5-2014

A comparison of dropout and weight decay for regularizing deep neural networks

Thomas Grant Slatton
University of Arkansas, Fayetteville

This thesis is approved.

Thesis Advisor:

Michael Gashler, Ph.D.

Thesis Committee:

Gordon Beavers, Ph.D.

Craig Thompson, Ph.D.

A Comparison of Dropout and Weight Decay for Regularizing Deep Neural Networks

An Undergraduate Honors College Thesis

in the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

by

Thomas Grant Slatton

April 23, 2014

Abstract

In recent years, deep neural networks have become the state-of-the art in many machine learning domains. Despite many advances, these networks are still extremely prone to overfit. In neural networks, a main cause of overfit is coadaptation of neurons which allows noise in the data to be interpreted as meaningful features. Dropout is a technique to mitigate coadaptation of neurons, and thus stymie overfit. In this paper, we present data that suggests dropout is not always universally applicable. In particular, we show that dropout is useful when the ratio of network complexity to training data is very high, otherwise traditional weight decay is more effective.

1 Motivation

1.1 Background

An *artificial neural network* (ANN) is a model of learning in which computations are performed by a set of interconnected *artificial neurons*. An artificial neuron is a polyadic function which takes real numbers as inputs, produces a weighted sum of the inputs, offsets that sum by a bias value, and outputs this biased value transformed by an *activation function*. Generally, a sigmoidal function is used as the activation function.

An ANN can be described as a weighted graph with a set of *input nodes*, *hidden nodes*, and *output nodes*. To produce a computation, the input nodes are activated according to the model input vector, then their descendants are activated (receiving inputs from their ancestors), etc. until the output nodes have been activated. The values of the output nodes are the model's output vector.

The type of ANN used in this paper is a *feedforward* neural network. That is, an ANN that contains no cycles. Specifically, we are interested in *multi-layer perceptrons* (MLP). An MLP is an ANN architecture which is formed by creating layers of nodes that form complete bipartite graphs with adjacent layers. Thus, an MLP consists of an input layer, any number of hidden layers, and an output layer.

1.2 Usage

ANNs gained widespread adoption in the field of artificial intelligence with the development of the *backpropagation* algorithm in the 1970s and 1980s. In the 1990s, simpler methods such as *support vector machines* (SVM) regularly outperformed ANNs due to the computational problem of training large enough ANNs to solve real world tasks.

In the last decade, ANNs have made a resurgence in the field due to the success of deep learning methods such as *deep neural networks* (DNN)[2] and *convolutional neural networks* (CNN)[4]. This recent success was largely due to an increase in computing power, GPU computing, and innovative training techniques.

At present, ANNs are the state-of-the-art in exercises such as handwriting recognition, image classification, and speech recognition. They have the potential to outperform leading systems and human experts at tasks such as medical diagnosis, financial predictions, and information retrieval.

1.3 Problems

Two key problems have always hindered ANNs: lack of computational power and *overfit*. Modern hardware implementations, such as multicore processors and GPUs, have helped alleviate the first problem. The latter, however, is more subtle and difficult to fix.

Overfitting is a state in which a model has learned its training data effectively, but it poorly generalizes beyond its training data. For example, in Figure 1, the data is slightly noisy, but generally linear. We present a linear fit, as well as a 6 degree polynomial fit. It can be seen that the 6 degree polynomial fits every data point, but fails to describe the overall linearity of the data.

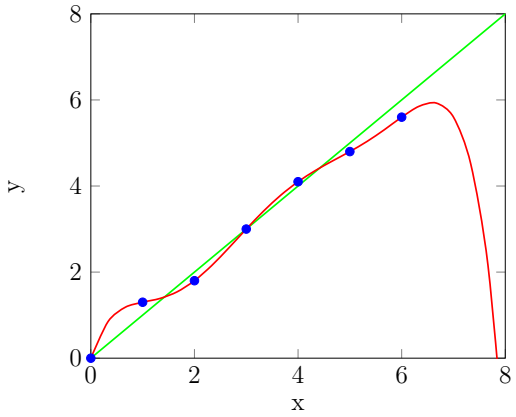

Figure 1: A typical example of overfit: The polynomial function matches every data point, but fails to capture the general linearity of the data.

The problem of overfit is common in ANNs because complex networks are required to perform interesting and meaningful computations, but added complexity in a network introduces the capacity to fit the function to noise, rather than general features.

The simplest way to avoid overfit is merely to gather more training data. By learning from a larger set of data, the likelihood of multiple data points all sharing the same anomalous aspect is reduced significantly, along with the likelihood of accidentally learning such an anomalous aspect as a general feature.

Unfortunately, in many cases, it is impossible or prohibitively expensive to gather more training data. In this event, there exist *regularization* methods which have experimentally been shown to reduce overfit. Regularization refers to altering the network performance function in some way to improve generalization. Typically, the altered performance function biases the network towards a simpler model. In accordance with Occam's Razor, it is often found that a simpler model provides a more general solution. In this paper, we compare two of these methods: *weight decay* and *dropout*.

1.4 Reasoning

Weight decay is a well-established regularization technique for ANNs. It has been throughly tested, expanded upon, and utilized. Dropout has only recently been reported on, and, to our knowledge, its limitations are not well explored.

While dropout has shown great success in both speech and image recognition [1], it is important for the research community to be aware of any possible pitfalls of new methods. In this work, we seek to clarify conditions in which a more traditional regularization scheme should be employed, so that ANNs may be used effectively on a diversity of problems.

2 Intuitive Overview

2.1 Weight Decay

Weight decay is a simple regularization method that works by scaling weights down in proportion to their current size (i.e. exponentially). To accomplish this, the error function can be modified to

$$E(w) = E_0(w) + \frac{1}{2}\lambda \sum_i w_i^2,$$

where E_0 is the original error function, and λ is a decay rate. It can be shown that this is equivalent to adding a term to the weight update function as in

$$\dot{w}_i \propto -\frac{\partial E_0}{\partial w_i} - \lambda w_i.$$

That is, during each weight update, the weight is adjusted according to the error function, but also scaled down in proportion to its size.

This helps regularize the network in two ways. Most obviously, it can be quickly seen that by decaying all weights, any weights that are not being used by the network to produce meaningful output will rapidly become negligible, rather than persisting in the network for no reason. While such errant weights may not affect training performance, they may reduce generalization during test time.

A less evident benefit of weight decay comes from recognizing that large weights introduce large jumps in the model function. Intuitively, smaller weights should produce a more subtle fluctuations and a smoother (and more general) function.

2.2 Dropout

Dropout is a more recent method, developed by Hinton *et al.* [2] that involves choosing a probability p (commonly $p = 0.5$), and randomly deactivating hidden nodes with probability p during training time. Figures 2 and 3 illustrate this process.

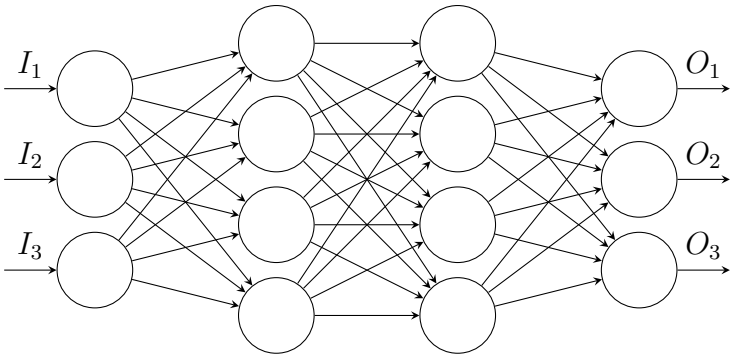

Figure 2: A typical feedforward ANN.

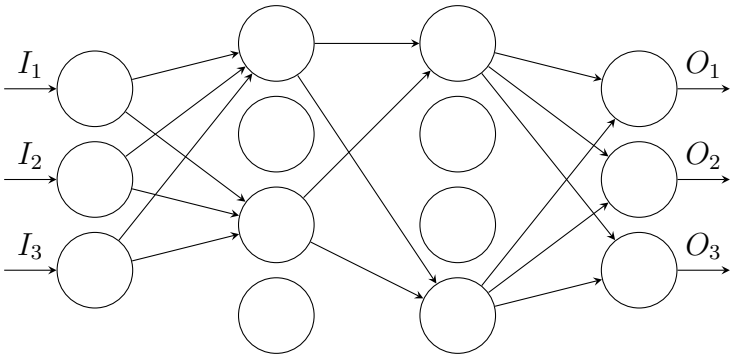

Figure 3: The same ANN with a dropout of $p = 0.5$ applied.

Thus, during each training iteration, the nodes and weights being altered represent a sub-network. The number of sub-networks within the network is exponential in the size of the network.

At test time, no nodes are dropped out, so to compensate for the increase in layer output (caused by all nodes being active, rather than a subset), the weights are all multiplied by $1 - p$. This has the effect of combining all of the $O(2^n)$ overlapping sub-networks that were created during training time.

Dropout helps prevent overfit by not allowing neurons to co-adapt to each other. Since a neuron cannot count on any other neuron to be active during any particular training iteration, the neuron must learn to receive inputs generally, rather than specifically.

More complex dropout schemes can be devised; dropping out hidden and visible units can be effective for some tasks, as well as using different probabilities for different layers. In these experiments, we only consider the most common method of dropping out hidden units with uniform probability.

3 Related Work

The problem of preventing overfit in machine learning models may be as old as machine learning itself. For ANNs in particular, a very early prescription was to carefully craft the

architecture of the network to match the problem and size of the data set [7]. That is, one can work to prevent overfit by designing a network in such a way that there is no room for overfit to occur. This essentially amounts to using as few nodes as possible while still providing adequate results. The reasoning behind this is that more nodes allow for more features to be detected, and detecting more features with not enough data allows noise to be mistaken for features.

A second, common overfit inhibitor is *early stopping*. Early stopping is used because predictive accuracy increases over time with respect to training data, but after a certain point decreases with respect to validation data as the network begins to overfit the training input. This is visualized in Figure 4.

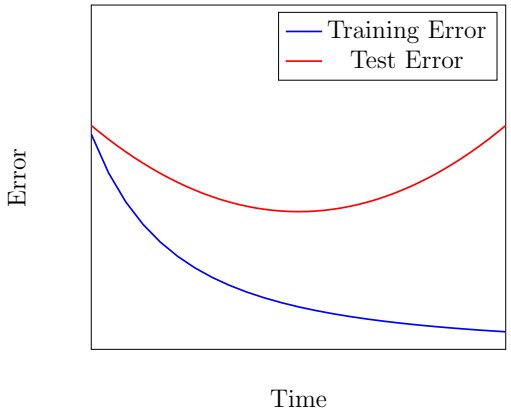

Figure 4: The optimal place to stop the network's training would be at the minimum of the red curve.

Both of these early methods are fairly effective, and do not involve altering the actual operations of the ANN. However, these solutions were surpassed by more robust and powerful techniques.

As described in the previous section, Krogh and Hertz [5] showed in 1992 that simply decaying the ANN's weights in proportion to their size sufficed to improve generalization a great deal by shrinking unneeded signals in the network.

Nowlan and Hinton proved that a simple exponential decay could be beaten by a more complex method [9]. They note that the modified error equation,

$$E(w) = E_0(w) + \frac{1}{2}\lambda \sum_i w_i^2,$$

will reward two weak interactions rather than one strong one because

$$\frac{w^2}{2} + \frac{w^2}{2} < w^2 + 0^2$$

Thus, they propose a significantly more complex formula using a prior which contains a thin and a wide Gaussian centered at zero, and what they refer to as a *responsibility function* which measures the "conditional probability that a particular weight was generated by a

particular Gaussian." Using this function, large weights are not necessarily penalized as much as they are in typical weight decay.

In a similar vein, Larsen and Hansen proposed their own modification to the cost function which adds a term they define as the *generalization error*, thus providing mathematical rigor to the typical notion of overfit. They derive an unbiased average generalization error estimator, while showing that it is not possible to know the exact generalization error without perfect knowledge of the joint input-output distribution[6]. They then use this estimator to optimize the weight decay function.

After various modifications to traditional weight decay had been thoroughly explored in the 1990s, new approaches were developed in the 2000s. Jin *et al.* performed experiments to test a system in which the hyper-parameters of the network were modified in a multi-objective evolutionary optimization function[3]. Using this method, one may simultaneously train several slightly altered models and produce an ensemble model of all of the networks. This method has certain advantages over traditional gradient-descent methods in that additional parameters can be added that do not need to be continuously differentiable. They show that such a method is viable, but do not compare it to existing methods.

The newest method to gain popularity, and a key target of this research, is dropout. Developed by Hinton *et al.* in 2012[1] and thoroughly documented by Srivastava the following year[10], dropout mitigates overfit by preventing complex interactions from forming between groups of neurons. In Srivastava and Hinton's work [10][1], several examples of where dropout improves ANNs are given. They show how it can be applied to typical ANNs, and also *deep belief networks*, also developed by Hinton[2]. However, there is little information provided about cases when dropout is *not* appropriate.

In this work, we seek to examine cases in which dropout is useful, and when a more typical regularization technique, such as weight decay, should be employed.

4 Technical Approach

To test and compare weight decay and dropout (and a control group, using neither), an ANN was implemented in C. This ANN had the following hyper-parameters:

$Learning\ Rate : \eta$

$Momentum : \mu$

$Dropout\ Probability : p$

$Weight\ Decay\ Rate : \lambda$

Since we are not using adaptive learning rates or adaptive momentum terms, a grid search was executed over each parameter. Learning rates were searched in {0.1, 0.01, 0.001}, momentum in {0.0, 0.1,..., 0.9}, and weight decay rate in {0.00001, 0.0001, 0.001, 0.01, 0.1}. Dropout probability remained constant at 0.5, though some researchers have reported success with values such as 0.7 for *extremely* large networks[1]. Srivastava reports roughly equal performance for dropout rates in the interval [0.4, 0.6], with a decline in performance above that[10].

4.1 Weight Decay

Implementing weight decay is straightforward. Where the previous weight-update code matches the formula

$$w_i^{t+1} = w_i^t - \eta \frac{\partial E_0}{\partial w_i} - \mu \Delta w_i^{t-1},$$

we must merely add a term to achieve weight decay:

$$w_i^{t+1} = w_i^t - \eta \frac{\partial E_0}{\partial w_i} - \mu \Delta w_i^{t-1} - \lambda w_i^t.$$

When $\lambda = 0$, this term has no effect, so weight decay can be easily disabled to return to typical behavior.

4.2 Dropout

Dropout can also be added to common ANN implementations without large modifications. It is possible to add any sort of preferred numerical flag to a node, then, for all code involving one or more nodes, simply skip the code if any of the nodes are dropped out. For a typical backpropagation implementation, this means ignoring dropped out nodes during the forward pass, and not updating any weights between two nodes during the backpropagation phase if either node is dropped out. One must also take care not to add dropped out values into the error sum during backpropation.

When training a dropout network, only a subnetwork (with size approximately equal to p times the original size) is trained each iteration. Thus, during test time, when no nodes are being dropped out, weights are multiplied by $1 - p$ to counteract the stronger weights created by training the subnetworks.

It should be noted that, like weight decay, dropout is easily disabled. When $p = 0$, all of the terms involving p are unaffected by it. The $1 - p$ multiplier becomes the identity, $random() < p$ is never true, etc. The performance hit from a small handful of extra *if* statements is entirely negligible in comparison to the backpropagation algorithm.

5 Validation

All experiments performed use the MNIST database[8]. We chose this dataset because it is well known and easily accessible to anyone wishing to perform more tests in this domain.

Due to the large amount of computation time required of the grid search (described in the previous section), the full 60,000 training samples were not used. At most, we used 10,000, and generally 1,000. This is not detrimental to the study because we are trying to compare two methods rather than break records on MNIST.

5.1 Network Complexity to Data Size Ratio

Before narrowing in on specific and detailed experiments, a broad search was performed over several architectures and dataset sizes to lead our search. A property worth investigating

further was the apparent tendency for dropout to perform better than weight decay in situations with high *overfit potential* and vice versa. We define overfit potential (OP) as

$$OP \propto network\ complexity / dataset\ size.$$

That is, increasing the network complexity (using one's favorite measure—typically a combination of number of hidden layers and layer width) or decreasing dataset size increases a system's propensity to overfit.

To test this property, we performed a grid search over a wide spread of systems with varying overfit potential. We present three cases—low, mid, and high OP—which illustrate the consensus of this experiment.

For the low OP case, we use a $784 - 50 - 10$ classifier and 1000 training samples. We see in Figure 5 that dropout performs better than no regularization at all, but is still beaten by weight decay.

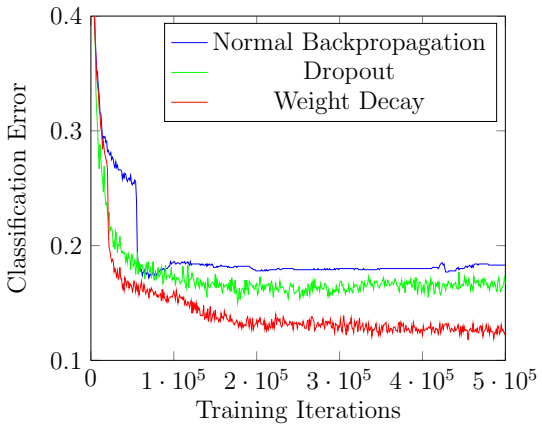

Figure 5: A $784 - 50 - 10$ classifier trained on 1000 samples from the MNIST data.

To increase the OP of the previous case, we increased the network size while decreasing the dataset size. The network had a $784 - 500 - 10$ topology and only 100 training samples were used. Figure 6 illustrates that dropout and weight decay perform comparably.

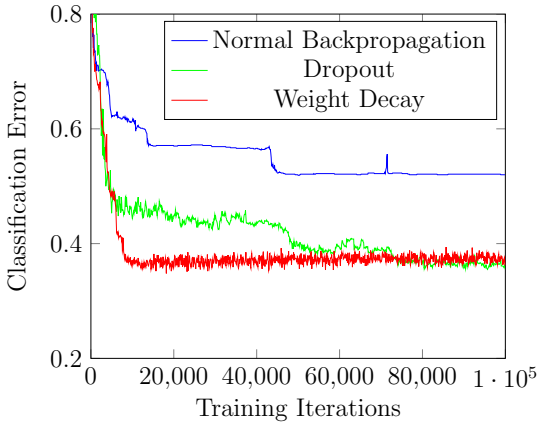

Figure 6: A $784 - 500 - 10$ classifier trained on 100 examples of the MNIST dataset.

For a high OP experiment, we increased the network size yet again. We tested a $784 - 500 - 50 - 500 - 784$ autoencoder on an absurdly small 100 samples. Figure 7 shows that dropout converges *significantly* more rapidly than weight decay and plain backpropagation, and its final converged MSE is also superior.

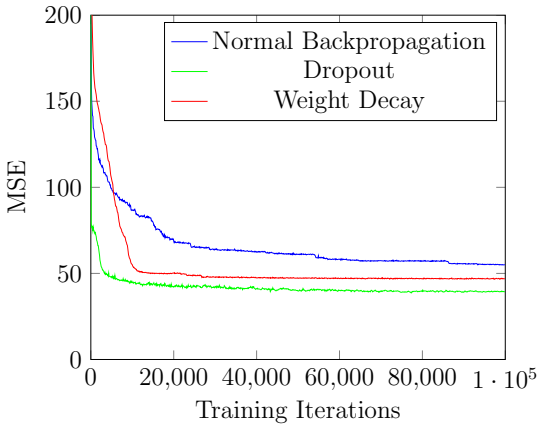

Figure 7: A $784 - 500 - 50 - 500 - 784$ autoencoder being trained on the MNIST digits.

For a more visual demonstration of the regularization methods, we present in Figure 8 the digit 9 sent through each autoencoder.

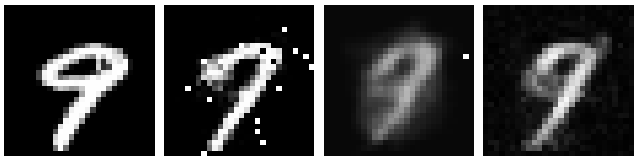

Figure 8: From left to right: Original, Normal Backpropagation, Weight Decay, Dropout

As can be seen in these images, and what is generally true upon visual inspection of the other several thousand images, the normal backpropagation algorithm severely overfits in a system with such a high OP. This can be seen in the white speckles that appear throughout the image. The image produced by the weight decay network is significantly less overfit, but still leaves something to be desired. The final image, produced by the dropout network, is an impressive reconstruction given that the rather large network was only trained on 100 samples.

The reason the weight decay network produces a slight "ghosting" is that, as Nowlan and Hinton tell us when they discuss alternative decay methods, the standard weight decay algorithm gives a lower cost to two weak connections rather than one strong connection[9]. Thus, the autoencoder recreates the image by layering many weak features, whereas the dropout network uses fewer strong features.

This concept—learning very robust features—is precisely what Hinton *et al.* intended when developing the dropout method. The other methods rely on multiple features all adapting together to be able to form a cohesive output, but the dropout network's feature detectors must learn to act alone.

5.2 Weight Decay and Dropout Can Work Together

It can be easily seen that weight decay operates on weights, while dropout operates on nodes. In several experiments, it was uniformly observed that, when dropout already performs well, adding a weight decay can improve it slightly. In Figure 9, we present the same $784 - 500 - 50 - 500 - 784$ autoencoder, but with a fourth model that implements both weight decay and dropout.

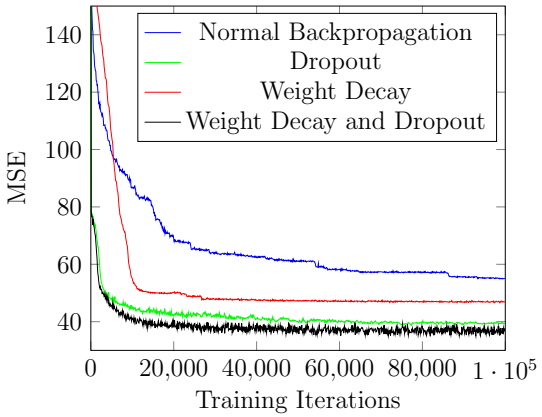

Figure 9: Weight Decay can improve Dropout

5.3 Dropout is Detrimental for Small Networks

Consider a $784 - 16 - 10$ network to be used in classifying MNIST digits. If one applies dropout to this network, the expected hidden layer size is 8. Given a hidden layer of size n, and dropout probability p, the probability of a dropped out layer size k is given by the formula

$$\frac{n!}{k!(n-k)!}p^k(1-p)^{n-k},$$

which is the binomial distribution. The reason this is important is in the analysis of dropped out hidden layer size over the course of training. For this particular network, the probability that the size of the hidden layer is less than 10 is 77.2%. This is detrimental because having a hidden layer smaller than the output size will generally produce a lossy function. With autoencoding, having a smaller hidden layer is necessary, but with classification it is generally undesirable.

With dropout it can be even more problematic because the smaller hidden layer is not constant, which can cause the feature detectors to thrash from feature to feature.

A generalized function for determining the probability that a layer of size n will be less than the output layer of size m is given by

$$\rho = \sum_{k=m-n+1}^{n} \frac{n!}{k!(n-k)!}p^k(1-p)^{n-k},$$

We demonstrate this problem in Figure 10 with the $784 - 16 - 10$ network previously mentioned.

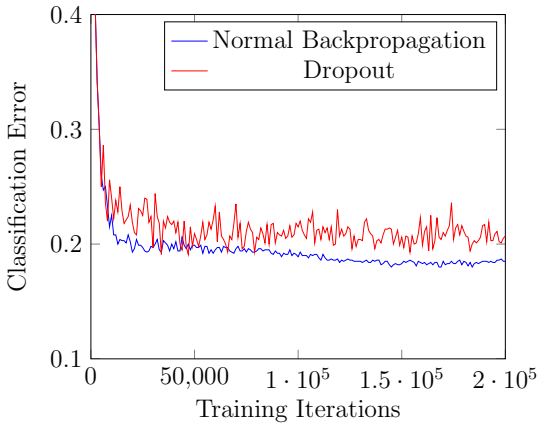

Figure 10: Dropout fails on small hidden layers.

As can be seen in the graph, not only does dropout fail to perform as well as standard backpropagation, but it also fails to smoothly converge as the restricted hidden nodes thrash during training.

6 Conclusion

Weight decay and dropout are two useful ANN regularization methods. In our experiments, we have explored the appropriate domains of each of these. In short, it appears that weight decay is more useful in networks with a smaller propensity to overfit—that is, networks with lower complexity and more training data.

We have confirmed the results of Hinton and Srivastava in as much as dropout is an effective method for preventing co-adaption between feature detectors, and have extended their research to show when preventing co-adaption is deleterious. Namely, when the size of hidden layer is too small, it is probable that a hidden layer will contain an insufficient number of feature detectors to converge, and will instead oscillate around a local minimum. We have provided a mathematical basis for this result, and numerical experiments to support it.

In support of dropout, we have found that in large, deep networks, dropout is both effective at minimizing error and speeding up convergence. We have also shown, in agreement with Srivastava's results, that weight decay and dropout can often be combined in these instances to yield slightly improved performance.

Future research on the topic of dropout includes testing performance in recurrent networks, and performance when combined with more complex weight decay methods (as previously described). Modifications of the dropout algorithm should be explored in an attempt to find a generalization that performs well for networks small and large.

References

[1] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arxiv*, 2012.

[2] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[3] Yaochu Jin, Tatsuya Okabe, and Bernhard Sendhoff. Neural network regularization and ensembling using multi-objective evolutionary algorithms. In *CEC2004*, pages 1–8, 2004.

[4] Alex Krizhevsky, Ilysa Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 1:4, 2012.

[5] Anders Krogh and John A. Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4:950–957, 1992.

[6] Jan Larsen and Lars Kai Hansen. Generalization performance of regularized neural network models. *Neural Networks for Signal Processing*, 4:42–51, 1994.

[7] Steve Lawrence, C. Lee Giles, and Ah Chung Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *AAAI-97*, pages 540–545, 1997.

[8] Yann LeCun, Corinna Cortes, and Cristopher J. C. Burges. The mnist database of handwritten digits.

[9] Steven J. Nowlan and Geoffrey E. Hinton. Simplifying neural networks by soft-weight sharing. *Neural Computation*, 4:473–493, 1992.

[10] Nitish Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, 2013.