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]:
```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

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
```

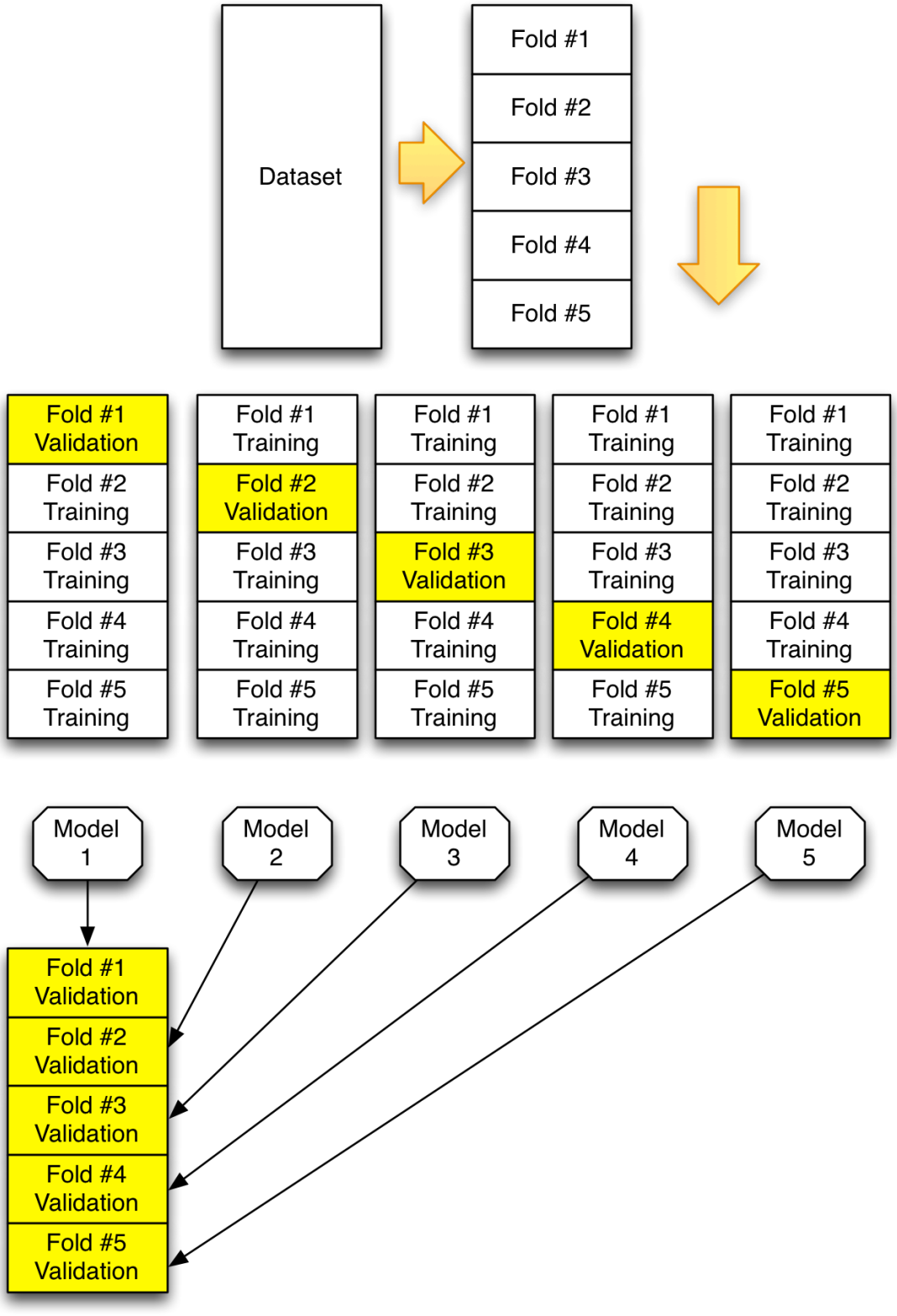
Part 5.2: Using K-Fold Cross-validation with Keras

You can use cross-validation for a variety of purposes in predictive modeling:

- Generating out-of-sample predictions from a neural network
- Estimate a good number of epochs to train a neural network for (early stopping)
- Evaluate the effectiveness of certain hyperparameters, such as activation functions, neuron counts, and layer counts

Cross-validation uses several folds and multiple models to provide each data segment a chance to serve as both the validation and training set. Figure 5.CROSS shows cross-validation.

Figure 5.CROSS: K-Fold Crossvalidation



It is important to note that each fold will have one model (neural network). To generate predictions for new data (not present in the training set), predictions from the fold models can be handled in several ways:

- Choose the model with the highest validation score as the final model.

- Preset new data to the five models (one for each fold) and average the result (this is an ensemble).
- Retrain a new model (using the same settings as the cross-validation) on the entire dataset. Train for as many epochs and with the same hidden layer structure.

Generally, I prefer the last approach and will retrain a model on the entire data set once I have selected hyper-parameters. Of course, I will always set aside a final holdout set for model validation that I do not use in any aspect of the training process.

Regression vs Classification K-Fold Cross-Validation

Regression and classification are handled somewhat differently concerning cross-validation. Regression is the simpler case where you can break up the data set into K folds with little regard for where each item lands. For regression, the data items should fall into the folds as randomly as possible. It is also important to remember that not every fold will necessarily have the same number of data items. It is not always possible for the data set to be evenly divided into K folds. For regression cross-validation, we will use the Scikit-Learn class **KFold**.

Cross-validation for classification could also use the **KFold** object; however, this technique would not ensure that the class balance remains the same in each fold as in the original. The balance of classes that a model was trained on must remain the same (or similar) to the training set. Drift in this distribution is one of the most important things to monitor after a trained model has been placed into actual use. Because of this, we want to make sure that the cross-validation itself does not introduce an unintended shift. This technique is called stratified sampling and is accomplished by using the Scikit-Learn object **StratifiedKFold** in place of **KFold** whenever you use classification. In summary, you should use the following two objects in Scikit-Learn:

- **KFold** When dealing with a regression problem.
- **StratifiedKFold** When dealing with a classification problem.

The following two sections demonstrate cross-validation with classification and regression.

Out-of-Sample Regression Predictions with K-Fold Cross-Validation

The following code trains the simple dataset using a 5-fold cross-validation. The expected performance of a neural network of the type trained here would be the score for the generated out-of-sample predictions. We begin by preparing a feature vector using the **jh-simple-dataset** to predict age. This model is set up as a regression problem.

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

# 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)

# Generate dummies for product
df = pd.concat([df,pd.get_dummies(df['product'],prefix="product")],axis=1)
df.drop('product', 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['subscriptions'] = zscore(df['subscriptions'])

# Convert to numpy - Classification
x_columns = df.columns.drop('age').drop('id')
x = df[x_columns].values
y = df['age'].values
```

Now that the feature vector is created a 5-fold cross-validation can be performed to generate out-of-sample predictions. We will assume 500 epochs and not use early stopping. Later we will see how we can estimate a more optimal epoch count.

In [4]:
```
EPOCHS=500

import pandas as pd
import os
import numpy as np
```

```
from sklearn import metrics
from scipy.stats import zscore
from sklearn.model_selection import KFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

# Cross-Validate
kf = KFold(5, shuffle=True, random_state=42) # Use for KFold classification
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]

    model = Sequential()
    model.add(Dense(20, input_dim=x.shape[1], activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')

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

    pred = model.predict(x_test)

    oos_y.append(y_test)
    oos_pred.append(pred)

    # Measure this fold's RMSE
    score = np.sqrt(metrics.mean_squared_error(pred,y_test))
    print(f"Fold score (RMSE): {score}")

# Build the oos prediction list and calculate the error.
oos_y = np.concatenate(oos_y)
oos_pred = np.concatenate(oos_pred)
score = np.sqrt(metrics.mean_squared_error(oos_pred,oos_y))
print(f"Final, out of sample score (RMSE): {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 (RMSE): 0.6814299426511208
Fold #2
Fold score (RMSE): 0.45486513719487165
Fold #3
Fold score (RMSE): 0.571615041876392
Fold #4
Fold score (RMSE): 0.46416356081116916
Fold #5
Fold score (RMSE): 1.0426518491685475
Final, out of sample score (RMSE): 0.678316077597408
```

As you can see, the above code also reports the average number of epochs needed. A common technique is to then train on the entire dataset for the average number of epochs required.

Classification with Stratified K-Fold Cross-Validation

The following code trains and fits the **jh**-simple-dataset dataset with cross-validation to generate out-of-sample. It also writes the out-of-sample (predictions on the test set) results.

It is good to perform stratified k-fold cross-validation with classification data. This technique ensures that the percentages of each class remain the same across all folds. Use the **StratifiedKFold** object instead of the **KFold** object used in the regression.

In [5]:
```
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
```

We will assume 500 epochs and not use early stopping. Later we will see how we can estimate a more optimal epoch count.

In [6]:
```
import pandas as pd
import os
import numpy as np
from sklearn import metrics
from sklearn.model_selection import StratifiedKFold
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

# np.argmax(pred,axis=1)
# Cross-validate
# Use for StratifiedKFold classification
kf = StratifiedKFold(5, shuffle=True, random_state=42)

oos_y = []
oos_pred = []
fold = 0

# Must specify y StratifiedKFold for
for train, test in kf.split(x,df['product']):
    fold+=1
    print(f"Fold #{fold}")

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

    model = Sequential()
    # Hidden 1
    model.add(Dense(50, input_dim=x.shape[1], activation='relu'))
    model.add(Dense(25, activation='relu')) # Hidden 2
    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=EPOCHS)

    pred = model.predict(x_test)

    oos_y.append(y_test)
    # raw probabilities to chosen class (highest probability)
```

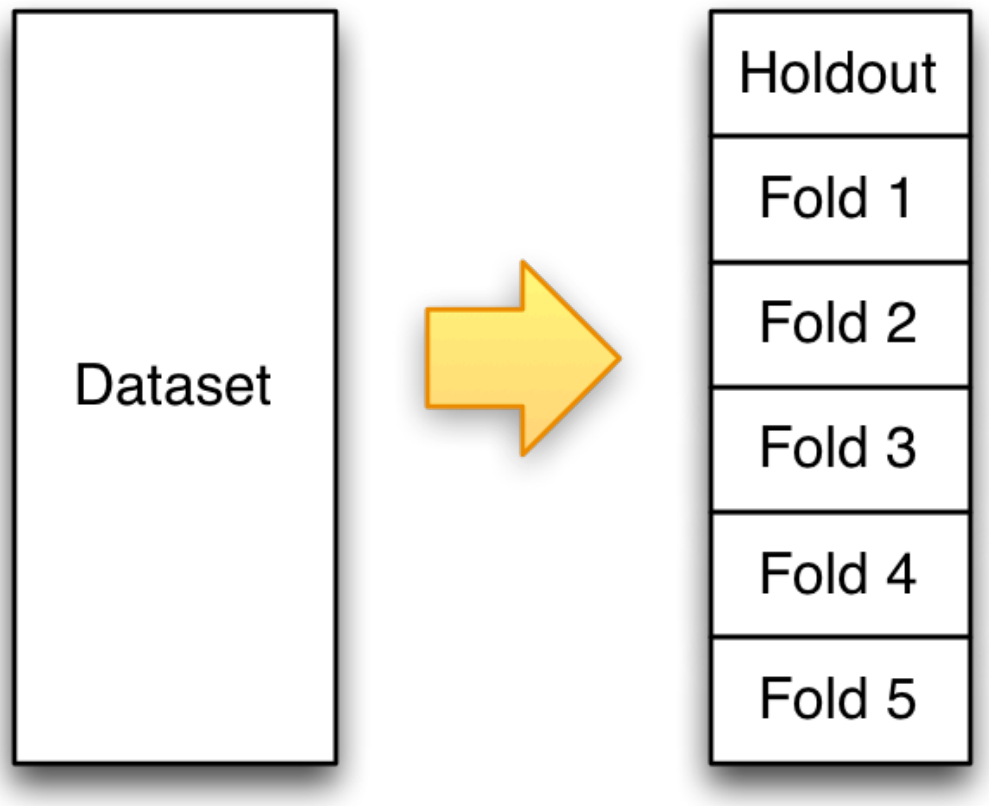
```
    pred = np.argmax(pred,axis=1)
    oos_pred.append(pred)

    # 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): 0.6325
Fold #2
Fold score (accuracy): 0.6725
Fold #3
Fold score (accuracy): 0.6975
Fold #4
Fold score (accuracy): 0.6575
Fold #5
Fold score (accuracy): 0.675
Final score (accuracy): 0.667
```

Training with both a Cross-Validation and a Holdout Set

If you have a considerable amount of data, it is always valuable to set aside a holdout set before you cross-validate. This holdout set will be the final evaluation before using your model for its real-world use. Figure 5. HOLDOUT shows this division.

Figure 5. HOLDOUT: Cross-Validation and a Holdout Set



The following program uses a holdout set and then still cross-validates.

In [7]:
```
import pandas as pd
from scipy.stats import zscore
from sklearn.model_selection import train_test_split

# 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)

# Generate dummies for product
df = pd.concat([df,pd.get_dummies(df['product'],prefix="product")],axis=1)
df.drop('product', 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['subscriptions'] = zscore(df['subscriptions'])

# Convert to numpy - Classification
x_columns = df.columns.drop('age').drop('id')
x = df[x_columns].values
y = df['age'].values
```

Now that the data has been preprocessed, we are ready to build the neural network.

In [8]:
```
from sklearn.model_selection import train_test_split
import pandas as pd
import os
import numpy as np
from sklearn import metrics
from scipy.stats import zscore
from sklearn.model_selection import KFold

# Keep a 10% holdout
x_main, x_holdout, y_main, y_holdout = train_test_split(
    x, y, test_size=0.10)


# Cross-validate
kf = KFold(5)

oos_y = []
oos_pred = []
fold = 0
for train, test in kf.split(x_main):
    fold+=1
    print(f"Fold #{fold}")

    x_train = x_main[train]
    y_train = y_main[train]
    x_test = x_main[test]
    y_test = y_main[test]

    model = Sequential()
    model.add(Dense(20, input_dim=x.shape[1], activation='relu'))
    model.add(Dense(5, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')

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

    pred = model.predict(x_test)
```

```
    oos_y.append(y_test)
    oos_pred.append(pred)

    # Measure accuracy
    score = np.sqrt(metrics.mean_squared_error(pred,y_test))
    print(f"Fold score (RMSE): {score}")


# Build the oos prediction list and calculate the error.
oos_y = np.concatenate(oos_y)
oos_pred = np.concatenate(oos_pred)
score = np.sqrt(metrics.mean_squared_error(oos_pred,oos_y))
print()
print(f"Cross-validated score (RMSE): {score}")

# Write the cross-validated prediction (from the last neural network)
holdout_pred = model.predict(x_holdout)

score = np.sqrt(metrics.mean_squared_error(holdout_pred,y_holdout))
print(f"Holdout score (RMSE): {score}")
```

```
Fold #1
Fold score (RMSE): 0.544195299216696
Fold #2
Fold score (RMSE): 0.48070599342910353
Fold #3
Fold score (RMSE): 0.7034584765928998
Fold #4
Fold score (RMSE): 0.5397141785190473
Fold #5
Fold score (RMSE): 24.126205213080077

Cross-validated score (RMSE): 10.801732731207947
Holdout score (RMSE): 24.097657947297677
```

In [ ]: