cs577 Assignment 1

Girish Rajani-Bathija
A20503736
Department of Computer Science
Illinois Institute of Technology

September 20, 2022

1. Explain the difference between training, validation, and testing data sets. Explain the need for such datasets.

The original dataset is normally broken down into training, validation, and testing data sets where each dataset plays a key role in creating the model for prediction.

The training dataset is, in most cases, the largest dataset out of the three simply because this is the data that is being used by the model to learn from or to train the model. It is important to feed the model with enough training data and the right type of data so that the predictions can be as accurate as possible. The training data allows the algorithm to identify patterns and decisions so that it can do the same with real-world data. By increasing the training data, we allow the model to get better and better since it will be able to see patterns and learn more efficiently. However, it is important that the training data is relevant to what we want the model to do because if we just keep feeding the model more and more data without taking into account the relevancy, then we can run into overfitting.

The validation dataset helps with evaluating various models and finding the values for the hyperparameters which will result in the best performance. It is used for optimization, and by adjusting the hyperparameters of the validation dataset, we can increase the accuracy of the model. The validation set consists of a small portion of the training dataset.

On the other hand, the testing dataset consists of new unseen data that is used to test the final model after it has been trained. It is important to test the accuracy of both the training dataset and the testing dataset where the goal should be to try and have these two accuracies be relatively similar. If, on one hand, the training dataset is not performing well, then more relevant data is needed to continue training the model. However, if the training dataset is significantly more accurate than the testing dataset then it is possible that overfitting has occurred.

The explanations for the validation dataset and the testing dataset may seem somewhat similar and so one may ask, what is the main difference between these two datasets and if the validation dataset can be omitted? When there is only one model being used then it is possible to omit the validation set, but if there are multiple models to select from, then the validation dataset plays an important role because it can be used to provide an early estimate on how the various models are performing without testing it with unseen data. This can help with optimizing and tuning the hyperparameters before the final testing is done. The validation dataset is used during the training phase to get a preliminary accuracy, and then only once the model has been fully trained and optimized, is it tested on unseen data to assess the final performance.

The need for such datasets exists because if you don't use these datasets then your model will be unreliable, resulting in biased performance and full of inaccuracies when used with real-world data. The end goal is to create a neural network that can adapt well to unseen data and make accurate predictions and in order to do so, it is crucial that the network is trained, validated, and tested.

2. Explain the 4 steps used in writing a network program using Keras (data, model, learning process, fitting).

   According to Francois Chollet in "Deep Learning with Python", the 4 steps used in writing a network program using Keras are:
   - **Define your training data** - This includes defining your input and target tensors. This is where the amount of data that will be used for the training dataset is specified.
   - **Define the model** - There are two ways in which a model can be defined; using the Sequential class or the functional API. With the Sequential class, the first step is to build the instance of the class, after which, various layers can be created as needed with activations such as relu, sigmoid, softmax, etc. When using the functional API, the input and output tensors are modified so that the model applies various layers as though they are functions.
   - **Configure the learning process** - When using the Sequential class, configuring the learning process is done by using the compile() method, which specifies three parameters: the optimizer, the loss function (example 'categorical_crossentropy'), and the metrics that will be monitored when training the data (eg: 'accuracy').
   - **Train the model using fitting** - The fit() method is used to train the model by passing the data to the model. This is done by specifying the input and target tensors along with the batch size and epochs through the fit() method. The batch size and the number of epochs allow the model to iterate through the training set multiple times as specified by the mentioned parameters.

3. Explain the basic parameters used to define a dense layer (number of units, and activation).

When defining a dense layer, the number of units and the activation is specified. Within the first layer, the input size, the number of units that specify the output size, and activation are specified but in the next layer, the input size is not specified because it will automatically take the input size that was mentioned in the first layer. The number of units must always be a positive integer number because it is the size of the output vector, which cannot be negative.

The activation is a function that transforms the input into output within the neural network. This allows the neural network to learn from the errors of the previous epoch and make adjustments necessary to reduce errors in future epochs. Some popular activation functions are relu, sigmoid, softmax, etc.

4. Explain the network configuration aspects when compiling the model (optimizer, loss, metrics). Explain the difference between loss and metrics.

The final step in creating a model is compiling. When compiling the model, there are three network configuration aspects which are optimizer, loss, and metrics. The optimizer allows us to tune the model by changing the parameters, which can help reduce the loss and make the predictions more accurate. The model weights, learning rate, etc are modified through the optimizer to achieve the goal of making accurate predictions.

The loss function is used to find errors and helps guide the optimizer. It lets the optimizer know whether it is moving in the right or wrong direction so that it can make modifications accordingly to improve results. This is done by taking the prediction from the model and comparing it with the expected output, then computing a score to show how well the network has done in predicting the expected output.

The metrics are used for monitoring during training and testing. A very common metric is 'accuracy' whereby the metric monitors the accuracy during training and testing.

If metrics are monitoring accuracy then one may ask, what is the difference between loss and metrics? To most people, loss and accuracy may seem very similar but are, in fact, different. In this case, the loss function is used to optimize the model. The value from the loss function is used by the optimizer to make modifications to increase the accuracy of the predictions. On the other hand, the metric is used to judge the model at the end. In most cases, when specified, the metric will return the accuracy and this is for us to look at and get an understanding of the model. The metric does not play any part in terms of optimization. For example, the metric may return accuracy = 98%. This is something that we can understand and conclude that the model has been trained well and is making accurate predictions. On the other hand, the loss function may return something like 0.5 and to us, we may not know what that loss is directly referring to but the optimizer is able to understand this and make adjustments to reduce this loss.

5. Explain the 5 basic arguments provided to fit (input, output, batch size, epochs, validation data).

The fit() function is used when training the model and allows the model to be evaluated during training. The input argument specifies the input training data that will be used to train the model. The output argument is simply the target data (desired output). The batch size is the number of samples that will be drawn to train the model during one iteration. The advantage of splitting the training data into smaller batch sizes is that it will use less memory since smaller batches are being used and it will also train faster since, after every iteration, the parameters can be updated. The disadvantage of using smaller batches instead of training all samples is that if the batch size is too small then this can affect the accuracy of the model.

Epochs specify the number of iterations allowed when training the data. After each epoch, the neural network will make adjustments to reduce the loss. Typically, with every epoch, the training loss decreases, and the training accuracy increases because of optimization. In some cases, when the number of epochs is large then, this can cause what is known as overfitting, where the model has learned too much specific to the training data and does not perform well on unseen data. This can be analyzed when plotting a training and validation accuracy graph, as shown below.

Validation data is the final basic argument within the fit() function. The validation data helps to monitor the loss and accuracy of the samples while they are being trained. After every epoch, the model will calculate the loss and accuracy of the validation data. This helps to plot the validation loss and accuracy, which can be used to determine if overfitting has occurred.

6. Explain the steps used to convert a variable length text string into a binary feature vector.

When feeding data into a model, the data has to be in the form of tensors. One form of tensors is a binary feature vector. This is a vector consisting of 0's and 1's. The process of converting a variable length text string into this binary feature vector starts by converting the text string into integers through a dictionary which is then converted into a binary feature vector.

The next step is to convert these indices into a binary feature vector. This can be done using one hot encoding. One hot encoding allows the encoding of the list of indices into vectors of 0s and 1s. This allows the algorithm to make better predictions, especially when the integers do not have a specific ranking. The way in which the binary feature vector is actually created is quite simple. Each index or integer value is an individual vector whereby the vector contains all 0's except for where the index is present, which is marked with a 1. We can see an example of this below.

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Figure 2 - Example of One Hot Encoding to convert a text string into a binary feature vector

As we can see from figure 2 above, we would like to convert the text strings Apple, Chicken, and Broccoli into a binary feature vector. The first step is label encoding which will map an integer value to each text string. Secondly, each index will correspond to a field. In the above example, Chicken is the 2nd integer or the 2nd field, so we put a 1 in that field and 0's everywhere else. This way, we have converted the text string 'Chicken' into a binary feature vector of [0,1,0] through one hot encoding.

7. Explain possible conclusions when observing training and validation loss graphs over epochs (underfitting and overfitting).

As explained briefly in question 5, the number of epochs can have an effect on the training and validation loss graphs such as resulting in underfitting or overfitting. The end goal is to get good generalization but since that is not something we can control, we focus instead on getting good optimization. When the model is initially being trained, both generalization and optimization are related in that when the loss on training data is less, the loss on test data will also be less. This scenario is what is known as underfitting. In this case, there is still more to be learned as the model has not yet learned every pattern on the training dataset. Eventually, there will be a point where the generalization will not improve further, and slowly, the validation accuracy will drop. This scenario is what is known as overfitting. In this case, the model is learning patterns that are very specific to the training dataset but unnecessary when it comes to the testing data or real-world unseen data.

When it comes to observing training and validation loss graphs to identify whether a model is overfitting or underfitting, we can look at the below example.

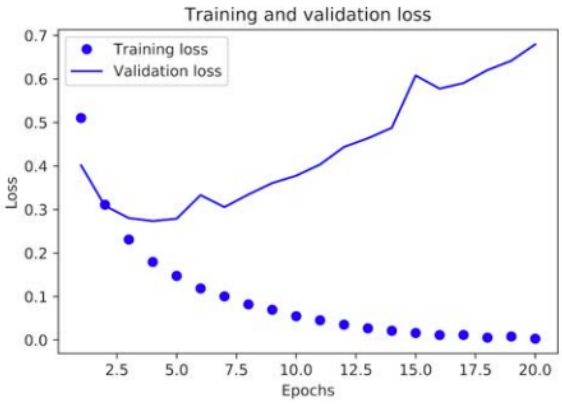

Figure 1 - Training and validation loss graph

If we look at the above graph, we can see that the validation loss slowly decreases until about 5 epochs. After that, the validation loss starts to fluctuate a little and then steadily increase. Any epoch less than 5 would result in underfitting because there is still space for the model to learn more from the training data. On the other hand, any epoch greater than 5 would result in overfitting because although the training loss continues to decrease after 5 epochs, the validation loss now starts to increase, which means that the model is becoming too specific to the training data and will not perform so well on test data. It will be optimal to stop training after 5 epochs.

8. Explain possible hyper-parameters that can be tuned (layers, units per layer, activation functions, loss).

Typically, the number of layers and the units per layer define the size of the model or the complexity. By increasing the layers, the model can understand more complex datasets. For example, if we are training a model for image classification, then having multiple layers is necessary to identify and understand the features of the image. By adjusting the number of layers, we can prevent instances such as overfitting and underfitting. A layer accepts input of a particular shape and then outputs tensors of a specified shape. If the model has multiple layers then Keras has built-in compatibility that allows the shape of the output to be the shape of the input from the previous layer.

The next possible hyper-paramater is the units per layer which refer to the number of artificial neurons in a layer. With a larger number of units per layer, the model will take longer to train, so depending on the complexity of the dataset and the required goals, the appropriate number of units per layer will have to be selected. If overfitting occurs, we can simply reduce the number of units per layer to reduce the complexity of the model, which can help increase generalization. Similarly, if the model needs to be more complex then this can be done by increasing the number of units per layer or also increasing the number of layers.

Activation functions introduce non-linearity into the artificial neural network. Without an activation function, the layers would consist of linear operations which are too restricted. There will be no difference between one layer and multiple layers since only linear operations will occur at each layer. Activation functions calculate the input values of the layer into the output value. This process then repeats with the next layer and so forth. Some common activation functions as mentioned earlier are relu, sigmoid, softmax, etc.

The loss function within the hyper-parameters has to do with how far the output is from the model when compared to the expected output. The predictions that were made from the model and the expected target are taken and the loss function computes the distance in between to give an idea as to how well the model has performed. The loss function evaluates the output. When the loss improves from the loss function, we know that the model is performing well. Examples of popular loss functions are cross-entropy and mean squared error.

9. Explain how a vector of predictions from a binary classifier with a logistic function in the output layer can be converted to class decisions.

For a binary classification problem, after converting the list of integers into a binary feature vector and building the model to make predictions, the predictions can be converted into class decisions by using a probability threshold. This threshold simply allows the data to be classified into one of two decisions based on the value of the prediction. Let us look at the IMDB binary classification example. The goal is to classify movie sentiments/reviews as either positive or negative. An example of the prediction of new data is a vector that outputs a probability in float. We can set a threshold of, let's say, 0.5 to convert the prediction probability into a class decision by saying that anything above 0.5 can be classified as positive sentiment, and everything below 0.5 can be considered negative sentiment.

10. Explain how one-hot-encoding is used to encode class labels of a multi-class classification problem.

One-hot encoding is more commonly used for categorical encoding to encode classes. One-hot-encoding simply takes a label and converts it into a vector with specified dimensions where all the entries would be 0 except for the label index, which would be 1's. For example, if we are looking at a token of words, firstly, each word will correspond to a specific number and we will get a list of integers. The list of integers, through one-hot-encoding, is turned into a vectorized tensor. Let us look at an example of how one-hot-encoding is used to encode a multi-class classification problem.

```
1 red,    green,  blue
2 1,      0,      0
3 0,      1,      0
4 0,      0,      1
```

Figure 2 - Showing an example of one-hot-encoding on multiple classes of colors

If we look at the example above, In order to work with multiple classes of colors, the labels are encoded into vector labels through one-hot-encoding. As previously mentioned, a vector is created and all the values are 0's except for where the index of the label is present, which is represented by 1's. In the above example, we can see that for the color red, there is a 1 on the red and 0's on the others. The 0's tend to be 'dummy values' for the purpose of vectorizing the data.

11. Explain the meaning of the output layer when using softmax as an activation function in it.

The softmax activation function tends to be used a lot within the last dense layer of the model. It is used in both multiclass and single-label classification. The softmax function is used to normalize the output layer by converting it into a probability distribution. Let's say, for example, the output size of the final layer is 3, then the model will output a vector of 3 probability scores where the sum of all these probabilities is 1. Each score represents the probability that the input value belongs to that output class. Let us look at an example below.

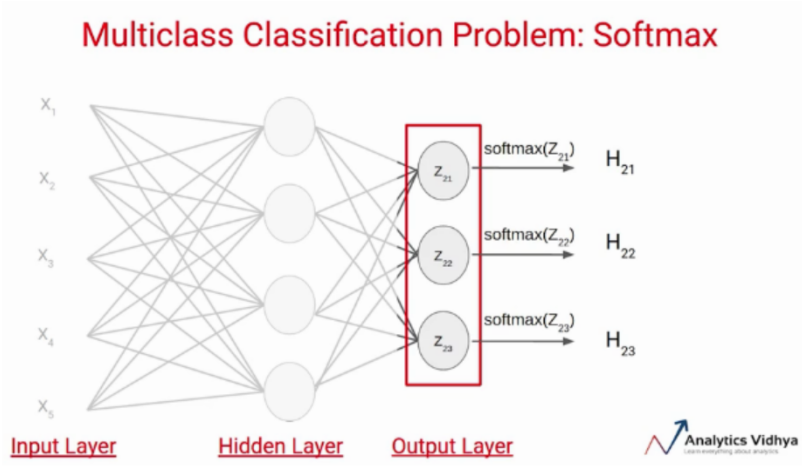

Figure 3 - Showing an example of a neural network with the output layer being a softmax activation function

In this example, we have 3 output classes, Z21, Z22, and Z23. Let us assume that the value of Z21 is 2.33, Z22 is -1.46, and Z23 is 0.56. We can then apply the softmax formula below to convert these output classes into probability values.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)}$$

Figure 4 - Showing the softmax formula where z represents the value of the output layer

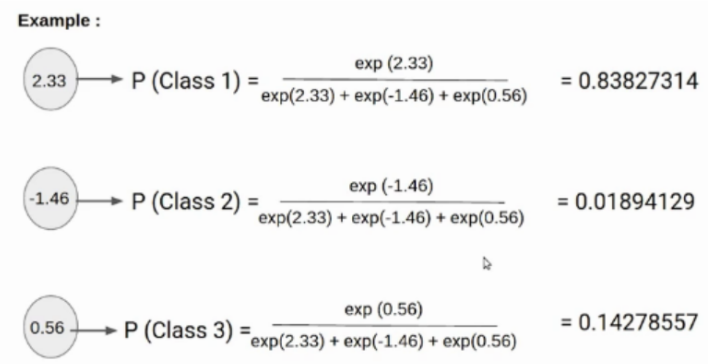

Figure 5 - Showing the probability values of the output layer after applying softmax activation

As shown in figure 6 above, after applying the softmax activation function, the values of the output layer have now changed into probabilities that show the odds that the input belongs to a certain output. In this particular case, we can conclude that the input belongs to class 1 because it has the highest probability. We can also see that the total of these 3 probabilities is 1.

12. Explain the difference between sparse-categorical-crossentropy and categorical-crossentropy.

Categorical-crossentropy is the recommended loss function that is used alongside the softmax activation function. Due to the fact that softmax outputs a probability distribution, categorical-crossentropy works well with softmax because it is used to minimize the distance between the probability distributions. This activation is used in multi-class classification.

The main difference between categorical-crossentropy and sparse-categorical-crossentropy is the use case. When a softmax activation is used and the targets are one-hot-encoded, it is typical to use categorical-crossentropy but when the labels are encoded as integers instead of one-hot-encoding then it is typical to use sparse-categorical-crossentropy. An example of a categorical-crossentropy use case would be for a 4-class classification where the output is a 4-dimensional vector such as [1,0,0,1], [1,0,1,1], [1,1,1,0], [1,1,0,1]. An example of sparse-categorical-crossentropy in the same 4-class classification would be [1], [2], [3], [4]. Therefore, depending on the dataset, the appropriate loss function should be used.

13. Assuming a dataset with 5 classes where each class is represented equally, what will be the accuracy of a random classifier?

The equation for the classification accuracy of a random classifier is:
Accuracy = 1 / k, where k is the number of classes
In this case, accuracy = 1 / 5 = 0.2, when converted to a % accuracy, is 20%

14. Explain how to normalize feature vectors to have equal mean and standard deviation. Explain the purpose of such normalization.

Normalization is part of data preprocessing for neural networks. Before the data is fed into the neural network, it is important to normalize feature vectors so that each has equal mean and equal standard deviation. This normalization is done by subtracting the mean from the training data and dividing it by the standard deviation. This will enable the feature vectors to have a mean of 0 and a standard deviation of 1.

Such normalization is important in order to scale the data because we do not want to feed data into the model that has much larger values compared to initial values, nor do we want to feed data that are in different ranges (for example, some feature vectors in the range of 0-1 and other feature vectors in the range of 100-200). This can cause the network to take much longer to train. By simply normalizing each feature vector, the network can train easier.

15. Explain the difference between MSE and MAE metrics. Which is easier to interpret?

Both MSE and MAE are metrics that are used to evaluate the accuracy of the model. MSE is used to compute the square of the difference between the prediction made by the model and the expected output (target). On the other hand, MAE is used to compute the absolute value of the difference between the prediction and the target. When performing regression, MSE is commonly used as the loss function and MAE is commonly used as the metric. This can be shown below.

```
from keras import models
from keras import layers

def build_model():
    model = models.Sequential()
    model.add(layers.Dense(64, activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
    return model
```

Because you'll need to instantiate the same model multiple times, you use a function to construct it.

Figure 6 - Showing a model defined using mse as the loss function and mae as the metrics

MAE is easier to interpret because if we look at the house prices prediction example that was discussed in class a few lectures ago, if we get an MAE of, let's say, 0.5, then we can conclude that the model prediction was off by $500. We are easily able to interpret and convert the MAE to something that we understand when compared to MSE, which is more complex.

16. Explain how to perform k-fold cross validation. When is k-fold cross validation needed?

K-fold cross validation splits the data into K partitions. The idea behind splitting the data into partitions is to use each fold once as testing data at some point and getting a validation score on each partition. Let us look at the process of how to perform k-fold cross validation while using an example. In this scenario, let us use a k-value of 3 which means the data is split into 3 folds. When training the model using k-fold cross validation, we first use the k-1 fold as the test data and then the remaining folds as the training data. In the first instance, the model will be trained on the 1st and 2nd folds and then validated on the 3rd fold as test data. These steps are then repeated by changing the k-value so the next step will be k-2 and so forth. By the end of it, we would have at some point, used each fold as part of the testing data and have a validation score for each fold. The final validation score is the average of the K validation scores. This process can also be illustrated in the diagram below.

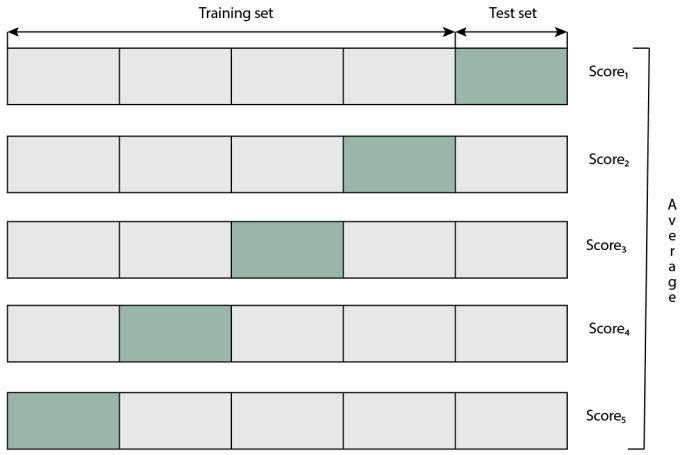

Figure 7 - Showing an illustration of how k-fold cross validation is performed

K-fold cross validation is needed when there is little data available. Initially, we learned that to train the model, we split the data into training and validation datasets. However, when there is little data available, the validation dataset will have a very small number allocated to it, which can result in the validation score fluctuating a lot. This can reduce the accuracy and reliability of the model being trained. Therefore, in instances where there is less data available, k-fold cross validation is used.

17. Explain when performing k-fold cross validation how to report the validation error and how to train the final model.

When performing k-fold cross validation, the validation error can be reported by plotting a graph of epochs against the validation mean absolute error (MAE). The MAE is calculated from the evaluation scores of the model after training it using k-fold cross validation. This is simply done by saving the validation score of the folds after each epoch in a validation score log and then plotting the graph using matplotlib. An example of this can be seen in the graph below.

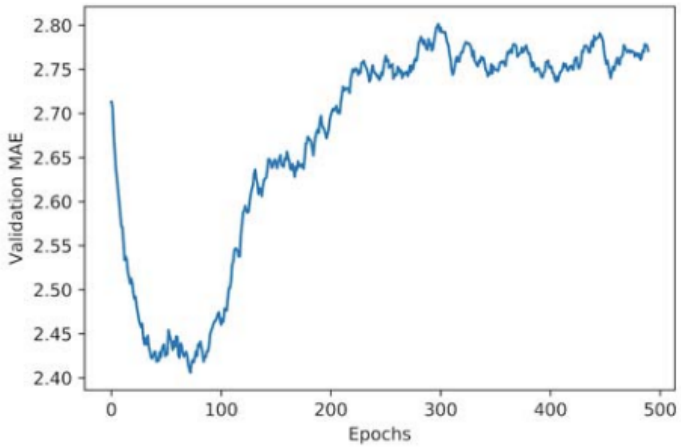

Figure 8 - Graph showing validation MAE by Epochs

From figure 8 above, we can see that after reporting the validation error after each epochs, we can identify when the model is being overfitted and make decisions such as reducing the number of epochs and tuning other parameters to create the final model.

When it comes to training the final model using k-fold cross validation, you analyze the validation error graph and adjust various parameters. If we take a look at the same example of predicting the house prices discussed earlier, in figure 8, after retraining the model with 500 epochs, we can conclude that the model starts to overfit after 80 epochs, so after training a fresh model with 80 epochs and adjusting the batch size from 1 to 16 as well as adjusting other parameters such as the size of hidden layers, the final model predictions were off by $2,550 which is an improvement from the $3,000 prediction of the initial model.

In conclusion, after every epoch, the validation MAE is calculated then after the model has been trained, a graph can be plotted to analyze how well the model has performed after k-fold cross validation. After that, the graph can be analyzed and changes to the parameters of the model can be made in order to train the final model. This model will have optimized parameters and is expected to perform better.