Using Dropout to Reduce Overfitting in Neural Networks

Joshua Paul Chapman
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
chapm250@morris.umn.edu

ABSTRACT

Perfecting unsupervised learning is a major goal in artificial intelligence. Deep neural networks using unsupervised training are a promising answer to this problem; although they are prone to overfitting, which limits their effectiveness. Methods such as validation, averaging, and L1 and L2 regularization have been used to reduce overfitting. While these methods have been successful in reducing overfitting they often take up enough resources that it becomes preferable to not use a neural network. Dropout is a technique that handles overfitting while taking up fewer resources than other methods, and generalizes to all types of neural networks. Maxout utilizes dropout and improves performance of neural networks even more by leveraging dropouts effects. This paper describes both dropout and maxout, as well as results of using each approach.

1. INTRODUCTION

The main topic of this paper is the introduction of two methods that improve the performance of deep neural networks by reducing overfitting, and approximately averaging many neural networks. Before dropout, very deep neural networks were often not thought to be a practical machine learning algorithm because they were prone to overfitting, which caused significant accuracy loss [6]. Dropout is a major improvement to deep neural networks because it is not only much better at reducing overfitting than other methods, it also generalizes to be effective with all neural network algorithms [8]. When it was tested, it improved performance in deep neural network with a variety of different data and architectures. Dropout's generality and universal improvement for deep neural networks has made it extremely popular. For this paper I will be focusing on a specific type of neural network algorithm called the restricted Boltzmann machine because it seems to have been subjected to the most testing. I will also introduce another neural network improvement that is built on top of dropout called maxout.

UMM CSci Senior Seminar Conference, May 2016 Morris, MN.

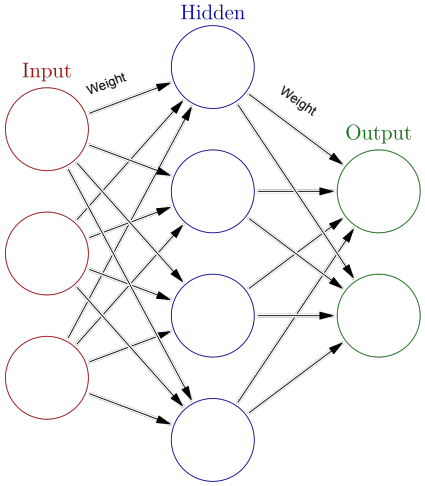

Figure 1: Shallow neural network

2. BACKGROUND

2.1 Deep Neural Networks

Deep neural networks are a special kind of neural networks so I will describe some properties of regular neural networks. Neural networks are very effective at finding complex patterns in data. This makes them very powerful at solving problems that humans are good at, which historically computers have not been good at, such as image recognition, speech recognition, translating, and market trends. What makes these problems so hard for computers is that they have very complex and dynamic patterns that cannot simply be programmed in by humans. For instance no two pictures of yourself are exactly the same, there is always something slightly different such as background, lighting, or position. Neural networks work better than regular algorithms for these problems because they can generalize to find these patterns in noisy data [12].

A *neural network* is a set of n nodes called *artificial neurons* which I will just refer to as neurons. The first layer is called the input which is composed of n neurons, each of which is connected to m neurons in the second layer called the *hidden layer*, each of which are connected to the output layer. Each edge between the neurons carries a weight. A visual representation is shown in figure 1 *Deep neural networks* work the same as neural networks, except there are multiple hidden layers. Multiple hidden layers allows them

to find more complex patterns in data. In order to distinguish between neural networks with one hidden layer and deep neural networks I will refer to neural networks with one hidden layer as *shallow neural networks*. I will use just the term "neural network" when the distinction between deep and shallow networks is irrelevant.

2.1.1 Activation Functions

In order for the inputs from a previous layer to be translated into an output, a neural network needs an activation function. *Activation functions* are equations that take in a set of inputs from a layer, and represent them as a single output value. Only neurons in the hidden layer and output layer have activation functions. The first activation function was the step function. It was relatively simple activation function that followed this formula

$$Output = \begin{cases} 0 \text{ if } & \sum_i x_i w_i <= thresholdValue \\ 1 \text{ if } & \sum_i x_i w_i > thresholdValue \end{cases}$$

where x is the input, w is the weight between the input and the neuron, and i is the range over input nodes. Step functions do not scale well to detecting more complex patterns because outputting 0s and 1s does not express much data. In order to create neural networks that could more accurately label complex patterns, the sigmoid function replaced the step function [1]. The sigmoid function can label more complex patterns because its activation function goes from the range zero to one. Sigmoid functions are also preferred over other activation functions because their derivatives are easy to calculate which makes it faster to train neural networks with more advanced training techniques. An example of the sigmoid function is

$$\text{Total} = \left(\sum_{i=0}^{inputs} w_i x_i \right) \tag{1}$$

$$\text{Output} = \text{SigmoidFunction(Total)} = \frac{1}{1 + e^{-Total}} \tag{2}$$

More recently rectified linear neurons, also called *ReLU*, have been replacing the sigmoid neurons. The ReLU neuron is much faster to compute because the activation function is simply max(0, x) where x is the net input to the hidden neuron [3].

2.1.2 Training

Before a neural network can be useful, it first must be trained. I will walk through how training works for static image recognition, although neural networks can be trained to recognize any pattern given proper data representation. There are many different ways to train a neural network, but all neural networks require some sort of training.

Initially all the weights in a neural network are the randomly assigned. There are many different ways to decide how to distribute these weights but the specifics are not relevant to this paper. In order to train a neural network to recognize static images, it must be handed many photos either labeled or unlabeled. The industry standard is to use 20% of the data for training and the rest for testing. In image recognition neural networks recognize relationships between pixels. For instance if you wanted to train it to tag a dog in a picture you would randomly hand it thousands or millions of pictures of dogs, and pictures without dogs. For the first few pictures it will map all the pixels of the pictures onto the neural network, assigning weights to groups of pixels that are similar in structure. Since it has processed few pictures so far, it might under-fit the data. Which is recognizing similarities in the picture that have nothing to do with dogs but are often in pictures with dogs, such as trees, humans, or grass. This is a reason why it is important to have a diverse set of pictures. If there is anything that is similar in all the pictures besides the essence of a dog it will have no way of knowing what is fundamental to what a dog looks like, or more generally what the pattern of what a dog looks like, as they come in many different sizes, colors, and environments. Eventually, given enough pictures, weights that represent the pixels that represent parts of the dog will be heavily correlated, such as the ears or tails for many different angles. It will be able to detect the *pattern* which represents a dog. A well trained neural network will eventually be able to use these parts together to provide evidence that the image has a dog in it. When handed a picture, the input layer receives groups of pixels from the image which are near each other. Then the hidden layer(s) try to find a correlation between the pixels that represent a dog. If it finds sufficient proof that a dog is in the picture it will be able to tag the cluster of pixels where there might be a dog. If it does not find enough proof that there is a dog in the picture, it will tag it as not having a dog. When the neural network has been fully trained the unique structure is called the *model*.

There are two major categories of neural network learning algorithms, *supervised* and *unsupervised* [9]. Supervised neural networks work by training the neural network using a labeled input set. After a training sample is sent through the supervised network an error, or *cost* is calculated. The cost is the difference between what the neural network thought the training sample was, and what it actually was given the input. Then that cost is *back propagated* through the weights; which adjusts them so that they reduce the cost [7]. There are many ways to calculate how to back propagate, but the specifics are not relevant for this paper. Unsupervised neural networks are not given labels for the input data; instead they find patterns in the data and output groups that have similar patterns. Supervised training does not require as much data, but requires people to label all the data. Unsupervised training needs much more data, and is more prone to find non-relevant patterns, but does not need humans to label data.

2.2 Overfitting in Neural Networks

Overfitting occurs when a model is overly complex, relative to how much data there is for training. The main cause for overfitting is that neurons begin to co-adapt [2]. A simple example of co-adaptation would be if there were two hidden neurons in the same layer, neuron A, and neuron B. If overfitting started to occur the input to neuron A could be [0.4, -0.6, 0.7, 0.1, 0.9], and the input to neuron B could be [0.4, -0.6, -0.7, -0.1, -0.9]. When these neurons output their result to the next layer, only the first two indexes actually convey any information: the rest cancel each other out. The relevant information becomes dependent on the not relevant information being canceled out. This leads to lower cost because neuron A and B are still decreasing the cost by a small net amount. Co-adaptations that actually occur in neural networks are typically much more complex, but the end result is still the same. Despite some techniques that have been developed to reduce overfitting, it has remained

a serious issue with neural networks.

These co-adaptations allow the neural network to "memorize" parts of the training data to decrease cost. This can happen because a training set is ran through a neural network many times. Given a complex neural network and a small enough training set it will stop learning the pattern it is supposed to learn and instead memorize random noise in the data to decrease cost. While this might decrease cost, it does not generalize to test data as the new data will not have the same random noise which will cause the neural network to increase error.

2.3 Methods to Prevent Overfitting

There are a few methods that researchers have developed to prevent overfitting, although they are often computationally expensive, or require performance sacrifices. One way to combat overfitting is to have an excess of diverse data. Unfortunately, this is very costly as you need to have the resources to get more data, and it will take longer to train the data to the neural network.

2.3.1 Regularization

Regularization is the most popular way of preventing overfitting. Regularization works by artificially reducing complexity in a model. Once a model becomes too complex, regularization starts reducing the influence of both noise and the real data by pulling the weights toward zero. This reduces overfitting and allows more complex models, but harms the predicting power of the model as complexity is intentionally crippled. There are two different types of regularization, L1 and L2. L1 regularization works by pulling low importance, or weights that already are not large relative to other weights, toward zero, sometimes reaching zero which effectively removes them. It does this because weights of low importance that develop later on are often focusing on noise. L2 regularization pulls all weights toward zero, but never allows them to get to zero [13]. This works because it is often not that useful that high importance weights be excessively high, as they just need to have a high enough weight to confirm that the pattern is present or not. Low importance weights that are often noise have much more to gain by having their weights increase so reducing their strength reduces their influence much faster than high importance.

2.3.2 Validation

Another popular way of preventing overfitting is validation. *Validation*, also called early stopping, is a method where you section off about 20% of the training data set to test on. If during training the cost decreases, but the accuracy of running the neural network on the validation set stays the same or decreases, then training stops [11]. This method works quite well to prevent overfitting but it requires sacrificing part of the data set which could be used for training. It can also be difficult to find a subset data that truly represents the rest of the data.

2.3.3 Averaging

Averaging the outputs is another way that researchers have tried to prevent overfitting. This is done by breaking up the input sets and training a neural network for each of them. Then instead of taking one output, you average the output probability of all the neural nets. It is simple yet effective. One major downside to this method is that creating multiple neural networks is computationally expensive as large neural networks can take days to fully train. This is why it is not often thought of as a reasonable choice.

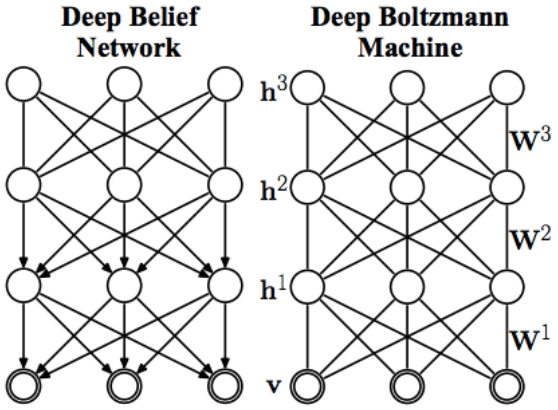

Figure 2: Deep Boltzmann Machine vs Deep Belief Network

2.4 Restricted Boltzmann Machines

For this paper, the main type of neural network I will be talking about is the Restricted Boltzmann Machine. *Restricted Boltzmann Machines* (RBMs), can be used to create deep neural networks. RBMs most often are used to create powerful unsupervised neural networks. A RBM is composed of a hidden layer and a visible layer where each visible neuron has an undirected connection to all hidden neurons. RBMs are used to increase the relationship between neurons. If it seems clear given the input that a visible neuron has a positive relationship with another visible neuron, it will transform the visible neuron to make it more probable that the hidden neuron that they share will activate. In order to make a deep neural network from an RBM the outputs from one RBM become the inputs to the next. A deep neural network that is formed only using RBMs is called a *deep Boltzmann machine* or DBM. A three layer deep Boltzmann machine is shown on the right in figure 2 here h represents hidden layers, v is the input layer, and the W represents weights. Every layer is a RBM. There is also the *deep belief network*, or DBN shown on the left in figure 2 which is similar except only the output and last hidden layer are restricted Boltzmann machines, and the rest of the layers operate like a normal neural network [5]. This is done so that the neural network can learn the patterns of its inputs with RBMs using unlabeled data. Then once most of the training data has gone through, a small amount of labeled training data can be presented so that the neural network part of the DBN can learn what the data it has categorized is named. The amount of labeled data needed to create an accurate model is much smaller than a regular neural network would need. In the real world this is a huge advantage because most of the data in the world is not labeled.

3. DROPOUT

3.1 Dropout Deep Neural Networks

Dropout is a technique introduced to prevent the problems of overfitting in deep neural networks, and average many models quickly [8]. Training a neural network with dropout is similar to usual training. The main difference in training a neural network with dropout is that there is a 0.5 probability that any neuron's weight will be set to 0. It has been found that the optimal probability to drop units is about 0.5 for hidden units, and 0.2 for visible units, although these numbers are somewhat arbitrary. Setting the weight to 0 means that that neuron will not be activated and can be thought of as dropped out. One large advantage to this is that you are essentially randomly sampling from 2^h different architectures where h is the amount of hidden neurons, and each architecture has the same weights. It is important that the architectures have the same weights because it has the effect of L2 regularization [10]. Unlike the L2 regularization I described earlier which works by pulling all weights toward 0, when a model shows signs of overfitting, dropout adjusts the weights toward the correct value. Although a caveat to dropping neurons is that training takes two to three times longer than without [8].

There is one more step to complete the dropout method, averaging the different architectures. This step is taken after training is complete. Averaging is done by dividing all the outgoing weights by $1/(1-p)$, where p is the probability of dropping a unit. Since 0.5 is the standard, you would halve all outgoing weights. It has been mathematically proven that this method will perform averaging while training on shallow neural networks. It cannot be proven mathematically that it scales to deep neural networks, but in practice it approximately does. Dropout does this much faster than previous methods, which required building many models from scratch with different training sets. Figure 4 depicts how a variety of neural network algorithms work with regularization and with dropout. When the lines start diverging is when overfitting starts to occur. Regularization minimizes damage done but does not improve much on the model. Dropout not only reduces overfitting but allows progress to be made.

While dropout works well to prevent overfitting there are some cases where it can increase error rate. To test this, dropout neural networks were compared to regular neural networks on data set sizes of 100, 500, 1000, 5000, 10000 and 50000 with the only difference being whether or not dropout was applied [8]. The results are displayed in figure 3. For very small data sets relative to the amount of parameters dropout increases error rate because there is so little data that overfitting happen regardless of dropout. This effect disappears as the data set gets large enough for dropout to become effective. It stops decreasing error once the data set gets to about 10000 because the data set got large enough overfitting did not occur. So there is a certain range of data that dropout is effective: not so small that overfitting occurs regardless of dropout, and not so large that overfitting is not an issue. One could argue that these are not major issues. If there is such a small data set that overfitting occurs despite dropout, it is a sign that there is not enough data for proper representation of the pattern; which is an issue that only more data can solve. If there is so much data relative to the size of the network that overfitting does not occur regardless

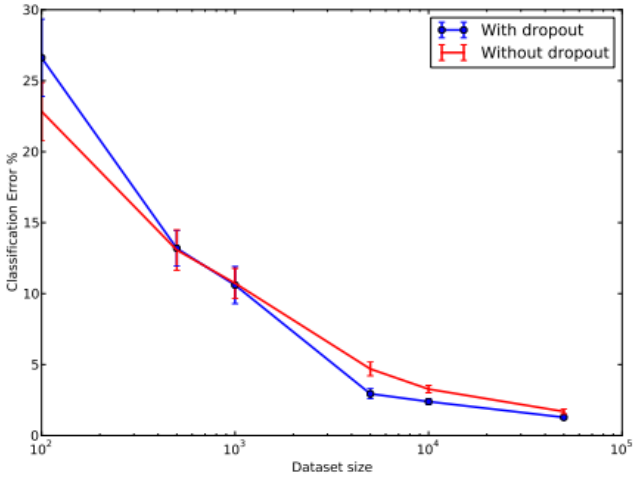

Figure 3: Dropout vs not Dropout

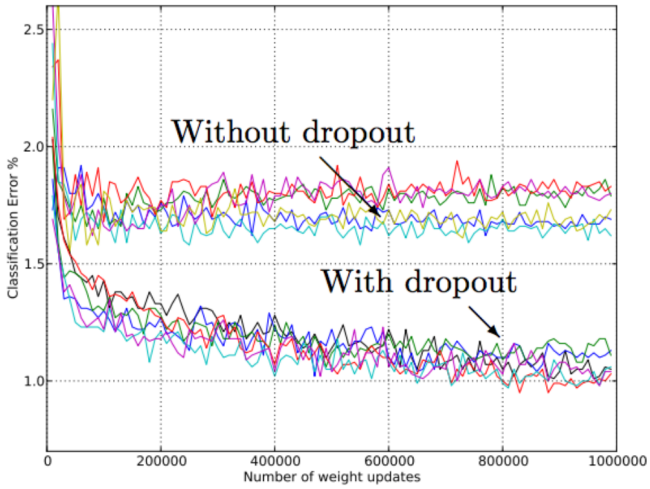

Figure 4: Dropout vs Regularization

if dropout is used or not, then the model would become more accurate if the size of the network were increased and dropout used.

3.2 Maxout Deep Neural Networks

Dropout is a powerful tool that can improve almost any deep neural network when applied, but activation functions have been optimized for older regularization techniques. Older regularization techniques were improved by making each neuron less influential so that overfitting could be detected faster. This allowed regularization to reduce the influence of noise as soon as it appeared, without reducing the influence of meaningful patterns. In contrast dropout starts preventing overfitting immediately instead of waiting for it to appear. Part of the reason that dropout works so well is because it makes each neuron more independently useful. Since dropout implements immediately, and makes neurons more independently useful, its performance is hindered by activation functions that are optimized for older regularization.

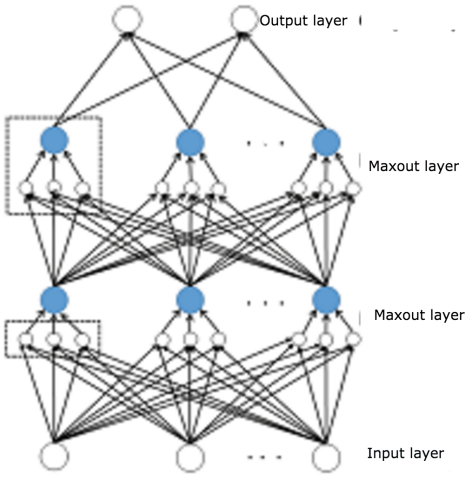

Figure 5: maxout

Maxout is a technique that solves this problem, and makes it so that the approximate model averaging that occurs with applying dropout to deep neural networks is closer to the actual model average [4]. It does this by introducing a different type of activation function: the maxout neuron. The maxout neuron is composed of one neuron that contains the activation function, called the *maxout operator*, and k neurons that receive outputs from the previous layer, and only outputs to the maxout operator. Because the k neurons only output to one neuron, and do not contain an activation function they are called *linear neurons*. An example of a maxout neuron is shown in the top rectangle in figure 5. The larger colored neuron is the maxout operator, and the small white neurons are the linear neurons. A hidden layer with maxout neurons is called a *maxout layer*.

By doing this a deep neural network acts more like a shallow neural network which means that when averaging with dropout the approximate averaging is going to be closer to actual averaging. Dropout training also works differently with a maxout deep neural network; instead of all neurons having the same probability of being dropped, the weights inside the maxout neuron are never dropped. For example the weights between the maxout operator and the linear neurons in figure 5 are never zero. The maxout neurons activation function is $h_i(x) = \max_{j\in[1,k]}(z_{ij})$, $z_{ij} = x^T W_{...ij}$ where h_i is the maxout neuron, x is the input from the previous layer, W is the weights, j represents the linear neurons specific to a maxout operator, i represents the specific maxout operator, and T means transpose. $z_{ij} = x^T W_{...ij}$ translates to calculate the weighted sums of the inputs for a given linear unit, and $\max_{j\in[1,k]}(z_{ij})$ means find the maximum value of all the linear units for a given maxout operator. [4]. Unlike the sigmoid function, the maxout neuron does not have an upper limit. This has the effect that neurons increase their influence much faster. This is not that different from ReLUs, the only difference being that maxout neurons do not have a lower bound 0. In fact a ReLU can be created using a maxout neuron with two linear neurons, one of which always outputs 0 to the maxout operator. ReLUs

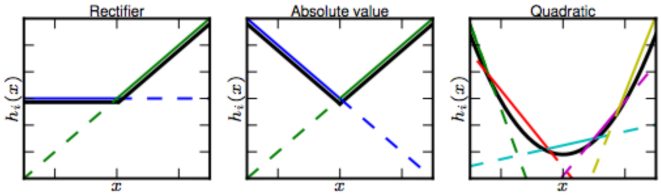

Figure 6: maxout activation functions

are not the only activation function that maxout neurons can create. It has been mathematically proven that maxout neurons can approximate any activation function given enough linear neurons [4]. A visual example is shown in figure 6. The dashed lines represent the linear neurons approximation, and the solid lines represent the true values. A ReLU and absolute value function can be created with two linear neurons, while the quadratic function needs five to be a good approximation. These activation functions are not hard coded in for each neuron but are learned during training because the weights between the linear neurons and the maxout operator change with training.

4. RESULTS

One of dropouts strengths is that it can improve almost any deep neural network. In order to prove this the researchers applied it to many different neural networks and compared the results to neural networks not using dropout. Table 1 is a description of the data used for comparison [8].

Table 1: Data labels

Data Set	Domain	Dimensionality	Training set	Test set %
MNIST	Vision	785 (28 x 28 greyscale)	60K	10K
TIMIT	Speech	2520 (120-dim, 21 frames)	1.1M frames	58K frames

MNIST is a bench marking image recognition test data set that is used with most new neural network techniques to make sure there is a fair comparison. As shown in all test cases; dropout beat all other equivalent method that did not use dropout, and by quite a significant margin. For this test the researchers also used another regularization technique that works well for dropout but is not relevant for this paper. When [4] was published this was the best score anyone had gotten with the MNIST data set. The method used was a deep Boltzmann machine using dropout with a logistic unit it achieved an error of 0.79. When the same architecture was used without dropout it had an error rate of 0.96. Similar results were achieved when comparing a regular neural network and a dropout neural network: without dropout it achieved an error rate of 1.60, with dropout error was reduced to 1.35.

This might not seem like a fair comparison because the dropout neural network had 3 layers and 1024 neurons, while the neural network had 2 layers with 800 neurons; but if the neural network had as many layers and neurons it would have became worse due to overfitting. So because of dropout neural networks are allowed to have more layers, and neurons without having to worry as much about overfitting. It is also important to see that the best regular neural network method was using Maxout with a slightly larger error of 0.94. Although it is hard to compare networks because of

the maxout neurons unique structure, we can compare the total number of neurons to get a rough estimate of how computationally expensive each method was. It is specified in the architecture column that maxout had two layers and (5x240) units. This means that there was five linear neurons inside each maxout neuron, and 240 maxout neurons. So we can calculate how many neurons there were total by doing 5*240 which comes out to 1200 total neurons. The closest method to that amount of neurons used ReLUs and received an error rate of 1.06.

Table 2: MNIST

Method	Unit Type	Architecture	Error %
Neural Network	Logistic	2 layers, 800 units	1.60
Dropout NN	Logistic	3 layers 1024 units	1.35
Dropout NN	ReLU	3 layers 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	3 layers 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	3 layers 8192 units	0.95
Dropout NN + max-norm constraint	Maxout	2 layers (5 x 240) units	0.94
DBN	Logistic	500-500-2000	1.18
DBM	Logistic	500-500-2000	0.96
Dropout DBN	Logistic	500-500-2000	0.92
Dropout DBM	Logistic	500-500-2000	0.79

In order to show that it works on different types of data they also looked at speech recognition. This data set called TIMIT is a recording of 680 speakers from 8 major dialects of American English reading sentences in a noise-free environment. For the regular neural network dropout decreased error rate by 1.6%. When using deep belief networks accuracy increased by 3%.

Table 3: TIMIT

Method	Phone Error Rate%
Neural Network (6 layers)	23.4
Dropout NN (6 layers)	21.8
DBN (4 layers)	22.7
DBN (6 layers)	22.4
DBN (8 layers)	20.7
Dropout DBN (4 layers)	19.7
Dropout DBN (8 layers)	19.7

5. CONCLUSIONS

Overfitting has prevented deep neural networks from being applied to many problems which they could otherwise have had great success. Dropout is an effective tool that reduces overfitting without having to hinder complexity. It does take two to three times longer to train a model, but for many problems the extra computing power is worth not having to find more training samples. Both dropout and maxout are major innovations in the field of neural networks and greatly expanded the amount of problems that neural networks can solve. One of the major benefits is that dropout and maxout have made unsupervised training much more viable, which means that we can apply neural networks to problems without the large amount of people power needed to label data; which was a major cost for training neural networks.

Acknowledgments

Thanks to KK Lamberty, Elena Machkasova, Snuffy Linder, and Andrew Latterner for their help and feedback.

6. REFERENCES

[1] Improve Neural Network Generalization and Avoid Overfitting. http://www.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html, 2016.

[2] P. Baldi and P. J. Sadowski. Understanding dropout. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. Curran Associates, Inc., 2013.

[3] G. Dahl, T. Sainath, and G. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613, May 2013.

[4] M. C. B. Goodfellow, Warde-Farley. Maxout networks. pages 1319–1327, 2013.

[5] G. Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[6] S. Lawrence, C. L. Giles, and A. C. Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *In Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, pages 540–545. AAAI Press, 1997.

[7] M. A. Nielsen. *neuralnetworksanddeeplearning*. Manning Publications Co., 2015.

[8] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[9] Stergiou, Siganos. Neural Networks. https://www.doc.ic.ac.uk/ nd/surprise-96/journal/vol4/cs11/report.html, 2016.

[10] L. Wager, Wang. Dropout Training as Adaptive Regularization. *ArXiv e-prints*, July 2013.

[11] Wikipedia. Cross-validation — Wikipedia, The Free Encyclopedia, 2016.

[12] Wikipedia. Deep learning — Wikipedia, The Free Encyclopedia, 2016.

[13] Wikipedia. Regularization — Wikipedia, The Free Encyclopedia, 2016.