UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Aggregation of Dropout Subnetworks

Paul Miller
Michele Alberti
Vinaychandran Pondenkandath
Rolf Ingold

Department of Informatics - Bachelor Project Report

Département d'Informatique - Departement für Informatik • Université de Fribourg - Universität Freiburg • Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland

phone +41 (26) 300 84 65 fax +41 (26) 300 97 31 Diuf-secr-pe@unifr.ch http://diuf.unifr.ch

Contents

Abstract

Deep neural networks are used in several areas of Machine Learning, including image recognition. However, the large number of neurons makes it harder for a neural network to generalize beyond training data. A relatively new method to prevent this problem is Dropout, described in the paper [1], which achieved good results on several datasets. This project tries to find a better way of aggregating the subnetworks that Dropout creates at test time by taking a look at their outputs at test time. We overview aggregation methods such as using the median and average of subnetwork outputs. In addition, we implement a voting system for subnetworks which improves performance by 0.2% on CIFAR-10 and 0.009% on MNIST. We also describe a scoring system for subnetworks which shows an accuracy increase of up to 0.2% on CIFAR-10 and 0.015% on MNIST over standard Dropout. These methods may be useful for getting better results from a neural network which has been trained using Dropout at the cost of increased testing time.

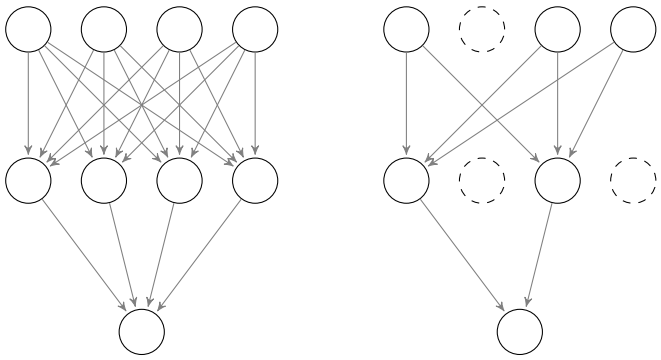
Keywords: Dropout, subnetworks, neural networks

1 Introduction

Deep neural networks contain a large number of neurons, which makes it possible to retain the complex relationships of input data. However, one downside is that the neural network will have less difficulty memorizing all of the training examples and may not be able to generalize beyond training data at test time. This issue, called overfitting, is a common problem in the field of Machine Learning and has no one-size-fits-all solution.

This issue is prevented by the relatively new method called Dropout, described in [1]. By setting a probability $0 < p < 1$ that the output of neurons will be retained during training, the Dropout technique trains some of 2^n possible thinned neural networks.

Figure 1: Standard vs Dropout training



(a) Without Dropout (b) Subnetwork example with Dropout

Figure 1 (a) shows an example of a forward pass during training without Dropout. Figure 1 (b) shows one of the subnetworks that could be trained by Dropout during test time. At test time, we multiply the output of neurons by the probability of their output being retained p and we use all neurons.

The effectiveness of this method is based on the fact that this allows the neural network to approximate the geometric mean of all possible subnetworks at test time without the need of computing outputs of an exponential number of subnetworks, giving every subnetwork the same weight. Furthermore, neurons have to act independently of others because they can't count on another neuron's output, which regularizes the neural network and prevents overfitting. The Dropout method achieved good results on several datasets including MNIST, SVHN, CIFAR-10, CIFAR-100 and ImageNet. A newer technique called DropConnect, introduced in [5], has shown better performance on many datasets by ignoring weights during training rather than neurons.

In standard neural networks trained without Dropout, we don't really have any way to get more information at test time, since we will only have one output for every input and we cannot determine how confident the neural network is. The goal of this project is to use additional information by computing the outputs of subnetworks created by Dropout at test time in hopes of finding a better way of aggregating the outputs of the thinned neural networks.

Figure 2: Aggregation of subnetworks at test time

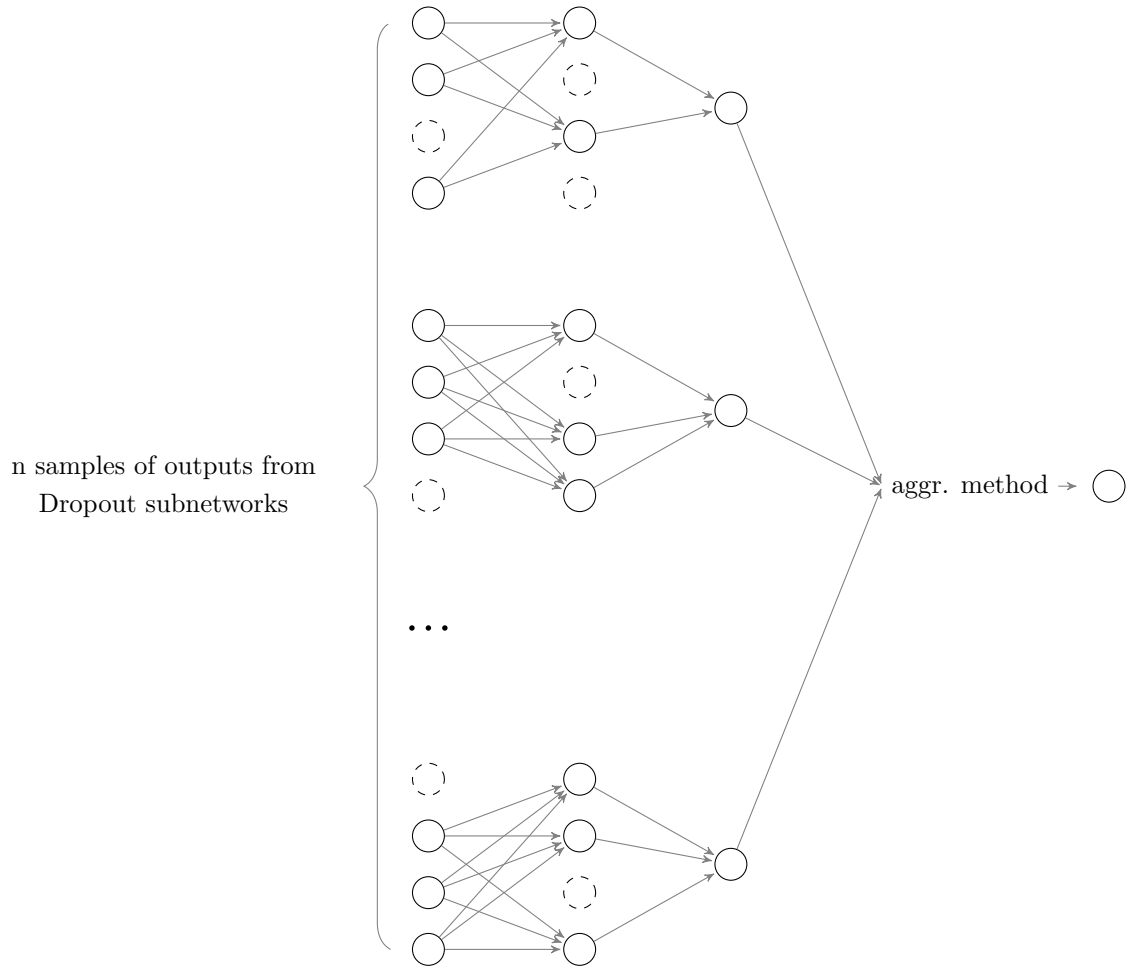


Figure 2 shows an example of what we try to do during this project. If we let Dropout ignore neurons during test time, we can gather a large number of outputs for the same input in the hopes of getting a better output than averaging all possible outputs, which is what the standard Dropout method does at test time by calculating an approximation of the geometric mean of all possible subnetworks.

To make this task easier, we have implemented our own class within DeepDIVA [2], which is a Deep Learning Framework which uses PyTorch. Unfortunately, we were not able to create a working version to load trained networks, so we retrained networks for every experimental run. However, this has the benefit of making our results more reliable, since we do not focus on one specific initialization, but it is a lot more time consuming. Moreover, we have chosen to work with the AlexNet model from [8] for our final results since it contains 2 Dropout layers with $p = 0.5$, is big enough to overfit and has good performance on the databases we have chosen. Furthermore, it doesn't use batch normalization, described in [7], which is another technique behaving differently at training time. Additionally we demonstrate results on the MNIST database of digit recognition and the CIFAR-10 database of object classification to obtain networks with higher accuracy (about 99.5% on MNIST) and with lower accuracy (80% on CIFAR-10). We used CIFAR-10 first and then checked results on MNIST because it is easier to see differences in accuracy, since there is more room for improvement[1].

[1]See experimental results in section 4

2 Contributions

This paper is structured as follows. Firstly, we will take a look at the subnetworks that Dropout creates and how they work together by computing the arithmetic mean and median of a sample of subnetworks to compare them with the standard Dropout technique using the CIFAR-10 and MNIST databases. Secondly, we will see if we can use standard deviation and the number of subnetworks which are giving different predictions to make a better decision for subnetwork aggregation. Then we will go over the two methods we have chosen to describe in this project, which are a voting system and a scoring system for subnetworks along with their results on the CIFAR-10 and MNIST databases. In addition we will see how we could improve them. Finally, we will detail further work and means of improvement.

3 Related work

The paper [1] describes the Dropout technique and how it reduces co-adaptation of features by approximating a geometric mean of an exponential number of models sharing parameters. This method trains and uses all thinned nets at test time efficiently but does not look in detail into other means of aggregating Dropout subnetworks.

The DropConnect method described in [5] offers a generalization of Dropout by acting on the weights the model instead of the neurons of fully connected neural networks. DropConnect has been shown to often outperform Dropout with very little difference in training time.

Using test-time Dropout, also referred to as Monte Carlo Dropout, has been used to estimate model uncertainty by Bayesian approximation, as seen in [9], therefore using the information from models that had been ignored so far. It studies the properties of Dropout's uncertainty to use it in deep reinforcement learning.

The follow-up paper [11] proposes a new variant of Dropout which obtains better uncertainty estimates and improves performance. It replaces the need for a grid-search over the Dropout probabilities by using a continuous relaxation of Dropout's discrete masks.

Recently, the application of Monte Carlo Dropout has also been shown that it can be used as a Bayesian estimator for deep neural networks (DNN) to improve speech enhancement in unseen noise and signal-to-noise ratio conditions in [10]. It uses Dropout to estimate model precision and dynamically select the DNN models based on their test data performance.

4 Preliminary results

4.1 Dropout subnetworks

What we have found to be a pretty good starting point is to compute the accuracy of the subnetworks that Dropout creates to see how they perform at test time.

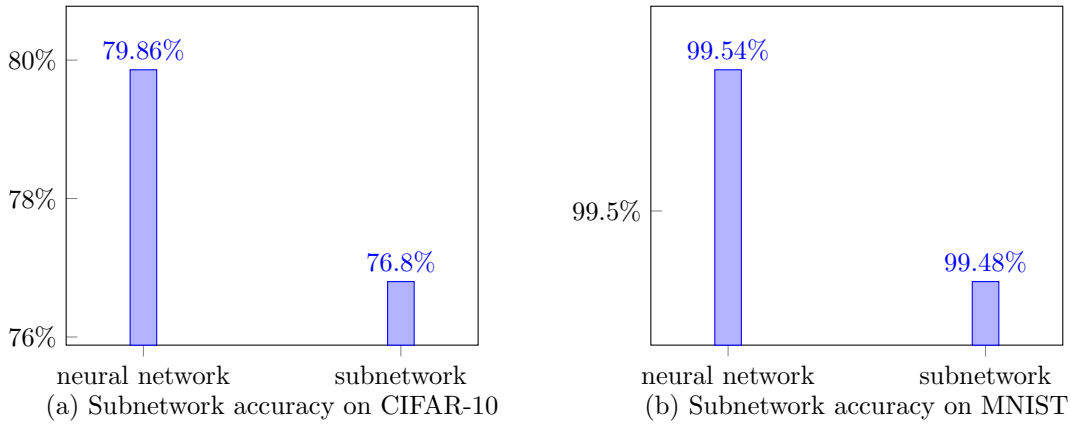
4.1.1 Experimental setup

By simply leaving the model in training mode in PyTorch, we will randomly pick one subnetwork at test time for each mini-batch. We did this by computing the standard Dropout output and running another forward pass with Dropout active (to get the output of a single subnetwork) for each mini-batch and evaluating both accuracies.

4.1.2 Experimental results

In the next figure we look at how the entire neural network compares to using one subnetwork at test time.

Figure 3: Accuracy of subnetworks



(a) Subnetwork accuracy on CIFAR-10 (b) Subnetwork accuracy on MNIST

Using the AlexNet model at 20 epochs (single run), Figure 3 (a) and Figure 3 (b) show the accuracy of subnetworks on CIFAR-10 and on MNIST respectively. Using all of the subnetworks at test time by multiplying the output of neurons by p, we can see that the Dropout technique increases accuracy by 3.06% on CIFAR-10 and 0.006% on MNIST. These values were taken from a single run on both databases only to give an idea of the accuracy of the subnetworks we will be working with.

4.2 Standard Dropout, arithmetic mean and median

In theory, multiplication at test time of the output of a neuron (Dropout) gives the same weight to all possible thinned subnetworks by approximating their geometric mean. However, a large amount of these 2^n subnetworks will never be trained and we may be able to gather information to better aggregate the subnetworks by computing the outputs of a sample of subnetworks. We can do this by putting the model in training mode and running multiple forward passes at test time for each batch, therefore giving us different outputs for the same input. This behaviour is specific to Dropout from how it regularizes the network (creating subnetworks which can act independently) and it is something that we might be able to take advantage of.

In this section, we chose to take a look at the arithmetic mean (average) and the median of outputs of samples of subnetworks to see how Dropout compares to other methods of averaging at test time.

4.2.1 Experimental setup

We consider the task of image classification on the CIFAR-10 and MNIST databases. Using the implementation of the AlexNet model of the DeepDIVA [2] framework at 20 epochs, we will calculate the accuracy of the standard Dropout technique. Moreover, we choose $N \in \{50, 100, 200, 500, 750\}$ forward passes at test time on CIFAR-10 and $N \in \{25, 50, 100, 200\}$ on MNIST to compute the average and median of outputs for each mini-batch and get their accuracy on the test set. By computing these values on 20 different networks for each number of samples, we are reducing the risk that results are dependent of the accuracy of the entire network and, by extension, of a specific initialization of the model. Results shown below are the average accuracy out of 20 runs for each method of aggregating subnetworks.

4.2.2 Results on CIFAR-10

Table 1: Accuracy loss/gain on CIFAR-10

nb samples	Dropout	median	average
50	**79.8745%**	–0.0645%	–0.023%
100	80.0775%	+0.1765%	**+0.2065%**
200	79.6835%	+0.1315%	**+0.15%**
500	80.279%	+0.1475%	**+0.156%**
750	79.863%	+0.089%	**+0.1175%**

Firstly, we see that Dropout performs worse than both other methods when we use more at least 100 samples of subnetworks. It looks like the standard Dropout method may average the outputs too much at test time and reduce the accuracy of the network.

Furthermore, using the arithmetic mean of at least 100 subnetwork outputs is often better than other methods. Taking the mean of 100 subnetwork outputs seems to be a good way to increase the accuracy of the AlexNet model on CIFAR-10. However, this is not always the case. Using 100 samples, the arithmetic mean performs better than Dropout 14 out of 20 runs so it isn't better than Dropout at all times. Out of these runs, the highest gain in accuracy by using the arithmetic mean over Dropout was 0.73% and the biggest loss in accuracy was 0.31%.

We see that the median has similar results than the arithmetic mean, which suggests that there are no significant outliers on one side of the spectrum making the network less accurate. However, it seems to have worse accuracy than the average of subnetworks.

We can see that using the mean of between 100 and 500 outputs of subnetworks improves accuracy by 0.11 to 0.21%. However, the gain of using the average of samples seems to be less significant when using a larger number of samples but we would need more tests to confirm this, such as using a higher number of samples.

The average and standard Dropout technique differ for approximately of 323.9 predictions (i.e. choose a different class), where Dropout classifies the input correctly 107.2 times, the average gets it right 127.85 times and both methods give wrong predictions 88.85 times. Keeping in mind that this is specific to this model and to the number of epochs we choose, this means that we could improve accuracy of the average of subnetworks by more than 0.1% on CIFAR-10 (since the test set contains 10'000 images) if we were able to pick the best of both methods at test time (gaining up to 0.3% over Dropout).

4.2.3 Results on MNIST

Table 2: Accuracy loss/gain on MNIST

nb samples	Dropout	median	average
25	**99.474%**	–0.0025%	–0.0035%
50	99.4905%	**+0.0045%**	+0.0025%
100	**99.467%**	–0.007%	–0.001%
200	99.464%	+0.006%	**+0.0065%**

We can see that the high accuracy of the network makes it harder for the average of subnetworks to improve its accuracy by using the means of a sample of subnetworks. This can be explained by the fact that the subnetworks have good accuracy and the Dropout method does work efficiently to get good results during test time.

We don't think it is worth trying to analyze the results from this table, since the differences are so small and the results seem inconclusive. However, we may be able to get more useful information if we increase the number of subnetwork samples or experiments.

From what we've seen with our CIFAR-10 results, using the average of about 100 samples instead of the standard Dropout method might be a good way to improve the accuracy of a network which doesn't have high accuracy by increasing computation time. However, the gains don't seem worth it, since it adds a lot more testing time.

We compute the accuracy of the median and average during all subsequent experiments to compare them to other methods which will try to improve accuracy. From these results, we know that we can sometimes improve the performance of a neural network trained with Dropout by computing the mean output of a sample of subnetworks. Furthermore, we learned that we could gain a lot of accuracy if we could distinguish a wrong prediction from a right one.

5 Observations

5.1 Standard deviation

Another method of aggregating Dropout subnetworks at test time that we looked into is using the standard deviation of outputs. We might be able to use this information to try and make a better decision as to the weight we give to a specific thinned subnetwork.

A natural choice is to see how the sample outputs vary in order to calculate how confident the network is. The next figure is the kind of situation we would want to find with the MNIST database for digit recognition. The outputs are the values for classes of digits 0 (top) to 9 (bottom).

Figure 4: Example of using standard deviation

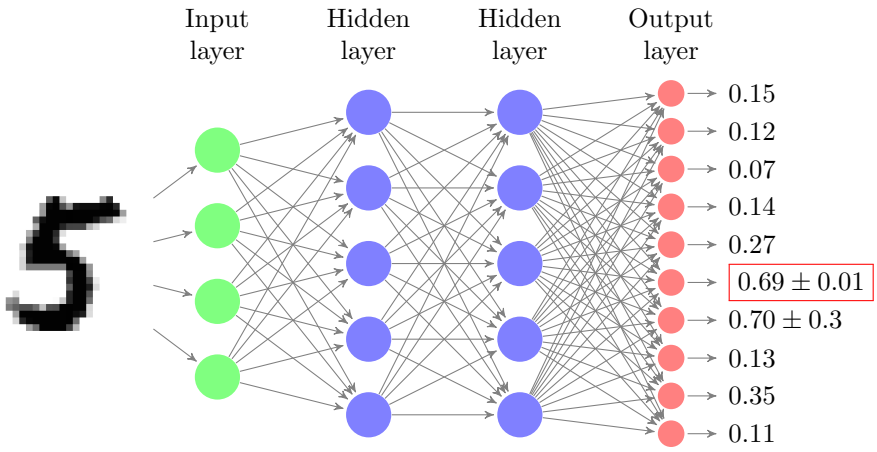


Figure 4 shows a situation that the standard Dropout technique would get wrong, but that we could get right if we use additional information on the output. We would choose the output that doesn't vary as much, because it shows that the neural network is more confident in its result.

The way we tried to implement this is by computing the standard deviation of the outputs of subnetworks and giving less importance to the predictions that vary the most, thinking that they would be the values that we are less sure about.

This method would have the benefit of using the outputs of a sample of subnetworks, but without too much computation, since we are only using them to compute the standard deviation. The simplest version of this would be to divide the output by the standard deviation of the outputs of subnetworks. This does not work for many reasons, for example:

- differences between low and high values are treated equally
- outputs and how much they vary depend on the loss function
- we do not take into account the fact that values change naturally (for example the fact that the digit 9 looks like the digit 4 in MNIST[2])

We tried to introduce different factors and reduce these effects for example by changing the range of the standard deviation values or normalizing the output, but we could not get to a point where we saw any improvement.

Another idea we had was trying to divide or subtract the standard deviation of subnetworks by the standard deviation during validation for each class. The idea was that, since classes vary differently compared to others, we could remove this effect and be left with how the network is varying and see how confident it is in its output for each class.

Furthermore, we tried to multiply the top value by the second value's standard deviation and vice versa in order to see if this could help choose between the top two values and prevent prediction

[2]More on this in subsection 7.1

errors. This would solve the situation described in Figure 4 while not changing the outputs of other classes.

From what we've observed, standard deviation does not seem to be a sign of uncertainty, but only the consequence of some subnetworks having higher output values.

To sum up, we were not able to use the standard deviation of outputs of subnetworks created by Dropout with these methods neither to get an idea of how confident a network is in its output, nor to improve the final output.

5.2 Conflicting predictions

Another calculation we found interesting is that if, when the network is wrong, we pick the class with second highest value for the average of outputs, we will avoid a lot of mistakes. For CIFAR-10, we could get more than half of the predictions right if we just picked the second class when we were wrong. This means that if we could switch our answer to the second prediction when we are wrong, we could highly increase accuracy. But knowing when we are wrong is no small feat.

A natural way for trying to figure out when we are wrong is to see if the subnetworks are classifying the same input differently. An idea would be to only adjust the outputs of subnetworks if there are subnetworks which predict different values, i.e. if the thinned neural networks aren't predicting the same value as the average of subnetwork outputs.

5.2.1 Experimental setup

Here are results of some tests performed with the AlexNet model on the MNIST and CIFAR-10 databases (average of 20 runs, 20 train epochs, 100 samples of subnetworks with Dropout active at test time). For every example in the mini-batch, we count the number of subnetworks that don't predict the same class as the mean of all subnetworks to determine if we can figure out when the average of subnetworks is giving the right prediction. We decided to take the average of subnetworks instead of the standard Dropout result in light of the experimental results of subsection 4.2 and because we do not have access to this standard Dropout result when the model is in training mode.

5.2.2 Experimental results

Table 3: Conflicting subnetworks when the average gives the right prediction

	MNIST	CIFAR-10
Average	0.2485	6.891
Maximum	53.95	77.8
Minimum	0	0

Table 4: Conflicting subnetworks when the average gives the wrong prediction

	MNIST	CIFAR-10
Average	20.9445	28.461
Maximum	55.9	80.6
Minimum	0	0

These values are dependent of the number of samples in the sense that we observe roughly the same ratios of conflicting and non-conflicting subnetworks given a different number of samples. On CIFAR-10, the average of conflicting subnetworks when the mean of outputs gives the wrong prediction is a bit over 28% of samples. When the mean of outputs gives the right prediction, this number is only 6.8%.

We can draw these conclusions from these results[3]:

[3]These assumptions are true for all experiments we have run, not only those using 100 subnetworks

- There are on average more conflicting subnetworks when the output is wrong
- A high number of conflicting subnetworks doesn't necessarily mean that the output is wrong
- A low number of conflicting subnetworks doesn't necessarily mean that the output is right
- The maximum number of conflicting subnetworks is always higher for the wrong predictions

This shows that the number of conflicting predictions could be used to have an idea of how confident we are in a prediction, but isn't enough by itself to know when the classification is wrong. In other words, if the number of conflicting subnetworks is higher, there is a higher probability that the mean of outputs is giving the wrong output class. However, we can't be sure about the output class only because there are no conflicting predictions since high and low numbers occur in both cases.

In light of this, we tried taking the second highest prediction when the number of conflicting subnetworks is higher than a certain threshold but this only added more errors.

5.3 What we tried using what we've seen so far

As we noted in the last remark when we compared Dropout, average and median on the CIFAR-10 database, we could gain at least 0.1% accuracy if we were able to distinguish between a good or a bad result. We could compute all 3 values (Dropout, average and mean of subnetworks) at test time and picking the "best" prediction (for example using the standard Dropout output when the average is wrong or vice versa). The main methods we have tried to choose the best output were:

- Using the output with the highest value for the predicted class (in hopes of getting the result which is most sure that the output class is the right one)
- Using the output with the lowest value for the predicted class (in hopes of getting a better average of all subnetworks which is more balanced)
- Using the output which has the highest difference between the top 2 predictions (in hopes of getting the result which is most sure that the output class and not another class)
- Using the output which has the lowest difference between the top 2 predictions (in hopes of getting the result which is more balanced)

The above methods did not improve accuracy. We could not find a way to distinguish between good and bad outputs when comparing the 3 different outputs of these methods. We also tried different functions to normalize the outputs in order to be able to compare them in a more meaningful way and distinguish good from bad outputs to no avail. In addition, we also attempted to use these criteria to pick one subnetwork as final output, in case this might be a good way to determine the best of the sampled subnetworks.

Furthermore, we have tried to use the number of conflicting predictions as a way to prevent from adding errors to the network. For example we tried to multiply or divide outputs by their standard deviation like taking the top two values and multiplying them by the other's standard deviation[4], but only when there is at least a certain number of subnetworks that disagree with the prediction from the mean of subnetwork outputs.

Moreover, we tried to find a link between the number of conflicting subnetworks and when a specific averaging method is right or wrong. For example, we were hoping that Dropout might be wrong more often than the mean of subnetworks when there are a lot of conflicting subnetworks. Or maybe the median would be the best output when there are more than 50% of conflicting subnetworks, but the values seemed to be more or less the same for each method.

We have also tried to only take the subnetwork which has the highest or lowest top value only when there are a certain number of conflicting subnetworks in order to see what effect this would have on accuracy. This seemed to have more or less the same effect of dropping the accuracy of about 5% on CIFAR-10 compared to standard Dropout, which seems to be worse than taking a random subnetwork at test time[5].

[4] As mentioned in subsection 5.1

[5] From the results of subsection 4.1, we see about 3% accuracy loss when using one subnetwork at test time

6 Voting system for subnetworks

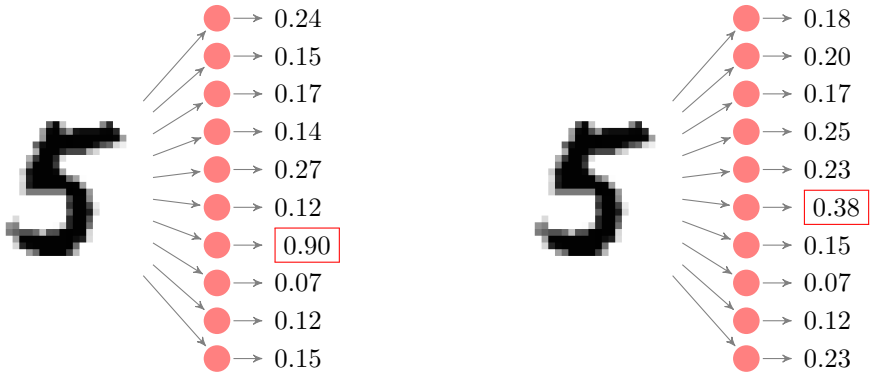
6.1 Description of a simple voting system

From results of the previous section, we have not found a way to judge how confident a neural network is in an output and cannot distinguish between a "good" output and a "bad" output and therefore don't know what subnetworks to give more weight to. We have seen that values not being close to each other isn't a reliable way to judge the confidence of a neural network. So maybe what we are doing by averaging outputs is actually reducing the accuracy of our network.

Dropout trains a lot of subnetworks and makes them work together at test time. By using the mean of a certain number of outputs, we will add up the outputs of all subnetworks regardless of the range of their values. By doing this, we give less weight to the subnetworks that have outputs that are close to each other for every class.

Here is an example of this situation on MNIST (with the output at the top being the output for the class 0):

Figure 5: How a voting system could help



(a) Output with different values

(b) Output with similar values

In Figure 5 (a), we have an example of output from a subnetwork which classifies the digit as a 6. Figure 5 (b) shows an example of a subnetwork which classifies the image correctly as a 5. Here is what happens if we take the arithmetic mean of both outputs:

Figure 6: Average of both outputs

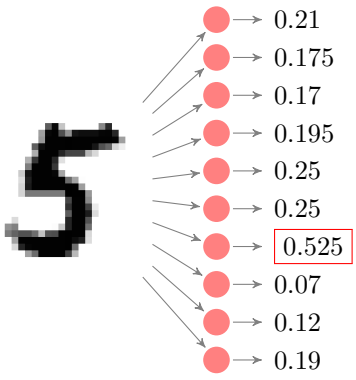


Figure 6 shows that both outputs are averaged to a point where the second subnetwork's output (Figure 5 (b)) has very little influence on the result. The solution to this would be to normalize the set of subnetwork outputs at test time.

A simple and effective way around this problem is by implementing a kind of voting system for subnetworks. Rather than taking the subnetwork's output in its entirety, we could just break it down to a single vote for a class. In Figure 5, we would have one vote for the class 6 and one vote for the class 5. This has the benefit of really giving the same weight to every subnetwork since every subnetwork only has 1 vote that counts as much as all others without too much computational strain. Furthermore it also is closer to the real situation by the fact that either the digit is a 5 or a 6 and that the network has been trained with examples that only have one output class. By analogy, if we ask a person what digit it is, they will only give a single answer. This method may also prevent us from comparing outputs of neural networks which don't have the same weights.

In case of a draw between two classes, we add the mean output for both of the highest classes to the result in order to decide between them. In the previous example, we would get 1.525 votes for class 6, 1.25 votes for class 5 and no votes for other classes. Even though the result still classifies the input wrong, we are closer to the result and this method may be able to prevent errors when using more subnetwork samples.

6.2 Experimental setup

Using the implementation of the AlexNet model of the DeepDIVA [2] framework on MNIST and CIFAR-10 at 20 epochs, we compute the accuracy of the standard Dropout technique and gather $N \in \{50, 100, 200\}$ subnetwork outputs at test time. Then we go through all subnetwork outputs at test time and count the number of votes for each output class. In case of a draw, we add the mean output for both of the highest classes to the vote total. Results shown below are the average accuracy out of 20 runs for each method of aggregating subnetworks.

6.3 Experimental results

Table 5: Voting system results on CIFAR-10

nb samples	Dropout	median	average	voting system
50	80.0125%	+0.172%	**+0.2535%**	+0.2185%
100	80.0905%	+0.115%	+0.129%	**+0.1405%**
200	79.9995%	+0.1825%	+0.2025%	**+0.2035%**

Table 6: Voting system results on MNIST

nb samples	Dropout	median	average	voting system
50	**99.439%**	–0.0075%	–0.009%	–0.0135%
100	99.466%	+0.0015%	**+0.009%**	+0.0015%
200	99.4725%	+0%	+0.0025%	**+0.009%**

We can see that the voting system improves performance over the standard Dropout technique on both databases when using more than 100 samples. Using more subnetwork outputs seems to increase accuracy of the voting system, which is logical since it will normalize outputs more effectively and more outputs should lead to better classification.

With the voting system, we can see up to 0.2035% accuracy gain over Dropout and 0.009% at the cost of test time. This may be a useful technique to increase accuracy of a neural network which contains Dropout layers at the cost of adding more test time. The downside of this method is that we might have to use a large amount of subnetworks at test time for it to be a viable option and we have to compute the average of subnetworks in case of a draw, which adds extra computational strain.

6.4 Alternate version using softmax

Another version of this voting system we tried was using the softmax function to normalize the subnetwork outputs instead of just reducing each subnetwork output to one simple vote for one output class. Due to the nature of the softmax function, if we apply it to the output of a subnetwork we will get an array with values which sum up to 1. By adding up the softmaxed subnetwork outputs and using this result as output for our model, we implement another voting system. This aggregation method gives one vote to each subnetwork which is divided over the output classes. The advantages of this system are that we don't need to handle draw cases, because they are improbable and we take into account all output class values. However, it will use more computations than the previous method.

6.5 Experimental setup

Using the implementation of the AlexNet model of the DeepDIVA [2] framework on MNIST and CIFAR-10 at 20 epochs, we compute the accuracy of the standard Dropout technique and gather $N \in \{50, 100, 200\}$ subnetwork outputs at test time. Then we apply the softmax function to subnetwork outputs at test time and use the sum as final output of the model. Results shown below are the average accuracy out of 20 runs for each method of aggregating subnetworks.

6.6 Experimental results

Table 7: Softmax voting system results on CIFAR-10

nb samples	Dropout	median	average	softmax system
50	79.838%	+0.035%	**+0.0915%**	–0.7765%
100	79.6505%	+0.1055%	**+0.167%**	–1.0645%
200	80.302%	+0.0675%	**+0.1025%**	–1.255%

Table 8: Softmax voting system results on MNIST

nb samples	Dropout	median	average	softmax system
50	99.468%	–0.006%	**+0.0005%**	+0.002%
100	99.493%	+0.004%	+0.003%	**+0.005%**
200	99.4375%	–0.004%	–0.0005%	**+0.001%**

We can see that this version of the voting system reduces accuracy on CIFAR-10 drastically. Since this does not happen with the other voting system, either applying the softmaxed outputs don't give enough weight to the highest output for a class in the final result or they give too much weight to other classes.

Results on the MNIST database show that it may improve accuracy in higher accuracy models. Because the improvements are not consistent for both databases, we do not see this as a viable method for aggregating subnetworks.

7 Scoring system for subnetworks

7.1 Mean of outputs during validation

We have also tried to collect information on the neural network during the last validation by gathering logits for each target class with Dropout active to calculate the mean of outputs from the classification layer. The idea behind this is to take advantage of the fact that classes have recognizable patterns. Here is an example of the mean of outputs during validation over MNIST with the AlexNet model after 20 epochs (simple run):

Table 9: Example of mean output for every class on MNIST

	0	1	2	3	4	5	6	7	8	9
input digit 0	**31.24**	–8.61	–3.64	–7.55	–1.08	–6.12	1.24	–4.77	–4.00	2.85
input digit 1	–3.17	**20.76**	–0.55	–7.47	0.01	–2.93	–1.75	0.90	–3.12	–3.57
input digit 2	–2.13	–0.89	**24.07**	0.93	–0.36	–9.20	–2.12	0.91	–3.95	–6.99
input digit 3	–10.49	–0.28	–4.30	**26.93**	–11.22	7.77	–7.32	–3.93	1.03	1.99
input digit 4	–10.55	–2.31	–3.31	–8.89	**32.87**	–6.57	–2.71	0.34	–3.42	4.92
input digit 5	–3.26	–5.13	–9.38	7.19	–10.43	**25.24**	1.62	–6.00	–1.44	1.34
input digit 6	3.55	–0.24	–5.08	–7.56	–3.09	6.44	**29.82**	–15.72	0.88	–9.38
input digit 7	–8.02	2.99	2.67	–1.79	2.37	–5.59	–9.98	**20.88**	–6.06	2.23
input digit 8	–4.22	–8.59	–0.22	0.56	–7.07	0.33	–3.27	–9.11	**32.44**	–0.31
input digit 9	–5.28	–5.52	–7.23	–3.56	9.04	–4.70	–11.27	0.54	0.63	**26.74**

Each line in the table above contains the mean output during validation of the logits layer when the target is a specific class.

For example, from the last line, we can see that, when given an input image of the digit 9, the neural network will have a harder time distinguishing it from a 4 than a 6. Furthermore, we see that the input of a 5 kind of looks like a 3 and vice versa. However, 6s look a bit like 5s but 3s don't look like 6s.

These values are inherent to how each class behaves in relation to the others. Therefore, we could know when we have an output which doesn't resemble what we expect. For example, if a neural network classifies an input image as a 9 but also has a high value for the output class 6, it would be good to doubt the entire output since we know that when the digit is a 9, it normally has a low output for the class 6.

7.2 Formula using the mean of outputs during validation

Based on the results of the previous subsection, we decided to use a formula that gives more weight to outputs which look more like the outputs we find during validation.

Consider the matrix $Y = (y_{ij})$ of size NxN, where N is the number of output classes, such as each line $[y_{i1}, ..., y_{iN}]$ contains the mean output of inputs from class $i \in [1, ..., N]$. For example, on MNIST, the first line of Y will contain the mean of outputs during validation when the input is the digit 0.

At test time, we compute K forward passes on the same input. For every forward pass $k \in [1, ..., K]$, we get an output $x^k = [x_1^k, ..., x_N^k]$, which predicts a class $h \in [1, ..., N]$ (i.e. the highest value in the array x^k is the h-th value). We compute the Euclidean distance (sum of squared differences) between the output x^k at test time and mean output of the class predicted by x^k:

$$\alpha_k = \sum_{i=1}^{N} (x_i^k - y_{hi})^2$$

After computing α_k for every forward pass k, we calculate $\forall k$:

$$\omega = \max(\alpha_k)$$

The result ω stores the maximum of the sum of squared differences of all forward passes. This will simply be the α_k value of the output that differs the most from what it should look like. We can then use this value to give outputs more weight when they are similar to what they should be:

$$z = \sum_{k=1}^{K} x^k \frac{(a\omega - b\alpha_k)}{\omega}$$

By using the formula above we use a weighted sum as final output, we give more weight to outputs that actually look like what we are expecting in order to include how confident we are in subnetwork outputs.

Another theory for why this would work is that the subnetwork outputs which don't resemble what we are expecting may be from subnetworks that use neurons that were not picked during training time or were not trained for the input class. This scoring system might also give more influence on the final output to subnetworks which have neurons that have been trained the most.

We decided to set $a = 1$ and $b = 1$, which will multiply the original outputs of subnetworks x^k by values in $[0, 1]$ (we will ignore the value that has the highest distance from the mean during validation). We decided to use these values for a and b and compare them to the other methods We have previously used (average, median, Dropout) to see if there is any improvement. We also tried to see if this system made it possible to use a smaller number of subnetworks at test time, to reduce computation time and attempt to make it a feasible alternative to Dropout's testing time routine.

A downside of this method compared to the voting system is that we must have a finite amount of subnetworks in order to compute the value ω so, once the result z is computed, we cannot decide to make more forward passes to get a more accurate results unless we recalculate all values.

7.3 Experimental setup

Using the implementation of the AlexNet model of the DeepDIVA [2] framework at 20 epochs, we calculate the mean of logits on the validation set with Dropout active during the last epoch of validation. Then we compute the accuracy of the standard Dropout technique and gather $N \in \{10, 25, 50, 100, 200, 500\}$ subnetwork outputs at test time for every mini-batch, which we can use to calculate the accuracy of the median and average of subnetwork outputs. Subsequently we evaluate the weighted sum given by our formula to determine the scoring system's output. The following results show the average accuracy over 20 runs on the CIFAR-10 and MNIST databases.

7.4 Experimental results

Table 10: Accuracy loss/gain on CIFAR-10

nb samples	Dropout	median	average	scoring system
10	**79.5805%**	–0.286%	–0.121%	–0.0825%
25	80.1255%	–0.029%	**+0.082%**	+0.0815%
50	80.0295%	+0.092%	+0.159%	**+0.1975%**
100	79.9675%	+0.1465%	+0.195%	**+0.2045%**
200	79.9145%	+0.065%	+0.0795%	**+0.118%**
500	79.8585%	+0.141%	+0.148%	**+0.1935%**

Table 11: Accuracy loss/gain on MNIST

nb samples	Dropout	median	average	scoring system
10	99.472%	–0.019%	–0.01%	**+0.015%**
25	99.474%	–0.0025%	–0.00325%	**+0.0065%**
50	99.4905%	+0.0045%	+0.0025%	**+0.009%**
100	99.467%	–0.007%	–0.001%	**+0.015%**
200	99.464%	+0.006%	+0.0065%	**+0.012%**

The results show that the scoring system improves accuracy of the network when using more than 25 samples of subnetworks. Moreover, they reveal that the scoring system is an improvement over using the mean of outputs in every single case (except with 25 samples on CIFAR-10, but the difference of 0.0005% is quite small).

We can see up to +0.2045% accuracy on CIFAR-10 with this scoring system and up to +0.015% accuracy on MNIST when using 100 samples. It is worth noting that the difference on MNIST is non-negligible, since state-of-the-art results contain about 0.21% error, from the paper [5]. If similar results could be obtained with the DropConnect method, we might be able to slightly improve accuracy.

These results are good since we see that the scoring system improves results when we use more than 25 samples, which is not the case with the other methods we have described in this paper. However, 100 samples seems to achieve the best improvement for both databases. We would need more tests to confirm these results, since these are small differences in accuracy.

We have also compared every run to see how often the scoring system performs worse than Dropout. When using 100 samples on MNIST, Dropout does achieve higher accuracy for 6 out of 20 runs. This ratio drops to 4 out of 20 runs for CIFAR-10. Therefore, using the scoring system isn't always better than Dropout but it is in most cases.

8 Conclusion

The Dropout technique prevents overfitting by training some of 2^n possible thinned neural networks by setting a probability $0 < p < 1$ that the output of n neurons will be retained. At test time, we multiply the output of neurons of Dropout layers by p and use all neurons, effectively approximating the geometric mean of all possible subnetworks.

The aim of this project was to inspect the subnetworks and how they work together at test time, hoping to be able to find a better method of aggregating subnetworks by using a sample of forward passes with Dropout active.

We have seen that the arithmetic mean and the median of subnetworks sometimes get better accuracy than Dropout, gaining on average up to 0.2% accuracy on CIFAR-10 over standard Dropout using the same model. However, it does not seem to be a viable alternative to Dropout, since it adds a lot of computation time but doesn't always improve the accuracy of the network.

Furthermore, we have described a voting system which tries to overcome the fact that some subnetworks output a different range of values than others and have more influence on the standard output by giving only one vote for an output class per subnetwork. Results show this is an effective way to normalize the outputs, which improves accuracy of the AlexNet model by 0.2% on CIFAR-10 and 0.009% on MNIST when using 200 subnetworks at test time.

Finally, we have outlined a scoring system for subnetworks which relies on a formula using the mean of outputs for each class. We will use this formula to compute a weighted sum of the outputs of subnetworks which gives more weight to outputs which look like what we expect by computing the Euclidian distance between the subnetwork output and the mean output during validation. Experimental results show improved accuracy on average of up to 0.2% on CIFAR-10 and 0.015% on MNIST when using 100 forward passes (less than the voting system). However, it does add a significant amount of computation time during testing and it might need some tweaking to find an optimal formula. We might also be able to use this technique to improve the DropConnect method.

9 Further work

9.1 Implementation

Although how we trained neural networks from scratch for every run and averaged 20 runs for our results does give a better approximation of what would happen usually, it would have been less time consuming to load a trained model to only run the test phase. We would need to store the mean output during validation during training to be able to run the scoring system described in section 7. This would also help get a better idea on what happens when we sample a bigger number of subnetworks.

Another improvement which could speed up computation time is to use PyTorch Tensors instead of NumPy arrays to be able to increase computational speed with CUDA GPUs. We decided to use the latter since we were trying to get a more general idea of methods of aggregating Dropout subnetworks rather than finding a viable method that would replace standard Dropout. Also, using a NumPy array with 3 axes to store the outputs of subnetworks seemed like the easiest option.

9.2 Aggregation methods

9.2.1 Mean and median

For the mean and the median of subnetworks, we would need to do more testing to confirm that there might be some benefit to using the average or median of a sample of subnetworks rather than using the standard Dropout aggregation. The results we showed, although promising, are a bit uncertain and do not always work. We weren't able to find out why they work sometimes or not, although this might be due to some subnetworks having more influence over the final aggregation of Dropout because of high outputs.

9.2.2 Improvements to the voting system

For the simple voting system, which tries to solve this problem by giving only 1 vote per subnetwork to one class, we could also try to give some smaller votes for the second highest prediction. This would helps us make a better decision in case of a draw and maybe remove the need to compute the mean of outputs therefore reducing computational strain when using a large number of samples. We could also choose to run another forward pass to get a deciding vote.

Furthermore, we could try to pick situations where the voting system gets the final output wrong and try to see how we could solve this.

For the softmax version of the voting system described in subsection 6.4, we could adjust the results by multiplying the result by a constant or by adding a vote to the highest output class (effectively using the sum of both voting systems as the result).

9.2.3 Improvements to the scoring system

For the formula of our scoring system, we could try to use different values for a and b to see what would be optimal. There might be different solutions given the accuracy of a neural network or even the database we are working with. We could also incorporate the voting system by replacing the subnetwork output by an array with only 1 vote before multiplying it by the coefficient we use in our formula.

Furthermore, we could decide to use this formula only when the subnetworks are conflicting (the value described in subsection 5.2) in order to decrease testing time since we have seen that these are the cases that the network most often gets wrong. We could also use different values for a and b in the formula given how many subnetworks are conflicting. For example, we might consider using higher values for a and b when there are a lot of conflicting subnetworks to make it easier to decide between the different options.

Instead of giving a subnetwork less weight when it doesn't look like what we expect, we could find what the output actually looks like. With the method described above, we only compare a subnetwork output with the mean during validation of the class which gives the highest value. If we computed the sum of squared differences for all classes, it would show if the differences are smaller

for values other than the highest prediction. The way we could implement this is by computing the sum of squared differences between the subnetwork output and the mean output for every class and we would then choose to predict the class which has the lowest sum (i.e. the one with values closest to the subnetwork's output). The simplest way of doing this would be to implement a voting system as seen in section 4. This system would have the same benefit of the voting system, such as giving only 1 vote for every subnetwork which will count as much as the others, and would also take advantage of the mean output of classes during validation. However, this would add a lot more computation time and these computations often aren't necessary.

9.3 More experiments

Even though taking 20 runs at 20 epochs seems like it is a good starting point to take a look at the methods, it is necessary to make more experiments to see if these results are conclusive. We have also tried using smaller convolutional neural networks and a network with 2 linear layers of 8192 neurons which seemed to give similar results, but we mainly focused on the AlexNet model.

We could also try different databases in order to get a more global view of how our methods are performing. We have seen that results are very different on the CIFAR-10 and MNIST databases, this might also be the case for other databases and this may give a better idea why.

In addition, it may be interesting to inspect the increase in computational time of both the voting and scoring systems.

9.4 DropConnect

Recent architectures are moving away from Dropout in favor of other means of regularization, such as Batch Normalization. This method is very effective at regularizing networks, especially convolutional layers.

As mentioned in the introduction, Dropout is sometimes replaced by the DropConnect method, which works by ignoring weights and not neurons, which can remain partially active. What we have done in this thesis could be transferred to a network which uses DropConnect, since it works in the same way and the subnetwork outputs may be more reliable.

References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15*, pp. 1929–1958, 2014.

[2] M. Alberti, V. Pondenkandath, M. Würsch, R. Ingold and M. Liwicki. DeepDIVA: A Highly-Functional Python Framework for Reproducible Experiments. *arXiv:1805.00329*, 2018.

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

[4] S. Wager, S. Wang, and P. Liang. Dropout training as adaptive regularization. In *Advances in Neural Information Processing Systems 26*, pp. 351–359, 2013.

[5] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, R. Fergus. Regularization of Neural Networks using DropConnect. *Proceedings of the 30th International Conference on Machine Learning*, pp. 1058–1066, 2013.

[6] O. Dekel and O. Shamir. Learning to classify with missing and corrupted features. In *Proceedings of ICML 25*, pp. 216–223, 2008.

[7] S.Ioffe, C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *arXiv:1502.03167 [cs.LG]*, 2015.

[8] A. Krizhevsky, I. Sutskever, G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Communications of the ACM 60 (6)*, pp. 84–90, 2017.

[9] Y. Gal and Z. Gharamani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. *arXiv:1506.02142*, 2015.

[10] Nazreen P.M. and A.G. Ramakrishnan. Using Monte Carlo dropout for non-stationary noise reduction from speech. *arXiv:1808.09432*, 2018.

[11] Y. Gal, J. Hron and A. Kendall. Concrete Dropout. *arXiv:1705.07832*, 2017.