Experiment Overfit Prevent Techniques on A Simple Neural Network on Pima Dataset

Pooya Danandeh

Pooya144@gmail.com

Department of Computer Science, University of Tabriz, Tabriz, Iran

Abstract

Overfitting is most common problem in machine learning and deep learning models people face with it. Overfitting means the model has high accuracy on training data but has low accuracy on unseen or test data. This can be dangerous in field of medicine and real time systems such as cancer detection and anomaly detection. In this paper we try to solve this problem to some extent with some techniques.

Keywords: Overfitting, Machine learning, Deep Learning

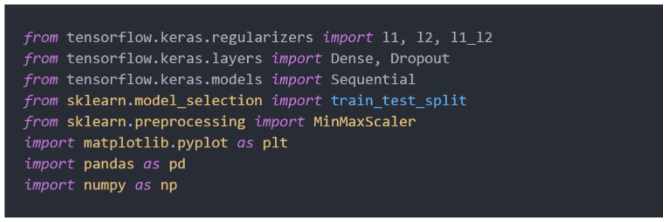
1 Introduction

Overfitting caused by some reasons such as excessive complexity of the model or excessive increase network weights. We try to prevent overfitting by control on model complexity and weights of network. We use 3 methods for this aim: Dropout, Regularization and Reduction. We will implement all these 3 methods in this paper using Keras library.

2 Implementation

2.1 Need Tools

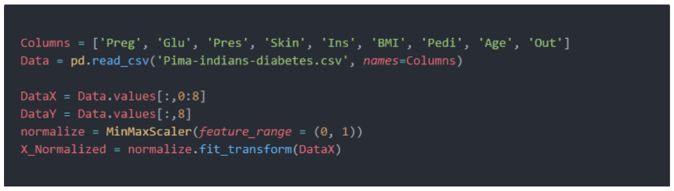
First, we should import all need tools.



```
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.models import Sequential
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

2.2 Load and Preprocessing Data

In this step we load pima dataset contains 9 columns (8 columns data and 1 column labels) and normalize them.

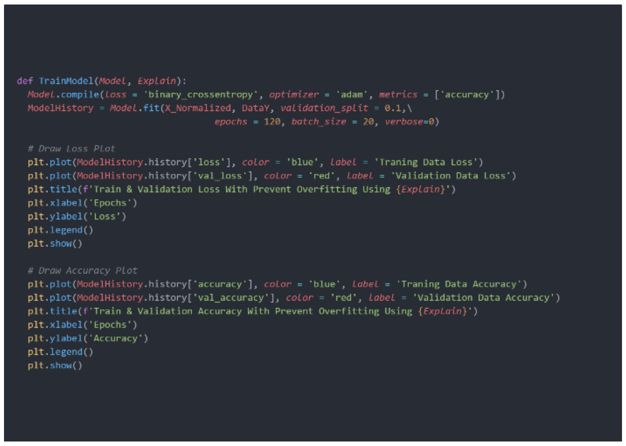


```
Columns = ['Preg', 'Glu', 'Pres', 'Skin', 'Ins', 'BMI', 'Pedi', 'Age', 'Out']
Data = pd.read_csv('Pima-indians-diabetes.csv', names=Columns)

DataX = Data.values[:,0:8]
DataY = Data.values[:,8]
normalize = MinMaxScaler(feature_range = (0, 1))
X_Normalized = normalize.fit_transform(DataX)
```

2.3 Train Model and Draw its Plots

We define a function that trains a model then draws accuracy and loss on train and validation data plots. We use 10% of dataset for validation and set number of epochs to 120 also batches size is 20.



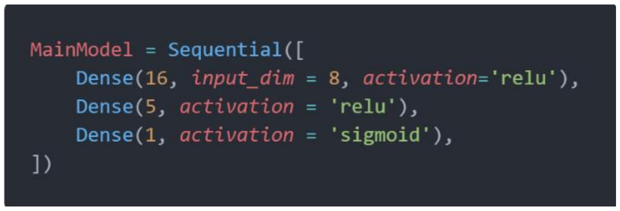
```
def TrainModel(Model, Explain):
  Model.compile(loss = 'binary_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
  ModelHistory = Model.fit(X_Normalized, DataY, validation_split = 0.1,\
                                 epochs = 120, batch_size = 20, verbose=0)

  # Draw Loss Plot
  plt.plot(ModelHistory.history['loss'], color = 'blue', label = 'Traning Data Loss')
  plt.plot(ModelHistory.history['val_loss'], color = 'red', label = 'Validation Data Loss')
  plt.title(f'Train & Validation Loss With Prevent Overfitting Using {Explain}')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()

  # Draw Accuracy Plot
  plt.plot(ModelHistory.history['accuracy'], color = 'blue', label = 'Traning Data Accuracy')
  plt.plot(ModelHistory.history['val_accuracy'], color = 'red', label = 'Validation Data Accuracy')
  plt.title(f'Train & Validation Accuracy With Prevent Overfitting Using {Explain}')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()
```

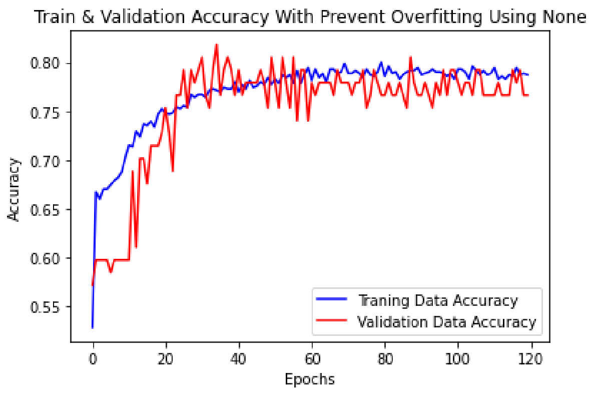
2.4 Main Model

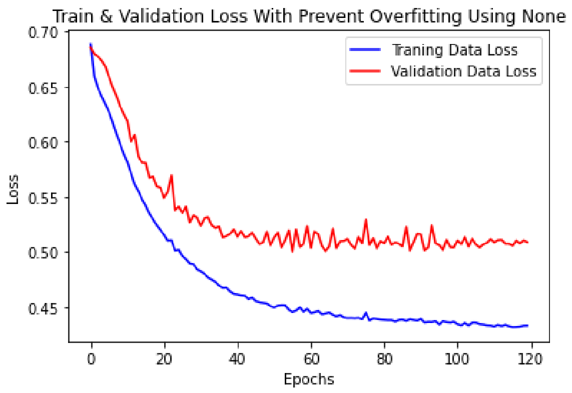
This model is the base, that means designed simple and without any try to prevent overfitting. This network has 2 layers (hidden layers) with Relu activation function. First layer has 16 neurons and second one has 8. We have also 8 neurons at input layer and 1 neuron for output with Sigmoid activation function.



```
MainModel = Sequential([
    Dense(16, input_dim = 8, activation='relu'),
    Dense(5, activation = 'relu'),
    Dense(1, activation = 'sigmoid'),
])
```

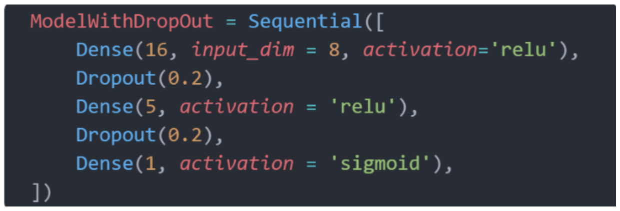
After training, we got these results:





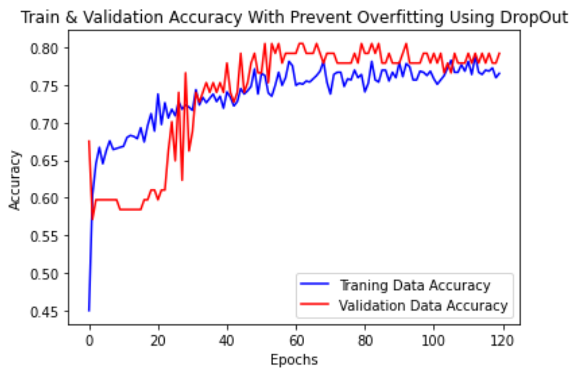
2.5 Apply Dropout

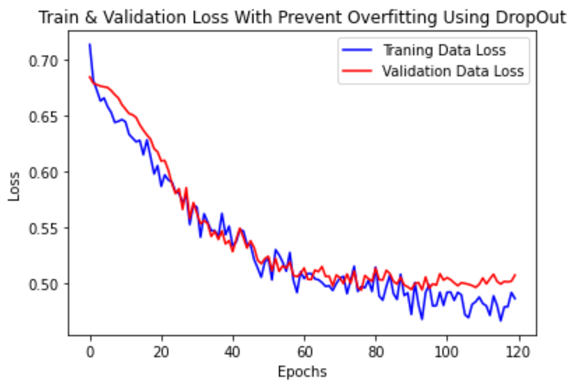
In this method some neurons drop out in every epoch, neuron drop out rate is 20%, that means 20% of every hidden layer neuron drop out in every epoch. This method controls network complexity.



```
ModelWithDropOut = Sequential([
    Dense(16, input_dim = 8, activation='relu'),
    Dropout(0.2),
    Dense(5, activation = 'relu'),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

After training, we got these results:



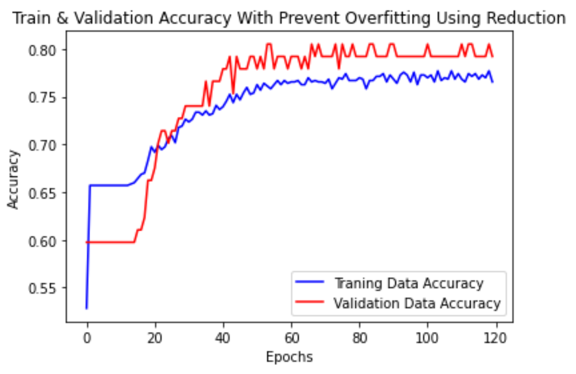


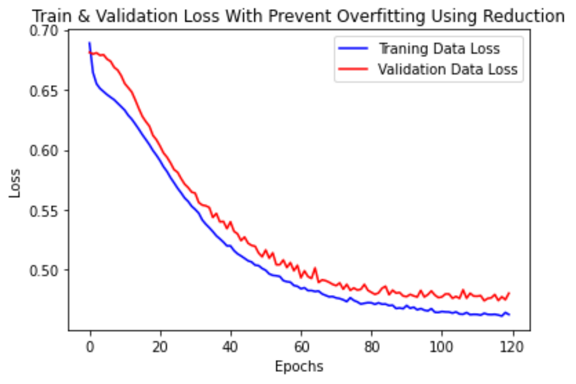
2.6 Apply Reduction

In Reduction we reduce number of neurons manually. This method controls the network complexity. We reduced first layer neurons to 8 and second layer to 3.

```
ModelReduced = Sequential([
    Dense(8, input_dim = 8, activation='relu'),
    Dense(3, activation = 'relu'),
    Dense(1, activation = 'sigmoid'),
])
```

Results after training:





2.7 Apply Regularization

In Regularization we prevent excessive increase network weights by defining Regularization Coefficient(λ). We have 3 kind of regularization for neural networks:

2.7.1 L1 Regularization

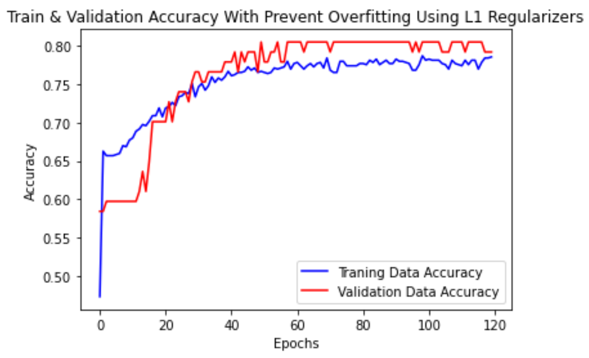
In L1 Regularization (Lasso) we add sum of absolute values of magnitude of coefficient to cost function as penalty.

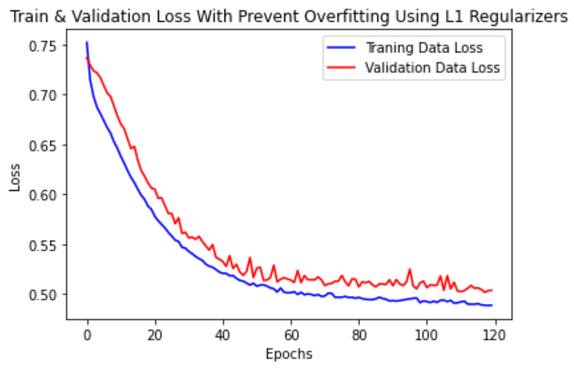
$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}|\beta_j|$$

In this equation, y_i is true label, β_i is network weights and λ is regularization coefficient. If λ be too small, our cost function is similar to main cost function (in main model) and large value of λ cause be underfitting in the network. So, we assume λ=0.001.

```
ModelWithL1Regularizers = Sequential([
    Dense(16, input_dim = 8, activation='relu',kernel_regularizer=l1(0.001)),
    Dense(5, activation = 'relu',kernel_regularizer=l1(0.001)),
    Dense(1, activation = 'sigmoid'),
])
```

And results after training:





2.7.2 L2 Regularization

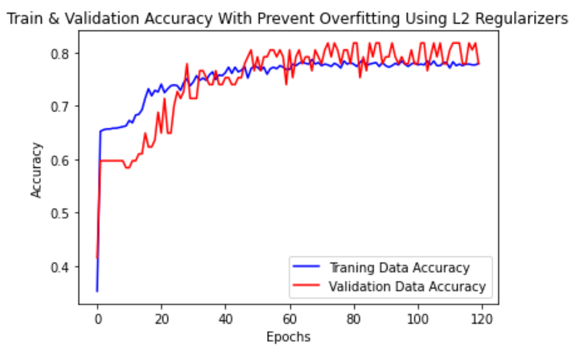
In L2 Regularization (Ridge) we add sum of squared values of magnitude of coefficient to cost function as penalty.

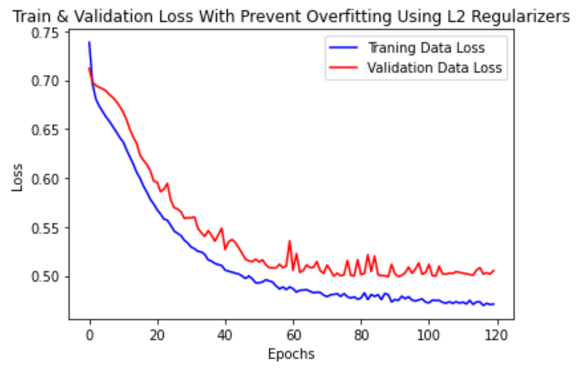
$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

such as L1 Regularization, y_i is true label, β_i is network weights and λ is regularization coefficient. Also, too small value of λ makes our cost function look like to main cost function (in main model) and large value of λ cause be underfitting in the network too. So, again we assume λ=0.001.

```
ModelWithL2Regularizers = Sequential([
    Dense(16, input_dim = 8, activation='relu',kernel_regularizer=l2(0.001)),
    Dense(5, activation = 'relu',kernel_regularizer=l2(0.001)),
    Dense(1, activation = 'sigmoid'),
])
```

After training this network we got these results:



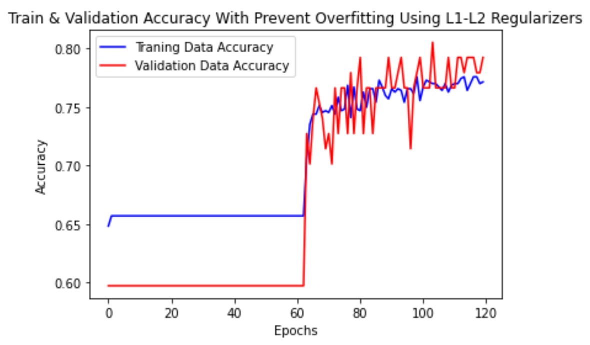


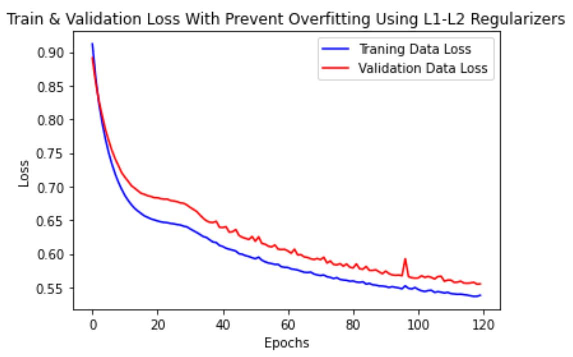
2.7.3 L1-L2 Regularization

Using L1 and L2 at the same time in the network layers

```
ModelWithL1L2Regularizers = Sequential([
    Dense(16, input_dim = 8, activation='relu',kernel_regularizer=l1_l2(0.001)),
    Dense(5, activation = 'relu',kernel_regularizer=l1_l2(0.001)),
    Dense(1, activation = 'sigmoid'),
])
```

After training:



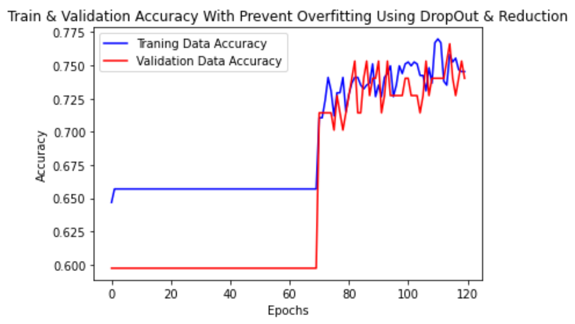


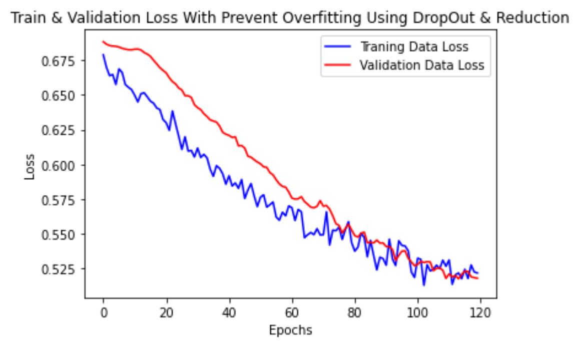
2.8 Apply Dropout and Reduction

In this step we apply Dropout and Reduction to the network at the same time.

```
ModelWithDropOutAndReduction = Sequential([
    Dense(8, input_dim = 8, activation='relu'),
    Dropout(0.2),
    Dense(3, activation = 'relu'),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

And results after training:



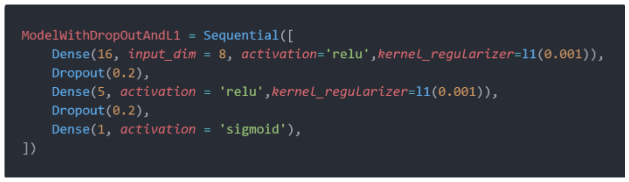


2.8 Apply Dropout and Regularization

In this step we apply Dropout and every one of 3 Regularization methods on the network at the same time.

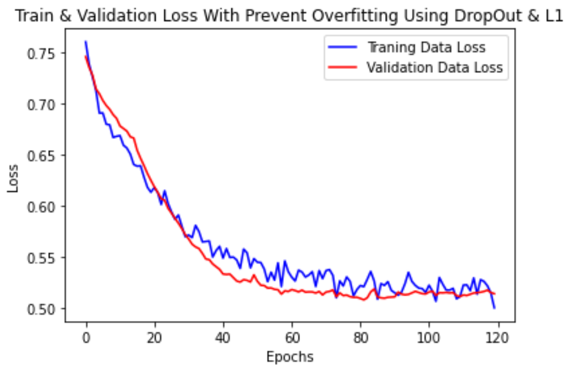
2.8.1 Apply Dropout and L1

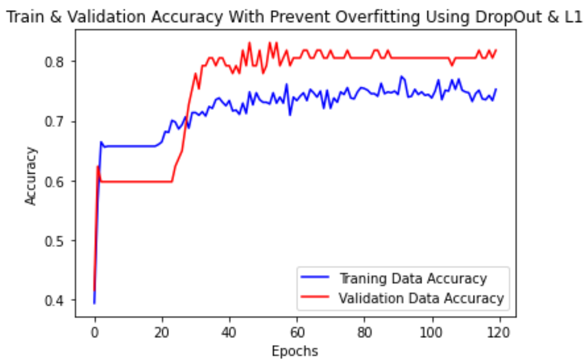
Dropout and L1 Regularization at the same time.



```
ModelWithDropOutAndL1 = Sequential([
    Dense(16, input_dim = 8, activation='relu',kernel_regularizer=l1(0.001)),
    Dropout(0.2),
    Dense(5, activation = 'relu',kernel_regularizer=l1(0.001)),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

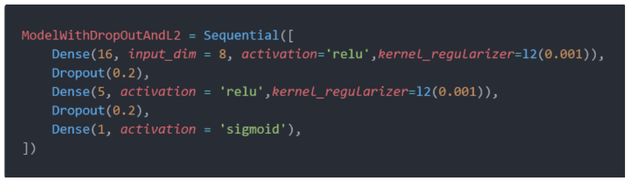
Results after training:





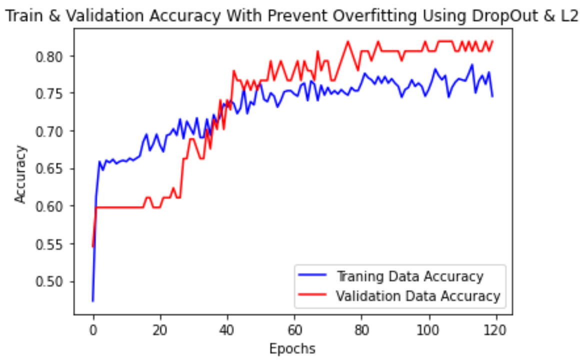
2.8.2 Apply Dropout and L2

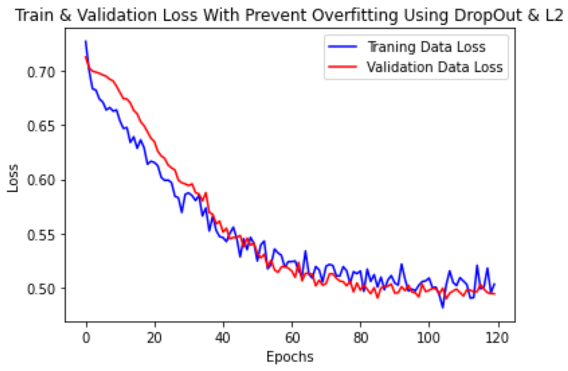
Dropout and L2 Regularization at the same time.



```
ModelWithDropOutAndL2 = Sequential([
    Dense(16, input_dim = 8, activation='relu',kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(5, activation = 'relu',kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

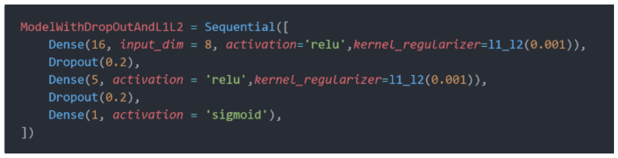
After training:





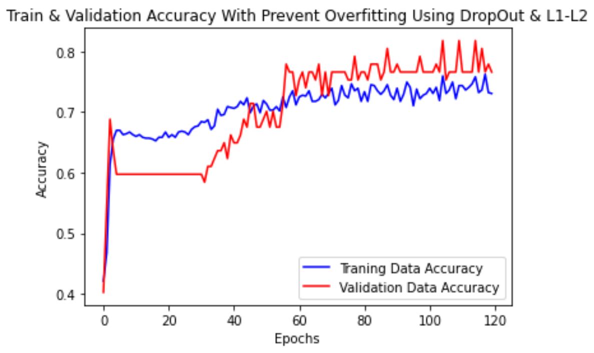
2.8.1 Apply Dropout and L1-L2

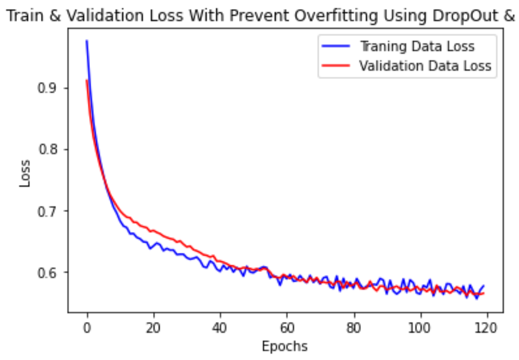
Dropout and L1-L2 Regularization at the same time.



```
ModelWithDropOutAndL1L2 = Sequential([
    Dense(16, input_dim = 8, activation='relu',kernel_regularizer=l1_l2(0.001)),
    Dropout(0.2),
    Dense(5, activation = 'relu',kernel_regularizer=l1_l2(0.001)),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

And results:



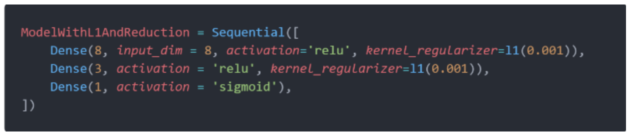


2.9 Apply Reduction and Regularization

In this step we will apply Reduction and every one of 3 Regularization methods on the network at the same time

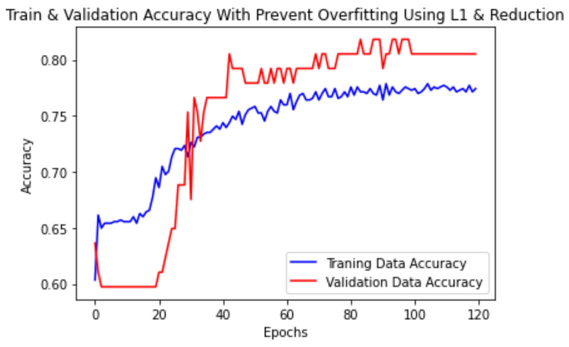
2.9.1 Apply Reduction and L1

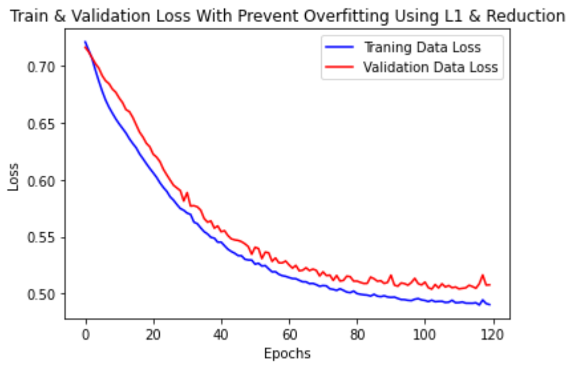
Reduction and L1 Regularization at the same time.



```
ModelWithL1AndReduction = Sequential([
    Dense(8, input_dim = 8, activation='relu', kernel_regularizer=l1(0.001)),
    Dense(3, activation = 'relu', kernel_regularizer=l1(0.001)),
    Dense(1, activation = 'sigmoid'),
])
```

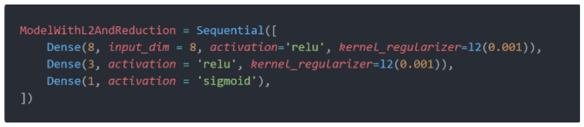
Results:





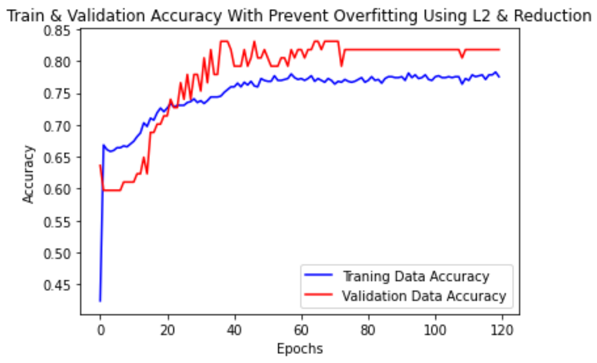
2.9.1 Apply Reduction and L2

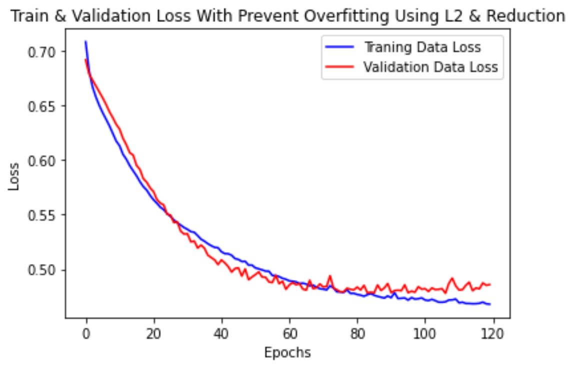
Reduction and L2 Regularization at the same time.



```
ModelWithL2AndReduction = Sequential([
    Dense(8, input_dim = 8, activation='relu', kernel_regularizer=l2(0.001)),
    Dense(3, activation = 'relu', kernel_regularizer=l2(0.001)),
    Dense(1, activation = 'sigmoid'),
])
```

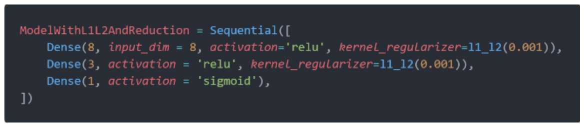
And the results:





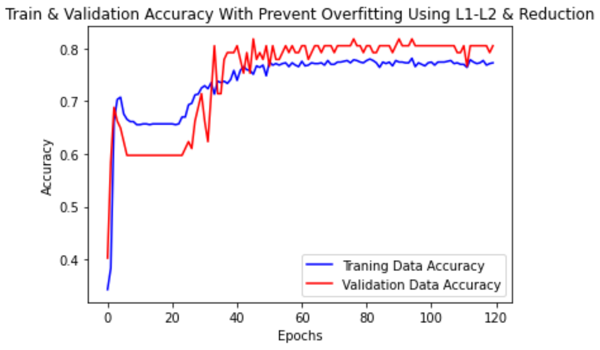
2.9.1 Apply Reduction and L1-L2

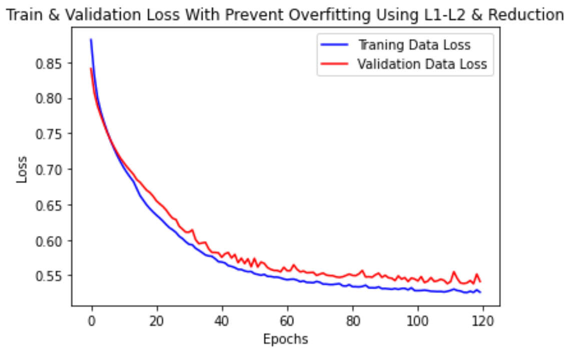
Reduction and L1-L2 Regularization at the same time.



```
ModelWithL1L2AndReduction = Sequential([
    Dense(8, input_dim = 8, activation='relu', kernel_regularizer=l1_l2(0.001)),
    Dense(3, activation = 'relu', kernel_regularizer=l1_l2(0.001)),
    Dense(1, activation = 'sigmoid'),
])
```

Result after training:



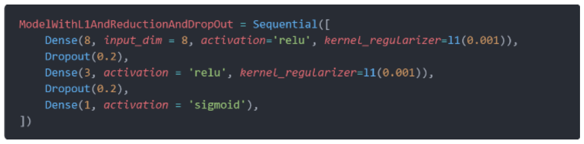


2.10 Apply All Methods

In this step we will apply Reduction, Dropout and every one of 3 Regularization methods on the network at the same time.

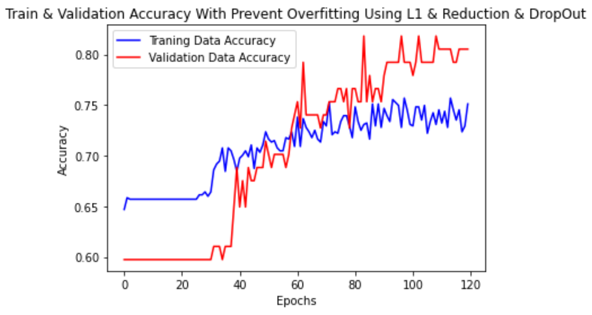
2.10.1 Dropout, Reduction and L1

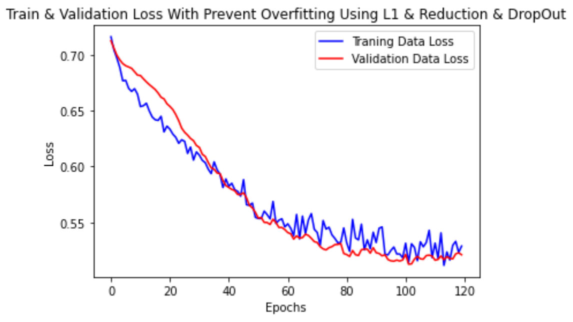
Dropout, Reduction and L1 Regularization at the same time.



```
ModelWithL1AndReductionAndDropOut = Sequential([
    Dense(8, input_dim = 8, activation='relu', kernel_regularizer=l1(0.001)),
    Dropout(0.2),
    Dense(3, activation = 'relu', kernel_regularizer=l1(0.001)),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

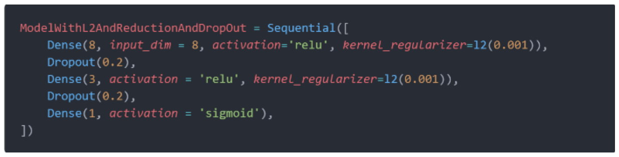
Results after training:





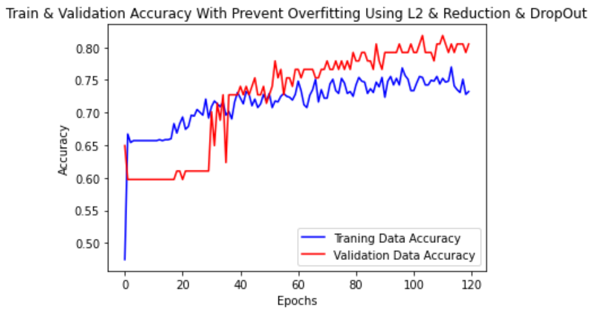
2.10.2 Dropout, Reduction and L2

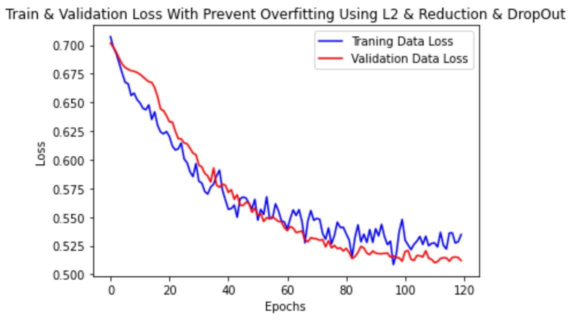
Dropout, Reduction and L2 Regularization at the same time.



```
ModelWithL2AndReductionAndDropOut = Sequential([
    Dense(8, input_dim = 8, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(3, activation = 'relu', kernel_regularizer=l2(0.001)),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

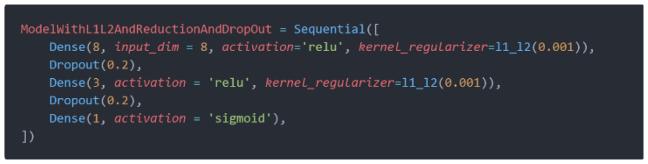
After training, we got these results:





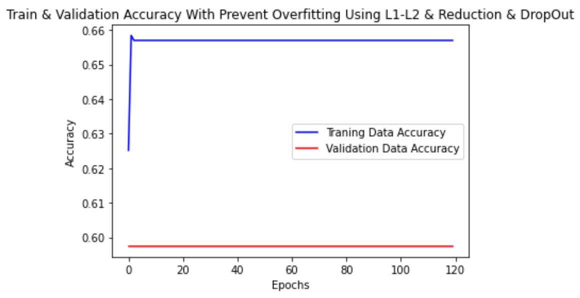
2.10.3 Dropout, Reduction and L1-L2

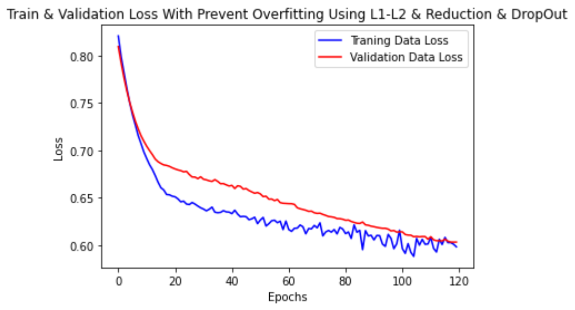
Dropout and Reduction and L1-L2 Regularization at the same time.



```
ModelWithL1L2AndReductionAndDropOut = Sequential([
    Dense(8, input_dim = 8, activation='relu', kernel_regularizer=l1_l2(0.001)),
    Dropout(0.2),
    Dense(3, activation = 'relu', kernel_regularizer=l1_l2(0.001)),
    Dropout(0.2),
    Dense(1, activation = 'sigmoid'),
])
```

Results after training:





3 Conclusion

If we check all plots of each methods introduced in this paper, we show “L1-Regularization” and “Dropout” better than other methods can prevent overfitting occurrence. And using all of these methods (L1-L2, Dropout and Reduction) gave us worst results.

4 References

[1] L1 and L2 regularization methods

https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

[2] L1 and L2 regularization in Machine-Learning

https://www.analyticssteps.com/blogs/l2-and-l1-regularization-machine-learning

[2] Simple Neural Network on Pima

https://www.kaggle.com/pooyadanandeh/pima-indians-diabetes

https://www.kaggle.com/pooyadanandeh/keras-pandas-practice-2