Build your first Neural Network to predict house prices with Keras

Mehul Patole(2017120046)
Vaishnavi Chavan(2018220065)
Shubham Parulekar(2017120044)

***Abstract*— Build your first Neural Network to predict house prices with Keras. Explore how to use a package called Keras to build our first neural network to predict if house prices are above or below median value.**

I. INTRODUCTION

In particular, we will go through the full Deep Learning pipeline, from:
Exploring and Processing the Data
Building and Training our Neural Network
Visualizing Loss and Accuracy Adding Regularization to our Neural Network Here we need Jupyter notebook set up with an environment that has the packages keras, tensorflow, pandas, scikit-learn and matplotlib installed.

II. INTRODUCTION TO NEURAL NETWORKS

Deep Learning is simply a subset of the architectures (or templates) that employs neural networks. "Neural networks" (more specifically, artificial neural networks) are loosely based on how our human brain works, and the basic unit of a neural network is a neuron. At the basic level, a neuron does two things: Receive input from other neurons and combine them together Perform some kind of transformation to give the neuron's output.

III. METHODOLOGY

Exploring and Processing the Data Before we code any ML algorithm, the first thing we need to do is to put our data in a format that the algorithm will want. In particular, we need to:

Read in the CSV (comma separated values) file and convert them to arrays. Arrays are a data format that our algorithm can process. Split our dataset into the input features (which we call x) and the label (which we call y). Scale the data (we call this normalization) so that the input features have similar orders of magnitude. Split our dataset into the training set, the validation set and the test set. If you need a refresher on why we need these three datasets. After reading the csv file Here, you can explore the data a little. We have our input features in the first ten columns:

Lot Area (in sq ft). Overall Quality (scale from 1 to 10). Overall Condition (scale from 1 to 10). Total Basement Area (in sq ft). Number of Full Bathrooms. Number of Half Bathrooms. Number of Bedrooms above ground. Total Number of Rooms above ground. Number of Fireplaces. Garage Area (in sq ft). In our last column, we have the feature that we would like to predict: Is the house price above the median or not? (1 for yes and 0 for no)Now that we have seen what our data looks like, we want to convert it into arrays for our machine to process: We now split our data set into input features (X) and the feature we wish to predict (Y). To do that split, we simply assign the first 10 columns of our array to a variable called X and the last column of our array to a variable called Y.Now we have split our data set into input features (X) and the label of what we want to predict (Y).The next step in our processing is to make sure that the scale of the input features are similar. Right now, features such as lot area are in the order of the thousands, a score for overall quality is ranged from 1 to 10, and the number of fireplaces tend to be 0, 1 or 2.This makes it difficult for the initialization of the neural network, which causes some practical problems. One way to scale the data is to use an existing package from scikit-learn.Then, we use a function called the min-max scaler, which scales the data set so that all the input features lie between 0 and 1 .Now,split our data set into a training set, a validation set and a test set. Now that we've got our architecture specified, we need to find the best numbers for it. Before we start our training, we have to configure the model by telling it which algorithm you want to use to do the optimization telling it what loss function to use telling it what other metrics you want to track apart from the loss function. We specify some of our settings (optimizer, loss function, metrics to track) with model.compile We train our model (find the best parameters for our architecture) with the training data with model.fit We evaluate our model on the test set with model.evaluate. We use matplotlib to visualize the training and validation loss / accuracy over time to see if there's over-fitting in our model.We used an architecture of neural network consisting of these layers:
Hidden layer 1: 32 neurons, ReLU activation.
Hidden layer 2: 32 neurons, ReLU activation.
Output Layer: 1 neuron, Sigmoid activation.

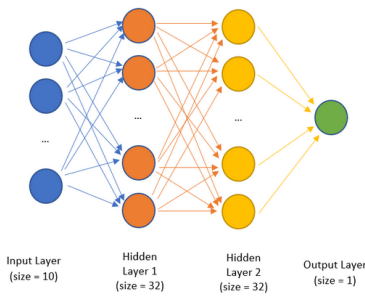

Figure1:Neural network architecture that we are using for our problem

IV. Analysis

Here, first we are visualizing the training loss and the validation loss.

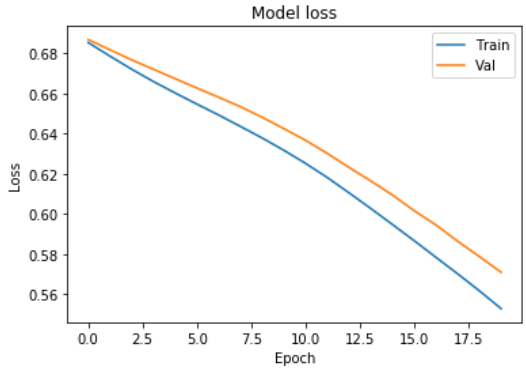

Figure1: A graph of model loss

We can do the same to plot our training accuracy and validation accuracy

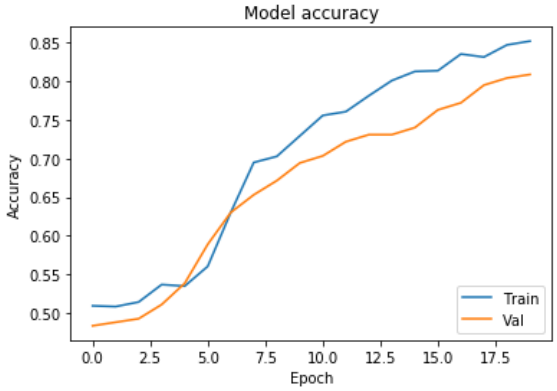

Figure1: Plot of model accuracy for training and validation set

Since the improvements in our model to the training set looks somewhat matched up with improvements to the validation set, it does not seem like over-fitting is a huge problem in our model. For the sake of introducing regularization to our neural network, we formulated a neural network that will badly over-fit on our training set as model 2.

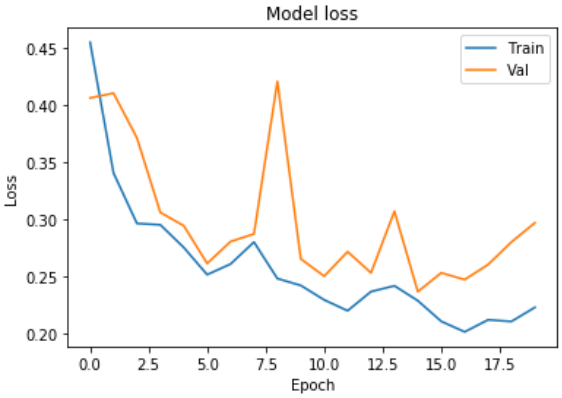

Figure1: Loss curves for over-fitting model

This is a clear sign of over-fitting. The training loss is decreasing, but the validation loss is way above the training loss and increasing. Below we can see plot of accuracy for over-fitting model.

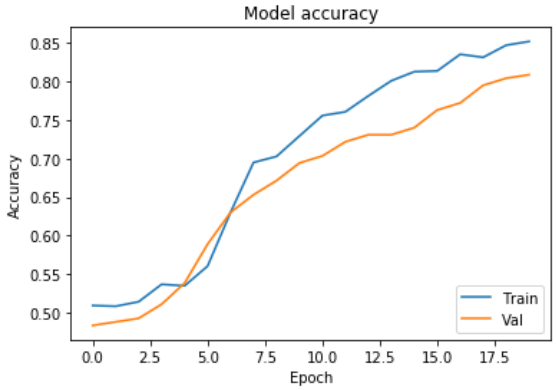

Figure1: Training and validation accuracy for our overfitting model

We can see a clearer divergence between train and validation accuracy as well. Then we tried out some of our strategies to reduce over-fitting (apart from changing our architecture back to our first model).We incorporated L2 regularization and dropout here. The reason we don't add early stopping here is because after we've used the first two strategies, the validation loss doesn't take the U-shape we see above and so early stopping will not be as effective. There are two main differences: Difference 1: To add L2 regularization, notice that we've added a bit of extra code in each of our dense layers.This tells Keras to include the squared values of those parameters in our overall loss function, and weight them by 0.01 in the loss function. Difference 2: To add Dropout, we added a new layer This means that the neurons in the previous layer has a probability of 0.3 in dropping out during training. We compiled it and run it with the same parameters as our Model 2 (the overfitting one).

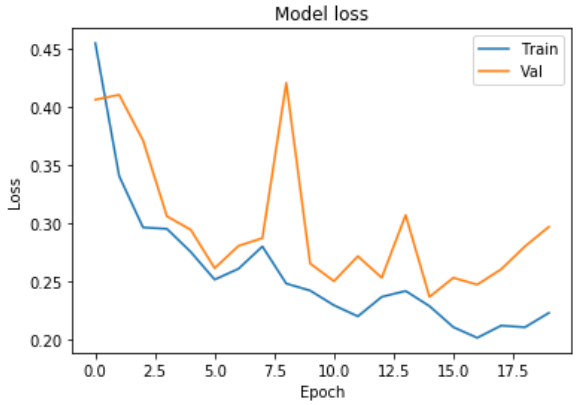

Figure1: Loss VS Epoch(Model 3)

Here we plotted the loss and accuracy graphs. The loss is a lot higher at the start, and that's because we've changed our loss function. To plot such that the window is zoomed in between 0 and 1.2 for the loss, we add an additional line of code (plt.ylim) when plotting.We can see that the validation

loss much more closely matches our training loss.

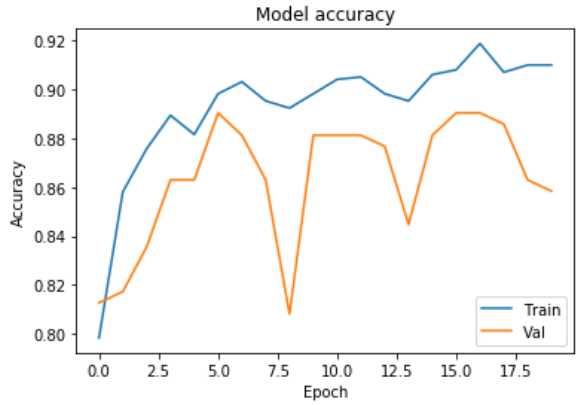

Figure1: Accuracy VS Epoch (Model 3)

Compared to our model in Model 2, we have reduced over-fitting substantially. And that's how we apply our regularization techniques to reduce over-fitting to the training set.

V. CONCLUSIONS

To deal with over-fitting, we can code in the following strategies into our model each with about one line of code:
L2 Regularization
Dropout
If we visualize the training / validation loss and accuracy, we can see that these additions have helped deal with over-fitting

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

REFERENCES

[1] https://hackernoon.com/build-your-first-neural-network-to-predict-house-prices-with-keras-3fb0839680f4
[2] https://www.investopedia.com/terms/n/neuralnetwork.asp
[3] Neural Networks and Deep Learning: A Textbook 1st ed. 2018 Edition
[4] https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/
[5] https://towardsdatascience.com/building-our-first-neural-network-in-keras-bdc8abbc17f5
[6] https://keras.io/getting-started/sequential-model-guide/same-stacked-lstm-model-rendered-stateful