AUTHOR(S): BARROW, E., EASTWOOD, M. and JAYNE, C.

TITLE: Selective dropout for deep neural networks.

YEAR: 2016

Publisher citation: BARROW, E., EASTWOOD, M. and JAYNE, C. 2016. Selective dropout for deep neural networks. Lecture notes in computer science, 9949, Neural information processing: Proceedings of 23rd International conference on neural information processing (ICONIP 2016), 16-21 October 2016, Kyoto, Japan. Cham: Springer [online], pages 519-528. Available from: https://doi.org/10.1007/978-3-319-46675-0_57.

Selective Dropout for Deep Neural Networks

Erik Barrow[1] and Mark Eastwood[1] and Chrisina Jayne[2]

[1] Coventry University, Coventry, West Midlands, UK
[2] Robert Gordon University, Aberdeen, Scotland, UK

Abstract. Dropout has been proven to be an effective method for reducing overfitting in deep artificial neural networks. We present 3 new alternative methods for performing dropout on a deep neural network which improves the effectiveness of the dropout method over the same training period. These methods select neurons to be dropped through statistical values calculated using a neurons change in weight, the average size of a neuron's weights, and the output variance of a neuron. We found that increasing the probability of dropping neurons with smaller values of these statistics and decreasing the probability of those with larger statistics gave an improved result in training over 10,000 epochs. The most effective of these was found to be the Output Variance method, giving an average improvement of 1.17% accuracy over traditional dropout methods.

Keywords: MNIST, Artificial Neural Network, Deep Learning, Dropout Network, Non-Random Dropout, Selective Dropout

1 Introduction

Dropout is an effective method for reducing overfitting in neural networks [1] that works by switching off neurons in a network during training to force the remaining neurons to take on the load of the missing neurons. This is typically done randomly with a certain percentage of neurons per layer being switched off. Dropout has also been previously tested on large well known data sets such as MNIST [2, 1], ImageNet [3, 4] which is significantly larger than MNIST, and CIFAR [5, 1].

We propose a new method of dropout that selectively chooses the best neurons (neurons which will have the biggest positive effect on the network if switched off) to be given a higher probability of being switched off on the assumption that dropout could be made more effective and efficient by not dropping neurons that should be forced to continue to learn. Our main contributions are 3 new methods of selecting these neurons as described in section 2.2.

Recently some other methods of altering the way dropout works have surfaced. These include a method of adaptive dropout that was explored by Ba and Frey, where they change the probability of neurons being switched off using a binary belief network, which runs separately from the neural network being trained [6]. This allows the network to find the optimum drop rate for any given neuron.

Another Modified dropout method was introduced by Duyckm et al, where they apply optimisation methods to dropout such as simulated annealing [7]. Dropout has also been used in an innovative way by Wan et al to create a new method called DropConnect, where a random subset of the weights on a neuron are randomly set to zero instead of all of a neurons weights [8].

Our experiments on Selective Dropout will be conducted using the MNIST dataset [2], a dataset consisting of pre-processed handwritten digits. MNIST is a well-known and well-used dataset in the field of deep learning and will provide results that can be easily benchmarked against other research. Number recognition is a widely used machine learning application, with applications such as number plate recognition, and automated bank teller machines. Dropout has also been previously tested on large ImageNet [3, 4] which is significantly larger than MNIST.

To test the effectiveness of selective dropout against that of the traditional dropout we will use a Deep Neural Network architecture. Since 2006 Deep learning has been an effective method of machine learning since the discovery of the use of pre-training deep networks with Restricted Boltzmann Machines [9]. All our experiments will be pre-trained by a stack of Restricted Boltzmann Machines in two different hidden layer architectures, these are [500,500,500] & [500,500,1000]. All experiments will be validated using 10 Fold validation to ensure an accurate test score for comparison of methods.

Section 2 outlines the methodology of the traditional dropout method and goes into detail on the various methods of selective dropout used for the experiments in this paper. Section 3 outlines the MNIST dataset used in this experiments, and Sections 4 & 5 outline the experiments results and conclusions.

2 Dropout Methodology

2.1 Traditional Dropout

The Traditional dropout method [1] is a method used to reduce the error rate of artificial neural networks, working by reducing the amount of overfitting that happens in a network and also helping to prevent the network settling in local minima. The dropout method does this by selecting a proportion of random neurons from the input layer through to the last hidden layer, and effectively switching them off for a training epoch. At the end of an epoch, the neurons are then switched back on and another set of random neurons are switched off. This switching off of neurons causes the remaining neurons that may not have learnt anything useful, to try and learn new features.

Neuron outputs are scaled for the use of the network after training, relative to the amount of neurons being switched off in that particular layer. This prevents issues occurring where neurons are overloaded with more inputs than expected. e.g. A network has 50 percent of neurons switched off on each layer during training, however after training all neurons are switched on. Neurons on a receiving layer would get double the expected input if the outputs are not scaled down by 50 percent.

If $\mathbf{x}^{(k)}$ are the node outputs at layer k, and p_k is the proportion of nodes retained during dropout for layer k, the node output during prediction is as shown in Equation 1 [1]:

$$\mathbf{x}^{(k)} = sigmoid(p_k \mathbf{W}^{(k)} \mathbf{x}^{(k-1)} + \mathbf{b}) \tag{1}$$

In section 4 the results of traditional dropout are represented by **TD**

2.2 Selective Dropout

Selective Dropout is different from traditional dropout in that it attempts to choose the neurons which it switches off by giving a higher drop probability to a neuron. By increasing the chances of switching off a selective set of neurons we could improve the effectiveness of the traditional dropout method by either better test results or faster network convergence. There are multiple ways in which a neuron can be selected, and this paper looks at selecting neurons based on the average change in weights between training iterations, the neurons average weight size, and the variance of a neurons output during a training iteration.

With the statistic calculated from each method, we can use a layers drop probability to calculate an individual drop probability for each neuron in the layer while keeping the mean of all probabilities assigned to those neurons the same as the drop percentage for each layer. The below formula (Equation 2) comes from the demands of having a probability that is not only proportionate to the calculated statistics, but also gives an average that is equal to the drop probability given to the network.

$$\mathbf{P} = C\mathbf{S}^{\alpha} \tag{2}$$

Equation 2 shows how this is calculated, where $\mathbf{P}$ is the calculated probability for all the neurons on a given layer, and $\mathbf{S}$ is the given statistics we are scaling probabilities by (Such as $\mathbf{avg_k}$ in equations 6 and 7, or $\mathbf{N\_Variance_k}$ in equation 8). C is a constant that allows us to calculate a probability that is proportionate to our statistics ($\mathbf{S}$) and the drop rate given to the network, shown by equation 3 below. This allows us to meet our demands that the calculated probability is proportionate to the statistics as well as the given drop rate. N is used in the equation to ensure that the mean of all the probabilities generated is equal to the layers drop probability specified in the network parameters, such that $N = C \sum \mathbf{S}^{\alpha}$.

$$C = \frac{N}{\sum \mathbf{S}^{\alpha}} \tag{3}$$

In equation 3, α is a constant between 0 and 1 chosen such that $Cmax(\mathbf{S}^{\alpha}) \leq 1$. This is chosen in numerical decrements starting at 1 until a solution is found. α is used in the equation to ensure that the result for a neurons probability is always less than or equal to 1, this also allows us to work with larger statistics values. It is also worth noting that an α of 0 would result in the traditional

dropout method. N is the number of neurons we expect to drop on any given layer, based on the size of the layer and the drop rate provided to the network. Equation 4 below shows how N is calculated, with p_k representing the neuron retention rate given to the layer (k) being calculated, such as the retention rate of 0.6 used for the hidden layers in our experiments.

$$N = size(\mathbf{S}) \times (1 - p_k) \tag{4}$$

Equation 2 will give a higher probability of being dropped to statistics with a high value. However in some of our experiments we need to give a higher probability to statistics with a small value. We can accomplish this with the following equation (Equation 5). In the equation 1.1 is used to ensure that no statistical values equal zero, as it is preferable that all neurons have some probability of being switched off.

$$\mathbf{S} = 1.1 \times max(\mathbf{S}) - \mathbf{S}; \tag{5}$$

The methods described below can only be applied to the neurons in the hidden layers of a network and cannot be applied to neurons in the input layer, as the input layer either has no incoming weights or the input layer output is the image being inputted, of which the network has no control over. Due to this we use the traditional dropout method on the input layer to select random neurons, and then the remaining hidden layers are run with our selective method to choose selective neurons.

Average Weight Change The method of Average Weight Change looks at the average of the differences between a neurons weights before and after a training iteration. The motivation of this method is that we can tell whether a neuron is still learning by the amount of change in its weights between epochs/batches and by how active the learning process is by the scale of this change.

The equation below (Equation 6) shows the method of finding the average weight change for each neuron where $\mathbf{avg}_k$ is the nodes input weight change average at layer k, and $\mathbf{W}_{jk}^{(i)}$ is the weight matrix feeding into layer k at the beginning of iteration i (Before training). n represents the number of weights feeding into a neuron from the nodes in the previous layer.

$$\mathbf{avg}_k = \frac{1}{n} \sum_{j=1}^{n} (|\mathbf{W}_{jk}^{(i)} - \mathbf{W}_{jk}^{(i-1)}|) \tag{6}$$

For neurons switched off during a dropout iteration there should be no change in their weights between iterations, which would cause the average change to be 0. In some selections as detailed below this would cause the neurons to either always be given a high probability of being selected to be switched off, or could cause the dropped neurons to alternate between a high and low probability of being switched off. To overcome this, the average weight change for neurons switched off are replaced with the average from the last time they were switched

on. To facilitate this the first iteration of training is performed without any dropout to allow a base average for each neuron to be obtained.

When the vector of the average change in weights for each neuron (as calculated in equation 6) has been obtained it can then be used to find the neurons with either the highest change in weights or the lowest change in weights, before then using algorithm 2 to assign a drop probability to these neurons. This allows us to probabilistically switch off a percentage of neurons which have either had a large change in weights or a small change in weights.

Giving a higher probability to neurons with the smallest average change in weights allows us to increase their probability of being switched off, and is much like sorting our statistic in Ascending order and giving high probabilities to the first x neurons in the vector. This could be considered as neurons that have finished learning as their weights are changing the least. In section 4 the results of average weight change dropout (dropping smallest average change) are represented by **WA**.

Giving a higher probability to neurons with a high average change in weights (as if sorting the statistics in Descending order), will allow us to increase the chances of neurons with a high change in weights being switched off. This could be considered as increasing the chances of switching off the neurons that are learning, in order to force those that are not learning to learn. In section 4 the results of average weight change dropout (dropping highest average change) are represented by **WD**.

Average Neuron Weight This method looks at the average weight for each neuron in a layer. The motivation behind this method is that neurons or weights that have learnt something tend to increase in size and start out very small or close to zero when initialised, allowing us to choose neurons that have or haven't done much learning.

To find the average weight for each neuron the following equation (Equation 7) is used where $\mathbf{avg}_k$ is the average input weight of a neuron on layer k and $\mathbf{W}_{jk}^{(i)}$ is the weight matrix for the current iteration i before training commences.

$$\mathbf{avg}_k = \frac{1}{n}\sum_{j=1}^{n}(|\mathbf{W}_{jk}^{(i)}|) \tag{7}$$

As with the previous method, the average of the weights can be used to give a drop probability to each neuron relative to the layer drop rate using equation 2. As dropout typically already replaces the weights of switched off neurons at the end of an iteration there is no need to keep a base representation for neurons switched off.

By giving the neurons with high average weight values higher drop probabilities (much like sorting the statistics in descending order) we can increase the chances of switching off neurons with high weights attached to them. The idea of this method is that weights are initialised close to zero and that weights that are closer zero will have learnt less and should be kept on to learn while the

neurons that have learnt are switched off. Another motivation for this method is regularisation, typical regularisation methods penalise large weights in some way, and this method also does so by preventing neurons with very large weights from increasing. In section 4 the results of average weight dropout (dropping highest average weight) are represented by **AD**.

On the other hand giving a high drop probability to neurons with low average weights (much like sorting the statistic in ascending order) allows us to increase the chances of switching off neurons where the average weight is of a small value. We expect that this could cause neurons with small initial weights to be given a high drop probability for a majority of the training, causing them to spend a majority of training time switched off and rarely being switched back on, in turn leaving the same neurons learning a majority of the time. We still run this experiment to give us some comparison of the effect between giving high probabilities to switching of neurons with high or low weight magnitudes. In section 4 the results of average weight dropout (dropping lowest average weight) are represented by **AA**.

Output Variance This method looks at the variance of outputs for a neuron over an iteration. The motivation behind this method is that you may be able to tell how much a neuron has learnt by the difference in the outputs it gives between images, i.e. a neuron that always outputs the same number no matter the input could be considered as not having learnt anything useful.

The variance is found by taking the output matrix from the previous iteration as shown by the equation below (Equation 8) where $\mathbf{N\_Variance}_k$ represents the variance of neurons of layer k and $\mathbf{X}_k^{(i-1)}$ represents the output matrix of layer k for iteration $i-1$. Variance is calculated along the dimension that holds all the outputs for a neuron over an epoch/batch.

$$\mathbf{N\_Variance}_k = variance(\mathbf{X}_k^{(i-1)}) \tag{8}$$

As with the Average Weight Change method, there is a need to keep a baseline for neurons switched off as switched off neurons would produce no output giving a variance of 0. This baseline is facilitated by running the first iteration with no dropout to allow for an output matrix where every neuron has an output. As with all previous methods the variance vector can be used to find neurons with high output variance or low output variance, and apply a probability to them with equation 2.

Giving a high probability to neurons with high output variance (much like ordering the statistic in descending order) will allow us to increase the chance of switching off neurons with high variance. High variance could indicate that the neuron has learnt an interesting feature and no longer needs training and as such can be switched off to allow neurons with a lower variance to learn. In section 4 the results of neuron output variance (dropping highest variance) are represented by **VD**.

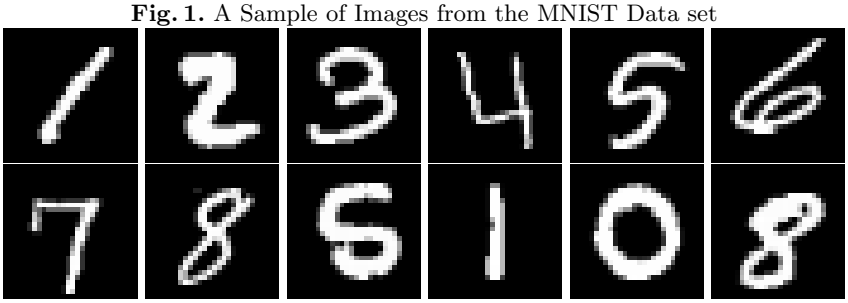
Giving a high probability to neurons with a small output variance (much like sorting the statistic in ascending order) allows us to increase the chance

of switching off neurons with low variance. Low variance may indicate that a neuron is outputting roughly the same value no matter the input and has learnt no interesting features. We expect that switching off these neurons that have not learnt, will cause the network to learn at a very slow rate and possibly be less efficient than the traditional dropout method. In section 4 the results of neuron output variance (dropping lowest variance) are represented by **VA**.

3 Dataset

The dataset used for this experiment was MNIST's handwritten number dataset [2]. MNIST consists of 60,000 training images and 10,000 test images of handwritten numbers in the range of 0-9, each of which 28x28 in size (784 pixels) and has been normalised and centred. We used the 60,000 training image set for training, validation, and testing to allow for 10 Fold validation. Figure 1 shows a small sample of MNIST images.

Fig. 1. A Sample of Images from the MNIST Data set



For our experiment, the MNIST dataset was split into 10 folds for data validation. Each fold contains 54,000 images for Training and Validation (at an approximate 90% and 10% split), and 6,000 images for testing. The Training images are used for the training of the Deep Neural network with the Validation images used to facilitate early stopping. The remaining test set images are solely used for testing the network after training and have no impact on the training process.

4 Experiments and Results

140 Experiments were conducted over a maximum of 10,000 epochs each. Each method ran 20 experiments consisting of 2 network architectures and 10 different folds of data. Each dropout method used the same pre-initialised weights from their respective architectures/folds Restricted Boltzmann Machine. The results of each fold were combined to get an average result for the testing of the method and architecture. Each network was trained with a drop rate of 20% for the input layer (using traditional dropout), and 40% on all remaining hidden layers (using our altered methods). Table 1 shows the results of the experiments for each method under each network architecture.

Table 1. 10 Fold Validated Experiment Results (Test Set), to 4 decimal places

Method	% Error [500,500,500]	% Error [500,500,1000]
TD - Traditional Dropout	4.7300	4.5933
WD - Weight Change (Descending)	7.0533	7.5783
WA - Weight Change (Ascending)	4.3167	4.2517
AD - Average Weight (Descending)	5.2983	5.6533
AA - Average Weight (Ascending)	4.3217	4.1733
VD - Output Variance (Descending)	14.4633	13.7833
VA - Output Variance (Ascending)	3.5167	3.4667

From the results listed above we can see that 3 of the networks did better than the traditional dropout method, these being Weight Change in Ascending order (**WA**) which was 0.378% better on average, Average Weight in Ascending order (**AA**) which was 0.414% better on average, and Output Variance in ascending order (**VA**) which was 1.17% better on average. We can also see that these methods worked for both sizes of architecture tested.

Several of these results were unexpected and the complete opposite of our expectations in Section 2.2. For the Average Weight Change method our experiments performed as expected, with neurons that had little change in weights being given a high probability of being switched off, suggesting that they had finished learning, while leaving neurons that were still learning switched on. As expected, giving a high probability of switching neurons with a large weight change off during training caused the network to be 2.744% worse off on average than the traditional dropout method.

With the Average Weights method, we expected neurons with larger weights to have learnt more, and that they could be given a higher probability of being switched off, allowing neurons with smaller weights to continue learning. However, this did worse than the traditional dropout method, coming out 0.814% worse off on average than the traditional dropout method. The method of switching off neurons with smaller weights did better than this, suggesting that leaving on larger weights gave better results. This could possibly indicate that similar to the weight change method, that it is better to leave neurons that are learning the most switched on.

We also expected that increasing the probability of switching off neurons with high Output Variance to perform well, indicating the neurons had learnt an interesting feature and could be switched off. However, this gave the worst results of all the experiments giving an average of being 9.462% worse than the traditional dropout method. Giving a high probability to neurons with low variance did much better than all the other experiments, suggesting leaving the neurons with high variance switched on is the best method. One possible explanation for this is that the worst neurons that were hindering the network were switched off for a majority of training, leaving the better neurons to continue to learn and refine their weights. It is also a possibility that the method could be acting as a form of network pruning [10] or sparsing [11], causing the network to prune or

sparse particular neurons for the majority or all of the training, consistent with what can be seen in biology.

5 Conclusion

To conclude we have found that the three methods we have presented can improve the effectiveness of the dropout method in deep neural networks by probabilistically dropping neurons with smaller values in the statistics given (Equations 6, 7, and 8). The best of these given methods was shown to be giving higher drop probability to neurons with a low output variance, yielding an average improvement of 1.17% over normal dropout.

Dropping neurons based on the smallest change in weights between iterations, and a smaller average weight size, also yielded improved results of 0.378% and 0.414% on average. Future work could explore the effects of the networks architecture size on these methods, as well as explore other new statistics that could be used to select neurons to be switched off.

References

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

[2] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11) (1998) 2278–2324

[3] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

[4] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) (2015)

[5] Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)

[6] Ba, J., Frey, B.: Adaptive dropout for training deep neural networks. In: Advances in Neural Information Processing Systems. (2013) 3084–3092

[7] Duyck, J., Lee, M.H., Lei, E.: Modified dropout for training neural network

[8] Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In: Proceedings of the 30th International Conference on Machine Learning (ICML-13). (2013) 1058–1066

[9] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7) (July 2006) 1527–1554

[10] Karnin, E.D.: A simple procedure for pruning back-propagation trained neural networks. Neural Networks, IEEE Transactions on **1**(2) (1990) 239–242

[11] Liu, B., Wang, M., Foroosh, H., Tappen, M., Pensky, M.: Sparse convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 806–814