LU TP 22-11

Bachelor Thesis in Theoretical Physics

Dynamic Stopping for Artificial Neural Networks

by

Daniel Lizotte

Abstract

The growing popularity of Artificial Neural Networks (ANN) demands continuous improvement and optimization of the training process to achieve higher-performing algorithms at a lower computational cost. During training an ANN learns to solve a problem by looking at examples, and will iteratively go over a dataset to reach an optimal performance. Usually the user needs to define a fixed number of iterations before viewing the final result and assessing the quality of training. The goal of the project presented in this paper was to optimize the training process by defining an **automatic** stopping criterion, which arrests training when good performance is achieved but further training would yield diminishing returns. The two methods explored to achieve this were based on the Plateau Detection coefficient (C) and the Gradient Correlation coefficient (G). The hypothesis was that these could be used across different problems and different networks with consistently good dynamic stopping. Testing over multiple different ANN parameters revealed that $G = 1$ produced a good stopping criterion over almost all scenarios, whereas C presented some weaknesses.

FYTK02
Bachelor's Thesis (15 credits ECTS)
January 2022
Supervisor: Patrik Edén

Popular Science Article

“Artificial Intelligence” is a term that is now ubiquitous in media, entertainment, social media and even the engineering world. Most people are confused about what it is, where it is and what it’s made of. The term is even thrown around in philosophical debates, in which people attempt to give meaning to the abstract concepts of “artificial” and “intelligence”. So, I would like to introduce an alternative term for AI: the Artificial Neural Network (a.k.a. ANN). Using “Neural Network” is useful because it is more specific: we are talking about a network made of neurons. Just like the human brain, an ANN is made up of individual units (called neurons) which communicate with each other. When they are all working together, they can perform complicated operations much faster than one node on its own.

The reason why ANNs are so fast and flexible is because they undergo a training stage, where the network ”practices” on training points and in response ”tunes” the values of its nodes, which affects the final output. This Bachelor Thesis takes a closer look at two ways in which training can be made more efficient, cutting down on computational resources and user supervision. We want to avoid training for too long, but we don’t want to stop too early either. Finding the point where the training is just right in an **automatic** way we call **dynamic stopping**. We tested two methods for dynamic stopping, called Plateau Detection and Gradient Correlation.

After studying how these methods behaved in different scenarios, it was shown that the Gradient Correlation method was **better** at predicting reliable stopping points than the Plateau Detection method, and that it would be worth it to explore it more. After all, this is only the beginning of the study of this coefficient, and if it turns out to be a reliable way of stopping training in ANNs, its widespread use could mean advantageous improvements in efficiency of these networks.

Contents

1

Introduction

The popularity of Artificial Neural Networks (ANNs) has seen a significant increase in the past decades due to the impressive flexibility of these algorithms. From facial recognition [1] to art generation [2], ANNs have been able to complete tasks that were thought to be impossible or very difficult for a computer. The strength of these networks lies in the fact that they find solutions using data and with no need for the developer to specify what operations need to occur. In contrast to regular algorithms, ANNs are plastic and go through a process called training. It has been shown that after training, ANNs can find solutions to complex problems much faster than traditional solvers [3].

The effectiveness and performance of ANNs depends on the quality of the training process. Since training data-sets and computer resources are limited, there is a need to optimize this process. Specifically, the training time is crucial in determining the cost and benefit of a network. Seen as training is an iterative process, each iteration brings the network closer to the best solution to the problem. One challenge is to stop the training at the point of diminishing returns: it is desirable to use as few iterations as possible without missing out on a good solution.

The methods analyzed in this thesis paper have the potential to alleviate both these problems by defining an **automatic** stopping criterion to arrest the networks development at the optimal point. The two methods are called **Plateau Detection** and **Gradient Correlation** coefficient. In short, the first method tries to detect when the training performance starts to reach a plateau. The second is a novel method, which compares the updates of the ANN proposed by different data-points, to calculate a coefficient of correlation. The network training can be arrested when the value of this coefficient is sufficiently low. These methods were tested empirically using an ANN written in Python using the Keras and Tensorflow libraries [4][5]. The goal was to compare the two methods in a variety of training situations and see if under visual inspection, values of their key parameters could be used for dynamic stopping.

2

Theory

An ANN is made of computational units called "**nodes**". A node receives a number of inputs, either from the data or from a previous layer of nodes. Each node then takes the sum of all the weighted inputs and puts it through an activation function. Then the node sends out the result of its activation function to other nodes. This structure is summarized in figure 2.1.

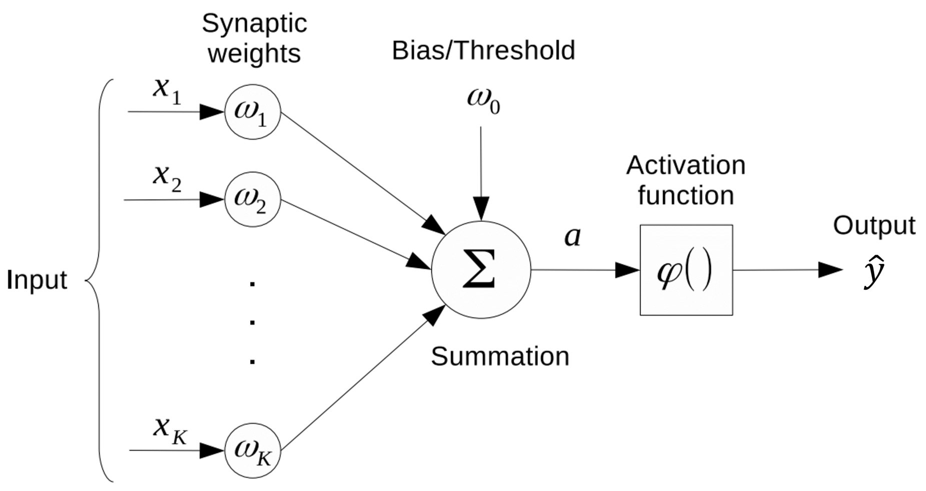


Figure 2.1: The structure of a node in an ANN. Figure from coursebook [6].

More precisely, the mathematical operation performed by a node is described in equation (2.1). Here x_k represents the different inputs, ω_k the weights by which the inputs are multiplied, φ is the activation function and $\hat{y}$ is the output of the node. Additionally ω_0 is a bias applied to a node.

$$a = \sum_{k=1}^{K} \omega_k x_k + \omega_0 \quad \Rightarrow \quad \hat{y} = \varphi(a) \tag{2.1}$$

Nodes can be combined together in intricate architectures where multiple layers work together to produce a final output. In figure 2.2 are some examples of so-called feed forward networks, in which data passes from source to output.

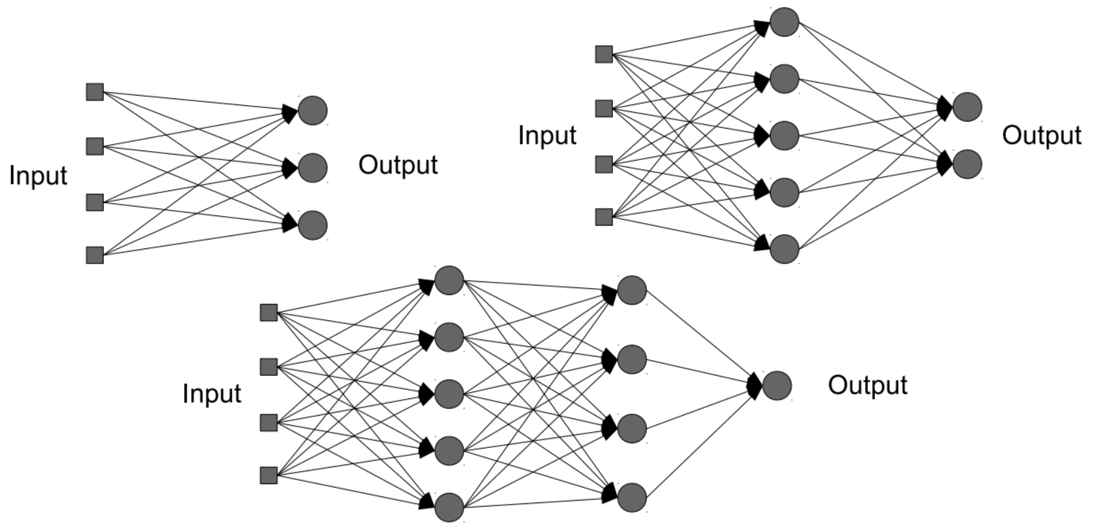


Figure 2.2: Feed forward networks with one, two or three layers of nodes. Figure from coursebook [6].

The weights between the nodes and the bias each node has, are the tunable parameters that determine how a network will transform inputs into outputs. By using methods such as *Stochastic Gradient Descent* [7] a network is trained over many examples of the task it is designed to complete, and adjusts its weights automatically to achieve the best accuracy.

2.1 Training and Gradient Descent

As mentioned above, the efficiency and performance of ANNs depend strongly on the training process. The goal of this process is to determine the best values for all tunable parameters, such that the network solves the problem with the least loss. The latter value is calculated using a loss function, which is selected based on the type of problem one is trying to solve, and will estimate a measure of how close the predicted outputs of the network are to the target data. For example, the Binary Cross-entropy loss is defined as follows:

$$E = \frac{1}{N}\sum_{n=1}^{N} E_n = -\frac{1}{N}\sum_{n=1}^{N}[y_n \cdot log\ \hat{y}_n + (1-y_n)\cdot log(1-\hat{y}_n)] \tag{2.2}$$

Where N is the number of data-points that the network trains on, also known as patterns, $\hat{y}_n$ is the output of the network for a pattern n and y_n is the desired output, also called the target, which is specified by the data-set. This loss is suitable for binary classification since y_n is either 0 or 1 and $\hat{y}_n$ lies in the open range between 0 and 1.

The network will compute the derivative ($g_k = \frac{\partial E}{\partial \omega_k}$) of the loss E with respect to each trainable weight ω_k and collect them into a vector called the gradient.

$$\boldsymbol{g} = \nabla_{\boldsymbol{\omega}} E \tag{2.3}$$

The *negative* gradient ($-\boldsymbol{g}$) "points" in the direction of quickest descent. It is also possible to define a gradient for each pattern n. This measure, seen in equation (2.4), is not directly used in training, but will become relevant for the calculation of the Gradient Correlation factor.

$$\boldsymbol{g}_n = \nabla_{\boldsymbol{\omega}} E_n \tag{2.4}$$

where E_n is defined implicitly in equation (2.3).

When the network is updated at the end of this training process, it effectively takes a "step" in the direction of quickest descent (the gradient) and all tunable parameters are adjusted. The size of this "step" can be changed by the user and is known as the **learning rate**. Equation (2.5) shows how weight updating is done every iteration i, by multiplying the gradient by the learning rate for each weight. The negative sign makes sure the gradient is pointing towards the minimum, not the maximum.

$$\omega_{i+1} = \omega_i + \mu \cdot (-\boldsymbol{g}_i) \tag{2.5}$$

where μ is the learning rate.

It is important to note that the size of the gradient itself decreases when one approaches the global optimum, leading to smaller and smaller steps. This means that while the network can quickly reach good performance which is *close* to the optimum, reaching the *true* optimum may take very long to reach, and could be practically impossible. In fact, because of the practical limitations (computer resources, time) it is undesirable to train over an excessive amount of iterations.

2.1.1 Stochastic Gradient Descent

When the gradient for every pattern (n) has been calculated and used for training, this is called an epoch. However, one epoch does not need to correspond to one weight update being performed. To increase the efficiency of the network and speed up training, the practice of **mini-batch updates** is often used. It consists of randomly dividing the training data-set $\mathcal{D}$ into parts called mini-batches $\mathcal{B}$. For instance, a data-set with $N = 200$ data-points could be divided into 4 equal batches of size $B = 50$. The only difference compared to regular training is that at the end of the first batch the network weights are being updated (a step is taken), which results in 4 iterations per epoch. More iterations means faster training, but it has a drawback when approaching the optimum: even though the batch elements are chosen randomly, the optimum for a batch is different from the global optimum. This means that the smaller the batch size, the more "unstable" the network will become close to the global optimum. Using new randomly defined mini-batches is called Stochastic Gradient Descent (SGD)[7] and it is a powerful way to speed up training.

2.2 Dynamic Stopping

While the most common methods of stopping the training of an ANN are to simply specify the total number of epochs or setting a threshold for the loss to be achieved, they are decided in an arbitrary way and rely on the interpretation of a user. The preferred way would be to have an automatic stopping criterion that can be applied across different types of networks and problems; we refer to this as **dynamic stopping**. The thesis aims to reproduce one method from a published Master thesis from Lund University [8] and testing a novel method developed for this project.

Dynamic stopping should not be confused with **early stopping**. The latter refers to solving the problem of generalization of an ANN. In short, generalization refers to how well the network performs at solving the general problem for new data generated under the same conditions as the training data. Every epoch the network performance is **tested** under this new data, known as validation set, to generate a validation loss. It has been shown that in general, validation loss will reach a minimum, and that after this point the network becomes "over-trained", leading to poor generalization [9]. A study from the Max-Planck institute [10] proposes that both problems of early stopping and dynamic stopping can be solved by studying the gradient at each point in training. Even though their proposed measure inspired the Gradient Correlation factor presented here, the implications for validation and generalization are outside the scope of this project.

2.2.1 The "plateau detection" method

This first method is derived directly from the Lund Univeristy master thesis titled "Childhood Habituation in Evolution of Augmenting Topologies"[8]. In his work, Moberg needed a simple method to determine when his network had achieved a stable training loss. More specifically, Moberg was comparing networks of different sizes and parameters, which would take a different number of epochs to reach their optimum. This meant that there would always be resources wasted during training - which lead to the development of the plateau detection method. Simply put, the method consists of looking at the training loss over multiple epochs and extracting the loss for two adjacent intervals (W_1, W_2). One uses intervals instead of individual loss measurements because the loss of the network fluctuates. Hence, the method takes the average of the loss in the two intervals and looks at the difference. If this difference is small enough the training is stopped. In equation (2.6) one can see that the threshold is set as a proportion c of the average values in the intervals.

$$|\frac{1}{W_1}\sum_{k=0}^{W_1} E_{i-k} - \frac{1}{W_2}\sum_{m=0}^{W_2} E_{i-W_1-m}| < c \cdot \frac{(\frac{1}{W_1}\sum_{k=0}^{W_1} E_{i-k} + \frac{1}{W_2}\sum_{m=0}^{W_2} E_{i-W_1-m})}{2} \tag{2.6}$$

In the equation above, W_1 and W_2 represent the sizes of the two intervals, E is the loss at a certain epoch, (i, k, m) are indices for the summation and c is the coefficient of proportionality, which will from now on be called the **Plateau Detection coefficient**. Figure 2.3 shows visually how the intervals are taken from the training error curve.

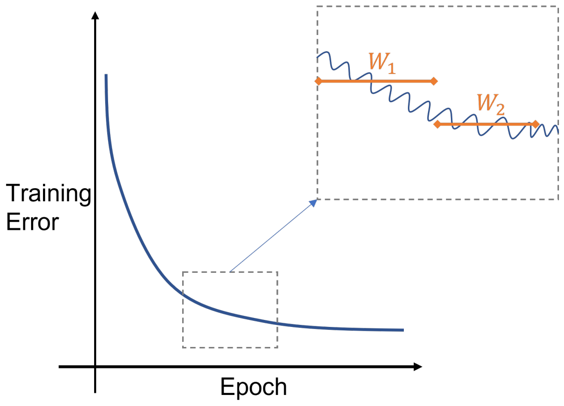


Figure 2.3: Visual representation of where the intervals are taken.

As per its current definition, the Plateau detection method needs 3 user-defined arguments: W_1, W_2 and c. However, W_1 and W_2 only rarely need to be different, and it is quite easy to determine the minimum width needed to smooth the random fluctuations of training. Hence, the most important factor to determine is c. Effectively, c indicates the necessary relative error between the two interval averages. Setting $c = 0.1$ will stop the training after the percentage error is lower than 10% .

One can also rearrange equation (2.6) to extract a measure C at each training step, as seen in equation (2.7). An important distinction is that C is calculated at **every epoch** of training, while c refers specifically to the value of the stopping criterion.

$$C = \frac{2|\frac{1}{W_1}\Sigma_{k=0}^{W_1} E_{i-k} - \frac{1}{W_2}\Sigma_{m=0}^{W_2} E_{i-W_1-m}|}{\frac{1}{W_1}\Sigma_{k=0}^{W_1} E_{i-k} + \frac{1}{W_2}\Sigma_{m=0}^{W_2} E_{i-W_1-m}} \tag{2.7}$$

In essence, this is similar to taking the slope of the loss between two points. Once the relative slope is low enough, we consider the network to have converged.

It is to be noted that one scenario where this method could prove ineffective is when the network hits a region of near constant loss. Since it has no way of knowing at which point the network has reached a global minimum, it will simply stop when the slope of the loss curve is ”flat enough”.

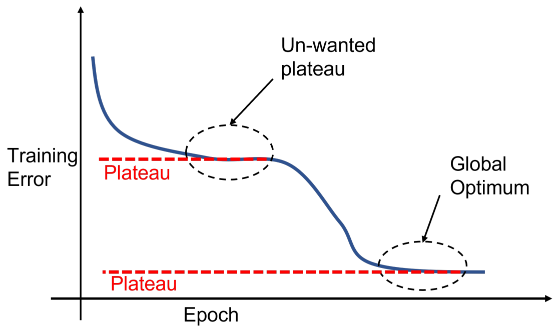


Figure 2.4: Visual representation of where the Plateaus are detected. In this case, the network would stop much earlier than the global optimum.

2.2.2 The Gradient Correlation factor

One can visualize the training process by looking at a two-dimensional space, considering only at the first two weights (ω_1, ω_2), as seen in figure 2.5.

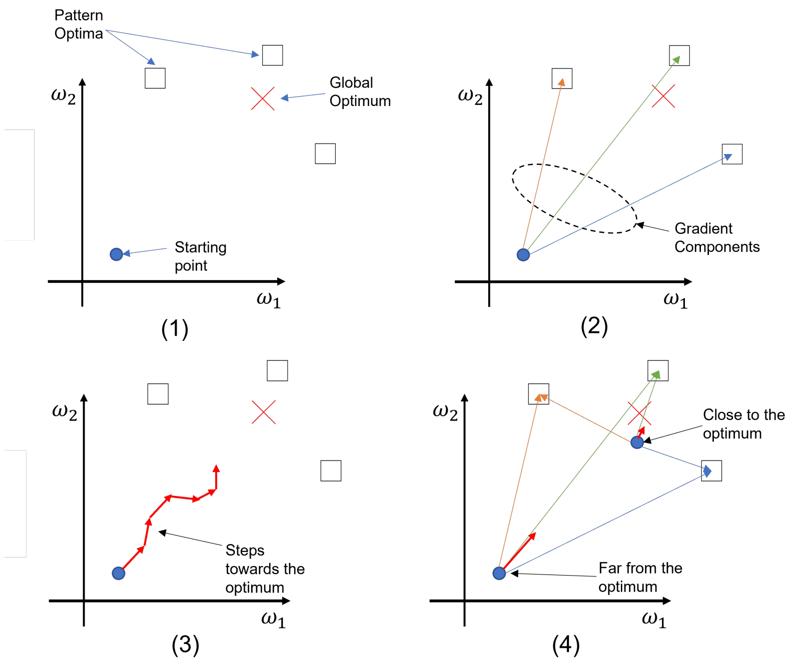


Figure 2.5: The diagrams describe the process of gradient descent. In every one, the x-axis represents the value of one weight, the y-axis the value of another. In practice, the weight space has as many dimensions as there are tunable parameters.

(1) The network starts at a certain point in weight-space and the objective is to reach the global optimum situated between the pattern optima. (2) The loss relative to each target determines the terms included in the gradient. (3) The gradient is a vector sum of all these terms. Every iteration (or epoch) the weights are updated to take a step towards the global optimum. (4) Far from the optimum the gradient terms will be strongly correlated. However, near the global optimum they will become less correlated. Precisely at the global optimum, one expects that all gradient terms will become perfectly anti-correlated (add up to zero).

The **Gradient Correlation** method makes use of the individual gradients to calculate the correlation between the different gradient terms (one for each pattern), and is a novel field of study. The gradient correlation coefficient (G) is defined as

$$G = \frac{|\Sigma_n \boldsymbol{g}_n|^2}{\Sigma_n |\boldsymbol{g}_n|^2} \tag{2.8}$$

where $\boldsymbol{g}_n$ is the gradient for pattern n defined in equation (2.4). It has one value for each trainable parameter of the network and "points" in the direction of the optimum for each pattern n. Similar to the variance, this coefficient gives a measure of how correlated the gradients are at every point in training. Intuitively, one can observe that at the beginning of training the gradients for all patterns point in roughly the same direction, which will result in a high G. As the network approaches the global optimum, the gradients for individual patterns start to point in different directions and become more uncorrelated, reducing the value of G. In fact, **at the global optimum** one expects that the sum of all gradients be zero, which means that G will also tend to zero.

$$\Sigma_n \boldsymbol{g}_n \longrightarrow 0 \quad \Rightarrow \quad G \longrightarrow 0$$

However, this is not very useful as G will go to zero as slowly as the network is converging, and we are interested in stopping the network training slightly before the global optimum is reached. If one expands the sum on the numerator of the definition of G, the coefficient can be re-written as follows:

$$|\Sigma_n \boldsymbol{g}_n|^2 = \Sigma_n |\boldsymbol{g}_n|^2 + \Sigma_n \Sigma_{m \neq n} \, \boldsymbol{g}_m \cdot \boldsymbol{g}_n \tag{2.9}$$

$$\Rightarrow G = 1 + \frac{\Sigma_n \Sigma_{m \neq n} \, \boldsymbol{g}_m \cdot \boldsymbol{g}_n}{|\Sigma_n \boldsymbol{g}_n|^2} \tag{2.10}$$

Where n and m are indices of two gradients from **different** patterns. One can observe that if all gradients are on average orthogonal, the sum of scalar products $\boldsymbol{g}_m \cdot \boldsymbol{g}_n$ goes to zero. Hence, the second term of the sum goes to zero and G goes to 1. This means that G will reach the value of 1 at some point in training independent of the number of patterns or network size. Thus, an interesting case is when the gradients are **uncorrelated**, which occurs after some training but before the global optimum.

If $G = 1$ is a good value to stop training it could have wide range of applicability to different networks and problems. One of the weaknesses of this method occurs when the number or patterns is very low, which can cause large fluctuations in the value of G. That is why it is best the calculate G over all patterns in the data-set $\mathcal{D}$ even when using mini-batches.

3

Data and Methods

To investigate the validity of the two dynamic stopping criteria, a problem and a model for the ANN were chosen. In this case, the problem is a simple binary classification in two dimensions, where two classes are generated from a gaussian distribution into two overlapping clouds. This allows the problem to be solved by a simple Multi Layered Perceptron (MLP) network, where the general structure is the following: Input layer (2 nodes), Hidden layer (n nodes), and Output layer (1 node). The data-set and the network are constructed in a Python[4] script using the Keras and Tensorflow[5] libraries.

3.1 Gaussian cloud binary classification

As can be seen in figure 3.1, the standard definition of the problem includes an overlap of the two clouds, which affects the maximum efficiency of the network. Because of the overlap, the network is never able to separate the two classes perfectly with one line. The size and overlap of the two clouds was changed to modify the problem's difficulty. This is the main reason why a synthetic data-set was chosen instead of using real data-sets.

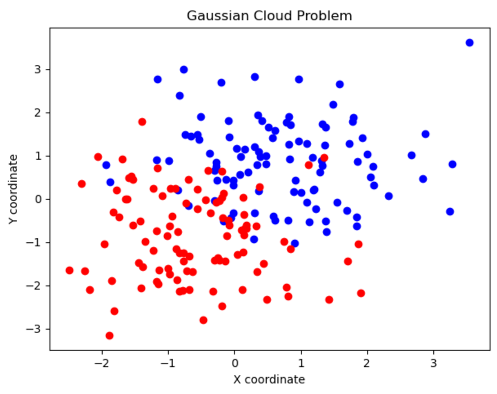


Figure 3.1: Visualization of the gaussian cloud problem. Class 0 is in red and class 1 is in blue. The distance between the centres of the distributions is 1.6 and the standard deviation of both distributions is 1.

The data-set seen above is the **default** used in all trials, except when the problem difficulty is varied. The two clouds have a standard deviation width parameter of

1 and a separation parameter of 1.6, to produce an overlap, with $N = 200$ data-points. The best decision boundary for this problem is a straight line. To change the problem difficulty, the shape of the separation boundary was made more complex. The first step was to enlarge one of the two clouds, which now has a value of 2. Then the distance between the clouds was altered to affect the difficulty. As one can see in figure 3.1, when shifting the clouds closer to each other, the decision boundary that solves the problems turns from a parabola into a circle.

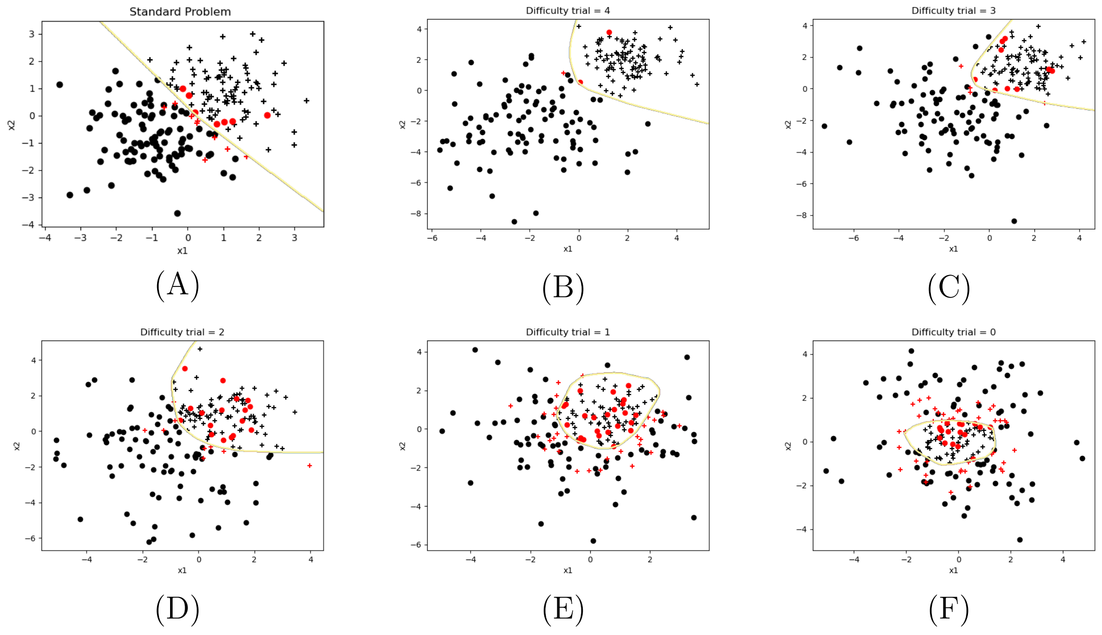


Figure 3.2: The datasets are ordered from A to F in increasing difficulty, as can be seen by the best decision boundary, which gets more complex.

Another consequence of the clouds moving closer together is that the maximum accuracy for the fully trained network will be lower, because more points are overlapping and will be mis-classified.

3.2 Python script

The Python script was responsible for generating the dataset, constructing the MLP and training it over multiple epochs. The biggest asset of the code is the Tensorflow library [5], which calculates the gradient of the network automatically using numerical differentiation. Using the same equations as described in the theory section it was possible to calculate the Gradient Correlation coefficient and the Plateau Detection Coefficient at every epoch of training. Systematic testing of variables made it possible to draw conclusions on the effectiveness of these coefficients in enabling dynamic stopping.

3.3 Testing parameters

To look at a wide range of scenarios, the network was trained to solve the gaussian cloud problem under various conditions. The objective was to observe whether it is possible to use the same value for G or c and obtain good performance across different parameters. These parameters were:

- Problem difficulty - *Changing the parameters of the gaussian clouds*
- Network size - *The number of hidden nodes*
- Learning rate - *The size of steps during weight update*
- Batch size - *The number of patterns used for one weight update*
- Dataset size - *The total number of datapoints*

Table 3.1 shows all the parameters that were held constant and all the default parameters. While one of the parameters is **varied** for investigation, the others remain at their **default value**.

Constant Parameters	
Total Epochs	200
Activation Function	ReLU
Optimizer	Adam
Intervals $W_1 = W_2$	5

Default Parameters	
Learning Rate	1E-3
Datapoints (N)	200
Batch Size	50
Number of hidden nodes	200
Cloud Distance Parameter	1.6
Cloud Width Parameter	1

Table 3.1: The constant parameters are the same throughout testing and only one of the default parameters is changed at a time.

The **hypothesis** was that the gradient correlation coefficient (G) and/or the plateau coefficient (C) could be set to a static value to obtain consistently good dynamic stopping. In particular, it was investigated whether $G = 1$ agreed with stopping points that seemed reasonable by visual inspection of the loss. The corresponding values of C were found, and it was tested if a single threshold value of c also could find reasonable stopping points.

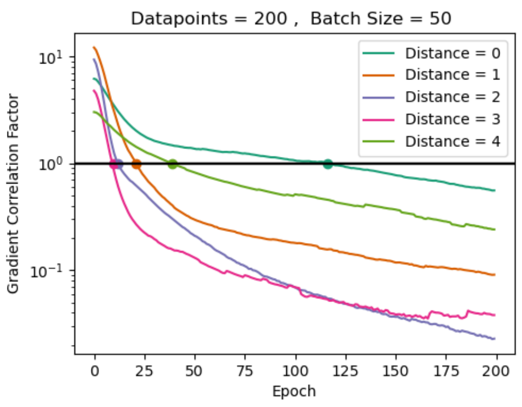
4

Results

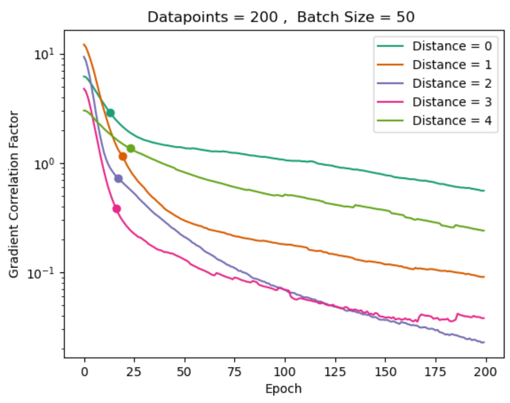
The data collected by running the simulations is presented here. Throughout each of the different trials, the graphs are arranged as follows: G over epoch on top, the loss over epoch in the middle and the value of C over epoch on the bottom. The left column represents the case where $G = 1$ and is therefore ordered (A) (B) (C). The right column represents the same curves, but this time the stopping criterion $C = c$ is set, and has labels (D) (E) (F). The dots mark the epochs where $G = 1$, or the epochs for a chosen constant value of C. Thus, the dots in different graphs relate the curves to each other. The values of the varied parameter are indicated in the legends of each figure, while all other variables are held **constant** or at **default** as described by table 3.1.

4.1 Problem Difficulty

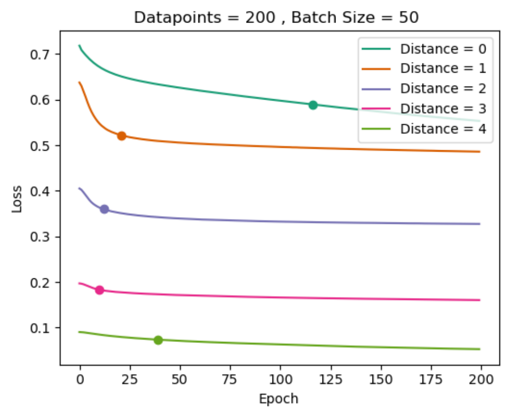
Figures 4.1A-F display the result of varying the problem difficulty as outlined in the previous section and seen in figure 3.1. As expected, reducing the distance results in a much higher minimum loss. Still, all networks reach the value of $G = 1$ within 200 epochs. By looking qualitatively at figure 4.1B, it appears that the stopping point determined by $G = 1$ is a pretty good indicator for when the network has converged. The parameter c was given the average value of 0.033 based on the stopping points generated by $G = 1$. Under this stopping criterion, the points are very similar to the Gradient Correlation method, except in the hardest problem (distance = 0), where the Plateau coefficient suggest stopping much earlier.



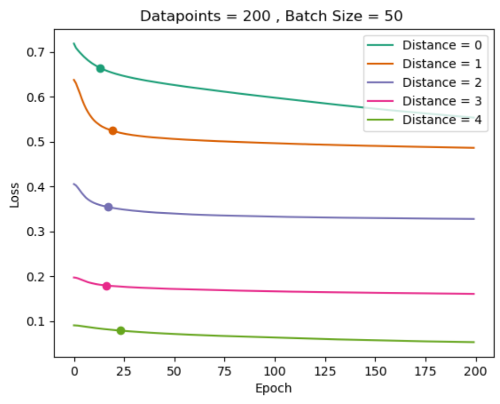
(A)



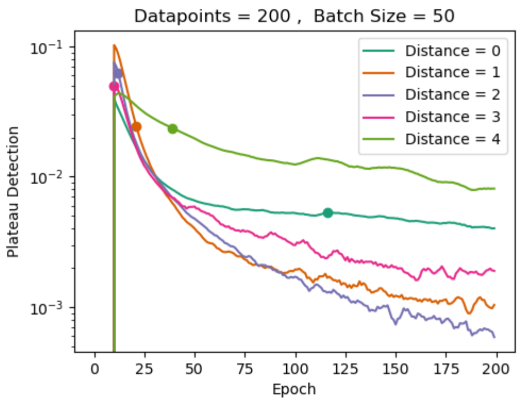
(D)



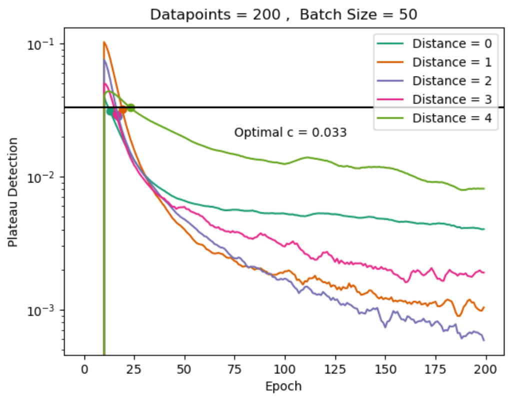
(B)



(E)



(C)

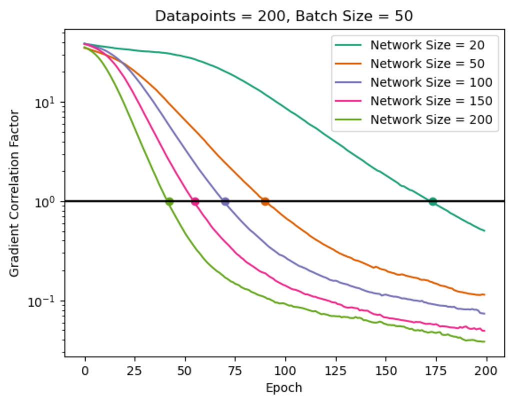


(F)

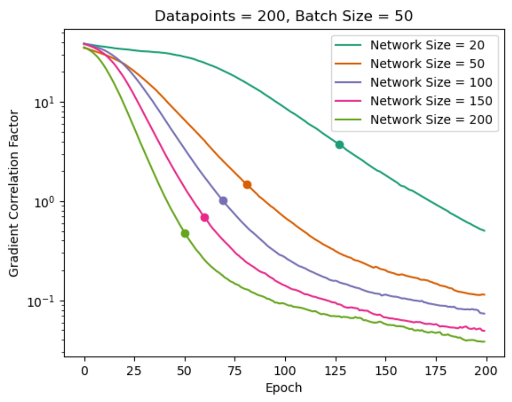
Figure 4.1: Colour curves represent different problem difficulty (Distance = 0 ⇒ highest difficulty). On the ***left*** *column the stopping points according to* $G = 1$ *are highlighted. On the* ***right*** *column are the stopping points for* $c = 0.033$*. Both methods are roughly in agreement, except for the* ***Distance = 0*** *trial.*

4.2 Network Size

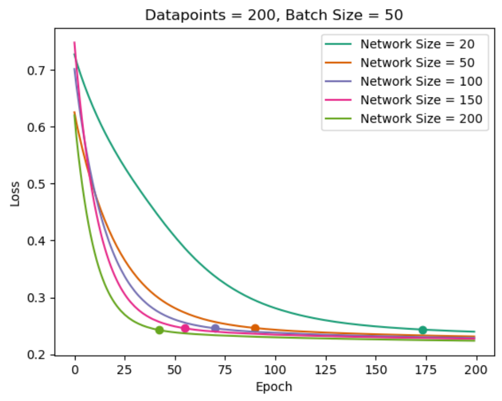
Moving on to testing the network size, this simulation provided some of the smoothest results of the investigation. Over a wide range of inputs, the loss, G and C curves show that **bigger** networks solve the problem faster than small ones. Figure 4.2B shows that $G = 1$ represents a qualitatively reasonable stopping point and that training beyond this point yields diminishing returns. However, the result for C does not behave as expected. It appears that a static value for G does not correspond to a static value for c. Setting c equals to its average $c = 0.015$ causes training of the smallest network to stop later and of the biggest network to stop earlier, as seen in figure 4.2E.



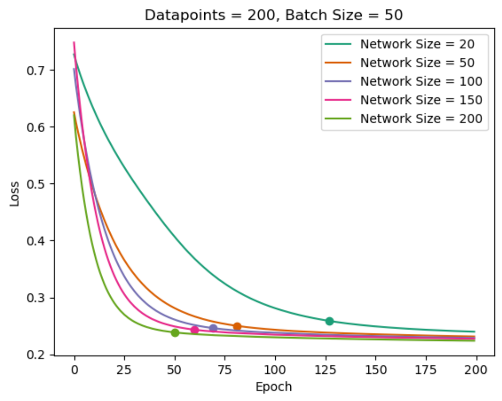
(A)



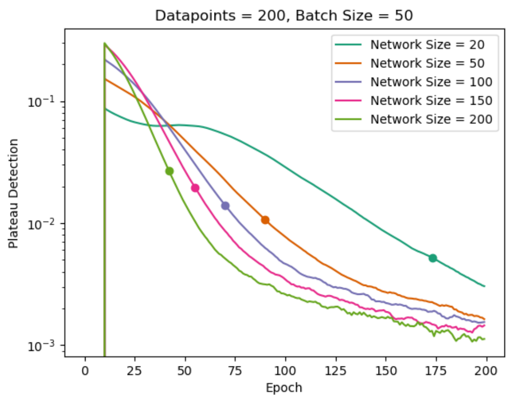
(D)



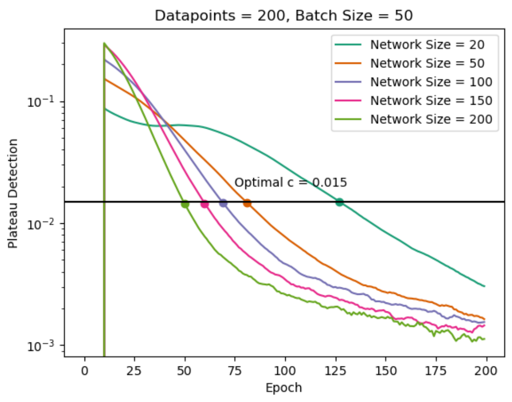
(B)



(E)



(C)

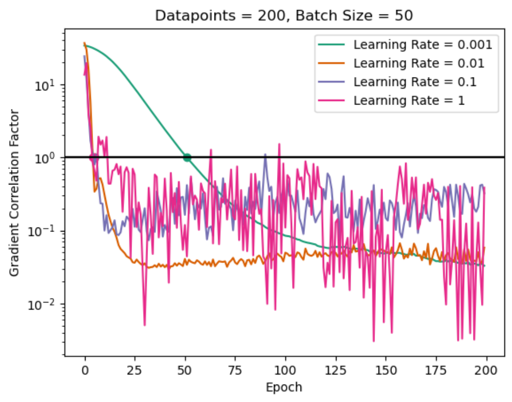


(F)

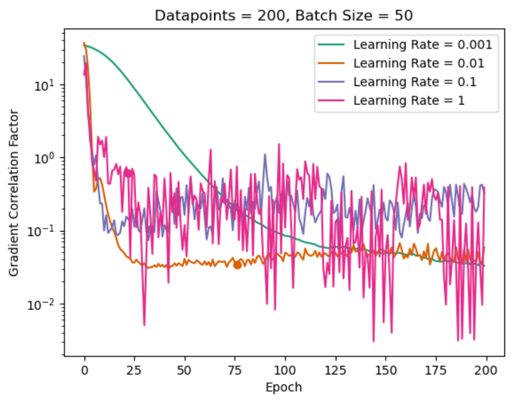
Figure 4.2: Colour curves represent different network size (Network Size = 200 $\Rightarrow$ largest). On the ***left*** *column the stopping points according to $G = 1$ are highlighted. On the* ***right*** *column are the stopping points for $c = 0.015$. Both methods are roughly in agreement.*

4.3 Learning Rate

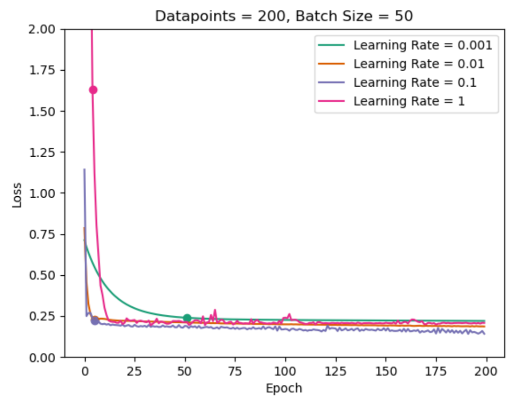
The learning rate was studied within the large range from 0.001 to 1, to highlight its effects on training. Learning rates smaller than this range produced no convergence within 200 epochs whereas higher learning rates produced heavy fluctuations. The test with learning rate 1 was the only example in this report where $G = 1$ **failed** to identify a reasonable stopping point – for all smaller learning rates G appears to be a good predictor for convergence. The large learning rate also highlights the challenge to find a good window size for the plateau method. First, C cannot be used to detect a stopping point before enough epochs have been run to create the windows (W_1, W_2), which creates a so-called "dead zone". However, with small window sizes fluctuations in loss can cause C to plummet to 0, creating a risk for false identification of a stopping point. For example, in figure 4.3D C drops so quickly that the chosen stopping point actually has lower c then the set $c = 0.007$. In this case this still results in an acceptable stopping point, as seen in figure 4.3E, but that may not be true for the general case.



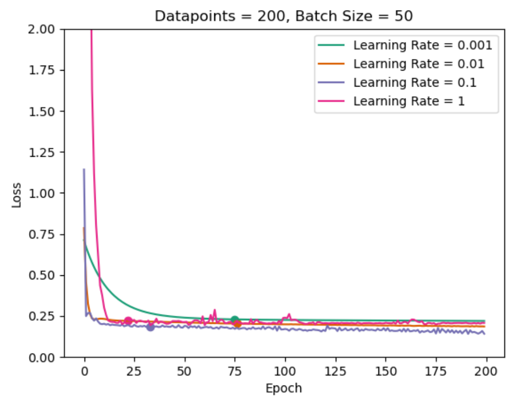
(A)



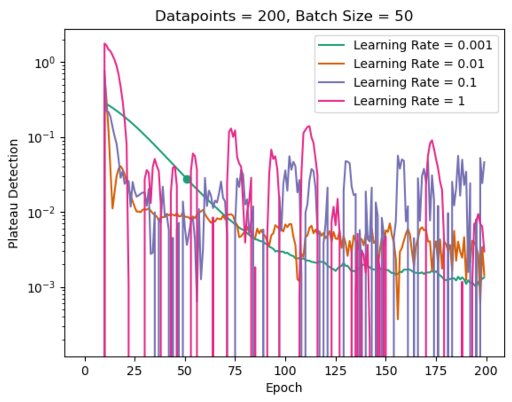
(D)



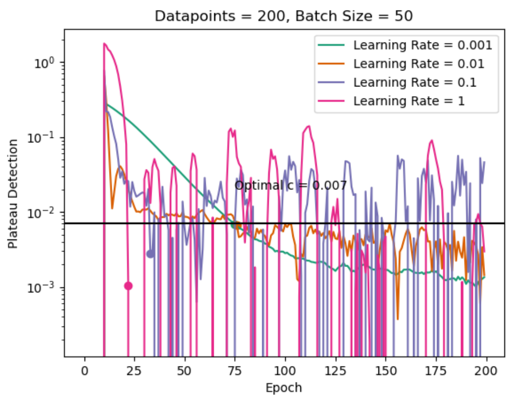
(B)



(E)



(C)

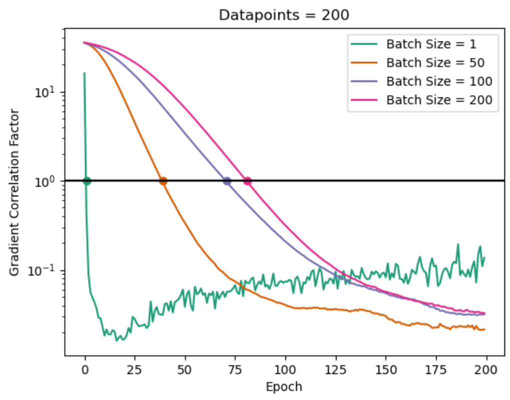


(F)

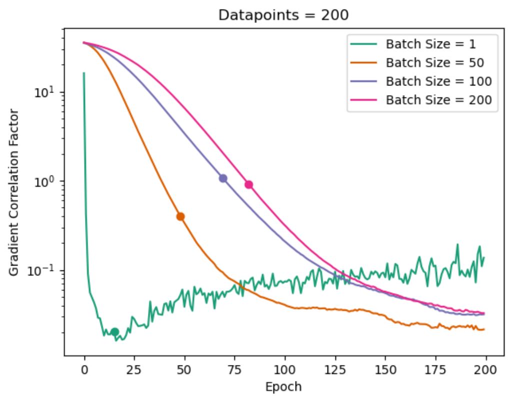
Figure 4.3: Colour curves represent different learning rate (Learning rate = 1 $\Rightarrow$ highest). On the ***left*** *column the stopping points according to $G = 1$ are highlighted. On the* ***right*** *column are the stopping points for $c = 0.007$. Learning rate = 1 is causing difficulties for both dynamic stopping methods.*

4.4 Batch Size

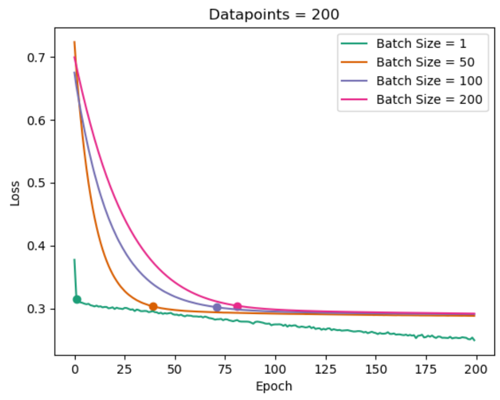
The study of Stochastic Gradient Descent showed a different curve when using a batch size of 1, which means that the network is updated after training on every single data-point. The network was able to converge in few epochs, since every epoch corresponds to 200 iterations. The graph of G, in figure 4.4A, reflects this. Since G is calculated at the end of every epoch, the network is now much closer to the optimum and G decreases rapidly. The subsequent fluctuating behaviour can be explained by the fact that updating one weight at the time is not anymore a straight path towards the global optimum, but rather a random chase in the direction of each individual pattern .The variable C on the other hand is shown to suffer stability under the fluctuations of the loss, seen in figure 4.4C. Additionally, the stopping point detected by $G = 1$ in batch-size 1 falls within the "dead zone" of C and is therefore not found in that curve. Having a batch size as large as the data-set delays the convergence because the network has undergone less iterations, and it gives the most smooth and stable calculation of each of the parameters (G, C and loss).



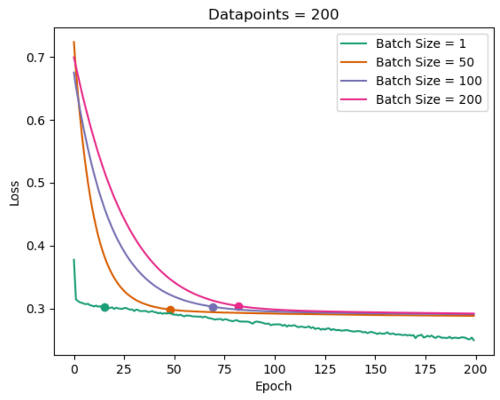
(A)



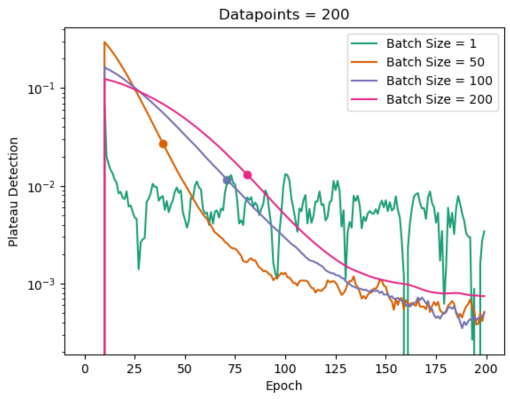
(D)



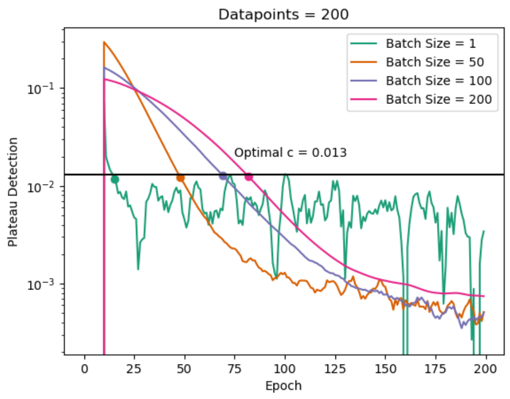
(B)



(E)



(C)

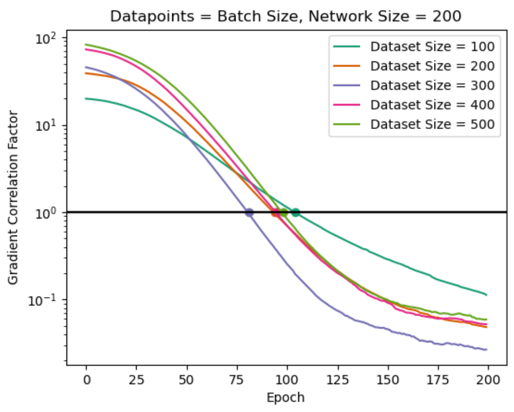


(F)

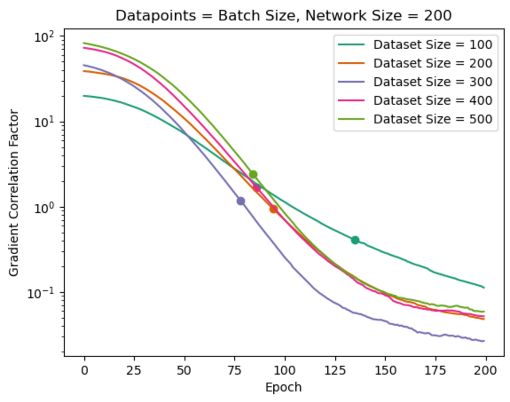
Figure 4.4: Colour curves represent different batch sizes (Batch size = 200 ⇒ largest). On the ***left*** *column the stopping points according to* $G = 1$ *are highlighted. On the* ***right*** *column are the stopping points for* $c = 0.013$*. Batch size = 1 has no identified stopping point on figure (C) since it happens too early ("dead zone"). Apart from the latter, the two methods are roughly in agreement.*

4.5 Total Data Size

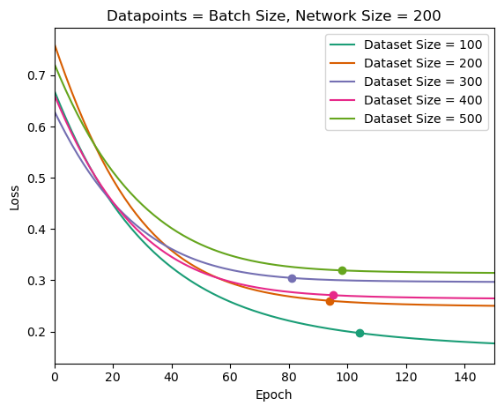
To preserve the same amount of iterations between all problems, the batch size was set equal to the total number of training points. The loss over epoch curve shows that changing the amount of data does not seem to affect the difficulty of the problem since it is in the simplest formulation (same-width clouds), but it can affect the minimum achievable loss. In figure 4.5A one can observe that the starting value of G is roughly the same as the total data-set. Despite this, all curves follow a similar trend and $G = 1$ occurs at similar epochs. Once again, one can see that $G = 1$ is located at a reasonable stopping point for the function. For the values of c, they also seem to cluster around an average, except for the smallest data-set. It could be argued that the smallest data-set is easier to solve as there are not so many points overlapping between the two clouds.



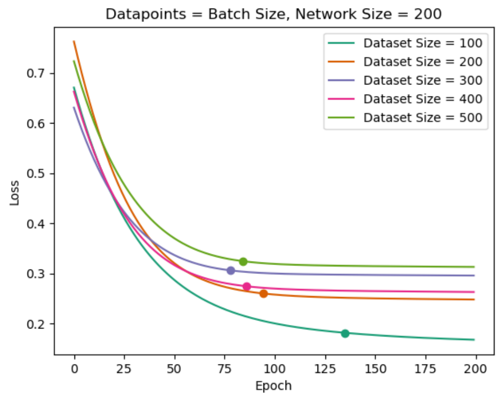
(A)



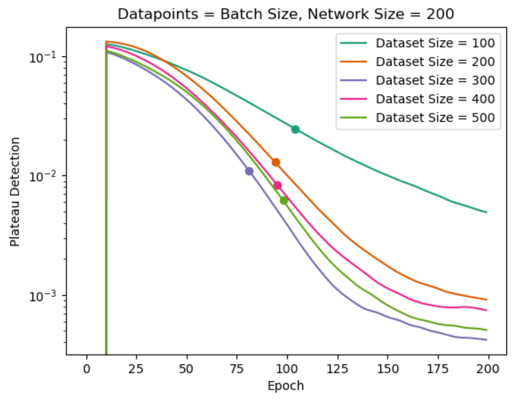
(D)



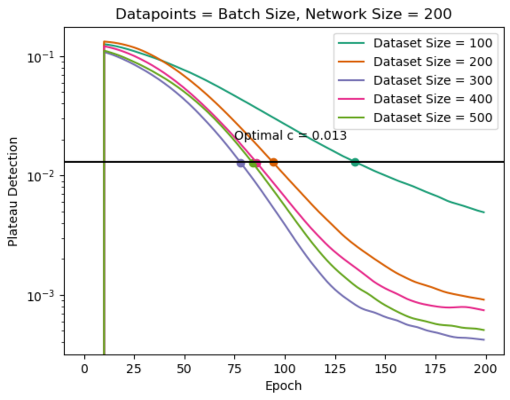
(B)



(E)



(C)



(F)

*Figure 4.5: Colour curves represent different data-set size (Dataset size = 500 ⇒ largest). On the **left** column the stopping points according to $G = 1$ are highlighted. On the **right** column are the stopping points for $c = 0.013$. Both methods roughly agree on the stopping points, except some discrepancy in Data-set size = 100.*

4.6 Results Summary

The results for each testing parameter can be summarized as follows:

1. The way **problem difficulty** was implemented, it did not always affect how fast a network converged, but it affected the minimum loss. The threshold $G = 1$ yielded relatively stable results and corresponding c were fairly clustered around an average of $c = 0.033$

2. **Network Size** directly affected how fast the problem was solved, where bigger networks achieved better performance in less epochs. Choosing $G = 1$ or the corresponding average $c = 0.015$ gave satisfactory stopping points.

3. Varying the **learning rate** to increase fluctuations revealed weaknesses in G and C. This is expected since the learning rate directly affects the size of the gradient. However, C was shown to be even weaker, failing to show convergence if it happens before a "dead zone" (10 epochs) and being unstable when the loss fluctuations are to large to be smoothed over intervals W_1 and W_2.

4. The experiment on **batch size** revealed that G is mostly sensitive to numbers of iterations. If the batch size is equal to 1, the network weights are updated 200 times for each epoch, causing it to converge more quickly and reducing the value of G sooner. The corresponding curves of C are again shown to be weak in the presence of fluctuations in the loss.

5. The **total data size** only affected the starting point for G, which is roughly equal to the data-set size. Otherwise, all solutions cluster around one point. The smallest data-set is expected to have better minimum training loss, since there are less data-points in the overlap region between the two clouds.

4.7 Discussion

The results indicate that a common G threshold is reliable to detect convergence in a wide range of situations, while the value of C fluctuates more. As seen in many figures, G varies rather rapidly with respect to epochs around $G = 1$. This adds benefit to the chosen value, since around this value the stopping epoch is fairly insensitive to small variations in G. Since the gaussian cloud was a relatively simple problem, increasing the learning rate to big values did not affect the training very negatively. However, as expected that for any problem, having extreme learning rates is undesirable for a stable solution, and as long as the solution is stable, G will be stable as well.

On the other hand, C was revealed to have some vulnerabilities. Large fluctuations in the loss can cause this method to miss the optimum completely. This effect could be dampened by increasing the size of the intervals to average over, but this introduces another problem: it increases the size of the "dead zone" at the beginning of the training, where not enough epochs have been acquired to compute the difference. Even at its most stable in figure 4.5E, the optimal positions indicated by the average c are not necessarily better than the ones given by $G = 1$ in figure 4.5B. Despite its simplicity, it still requires more arguments to define a stopping criterion for the network compared to $G = 1$, since one needs a threshold c and interval sizes W_1 and W_2.

Of course, one needs to acknowledge that the results were conducted on a simple 2-D problem with an MLP, and that they don't necessary apply to other problems or types of networks. This could explain the surprising result that larger networks solved the problem faster, as seen in figure 4.2B. This appears counter-intuitive, since a larger network has more parameters to train, and is in contrast with the findings of Moberg [8]. Given that the default gaussian cloud problem can be solved with a 2-node MLP, one explanation could be that since all network sizes used far exceeded this number, larger networks could reach the solution faster simply because they had more randomly initialized nodes.

One of the biggest difficulties in drawing the conclusions for this project is in fact the evaluation of the *extent of convergence*. In this case, the "good stopping point" was determined by balancing the difference between achieved loss and minimum loss, against the relative time saved (in epochs) compared to training for all 200 epochs. Despite this being highly subjective, it is still difficult to train any network without some extent of user-defined parameters. An interesting extension to the project would include comparing a quantitative measurement of the extent of convergence to G. If one wanted to optimize the generalization of the network, the point of minimum validation loss would indicate an **objective** optimum stopping point, as proposed in [10], which could then be compared to the value of G. The calculation of G adds a small computational cost at every epoch, needing to calculate the sum of vectors etc., but this could be considered negligible compared to the long-term benefits of reliable dynamic stopping, such as reducing "wasted" computational time.

5 Conclusion

In conclusion, the systematic testing of parameters revealed that the gradient correlation factor (G) could be a good candidate for dynamic stopping of ANNs, ensuring that training comes reasonably close to convergence, without losing time on excessive epochs. In the experiments shown, it is more flexible than the plateau detection method as long as the learning rate is not set too high. However, in the volatile case of high learning rate, plateau detection also fails at detecting a good stopping point reliably. The significance of setting $G = 1$ is observed, as it seems to indicate the point of diminishing returns in training. This can be explained by understanding that $G = 1$ means that the gradients of the network are uncorrelated. At this point, the network is taking steps in random directions to find the optimum. This is less efficient than following a strongly correlated gradient. Hence, after $G = 1$ the network trains at a slower pace. In essence, the project represents a proof of concept for the gradient correlation factor, and it shows promising results. But a lot more research is needed to determine whether this measure will work across different problems and network types.

Bibliography

[1] P. Kaur, K. Krishan, S. K. Sharma, and T. Kanchan, "Facial-recognition algorithms: A literature review," *Medicine, Science and the Law*, vol. 60, no. 2, pp. 131–139, 2020. PMID: 31964224.

[2] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial networks," in *NIPS*, 2014.

[3] M. Dissanayake and N. Phan-Thien, "Neural-network-based approximations for solving partial differential equations," *communications in Numerical Methods in Engineering*, vol. 10, no. 3, pp. 195–201, 1994.

[4] Python Software Foundation, "Python language reference, version 3.8." Available on www.python.org, 2021.

[5] Tensorflow Software Foundation, "Tensorflow api reference, version 2.7." Available on www.tensorflow.org, 2021.

[6] P. Edén and M. Ohlsson, *Lecture Notes on Introduction to Artificial Neural Networks and Deep Learning (FYTN14/EXTQ40/NTF005F)*. Lund University, 2020.

[7] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, p. 400–407, 1951.

[8] A. Moberg, "Childhood Habituation in Evolution of Augmenting Topologies (CHEAT)," *Master Thesis*, Aug 2020.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Machine Learning Basics*, ch. 5. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig, "Early stopping without a validation set," *arXiv preprint arXiv:1703.09580*, 2017.