Dropout Regularization Sonar Project 4 for the Navy: Mines vs. Rocks

Total Marks: 100 points
Project Supervisor: Mr. Inam Ul Haq, CTO Zaavia

A simple and powerful regularization technique for neural networks and deep learning models is dropout.

In this project you will discover the dropout regularization technique and how to apply it to your models in Python with Keras.

After doing this project you will know:

- How the dropout regularization technique works.
- How to use dropout on your input layers.
- How to use dropout on your hidden layers.
- How to tune the dropout level on your problem.

Step 1. Description of the Dataset

The dataset we will use in this tutorial is the Sonar dataset.

This is a dataset that describes sonar chirp returns bouncing off different services. The 60 input variables are the strength of the returns at different angles. It is a binary classification problem that requires a model to differentiate rocks from metal cylinders.

You can learn more about this dataset on the UCI Machine Learning repository: https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)

We have downloaded the dataset for free and placed it in the project directory with the filename sonar.csv. You can also directly download the dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data

It is a well-understood dataset. All of the variables are continuous and generally in the range of 0 to 1. The output variable is a string “M” for mine and “R” for rock, which will need to be converted to integers 1 and 0.

A benefit of using this dataset is that it is a standard benchmark problem. This means that we have some idea of the expected skill of a good model. Using cross-validation, a neural network should be able to achieve performance around 84% with an upper bound on accuracy for custom models at around 88%.

Step 2: Dropout Regularization For Neural Networks

Dropout is a regularization technique for neural network models proposed by Srivastava, et al. in their 2014 paper Dropout:

A Simple Way to Prevent Neural Networks from Overfitting:
http://jmlr.org/papers/v15/srivastava14a.html

You can download the PDF from here:
http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass.

As a neural network learns, neuron weights settle into their context within the network. Weights of neurons are tuned for specific features providing some specialization. Neighboring neurons become to rely on this specialization, which if taken too far can result in a fragile model too specialized to the training data. This reliant on context for a neuron during training is referred to complex co-adaptations.

You can imagine that if neurons are randomly dropped out of the network during training, that other neurons will have to step in and handle the representation required to make predictions for the missing neurons. This is believed to result in multiple independent internal representations being learned by the network.

The effect is that the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.

Step 3: Dropout Regularization in Keras
Dropout is easily implemented by randomly selecting nodes to be dropped-out with a given probability (e.g. 20%) each weight update cycle. This is how Dropout is implemented in Keras. Dropout is only used during the training of a model and is not used when evaluating the skill of the model.

Next we will explore a few different ways of using Dropout in Keras.

As stated above the project will use the Sonar dataset. This is a binary classification problem where the objective is to correctly identify rocks and mock-mines from sonar chirp returns. It is a good test dataset for neural networks because all of the input values are numerical and have the same scale.

We will evaluate the developed models using scikit-learn with 10-fold cross validation, in order to better tease out differences in the results.

There are 60 input values and a single output value and the input values are standardized before being used in the network. The baseline neural network model has two hidden layers, the first with 60 units and the second with 30. Stochastic gradient descent is used to train the model with a relatively low learning rate and momentum.

The full baseline model is listed below.

```
# Baseline Model on the Sonar Dataset
import numpy
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.wrappers.scikit_learn import KerasClassifier
from keras.constraints import maxnorm
from keras.optimizers import SGD
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load dataset
dataframe = read_csv("sonar.csv", header=None)
dataset = dataframe.values
# split into input (X) and output (Y) variables
X = dataset[:,0:60].astype(float)
Y = dataset[:,60]
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(Y)
encoded_Y = encoder.transform(Y)

# baseline
def create_baseline():
	# create model
	model = Sequential()
	model.add(Dense(60, input_dim=60, kernel_initializer='normal', activation='relu'))
	model.add(Dense(30, kernel_initializer='normal', activation='relu'))
	model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
	# Compile model
	sgd = SGD(lr=0.01, momentum=0.8, decay=0.0, nesterov=False)
	model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
	return model

numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_baseline, epochs=300, batch_size=16,
verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Running the example generates an estimated classification accuracy of 86%.

Baseline: 86.04% (4.58%)

Step 4: Using Dropout on the Visible Layer
Dropout can be applied to input neurons called the visible layer.

In the code below we will add a new Dropout layer between the input (or visible layer) and the first hidden layer. The dropout rate is set to 20%, meaning one in 5 inputs will be randomly excluded from each update cycle.

Additionally, as recommended in the original paper on Dropout, a constraint will be imposed on the weights for each hidden layer, ensuring that the maximum norm of the weights does not exceed a value of 3. This is done by setting the kernel_constraint argument on the Dense class when constructing the layers.

The learning rate was lifted by one order of magnitude and the momentum was increase to 0.9. These increases in the learning rate were also recommended in the original Dropout paper.

Continuing on from the baseline code above, in the code below exercise the same network with input dropout.

```
# dropout in the input layer with weight constraint
def create_model():
        # create model, insert code here



        # Compile model
        sgd = SGD(lr=0.1, momentum=0.9, decay=0.0, nesterov=False)
        model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
        return model

numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_model, epochs=300, batch_size=16,
verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
print("Visible: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

Running the example provides a small drop in classification accuracy, at least on a single test run.

Visible: 83.52% (7.68%)

Step 5: Trying to Improve Performance
It is possible that additional training epochs are required or that further tuning is required to the learning rate. Please try to make these changes in the model developed in step 4 and see if the performance improves.

Step 6: Using Dropout on Hidden Layers
Dropout can be applied to hidden neurons in the body of your network model.

In the code below Dropout will be applied between the two hidden layers and between the last hidden layer and the output layer. Again a dropout rate of 20% is used as is a weight constraint on those layers.

```
# dropout in hidden layers with weight constraint
def create_model():
        # create model, insert code here



        # Compile model
        sgd = SGD(lr=0.1, momentum=0.9, decay=0.0, nesterov=False)
        model.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])
        return model

numpy.random.seed(seed)
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasClassifier(build_fn=create_model, epochs=300, batch_size=16,
verbose=0)))
pipeline = Pipeline(estimators)
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
results = cross_val_score(pipeline, X, encoded_Y, cv=kfold)
print("Hidden: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

We can see that for this project and for the chosen network configuration that using dropout in the hidden layers did not lift performance. In fact, performance is worse than the baseline.

It is possible that additional training epochs are required or that further tuning is required to the learning rate.

Hidden: 83.59% (7.31%)

Step 7: Trying to Improve Performance
It is possible that additional training epochs are required or that further tuning is required to the learning rate. Please try to make these changes in the model developed in step 6 and see if the performance improves.

Step 8: Tips For Using Dropout
The original paper on Dropout provides experimental results on a suite of standard machine learning problems. As a result they provide a number of useful heuristics to consider when using dropout in practice. Please try out the following mini-steps:

Step 8.1: Try Different Dropout values
Generally, use a small dropout value of 20%-50% of neurons with 20% providing a good starting point. A probability too low has minimal effect and a value too high results in under-learning by the network.

Step 8.2: Try using a Larger network
Use a larger network. You are likely to get better performance when dropout is used on a larger network, giving the model more of an opportunity to learn independent representations.

Step 8.3: Try using Dropout on both visible and hidden units
Use dropout on incoming (visible) as well as hidden units. Application of dropout at each layer of the network has shown good results.

Step 8.4: Try using large learning rate with decay and larger momentum
Use a large learning rate with decay and a large momentum. Increase your learning rate by a factor of 10 to 100 and use a high momentum value of 0.9 or 0.99.

Step 8.5: Try constraining the size of the network weights
Constrain the size of network weights. A large learning rate can result in very large network weights. Imposing a constraint on the size of network weights such as max-norm regularization with a size of 4 or 5 has been shown to improve results.

Step 9: Read More Resources on Dropout, and try implementing them
Below are some resources that you can use to learn more about dropout in neural network and deep learning models.

Dropout: A Simple Way to Prevent Neural Networks from Overfitting (original paper).
http://jmlr.org/papers/v15/srivastava14a.html

Improving neural networks by preventing co-adaptation of feature detectors.
https://arxiv.org/abs/1207.0580

How does the dropout method work in deep learning? on Quora.
https://www.quora.com/How-does-the-dropout-method-work-in-deep-learning-And-why-is-it-claimed-to-be-an-effective-trick-to-improve-your-network

What did you achieve in this Project
In this project, you discovered the dropout regularization technique for deep learning models. You learned:

- What dropout is and how it works.
- How you can use dropout on your own deep learning models.
- Tips for getting the best results from dropout on your own models.