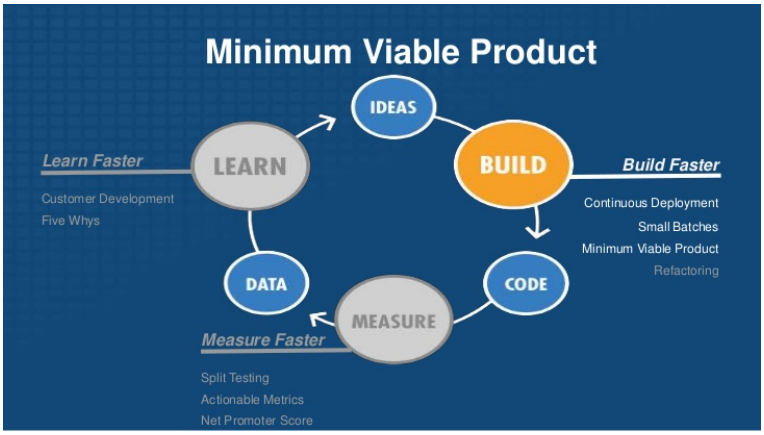


Figure 33: Lean Development Cycle

Here the MVP was a 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 pythons `numpy` 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 saved locally and could be displayed as necessary.

7.4 Evaluation

In order to effectively evaluate the success of this product, there are three perspectives that must be addressed: the perspective of the neural network researcher and the insight concerns; the success of the product as a visualisation and the success of the implementation with respect to the previous two goals.

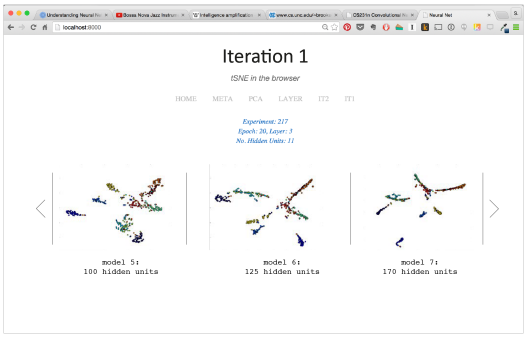


Figure 34: Screenshot iteration 1

7.4.1 Neural Network Perspective

The response from a small test sample was fairly clear - while the animations were useful in highlighting that the network changed over time, it was a highly unsatisfactory product for a couple of important reasons:

- The first and most important consideration was the lack of ability to pause the animations to better understand one abnormal tSNE plot. 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.
- The second reason, was even if there was the ability to pause, the detail in the plots was limited due to filesize constraints - so where a researcher to stop the animation, not much could be learnt.

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.

However, in response to the needs of the neural network reseachers the product was deemed to be of too low resolution - and while increasing the resolution was possible, it was decided that the plots should next be created using scaleable vector graphics. This would ensure the quality of the visualisation even upon infinite zooming - a useful quality where datasets can have several thousands of points.

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 user to grapple with lots of moving parts. Again, causing friction to the tool would ensure that it was never used.

8 Iteration 2

8.1 Online and interactive

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

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 that seeks to be interactive and online, and therefore also enabling the use of JavaScript and thus interactivity. In addition to all of the previous Tufte rules that we have successfullly achieved, there comes two more by bringing the data online.

- Through the use of a common interactive concept for the web *tooltips*, which is simply text that hovers above the object that the mouse is focusing upon at any one moment in time, we can satisfy Tufte's sixth principle to *visually highlight your message*.
- In addition, where previously in 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 the 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

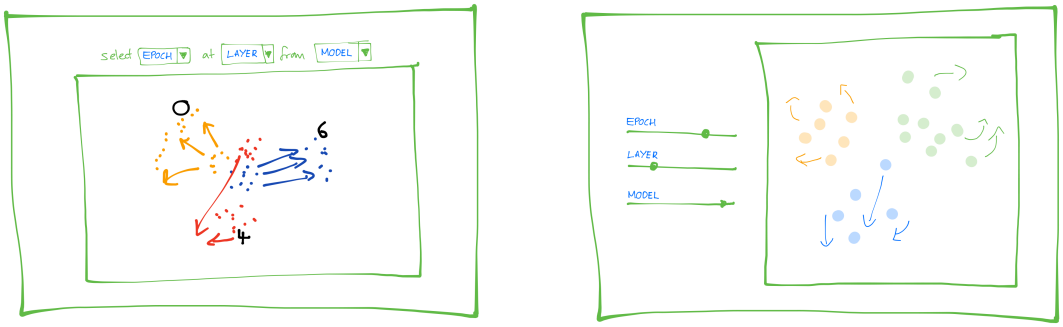
The previous idea was for the animations to load after each model had trained - displaying the full transformation of a model from one that classified poorly with certain distinct characteristics, to one that classified well - potentially again displaying specific characteristics.

This version differed in it's approach. Influenced by Andrej Karparthy's tSNE visualisations, this model visualised the optimising of the tSNE cost function over time, or with respect to certain 'steps'. As shown below in the initial sketch designs below the aim was to show how tSNE functioned pulling apart the classes.

The value of doing this would be to observe how the local and global structures emerged over the course of the tSNE optimisation to a 2D representation that more accurately represented the multiple dimensions of the neural network activation data.

In addition the researcher should be able to choose from the models already stored in the database by selecting a plot based on simple criteria such as:

- Experiment: choose from all the experiments run since the visualisation tool was created.
- Epoch: choose which epoch you want to run, to be able to compare earlier epochs with later ones - testing if the model was indeed still training even if the Loss values had plateaued - enabling you to see if those last few annoying samples were slowly being better classified. For example the unfortunate event where Googles algorithms identified people with Black skin as monkeys - a severe mistake, and one that caused them significant embarrassment.
- Layer: looking by layer shows 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.



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

Figure 35

8.3 Architecture

In order to bring the content online, the author spent a sizeable amount of time learning to use the *MEAN* development stack: Mongo (for the database), Express (for the routing), Angular (for the frontend interaction) and Node (the server).

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

Now that the project is entirely written in javascript, it seemed pointless to have a Python system crunching the tSNE data as well. Far better infact to simply store in the database and call from the JavaScript sever to feed to the front end via. a *RESTful API*. This therefore required a JavaScript implementation of the tSNE algorithm as well.

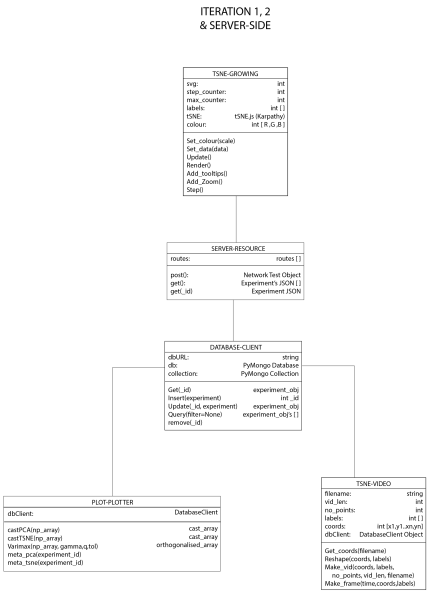


Figure 36: UML for iterations 1 & 2

8.3.1 tSNE.js

Andrej Karpathy, a Stanford PhD student researching neural networks has developed a series of JavaScript based projects that relate to neural networks. While his famous projects is a implementation of a convolutional neural network, he also openly supplies an implementation of the tSNE

algorithm in Javastript - `tSNE.js`.

This second iteration used tSNE.js and followed the example given by Karpathy online in order to produce a tSNE plot that grows over time. The graph below is his implementation applied to word-embeddings (**?**)

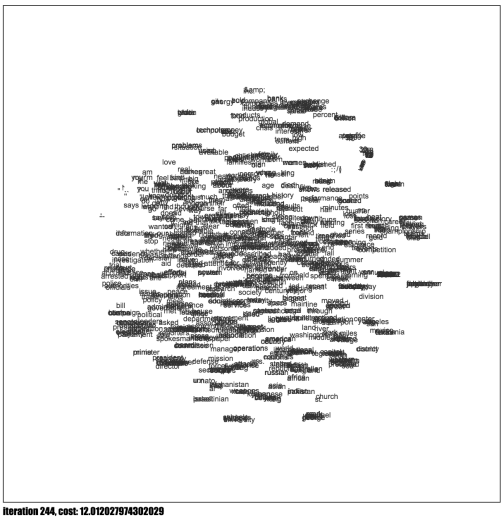


Figure 37: Andrej Karpathy Implementation

8.3.2 Node Server

Node.js 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.(Dahl 2009)

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 uninstallation 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 DL community.

8.3.3 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 at supporting 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 task of exploring visualised data with the aim of deriving new insights from such data.

D3 uses a sophisticated method of joining data with the DOM. With three simple commands (Enter, Update, Exit) it enables you to tell D3 the relationship you want to exist between your data and your SVG. For example, you might want circle elements to correspond to data. Instead of telling D3 to create circles and then collect all the circles and assign each datapoint a circle, as you might do with jQuery.

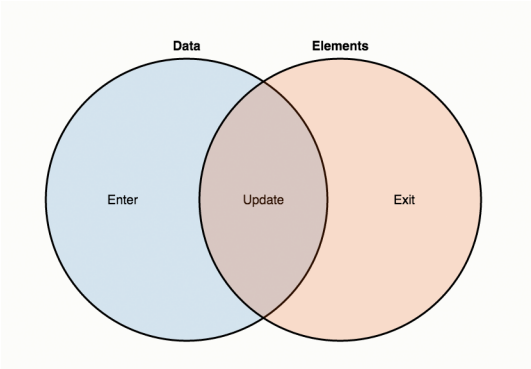


Figure 38: D3 Data Binding

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 (left). Then, any remaining unbound circles produce the exit selection (right). Often these are the points we want to remove. The significance of this is that a scatterplot can be created with not much more code that the following code:

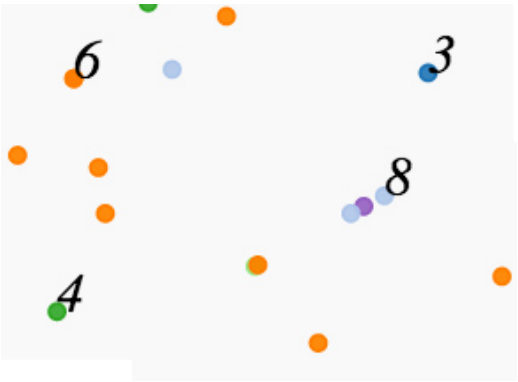
```
var circle = svg.selectAll("circle")
    .data(data);

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 39: 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 powerful tools.

D3 is not just a great way of creating the scalable vector graphics that we require to enable researchers to look both at the global and local structure of the tSNE plots, but also allows us, with relative ease, to bring more detail to the plots. Coding in text tooltips for example ensure that we meet Tufte's principle on focusing in on detail.

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 adaptabel Tooltip d3.js Code

Figure 40

8.4 Evaluation

In order to retain some consistency, the evaluation methods were retianed the same. Assessing the quality of the product under the three main categories of neural network research usefulness, visualisation quality and implementation quality.

This is a screenshot of the complete second iteraction once the cost function has been optimised and the image zoomed in upon. It demonstrate the high-fidelity of the d3.js svg graphic, and shows a marked improvement upon the low fidelity pixelated animations.

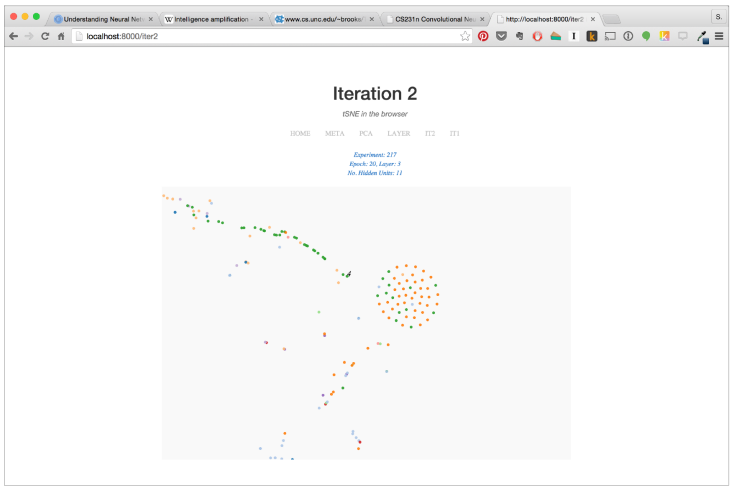


Figure 41: Zoomed in screenshot

Below are a number of screen shots depicting the final product during the stages of tSNE optimisation over it's cost function given at epoch 20, Layer 3 for a derivation of the Hinton network but with only 11 Hidden ReLU units in each layer.

(a) (b) (c)

Figure 42: Screenshots of growth from iteration two

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 user 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 changes that clearly needed to be made:

The version was criticised for taking too long in the processing of each tSNE plot. This makes it hard for users 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 and 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 addressed in version three.

8.4.2 Visualisation Perspective

While in the neural network critisism above it's clear to see that there are a number of visualisation improvements that needed to be addressed, there were significant visualisation discoveries made for the authors perspective. Most notably was 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 the 'WOW' factor, however in this instance they provide an important functional use, and allow the eye to easily follow specific points trajectories in space. This is useful in allowing the user to observe anomalies or perfulaiar changes in the data over time.

The introduction of transitions here is significant, and should 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 The Product

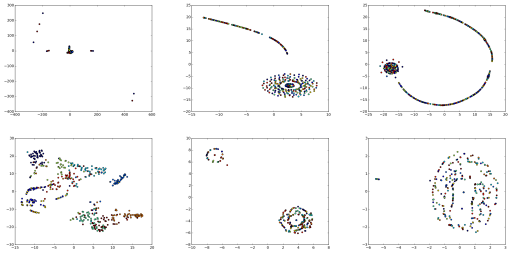


Figure 43: Epoch 2, Layer 1 from different models

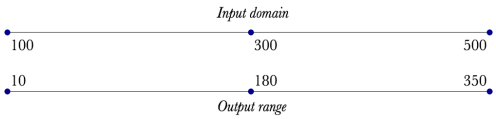


Figure 44: d3 scaling elements

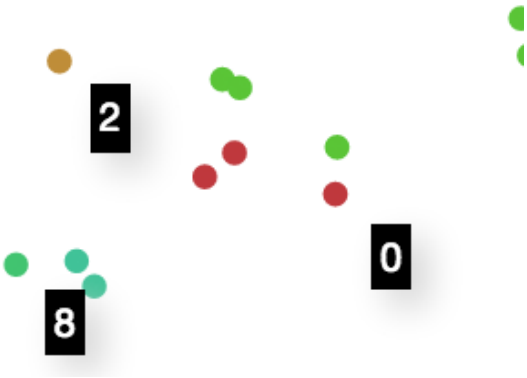


Figure 45: Tooltips

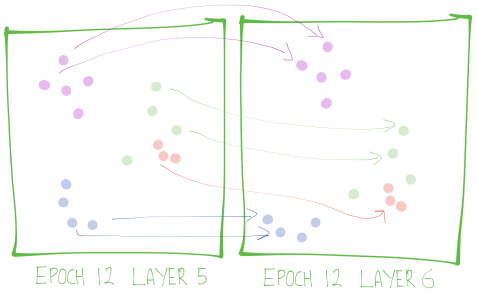


Figure 46: Wireframe Design

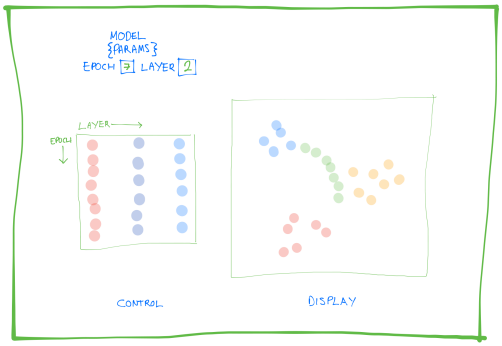


Figure 47: Wireframe Design

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

Figure 48: simple d3 Transitions


```
{
  "_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": {
    "PCA_COORDS": [2,2,5,4,9,12],
    "META_COORDS": [0,1,5,1,7,12],
    "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 49: Final Simplified JSON scheme for PyMongo

9.2 Analysis: Neural Network Response

9.3 Analysis: Visualisation Response

9.4 Analysis: Implementation Response

10 Iteration 4: metaSNE & Image tooltips

10.1 The Product

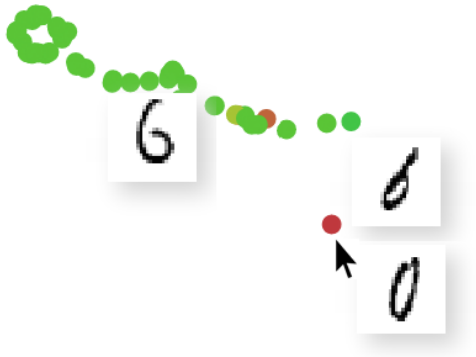


Figure 50: Tooltips

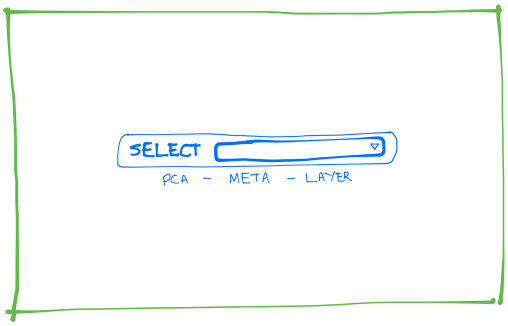


Figure 51: Wireframe Design

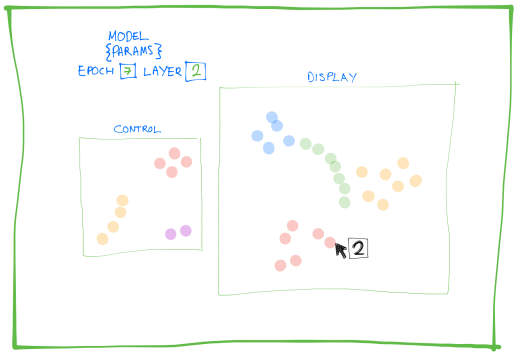


Figure 52: Wireframe Design

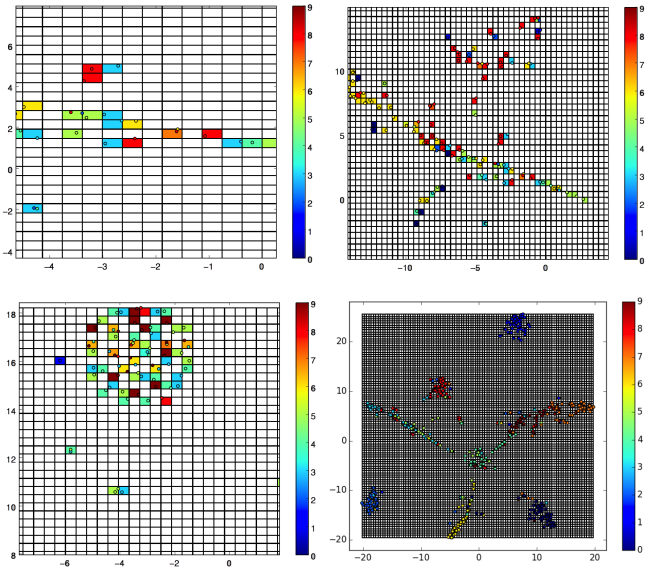


Figure 53: Imagifying Scatterplots

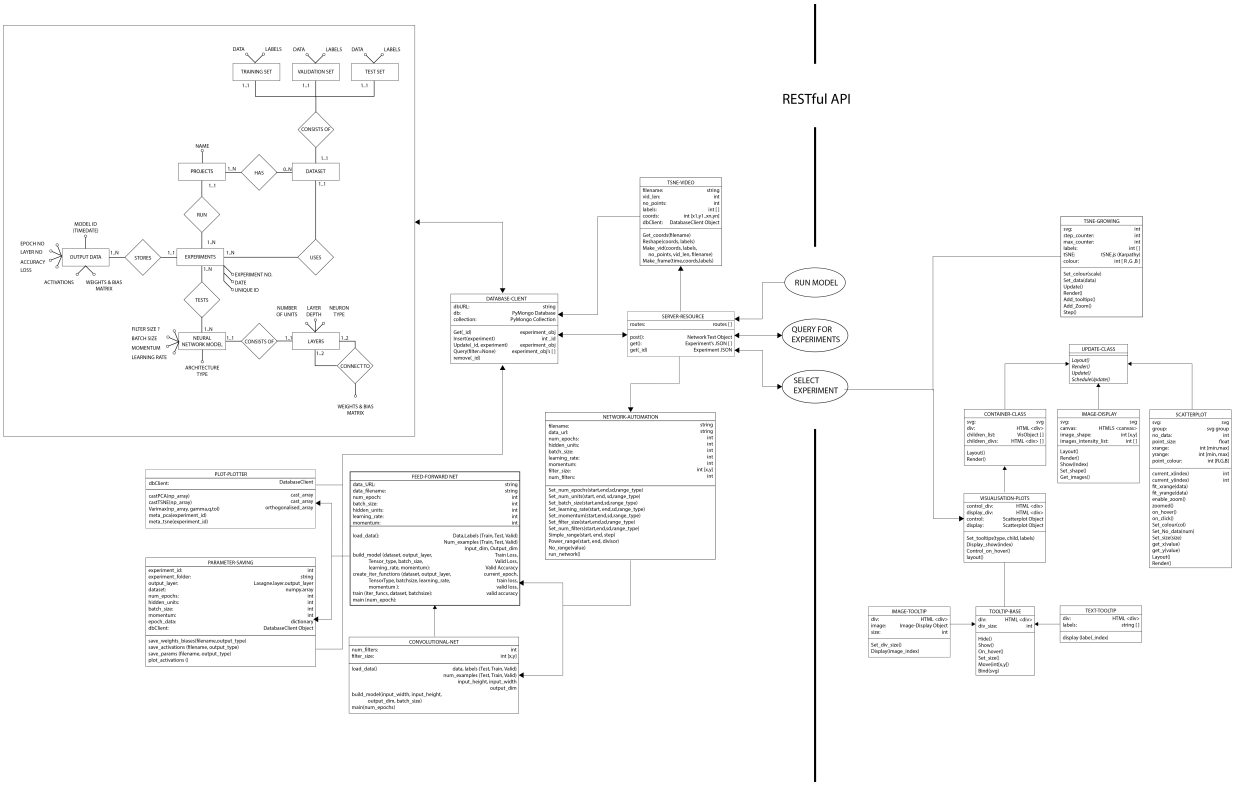
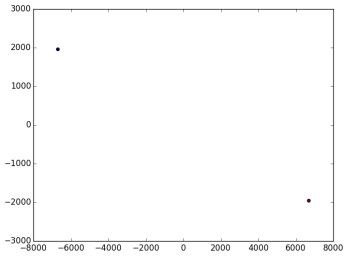
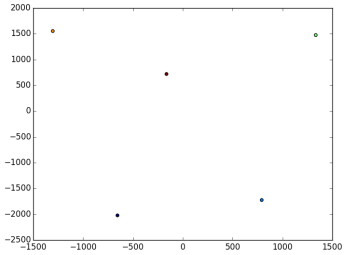


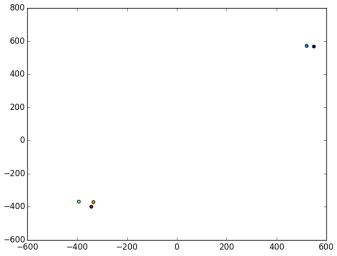
Figure 54: UML, ER, Scheme diagram hybrid



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



(b) Barnes Hut SNE after applying Principle Components Analysis



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

Figure 55

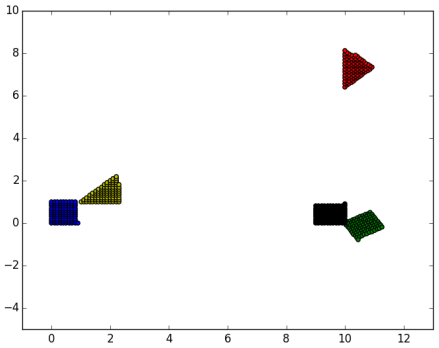
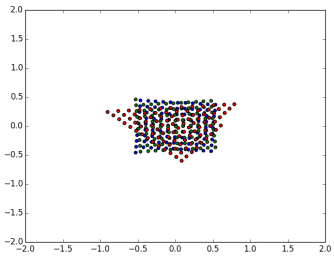
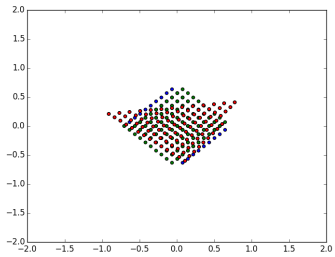


Figure 56: The test dataset



(a) Application of Principle Component Analysis



(b) Application of Varimax Orthogonalisation

Figure 57

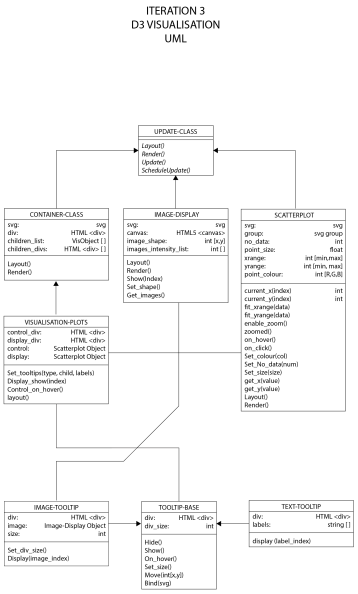


Figure 58: UML, ER, Scheme diagram hybrid

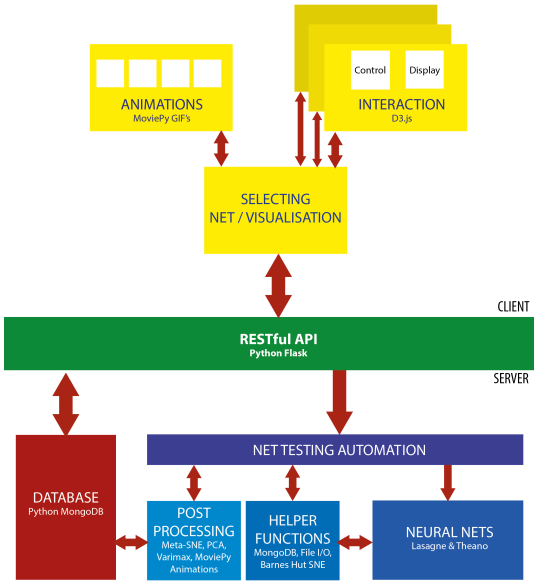


Figure 59: The architecture behind the whole system

10.2 Analysis: Neural Network Response

11 Conclusions and Future Work

11.1 Expanding the Automatic Neural Network

11.2 Widening the Visualisation Toolbox

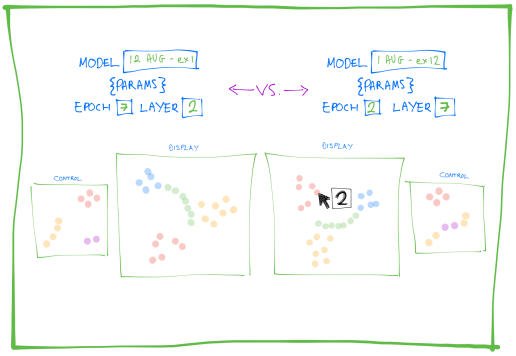


Figure 60: Wireframe Design

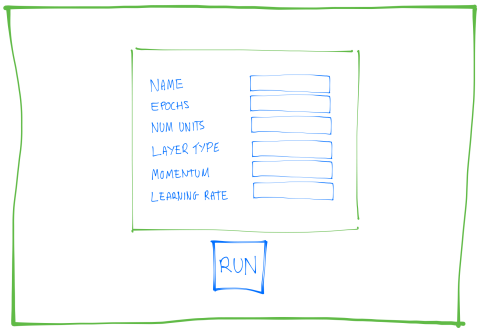


Figure 61: Wireframe Design

11.3 Using different Visualisation UI techniques

11.4 Adapting an API for other Neural Network Packages

12 Structure

Which Section Question / Implementation Goal My Solution / Design What did I need to Explore Theory or Software What did I try, Implementation How successful / unsuccessful was it? - pro's, con's, comparison Significance to main goal Outcome of the exploration, and significant questions left to answer What to explore next?

13 Progress Summary

13.1 Investigation and Data Collection

13.1.1 Survey

Deep Neural Networks sometimes contain hundreds of parameters which one can tweak, and a vast array of elements that may be added or subtracted from the standard network architectures described earlier.

While it would be great to visualise everything in this project - unfortunately this isn't feasible, and so in order to establish which areas to start on, a survey has been created to distribute amongst the vast deep learning community.

There are three components to the survey:

- **Working Environment:** In order to assess which areas would be most effective in terms of increasing efficiency of work, the aim is to find out which broad areas of working with DNNs take up most time.

  To ensure that any tools developed throughout this project are suitable to both the academic and practitioner communities, the survey aims to find out which packages and languages people are most familiar with.

- **Training Methods** In order to develop visualisations that are immediately useful, the survey aims to find out what the most frustrating problems are for the researcher, and to avoid duplication of what exists already - ask if they have found effective solutions to these problems.

  Two other questions aim to diagnose which parameters are A most important to producing a working network, and B most commonly tweaked.

- **Visualisation** The section begins by showing five images of DNNs being visualised in a variety of different ways, with the aim of clarifying any misunderstandings about what it means to visualise these networks. As a bonus, the survey uses these to deduce which appear to be most useful - each is in it's own distinct category of visualisation.

  Having established what visualisations may be possible, the survey continues to ask more focussed questions aimed at understanding if people have had prior experience with visualisation, where and why they think they would use it, and how they would like to interact with it if such a thing existed. This information is both explicitly asked for an implicitly deduced by a series of free-text and selection questions.

13.2 ANN Visualisation: Representations

13.2.1 Overview

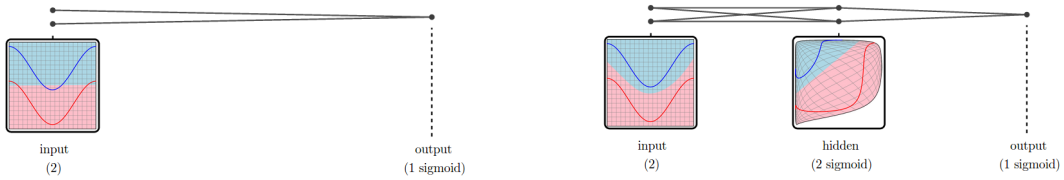
Dimensionality reduction techniques provide useful tools for visualising data with greater than three dimensions - most neural networks. As seen above, they allow the user to explore both global and local patterns within their data visually, providing perhaps previously unseen insights into their data-set and in it's manipulations.

The structure that dimensionality reduction tries to preserve when translating to two, or three, dimensional space can be seen as a higher-order *representation* of the data. 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). Therefore exploring these representations from a visual perspective might help researchers inspected and explore this space and it's here that they are likely to see which features are contributing to the learning, which intermediate concepts - such as higher-level features - are being created by the hidden layers. Importantly, these representations are likely to be distributed (Hinton 1986) such that each concept is encoded in the activations of any number of the networks nodes, making understanding these concepts a greater problem than simply understanding the decision surface on singular neurons, but one that requires representations across all nodes. Ultimately understanding these better should provide a method to guide the training process that is less situated in trial and error.

13.2.2 Space Transformation

Representations are created to perform easier classification in the latter stages of a neural network. In order for the a network to classify the points as belonging to either one or the other it must seek to find a linear separation of the two (Olah 2014*b*).

In the input space the network requires a relatively complex line to divide two curves on a plane. However 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 62: 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 and this can be seen above as well. 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).

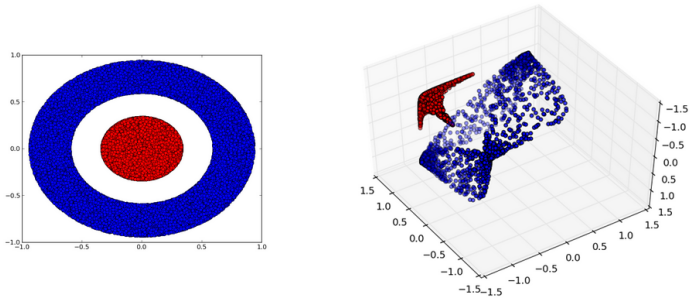


Figure 63: Three Node Warping

Another example, is one that cannot be warped simply in two dimensions, but requires a third, such as a circle within a circle:

$$A = \ xld(x, 0) < 1/3 \ B = \ xl2/3 < d(x, 0) < 1$$

It is impossible for a neural network to classify this without having a layer with greater than 3 hidden neurons. The requirement of the network to find a hyperplane that separates A and B in some final representation will not be possible no matter how the space is warped - the network requires an extra dimension. Visualisation demonstrates the network struggle to perform this. However, if we add a third neuron, the problem becomes trivial - with a three dimensional representation of the data. This spatial transformation occurs in even more complex datasets with numerous dimensions (Carlsson et al. 2008) such as images. however, while it is less easy to visualise these the intuition is useful and may lead to discovering appropriate ways of showing the same transformations in multi-dimensional space.

13.2.3 Representation of word embeddings

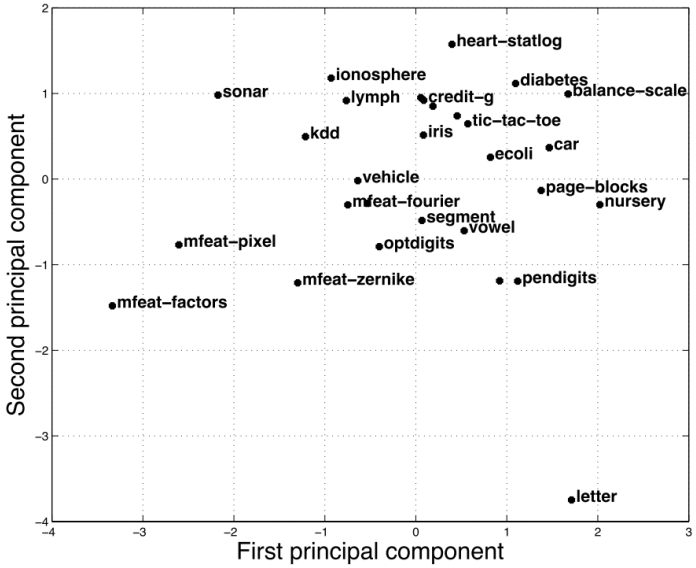


Figure 64

A word embedding $\boldsymbol{W}$ **:** $\boldsymbol{words} \rightarrow \mathbb{R}^{\boldsymbol{n}}$ is a parametrized function that maps words to high-dimensional vectors (Bengio et al. 2003). If these are then passed through a learned representation $\mathbb{R}$ of word-space we can classify the words.

In a word-embedding when you switch a word for its synonym or for another within its class - "a few people sing well" versus "a couple of people sing badly" - then while it appears the input has changed a lot, if $\boldsymbol{W}$ maps synonyms (few $\rightarrow$ couple) and classes (well $\rightarrow$ badly) close together, then from the perspective in the representation $\mathbb{R}$ very little actually changes.

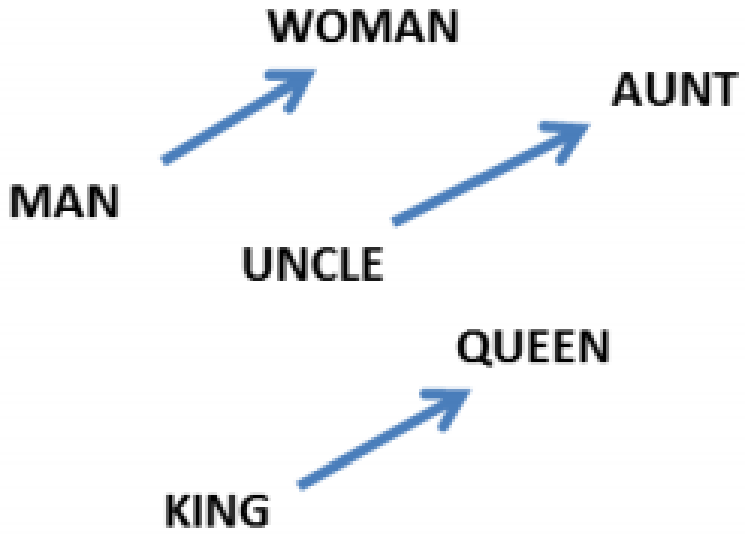


Figure 65

One way to get a feel for this word embedding is by using t-SNE to visualise the data. This displays words that are similar close together. The words appear to have have a physical representation - here analogies between words are encoded in the vector difference between words (Olah 2014*a*), for example:

$$W(\text{“}woman\text{”}) - W(\text{“}man\text{”}) \simeq W(\text{“}aunt\text{”}) - W(\text{“}uncle\text{”}) \simeq W(\text{“}queen\text{”}) - W(\text{“}king\text{”})$$

The intuition here is that the word embedding has learnt to categorise gender consistently, and it's clear to see that the model has likely learnt a gender representation.

Translation from English into French sentences is achieved by understanding these representation's within two recurrent neural networks (Sutskever et al. 2014) . The first processes the English, word by word, to produce a representation of it. The second takes the representation of the English sentence and sequentially outputs the translated words.

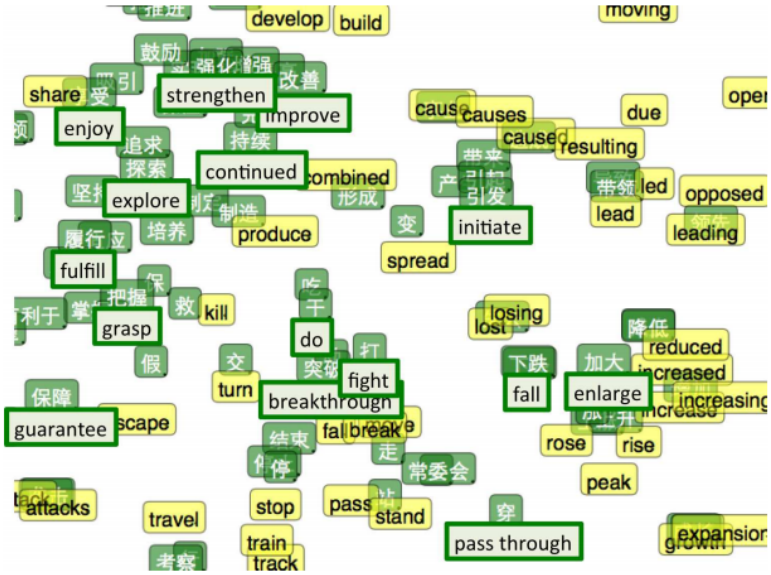


Figure 66: English and Chinese word representation plot

An interesting discovery made possible by the visualisation of this system, is that the representation taken at the intersection of the two languages was heavily dominated by the first word of the sentence. Spotting this would have been near impossible in other non-visual depictions of the data, and it allows certain theories to be drawn about what the network is actually doing in order to process the information correctly. (Olah 2014*d*) notes a number of possible deductions from this information.

13.2.4 Hidden Layer Representations

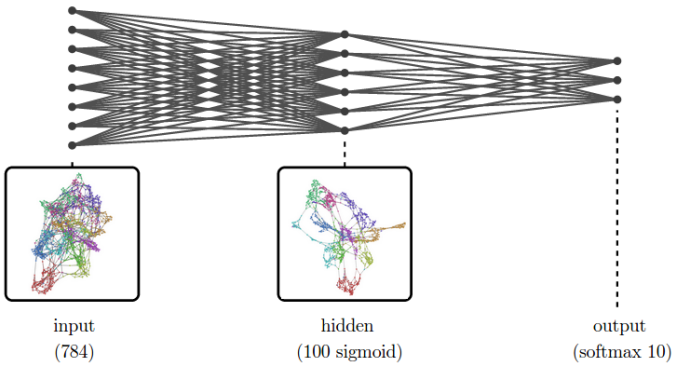


Figure 67: Force Directed Graph at input and hidden representation

Another level of insights provided by representations can be attained by comparing representation's across layers. One interesting visual example of a representation, produced by (Olah 2014*d*), is of the MNIST dataset. A nearest neighbour, force-directed graph that is used to show classes which are fairly tangled and chaotic at the input, where it's easy to assume little classification. However by the next layer because the model has already been trained to distinguish digit classes, the hidden layer

has learned to transform the data into an alternate representation that is easier to classify, and easier for a researcher to discern if the classification is performing as expected.

13.2.5 Transfer Function Representations

In addition to examining the representations at any given layer, it's possible to compare representations provided by different transfer functions.

Each Neuron warps the space it interacts in rather differently. Using Principle Component Analysis as a dimensionality reduction technique, it becomes easy to understand this deformation of input space.

With a five unit sigmoid layer projected into three dimensions using PCA, its clear to see that the representation is very much like a cube. This intuitively makes sense, as sigmoid units tend to give values close to 0 or 1, and less frequently produce values in the middle. If the transformation is performed across the five dimensions with the sigmoid layer, then there ends up being a concentration at the corners and edges - thus creating a high-dimensional cube.

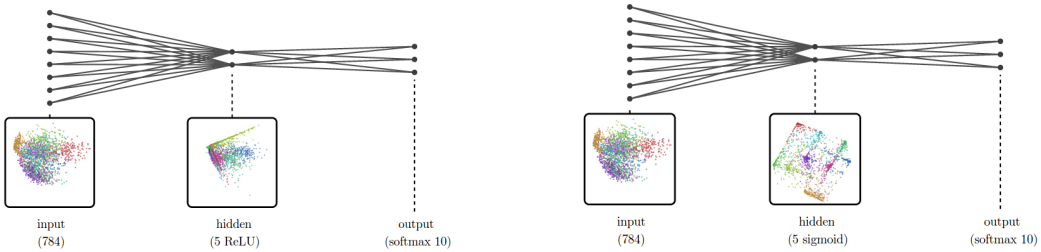


Figure 68: ReLU neuron versus Sigmoid neuron representation

If PCA is applied to a five unit ReLU layer, then a different geometric fingerprint is seen. The ReLU has a high probability of having zero activations - resulting in lots of points tending to the origin, or along the axes. Again in high-dimensions, this takes a physical representation and looks like a series of spikes originating from zero.

The interesting point to note is that very quickly it becomes possible to come to intuitive conclusions about how our data is being manipulated by the different functions. Far clearer certainly than if we were to simply look at the weights, activations and biases on a spreadsheet.

13.2.6 Isometries in Representations

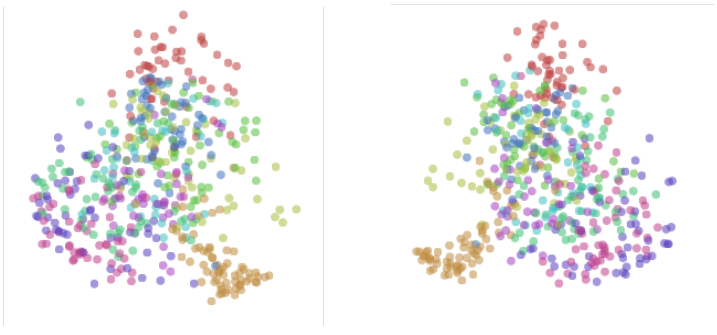


Figure 69: A flipped representation

Visualised representations effectively form a geometric footprint of the transformed data. This is not the same every time we train the network and can change depending on small variables. This makes it likely that we could end up with several minor variations of the same dataset.

Sometimes a different representation means something significant, like learning a new characteristic of the data, and at other times the new representation is simply an insignificant transformation in isometries, like rotation or flipping - where nothing new is really learnt. It's important to reduce the chances of seeing these isometries as they can confuse what experts learn from the data.

What is required is a form of representation that encodes only meaningful differences. In dimensionality reductions, such as PCA or t-SNE, we are primarily concerned with distance between points as this holds the important notion of similarity and difference.

(Olah 2014*d*) states 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 with removal of isometries required.

The issue with d_x however is that it is a function on a very high-dimensional continuous space, caused by the need to consider the distance between functions as infinite dimension vectors.

$$D_X = \begin{bmatrix} d_X(x_0, x_0) & d_X(x_1, x_0) & d_X(x_2, x_0) & ... \\ d_X(x_0, x_1) & d_X(x_1, x_1) & d_X(x_2, x_1) & ... \\ d_X(x_0, x_2) & d_X(x_1, x_2) & d_X(x_2, x_2) & ... \\ ... & ... & ... & ... \end{bmatrix}$$

Here we require another application of dimensionality reduction - again reapplying t-SNE, PCA or some other technique. *meta-SNE* is a recently introduced variation of t-SNE by (Olah 2014*d*) that performs the above flattening of distance matrices. This meta-SNE visualisation of distance shows how much representations disagree about which points are similar and which are different - allowing us to have quick overview of when a network has learnt an entirely new representation or not. This moves the position up from looking at representations, to the space of representations.

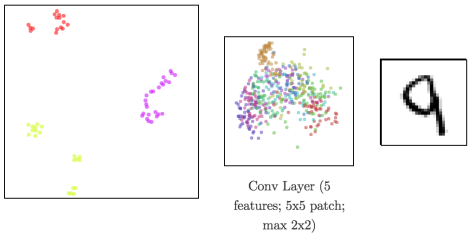


Figure 70: Meta-SNE representation → t-SNE representation → MNIST digit

It is possible here that this space could be used to see how current model representations compare to other 'landmark' representations from past experiments. If the models first layer representation is in the same place as a really successful model, then that's likely to be positive. If however it's tending towards a cluster that had some specific poor quality, the researcher would know to adjust for this too. This provides us with some qualitative feedback during the training of the neural network.

13.2.7 New Visual Encodings for Deep Learning (REJIG TO MAKE MORE ME!!!)

Figure 71: MNIST embedded digit plot

Figure 72: Ones visualised by tSNE

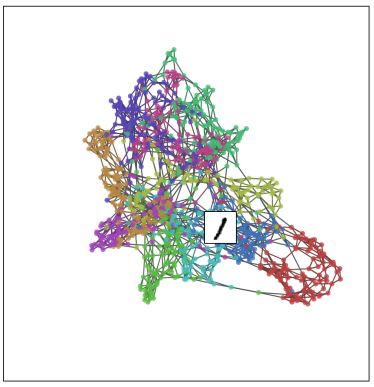


Figure 73: Three Nearest Neighbour Graph

While there are a number of established best practices for visualising low dimensional data as explored above, many of these simply don't work when it comes to exploring neural networks - which are typically multi-dimensional. Labelling axes quickly becomes ridiculous when multiplied by 10,000 variables. Giving units when comparing very different types of data under one visual representation becomes equality redundant.

It is important to recognise the fundamentals learnt from two dimensional visualisations, and extrapolate them when applying to more complex data sets.

(Olah 2014*d*) suggests a couple of principles to consider when visualising high-dimensional data that at first seem obvious, but in practice are rather hard to achieve:

- There must be a way to interrogate individual data points
- There must be a way to get a high-level view of the data

One way to encode the data such that these rules are met is to make the visualisations interactive and allow the viewer to zoom in for detail, or expand out for a high-level view.

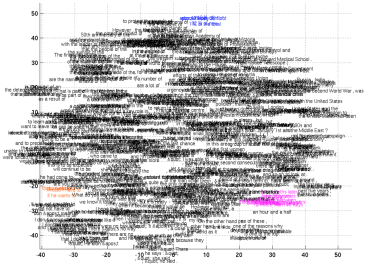


Figure 74: Phrase embedded plot

Interaction isn't always necessary however, and some attempts have been made to show data in a flat two dimensional representation using dimension reduction, which will be explored in depth a little later. The plot of the MNIST data set, a set of handwritten digits from one to nine, by (Maaten & Hinton 2008) provides a clear non-interactive view of the data where spotting patterns such as the angle deformation of the '1' characters across a class clustering, or spotting simple misclassification becomes easy.

As with exploring two dimensional data its important to remember that there is no one rule that fits all. A less successful example of embedding data within the plot itself is by (Cho et al. 2014) who attempt to visualise phrases. Here you can see that the data become messy incredibly quickly, and that perhaps interaction would be a better method of ensuring we retain Olahs principles.

Providing a user with the tools to control the data being visualised is incredibly important in engaging the user in the discovery process. The user must be able to change important network parameters, and immediately see the effects of such a change. They must also be able to compare and

contrast different portions of the data through selection, and control the rate at which this change in information is depicted so as to allow them to discover patterns for themselves.

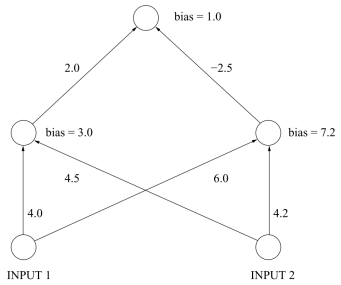


Figure 75: Simple Neural Network

A compelling argument is made by for interaction when exploring scientific data visually. He shows that there is often a situation where the data is so dense, where there is simply too much to explore in an effective way, that interaction in the only solution; instead hovering over the points and being provided with a tool-tip that demonstrates the points value. Interactive filtering can help, allowing the user to choose some number of easy to visualise classes.

13.2.8 Categorising and Understanding Academic Visualisations

One area within the field of Deep Learning that uses vast amounts of visual material, if aggregated, is across the academic body of research.

Graphs, charts, 3D planar surfaces, diagrams, morphed images, and more, all contribute a significant amount to helping readers understand, and writers explain new concepts and cutting edge research. It seems then, to be a good place to start examining if one wants to understand the types of things the community chooses to remove from mathematical syntax, and place in a visual form.

The body of visualisations collected is approximately 200, and growing. Each visualisation is categorised under the following headings:

- **The type of visualisation:** This information will provide a useful data point about the types of visualisation most understood and favoured by the community. This will make it far easier to produce visualisations that have the right level of explanation required to make any new visualisation types easy to process.
- **Any comparisons made:** This will provide invaluable information about which parameters researchers most often use to make decisions about performance, and ultimately lead to change in their models. Understanding this will help to ensure that visualisations produced for the case-study experiment are not 'overfitting' to the case-study, and are actually still useful to the community as a whole.
- **Any axis-labelling, or annotations:** Similar to above, this will provide an understanding of the features and scales that the community most values. For example, error rate as a percentage features heavily against time - but also occasionally features against other variables. Again, this provides a useful reference point as to what the community of researchers is most interested in, and allows this project to progress without needed to be an expert in the field.

Please see appendix A

13.2.9 Collecting Iterative Visualisations: Sketches

The process of acquiring this information has only just begun, however it will provide an invaluable insight into forms of visualisation that are not confined to those that have been iterated upon and refined for the purpose of publication.

The data collected in this research are sketches, quick diagrams and 'hacky' visualisations made by software - ideally accompanied by some description of what the researcher was trying to explain, or understand.

With this information, it will become far clearer what is required in terms of content for visualisations made for understanding and exploration as opposed to visualisations made for explanation. Ultimately this information should allow visualisation to be targeted towards helping researchers make key decisions.

13.3 Understanding Literature

Thus far; a number of papers have been read, a number of tutorials undertaken, and a number of online lectures watched - both in the discipline of visualisation and in working with deep neural networks.

13.4 Clarifying Goals

The goals set out when proposing this project appeared to be quite clear. However, as with any project, the more understanding you have of a topic - the more you realise you didn't understand before. This has been made incredibly clear, and even early research into what would be valuable for those implementing deep neural networks has changed the course of how this project will work. Hopefully it is more on the right tracks now.

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&a &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+d*

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

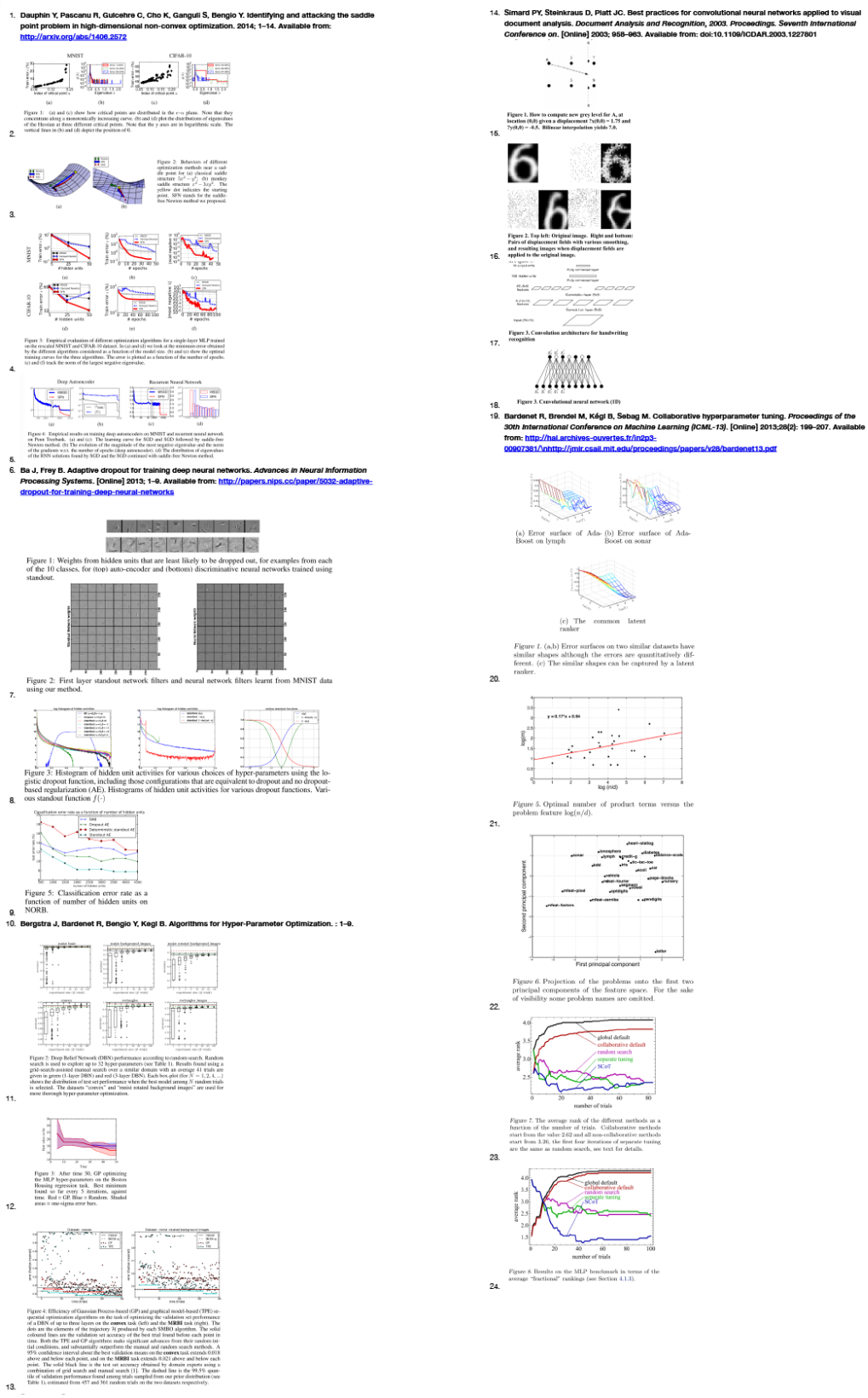


Figure 76: Sample of Classifying Image Data

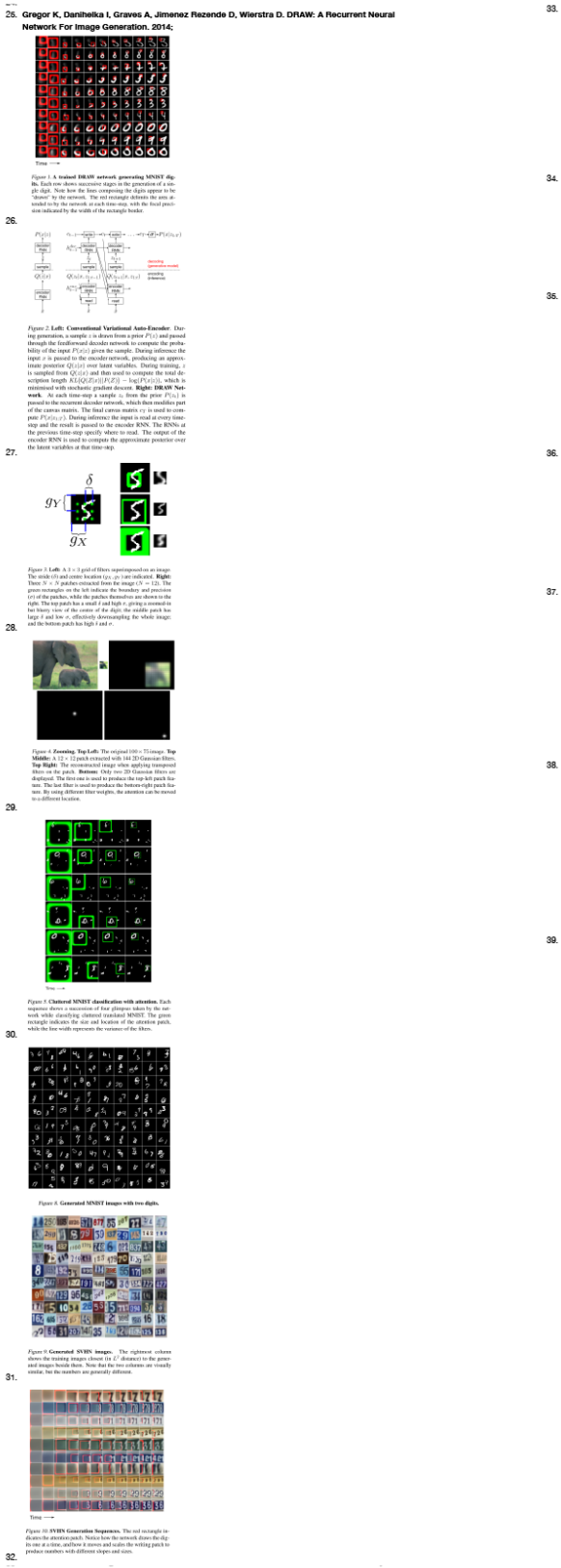


Figure 77: Sample of Classifying Image Data

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 Classification 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 78: Sample of Analysis of Image Data