Controlled Dropout: a Different Approach to Using Dropout on Deep Neural Network

https://ieeexplore.ieee.org/document/7881693

2017A3PS0205P Adithya Shankar Bhattiprolu
2017A7PS0061P Aayush Atul Verma
2017B4PS0506P Jyoti Ranjan Pagoda

Introduction

The occurrence of overfitting while building neural network architecture is a very common problem encountered by those working in the Deep learning community.

Overfitting is when the model learns noise and other irrelevant data and tries to fit the training dataset perfectly thus reducing the training error but when this same network is exposed to a test sample it fares poorly. Overfitting affects generalisation drastically.

As a result over the years a lot of study and research has gone into finding ways to reduce the problem of overfitting.

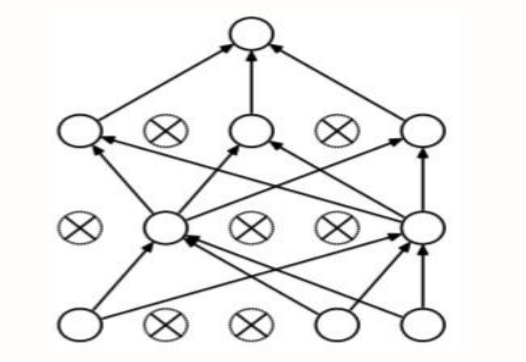
This led to creation of regularization techniques. These techniques when used while training of a Neural network will help reduce the possibility of overfitting.

Dropout

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.

In dropout, neurons are randomly dropped out of training with a pre -defined probability. (It is a hyper parameter that can be adjusted to get optimal performance.)

This is done in order to prevent overfitting and reduce the dependence of neurons on one another during training.



Controlled Dropout

The proposed method in the research paper is controlled dropout. It is claimed to be a modification of the existing dropout technique in terms of computational efficiency.

Unlike traditional dropout, controlled dropout intentionally picks which neurons to drop and accordingly the weights corresponding to that neuron are dropped for forward propagation too.

This is done in order to reduce the number of redundant mathematical computations that will be done in traditional dropout. (Storing values of zero at each node that has switched off)

Controlled Dropout

It is used to regularise deep neural networks and reduce the computation time. An example of the proposed technique with a dropout rate of 0.5 is shown below. Units from (1,1) to (6,1) and (1,5) to (6,6) are dropped to obtain a well-formed rectangle from (1,2) to (6,4) in matrix a(l). Corresponding to these, the W(l) matrix has hatched pattern squares which constitute the redundant zero element multiplication computation, forming an organized shape as opposed to the traditional method.

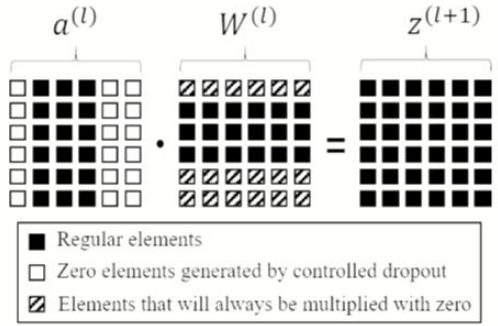


Figure a. Controlled dropout

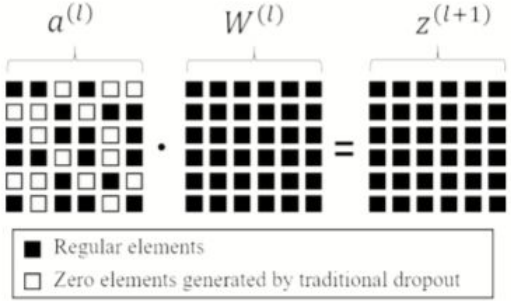


Figure b. Traditional dropout

Controlled Dropout

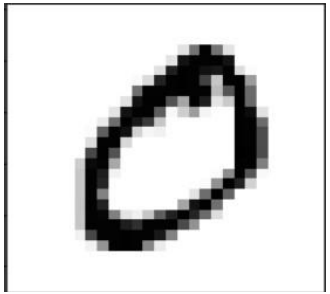
We want the rectangle in a(l) to be chosen randomly on every iteration and every layer, so we choose a random number between 1 and the total number of units. This will be first active neuron. Since the dropout rate is an input, we know the number of units that should be active and hence we can find the last index that should be active.
Hence before each matrix multiplication we calculate the required range of neurons and corresponding weights that should be active, and accordingly only take these subsets of the a(l) and W(l) matrices for multiplication to avoid redundant multiplication operations.

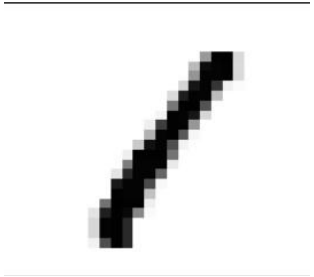
Dataset and Pre Processing

The dataset used for the project was the MNIST dataset.

It consists of black and white images of size 28 x 28 of handwritten digits in the range 0 to 9.

The images were initially in Uint8 format, We converted them into floating point format and divided each pixel value by 255 in order to scale all the pixel intensity values to the range of 0 to 1 for easier computation





Model

The architecture used for implementing Controlled dropout and Regular dropout is an ANN with 2 hidden layers. (784 - 2000 - 2000 - 10).

The activation function used for the hidden layers is ReLU and softmax function for the output layer.

The loss function used was Multi class cross entropy and ADAM was used as an optimizer for weight adjustment.

Hyperparameter Tuning

Controlled dropout was implemented using Pytorch framework while regular dropout was implemented using Keras.

Probability of dropout was fixed at 0.5 and learning rate in both cases was 0.00005. No dropout has been applied to the input or the output layer.

For the training task we implemented 3 separate cases for both networks and recorded the data obtained -

a) 15000 training images with 310 epochs and a batch size of 256
b) 20000 training images with 230 epochs and a batch size of 256
c) 25600 training images with 180 epochs and a batch size of 256

Points Regarding Training of Network

For testing the performance of the network we used a test dataset that comprised of 1/10th number of images in the training dataset for all runs.

The values were chosen in order to ensure a total of 18000 weight updates.

All values specified in the following tables are final values obtained after averaging the values obtained after 5 successive runs at the same time on the same pc.

Below we provide 2 tables each corresponding to a run of the same program at different dates. Though we had multiple runs we decided to document only 2 of them to showcase the results.

Problems We Faced

Initially our models for both traditional as well as controlled dropout was overfitting the data. (The validation accuracy kept dropping while the training accuracy kept increasing.). This was solved by increasing the number of training images to a suitable value.

Another problem we faced was that the validation accuracy was consistently greater than the training accuracy during training.

It was mentioned online that for high values of dropout probability it was possible as only a few neurons are being trained during the training stage and hence its accuracy is low but in the validation stage the entire network is used as a result its accuracy is higher. The problem was overcome by tuning the optimizer (Adam) and reducing the dropout rate to 0.5

Problems we faced

Problems while utilising Google colab. The same program without changing anything took different times to run at different times of the day. This caused a problem because we were not able to accurately record the training time.

We looked online on google Colab's official "Help" site. Here it was stated that the resources provided by Colab such as type of GPU, idle timeout periods and other factors vary dynamically and this leads to allocation of different types of resources which in turn affected the model's performance.

Despite this in all of our runs of both models on the same pc lead to the same result of Controlled dropout converging faster than traditional dropout.

Google Colab Usage Limits

What are the usage limits of Colab?

Colab is able to provide free resources in part by having dynamic usage limits that sometimes fluctuate, and by not providing guaranteed or unlimited resources. This means that overall usage limits as well as idle timeout periods, maximum VM lifetime, GPU types available, and other factors vary over time. Colab does not publish these limits, in part because they can (and sometimes do) vary quickly.

GPUs and TPUs are sometimes prioritized for users who use Colab interactively rather than for long-running computations, or for users who have recently used less resources in Colab. As a result, users who use Colab for long-running computations, or users who have recently used more resources in Colab, are more likely to run into usage limits and have their access to GPUs and TPUs temporarily restricted. Users with high computational needs may be interested in using Colab's UI with a local runtime running on their own hardware. Users interested in having higher and more stable usage limits may be interested in Colab Pro.

Figure. Screenshot of Question from Google Colab's help section explaining the problem we faced.

Quantitative Results

A)

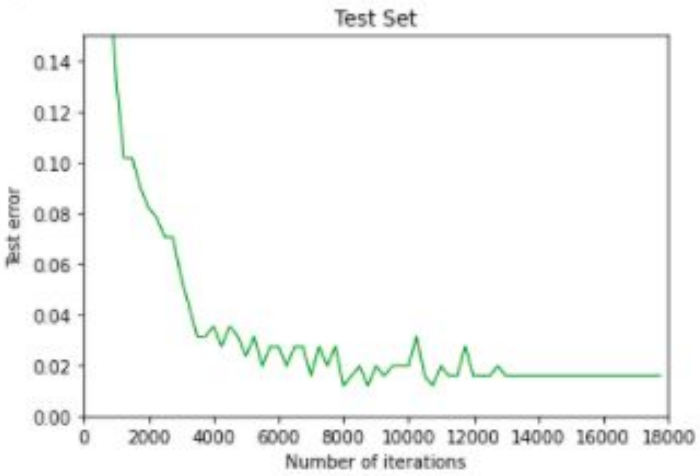
Case	Time for convergence (Dropout)	Time for convergence (Controlled Dropout)	Test Accuracy (Dropout)	Test Accuracy (Controlled Dropout)
a) 15000 training images	82.535 seconds	42.125 seconds	96.266%	94.322%
b) 20000 training images	79.415 seconds	41.823 seconds	97.000%	94.925%
c) 25600 training images	71.581 seconds	41.826 seconds	96.718 %	95.849%

Quantitative Results

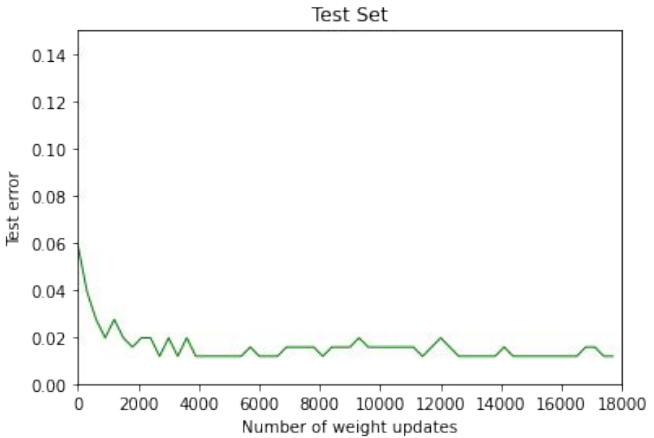
B)

Case	Time for convergence (Dropout)	Time for convergence (Controlled Dropout)	Test Accuracy (Dropout)	Test Accuracy (Controlled Dropout)
a) 15000 training images	177.456 seconds	110.231 seconds	96.733%	95.341%
b) 20000 training images	174.485 seconds	109.511 seconds	96.700%	95.462%
c) 25600 training images	165.987 seconds	109.32 seconds	96.914%	96.102%

Graphical Observations



a) Controlled Dropout



b) Traditional Dropout

Figure. Test error of Controlled and Traditional Dropout against number of weight updates for the case with 25600 training images.

Conclusions

In all three cases the final test accuracy of both the networks was approximately the same. Controlled dropout initially showed sudden spikes in accuracy during training whereas traditional dropout showed a much smoother drop in test error as the iterations proceeded.

Increasing the test set size also showed an increasing trend of generalisation in both traditional as well as controlled dropout but the traditional dropout technique seemed to converge faster than controlled dropout initially till around 4000 updates.

Conclusions

The execution time does not vary much if the number of weight updates during training are kept constant, in spite with a change in training size.

However, in all three cases, the time spent per weight update in controlled dropout was considerably lower. Hence even though both techniques gave a similar accuracy, controlled dropout finished its training and converged much faster than traditional dropout in terms of time.

The networks were trained on Google Colab using a GPU. All times specified are in accordance with the GPU used on Google Colab

Thank You