VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Learning Logical Activations in Neural Networks

Jack Laurenson

Supervisor: Marcus Frean

20th October 2016

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

With the advent of deep learning, neural networks have enjoyed spectacular success in recent years across a broad a range of classification problems. A crucial component of neural networks is the activation function at each neuron. Popular activation functions such as the logistic sigmoid, hyperbolic tangent, or rectified linear unit have been employed with varying success in classification. However, given a fully trained model, there is no clear interpretation of what the weighted sums on these activations represent. That is, despite their outstanding success, neural networks still remain a black box with respect to intelligibility. This project proposes two new types of activation functions inspired by boolean logic gates, these are called the Noisy-OR and Noisy-AND activation functions. We show that networks containing logical activations achieve performance comparable to standard multi-layer-perceptrons, demonstrate how their weights can be interpreted meaningfully in a fully trained neural network and elucidate the benefits of this interpretation.

1 Introduction

This report begins with a brief introduction to neural networks in section 2. We provides a summary of the literature regarding activation functions. When referring to our newly proposed functions, we use the term logical activation. All other activations are referred to as standard activations. Regarding the network as a whole, this report will consider Multi-Layer-Perceptrons (MLP's, introduced in detail below) a conventional architecture for feedforward neural networks. MLP's that contain purely standard activations will be referred to as standard MLP's, whereas MLP's containing logical activations are referred to as Logical Neural Networks (LNN's). We make the distinction between pure LNN's which only contain logical activations and mixed LNN's which contain standard and logical activations.

Judea Pearl introduces the concept of the *Noisy-OR* relation [1] to represent non-deterministic logical relationships in Bayesian Networks [2]. Pearl demonstrates how probabilities can represent the relevance of an input (or underlying variable) with noise, i.e. some uncertainty regarding that inputs relevance to the output. In the second part of section 2, we elaborate on Pearl's work, explaining the necessary background to discuss noisy activations.

Section 3 outlines our models for a logical activation functions, which are inspired by binary logical gates and the Noisy-OR relation. Firstly, we present each model in terms of the noise of each input, i.e. the uncertainty in the relevance of each input. We then propose a re-parameterisation in terms of weights and biases that is appropriate for deployment in a Neural Network. Our main goal in this paper is to demonstrate that for these activations, learning is feasible in the context of stochastic gradient descent and furthermore, that these model lead to a more interpretable neural network.

Section 4 compares the performance of LNN's with standard MLP's to demonstrate their learning capacity. We compare a range of architectures at different depths, testing both mixed and pure LNN's. Each network is trained and evaluated on MNIST [3], a handwritten digit classification problem. We suggest MNIST is a good data set for evaluating whether or not learning is feasible and evaluating the interpretability of a neural network.

Firstly, this is because MNIST is a non-trivial problem, it cannot be solved accurately by a simple classifier such as Naive Bayes [4] or KNN [5]. Therefore demonstrating that LNN's have comparable performance to standard MLP's on this problem shows that LNN's can learn complicated relationships. Secondly, digit recognition provides an excellent benchmark for testing interpretability. That is, as humans we are acutely aware of how the various substructures within a 6, combine to make a 6. So we can verify the interpretability of an architecture, by examining the weights of the network and investigating whether these substructures exist.

Section 5 takes a detailed look at the interpretability of MLP's demonstrating the superior interpretability of pure LNN models. We present an in-depth theory for why the optimal weights of a logical activation have an interpretable structure. For fully trained LNN's and standard MLP's, we compare the lower level features formed after training. Section 5 then proceeds to demonstrate how a pure LNN can be interpreted and how this method is not applicable to a standard MLP. We conclude this section but elucidating the benefits of interpreting neural networks, highlighting why further research with respect to logical activations is important.

2 Background

2.1 Neural Network Introduction

This introduction is not intended to be a completely general description of Neural Networks (NN's) as applied to image classification, nor is it designed to fully instruct the reader unfamiliar with NN's. The main goal of this introduction is to set up a context in which different architectures and activation functions may be described.

A neural network is a function approximator. Consider the MNIST dataset, a single image can be expressed as $I_i = (X_i, Y_i)$ where X_i is the vector of pixels for image i and Y_i is the true digit class (e.g. 8). The problem can be described as finding a correct function $F(X)$ from the pixels (inputs) to the digit class (output). An NN is trained in a supervised learning context to recognise unseen digits, that is, approximate the true pixel-to-digit class function.

A feed-forward Multi Layer Perceptron (MLP) is composed of sequential layers of nodes called neurons, with a total number of layers L. Specifically, an input layer is connected to one or more hidden layers, followed by an output layer. For a given image X_i, the input layer merely outputs the pixels, and each sequential layer performs a computation on the preceding layers outputs, where the output layer performs the final computation and returns the approximation for Y_i, denoted as a_L.

The neurons are *connected* in the sense that that a neuron in layer l receives a weighted sum of the outputs in layer $l-1$:

$$z_l = W_l a_{l-1} + b_l \tag{1}$$

where z_l is the vector of inputs to layer l and a_{l-1} is the vector of outputs (activation) from the previous layer. The matrix of weights W_l and vector of biases b_l are learnable parameters, which may be updated to produce a better approximation with a learning algorithm.

Each neuron has an activation function, which is a nonlinear function of z_l shown below:

$$\begin{aligned} a_l &= \phi(z_l) \\ &= \phi(W_l a_{l-1} + b_l) \end{aligned} \tag{2}$$

where ϕ is the activation function, and a_l is the vector of outputs (activations) that will be fed forward to the next layer.

For classification problems, the MLP is designed such that its output vector a_L, contains elements a_{Li} where $\forall i, 0 \leq a_{Li} \leq 1$. Each Y_i is encoded as a one-hot vector, with the number of elements equal to the number of classes. This way, the network can train to produce a_L's that are as close as possible to each Y_i. One-hot encoding is used because it is much easier to learn then using a single scalar integer to represent the predicted class [6].

That concludes the description of the structure of a neural network. However, we must have an appropriate choice of W and b for the network to be a useful predictor of digit class. A learning

algorithm is used to iteratively updated W and b such that the network becomes a better predictor. NN's are trained with an algorithm called back-propagation [7] (backProp) using stochastic gradient descent (SGD) [8].

To train an algorithm using SGD, we require a differentiable function that measures the quality of our approximation, i.e. a differentiable function that quantifies how good (or how bad) our MLP's predictive power is. For this report we will use the cross entropy loss function shown in equation 3:

$$E(W,b) = \sum_{j=1}^{m}\sum_{i=1}^{d}(Y_{ij}log(a_{Lij}) - (1-Y_i)log(1-a_{Lij})) \tag{3}$$

where $E(W,b)$ is the loss, with respect to all weights and biases of the network. m is the number of samples in the current mini-batch, while d is the number of classes/outputs. We optimise W and b by minimising $E(W,b)$. A perfect classification would result in a loss of zero, where for each $Y_{ij} = 1$ implies $a_{Lij} = 1$ and for each $Y_{ij} = 0$ implies $a_{Lij} = 0$.

Stochastic gradient descent is a very general algorithm for finding optimal values for a set of parameters θ. It may be applied to non-convex functions and is particularly useful if the exact solution is analytically or computationally intractable. In the context of a neural network, SGD is applied to optimise the set of weight matrices and bias vectors by minimising $E(W,b)$. The standard SGD update rule for a given weights matrix is shown below:

$$W_l = W_l - \lambda \bigtriangledown E_l \tag{4}$$

$\bigtriangledown E_l$ is the maximum directional derivative, i.e. the direction in which the gradient of $E(W,b)$ is the steepest with respect to W_l. λ is known as the *learning rate*, a scalar constant which determines the degree to which the weights decrease in the direction of $\bigtriangledown E_l$. $\bigtriangledown E_l$ is determined by back-propagation, a technique that computes the gradients of one layer by applying the chain rule using gradients of the previous layer. Without backProp, training MLP's would be computationally prohibitive.

SGD is stochastic, in the sense that rather than computing the gradients based on all training observations at once, they are computed for random mini-batches of the input. Matrix multiplications have a complexity of $\mathcal{O}(mp_{l-1}p_l)$ where m is the mini-batch size, and p_{l-1} and p_l are successive layers in an MLP. BackProp is the most expensive part of training a network and seeing as it is comprised primarily of matrix multiplications it pays to keep the mini-batch size m small. Furthermore, choosing a smaller mini-batch allows for quicker convergence and SGD tends to achieve higher classification accuracies than gradient descent, which uses the entire training set each batch [8].

A neural network is considered *deep* when it has many hidden layers, traditionally this is more than about 3 or 4 layers. However recent advancements successfully employ up to 152 hidden layers [9]. Deep Learning achieves better results than shallow networks on image classification datasets and has been shown to avoid local optima better than shallow networks do.[10].

2.2 Summary of Activation Functions

Introducing a nonlinear activation function at each neuron allows a network to approximate a wide range of possible functions, such as a function from an image's pixels to its digit class. This section describes the activation functions that are commonly applied in the NN literature, presenting the theoretical and empirical rational for using these functions.

Sigmoid

When training a neural network with back-propagation was first shown to be feasible [7] the activation function used was a sigmoid, the logistic function:

$$a_l = \frac{1}{1 + e^{-z_l}} \tag{5}$$

The sigmoid neuron is loosely speaking, biologically inspired. A biological neuron will either fire or not fire a signal to another neuron based on its incoming signals. A sigmoid function models this behaviour by approximating the step function, particularly for large for large elements of W.

Hyperbolic Tangent

A second commonly applied activation is the hyperbolic tangent function:

$$a_l = tanh(z_l) \tag{6}$$

Both the logistic sigmoid and the hyperbolic tangent have problems with saturating gradients, where for the extreme values of the gradients are low and therefore training is very slow. Begnio et al, 2010 [11] demonstrate that on image classification problems such as MNIST and CIFAR-10, a logistic sigmoid is more susceptible to saturation than the hyperbolic tangent, particularly as the number of layers in the network is increased. The hyperbolic tangent function has a range $-1 < a < 1$, and is only flat when $|a| \approx 1$. Hence, the gradients for W in a tanh neuron are not susceptible to saturation at $a \approx 0$ unlike the logistic sigmoid.

Rectified Linear Unit

However, Krizhevsky et al, 2012 [12] popularised a radically different kind of activation function, the Rectified Linear Unit (ReLU) [13]. Krizhevsky et al, shattered previous performance bench marks on the ImageNet dataset [14] by employing rectified linear units in a deep convolutional neural network. The ReLU activation function is shown below:

$$a_l = max(0, z_l) \tag{7}$$

Presently, ReLU's are the most popular activation function achieving the best performance across image classification data sets [9, 15, 16]. The success of ReLU's is partially attributed to the fact

that unlike their sigmoid counterparts, their gradients do not saturate [17]. The gradient of a ReLU is always one when $z > 0$, which implies no slow down in training. Indeed Krizhevsky et al, 2012 [12] demonstrate that a network composed of ReLU's trains several times faster then a tanh neuron on the CIFAR-10 image data set.

Softmax

The softmax activation [18] is used in the final layer of a neural network for classification problems:

$$a_{Li} = \frac{e^{z_i}}{\sum_{j=1}^{p} e^{z_j}} \tag{8}$$

for all p elements i of the output vector a_L. Hence p is the number of output nodes and each z_j is the weighted sum at each of these nodes. The softmax is useful because it transforms vectors over the entire range of the reals to the range $0 \leq a_L \leq 1$. Furthermore, if a given z_i is greater than all other z_j's then a_{Li} will be exponentially larger than each a_{Lj}, which helps when training one hot encoded outputs [19].

For each activation function described above (aside from the softmax), it has been proved that a neural network composed of such functions is a universal function approximator. That is, with a single hidden layer, a neural network made up of one of these activations functions, can approximate any continuous function (where increasing the number of hidden units reduces the error of the approximation). Cybenko, 1989 first showed this result for a neural network composed of logistic sigmoids [20]. Hornik, 1991 introduced more general conditions which an activation must satisfy to serve as a universal approximator and noted that the multi-layered architecture of an NN is what allows for universality [21].

Knowing that a neural network can approximate any continuous function given the correct set of weights and biases is promising, however the challenge lies in choosing an activation function that allows a learning algorithm such as backProp with SGD to learn the correct set of weights and biases.

There is no hard theory of activation functions, no mathematical proof that demonstrates why ReLU's are superior when applied to image classification. As it stands, the main reason for the popularity of ReLU's is their strong performance on image classification data sets.

2.3 Bayesian Networks

Bayesian Networks are a type of probabilistic graphical model that formulate a principled way of reasoning about an uncertain real world. A Bayesian network is an agent that consists of a predefined directed acyclic graph (DAG) where each node represents some underlying feature of the real world. Consider the following example construction, if the agent's conception of reality is the (28 *28) grid of possible pixels of an MNIST image, two nodes could be the pixels at (3, 4) and (5, 2). These pixels are parent nodes that could be connected to a child node that represents a higher level feature which in turn could be connected to a node that represent the presence of a particular digit.

The edges exist between these nodes by construction, but the probabilities on these edges are trained using Belief Propagation [22]. These probabilities generally represent the conditional dependence of one node on another. While obtaining such a structure would be richly interpretable, in practice the training Bayesian networks does not scale well because the number of parameters they have is exponential in the number of parents nodes. Given that a Bayesian Network applied to MNIST would consist of at least 784 pixels (potential parent nodes), using one for this problem is infeasible.

Noisy-OR

One particular variant of the relation between a node and its parent nodes is termed the Noisy-OR relation [4], developed by Judea Pearl. Consider a binary node D in a Bayesian network, with three binary parent nodes, S_1, S_2 and S_3. We prescribe the following structure to the network, D will be present if S_1 OR S_2 OR S_3 present, in the logical sense. However, we specify some uncertainty ε_i regarding whether S_i influences D. Assuming independence of predecessor nodes, we can express the probability$D = 1$ as follows:

$$P(D = 1|S_1, S_2, S_3) = 1 - \prod_{i=1}^{3} \varepsilon_i \tag{9}$$

In the following section, we describe an extension of Pearl's Noisy-OR applied in MLP's. Rather than predefining the structure of nodes, we learn that structure using SGD and backProp. Furthermore, we generalise Pearl's Noisy-OR by considering parent nodes as probabilities, rather than binary nodes. In section 5, we proceed to show that networks containing such relations are interpretable, like Bayesian Networks are.

3 Logical Activation Functions

3.1 Motivation

Deep convolutional neural networks are theorised work by automatically extracting localised features from an image at the base layer, and building higher level abstractions of these features at preceding layers [23]. We hypothesise that an image's class could be determined by a logical combination of these higher level features. For example, consider a network trained on the MNIST data set. A high level abstraction could be F_1, a downward stroke at the top left and F_2, horizontal bar at at the bottom of the image, The presence of F_1 *and* F_2 suggest the digit 2. Figure 1 depicts this logical relationship.

Given the universal approximation status of standard MLP's and the fact that in practice such networks can achieve a very high classification accuracy on MNIST [24, 25], standard MLP's can learn this relationship. However, as we shall demonstrate in section 5, the equivalence of infinitely many optimal weight solutions makes it difficult for us to infer what relationships have been learnt. We set out to design activations which do not suffer from this equivalence and demonstrate clearly interpretable relationships whilst still achieving reasonable classification accuracy.

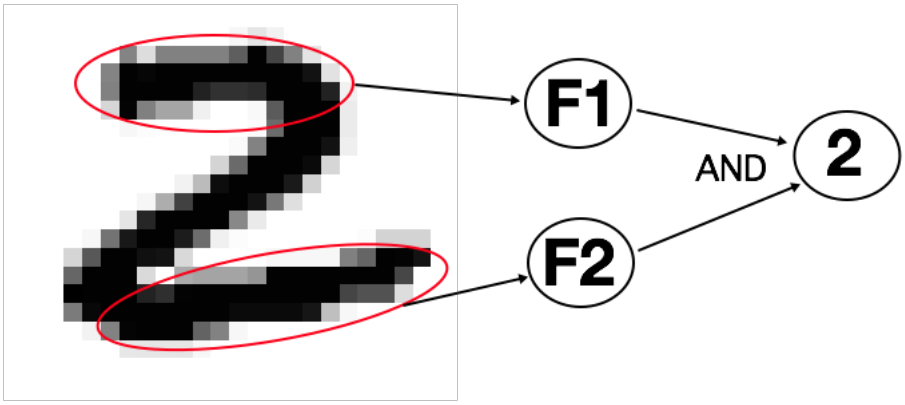

Figure 1: Illustration of a logical combination of features that might be learned by an agent. The two is taken from the MNIST data set.

In this section, we propose two activations based on logical OR and logical AND. Our motivation for networks containing these activations is based on two premises. If classification problems are well represented by logical combinations of lower level features, then a correctly parameterised logical activation should be able to capture this structure. More importantly, such a network would be more interpretable to a data scientist because OR and AND have a directly interpretable relationship with respect to their inputs.

3.2 Noisy OR

A logic gate only receives binary inputs and is non-differentiable, therefore it is unsuitable for an MLP trained with backProp. However, using Pearl's Noisy-OR we can derive a differentiable activation based on equation 9. Note that in equation 9, the errors (noises) ε_i are defined on binary nodes (inputs). To reiterate, the noise represents an uncertainty regarding a given inputs influence on the output. For example if $\varepsilon_i = 0.4$ the agent believes there is a 60% chance the input i influences its output node through a logical OR with the other inputs. Or conversely, a 40% chance that the input is irrelevant, i.e. a 40% chance the input can be ignored.

To apply Noisy-OR in an MLP, we need to generalise the model for real inputs in the range $0 \leq x_i \leq 1$. That is, we model the input nodes as probabilities, rather than binary inputs. Each x_i represents the agents belief as to the probability of the *ith* node being *on*. This means that the probability of an input node being irrelevant to the Noisy-OR relation is not only influenced by its noise ε_i, but also x_i, the probability of that input node being *on*.

Therefore we need some function of $f(\varepsilon, x)$ that describes the independent probability of irrelevance for input i, with respect to the output a_l. Such a function must satisfy the following properties:

1. $\varepsilon = 1$ implies $f(\varepsilon, x) = 1$
2. $x = 0$ implies $f(\varepsilon, x) = 1$
3. Monotonically non-decreasing in ε but monotonically non-increasing in x

That is, 1) if $\varepsilon_i = 1$ then no matter what x_i is, the input i cannot influence a_l so $f(1, x_i)$ must be 1. 2), if the input i certainly does not exist, ($x_i = 0$) then the no matter what the noise is it cannot influence a_l so $f(\varepsilon_i, 1)$ must be 1. 3) if the noise increases, or the probability that input i exists decreases, then the probability of input i's irrelevance must increase (or at least remain at 1, if it is at 1).

We propose the following function to satisfy these properties, displayed in figure 2:

$$f(\varepsilon, x) = \varepsilon^x \tag{10}$$

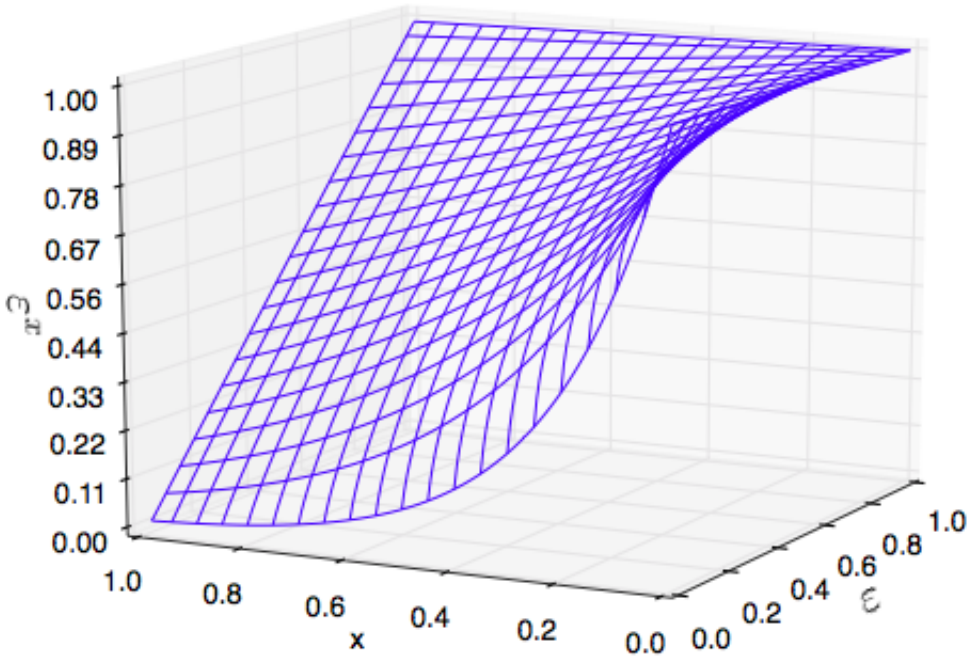

Figure 2: $f(\varepsilon, x) = \varepsilon^x$, i.e the irrelevance of a given input with respect to a Noisy-OR activation's output. It is easy to see that this function satisfies the properties described above.

We are now in a position to describe the Noisy-OR activation function. A discrete OR function takes p discrete binary inputs, and returns $a = 0$ if all the inputs are 0, or $a = 1$ otherwise. We propose the continuous function, Noisy-OR:

$$a_l = 1 - \prod_{i=1}^{p} \varepsilon_i^{x_i} \tag{11}$$

Where p is the number of inputs. x_i is the *ith* input, which is constrained to the range $0 \leq x_i \leq 1$. ε_I is the noise on the *ith* input, ε_i is constrained to $0 < \varepsilon_i \leq 1$.

Consider the case where there is no noise on the inputs, i.e. every $\varepsilon_i \to 0$ and where the inputs are binary. If any input $x_i = 1$, then $\varepsilon_i^1 \to 0$ which means the entire product in equation 6 is zero, so it

outputs 1. In this case, the only way that Noisy-OR can output 0 is if all the inputs are 0, as $\varepsilon_i^0 = 1$. Therefore, when all $\varepsilon_i^1 \to 0$ and all inputs are binary, Noisy-OR will compute a discrete logical OR gate across *all* inputs.

Suppose a given $\varepsilon_k = 1$ for an input k. For all values that input x^k can take on, $\varepsilon_i^{x_k} = 1$. That is, input x_k is ignored, it does not influence the product in equation 6. In general, increasing ε_i will increase $\varepsilon_i^{x_i}$ increasing the irrelevance of input i. Similarly the larger x_i is, the greater the certainty the network has regarding the presence of the feature x_i, and the less irrelevant input i becomes.

We will also introduce an error with no input, a bias ε_b:

$$a_l = 1 - \prod_{i=1}^{p} (\varepsilon_i^{x_i}) \varepsilon_b \tag{12}$$

To apply the Noisy-OR activation function within a neural network, we re-parameterise equation 7:

$$\begin{aligned} a_l &= 1 - \prod_{i=1}^{p} (\varepsilon_i^{x_i}) \varepsilon_b \\ &= 1 - e^{\sum_{i=1}^{p} log(\varepsilon_i) x_i + log(\varepsilon_b)} \\ &= 1 - e^{-\sum_{i=1}^{p} w_i x_i + b_i} \end{aligned} \tag{13}$$

Where $w_i = -log(\varepsilon_i)$ and $b = -log(\varepsilon_b)$, $0 < \varepsilon_i, \varepsilon_b \leq 1 \to 0 \leq w_i, b < \infty$.

Therefore the weighted sum for a Noisy-OR neuron may be expressed as $z_l = Wx + b$, where W and b are constrained to be non-negative. Hence the activation function for Noisy-OR may be expressed compactly as:

$$a_l = 1 - e^{-z_l} \tag{14}$$

3.3 Noisy AND

A discrete AND function takes p discrete binary inputs, and returns $a = 1$ if all the inputs are 1, or $a = 0$ otherwise. We propose the continuous logical AND function, Noisy-AND:

$$a_l = \prod_{i=1}^{p} (\varepsilon_i^{1-x_i}) \varepsilon_b \tag{15}$$

Where p is the number of inputs. x_i is the *ith* input, which is constrained to the range $0 \leq x_i \leq 1$. ε_I is the noise on the *ith* input, ε_i is constrained to $0 < \varepsilon_i \leq 1$

Setting $\varepsilon_b = 1$, consider the case where there is no noise on all the inputs, i.e every $\varepsilon_i \to 0$ and where the inputs are binary. If any input $x_i = 0$, then $\varepsilon_i^1 \to 0$ which means the entire product in

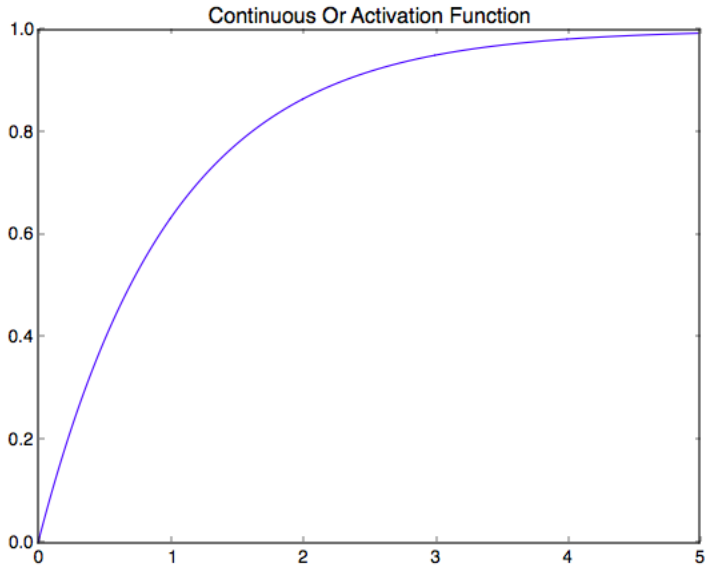

Figure 3: The Noisy-OR activation function expressed in terms of the weighted sum z_l. We see that as z_l increases, the gradient reduces at an exponential rate.

equation 7 is 0, so the function outputs 0. In this case, the only way that Noisy-AND can output 1 is if all the inputs are 1, as $\varepsilon_i^0 = 1$. Therefore, when all $\varepsilon_i^1 \to 0$ Noisy-AND will compute a discrete logical AND gate across *all* inputs.

ε_i behaves in a very similar manner to that in Noisy-OR, as ε_i increases $\varepsilon_i^{1-x_i}$ has less impact on the product and therefore less impact on the resulting activation. Furthermore, the larger x_i is, the closer $\varepsilon_i^{1-x_i}$ is to 1. That is, the larger x_i, the more input i is considered *on*.

To apply the Noisy-AND activation function within a neural network, we re-parameterise equation 10:

$$\begin{aligned} a_l &= \prod_{i=1}^{p} (\varepsilon_i^{1-x_i}) \varepsilon_b \\ &= e^{\sum_{i=1}^{p} log(\varepsilon_i)(1-x_i) + log(\varepsilon_b)} \\ &= e^{-\sum_{i=1}^{p} w_i(1-x_i) + b_i} \end{aligned} \tag{16}$$

Where $w_i = -log(\varepsilon_i)$ and $b = -log(\varepsilon_b)$, $0 < \varepsilon_i, \varepsilon_b \leq 1 \to 0 \leq w_i, b < \infty$ as with Noisy-OR.

The weighted sum for Noisy-AND is $z_l = W(1-x) + b$, where W and b are constrained to be non-negative. With this interpretation for z_l, the activation function for Noisy-AND may be expressed compactly as:

$$a_l = e^{-z_l} \tag{17}$$

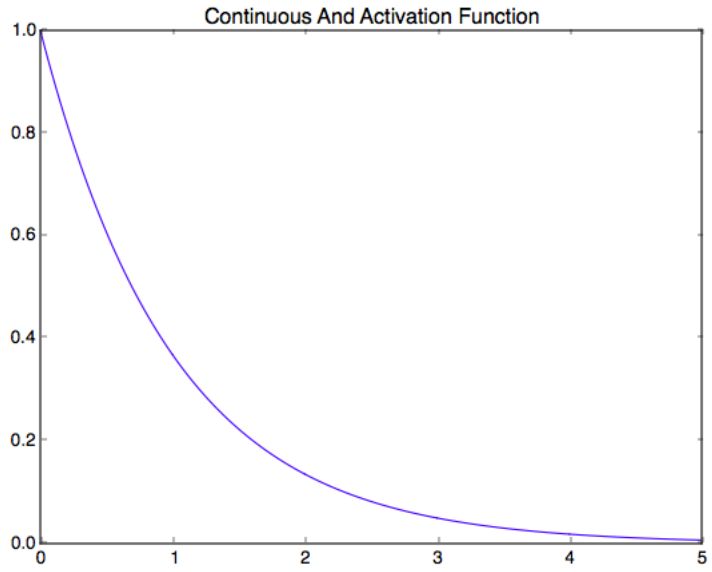

Figure 4: The Noisy-AND activation function expressed in terms of the weighted sum z_l. We see that as z_l increases, the gradient reduces at an exponential rate.

3.4 Logical Activations in Neural Networks

We consider a broad range of architectures for our logical activations. Typically, we propose placing a layer or two layers of logical activations prior to the softmax at the end of the network. Here, the network can learn to form abstract high level combinations of low level features. In the models we present, sigmoids are used at lower layers which are intended as feature extractors.

However, using networks entirely composed of pure activations function is a boon to interpretability. We propose a pure LNN where Noisy-OR neurons are applied in the first hidden layer. It is feasible to learn a logical OR of pixels, i.e. a low level feature is unlikely to exist in the same exact place each time but if it could be located at the pixel (i, j) or the nearby pixel (k, l). By contrast, a logical AND of raw pixels is difficult to learn, given the noise in the feature's exact location. Preliminary experiments using Noisy-AND neurons in the first hidden layer confirmed their infeasibility as low level feature extractors.

In our parameterisation of Noisy-OR and Noisy-AND, we have taken the care to design a function that when fully trained will exhibit sparse weight representations. Section 5 outlines why we theorise these activations will exhibit sparse weights, and provide empirical evidence to demonstrate that is the case. With respect to interpretation this sparsity is desirable. That is, we would like each neuron to learn to recognise a few import features and take a logical combination of those, whilst considering the rest of the features irrelevant.

4 Performance Comparison

To establish the feasibility of using Noisy-OR or Noisy-AND in an MLP, we apply each in an architecture targeted at the MNIST digit classification problem. We compare networks where the

hidden layers are composed entirely of sigmoids (with a softmax output) to one pure LNN and various mixed LNN's. In every architecture, the outputs are 10 neurons intended to capture each digit with one-hot encoding.

4.1 MNIST

The MNIST data set is composed of 70,000 handwritten digits. The training and test set are pre-specified, [3] split into 60,000 training images and 10,000 test images. For training the final models we adhere to this split, however for hyper-parameter selection we will split the training images into a training set of 50,000 images and a validation set of 10,000 images.

4.2 Network Architecture

We propose 9 different architectures which contain logical activations. Each architecture begins with a feature extraction layer composed of one or more sigmoids, or a noisy-OR layer. This is followed by one or more layers of different logical activations, and a softmax at the end for classification. We compare these models to 3 architectures composed of sigmoids followed by a softmax. All 12 models are displayed in table 1.

Architecture	Sizes
or-and	784-30-10
sig-or	784-30-10
sig-and	784-30-10
sig-or-sm	784-30-30-10
sig-and-sm	784-30-30-10
sig-sig-or-sm	784-30-30-30-10
sig-sig-and-sm	784-30-30-30-10
sig-or-and-sm	784-30-30-30-10
sig-and-or-sm	784-30-30-30-10
sig-sig-sig-sm	784-30-30-30-10
sig-sig-sm	784-30-30-10
sig-sm	784-30-10

Table 1: The twelve models constructed for training on the MNIST data set. The first layer corresponds to the 784 input pixels, whilst the remaining numbers in the sizes column denote the number of activations at each consecutive layer. For example, in *sig-or-sm* the 784 input pixels are connected to 30 sigmoid units, followed by 30 Noisy-OR units connected to 10 softmax units which produce the models output.

4.3 Hyper-Parameter Selection

Given the unique nature of these logical activations, there is no literature that helps to prescribe how to set hyper-parameters in networks that contain logical activations. Therefore, we perform a grid-search across a broad range of hyper-parameters.

We choose the hyper-parameter grid base on:

1. The theoretical properties of each model
2. Preliminary experiments

Weight Initialization

Initialising the weights and with a standard normal distribution has proven effective for standard activation functions [6]. Such a scheme is infeasible for logical activations, as it would allow the weighted sum z_l to be negative, which is not permitted. For the weight initialisation of our logical activations, we adopt the folded normal distribution [26] (the absolute value of the normal distribution) and adjust the weights produced by a scaling factor c. This scaling factor will be determined by the grid search. We present an heuristic argument for the grid of c's used.

Consider equation 14: $a_l = 1 - e^{-z_l}$ for the Noisy-OR activation. Recall that the weighted sum z_l must be positive. Figure 3 plots this equation, its slope may be expressed as $\frac{\delta a_l}{\delta z_l} = e^{-z_l}$. That is, the slope is exponentially decreasing starting from $\frac{\delta a_l}{\delta z_l} = 1$ at $z_l = 0$. Since all the weights and biases are positive, we would like to choose small weights, such that $z_l \approx 0$ and hence the gradients with respect to the weights and biases are larger, than they would be for a large z_l.

Therefore we choose the weight scale c to be a grid beginning at 1, and decrease by a factor of 10 for each entry in the grid. Note that small weights have an intuitive interpretation in terms of logical OR. If all the weight are near zero, this implies the underlying errors are near 1, $\varepsilon_i \approx 1$ (recall $\varepsilon_i = e^{-w_i}$). That is, all the inputs are ignored to begin with. This is approximately equivilent to an OR gate which has all zeros as its input. Hence the increase of just a single weight (equivalently the reduction of a single error) will have a large impact on the activation, like a single input becoming one in an OR whilst all others are zero.

In other words, the gradient with respect to any one input is large at initialisation time, prompting better training. Furthermore, as noted above we would expect a logical representation to be sparse. If the optimal approximation is sparse (which we demonstrate in section 5), then we have a better chance of obtaining it by starting all weights at zero and allowing SGD to increase the few weights that are relevant.

Similarly the logical AND function (see figure 4) has the gradient $\frac{\delta a_l}{\delta z_l} = -e^{-z_l}$ only differing in sign when compared to continuous OR. Therefore by our heuristic argument, we would also like small weights and biases. Like Noisy-OR, small weights and biases imply $\varepsilon_i \approx 1$. In the context of logical AND, this is approximately equivalent to an AND gate with all ones as the inputs. So a single input changing to zero (or noise ε decreasing below 1) would have a large effect on the activation. That is, the gradients are more receptive at lower weights. Again if the optimal approximation is sparse, the initialisation should benefit training.

4.3.1 Learning Rate

The derivate of the sigmoid function and the softmax in terms of a_l is:

$$\frac{\delta a_l}{\delta z_l} = a_l(1 - a_l) \tag{18}$$

whereas for Noisy-OR and Noisy-AND this is a_l and $-a_l$ respectively. For each model $0 < a_l < 1$, which implies for a given a_l, the gradient in Noisy-OR and Noisy-AND always has a larger magnitude than that for a sigmoid. Therefore, we would expect the logical activations to require smaller learning rates. We prescribe a schedule of learning rates that is lower than those prescribed for the softmax and sigmoid networks.

4.3.2 Selection Methodology

Activation Function	Hyper-Parameter Grid
$\lambda_{sigmoid}$ **Sigmoid/Softmax**	3.0, 1.0, 0.3, 0.1, 0.03
$\lambda_{logical}$ **OR/AND**	1.0, 0.3, 0.1, 0.03, 0.01
weight scale OR/AND	1.0, 0.1, 0.01, 0.001, 0.0001

Table 2: Hyper-Parameter grid based on heuristic arguments above.

Setting different hyper-parameters for each individual layer and performing an exhaustive grid search would be combinatorially prohibitive. However, standard activation functions are scarcely trained with different learning rates [23]. Therefore, all sigmoid and softmax neurons will have the same λ for any given run (for all architectures). For networks with no logical activations, this means the grid is simply one dimension in $\lambda_{sigmoid}$.

For architectures with a single logical activation, all $\lambda_{sigmoid}$ are considered, all $\lambda_{logical}$ (on the logical layers) and all weight scales are considered, for a total of 125 hyper-parameter combinations.

For those architectures with two logical activations, we assign priority to weight scaling. That is, all combinations of weight scaling are considered on the logical layers, but the same learning rate is used at each layer. This limits the total number of hyper-parameter combinations to 125.

Each run corresponded of 50 epochs with a mini-batch size of 10, after 25 iterations every learning rate in the network was divided by 3 to fine tune convergence. The following table presents the results of the grid search, showing the hyper-parameter combination which found the lowest test error on the validation set:

Architecture	$\lambda_{sigmoid}$	$\lambda_{logical}$	Weight Scale OR/AND	Error %
sig-or-sm	0.3	0.3	10^{-2}	3.17
sig-and-sm	0.03	0.1	10^{-2}	3.20
sig-sig-or-sm	0.03	0.03	10^{-2}	3.18
sig-sig-and-sm	0.1	0.1	10^{-3}	3.37
sig-or-and-sm	0.1	0.1	$10^{-2}/10^{-3}$	3.73
sig-and-or-sm	0.3	0.3	$10^{-3}/10^{-3}$	3.66
or-and	-	0.1	$10^{-2}/10^{-3}$	8.67
sig-or	0.3	0.3	10^{-2}	7.81
sig-and	0.3	0.3	10^{-3}	4.15
sig-sig-sig-sm	0.1	-	-	3.30
sig-sig-sm	0.03	-	-	3.13
sig-sm	0.1	-	-	3.41

Table 3: Final hyper-parameters chosen for each model based on 10,000 validation images after training on 50,000 images. The Error column is the classification error obtained on the validation set. Where two logical activations are present the first weight scale corresponds to the OR activation and the second to AND.

4.4 Comparison

With the hyper-parameters obtained for each model, we run full experiments on each model and compare classification error. Each architecture is run for 60 epochs, where after 30 epochs all learning rates are decreased by a factor of 3. For each architecture, the experiment is run 30 times for statistical significance. It is worth noting that for each run a new network is generated to guard against a single favourable/unfavourable weight initialisation. The results are presented in figures 5, 6, 7 and 8 and table 4.

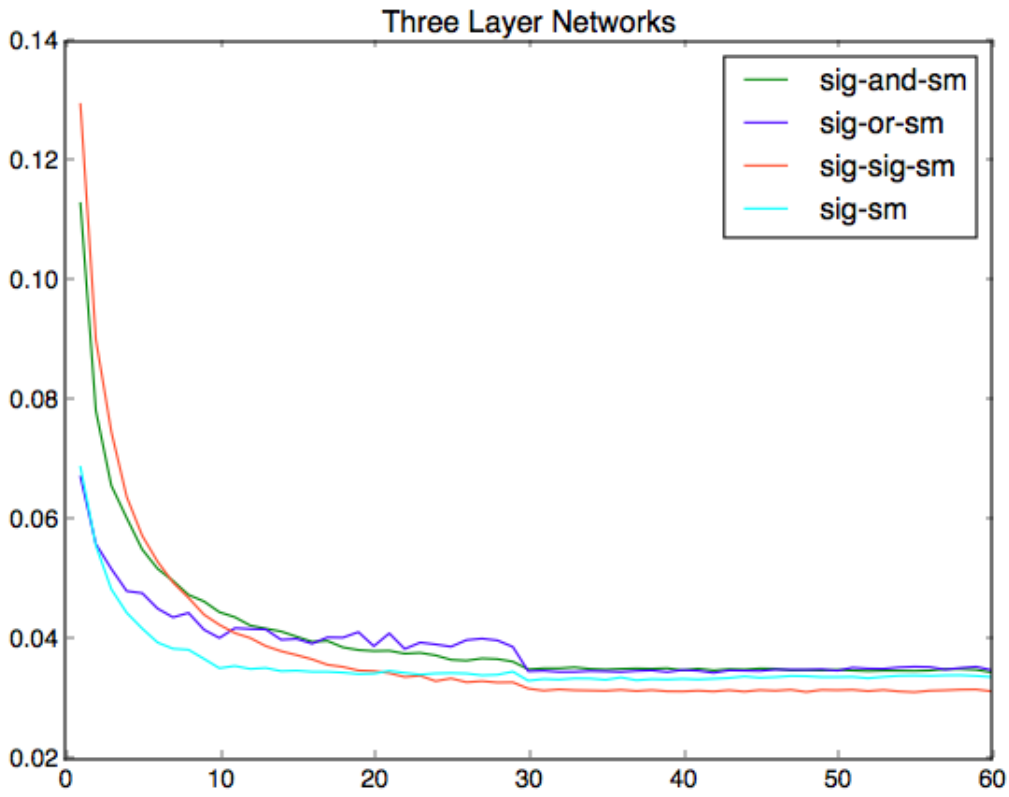

Figure 5: Comparison of three layer architectures, and a two layer model *sig-sm* The hyper-parameters used are those derived from the grid search. The graph displays the test classification error at the end of each epoch of training.

Figure 5 compares two three layer mixed LNN's with two standard architectures. Both mixed LNN architectures achieve comparable performance to the standard MLP's. It is interesting to note that to begin with, both LNN's train faster than the other three layer model *sig-sig-sm*. *sig-or-sm* appears to have a more noisy learning path way, although this may because the optimal learning rates for this model are higher than for the others.

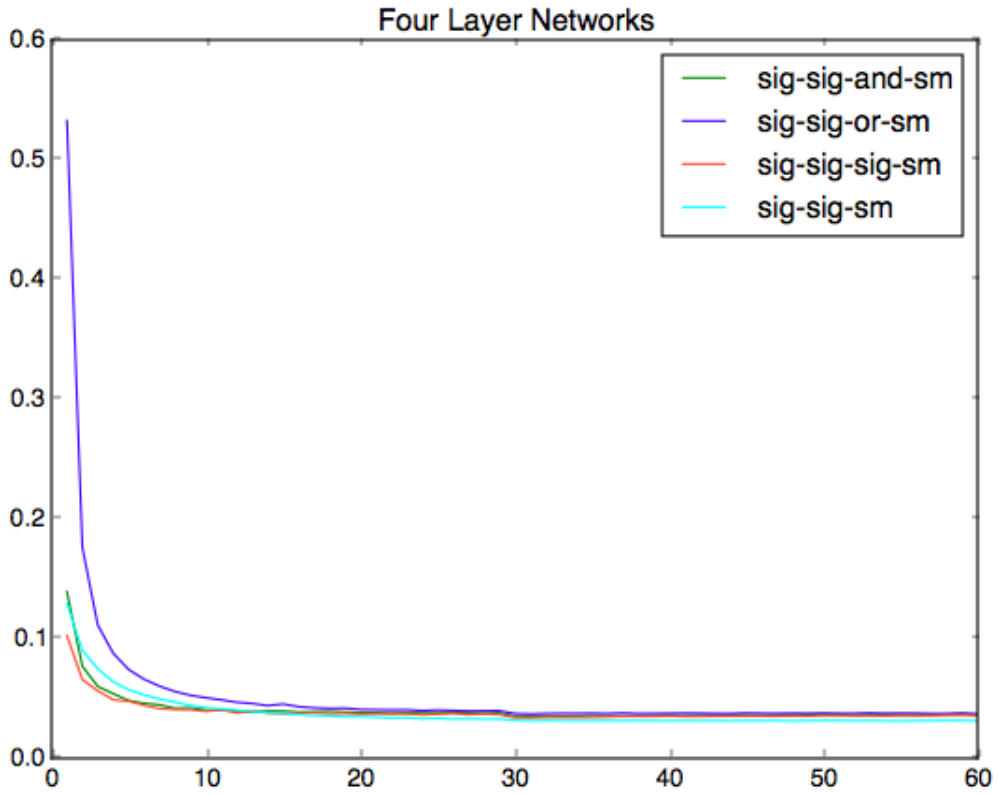

Figure 6: Comparison of four layer architectures, and a three layer model *sig-sig-sm* The hyper-parameters used are those derived from the grid search. The graph displays the test classification error at the end of each epoch of training.

Figure 6 compares two four layer mixed LNN's with two standard MLP's. Again the results are comparable with standard MLP's. However it is worth noting that increase in depth has made the results worse for all classes of MLP (see table 4). For example, *sig-or-sm* has a final classification error of 3.49, whereas its four layer counterpart *sig-sig-or-sm* has a worse classification error of 3.73. Similarly, the standard MLP *sig-sig-sm* has a final classification error of 3.13, while *sig-sig-sig-sm* has a final classification error of 3.43.

While the increased depth does not increase the representational power of our networks on MNIST, it is still useful to know that mixed LNN's retain comparable performance to standard MLP's with increased depth.

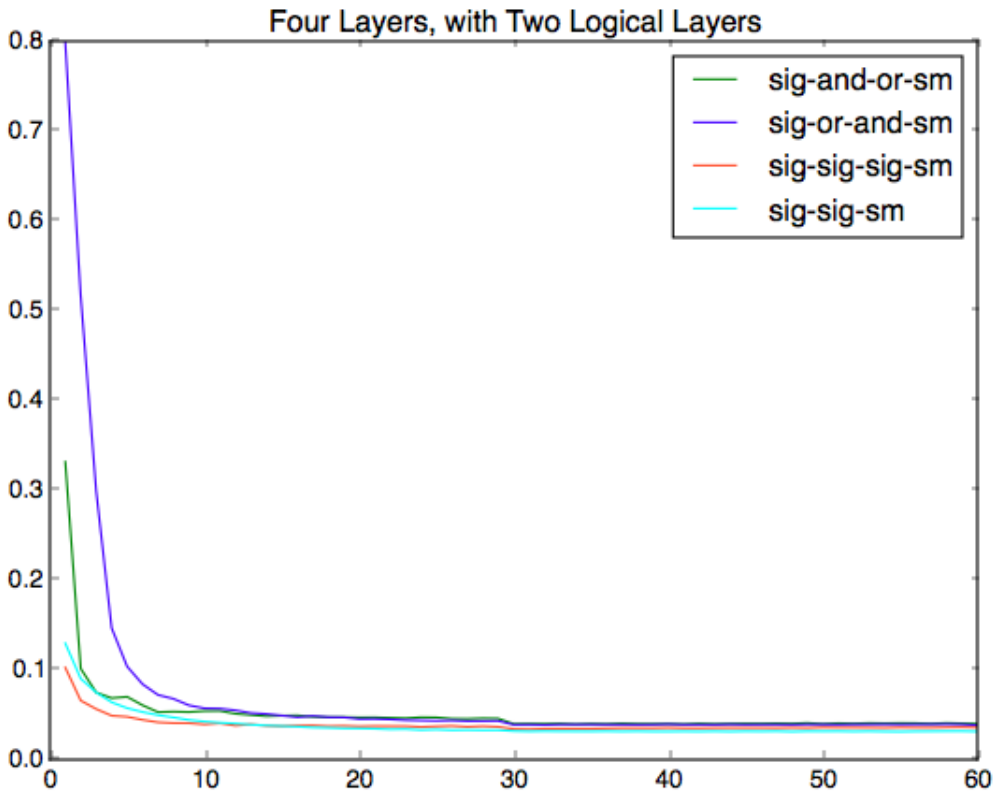

Figure 7: Comparison of four layer architectures, specifically the LNN's with two activations, and two standard models. The hyper-parameters used are those derived from the grid search. The graph displays the test classification error at the end of each epoch of training.

The experiments displayed in figure 7 are designed to assess the effectiveness of mixed LNN's with multiple logical layers. While it is not the focus of this work, providing a proof of concept for deeper LNN's is an important part of demonstrating their feasibility. *sig-or-and-sm* although slow to train, does not significantly differ in performance to *sig-sig-or-sm*. However, *sig-and-or-sm* has a final classification error of 4.00% whereas *sig-sig-and-sm* is lower at 3.64%.

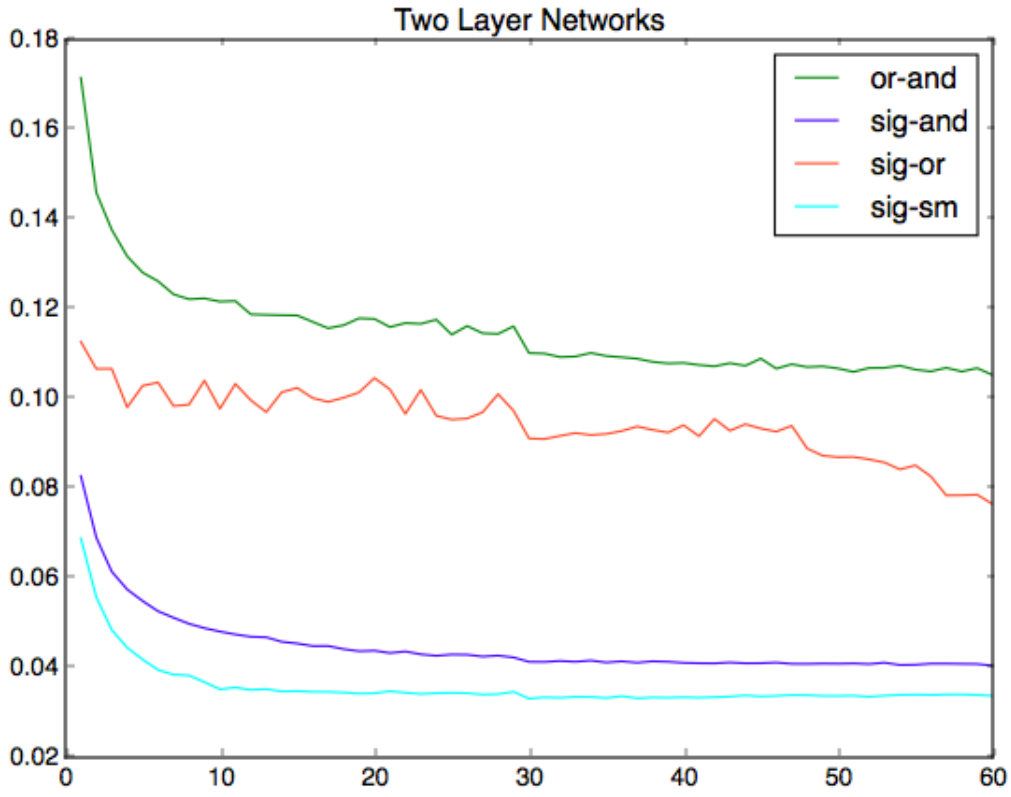

Figure 8: Comparison of two layer architectures. *or-and* is a pure LNN, *sig-or* and *sig-and* are mixed LNN's and *sig-sm* is a standard MLP. The hyper-parameters used are those derived from the grid search. The graph displays the test classification error at the end of each epoch of training.

Two layer networks are the only LNN's that are significantly outperformed by standard MLP's. In particular, final classification error for *or-and* is nearly three times that of the standard MLP *sig-sm*. *or-and* and *sig-or* train particularly slowly, but training beyond 60 epochs does not significantly reduce their classification error.

It is interesting to note that both networks layers containing a noisy-OR model exhibit a similar training pattern. That is ,both these models train slowly and exhibit considerable noise (even after averaging over 30 runs this can still be observed). By contrast *sig-and* exhibits a similar training pattern to *sig-sm*. This difference has not been investigated in this report but should be noted for future work, particularly work regarding performance improvements for logical activations

Architecture	Final Test Error %
sig-or-sm	3.49
sig-and-sm	3.45
sig-sig-or-sm	3.73
sig-sig-and-sm	3.64
sig-or-and-sm	3.84
sig-and-or-sm	4.00
or-and	10.32
sig-or	7.65
sig-and	4.05
sig-sig-sig-sm	**3.58**
sig-sig-sm	**3.13**
sig-sm	**3.34**

Table 4: Final test error averaged across all 30 runs of each architecture.

The results demonstrate that for standard MLP architectures, mixed LNN's have comparable performance to networks composed of standard activation functions. Pure LNN's despite being the only models that we can fully interpret, do not achieve comparable performance but nonetheless achieve a high enough classification accuracy for inference to be worthwhile.

5 Towards Interpretability

Neural networks are renowned for being a black box, a data scientist is unable to look at the individual weights and meaningfully interpret the learned relationships between the target class and inputs. We propose a method for pure LNN's, that is a promising step towards interpreting the relationship between inputs and class in a neural network. We compare this method for pure LNN's, mixed LNN's and standard MLP's, demonstrating the superior interpretability of pure LNN's.

5.1 Low Level Features

In this analysis, we will visualise lower level features using feature maps for the first layer of each network. These feature maps are simple images of the weights feeding into a single neuron, and are shaped according to the original image for inference purposes. In the first layer of the network, this image corresponds to the 784 $(28 * 28)$ weights on each individual pixel. In the grey-scale feature maps, the darker regions correspond to more positive weights, whereas the whiter regions are more negative weights or weights that are closer to zero in the context of a logical activations (which have no negative weights). An example taken from a fully trained *sig-sig-sm* model is shown in figure 9.

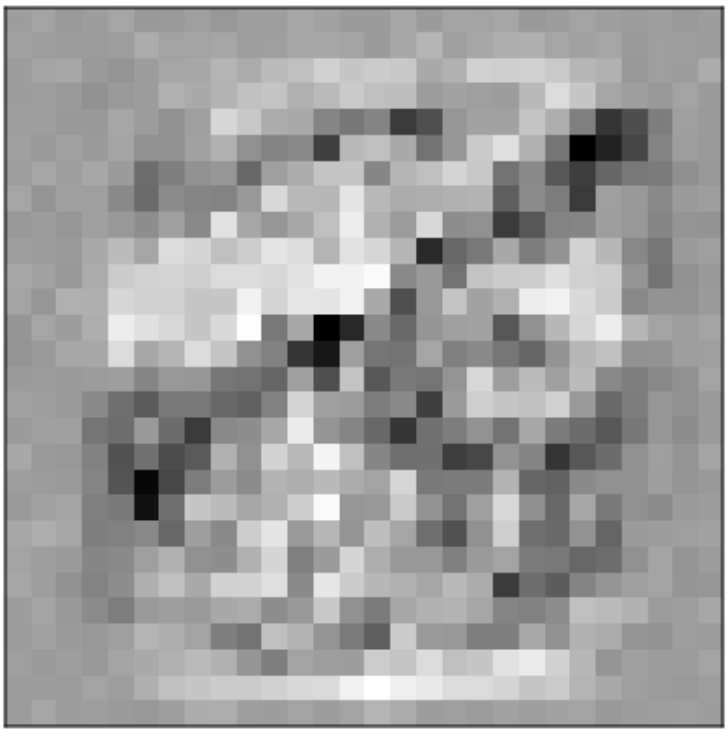

Figure 9: A grey-scale feature map, showing the weights feeding into a single neuron in the first hidden layer for the model *sig-sig-sm*. Note that the more white the more negative a weight is (or close to zero in the context of logical activations) and the more black the more positive. While we can see that some learned structure exists, it is very difficult to discern exactly what this structure means. For all conventional activation functions, the learned weights have this problem, where the structure is uninterpretable.

To test the interpretability of each class of networks we will visualise feature combinations in the pure LNN *or-and*, the mixed LNN *sig-and* and the standard MLP *sig-sm*. Figure 10 shows 5 randomly selected feature maps from the first layer of each network.

Referring to figure 10,we see in both cases where sigmoids form the lower level feature maps there clearly exists some structure, however it is not clear what this structure means. Furthermore, there is noise in that structure. Specifically, the semantic structure that would be useful to data scientist is mixed across features.

The problem is indirectly highlighted in [27]. In [27], Baldi demonstrates that a linear network learns semantically meaningful content. For example, Baldi shows a linear auto-encoder with a single hidden layer learns the first p principle components of the data (where p is the number of hidden units). However, he demonstrates that because there are multiple layers, there are infinitely many equivalent weight matrices that can represent these principle components. So a data scientist, without decomposing the weight matrices properly is unable to discern these principle components or even infer that they were learned.

This problem of equivalence highlighted by Baldi exists for all the standard activation functions. The general point we make is that if there was an underlying structure meaningful to a human, it would not be readily inferable because of that structure's equivalence with many other un-

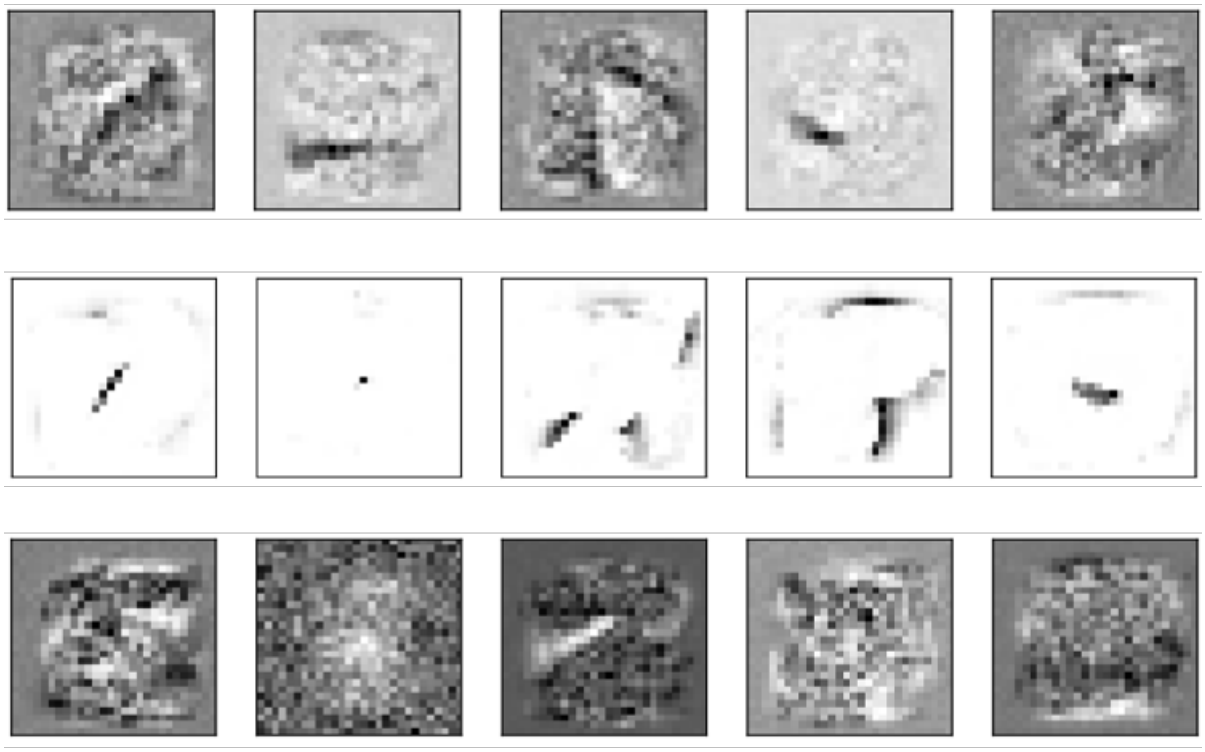

Figure 10: Feature maps for the weights feeding into 5 randomly selected neurons, in the first layer of 3 different neural networks. Top: The standard neural network *sig-sig-sm*, where each diagram represents the weights feeding into a sigmoid neuron in the first layer. Middle: The weights feeding into 5 Noisy-OR neurons in the first layer of *or-and*. Bottom: The mixed LNN, *sig-and*. Despite its logical layer, the weights feeding into its first layer (sigmoids) are difficult to interpret.

interpretable weight matrices. To our knowledge, no one has achieved a proper decomposition of the weights in the context of non-linear activations, which would be an incredibly useful tool. Such a decomposition is beyond the scope of of our work here. We turn our attention to the feature maps produced by a Noisy-OR first layer.

The Noisy-OR neurons from the first layer of *or-and*, lack the noisy looking feature overlap that is present with sigmoids. By contrast, their weights come to represent very distinct features focusing on a specific region of the input. That is, not only does a Noisy-OR neuron learn to capture the localised feature interactions inherent in any image recognition problem, but it does it in such a way that we can interpret which pixels the neuron is assigning credit to. This is in accordance with our central hypothesis regarding logical activations, that they would learn to ignore most inputs and focus on a few relevant features.

We present our theoretical rational for choosing the Noisy-OR (and Noisy-AND) activation functions, and demonstrate how this is confirmed by the empirical evidence in figure 10. Recall the activation function for a Noisy-OR neuron (ignoring the bias term for brevity). We rearrange the equation for the sake of demonstrating our point:

$$
\begin{aligned}
a_l &= 1 - e^{-z_l} \\
&= 1 - e^{-\sum_{i=1}^{p} w_i x_i} \\
&= 1 - \prod_{i=1}^{p} e^{-w_i x_i}
\end{aligned}
$$

Consider any single input pixel that has a non-zero weight. If that pixel is active, it will cause an exponential decrease in the magnitude of e^{-z_l} and an exponential increase of the activation value. That is, the logical activation is a product of inputs (with exponential weights), so it only takes a single input to increase the activation substantially.

For example, suppose a useful feature is the horizontal bar at the top of a 5, or a 7. Suppose that we had a single OR neuron that represented this feature, with weights all near zero apart from the region where the horizontal bar on a 5 or 7 usually is. The inclusion of a single large weight to the bottom of the image, would invalidate the representation of this feature, as this input alone can radically alter the activation (particularly if for a given image, $x_i = 0$ for all truly relevant pixels).

Therefore, such a feature may only be represented by weights all near zero, apart from the region where the horizontal bar on a 5 or 7 usually is. In contrast, sigmoid or ReLU activations exhibit some form of additive equivalence, where a single weight into 1 neuron can be equivalently represented as a sum of smaller weights into other neurons.

So given an LNN achieve reasonable classification accuracy, that is, learn good feature representation (which we have demonstrated empirically in section 4). Then these feature representation should manifest in the weight matrices of *individual* neurons, that is, weight matrices that only contain the information specific to one feature and are zero elsewhere.

5.2 Combination Layer

Of equal importance for the interpretation of these models is the second layer, i.e, the weighted combinations of lower level features. Figure 11 illustrates a stark difference between the LNN's and the standard architecture. We see that in a standard MLP the weights retain a Gaussian distribution, even after training. For logical activations the trained weights follow a radically different distribution, with a few large weights but most near zero.

For the second hidden layer *sig-sm*, it is unclear what these weights could represent. The network has achieved a final classification accuracy of error of just 3.43%, so it has clearly learnt a meaningful structure. However without the kind of decomposition alluded to above, the interpretation remains illusive to human data scientists.

By contrast for the LNN's, we can easily interpret the large weights as the few input features which the logical activation has learnt to pay attention to. By the same mechanism described in the analysis of the first layer, each logical activation ignores the bulk of the inputs, and takes a logical combination of select features.

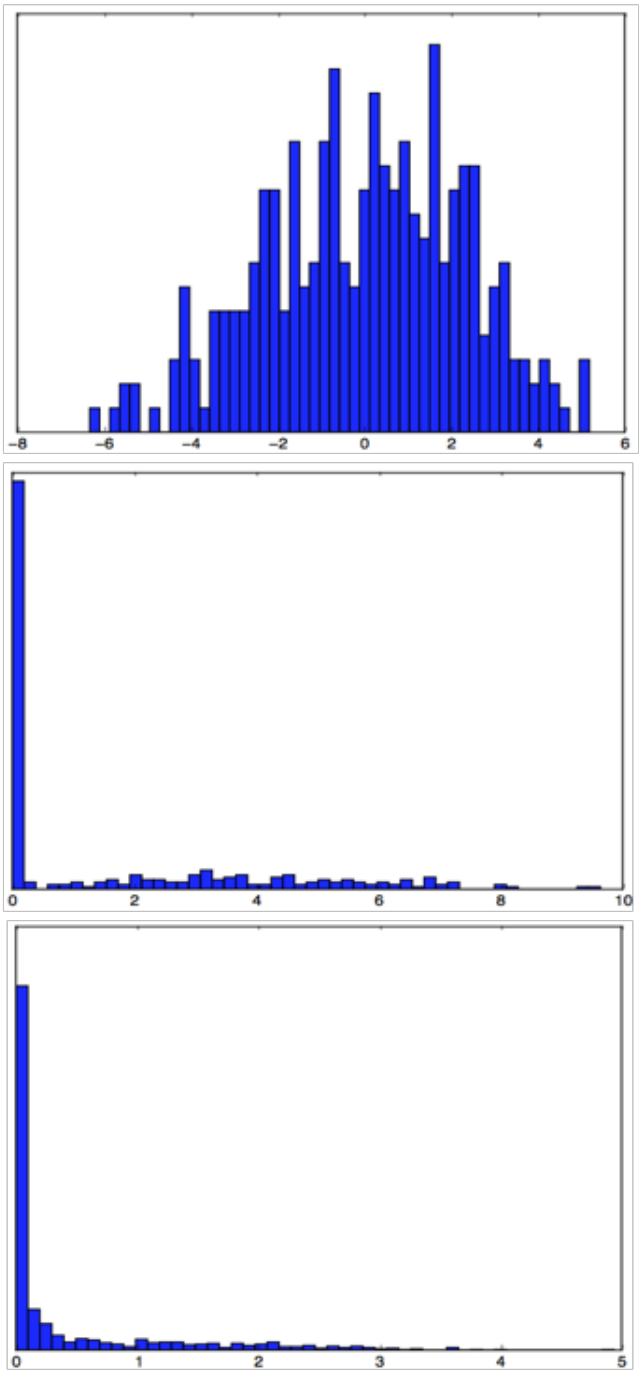

Figure 11: Histograms depicting the distribution of weights at the second layer of each of our proposed models. Top: *sig-sm*, the weights in the second layer still follow a normal distribution, even after training. Middle: *or-and*, the distribution is multi-modal, with most of the weight near zero, and a few large weights roughly between 2 and 6. Bottom: *sig-or*, again the overwhelming majority of the weights are near zero, but the distribution is not multi-modal, with less weights at higher magnitudes.

5.3 Interpretation

Consider the pure LNN *or-and*. Each AND neuron, is one of 10 outputs that correspond to a digit class. Let us choose a target class D for inference, such as 3. For the weights feeding into the output neuron that represents 3, we know that most are zero, while some weights are large. We choose all the large weights feeding into this neuron. More formally, we choose every weight feeding into this neuron that exceeds some threshold t. For these weights, we examine the features determined in the first hidden layer (the OR layer), and interpret them with respect to logical AND. In practice we choose $t = 1$. The determined features are shown in figure 12.

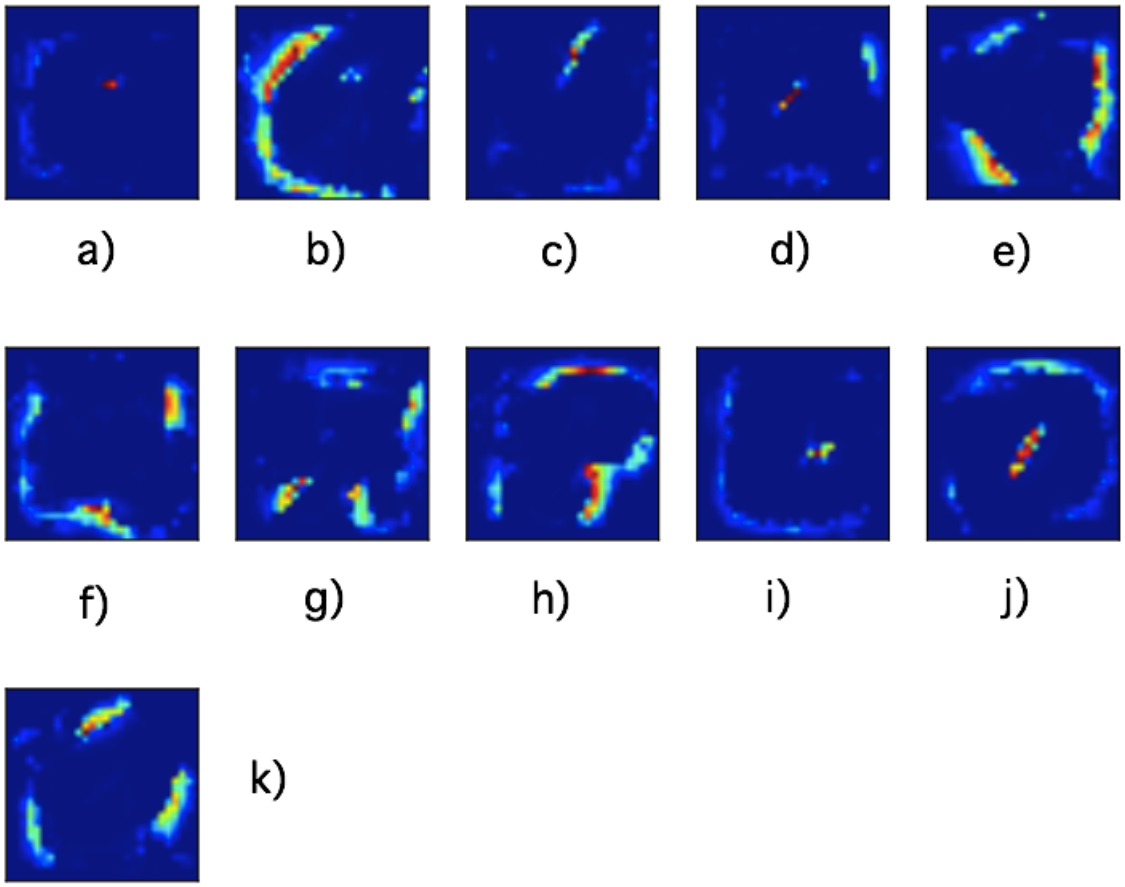

Figure 12: The coloured feature maps represent the size of the weight on each pixel for a given first layer neuron. Dark blue represents a zero weight, whereas green though to yellow through to red represents increasingly large weights. These are the first layer OR features that are most significant to the output neuron that represents 3 in a fully trained *or-and* LNN. These are chosen by selecting the feature maps that have large second layer weights on them where $w > t = 1$.

Each image represents the weights into a single OR neuron in the *or-and* architecture. For any feature, a small subset of the pixels is required to cause that feature to activate, seeing as these are weights on a Noisy-OR activation. Feature e) has learnt to to require the right hand side of a 3 or the upwards flick at the bottom. Whereas feature h) recognises the curve at the top of a 3, or the bottom right curve. Furthermore, features d) and j) represent the central flick of a 3.

Most of the features are not entirely unique to a 3. For example feature h) could be more useful for classifying a 9. Some features such as b) seem better suited to other features such as 6 or 0 (however the larger weights are concentrated at the top flick, which is present in a 3). It is important to recognise that the network would have to learn to share features, seeing as there are only 30 lower level feature maps and 10 digits to share these.

These features partially give us insight as to what parts of a 3, make a written digit a 3. In particular, the curves at the top right, bottom left, and bottom right occur with high intensity and frequently across these features. Of course, interpretation with respect to image recognition is largely an academic problem. For pure interpretation purposes, a human does not need to know what shapes make up a digit, a child can describe what makes a 3, a 3. However, feature interpretation becomes very useful where the relevant substructures are not fully known such as scans for cancer patients.

For example, consider a data set of cancer patients (where the tumour class, benign or malignant is known). The doctors may have an idea of what constitutes a malignant tumour, however they don't fully understanding all the visual cues that make a tumour malignant. An interpretable network such as a pure LNN (provided it was accurate enough) could identify part of the images that are important in the classification of tumours, part that human experts did not realise were important. Our work here specifically targets a problem for which the substructures are well known, as to demonstrate the feasibility of interpretation.

That said, a second benefit of being able to interpret the composition of learned features is obtaining insights about the potential problems of the network. That is, obtaining insights about why the network made certain misclassifications, problems that could be apparent across data sets. By identifying such problems, we are ultimately able to make adjustments to improve the design of our network more rigourously. For example, the network that evolved the features in figure 12 had about a 90% classification accuracy, meaning it is misclassifying about 1000 digits in the test set. It is easy to see *why* a 3 might be misclassified as a 6 and vice versa, given the features in figure 12.

With the problem clearly identified we can begin to consider solutions for dealing with it. For example, if the features shared between 6 and 3 are too common we might hypothesise greater network width at the first hidden layer will help improve performance, by introducing a wider variety of features. Testing a wider model, we can again examine the lower level features to identify if this hypothesis was correct and continue that process.

Using standard activations we could also hypothesise the network required a wider variety of features, but the crucial difference is that it is a lot harder to identify whether such a problem exists, because we cannot interpret what has been learnt. We present an attempt that interpreting *sig-and* in the same way we interpreted *or-and*. That is, for the output 3 we choose all weights exceeding the threshold $t = 1$ and examine the low level features that those large weights are coefficients for.

Figure 13 shows all the features for which there is large weights feeding into the output 3. The model is *sig-and* so we interpret by taking a logical AND of all the features, i.e. all of these features must activate for the network to output 3. However, it is not clear what these individual feature maps represent due to the problems with equivalence described above. In short, we are unable to interpret networks that utilise standard activations, in our logical interpretation framework.

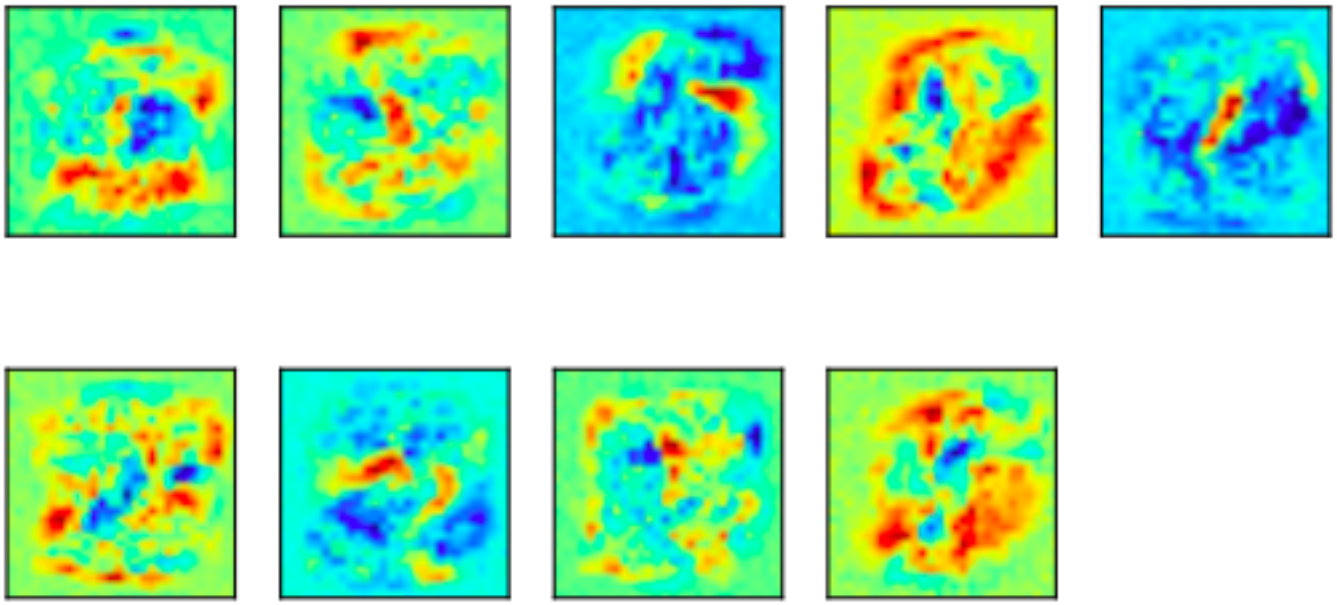

Figure 13: The first layer sigmoid features that are most significant to the output neuron that represents 3 in a fully trained *sig-and* LNN. These are chosen by selecting the feature maps that have large second layer weights on them where $w > t = 1$.

6 Conclusion

Deep Learning has proved incredibly useful in the last 5 years, and is being deployed in commercial applications in domains such as object recognition, speech classification, and machine translation at an increasing rate [23]. While simply achieving good performance in these domains is a huge achievement and very useful to society, we would also benefit from understanding the patterns and features these deep architectures learn.

Interpreting what a powerful neural network has learned would primarily allows us to discover semantically meaningful patterns in the data, allowing a human to take action based on the identified patterns. We highlighted tumour recognition in section 5, as an incredibly valuable application of such interpretation. We could have even more to gain in the domain of non-image classification problems, such as identifying whether a debtor will repay their loan or not. That is, humans are evolutionarily adept at recognising patterns in images but patterns in less natural data sets such as information regarding debtors, are be more difficult to recognise

In this report, we design new activation functions that are a step towards the goal of powerful and interpretable neural networks. Drawing from Bayesian Networks, and Pearl's Noisy-OR relation we derive more general relations that are valid for continuous inputs, under noise (uncertainty of influence on the relation). These models can be re-parameterised using basic algebra, to serve as logical activations in a neural network.

We present a clear case for why one would want to use logical activations in a neural network. In section 3 we discuss the inherent interpretability of AND's or OR's of inputs. Whilst in section 5, we discuss why logical activations given correctly trained weights can be interpreted in a logical manner. We introduce the notion of Logical Neural Networks (LNN's), MLP's that contain logical activations.

Section 4 demonstrates that these LNN's can learn a reasonably challenging problem, classifying the MNIST data set. We show that mixed LNN's which contain sigmoids and a softmax as well as logical activations, achieve performance comparable to standard MLP's. Our single pure LNN *or-and* achieve 90% classification accuracy on MNIST. Although lower then the other models, this is still high enough to make the model worth interpreting.

We then demonstrated how a pure LNN could be interpreted, using the MNIST results as an example. The feature maps in figure 11 showed that the individual weight matrices of *or-and* could be interpreted as the features that make up a given digit. We then described how the second layer allows us to consider a logical combination of easily recognisable lower level feature maps. The entire process results in a fairly effortless way to interpret a pure LNN, and paves the way for the interpretation of more complicated models.

This work is not necessarily close to achieving the lofty applications for high performance neural network interpretation stated above. However it is a solid proof of concept, using activations that are yet to have years of optimisations from a community of neural network researchers. That is, future work to develop new activations and improve LNN architectures would be worthwhile steps towards achieving high performance, interpretable neural networks.

References

[1] Judea Pearl. *Probabilistic Reasoning In Intelligent Systems*. 1988.

[2] Judea Pearl. "Bayesian Networks: A Model of Self-Activated Memory for Evidential Reasoning". In: (1985).

[3] Corinna Cortes Yann LeeCun and Christopher J.C Burges. *The MNIST Database of Handwritten Digits*. URL: `http://yann.lecun.com/exdb/mnist/`.

[4] Stuart J. Russell and Peter Norvig. *Artificial Intelligence A Modern Approach*. 1995.

[5] N. S. Altman. "An Introduction to Kernel and Nearest Neighbour nonparametric regression". In: (1992).

[6] Genevieve. B. Orr Yan LeCun Leon Bottou and Klaus-Robert Muller. "Efficient BackProp". In: (1998).

[7] Geoffrey Hinton David Rumelhart and Ronald Williams. "Learning representations by back-propagating errors". In: (1986).

[8] Sebastian Ruder. *An Overview of Gradient Descent Optimization Algorithms*. 2016. URL: `http://sebastianruder.com/optimizing-gradient-descent/`.

[9] Shaoqing Ren Kaiming He Xiangyu Zhang and Jian Sun. "Empirical Evaluation of Rectified Activations in Convolution Network". In: (2015).

[10] Anna Choromanska Mikael Henaff Michael Mathieu Gerard Ben Arous and Yan LeCun. "The Loss Surfaces of Multilayer Networks". In: (2015).

[11] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: (2010).

[12] Ilya Sutskever Alex Krizhevsky and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: (2012).
[13] Vinod Nair and Geoffrey Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: (2010).
[14] Jia Deng Wei Dong Li-Jia Li Kai Li Fei-Fei and Richard Socher. "ImageNet: A Large-Scale Hierarchical Image Database". In: (2009).
[15] Sergey Ioffe and Christian Szegeedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariant Shift". In: (2015).
[16] Thomas Unterthiner Djork-Arne Clevert and Sepp Hochreiter. "Fast And Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: (2015).
[17] Tiangi Chen Bing Xu Naiyan Wang and Mu Li. "Empirical Evaluation of Rectified Activations in Convolution Network". In: (2015).
[18] Christopher M Bishop. *Pattern Recognition and Machine Learning*. 2006.
[19] Kevin L. Priddy and Paul E. Keller. *Artificial Neural Networks: An Introduction*. 2005.
[20] G. Cybenko. "Approximation of Superpositions of a Sigmoidal Function". In: (1989).
[21] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: (1991).
[22] Judea Pearl. "Reverend Bayes on inference engines: A distributed hierarchical approach". In: (1982).
[23] Yoshua Bengio Yann LeCun and Geoffrey Hinton. "Deep Learning". In: (2015).
[24] Jia-Ren Chang and Yong-Sheng Chen. "Batch-normalized Maxout Network in Network". In: (2015).
[25] Ueli Meier Dan Cires and Jurgen Schmidhuber. "Multi-column Deep Neural Networks for Image Classification". In: (2012).
[26] L S Nelson F.C Leone and R B Nottingham. "The Folded Normal Distribution". In: (1961).
[27] Pierre Baldi and Kurt Hornik. "Neural Networks and Principle Component Analysis, Learning from Examples Without Local Minima". In: (1988).