Neural Networks in Application

Created by student

Date: December 13, 2024

Middlesex University

MSO3255 Neaural Networks and Deep Learning

Assessment: Ethical and Technical Analysis

Contents

1 Neural Network Design and Implementation

1.1 Problem and Data Description

In this section, we will discuss the selected data set and evaluate its accuracy and potential improvements.

My area of interest outside of Mathematics and Data Science is education in general- specifically, exploring ways to improve learning for greater academic achievement. It is undeniable that the number of hours spent studying is a key factor for academic success. However, from my perspective it could also indicate that the material is not delivered efficiently, which may lead students to feel overwhelmed, anxious, and stressed. It would be interesting to test these theories using dataset.

The chosen dataset **student lifestyle dataset** is available to download at Kaggle. The dataset has been created via Google Form surveys completed by students from various colleges in India. The dataset consists of 2000 records with information about study habits, sleep patterns, physical activities, and social interactions, covering a total of eight columns. According dataset's authors, CGPA values, used in India, were converted to GPA to ensure usability for international researchers. The data was collected recently and covers the academic year from August 2023 to May 2024, which is important since we aim to research the current state of education.

However, the dataset has certain limitations. Notably, it lacks information about the specific programs students are studying. It is widely believed that students in certain programs, including STEM, tend to be more overworked and stressed compared to those in arts or humanities programs. Including such information would provide deeper insights and improve the accuracy of analyses.

This data set is suitable for Neural Network modeling as it contains a sufficiently large number of records, and we assume that individual records are not linearly dependent. Neural Networks are also a good choice when dealing with multiple features, as they can process and highlight the correlations between them effectively.

The data set can be used for Regression tasks, such as predicting GPA scores, or for Classification tasks, such as predicting Stress Levels or categorizing students into performance brackets derived from GPA scores.

1.2 Neural Network Architecture

Before we move to the actual creating model we need to prepare our data. We will start by analyzing its structure. Our target column will be "Stress_Level". We noticed that the labels are recorded as "objects". To make data processing easier, we converted these labels into the numerical values {0,1,2}. Additionally, we observed that the labels are not distributed evenly:

```
from collections import Counter
y = data['Stress_Level']
# Check the class distribution
print("Original class distribution:", Counter(y))
```

Original class distribution: Counter(2: 1029, 1: 674, 0: 297)

This imbalance is an important factor to consider during designing a neural network, as it could lead to biased predictions.

We highlight this issue because the proposed solution reduces the number of records from 2000 to 891. This smaller number may not be sufficient for a deep network. We also assume that there is a near-linear correlation between features and labels, so a shallow network with two hidden layers should be suitable for the tasks. We understand here that additional hidden layers can be added if performance improvements are observed.

We aim to build a feedforward neural network where each neuron in a layer connects to every neuron in the next layer, achieved using ***Dense*** layers. For the first two hidden layers we used an activation function ***ReLU***, applying a rectified linear unit activation function. To prevent overfitting, we added a regulation technique ***Dropout***, which randomly "forgets" some neurons during training, forcing neurons to learn independently. In the last layer, ***softmax*** converts a vector of values into a probability distribution for the three labels. [activation fun]

The number of neurons per layer varies with network depth. Typically, the first layer contains more neurons to capture a wide range of features. The following layers reduce dimensionality, focusing on abstract and high-level features. The output layer produces 3 neurons matching the three possible labels. Section 4 shows two model variants: a shallow network with two hidden layers and a deeper network designed to enhance prediction accuracy. Both follow a general structure described above.

The prediction type dictates the loss function class.[loss fun] With three labels we decided to classify the outcome by calculating the probabilities. Hence, *CategoricalCrossentropy* is appropriate. In our model, we used ***SparseCategoricalCrossentropy*** because our labels are integers, not a one-hot representation.

The loss function also determines the performance metrics. [metrics] Our model uses ***SparseCategoricalAccuracy***, which calculates how often predictions match labels by dividing the accumulated local variables, *total* and *count*.[sparse metrics]

1.3 Impact of design on performance

When designing neural networks, we face numerous choices that affect model performance. In many ways, this process combines knowledge, experience and intuition, as there is no single correct solution. Each dataset must be treated

uniquely, meaning there is no universal recipe to follow. Additionally, the number of variables in network design is significant: from activation functions, the number of neurons, through optimizers to the size of the batch and number of epochs. Lastly, we must have clear expectations for our model's performance. We can aim to be as much as accurate as possible, but it may be the case that it will take longer to train neurons. In some cases, lower accuracy can be acceptable if faster results are required.

In this section, we present the performance measurements of different model variations, which guided our design of the initial model. In section 2 of this paper we will further evaluate the model. For now, we will focus on model accuracy and training time. We determined 50 epochs to be sufficient. It will not be shown, but that decision was checked by plotting accuracy after each epoch. The key variables tested in our two model variations were batch size and optimizer selection. It is important to note that accuracy may slightly vary due to randomness in parts of the code. Also, training time depends on the hardware or internet speed when running the code in an online environment.

We began by creating **Dummy model** to establish a baseline for performance. Its accuracy was very low, so we have a lot of room for improvement: **Accuracy: 0.3854748603351955**.

As previously mentioned, we started with a shallow model to evaluate its performance. We did not add a Dropout regulator before taking the measurements. Our findings are presented below.

Optimizer	Adam		Adagrad		AdamW		SGD		rmsprop	
batch size	10	20	10	20	10	20	10	20	10	20
accuracy	0.955	0.955	0.698	0.676	0.955	0.955	0.737	0.67	0.95	0.94
time	14.22	8.87	10	7.05	13.87	9.37	8.76	8.27	10.74	7.43

In addition to the measurements above, we tested how the model would behave when adding *Dropout(0.3)* with Optimizer *Adam*. We did not observe a significant difference in performance. The accuracy changed to 0.9441 (Dropout in 1 layer) and 0.9497 (Dropout in 2 layers).

We also observed that some optimizers are more sensitive to batch size. Generally, we see that the smaller the batch, the longer it takes to train neurons, but on the flip side, it improves accuracy. Based on our experiments, we decided to adopt the *Adam* optimizer with a batch size of 20, as it offered high accuracy while keeping training time relatively short.

Although the results for the shallow model were very good, we also tested a deeper neural network, which is presented in section 4. Despite adding layers and introducing dropouts, there was no improvement in accuracy. However, the design significantly increased the training time.

1.4 Mathematical Analysis

The model consists of an input layer, two hidden layers, and an output layer. Each layer is defined by a specific number of neurons and an activation function, which together determine the transformations applied to the input data. For our model, we use a dataset with 891 entries and 6 features, so our single input neuron is 6-dimensional. The entire dataset is represented as input $X \in \mathbb{R}^{891\times 6}$.

During forward propagation in the neural network, each layer transforms neurons according to a defined function:

$$z_n = W_n x + b_n, \quad h_n = f(z_n),$$

where:

- n is an iteration of a hidden layer;
- W_n represents weights matrix;
- b_n is a bias vector;
- h_n is the output of the layer;
- f is an activation function

We can apply it to our model, so the first layer is:

$$z_1 = W_1^{32\times 6} X + b_1, \quad h_1 = ReLU(z_1).$$

Second layer

$$z_2 = W_2^{16\times 6} X + b_2, \quad h_2 = ReLU(z_2),$$

and for the output layer, we have

$$z_3 = W_3 h_3 + b_3, \quad \hat{y} = Softmax(z_3),$$

where $\hat{y} \in \mathbb{R}^3$ is the predicted probability distribution over 3 classes.

Function ReLU (Rectified Linear Unit) is one of the most popular activation functions in deep learning as it addresses the vanishing gradients problem. It is defined as:

$$(x)^+ = \begin{cases} 0 & if \quad x \leq 0 \\ x & if \quad x > 0 \end{cases} = max(0, x).$$

Softmax converts a vector of raw prediction scores (often called logits) from the neural network into probabilities. In the classification networks, the last layer provides an output vector $z = [z_1, z_2, \cdots, z_K]$, where K is a number of the classes, in our model case is 3, and z_i is a logit of i-class. The formula for each class $i \in K$ is:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for \quad \forall i \in (1, \cdots, K) \tag{1}$$

The exponential function e^{z_i} plays a crucial role in Softmax, as even a small increase in logit value results in a larger probability, while small logits result in near-zero probabilities. [softmax]

The final output $\hat{y}$ is compared with the true labels y using our chosen loss function *sparse categorical crossentropy.*

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N} log(\hat{y}_{i,y_i}) \tag{2}$$

Forward propagation computes the values of hidden units, the output o, and loss $\mathcal{L}$. During backpropagation (backward propagation of errors), the neural network adjusts its weights and biases by minimizing the loss function $\mathcal{L}$ through gradient descent. The process begins by calculating $\Delta(o, o)$ to determine $\frac{\partial \mathcal{L}}{\partial o}$ and then computes each $\delta(h_r, o)$ in the backward direction. Next, gradients with respect to each neuron's weights are calculated as follows:

$$\frac{\partial \mathcal{L}}{\partial w_{(h_{r-1},h_r)}} = \Delta(h_r, o) \cdot h_{r-1} \cdot \Phi'(a_{h_r}),$$

where a_{h_r} is the value calculated in hidden layer h_r *before* applying the activation function $\Phi(\cdot)$.

Backpropagation for softmax works slightly differently because the function processes multiple inputs. Specifically, softmax converts k real-valued predictions $x_1 \dots x_k$ into probabilities (see Equation 1). So, to compute the derivative of loss $\mathcal{L}$ with respect to $x_1 \dots x_k$, we need to calculate each $\frac{\partial \mathcal{L}}{\partial o_i}$ and also $\frac{\partial o_i}{\partial x_j}$. [Aggarwal]

Since softmax is almost always paired with cross-entropy loss function, and in the case of our model sparse categorical cross-entropy (see Equation 2), the value of $\frac{\partial \mathcal{L}}{\partial x_i}$ has a simple form:

$$\frac{\partial \mathcal{L}}{\partial x_i} = \sum_{j=1}^{k} \frac{\partial \mathcal{L}}{\partial o_j} \cdot \frac{\partial o_j}{\partial x_j} = o_i - y_i$$

Further backpropagation proceeds like we talked about earlier.

2 Model evaluation

2.1 Model performance

In section 1.3 we briefly analyzed the performance of the model's variants. Here we will analyze it deeper and discuss why further improvements should be considered.

We applied our neural network and called for the metrics report, which gave us a deeper insight into the model performance. The reported **learning accuracy** after the last epoch is **0.98**, while **prediction accuracy** as is 0.95.

At first glance, it seems like we created a nearly perfect model. However, is it truly as effective as it appears?

We examined several performance plots, starting with the accuracy over epochs.

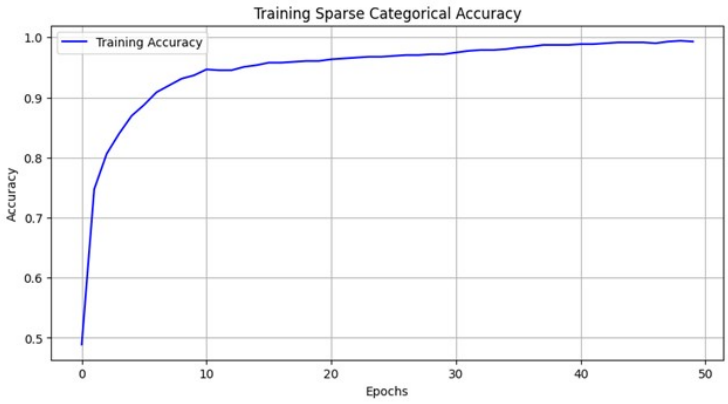

Figure 1: Training Sparse Categorical Accuracy

We can observe that our neural network quickly achieves 90% accuracy, within the initial epochs. While this might initially seem positive, it could also be an indicator of overfitting, where neurons begin to rely on each other despite the application of Dropout regularization.

A decreasing curve of the loss function plot indicates that the model is learning and improving its predictions over with each epoch. Flattening Loss suggests that the model is converging, and further training may yield diminishing returns. In our case, the loss curve is promising, constantly decreasing.

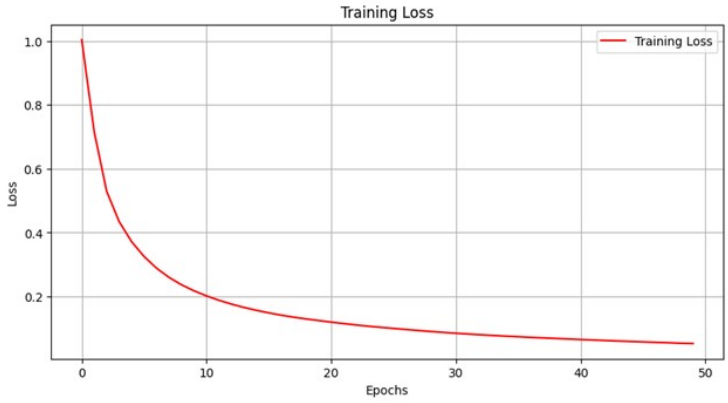

Figure 2: Training Loss Function

Next, we analyze the confusion matrix which provides a detailed breakdown of the model's performance by showing true positives, true negatives, false positives, and false negatives. There are many features which can be extracted from the confusion matrix, like accuracy, precision, F1 score etc. While these metrics could be recalculated from the confusion matrix, we have already obtained them from the output window during our evaluation process.

Loss: 0.13278375566005707
Accuracy (SparseCategoricalAccuracy): 0.9497206807136536
Training time is 3.7636899948120117
Classification Report:

	precision	recall	f1-score	support
0	0.95	0.98	0.97	62
1	0.97	0.92	0.94	61
2	0.93	0.95	0.94	56
accuracy			0.95	179
macro avg	0.95	0.95	0.95	179
weighted avg	0.95	0.95	0.95	179

Table 1: Metrics Output for Classification model

The confusion matrix can help identify the irregulations of the model's performance. For instance, high accuracy with low precision or recall could indicate that the model predicts the majority class more often. Since we balanced our data preprocessing stage, this issue should not concern our model.

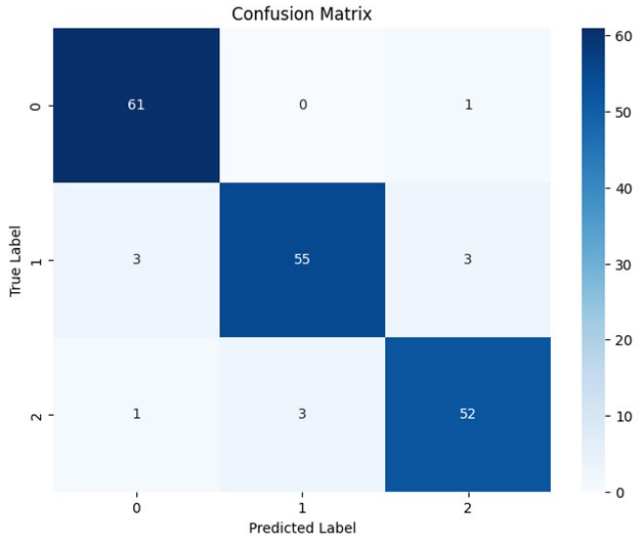

Figure 3: Confusion matrix for Classification model

Overall, our confusion matrix is well balanced, slightly more often predicting 0 as the outcome (see recall and F1-score in Table 1). It could be the issue of learning on features not related to the outcome or the neural network not being able to generalize better.

To investigate it further, we calculated the importance of the features. With the outcome below:

	Feature	Mean SHAP Value
0	Study_Hours_Per_Day	0.258717
2	Sleep_Hours_Per_Day	0.123369
4	Physical_Activity_Hours_Per_Day	0.076855
3	Social_Hours_Per_Day	0.051614
1	Extracurricular_Hours_Per_Day	0.041332
5	GPA	0.021541

Table 2: Features importance SHAP Score

Features that are not significant to the model can introduce random noise, and potentially produce misleading high accuracy, which is often accompanied by very high training accuracy.

To address this, we have created a new neural network, excluding three less important features from the data. Next, we analyzed how changes in hyperparameters affected the model's behaviour. Interestingly, we observed that if an improved model keeps batch size 20 as the original model, it achieves the same accuracy in a shorter time. Increasing the batch size to 50, we allowing a higher number of neurons to interact with each other, and that results in an accuracy increases to 0.96.

This highlights importance of feature selection when building statistical, ML models or neural networks. Carefully selecting relevant features ensures that model focuses on meaningful patterns.

Loss: 0.12153726816177368
Accuracy (SparseCategoricalAccuracy): 0.9608938694000244
Training time is 4.92718768119812

Classification Report:

	precision	recall	f1-score	support
0	0.97	1.00	0.98	62
1	0.95	0.95	0.95	61
2	0.96	0.93	0.95	56
accuracy			0.96	179
macro avg	0.96	0.96	0.96	179
weighted avg	0.96	0.96	0.96	179

Table 3: Metrics Output for Improved Classification model

2.2 Limitations

Our neural network faces several limitations, which we tried to mitigate.

- **Data bias**. As discussed earlier biased data results in biased model and biased prediction. To mitigate this issue, we removed excess input from the majority class to balance the dataset. Due to the randomization of the process, we are introducing a risk of omitting data points with valuable information.
- **Risk of overfitting with small data**. To match minority class sinificantly reduced the number of the data points creating this way relatively small dataset. Additionally, small number of classes lacks diversity. This increases risk of neurons memorizing the data instead of learning patterns. We mitigated that risk by introducing a Dropout in the hidden layers.
- **Lack of Feature-Specific Insights**. It is challenging to understand which features contribute most to the model's prediction. For example,the dataset records are numbers of hours spent daily on various activities. It has not been checked whether sum of those numbers do not exceed 24h, which should have been checked during preprocessing the data. To address that issues partialy, we do used SHAP to interpret feature importance and refine the model.
- **Vanishing Gradient**. The vanishing gradient problem is common issue when updating weights during backpropagation. To consider this, we used ReLU activation function, which is less prone to vanishing gradients. Additionally, reduction the number of epochs could be considered if neurons stop learning earlier in the training process.

3 Ethical considerations in Neural Network Applications

Neural networks are a powerful tool in artificial intelligence. With the growing implementation of that tool in people's daily lives, ethical concerns have arisen, as they should be. Applications of neural networks face many challenges, and general public awareness, as well as skills and ethical consciousness of programmers, is much needed. Here we will discuss several ethical challenges we faced or that could potentially become an issue if someone decides to use the model for their work.

The primary concern of this model is the dataset itself. It was clear to us that the data was biased. We do not question the purpose for which the data was collected. Perhaps the survey aimed to research how stressed students are, so the answer would be that the majority of them are stressed. However, if we want to define which factors increase the risk of anxiety or simply predict it, we should use unbiased data. If we build the neural network on biased data, our

model will predict a high level of stress for most new entries, even if indications are not present. That type of issue should be mitigated in the early stages of creating neural networks. Programmers should be particularly careful with sensitive data, such as in hiring, where gender, race, or socioeconomic status may lead to unfair outcomes.

Another issue stems also from the data. Although we used unbiased data to train our neural network, the results could be presented incorrectly. When we started to work on the network, we aimed to find how specific factors affect student mental health and potentially use that model to improve students' lives. In datasets where features are not clearly understood, it is hard to interpret the results. It is well-known that noisy or unrelated data affects accuracy, which could lead to incorrect conclusions. According to the dataset authors, the survey was addressed to random students from different universities. However, in our opinion, features like the name of the program or department, as well as gender would be very useful for this network. To mitigate the issue of explainability we used SHAP to narrow down the decision-making process.

Additionally, our neural network is shallow, making it more transparent and easier to understand. However, this is not always the case for deep neural networks, which raises yet another risk of misuse and misinterpretation. The general public often assumes that if a neural network provides a result, it must be correct. This opens the door to potential discrimination, wrongful decisions, or overlooking errors. No network is perfect, nor should it be considered so. For example, in tasks such as fraud detection, load decisions, or law enforcement, neural networks should always incorporate human oversight and communicate their limitations to the users. In the case of our model, if we claimed it could predict the anxiety level of any given student, some may try to use it for students from different areas or even countries, which could lead to inaccurate predictions. Moreover, identifying study hours as the main factor of student distress may result in poor decisions, like reducing academic curriculum or assigning less homework, rather than researching the root causes of stressful learning environments.

In conclusion, applications of neural networks should be approached with care. By addressing ethical concerns such as bias, transparency or privacy, programmers ensure that technology is used fairly and responsibly.

4 Code source

The code used for designing the neural network discussed in this paper is available on GitHub, or copy

https://github.com/marta4cod/NNAssessment1

to the browser.

5 References

References

[activation fun] Keras documentation: https://keras.io/api/layers/activations/

[loss fun] Keras documentation: https://keras.io/api/losses/

[metrics] Keras documentation: https://keras.io/api/metrics/

[sparse metrics] Keras documentation: https://keras.io/api/metrics/accuracy_metrics/#sparsecategoricalaccuracy-class/

[softmax] https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

[Aggarwal] Aggarwal, C.C., 2018. Neural networks and deep learning (Vol. 10, No. 978, p. 3). Cham: springer.