How Dropout Layer reduces overfitting in Neural Network

In this example, we will build two prediction models using Keras. In one model we'll only use only **dense layers** and in another model, we will add the **dropout layer**. We will then compare the performance of these two models where the evaluation criteria will be the prediction result obtained on the test dataset.

In this problem, we will generate random data samples with **"make_circles"** function of Keras and visualize it with a scatter plot using the Matplotlib library. Then we proceed to perform binary classification on the dataset with the model we build with Keras.

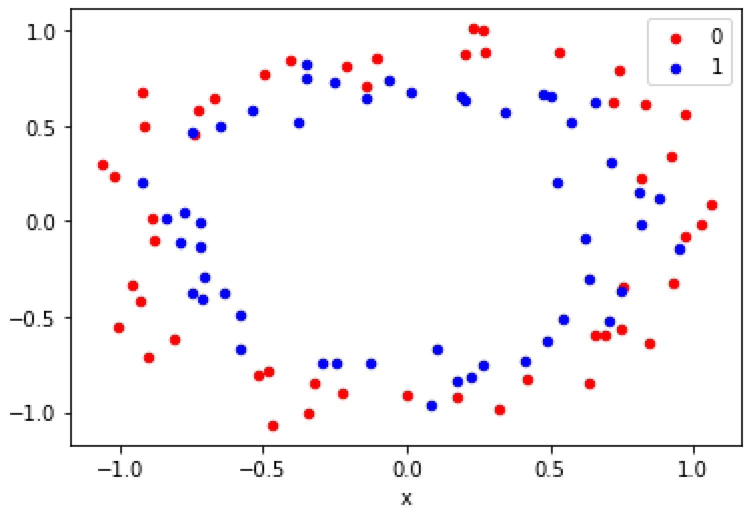
Generate Dataset

```
# generate two circles dataset
from sklearn.datasets import make_circles
from matplotlib import pyplot
from pandas import DataFrame
```

```
from sklearn import datasets

# generate 2d classification dataset
X, y = make_circles(n_samples=100, noise=0.1, random_state=1)
```

```
# scatter plot, dots colored by class value
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue'}
fig, ax = pyplot.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key,
color=colors[key])
pyplot.show()
```



Building a Model without Dropouts

The next step is to split the dataset into a training set and testing set. Now it’s time to create the model without a dropout layer.

```
X, y = make_circles(n_samples=100, noise=0.1, random_state=1)
# split into train and test
n_train = 30
trainX, testX = X[:n_train, :], X[n_train:, :]
trainy, testy = y[:n_train], y[n_train:]
```

Here Keras in-built models and layers are imported, we use **Sequential** model. After this, a couple of **Dense layers** are added, one with **relu** activation and the other one with **sigmoid** activation. Lastly, this model is compiled and we look to calculate the **accuracy** of the results with metrics parameter set to “accuracy”.

```
from keras import models
from keras import layers

# define model
model = models.Sequential()
model.add(layers.Dense(500, input_dim=2, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Now the above model is fitted and trained over the data available with us. Here we also specify the number of epochs(iterations).

```
# fit model

history = model.fit(trainX, trainy, validation_data=(testX,
testy), epochs=4000, verbose=0)
```

At last, we will check the results of the model on testing data. The results will contains the **accuracy** value, thus suggesting how many values were correctly classified and how many weren’t classified correctly.

```
# evaluate the model
_, train_acc = model.evaluate(trainX, trainy, verbose=0)
_, test_acc = model.evaluate(testX, testy, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

Output:

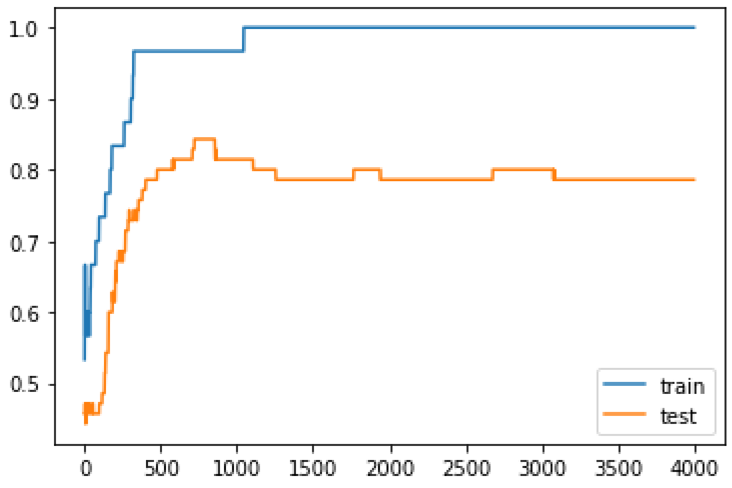
```

Train: 1.000, Test: 0.786
```

Clearly, the result shows that **training accuracy** is **“1.000”** which means 100% accuracy but we are not concerned with this value, the actual value to be looked at is the **testing accuracy**. It has resulted as **“0.786”** i.e. almost **79%** accuracy.

With the help of the matplotlib library, we plot the testing and training accuracy values and clearly, it has **overfitted**.

```
# plot history
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='test')
pyplot.legend()
pyplot.show()
```



Building a Model using Dropouts Layer

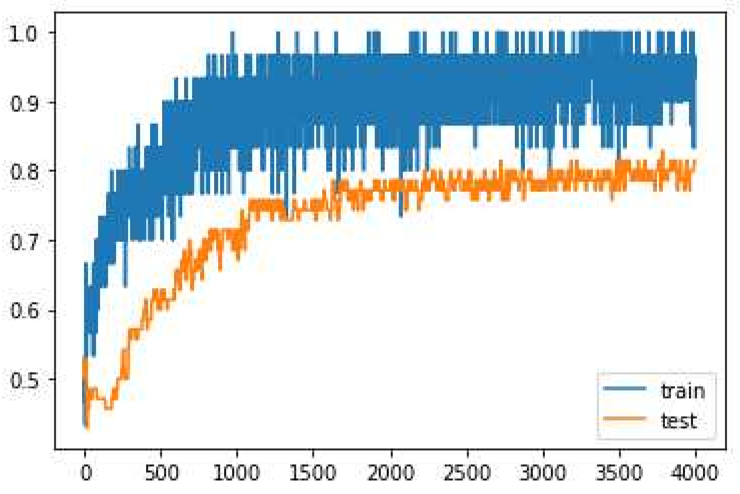
Now it’s time to use the Dropout layer in the model and see whether the performance will be boosted or not.

Again we’ll be using the same dataset, the model will now have another layer i.e. dropout layer. We have passed **“0.5” in the Dropout function**, this will actually drop 50% of input values for regularizing the dataset and the model as well.

```
cc
```

Output:

```
Train: 0.967, Test: 0.814
```



You can see for yourselves, there is a slight dip in training accuracy which is actually insignificant but the testing accuracy has increased. **Testing Accuracy** is **“0.814”** which means **81%** and the is **NO overfitting**.

So this comparison-based example clearly shows that the **dropout layer** can boost the model performance and avoid overfitting.

Tips on using Dropout Layer

We will end this article by discussing some tips that one must remember while working with the Dropout layer.

- Experiments show that the dropout value should range from 20% to 50%. In case you choose values below this range then the desired effects won’t be visible and if the values above this range are chosen then the network may suffer from under-learning.

- If the resources permit, try to use a larger network as this has more probability of giving better results. This is also because the model will get more opportunities to learn.
- Remember that the dropout layer can be used on all the visible i.e. input units and also on hidden units. Results have been pretty good if dropout is used at each layer of a network.

- **Also Read** – Different Types of Keras Layers Explained for Beginners
- **Also Read** – Keras vs Tensorflow vs Pytorch – No More Confusion !!

Conclusion

The tutorial explained the Keras DropoutLlayer function and its parameters, where we discussed the importance of the dropout layer. We also covered examples to show how dropout can reduce overfitting.