# Fraud Detection using Neural Networks

## Introduction to Neural Networks and Autoencoders

Neural networks are expanding the limits of what machines can do and revolutionizing the way we solve complex problems. As a Master of Data Analytics student with a specialty in Finance, Banking, and Insurance, I developed a strong interest in the applications of neural networks in the financial industry. This project represented a great opportunity for me to deepen my understanding of neural networks and explore the applications of neural networks in the financial industry. The focus of this project is utilizing an effective neural network structure to detect credit card fraudulent transactions. Fraud detection is among the highest priorities for the financial industry, and neural networks can be leveraged to revolutionize how fraud detection works.

Before diving into the analysis and building neural networks to detect credit card fraud, we will first explain the concept of neural networks and how they work. To clearly understand the concept of neural networks we will first explain what is machine learning. Machine learning is the science of getting computers to learn and act without explicit programming. Neural networks are a type of machine learning algorithm that is inspired by the structure of the biological brain. The design of neural networks simulate densely interconnected brain cells inside a computer, so the computer can learn things and make decisions in a human-like way.

In this section, we will look at neural networks organization, components, hyperparameters, learning, and learning paradigms. We will start with neural networks organization.

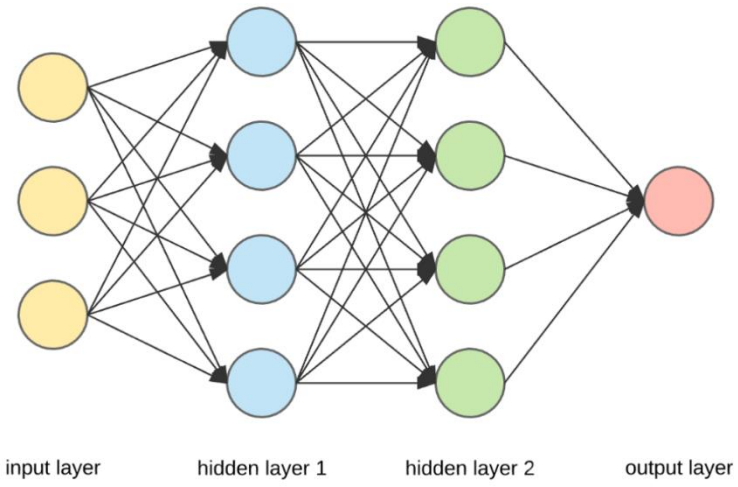



Image 1: Basic Neural Network Architecture

Neural networks are organized in multiple layers that consist of 3 types. An input layer, hidden layers, and an output layer. The input layer takes the initial data for the neural network. Then the hidden layers between the input and the output are where all the computations are done. Finally, the output layer produces the final result. Note that between any two layers in a neural network, it's possible to have multiple connection patterns. Given that we now know the basic organization of a neural network, we will look into the components that make a neural network. A neural network consists of neurons, weights, and propagation function.

> **Neurons:** A neuron is a mathematical function that models a biological neuron. Neurons are designed to receive an input such as an image, for example, combine that input with their internal state using what is known as an activation function, and then make an output using an output function. The initial input to the neurons comes from the data we are analyzing. That could be for example emails, images, or documents. The final output of the neurons is what accomplishes the task on hand. This could be classifying whether a transaction is fraudulent or not.

**Weights:** Neurons in a network are connected via one or more connections. The output of a neuron, in any layer of the neural network except the output layer, is the input to another neuron. Each connection between the neurons has a weight. This weight determines the relative importance of that connection.

**Propagation function:** A propagation function is designed to calculate the input to a neuron from the outputs of its predecessor neurons and their connections as a weighted sum. Sometimes a bias term is added to create a successful learning model. A bias value allows us to shift the activation function to the right or left and get a better fit for the data.

Here is a detailed image of a node in a neural network to further illustrate the function of each neural network component.

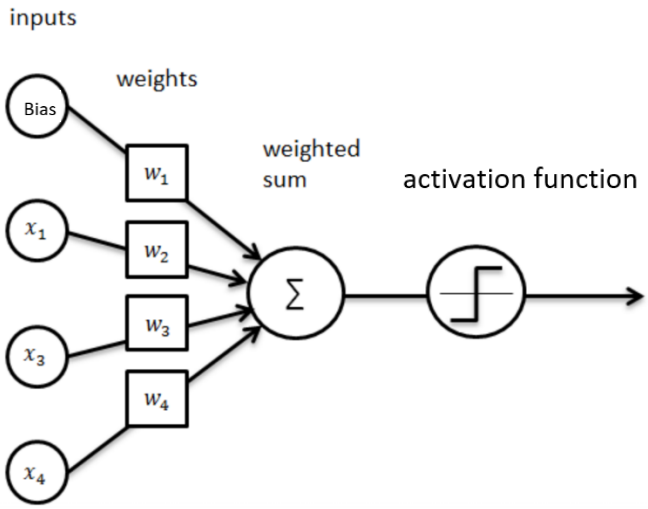



Image 2: A Node from a Neural Network

Next, we will look at what are hyperparameters and how they are used by neural networks. A neural network hyperparameter is a constant parameter whose value is set before the learning process begins. The learning process here refers to the process of optimizing the weights and bias terms in a network. Hyperparameters in neural networks include the number of hidden layers and learning rate. The number of hidden layers determines the network structure and the learning rate determines how the network is trained.

So far we have discussed neural network organization structure, components, and hyperparameters. Next, we will look at how neural networks learn. The process of learning for a neural network refers to how a network adapts by considering sample examples to better handle a task. The learning process involves adjusting the weights between the neurons of the network to minimize the observed errors and improve the result's accuracy. There are three important concepts related to the learning process of neural networks. Those concepts are learning rate, cost function, and backpropagation.

**Learning rate:** The learning rate refers to the size of the corrective steps that a model takes to adjust for errors in each observation. A low learning rate means that a neural network will take a longer training time but has a potential for greater accuracy and vice versa.

**Cost function:** A cost function measures how good a neural network output is using a training sample. In this project, we will use the mean square error as a cost function. This can be mathematically represented as:

$$C(\mathbf{y}, \mathbf{o}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - o_i)^2$$

where “y” is the observation sample and “o” is the network output.

**Backpropagation:** Backpropagation is a method used to adjust the weights that connect neurons in a network to compensate for each error found in the learning process. This can be done by calculating the gradient of the cost function with respect to the weights in the network.

Now we will look into neural networks' learning paradigms. There are two main learning paradigms: supervised learning and unsupervised learning.

**Supervised learning:** When the neural network is presented with a set of inputs along with the desired output labels, this is referred to as supervised learning. The network goal here is to discover a rule that enables it to map the input to the output.

**Unsupervised learning:** When the neural network is presented with a set of inputs but no desired outputs, this is referred to as unsupervised learning. The network goal here is to find structure and patterns on its own.

We have introduced important concepts to explain neural networks organization, components, hyperparameters, learning, and learning paradigms. This introduction helps us understand what are neural networks and how they work. In this project, we will use a type of neural networks called autoencoder. So before we dive into the analysis we will give a quick introduction to autoencoders using the concepts we explained above.

Autoencoders are a specific type of neural networks where the input is the same as the output. This type of neural network compresses the input into a lower-dimensional code, then reconstructs the output from this code. A code here basically means a compact summary of the input.

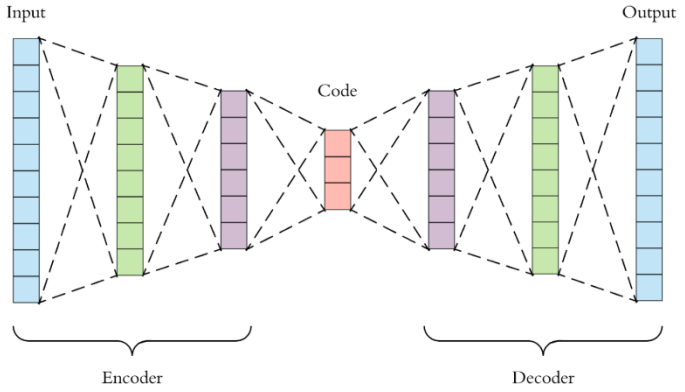



Image 3: Basic Autoencoder Architecture

Autoencoders are designed with restrictions that force them to approximate the output from the input instead of duplicating the information. In other words, they are designed to preserve only the most relevant aspects of the data. This can be achieved by restricting the number of nodes in the middle layer to be smaller than the nodes in the input layer. The basic autoencoder architecture consists of an encoder, a decoder, and a code. The encoder and decoder can be defined as transitions $\phi, \psi$ such that:

$$\phi : \mathcal{X} \to \mathcal{F}$$
$$\psi : \mathcal{F} \to \mathcal{X}$$
$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \|X - (\psi \circ \phi)X\|^2$$

where $X$ is the given input. In a simple case with one hidden layer the encoder takes an input $\mathbf{x} \in \mathbb{R}^d = \mathcal{X}$ and maps it to $\mathbf{h} \in \mathbb{R}^p = \mathcal{F}$: $\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$, where $\mathbf{h}$ is the code, $\sigma$ is the activation function, $\mathbf{W}$ is the weight matrix, $\mathbf{b}$ is a bias vector, and $\mathbf{x}$ is an input. The weight matrix and bias vector of an autoencoder are initialized randomly and then updated through backpropagation. The decoder stage maps $\mathbf{h}$ to the reconstruction $\mathbf{x}'$ of the same shape as $\mathbf{x}$: $\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$, where $\sigma'$, $\mathbf{W}'$, and $\mathbf{b}'$ for the decoder may be unrelated to the corresponding $\sigma$, $\mathbf{W}$, and $\mathbf{b}$ for the encoder. In the training process, the autoencoders aim to minimize a cost function/loss function such as the square errors where the loss can be defined as:

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

where $\mathbf{x}$ is averaged over some input training set.

When initializing an autoencoder there are 4 hyperparameters to specify. Those hyperparameters are code size which is the size of nodes in the middle layer, number of layers, number of nodes per layer, and a cost function. Furthermore, autoencoders don't require labeled inputs and therefore can be used as an unsupervised learning model.

## Dataset

The dataset used in this project comes from Kaggle and was uploaded by researchers from the Université Libre de Bruxelles in Belgium. The name of the dataset is "Credit Card Fraud Detection, Anonymized credit card transactions labeled as fraudulent or genuine". There is a link to the dataset in the references section. The dataset contains credit card transactions by European cardholders. The transactions in the dataset occurred during two days in September 2013. Here are some quick facts about the dataset:

- Total number of transactions in the dataset are 284,807
- The dataset has 492 fraudulent transactions
- The dataset is highly unbalanced since the number of fraudulent transactions only accounts for 0.172% of all transactions in the dataset

The original features in the dataset were not provided for confidentiality except for the features Time and Amount. Besides Time and Amount, the dataset only contains numerical input variables that are a result of principal component analysis(PCA) transformation. PCA is a dimensionality reduction technique that applies a transformation to a given set of training examples by projecting into the vector space generated by the eigenvectors of the covariance matrix. PCA aims to find the most relevant features in the data, so we can get a simplified representation of it. The table below breakdown the features in the dataset and their descriptions.

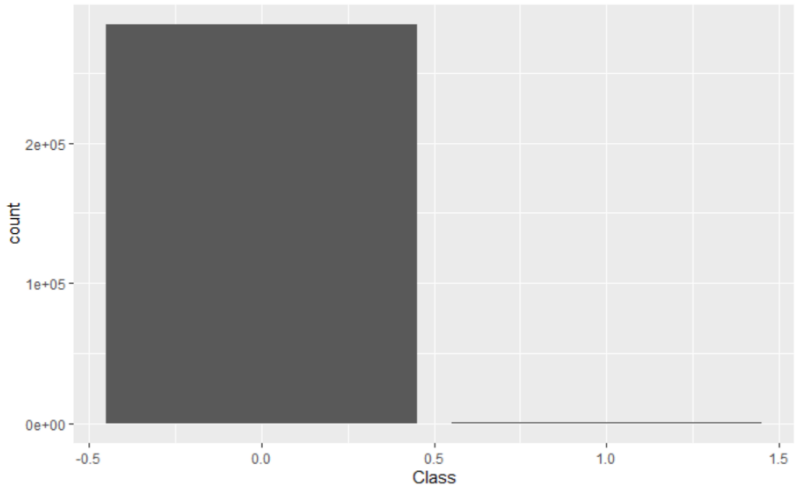
| Feature | Description |
|---|---|
| **V1, V2, V3, …. , V28** | The principal components obtained from PCA |
| **Time** | Seconds elapsed between each transaction and the first transaction in the dataset |
| **Amount** | Transaction amount |
| **Class** | Response variable which is either 1 to indicate a fraudulent transaction or 0 otherwise |

## Exploratory Data Analysis

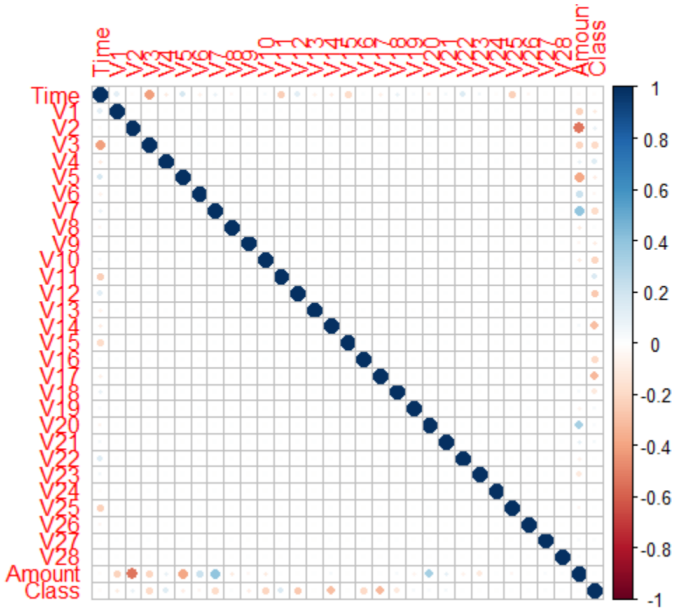
In this section, we will focus on exploratory data analysis to gain a higher-level understanding of our dataset. However, before we start with the exploratory data analysis we will do some data cleaning to ensure the quality of our data. For the data cleaning step we have:

- Verified that there are only two binary classes in the dataset. Class 1 represents a fraudulent transaction and 0 represents a genuine transaction. The R code used to verify the number of classes can be found in section 3.1 of the R code appendix.
- Verified that the timestamps for all transactions in our dataset correspond to two days as described on Kaggle. The R code used for this step can be found in section 3.2 of the R code appendix
- Verified that there are no missing values in the dataset. The R code used for this step can be found in section 3.3 of the R code appendix

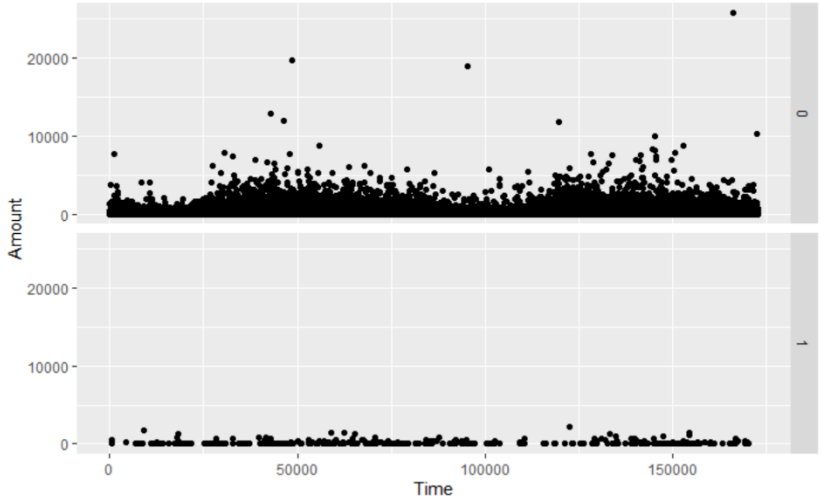
Now that we have completed our data cleaning step, we will move on to the exploratory data analysis. We will start by plotting a bar chart to get a visual of the distribution of the binary classes in our dataset.



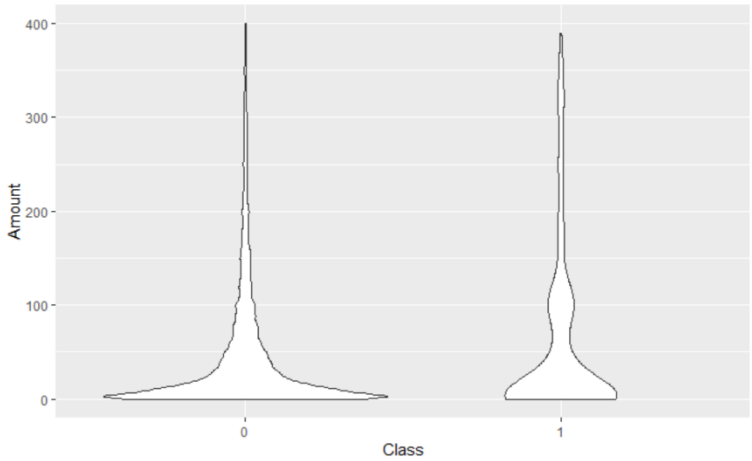

As we can see from the bar chart our dataset is highly unbalanced. Most of the transactions in the dataset are genuine. Knowing that our dataset is highly unbalanced will play an important role in the analysis. Next, we look at features correlation in the dataset through a correlogram.




The correlation matrix shows that none of the V1 to V28 PCA components have any correlation with each other. We also notice that there are some positive and negative correlations between Class and some V components. However, there is no correlation between Class and Time or Class and Amount. Next, we look at how the amount of each transaction varies by time for each binary class.



The pattern for the two classes appears to be similar. We can see from the plot that fraudulent transaction amounts are lower than the amounts of genuine transactions. Perhaps this happens because transactions under a certain amount don't require a PIN as the criminal can use the Tap To Pay feature of the credit card. Also, maybe this is a strategy so that the credit card owner doesn't notice the fraudulent transactions. Let's filter out all transactions below 400 and examine transaction amounts by class to gain deeper insights.




It appears that fraudulent transactions are anomalously centered around 100. In Canada, we know that the maximum amount to use the Tap To Pay without entering a PIN is 100. Perhaps, it's the same in Europe and financial criminals can steal credit card information but not the PINs. This can happen if the credit card information was stolen using wireless scanners in public areas for example.

## Data Analysis

We will start the exciting part of this project which is building an autoencoder neural network to detect credit card fraudulent transactions. Before we build the network, we want to explain the logic behind choosing autoencoder neural networks for this project. Our dataset comes with a binary classification for transactions to indicate whether the transaction was a fraud or not. We know that our dataset is unbalanced but by using the right sampling techniques, we could have implemented logistic regression, random forests or support vector machines to analyze the dataset. However, there are two main advantages that autoencoders offer over those techniques:

- Autoencoders don't require labeled datasets. They are algorithms for unsupervised learning. In real life, it's difficult and sometimes unrealistic to get labeled data for fraudulent transactions. Using autoencoders will allow us to analyze the data using techniques that are useful and applicable to real scenarios in the financial industry.
- Even when we have a labeled dataset for fraudulent transactions, the accuracy of those labels is questionable in complex scenarios. Let's say that Bob is a criminal and decided to commit fraud using their account. After the fraudulent transaction, Bob has completed normal activities in his account. Should a financial institution flag all of Bob's transactions or only a subset? Many may argue that the transactions were committed by a criminal. Therefore, all account activities should be flagged. This can introduce bias into the model. In these complex scenarios, relying on the labels in the dataset may not lead to accurate analysis. Instead, we should approach the problem as an anomaly detection problem. This is what autoencoder neural networks enable us to do.

Before we build our autoencoder neural network and train our model we will split our dataset into a test set and a training set. Splitting the data into a test set and a training set will enable us to evaluate how well the model does with data outside the training set. We chose to use 80% of the data as a training set and 20% of the data as a test set. Also, following the reasons we discussed above we decided to treat this problem as an

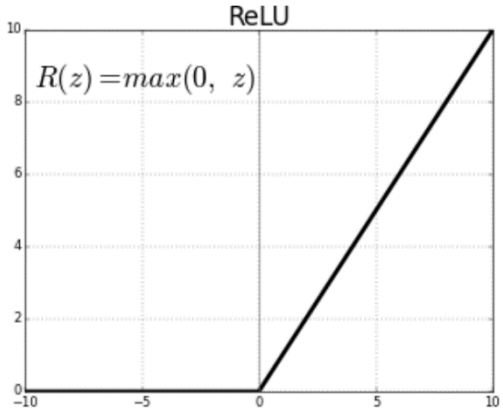
unsupervised problem. Therefore, we will be ignoring all the labels in the training dataset by excluding the Class and Time feature from the training dataset. The R code used to split the data into two sets and exclude the labels from the data can be found in section 5.1 in the R code appendix.

Now that our dataset is ready we will build our autoencoder neural network in R using Keras, which is an open-source neural network library. We are going to experiment with different autoencoder neural network architectures to compare which architecture yields the best results.

- **First autoencoder neural network architecture(6 layers in total, 3 encoders, and 3 decoders):**

| Network component | How many |
|---|---|
| Encoder: Neurons in the input layer | 29 |
| Encoder: Neurons in the outer encoder | 15 |
| Encoder: Neurons in the inner encoder | 10 |
| Decoder: Neurons in the inner decoder | 10 |
| Decoder: Neurons in the outer decoder | 15 |
| Decoder: Neurons in the output layer | 29 |

The R code for this network is in section 5.2 in the R code appendix. For this architecture, we used ReLU as an activation function. Here is a graph illustrating how the ReLU function behaves.

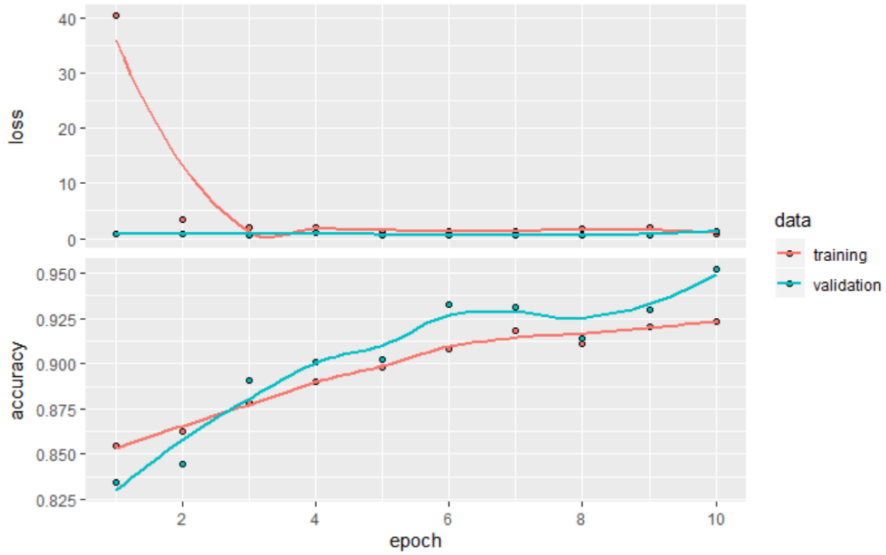



As we see from the graph the ReLU is half rectified. The value of the function is zero when the input is less than zero. Then the value of the function is equal to the input when the input is zero or above. Furthermore, for the network hyperparameters we will use mean square error as a cost function, and a batch size, which is the number of training examples in one pass, of 30. The fit function in the Keras library also enables us to specify the number of epochs when fitting the model. We will set the number of epochs to 10. Moreover, there is an option in the Keras neural network fit function to set a part of the training data as a validation dataset. Validation data is not the same as test data, which we set aside earlier. The validation data here is used to evaluate the cost function at the end of each epoch. For this neural network, we will set the validation data split to 0.2. This means that 20% of the training data will be used as a validation dataset. Here is a summary table of the parameters we set to train the model:

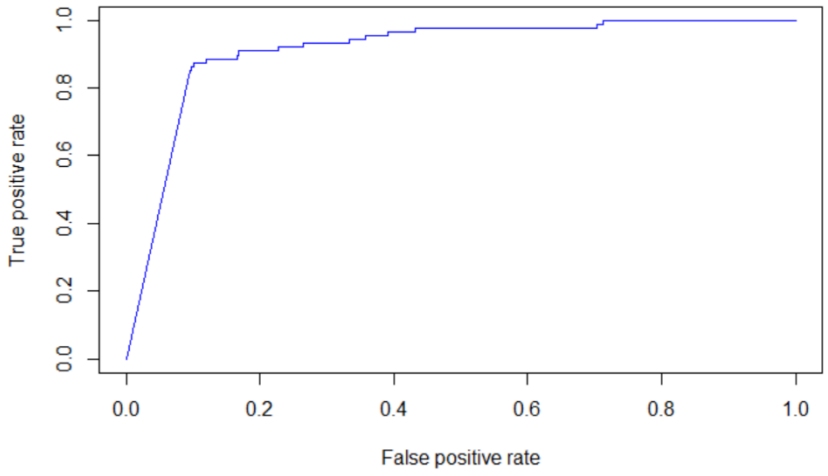
| Parameter | Value |
|---|---|
| Cost function | Mean square error |
| Batch size | 30 |
| Epochs | 10 |
| Validation Data Split | 0.2 |

The graph below shows how the loss and accuracy of the network change with each epoch. There is a clear trend showing there is an increase in accuracy as epoch increases. We also can see that the rate

at which the accuracy increase for the training data starts to decrease after epoch 8. However, that rate increase for the validation dataset.




We will now use our trained network to make predictions using the test dataset that we set aside before the network training. Then, we will look at anomalously large reconstruction errors to determine the accuracy of the network. We will define a large reconstruction error to be above 40. We came to find that this threshold for reconstruction errors is suitable by experimentation. Finally, we will plot a precision-recall curve to evaluate the model's performance. We decided to use a precision-recall curve instead of a confusion matrix because of the high class ratio imbalance in the dataset. The R code for making predictions using the trained network, defining the threshold for reconstruction errors, and plotting the precision-recall curve is in section 5.3 of the R code appendix.




We see from the plot above that the model performance is quite satisfactory. Let's see if we can improve the results by trying another autoencoder neural network architecture.

- **Second autoencoder neural network architecture(10 layers in total, 5 encoders, and 5 decoders):**

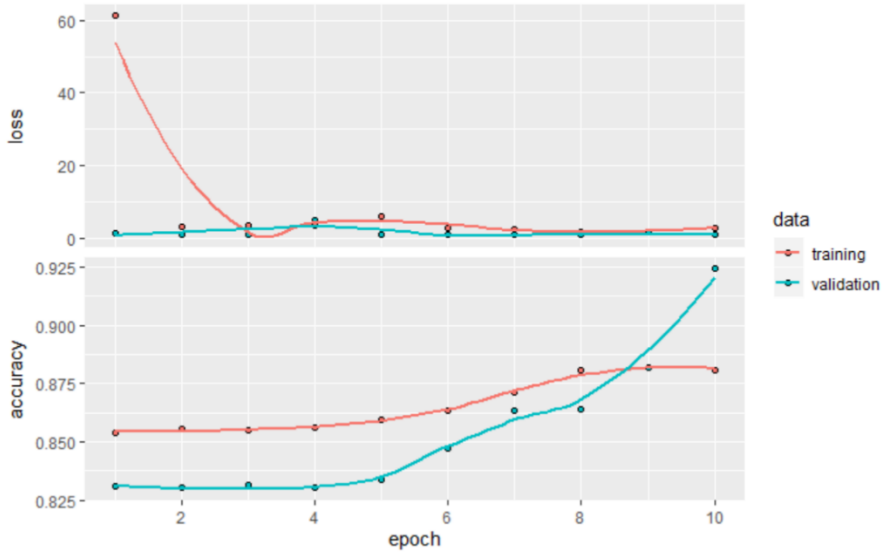
| Network component | How many |
|---|---|
| Encoder: Neurons in the input layer | 29 |
| Encoder: Neurons in the second encoder | 20 |
| Encoder: Neurons in the third encoder | 12 |
| Encoder: Neurons in the fourth encoder | 6 |

| | |
|---|---|
| **Encoder: Neurons in the fifth encoder** | 3 |
| **Decoder: Neurons in the fifth decoder** | 3 |
| **Decoder: Neurons in the fourth decoder** | 6 |
| **Decoder: Neurons in the third decoder** | 12 |
| **Decoder: Neurons in the second decoder** | 20 |
| **Decoder: Neurons in the output layer** | 29 |

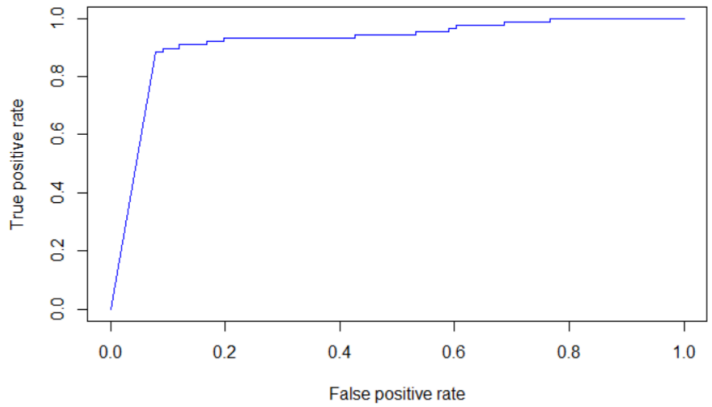
The R code for this network is in section 5.4 in the R code appendix. Similar to the first neural network we used ReLU as an activation function. Here is a summary table of the network parameters we chose:

| Parameter | Value |
|---|---|
| **Cost function** | Mean square error |
| **Batch size** | 30 |
| **Epochs** | 10 |
| **Validation Data Split** | 0.2 |

The graph below shows how the loss and accuracy of the network change with each epoch:




There is an increase in accuracy and a decrease in the loss as the epoch increase. We note that up until approximately the 9$^{th}$ epoch the training accuracy was higher than the validation accuracy. Then the validation accuracy increased significantly to surpass the training accuracy. We will get a closer look at the network performance through a precision-recall curve.




The performance of the network is pretty reasonable, but not as good as the first network we built. We will compare the performance of the two networks in-depth in the discussion section. The R code for making predictions using this trained network, defining the threshold for reconstruction errors, and plotting the precision-recall curve is in section 5.5 of the R code appendix.

## Discussion

We built two autoencoder neural networks to analyze credit card transactions to identify fraudulent transactions. Both of the neural networks achieved an accuracy of more than 90%. However, the first network architecture yielded better results. We designed the first neural network with 6 layers in total. Three of those layers are encoders and the other three are decoders. We also designed the network to have gradually decreasing neurons in each layer up until the last encoder layer. Then we gradually increase the number of neurons up until the output layer. The reason behind this design is that we wanted each subsequent layer of the encoders to have fewer neurons than the layer preceding it. This is to prevent the network from learning the identity function and copying the information instead of compressing it. The decoder layers gradually decoded the information that the encoder layer compressed. The second neural network was designed with more layers. There was a total of 10 layers. Five of those layers are encoders and the other five were decoders. We also designed the second network to have a gradually decreasing neurons in each layer up until the last encoder layer. Then we gradually increase the number of neurons up until the output layer. The results show that the first network yielded better results than the second network. We believe that this is because the second network has fewer neurons in the last encoder layer. The last encoder layer of the second network has only 3 neurons comparing with 10 neurons in the first network. This means that the second network compressed the information to a further degree than the first network did. The additional compression led to losing important aspects of the data, and that's why the reconstruction error for the second network was greater than that of the first network. Getting the right network architecture requires a lot of experimentation. We learned that for our dataset 3 neurons in an encoder layer aren't optimal.

Overall, we were able to build an autoencoder neural network that identified credit card fraud with great accuracy. We did this using unlabeled data after stripping away the labels in our dataset. This makes the techniques we developed in this project useful in real applications for fraud detection since most transaction data in our world are unlabeled. Given more time it would be interesting to analyze the same data set using logistics regression, random forests, and support vector machines and compare the results of each technique.

## References


- Towards Data Science(Applied Deep Learning): https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
- Towards Data Science(Neural Networks Hyperparameters): https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a
- Towards Data Science(Neural Networks and Backpropagation): https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a
- Keras Documentation (Installation Guide): https://keras.io/
- Towards Data Science(Activation functions): https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6
- Kaggle (Dataset): https://www.kaggle.com/mlg-ulb/creditcardfraud/data
- Pathmind: https://pathmind.com/wiki/neural-network
- Data Driven Investor: https://medium.com/datadriveninvestor/deep-autoencoder-using-keras-b77cd3e8be95
- Keras Blog: https://blog.keras.io/building-autoencoders-in-keras.html
- Keras Documentation: https://keras.io/models/sequential/
- Wikipedia: https://en.wikipedia.org/wiki/Artificial_neural_network

**R Code Appendix Summary**

| Section | Content |
|---|---|
| Section 1 | Installing Keras |
| Section 2 | Loading the dataset into R-Studio |
| Section 3 | Data Cleaning. This section is divided into 3 subsections |
| Section 3.1 | Verifying the number of binary classes in the data |
| Section 3.2 | Verifying that the timestamps in the dataset correspond to two days in total |
| Section 3.3 | Verifying that there are no missing values in the dataset |
| Section 4 | Exploratory data analysis. This section is divided into 4 subsections |
| Section 4.1 | Plotting binary classes distribution |
| Section 4.2 | Calculating a correlation matrix and plotting a correlogram |
| Section 4.3 | Plotting transaction amount versus time by class |
| Section 4.4 | Filtering out transactions less than 400 and plotting transaction amounts by class |
| Section 5 | Data Analysis and building two autoencoder neural networks. This section is divided into 5 subsections. |
| Section 5.1 | Splitting the dataset into a training set and a testing set. Also, dropping Time and Class features from the training set |
| Section 5.2 | Building the first autoencoder neural network with 6 layers in total and plotting the accuracy and loss values of the network for each epoch |
| Section 5.3 | Making predictions using the first autoencoder trained network and test data, defining the threshold for reconstruction errors, and plotting the precision-recall curve |
| Section 5.4 | Building the second autoencoder neural network with 10 layers in total and plotting the accuracy and loss values of the network for each epoch |
| Section 5.5 | Making predictions using the second autoencoder trained network and test data, defining the threshold for reconstruction errors, and plotting the precision-recall curve |