Chapter 5

Training & Evaluation

- Mean Squared Error
- Sensitivity & Specificity
- ROC Curve
- Simulated Annealing

So far we've seen how to calculate a neural network based on its weights; however, we have not seen where these weight values actually come from. Training is the process where a neural network's weights are adjusted to produce the desired output. Training uses evaluation, which is the process where the output of the neural network is evaluated against the expected output.

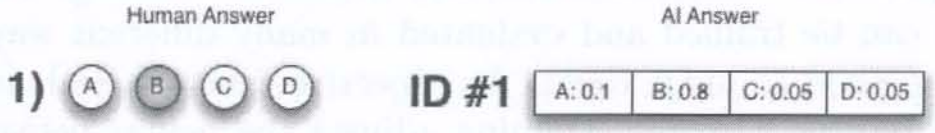
This chapter will cover evaluation and introduce training. Because neural networks can be trained and evaluated in many different ways, we need a consistent method to judge them. An objective function evaluates a neural network and returns a score. Training adjusts the neural network in ways that might achieve better results. Typically, the objective function wants lower scores. The process of attempting to achieve lower scores is called minimization. You might establish maximization problems, in which the objective function wants higher scores. Therefore, you can use most training algorithms for either minimization or maximization problems.

You can optimize weights of a neural network with any continuous optimization algorithm, such as simulated annealing, particle swarm optimization, genetic algorithms, hill climbing, Nelder-Mead, or random walk. In this chapter, we will introduce simulated annealing as a simple training algorithm. However, in addition to optimization algorithms, you can train neural networks with backpropagation. Chapter 6, "Backpropagation Training," and Chapter 7, "Other Propagation Training," will introduce several algorithms that were based on the backpropagation training algorithms introduced in Chapter 6.

5.1 Evaluating Classification

Classification is the process by which a neural network attempts to classify the input into one or more classes. The simplest way of evaluating a classification network is to track the percentage of training set items that were classified incorrectly. We typically score human examples in this manner. For example, you might have taken multiple-choice exams in school in which you had to shade in a bubble for choices A, B, C, or D. If you chose the wrong letter on a 10-question exam, you would earn a 90%. In the same way, we can grade computers; however, most classification algorithms do not simply choose A, B, C, or D. Computers typically report a classification as their percent confidence in each class. Figure 5.1 shows how a computer and a human might both respond to question #1 on an exam:

Figure 5.1: Human Exam versus Computer Classification



As you can see, the human test taker marked the first question as "B." However, the computer test taker had an 80% (0.8) confidence in "B" and was also somewhat sure with 10% (0.1) on "A." The computer then distributed the remaining points on the other two. In the simplest sense, the machine would get 80% of the score for this question if the correct answer were "B."

The machine would get only 5% (0.05) of the points if the correct answer were "D."

5.1.1 Binary Classification

Binary classification occurs when a neural network must choose between two options, which might be true/false, yes/no, correct/incorrect, or buy/sell. To see how to use binary classification, we will consider a classification system for a credit card company. This classification system must decide how to respond to a new potential customer. This system will either "issue a credit card" or "decline a credit card."

When you have only two classes that you can consider, the objective function's score is the number of false positive predictions versus the number of false negatives. False negatives and false positives are both types of errors, and it is important to understand the difference. For the previous example, issuing a credit card would be the positive. A false positive occurs when a credit card is issued to someone who will become a bad credit risk. A false negative happens when a credit card is declined to someone who would have been a good risk.

Because only two options exist, we can choose the mistake that is the more serious type of error, a false positive or a false negative. For most banks issuing credit cards, a false positive is worse than a false negative. Declining a potentially good credit card holder is better than accepting a credit card holder who would cause the bank to undertake expensive collection activities.

A classification problem seeks to assign the input into one or more categories. A binary classification employs a single-output neural network to classify into two categories. Consider the auto MPG data set that is available from the University of California at Irvine (UCI) machine learning repository at the following URL:

https://archive.ics.uci.edu/ml/datasets/Auto+MPG

For the auto MPG data set, we might create classifications for cars built inside of the United States. The field named *origin* provides information on

the location of the car assembly. Thus, the single-output neuron would give a number that indicates the probability that the car was built in the USA.

To perform this prediction, you need to change the *origin* field to hold values between 1 and the low-end range of the activation function. For example, the low end of the range for the sigmoid function is 0; for the hyperbolic tangent, it is -1. The neural network will output a value that indicates the probability of a car being made in the USA or elsewhere. Values closer to 1 indicate a higher probability of the car originating in the USA; values closer to 0 or -1 indicate a car originating from outside the USA.

You must choose a cutoff value that differentiates these predictions into either USA or non-USA. If USA is 1.0 and non-USA is 0.0, we could just choose 0.5 as the cutoff value. Consequently, a car with an output of 0.6 would be USA, and 0.4 would be non-USA.

Invariably, this neural network will produce errors as it classifies cars. A USA-made car might yield an output of 0.45; however, because the neural network is below the cutoff value, it would not put the car in the correct category. Because we designed this neural network to classify USA-made cars, this error would be called a false negative. In other words, the neural network indicated that the car was non-USA, creating a negative result because the car was actually from the USA. Thus, the negative classification was false. This error is also known as a type-2 error.

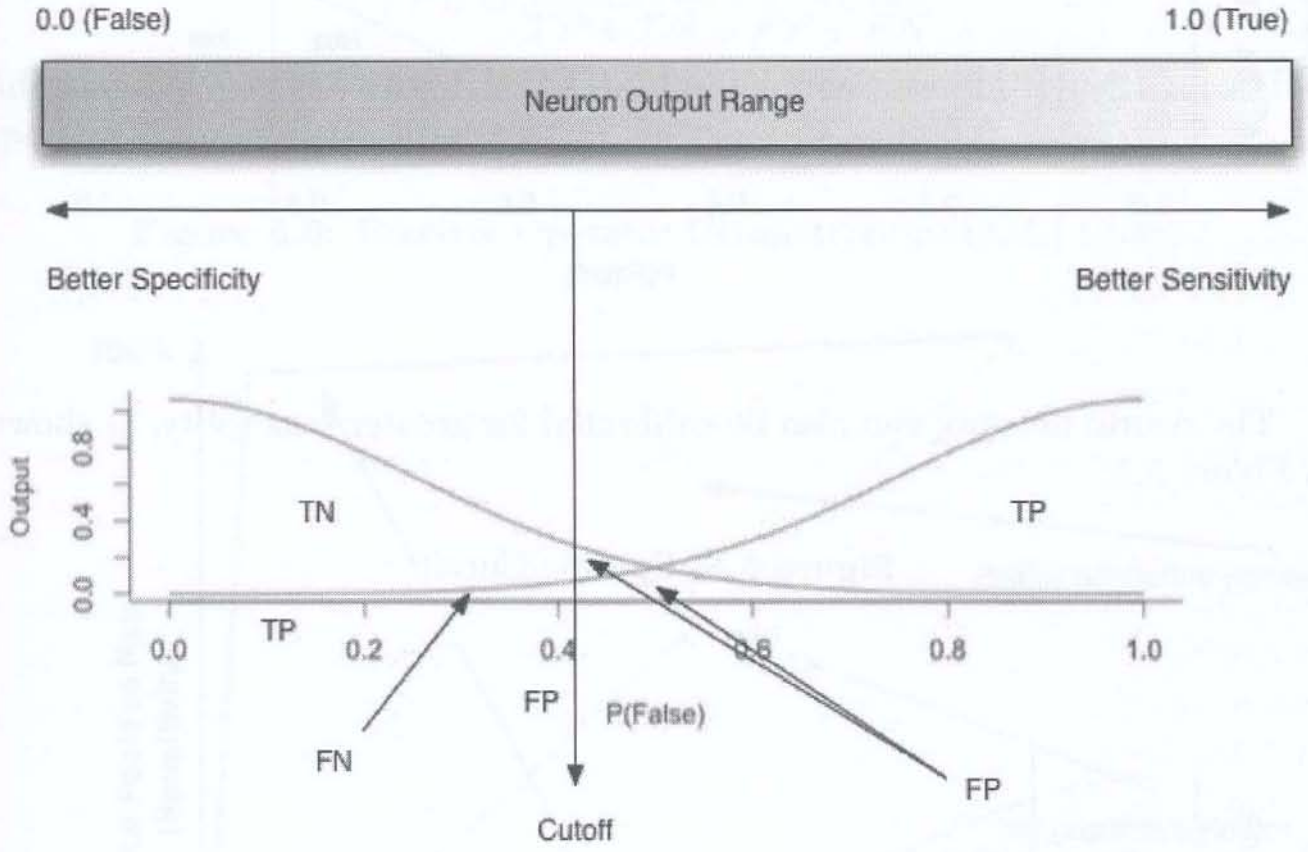
Similarly, the network might falsely classify a non-USA car as USA. This error is a false positive, or a type-1. Neural networks prone to produce false positives are characterized as more specific. Similarly, neural networks that produce more false negatives are labeled as more sensitive. Figure 5.2 summarizes these relationships between true/false, positives/negatives, type-1 & type-2 errors, and sensitivity/specificity:

Figure 5.2: Types of Errors

True vs False Positives	Type-1 Error	Sensitivity of Test
True vs False Negatives	Type-2 Error	Specificity of Test

Setting the cutoff for the output neuron selects whether sensitivity or specificity is more important. It is possible to make a neural network more sensitive or specific by adjusting this cutoff, as illustrated in Figure 5.3:

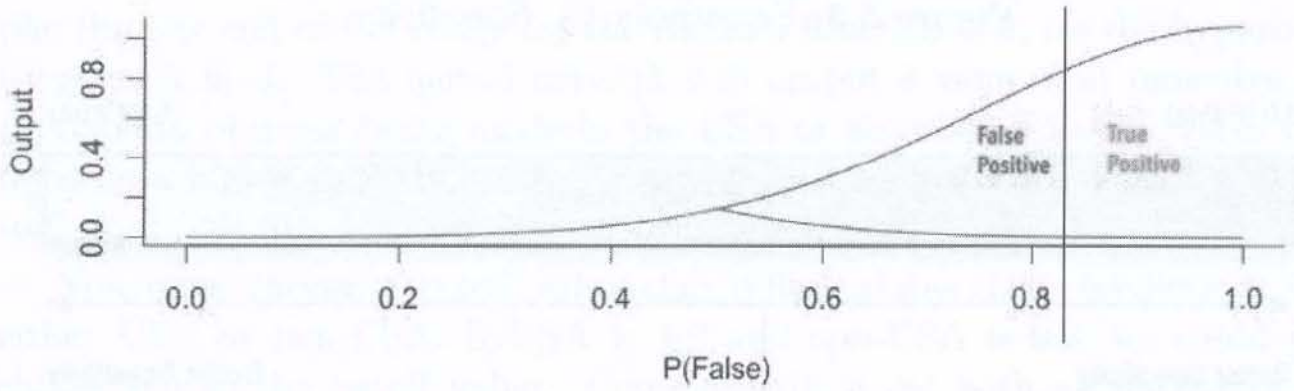
Figure 5.3: Sensitivity vs. Specificity



As the limit line moves left, the network becomes more specific. The decrease in the size of the true negative (TN) area makes this specificity evident. Conversely, as the limit line moves right, the network becomes more sensitive. This sensitivity is evident in the decrease in size of the true positive (TP) area.

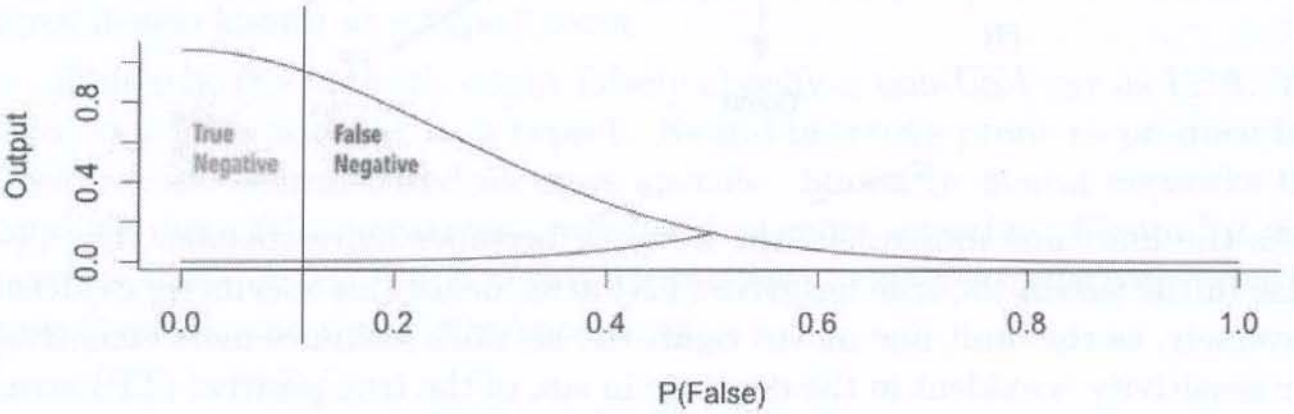
Increases in sensitivity will usually result in a decrease of specificity. Figure 5.4 shows a neural limit designed to make the neural network very sensitive:

Figure 5.4: Sensitive Cutoff



The neural network can also be calibrated for greater sensitivity, as shown in Figure 5.5:

Figure 5.5: Specific Cutoff



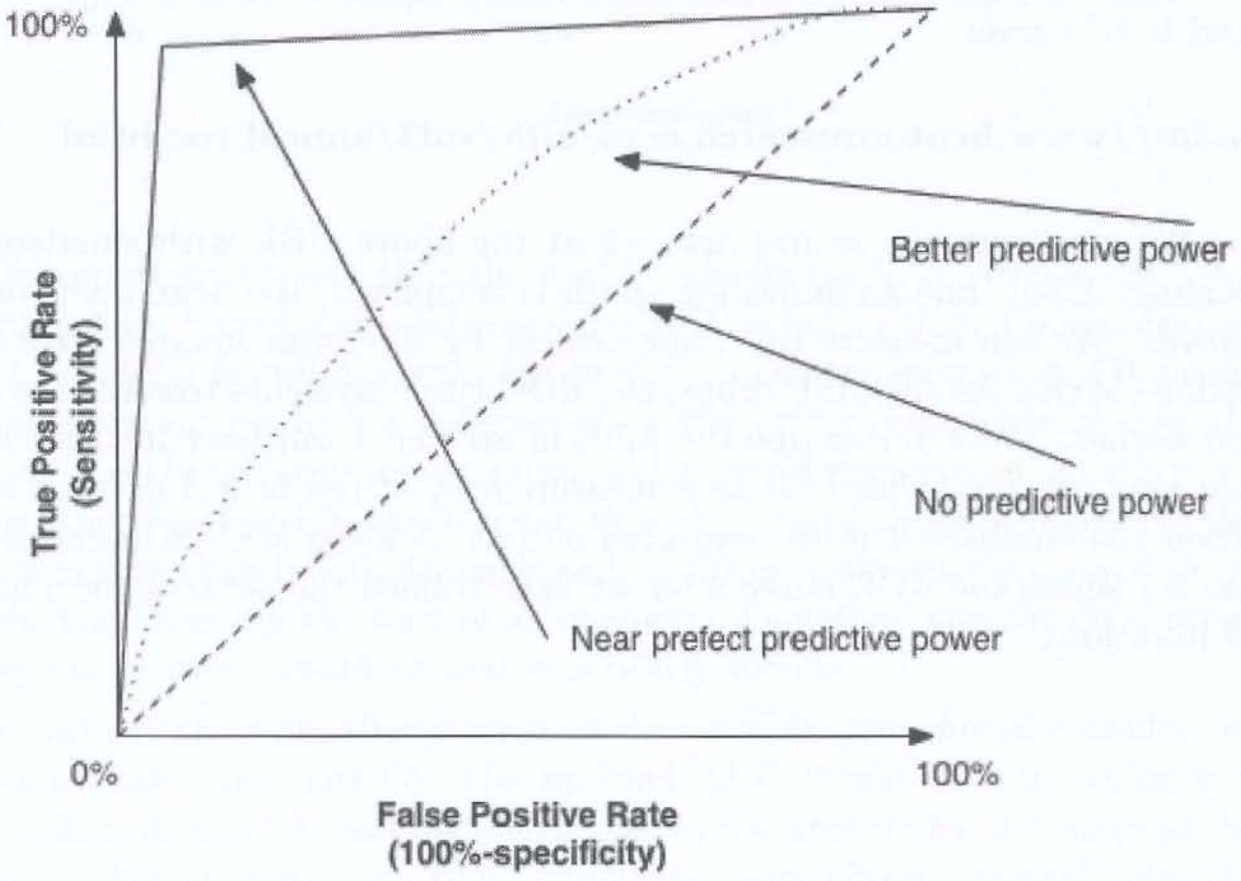
Attaining 100% specificity or sensitivity is not necessarily good. A medical test can reach 100% specificity by simply predicting that everyone does not have the disease. This test will never commit a false positive error because it never gave a positive answer. Obviously, this test is not useful. Highly specific or sensitive neural networks produce the same meaningless result. We need a

way to evaluate the total effectiveness of the neural network that is independent of the cutoff point. The total prediction rate combines the percentage of true positives and true negatives. Equation 5.1 can calculate the total prediction rate:

$$TPR = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Additionally, you can visualize the total prediction rate (TPR) with a receiver operator characteristic (ROC) chart, as seen in Figure 5.6:

Figure 5.6: Receiver Operator Characteristic (ROC) Chart



The above chart shows three different ROC curves. The dashed line shows an ROC with zero predictive power. The dotted line shows a better neural network, and the solid line shows a nearly perfect neural network. To understand how to read an ROC chart, look first at the origin, which is marked

by 0%. All ROC lines always start at the origin and move to the upper-right corner where true positive (TP) and false positive (FP) are both 100%.

The y-axis shows the TP percentages from 0 to 100. As you move up the y-axis, both TP and FP increase. As TP increases, so does sensitivity; however, specificity falls. The ROC chart allows you to select the level of sensitivity you need, but it also shows you the number of FPs you must accept to achieve that level of sensitivity.

The worst network, the dashed line, always has a 50% total prediction rate. Given that there are only two outcomes, this result is no better than random guessing. To get 100% TP, you must also have a 100% FP, which still results in half of the predictions being wrong.

The following URL allows you to experiment with a simple neural network and ROC curve:

http://www.heatonresearch.com/aifh/vol3/anneal_roc.html

We can train the neural network at the above URL with simulated annealing. Each time an annealing epoch is completed, the neural network improves. We can measure this improvement by the mean squared error calculation (MSE). As the MSE drops, the ROC curve stretches towards the upper left corner. We will describe the MSE in greater detail later in this chapter. For now, simply think of it as a measurement of the neural network's error when you compare it to the expected output. A lower MSE is desirable. Figure 5.7 shows the ROC curve after we have trained the network for a number of iterations:

Figure 5.7: ROC Curve

It is important to note that the goal is not always to maximize the total prediction rate. Sometimes a false positive (FP) is better than a false negative (FN.) Consider a neural network that predicts a bridge collapse. A FP means that the program predicts a collapse when the bridge was actually safe. In this case, checking a structurally sound bridge would waste an engineer's time. On the other hand, a FN would mean that the neural network predicted the bridge was safe when it actually collapsed. A bridge collapsing is a much worse outcome than wasting the time of an engineer. Therefore, you should arrange this type of neural network so that it is overly specific.

To evaluate the total effectiveness of the network, you should consider the area under the curve (AUC). The optimal AUC would be 1.0, which is a 100% (1.0) x 100% (1.0) rectangle that pushes the area under the curve to the maximum. When reading an ROC curve, the more effective neural networks have more space under the curve. The curves shown previously, in Figure 5.6, correspond with this assessment.

5.1.2 Multi-Class Classification

If you want to predict more than one outcome, you will need more than one output neuron. Because a single neuron can predict two outcomes, a neural network with two output neurons is somewhat rare. If there are three or more outcomes, there will be three or more output neurons. *Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms* does show a method that can encode three outcomes into two output neurons.

Consider Fisher's iris data set. This data set contains four different measurements for three different species of iris flower. The following URL contains this data set:

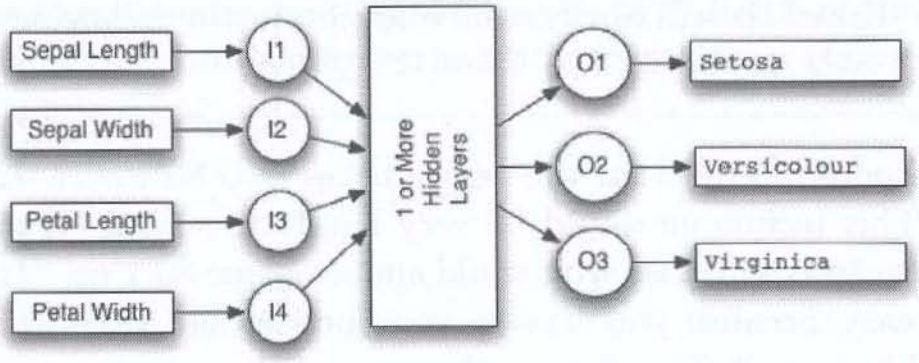
https://archive.ics.uci.edu/ml/datasets/Iris

Sample data from the iris data set is shown here:

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
7.0,3.2,4.7,1.4,Iris-versicolour
6.4,3.2,4.5,1.5,Iris-versicolour
6.3,3.3,6.0,2.5,Iris-virginica
5.8,2.7,5.1,1.9,Iris-virginica
```

Four measurements can predict the species. If you are interested in reading more about how to measure an iris flower, refer to the above link. For this prediction, the meaning of the four measurements does not really matter. These measurements will teach the neural network to predict. Figure 5.8 shows a neural network structure that can predict the iris data set:

Figure 5.8: Iris Data Set Neural Network



The above neural network accepts the four measurements and outputs three numbers. Each output corresponds with one of the iris species. The output neuron that produces the highest number determines the species predicted.

5.1.3 Log Loss

Classification networks can derive a class from the input data. For example, the four iris measurements can group the data into the three species of iris. One easy method to evaluate classification is to treat it like a multiple-choice exam and return a percent score. Although this technique is common, most machine learning models do not answer multiple-choice questions like you did in school. Consider how the following question might appear on an exam:

```
1. Would an iris setosa have a sepal length of 5.1 cm,
   a sepal width of 3.5 cm, a petal length of 1.4 cm, and
   a petal width of 0.2 cm?
A) True
B) False
```

This question is exactly the type that a neural network must face in a classification task. However, the neural network will not respond with an answer of "True" or "False." It will answer the question in the following manner:

```
True: 80%
```

The above response means that the neural network is 80% sure that the flower is a setosa. This technique would be very handy in school. If you could not decide between true and false, you could simply place 80% on "True." Scoring is relatively easy because you receive your percentage value for the correct answer. In this case, if "True" were the correct answer, your score would be 80% for that question.

However, log loss is not quite that simple. Equation 5.2 is the equation for log loss:

$$\text{log loss} = -\frac{1}{N}\sum_{i=1}^{N}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{5.2}$$

You should use this equation only as an objective function for classifications that have two outcomes. The variable *y-hat* is the neural network's prediction, and the variable y is the known correct answer. In this case, y will always be 0 or 1. The training data have no probabilities. The neural network classifies it either into one class (1) or the other (0).

The variable N represents the number of elements in the training set–the number of questions in the test. We divide by N because this process is customary for an average. We also begin the equation with a negative because the log function is always negative over the domain 0 to 1. This negation allows a positive score for the training to minimize.

You will notice two terms are separated by the addition (+). Each contains a log function. Because y will be either 0 or 1, then one of these two terms will cancel out to 0. If y is 0, then the first term will reduce to 0. If y is 1, then the second term will be 0.

If your prediction for the first class of a two-class prediction is *y-hat*, then your prediction for the second class is 1 minus *y-hat*. Essentially, if your prediction for class A is 70% (0.7), then your prediction for class B is 30% (0.3). Your score will increase by the log of your prediction for the correct

class. If the neural network had predicted 1.0 for class A, and the correct answer was A, your score would increase by log (1), which is 0. For log loss, we seek a low score, so a correct answer results in 0. Some of these log values for a neural network's probability estimate for the correct class:

- -log(1.0) = 0
- -log(0.95) = 0.02
- -log(0.9) = 0.05
- -log(0.8) = 0.1
- -log(0.5) = 0.3
- -log(0.1) = 1
- -log(0.01) = 2
- -log(1.0e-12) = 12
- -log(0.0) = negative infinity

As you can see, giving a low confidence to the correct answer affects the score the most. Because log (0) is negative infinity, we typically impose a minimum value. Of course, the above log values are for a single training set element. We will average the log values for the entire training set.

5.1.4 Multi-Class Log Loss

If more than two outcomes are classified, then we must use multi-class log loss. This loss function is very closely related to the binary log loss just described. Equation 5.3 shows the equation for multi-class log loss:

$$\text{multi-class log loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(\hat{y}_{i,j}) \tag{5.3}$$

In the above equation, N is the number of training set elements, and M represents the number of categories for the classification process. Conceptually,

the multi-class log loss objective function works similarly to single log loss. The above equation essentially gives you a score that is the average of the negative-log of your prediction for the correct class on each of the data sets. The inner most sigma-summation in the above equation functions as an if-then statement and allows only the correct class with a y of 1.0 to contribute to the summation.

5.2 Evaluating Regression

Mean squared error (MSE) calculation is the most commonly utilized process for evaluating regression machine learning. Most Internet examples of neural networks, support vector machines, and other models apply MSE (Draper, 1998), shown in Equation 5.4:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{5.4}$$

In the above equation, y is the ideal output and *y-hat* is the actual output. The mean squared error is essentially the mean of the squares of the individual differences. Because the individual differences are squared, the positive or negative nature of the difference does not matter to MSE.

You can evaluate classification problems with MSE. To evaluate classification output with MSE, each class's probability is simply treated as a numeric output. The expected output simply has a value of 1.0 for the correct class, and 0 for the others. For example, if the first class were correct, and the other three classes incorrect, the expected outcome vector would look like the following:

```
[1.0, 0, 0, 0]
```

You can use nearly any regression objective function for classification in this way. A variety of functions, such as root mean square (RMS) and sum of squares error (SSE) can evaluate regression, and we discussed these functions in *Artificial Intelligence for Humans, Volume 1: Fundamental Algorithms.*

5.3 Training with Simulated Annealing

To train a neural network, you must define its tasks. An objective function, otherwise known as scoring or loss functions, can generate these tasks. Essentially, an objective function evaluates the neural network and returns a number indicating the usefulness of the neural network. The training process modifies the weights of the neural network in each iteration so the value returned from the objective function improves.

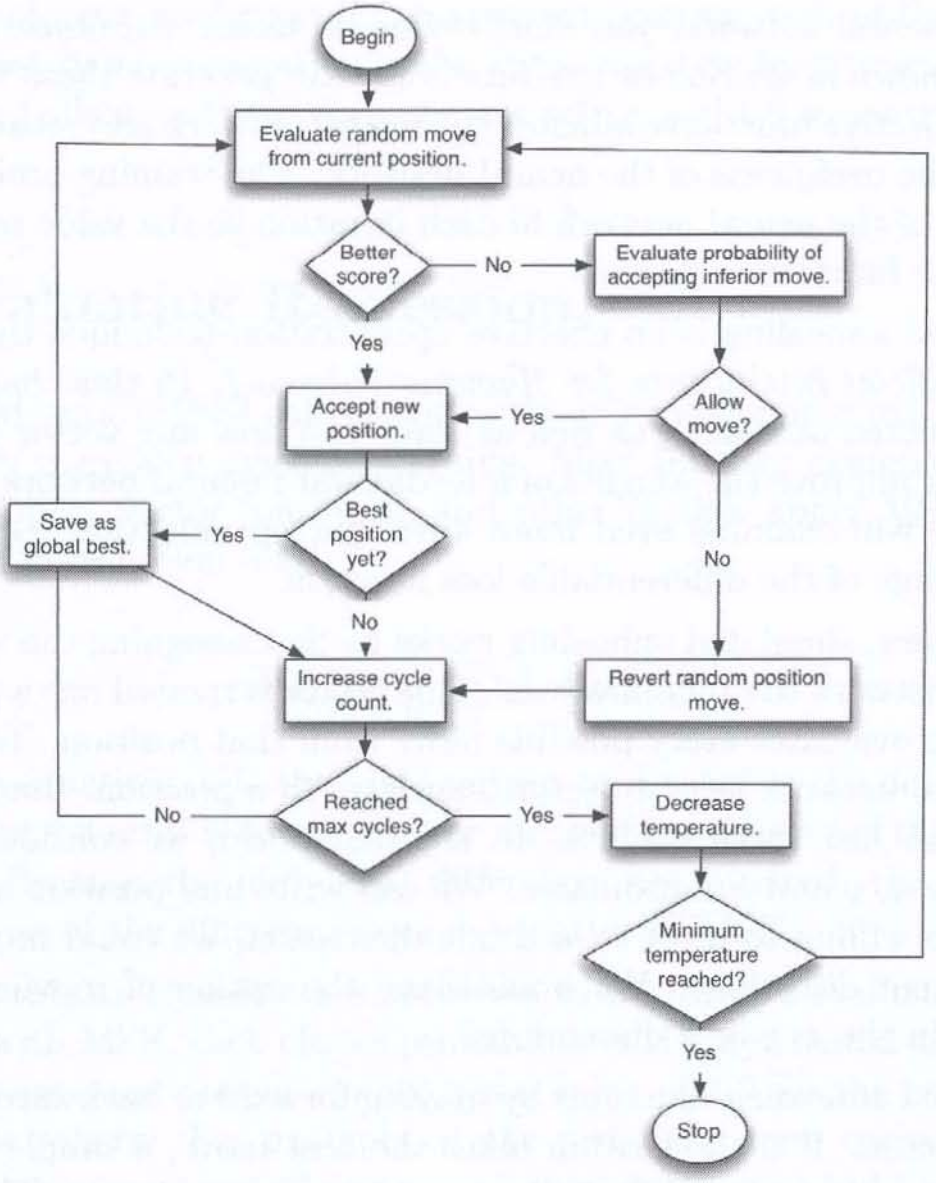
Simulated annealing is an effective optimization technique that we examined in *Artificial Intelligence for Humans Volume 1*. In this chapter, we will review simulated annealing as well as show you how any vector optimization function can improve the weights of a feedforward neural network. In the next chapter, we will examine even more advanced optimization techniques that take advantage of the differentiable loss function.

As a review, simulated annealing works by first assigning the weight vector of a neural network to random values. This vector is treated like a position, and the program evaluates every possible move from that position. To understand how a neural network weight vector translates to a position, think of a neural network with just three weights. In the real world, we consider position in terms of the x, y and z coordinates. We can write any position as a vector of 3. If we are willing to move in a single dimension, we could move in a total of six different directions. We would have the option of moving forward or backwards in the x, y or z dimensions.

Simulated annealing functions by moving forward or backwards in all available dimensions. If the algorithm takes the best move, a simple hill-climbing algorithm would result. Hill climbing only improves scores. Therefore, it is called a greedy algorithm. To reach the best position, an algorithm will sometime need to move to a lower position. As a result, simulated annealing very much follows the expression of two steps forward, one step back.

In other words, simulated annealing will sometimes allow a move to a weight configuration with a worse score. The probability of accepting such a move starts high and decreases. This probability is known as the current temperature, and it simulates the actual metallurgical annealing process where a metal cools and achieves greater hardness. Figure 5.9 shows the entire process:

Figure 5.9: Simulated Annealing



A feedforward neural network can utilize simulated annealing to learn the iris data set. The following program shows the output from this training:

```
Iteration #1, Score=0.3937, k=1,kMax=100,t=343.5891,prob=0.9998
Iteration #2, Score=0.3937, k=2,kMax=100,t=295.1336,prob=0.9997
Iteration #3, Score=0.3835, k=3,kMax=100,t=253.5118,prob=0.9989
Iteration #4, Score=0.3835, k=4,kMax=100,t=217.7597,prob=0.9988
Iteration #5, Score=0.3835, k=5,kMax=100,t=187.0496,prob=0.9997
Iteration #6, Score=0.3835, k=6,kMax=100,t=160.6705,prob=0.9997
Iteration #7, Score=0.3835, k=7,kMax=100,t=138.0116,prob=0.9996
```

```
...
Iteration #99, Score=0.1031, k=99,kMax=100,t=1.16E-4,prob=2.8776E
    -7
Iteration #100, Score=0.1031, k=100,kMax=100,t=9.9999E-5,prob
    =2.1443E-70
Final score: 0.1031
[0.22222222222222213, 0.6249999999999999, 0.06779661016949151,
    0.04166666666666667] -> Iris-setosa, Ideal: Iris-setosa
[0.1666666666666668, 0.4166666666666663, 0.06779661016949151,
    0.04166666666666667] -> Iris-setosa, Ideal: Iris-setosa
...
[0.6666666666666666, 0.4166666666666663, 0.711864406779661,
    0.9166666666666666] -> Iris-virginica, Ideal: Iris-virginica
[0.5555555555555555, 0.20833333333333331, 0.6779661016949152,
    0.75] -> Iris-virginica, Ideal: Iris-virginica
[0.611111111111111, 0.4166666666666663, 0.711864406779661,
    0.7916666666666666] -> Iris-virginica, Ideal: Iris-virginica
[0.5277777777777778, 0.5833333333333333, 0.7457627118644068,
    0.9166666666666666] -> Iris-virginica, Ideal: Iris-virginica
[0.44444444444444453, 0.4166666666666663, 0.6949152542372881,
    0.7083333333333334] -> Iris-virginica, Ideal: Iris-virginica
[1.178018083703488, 16.66575553359515, -0.6101619300462806,
    -3.9894606091020965, 13.989551673146842, -8.87489712462323,
    8.027287801488647, -4.615098285283519, 6.426489182215509,
    -1.4672962642199618, 4.136699061975335, 4.20036115439746,
    0.9052469139543605, -2.8923515248132063, -4.733219252086315,
    18.6497884912826, 2.5459600552510895, -5.618872440836617,
    4.638827606092005, 0.8887726364890928, 8.730809901357286,
    -6.4963370793479545, -6.4003385330186795, -11.820235441582424,
    -3.29494170904095, -1.5320936828139837, 0.1094081633203249,
    0.26353076268018827, 3.935780218339343, 0.8881280604852664,
    -5.048729642423418, 8.288232057956957, -14.686080237582006,
    3.058305829324875, -2.4144038920292608, 21.76633883966702,
    12.151853576801647, -3.6372061664901416, 6.28253174293219,
    -4.209863472970308, 0.8614258660906541, -9.382012074551428,
    -3.346419915864691, -0.6326977049713416, 2.1391118323593203,
    0.44832732990560714, 6.853600355726914, 2.8210824313745957,
    1.3901883615737192, -5.962068350552335, 0.502596306917136]
```

The initial random neural network starts out with a high multi-class log loss score of 30. As the training progresses, this value falls until it is low enough for training to stop. For this example, the training stops as soon as the error

falls below 10. To determine a good stopping point for the error, you should evaluate how well the network is performing for your intended use. A log loss below 0.5 is often in the acceptable range; however, you might not be able to achieve this score with all data sets.

The following URL shows an example of a neural network trained with simulated annealing:

http://www.heatonresearch.com/aifh/vol3/anneal_roc.html

5.4 Chapter Summary

Objective functions can evaluate neural networks. They simply return a number that indicates the success of the neural network. Regression neural networks will frequently utilize mean squared error (MSE). Classification neural networks will typically use a log loss or multi-class log loss function. These neural networks create custom objective functions.

Simulated annealing can optimize the neural network. You can utilize any of the optimization algorithms presented in Volumes 1 and 2 of *Artificial Intelligence for Humans*. In fact, you can optimize any vector in this way because the optimization algorithms are not tied to a neural network. In the next chapter, you will see several training methods designed specifically for neural networks. While these specialized training algorithms are often more efficient, they require objective functions that have a derivative.