Open in Colab

T81-558: Applications of Deep Neural Networks

Module 5: Regularization and Dropout

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 5 Material

- Part 5.1: Part 5.1: Introduction to Regularization: Ridge and Lasso [Video] [Notebook]
- Part 5.2: Using K-Fold Cross Validation with Keras [Video] [Notebook]
- Part 5.3: Using L1 and L2 Regularization with Keras to Decrease Overfitting [Video] [Notebook]
- **Part 5.4: Drop Out for Keras to Decrease Overfitting** [Video] [Notebook]
- Part 5.5: Benchmarking Keras Deep Learning Regularization Techniques [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: not using Google CoLab
```

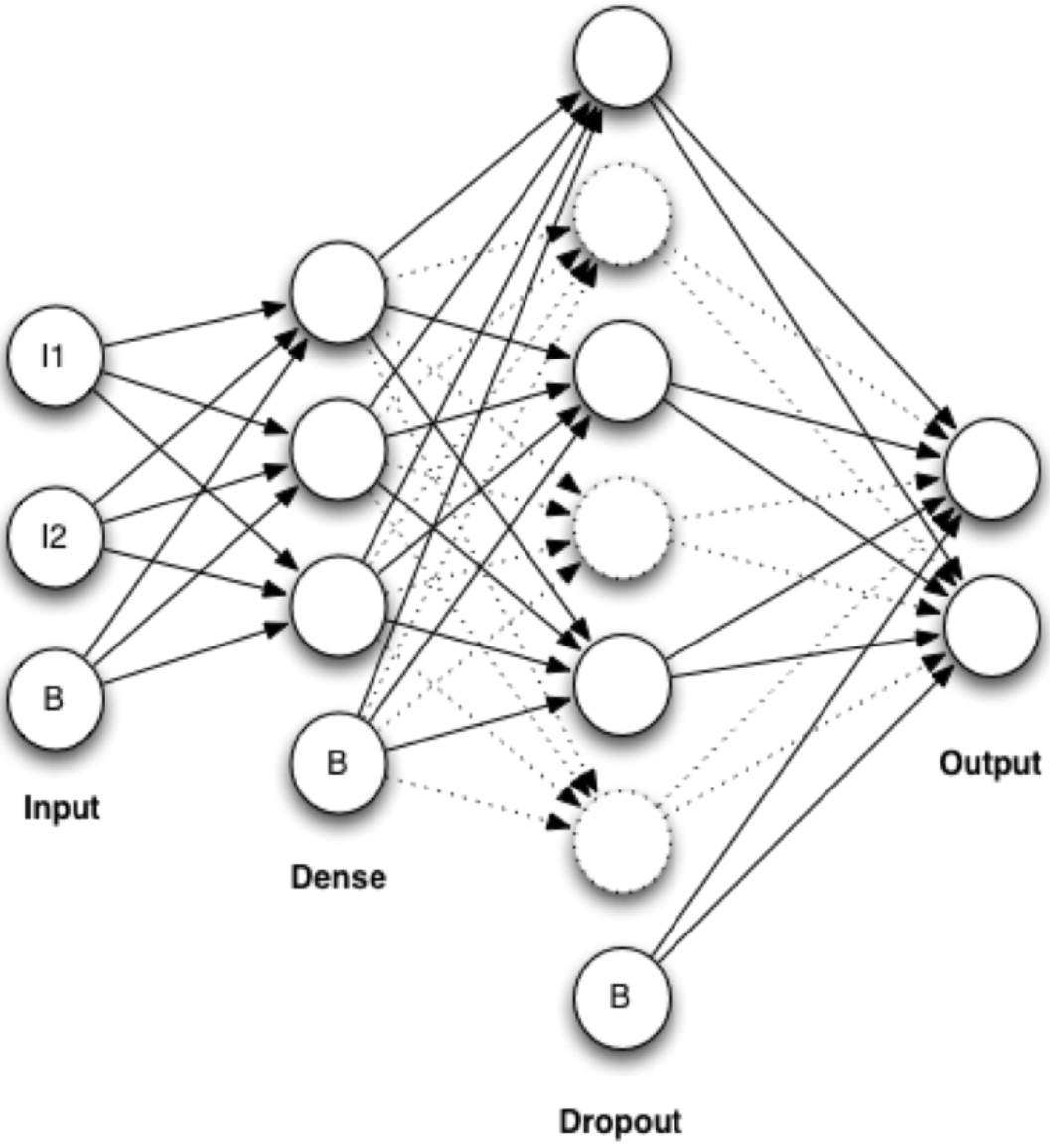
Part 5.4: Drop Out for Keras to Decrease Overfitting

Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov (2012) introduced the dropout regularization algorithm. [Cite:srivastava2014dropout] Although dropout works differently than L1 and L2, it accomplishes the same goal—the prevention of overfitting. However, the algorithm does the task by actually removing neurons and connections—at least temporarily. Unlike L1 and L2, no weight penalty is added. Dropout does not directly seek to train small weights. Dropout works by causing hidden neurons of the neural network to be unavailable during part of the training. Dropping part of the neural network causes the remaining portion to be trained to still achieve a good score even without the dropped neurons. This technique decreases co-adaptation between neurons, which results in less overfitting.

Most neural network frameworks implement dropout as a separate layer. Dropout layers function like a regular, densely connected neural network layer. The only difference is that the dropout layers will periodically drop some of their neurons during training. You can use dropout layers on regular feedforward neural networks.

The program implements a dropout layer as a dense layer that can eliminate some of its neurons. Contrary to popular belief about the dropout layer, the program does not permanently remove these discarded neurons. A dropout layer does not lose any of its neurons during the training process, and it will still have the same number of neurons after training. In this way, the program only temporarily masks the neurons rather than dropping them. Figure 5.DROPOUT shows how a dropout layer might be situated with other layers.

Figure 5.DROPOUT: Dropout Regularization



The discarded neurons and their connections are shown as dashed lines. The input layer has two input neurons as well as a bias neuron. The second layer is a dense layer with three neurons and a bias neuron. The third layer is a dropout layer with six regular neurons even though the program has dropped 50% of them. While the program drops these neurons, it neither calculates nor trains them. However, the final neural network will use all of these neurons for the output. As previously mentioned, the program only temporarily discards the neurons.

The program chooses different sets of neurons from the dropout layer during subsequent training iterations. Although we chose a probability of 50% for dropout, the computer will not necessarily drop three neurons. It is as if we flipped a coin for each of the dropout candidate neurons to choose if that neuron was dropped out. You must know that the program should never drop the bias

neuron. Only the regular neurons on a dropout layer are candidates. The implementation of the training algorithm influences the process of discarding neurons. The dropout set frequently changes once per training iteration or batch. The program can also provide intervals where all neurons are present. Some neural network frameworks give additional hyper-parameters to allow you to specify exactly the rate of this interval.

Why dropout is capable of decreasing overfitting is a common question. The answer is that dropout can reduce the chance of codependency developing between two neurons. Two neurons that develop codependency will not be able to operate effectively when one is dropped out. As a result, the neural network can no longer rely on the presence of every neuron, and it trains accordingly. This characteristic decreases its ability to memorize the information presented, thereby forcing generalization.

Dropout also decreases overfitting by forcing a bootstrapping process upon the neural network. Bootstrapping is a prevalent ensemble technique. Ensembling is a technique of machine learning that combines multiple models to produce a better result than those achieved by individual models. The ensemble is a term that originates from the musical ensembles in which the final music product that the audience hears is the combination of many instruments.

Bootstrapping is one of the most simple ensemble techniques. The bootstrapping programmer simply trains several neural networks to perform precisely the same task. However, each neural network will perform differently because of some training techniques and the random numbers used in the neural network weight initialization. The difference in weights causes the performance variance. The output from this ensemble of neural networks becomes the average output of the members taken together. This process decreases overfitting through the consensus of differently trained neural networks.

Dropout works somewhat like bootstrapping. You might think of each neural network that results from a different set of neurons being dropped out as an individual member in an ensemble. As training progresses, the program creates more neural networks in this way. However, dropout does not require the same amount of processing as bootstrapping. The new neural networks created are temporary; they exist only for a training iteration. The final result is also a single neural network rather than an ensemble of neural networks to be averaged together.

The following animation shows how dropout works: animation link

In [2]:
```
import pandas as pd
from scipy.stats import zscore

```

```
# Read the data set
df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/jh-simple-dataset.csv",
    na_values=['NA','?'])

# Generate dummies for job
df = pd.concat([df,pd.get_dummies(df['job'],prefix="job")],axis=1)
df.drop('job', axis=1, inplace=True)

# Generate dummies for area
df = pd.concat([df,pd.get_dummies(df['area'],prefix="area")],axis=1)
df.drop('area', axis=1, inplace=True)

# Missing values for income
med = df['income'].median()
df['income'] = df['income'].fillna(med)

# Standardize ranges
df['income'] = zscore(df['income'])
df['aspect'] = zscore(df['aspect'])
df['save_rate'] = zscore(df['save_rate'])
df['age'] = zscore(df['age'])
df['subscriptions'] = zscore(df['subscriptions'])

# Convert to numpy - Classification
x_columns = df.columns.drop('product').drop('id')
x = df[x_columns].values
dummies = pd.get_dummies(df['product']) # Classification
products = dummies.columns
y = dummies.values
```

Now we will see how to apply dropout to classification.

In [3]:

```
########################################
# Keras with dropout for Classification
########################################

import pandas as pd
import os
import numpy as np
from sklearn import metrics
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras import regularizers

# Cross-validate
kf = KFold(5, shuffle=True, random_state=42)

oos_y = []
oos_pred = []
fold = 0

for train, test in kf.split(x):
    fold+=1
```

```
    print(f"Fold #{fold}")

    x_train = x[train]
    y_train = y[train]
    x_test = x[test]
    y_test = y[test]

    #kernel_regularizer=regularizers.l2(0.01),

    model = Sequential()
    model.add(Dense(50, input_dim=x.shape[1], activation='relu')) # Hidden 1
    model.add(Dropout(0.5))
    model.add(Dense(25, activation='relu', \
                activity_regularizer=regularizers.l1(1e-4))) # Hidden 2
    # Usually do not add dropout after final hidden layer
    #model.add(Dropout(0.5))
    model.add(Dense(y.shape[1],activation='softmax')) # Output
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    model.fit(x_train,y_train,validation_data=(x_test,y_test),\
              verbose=0,epochs=500)

    pred = model.predict(x_test)

    oos_y.append(y_test)
    # raw probabilities to chosen class (highest probability)
    pred = np.argmax(pred,axis=1)
    oos_pred.append(pred)

    # Measure this fold's accuracy
    y_compare = np.argmax(y_test,axis=1) # For accuracy calculation
    score = metrics.accuracy_score(y_compare, pred)
    print(f"Fold score (accuracy): {score}")


# Build the oos prediction list and calculate the error.
oos_y = np.concatenate(oos_y)
oos_pred = np.concatenate(oos_pred)
oos_y_compare = np.argmax(oos_y,axis=1) # For accuracy calculation

score = metrics.accuracy_score(oos_y_compare, oos_pred)
print(f"Final score (accuracy): {score}")

# Write the cross-validated prediction
oos_y = pd.DataFrame(oos_y)
oos_pred = pd.DataFrame(oos_pred)
oosDF = pd.concat( [df, oos_y, oos_pred],axis=1 )
#oosDF.to_csv(filename_write,index=False)
```

```
Fold #1
Fold score (accuracy): 0.68
Fold #2
Fold score (accuracy): 0.695
Fold #3
Fold score (accuracy): 0.7425
Fold #4
Fold score (accuracy): 0.71
Fold #5
Fold score (accuracy): 0.6625
Final score (accuracy): 0.698
```

In [ ]: