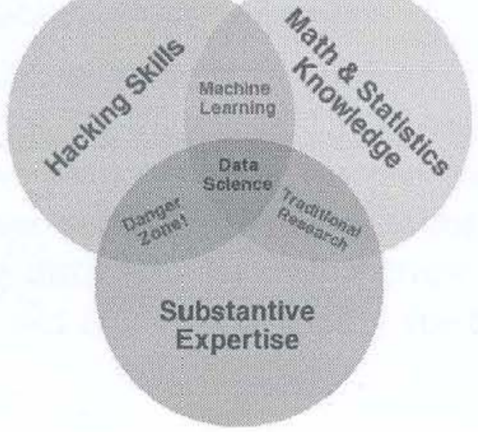
# Chapter 16

# Modeling with Neural Networks

- Data Science
- Kaggle
- Ensemble Learning

In this chapter, we present a capstone project on modeling, a business-oriented approach for artificial intelligence, and some aspects of data science. Drew Conway (2013), a leading data scientist, characterizes data science as the intersection of hacking skills, math and statistics knowledge, and substantive expertise. Figure 16.1 depicts this definition:

**Figure 16.1:** Conway's Data Science Venn Diagram



Hacking skills are essentially a subset of computer programming. Although the data scientist does not necessarily need the infrastructure knowledge of an information technology (IT) professional, these technical skills will permit him or her to create short, effective programs for processing data. In the field of data science, we refer to information processing as data wrangling.

Math and statistics knowledge covers statistics, probability, and other inferential methods. Substantive knowledge describes the business knowledge as well as the comprehension of actual data. If you combine only two of these topics, you don't have all the components for data science, as Figure 16.1 illustrates. In other words, the combination of statistics and substantive expertise is simply traditional research. Those two skills alone don't encompass the capabilities, such as machine learning, required for data science.

This book series deals with hacking skills and math and statistical knowledge, two of the circles in Figure 16.1. Additionally, it teaches you to create your own models, which is more pertinent to the field of computer science than data science. Substantive expertise is more difficult to obtain because it is dependent on the industry that utilizes the data science applications. For example, if you want to apply data science in the insurance industry, substantive knowledge refers to the actual business operations of these companies.

To provide a data science capstone project, we will use the Kaggle Otto Group Product Classification Challenge. Kaggle is a platform for competitive data science. You can find the Otto Group Product Classification Challenge at the following URL:

**https://www.kaggle.com/c/otto-group-product-classification-challenge**

The Otto Group was the first (and currently only) non-tutorial Kaggle competition in which we've competed. After obtaining a top 10% finish, we achieved one of the criteria for the Kaggle Master designation. To become a Kaggle Master, one must place in the top 10 of a competition once and in the top 10% of two other competitions. Figure 16.2 shows the results of our competition entry on the leaderboard:

**Figure 16.2:** Results in the Otto Group Product Classification Challenge

331 ↓3 Jeff Heaton 0.42881 52 Mon, 18 May 2015 14:34:37

The above line shows several pieces of information.

- We were in position 331 of 3514 (9.4%).
- We dropped three spots in the final day.
- Our multi-class log loss score was 0.42881.
- We made 52 submissions, up to May 18, 2015.

We will briefly describe the Otto Group Product Classification Challenge. For a complete description, refer to the Kaggle challenge website (found above). The Otto Group, the world's largest mail order company and currently one of the biggest e-commerce companies, introduced this challenge. Because the group has many products sold over numerous countries, they wanted to classify these products into nine categories with 93 features (columns). These 93 columns represented counts and were often 0.

The data were completely redacted (hidden). The competitors did not know the nine categories nor did they know the meaning behind the 93 features. They knew only that the features were integer counts. Like most Kaggle competitions, this challenge provided the competitors with a test and training data set. For the training data set, the competitors received the outcomes, or correct answers. For the test set, they got only the 93 features, and they had to provide the outcome.

The competition divided the test and training sets in the following way:

- Test Data: 144K rows
- Training Data: 61K rows

During the competition, participants did not submit their actual models to Kaggle. Instead, they submitted their model's predictions based on the test data. As a result, they could have used any platform to make these predictions. For this competition there were nine categories, so the competitors submitted a nine-number vector that held the probability of each of these nine categories being the correct answer.

The answer in the vector that held the highest probability was the chosen class. As you can observe, this competition was not like a multiple-choice test in school where students must submit their answer as A, B, C, or D. Instead, Kaggle competitors had to submit their answers in the following way:

- A: 80% probability
- B: 16% probability
- C: 2% probability
- D: 2% probability

College exams would not be so horrendous if students could submit answers like those in the Kaggle competition. In many multiple-choice tests, students have confidence about two of the answers and eliminate the remaining two. The Kaggle-like multiple-choice test would allow students to assign a probability to each answer, and they could achieve a partial score. In the above example, if A were the correct answer, students would earn 80% of the points.

Nevertheless, the actual Kaggle score is slightly more complex. The program grades the answers with a logarithm-based scale, and participants face heavy penalties if they have a lower probability on the correct answer. You can see the Kaggle format from the following CSV file submission:

```
1,0.0003,0.2132,0.2340,0.5468,6.2998e
    -05,0.0001,0.0050,0.0001,4.3826e-05
2,0.0011,0.0029,0.0010,0.0003,0.0001,0.5207,0.0013,0.4711,0.0011
3,3.2977e-06,4.1419e-06,7.4524e-06,2.6550e-06,5.0014e
    -07,0.9998,5.2621e-06,0.0001,6.6447e-06
4,0.0001,0.6786,0.3162,0.0039,3.3378e-05,4.1196e
    -05,0.0001,0.0001,0.0006
5,0.1403,0.0002,0.0002,6.734e
    -05,0.0001,0.0027,0.0009,0.0297,0.8255
```

As you can see, each line starts with a number that specifies the data item that is being answered. The sample above shows the answers for items one through five. The next nine values are the probabilities for each of the product classes. These probabilities must add up to 1.0 (100%).

## 16.0.1 Lessons from the Challenge

Having success in Kaggle requires you to understand the following topics and the corresponding tools:

- Deep Learning - Using H2O and Lasagne
- Gradient Boosting Machines (GBM) - Using XGBOOST
- Ensemble Learning - Using NumPy
- Feature Engineering - Using NumPy and Scikit-Learn
- GPU is really important for deep learning. It is best to use a deep learning package that supports it, such as H2O, Theano or Lasagne.
- The t-SNE visualization is awesome for high-dimension visualization and creating features.
- Ensembling is very important.

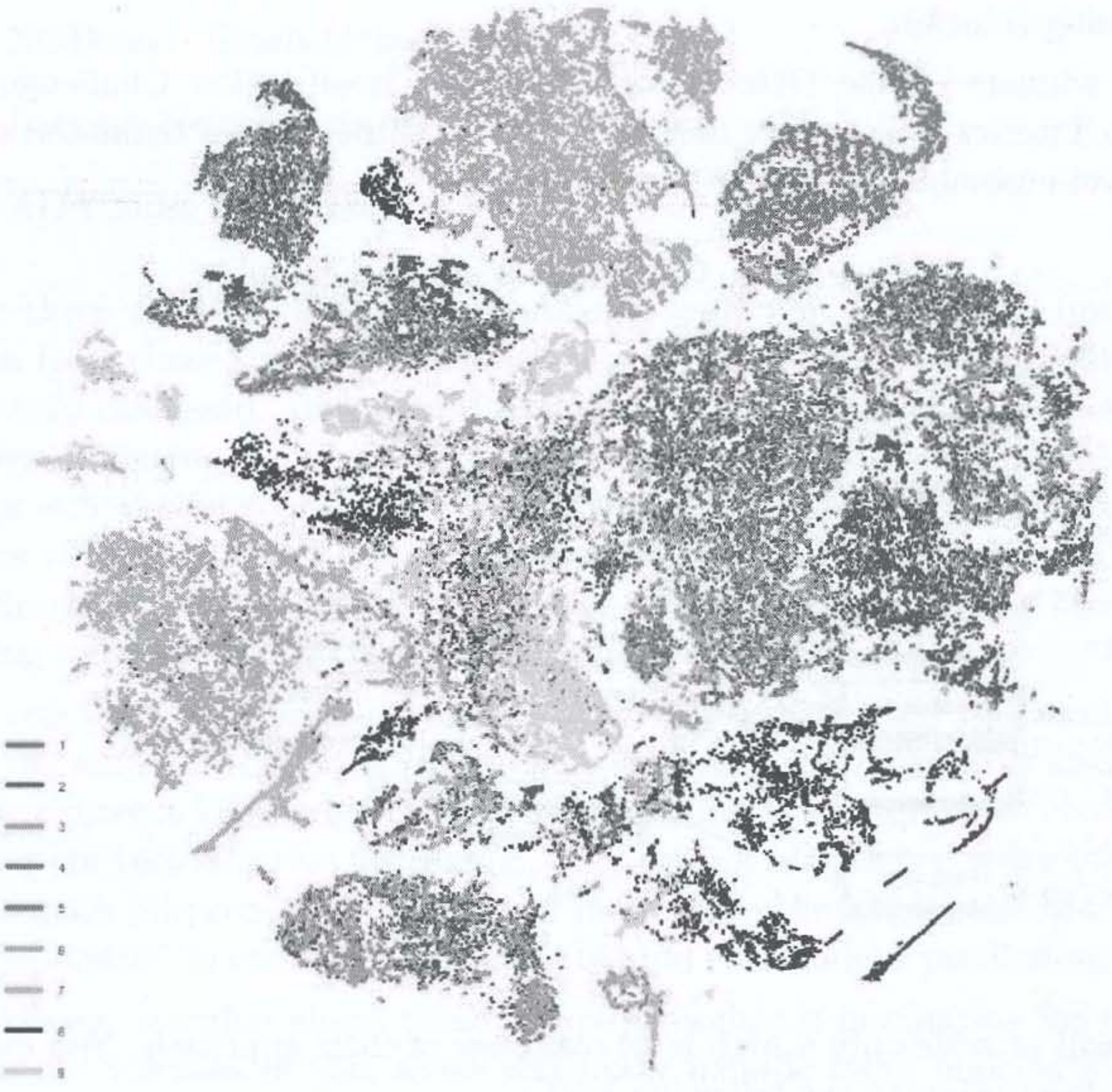
For our submission, we used Python with Scikit-Learn. However, you can use any language capable of generating a CSV file. Kaggle does not actually run your code; they score a submission file. The two most commonly used programming languages for Kaggle are R and Python. Both of these languages have strong data science frameworks available for them. R is actually a domain-specific language (DSL) for statistical analysis.

During this challenge, we learned the most about GBM parameter tuning and ensemble learning. GBMs have quite a few hyper-parameters to tune, and we became proficient at tuning a GBM. The individual scores for our GBMs were in line with those of the top 10% of the teams. However, the solution in this chapter will use only deep learning. GBM is beyond the scope of this book. In a future volume or edition of this series, we plan to examine GBM.

Although computer programmers and data scientists might typically utilize a single model like neural networks, participants in Kaggle need to use multiple models to be successful in the competition. These ensembled models produce better results than each of the models could generate independently.

We worked with t-SNE, examined in Chapter 15, "Visualization," for the first time in this competition. This model works like principal component analysis (PCA) in that it is capable of reducing dimensions. However, the data points separate in such a way that the visualization is often clearer than PCA. The program achieves the clear visualization by using a stochastic nearest neighbor process. Figure 16.3 shows the data from the Otto Group Product Classification Challenge visualized in t-SNE:

**Figure 16.3:** Challenge t-SNE

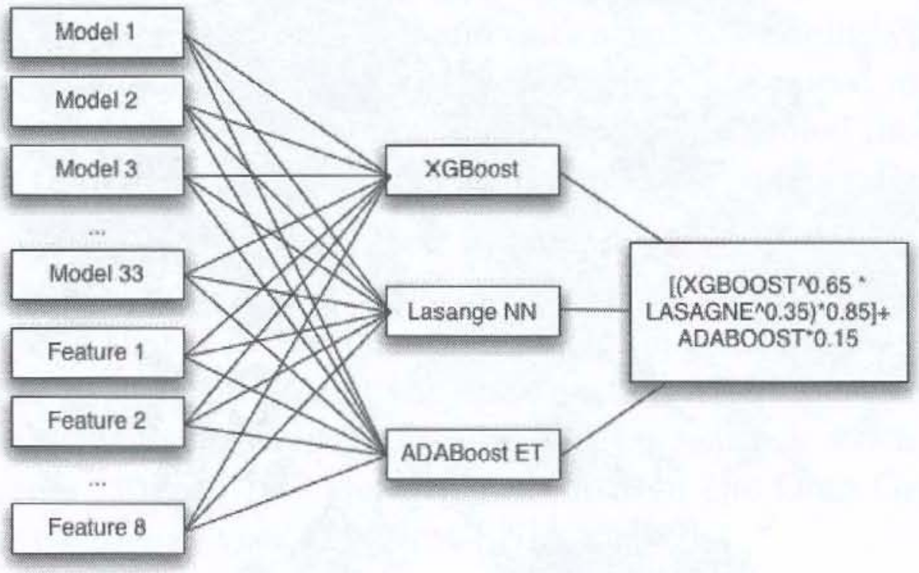


## 16.0.2 The Winning Approach to the Challenge

Kaggle is very competitive. Our primary objective as we entered the challenge was to learn. However, we also hoped to rank in the top 10% by the end in order to reach one of the steps in becoming a Kaggle master. Earning a top 10% was difficult; in the last few weeks of the challenge, other competitors knocked us out of the bracket almost daily. The last three days were especially turbulent. Before we reveal our solution, we will show you the winning one. The following description is based on the information publically posted about the winning solution.

The winners of the Otto Group Product Classification Challenge were Gilberto Titericz & Stanislav Semenov. They competed as a team and used a three-level ensemble, as seen in Figure 16.4:

**Figure 16.4:** Challenge Winning Ensemble



We will provide only a high-level overview of their approach. You can find the full description at the following URL:

**https://goo.gl/fZrJA0**

The winning approach employed both the R and Python programming languages. Level 1 used a total of 33 different models. Each of these 33 models provided its output to three models in level 2. Additionally, the program generated eight calculated features. An engineered feature is one that is calculated based on the others. A simple example of an engineered feature might be body mass index (BMI), which is calculated based on an individual's height and weight. The BMI value provides insights that height and weight alone might not.

The second level combined the following three model types:

- XGBoost - Gradient boosting
- Lasange Neural Network - Deep learning
- ADABoost Extra Trees

These three used the output of 33 models and eight features as input. The output from these three models was the same nine-number probability vector previously discussed. It was as if each model were being used independently, thereby producing a nine-number vector that would have been suitable as an answer submission to Kaggle. The program averaged together these output vectors with the third layer, which was simply a weighting. As you can see, the winners of the challenge used a large and complex ensemble. Most of the winning solutions in Kaggle followed a similar pattern.

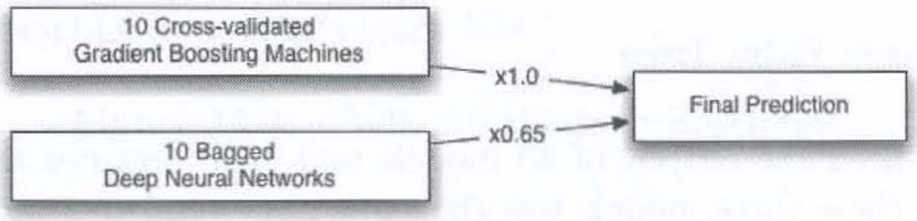
A complete discussion on exactly how they constructed this model is beyond the scope of this book. Quite honestly, such a discussion is also beyond our own current knowledge of ensemble learning. Although these complex ensembles are very effective for Kaggle, they are not always necessary for general data science purposes. These types of models are the blackest of black boxes. It is impossible to explain the reasons behind the model's predictions.

However, learning about these complex models is fascinating for research, and future volumes of this series will likely include more information about these structures.

### 16.0.3 Our Approach to the Challenge

So far, we've worked only with single model systems. These models that contain ensembles that are "built in", such as random forests and gradient boosting machines (GBM). However, it is possible to create higher-level ensembles of these models. We used a total of 20 models, which included ten deep neural networks and ten gradient boosting machines. Our deep neural network system provided one prediction, and the gradient boosting machines provided the other. The program blended these two predictions with a simple ratio. Then we normalized the resulting prediction vector so that the sum equaled 1.0 (100%). Figure 16.5 shows the ensemble model:

**Figure 16.5:** Our Challenge Group Entry


10 Cross-validated Gradient Boosting Machines
x1.0
Final Prediction
x0.65
10 Bagged Deep Neural Networks

You can find our entry, written in Python, at the following URL:

**https://github.com/jeffheaton/kaggle-otto-group**

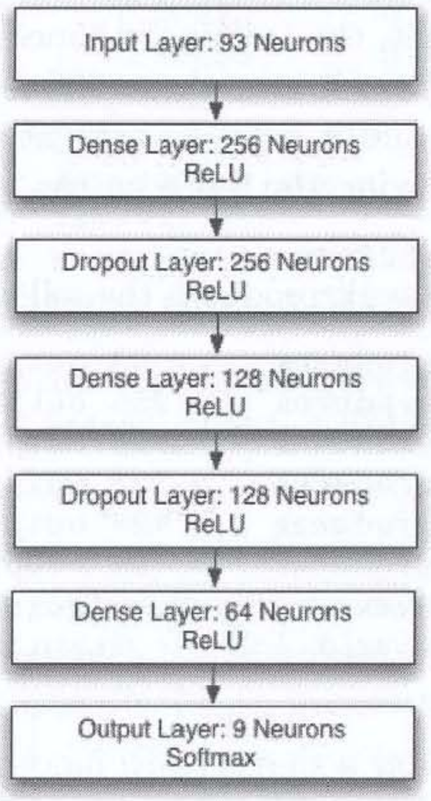
## 16.1 Modeling with Deep Learning

To stay within the scope of this book, we will present a solution to the Kaggle competition based on our entry. Because gradient boosting machines (GBM) are beyond the subject matter of this book, we will focus on using a deep neural network. To introduce ensemble learning, we will use bagging to combine ten trained neural networks together. Ensemble methods, such as bagging, will usually cause the aggregate of ten neural networks to score better than a single neuron. If you would like to use gradient boosting machines and replicate our solution, see the link provided above for the source code.

### 16.1.1 Neural Network Structure

For this neural network, we used a deep learning structure composed of dense layers and dropout layers. Because this structure was not an image network, we did not use convolutional layers or max-pool layers. These layer types required that input neurons in close proximity have some relevance to each other. However, the 93 input values that comprised the data set might not have been relevant. Figure 16.6 shows the structure of the deep neural network:

**Figure 16.6:** Deep Neural Network for the Challenge


Input Layer: 93 Neurons
Dense Layer: 256 Neurons ReLU
Dropout Layer: 256 Neurons ReLU
Dense Layer: 128 Neurons ReLU
Dropout Layer: 128 Neurons ReLU
Dense Layer: 64 Neurons ReLU
Output Layer: 9 Neurons Softmax

As you can see, the input layer of the neural network had 93 neurons that corresponded to the 93 input columns in the data set. Three hidden layers had 256, 128 and 64 neurons each. Additionally, two dropout layers each had layers of 256 and 128 neurons and a dropout probability of 20%. The output was a softmax layer that classified the nine output groups. We normalized the input data to the neural network to take their z-scores.

Our strategy was to use two dropout layers tucked between three dense layers. We chose a power of 2 for the first dense layer. In this case we used 2 to the power of 8 (256). Then we divided by 2 to obtain each of the next two dense layers. This process resulted in 256, 128 and then 64. The pattern of using a power of 2 for the first layer and two more dense layers dividing by 2, worked well. As the experiments continued, we tried other powers of 2 in the first dense layer.

We trained the network with stochastic gradient descent (SGD). The program divided the training data into a validation set and a training set. The SGD training used only the training data set, but it monitored the validation set's error. We trained until our validation set's error did not improve for 200 iterations. At this point, the training stopped, and the program selected the best-trained neural network over those 200 iterations. We refer to this process as early stopping, and it helps to prevent overfitting. When a neural network is no longer improving the score on the validation set, overfitting is likely occurring.

Running the neural network produces the following output:

```
Input      (None, 93)  produces    93 outputs
dense0     (None, 256) produces   256 outputs
dropout0   (None, 256) produces   256 outputs
dense1     (None, 128) produces   128 outputs
dropout1   (None, 128) produces   128 outputs
dense2     (None, 64)  produces    64 outputs
output     (None, 9) roduces     9 outputs
epoch    train loss    valid loss    train/val    valid acc
-------  ------------  ------------  -----------  -----------
      1       1.07019       0.71004      1.50723      0.73697
      2       0.78002       0.66415      1.17447      0.74626
      3       0.72560       0.64177      1.13061      0.75000
      4       0.70295       0.62789      1.11955      0.75353
      5       0.67780       0.61759      1.09750      0.75724
...
    410       0.40410       0.50785      0.79572      0.80963
    411       0.40876       0.50930      0.80260      0.80645
Early stopping.
Best valid loss was 0.495116 at epoch 211.
Wrote submission to file las-submit.csv.
Wrote submission to file las-val.csv.
Bagged LAS model: 1, score: 0.49511558950601003, current mlog:
   0.379456064667434, bagged mlog: 0.379456064667434
Early stopping.
Best valid loss was 0.502459 at epoch 221.
Wrote submission to file las-submit.csv.
Wrote submission to file las-val.csv.
Bagged LAS model: 2, score: 0.5024587499599558, current mlog:
    0.38050303230483773, bagged mlog: 0.3720715012362133
  epoch     train loss      valid loss      train/val       valid acc
  -------  -------------  -------------  --------------  -------------
      1      1.07071          0.70542        1.51785         0.73658
      2      0.77458          0.66499        1.16479         0.74670
...
    370      0.41459          0.50696        0.81779         0.80760
    371      0.40849          0.50873        0.80296         0.80642
    372      0.41383          0.50855        0.81376         0.80787
Early stopping.
Best valid loss was 0.500154 at epoch 172.
Wrote submission to file las-submit.csv.
Wrote submission to file las-val.csv.
Bagged LAS model: 3, score: 0.5001535314594113, current mlog:
    0.3872396776865103, bagged mlog: 0.3721509601621992
...
Bagged LAS model: 4, score: 0.4984386022067697, current mlog:
    0.39710688423724777, bagged mlog: 0.37481605169768967
...
```

In general, the neural network gradually decreases its training and validation error. If you run this example, you might see different output, based on the programming language from which the example originates. The above output is from Python and the Lasange/NoLearn frameworks.

It is important to understand why there is a validation error and a training error. Most neural network training algorithms will separate the training set into a training and validation set. This split might be 80% for training and 20% for validation. The neural network will use the 80% to train, and then it reports that error as the training error. You can also use the validation set to generate an error, which is the validation error. Because it represents the error on the data that are not trained with the neural network, the validation error is the most important measure. As the neural network trains, the training error will continue to drop even if the neural network is overfitting. However, once the validation error stops dropping, the neural network is probably beginning to overfit.

## 16.1.2 Bagging Multiple Neural Networks

Bagging is a simple yet effective method to ensemble multiple models together. The example program for this chapter trains ten neural networks independently. Each neural network will produce its own set of nine probabilities that correspond to the nine classes provided by Kaggle. Bagging simply takes the average of each of these nine Kaggle-provided classes. Listing 16.1 provides the pseudocode to perform the bagging:

**Listing 16.1:** Bagging Neural Network

```
# Final results is a matrix with rows = to rows in training set
# Columns = number of outcomes (1 for regression, or class count
   for classification)
final_results = [][]
for i from 1 to 5:
   network = train_neural_network()
   results = evaluate_network(network)
   final_results = final_results + results

# Take the average
final_weights = weights / 5
```

We performed the bagging on the test data set provided by Kaggle. Although the test provided the 93 columns, it did not tell us the classes that it supplied. We had to produce a file that contained the ID of the item for which we were answering and then the nine probabilities. On each row, the probabilities should sum to 1.0 (100%). If we submitted a file that did not sum to 1.0, Kaggle would have scaled our values so that they did sum to 1.0.

To see the effects of bagging, we submitted two test files to Kaggle. The first test file was the first neural network that we trained. The second test file was the bagged average of all ten. The results were as follows:

- Best Single Network: 0.3794
- Five Bagged Networks: 0.3717

As you can see, the bagged networks achieved a better score than a single neural network. The complete results are shown here:

```
Bagged LAS model: 1, score: 0.4951, current mlog: 0.3794, bagged
    mlog: 0.3794
Bagged LAS model: 2, score: 0.5024, current mlog: 0.3805, bagged
    mlog: 0.3720
Bagged LAS model: 3, score: 0.5001, current mlog: 0.3872, bagged
    mlog: 0.3721
Bagged LAS model: 4, score: 0.4984, current mlog: 0.3971, bagged
    mlog: 0.3748
Bagged LAS model: 5, score: 0.4979, current mlog: 0.3869, bagged
    mlog: 0.3717
```

As you can see, the first neural network had a multi-class log loss (mlog) error of 0.3794. The mlog measure was discussed in Chapter 5, "Training & Evaluation." The bagged score was the same because we had only one network. The amazing part happens when we bagged the second network to the first. The current scores of the first two networks were 0.3794 and 0.3804. However, when we bagged them together, we had 0.3720, which was lower than both networks. Averaging the weights of these two networks produced a new network that was better than both. Ultimately, we settled on a bagged score of 0.3717, which was better than any of the previous single network (current) scores.

## 16.2 Chapter Summary

In the final chapter of this book, we showed how to apply deep learning to a real-world problem. We trained a deep neural network to produce a submission file for the Kaggle Otto Group Product Classification Challenge. We used dense and dropout layers to create this neural network.

We can utilize ensembles to combine several models into one. Usually, the resulting ensemble model will achieve better scores than the individual ensemble methods. We also examined how to bag ten neural networks together and generate a Kaggle submission CSV.

After analyzing neural networks and deep learning in this final chapter as well as the previous chapters, we hope that you have learned new and useful information. If you have any comments about this volume, we would love to hear from you. In the future, we plan to create additional editions of the volumes to include more technologies. Therefore, we would be interested in discovering your preferences on the technologies that you would like us to explore in future editions. You can contact us through the following website:

**http://www.jeffheaton.com**