**Final project**

**Dr. Yakir berchencko**

**Authors – Doron Levi, Amir Yativ**

**Ben Gurion University**

**Final Project - Report**

**General Background and Presentation of the Project's Essence**

Neural networks are computational systems that play a central role in the fields of artificial intelligence and machine learning. Their primary purpose is to process information, learn from it, and draw conclusions. The underlying concept is to build a model composed of tiny computational units called "neurons," which are interconnected in a structure resembling a network. Each neuron contributes to solving the problem by performing simple calculations and passing the results to neurons connected to it in the next layer (Figure 1). Through these simple calculations, more complex functions can be created, with the goal of accurately representing the behavior and patterns present in the underlying data. This enables the prediction and classification of information in a qualitative and efficient manner.

**Figure 1:** example of a fully connected neural network with input, hidden, and output layers.

Top of Form

Bottom of Form



The neural networks examined in this project are trained by presenting examples of problems and their corresponding solutions. This process, known as "training," involves the network adjusting the connections between neurons to recognize patterns and enhance the accuracy of its predictions. The network begins with an input layer, where raw data relevant to the phenomenon under investigation is received. This data is then passed through intermediate layers, known as "hidden layers," where complex computations are performed to identify essential features and patterns within the data. Finally, the results are obtained in the output layer, which provides a prediction or classification based on the input information.

Neural networks are applied in various fields, including image recognition, natural language processing (understanding text and speech), quantitative forecasting, and many other areas. These capabilities make neural networks a powerful and innovative tool in the realm of modern technology.

Our project focuses on comparing two different approaches to utilizing neural networks, with the aim of exploring and analyzing the findings, and ultimately improving the machine learning process through neural networks.

The first approach we concentrated on involved constructing and training a basic default neural network, where the key hyperparameters of this network include: a fully connected network with multiple hidden layers (ranging from 4 to 60 layers depending on the experiment), 10 neurons per layer, a BFGS optimizer, and a ‘ReLU’ activation function (detailed explanations of these hyperparameters and their impact can be found in the Literature Review section). This network serves as a baseline for comparing other learning approaches to evaluate the quality of the predictions obtained, with a focus on minimizing the Mean Squared Error (MSE) — a metric that reflects the quality of regression models .

Alternatively, the second approach focuses on 'Ensemble’ Learning, where we employed a multitude of neural networks sharing the same architecture as the baseline network, which served as the reference point (a fully connected network with identical layers and neuron counts per layer as the baseline). To carry out the learning process, we employed a group of identical neural networks and calculated an appropriate weight for each network (denoted as vector β, more details on this can be found in the Literature Review ). After computing this vector, the output results of each network were used to produce an improved final prediction.

After calculating the 𝝱 vector, we can use it to make predictions for other samples. To do so, we need to apply the same networks that were used to determine the 𝝱 vector and run the desired sample through them. Once we have obtained the predictions from all the networks, we will multiply them by the 𝝱 vector, resulting in the desired prediction.

The overarching goal of the project was to investigate whether using an ‘Ensemble’ learning-based algorithm could enhance the quality of learning and predictive capability in regression tasks.

**Literature review**

In this paper, we will compare the predictive capabilities of a basic and deep neural network with an ‘Ensemble’ algorithm, which we will elaborate on in subsequent sections. We did not find existing literature that could serve as a foundation for this comparison. As we could not locate any existing studies that directly support this comparison, we drew upon the book *" Neural Networks From Scratch* (*Harrison Kinsley & Daniel Kukieła, 2020)"* to construct a robust and effective deep neural network. Additionally, we leveraged the expertise of our instructor, Dr. Berchenko Yakir, to guide the understanding and development of the ‘Ensemble’ algorithm. The following section will outline the methodology used to construct the ‘Ensemble’ algorithm, focusing on the relevant variables.

**M** – This matrix represents a collection of predictions based on the existing data. The matrix is structured such that the rows correspond to the samples used in the experiment, while the columns represent simple and basic neural networks that have not yet undergone any learning process. Each cell within the matrix contains the output of the neural network for the corresponding sample. For instance, in the matrix **M**, the entry at position [1][2] displays the output of the second neural network after it was applied to the first sample.

**Y** – This vector consists of the true values of the samples at our disposal.

**𝝱** – This vector is composed of scalars, and its multiplication by our prediction matrix (**M**) should yield an accurate estimate of the true values.

To facilitate a comparison with the ‘Ensemble’ algorithm, we implemented a basic deep neural network as a baseline for our study. This network was configured with the following parameters: a BFGS optimizer, a ReLU activation function, and four layers, each containing 10 neurons. This foundational network serves as a reference point, allowing us to evaluate its performance against the ‘Ensemble’ model. The ‘Ensemble’ model is constructed by aggregating biases and weights derived from random sub-sampling of independently built networks.

The construction of the ‘Ensemble’ network involved initially creating several basic neural networks (which remained untrained), denoted by X. Subsequently, we sampled X samples from the dataset and processed them through the constructed networks. The output for each sample was recorded in a matrix M, which is a square matrix. Based on this matrix, we formulated the vector 𝝱, composed entirely of scalars, calculated using the following formula:

To solve this equation, we transformed it into the following form:

Top of Form

Bottom of Form

We identified the appropriate β value.

**Application of the Algorithm for Prediction Using the β Vector**

Once the β vector has been determined, the algorithm can be applied in the following manner:

1. **Selection of the Sample**: First, choose the sample for which a prediction is desired.
2. **Input to Neural Networks**: Next, input the selected sample into the X neural networks that were utilized to create the matrix M (the same networks used in the computation of the matrix).
3. **Prediction Calculation**: Finally, multiply the outputs obtained from the neural networks in step 2 by the β vector to generate the desired prediction.

The process of applying the β vector for prediction is grounded in the principle of weighted averaging. Each neural network in the ensemble generates an output based on the input sample, and the β vector acts as a set of coefficients that determines the contribution of each neural network's output to the final prediction. The β vector is calculated during the training phase, where it is optimized to minimize the difference between the predicted values and the actual target values in the training data. This optimization ensures that the resulting prediction is as close as possible to the true values, making the ensemble method more robust and accurate compared to relying on a single neural network.

This approach works effectively because the ensemble of neural networks, each with its unique perspective on the data, can capture a broader range of patterns and relationships within the data. By assigning different weights (through the β vector) to the outputs of these networks, the algorithm can emphasize the strengths of each network while mitigating their individual weaknesses. The final prediction combines the strengths of each neural network in the ensemble, leveraging their diverse perspectives to produce a more balanced and accurate outcome, guided by the carefully calibrated β vector.

The ‘Ensemble’ algorithm is predicated on the assumption that the random construction of deep neural networks without training tends to produce simple functions. In the paper "*Simplicity Bias in Overparameterized Machine Learning(Anonymous, 2023)",* it is argued that there is a simplicity bias in overparameterized models, suggesting that our algorithm is likely to exhibit strong generalization capabilities. Additionally, the paper notes that the type of optimizer does not influence the simplicity bias. In our experiment, we choose to use the BFGS optimizer primarily due to its computational efficiency. BFGS is recognized for its efficiency compared to other optimizers, such as SGD (Stochastic gradient descent), owing to its computation technique that avoids heavy mathematical operations and utilizes function approximations to perform the necessary calculations. Regarding another hyperparameter, the activation function, we selected ‘ReLU’ after trial and error with other activation functions, such as the ‘Sigmoid’ function. Our tests indicated that ‘ReLU’ outperformed the alternatives. For determining the number of hidden layers and the optimal number of iterations, we conducted experiments to identify the optimal values for each hyperparameter, as detailed later. The learning rate hyperparameter was set to the recommended value of 0.01.

**Methodology**

Our experiments were designed to compare the predictive accuracy between an ‘Ensemble’ neural network learning approach, which we created from scratch, and a basic default neural network, also created from scratch. The main objective of these experiments is to determine whether high-quality and efficient machine learning can be achieved using an ‘Ensemble’ neural network model compared to a basic default network, particularly in scenarios where there is a relatively large number of features compared to the number of samples (an "overparameterized" model).

We collected the data for the experiment from the Kaggle website, focusing on housing prices in Australia. We chose this dataset because of its relatively high number of features (80 features) compared to the number of samples (1,460 samples).

The objective of using these data was to examine whether the quality of learning in situations with an excess of parameters can be improved through the ‘Ensemble’ learning approach by either not training or only partially training the neural networks. This method may help in creating simple functions with a high probability, in accordance with the simplicity bias theory, and subsequently performing an intelligent weighted averaging of these functions.

After reviewing all the raw data and understanding its significance, we documented the information in an explanatory file that details the meaning of each feature. We then began filtering the raw data to exclude variables that had no value for the learning task. Additionally, we processed the data and divided it into several subgroups (performed Feature Representation), with each subgroup undergoing appropriate processing to maximize the quality of the network results we constructed.

The methods we used for feature representation are:

* **Normalization**: Adjusting the data to a range between 0 and 1, aimed at improving the stability and performance of the algorithm. We applied this in two ways:

1. Directly converting the range of values to the interval between 0 and 1 (for example, dividing by 10 features where ratings are given on a scale of 1 to 10).
2. Replacing a numerical value with its corresponding percentile relative to the rest of the samples (e.g., apartment size).

* **One-Hot Encoding**: Transforming categorical variables into a vector that makes the categories accessible (both for those with an ordinal significance and those without), by creating a binary column for each category.
* **Binarization**: A process in which categorical data is converted into binary values (0 / 1) based on the original data. The goal of this action is to simplify the dataset and make it more suitable for machine learning algorithms.

Once the database was ready, we began developing the neural networks in Python. Initially, we created a basic default neural network, drawing from the book "*Neural Networks from Scratch (Harrison Kinsley & Daniel Kukieła, 2020)*", while deliberately avoiding the use of Python libraries, except for ‘pandas’ for data processing, ‘numpy’ for mathematical computations, and ‘scipy.optimize’ which assisted in creating the BFGS optimizer. We avoided using additional libraries to gain better control over the network's settings on one hand, and to gain a deeper understanding of how the neural network we were using operates on the other. The key hyperparameters we defined for this network included: a Fully Connected network with 4 hidden layers, 10 neurons in each layer, a maximum of 100 iterations, the BFGS optimizer, and the ‘ReLu’ activation function.

After developing the basic default neural network, we proceeded with the development of a neural network based on the ‘Ensemble’ approach. In this network, similar to the default neural network, we developed everything from scratch in Python (using only the “pandas” library for data processing and the “numpy” library for mathematical calculations). The main principle guiding us in building this network was to run a large number of neural networks with the same structure. Here too, similar to the default neural network, the key hyperparameters we defined were: a fully connected network with 4 hidden layers and 10 neurons in each layer. By running the data through these networks, we found a vector, 𝝱, which allows us to understand the weight assigned to each network in order to find the optimal solution. This process is essential and critical for performing the prediction using the ‘Ensemble’ approach.

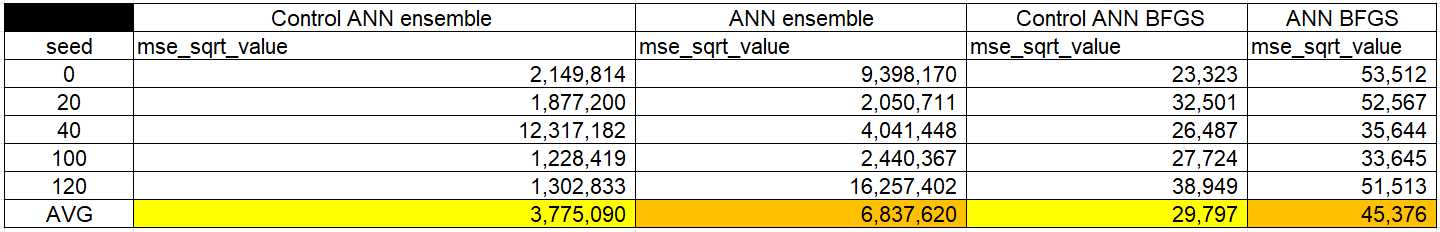
During this project, we conducted three main experiments using neural networks that we developed:

1. **Objective:** The purpose of this experiment was to evaluate the quality of the code we wrote and ensure it functions logically as intended. Additionally, we wanted to assess whether there is an improvement in the MSE metric when using an ‘Ensemble’-based neural network learning algorithm compared to a basic default neural network learning algorithm.  
    **Experiment Procedure:** For each neural network we built from scratch (both the default and the ‘Ensemble’), we created a parallel network using Python libraries such as ‘sklearn.neural\_network’ and ‘sklearn.metrics’. These networks served as control networks to verify the quality of our code, ensuring that the MSE metric was on the same scale as the networks we developed from scratch. Furthermore, we compared the results of the two networks we built from scratch to understand which one achieves better MSE results, helping us assess the quality of the networks. To ensure the comparison was as objective as possible, we maintained an identical structure for all the networks included in this experiment, consisting of a Fully Connected network with 4 hidden layers, 10 neurons in each layer, a BFGS optimizer, and a ‘ReLu’ activation function. Additionally, to ensure that the results were not based on outlier values resulting from edge cases, we conducted the experiment 8 times and averaged the MSE results obtained during each run.
2. **Objective:** The aim of this experiment was to examine the impact of the depth of the neural network (the number of hidden layers in each network) on the quality of the resulting predictions, and to identify the optimal network depth for both the basic default neural network and the ‘Ensemble’ network. In this experiment as well, we evaluated the quality of the predictions using the MSE metric.  
    **Experiment Procedure:** We examined the optimal number of layers for the basic default network, with the range of layers tested varying between 10 and 40, in increments of 10 layers. Additionally, we tested the optimal number of layers for the ‘Ensemble’ network, with the range of layers tested varying between 10 and 60, also in increments of 10 layers. To ensure that other hyperparameters did not influence the results, we kept all other parameters constant: a Fully Connected network, 10 neurons in each layer, a BFGS optimizer, and a ‘ReLu’ activation function. To ensure that the results were not influenced by outlier values or edge cases, as in the previous experiment, we repeated the experiment 8 times and averaged the resulting MSE values.
3. **Objective:** The goal of this experiment was to investigate the effect of partial training of the ‘Ensemble’ network compared to the starting point where the ‘Ensemble’ network is not trained at all, in an attempt to improve the prediction quality of this model. In this experiment as well, we used the MSE metric to evaluate the prediction quality.  
   **Experiment Procedure:** First, we determined the optimal maximum number of iterations to be set as a parameter for the basic default neural network to perform optimal training and minimize the MSE metric. We conducted the experiment for Max iteration values ranging from 50 to 500, in increments of 50 iterations. To isolate the Max iteration from other hyperparameters, we used the same hyperparameters in every run: a Fully Connected network with 4 hidden layers, 10 neurons in each layer, a BFGS optimizer, and a ‘ReLu’ activation function. Again, to ensure that the results were not based on outlier values resulting from edge cases, we conducted the experiment 10 times and averaged the MSE results obtained. Next, we began comparing partial training of the basic neural network with partial training of the ‘Ensemble’ neural network. For each network, we varied the maximum iterations between 5 and 65 in increments of 10, while keeping the other hyperparameters constant (similar to previous experiments). Additionally, to ensure that the results were not influenced by outliers or extreme cases, we conducted the experiment five times and calculated the average of the resulting MSE metrics.

**Results**

**in the first** **experiment** we conducted, our aim was to assess the quality of the code we wrote and ensure that the neural networks we developed functioned logically as intended. The attached table (Table 1) displays the results of five runs for each network we developed, as well as for the corresponding benchmark network. Additionally, we calculated the average results of the runs for each network.

**Table 1**: Quality Assessment of Developed Networks and Networks Based on Libraries



In comparing the ANN BFGS with its benchmark network, Control ANN BFGS, we observe that the results are of the same order of magnitude (averaging 29,7979 in the control network and 45,376 in the network we developed from scratch). This outcome supports the assertion that the BFGS network we developed indeed functions as expected. However, there is a slight advantage in favor of the control network, which might be attributed to the fact that libraries sometimes use sophisticated weight initialization methods that can enhance network performance. In our case, we chose to initialize the weights randomly, which may have impacted the learning process.

When comparing the ANN ‘Ensemble’ to its control network, the Control ANN ‘Ensemble’, it can be observed that the results obtained are of the same order of magnitude (an average of 3,755,090 for the control network and 6,837,620 for the network we developed from scratch). This outcome further supports the claim that the ‘Ensemble’ network we developed indeed functions as expected. However, a slight advantage can be seen in favor of the control network, which could be due to the fact that libraries sometimes use sophisticated weight initialization techniques that can enhance the network's performance. In our case, we chose to initialize the weights randomly, which may have impacted the learning process itself.

A comparison of the average results obtained for the networks we developed using BFGS and ‘Ensemble’ methods indicates a significant difference between these networks (45,376 and 6,837,620, respectively). This result suggests that the learning quality of the BFGS network is significantly better than that of the ‘Ensemble’ network we developed. Based on these results, it can be inferred that the ‘Ensemble’ network performs learning less effectively than the BFGS network, which may be due to the assumptions we made about the model not being fully realized in practice.

**In the second experiment** we conducted, we aimed to determine the optimal number of hidden layers for each network we developed. The attached table (Table 2) shows the results of eight runs for each network, where the only difference between the networks is the number of hidden layers (all other parameters are identical across the networks). After obtaining the results, we calculated the average results for each network.

**Table 2:** Results of the Root Mean Square Error (RMSE) in relation to the number of hidden layers and the average of these results in the basic default network.

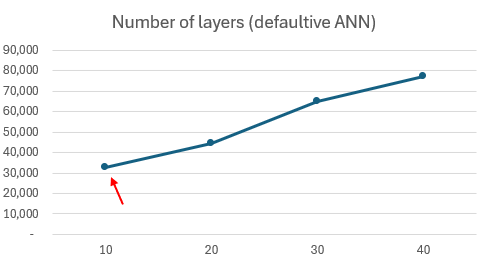
A table with numbers and a yellow tab

Description automatically generated

In the basic default neural network we developed, we found that the ideal number of hidden layers is 10, as the average root MSE in this case is 32.779. As the number of hidden layers increases, the root MSE metric also increases, indicating that these networks provide less accurate results.

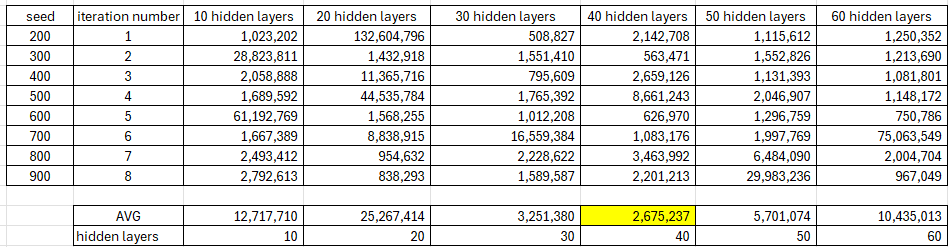
In the graph (Figure 2), it can be observed that for this neural network, as the number of hidden layers increases, the average root MSE also increases. Therefore, with 10 hidden layers, the best result is achieved.

**Figure 2:** The average results of the Root Mean Square Error (RMSE) metric in relation to the number of hidden layers in the basic default network.



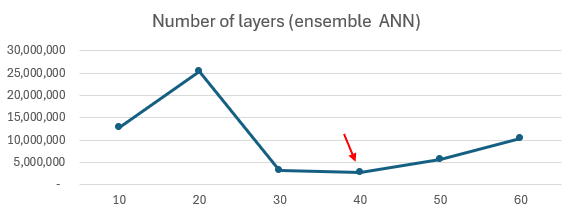
In the ‘Ensemble’ network we developed, we found that the optimal number of hidden layers is 40, as shown in the table (Table 3). This configuration resulted in an average root MSE of 2,675,237.

**Table 3:** results of the root MSE Index in relation to the number of hidden layers and their average in the ‘Ensemble’ network



In the graph (Figure 3), it can be observed that the optimal number of hidden layers for the ‘Ensemble’ network is 40, as this is the point at which the lowest RMSE (Root Mean Square Error) value is achieved. When the number of hidden layers is either less than or greater than 40, there is a significant increase in this metric, indicating that the network learns less effectively.

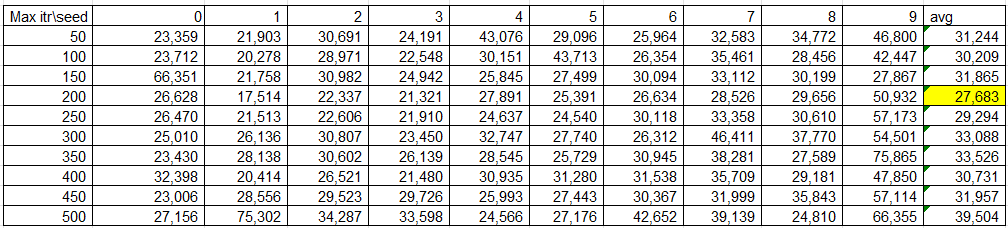
**Figure 3**: The average results of the Root Mean Square Error (RMSE) measure in relation to the number of hidden layers in the ‘Ensemble’ network.



**In the third experiment** we conducted, we aimed to examine how partial training of the neural networks we developed affects their performance. While our primary focus was on the ‘Ensemble’ network, we also tested the default network to compare the quality of results between the ‘Ensemble’ and the default configurations.

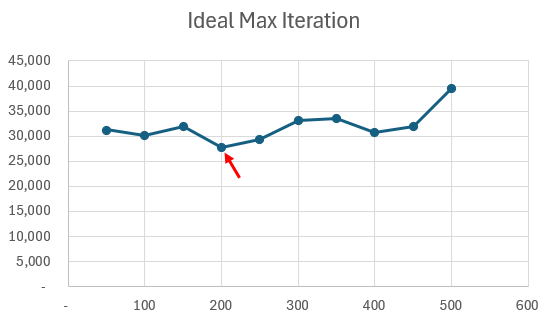
In the attached table (Table 4), the results of ten runs on the default network are presented. For each network we developed, the only difference was the maximum number of iterations, while all other parameters remained the same across the networks. It can be observed that with 200 max iterations, the lowest and therefore best root mean square error (RMSE) value (27.683) was achieved.

**Table 4**: Examining the optimal maximum number of iterations for the default network



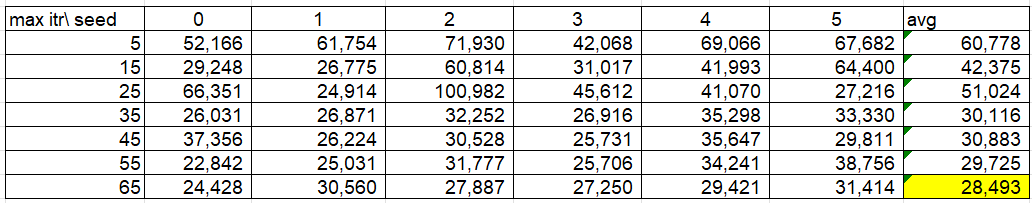
The graph (Figure 4) shows the average Root Mean Squared Error (RMSE) relative to the maximum number of iterations for the default network. It can be observed that the minimum average RMSE is obtained when the maximum number of iterations is 200 (27.683). At this point, the RMSE value is significantly lower compared to the other iterations.

**Figure 4:** The graph shows the average MSE (Mean Squared Error) in relation to the number of iterations.



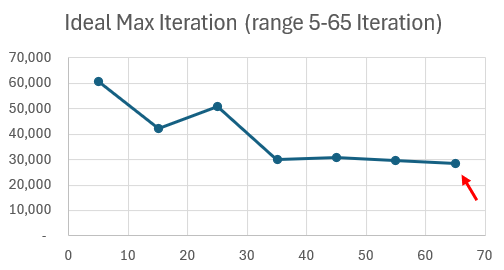
In the attached table (Table 5), you can see the results of six runs of the default network. In each run, only one parameter was changed – the maximum number of iterations – while all other parameters remained the same across all networks. It is evident that with 65 maximum iterations, the Root Mean Square Error (RMSE) metric is the lowest, indicating better performance (28.493).

**Table 5**: Partial training of the maximum number of iterations for the default network.



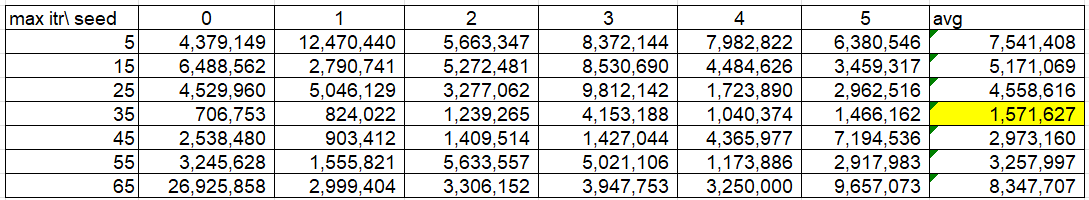
The graph (Figure 5) shows that the Root Mean Square Error (RMSE) metric varies according to the number of maximum iterations set for the training. As the number of iterations increases, the RMSE gradually decreases, indicating an improvement in the model's performance. Clearly, the minimum average RMSE is achieved when the maximum number of iterations is set to 65, with an RMSE value of 28.493. Therefore, according to the graph, setting the max iteration to 65 is the ideal choice to achieve the best performance of the model

**Figure 5:** Average Results of the Partial Training for the Maximum Number of Iterations in the Default Network.



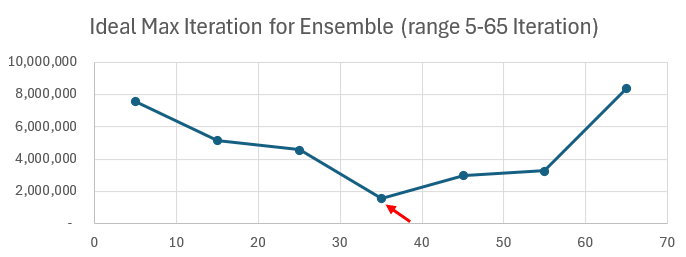
In the attached table (Table 6) , you can see the results of six runs for the ‘Ensemble’ network. The only difference between the networks is the maximum number of iterations, while all other parameters remained identical for all of them. It is evident that with 35 iterations, the lowest and therefore best Root Mean Squared Error (RMSE) value is obtained, with a value of 1,571,627.

**Table 6:** Partial training with the maximum number of iterations for the ‘Ensemble’ network



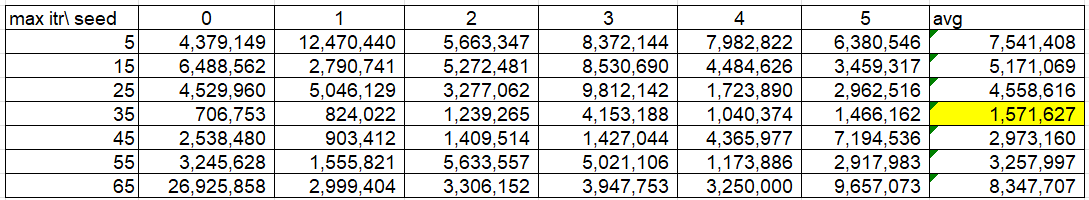
The graph (Figure 6) shows that the minimum average value of the root mean square error (RMSE) is achieved when the maximum number of iterations is 35, with a value of 1,571,627. This indicates that at this point, the model's performance is the best among all the iterations tested, as the metric value is the lowest, signifying a smaller error and higher accuracy of the model.

**Figure 6:** Partial training average of the maximum number of iterations for the ‘Ensemble’ network.



In the attached table (Table 6), the results of six runs for the ‘Ensemble’ network are presented. The only difference between the networks is in the maximum number of iterations, while all other parameters remained the same for all of them. It can be observed that with 35 iterations, the lowest and therefore the best Root Mean Squared Error (RMSE) is achieved, with a value of 1,571,627.

**Table 6:** Partial training with the maximum number of iterations for the ‘Ensemble’ network



In the graph (Figure 7), it can be observed that the minimum average of the Root Mean Square Error (RMSE) index is obtained when the maximum number of iterations is 35, with a value of 1,571,627. This means that at this point, the model achieves its best performance among all the iterations tested, as the RMSE value is the lowest, indicating a smaller error and higher accuracy of the model.

**Figure 7:** Partial training average of the maximum number of iterations for the ensemble network.

A graph with a line and a red arrow

Description automatically generated

**Discussion**

The main conclusion from the experiments we conducted is that the ‘Ensemble’ network failed to provide predictions of equivalent quality to those of the default neural network. We hypothesize that potential reasons for this may include discrepancies between our underlying assumptions related to the model and the actual conditions. To investigate whether there were issues with our assumptions, we conducted an additional experiment (Experiment 2) designed to assess if the network size affected the experiment's outcomes. As indicated by the relationship between the network size and its learning ability, increasing the number of hidden layers did indeed improve predictive performance, but only to a limited extent. From this, we can infer that the relatively small size of the network did hinder its ability to perform quality learning, but this was not the primary reason for its failure to provide predictions of satisfactory quality, unlike the default neural network.  
Additionally, we investigated whether partial training of the network could improve learning quality. Similar to the previous experiment, partial training did enhance performance to some extent, but not sufficiently. This might be due to the trade-off between increasing the number of iterations, which allows the function to better adjust between the input and the resulting prediction, and the increasing complexity of the network after too many iterations. In cases like the ‘Ensemble’ function, the network’s overfitting due to averaging with the 𝝱 vector might lead to an overly complex function that fails to provide high-quality predictions. Therefore, after an initial increase in the prediction metric, it began to decline after a maximum of 45 iterations. From this, we can conclude that partial training indeed improved the network's performance and enabled more effective learning, but even in this case, it was not the primary reason for its failure to provide predictions of sufficient quality compared to the default neural network.  
As a result, we conclude that there may be additional assumptions that were not met in the model we developed using the ‘Ensemble’ approach:

1. The relationship between the data and the true values may not be as straightforward as we initially assumed. The algorithm we developed is based on the assumption that there is a simple linear relationship between the data and the true predictive values. Therefore, we chose to represent it using simple networks that were randomly initialized with a bias of 0 and a vector of scalars (vector 𝝱). However, this approach may fail to capture the true complexity required for data analysis and quality prediction.
2. Additionally, the issue may lie in the fact that there is no perfect fit for the data. In such a case, even if we combine simple yet different neural networks, the result may become overly complex. When performing a weighted average of several such networks, a function that is too complex may be created, which fails to accurately estimate housing prices. Instead of improving prediction accuracy, combining overly complex functions could impair performance, as the network may struggle to identify the basic patterns guiding the data. In such cases, the pursuit of a perfect fit might lead to a loss of generalization ability, which is crucial for creating a reliable and accurate predictive model.

For future projects related to ‘Ensemble’ networks, we recommend focusing on two main directions: First, it is essential to explore additional hyperparameters of the model and investigate whether tuning them can enhance the model's performance, thereby making the Ensemble approach a viable option for use in data analysis and machine learning. Second, it is advisable to evaluate the performance of the Ensemble network on various datasets and data types to further expand the knowledge in this field regarding the effectiveness of Ensemble networks across a broader range of scenarios. Insights from these studies could shed light on the potential of the approach and assist in improving its application in the future.

**Bibliography**:

* Kinsley, H., & Kukieła, D. (2020). Neural Networks from Scratch in Python. Sentdex Kinsley Enterprises Inc.
* Anonymous. (2023). Simplicity Bias in Overparameterized Machine Learning. Paper under review for the International Conference on Learning Representations (ICLR).

**נספחים**

התכונות שנמצאות בכל קטגוריה:

את התכונות הבאות חילקנו לפי קטגוריות ( one hot encoding ):MSZoning, Alley, LotShape, LandContour, LotConfig, LandSlope, Neighborhood, Condition1, Condition2, BldgType, HouseStyle, RoofStyle, Exterior1st, Exterior2nd, MasVnrType, Foundation, Heating, Electrical, Functional, GarageType, PavedDrive, Fence, SaleType, SaleCondition.

את התכונות הבאות חילקנו לאחוזונים: LotFrontage, LotArea, YearBuilt, YearRemodAdd, MasVnrArea, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, 1stFlrSF, 2ndFlrSF, LowQualFinSF, BsmtFullBath, BsmtHalfBath, FullBath, HalfBath, BedroomAbvGr, KitchenAbvGr, TotRmsAbvGrd, Fireplaces, GarageYrBlt, GarageCars, GarageArea, WoodDeckSF, OpenPorchSF, EnclosedPorch, 3SsnPorch, ScreenPorch, PoolArea, MiscVal, MoSold, YrSold.

לתכונות הבאות שינינו את הסקאלה: OverallQual, OverallCond,

את התכונות הבאות המרנו למספרים: MSSubClass, ExterQual, ExterCond, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, HeatingQC, KitchenQual, FireplaceQu, GarageFinish, GarageQual, GarageCond, PoolQC,

קבוצת התכונות שהמרנו לייצוג בינארי - CentralAir

קבוצת התכונות שהורדנו מבסיס הנתונים - Utilities, BsmtFullBath, MiscFeature