**Dropout Regularization in Deep Learning**

* 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.

**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) ([download the PDF](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 come to rely on this specialization, which, if taken too far, can result in a fragile model too specialized for the training data. This reliance on context for a neuron during training is referred to as complex co-adaptations.

You can imagine that if neurons are randomly dropped out of the network during training, 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 capable of better generalization and less likely to overfit the training data.

## Dropout Regularization in Keras

Dropout is easily implemented by randomly selecting nodes to be dropped out with a given probability (e.g., 20%) in 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, let’s explore a few different ways of using Dropout in Keras.

The examples will use the [Sonar dataset](http://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)). This is a binary classification problem that aims to correctly identify rocks and mock-mines from sonar chirp returns. It is a good test dataset for neural networks because all the input values are numerical and have the same scale.

The dataset can be [downloaded from the UCI Machine Learning repository](http://archive.ics.uci.edu/ml/machine-learning-databases/undocumented/connectionist-bench/sonar/sonar.all-data). You can place the sonar dataset in your current working directory with the file name sonar.csv.

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

There are 60 input values and a single output value. 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  from pandas import read\_csv  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.optimizers import SGD  from scikeras.wrappers import KerasClassifier  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  # 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\_shape=(60,), activation='relu'))  model.add(Dense(30,  activation='relu'))  model.add(Dense(1, activation='sigmoid'))  # Compile model  sgd = SGD(learning\_rate=0.01, momentum=0.8)  model.compile(loss='binary\_crossentropy', optimizer=sgd, metrics=['accuracy'])  return model    estimators = []  estimators.append(('standardize', StandardScaler()))  estimators.append(('mlp', KerasClassifier(model=create\_baseline, epochs=300, batch\_size=16, verbose=0)))  pipeline = Pipeline(estimators)  kfold = StratifiedKFold(n\_splits=10, shuffle=True)  results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)  print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100)) |

**Note**: Your [results may vary](https://machinelearningmastery.com/different-results-each-time-in-machine-learning/) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

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

|  |  |
| --- | --- |
| 1 | Baseline: 86.04% (4.58%) |

## Using Dropout on the Visible Layer

Dropout can be applied to input neurons called the visible layer.

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

Additionall

Additionally, as recommended in the original paper on Dropout, a constraint is 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 increased to 0.9. These increases in the learning rate were also recommended in the original Dropout paper.

Continuing from the baseline example above, the code below exercises the same network with input dropout:

|  |
| --- |
| # Example of Dropout on the Sonar Dataset: Visible Layer  from pandas import read\_csv  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.layers import Dropout  from tensorflow.keras.constraints import MaxNorm  from tensorflow.keras.optimizers import SGD  from scikeras.wrappers import KerasClassifier  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  # 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)    # dropout in the input layer with weight constraint  def create\_model():  # create model  model = Sequential()  model.add(Dropout(0.2, input\_shape=(60,)))  model.add(Dense(60, activation='relu', kernel\_constraint=MaxNorm(3)))  model.add(Dense(30, activation='relu', kernel\_constraint=MaxNorm(3)))  model.add(Dense(1, activation='sigmoid'))  # Compile model  sgd = SGD(learning\_rate=0.1, momentum=0.9)  model.compile(loss='binary\_crossentropy', optimizer=sgd, metrics=['accuracy'])  return model    estimators = []  estimators.append(('standardize', StandardScaler()))  estimators.append(('mlp', KerasClassifier(model=create\_model, epochs=300, batch\_size=16, verbose=0)))  pipeline = Pipeline(estimators)  kfold = StratifiedKFold(n\_splits=10, shuffle=True)  results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)  print("Visible: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100)) |

**Note**: Your [results may vary](https://machinelearningmastery.com/different-results-each-time-in-machine-learning/) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

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

|  |  |
| --- | --- |
| 1 | Visible: 83.52% (7.68%) |

**Using Dropout on Hidden Layers**

Dropout can be applied to hidden neurons in the body of your network model.

In the example below, Dropout is 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.

|  |
| --- |
| # Example of Dropout on the Sonar Dataset: Hidden Layer  from pandas import read\_csv  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense  from tensorflow.keras.layers import Dropout  from tensorflow.keras.constraints import MaxNorm  from tensorflow.keras.optimizers import SGD  from scikeras.wrappers import KerasClassifier  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  # 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)    # dropout in hidden layers with weight constraint  def create\_model():  # create model  model = Sequential()  model.add(Dense(60, input\_shape=(60,), activation='relu', kernel\_constraint=MaxNorm(3)))  model.add(Dropout(0.2))  model.add(Dense(30, activation='relu', kernel\_constraint=MaxNorm(3)))  model.add(Dropout(0.2))  model.add(Dense(1, activation='sigmoid'))  # Compile model  sgd = SGD(learning\_rate=0.1, momentum=0.9)  model.compile(loss='binary\_crossentropy', optimizer=sgd, metrics=['accuracy'])  return model    estimators = []  estimators.append(('standardize', StandardScaler()))  estimators.append(('mlp', KerasClassifier(model=create\_model, epochs=300, batch\_size=16, verbose=0)))  pipeline = Pipeline(estimators)  kfold = StratifiedKFold(n\_splits=10, shuffle=True)  results = cross\_val\_score(pipeline, X, encoded\_Y, cv=kfold)  print("Hidden: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100)) |

**Note**: Your [results may vary](https://machinelearningmastery.com/different-results-each-time-in-machine-learning/) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

You can see that for this problem and the chosen network configuration, using Dropout in the hidden layers did not lift performance. In fact, performance was worse than the baseline.

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

|  |  |
| --- | --- |
| 1 | Hidden: 83.59% (7.31%) |

**Dropout in Evaluation Mode**

Dropout will randomly reset some of the input to zero. If you wonder what happens after you have finished training, the answer is nothing! In Keras, a layer can tell if the model is running in training mode or not. The Dropout layer will randomly reset some input only when the model runs for training. Otherwise, the Dropout layer works as a scaler to multiply all input by a factor such that the next layer will see input similar in scale. Precisely, if the dropout rate is �, the input will be scaled by a factor of 1−�.

**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.

* 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.
* 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.
* Use Dropout on incoming (visible) as well as hidden units. Application of Dropout at each layer of the network has shown good results.
* 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.
* 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.