ERASMUS UNIVERSITY ROTTERDAM

Erasmus School of Economics

Bachelor Thesis (Econometrie & Operationele Research)

Improving Artificial Neural Network Classification through Neuron Specialization

Gianni Brauwers

Student ID: 452389

Supervisor: A. Castelein (castelein@ese.eur.nl)

Second assessor: F.J.L. van Maasakkers (vanmaasakkers@ese.eur.nl)

Date final version: July 7, 2019

Abstract

In this paper, a novel artificial neural network extension is proposed that draws inspiration from the fact that actual biological brains consist of various types of neurons. The method known as *"neuron specialization"* consists of training not only the output neurons, but also the hidden neurons to activate for specific classes. This creates neurons that the output neurons can easily rely on and improves the interpretation of the hidden neurons. The first part of this research consists of exploring the neural network model intricacies and performances. Afterwards, the neural network extension is tested using two case studies: a marketing case and an image recognition case. The neural specialization extension is able to achieve significant performance boosts and interpretation improvements for several different neural network architectures.

Contents

1 Introduction

Artificial Neural Networks are an extremely popular Machine Learning method that are based on the efficient learning structure of real neurons and have abundantly many applications (Liu et al., 2017). Although neural network models can achieve state-of-the-art results in many different fields, there are still crucial differences between a neural network and the actual biological brain structure that it is based on. For example, the human brain is split up into different sections with each section being responsible for different cognitive abilities. There are separate sections and neurons assigned to interpret language, evaluate input signals from vision, store memories etc. (Hines, 2018). Within these sections, different types of neurons are found that are thought to have specific functions, like face neurons are used to recognize faces (Axelrod et al., 2019). One could say these various neuron types have particular roles. Unfortunately, researchers have not yet found a definite way to fully classify all these neurons, which means it is not known how specific and well-defined these functions are (Kepecs and Fishell, 2014). Nonetheless, even though we are far from being able to fully understand and simulate these complexities of actual neurons found in the human brain, the concept of neurons with roles or specializations can still be applied to an Artificial Neural Network made for classification purposes.

In this paper, a novel Artificial Neural Network extension is proposed that incorporates the concept of neuron specialization by assigning roles to neurons. This is implemented by training neurons to only activate for a specific class or classes. Thus, each node is specialized to extract features for those particular classes. The concept of neuron specialization does not only hold merit due to its biological similarities. In this paper, the actual practical advantages and disadvantages of this method will be discussed in detail.

The first step in this research is exploring the workings of the neural network architecture and ensuring the mechanisms and performances of this model are clear. This is done by evaluating a regular neural network implementation and comparing it to another typical classification technique: the Multinomial Logistic Regression model. As such, the first research question to answer is as follows:

How does an Artificial Neural Network perform compared to a standard classification method such as a Multinomial Logistic Regression model?

Similarly to Agrawal and Schorling (1996), the methods are assessed in a marketing context where the models are required to predict the purchase decisions made by customers. Many factors are taken into account when evaluating the performances. The most obvious performance measure would be the prediction error rate, since the main goal of any classification method is generally to accurately classify or predict data records. Nevertheless, other factors such as fitting times, prediction times and interpretation are also taken into account. After the exploratory research, the neural network model will be extended by introducing specialized neurons. To test this extension, not only will the two marketing data sets mentioned before be used, but also two image classification databases. The performances of the neuron specialization model will be compared to the previous model implementations for the purpose of answering the second research question:

Can Artificial Neural Network models be improved through the use of neuron specialization?

The outline of this paper is as follows. The paper starts with a literature review on the methods and problem domains in Section 2. Then, the multinomial logit model, neural network model, and the neuron specialization extension are discussed in Section 3. Following that, an analysis is given of the four datasets used in this paper in Section 4. Lastly, the model architectures and comparisons are discussed in Section 5, followed by the final conclusions in Section 6.

2 Literature Review

The first part of this research is an exploratory research that is performed through a model comparison similarly to Agrawal and Schorling (1996), using similar models and data. In the paper by Agrawal and Schorling (1996), a comparison between a neural network and a multinomial logistic regression model is made based on their performances when forecasting market shares. The authors used different data sets and various data groupings to test and compare these models in different situations. They conclude that in the majority of situations, the neural network outperforms the multinomial logistic regression model. However, it was also concluded that the results of the logistic regression model are significantly easier to interpret. Nevertheless, neural networks still are a popular choice when performing such a prediction task. For example, Fish et al. (2004) used a neural network implementation combined with a genetic algorithm to model brand shares.

In contrast to Agrawal and Schorling (1996) who predicted brand market shares, the model comparison in this paper are made based on individual brand choice predictions. Brand choice modelling is a topic that has been researched quite extensively in the past. For example, (Paap and Franses, 2000) used a dynamic multinomial probit model with different long-run and short-run effects to model the brand choices for saltine crackers. Other models such as Mixed- and Latent Markov Models (Poulsen, 1990), and Multiple Brand Choice models (Baltas, 2004) have been developed for this problem domain. Unsurprisingly, neural networks are also a popular choice for this application. For example, Vroomen et al. (2004) use a neural network to model consideration sets and brand choice. Bentz and Merunka (2000) even propose a hybrid method consisting of both a neural network model and the multinomial logistic model to predict instant coffee brand choices.

The second part of this research consists of testing the proposed extension of neuron specialization. As to my knowledge, there is no mention in the literature of this concept. To test this method, not only will it be implemented for brand choice modelling, but also for image classification. Just like brand choice modelling, image classification is an extensively researched topic, even having several competitions such as the ILSVR Challenge (Russakovsky et al., 2015) attached to the subject. Many classification methods such as Linear Models, K-Nearest Neighbours, Neural Networks and many others can be used to classify images. For example, LeCun et al. (1995) compare and contrast the performances of several classification methods when used for classifying images of handwritten digits. Bhatnagar et al. (2017) use Convolutional Neural Networks to classify images of fashion products and achieve state-of-the-art results. In the literature one can often find that these (Deep) Convolutional Neural Networks generally obtain the best results in this problem domain (LeCun et al., 1995) (Russakovsky et al., 2015).

3 Methodology

This section covers the methods and techniques used in this paper. Section 3.1 starts by explaining the basics of the multinomial logistic regression model. Section 3.2 continues with the most important aspects of the neural network model and Section 3.3 explains in detail the specialized neuron extension. Finally, an overview of the implementation and testing methods is given in Section 3.4. To be clear, this paper is not meant as a complete learner's guide to neural networks and multinomial logistic regression. As such, this section will only discuss the basics of these models and the elements that are necessary to understand the results. For further information on the multinomial logit model and the neural network model, one can look into the works by Paap (2001) and Haykin (1994).

3.1 Multinomial Logistic Regression Model

The model used to benchmark the neural network model is the multinomial logistic regression model. The multinomial logistic regression model is a regression model based on the softmax function and is specifically designed for classification. This model predicts the probabilities of a record being a label by evaluating the explanatory variables attached to that record. Two different versions of this model are used in this paper. Firstly, the standard multinomial logistic regression model which will be referred to as the MNL model. Suppose a record r_i consists of n_x explanatory variables that are contained in the vector $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,n_x})$. Then, the probability of the label y_i of record r_i being equal to j is given by the following equations:

$$Pr(y_i = j|X_i) = P_{i,j} = \frac{exp(\beta_{0,j} + X_i'\beta_j)}{1 + \sum_{h=0}^{J-1} exp(\beta_{0,h} + X_i'\beta_h)} \text{ for } j = 0, ..., J-1, \tag{1}$$

$$Pr(y_i = J|X_i) = P_{i,J} = \frac{1}{1 + \sum_{h=0}^{J-1} exp(\beta_{0,h} + X_i'\beta_h)}. \tag{2}$$

Where J is the total number of classes, y_i is the label of record i, $\beta_j = (\beta_{1,j}, \beta_{2,j}, ..., \beta_{n_x,j})$ is the weights vector for class j, and $\beta_{0,j}$ is the bias parameter of class j. To prevent identification issues, β_J is a fixed vector of zeros and $\beta_{0,J}$ is also fixed to zero.

Secondly, a variant of the multinomial logistic regression model called the conditional logistic regression model, or CNL, is also used. This variant is better suited to data where there are class-specific explanatory variables. Suppose a record r_i has n_z explanatory variables for each class that are contained in the vectors $Z_{i,j} = (z_{i,j,1}, z_{i,j,2}, ..., z_{i,j,n_z})$ for $j = 1, ..., J$. Then, the probability of the label y_i of record r_i being equal to j is equal to

$$Pr(y_i = j|Z_i) = P_{i,j} = \frac{exp(\beta_{0,j} + Z'_{i,j}\gamma)}{\sum_{h=0}^{J} exp(\beta_{0,h} + Z'_{i,h}\gamma)} \text{ for } j = 0, ..., J. \tag{3}$$

Where $\gamma = (\gamma_1, \gamma_2, ..., \gamma_{nz})$ is a vector of weights, $\beta_{0,j}$ is the bias parameter for class j, and $Z_i = (Z_{i,1}, Z_{i,2}, ..., Z_{i,J})$ is the matrix containing all the class-specific explanatory variables for each class. To prevent identification issues, the bias parameter $\beta_{0,J}$ is fixed to zero. The weights in both the CNL as well as the MNL model can be learned through the maximum likelihood estimation method. The likelihood function that needs to be maximized can be defined as follows:

$$L(\theta) = \prod_{i=1}^{n} \prod_{h=1}^{J} P_{i,h}^{I_{[y_i=h]}}. \tag{4}$$

Where n is the total number of records contained in the training set, θ represents all the parameters that need to be learned and $I_{[y_i=h]}$ is an indicator function that is equal to 1 if $y_i = h$ and 0 otherwise. The problem with this function, however, is that the term $P_{i,h}$ is between 0 and 1, and the product of such numbers becomes extremely small very fast. To prevent this problem, the log-likelihood function is used instead:

$$l(\theta) = \sum_{i=1}^{n} \sum_{h=1}^{J} I_{[y_i=h]} log(P_{i,h}). \tag{5}$$

By maximizing $l(\theta)$, parameter estimates for θ can be derived which can then be used to predict new records. The estimated coefficients can quite easily be interpreted using the odds ratios $\Omega_{j|l}(X_i)$, or the log of these ratios. These ratios can be defined as follows:

$$\Omega_{j|l}(X_i) = \frac{Pr(y_i = j|Z_i)}{Pr(y_i = l|Z_i)} = \frac{exp(\beta_{0,j} + Z'_{i,j}\gamma))}{exp(\beta_{0,l} + Z'_{i,l}\gamma))}, \tag{6}$$

$$log(\Omega_{j|l}(X_i)) = (\beta_{0,j} - \beta_{0,l}) + (Z'_{i,j} - Z'_{i,l})\gamma. \tag{7}$$

For example, suppose that $\gamma_k > 0$, then an increase in $z_{i,j,k} - z_{i,l,k}$ would result in an increase in the log-odds ratio making it more likely that class j is chosen over class l. Lastly, the way new records are classified is done by simply predicting the label that has the highest estimated probability to be the correct label based on the estimated coefficients $\hat{\theta}$: $\hat{y}_i = argmax_h(\hat{P}_{i,h})$.

3.2 Artificial Neural Network

The inspiration behind the Artificial Neural Network model is to imitate the structure of the biological brain. A neural network is a layered structure of neurons where the knowledge of the network is contained within the connections, or weights, between the neurons. The very first layer of the network contains the input variables and the final layer contains the output neurons. The layers in between these are called hidden layers. When a data record is evaluated using the input variables, the activations of the input layer neurons activate the neurons in the first hidden layer through the connections. This way, the activations travel through the network till the output layer is reached where the final prediction is determined by predicting the label of the output node with the highest activation value. The idea is that is that each neuron in each layer captures a feature in the data, and each layer combines the features of the previous layer into larger features. When trying to understand what features a neuron can capture, one can think of price differences between brand choices or patterns in purchase behaviour in a marketing dataset. Another example would be the shapes and components of different digits when classifying images of digits. Before moving on to the more in-depth workings of a neural network, some notation is introduced with the aid of Figure 1. The notation used in this paper is highly similar to the notation defined in the Stanford University *CS230* course (Ng and Katanforoosh).

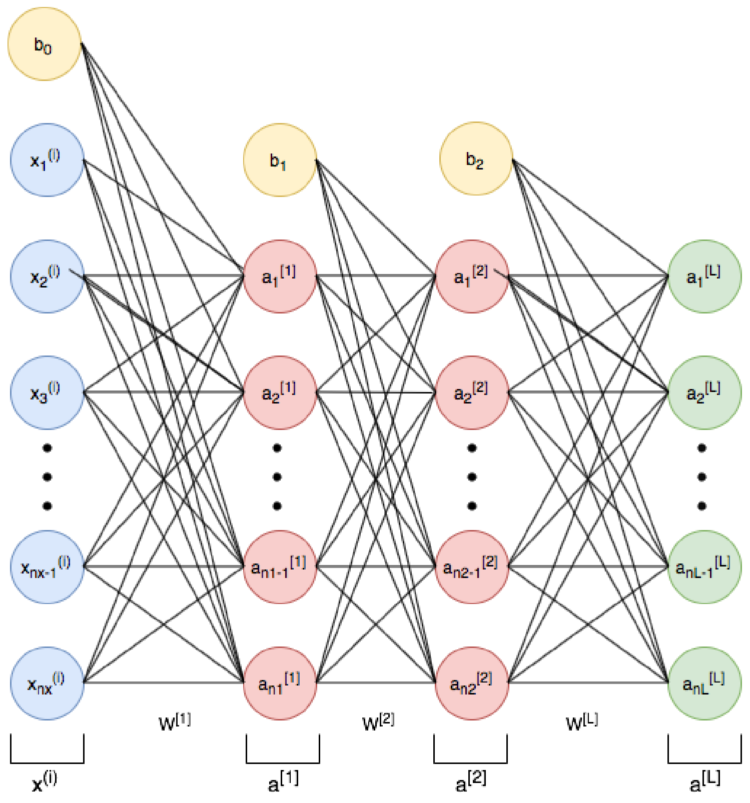

Figure 1: Architecture of a Neural Network model with two hidden layers. Input neurons are highlighted in blue, bias neurons in yellow, hidden neurons in red, and output neurons in green.

Nomenclature

$y^{(i)}$	The correct label of record i.
$x^{(i)}$	Vector of input variables for record i: $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_{n_x-1}^{(i)}, x_{n_x}^{(i)})$.
$t^{(i)}$	Target vector for record i: $t^{(i)} = (t_1^{(i)}, t_2^{(i)}, ..., t_{n_L-1}^{(i)}, t_{n_L}^{(i)})$.
$a^{[l]}$	Vector of the activations of the neurons in layer l: $a^{[l]} = (a_1^{[l]}, a_2^{[l]}, ..., a_{n_l-1}^{[l]}, a_{n_l}^{[l]})$.
b_l	Bias neuron of layer l.
$W^{(l)}$	Weight matrix of size $n_{l-1} \times n_l$ containing the weights between the neurons in the $(l-1)$'th layer and the (l)'th layer.
L	Number of layers ($L = 3$ in the example architecture of figure 1).
n	Number of records in the training data.
n_x	Number of input neurons.
n_l	Number of neurons in layer l.

The activation of a neuron is determined by the linear combination of the activations in the previous layer multiplied by the corresponding set of weights. However, neural networks are often used in highly complex situations which usually result in non-linear relations. Thus, the linear activations of the neurons are not sufficient to capture all the relations in the data. As such, every neural network needs activation functions to transform the inputs of the neurons to introduce non-linearity into the network. Suppose there is a hidden neuron h in layer $l > 1$ with activation function $R_h(x)$. Then, the activation of neuron h is equal to $R_h(a^{[l-1]'}W^{[l]}_{*,h})$. A classic activation function is the sigmoid function. However, this function often suffers from various problems such as vanishing gradients and a non-zero mean that slow down the training process (LeCun et al., 1998). These problems can be solved by using a Rectified Linear Unit (ReLu) activation function (Glorot et al., 2011) or the leaky ReLu function (Maas et al., 2013). For the output layer however, a softmax function is recommended for multinomial classification tasks (Dunne and Campbell, 1997).

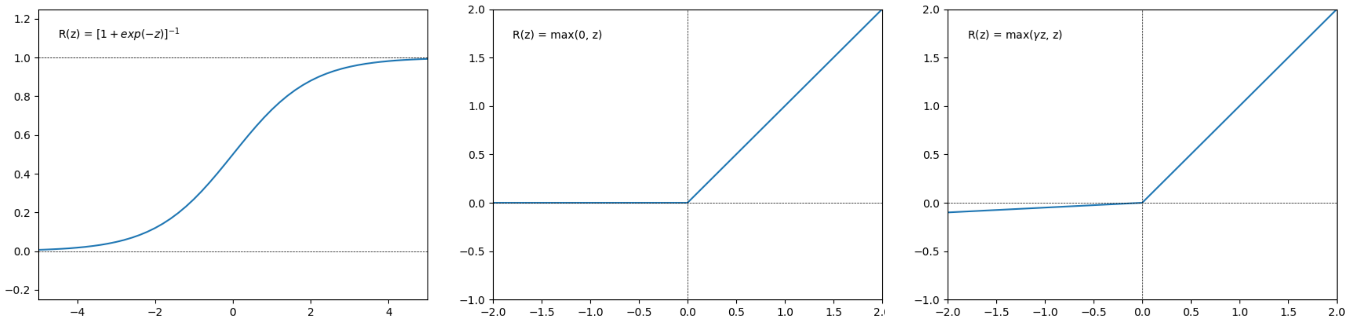

Figure 2: Plots of the sigmoid function, the Relu function, and the leaky ReLu function, respectively.

The weights in the network are trained by defining a loss function and minimizing it. The loss function can be defined in various ways. A classic form of the loss function is the Mean Squared Error (MSE) loss, but a more suitable function for a multinomial classification task is the Cross-Entropy loss function (Kline and Berardi, 2005);

$$\text{MSE: } E = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n_L}(t_j^{(i)} - a_j^{[L]})^2, \qquad \text{Cross-Entropy: } E = -\sum_{i=1}^{n}\sum_{j=1}^{n_L} t_j^{(i)} log(a_j^{[L]}). \tag{8}$$

If a record i has label $y^{(i)} = c$, then the target value $t_c^{(i)}$ must be equal to 1 and $t_j^{(i)}$ for $j \neq c$ must be equal to 0. To minimize the loss function, the familiar optimization method called gradient descent is used due to its simplicity. The entire training process consists of two steps:

Forward propagation: First, a training record is fed through the network and the activations of every single neuron in the network are recorded.
Backpropagation: Next, the errors, or the loss, at the output nodes are passed back through the network and are used to update the weights along the way according to a gradient descent step.

The backpropagation step is arguably the most important step to understand when trying to comprehend neural networks. To update the weights that are further down the network, it is required to know the error at the corresponding hidden neurons. The errors at the hidden neurons are calculated based on the errors from the output layer. During implementation, the output errors are quite literally moved from the output layer through the hidden layers. For clarity, a gradient descent step for a random weight $w_{i,j}^{[l]}$ in the network can be determined using Formula 9:

$$\Delta w_{i,j}^{[l]} = -\alpha * \frac{\partial E}{\partial w_{i,j}^{[l]}}. \tag{9}$$

Where $\Delta w_{i,j}^{[l]}$ is the update to the weight $w_{i,j}^{[l]}$, E is the total loss function, and α is a predetermined learning rate. However, before one can train the weights, the weights must first be initialized. Due to the nature of the algorithm, if we initialize all the weights the same, for example all weights are initialized to zero, then all the updates to the weights will be identical. This would make a significant part of the network useless. Fortunately, this can be prevented by simply initializing the weights based on a random distribution such as the normal or the uniform distribution.

3.3 Neuron Specialization

The goal of any neural network is to train the output neurons such that they are only equal to 1 for records with the correct label and 0 otherwise. As such, one could say that these output nodes already have a role or specialization. However, for a specific output node to activate, it needs to use the features in the previous layer. As such, it would be ideal if these features act similarly to the output node. If there are neurons in the previous layer that also only equal to 1 for records with the correct label and 0 otherwise, then the output node can very reliably use these features. This is the premise behind this method. Neuron specialization is a technique where not only the output nodes, but also hidden nodes are trained to activate for specific classes. This boils down to a relatively simple extension of the loss function. Namely, a loss function is not only defined for the output layer, but also for the hidden layers. Thus, the total loss is the sum of the loss function of every single layer:

$$E_{spec} = \sum_{l=1}^{L} \tau^{[l]} E_{spec}^{[l]}. \tag{10}$$

Where E_{spec} is the total loss function in a specialized neuron network, $E_{spec}^{[l]}$ is the loss of layer l in a specialized neuron network, and $\tau^{[l]}$ is a predetermined variable known as the "importance factor" of layer l and is set somewhere between 0 and 1. The importance factor is necessary to alleviate the restriction of the extra loss terms in the hidden layers. Namely, one of the problems with a neural network is that the error generally decreases when backpropagated through the network due to the zero-mean initialization of the weights. As such, if the loss functions of the hidden layers are not properly scaled, then they might drown out the error from the output layer, the error that actually matters. This is also why the parameters are called "importance factors". Namely, the loss of the output layer is the most important since this determines the actual outputs. As such, the importance factor for this layer $\tau^{[L]}$ will be set equal to 1. The last hidden layer is not the layer that determines the output, but it does directly influence the output layer. As such the importance factor will be fairly large but not quite equal to 1. The idea is that the layers that are further away from the output layer directly influence the output loss to a lesser extent and their loss functions are therefore less important to focus on. Thus, one should have decreasing importance factors when going backwards from the output layer. Now suppose that every single layer in a neural network uses MSE as the loss function. Then, the total loss that needs to be minimized is equal to

$$E_{spec} = \sum_{l=1}^{L} \tau^{[l]} \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n_l} (t_j^{[l](i)} - a_j^{[l]})^2 = \frac{1}{n} \sum_{i=1}^{n} \sum_{l=1}^{L} \tau^{[l]} \sum_{j=1}^{n_l} (t_j^{[l](i)} - a_j^{[l]})^2. \tag{11}$$

To accommodate for the new neuron roles, the notation of $t_j^{(i)}$ has been changed to $t_j^{[l](i)}$, which now indicates the target value of neuron j in layer l for record i. These target values are determined by the roles of the neurons. The role of neuron j in layer l is indicated by $r_j^{[l]}$. This role is defined as an integer between 0 and n_L (exclusive). However, for reasons that will be explained later, it is advantageous to not only exclusively have specialized neurons in the network. As such, $r_j^{[l]}$ is defined such that it can also take the value of -1 to indicate a regular neuron. Using this information, the target variable can be defined as follows:

$$t_j^{[l](i)} = \begin{cases} 1, \text{ if } r_j^{[l]} = y^{(i)} \\ a_j^{[l]}, \text{ if } r_j^{[l]} = -1 \\ 0, \text{ else} \end{cases}$$

The new loss function described in formula 10 can again be minimized using the backpropagation method and gradient descent as explained in section 3.2. Fortunately, the new gradient descent step is easy to derive:

$$\Delta w_{i,j}^{[k]} = -\alpha * \frac{\partial E_{spec}}{\partial w_{i,j}^{[k]}} = -\alpha * \frac{\partial}{\partial w_{i,j}^{[k]}} \sum_{l=1}^{L} \tau^{[l]} E_{spec}^{[l]} = -\alpha * \sum_{l=1}^{L} \tau^{[l]} \frac{\partial E_{spec}^{[l]}}{\partial w_{i,j}^{[k]}}. \tag{12}$$

From Formula 12, it is easy to see that to determine the weight update, one can simply sum over the derivatives of each loss function for every layer. The term $\frac{\partial E_{spec}^{[l]}}{\partial w_{i,j}^{[k]}}$ is the exact same gradient as one would use for a loss function of a regular neural network and can be derived in the exact same way. Due to the fact that $\frac{\partial E_{spec}^{[l]}}{\partial w_{i,j}^{[k]}} = 0$ if $k > l$, this could be interpreted as backpropagating the error from each layer individually. Yet, a significantly simpler implementation exists. As explained in Section 3.2, backpropagation boils down to moving the error from the output layer back through the hidden layers and updating the weights along the way. Suppose the error from the output layer has been backpropagated to the last hidden layer. Now both the backpropagated error from the output layer, as well as the error of the specialized neurons in the current hidden layer need to be backpropagated further through the network. Thus, one can simply add these two errors together and backpropagate the resulting sum. As such, the backpropagation algorithm is adjusted as follows:

```
Algorithm 1 Backpropagation with specialized neurons:
Initialize $current\_layer$ equal to $L$.
Initialize $E_{spec}^{B}$ equal to a vector of zeros of length $n_L$. (This variable will be used to keep track of the backpropagated error)
Step 1: Calculate the error $E_{spec}^{[current\_layer]}$ at the neurons in the current layer.
Step 2: $E_{spec}^{B} = E_{spec}^{B} + \tau^{[current_l ayer]} E_{spec}^{[current\_layer]}$
Step 3: Propagate $E_{spec}^{B}$ back one layer and update the weights $W^{[current\_layer]}$ according to a gradient descent step. Set $E_{spec}^{B}$ equal to the backpropagated errors.
Step 4: $current\_layer = current\_layer - 1$
Step 5: If $current\_layer = 0$, stop. Else, go back to step 1.
```

This implementation adds very little complexity to the algorithm compared to the regular backpropagation method. The only difference being that one must calculate the errors at every layer instead of only the output layer, but this takes very little time in practice. Additionally, if the hidden layers in the neural network do not contain any specialized neurons, then the $E_{spec}^{[current\_layer]}$ term will simply be equal to zero at the hidden layers. Thus, the regular backpropagation algorithm can be recovered.

Another aspect that is worth mentioning is weight initialization in the case of neuron specialization. By all means, one can simply initialize all the weights completely randomly as is done with regular neural networks. However, the fact that the behaviour of specialized neurons can be predicted is information that can be exploited. Namely, if neurons in different layers have the same role, then it is to be expected that these neurons have similar activation patterns. Thus, it is to be expected that these neurons will eventually have strong positive connections between them. As is often said in the fields of psychology and neuroscience, *"Neurons that fire together, wire together"*. Using the fact that one can predict which neurons will fire together, an improved initialization can be achieved by giving these neurons strong positive weights from the start. Similarly, specialized neurons that do not have the same role can be given negative weights.

As mentioned before, it is not optimal to create a neural network with only specialized neurons. The reason for this is the fact that the constraints on these neurons are quite restrictive. Essentially, a specialized neuron is trained to activate *only* for a specific class and no others. This implies two things. First, the pattern captured by this neuron has to exist for every single record of that class. While this makes the feature quite reliable, this also means that features that only exist within a part of the class data cannot be captured even though those patterns are exclusive to that class. Furthermore, features that are shared between classes cannot be captured by specialized neurons. The latter problem can be solved by simply introducing neurons are specifically meant to extract shared features. Although, this does significantly increase the amount of neurons needed to capture all the shared features due to the many combinations of classes possible. For ten classes this is manageable. However, assume there are 100 classes. Then there are $\binom{100}{2} = 4950$ different combinations of two classes. Let alone if one wants to include features shared between three or more classes. Both problems are in part solved by the importance factors $\tau^{[l]}$ since these alleviate the strict restrictions of the specialized neurons. Yet, another solution for both problems is to create a hybrid network consisting of both the very specific specialized neurons and the highly flexible standard neurons.

3.4 Implementation and Testing

The methods mentioned are implemented using the programming language *Python* (Rossum, 1995). The methods are implemented from scratch using only the highly useful mathematical computation packages from the *SciPy* library (Jones et al., 01) as support. The *Python* implementations for the models are based on code from a previous paper (Brauwers et al., 2019). The methods are tested on a 3.4 GHz Intel Core i5 processor.

The exploratory research into the workings of a neural network is performed using two brand choice datasets about Catsup purchases and Cracker purchases due to their simplicity. The neural network implementation is evaluated on performance and compared to a multinomial logistic regression model for reference. Furthermore, the interpretation possibilities of both models are also analysed. After, the neural specialization is implemented to see how this influences the performance and interpretation of the neural network. To get more definitive results, the neuron specialization method is also tested using two image classification datasets focussing on handwritten digit recognition and clothing item classification.

To tune the hyper-parameters for the models used in this paper, cross-validation is used. The test sets are only used to obtain the final results and not during the process of tuning the model parameters. For the two image classification datasets, the data has already been split into a training and test set. As such, a 5-fold cross validation with stratified sampling from the training data is simply implemented. However, the marketing datasets first have to be split up into testing and training sets. For the Catsup dataset, the last two records of each individual is used as the testing set and the two purchases before those are used as the validation set during cross-validation. For the Cracker dataset, a similar split of the data is made but with the last five purchases of each individual. This way the test sets and the validation sets for both datasets contain approximately 20% of the data.

4 Data

4.1 Brand Choice Datasets

The two marketing datasets used in this research contain cross-sectional data about Catsup and Cracker purchases and are accessed through the *R* (R Core Team, 2017) package *Ecdat* (Croissant, 2016). The Catsup dataset (McFadden and Train, 2000) contains data from multiple purchase decisions of 300 individuals combining to a total of 2798 observations. There are 4 possible brand choices with each brand having 3 specific marketing-mix variables. The *display* variable is a binary variable that indicates whether the brand is on display at the time of purchase. The *feature* variable is also a binary variable that indicates whether there is a newspaper feature advertisement for that brand. Finally, the *price* variable is a continuous variable that indicates the price of that brand. Thus, in total the Catsup dataset contains 12 explanatory variables.

The Cracker dataset (Jain et al., 1994) contains data from 136 individuals granting a total number of 3292 purchase observations. Again, this dataset contains 4 brand choices with each brand having 3 marketing-mix variables. These marketing-mix variables are defined exactly the same as for the Catsup dataset. Both datasets are adjusted such that each purchase record also contains the marketing-mix variables of the previous two purchase records so that patterns through time can also be investigated. To investigate brand loyalty, the purchase decisions of the previous two purchases are also included in the form of dummy variables.

4.2 Image Classification Datasets

The first image classification database that is used is the well-known Machine Learning dataset MNIST constructed by LeCun and Cortes (2010). This database is a modified subset of the National Institute of Standards and Technology (NIST) database and is famous for being a highly suitable dataset for testing Machine Learning methods on. It contains 70000 images of handwritten digits. These images are split up into a training set of 60000 images and a testing set of 10000 images. The images of the digits are of size 28 by 28 pixels where each pixel is represented by a single grayscale value ranging from 0 (white) to 255 (black).

An alternative to the MNIST database is the Fashion-MNIST database released by Xiao et al. (2017). Xiao et al. (2017) argue that there are some significant shortcomings with the MNIST dataset. Namely, the images are generally too easy to classify and ideas tested on MNIST may not transfer well to other datasets. As such, the Fashion-MNIST dataset was designed to be a direct replacement for MNIST. Hence, this database is exactly the same in structure and format as the MNIST set, but it contains images of clothing articles instead of handwritten digits. Similarly to MNIST, it has a training set of 60000 images and a testing set of 10000. The images are all of format 28 by 28 pixels with each pixel being represented by a grayscale value. Finally, both the MNIST and the Fashion-MNIST datasets are formatted such that the images become row vectors with the image label at index 0 and the pixel values after that.

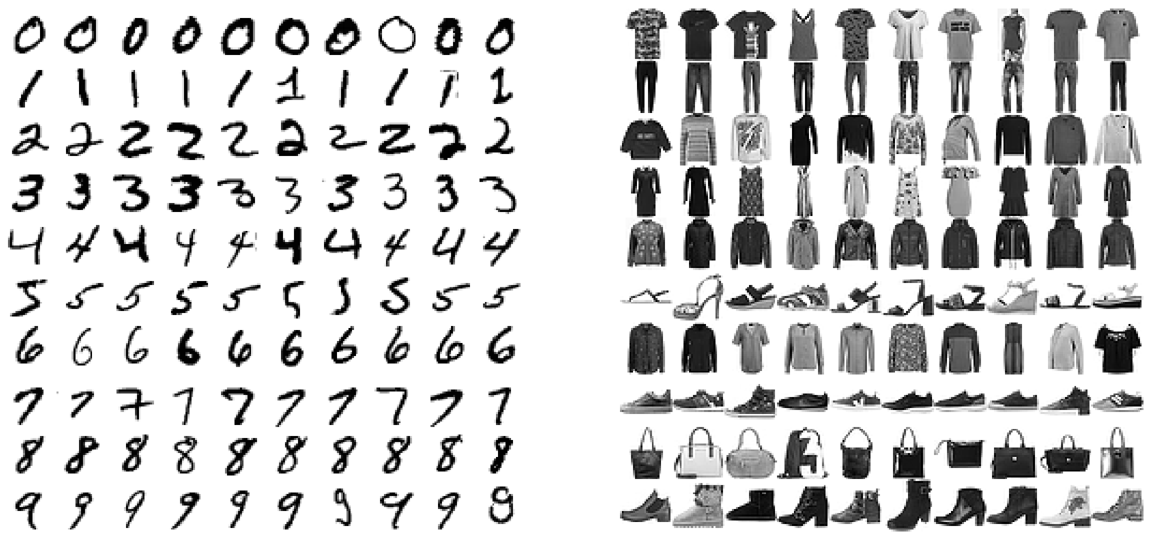

Figure 3: Example images taken from MNIST (left) and Fashion-MNIST (right) training sets.

The pictures found in Figure 3 are the first ten pictures of every class found in the training sets. Starting with the MNIST images, one can observe that there is quite a bit of variety in writing style, especially in the slanting of the digits and the way the loops are formed. Nevertheless, the classes are still quite easily differentiated from each other. The Fashion-MNIST images, on the other hand, contain significantly more variation within the classes themselves. There is a large variety in the shapes and patterns of different sandals, bags and dresses, for example. There are also many different shades present within the classes which is significantly less prominent in the MNIST images. These factors indicate that the Fashion-MNIST data poses a larger challenge for the classifiers than the MNIST dataset.

5 Results

In this section, the results from implementing the various models on the different datasets will be discussed. Section 5.1 starts with a performance comparison between the multinomial logit model and the neural network model based on the marketing datasets. A comparison based on the interpretation possibilities is also made. Then, the neuron specialization extension of the neural network model is evaluated. Section 5.2 continues with a similar analysis for the same methods based on the image classification datasets.

5.1 Brand Choice Prediction

Similarly to Agrawal and Schorling (1996), the neural network setup used for the marketing datasets is fairly small with only one hidden layer consisting of 20 neurons. The neural network models are trained with mini-batch gradient descent with a batch size of 32 and a decaying learning rate. The learning rates for the Catsup dataset starts at 0.3 and quickly decays to 0.01. For the Cracker dataset, the learning rate starts at only 0.05 since the training process converges significantly faster for this dataset. The loss function that is minimized in this process is the cross-entropy loss function with a softmax activation function. The hidden neurons in the network have leaky ReLu activation functions with a leaky factor of 0.01. Furthermore, the input values are normalized between 0 and 1 to speed up convergence. The weights are initialized from a standard normal distribution that is scaled using the method proposed by He et al. (2015) to prevent exploding gradients; the weights are multiplied with a factor equal to $\sqrt{\frac{2}{n_l}}$, with n_l being the number of input neurons in the previous layer. The neural network training is stopped early after 100 epochs to prevent overfitting. Since both datasets contain choice-specific variables, the conditional logistic model is used to compare the neural network with. The log-likelihood function of the CNL model is maximized using the BFGS optimization method (Fletcher, 1987) using a line search based on the Wolfe conditions (Wolfe, 1969) to determine the step sizes. The CNL model is trained till convergence.

Table 1: Model results for the marketing datasets.

Catsup Dataset	Neural Network	CNL		Cracker Dataset	Neural Network	CNL
Fitting Time (s)	9.59	38.74		Fitting Time (s)	12.17	56.70
Prediction Time (ms)	0.00136	0.000845		Prediction Time (ms)	0.00124	0.000701
Error Rate (%)	31.67	30.83		Error Rate (%)	19.56	18.68

Firstly, one would expect the neural network to outperform the CNL model based on the error rate due to the neural network's improved ability to capture non-linear relations. Yet, the error rates of the CNL model are lower for both datasets. This could be explained by the fact that the neural network has significantly many more parameters that allow the model to overfit to a much higher degree. Another explanations could be that the CNL model performs better simply due to the BFGS algorithm being superior to the gradient descent algorithm. Although the BFGS algorithm is also the reason for the fitting times being significantly higher for the CNL model. However, fitting times are generally not extremely important in practice since one only has to train the model once and the times achieved here are below one minute anyway. What does matter in practice however, is the prediction time per record. If large amounts of records need to consistently be classified, then it is important that the prediction times are as low as possible. As one can observe, the prediction times of the neural network are significantly higher due to the extra layer of calculations.

Sometimes one needs to predict records as accurately as possible, yet other times one may want to find out how the explanatory variables influence these predictions. This is the concept of “prediction versus explanation" and shows that not only the prediction performances are important, but also the interpretability of the model. As such, the next step is to compare the interpretation possibilities for both models. The model interpretations are compared using the coefficient estimates learned for the Cracker dataset. The CNL coefficients for this dataset can be found in Table 2. The subscripts of the variables indicate the time at which the variable was recorded.

Table 2: CNL model coefficient estimates.

Variable	$display_t$	$feature_t$	$price_t$	$display_{t-1}$	$feature_{t-1}$	$price_{t-1}$	$purchase_{t-1}$	$display_{t-2}$	$feature_{t-2}$	$price_{t-2}$	$purchase_{t-2}$
Coefficient	0.1674	0.6303	-2.9867	-0.06854	-0.1128	1.7502	1.4791	0.06041	-0.09656	0.92269	1.0915

The coefficients from the CNL model can very easily be interpreted using log odds ratios as defined in Formula 6. For example, since the estimated coefficient for $price_t$ is negative, if the price for brand *sunshine* is higher than the price of brand *kleebler* at time t, then the probability of the individual choosing for *sunshine* decreases. Similarly, since the estimated coefficient for $purchase_{t-1}$ is positive, if the individual bought brand *sunshine* at time $t-1$, the probability of the individual choosing *sunshine* again increases. These are all logical patterns that are to be expected. The interpretation of the coefficients of a neural network is substantially more difficult. Tracking the exact influence of a specific variable on a particular class is challenging due to the many connections and the activation functions. All one can really do is investigate the neurons in the first hidden layer of the network. The coefficient estimates for a single neuron from the hidden layer of the neural network trained on the Cracker dataset are shown in Table 3.

Table 3: Neural network weight estimates for a single neuron.

	$display_t$	$feature_t$	$price_t$	$display_{t-1}$	$feature_{t-1}$	$price_{t-1}$	$purchase_{t-1}$	$display_{t-2}$	$feature_{t-2}$	$price_{t-2}$	$purchase_{t-2}$
sunshine	-0.0699	0.0339	-0.2367	-0.1549	-0.3591	-0.1260	0.3110	-0.0396	0.0930	0.1149	-0.3586
kleebler	-0.0812	-0.2636	-0.1149	0.4369	-0.2977	-0.5134	0.2135	-0.0720	0.2152	-0.3760	-0.0480
nabisco	-0.0730	-0.1876	0.2336	-0.2297	0.2897	-0.3046	0.1884	-0.4220	-0.0118	-0.1735	0.4838
private	-0.4824	0.1027	0.0729	0.0592	0.0106	0.5593	-0.0150	-0.0102	0.1265	0.1137	-0.0416

Each hidden neuron has a weight attached to every single input neuron, which can make interpretation confusing. For example, the coefficients of $purchase_{t-1}$ and $purchase_{t-2}$ are quite large for brand *nabisco*. Since individuals often like to buy the same brands as also demonstrated by the

CNL model, this would indicate that this neuron will have a positive activation when a *nabisco* prediction is made. Yet, the $price_t$ also has a positive coefficient for the *nabisco* brand. Apparently, this neuron has a higher activation when the individual has chosen *nabisco* in the past, but also when the price of *nabisco* is high. Such patterns generally do not make sense to us, yet the model still uses such patterns to correctly classify records. The neural network model does not seem to perform particularly well on these datasets, but perhaps the neuron specialization can help the model.

The neuron specialization model has the exact same architecture as the standard neural network implementation, but half the neurons in the hidden layer are specialized. These specialized neurons use a mean squared error loss function since it works better with the ReLu activation functions and is significantly easier to calculate. Furthermore, the importance factor is set to 0.5. Also, the weight initialization is adjusted slightly to make use of the fact that the behaviours of the neurons are predictable. Namely, weights between specialized neurons are still initialized from the same normal distribution, but weights between neurons with the same role are made strictly positive and weights between neurons with different specializations are made strictly negative.

Table 4: Specialized neuron model results for the marketing datasets.

	Catsup	Cracker
Fitting Time (s)	10.82	11.47
Prediction Time (ms)	0.00131	0.00141
Error Rate (%)	31.50	18.97

Although the neuron specialization achieves slightly lower error rates than the standard neural network, it still does not outperform the CNL model. The fitting times and prediction times seem to be quite similar to the standard neural network. Again, it is also important to inverstigate the interpretability of the model. As such, the weights for a neuron specialized to extract patterns for the *nabisco* brand are presented in Table 5.

Table 5: Neural network weight estimates for a single neuron that is specialized to extract patterns for the *nabisco* class.

	$display_t$	$feature_t$	$price_t$	$display_{t-1}$	$feature_{t-1}$	$price_{t-1}$	$purchase_{t-1}$	$display_{t-2}$	$feature_{t-2}$	$price_{t-2}$	$purchase_{t-2}$
sunshine	0.0008	0.0093	0.2127	0.0614	0.0855	0.2440	0.0495	0.0300	0.0711	0.1280	0.0346
kleebler	0.0562	-0.1170	0.1535	0.0062	-0.0136	-0.2178	-0.0550	-0.0014	0.0273	-0.2095	0.1024
nabisco	0.0075	0.0719	-0.5609	-0.0317	-0.0208	0.0643	0.4104	0.0253	0.0722	0.1373	0.3159
private	0.0516	-0.0992	-0.0173	0.0230	-0.0335	0.0755	-0.1058	-0.0999	0.2171	0.0520	-0.0367

The patterns found within this specialized neuron are more logical than the ones found in Table 3. For example, when $price_t$ for nabisco increases, the neuron will have a lower activation value. Inversely, when the prices of other brands increase, the activation of this neuron also increases. Moreover, when the individual chose *nabisco* in the past, the activation of this neuron will also increase. These are patterns that are easy to understand and allow one to more easily find out which variables influence the choice for *nabisco* positively or negatively. The performances and the interpretability of the neural network model certainly seem to improve when neuron specialization is introduced. Yet, the CNL model still outshines the neural network model either way. To get more definitive answers on the effects of neuron specialization, the neuron specialization extension is also tested on datasets where the neural network model performs significantly better.

5.2 Image Classification

For the image classification datasets, various model architectures are tested for comparison purposes. Multiple standard three-layered neural network architectures are implemented with and without neuron specialization. Except for the learning rates and the hidden layer structures, the neural network models used for image classification have the exact same architecture as the ones used for the brand choice prediction datasets as it is a general classification architecture that can be used for any classification task. The learning rate used is equal to 0.05 for the MNIST dataset and 0.01 for the Fashion-MNIST dataset as this suits the data better. For the specialized neuron models, the importance factors for the hidden layers are, in order, 0.01 and 0.5. Due to time and hardware restrictions it was not possible to train every model till convergence. As such, every neural network model in this list has been trained for only 50 epochs which still produces adequate results. Since these datasets do not contain any class-specific variables, the MNL model is used to benchmark the neural network. The performances for the MNIST dataset are displayed in Table 6.

Table 6: MNIST test results. (NN = Neural Network, HU = Hidden Units)

Classifier	Test Error (%)	Prediction Time (ms)	Fitting Time (s)
MNL model	8.04	0.0112	2035.48
3-layer NN, 400 + 400 HU	1.86	0.0505	1680.03
3-layer NN, 400 + 400 HU, 50% specialization	1.50	0.0491	1722.41
3-layer NN, 800 + 800 HU	1.69	0.0763	3760.00
3-layer NN, 800 + 800 HU, 50% specialization	1.26	0.0715	3739.46
3-layer NN, 1200 + 1200 HU	1.71	0.0921	7594.12
3-layer NN, 1200 + 1200 HU, 50% specialization	1.34	0.0919	7751.95

First of all, it is clear that the MNL model is not the most suitable model for this problem. Although the classification rates are surprisingly high and the prediction times are lower than those of the neural network models, the error rates of the neural network models are significantly better. Furthermore, the error rate of every standard neural network model is improved significantly by adding neuron specialization. Moreover, even the specialized neuron model with only 400 neurons in each hidden layer has a lower test error rate than every single standard model. To investigate this increase in performance, it may be interesting to examine the test and training errors after every epoch and plot the error minimization process. In Figure 4, the training and test errors of the three-layered neural network with 800 hidden units in each layer have been plotted, with and without specialization.

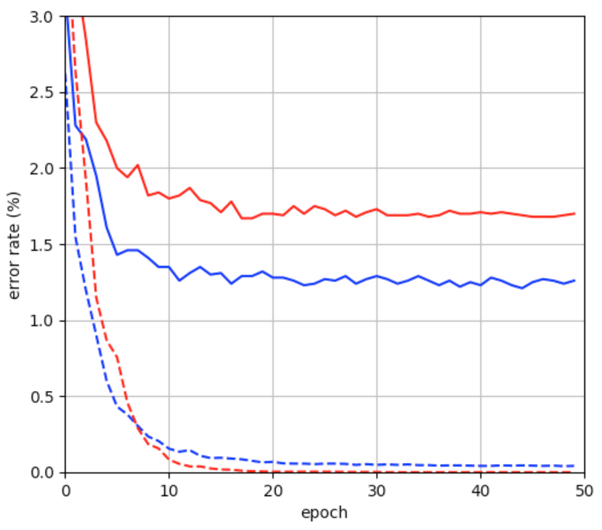

Figure 4: Plot of the test errors (full lines) and training errors (dashed lines) for the specialized (blue) and non-specialized (red) models.

Firstly, both models seem to overfit quite extensively. Yet, the addition of the specialized neurons still provides consistently lower test errors throughout the entire training process. Furthermore, the training error for the specialized neuron model is minimized faster in the beginning, but it quickly slows down and is overtaken by the training error of the regular model. This is most probably caused by the fact that the standard model fully focuses on minimizing the training error at the output neurons while the specialized neuron model must also focus on training the specialized neurons in the hidden layers. The same models used for the MNIST dataset have also been implemented on the Fashion-MNIST data. The results are shown in Table 7.

Table 7: Fashion-MNIST test results. (NN = Neural Network, HU = Hidden Units)

Classifier	Test Error (%)	Prediction Time (ms)	Fitting Time (s)
MNL Model	15.79	0.0101	1280.36
3-layer NN, 400 + 400 HU	10.15	0.0423	1612.59
3-layer NN, 400 + 400 HU, 50% specialization	9.96	0.0430	1629.43
3-layer NN, 800 + 800 HU	9.94	0.0770	3630.31
3-layer NN, 800 + 800 HU, 50% specialization	9.31	0.0829	3708.13
3-layer NN, 1200 + 1200 HU	9.77	0.0914	7534.73
3-layer NN, 1200 + 1200 HU, 50% specialization	9.23	0.0911	7762.42

Similar results as for the MNIST data are achieved for the Fashion-MNIST dataset. Each neural network model with neuron specialization sees a consistent improvement in test error rate compared to its standard non-specialized counterpart. Although, the relative improvement is smaller for this dataset. The MNIST models achieved approximately a 20 percent decrease in the amount of misclassified test records while the Fashion-MNIST models achieve only around a 5 percent decrease. For further comparison, the test error results of various other classifiers are shown in Table 8 and Table 9. From these tables one can see that the methods used in this paper are quite competitive compared to other methods.

Table 8: MNIST error rates of various classifiers implemented by Lecun et al. (1998).

Classifier	Error Rate (%)
2-layer NN, 300 HU	4.7
3-layer NN, 500+150 HU	2.95
K-NearestNeighbors, L2 Distance	2.4
SVM, Gaussian	1.4
CNN LeNet-5 with distortions	0.8

Table 9: Fashion-MNIST benchmark results. (Xiao et al.).

Classifier	Error Rate (%)
K-NearestNeighbors, L2 Distance	14.0
MLPClassifier, 100 HU	12.3
RandomForest, 100 estimators	12.1
GradientBoosting, 100 estimators	11.2
SVC, Poly	10.3

It is clear now that specialized neurons can improve performance of neural network models. The question that remains is why this is possible. In theory, the extension of the loss function should only make the model perform less adequately since it shifts the focus away from minimizing the actual output error terms. To understand why the specialized neurons still increase model accuracy significantly, one can inspect the behaviours of the neurons themselves. In Figure 6, the average activations of the specialized neurons are depicted. The model used for this is, again, the three-layered neural network with 800 hidden units in each layer with 50% specialization. The averages are calculated for each hidden layer separately to investigate the effects of the importance factors. The averages are obtained by first calculating the activations of the neurons for all the 10,000 records in the MNIST test set. Then, the averages are calculated for both the activations when neurons activate for the correct class they were assigned to, and the activations when they activate for the incorrect class. The averages are then taken over both the amount of neurons and the amount of records.

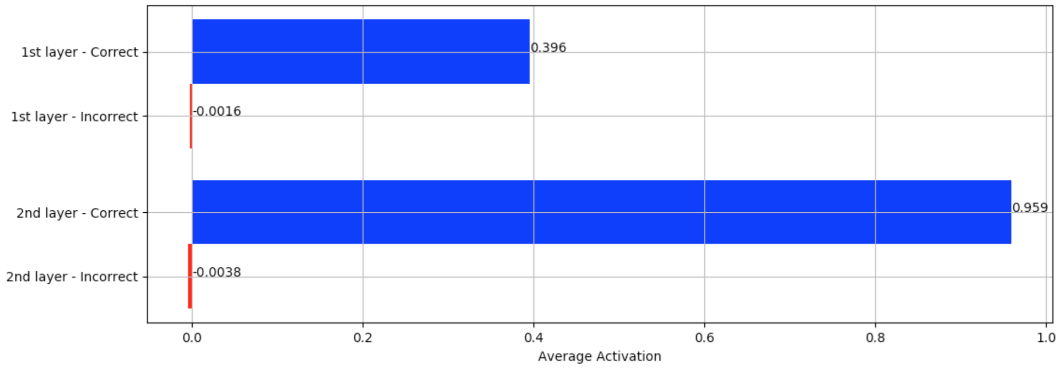

Figure 5: Bar chart depicting the average activations of the specialized neurons in the first and the second layer for the class that they were assigned to (correct) and the classes they were not assigned to (incorrect).

It is quite clear that the behaviours of the neurons are influenced quite significantly by the specializations. The neurons in the second layer on average behave almost perfectly as expected. They on average have an activation of almost 1 for the class they were assigned to and almost always have an activation of close to 0 for classes they were not assigned to. The neurons in the first layer do not fully behave as perfect, but this was to be expected due to the smaller importance factor of 0.01. Nevertheless, it is quite clear that the neurons, especially those in the second layer, are highly reliable neurons that the output layer can use to more easily correctly classify records.

This already shows how these specialized neurons have helped improve performance, but a closer inspection of the individual neurons is also insightful. In figure 6, typical images of both regular neurons and specialized neurons are depicted. Each square depicts the weights of neurons trained to extract features from the MNIST dataset. The rows in the right picture represent each digit in order, where each row contains neurons that are specialized to extract features for that specific digit.

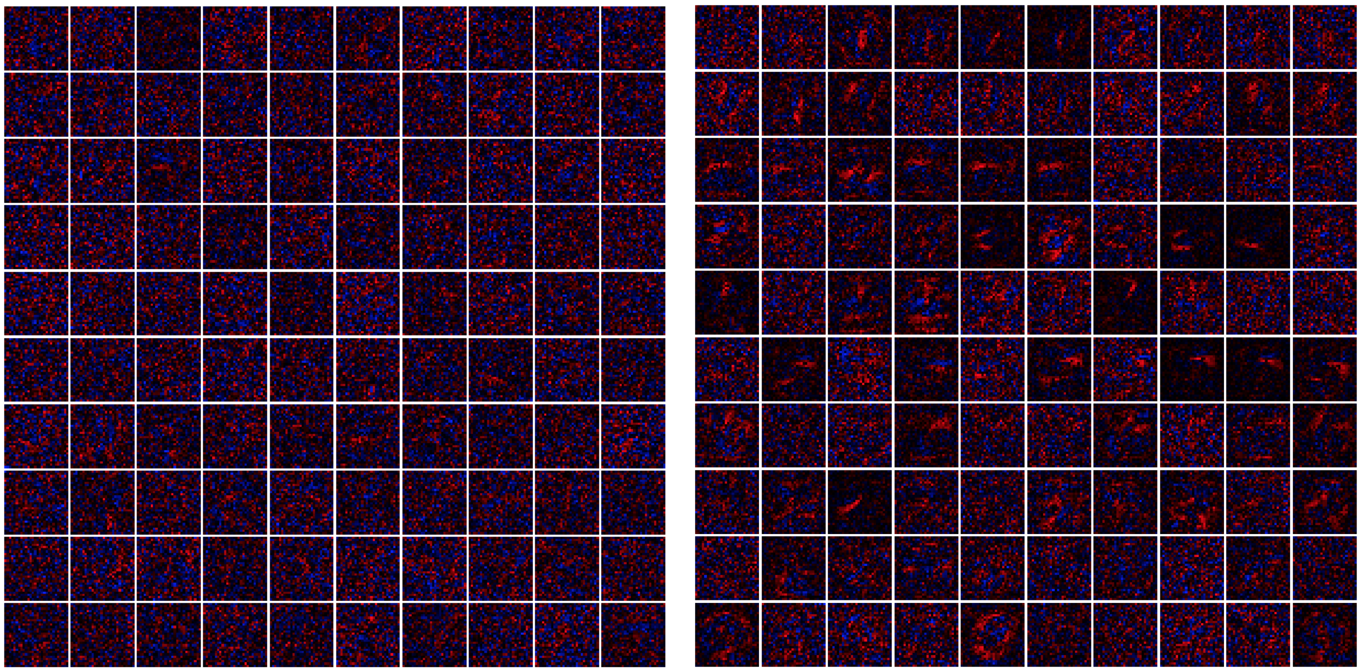

Figure 6: Typical example plots of the weights of regular neurons (left) and specialized neurons (right). Blue pixels indicate positive weights while red pixels indicate negative weights. The brighter the colour of the pixel, the larger the value of the weight.

At first sight, the depictions on both sides seem to be relatively noisy collections of pixels, however there are still patterns to be found. The pictures on the left seem to contain clusters and patterns of pixels depicting the various features of the different digits. Yet, it is particularly difficult to trace how these patterns fit into the shapes of the digits. The pictures of the specialized neurons, however, contain more well-defined and significantly easier to recognize patterns. Each neuron seems to represent a different form or aspect of the corresponding digit. For example, the first row contains neurons that extract features for the digit *zero*. One can see that, even though some images are more clear than others, each neuron in this row incorporates the circular shape of the digit in one way or another. Similarly for the neurons depicting the digit *one*, one can observe that each image consists of the general long familiar shape of the digit. Yet, each pattern is different in the specific shape, size or slanting and seems to represent a different aspect of the digit. The depictions of other digits are perhaps relatively more difficult to recognize, but the patterns are most certainly there. It is clear that the specialization of the neurons has had a significant effect on their behaviour.

6 Conclusion

In this paper, a novel machine learning technique known as “neuron specialization" was proposed. The research into this topic started with an introduction into the workings of a neural network by comparing it to a logistic regression model using two marketing datasets. In contrast to the paper by Agrawal and Schorling (1996), the results obtained in this paper from these datasets show that neural networks are not always preferable in a marketing context. For such a simpler problems, a better alternative may be a simpler model such as multinomial logistic regression which has significantly better interpretation possibilities and better performances. Nevertheless, the image classification datasets do show that neural network models perform significantly better when used for a more complex problem with more data. Thus, the answer to the first research question, *“How does an Artificial Neural Network perform compared to a standard classification method such as a Multinomial Logistic Regression model?"*, would be that neural networks are preferred for complex problems with more data, while multinomial logit models are preferred for simpler tasks such as brand-choice prediction. Just to be clear, brand-choice prediction is not exactly “simple" since predicting human behavior is extremely difficult. However, due to the small size of the data and the relatively small amount of variables, there are not many complex patterns to be found.

The second part of this research was to investigate the effects of implementing the neuron specialization extension. This extension was implemented and tested for both the marketing and the image classification datasets to answer the following research question: *“Can Artificial Neural Network models be improved through the use of neuron specialization?"* For the marketing datasets, the neuron specialization seems to improve the neural network only slightly and the multinomial logit model is preferred anyway. However, for the image classification datasets, the neural network with the neuron specialization extension implemented had consistently lower error rates and easier to interpret neurons. As such, it can be concluded that neuron specialization can improve classification neural networks.

The question that remains however is to what extent the neuron specialization can be utilized. The image classification datasets used in this paper were arguably easier than some other highly complex datasets such as ImageNet (Russakovsky et al., 2015) or the Open Images Dataset (Krasin et al., 2017). It is questionable if neuron specialization will work for these datasets. Namely, the Fashion-MNIST dataset has around ten times higher error rates than the regular MNIST dataset. With this increase in complexity, the relative performance boost that neuron specialization provides seems to drop. Furthermore, the datasets used in this research consisted of either four or ten classes. However, when a dataset contains thousands of classes, it is practically impossible to assign neurons to every single class. Due to hardware and time constraints, it was not possible to investigate the effects of such larger datasets in this paper. Thus, further research is required to find out if neuron specialization can work for larger and more complex datasets.

References

Agrawal, D. and Schorling, C. (1996). Market share forecasting: An empirical comparison of artificial neural networks and multinomial logit model. *Journal of Retailing*, 72(4):383–407.

Axelrod, V., Rozier, C., Malkinson, T. S., Lehongre, K., Adam, C., Lambrecq, V., Navarro, V., and Naccache, L. (2019). Face-selective neurons in the vicinity of the human fusiform face area. *Neurology*, 92(4):197–198.

Baltas, G. (2004). A model for multiple brand choice. *European Journal of Operational Research*, 154(1):144–149.

Bentz, Y. and Merunka, D. (2000). Neural networks and the multinomial logit for brand choice modelling: a hybrid approach. *Journal of Forecasting*, 19(3):177–200.

Bhatnagar, S., Ghosal, D., and Kolekar, M. H. (2017). Classification of fashion article images using convolutional neural networks. In *2017 Fourth International Conference on Image Information Processing (ICIIP)*, pages 1–6. IEEE.

Brauwers, G., Ngo, H., Riezebos, M., and Warners, A. (2019). Classification of handwritten digits by (convolutional) neuralnetworks.

Croissant, Y. (2016). Data sets for econometrics. https://cran.r-project.org/web/packages/Ecdat/. Accessed: 08-05-2019.

Dunne, R. A. and Campbell, N. A. (1997). On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer.

Fish, K. E., Johnson, J. D., Dorsey, R. E., and Blodgett, J. G. (2004). Using an artificial neural network trained with a genetic algorithm to model brand share. *Journal of Business research*, 57(1):79–85.

Fletcher, R. (1987). *Practical Methods of Optimization; (2Nd Ed.)*. Wiley-Interscience, New York, NY, USA.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Hines, T. (2018). Anatomy of the brain. https://mayfieldclinic.com/pe-anatbrain.htm.

Jain, D. C., Vilcassim, N. J., and Chintagunta, P. K. (1994). A random-coefficients logit brand-choice model applied to panel data. *Journal of Business & Economic Statistics*, 12(3):317–328.

Jones, E., Oliphant, T., Peterson, P., et al. (2001–). SciPy: Open source scientific tools for Python. [Online; accessed 12-04-2019].

Kepecs, A. and Fishell, G. (2014). Interneuron cell types are fit to function. *Nature*, 505(7483):318.

Kline, D. M. and Berardi, V. L. (2005). Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computing & Applications*, 14(4):310–318.

Krasin, I., Duerig, T., Alldrin, N., Ferrari, V., Abu-El-Haija, S., Kuznetsova, A., Rom, H., Uijlings, J., Popov, S., Veit, A., Belongie, S., Gomes, V., Gupta, A., Sun, C., Chechik, G., Cai, D., Feng, Z., Narayanan, D., and Murphy, K. (2017). Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

LeCun, Y., Bottou, L., Orr, G., and Muller, K. (1998). Efficient backprop in neural networks: Tricks of the trade (orr, g. and müller, k., eds.). *Lecture Notes in Computer Science*, 1524(98):111.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., Denker, J., Drucker, H., Guyon, I., Muller, U., Sackinger, E., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *International conference on artificial neural networks*, volume 60, pages 53–60. Perth, Australia.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., and Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, 234:11–26.

Maas, A. L., Hannun, A. Y., and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3.

McFadden, D. and Train, K. (2000). Mixed mnl models for discrete response. *Journal of applied Econometrics*, 15(5):447–470.

Ng, A. and Katanforoosh, K. Standard notations for deep learning. https://cs230.stanford.edu/files/Notation.pdf. Accessed: 20-06-2019.

Paap, P. H. F. . R. (2001). *Quantitative Models in Marketing Research.* Cambridge: Cambridge University Press.

Paap, R. and Franses, P. H. (2000). A dynamic multinomial probit model for brand choice with different long-run and short-run effects of marketing-mix variables. *Journal of Applied Econometrics*, 15(6):717–744.

Poulsen, C. S. (1990). Mixed markov and latent markov modelling applied to brand choice behaviour. *International Journal of Research in Marketing*, 7(1):5–19.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Rossum, G. (1995). Python reference manual. Technical report, Amsterdam, The Netherlands, The Netherlands.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Vroomen, B., Franses, P. H., and van Nierop, E. (2004). Modeling consideration sets and brand choice using artificial neural networks. *European Journal of Operational Research*, 154(1):206–217.

Wolfe, P. (1969). Convergence conditions for ascent methods. *SIAM Review*, 11(2):226–235.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion mnist benchmark dashboard. `http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/#`. Accessed: 22-06-2019.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Appendix

Code Directory

The code used in this paper is provided separately in a different file. As such, this section describes in short the directory and the scripts within.

```
Code
 ├── CNL
 │    ├── CNL_testing.py:
 │    │   Run this script to test the CNL model.
 │    │
 │    └── CNL.py:
 │        This script contains the implementation of the CNL model.
 │
 ├── MNL
 │    ├── MNL_testing.py:
 │    │   Run this script to test the MNL model.
 │    │
 │    └── MNL.py:
 │        This script contains the implementation of the MNL model.
 │
 └── NeuralNetwork
      ├── NeuralNetwork_testing.py:
      │   Run this script to test the Neural Network model.
      │
      └── NeuralNetwork.py:
          This script contains the implementation of the neural network model with neuron
          specialization that can be turned on or off.
```