cs577 Assignment 1

Jorge Gonzalez Lopez
A20474413
Department of Computer Science
Illinois Institute of Technology
February 9, 2021

Abstract

The report has to main sections: the first one formed by a list of 17 questions and the second one by a description of 4 different neural networks algorithms developed to solve the questions asked.

1. Questions:

1. When it comes to generating a Deep Learning a model, the most important thing is the data. However, to be able to return the actual performance of the model, it is necessary to split the data into three different data sets:
    a. Training data set: As its name implies, it is needed to train or fit the model and therefore, it corresponds to the data with which the model weights will be updated.
    b. Validation data set: It is used to check how the model generalizes to other 'unknown' data and to tune the hyperparameters of the model so that it performs better with data with which the model has not been trained on.
    c. Testing data set: It is used to provide the final evaluation of the model as it consists of data that has not been used until the hyperparameters are completely tunned and the final model trained. It gives an accurate evaluation of the model in case that it has fit the validation data in the process.

2. To successfully write a network program using Keras, 4 steps must be used [1]:
    a. Data: The input and output (target) of the model have to be identified and defined. Furthermore, many times, the data must be cleaned or preprocessed.
    b. Model: The second step is defining the neural network model. To do so, the Keras Sequential class allows to pile up different network layers to generate a model.
    c. Learning process: It gets done by running the command compile() on the model that has been just defined. The arguments that must be specified to configure the learning process are the optimizer (i.e., SGD, RMSprop, Adam…), the loss function (i.e., CategoricalCrossentropy, MeanSquaredError…) and the metrics (i.e., accuracy, MeanSquaredError…).
    d. Fitting: It gets done by running the command fit() on the model that has been just compiled. This command trains the model with the input and output data that is passed in the arguments as tensors. The number of epochs and the batch size are also parameters that have to be defined to set the training time and the amount of data with which the weights are going to be updated throughout every epoch.

3. The basic parameters of the Dense layer in Keras are [2]:
    a. Number of units: number of neurons that the layer will have.
    b. Activation: The activation function is the second operation that every neuron (unit) performs to generate its output value (sigmoid, relu, softmax…).

4. To compile the model in Keras, the basic arguments to configure the learning process are [3]:
    a. Optimizer: The way in which the maximum or minimum values of the cost function as well as the values of the variables in which those maxima and minima are found. Some examples would be Gradient Descent, RMSprop or Adam.
    b. Loss: It can be described as the function to compare the model's predictions with the actual output values in order for the training parameters to get more accurate with time.
    c. Metrics: It is another function used to evaluate the performance of the model. However, the difference with the Loss function is that the results of this function are not considered in the training process.

5. Likewise, to train the model, the basic arguments that should be always specified are the following [3]:
    a. Input: The training data set with which the model is intended to learn some rules to solve a specific task.
    b. Output: The corresponding target data of the input. It is needed so the loss function can be computed in supervised learning.
    c. Batch size: Number of samples with which the weights are going to be updated throughout a single epoch.
    d. Epochs: Number of cycles through all the samples of the training dataset (time spent training the model).
    e. Validation data: The validation data set to compute the loss and the metrics after every epoch.

6. First, it is necessary to create a vector with the same dimensions than the total number of possible words that would like to be considered (if the text strings are sentences) or the total number of letters in the vocabulary (if the text strings are words). Next, this vector is initialized with zeros and then, a value of '1' is given to the index of the corresponding word/letter if it is included in the sentence/word that is being analyzed. This method is called One-hot Encoding.

7. There are two main issues when it comes to training a Deep Learning Model. The first one would that both, the training error and the validation error, are very big. This means that the model suffers from high bias and therefore, it is underfitted (a bigger network or a longer time training are needed). The second problem would be that even though the training error is small, the validation one is much bigger. Therefore, the model suffers from high variance and overfitting, thus it does not generalize well to new data (some regularization or more training data are needed).

8. Some of possible hyper-parameters that can be tuned are:
    a. Layers: It correspond to the degree of depth of the model. More layers increase the size and complexity of the network; hence it can learn more complicated tasks.
    b. Units per Layer: The number of neurons in each layer. More neurons also imply a bigger and more complex network.
    c. Activation functions: There are many different activation functions and some of them perform better in specific tasks. For example, in a regression problem a linear function is recommended. Some of the most used functions are sigmoid, hyperbolic tangent, rectified linear unit or softmax.
    d. Loss: It is the actual way in which the network compares its predictions with the real output values. There are different types as Mean Squared Error (for a regression task) or Binary Cross entropy and Categorical Cross entropy (for binary classification and multiclass classification respectively)

9. The predicted values using a logistic function are numbers in the range 0 to 1. Therefore, by specifying a threshold, if the predicted value is bigger than the threshold, the class selected would be 1 and otherwise, the class chosen would be 0.

10. When facing a multi-class classification problem, there are several ways to get the target data. One of the most used ones is One-hot Encoding. This method generates a vector of the same length than the total number of classes with all values equal to zero and then, writes a one in the corresponding position of the actual label. For example, to get a model that classifies oranges, apples and bananas in pictures, the target value of an apple picture would be [0,1,0] with One-hot Encoding.

11. The softmax function is a generalization of the logistic function for multi-class classification tasks, so that it generates a vector of probabilities denoting how likely it is that a sample would be each of the classes. For example, a possible output in the example in question 10 would be: [0.2, 0.7, 0.1].

12. The only difference between sparse categorical cross entropy and categorical cross entropy would be the type of output data that is going to be introduced to the network. For the categorical cross entropy, one-hot encoded vectors must be used and for the sparse one, the target data would be just integers in the range 1 to the total number of classes [4].

13. If there are 5 equiprobable classes, a random classifier would have an accuracy of:
$$p = 100\ \frac{1}{5} = 20\ \%$$

14. The mean and the standard deviation of each feature has to be calculated. Then, the mean has to be subtracted to each feature value and then, divide the result by the standard deviation. This way, all features will be in the same range (zero mean and a normalized distribution). The purpose of the normalization is so that features with low values have the same significance for the prediction as features with very high values.

15. Both, the MSE and MAE, give an average of the errors however, since with MSE the errors are squared before they are averaged, it gives a greater significance to larger

errors. Therefore, to consider the deviated data MSE should be used and if it wants to be ignored, MAE should. MAE is easier to interpret as it is a linear function and not a quadratic one [5].

16. K-fold cross validation is used to evaluate the performance of a model when there are not many data available. To do so, all the data is shuffle and divided into k groups. Then, the model gets trained k times (trained with k-1 groups and validated with the remaining one). K-Fold is needed when there is not much data to train, validate and test the model.

17. When using K-Fold cross validation, the validation error corresponds to the mean of the k validation scores and the final model is trained with all the data available.

2. Programming

1. Tensorflow playground:

Problem statement:

Different network architectures have to be configured to try to classify correctly all the train and validation data sets of the Tensorflow playground.

Implementation details:

a. Circle: The network used has two input features (x_1 and x_2) and two hidden layers (the first one with 3 neurons and the second with 1 neuron). The hyper-parameters are:

i. Learning rate: 0.1
ii. Activation: ReLu
iii. Regularization: None
iv. Batch size: 10
v. Number of epochs: 347

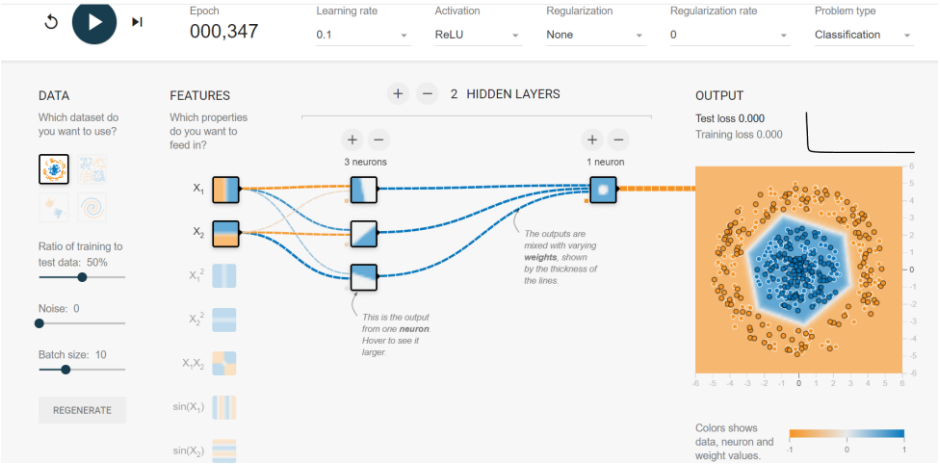

Figure 1.1: Classification task with circle dataset.

b. Exclusive OR: In this case, just by introducing a new feature generated as $x_1 * x_2$, a very simple network with a one neuron hidden layer classifies correctly all the data. The hyper-parameters are set as:

i. Learning rate: 1
ii. Activation: ReLu
iii. Regularization: None
iv. Batch size: 10
v. Number of epochs: 39

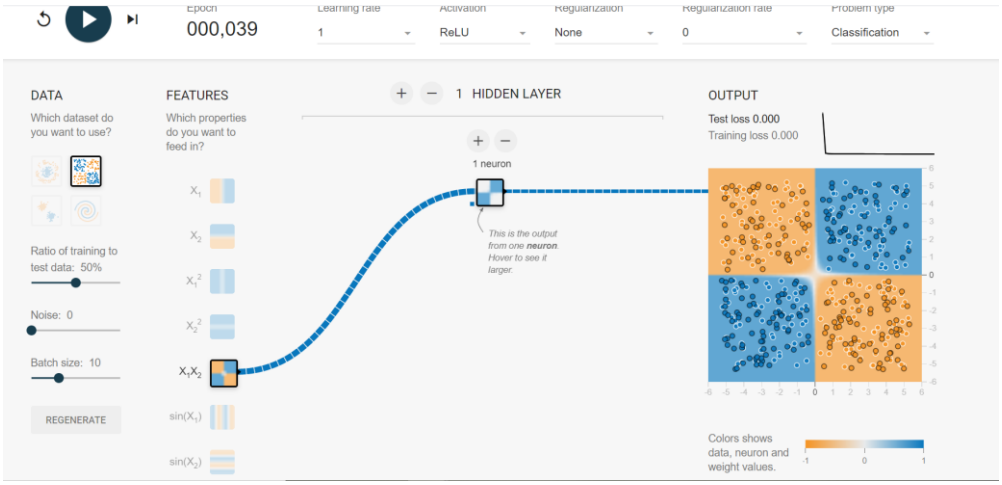

Figure 1.2: Classification task with XOR dataset.

c. Gaussian: In this case, just by introducing the two features x_1 and x_2, a very simple network with a one neuron hidden layer classifies correctly all the data. The hyper-parameters are set as:

i. Learning rate: 3
ii. Activation: ReLu
iii. Regularization: None
iv. Batch size: 10
v. Number of epochs: 24

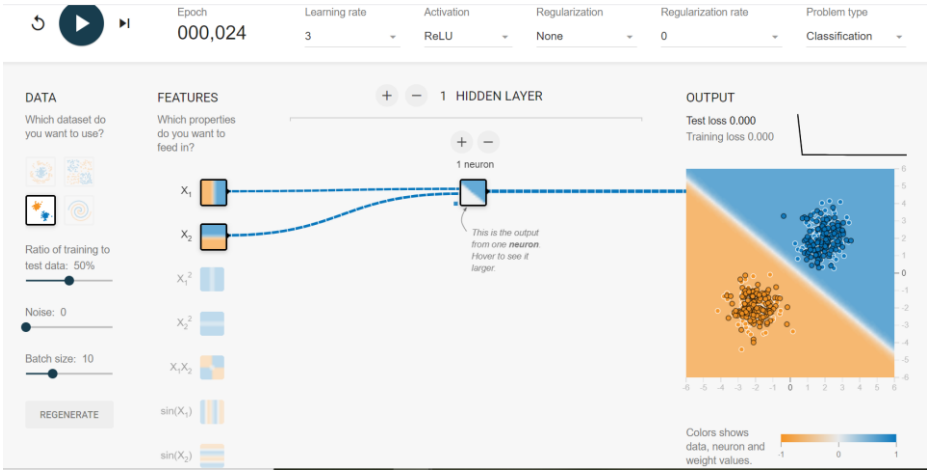

Figure 1.3: Classification task with Gaussian dataset.

d. Spiral: In this case, the data set is harder to classify. A network with 2 hidden layers of 6 neurons each has been created, introducing all seven features. The other hyper-parameters used are:

i. Learning rate: 0.1
ii. Activation: ReLu
iii. Regularization: L2
iv. Regularization rate: 0.001
v. Batch size: 10
vi. Number of epochs: 1040

It could not classify all the training and testing samples correctly, having a 0.012 test loss and a 0.004 training loss.

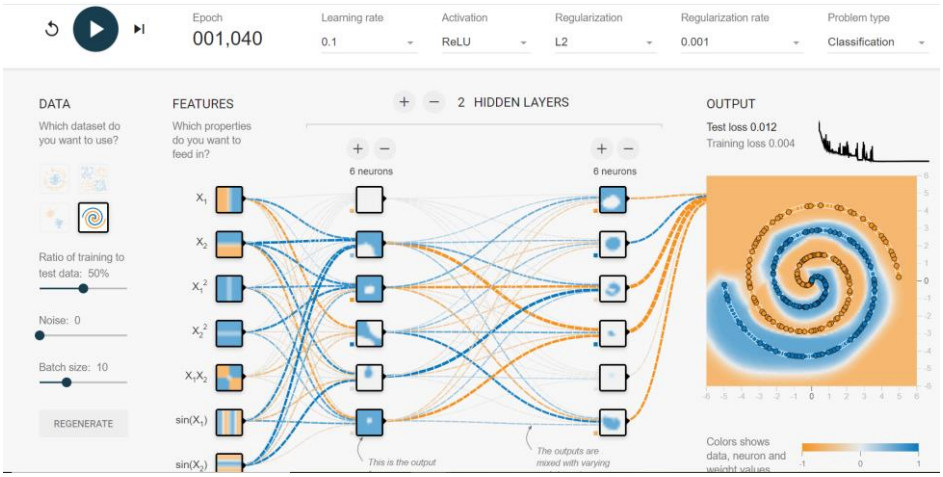

Figure 4: Classification task with Spiral dataset.

2.

Problem statement:

A model to classify images with 3 different classes from the CIFAR10 dataset.

Proposed Solution:

The data is formed by 10 different classes, so the images of the first 3 are the only ones that are going to be considered.

Two numpy arrays are created to train the model, the images of the 3 classes (X) and their labels (Y). The labels are transformed to a vector of zeros and ones with One-hot Encoding and the X and Y arrays are split up into training and validation data with a 70:30 ratio, respectively.

Finally, a Neural Network is designed with the following design:

- First a rescaling layer to turn input pixel values to the range 0 to 1.
- 8 hidden layers with 128 units each and the ReLU activation function.
- Output Layer with 3 units (number of classes) and Softmax activation function (as One-Hot Encoding is being used).

Then, an Adam optimizer is going to be used, with a learning rate of 1e-2, and a categorical loss entropy and accuracy for the loss function and metric function, respectively. Finally, an initial batch size of 512 samples and 50 epochs are going to be considered for the training.

Implementation details:

The model's validation error was much greater than the training one and sometimes it also diverged. Therefore, a reduction of the learning rate has been applied (1e-4) with a subsequent incrementation of the time training (250 epochs). At the same time, to reduce the overfitting, the hidden layers have been cut out to 4 with 64 neurons (a simpler model is less likely to overfit the data). Finally, the batch size was increased to 1024 samples to have less variations in the final plots, see Figure 2.1.

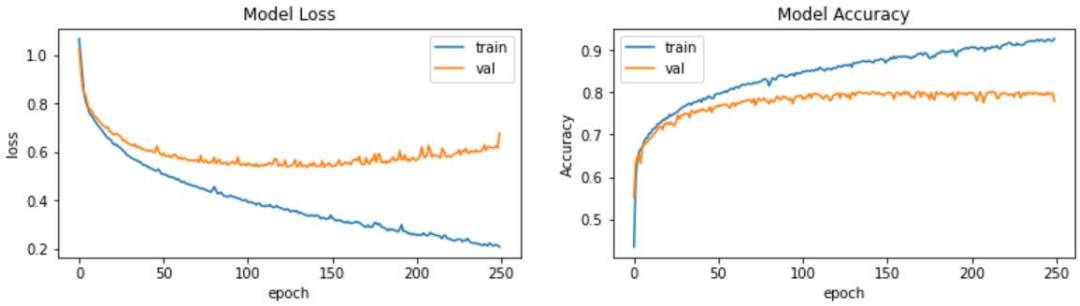

Figure 2.1: Model results with 4 hidden layers of 64 units.

This time, the validation loss gets worse after the 125 epochs approximately, thus a little overfit still remains. Therefore, a L2 regularization in all the layers has been applied (increasing the regularization rate until the validation loss does not increase with time).

It is important to note that an additional option has been implemented to be able to train or load the model depending on a Boolean flag call *Train*.

Results and discussion

The final model results are shown in Figure 2.2. The validation results improve to a certain degree, in which they get stacked (even though the training results keep improving).

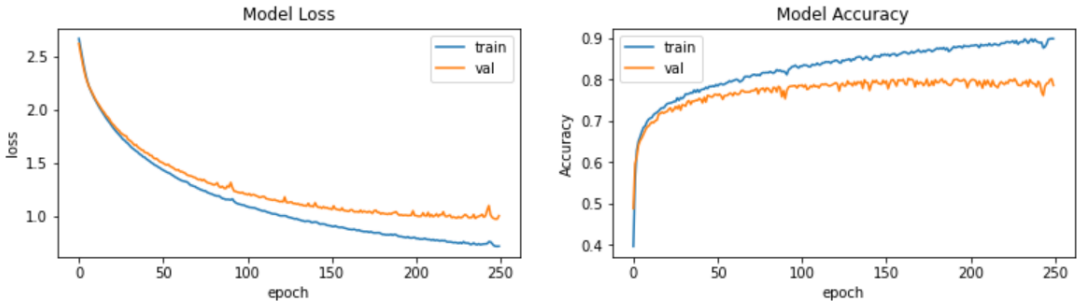

Figure 2.2: Final model results.

With this model, the results over the test_batch are the following:

loss: 0.9794 - accuracy: 0.8033

3.

Problem statement:

Build a model to do a classification of emails as spam or not spam.

Proposed Solution:

The data comes as a text file, so the numpy function loadtext is used.
Once loaded, it is split into the input data or X (all but last columns) and target data or Y (last column), which is reshaped to get sure it maintains the right dimensions.
Then, the columns of the input data are normalized by subtracting their mean values and dividing by their standard deviation. This way, all columns have a zero mean and a similar range of values.
Next step is to shuffle the data to get a balance input regarding the spam and not spam emails when dividing the dataset into the train and validation sets.
Finally, the first 70 % of the data comprise the training set and the remaining 30 % the validation set.

Finally, a Neural Network is designed with the following design:

- 4 hidden layers with 64 units each and the ReLU activation function.
- Output Layer with 1 unit and sigmoid activation function (binary classification).

Then, an Adam optimizer is going to be used, with a learning rate of 1e-2, and a binary cross entropy and accuracy for the loss function and metric function, respectively. Finally, an initial batch size of 1024 samples and 50 epochs are going to be considered for the training.

Implementation details:

As in last model, the validation error was much greater than the training one. Therefore, to reduce the overfitting, several network architectures have been tried until the complexity has been reduced to a simple 2 hidden dense layers of 4 neurons each. At the same time, as there are not many training data, the batch size has been increased to 2048 samples, thus the training results improve in a smoother way.

It is important to note that an additional option has been implemented to be able to train or load the model depending on a Boolean flag call *Train*.

Results and discussion:

The final model results are shown in Figure 3.1. After the 100 epochs, the validation loss and accuracy do not improve any more, but the results are pretty similar to the training ones.

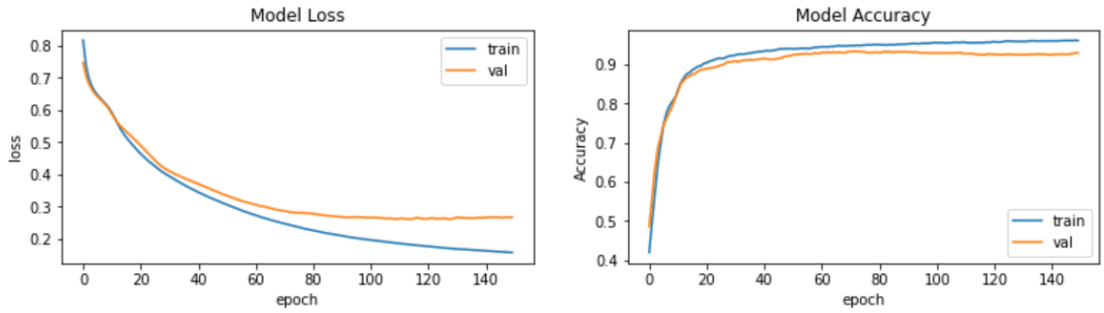

Figure 3.1: Final model results.

And the numeric results for the validation set are:

loss: 0.2942 - accuracy: 0.9219

4.

Problem statement:

Build a regression model to predict the crime rate in a certain community.

Proposed Solution:

The data comes as a text file, but with missing values marked as '?', thus a different approach has been taken.
First, the file containing the attributes names is read to obtain the names of the columns, then the pandas read_csv function is used specifying the columns names just obtained. Then, the data is split up in inputs (all columns but Violent Crimes per Population) and target (Violent Crimes per Population column).
Now, with the pandas' functions, the missing values of the inputs are replaced by NaN values so that the columns containing them can be easily removed from the dataset and, at the same time, the non-predictive columns (as the community name) are also removed.
Finally, though the data is already normalized, it is randomly shuffle with a seed to always get the same results and the training and validation datasets are generated with a samples ratio of 70-30.

Finally, a function which generates and train a Neural Network is developed with the following design:

- 2 hidden layers with 64 units each and the ReLU activation function.
- Output Layer with 1 unit and no activation function (Regression model).

A RMSprop optimizer is going to be used, with a learning rate of 1e-3, and a mean squared error and mean absolute error for the loss function and metric function, respectively. Finally, an initial batch size of 128 samples and 150 epochs are going to be considered for the training.

Implementation details:

As there are not many data within this dataset, a K-Fold validation has been considered. To do so, the function k-Fold from sklearn has been used with k =

5. For each iteration, the data is divided into 5 sets, using 4 of them for training and the remaining one for validating the model. Then, the validation MAE is saved so that the mean MAE over all the k models is computed and plotted out.

As in last model, to avoid overfitting and to smooth out the plots, L2 regularization and a batch size of 512 has been chosen obtaining the mean MAE in Figure 4.2.

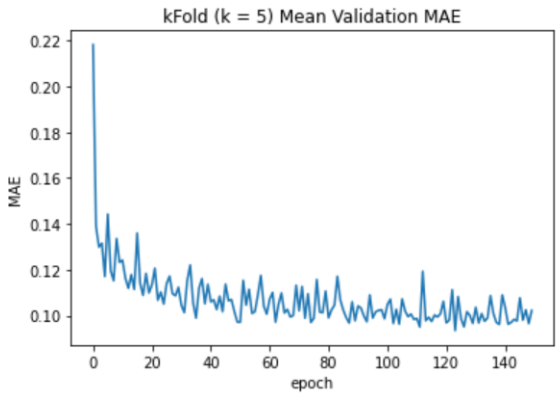

Figure 4.2: Mean MAE over all the k-Fold models.

It is important to note that an additional option has been implemented to be able to train or load the model depending on a Boolean flag call *Train*.

Results and discussion

Finally, once the validation results have been obtained, the model is trained with all the training data available. The final results for this model for the training and testing data can be seen in Figure 4.1.

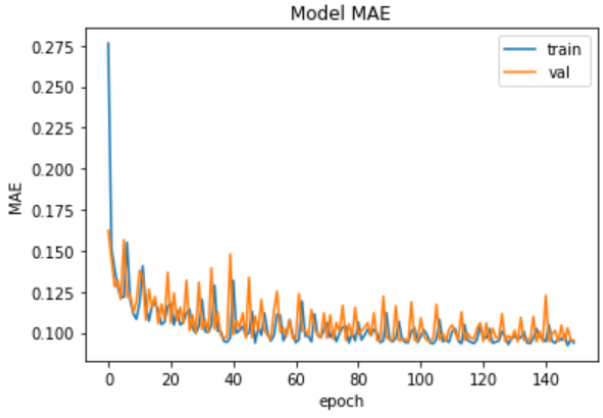

Figure 4.1: Final model results.

And the numeric results for the test data set are:

loss: 0.0282 - mae: 0.0952

3. References

[1] The Keras 4 Step Workflow. (n.d.). Retrieved February 15, 2021, from https://www.kdnuggets.com/2018/06/keras-4-step-workflow.html

[2] Team, K. (n.d.). Keras documentation: Dense layer. Retrieved February 15, 2021, from https://keras.io/api/layers/core_layers/dense/

[3] Team, K. (n.d.). Keras documentation: Model TRAINING APIs. Retrieved February 15, 2021, from https://keras.io/api/models/model_training_apis/

[4] Team, K. (n.d.). Keras documentation: Losses. Retrieved February 15, 2021, from https://keras.io/api/losses/

[5] Jj. (2016, March 23). MAE and RMSE - which metric is better? Retrieved February 15, 2021, from https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d