Chapter 6

Weight Initialization

- Ranged Random Weights
- Trained and Untrained Neural Networks
- Nguyen-Widrow Weight Init

Neural networks must start with their weights initialized to random numbers. These random weights provide the training algorithms with a starting point for the weights. If all of the weights of a neural network were set to zero, the neural network would never train to an acceptable error level. This is because a zeroed weight matrix would place the neural network into a local minimum from which it can never escape.

Often the weights of neural networks are simply initialized with random numbers between a specific range. The range -1 to +1 is very popular. These random weights will change as the neural network is trained to produce acceptable outputs. In the next section we will take a look at what trained and untrained neural networks look like.

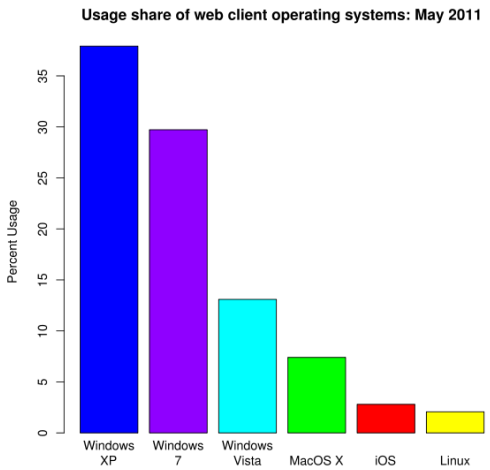
Later in this chapter we will see that these random weights can be modified to help the neural network to train faster. The Nguyen-Widrow weight initialization algorithm is a popular technique to adjust these starting weights. The Nguyen-Widrow weight initialization algorithm puts the weights into a position that is much more conducive to training. This means fewer training iterations to get the neural network to an acceptable error rate.

6.1 Looking at the Weights

In previous chapters we've looked at the weights of a neural network as an array of numbers. You can't typically glance at a weight array and see any sort of meaningful pattern. However, if the weights are represented graphically patterns begin to emerge.

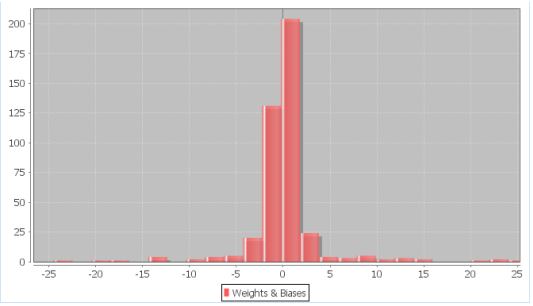
One common way to view the weights of a neural network is with a special type of chart called a histogram. A histogram is made up of vertical bars that count the number of occurances in a population. You've probably seen histograms many times before. For example, Figure 6.1 shows the popularity of operating systems.

Figure 6.1: Histogram of OS Popularity (from Wikipedia)



The y-axis shows the number of occurrences of each of the groups in the y-axis. We can use a histogram to look at the weights of a neural network. You can typically tell a trained, from a untrained neural network by looking at this histogram. Figure 6.2 shows a trained neural network.

Figure 6.2: A Trained Neural Network



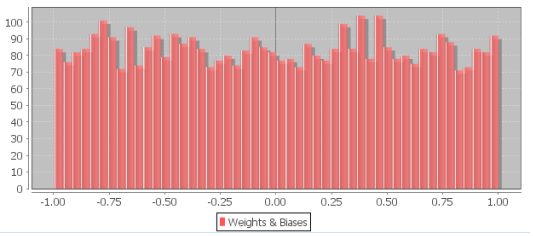
A neural network histogram is the same as concept as the operating system histogram shown earlier. The y-axis specifies how many weights fell into the ranges specified by the numbers on the x-axis. This allows you to see the distribution of the weights.

Most trained neural networks will look something like the above chart. Their weights will be very tightly clustered around zero. A trained neural network will typically look like a vary narrow gaussian curve.

6.2 Ranged Randomization

In the last section we saw what a trained neural network looks like in a weight histogram. Untrained neural networks can have a variety of appearances. How the weight histogram will look will be determined by the weight initialization method used.

Figure 6.3: A Ranged Randomization



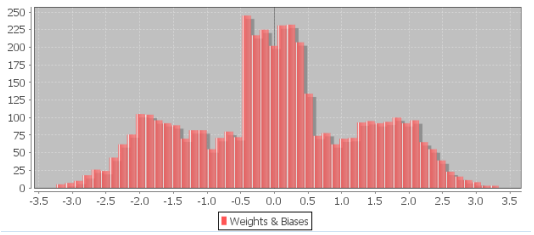
Range randomization produces a very simple looking chart. The more weights there are the flatter the top will be. This is because the random number generator should be giving

you an even distribution of numbers. If you are randomizing to the range of -1 to 1, you would expect to have approximately the same number of weights above zero as below.

6.3 Using Nguyen-Widrow

We will now look at the Nguyen-Widrow weight initialization method. The Nguyen-Widrow method starts out just like the range randomized method. Random values are chosen between -0.5 and +0.5. However, a special algorithm is employed to modify the weights. The histogram of a Nguyen-Widrow weight initialization looks like Figure 6.4.

Figure 6.4: The Nguyen-Widrow Initialization



As you can see, the Nguyen-Widrow initialization has a very distinctive pattern. There is a large distribution of weights between -0.5 and 0.5. However it then gradually rises and then rapidly falls off to around -3.0 and +3.0.

6.3.1 Performance of Nguyen-Widrow

You may be wondering how much of an advantage Nguyen-Widrow can have have. The following shows the average number of training iterations needed to train a neural network initialized by range random and Nguyen-Widrow.

```
Average iterations needed (lower is better)
Range random: 502.86
Nguyen-Widrow: 454.88
```

As you can see from the above information, the Nguyen-Widrow outperforms the range randomizer.

6.3.2 Implementing Nguyen-Widrow

This technique was invented by Derrick Nguyen and Bernard Widrow. It was first introduced in their paper "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights" in the Proceedings of the International Joint Conference on Neural Networks, 3:21-26, 1990.

To implement an Nguyen-Widrow randomization first initialize the neural network with random weight values in the range -0.5 to +0.5. This is exactly the same technique as was described earlier in this article for the ranged random numbers.

The Nguyen-Widrow randomization technique is efficient because it assigns each hidden neuron to a range of the problem. To do this we must map the input neurons to the hidden neurons. We calculate a value, called beta, that establishes these ranges. You can see the calculation of beta in Equation 6.1.

$$\beta = 0.7h^{\frac{1}{i}} \tag{6.1}$$

The variable **h** represents the number of hidden neurons in the first hidden layer, whereas the variable **i** represents the number of input neurons. We will calculate the weights taking each hidden neuron one at a time. For each hidden neuron we calculate the Euclidean norm for of all inputs to the current hidden neuron. This is done with Equation 6.2.

$$n = \sqrt{\sum_{i=0}^{i<w_{max}} w_i^2} \tag{6.2}$$

Beta will stay the same for every hidden neuron. However, the norm must be recalculated for each hidden neuron. Once the beta and norm values have been calculate, the random weights can be adjusted. The equation below shows how weights are adjusted using the previously calculated values.

$$w_{t+1} = \frac{\beta w_t}{n} \tag{6.3}$$

All inbound weights to the current hidden neuron are adjusting using the same norm. This same process is repeated for each hidden neuron.

You may have noticed that we are only specifying how to calculate the weights between the input layer and the first hidden layer. The Nguyen-Widrow method does not specify

how to calculate the weights between a hidden layer and the output. Likewise the Nguyen-Widrow method does not specify how to calculate the weights between multiple hidden layers. All weights outside of the first layer and first hidden layer are simply initialized to a value between -0.5 and +0.5.

6.3.3 Nguyen-Widrow in Action

We will now walk through the random weight initialization for a small neural network. We will look at a neural network that has two input neurons and a single output neuron. There is a single hidden layer with two neurons. Bias neurons are present on the input and hidden layers.

We begin by initializing the weights to random numbers in the range -0.5 to +0.5. The starting weights are shown here.

```
Weight 0: H1->O1: 0.23773012320107711
Weight 1: H2->O1: 0.2200753094723884
Weight 2: B2->O1: 0.12169691073037914
Weight 3: I1->H1: 0.5172524211645029
Weight 4: I2->H1: -0.5258712726818855
Weight 5: B1->H1: 0.8891383322123643
Weight 6: I1->H2: -0.007687742622070948
Weight 7: I2->H2: -0.48985643968339754
Weight 8: B1->H2: -0.6610227585583137
```

First we must calculate beta, which is given in Equation 6.1. This calculation is shown here.

```
Beta = 0.7 * ( hiddenNeurons ^ (1.0/inputCount)
```

Filling in the variables we have.

```
Beta = 0.7 * ( 2.0 ^ (1.0/2.0)) = 0.9899
```

We are now ready to modify the weights. We will only modify weights three through eight. Weights zero through two are not covered by the Nguyen-Widrow algorithm, and are simply set to random values between -0.5 and 0.5.

The weights will be recalculated in two phases. First we will recalculate all of the weights from the bias and input neurons to hidden neuron one. To do this we must calculate the Euclidean norm, or magnitude, for hidden neuron one. From Equation 6.2, we have the following.

```
Norm Hidden 1 = sqrt( (weight 3)^2 + (weight 4)^2 + (weight 5)^ 2)
```

Filling in the weights we have the following.

```
Norm Hidden 1 = sqrt((0.5172^2) + (-0.5258)^2 + (0.8891^2)) = 1.155
```

We will now look at how to calculate the first weight.

```
New Weight =( beta * Old Weight)/Norm Hidden 1
```

Filling in values we have.

```
New Weight =( 0.9899 * 0.5172) / 1.155 = 0.4432
```

All three weights feeding hidden the first neuron are transformed in this way.

```
0.5172524211645029 -> 0.44323151482681195 (as just seen)
-0.5258712726818855 -> -0.4506169739523903
0.8891383322123643 -> 0.7618990530577658
```

We now repeat the same process for neuron two. The value for beta stays the same. However, the magnitude must be recalculated. The recalculated magnitude for neuron two is given here.

```
Neuron Two Magnitude = 0.8227815750355376
```

Using this we can now recalculate weights six through eight.

```
-0.007687742622070948 -> -0.009249692928269318
-0.48985643968339754 -> -0.5893825884595142
-0.6610227585583137 -> -0.79532547274779
```

This results in the final values for all of the weights.

```
Weight 0: H1->O1: 0.23773012320107711
Weight 1: H2->O1: 0.2200753094723884
Weight 2: B2->O1: 0.12169691073037914
Weight 3: I1->H1: 0.44323151482681195
Weight 4: I2->H1: -0.4506169739523903
Weight 5: B1->H1: 0.7618990530577658
Weight 6: I1->H2: -0.009249692928269318
Weight 7: I2->H2: -0.5893825884595142
Weight 8: B1->H2: -0.79532547274779
```

These weights, though still random, better utilize the hidden neurons. These are the weights that we will begin training with.

6.4 Chapter Summary

In this chapter we saw how the weights of a neural network are initialized. These weights must be set to random values. If the weights were all set to zero, the neural network would never train properly. There are several different ways that the weights of a neural network are commonly initialized.

The first is range randomization. This process initializes each weight of the neural network to a random value in a specific range. The range used is typically -1 to 1 or 0 to 1. Though range randomization can provide very good results for neural network training, there are better methods.

Nguyen-Widrow weight initialization can provide significantly better training times than simple range randomization. The Nguyen-Widrow randomization technique is efficient because it assigns each hidden neuron to a range of the problem.

So far all training presented has focused on attempting to minimize the error. In the next chapter we will look the Levenberg Marquardt (LMA) training algorithm. LMA can often train in fewer iterations than resilient propagation (RPROP).