**ASSIGNMENT REPORT**

**UNDERSTANDING ARTIFICIAL INTELLIGENCE**

**OSAGIE ELLIOT AIBANGBEE**

**202115576**

**MSC ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

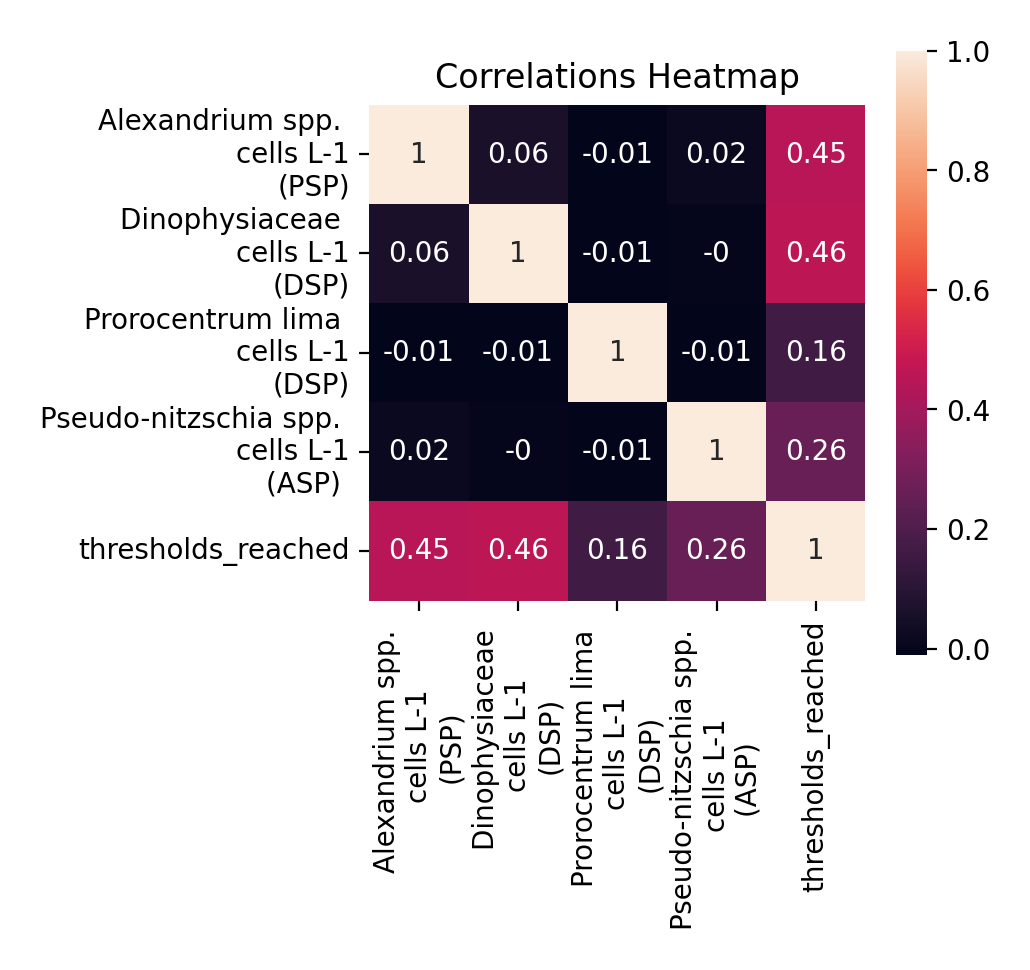
**December 2021**

**COMPONENT 1**

**WATER QUALITY ANALYSIS**

After cleaning the phytoplankton data set and selecting only these 4 columns ("Alexandrium spp. \ncells L-1\n(PSP)", "Dinophysiaceae \ncells L-1\n(DSP)", "Prorocentrum lima \ncells L-1\n(DSP)", "Pseudo-nitzschia spp. \ncells L-1\n(ASP) "), I proceeded to use the information given about the threshold concentration levels of each sample to create a column of binary values indicating where each phytoplankton sample is above their respective threshold values, with 1 indicating True and 0 indicating False. This was then used to create a column serving as a general indicator for a sample having an above threshold concentration level for any of its constituents. The general indicator column had a value 1 (True) for a sample whose concentration level exceeded the specified trigger level for any of the phytoplanktons mentioned above. After which a total of 96 samples were detected as having an above threshold concentration for one of the phytoplanktons. This new information was used as the train to a neural network model for binary classification.

The below Heatmap shows a positive correlation between target label (thresholds\_reached) and samples of Alexandrium Spp and Dinophysiaceae. The correlation thresholds\_reached has with these two features is stronger than those of Pseudo-nitzschia spp and Prorocentrum lima.



**FIRST TIME SET UP**

Network Model Structure:

IL – Input Layer; HL – Hidden Layer; OL – Output Layer; AF – Activation Function

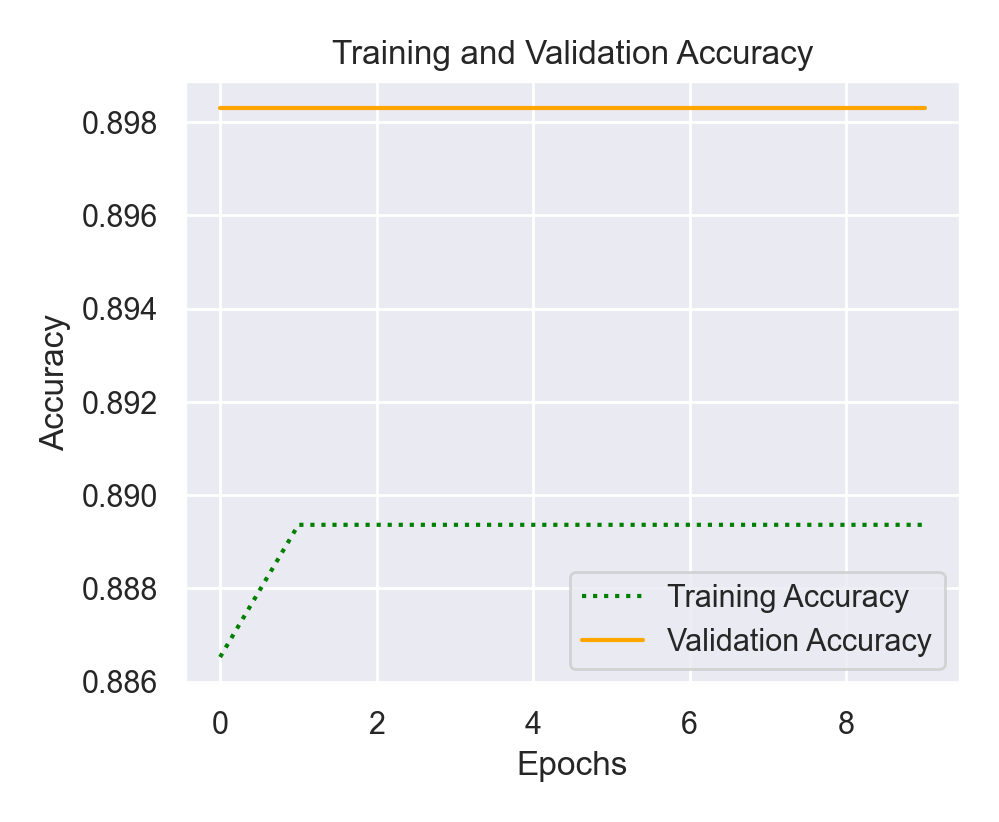
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **IL** | **HL-1** | **HL-2** | **HL-3** | **OL** |
| **Type** | Dense | Dropout | Dense | Dropout | Dense |
| **Units/Rate** | 1000 | 0.9 | 5 | 0.2 | 1 |
| **AF** | relu |  | relu |  | relu |
| **Input Shape** | (4, ) |  |  |  |  |

Network Model Configuration:

|  |  |
| --- | --- |
| **Metric** | Accuracy |
| **Loss Function** | Categorical Crossentropy |
| **Optimizer** | adam |

Network Model Fitting:

|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Batch Size** | **Training Data** | **Validation Data** |
| 10 | 16 | [x\_train, y\_train] | [x\_test, y\_test] |



The above plot shows the change in accuracy and loss scores per training loop. The first training loop generated a training accuracy of 0.8851 and a validation accuracy of 0.8983. While the second iteration resulted in a training accuracy of 0.8894 and the validation accuracy remained at 0.8983, indicating that there was only a slight increase in the training accuracy from 0.8851 to 0.8894. Subsequent iterations generated a constant value of 0.8894 as training accuracy, and 0.8983 as validation accuracy. This indicates that no optimization took place after the second iteration.

Network Model Final Evaluation:

|  |  |
| --- | --- |
| **Testing Data** | [x\_test, y\_test] |
| **Test Loss** | nan |
| **Test Accuracy** | 0.8983 |

Upon the final evaluation using the same test dataset that was used as validation set, an accuracy of 0.898 and a loss of nan was derived. And this leaves much to be desired as to the structure and configuration of the network model.

**MAKING ARCHITECTURAL CHANGES**

1. ACCURACY ACROSS 3 ARCHITECTURAL CHANGES
2. Adding/Dropping a number of layers:

Adding more layers, accuracy was **0.8983,** and loss was **nan**.

Network Model Structure:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **IL** | **HL-1** | **HL-2** | **HL-3** | **HL-4** | **HL-5** | **HL-6** | **HL-7** | **HL-8** | **HL-9** | **HL-10** | **OL** |
| **Type** | Dense | Dropout | Dense | Dense | Dense | Dense | Dropout | Dense | Dense | Dense | Dense | Dense |
| **Units/Rate** | 1000 | 0.9 | 5 | 5 | 5 | 5 | 0.2 | 5 | 5 | 5 | 5 | 1 |
| **AF** | relu |  | relu | relu | relu | relu |  | relu | relu | relu | relu | relu |
| **Input Shape** | (4, ) |  |  |  |  |  |  |  |  |  |  |  |

Network Model Configuration:

|  |  |
| --- | --- |
| **Metric** | Accuracy |
| **Loss Function** | Categorical Crossentropy |
| **Optimizer** | adam |

Network Model Fitting:

|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Batch Size** | **Training Data** | **Validation Data** |
| 10 | 16 | [x\_train, y\_train] | [x\_test, y\_test] |

The above plot shows the change in accuracy and loss scores per training loop after adding 7 layers to the previous 5 as shown in this model structure table (1a). The first training loop generated a training accuracy of 0.8809 and a validation accuracy of 0.8983. While the second iteration resulted in a training accuracy of 0.8894 and the validation accuracy was 0.8983, indicating that there was a slight increase in the training accuracy. Subsequent iterations generated a constant value of 0.8894 as training accuracy, and 0.8983 as validation accuracy. This indicates that no optimization took place after the second iteration.

Network Model Final Evaluation:

|  |  |
| --- | --- |
| **Testing Data** | [x\_test, y\_test] |
| **Test Loss** | nan |
| **Test Accuracy** | 0.8983 |

Upon the final evaluation using the same test dataset that was used as validation set, an accuracy of 0.898 3 and a loss of nan was derived. This shows that adding more layers did not improve the network’s performance compared to the first time. Still, much can be done about the structure and configuration of the network model that can lead to an improvement.

Dropping a number of layers:

Accuracy was **0.8983**, and Loss was **nan.**

Network Model Structure:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **IL** | **HL-1** | **OL** |
| **Type** | Dense | Dropout | Dense |
| **Units/Rate** | 1000 | 0.2 | 1 |
| **AF** | relu |  | relu |
| **Input Shape** | (4, ) |  |  |

Network Model Configuration:

|  |  |
| --- | --- |
| **Metric** | Accuracy |
| **Loss Function** | Categorical Crossentropy |
| **Optimizer** | adam |



Network Model Fitting:

|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Batch Size** | **Training Data** | **Validation Data** |
| 10 | 16 | [x\_train, y\_train] | [x\_test, y\_test] |

The above plot shows the change in accuracy and loss scores per training loop after dropping 2 layers from the first network model (of 5 layers), as shown in this model structure table (1b). The first training loop generated a training accuracy of 0.8879 and a validation accuracy of 0.8983. While the second iteration resulted in a training accuracy of 0.8894 and the validation accuracy was 0.8983, indicating that there was a slight increase in the training accuracy. Subsequent iterations generated a constant value of 0.8894 as training accuracy, and 0.8983 as validation accuracy. This also indicates that no optimization took place after the second iteration.

Network Model Final Evaluation:

|  |  |
| --- | --- |
| **Testing Data** | [x\_test, y\_test] |
| **Test Loss** | nan |
| **Test Accuracy** | 0.8983 |

Upon the final evaluation using the same test dataset that was used as validation set, an accuracy of 0.898 3 and a loss of nan was derived. This shows that adding more layers did not improve the network’s performance compared to the first time. Much can be done about the structure and configuration of the network model that can lead to an improvement.

1. Adjusting hyperparameters:

Changing Output Activation Function to Sigmoid, and Loss Function to Binary Crossentropy

Changing activation function of output layer from ‘relu’ to ‘sigmoid’ and using a binary cross entropy loss function resulted in an accuracy of **0.9209.**

Network Model Structure:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **IL** | **HL-1** | **HL-2** | **HL-3** | **OL** |
| **Type** | Dense | Dropout | Dense | Dropout | Dense |
| **Units/Rate** | 1000 | 0.9 | 5 | 0.2 | 1 |
| **AF** | relu |  | relu |  | sigmoid |
| **Input Shape** | (4, ) |  |  |  |  |

Network Model Configuration:

|  |  |
| --- | --- |
| **Metric** | Accuracy |
| **Loss Function** | Binary Crossentropy |
| **Optimizer** | adam |

Network Model Fitting:

|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Batch Size** | **Training Data** | **Validation Data** |
| 10 | 16 | [x\_train, y\_train] | [x\_test, y\_test] |



The above plot showing the change in accuracy and loss scores per training loop illustrates an improvement in the training accuracy and loss across 10 epochs.

Network Model Final Evaluation:

|  |  |
| --- | --- |
| **Testing Data** | [x\_test, y\_test] |
| **Test Loss** | 0.3120 |
| **Test Accuracy** | 0.9209 |

Upon the final evaluation using the same test dataset that was used as validation set, an accuracy of 0.9209 and a loss of 0.3120 was derived. And this is an encouraging improvement on the first model. Meaning that sigmoid activation function in the output layer is the better choice than using relu based on the task at hand. And a loss function of binary crossentropy suits this task which is a binary classification.

1. Creating a separate validation set, normalizing inputs (training, validation, and test set) with Scikitlearn’s Standardscaler object, using binary cross entropy loss function, reducing epochs to 3, changing number of units in the second dense layer to 50:

All these resulted in an accuracy of **1.0,** and loss score is **0.0303**

Network Model Structure:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **IL** | **HL-1** | **HL-2** | **HL-3** | **OL** |
| **Type** | Dense | Dropout | Dense | Dropout | Dense |
| **Units/Rate** | 1000 | 0.9 | 50 | 0.2 | 1 |
| **AF** | relu |  | relu |  | sigmoid |
| **Input Shape** | (4, ) |  |  |  |  |

Network Model Configuration:

|  |  |
| --- | --- |
| **Metric** | Accuracy |
| **Loss Function** | Binary Crossentropy |
| **Optimizer** | adam |

Network Model Fitting:

|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Batch Size** | **Training Data** | **Validation Data** |
| 3 | 16 | [normalized\_x\_train, y\_train] | [normalized\_x\_val, y\_test] |

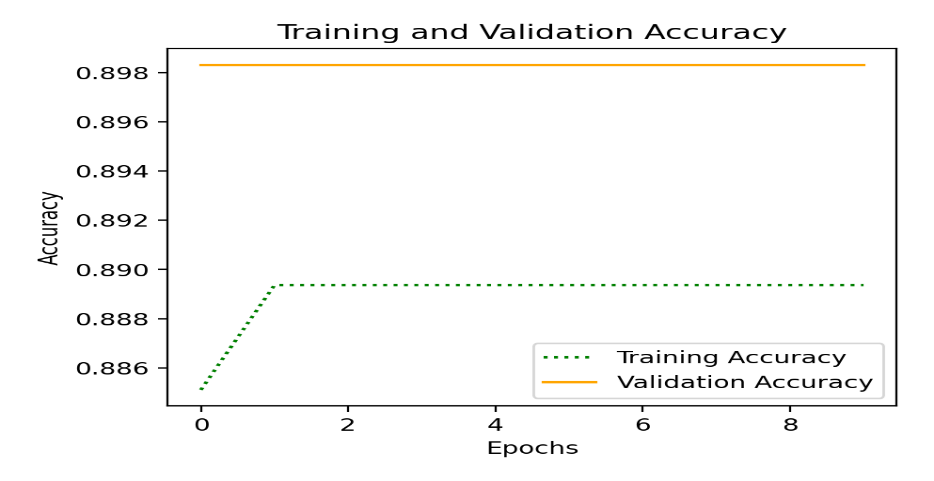


Network Model Final Evaluation:

|  |  |
| --- | --- |
| **Testing Data** | [normalized\_x\_test, y\_test] |
| **Test Loss** | 0.0303 |
| **Test Accuracy** | 1.0 |

Upon the final evaluation using the same test dataset that was used as validation set, an accuracy of 1.0 and a loss of 0.0303 was derived.

LIKELY CAUSE(S) OF RESULTING ACCURACY SCORES

Initially, the network model was created consisting of 5 layers with a dense input layer, three hidden layers (one dense layer in-between two dropout layers), and a dense output layer. Each dense layer had a rectified linear unit (ReLU) activation function, the chosen optimizer was ‘adam’, the loss function was ‘categorical cross entropy’, and the metrics was ‘accuracy’. The model was then fitted to the training set and validated on the test set, with batch size of 16 and 10 epochs. The model’s performance during this training can be seen in the graph below.

Things wrong with this model include categorical crossentropy is a loss function that performs best in classifications involving more than two classes/categories. The number of epochs is unnecessary for the size of this dataset. The network should have been trained and validated on training and validation sets and then tested on a reserved test set, which would help us to know how well the model truly performs on new data. For the purpose of this classification task, a better activation function for the output layer would be ‘sigmoid’ (rather than ‘relu’).

1. **Different number of layers:**

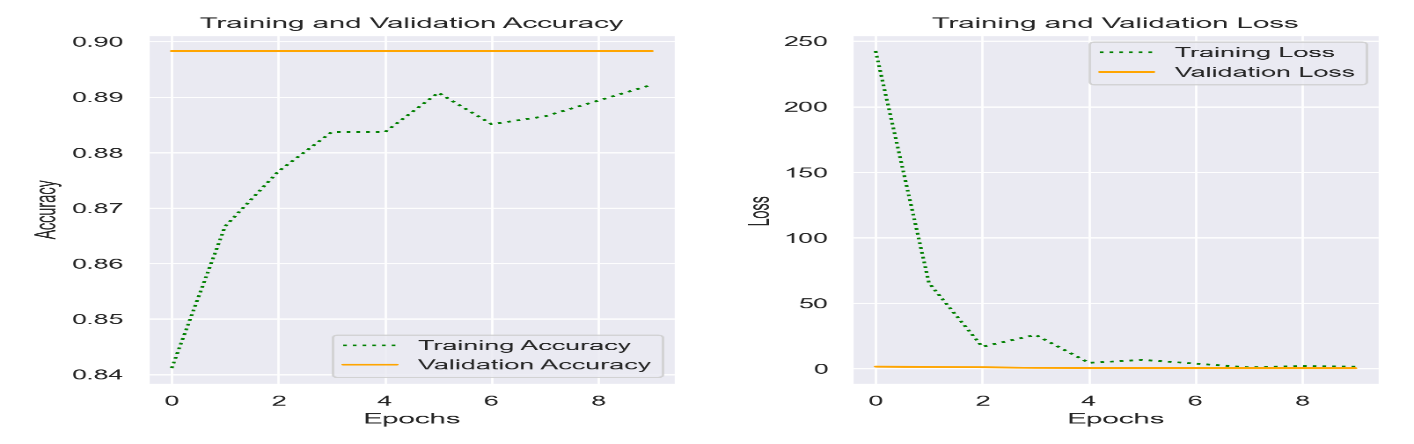
Adding/Dropping a number of layers

Adding 7 hidden layers (as shown in the corresponding structure table) to the existing 5 layers did not produce any change in performance of the network. Likewise, dropping the two dropout layers from the model did not change the performance of the model because the number of layers has no effect on the data.

Therefore, only changing the number of layers appears not to have any significant effect on the model’s performance. This could be because of the following reasons: wrong loss function being used, suboptimal activation function (‘relu’) used in the output layer, values of the input do not all have same range of values, validation and testing was done on the same test set.

1. **Adjusting Hyperparameters**

Changing activation function of output layer from ‘relu’ to ‘sigmoid’ and using a binary cross entropy loss function

I used a sigmoid/logistic activation function in the output layer, and a binary cross entropy as loss function because crossentropy outputs probabilities and the sigmoid function outputs values between 0 and 1, and this label (y) has only two classes. This resulted in a change in accuracy score during training as shown in the graph below

Looking at the graph reveals the performance of the model stops to improve by the third iteration (epoch 2). Thus, it might help if we changed epochs to three in addition to other adjustments.

1. **Creating a separate validation set, normalizing inputs (training, validation, and test set) with Scikitlearn’s Standardscaler object, using binary cross entropy loss function, reducing epochs to three:**

An output layer with sigmoid activation function was used, and binary cross entropy was chosen as loss function. A sigmoid activation function (which only outputs values between 0 and 1) was used because it’s most appropriate for binary classification. Epochs was changed to three because from previous trials it was observed that there was no significant improvement in the model’s performance after two to three epochs. A separate validation set was created for validation during training to prevent information leakage from optimizing to the test set, and to get a better idea of how the model performs with unseen data (test set). StandardScaler object uses the mean and standard deviation to normalize values of a given numeric column such that it is centered around 0. This brought all the input values within the same range.

**EFFECT OF OPTIMIZATION FUNCTION ON A NETWORK**

The optimization function (optimizer) is primarily responsible for making necessary adjustments to a network’s parameters during the training process. It works in conjunction with the loss function during training, by using the loss score (based on performance metric) generated by the loss function after each training loop, to fine-tune the model’s hyperparameters in a direction that reduces loss and increases accuracy. It uses stochastic gradient descent to implement this performance improvement.

**INCLUDING MORE THAN 4 LAYERS TO THE MODEL**

Doing this alone without changing other parameters (e.g changing the loss function or epochs, input transformation, etc.) did not affect the performance of the network. A deep neural network is one that has 3 or more hidden layers. So doing this simply increases the depth of the network without any noticeable improvement in its performance. When increasing the depth of a network, special consideration should be given to their activation functions, as ReLU tends to perform better with deeply connected layers than sigmoid and hyperbolic tangent because the vanishing problem (Brownlee, 2020)**.**

**THE EFFECT OF THE DATA SIZE ON YOUR ACCURACY**

In separate attempts, using 50% of data set to train and validate the network model gave an accuracy score of **0.8914** and a loss of **nan**. While training and validating on 30% of data set result in an accuracy of **0.8964.** Evidently, the size of the data set does not have a significant effect on the accuracy of this network. A large dataset is not required for a neural network to obtain a high accuracy in a classification task (Harb & Jayousi, 2013).

**COMPONENT 2**

**MULTI-OBJECT RECOGNISER**

Network Model Structure:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **IL** | **HL-1** | **HL-2** | **HL-3** | **HL-4** | **HL-5** | **HL-6** | **HL-7** | **OL** |
| **Type** | Conv2D | MaxPooling2D | Conv2D | MaxPooling2D | Conv2D | MaxPooling2D | Flatten | Dense | Dense |
| **Units** | 16 |  | 32 |  | 64 |  |  | 128 | 4 |
| **AF** | relu |  | relu |  | relu |  |  | relu | None |
| **Input Shape** | (4, ) |  |  |  |  |  |  |  |  |

Network Model Configuration:

|  |  |
| --- | --- |
| **Metric** | Accuracy |
| **Loss Function** | Sparse Categorical Crossentropy |
| **Optimizer** | adam |

Network Model Fitting:

|  |  |  |  |
| --- | --- | --- | --- |
| **Epochs** | **Batch Size** | **Training Data** | **Validation Data** |
| 10 | 16 | [normalized\_x\_train, y\_train] | [normalized\_x\_val, y\_test] |

**How long does the network need to train until reaching an accuracy of 95% (or does it not reach this level at all)?**

The highest accuracy of my model is 0.67. I never got an accuracy of 95%. I think it would take days of training on a larger dataset to achieve that sort of accuracy

**What is the tradeoff between using many layers (i.e. having a “deeper” network) and accuracy? And layers and time?**

A Layer consists of nodes which map input to output. How a layer carries this out is determined by its activation function. Layers learn relationships between input values and their corresponding label values by performing tensor operations (such as dot operation) on inputs values, and outputting the result to the next layer. Thus, the following layer receives a different representation of the input, as effected by the preceding node. Generally speaking, the accuracy of a network model depends on not just the number of layers but other hyperparameters like the weights, activation function, etc. Ideally, a deep network has the capacity to learn and produce better accuracy. However, this is heavily reliant on the size of the dataset.

The higher the number of layers in a network, the more the number of tensor operations it needs to perform. Therefore, the more time it needs to finish each training loop. Some layers perform better for certain tasks than others.

More layers mean the network is able to learn more transformations of the input. However, for smaller datasets, this can easily lead to overfitting. Thus, when training to a small dataset, it is advisable to use few layers to prevent overfitting the model to the training data.

**What is the effect of changing the pooling mechanism, e.g. average vs max?**

For images, average pooling tend to make the image smoother and max pooling brings out brighter shapes. Thus max pooling is better for dark background images.

**How well does your network do at classifying these images?**

**Does fine-tuning make a difference?**

Fine-tuning hyper parameters like changing the activation function of the output layer from none to ‘softmax’ for this multinomial classification resulted in slightly better accuracy.

**COMPONENT 3**

**DISCUSSION OF ETHICS IN AI**

**Energy and Policy Considerations for Deep Learning in NLP**

Two of the most common sources of atmospheric carbon emission are burning of substances consisting of high amount of carbon, and deforestation (destruction of plants). A lion’s share of the atmospheric CO2 is absorbed by plants, and the oceans making them more acidic (Encyclopedia.com, 2021). The increasing level of atmospheric CO2 is known to be one of the major contributors to global warming.

This paper spotlights the negative impact(s) of deep learning models performing natural language processing (NLP) tasks on the environment, and their financial implications on research. Emphasis was placed on the huge amount of CO2 (carbon dioxide) being released into the atmosphere during training and development of these models. Modern technological advances such as availability of computing resources capable of high accuracy of neural networks performing NLP tasks has been linked to high energy usage (environmental cost) and financial cost (including hardware requirements and electricity expenditure). This growing trend can be attributed to the simple fact that technologically-advanced models which require a large amount of sophisticated computing resources tend to produce the best performance. Thus, energy used and carbon emitted as part of the costs of model development has increased exponentially. In the paper, it was estimated that the average amount of CO2 it takes to train and develop NLP models (including tuning and experimentation) is 78, 468 lbs. This is equivalent to seven times of the average amount being used by a human being per year. It is a cause for concern, especially because it has been reported that there is need to drastically reduce the rate of CO2 release by 50% within the next ten years (as of the year of publication, 2019) to prevent the occurrences of more natural disasters.

To estimate the amount of CO2 released and electricity used, they obtained approximations of the requisite energy (in kilowatts) for training common NLP models. And, to gain a good understanding of the complete computational requirements, a case study was done on a sophisticated NLP pipeline. Expended energy was determined in this research by training models without changing the original settings and recording the amount of power being spent by GPU and CPU. During Training, only ELMo needed three NVIDIA GTX 1080 Ti GPUs, while one NVIDIA Titan X GPU each was enough for others. The average GPU power usage was taken from the NVIDIA System Management Interface, and CPU’s was taken from Intel’s Running Average Power Limit interface. Approximated times of training completion were taken from the original papers. Power expenditure was computed using designated formula, and approximations of CO2 emission was derived based on calculated values in the U.S EPA.

Models analyzed are the Transformer model, ELMo, BERT, NAS, GPT-2. The research revealed that the most efficient hardware for model training and development is TPU due to the next to zero amount of CO2 they release. Training a model was found to be more cost-effective than tuning them to different datasets.

They recommended that AI practitioners (who tend to focus on model performance) should document the time and resources required to train and retrain models. Also, priority should be given to the effective management of hardware and efficient model structure.

CO2 emission lingers, despite being a universal challenge, partly because its adverse effects are not immediately visible on the environment, and brings economic benefits to a lot of people (even countries). This research revealed that the exponential rise in the amount of CO2 released in AI-related activities is too huge a price to be paid.

Thus, one way of checking this perennial trend would be for international bodies like the Association for the Advancement of Artificial Intelligence (AAAI), to be much firmer and outspoken in their stance against the contribution of AI-related activities to global warming, by making the industry fully aware of green-house implications associated with adopting certain methods, and/or using certain hardware to train and develop models. AI practitioners should be more conscious of what source of energy their hardware uses (Strubell et al., 2020). A regulatory body should be set up and charged with keeping AI industry-related CO2 release and energy usage to the barest minimum. World bodies (such as the WHO and UN) should set up measures such as penalizing defaulters through fines/taxes, and recognition of those who adhere to best practice, promoting the planting of more trees among AI practitioners.

However, a more holistic approach to solving global warming would be a widespread adoption of renewable energy by countries and international organizations. Doing this would significantly curb the indiscriminate release of atmospheric CO2, which would eventually have a trickle-down effect on the AI practitioners in research and industry.

**REFERENCES**

Encyclopedia.com (2019) Carbon Dioxide (CO2) Emissions

Available Online: <https://www.encyclopedia.com/environment/energy-government-and-defense-magazines/carbon-dioxide-co2-emissions> [Accessed 11/12/2021]

Harb, A. & Jayousi, R. (2013) Database and Data Mining. *The 6th International Conference on Information Technology (ICIT 2013)*

Jason Brownlee (2020) *A Gentle Introduction to the Rectified Linear Unit (ReLU)*

Available Online: <https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=The%20rectified%20linear%20activation%20function%20or%20ReLU%20for,easier%20to%20train%20and%20often%20achieves%20better%20performance> [Accessed 12/12/2021]

Strubell, E., Ganesh, A. & McCallum A. (2020) Energy and Policy Considerations for Modern Deep Learning Research. *The Thirty-Fourth AAAI Conference on Artificial Intelligence (AAAI-20)*