CS5011: Assignment 4 – Learning

Automated Algorithm Selection using Neural Networks

Student ID: 220029176

Date of Submission: 10/05/2023

Table of Contents

1. Introduction

This report presents the implementation and analysis of an Automated Algorithm Selection (AS) system using artificial neural networks (ANNs). The goal of the AS problem is to build an automated selector that can choose the best algorithm for a given instance, to minimise the expected cost metric. In Section 1, the AS problem is modelled as a regression task, and an ANN is built to predict the cost of every algorithm in the portfolio based on instance features. In Section 2, the AS problem is modelled as a multi-class classification task, where both a basic and advanced approach are implemented and compared with the results obtained in Part 1. Section 3 involves the implementation of two extensions: the cost-sensitive pairwise classification approach, and an AS system based on Random Forest models.

1.1 Parts Implemented

ANN	**Status**
Part One: Regression	Attempted, Tested, Fully Working.
Part Two: Classification	Attempted, Tested, Fully Working.
Part Two: Advanced Classification	Attempted, Tested, Fully Working.
Extension: Binary Classification	Attempted, Tested, Partially Working.
Extension: Random Forest Classification	Attempted, Tested, Fully Working.

2. Design & Implementation

2.1 The Metrics

The average loss, accuracy, average cost, SBS cost, VBS cost and, the SBS-VBS gap were all used as metrics for this assessment as per the specification. These metrics were calculated differently based on the type of model. As the regression model's goal is to find the algorithm with the best (minimum) predicted performance, the argmin function is used to find the minimum value in the prediction vector. Alternatively, the classification models want to find the algorithm with the highest probability of having the lowest cost. To do this, the argmax function is used, which returns the index of the maximum value in the probability distribution.

2.2 The Dataset

All the datasets were split into 80:20 for the training and validation sets respectively. A validation set helps monitor the model's performance on unseen data during training by comparing training loss and validation loss. The split ratio was chosen as there was enough data for the training model to learn the underlying patterns and relationships effectively with 80% of the data. The datasets used in Parts One and Two Basic, inherit the PyTorch dataset class with similar functionality to suit the models. For the implementation of Part Two Advanced, a performance parameter was added to the dataset to be used in the loss function.

The datasets are then iterated using the Dataloader class with batches of 32. This size was chosen to help the models converge quicker and to act as an implicit regularisation, see section 3.5 in Testing. The

provided dataset was not standardised (MiniZinc, 2012).Therefore, the data was standardised using the min-max standardisation method, shown in Equation 1.

$$x_{scaled} = \frac{x - x\_min}{x\_\max - x\_min}$$

Equation 1: Min-Max Equation

2.2 Neural Network Architecture

The neural network for Parts One and Two (Basic and Advanced) have the same architecture. A three fully connected linear layers, with rectified linear unit (ReLU) activation functions after the first and second layers. The number of hidden units in the first and second hidden layers is determined by calculating the sum of the input and output dimensions and multiplying the value by two thirds. This technique to determine the optimum number of neurons is based on the literature, *Introduction to Neural Networks for Java* (Heaton, 2008).

The three-layer architecture, including an input layer, hidden layer, and output layer, provides sufficient capacity for the model to learn from the data whilst keeping the overall computational complexity to a minimum. Rectified Linear Unit activations were implemented to introduce non-linearity into the model, which enables the network to learn complex patterns and relationships in the data.

2.3 Regression Model

Adam was chosen as the optimiser with a learning rate of 0.01 and weight decay of 0.001. The learning rate is a crucial hyperparameter that determines how much the model's weights are updated during each iteration of the optimisation process. During the testing phase of the model, this value proved to be reliable producing a lower loss due to the lower risk of overshooting the optimal solution. However, this meant that the model took longer to converge, hence why the patience was set to a larger epoch value. Mean square error (MSE) loss was used for the regression model, this is a standard loss function for regression problems due to its differentiable properties as well as the emphasising large errors. The reduction parameter was tested to view the impact on the loss metric. The reduction parameter controls how the individual squared differences between predicted and actual values are combined to produce the final loss value. The mean reduced MSE performed better than the sum reduced MSE and was therefore used in the final model.

2.4 Basic Classification Model

The optimiser, learning rate and, the weight decay is the same as the regression model. The loss function for the classification model was cross entropy loss. Cross-entropy loss measures the difference between two probability distributions: the true distribution of class labels and the predicted distribution from the model. This makes it a natural fit for classification problems, where the goal is to correctly assign probabilities to class labels. The reduction was tested similarly to the regression loss function and the results seen in testing show the sum reduced cross entropy loss demonstrated the best performance.

2.5 Advanced Classification Model

The optimiser, learning rate and, the weight decay is the same as the regression model. The loss function for the advanced classification model was a custom function. This loss function incorporated the cross-entropy loss function with a regret value. The regret value calculates the difference between the cost of the algorithm chosen by the model and the cost of the optimal algorithm for each instance. Regret thus

measures the model's performance against an ideal benchmark, pushing the model to minimise the disparity between its decisions and the optimal ones.

The final loss is a weighted combination of the Cross-Entropy loss and the Regret loss. The hyperparameter alpha is used to balance the influence of these two components. A higher alpha prioritises the reduction of Cross-Entropy loss, while a lower alpha prioritises Regret. This dual-aimed loss function drives the model to not only be accurate in its classifications but also to make choices that minimise the overall cost.

2.6 Pairwise Classification Model

The pairwise model was attempted and fully implemented. However, the model is not fully working. The accuracy produced by the model ranges from 0.01 to 0.02 and the SBS-VBS gap is 4500 - 4600. The model is designed to predict the performance differences between pairs of algorithms based on the provided features. A custom PyTorch Dataset was created to manage the data for these pairwise comparisons. Data normalisation was applied to ensure a consistent scale across the features. A custom loss function was used to compute the error between predicted and actual performance differences during training. It encourages the model to accurately predict which algorithm performs better.

All the evaluation is completed in the model instead of the evaluation file. This was to avoid any unnecessary complexities in the evaluation file and to speed up the process of debugging the model. Although a final model could not be established, the key infrastructure has been implemented and the key ideas have been displayed.

2.7 Random Forest Classification Model

The random forest model utilises the sklearn library, the model was built using the random forest classifier which allow for a quick implementation of the model. The evaluation also utilises the sklearn library, the same evaluation metrics as Parts One and Two are implemented. One issue that is occurring with this model is that there is no loss value. Therefore, a –1 value has been supplemented.

3 Testing

All the testing was completed before the final models were decided to determine the best model for the given task. To keep the results as accurate as possible, a random seed was incorporated into the system. This allowed the results to be reproducible throughout the testing. The SBS-VBS gap was chosen as the metric for the testing as this is the primary metric within this assessment. Therefore, by focussing on the gap, the system can be optimised accurately.

The neural network architecture is tested as the structure of the model can heavily influence the computational time taken to train the model. The weight decay is a powerful tool that can help prevent overfitting within the model. However, if the weight decay is too high, the model can become unreliable. Therefore, a right balance must be investigated for our given dataset. The learning rate is a key component within the training process that determines the effectiveness of the model. A small learning rate can lead to greater control over the predictions whilst increasing the time to converge. Alternatively, if the learning rate is too high then the optimal value could be neglected leaving to a less effective model. The learning rate and the weight decay are both parts of the optimisation function which forms the essential component in the training process. Therefore, a variety of optimisations will be tested to determine the appropriate choice for each model. Finally, another important aspect of the training process is the batch size. Smaller batch sizes introduce more noise into the gradient estimates, which can lead to faster convergence and better generalisation due to the implicit regularisation effect. However, smaller batch

sizes may also result in a less stable convergence. Larger batch sizes provide more accurate gradient estimates but can slow down convergence and potentially result in poorer generalisation. Thus, a balance needs to be determined.

3.1 Neural Network Architecture

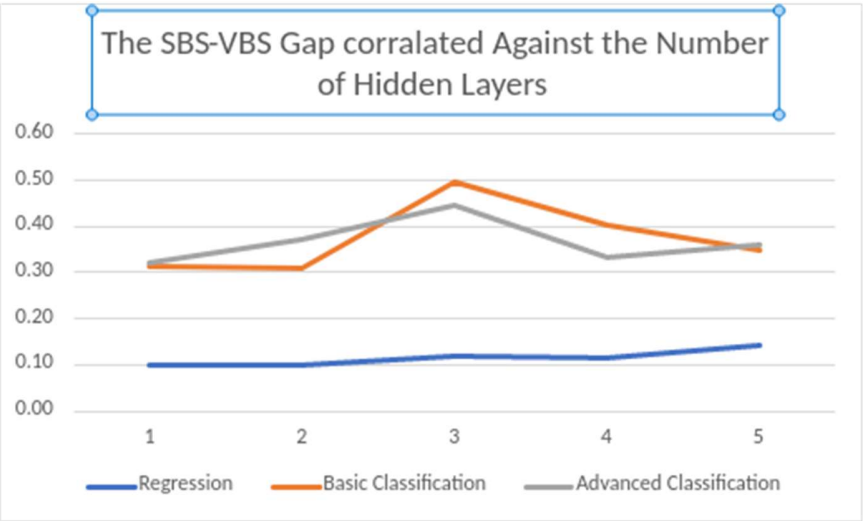

Figure 1: A Graph Showing How the Number of Hidden Layers Effects the SBS-VBS Gap for Each Model

All the neural networks were originally built using one hidden layer, this base model was chosen as it is recommended, when building a new neural network, to start from one hidden layer and build onto it to achieve further clarity of the data and establish the best architecture (Gad, 2018). From Figure 1, we can see that the basic and the advanced classification models as well as the regression model favour a lower number of hidden layers, indicating that the dataset is not complex in nature and will require a small number of hidden layers. This conclusion is further reinforced as we observe that as the number of hidden layers increases in Figure 1, the SBS-VBS gap also increases.

3.2 Weight Decay

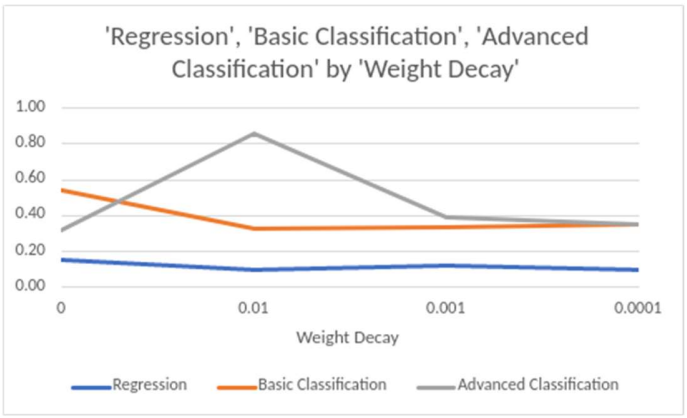

Figure 2: A Graph Showing How the Weight Decay Effects the SBS-VBS Gap for Each Model

The base model for the weight decay is zero as this is the base value provided by the PyTorch library. The standard value used for weight decay traditionally falls in the range of 0.001 and 0.0001 (Krogh & Hertz, 1991). Therefore, the weight tested are 0, 0.01, 0.001 and, 0.0001. From the data seen in Figure 2, we can observe that the best weight decays are 0.001 and 0.0001 for our models. However, as previously mentioned, this hyperparameter is used to prevent overfitting. Therefore, the lower value may, though predicting the smallest gap, be the most overfit. To evaluate this statement further we can refer to the validation-train loss line graph, see Figure 3. From these graphs we can see that for a weight decay of 0.0001 the validation loss line (orange) does separate from the training loss line (blue) indicating that the models are potentially overfit to the data which is caused by the low weight decay. Therefore, a weight decay of 0.001 will be applied to all models to provide a sufficient weight decay without underfitting or overfitting the data.

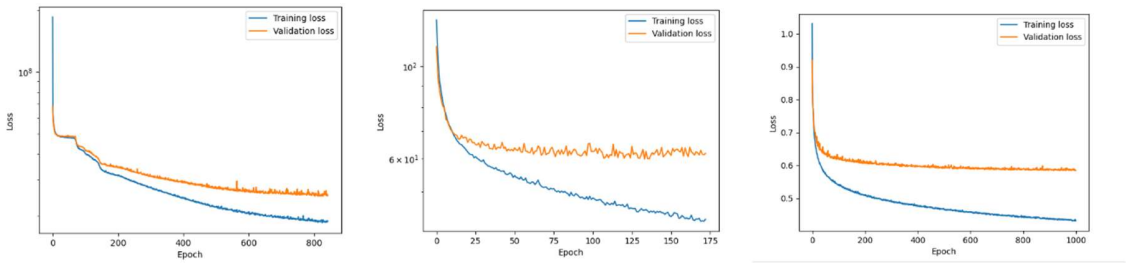

Figure 3: The Average Loss over Epochs for the Regression (Left), Basic Classification (Middle) and, Advanced Classification (Right) with a Weight Decay of 0.0001

3.3 Optimisation Function

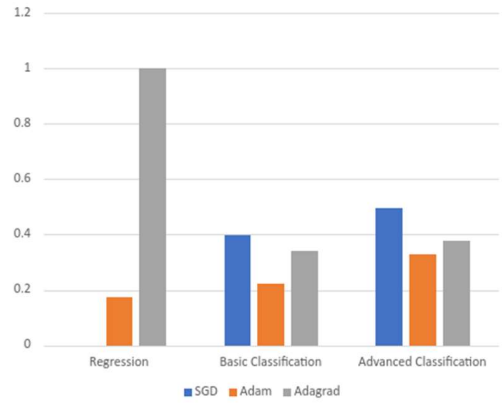

Figure 4: A Graph Showing How Each Model Reacts to Each Optimisation Function

After researching the various algorithms presented by the PyTorch library, Adam, Adagrad and, SGD were determined to be the most effective algorithms to test and evaluate, this is because they have a large history with neural networks (Heaton, 2008). From Figure 4, we can see how the different optimisation functions effect each model. Adagrad and SGD are both incompatible with the regression model, the cause of this may be that they are stuck in a non-optimal local minimum. However, a more likely cause for the results seen with Adagrad is the aggressive nature of Adagrad on the learning rate, causing the learning process to be prematurely slowed down before reaching the optimal solution. Overall, the Adam optimiser is seen to be the most effective optimiser for every model, this is an expected behaviour as

Adam is designed specifically for training deep neural networks, combining the advantages of Adagrd and RMSProp (Kingma & Ba, 2014).

3.4 Learning Rate

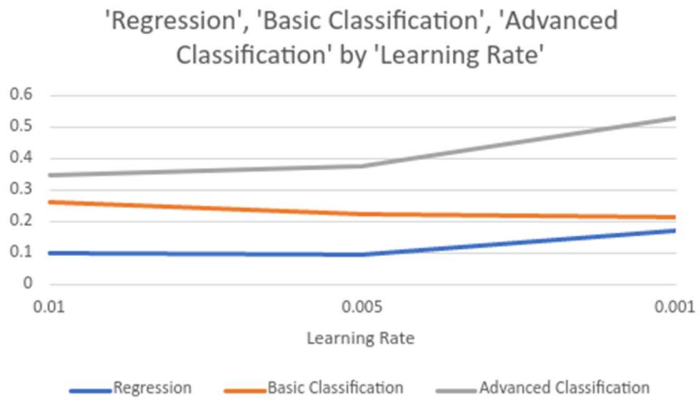

Figure 5: A Graph Showing How the Learning Rate Affects the SBS-VBS Gap for Each Model

The base model for the learning rate is 0.001 as this is the base value provided by the PyTorch library. The standard value used for learning rate traditionally falls in the range of the base value and 0.01 (Smith, 2017). Therefore, the weight tested are 0.01, 0.005 and, 0.001. From the results seen in Figure 5, as the learning rate increases, as does the SBS-VBS gap. Therefore, a learning rate of 0.01 will be used on all models.

3.5 Batch Size

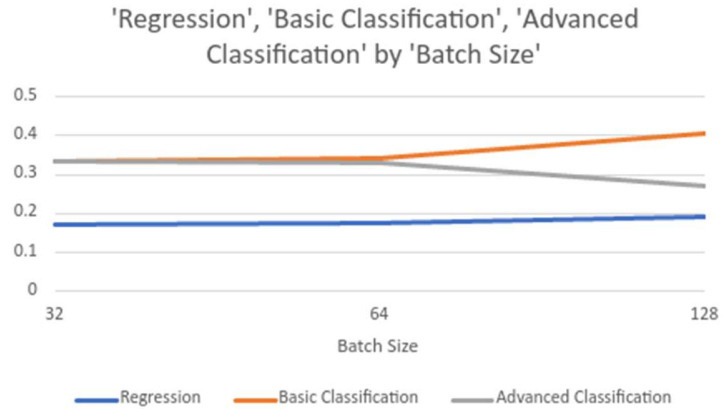

Figure 6: A Graph Showing How the Weight Decay Affects the SBS-VBS Gap for Each Model

A batch size of 64 was chosen as a start point as this value is small enough to avoid some of the time and space problems caused by large batch sizes, but large enough to take advantage of the speedups from vectorised operations and parallel processing capabilities of modern GPUs (Mishkin, Sergievskiy , & Matas, 2017). 32 and 128 were chosen to evaluate how batch size effects the loss. From Figure 6, we can observe that as the batch size increases, as does the SBS-VBS gap. Therefore, a small batch size of 32 will be introduced to the training process.

4 Evaluation

Evaluation Metrics	Regression	Basic Classification	Advanced Classification	Pairwise Classification	Random Forest Classification
Loss	22455427.7520	0.9544	0.5894	2509.1872	NA
Accuracy	0.3864	0.6781	0.6233	0.0248	0.7104
Average Cost	4483.5271	5463.3811	6198.4093	15601.6821	5430.8595
SBS-VBS Gap	0.1028	0.3118	0.4686	4688.3791	0.3048

Table 1: A Table of the Evaluation Metrics for Each Final Model

Firstly, the SBS-VBS gap metric seen within Table 1 were calculated based on the static values for VBS and SBS of 8690.0126 and 4001.6335, respectfully.

The Regression model appears to do well at optimising for cost and the SBS-VBS gap, which are the main aims of this task. However, the model appears to struggle with accuracy, which could be because it's primarily designed to reduce cost rather than to increase accuracy. The Basic Classification model, in contrast, is more focused on accuracy and predicting the correct labels, which results in higher accuracy but at the cost of a higher gap and cost. The Advanced Classification model seems to provide a good balance between the two, effectively integrating the Regret into the training and loss process. It could potentially perform the best with further optimisation of parameters, particularly the alpha value. The Random Forest Classification model, despite not being designed specifically for this task, also performs well, particularly in terms of accuracy, thanks to its ability to manage the complexity of the task and capture complex patterns in the data.

From the results, the Pairwise Classification model appears to not be functioning as intended. Because of this, the evaluation metrics are hard to evaluate and assess when compared to the other models. Therefore, this model will not be fully evaluated until future work is achieved to demonstrate suitable evaluation metrics.

In conclusion, of the models tested, the Random Forest Classifier delivered the highest accuracy, and a reasonably low SBS-VBS gap, suggesting it may be the most effective model for this specific task. However, it's important to note that each model has its strengths and weaknesses, and the choice of model can depend on the specific requirements of the task. For example, if the primary goal is to minimise the SBS-VBS gap, the regression model may be preferred despite its lower accuracy.

5 Conclusion

In conclusion, the exploration of the specified model architectures, combined with the innovative application of the loss functions, has yielded insightful results for the task. The performance of the proposed models has been evaluated with key metrics, including loss, accuracy, average cost, and the SBS-VBS gap. The final results obtained are quite promising, considering the complexity of the task at hand.

Additionally, the use of different loss functions, such as Mean Squared Error (MSE) for regression models, and Cross-Entropy Loss for classification models, have further helped in the effective training of the models. Moreover, the implementation of batch size normalisation and appropriate scaling of the input

features have contributed to the model's robustness against overfitting and underfitting, hence improving the model's generalisability on unseen data.

Future work will focus on the further optimisation of these models, exploring more sophisticated neural network architectures, and fine-tuning the hyperparameters to achieve even better results. Additionally, work would be needed to improve the accuracy of the Pairwise model and adjust the evaluation step to suit the model.

6 References

Gad, A. (2018, June 27). *towardsdatascience.* Retrieved from Beginners Ask "How Many Hidden Layers/Neurons to Use in Artificial Neural Networks?": https://towardsdatascience.com/beginners-ask-how-many-hidden-layers-neurons-to-use-in-artificial-neural-networks-51466afa0d3e

Heaton, J. (2008). *Introduction to Neural Networks for Java, 2nd Edition.* Heaton Research, Inc.

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.

Krogh, A., & Hertz, J. A. (1991). A Simple Weight Decay Can Improve Generalization. *International Conference on Neural Information Processing Systems*, 950–957.

MiniZinc. (2012). *minizinc*. Retrieved from MiniZinc Challenge 2012: https://www.minizinc.org/challenge2012/results2012.html

Mishkin, D., Sergievskiy , N., & Matas, J. (2017). Systematic evaluation of convolution neural network advances on the Imagenet. *Computer Vision and Image Understanding*, 11-19.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. *IEEE winter conference on applications of computer vision (WACV)*, 464-472.