Learning a linearly separable rule

Neural Networks and Computation Intelligence

Ivo de Jong (s3174034)
Travis Hammond (s2880024)

January 2020

1 Introduction

In assignment 1 it was found that the Rosenblatt perceptron algorithm is very effective at finding a linear separation plane in a dataset if one exists. However, one disadvantage of the Rosenblatt algorithm is that it stops when it finds *any* perceptron that satisfies this separation. For most purposes the desired separation plane is the one that best approximates the "true" separation rule that generated the dataset, rather than just satisfying the datapoints and labels in the training data. This goal is commonly referred to as generalisability: how well a perceptron would keep performing when exposed to novel data.

It can be said that a perceptron with low generalisability will start misclassifying data as soon as the data changes slightly. This can be found when the plane touches "the edge" of some data points as demonstrated by the black lines in figure 1. This figure also indicates how high generalizability can be achieved: placing the separation plane in the middle of this area, as far away from any individual data point as possible.

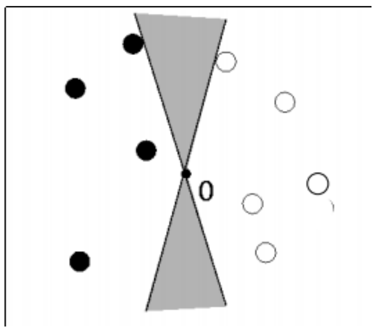

Figure 1: Possible perceptrons given a set of data. Image courtesy of Michael Biehl

1.1 The perceptron of maximal stability

The perceptron of maximal stability aims to achieve high generalizability, with a process that leads to the separation plane which is in the middle of the possible separation planes. It aims to achieve this by maximizing the distance that it has to the nearest data point. This distance between the separation plane and a data point is referred to as the stability of said datapoint and can be determined by $\kappa^\mu = \dfrac{w \cdot \xi^\mu S^\mu}{|w|}$, where κ^μ is the stability of datapoint μ, ξ^μ is its N-dimensional feature vector, S^μ is its label, and w is the N-dimensional weight vector of the perceptron.

As alluded to, the stability of the perceptron itself is defined by the datapoint which is closest to it, that is to say, the datapoint with the lowest stability. This defines the stability of the perceptron as $\kappa(w) = \min_\mu\{\kappa^\mu\}$.

1.2 Minover

The minover algorithm aims to find this perceptron of maximal stability. The intuitive way it works is by constantly having the perceptron "step away" from the datapoint with the lowest stability. This will allow the perceptron to converge to the perceptron of maximal stability. Unfortunately, due to the incessant stepping, the algorithm doesn't terminate nicely in the way that the Rosenblatt algorithm does. Instead, a stopping criterion has to be defined. The exact details of the implementation for this specific report will be discussed in the method section.

1.3 Adatron

The adatron algorithm also finds the perceptron of optimal stability, but instead of updating the weight vector it operates directly on the embedding strengths. It converges somewhat more smoothly than the minover algorithm, but also does not terminate early so a stopping criterion is also set.

1.4 Generating data

In this experiment the goal is not to find any linear separation plane, but instead the goal is to find the best one. Similar to the Rosenblatt experiment data will be generated as a N-dimensional Gaussian distribution around the origin with a variance of 1. The main difference here is that the data points are not randomly labelled, but instead are labeled based on a random perceptron w^*. This guarantees that a linear separation exists, and it also allows us to see how close the approximated perceptron of maximal stability is to the perceptron which actually determined the labelling (w^*).

An interesting note to be made here is that the data is still randomly distributed around the origin, not based on the perceptron. Looking back at figure 1 it becomes clear that any random w^* drawn in the gray area will draw the same labelling. This means that the w^* does not need to be the perceptron of maximal stability.

1.5 Determining generalization error

After a training run there will be 2 perceptrons that form the same separation of data, but will most likely have different separation planes. For clarity, these two perceptrons are the perceptron that labelled the data w^* and the approximate perceptron of maximal stability, which will be referred to as w^{opt}.

To determine the similarity between these, the generalization error is computed as defined by $\epsilon_g = \frac{1}{\pi} \arccos \frac{w^{opt} \cdot w^*}{|w^{opt}||w^*|}$.

1.6 Predictions

Since w^* ends up being any random valid separation plane after labelling, while the perceptron of maximal stability will be the "middle" separation plane it is not expected that the generalization error will diminish to 0. Instead it is expected that as the number of datapoints increases, the range in which w^* may be decreases, so the generalization error will decrease. For a larger number of dimensions it is expected that the generalization error will remain higher, as there is more "space" for the perceptrons to be in.

1.7 Bonus

For additional insight, the stabilities of the perceptron of maximal stability ($\kappa(t_{max})$ will also be determined for data that is randomly labeled. In this case it can be expected that the stabilities will go down as more data is added, as there is less "space" for the perceptron to move away. It is expected that for large dimensions stabilities will remain higher as more datapoints are added.

To add beyond that the generalization error between w^* and a perceptron trained by the Rosenblatt algorithm will be determined. This will converge to any linear separation within the bounds of the data. With figure 1 as a mental model of the situation, comparing a "random" valid separation plane from the Rosenblatt algorithm to another "random" valid separation plane of w^* is expected to have a higher generalization error than the comparison between the "random" w^* and the "middle w^{opt}.

2 Method

This method section will consist of three components. Firstly it will discuss the data generation. Next it will discuss the minover algorithm. Lastly it will tie some ends together by defining the parameters that the experiment will be performed over. Parts of the Bonus component will be omitted, such as the Rosenblatt algorithm, as this is already more thoroughly described in assignment 1.

2.1 Generating data

As briefly mentioned in the respective subsection in the introduction, the features of the datapoints will be randomly generated.

Firstly, P N-dimensional datapoints will be generated. Each dimension in each datapoint is a randomly sampled Gaussian with a mean 0 and a variance of 1. This forms a Gaussian "cloud" around the origin. This allows a perceptron through the origin to draw varying separations.

Next, to base labels on w^* we determine a random w^*. w^* will be an N-dimensional weight vector, where all weights are uniformly drawn. w^* is under the requirements that $|w^*|^2 = N$. Based on this we can define labels for each datapoint μ as $S^\mu = sign(w^* \cdot \xi^\mu$.

For the extra experiment where stabilities for randomly labeled data are determined the labels are instead sampled uniformly from the set $0, 1$.

2.2 Minover

The goal of the minover algorithm is to find the perceptron of the maximal stability. That is, to find $w_{opt} = \text{argmax}_{R^N}[\min_\mu(\dfrac{w \cdot \xi^\mu S^\mu}{|w|})]$. The way this is approached is expressed in listing 2.

Listing 1: Minover algorithm implemented in Python

```
import numpy as np

(xi, S) = generate_data(P, N)
w = np.zeros(N)

# Epoch loop
for t in range(t_max):
    # Data loop
    min_stability = None
    for xi_v, S_v in zip(xi, S):
        # Find the datapoint v = (xi_v, S_v)
            # with minimal stability
        # Min stability ~ abs(w .* xi_v * S_v)
        stability = abs(np.dot(w, xi_v * S_v))

        if min_stability is None
                or stability < min_stability[2]:
            min_stability = (xi_v, S_v, stability)

    # Update the weight vector with v = (xi_v, S_v)
    min_xi_v, min_S_v, stability = min_stability
    w += (min_xi_v * min_S_v) / N

return generalization_error(w, w_star)
```

The code in listing 2 demonstrates the Minover algorithm. It starts of with $w(0)$ as an N-dimensional 0 vector. For a maximum number of epochs, as defined by t_max, the algorithm goes over all datapoints and determines a proxy of the stability.

Note that the code shows $\kappa(\mu) = abs(w \cdot \xi^{\mu} S^{\mu}$, rather than following the definition provided in the introduction. This is because it only needs to find which stability is the lowest, so dividing all stabilities by the same $|w(t)|$ is meaningless. Note that for the bonus experiment where the actual stabilities for randomly labelled data are determined the original formula is used.

With the datapoint of minimal stability found, the weight vector is updated by the Hebbian term, giving $w(t+1) = w(t) + \dfrac{\psi^{\mu(t)} S^{\mu(t)}}{N}$, where $\mu(t)$ indicates the datapoint of minimal stability at epoch t.

Once the maximum number of epochs has been reached the generalization error between $w(t_{max})$, prev and w^* is determined according to the formula for ϵ_g defined in the introduction. At this point it is good to note that due to a finite number of iterations $w(t_{max})$ will remain an approximation of w^{opt}, as there will at best be some small "jittering" around w^{opt}, and at worst the algorithm will have had insufficient time to converge.

2.3 Adatron

The AdaTron algorithm uses tricks from quadratic optimization techniques to find the optimally stable perceptron by operating directly on embedding strengths, where at each timestep: $x^{\mu(t)}(t+1) = \max(0, x^{\mu(t)}(t) + \eta(1 - E^{\mu(t)}))$.

Listing 2: Minover algorithm implemented in Python

```
import numpy as np

(xi, S) = generate_data(P, N)
x = np.zeros(P)
C, eta = generate_C_eta(xi, S)

# Epoch loop
for t in range(t_max):

    # Data loop
    for v, (xi_v, S_v) in enumerate(zip(xi, S)):

        E_v = np.dot(C, x)[v]
        x[v] = max(0, x[v] + eta * (1 - E_v))
```

The code for the AdaTron algorithm looks simpler, but the math behind it is rather more complicated so we will not describe it in detail here. However, it would be good to mention that C is a PxP matrix where $C^{uv} = \frac{1}{N} * S^u * S^v * \xi^u * \xi^v$, and eta is bound by $0 < \eta < \frac{2}{C^u u}$ for all u. To determine the generalization error, the weight vector is constructed from the embedding strengths as follows: $w = \frac{1}{N} * \sum_{\mu}(x^{\mu} * \xi^{\mu} * S^{\mu})$

2.4 Experiment parameters

There is a specific relation between the number of datapoints and the number of dimension that we intend to explore. In fact, as the number of dimensions increases, the "density" of data decreases. Therefore we define $\alpha = P/N$ as the ratio of the number datapoints to the number of dimensions.

To gain a good understanding of how the generalization errors at t_{max} are affected by α we defined $\epsilon_g(\alpha) = \epsilon_g^{\alpha}(t_max)$. What this means is that $\epsilon_g(\alpha)$ forms a function of how generalization error is affected by the ratio between the datapoints and the dimensions.

For a sufficiently accurate measure of $\epsilon_g(\alpha)$ for any given α we need to perform a certain number of repetitions of generating data, labelling data according to a random w^*, learning a $w(t_{max}$, and calculating $\epsilon_g(\alpha)$. For this experiment that number of repetitions is set to 40, so each measurement of ϵ_g becomes the average of 40 identical measurements. In the bonus-experiments this parameter is kept the same.

Another static parameter that needs to be defined is t_max, the maximum number of epochs. Some trial-and-error seemed to suggest that $t_max = 100$ is reasonably sufficient to gain some convergence, while most importantly minimizing computational time. In this bonus-experiments this parameter is also kept the same. For the bonus-experiment involving the Rosenblatt algorithm, its t_max was also set to 100.

Lastly, and perhaps the most interesting are the parameters that form the different functions. A sufficiently small step-size for α was set to 0.25. To get a good range of low-to-high α values a minimum was set at 0.25, with a maximum at 10. A slight note that may be added to this is that if α is very small, $P = \alpha * N$ may be less than 1. In these cases P would have been rounded up to 1 to prevent computational problems. As the number of dimensions in this experiment was at least 5 this was not a problem, but it is something to be aware of for adaptation purposes. For the bonus-experiments the upper bound for α was set to 6.

To add another dimension of analysis (pun intended) the number of dimensions N was also varied. To keep the number of training runs somewhat minimized 3 different number of dimensions were used for all experiments, specifically $\{5, 20, 150\}$. This would give some indication of how more dimensions may affect behaviour, but is not sufficient to show exactly what shape this interaction has.

3 Result and Discussion

The main result to examine is of course the graph plotting $\epsilon_g\alpha$ for different numbers of dimensions. This can be found in figure 2.

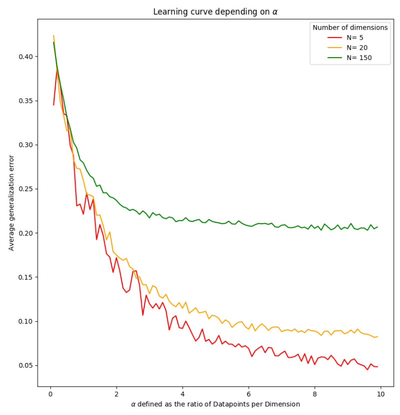

Figure 2: $\epsilon_g^N \alpha$ for various N

This figure firstly shows that as α increases the generalization error decreases. This makes perfect sense, as when looking back at figure 1 we can see that with an increased number of datapoints the space that the perceptrons can be in becomes smaller. This forces them to be closer together. This does have diminishing returns, as is to be expected, because the probability of a random extra datapoint "cutting through" the space of valid separation planes becomes smaller when this space is already smaller.

An interesting behaviour that this does show is that for higher dimensions the generalization error will remain higher, despite a large number of datapoints. This is a somewhat curious finding, and not what was expected. This may be because a higher dimensional "volume" has more space that perceptrons can be different in. It might also be the case that this behaviour is simply caused by an insufficient number of training epochs, and that the perceptrons simply need more time to converge for such high numbers of dimensions.

3.1 Bonus: Stability for random labels

The stability of $w(t_{max})$ as a function of α was examined for randomly labeled data, as well as for data determined by a w^*. The results of this can be seen in figure 3.

This figure shows the maximum stability will be lower as more datapoints exists. This is to be expected, because when there are more datapoints there is also less space for the perceptron to be "far away" from the datapoints. This effect appears to be more influential for higher numbers of dimensions. This is a bit "tricky", as in the main experiment it was found that there was somehow "more space" in the higher dimensions, as the perceptron was further away from w^*. The combined conclusion of these effects seem to indicate that the number

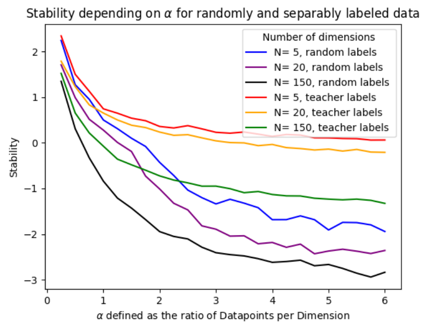

Figure 3: Stability for separably labeled data and randomly labeled data

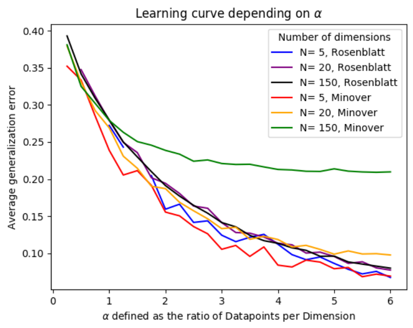

Figure 4: $\epsilon_g^N \alpha$ for Rosenblatt and Minover

of epochs was too low.

This graph also shows the maximum stability for randomly labeled data. This is much lower (i.e. worse) than in the data labeled according to w^*. This is because the data is not nicely separated, so every way the perceptron turns it will still be misclassifying part of the data.

3.2 Bonus: Rosenblatt vs. Minover

Another bonus experiment that was performed compared $\epsilon_g(\alpha)$ for the Rosenblatt algorithm against that of the Minover algorithm. The results are shown in figure 4.

The first thing to note about this graph is the missing values. For some of the lower α there was no value from the Rosenblatt algorithm. This is because in these cases there is a chance that with the original 0 vector classification is already perfect. If it comes out as such the calculation for the generalization error includes a division by 0, leaving it without value.

Regardless, some insights can still be drawn. For 5 dimensions we can see that minover tends to have a lower generalization error than Rosenblatt. This

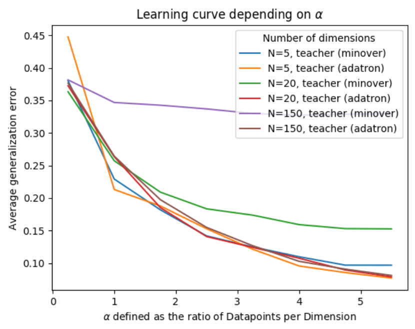

Figure 5: $\epsilon_g^N \alpha$ for Adatron and Minover

makes sense, as the ”middle” plane is likely to be closer to a random w^* than a ”random” plane in the model from figure 1. However, we see that for a large α this difference becomes minimal. This could be because as the number of datapoints decreases, the space of possible valid separation planes also decreases. This leaves no space to distinguish between the ”middle” of the space and a random separation plane from Rosenblatt.

For $N = 150$ dimensions we see this effect completely disappearing, where the minover algorithm performs generally poorly, especially compared to the Rosenblatt algorithm. This again seems to indicate that it did not have enough epochs to converge to an appropriate approximation of w^{opt}.

3.3 Bonus: Minover vs. AdaTron

Finally, in another bonus experiment we compare the behaviour of $\epsilon_g(\alpha)$ between the Minover and AdaTron algorithms. In figure 5 we can see that for large N, Minover does not perform so well anymore, while AdaTron still monotonously decreases the generalization error. This can be explained by the oscillations of the weight vector in the Minover algorithm that happen when choosing alternating data points with minimal stability to update from, while AdaTron is guaranteed to monotonously decrease the cost function at every update.

3.4 Conclusion

In general a conclusion can be drawn that minover has a lower generalization error than Rosenblatt. That means that it is more effective at it's purpose of finding a linearly separable rule.

In this case the features of the datapoints are not generated based on the linearly separable rule that is to be learnt, so there will always be an error in that sense.

The main weakness of the experiment did end up being the maximum number of epochs. When comparing the performance to the Rosenblatt this starts to make some sense. In the Rosenblatt algorithm $t_max = 100, P = 150$ gave the perceptron up to $100 * 150 = 15000$ Hebbian terms. In the minover algorithm each epoch only gives 1 Hebbian term, resulting in a total of 100 Hebbian terms. This proved to be acceptable for lower numbers of datapoints, but problematic for higher numbers.

We can conclude that Minover does find perceptrons of better stability, provided that enough training epochs are performed. A more appropriate stopping criterion could be determined by the angular change of the weight vector. If the vector seems to move minimally it could be conclude that convergence was reached.

Either way, training time with the Minover algorithm will be much longer than with Rosenblatt, but the result would also be better at generalizing.

4 Teamwork

Both Travis and Ivo worked on the code for this assignment, which was largely based on the code of the previous assignment. Both members contributed to the bonus problems and the report.