

1. Discrete Perceptron

The average period of convergence to the desired result is a function of the learning rate Alpha. To demonstrate figure 1 is included, done as a average of 100 trainings with Alpha set to 0.1, 0.2, 0.3, 0.4, 0.5, etc.. till 1.0, for AND (left) and OR (right):

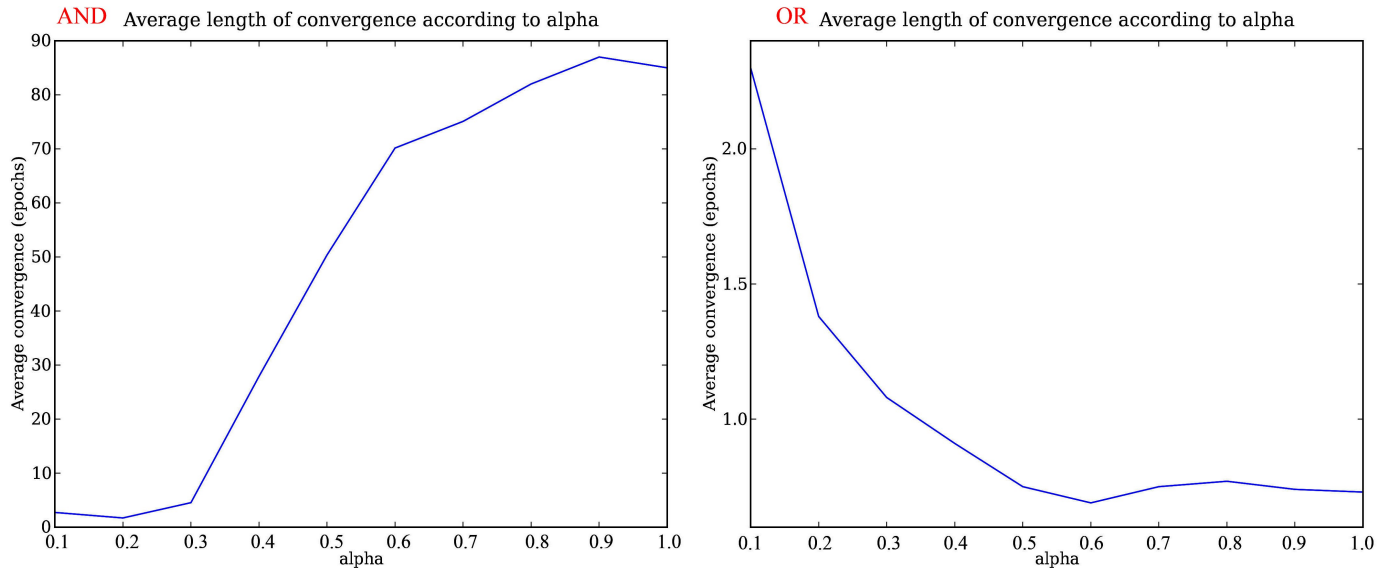


Figure 1.

Training done 100 times for various values of Alpha, taking 100 Epochs. On the left side is the AND training set,. We see that with alpha approaching 0.9, the average convergence period approaches the maximum Epochs available. On the right side we see that when training the perceptron on a OR training set, higher alpha means a faster convergence. Also note that on the plot on the right side, the maximum average convergence rate is cca 2.4 Epochs.

When training the perceptron to act as a binary classifier for the function AND, i found out that good learning rates are < 0.3 . When using learning rates > 0.3 the perceptron can never converge, as shown on figure 2, but it can also happen that the weights have been randomly initialized to good values, and the perceptron converges in one or two epochs (figure 3).



Figure 2.

Training the perceptron with AND training set and with learning rate $\alpha = 0.4$. The perceptron's weights enter a loop, with error rates between 1 and 3.

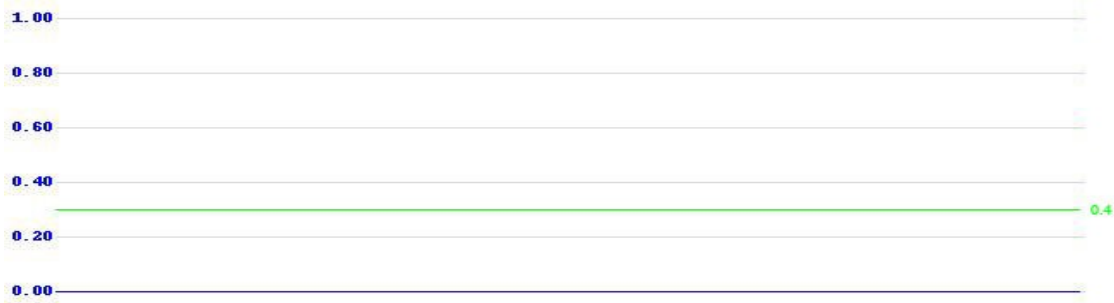


Figure 3.

Training the perceptron with AND training set, and with learning rate $\alpha = 0.4$. The perceptron is randomly initialized to a good result.

When training the perceptron as a binary classifier for OR, i found out that it is most likely to be correctly trained when using any of the alpha's between 0 and 1, it only has a impact on the length of the training. (see figure 1 right).

I trained the perceptron with Alpha = 0.01 to get a longer lasting convergence. The output of the perceptron along with its weights is in the file **and_output.txt**. The separating hyperplane and its progress is shown on figure 4.

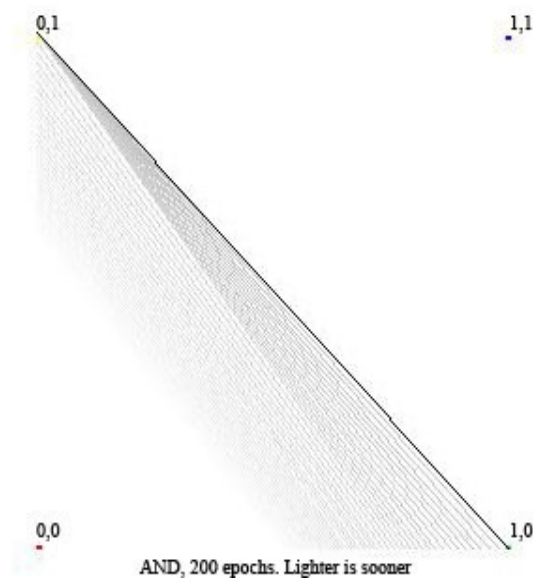


Figure 4.

The development of the separating hyper-plane over 200 Epochs. The black line is the final state of the hyper-plane. It can be seen that it really separates two classes of the AND training set.

1. Continuous Version (d)

There is only one slight difference between the discrete and continuous perceptron used for separating AND and OR classes. Here is some output from a training on the OR training set:

```
krok: 191 vahy: [2.6040045527268232, 2.5699759799876598] vstup: [['1', '0'], ['1', '1'], ['0', '0'], ['0', '1']] vystup:
[0.93083526991593957, 0.99434252025638559, 0.5, 0.92862557873407592] zelany: ['1', '1', '0', '1']
krok: 192 vahy: [2.6083968600705125, 2.5745001722000769] vstup: [['0', '1'], ['1', '1'], ['1', '0'], ['0', '0']] vystup:
[0.9289041096200612, 0.99439331968940481, 0.93114006865797849, 0.5] zelany: ['1', '1', '1', '0']
krok: 193 vahy: [2.6127696037405266, 2.5790025453940659] vstup: [['0', '1'], ['1', '0'], ['0', '0'], ['1', '1']] vystup:
[0.92920231383832641, 0.93140003605246358, 0.5, 0.99446503229271099] zelany: ['1', '1', '0', '1']
```

As we can see, the difference between discrete and continuous is that, that the discrete gives 0.5 for 0, and that there is never a output equal to 1, but very close to it. So the interpretation of the results has to be changed so, that a vector belongs to the "1" class when the result is very close to 1, and viceversa with 0,5 and 0.

2. Continuous version in 25 Dimensions

The perceptron was trained with the 25D data from perceptron2.dat file. It was trained for 200 Epochs, with Alpha = 0.03. Figure 5 shows the graph of five independent trainings.

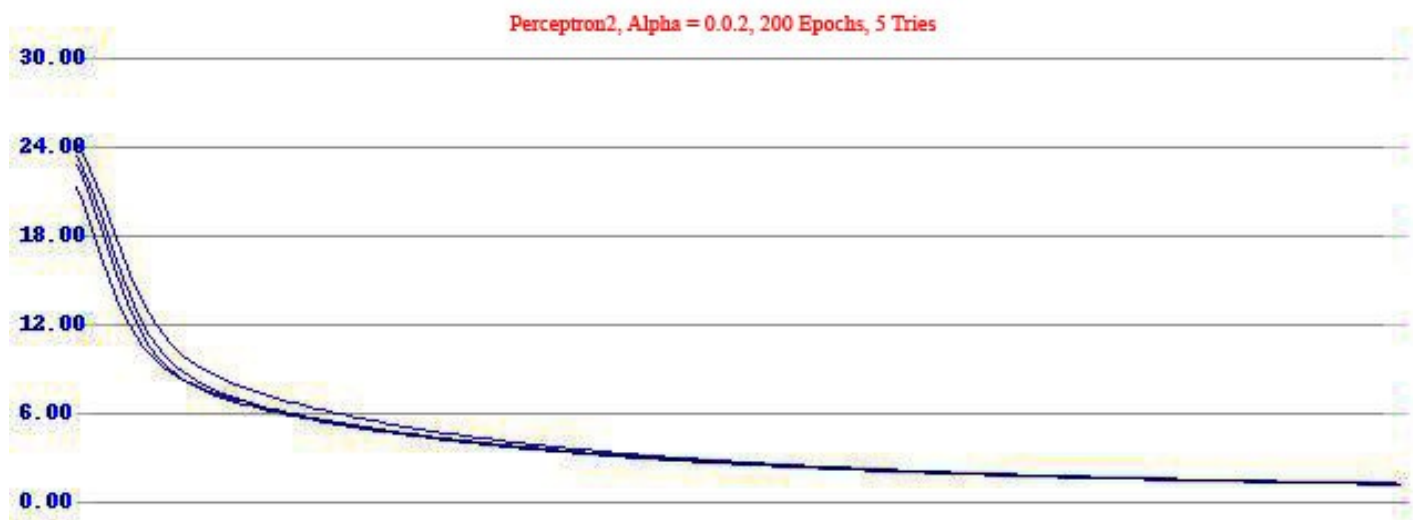


Figure 5.
Continuous perceptron operating on 25 Dimensions.

The perceptron was trained with alpha = 0.03 for 200 Epochs and then for 300 Epochs. When trained for 200 Epochs, the average success was cca 91.93 (measured as the average difference between the desired result and the actual result, in case of a number close to zero reversed like this: $1 - \text{result}$, and then multiplied by 100). When trained for 300 Epochs, the average success was cca 94.07. Which is a quite good result allready.