# Reproducibility Report on Adversarial Dropout for Supervised and Semi-Supervised Learning

Alexander Bratyshkin, David Lougheed, Camilo Garcia

*COMP551 - Applied Machine Learning*
*Track 3, Paper #10* *April 20, 2018*

alexander.bratyshkin@mail.mcgill.ca 260684228
david.lougheed@mail.mcgill.ca 260674302
camilo.e.garcia@mail.mcgill.ca 260657037

***Abstract*— In this report we analyze the results obtained when implementing the ideas and experiments presented in "Adversarial Dropout for Supervised and Semi-Supervised Learning". The paper being reproduced combines the already known dropout and adversarial training methods to introduce adversarial dropout as a training technique that yields sparser, less overfitting networks when compared to standard dropout. We trained two CNNs with Adversarial dropout (*$p$*=*1.0* and *$p$*=*0.5*) on the MNIST dataset. We were able to successfully reproduce the empirical results presented by the original paper, with a test score 0.12% higher than that of a base CNN, which confirms that the claims made by the paper regarding generalization have at least some foundation to them. As for sparsity, the mean of the absolute value of the weights of the adversarial dropout model was ~0.003 lower than that of a standard model with a high degree of statistical significance. Considerable time was required to understand and analyse the background concepts of the paper and the mathematical transformations and algorithm proposed.**

***Keywords*— Adversarial Training, Dropout, MNIST, Adversarial Dropout, CNN**

## I. Background

We begin the analysis of the original paper's results by summarizing the two core concepts leveraged by adversarial dropout, as well as the proposed technique itself.

### *A. Dropout Method*

Recall that a fully connected neural network assigns weight to most of the parameters. Dropout is a technique intended to act as a regularizer in neural networks by preventing strong co-adaptation between its features. It was presented in the paper Dropout: A Simple Way to Prevent Neural Networks from Overfitting (Hinton et al. 2012; Srivastava et al. 2014). It effectively disconnects at random neural units during training with a probability hyperparameter *1-p* so that a reduced network has to minimize the loss function. In practical terms, the neurons' contributions to the total network is temporarily removed. After training is complete, all nodes are connected with their revised weights. Testing proceeds using all the activations, but reduced by a factor of *p* to account for the missing neurons during the training. Note that the reduction of the feature space has the secondary positive effect of reducing training requirements in time and space required for its computation. Dropout has thus been proven to be an excellent technique for countering overfitting in neural networks (Hinton et al. 2012).

### *B. Adversarial Training*

A current field of research, this technique aims to train more generalizable machine learning models, and not only just neural networks. Szegedy et al. (2013) demonstrated that standard networks are vulnerable to very minimal noise if the perturbation vectors' direction is sensitive to the model's assignment of labels given the inputs. Adding small but intentionally worst-case perturbations to the training samples yields more robust models, evident in the considerably reduced test error. Note that these perturbations are infinitesimal, meaning that, with for example an image, the noise wouldn't even be visible to the human eye. The current method of defining the adversarial perturbation for supervised training problems is through a unit vector of additive noise applied to the input vector. It requires access to the labels, because they are part of the cost function that the adversarial perturbation is attempting to maximize. Virtual Adversarial Training (Miyato et al., 2016) is the current approach for the derivation of adversarial inputs for semi-supervised training. The key idea is to regularize the model such that given a sample, the model will generate an equal output distribution as the one it would produce on an adversarial perturbed sample.

### C. *Adversarial Dropout*

Combining both concepts intended to improve the generalization of neural networks, adversarial dropout is a training technique where adversarial dropout masks are constructed by altering a random dropout mask. There is an imposed restriction on adversarial changes to prevent severely adversarial, potentially disconnected and therefore nonsensical layers. As was the case with standard adversarial training, the direction of the perturbations is optimized adversarially to the model's label assignment. The gradients of the divergence error (a metric of difference between the true label assignment and standard feed-forward prediction using current weights, not including the adversarial dropout layer) with respect to the inputs of the adversarial dropout layer are gathered and the algorithm proposed by Park et al. iterates until the given threshold restriction boundary by applying the following two actions, which are inherently adversarial:

- Turn off neurons (by setting the corresponding mask position to 0) that decrease the error – i.e. have a negative Jacobian vector entry, and are currently 1 (on) in the dropout mask.
- Turn on neurons (by setting the corresponding mask position to 1) that increase the error – i.e. have a positive Jacobian vector entry, and are currently 0 (off) in the dropout mask.

The threshold boundary is approximately the square root of the maximum number of adversarially modified neurons allowed as compared to the original dropout mask, and is controlled by the below-described hyperparameter $\delta$ (which is the proportion of the number of inputs that are considered in relation to the norm of the difference of the adversarial mask and the original random mask).

This layer has two hyperparameters; *delta* ($\delta$), which controls the intensity of the adversarial perturbation; and *p*, the probability with which the node will remain active (i.e. nodes are dropped with a $1-p$ probability).

## II. Method

Next, we present the goals we had for the the implementation of the paper's methods and experiment and the metrics we recorded in preparation for the comparison with the paper's claims, as well as details on challenges faced during implementation and design choices throughout the process.

TABLE 1
The implemented CNN architecture

| **Name** | **Description** |
|---|---|
| input | 28x28 image |
| conv1 | 32 filters, 3x3, stride=1, ReLU |
| pool1 | Maxpool 2x2, stride=2, pad=1 |
| drop1 | Dropout $p$=0.5 |
| conv2 | 64 filters, 3x3, stride=1, ReLU |
| pool2 | Maxpool 2x2, stride=2, pad=1 |
| drop2 | Dropout $p$=0.5 |
| conv3 | 128 filters, 3x3, stride=1, ReLU |
| pool3 | Maxpool 2x2, stride=2, pad=1 |
| adt | Adversarial Dropout, $p=0.5$, $\delta=0.005$ |
| dense1 | Fully connected 1152 → 625 |
| dense2 | Fully connected 625 → 10 |

### A. *Implementation Goals*

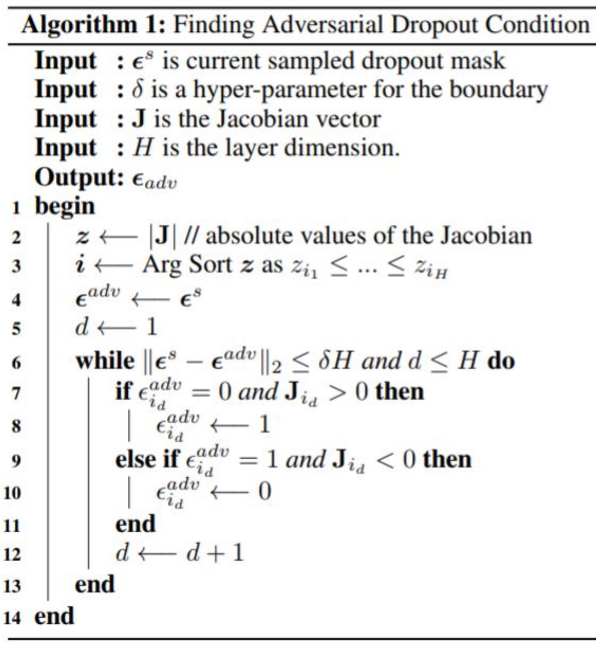
We aim to reproduce the theoretical concept of adversarial dropout described in the previous section for a CNN trained on the MNIST dataset in the context of supervised training. We leveraged all the information available in the paper regarding the architecture of the network, as well as the layer parameters and the model's hyperparameters, but included some modifications. With the results obtained through testing of our slightly-modified CNN we will discuss the claims made by the authors of the original paper regarding the improved generalization performance, the sparsity of the modified network when compared to the same CNN without the adversarial dropout, and finally the claim that the new introduced technique acts as a regularization term. It is necessary to note that Park et al. analyzed their technique for other use cases, such as for supervised and semi-supervised learning tasks on the SVHN and CIFAR-10 datasets. We decided to opt for the easier exercise on the MNIST for several reasons. The main selling point of adversarial dropout is that it's supposed to be a generalizable technique. The paper does not introduce us to new models – it presents a new technique which should be applicable to most models. Hence, we felt that the main challenge resided in the implementation of the adversarial dropout, not in the replication of the models on which we would test the performance of the algorithm, given that all provided implementations were CNNs. It seemed more reasonable to not spend time figuring out the irrelevant (for the purpose of this project) intricacies of SVHN and CIFAR-10 and instead to allocate these efforts towards more meaningful undertakings such as the parsing of the theory behind adversarial dropout, its implementation, etc. Moreover, as we began familiarizing ourselves with the contents of the paper, we realized that the description of the algorithm Park et al. provide (see Figure 1) was destined to perform poorly in terms of run-time (our intuition proved to be correct, as evidenced by the following sections). SVHN and CIFAR-10 are two datasets which are larger in size to MNIST, and Park et al.'s model for dealing with the former problems is much deeper and more complex than the one used for MNIST digit recognition. To reduce our inconvenience during the training process, we picked the simpler model of the two.

## B. *Initial Implementation*

Using the capacities of Pytorch as our main deep learning library, Numpy for concrete math operations, and Google's Colab for the computational resources, we implemented a CNN based on the one presented by the authors in their respective section dealing with the MNIST dataset. We introduced minimal variations to the structure of the network for technical reasons (see Table 1); namely, the output size of the convolutional subnetwork's final layer and thus the input size of the "dense1" layer were changed to 1152. Firstly, we felt that deviating from the original architecture would validate the claim that the technique generalizes well and not only in the exact layer structure the authors implemented. It is expected that adversarial dropout yields similar improvements on all machine learning models compatible with the standard dropout technique. Second, as mentioned in the original article, the authors' framework of choice for all of their implementations was Tensorflow. The padding used in their implementation is set to the parameter "*same*", which is specific to the Tensorflow library – it automatically infers left and right paddings of the filter. Pytorch, on the other hand, does not offer this parameter, hence we were unable to replicate exactly the automatic paddings that Tensorflow added at each layer. We instead used Pytorch 1-column 0-padding to ensure image sizes that could be convolved with multiple filters, which should not have a noticeable effect on learning performance on the MNIST dataset other than inherent effects of having less neurons.

FIGURE 1

Pseudocode for the Adversarial Dropout Condition Derivation.



```
Algorithm 1: Finding Adversarial Dropout Condition
Input  : $\epsilon^s$ is current sampled dropout mask
Input  : $\delta$ is a hyper-parameter for the boundary
Input  : $\mathbf{J}$ is the Jacobian vector
Input  : $H$ is the layer dimension.
Output: $\epsilon_{adv}$
1  begin
2      $z \longleftarrow |\mathbf{J}|$ // absolute values of the Jacobian
3      $i \longleftarrow$ Arg Sort $z$ as $z_{i_1} \leq ... \leq z_{i_H}$
4      $\epsilon^{adv} \longleftarrow \epsilon^s$
5      $d \longleftarrow 1$
6      while $\|\epsilon^s - \epsilon^{adv}\|_2 \leq \delta H$ and $d \leq H$ do
7          if $\epsilon^{adv}_{i_d} = 0$ and $\mathbf{J}_{i_d} > 0$ then
8              $\epsilon^{adv}_{i_d} \longleftarrow 1$
9          else if $\epsilon^{adv}_{i_d} = 1$ and $\mathbf{J}_{i_d} < 0$ then
10             $\epsilon^{adv}_{i_d} \longleftarrow 0$
11         end
12         $d \longleftarrow d + 1$
13     end
14 end
```

The adversarial dropout algorithm implemented initially was an exact replica of the pseudo-code presented by the authors in the original paper (see Fig. 1). The model was trained for 100 epochs and a learning rate of 0.001, in faith to the authors' own hyperparameter tuning. Additionally, we sampled the dropout mask $\varepsilon^s$ with a probability p=1, just like Park. et al did, which seemed like an odd detail of the implementation since the randomness element of the mask appeared to be completely removed (more on this in the challenges section). The adversarial dropout was applied only on the final layer of the convolutional portion of the network, i.e. right before the fully connected layers. The loss function chosen for the calculation of the temporary loss which serves for the calculation of temporary gradients (which, in turn, are necessary for the computation of the Jacobian vector product used in the algorithm presented), was KL divergence with regularization parameter $\lambda = 1$, although Park et al. state that it can be used interchangeably with QE loss (which we also implemented in our code, but did not use formally). As for the actual cost function used for the training and backpropagation of the model, since there was no mention about it in the initial article, we decided to opt for the standard cross entropy loss function because of our previous experience with it and the demonstrated good results it has shown during the Kaggle competition and with general classification problems.

## C. *Modified Implementation*

Subsequently, after noticing long delays in the training process, we implemented an alternative implementation of the pseudocode in Fig. 1 using bitwise operations on bit arrays. First, the randomly generated dropout mask $\epsilon^s$ is converted to a bit array. Then, an initial replacement mask is calculated, where any 1s indicate a candidate entry in $\epsilon^s$ for modification and and 0s indicate ignored entries, using the following bitwise operations, where a comparison operation represents a bit array containing the result of that comparison entry-wise for the whole array:

```
[ϵˢ XOR (Jacobian > 0)] AND Jacobian ≠ 0
```

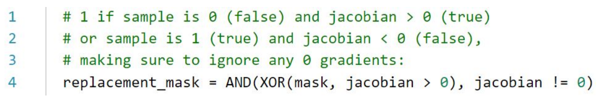
This is a direct translation of the conditionals within the while loop, meaning that selection criteria in this versus the original yield the same results (see Figure 2). To simulate the loop, we can solve for how many nodes need to be changed based on the hyperparameter $\delta$. The loop will terminate after $\sqrt{(\epsilon_1{}^s - \epsilon_1{}^{adv})^2 + \ldots + (\epsilon_H{}^s - \epsilon_H{}^{adv})^2}$ is equal to $ceil(\partial H)$. With the MNIST model, a value of $\partial = 0.005$ was used, and with our modified dense layer, $H = 1152$, resulting in a total of $ceil(\partial H)^2 = 36$ possible neurons switched from their original random dropout state. By taking the top $min(36, |\varepsilon_{nonzero}|)$ bits that met the inner conditions in the index order specified (magnitude of the Jacobian vector entries, small to large) and flipping them, the loop condition was replicated exactly as well.

To check that these optimization modifications did in fact keep the correct algorithm performance, we ran 10 epochs testing the norm of the adversarial mask generated using the original algorithm (see Figure 1) and the new algorithm – in all cases, it was found to be 0 indicating that they generated identical adversarial dropout masks.

Switching out the implementation, an improvement in runtime of approximately 85% was observed, almost certainly as a direct result of the bitwise operations. The average epoch duration for this implementation is around 128 seconds (2.15 minutes), as compared to around 600 seconds for the naive implementation. At 100 epochs, this yields a total training time of about three hours as opposed to a theoretical 14 hours it would have taken the original implementation.

FIGURE 2

Adversarial Dropout Bit-mask Implementation (Inner loop conditions).



```
# 1 if sample is 0 (false) and jacobian > 0 (true)
# or sample is 1 (true) and jacobian < 0 (false),
# making sure to ignore any 0 gradients:
replacement_mask = AND(XOR(mask, jacobian > 0), jacobian != 0)
```

## D. *Baseline Model*

Although Park et al. evaluated their results against a self-ensembling model (Laine and Aila 2016), it appeared to us rather strange that no comparison was provided against a model trained simply with regular dropout in the place of the adversarial dropout. Generally, we would want to compare performance against a more popular (and in this case foundational) technique like Hinton’s original dropout. It is why we found it confusing that Park et al. chose Laine and Aila’s model as an authority, especially considering the fact that their original paper does not even offer benchmarks for the MNIST problem, only results based on SVHN and CIFAR-10. Therefore, to further verify the claim that adversarial dropout is a generalizable technique that should result in improvement versus standard dropout, we decided to introduce a minimal variation to the structure presented in Table 1 to see if, objectively, their claims are true. Namely, we replaced the adversarial layer (i.e. completely neglected the computation of the adversarial dropout noise vector) with a regular dropout layer, keeping the dropout probability of p=0.5. In this way, we should be able to tell if the additional computation time incurred by the adversarial dropout is ‘worth it’ compared to a well-established standard.

## E. *Assessments and Measures*

As the main argument in favor of adversarial dropout is the improvement in generalization performance in the context of supervised classification, we evaluated the performance based on test set accuracy and test loss (i.e. single-fold cross validation), keeping in mind that the dataset is uniformly distributed and we are assuming the random sampling will yield approximately the same ratio of labels for each class in the train/test split. With respect to the alleged improvements on sparsity and regularization, the weights for the first hidden layer were stored for both models for subsequent analysis. To learn the weights of all models, we used Adam optimization as in the paper’s implementation. The discussion of the differences and similarities between our results and those presented in the original paper are discusses in the following section.

## III. Results

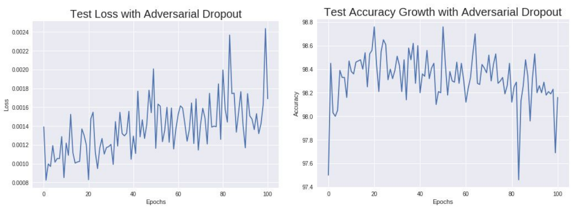
Thankfully, due to Pytorch's very convenient data loading and processing options, we were able to load the MNIST dataset directly with a trivial function call to the library's prestored datasets. Similarly, the shuffling of the data, as well as the train and test splits of the full dataset, were done with predefined methods inside of Pytorch. 60,000 images of size 28x28 were used for training out of the total 70,000, with the remaining 10,000 employed for testing.

No concrete hyperparameter tuning was performed, since the supposed best combination of the hyperparameters for this experiment was already provided by Park et al., and we wanted to attempt to match the results they observed.

As such, we trained our first model with adversarial dropout and a dropout mask of $\varepsilon^s$ sampled with a probability of p=1 as an input to the calculation of the adversarial dropout mask. This was done over 100 epochs, with batch size 64, and $\delta$ = 0.005. The results (i.e. test accuracy and loss) were averaged over two runs of the entire experiment. Note that we were unable to do eight consecutive runs as supposedly performed by the authors of the original paper due to time constraints. Each of our epochs, with our optimized version of the adversarial dropout condition algorithm, took on average 128 seconds to run, and around four to five times as long with the previously-discussed naive algorithm implementation. Consequently, we had to wait approximately three and a half hours before we could obtain the results over 100 epochs. All of this taken into account, we achieved an average test accuracy of **98.18%** after the iterations were done over 10,000 test samples, with a loss of **0.0015**.

FIGURE 3

Test Loss (Left) and Accuracy (Right) Evolution with Dropout Mask $\varepsilon^s$ Sample of All Ones (p=1.0) Over 100 Epochs

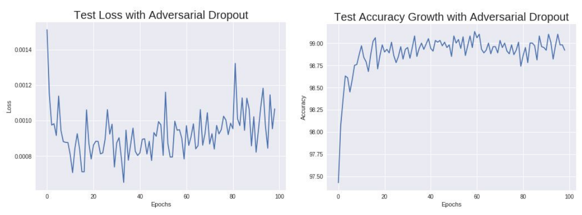


As we can see in Figure 3, the test accuracy fluctuates quite a lot, and the test loss increases rather than decreases over the epochs, indicating divergence. Therefore, under the suspicion that the cause of this divergence is the odd setting of the sampling probability of the dropout noise vector to 1 (all neurons left connected), we decided to run another experiment using a truly randomly sampled dropout mask as the basis for adversarial mask generation.

For our second experiment on the model trained with an adversarial dropout mask, we decided to, at each epoch, create our own truly random dropout mask $\varepsilon^s$ with probability p=0.5, which was then subsequently passed as an input to our optimized algorithm for finding the adversarial dropout condition. We kept the other hyperparameters the same as in the previous experiment, i.e. 100 epochs, with batch size 64, and $\delta$ = 0.005. An average test accuracy of **98.91%** was obtained on the last epoch, an increase of 0.73 percentage points over the previous experiment, while in some cases even peaking at 99.15% maximum test accuracy. As for the test loss, we obtained **0.0011**, which represents a reduction of 0.0004 versus the all-ones starting dropout mask model.

FIGURE 4

Test Loss (Left) and Accuracy (Right) Evolution with Dropout Mask $\varepsilon^s$ Randomly Sampled With Probability 0.5 Over 100 Epochs



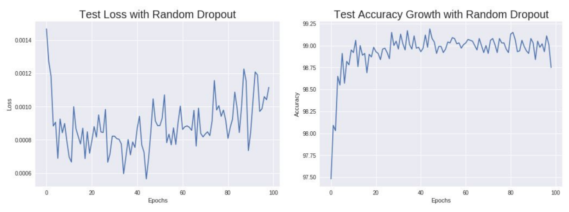
By comparing Figure 4 with Figure 3, we can notice a much smoother growth pattern in the former, with the latter displaying more drastic fluctuations as far as test accuracy is concerned. Additionally, the test loss shows a much clearer decrease trend over the long run for the second experiment, only diverging later, whereas the first one shows almost immediate divergence. Although, indeed, considerably more time should be spent on evaluating this hypothesis, we are nonetheless led to believe that, contrary to Park's claim, the randomness of the dropout mask $\varepsilon^s$ does have an effect on the overall performance of the model (more on this in the following section).

As for our own baseline model, we, again, trained it with the same set of hyperparameters,. i.e. 100

epochs, with batch size 64, and $\delta = 0.005$, though this time we replaced the adversarial dropout with just a regular dropout right before our first dense layer, as per Hinton et al.'s original proposal. This regular dropout was done with a probability of p=0.5, the standard convention suggested in the same paper, and matching the starting dropout mask used for our second adversarial dropout implementation. We obtained a test accuracy of **98.75%** on the last epoch, which represents only a decrease of 0.16 percentage points in regards to our second adversarial experiment. However, the maximum average value that the regular dropout was able to achieve over all of its epochs was 99.22%, which is *higher* than the one obtained for the second experiment (adversarial dropout with p=0.5). Moreover, the test loss was identical to the one previously obtained, i.e. **0.0011**. The main advantage observed during this procedure, however, was the fact that it was significantly faster to execute. Every epoch took, on average, 14 seconds to run, compared to the run-time of the first experiment, which averaged at 128 seconds per epoch, which is almost ten times as long.

FIGURE 5

Test Loss (Left) and Accuracy (Right) Evolution with Regular Dropout Over 100 Epochs



Figures 4 and 5 show relatively similar growth and evolution dynamics for the test loss and accuracy, especially in contrast to Figure 3.

TABLE 2

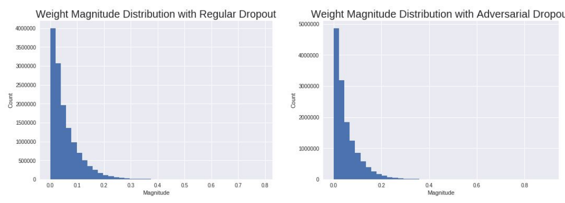
Summary of Test Results Over All Experiments

| Experiment | Test accuracy | Test loss |
|---|---|---|
| Adversarial dropout with $\varepsilon^s$ dropout mask sampled with probability of 1 | 98.18% | 0.0015 |
| Adversarial dropout with $\varepsilon^s$ dropout mask sampled with probability of 0.5 | 98.91% | 0.0011 |
| Regular dropout | 98.75% | 0.0011 |

Finally, after verifying that the performance of the model with adversarial dropout is somewhat similar to that of the regular dropout (though not better to a *statistically significant* degree, which is not necessarily either in conflict with or supporting Park et al.'s presented results), we set out to validate the claim that the adversarial dropout makes the model sparser. As Hinton et al. (2012) state in their original paper, dropout results in sparse activations of the hidden units. Therefore, we took the mean and the variance of the value of the weights leading to the first dense layer of both the baseline model and the better adversarial model of the two (initializing adversarial dropout with p=0.5). These values were taken only after a shortened epoch count of 20, because it is not false to assume that the networks' sparsity levels will change significantly given that the loss and test accuracy do not exhibit any large improvements after the first 10 or so epochs (see Figures 4 and 5). Other hyperparameters were maintained. Given these conditions, for the regular dropout we obtained a mean weight value of **0.054945** and a variance of 0.00286944. For the adversarial dropout, we found a mean weight value of **0.0525915** and a variance of 0.0028134. From this, we can observe that the means of both weight distributions are many standard deviations apart, with the mean of the weights of the layer after the adversarial dropout being significantly lower than that of the regular dropout. Consequently, we can conclude that the adversarial dropout model is sparser, verifying the second of Park et al.'s claims regarding their proposal. The weight magnitude distributions can be found in Figure 6.

FIGURE 6

Weight Magnitude Distributions for Regular Dropout (Left) and Adversarial Dropout (Right)



## IV. Discussion & Challenges

### A. Comparison of Results

It is important to take into consideration the differences between the original paper's experiments and the implementation we present in this paper. For example, we present results for both a fully randomized dropout mask (p=0.5) and one with the same probability (p=1) presented in the original paper. Another example is our improved implementation to derive the adversarial dropout condition which reduced by 85% the running time of each epoch, and while no running time metric was explicitly presented by the authors to which we could compare, for technical purposes and time limitations regarding this paper it proved to be a positive improvement. Another factor to take into consideration when comparing results is that we did not tune hyper-parameters as the goal of our paper is not to find better baselines but rather to corroborate that the architecture and technique presented by the authors does indeed yield the results they present. The performance metrics for our experiment with the random mask of probability p=0.5 (see Table 2) at least partially validate the authors claim that a CNN with adversarial dropout outperforms a base CNN, in our case by 0.16 percentage points after 100 epochs. Though this is not statistically conclusive of the *consistent* superiority of adversarial dropout over regular dropout, it does at least show that it exhibits a comparable performance; we must recall that we did not run the train the model as many times as the original authors, nor did we perform *k*-fold cross-validation due to time limitations. With respect to the claims regarding sparsity, however, the metrics we recorded comparing random and adversarial dropout after 20 epochs yielded much more assertive results in regards to Park et al.'s claims about improvements seen in the sparsity of neural networks with adversarial dropout: we observe a value of ~0.05259 for the mean magnitude of the weights in the adversarial dropout model as opposed to a mean of ~0.0549 for the standard model, implying that the former is much sparser and thus less prone to overfitting. Hence, we obtain a model which improves upon a base learner at least in terms of sparsity and even potentially in terms of performance due to the addition of an adversarial dropout layer.

### B. The Random Sample Mask Issue

As previously mentioned, this was a rather central issue surrounding our implementation. Throughout Park et al.'s whole paper, the authors refer to the so-called $\varepsilon^s$ model parameter, which represents a sampled random dropout mask instance, i.e. a dropout noise vector which was generated with a predetermined dropout probability $p$. It appears to play a significant role in the computation of the adversarial dropout because, firstly, it is the initial value that the adversarial dropout noise vector receives (see line 4 in Figure 1), secondly, it is used to calculate termination condition of the algorithm (see line 6 in Figure 1) and, lastly, has a definite effect on the overall value of the adversarial dropout condition, for the adversarial dropout actually performs an infinitesimal adversarial variation upon this original $\varepsilon^s$ vector. However, in their sampling process of $\varepsilon^s$ for the first model contained in the appendix of the original paper, the authors proposed a sampling with probability p=1, i.e. feeding a vector full of ones (of the same size as the linear layer succeeding the final max pooling layer) to the algorithm calculating the adversarial dropout condition. The reason this appeared strange to us is the following: suppose that we do use an $\varepsilon^s$ sampled with p=1 – then, it would seem particularly odd if any significant dropout would be happening at all. The first part of the exit condition of the while loop on line 6 of Figure 1, i.e. the difference of the squared means of the two vectors, imposes a restriction on the difference of the dropout condition so as to make this divergence minimal. If we were to attempt to calculate the adversarial dropout by optimizing adversarially to all gradients (or, equivalently, by iterating over all of the Jacobian vector product), we would risk obtaining a vector which would "shut down" all neurons of a layer. The $\delta$ hyperparameter gives us the ability to control the number of gradients against which we want to optimize our adversarial condition, so as to warrant that the feasible space for the adversarial dropout can only "kill" a miniscule amount of the most active neurons. Conversely, if we were to proceed with the $\varepsilon^s$ vector of all ones, we would obtain an adversarial vector which would only result in a minimal amount of dead neurons, thus increasing the chances of overfitting. In hopes of clearing up our confusion, we decided to reach out to one of the authors of the paper, specifically Sungrae Park, via electronic correspondence. He acknowledged our concern, and his justification for such a decision was that "when [he] changed the initial state of dropout mask [...], [he] got similar performances" (quote lifted directly from the e-mail). In essence, his choice was made based on empirical observations through a trial-and-error approach, rather than through some particular theoretical foundation. However, as evidenced in our results section, the dropout mask generated with probability p=1 performed worse in terms of test accuracy than the one generated with probability p=0.5 by a rather substantial amount of 0.73%, legitimizing our aforementioned claims, which leads us to believe that perhaps the authors should have made a more thorough checking of their experiment.

## C. *Run-time Performance*

Perhaps the main drawback of the whole adversarial process, at least as far as our implementation was concerned, was a speed issue. Indeed, as mentioned above, not only did the implementation of the rudimentary adversarial algorithm provided in the original paper have a dreadful execution time, but even our optimized version, which is almost twice as fast, ran for up to ~2.5 minutes per epoch on some occasions, and this is on a very lightweight dataset such as MNIST – one could only begin to imagine the bottleneck horrors on problems which require gigabytes of more complex data. The very nature of the algorithm depends on the fact that we must process the data sequentially, i.e. batch per batch, as well as each entry for each batch, and this for every single training epoch. It is true that this process could be parallelized with the power of GPUs, since for every entry in a batch, which is stored in a matrix, we could've calculated an individual adversarial vector without depending on the results of the other entries (rows). However, the authors of the original paper did not provide any optimization suggestions or descriptions of more advanced techniques for computing the adversarial dropout, and our own knowledge of CUDA unfortunately falls short in the face of such a task. Additionally, it is worthy to note that the calculation of the adversarial dropout condition wasn't the sole reason for the run-time handicap (although very majoritarily so): during the training phase, we were required to feed our samples forward through both the pre-adversarial layers of the network and the post-adversarial layers while simultaneously storing the outputs of the last hidden convolutional layer in a buffer so as to obtain both the gradients as well as the temporary loss needed for the calculation of the Jacobian product vector. Then, after the adversarial dropout was found, we performed an entry-wise multiplication of it and the outputs of the stored hidden layer. The product was then propagated *again* through the dense layers, after which a regular cross entropy loss was computed. Evidently, though not as much as the adversarial dropout calculation, this still represented a significant delay if compared against a model trained only with regular dropout.

## D. *Resource Cost*

Because the MNIST dataset that we used to replicate the experiments of the paper is relatively lightweight by machine learning standards, we did not require access to any additional computational resources besides Google's Colab GPU accelerator. However, initial training times for the code presented under section *II Method, B. Initial implementation,* proved to be long enough to hinder the development process of the group. As we required complete results before being able to perform any transformations to the model and its hyperparameters. Our attempt at a better running time, presented in *II Method, C. Modified Implementation,* aimed to accelerate the implementation of the paper. Nonetheless, the largest part of our resources was spent in the static analysis of the paper and its supporting sources. During this early stage we chose to communicate with the author in order to clarify ambiguities in the initialization of the dropout mask. Development effort was considerably reduced due to the previous experience of the team members using Pytorch's library and the CNN theory applied during the Kaggle competition. The last point of interest with respect to the resources required to accomplish this project was the adaptation of the notation, API methods and library constants provided by the framework chosen by the authors, Tensorflow, to our framework of choice Pytorch. While it was part of our design goals to stress test the technique under analysis, we did not want to work completely tangentially to the original architecture.

## V. CONCLUSION

The adversarial dropout technique proposed by Park et al. turned out to be an effective improvement upon the regular state-of-the-art method of Hinton et al.'s (2012) simple dropout. Although the gain in test performance is not particularly notable, adversarial dropout did offer a significant increase to the sparsity of our CNN model for the MNIST digit recognition problem by leveraging the philosophy of adversarial training to provide an enhancement to regular dropout. Nonetheless, the algorithm presented in Park et al.'s paper does impose a severe runtime bottleneck, and thus additional optimization efforts are necessary for the future. In spite of this, we believe that adversarial dropout is a positive step towards the resolution of the overfitting issue in neural networks and is definitely a technique worthy of attention.

## VI. Acknowledgment

We are greatly indebted to Sungrae Park of the Department of Industrial and System Engineering (KAIST) who was exceptionally helpful, insightful and friendly throughout our inquiries regarding his paper and the design choices of his experiments.

## VII. Statement of Contributions

**Alexander Bratyshkin:** I worked on interpretation of theory, largely on the implementation of the algorithm and the models, and also on the writing and editing of the report and executive summary.

**David Lougheed:** I also worked on the interpretation of theory in the paper, especially with the algorithm of interest; was involved in the creation of the optimized version of the algorithm, and participated in the writing and editing of the report and executive summary.

**Camilo Garcia:** I worked on interpreting and researching methods presented in Park et al., assisted with our implementation of the methods described, and wrote a large part of the report and executive summary.

We hereby state that all the work presented in this report is that of the authors.

## VIII. References

Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. R. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv*:1207.0580.

Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15(1):1929–1958.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv*:1312.6199.

Miyato, T.; Dai, A. M.; and Goodfellow, I. 2016. Virtual adversarial training for semi-supervised text classification. *stat* 1050:25.

Laine, S., and Aila, T. 2016. Temporal ensembling for semi-supervised learning. *arXiv preprint arXiv: 1610.02242.*

## Appendix

### A. Executive Summary

We chose track number three for the final project. It required us to analyze and corroborate claims made by a scientific paper in a current research field of machine learning by implementing by ourselves the experiments that supported those claims. In this report we analyze the results obtained when implementing the experiments presented in Adversarial Dropout for Supervised and Semi-Supervised Learning (Park, et al. 2017). Because the paper being reproduced leverages recent techniques and concepts not seen in COMP551, we spent considerable time resources researching the two main building blocks of the technique presented by the authors: the dropout method and adversarial training. The first one, introduced in the paper Dropout: A Simple Way to Prevent Neural Networks from Overfitting (Hinton et al. 2012; Srivastava et al. 2014), aims to reduce overfitting. It achieves better test performance by disconnecting, at random, neural units during training with a probability so that a reduced network has to minimize the cost function. The second, proposed by Szegedy et al. (2013) targets the vulnerability of networks to very minimal noise if the perturbation vectors' direction is sensitive to the model's assignment of labels given the inputs. By adding small but intentionally worst-case perturbations to the training samples, the training phase yields more robust models which can be seen in a considerably reduced test error. The introduced technique, adversarial dropout, unifies both concepts into a layer which is intended to adversarially optimize a given dropout mask to train a sparser, less overfitting CNN when compared to a network with a standard dropout layer.

In order to test the authors' claims, we trained two CNNs with adversarial dropout on the MNIST handwritten digit recognition problem. We attempted to test the boundaries of the technique by slightly modifying the architecture of the CNN from the one presented in the paper in order to verify that the performance improvements generalized to multiple architectures and not only on a single limitated setup. Both models were inspired from Park et al.'s original implementation, deviated only by slight variations like hidden layer sizes. The first model was trained with a drop mask probability initialized with probability p=1.0 (all ones, a very counter-intuitive decision from the authors which we criticize in our report and which forced us to reach out to Sungrae Park in hopes of clarification), and the subsequent model with a drop mask probability sampled at p=0.5. While the former model did not outperform a base model, the latter embodied the characteristics stated by the original authors – a test accuracy performance improvement of 0.12 percentage points was observed when compared to a base learner, though we found that such a small deviation did not represent much statistical significance overall. However, this outcome was still valuable for us in conjunction with the sparsity result because it implied that, at least, adversarial dropout did not hinder the performance. The mean magnitude of the weights of the adversarial dropout model's first fully-connected layer was 0.003 lower than that of a standard model, which indicates that, indeed, adversarial dropout helps produce sparser networks, thus resulting in models less prone to overfitting. However, despite coming up with an enhancement for the original algorithm which improved the run-time by 85%, the adversarial technique slowed down the training process tenfold compared to a regular dropout layer (128 vs. 14 seconds, respectively). In light of the noticeable increase in sparsity, though, and similar testing score for a CNN, we feel confident in our assertion that Park et al.'s work signifies a positive development towards the resolution of overfitting issues in neural networks. Perhaps it is in the interest of researchers to come up with methods of optimizing adversarial dropout by utilizing the full parallelization capacities offered by CUDA or similar methods.