Predictive Modeling
Homework 3. Neural Networks application

Jorge Alejandro Díaz Sánchez

Introduction

For this homework, I am analyzing the dataset provided at https://archive.ics.uci.edu/dataset/374/appliances+energy+prediction.

The collected dataset contains temperature and humidity measurements within various rooms in a house, alongside climatic measurements outside the house. The primary aim is to ascertain whether the total energy consumption within the house can be stimulated using a neural network model. The total energy consumption is the sum of two variables: Appliances and Light.

Data Description

The dataset contains the following columns:

Variable Description	Explanation
date time	Date and time of the recording (in the format: year-month-day hour:minute:second)
Appliances	Energy consumption from appliances (in Wh)
lights	Energy usage of light fixtures in the house (in Wh)
T1	Temperature in the kitchen area (in Celsius)
RH_1	Humidity in the kitchen area (in %)
T2	Temperature in the living room area (in Celsius)
RH_2	Humidity in the living room area (in %)
T3	Temperature in the laundry room area (in Celsius)
RH_3	Humidity in the laundry room area (in %)
T4	Temperature in the office room (in Celsius)
RH_4	Humidity in the office room (in %)
T5	Temperature in the bathroom (in Celsius)
RH_5	Humidity in the bathroom (in %)
T6	Temperature outside the building (north side) (in Celsius)
RH_6	Humidity outside the building (north side) (in %)
T7	Temperature in the ironing room (in Celsius)
RH_7	Humidity in the ironing room (in %)
T8	Temperature in teenager room 2 (in Celsius)
RH_8	Humidity in teenager room 2 (in %)
T9	Temperature in the parents' room (in Celsius)
RH_9	Humidity in the parents' room (in %)

To	Temperature outside (from Chièvres weather station) (in Celsius)
Pressure (from Chièvres weather station)	Atmospheric pressure (from Chièvres weather station) (in mm Hg)
RH_out	Humidity outside (from Chièvres weather station) (in %)
Windspeed (from Chièvres weather station)	Wind speed (from Chièvres weather station) (in m/s)
Visibility (from Chièvres weather station)	Visibility (from Chièvres weather station) (in km)
Tdewpoint (from Chièvres weather station)	Dewpoint temperature (from Chièvres weather station) (in °C)
rv1	Random variable 1 (nondimensional)
rv2	Random variable 2 (nondimensional)

Date time, rv1 and rv2 will be excluded from the analysis.

The target is the sum of “Appliances” and “light”.

There are in total 24 descriptors, 1 target and 19,735 samples in this dataset.

Data Preprocessing

There are no missing values. We can proceed to see analyze the skewness:

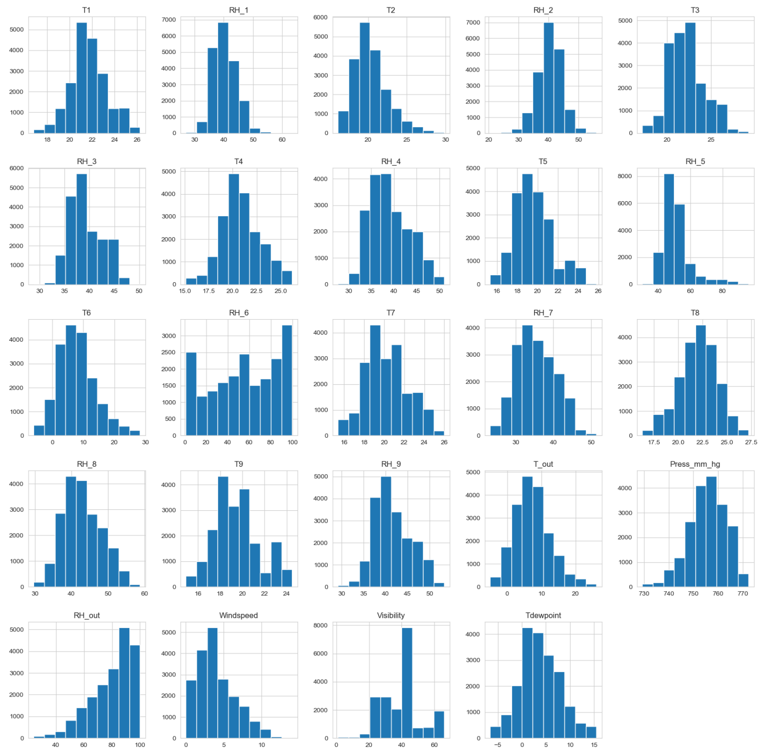

Figure 1 Skewness of predictors

The distribution of the data seems to be fair. There is not much skewness visually for most of the variables. I am determining a range of [-1.5,1.5] to be an acceptable skew value. There is only one variable that is out of this range, which is RH5, with a skew value of 1.8668. I am applying a power transformation $a = a^{(1/3)}$, and it brings down the skew to 1.4165.

Furthermore, I apply z-score normalization to each variable to ensure a standardized scale across all features.

These two processes are crucial to prevent saturation within the neural network. Saturation can cause neurons to cease providing information or become less responsive, complicating the learning process. Normalizing the data helps maintain a balanced scale among the variables, ensuring that no particular feature dominates the others and allowing the network to effectively learn the underlying patterns

Neural network Model

Now, let’s start analyzing how the neural network could be implemented. I am using RMSE for the loss function. What I want to determine is the following:

- Number of hidden layers
- Number of neurons per layer
- Learning Algorithm
- Epochs

Since there are 24 inputs, and therefore, 24 neurons in the input layer, let’s start with a single hidden layer with 24 neurons and a sigmoid activation function. Starting with SGD (Gradient Descend), I am using a learning rate of 0.001 and 500 epochs:

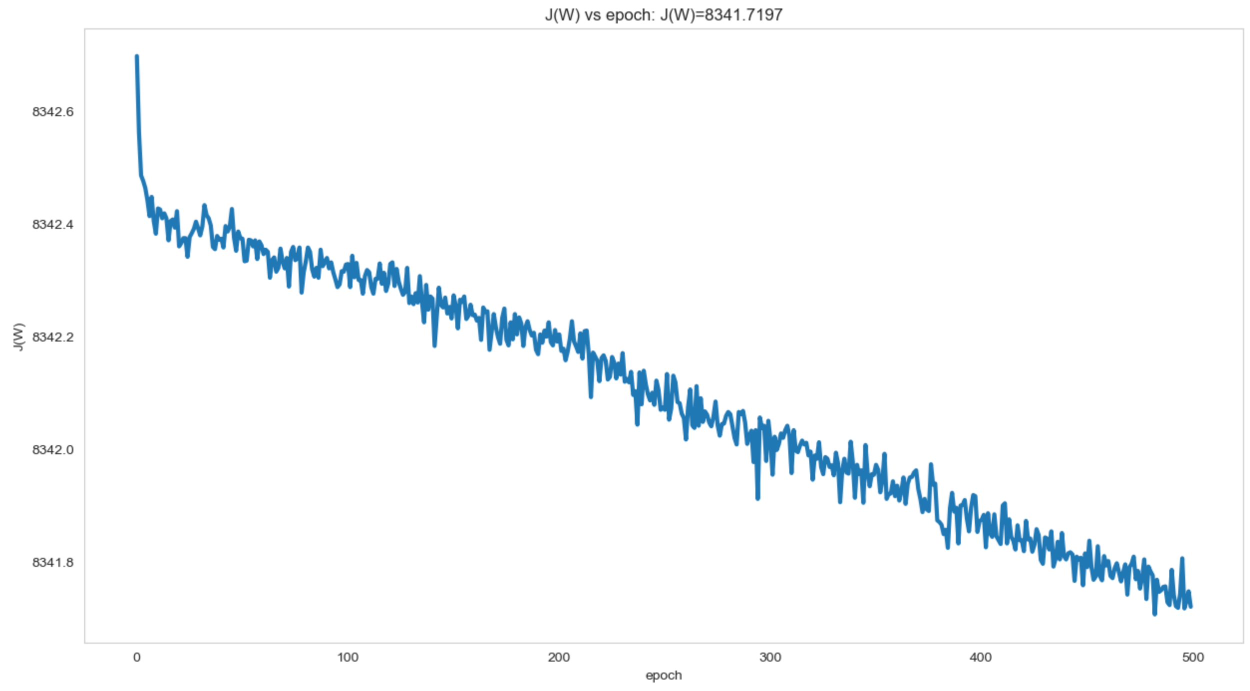

Figure 2 Learning curve

As seen the Figure 2, I could probably lower the learning rate to avoid the oscillation observed, but it seems to be converging to an RMSE greater than 8300, and it is giving an R2 score of 0.25 for training and 0.22 for testing.
If I add a momentum of 0.5, it seems to work a little bit better, but not much greater scores:

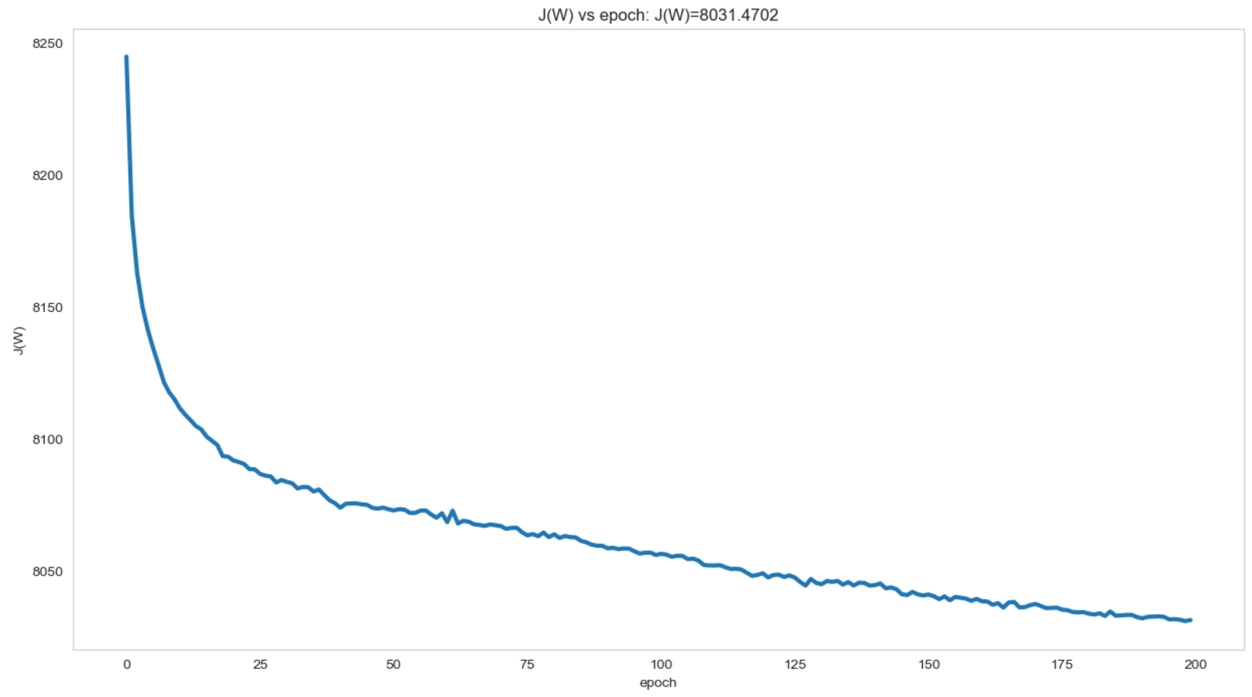

Figure 3 SGD with momentum

When adding a second layer of neurons, we start decrementing the RMSE:

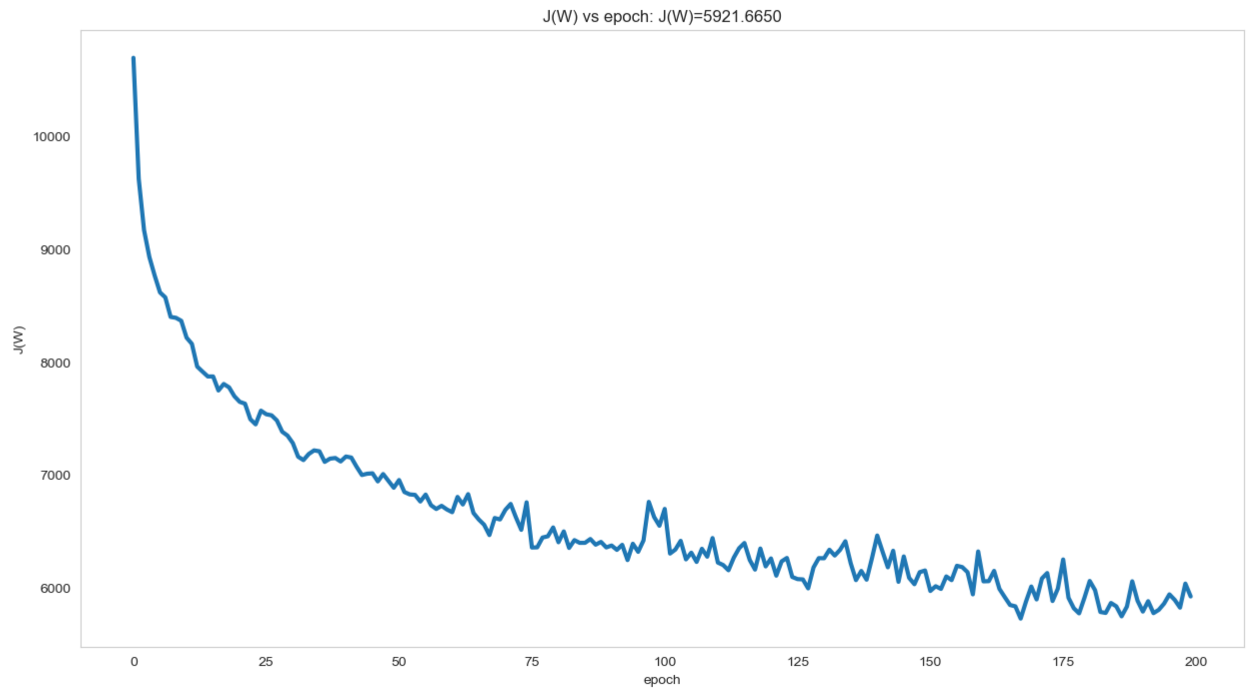

Figure 4 Learning with 2 layers of 24 neurons

I kept the momentum, but I added a second layer of 24 neurons. It now seems to be somewhere around 6000 for RMSE.
If I keep all configurations, but I change the algorithm to Adam, it improves, and RMSE goes below 4000:

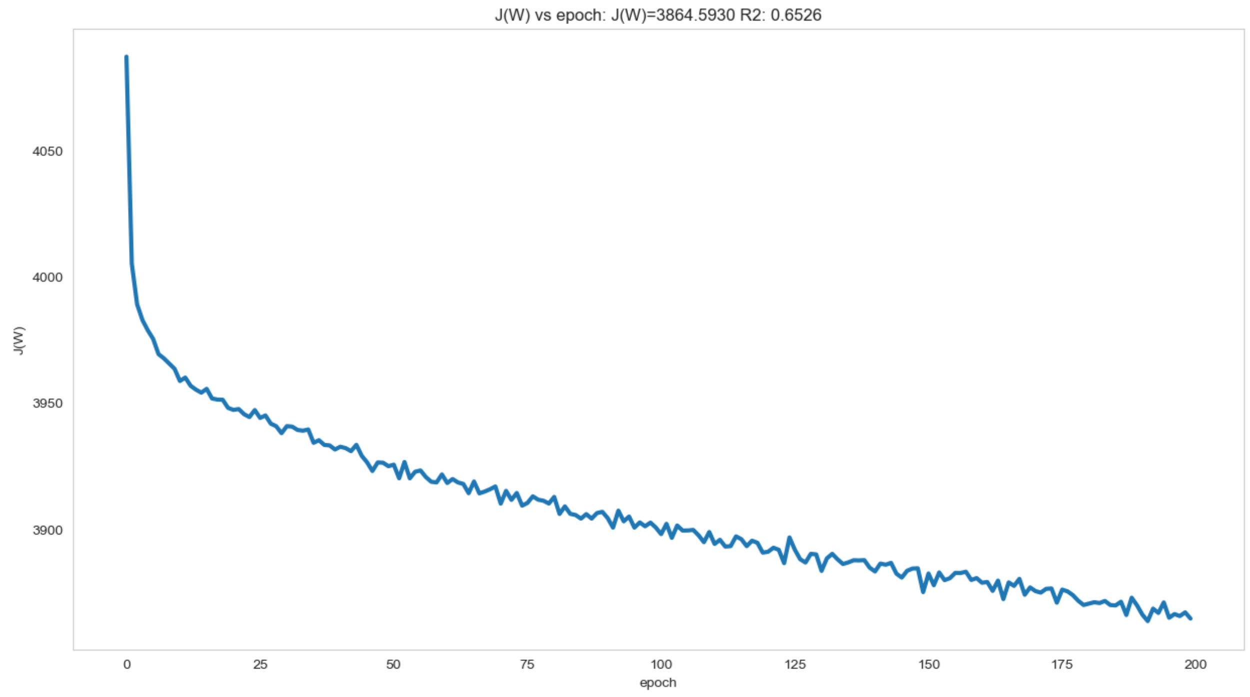

Figure 5 Learning curve with Adam

I am seeing interesting results here, with only 200 epochs, it goes to 3864 RMSE, and an R2 for the training of 0.6526, although R2 for testing is still only 0.32. With two layers of 24 neurons, it seems to be getting stuck around these values.

If I double the neurons per hidden layer, meaning 48 per layer, it improves significantly:

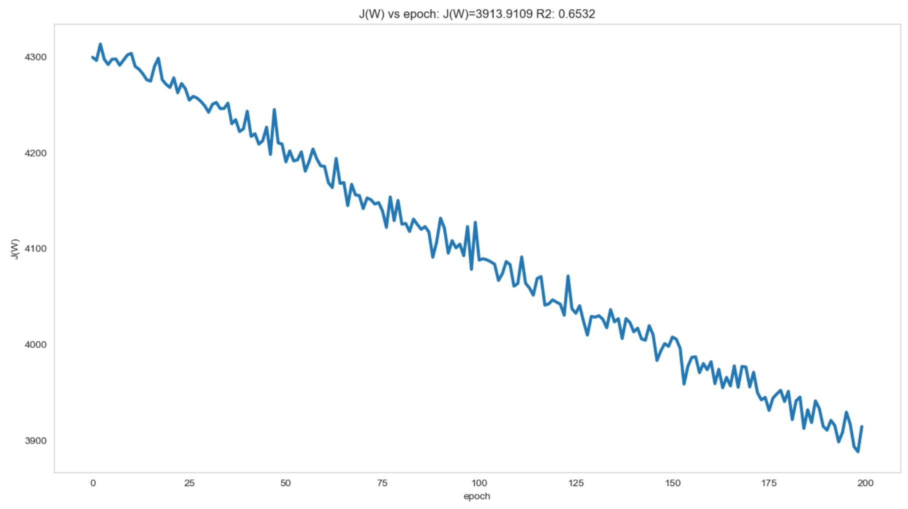

Figure 6 RMSE with 2 layers of 48 neurons

Even though the RMSE did not improve, the testing set now goes up to 0.4724 for R2 score, against the previous 0.32. It seems to be learning still, so I am leaving it to 1000 epochs:

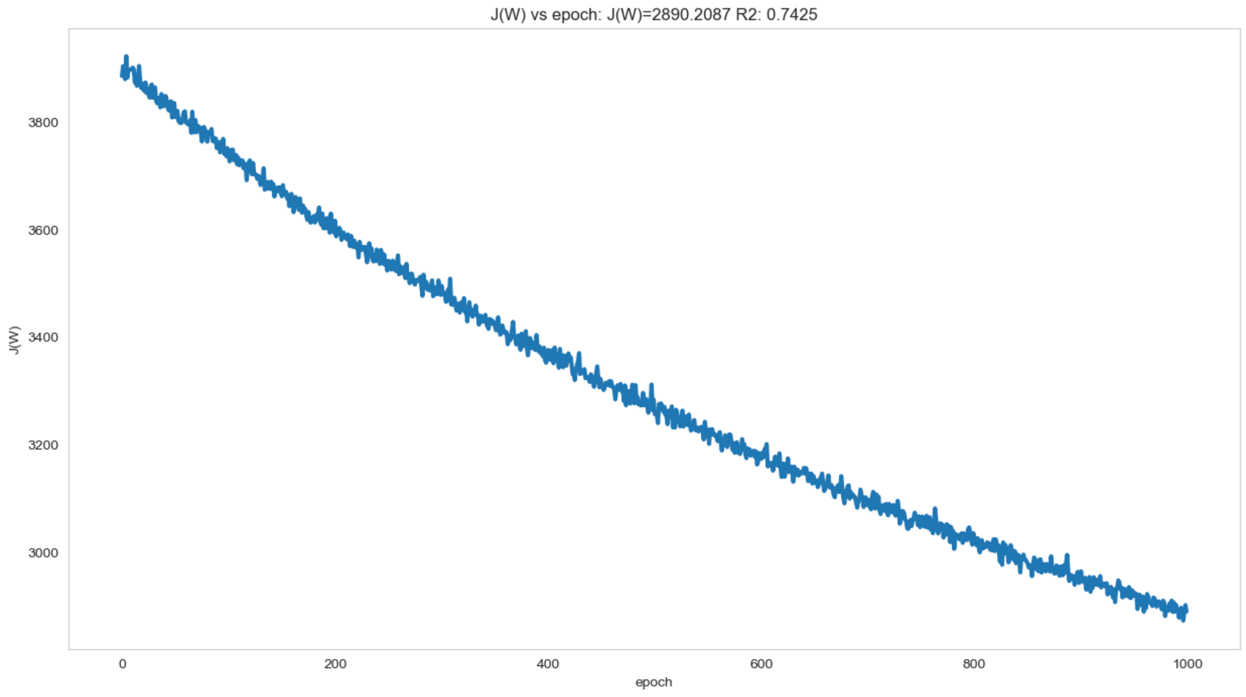

Figure 7 Experiment with 1000 epochs

We get to an RMSE of 2890, an R2 score of 0.74 for training, and 0.5053 for testing, it seems to be getting better with two hidden layers of 48 neurons. Since it is still going down, I'll just keep giving more epochs:

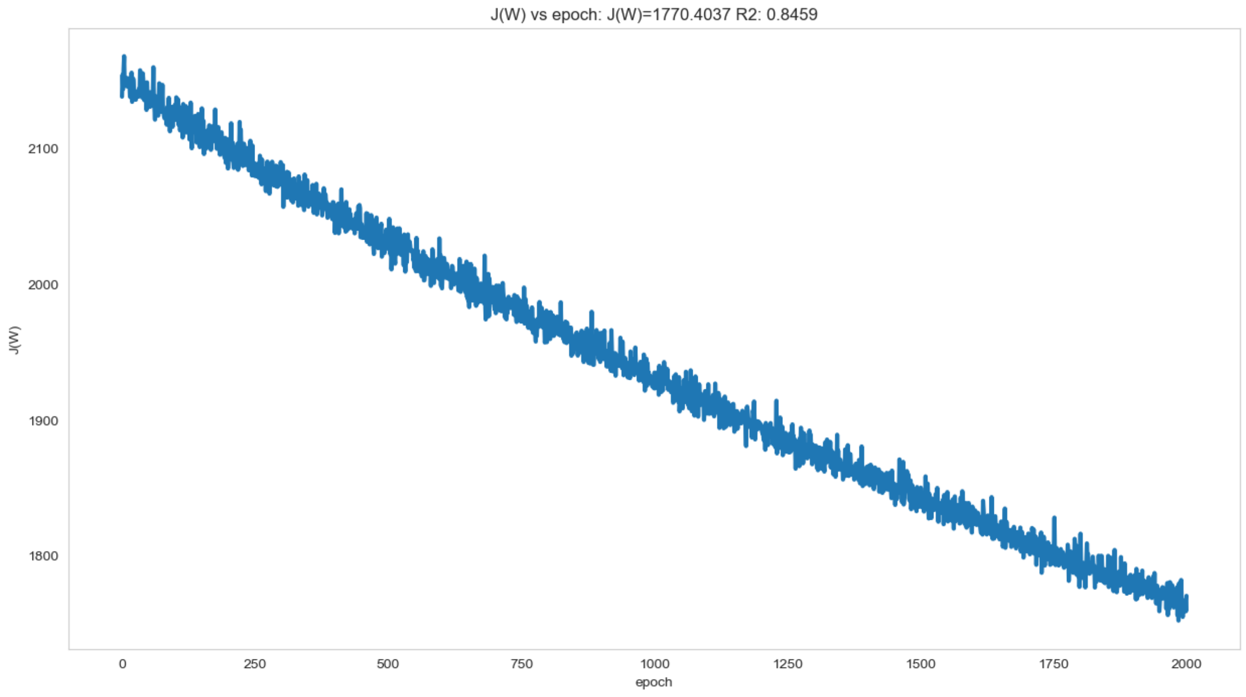

Figure 8 Using 2000 epochs

If I leave it with 2000 epochs, it goes down to an RMSE of 1770, and the R2 score for testing gets to 0.5205 which is better. I could try to decrease the learning rate, but it feels like it would need a lot more epochs to go down.
If I add a third layer of 48 neurons, then this happens:

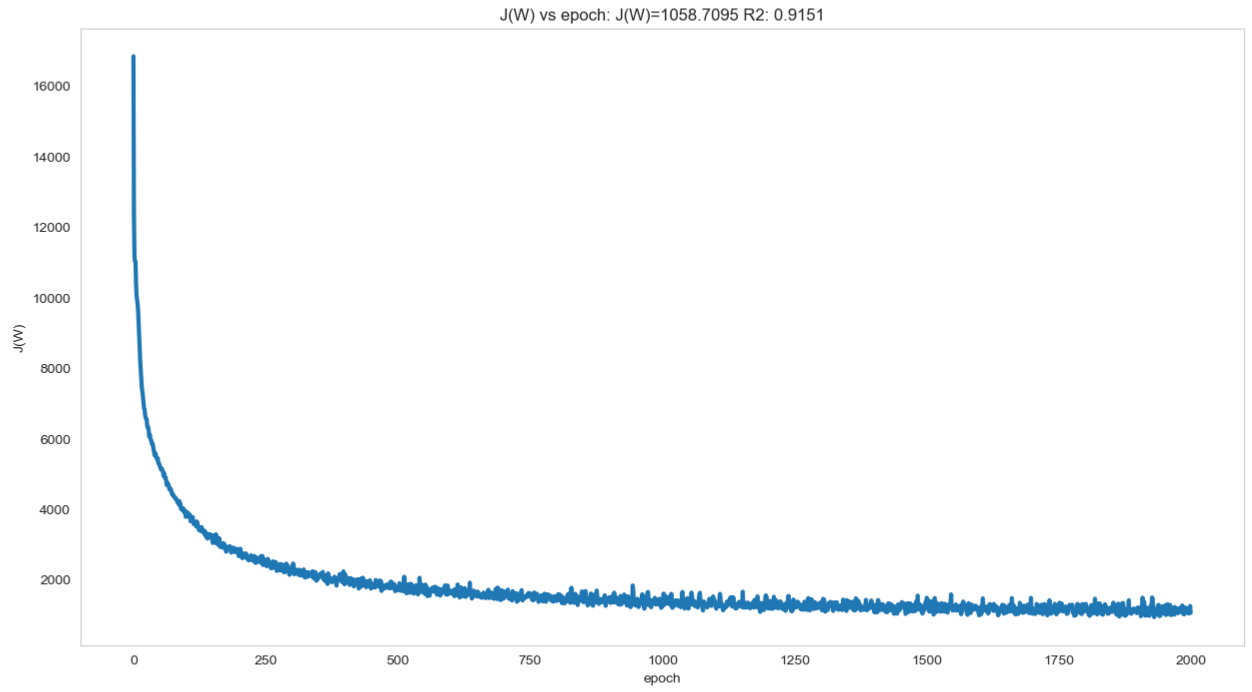

Figure 9 Using 3 hidden layers of 48 neurons

The RMSE in testing goes down to 1058, getting an amazing 0.9151 R2 score, unfortunately, for testing R2 score decreases to 0.4477, which could imply an overfitting of the training data.

I do not think a third layer will be helpful. I am keeping two layers and setting each hidden layer to 72 neurons. So, I am using 2 hidden layers of 72 neurons with a sigmoid activation, Adam method with a learning rate of 0.01 and 500 epochs.

I was able to get an R2 score of 0.92 for training and 0.52 for testing:

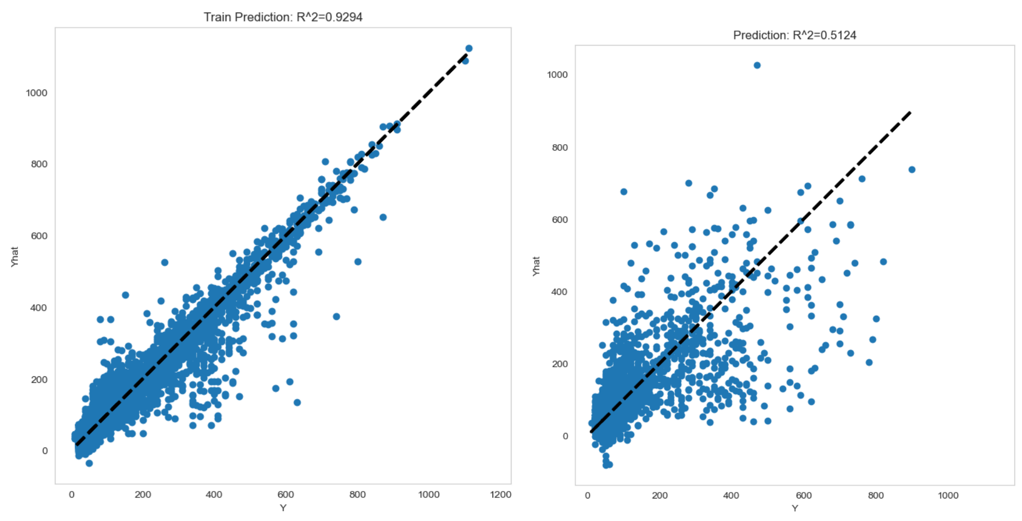

Figure 10 Train and test actual vs predicted

If we see however the loss function graph, we can see the learning rate might be too high:

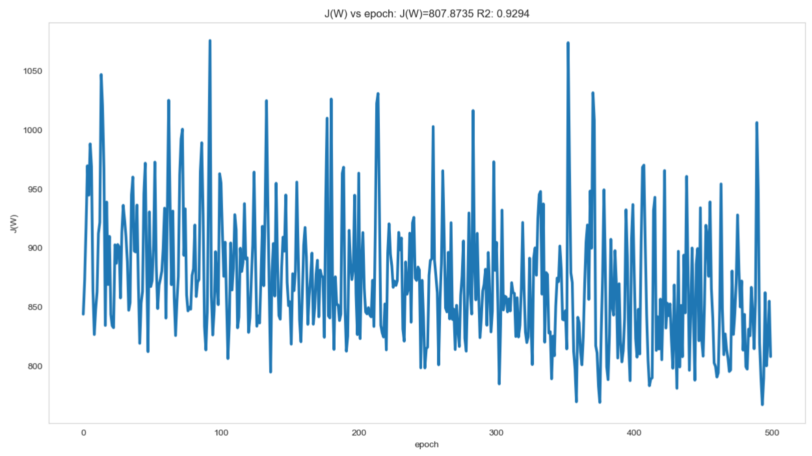

Figure 11 Loss function 2 hidden layers of 72 neurons

If I modify the learning rate to 0.0001, then we can see a much clearer convergence in the loss function:

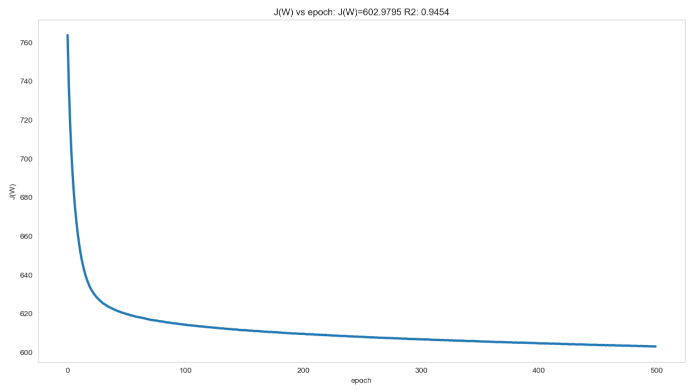

Figure 12 Loss function with a learning rate of 0.0001 for Adam

The R2 score for the train set goes up a little bit and the test seems to be almost the same, but it seems to be a more stable convergence:

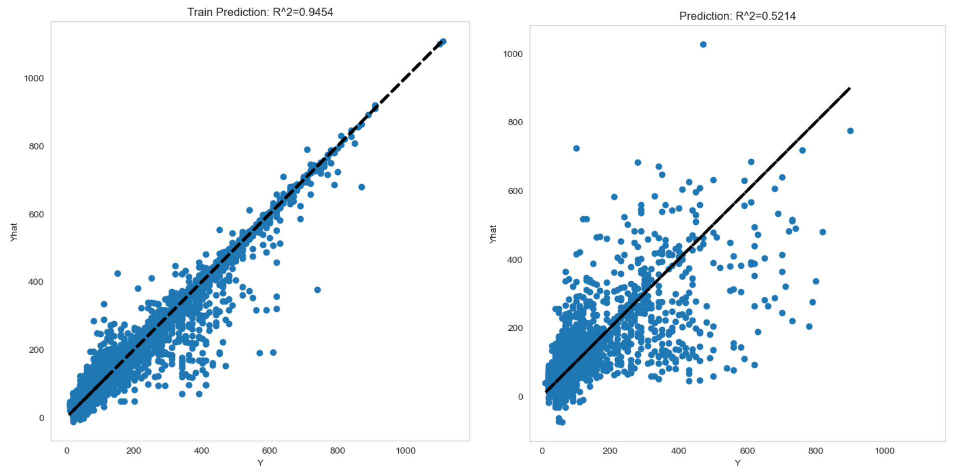

Figure 13 Predictions for train and test with lr=0.0001

This seems to be working fine since I want to avoid overfitting. I am using this architecture and I am putting it through gridsearchcv to determine the best hyperparameters.

Grid Search

So, for gridsearch I am keeping a specific configuration:

- 2 hidden layers of 72 neurons each.
- Adam algorithm.
- Linear output.
- 500 epochs.

To search I am using combinations of these parameters:

```
learning_rates = [0.0001, 0.00005, 0.001]
betas = [0.9, 0.99, 0.8]
activation = ["sigmoid", "tanh", "relu"]
```

After running the grid search, I found that these are the best-proposed parameters:
Best parameters found: {'activation': 'tanh', 'beta': 0.99, 'lr': 0.001}

This is interesting as I had only been testing with sigmoid activations, but tanh seems to be better fitted.

The R2 found by GS (gridsearch) in testing is 0.4477. I could not find a way to see the history to plot the loss function. Even though I know it won't be same, I will try again with these parameters and 1000 epochs to see if we can find something more interesting.

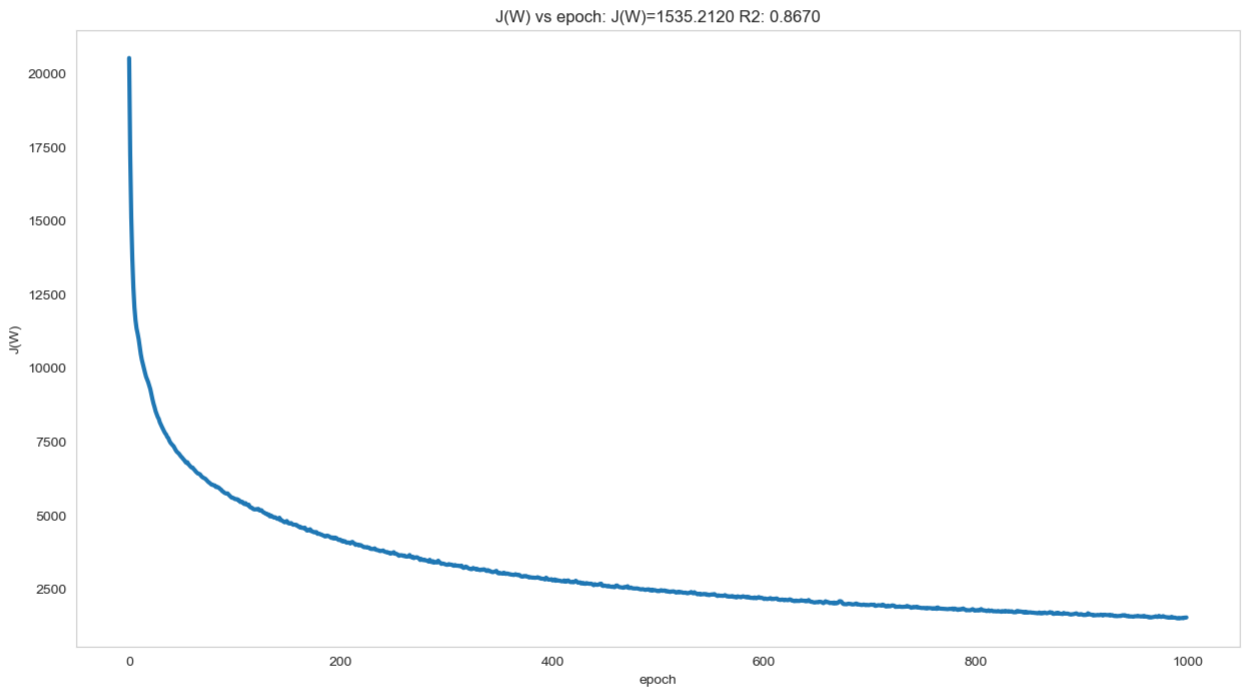

Figure 14 Loss function with parameters found in GS and 1000 epochs

The decrement looks fine. We can see that it starts to converge, and looks like it does not fit as well as the sigmoid for the train set. Let's compare with test:

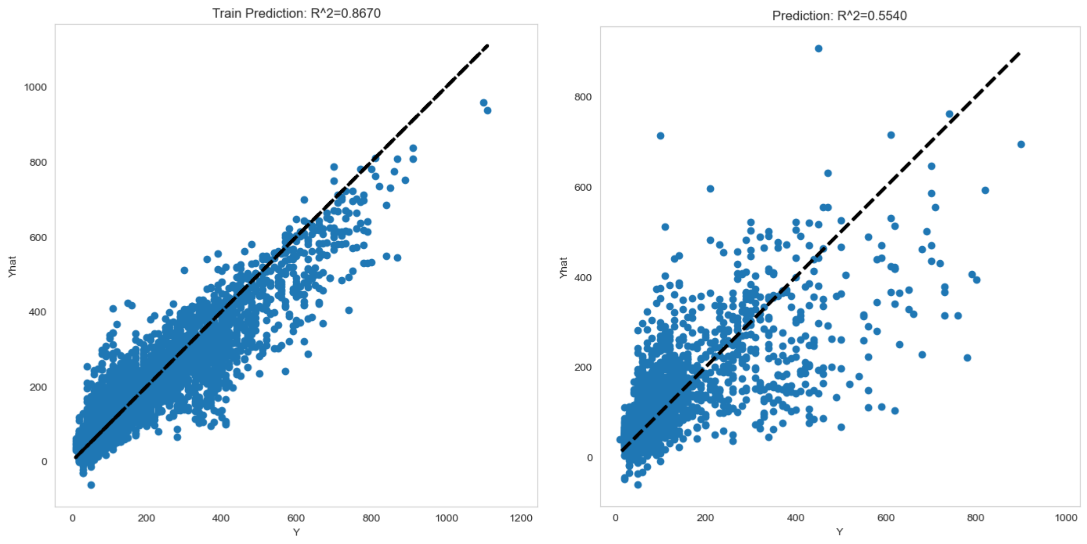

Figure 15 Testing vs Training with GS parameters and 1000 epochs

Even if the R2 score in training is not as high as it was when I used sigmoid activations, it performs better when I do it in the test. This may be an indication that using tanh is mitigating the overfitting, and is able to do a better generalization.

Conclusions

The final model proposed is:

- Input layer (24)
- Hidden layer 1 (72) - tanh
- Hidden layer 2 (72) – tanh
- Output layer (1) – Linear

Training with Adam lr=0.001, beta_1=0.99, and 1000 epochs.
Using "RMSE" as the loss function, and "R2" as an indicator for the best model.

Exploring hyperparameters in this activity was highly intriguing. The multitude of available hyperparameters presents a significant challenge in determining the most effective ones to utilize. Moreover, the non-deterministic nature of the training process adds an additional layer of complexity, as identical results may not be replicated. This variability calls for careful consideration and a balance between exploring the vast hyperparameter space and the quest for reproducible and robust model performance.

One interesting observation is the consistency exhibited by similar configurations during multiple training runs, even though they might not yield identical results each time. There's a tendency for these configurations to converge towards comparable outcomes unless a stroke of luck leads to the discovery of an exceptionally favorable local minimum.

I believe initial manual exploration is crucial. Engaging in a visual inspection of the neural network can effectively narrow down the search area. By identifying the set of parameters exhibiting better performance, we can then automate the search, exploring variations of these configurations to enhance results. Commencing with an exhaustive grid search from the beginning might be quite demanding, potentially leading to unnecessary waste of time and resources. By initially identifying and discarding poorly performing combinations, we can streamline the search process, focusing on promising configurations for more efficient hyperparameter tuning.

I think that tuning neural networks can be regarded as a heuristic process, lacking a deterministic approach. Nonetheless, following a structured methodology is crucial to discover optimal results.

While there's ample room for further exploration, the extent of investigation depends on the available time and the level of interest in the specific study. I believe that the obtained results within the dedicated time frame for this task (approximately a week and a half) offer intriguing insights into the model's performance