The Effect of Neural Network Complexity on Learning and Training in Regression Tasks

Salomón Cardeño Luján
Mathematical Engineering
School of Applied Sciences and Engineering
EAFIT University
scardenol@eafit.edu.co

***Abstract*—This study investigates the relationship between neural network architecture and the progress of training and learning in a regression task. The study implemented a neural network without bias in Python, beginning with a single hidden layer containing one neuron and gradually increasing the complexity of the architecture to three hidden layers containing five neurons each. The dataset was split into three subsets: a training set, a validation set, and a test set. The training set was used to train the neural network and to monitor and analyze the progress of training. The network was trained on the training set using three different learning rates (0.2, 0.5, and 0.9), and the progress of the learning process was also analyzed using the validation set. The best, worst, and median models were selected based on a trade-off between the lowest network error (average energy) and the most parsimonious model (lowest number of model parameters). The performance of the best, worst, and median models were then evaluated on the test set to estimate the generalization error. The results suggest a relationship between network architecture and training and learning progress, where simpler architectures led to better training and learning performance. Additionally, the optimal learning rate varied with the complexity of the network architecture. These findings provide insights into the importance of carefully designing neural network architectures and selecting appropriate learning rates for different tasks, suggesting that increasing the complexity of a neural network architecture may not always lead to improved performance in regression tasks.**

***Keywords*—Neural network, Regression task, Training, Learning, Python.**

I. Introduction

Artificial neural networks have become increasingly popular for a wide range of applications, including regression tasks. In recent years, there has been a growing interest in understanding the impact of network architecture on the performance of these models. Specifically, researchers have explored the role of the number of hidden layers and neurons in the hidden layers on the training and learning progress of the neural network [1], [2]. With their findings suggesting that the network complexity does have a relevant impact in performance.

The goal of this study is to investigate if there exists a relationship between network architecture and training and learning progress in a regression task. Specifically, we aim to study the performance of a neural network that goes from a single layer architecture with 1 hidden layer and 1 neuron in the hidden layer to 3 hidden layers and 5 neurons in the hidden layer. To assess the performance of the models, we used three learning rates (0.2, 0.5, and 0.9) and compared the best, worst, and median models based on their network error (average energy) and the number of model parameters. This process is usually described in literature as going from a shallower (simpler) network to a deeper (more complex) network. An example to illustrate both architectures is shown in figure 1 below.

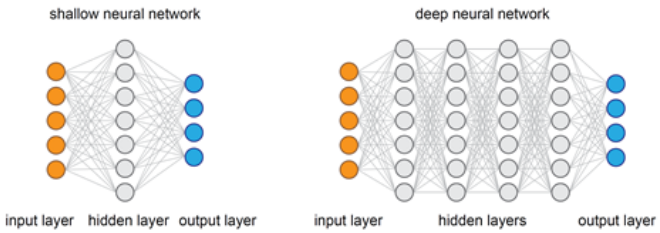

Fig. 1. Illustration of shallow and deep neural networks. Link to source.

The architecture of a neural network plays a crucial role in its ability to learn and generalize to new data. The number of layers and neurons in a neural network determines its complexity, which can impact both its learning speed and its ability to avoid overfitting. As such, it is important to study the relationship between network complexity and training performance to gain a better understanding of the behavior of neural networks. By investigating the impact of network architecture on training and learning progress, we can gain insights into how to design more efficient and effective neural networks for a variety of tasks. In this study, we investigate the effect of increasing network complexity on training and learning performance in a regression task, in which the goal is to predict a continuous output value from a set of input features.

II. Methodology

The methodology employed in this study involves the implementation and training of neural networks with varying degrees of complexity, ranging from a single hidden layer with one neuron to a network with three hidden layers and five neurons in each hidden layer. The networks were trained on a regression task, with the objective of predicting a continuous output value from a set of input features. The training process

was optimized using a gradient descent algorithm, and early stopping was used to prevent overfitting. The behavior of the network was monitored and analyzed using the network error and gradient of the output layer. The data set used for training and testing was provided by the course professor, and contains two input features and one output. The data was normalized using min-max normalization to bring it into the $[0, 1]$ interval, and was split into training, validation, and testing sets with a ratio of 60%, 20%, and 20%, respectively. In this section, we provide a detailed description of the methodology used to carry out the study, including the implementation and training of the neural networks, the data pre-processing steps, and the selection of the best models based on network error and parsimony.

A. Data

The data consists of 2 input features and 1 output with 3000 observations of a dynamical phenomenon. The actual nature and context of the data its unknown, but upon inspection we see that the data is in different scales and that it's all numerical, hence the nature of the regression task. The first 2 columns where selected as the input features and the last one as output.

We take a non-random sample of the first 500 observations to preserve the time nature of the data where the first 60% corresponds to the training set, the following 20% corresponds to the validation set and so on.

In order to ensure that all inputs features are treated equally and to deal with the different scale of both the inputs and outputs, we perform a min-max normalization to scale the data to $[0, 1]$ with the usual formula

$$X_{norm} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Where X corresponds to the feature or output to be normalized, $X_{\min}$ to the minimum of said data, $X_{\max}$ to the maximum and X_{norm} to the resulting data.

B. The role of the train, validation and test sets

In machine learning, the role of the training, validation, and test sets is essential for building accurate and reliable models. The training set is used to train the model by adjusting the weights of the network and minimizing the training error (average energy error). After training, the model's performance is evaluated on the validation set, which helps in selecting the best model based on a predefined selection criterion. The validation set is not used for training and is kept separate to ensure that the model is not overfitting to the training data. Finally, the test set is used to evaluate the model's performance on new data that the model has not seen before. This ensures that the model's performance is not biased towards the training or validation data and gives a more realistic estimate of the model's performance.

C. Initial conditions: the weights

The networks we consider are dense, meaning that each neuron from a specific layer its connected to each other neuron from the next layer. The strenght of said connections are represented mathematically as the weights $\mathbf{W}$. We constrain the initial space for the weights so that each weight is in the hypercube $[-1, 1]$. In practice, we randomly select the values for each weight from that interval with equal probability (uniform random selection).

D. Network error or loss function

To define the loss function we use, we first need to define the instantaneous energy error [3]

$$\mathcal{E} = \frac{1}{2} \sum_{j \in L}^{N} e_j^2$$

Where N is the number of observations, j are all the neurons in the field L and $e_j = d_j - y_j$ is the error defined as the difference between the desired output d_j and the real output y_j. The instantaneous energy error can be interpreted as a measure of the lack of effectiveness of all the neurons j in the field L. Now we define the loss function as the average energy

$$\mathcal{E}_{av} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{E}$$

Where we average over all observations n. The average energy error can be interpreted as the complete network error.

E. Hyperparameters and experiment

The idea is to increase the model complexity in base or number of neurons, and in depth or number of layers. We start with a shallow network of 1 neuron at its base and 1 layer of depth and work our way up by gradually increasing the base and then the depth until reaching the deepest most complex network considered of 5 neurons at its base and 3 hidden layers of depth. We consider the same number of neurons for each hidden layer, meaning we end up with 15 models so far and not the entire possible combinations. As for the learning rate, we consider three: 0.2, 0.5 and 0.9, resulting in a total of 45 different models or network architectures.

We consider a sigmoid activation function for every layer and this choice remains constant throughout the whole experiment. The number of maximum epochs was set to 50 where we update the weight parameters with gradient descent on each epoch. An early stopping is possible if the network error is less than $1e - 2$.

F. Selection criterion

Based on the role of the train, validation and test sets we apply a model selection criterion using the validation set. We select the best model as the model that results with the lowest network error (energy average) in the last epoch and its the most parsimonious (lowest number of parameters). Initially, we used the Bayesian Information Criteria not as a loss function, but as a score of both criteria. The BIC can be defined as

$$BIC = k \ln N + 2 \ln \mathcal{E}_{av}$$

Where k is the number of model parameters, N is the number of observations and $\mathcal{E}_{av}$ is the average energy (network error).

This expression was deduced from the classical expression where the context its to maximize the likelyhood estimator, but because it is equivalent to minimize the negative of the loss function we replace the second termn and obtain the above expression. To calculate the number of model parameters its easy to deduce them from the context of the architecture we are considereing

$$k = n * l_i * L * m$$

Where n is the number of inputs, l_i is the number of neurons on each hidden layer, L is the number of hidden layers and m is the number of outputs. Using the BIC as a score for model selection appeared to be promesing but it was concluded after further experimentation that it resulted in a score highly biased to the model's parsimony overshadowing the importance of the network error.

Because of this, a simpler yet effective criterion was considered: order the models by their last epoch network error (lowest to highest) and order that resulting list by their number of parameters (lowest to highest). This results in the first element being the best model, the last being the worst model and the median being the median model which is the $\lfloor l/2 \rfloor$ element of that list, where l is the length of the list. A seed its used in order to make reproducible results.

G. Code implementation

The whole code implementation was done in the operating system `Microsoft Windows 11 Home Single Language v.10` and the programming language `Python 3.10.9` as native as possible with the use of the libraries `numpy`, `pandas`, `matplotlib.pyplot` and the native library `os`. The input data was provided originally as `datosIA.mat` which is a Matlab file but was converted to the text file `datosIA.txt` to avoid using uncommon libraries. The neural network is implemented without bias and in batch. The algorithm used to update the weights on each epoch of the backpropagation is gradient descent. The experiment described before literally translates to a nested for loop with 3 levels from inner to outter: first we iterate over the base or number of neurons l_i, then we iterate over each of the learning rates lr and then we iterate over the depth or number of hidden layers L of the network.

The script takes the input data and outputs the plot results to a folder. The input data is provided with the script.

III. Results and Discussion

Because of how the experiment was designed, we gathered the results by each layer and, therefore, are presented here in the same per depth manner.

A. One hidden layer $L = 1$

In the case of one hidden layer $L = 1$ the base number was increased from $l_i = 1$ to $l_i = 5$ neurons per hidden layer for learning rates of $lr = 0.2, 0.5, 0.9$. In the case of the training set, figure 2 below shows the network error and figure 3 the local gradient for the output layer for the best (blue), worst (orange) and median (green) models.

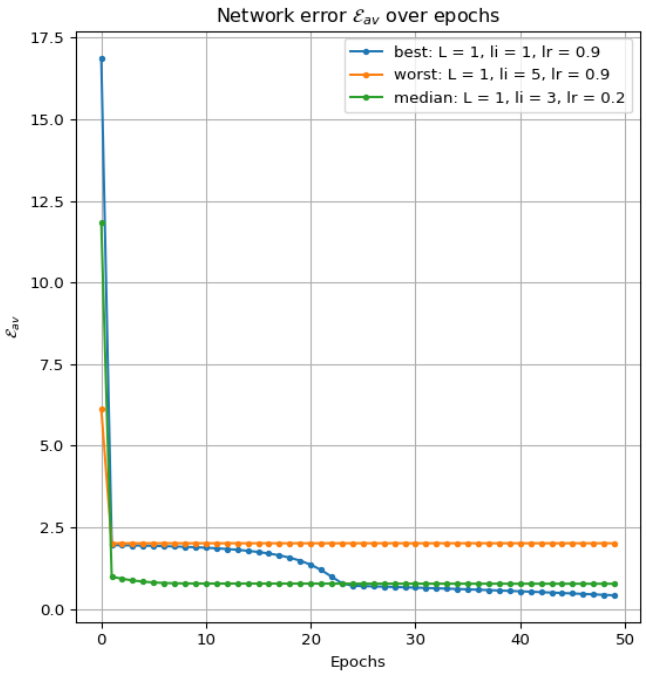

Fig. 2. Network error of the train set for the best, worst and median models with one hidden layer.

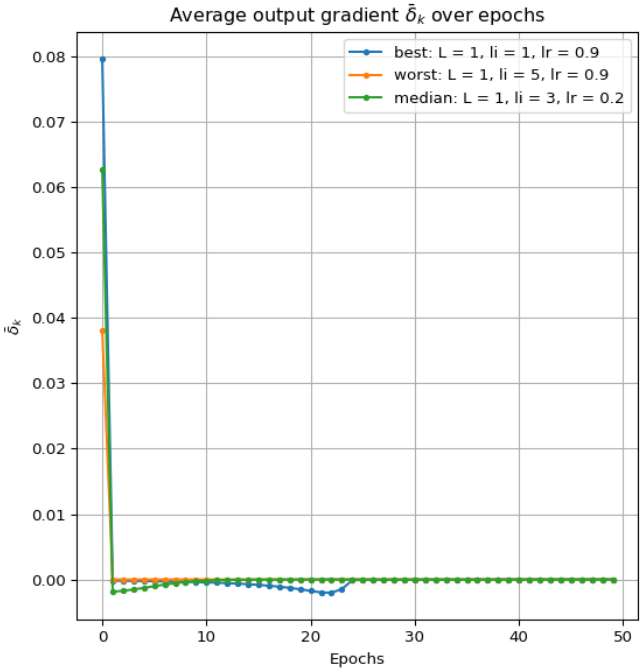

Fig. 3. Local gradient of the output layer of the train set for the best, worst and median models with one hidden layer.

The best model for this depth was a network with the simplest architecture having 1 hidden neuron and a learning rate of 0.9. The worst model was the most complex network

for this depth with 5 hidden neurons and a learning rate of 0.9. The meadian model was a network with 3 hidden neurons and the lowest learning rate of 0.2. We observe that the network error for the best model was the lowest at the last epoch even after starting from the highest error on the first epoch, on the other hand the worst model had the highest network error on the last epoch even after starting from the lowest error on the first epoch. The network error for the median model was more consistent, starting with an error about the middle of the previous errors on the first epoch but ending with an error closer to the best model on the last epoch. On al the three models the highest change of network error happened from the first epoch to the second. By looking at the resulting average output gradient of the output layer we see the same starting behaviour which makes sense as a higher network error implies a higher average output gradient because of their dependency on the error e_j. We observe that the gradient of the best model had a similar behaviour to the median model from epoch 2 until epoch 10 and then deviated from epoch 11 to 24 until converging at 0. Because of the connection of both measures, this behaviour of the gradient explains the behaviour for the same epoch intervals when observing the network error.

In general, the training behaviour for this depth was consistent with what was expected of a desired learning set stage: evidence of minimization of the network error or loss function.

In the case of the validation set figure 4 below shows the network error and figure 5 shows the local gradient of the output layer for the best (blue), worst (orange) and median (green) models.

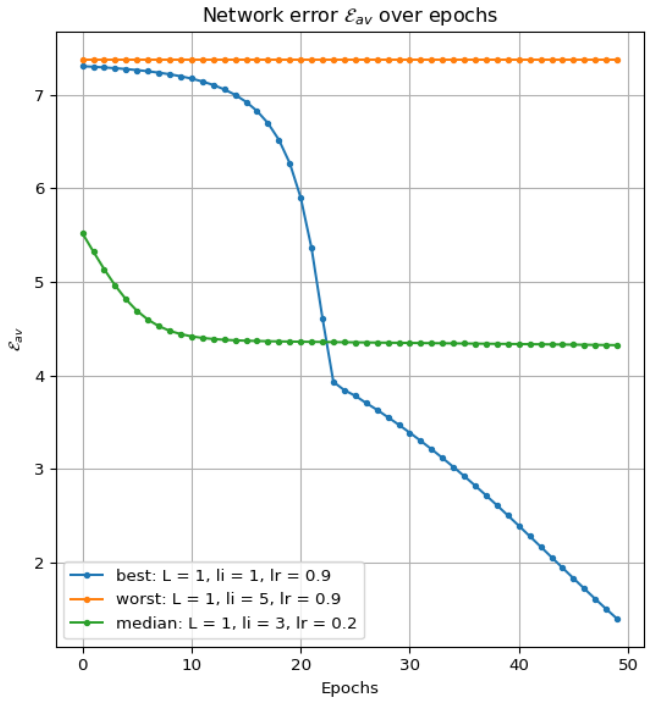

Fig. 4. Network error of the validation set for the best, worst and median models with one hidden layer.

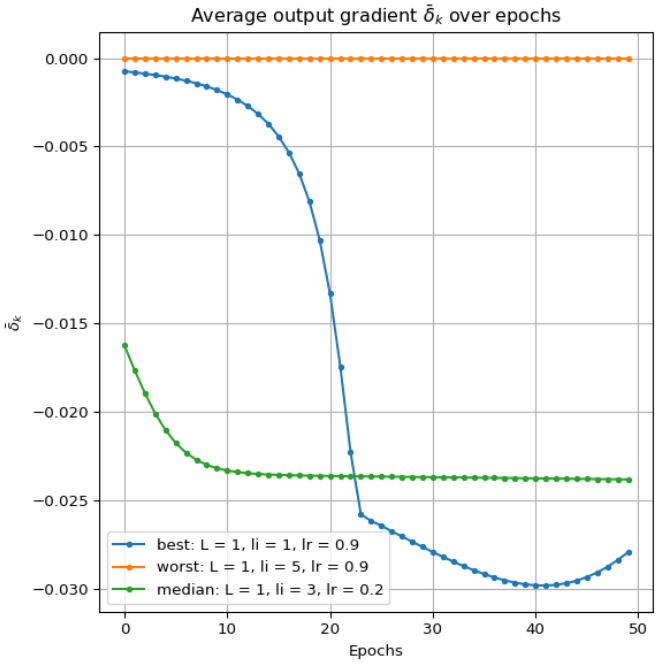

Fig. 5. Network error of the validation set for the best, worst and median models with one hidden layer.

By providing unseen data to evaluate the training performance of each network we guarantee that a model selection from this results is not biased to the training set. From the network error we observe the reason for each model selection. At the very worst we have the model with the highest number of parameters for this depth and which presented no underfit or overfit in the scale comparable to the other models as the error did not increase nor decrease, this same behaviour can be observed in the output gradient which was not active thus explaining said behaviour. In the case of the median model we observe a slow decrease in the network error in the first 10 epochs and then an even slower convergence into a fixed value. This behaviour is explained by a very similar behaviour of the output gradient. The best model showed the highest change in error. It started with a very high network error similar to the worst model, then decreased slowly until around epoch 15 when it started to decrease rapidly and about approximately a network error of 4 it changed to an almost lineal descent resulting in the lowest network error. This behaviour can be explained by observig the output gradient for the best model which behaves the same until it reaches the same epoch where the network error changed its behaviour and also changes to a parabolic behaviour.

In general, the validation behaviour for this depth was consistent with the model selection and provided an unbiased evaluation of every model training progress.

Lastly, in the case of the test set figure 6 below shows the output predictions for the best, worst and median models compared to the real output.

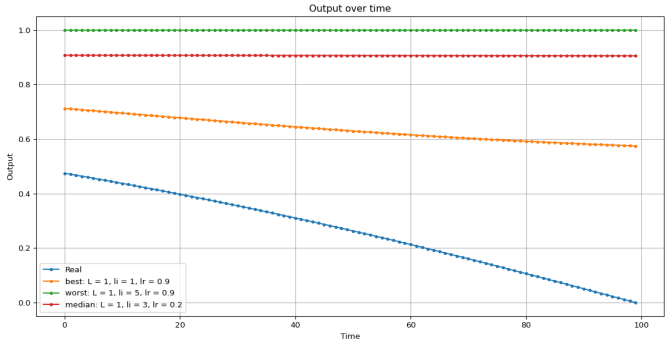

Fig. 6. Output preditction of the best, worst and median models with one hidden layer compared to the real output.

We observe that even the best model could not generalize good enough to result in a closer behaviour to the real output, but the fact that its the closest behaviour to the real output and the difference with the predictions from the median and worst model shows that there is in fact a learning progress.

From these results we can conclude that for a depth of $L = 1$ there is a relationship between network architecture and the progress in training and learning.

B. Two hidden layers $L = 2$

The same methodology for a depth of one layer was applied to the case of two hidden layers $L = 2$. In the case of the training set, figure 7 below shows the network error and figure 8 the local gradient for the output layer for the best (blue), worst (orange) and median (green) models. The worst model converged to a higher network error and had its gradient stopped at epoch 2, resulting in an early bad convergence to the highest network error.

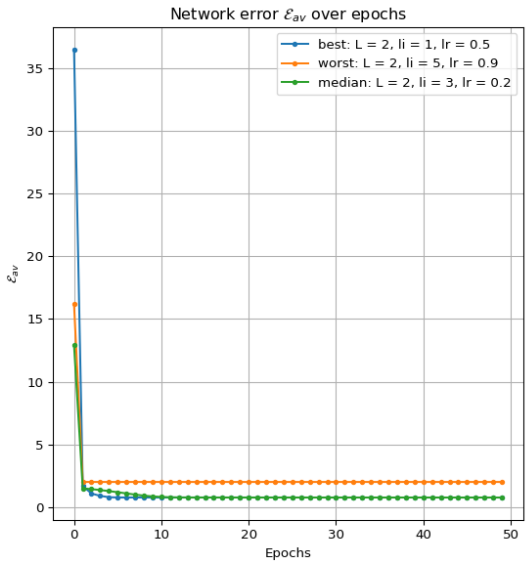

Fig. 7. Network error of the train set for the best, worst and median models with two hidden layers.

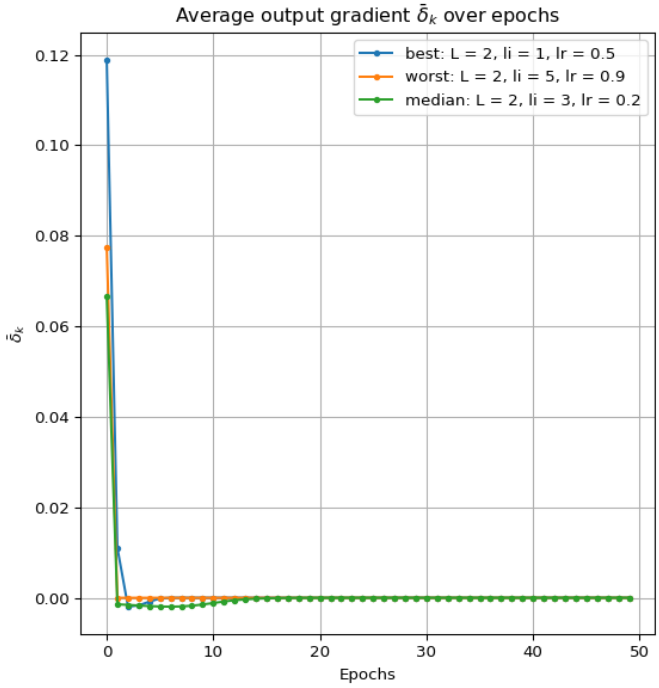

Fig. 8. Local gradient of the output layer of the train set for the best, worst and median models with two hidden layers.

We observe that the best, worst and median models for this depth resulted in the same number of base neurons as the same models for a depth of one layer, but the learning rate was different for the best model with a lower rate of 0.5 for the case of a depth of 2 hidden layers. The best model showed the same behaviour of starting at the highest network error and output gradient and then finishing in the lowest value for both measures. The behaviour of the median model was similar to that of the best model but it started with the lowest values of network error and output gradient. The results obtained in the training set were consistent to what was expected in the minimization of the network error.

In the case of the validation set, figure 9 below shows the network error and figure 8 the local gradient for the output layer for the best (blue), worst (orange) and median (green) models.

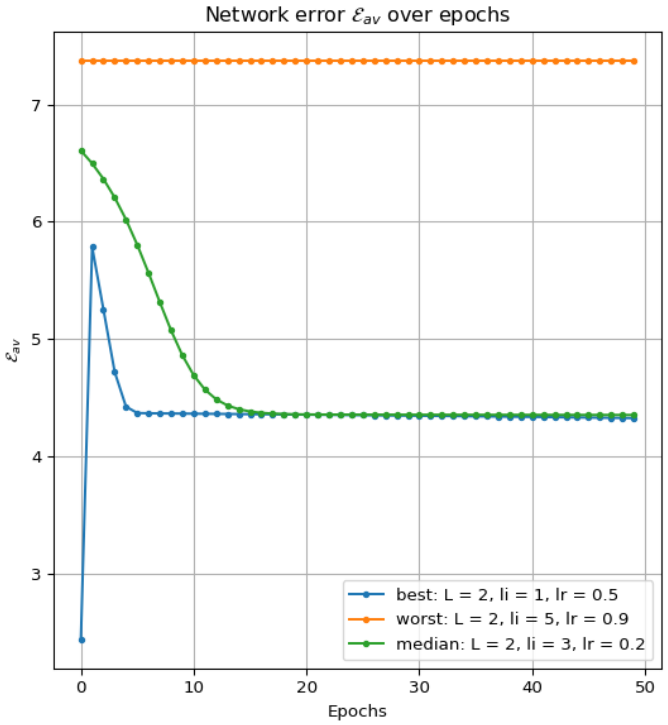

Fig. 9. Network error of the validation set for the best, worst and median models with two hidden layers.

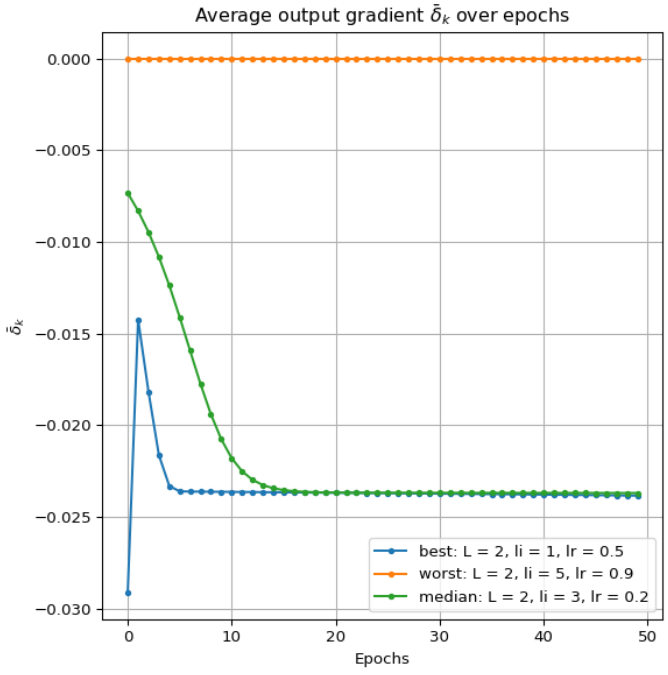

Fig. 10. Local gradient of the output layer of the validation set for the best, worst and median models with two hidden layers.

The worst model converged to a higher network error and had its gradient stopped at epoch 2, resulting in an early bad convergence to the highest network error. The results obtained for the behaviour of the network error and average output gradient for the case of the worst and median models were very similar to that of the previous depth. On the other hand, the behvaiour of the best model was different by actually starting with the lowest network error and gradient, highly increasing both values at the second epoch and then slowly decreasing until convergence due to its lower learning rate.

Lastly, in the case of the test set figure 11 below shows the output predictions for the best (orange), worst (green) and median (red) models compared to the real output (blue).

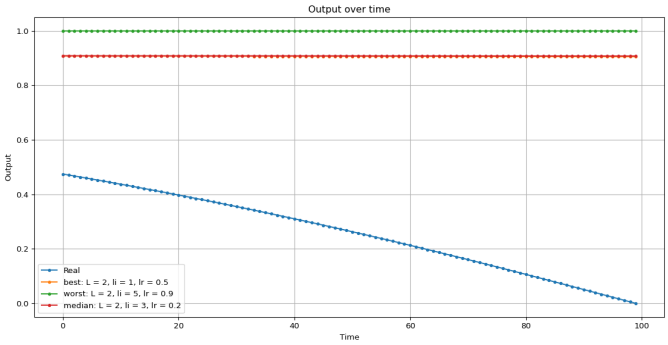

Fig. 11. Output preditction of the best, worst and median models with two hidden layer compared to the real output.

The results for this depth were very similar for the median and worst models from the previous depth of one layer. Nonetheless, the predictions of the best model resulted worst in this depth with a behaviour almost exactly as the median model.

From these results we can conclude that for a depth of $L = 2$ there is a relationship between network architecture and the progress in training and learning.

C. *Three hidden layers* $L = 3$

The same methodology for a depth of two layers was applied to the case of three hidden layers $L = 3$. In the case of the training set, figure 12 below shows the network error and figure 13 the local gradient for the output layer for the best (blue), worst (orange) and median (green) models.

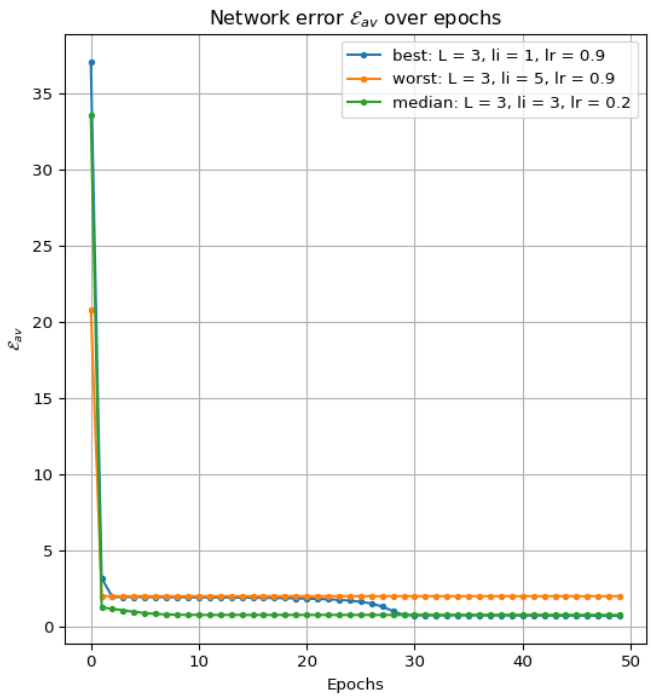

Fig. 12. Network error of the train set for the best, worst and median models with three hidden layers.

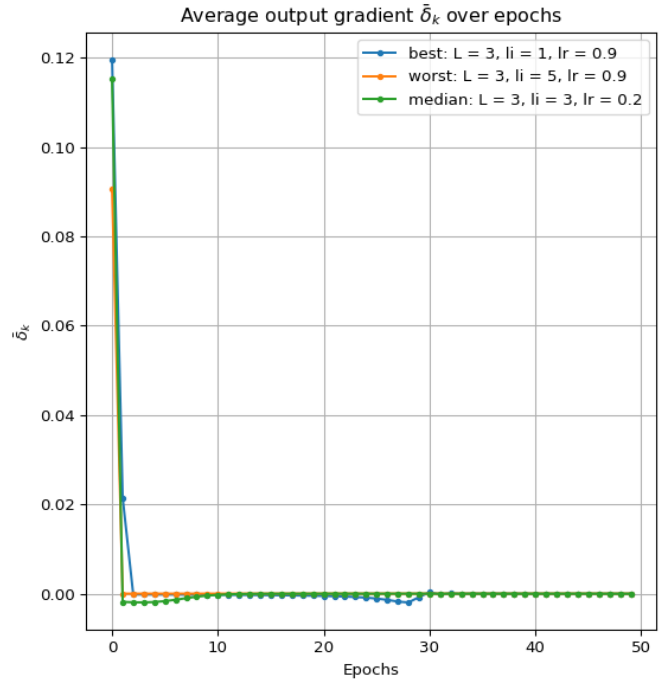

Fig. 13. Local gradient of the output layer of the train set for the best, worst and median models with three hidden layers.

The worst model converged to a higher network error and had its gradient stopped at epoch 2, resulting in an early bad convergence to the highest network error. We observe that the best, worst and median models for this depth resulted in the same number of base neurons as the same models for a depth of two layers, but the learning rate was different for the best model with a higher rate of 0.9 for the case of a depth of 3 hidden layers. The best model showed the same behaviour of starting at the highest network error and output gradient and then finishing in the lowest value for both measures, but its behaviour was similar to the worst models on early epochs. The behaviour of the median model was similar to that of the best model but it started with a lower value of network error and output gradient. The results obtained in the training set were consistent to what was expected in the minimization of the network error.

In the case of the validation set, figure 14 below shows the network error and figure 13 the local gradient for the output layer for the best (blue), worst (orange) and median (green) models. The worst model converged to a higher network error and had its gradient stopped at epoch 2, resulting in an early bad convergence to the highest network error.

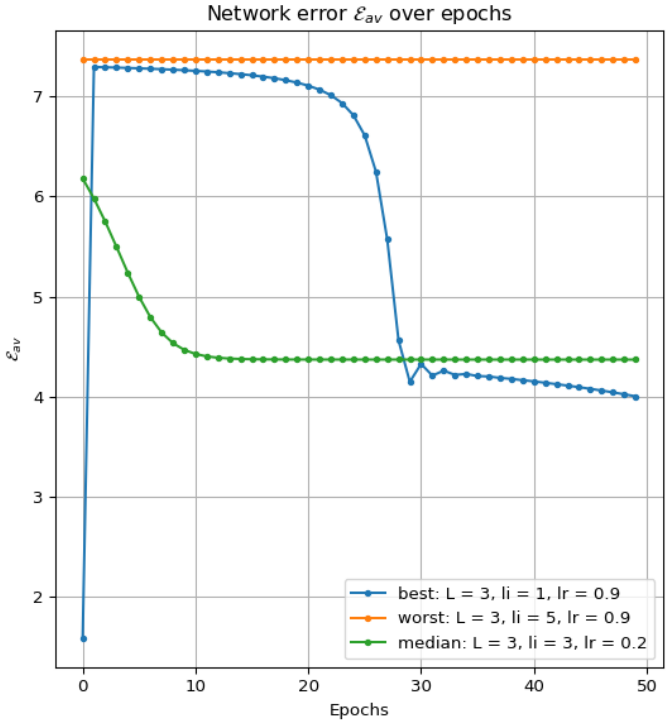

Fig. 14. Network error of the validation set for the best, worst and median models with three hidden layers.

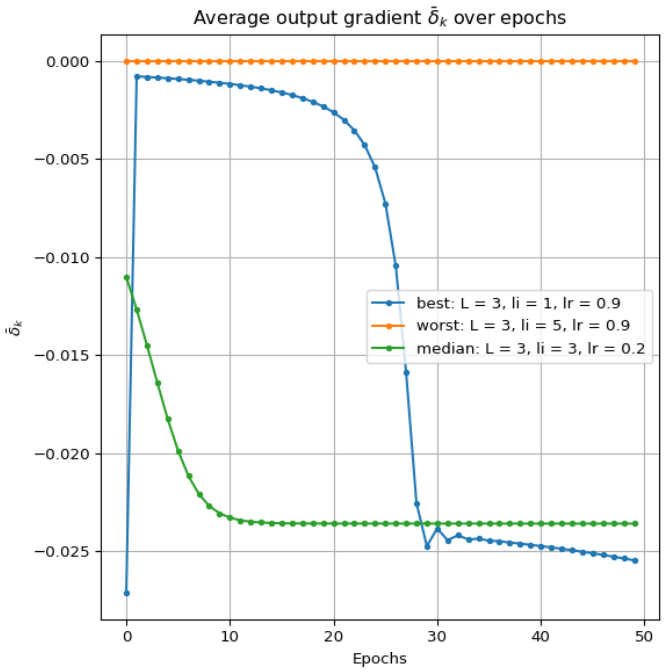

Fig. 15. Local gradient of the output layer of the validation set for the best, worst and median models with three hidden layers.

The results obtained for the behaviour of the network error and average output gradient for the case of the worst and median models were very similar to that of the previous depth. On the other hand, the behvaiour of the best model was different by actually starting with the lowest network error and gradient, highly increasing both values at the second epoch reaching almost the same value as the worst model and then slowly decreasing until epoch 20 were a higher decrease started until some oscilatory behaviour at iteration 30 and the slowly decreasing.

Lastly, in the case of the test set figure 16 below shows the output predictions for the best (orange), worst (green) and median (red) models compared to the real output (blue).

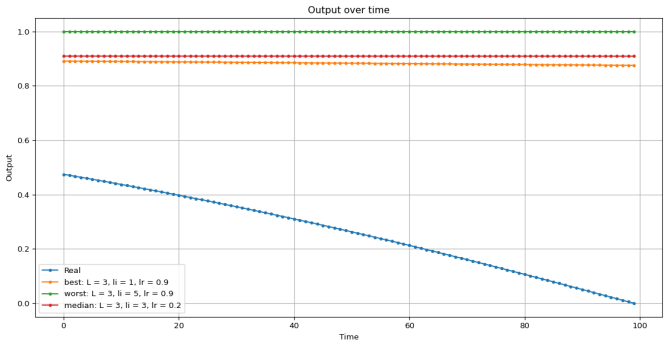

Fig. 16. Output preditction of the best, worst and median models with three hidden layer compared to the real output.

The results for this depth were very similar for the median and worst models from the previous depth of two layers. Nonetheless, the predictions of the best model resulted better in this depth than in the case of a lower depth of two hidden layers.

IV. Conclusion

The results suggest enough evidence to conclude that there exists a relationship between network architecture and the progress in training and learning. In this particular regression task the relationship showed that a less complex model results in a better progress in training and learning, leaning towards higher learning rates than lower learning rates. One thing to notice is that the best model was worst when increasing the depth from one layer to two layers, but was best when incresing from two layers to three layers, nonetheless, the more shallow network performed better overall.

References

[1] J. de Villiers and E. Barnard, "Backpropagation neural nets with one and two hidden layers," in IEEE Transactions on Neural Networks, vol. 4, no. 1, pp. 136-141, Jan. 1993, doi: 10.1109/72.182704.

[2] M. Bianchini and F. Scarselli, "On the Complexity of Neural Network Classifiers: A Comparison Between Shallow and Deep Architectures," in IEEE Transactions on Neural Networks and Learning Systems, vol. 25, no. 8, pp. 1553-1565, Aug. 2014, doi: 10.1109/TNNLS.2013.2293637.

[3] Course notes.