Effect of parameter tuning on reducing the number of queries required to perform model stealing

Floris van Veen
Supervisor(s): Chi Hong, Jiyue Huang, Stefanie Roos
EEMCS, Delft University of Technology, The Netherlands
June 19, 2022

A Dissertation Submitted to EEMCS faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering

Abstract

Model extraction attacks are attacks which generate a substitute model of a targeted victim neural network. It is possible to perform these attacks without a preexisting dataset, but doing so requires a very high number of queries to be sent to the victim model. This is otfen in the realm of several million queries. The more difficult the dataset, the more queries required to gain an accurate substitute model. Through each state-of-the-art model extraction algorithm, one thing that is not thoroughly optimised are the hyperparameters of the models, and optimizing them has been found to have a strong impact on accuracy of the substitute model. To attempt to reduce the number of queries required, research has been done to find the effects of optimizing hyperparameters for both MNIST and fashion-MNIST datasets. This is done through grid search and random search. The results show that proper hyperparameter tuning can reduce the number of queries required to perform model stealing if they are not already optimized. Examples include requiring 125000 + queries to achieve 95% accuracy for the MNIST dataset with some hyperparameter combinations to only requiring 15000.

1 Introduction

As machine learning and deep neural networks are being used in more applications than before, it is becoming increasingly important to ensure that they are all working as intended, and that malicious parties cannot exploit issues that are found. One such issue is that of adversarial attacks [11]. There are multiple types of adversarial attacks [30]. The most common type of adversarial attack is evasion attacks that can create images which avoid correct classification from the neural network [12], while being nearly indistinguishable from the original image [28]. Another important type are model extraction attacks, which allows an attacker to make a substitute 'copy' of a model which has been made available to query [30]. A copy of this victim model can be made by having access to the same dataset and training another neural network from these samples, but model extraction allows attackers to create a 'copy' without any prior training set [29]. The only requirement is that the attacker can make queries to the victim.

Model stealing attacks may be done for many different reasons [13]. These reasons can include people wanting to make use of the victim model, to competition, to utilising it for monetary gain [23]. Different attackers can have different goals, and that will correspond to different types of attacks.

This leads us to the research question: *How can we reduce the number of queries which are required to perform model stealing?*

In this work, we look at the ways that model stealing are applied, and methods in which the number of queries which are required to perform them can be reduced. We study and apply methods to tune an existing data free model extraction algorithm DFME [29] to require fewer queries in order to gather an accurate copy of the victim model. This is done through analysis of the existing algorithms, and on any aspects which are either less researched or could be improved. Considering the time constraints given, the chosen methods for reducing the number of queries must also take that into consideration.

- First, the information which is available is discussed, and how they relate to model stealing attacks
- Secondly, the problem is formally described and the potential solutions are discussed
- A methodology which is used to find the optimal solution to the parameters is detailed.
- Our contributions to the research question are discussed
- The results of the research is displayed, discussed and evaluated
- Finally, improvements that can be performed in the future are introduced, and anything that could not be performed during the experiment for any reason.

2 Related work

As of now, there are relatively few robust model stealing algorithms, and each use slightly different theories to create substitute models. Here, we discuss these algorithms and other areas of research which are relevant to the theory used in model stealing.

2.1 Data Free Model Extraction

DFME [29] is a model stealing algorithm published in 2021, and primarily made use of the SVHN [25] and CIFAR-10 [18] datasets. Truong et al. (2021) [29] have created a model stealing algorithm that makes use of a generator that takes in a vector of random noise, and utilises a resulting image to train a student model. Zeroth order gradient estimation is used to approximate the victim dataset and improve the generator's generated images. A single loss function is used to improve the results of the generator and student models, though they both aim to do different things with it. Truong et al. (2021) [29] have managed to gain a model with 99% victim accuracy on the SVHN dataset with a query budget of 2 million queries, and a 92% accuracy with regards to the victim model on the CIFAR-10 dataset with a query budget of 20 million. It has been found that both the choice of loss function and hyperparameters have an impact on the accuracy which can be achieved, though the extent of which has not been fully researched by them.

2.2 MAZE

MAZE [15] is a data free model stealing attack algorithm published in 2020, which uses zeroth order gradient estimation to perform model stealing. In contrast to DFME, there are two loss functions: one for the generator and one for the student model. Kariyappa et al. (2020) [15] also utilise an 'experience replay', where the student is retrained on previously seen examples to avoid catastrophic forgetting. This has been found to improve the accuracy of the student model by an average of 7.3%. The generator has 3 convolutional layers.

2.3 Model Stealing Against Inductive Graph Neural Networks

Shen et al. (2021) [27] have created a model stealing method that works against graph neural networks [27], as opposed to image based networks. To accomplish this, 2 unique attacks were constructed to be used in different scenarios. Both these attacks are robust for graph neural networks, and have somewhat comparable results to image based neural networks.

2.4 Data free knowledge distillation

Data free model distillation is a technique used to 'distill' all the information known in a larger teacher model, to a smaller student model [24]. The teacher can be trained on a large dataset or an ensemble of separately trained models [1], which altogether give individual strong results. A variety of model compression techniques can be utilised to better distill and fine tune the model, which is then taught to the student [24], [21]. Several knowledge distillation techniques have been created, including data free model distillation [21] and even adversarial based distillation [6]. These form a similar basis to model extraction, and suggest its feasibility [29].

2.5 Generative adversarial models

A Generative Adversarial Network (GAN) is a generative network which learns from a target generator being pitted against a discriminator [9]. The generator generates its own fake samples and the discriminator receives a given sample and determines if the provided sample is a real sample from a target dataset or if it was fabricated by the generator. The 'loser' of this exchange updates their model and improves itself accordingly. Through repeated iterations, the generator improves its image generation and the discriminator improves its assessment of the dataset [9], though this can be somewhat inconsistent [22]. We make use of two competing networks in a similar fashion in order to allow the student to learn as quickly as possible.

2.6 Effect of hyperparameters on optimizers

Hyperparameters of an optimizer for a neural network are very important when training neural networks. Optimizers attempt to minimize the loss in a model [2] and as such have an influence on the learning rate of the model. Properly tuned hyperparameters can improve accuracy, robustness and reduce computing costs [8], [5]. Individual hyperparameter changes can have observable effects on a model, but when used together the result does not directly correspond to the sum of the individual changes [5].

2.7 Hyperparameter optimization algorithms

There are many algorithms with the express purpose of picking the best combinations of hyperparameters to ensure the best performance of the target neural network [2]. Each algorithm has its own strengths, weaknesses and specific use cases, but we primarily make use of grid search [19] and random search [3]. This is because they are relatively simple algorithms to implement and their impact is very easy to view and replicate (though random search is random so it is not always completely reproducible).

2.8 Novelty of research

This research is deemed novel as there is no prior research observing methods to directly reduce the number of queries required to perform model stealing. To the extent of our knowledge, there is no research directly observing the effect of hyperparameters on model stealing.

3 Problem Description

This section briefly outlines the problem which is faced and what can possibly be done about it.

While performing model stealing with high accuracy is possible through multiple different methods, one thing remains consistent between them all: They require an extremely high number of queries to get any kind of results [29], [15]. The exact number depends on the complexity of the dataset [29], [15], but even copying simple datasets such as MNIST [7] require millions of queries. To gain a substitute model of the SVHN dataset [25], the DFME algorithm required 2M queries to gain up to 99% model accuracy [29], and this is not an outlier - model extraction is very difficult to perform [13]. When this algorithm is applied to other, more complex datasets, the number of queries required to achieve high accuracy can increase dramatically.

In order to reduce the number of queries required to create an accurate model, several approaches were considered:

1. The first considered approach was to improve the optimiser. Different optimisers have different strengths and weaknesses [10], and should be utilised accordingly. The optimisers used in the algorithm initially are Adam and SGD.
2. The second considered approach was to improve the outputs of the generator, to make it closer to the actual type of data which the victim model will be creating. This could be done by using concepts from GAN's [9], or by changing input noise vector in some way. When observing the resulting images which were generated, even after 87% similarity to a CIFAR-10 [18] victim model the images showed absolutely no similarity to the actual dataset [29].
3. The third approach relates to the initial hyperparameters of the algorithm. Of all the citations for model extraction, only DFMEconsidered the effects that hyperparameter tuning could have, and from their preliminary testing, it appeared that extraction accuracy could be sentitive to the choice made [29]
4. A more advanced approach would be to alter the found algorithms in some way to improve the learning rate and reduce the number of queries required. This could be done by utilising various techniques found in multiple model stealing algorithms, or separately from it.

The third approach was chosen, after weighing the chances of reducing the number of queries required and the time constraints. Utilising different optimisers were unlikely to realistically reduce the number of queries required as some of

the best optimisers were already being utilised in appropriate conditions [10]. Regardless of this, the effects of using different optimisers can be briefly studied and it is possible to test if there is a more optimal use for the SGD [26] and Adam [17] optimisers.

Improving the outputs of the generator was deemed unlikely to have a large impact, as the generator maximising loss already causes it to search for the optimal image to train the student model on. Utilising another GAN [9] to train images which are accurate to the target model is also not feasible as it requires samples first to be able to train, which we do not have access to.

As for why the final option was not chosen, while it is likely that an alternate, optimal solution exists which can reduce the number of queries which are required, it is unrealistic to find, implement and test this in the time which is given.

This leaves hyperparameters. Theoretically, initial hyperparameters can influence the number of queries which are required to make a copy of a system like this. Initial testing showed that there was a large discrepancy in the rates that the student learned from the victim model depending on the hyperparameters chosen.

4 Methodology

This section outlines the steps which are taken to reduce the number of queries required while performing model stealing. This was primarily done through parameter optimisation, and the process for this is detailed.

To achieve the goal of reducing the number of queries required to perform model stealing, data collection was done systematically and applied accordingly.

Initially, investigation was done to view the relation between the rate in which the accuracy increases and the maximum accuracy which we could achieve. The way in which different hyperparameters affect the final accuracy, other hyperparameters, and how the optimal changes between datasets was also investigated. This information is compared to other results which are found by our experiments, and if applicable, results found in other instances of research (such as [15], [29]).

Initial tests were done by simply observing the effect of tuning hyperparameters manually to gather information on the learning curves such as the rate in which the accuracy of the student model increases in relation to the victim model. Initial values for the hyperparameters are based off of those chosen in [29], and iterated based on the results gathered. This is done to gather a better idea of the range of values which could feasibly give a positive result for the dataset.

Once the initial information is gathered, the information found is applied to various optimisation algorithms, namely grid search and random search algorithms. The number of queries which are used during these processes is recorded and factored in for the final result. After the optimization algorithms are completed, a comparison between the arbitrarily chosen hyperparameters and the optimized set is done. This is first performed on the MNIST dataset [7], and secondly performed on the fashion-MNIST dataset [31].

4.1 Model extraction algorithm

Algorithm 1 The Data free model extraction algorithm used

Require: Epoch range E, batch size for generator n_G, epoch iterations e_itr, generator iterations g_itr, student iterations s_itr student learning rate lr_s, generator learning rate lr_g, step timings $step$, scale $scale$, noise vector dimension z_dim

Ensure:
(Optional) target student accuracy T_acc
$optim_S \leftarrow$ optimizer(lr_s)
$Scheduler_S \leftarrow$ scheduler($optim_S$, $step$, $scale$)
$optim_G \leftarrow$ optimizer(lr_g)
$Scheduler_G \leftarrow$ scheduler($optim_G$, $step$, $scale$)
$e \leftarrow 0$
for e = 1 ... E **do**
  for $itr = 1...e\_itr$ **do**
    for $e_G = 1...g\_itr$ **do**
      $z \leftarrow randN(n_G, z\_dim)$
      $x \leftarrow Generate(z)$
      Query V using x
      approximate gradient $\Delta\theta_G L(x)$
      Apply to Opt_G
    end for
    for $e_S = 1...s\_itr$ **do**
      $z \leftarrow randN(n_G, z\_dim)$
      $x \leftarrow Generate(z)$
      Query V using x
      Calculate $V(x), L(x), \Delta\theta_S L(x)$
      Apply back propagation to gen using $L(x)$
      Apply to Opt_S
    end for
  end for
  Apply to $scheduler_S$
  Apply to $scheduler_G$
  if $student\_accuracy \geq T\_acc$ **then Break**
  end if
end for
Return e, $student\_accuracy$

The algorithm used can be seen in Algorithm 1. It is a slightly modified version of the DFME [29] algorithm, changed to operate with a number of epochs rather than continuing until a certain query budget is completed.

Initially the algorithm takes the arguments:

- epoch range E - Number of epochs the algorithm will run for. Influences the number of queries made by having another iteration in which the student and generator query the victim.
- batch size n_G - Amount of images generated by the generator in each 'cycle'. Influences the number of queries made as each image in the batch needs to be compared.
- epoch iterations e_itr - Number of times the process is repeated per epoch, before the schedulers are applied.

Influences the number of queries made as the entire process which is performed before the epoch 'completes' is repeated accordingly.

- student/generator iterations s/g_itr - The number of times the student or generator are individually repeated per epoch iteration These influence the number of queries made as this causes the student and generator to repeat their entire process according to the specifications.
- student learning rate lr_s - Learning rate for the student optimizer
- generator learning rate lr_g - Learning rate of the generator optimizer
- step timings $step$ - Percent of the way through the entire iteration process that the scale is utilised
- scale $scale$ - The multiplier that is applied to the larning rate of both the student and generators on each step timing
- noise vector dimension z_dim - the dimension of the noise vector

The student is our 'final' substitute network.

Schedulers for the student and generators are initialized using the learning rate, optimizers, steps and scale. The algorithm repeats for each epoch which is entered. Each epoch is repeated a number of times depending on the set epoch iterations. For this experiment, the epoch iterations is always set to 50.

First, the generator is worked on for the amount of times that is specified in g_itr. In this, a noise vector is generated and the victim is queried using the resulting image. The gradient is approximated and that is fed back to the generator network.

Secondly, the same is applied to the student but instead of approximating the gradient of the victim model, the loss L(x) is used to back propogate the generator, and the optimizer step is applied.

If the student accuracy is calculated to be higher than or equal to the target accuracy, then the epoch loop breaks and the number of queries is returned, along with the student model. Otherwise, it continues until all the epochs are complete.

4.2 Grid search

The first method of hyperparameter optimization is grid search. Grid search is an extremely simple algorithm, but is also very computationally intensive as it suffers from the curse of dimensionality [4]. This is because it is an exhaustive search algorithm, which covers all combinations of the sets of input parameters. This allows us to find the optimal combination of hyperparameters from the set of parameters which are added.

A very obvious issue with grid search is the sheer amount of computation which is required, which also lends itself to requiring a very large amount of queries as the number of parameters added are increased. Adding additional parameters very quickly increases the number of iterations which need to be done, and as such the parameters should be chosen carefully. The set of chosen hyperparameters can be seen in Table 1

lr_s	lr_g	step size	scale
0.001	0.0001	[0.08, 0.4, 0.9]	0.3
0.0035	0.0002	[0.1, 0.5, 0.9]	
0.006	0.0003	[0.12, 0.6, 0.9]	
0.0085	0.0004		

Table 1: Hyperparameters searched through in grid search for MNIST

4.3 Random search

The second algorithm that is used to optimise the hyperparamers is random search. Random search is also a simple algorithm, but does not cover an exhaustive range like grid search. Instead it only covers a range of values which the specified hyperparameters can cover. The algorithm then randomly selects values within these ranges for each hyperparameter that is done, and repeats until the specified number of iterations has been reached. If the current iteration reaches a target accuracy in fewer iterations than a saved minimum, it is saved as the current best and the student accuracy, number of iterations and current hyperparameters are saved. At the end, the hyperparameters which reached the target accuracy in the lowest number of iterations is returned. This can be seen in Appendix A. Every iteration is recorded so information of the suboptimal hyperparameters are also saved.

4.4 Fashion-MNIST

To observe the effects of hyperparameter tuning on a more difficult dataset, random search and grid search will also be applied onto the Fashion-MNIST dataset. Fashion-MNIST was chosen because it has a similar structure to MNIST, and as such requires minimal tuning to the algorithm. The only thing that is required is to train a neural network on the Fashion-MNIST dataset. As this dataset is much more difficult than MNIST, it is expected that the accuracy which can be achieved by the model stealing algorithm is much lower, and the differences in effectiveness of hyperparameter tuning should be more visible as a result.

5 Experimental results - MNIST

In this section, we will outline the results which were gathered through the experimental data which was gathered and discuss their implications.The victim model used has an accuracy of 99.55%.

5.1 MNIST - Individual hyperparameter changes

Here, the effects of tuning individual hyperparameters are observed and their implications are discussed.

The specific hyperparameters which have been tuned in this investigation are the learning rate for the student, the learning rate for the generator, the batch size (which affects the number of queries which are done in a single epoch), step timing and the scalar which is applied to the learning

rates when the steps are performed. In Figure 1 we can see the effect of changing just the student learning rate. The default hyperparameters can be seen in Appendix C. A full set of 50 epochs with a batch size of 25 results in 125000 queries being made. In these 12500 queries, we can see that the accuracy levels out at around 97% for all three values of lr_s which are around the order of magnitude 0.01. We can observe the same thing in graph (Graph still needs to be implemented)

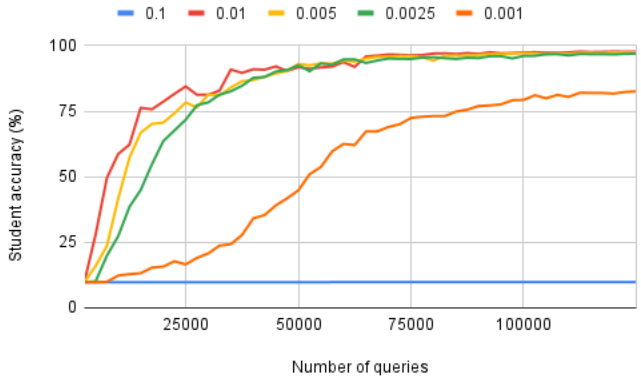

Figure 1: learning rate when lr_s is changed

5.2 MNIST Grid search results

Here, we can see the results of running Grid search on the target model. The full table of results is shown in Appendix D, and the set of hyperparameters which were searched through can be seen in Table 1

The condensed table of results can be found in Table 2. The first row depicts the optimal hyperparameters given in the first iteration, and the same follows for the second and third rows. The fourth row shows the combination of hyperparameters which gives the average lowest number of epochs. When running the same set of hyperparameters, the number of epochs which is required to reach 95% accuracy changes slightly each time.

The cause for this discrepancy is due to the randomness of the input vector into the generator, and is explored in Section 7. The effect of this randomness can be quite pronounced, as we can observe in the first row. The range between the lowest number of epochs and the highest is very high, from 7 to 21. This triples the number of queries required, and as such simply using these optimal hyperparameters from a single use of grid search can lead to suboptimal results. This holds for all three different 'optimal' parameters. Instead, row 4 could be considered the 'optimal' as it has the averaged best result.

5.3 MNIST Random search results

As random search gives random sets of hyperparameters, the results cannot be averaged out like what is seen in grid search. Instead, the full graph can be seen in Appendix F, and the graph of optimal results can be seen in Table 3. As we can see from the optimal results, the number of epochs which are required to achieve 95% accuracy hovers around the same area,

			Num epochs		
Lr_s :	LR_g	Step timings	1	2	3
0.0035	0.0003	[0.12, 0.6, 0.9]	7	21	11
0.0035	0.0004	[0.1, 0.5, 0.9]	16	8	14
0.0085	0.0003	[0.1, 0.5, 0.9]	10	19	6
0.0085	0.0004	[0.08, 0.4, 0.9]	10	8	13

Table 2: Table depicting the number of epochs it took to reach 95%. The green highlights depict the optimal hyperparameters for each iteration

				Ex	
Lr_s	LR_g	Step timings	Scale	1	2
0.00833	0.0007	[0.082, 0.648, 0.832]	0.388	6	-
0.00367	0.0007	[0.204, 0.369, 0.906]	0.352	-	8

Table 3: Optimal hyperparameters from random search and the number of epochs required to reach a student accuracy of 95%

from 6 to 8 epochs. It is likely that if we ran these same hyperparameters again that they would give different results, similarly to what is seen in the grid search results. What we do find however, is that even with a random selection of hyperparameters, the amount of epochs required to achieve this level of accuracy does not change very much. Instead, we find that a good selection of hyperparameters is important to reduce the number of epochs required, but only to a certain extent.

Past finding hyperparameters which are in the correct order of magnitude, further optimizing them are not incredibly effective. We do not see a significant difference in the number of queries required to achieve a desired target accuracy. Instead, the results we see are often influenced by the variation observed in Table 2, and further optimization which is performed in the grid search algorithm is not very useful for a relatively simple dataset such as MNIST.

5.4 MNIST Post optimization

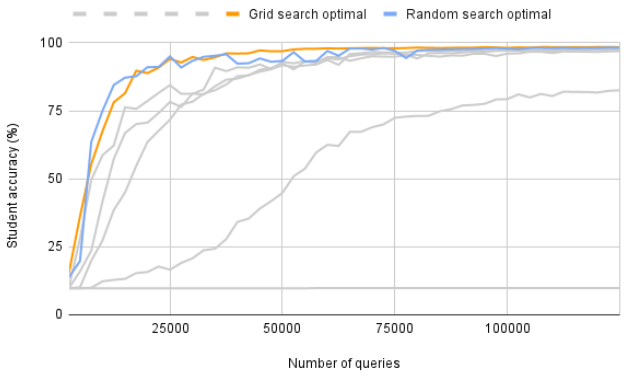

Figure 2: Difference between optimised results and previous results

After the results of grid and random search were returned, the effects of both were recorded and displayed in Figure 2. Compared to the previous results, both the results from grid

	Lr_S	Lr_G	Step timings	scale	batch	epochs
Ex 1	0.001	0.0001	[0.1, 0.5, 0.9]	0.3	25	50
Ex 2	0.01	0.001	[0.1, 0.5, 0.9]	0.3	25	50
Ex 3	0.01	0.001	[0.3, 0.5, 0.9]	0.3	25	50

Table 4: Initial hyperparameters used to observe accuracy with queries for Fashion-MNIST

	Lr_S	Lr_G	Step timings	scale	batch	epochs
Ex1	0.01	0.0001	[0.1, 0.5, 0.9]	0.3	50	50
Ex2	0.01	0.001	[0.1, 0.5, 0.9]	0.3	50	100
Ex3	0.001	0.0006	[0.17, 0.5, 0.9]	0.225	50	100

Table 5: Hyperparameters used when increasing the number of epochs in Fashion-MNIST

and random search performed significantly better. The accuracy for grid search increases at a much more steady rate than the random search result. This is likely due to the fact that this is the set of hyperparameters which has the lowest average number of required epochs over multiple repetitions, while random search gave results with only a single trial making more likely to give a result with more 'volatility'.

6 Experimental results - Fashion-MNIST

Here, we can see how hyperparameter tuning affects the results of a more difficult dataset: Fashion-MNIST [31]. It is expected that the number of queries required to perform model stealing to a reasonable accuracy will be much higher, and that any changes will leave much larger impacts on the number of queries required than were found for MNIST. The victim model has an accuracy of 91.80%.

6.1 Fashion-MNIST Individual hyperparameter changes

Here, the effects of tuning individual hyperparameters are observed and their implications are discussed for the dataset of fashion-MNIST.

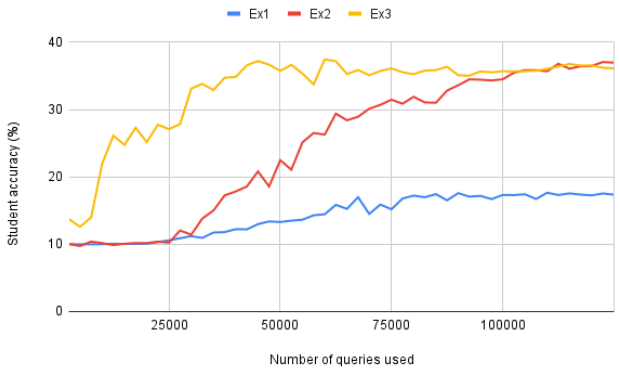

Figure 3: Initial experiments run for the learning rate of Fashion-MNIST. Ex1, 2 and 3 correspond to the parameters shown in Table 4

The first set of hyperparameters were chosen based off of the same parameters that MNIST was initialised to. Initially, observations were made with regards to how the student model learned with the same number of queries as for the MNIST dataset. The student learning rate and step timings were slowly raised as more experiments were performed, to observe exactly how the learning rate changed. The hyperparameters used can be seen in Table 4, and the graph of the student accuracy can be seen in Figure 3. As expected, the new dataset is much more difficult to copy than MNIST, only reaching a maximum of 37% accuracy in 125000 queries. More queries are required to view if the maximum accuracy continuously increases as more queries are utilised, or if the maximum accuracy levels out.

As such, a larger batch size was utilised along with more epochs. This increases the number of queries per epoch, and increases the duration of time in which the model is trained for. The exact hyperparameters chosen can be seen in Table 5, and the graph of the results can be seen in Figure 5. Increasing the batch size does have an impact on the maximum accuracy the model reaches. As can be observed, the accuracy peaks at 45%, as opposed to the old peak of 37%. We cannot draw direct conclusions of the effects of the batch size or number of epochs here, as the steps and scale also impact this. It is possible that instead, the step timing is suboptimal and the experiments run in Figure 3 would be more optimal without scaling at all.

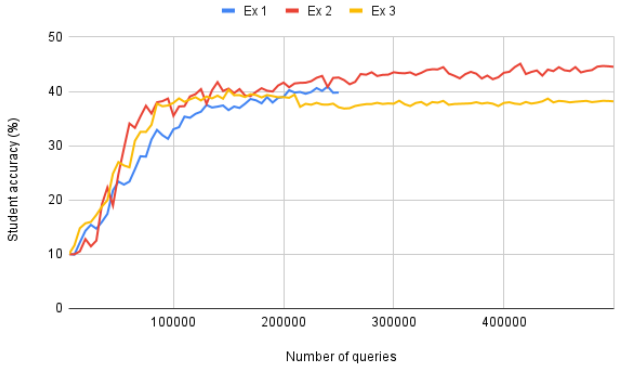

Figure 4: Experiments run while increasing number of queries available. Exact hyperparameters are visible in Table 5

Regardless, we can observe that using more queries does have an impact on the accuracy of the student model. This reflects the observations found in [29] and [15]. To continue this, the number of queries which were used were increased to 3 million, and the impact it had on learning rate was also recorded. These results can be seen in Figure 5. Both these sets of hyperparameters are identical to those found in Figure 4, except for the batch size. We observe that the choice in initial hyperparameters has a strong impact on the number of queries which is required to achieve middling levels of accuracy. While it is possible to reach 70% accuracy with significantly fewer queries through better hyperparameter choice, it is not known if this continues to higher levels, such as to (95% +) victim accuracy. It is possible that certain hyperparameter choices may never reach that accuracy at all.

	Lr_S	Lr_G	Step timings	scale	batch	epochs
Ex 1	0.01	0.001	[0.1, 0.5, 0.9]	0.3	300	100
Ex 2	0.01	0.001	[0.3, 0.5, 0.9]	0.3	300	100

Table 6: hyperparameters used to test accuracy with 3 million queries

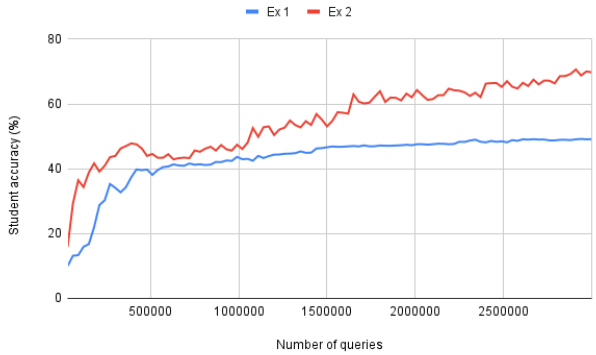

Figure 5: Experiments run to 3 million queries, hyperparameters shown in Table 6

6.2 Fashion-MNIST Grid search results

The results for Grid search can be seen in Appendix G, and the grid of hyperparameters chosen can be found in Table 15 in Appendix I. The reason that the batch size was chosen to be 50 and for the maximum number of epochs was set to 50 was was to allow for somewhat reasonable computation times. This is also the reason that the number of varying hyperparameters has been decreased. The target accuracy for these iterations were to 60%, but was never reached.

The results can be seen in Table 7. Both experiments yielded the same optimal set of hyperparameters, but their accuracy was below 40% each time. This is in the same range as Figure 3, both in accuracy and number of queries used. As the datasets become more difficult, the range of effective hyperparameters becomes more difficult to properly evaluate, and as such grid search becomes less and less useful. This is what is expected from grid search, but it is already clear that is is relatively ineffective for finding a good range of viable hyperparameters for more difficult datasets.

					accuracy	
Lr_s :	LR_g	Step timings	scale	epochs	Ex 1	Ex 2
0.006	0.0004	[0.2, 0.6, 0.933]	0.3	50	38.06	39.41

Table 7: Results for optimal hyperparameter results for fashion-MNIST grid search. There is only one column as the optimal was the same both times

6.3 Fashion-MNIST Random search results

The full set of results can be found in Appendix H. Of these results, the optimal hyperparameter combinations are shown in Table 3. Both these results give noticeably higher accuracies than the grid search results in Table 7. As both of these experiments use 125000 queries each, they prove to be significantly better than the results found in Figure 4, giving higher accuracies while using one quarter the number of queries. We can already observe that to an extent, hyperparameter tuning is effective at reducing the number of queries required to achieve certain accuracies while performing model stealing. As the algorithm used is currently unable to match the results shown in DFME and MAZE, we are unable to verify if this same effect can also be observed when attempting to perform model stealing to a significantly higher target student accuracy.

					ex	
Lr_s	Lr_g	step timing	scale	epochs	1	2
0.00857	0.000734	[0.21, 0.6, 0.929]	0.438	50	49.49	-
0.00666	0.000897	[0.293, 0.329, 0.895]	0.425	50	-	46.64

Table 8: Results for the optimal set of hyperparameters for random search in Fashion-MNIST

6.4 Fashion-MNIST after optimization

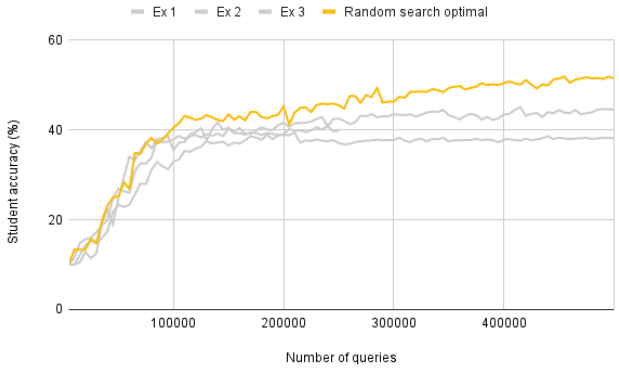

Figure 6: Graph displaying the optimized results found from random search

Post optimization, we can see from Figure 6 that the learning rate of the new optimal set of hyperparameters found by random search outperforms all previously arbitrarily chosen sets of hyperparameters. The student appears to learn at an improved rate overall, and this combined with the results from MNIST shows that hyperparameter tuning can be a viable method to reduce the number of queries required to perform model stealing.

If scaled up to 3 million queries however, the effectiveness of the results are less visible. This can be seen in Figure 7. Scaling up batch sizes does not always result in a direct increase in learning, as also explained in [5]

6.5 Optimizers

Neural network optimizers are extremely important when minimizing the loss of a neural network [2]. This holds true for our student and generator models, where each utilises a chosen optimizer to ensure that the student manages to minimize loss, while the generator seeks to maximise it as shown in algorithm 1. As the choice in optimizer has a direct impact on the results of the models, investigation should also be performed on what combination of optimizers are optimal, and for what situations.

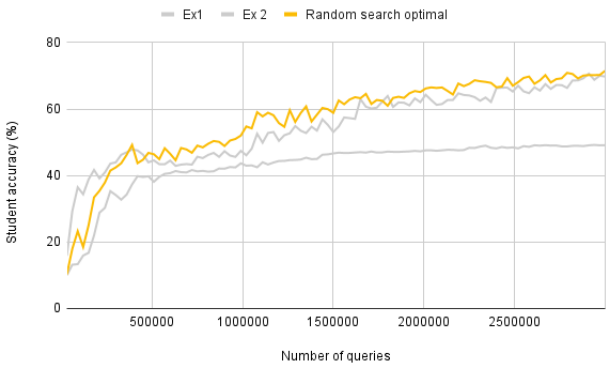

Figure 7: Difference between optimised results and previous results

Only relatively few changes have been made for optimizers. Our default has the student using SGD and the generator uses Adam. To verify whether this was the optimal usage, the student and generators have been tested using all combinations of SGD and Adam. Initial tests show that for fashion-MNIST, student - SGD and generator - Adam is the clear optimum. For MNIST brief experiments in both grid and random search show that setting both to SGD could improve results, as experiments have returned optimal results of 4 epochs required to achieve 95% accuracy, as opposed to minima of 6 for the standard setup. Full results can be seen in Appendices J, K. The reasons for this are not very clear at the moment, and should be explored more in the future.

7 Random number generator discussion

This section will discuss the effects of randomness found in each experiment run.

When looking at the results found in Table 2, we run into an issue regarding what is 'optimal'. For the three experiments, the 'optimal' set of hyperparameters is different each time, and the number of epochs (and as such, queries) which are required to achieve the target accuracy changes significantly each time. None of the resulting optimal results give the lowest average number of queries.

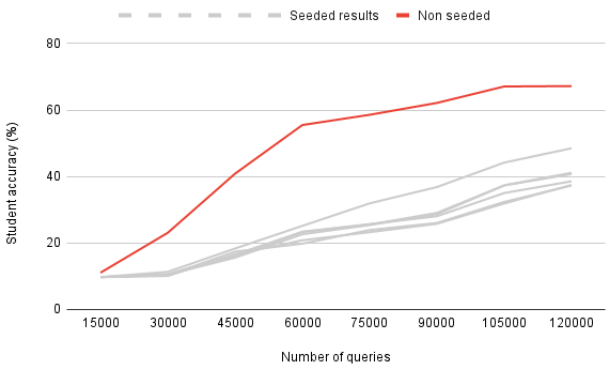

Figure 8: Brief experiments with random seeding. 6 seeded vs 1 non seeded

To study the effect of the randomness of the input vector, a seeded experiment was conducted. The noise vector was set to random seed (3). The results can be seen in Figure 8, compared to a non seeded result performed immediately afterward. This experiment was conducted with a single set of arbitrarily chosen hyperparameters with a batch size of 15 to 8 epochs on the MNIST dataset.

The 6 sets of seeded results are not exactly the same but are very similar, meanwhile the non seeded iteration performs quite differently. This provides evidence that even with the same set of hyperparameters, the randomness of the input vector does have quite a significant impact on the performance of individual experiments. This means we must repeat all optimization algorithms multiple times to draw more accurate conclusions.

8 Conclusion

Overall, hyperparameter tuning is a viable method to reduce the number of queries required to perform model stealing, provided you are able to first perform the hyperparameter optimization beforehand. There are noticeable effects when performed on simple datasets such as MNIST, but this has not been fully tested to very high target accuracies for more difficult datasets such as fashion-MNIST. The highest target accuracy which has been achieved in this study for fashion-MNIST is 0.79x the target model's accuracy of 91.80%. Still, proper tuning does seem quite effective at reducing the number of queries required to achieve high accuracies.

The grid and random search algorithms were used to optimize hyperparameters. While generally random search gave better results, the randomness of the input vector to the generator causes some issues when determining the optimal set of hyperparameters. More robust hyperparameter opimization algorithms could be utilised to see if they can further improve results.

The neural network optimizers were briefly tested to see if using SGD for the student and Adam for the generator was the optimal combination for these optimizers. For fashion-MNIST this proved true, while for MNIST there seemed to be some positive effect when using SGD for the generator as well. This should be further tested however.

8.1 Future work

This section outlines possible improvements and additional research that can still be done

1 - While the simple hyperparameter optimization algorithms yielded decent results, more advanced algorithms such as Bayesian hyperparameter optimization could prove more effective. Research can be done to view the effects they can have.

2 - Attempt to recreate the DFME experiment and test if optimizing the hyperparameters there allows for fewer queries to be used with their stealing of a CIFAR-10 model compared to the parameters they used

3 - Research if different optimizers than simply SGD and Adam give better results for the student and generators. This should also research why SGD for both the student and generator appears to improve results for MNIST.

9 Responsible Research

The research found here is purely done for hypothetical purposes, and should not be performed realistically. Intellectual property theft is illegal [14], and should not be utilised on real neural networks. Research behind model stealing is important in order to gather a greater understanding of it, both how it is performed and its limitations. There are already papers researching model stealing and methods to defend against it [20], [16]. Understanding the extent that hyperparameters can influence the number of queries which are required to perform model extraction has little effect on real world extraction attacks.

Ensuring that the work is reproducible is also very important. To this extent, the repository is made public [1], and all algorithms and values used are available in the appendices. The repository contains easy to change specifications to run the code, and includes the original results from each experiment.

References

[1] Umar Asif, Jianbin Tang, and Stefan Harrer. Ensemble knowledge distillation for learning improved and efficient networks, 2019.

[2] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012.

[4] Aloïs Bissuel. Hyper-parameter optimization algorithms: a short review, 2019.

[5] Thomas M. Breuel. The effects of hyperparameters on sgd training of neural networks, 2015.

[6] Yoojin Choi, Jihwan Choi, Mostafa El-Khamy, and Jungwon Lee. Data-free network quantization with adversarial knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

[7] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[8] Xianping Du, Hongyi Xu, and Feng Zhu. Understanding the effect of hyperparameter optimization on machine learning models for structure design problems. *Computer-Aided Design*, 135:103013, jun 2021.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[1] https://github.com/bwmfvanveen-II/BRP-AdversarialAttacks.git

[10] Ayush Gupta. A comprehensive guide on deep learning optimizers, 2021.

[11] Christian Szegedy Ian J. Goodfellow, Jonathon Shlens. Explaining and harnessing adversarial examples. 2015.

[12] 'ilmoi'. Evasion attacks on machine learning (or "adversarial examples"), 2019.

[13] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks, 2019.

[14] Jose Rivera Jennifer Corbett. Intellectual property theft, 2022.

[15] Sanjay Kariyappa, Atul Prakash, and Moinuddin Qureshi. Maze: Data-free model stealing attack using zeroth-order gradient estimation, 2020.

[16] Sanjay Kariyappa and Moinuddin K Qureshi. Defending against model stealing attacks with adaptive misinformation, 2019.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[18] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[19] Steven M LaValle, Michael S Branicky, and Stephen R Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *The International Journal of Robotics Research*, 23(7-8):673–692, 2004.

[20] Taesung Lee, Benjamin Edwards, Ian Molloy, and Dong Su. Defending against machine learning model stealing attacks using deceptive perturbations, 2018.

[21] Raphael Gontijo Lopes, Stefano Fenu, and Thad Starner. Data-free knowledge distillation for deep neural networks, 2017.

[22] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[23] Nicolas Papernot Matthew Jagielski. In model extraction, don't just ask 'how?': Ask 'why?', 2020.

[24] Fatemehsadat Mireshghallah, Arturs Backurs, Huseyin A Inan, Lukas Wutschitz, and Janardhan Kulkarni. Differentially private model compression, 2022.

[25] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[26] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[27] Yun Shen, Xinlei He, Yufei Han, and Yang Zhang. Model stealing attacks against inductive graph neural networks, 2021.

[28] Mark Stone. Why adversarial examples are such a dangerous threat to deep learning, 2020.

[29] Jean-Baptiste Truong, Pratyush Maini, Robert J. Walls, and Nicolas Papernot. Data-free model extraction, 2020.

[30] Kyle Wiggers. Adversarial attacks in machine learning: What they are and how to stop them, 2021.

[31] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Appendix start

A Grid search algorithm

Algorithm 2 Grid search algorithm

Require: Set of student learning rates SLr_s,
Set of generator learning rates SLr_g
Set of step timings $Sstep$,
Set of scales to apply in the step timings $Sscale$

Ensure:
$bestNumEpochs \leftarrow Max\_Value$
$bestAccuracy \leftarrow 0$
$bestArgsSetup$
for S_n = SLr_s1 to SLr_sn **do**
  for G_n = SLr_g1 to SLr_gn **do**
    for $step_n$ = $Sstep1$ to $Sstepn$ **do**
      for $scale_n$ = $Sscale1$ to $Sscalen$ **do**

        $numEpochs, studentAccuracy = run$ Algorthm 1 with params(S_n, G_n, $step_n$, $scale_n$)

        if $numEpochs < bestNumEpochs$ **then**
          $bestNumEpochs = numEpochs$
          $bestAccuracy = studentAccuracy$
          $bestArgsSetup$ = (S_n, G_n, $step_n$, $scale_n$)
        end if
      end for
    end for
  end for
end for
Return bestArgsSetup, bestNumEpochs, and bestAccuracy

B Random search algorithm

Algorithm 3 Random search algorithm

Require: Range of student learning rates RLr_s,
Range of generator learning rates RLr_g
Range of step timings $Rstep$,
Range of scales to apply in the step timings $Rscale$,
number of iterations desired $iters$

Ensure:
$bestNumEpochs \leftarrow Max\_Value$
$bestAccuracy \leftarrow 0$
$bestArgsSetup$
for n = 1 to $iters$ **do**
  S_n = Random within RLr_s
  G_n = Random within RLr_g
  $step_n$ = Random within $Rstep$
  $scale_n$ = Random within $Rscale$
  $numEpochs, studentAccuracy = run$ Algorthm 1 with params(S_n, G_n, $step_n$, $scale_n$)

  if $numEpochs < bestNumEpochs$ **then**
    $bestNumEpochs = numEpochs$
    $bestAccuracy = studentAccuracy$
    $bestArgsSetup$ = (S_n, G_n, $step_n$, $scale_n$)
  end if
end for
Return bestArgsSetup, bestNumEpochs, and bestAccuracy

C Default hyperparameters for individual hyperparameter changes

hyperparameter	value
lr_S	0.001
lr_G	0.0001
step timings	[0.1, 0.5, 0.9]
scale	0.3
number of epochs	50
batch size	25
Epoch iters	50

Table 9: Default hyperparameters for viewing the learning rate of MNIST

D Grid search average results

$\mathbf{Lr}_s$	$\mathbf{LR}_g$	Step timings	epochs	S_acc
0.001	0.0001	[0.08, 0.4, 0.9]	50.00	80.68
		[0.1, 0.5, 0.9]	50.00	77.94
		[0.12, 0.6, 0.9]	50.00	89.30
	0.0002	[0.08, 0.4, 0.9]	50.00	91.84
		[0.1, 0.5, 0.9]	50.00	94.51
		[0.12, 0.6, 0.9]	39.33	94.33
	0.0003	[0.08, 0.4, 0.9]	50.00	91.98
		[0.1, 0.5, 0.9]	24.00	95.39
		[0.12, 0.6, 0.9]	28.33	95.54
	0.0004	[0.08, 0.4, 0.9]	46.00	92.55
		[0.1, 0.5, 0.9]	36.00	95.33
		[0.12, 0.6, 0.9]	24.00	95.16
0.0035	0.0001	[0.08, 0.4, 0.9]	24.33	95.66
		[0.1, 0.5, 0.9]	34.00	94.80
		[0.12, 0.6, 0.9]	19.00	95.22
	0.0002	[0.08, 0.4, 0.9]	17.33	95.10
		[0.1, 0.5, 0.9]	15.33	95.47
		[0.12, 0.6, 0.9]	22.67	95.54
	0.0003	[0.08, 0.4, 0.9]	15.33	95.37
		[0.1, 0.5, 0.9]	12.33	95.73
		[0.12, 0.6, 0.9]	13.00	95.08
	0.0004	[0.08, 0.4, 0.9]	15.33	95.66
		[0.1, 0.5, 0.9]	12.67	95.68
		[0.12, 0.6, 0.9]	11.33	95.16
0.006	0.0001	[0.08, 0.4, 0.9]	38.67	95.26
		[0.1, 0.5, 0.9]	22.00	95.57
		[0.12, 0.6, 0.9]	38.00	94.96
	0.0002	[0.08, 0.4, 0.9]	18.33	95.81
		[0.1, 0.5, 0.9]	16.67	95.63
		[0.12, 0.6, 0.9]	13.33	95.37
	0.0003	[0.08, 0.4, 0.9]	16.33	95.42
		[0.1, 0.5, 0.9]	15.67	95.40
		[0.12, 0.6, 0.9]	14.67	95.41
	0.0004	[0.08, 0.4, 0.9]	13.00	95.56
		[0.1, 0.5, 0.9]	11.67	95.39
		[0.12, 0.6, 0.9]	13.67	95.59
0.0085	0.0001	[0.08, 0.4, 0.9]	28.00	95.40
		[0.1, 0.5, 0.9]	24.67	95.41
		[0.12, 0.6, 0.9]	23.00	95.45
	0.0002	[0.08, 0.4, 0.9]	21.33	95.75
		[0.1, 0.5, 0.9]	13.00	95.46
		[0.12, 0.6, 0.9]	22.33	95.45
	0.0003	[0.08, 0.4, 0.9]	17.33	95.51
		[0.1, 0.5, 0.9]	11.67	95.97
		[0.12, 0.6, 0.9]	12.67	95.36
	0.0004	[0.08, 0.4, 0.9]	10.33	95.16
		[0.1, 0.5, 0.9]	13.33	96.12
		[0.12, 0.6, 0.9]	16.00	95.32

Table 10: Average results of three tests of grid search with the following hyperparameters. The step size was a constant 0.3, and was omitted from the graph

E Range of hyperparameters for random search

		min	max
	Lr_s	0.0001	0.01
	Lr_g	0.0001	0.001
step timings	1	0.01	0.3
	2	0.31	0.7
	3	0.8	0.99
	Scale	0.15	0.45

F Raw data for random search

Lr_s:	LR_g	Step timings	scale	epochs	accuracy
0.00116	0.00016	[0.0922, 0.6714, 0.9458]	0.3522	32	95.55
0.00872	0.00067	[0.0217, 0.5352, 0.8727]	0.4434	16	95.60
0.00154	0.00052	[0.2559, 0.4127, 0.9866]	0.3327	10	95.05
0.00407	0.0002	[0.2273, 0.6839, 0.9548]	0.2574	15	95.26
0.007	2.87E-05	[0.0910, 0.3368, 0.81131]	0.3914	50	93.58
0.00318	0.00059	[0.1556, 0.3257, 0.9137]	0.2237	7	95.02
0.00903	0.00023	[0.1842, 0.3495, 0.9650]	0.2151	18	95.20
0.00637	0.00013	[0.1270, 0.5846, 0.9736]	0.177	20	95.11
0.00172	0.00019	[0.2863, 0.3523, 0.8663]	0.1919	14	96.52
0.00174	0.00046	[0.1651, 0.3505, 0.8230]	0.1648	9	95.07
0.00046	0.00093	[0.1288, 0.5446, 0.8819]	0.2846	32	95.43
0.0055	0.00092	[0.0135, 0.6860, 0.9671]	0.2037	18	95.34
0.00052	0.00085	[0.0842, 0.3160, 0.8920]	0.2442	50	86.77
0.00098	0.00079	[0.1990, 0.4929, 0.9845]	0.2414	23	95.42
0.0094	0.0005	[0.1300, 0.5769, 0.9235]	0.1597	11	95.95
0.00263	0.00064	[0.0431, 0.5947, 0.9888]	0.322	14	95.79
0.00546	0.00031	[0.2515, 0.5735, 0.9819]	0.4429	10	95.08
0.00939	0.00071	[0.2183, 0.6430, 0.9003]	0.2443	11	95.16
0.00104	0.00047	[0.1948, 0.3983, 0.8219]	0.2804	17	95.65
0.0006	0.00037	[0.2008, 0.4981, 0.8078]	0.3182	50	92.11
0.00575	0.00096	[0.2652, 0.3497, 0.8873]	0.1682	40	95.12
0.00346	0.00021	[0.2260, 0.3830, 0.9043]	0.2613	13	95.65
0.00832	0.00072	[0.0817, 0.6479, 0.8323]	0.3883	6	95.68
0.00256	0.00027	[0.1546, 0.3151, 0.8676]	0.1852	32	95.00
0.00967	7.43E-05	[0.2881, 0.6651, 0.9353]	0.319	41	95.06

Table 11: Full results for the first set of random search with the following parameters

G Fashion MNIST average grid search results

Lr_s :	LR_g	Step timings	epochs	S_acc
0.001	0.0001	[0.1, 0.5, 0.9]	50	11.26
		[0.2, 0.6, 0.933]	50	13.03
	0.0004	[0.1, 0.5, 0.9]	50	13.67
		[0.2, 0.6, 0.933]	50	20.02
0.006	0.0001	[0.1, 0.5, 0.9]	50	16.13
		[0.2, 0.6, 0.933]	50	27.85
	0.0004	[0.1, 0.5, 0.9]	50	26.51
		[0.2, 0.6, 0.933]	50	38.74
0.0085	0.0001	[0.1, 0.5, 0.9]	50	19.74
		[0.2, 0.6, 0.933]	50	20.82
	0.0004	[0.1, 0.5, 0.9]	50	25.26
		[0.2, 0.6, 0.933]	50	30.54

Table 12: Table of averaged grid search results performed for Fashion MNIST

H Fashion MNIST random search results

Lr_s :	LR_g	Step timings	scale	epochs	accuracy
0.00549	8.60E-05	[0.083, 0.315, 0.934]	0.256	50	21.2
0.0063	0.000871	[0.145, 0.485, 0.898]	0.39	50	42.67
0.00433	0.000489	[0.061, 0.328, 0.936]	0.431	50	41.98
0.00699	0.000462	[0.019, 0.615, 0.85]	0.298	50	35.95
0.00076	2.45E-04	[0.054, 0.622, 0.87]	0.221	50	13.14
0.00748	0.000896	[0.018, 0.689, 0.879]	0.186	50	32.17
0.00831	0.000778	[0.247, 0.644, 0.901]	0.411	50	41.2
0.00066	0.000475	[0.255, 0.553, 0.825]	0.263	50	29.98
0.00857	0.000734	[0.21, 0.6, 0.929]	0.438	50	49.49
0.00089	0.000594	[0.268, 0.419, 0.815]	0.312	50	29.29
0.00804	0.000711	[0.069, 0.443, 0.846]	0.314	50	33.83
0.00506	0.000701	[0.065, 0.504, 0.884]	0.172	50	24.27
0.00754	0.000935	[0.265, 0.338, 0.865]	0.363	50	39.7
0.00994	0.000659	[0.279, 0.394, 0.923]	0.312	50	43.44
0.0056	0.000826	[0.073, 0.419, 0.879]	0.342	50	42.27
0.00248	0.000682	[0.047, 0.468, 0.96]	0.392	50	32.57
0.00084	0.000489	[0.154, 0.439, 0.876]	0.437	50	30.16
0.00695	0.000474	[0.097, 0.555, 0.979]	0.285	50	38.9
0.00309	0.000776	[0.066, 0.514, 0.856]	0.202	50	35.93
0.00877	0.000978	[0.014, 0.607, 0.831]	0.2340	50	39.37
0.00751	0.000645	[0.275, 0.4, 0.965]	0.17	50	35.32
0.00531	0.00015	[0.102, 0.677, 0.873]	0.245	50	33.19
0.00872	0.000141	[0.016, 0.553, 0.91]	0.204	50	19.16
0.004	9.60E-05	[0.148, 0.422, 0.982]	0.421	50	29.32
0.00651	4.57E-04	[0.217, 0.479, 0.882]	0.178	50	34.58
0.00531	0.00015	[0.102, 0.677, 0.873]	0.245	50	33.19
0.00666	1.90E-05	[0.031, 0.518, 0.909]	0.328	50	19.71

Table 13: Table of the first set of results for Fashion MNIST random search

Lr_s :	LR_g	Step timings	scale	epochs	accuracy
0.00557	6.07E-04	[0.223, 0.681, 0.917]	0.328	50	33.71
0.00666	0.000897	[0.293, 0.329, 0.895]	0.425	50	46.64
0.00618	0.000784	[0.158, 0.358, 0.856]	0.396	50	43.53
0.00605	0.000889	[0.128, 0.504, 0.813]	0.423	50	42.92
0.00189	6.75E-04	[0.297, 0.32, 0.951]	0.412	50	34.18
0.00227	0.000858	[0.086, 0.484, 0.954]	0.24	50	37.8
0.00828	0.000174	[0.134, 0.408, 0.923]	0.252	50	31.95
0.00727	0.000744	[0.016, 0.518, 0.985]	0.418	50	40.16
0.00476	0.000802	[0.013, 0.373, 0.923]	0.24	50	31.69
0.00024	0.000226	[0.041, 0.5, 0.84]	0.271	50	9.55
0.00125	0.000349	[0.156, 0.674, 0.931]	0.159	50	27.16
0.00195	0.000978	[0.286, 0.534, 0.902]	0.385	50	37.9
0.00411	0.000724	[0.215, 0.501, 0.811]	0.243	50	35.17
0.00508	0.000759	[0.268, 0.502, 0.808]	0.162	50	39.47
0.00276	0.00047	[0.211, 0.517, 0.911]	0.173	50	29.57
0.00491	0.000261	[0.292, 0.616, 0.905]	0.256	50	20.94
0.00629	0.0002	[0.098, 0.36, 0.868]	0.427	50	38.65
0.00036	2.40E-05	[0.192, 0.511, 0.838]	0.449	50	13.2
0.00301	0.000493	[0.035, 0.376, 0.934]	0.206	50	24.02
0.00947	0.000376	[0.073, 0.428, 0.839]	0.1690	50	23.87
0.00952	0.000464	[0.177, 0.376, 0.941]	0.297	50	32.42
0.00852	0.00067	[0.25, 0.39, 0.913]	0.302	50	45.41
0.00751	0.000936	[0.114, 0.627, 0.867]	0.348	50	39.11
0.00666	1.90E-05	[0.031, 0.518, 0.909]	0.328	50	19.71
0.00488	8.50E-04	[0.133, 0.351, 0.903]	0.156	50	30.97

Table 14: Table of the second set of results for Fashion MNIST random search

I Fashion MNIST hyperparameters used

lr_s	lr_g	step timings	batch	epochs
0.001	0.0001	[0.1, 0.5, 0.9]	50	50
0.006	0.0004	[0.2, 0.6, 0.933]		
0.0085				

Table 15: Hyperparameters used in grid search for fashion MNIST

		min	max
	Lr_s	0.0001	0.01
	Lr_g	1e-05	0.001
step timings	1	0.01	0.3
	2	0.31	0.7
	3	0.8	0.99
	scale	0.15	0.45

Table 16: Hyparparameter ranges used in random search for Fashion-MNIST

J Optimizer experiment optimal results - Grid search

There are far too many experiments run to properly put them in the correct format at this time, and as such only the optimal results are displayed here. The full set of results can be found in the repository [2]

J.1 MNIST

Grid of student learning rates: [0.001, 0.0035, 0.006, 0.0085]
Grid of generator learning rates: [0.0001, 0.0002, 0.0003, 0.0004]
Grid of step timings: [[0.08, 0.4, 0.9], [0.1, 0.5, 0.9], [0.12, 0.6, 0.9]]
Grid of scales used: [0.3]

Generator - SGD, Student - SGD
Experiment 1
The optimal set of hyperparameters were:
Dataset: MNIST
Learning rate of S: 0.0085
Learning rate of G: 0.0003
z_dim: 128

[2] https://github.com/bwmfvanveen-II/BRP-AdversarialAttacks.git

Step timings: [0.08, 0.4, 0.9], and scale: 0.3
Final student accuracy: 87.42Total time taken to run: 1966.7234036922455 seconds
Total number of queries made: 830000
Number of epochs taken to achieve this result: 2

Experiment 2
The optimal set of hyperparameters were:
Dataset: MNIST
Learning rate of S: 0.006
Learning rate of G: 0.0003
z_dim: 128
Step timings: [0.08, 0.4, 0.9], and scale: 0.3
Final student accuracy: 95.41Total time taken to run: 3394.7809166908264 seconds
Total number of queries made: 1472500
Number of epochs taken to achieve this result: 4

Generator - Adam, Student - Adam
Experiment 1
Dataset: MNIST
Learning rate of S: 0.001
Learning rate of G: 0.0004
z_dim: 128
Step timings: [0.1, 0.5, 0.9], and scale: 0.3
Final student accuracy: 95.02Total time taken to run: 11066.722125768661 seconds
Total number of queries made: 3825000
Number of epochs taken to achieve this result: 9

Experiment 2
Dataset: MNIST
Learning rate of S: 0.001
Learning rate of G: 0.0004
z_dim: 128
Step timings: [0.1, 0.5, 0.9], and scale: 0.3
Final student accuracy: 95.64Total time taken to run: 8657.446615219116 seconds
Total number of queries made: 3600000
Number of epochs taken to achieve this result: 7

J.2 Fashion-MNIST

Grid of student learning rates: [0.001, 0.006, 0.0085]
Grid of generator learning rates: [0.0001, 0.0004]
Grid of step timings: [[0.1, 0.5, 0.9], [0.2, 0.6, 0.933]]
Grid of scales used: [0.3]

Generator - SGD, Student - SGD
The optimal set of hyperparameters were:
Dataset: FASHIONMNIST
Learning rate of S: 0.0085
Learning rate of G: 0.0004
z_dim: 128
Step timings: [0.1, 0.5, 0.9], and scale: 0.3
Final student accuracy: 36.06Total time taken to run: 3277.818016052246 seconds
Total number of queries made: 1500000
Number of epochs taken to achieve this result: 50

Generator - Adam, Student - Adam
The optimal set of hyperparameters were:
Dataset: FASHIONMNIST
Learning rate of S: 0.006
Learning rate of G: 0.0004
z_dim: 128
Step timings: [0.2, 0.6, 0.933], and scale: 0.3
Final student accuracy: 37.02Total time taken to run: 3740.83221411705 seconds
Total number of queries made: 1500000
Number of epochs taken to achieve this result: 50

K Optimizer experiment optimal results - Random search

K.1 MNIST

Range of student learning rates: [0.0001, 0.01]
Range of generator learning rates: [1e-05, 0.01]
Range of step timings: [[0.01, 0.31, 0.8], [0.3, 0.7, 0.99]]
Range of scales used: [0.05, 0.5]

Generator - SGD, Student - SGD
Experiment 1
Dataset: MNIST
Learning rate of S: 0.00492
Learning rate of G: 0.009116

z_dim: 128
Step timings: [0.035, 0.492, 0.953], and scale: 0.245
Final student accuracy: 96.25Total time taken to run: 4335.1783266067505 seconds
Number of epochs taken to achieve this result: 6

Experiment 2
Dataset: MNIST
Learning rate of S: 0.00713
Learning rate of G: 0.000475
z_dim: 128
Step timings: [0.281, 0.496, 0.803], and scale: 0.381
Final student accuracy: 95.55Total time taken to run: 4310.891749858856 seconds
Number of epochs taken to achieve this result: 4

Generator - Adam, Student - Adam

The optimal set of hyperparameters were:
Experiment 1
Dataset: MNIST
Learning rate of S: 0.00158
Learning rate of G: 0.002818
z_dim: 128
Step timings: [0.13, 0.663, 0.871], and scale: 0.221
Final student accuracy: 96.65Total time taken to run: 3371.5163402557373 seconds
Number of epochs taken to achieve this result: 4

Experiment 2
The optimal set of hyperparameters were:
Dataset: MNIST
Learning rate of S: 0.00402
Learning rate of G: 0.004536
z_dim: 128
Step timings: [0.199, 0.482, 0.971], and scale: 0.364
Final student accuracy: 95.16Total time taken to run: 3166.7564690113068 seconds
Number of epochs taken to achieve this result: 3

K.2 Fashion-MNIST

Range of student learning rates: [0.001, 0.006, 0.0085]
Range of generator learning rates: [0.0001, 0.0004]
Range of step timings: [[0.1, 0.5, 0.9], [0.2, 0.6, 0.933]]
Grid of scales used: [0.3]

Generator - SGD, Student - SGD
The optimal set of hyperparameters were:
Dataset: FASHIONMNIST
Learning rate of S: 0.00509
Learning rate of G: 0.000155
z_dim: 128
Step timings: [0.183, 0.579, 0.933], and scale: 0.384
Final student accuracy: 38.57Total time taken to run: 7556.132112979889 seconds
Number of epochs taken to achieve this result: 50

Generator - Adam, Student - Adam
Dataset: FASHIONMNIST
Learning rate of S: 0.00191
Learning rate of G: 0.003683
z_dim: 128
Step timings: [0.081, 0.446, 0.883], and scale: 0.368
Final student accuracy: 41.6Total time taken to run: 7188.755979537964 seconds
Number of epochs taken to achieve this result: 50