Analysis of Dropout in Deep Networks

Satya Narayan Shukla
College of Information and Computer Sciences
University of Massachusetts Amherst
snshukla@cs.umass.edu

Abstract

This work presents an analysis of dropout in deep networks with rectified linear units and the quadratic loss. The results expose surprising differences between the behavior of dropout and more traditional regularizers like weight decay. For example, on some simple data cases dropout training produces negative weights even though the output is the sum of the inputs. This provides a counterpoint to the suggestion that dropout discourages co-adaptation of weights. Moreover, dropout penalty can grow exponentially in the depth of the network while the weight-decay penalty remains essentially linear, and that dropout is insensitive to various re-scalings of the input features and network weights.

1 Introduction

The 2012 ImageNet Large Scale Visual Recognition challenge was won by the University of Toronto team by a surprisingly large margin. In an invited talk at NIPS, Hinton (2012) credited the dropout training technique for much of their success. It is basically a technique for addressing the problem of overfitting in large networks. Dropout training is a variant of stochastic gradient descent where we randomly drop units (along with their connections) from the neural network during training. The gradient calculation and updates are performed on the reduced network and the dropped out nodes are restored before the next iteration. Since the 2012 ImageNet challenge, dropout has been successfully applied to deep neural networks to improve performance in a variety of domains (Dahl, 2012; Deng et al., 2013; Dahl et al., 2013; Kalchbrenner et al., 2014; Chen and Manning, 2014), and is widely used (Schmidhuber, 2015; He et al., 2015; Szegedy et al., 2015; Yang et al., 2016). It has been incorporated into popular packages like TensorFlow, Torch, and Caffe as well.

By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure 1. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks.

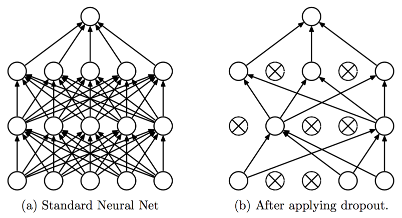

Figure 1: Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

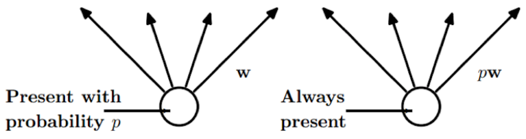

Figure 2: left: training time, right: test time

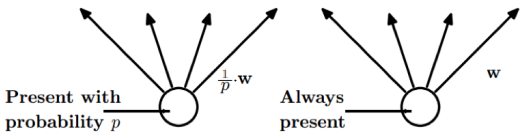

Figure 3: left: training time, right: test time

Applying dropout to a neural network amounts to sampling a "thinned" network from it. The thinned network consists of all the units that survived dropout (Figure 1b). A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. These networks all share weights so that the total number of parameters is still $O(n^2)$, or less. For each presentation of each training case, a new thinned network is sampled and trained. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with extensive weight sharing, where each thinned network gets trained very rarely, if at all.

There are two variants of dropout. In first variant, a unit is retained with probability p during training, the outgoing weights of that unit are multiplied by p at test time as shown in figure 2. While in second, no scaling is applied during test time, the retained units at training time are scaled up by a factor $1/p$ as shown in figure 3. This ensures that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time.

Weight decay training optimizes the empirical error plus an L_2 regularization term, $\lambda||\boldsymbol{w}||^2/2$, so we call $\lambda||\boldsymbol{w}||^2/2$ the L_2 penalty of $\boldsymbol{w}$ since it is the difference between training criterion evaluated at $\boldsymbol{w}$ and the empirical loss of $\boldsymbol{w}$. By analogy, we define the dropout penalty of $\boldsymbol{w}$ to be the difference between the dropout training criterion and the empirical loss of $\boldsymbol{w}$. Dropout penalties measure how much dropout discriminates against weight vectors, so they are key to understanding dropout's inductive bias.

The primary focus of this work is to examine the effect of dropout on the learning algorithm, expose surprising differences between the behaviour of dropout and other traditional regularizers such as weight decay and finally extend our understanding of why dropout succeeds.

2 Related Work

There has been numerous explanations behind the success of dropout. Hinton et al. (2012) suggest that dropout controls overfitting by preventing co-adaptation of weights. Baldi and Sadowski, (2013) and Bachman et al., (2014) view dropout as an ensemble method combining the different network topologies resulting from the random deletion of nodes. Wager et al. (2014) observe that in 1-layer networks dropout essentially forces learning on a more challenging distribution akin to 'altitude training' of athletes.

Most of the analysis of dropout has been focused on single layer setting with just input and output layer. Wager et al. (2013) considered the case where the distribution of label y given feature vector x is a member of the exponential family, and the log-loss is used to evaluate models. They pointed out that, in this situation, the criterion optimized by dropout can be decomposed into the original loss and a term that does not depend on the labels. They then gave approximations to this dropout regularizer and discussed its relationship with other regularizers. But as shown in next section, many aspects of the behavior of dropout and its relationship to other regularizers are qualitatively different when there are hidden units.

Wager et al. (2014) considered dropout for learning topics modeled by a Poisson generative process. They exploited the conditional independence assumptions of the generative process to show that the excess risk of dropout training due to training set variation has a term that decays more rapidly than the straightforward empirical risk minimization, but also has a second additive term related to document length. They also discussed situations where the model learned by dropout has small bias.

Baldi and Sadowski (2014) analyzed dropout in linear networks, and showed how dropout can be approximated by normalized geometric means of subnetworks in the nonlinear case. Gal and Ghahramani (2015) described an interpretation of dropout as an approximation to a deep Gaussian process.

3 Model Description

This section describes the neural network model. We will analyze fully connected layered networks with K inputs, one output, d layers (counting the output, but not the inputs), and n nodes in each hidden layer. Let $\boldsymbol{z}^l$ denote the vectors of input into layer l, $\boldsymbol{y}^l$ denote the vector of outputs from layer l ($\boldsymbol{y}^0 = \boldsymbol{x}$ as input). W^l and $\boldsymbol{b}^l$ be the weights and biases at layer l. The feedforward operation of a standard neural network can be described as (for any hidden unit i)

$$z_i{}^{l+1} = \boldsymbol{w}_i{}^{l+1}\boldsymbol{y}^l + \boldsymbol{b}_i^{l+1}$$
$$y_i{}^{l+1} = f(z_i{}^{l+1})$$

where f is any activation function, for example, $f(x) = \frac{1}{1+\exp(-x)}$. With dropout, the feedforward operation becomes

$$r_j^l \sim \text{Bernoulli}(p)$$
$$\widetilde{\boldsymbol{y}}^l = \boldsymbol{r}^l * \boldsymbol{y}^l$$
$$z_i{}^{l+1} = \frac{\boldsymbol{w}_i{}^{l+1}\widetilde{\boldsymbol{y}}^l}{p} + \frac{\boldsymbol{b}_i^{l+1}}{p}$$
$$y_i{}^{l+1} = f(z_i{}^{l+1})$$

Here $*$ denotes an element-wise product. For any layer l, $\boldsymbol{r}^l$ is a vector of Bernoulli random variables which has probability p of being 1. This vector is multiplied elementwise to the output of the layer l to create thinned outputs and then this thinned output is used as input to the next layer.

Throughout this paper $\mathcal{W}$ is used to denote particular setting of the weights and biases in the network and $\mathcal{W}(\boldsymbol{x})$ to denote the network's output on input $\boldsymbol{x}$ using $\mathcal{W}$. The hidden nodes are ReLUs, and the output node is linear. $\mathcal{W}$ can be decomposed as , $(W_1, \boldsymbol{b}_1, \cdots, W_{d-1}, \boldsymbol{b}_{d-1}, \boldsymbol{w}, b)$ where each W_j is the matrix of weights on connections from the (j - 1)th into the jth hidden layer, each b_j is the vector of bias inputs into the jth hidden layer, $\boldsymbol{w}$ are the weights into the output node, and b is the bias into the output node.

Square loss is used here so the loss of $\mathcal{W}$ on example $(\boldsymbol{x}, y)$ is $(\mathcal{W}(\boldsymbol{x}) - y)^2$. We will refer to a joint probability distribution over examples $(\boldsymbol{x}, y)$ as an example distribution. The risk is the expected loss with respect to an example distribution P. Therefore, the risk of $\mathcal{W}$ is defined as

$$R_P(\mathcal{W}) \overset{\text{def}}{=} \mathbf{E}_{(\boldsymbol{x},y)\sim P}((\mathcal{W}(\boldsymbol{x}) - y)^2)$$

Dropout training independently removes nodes in the network. Each non-output node is dropped out with the same probability q, so $p = 1 - q$ is the probability that a node is kept. (The output node is always kept; dropping it out has the effect of canceling the training iteration.) When a node is dropped out, the node's output is set to 0. To compensate for this reduction, the values of the kept nodes are multiplied by 1/p. With this compensation, the dropout can be viewed as injecting zero-mean additive noise at each non-output node (Wager et al., 2013). A dropout pattern is a boolean vector indicating the choices, for each node in the network, of whether the node is kept (1) or dropped out (0). For a network $\mathcal{W}$, an input $\boldsymbol{x}$, and dropout pattern $\mathcal{R}$, let $\mathcal{D}(\mathcal{W}, \boldsymbol{x}, \mathcal{R})$ be the output of $\mathcal{W}$ when nodes are dropped out following $\mathcal{R}$ (including the $1/p$ scaling of kept nodes' outputs). The goal of dropout training on an example distribution P is to find weights and biases minimizing the **dropout criterion** for a given dropout probability:

$$J_D(\mathcal{W}) \overset{\text{def}}{=} \mathbf{E}_{\mathcal{R}}\mathbf{E}_{(\boldsymbol{x},y)\sim P}((\mathcal{D}(\mathcal{W}, \boldsymbol{x}, \mathcal{R}) - y)^2)$$

Since the selection of dropout pattern and example from P are independent, the order of expectations can be swapped which leads to,

$$J_D(\mathcal{W}) \overset{\text{def}}{=} \mathbf{E}_{(\boldsymbol{x},y)\sim P}\mathbf{E}_{\mathcal{R}}((\mathcal{D}(\mathcal{W}, \boldsymbol{x}, \mathcal{R}) - y)^2) \qquad (1)$$

Above equation indicates when something is true about the dropout criterion for a family of distributions concentrated on single examples, then (usually) the same thing will be true for any mixture of these single-example distributions.

4 Properties of Dropout

4.1 Dropout is scale-free

Dropout in deep network is independent of the scale of the input features. Let A be a square diagonal matrix with full rank and $(A\boldsymbol{x}, y)$ be the scaled version of $(\boldsymbol{x}, y)$. Consider the original network $\mathcal{W} = (W_1, \boldsymbol{b}_1, \cdots, W_{d-1}, \boldsymbol{b}_{d-1}, \boldsymbol{w}, b)$ and let $\mathcal{W}' = (W_1 A^{-1}, \boldsymbol{b}_1, \cdots, W_{d-1}, \boldsymbol{b}_{d-1}, \boldsymbol{w}, b)$. For any $\boldsymbol{x}$, $\mathcal{W}(\boldsymbol{x}) = \mathcal{W}'(A\boldsymbol{x})$, as A^{-1} cancels the effect of scaling i.e. A before it gets to the rest of the network. Furthermore, for any dropout pattern $\mathcal{R}$, we have $\mathcal{D}(\mathcal{W}, \boldsymbol{x}, \mathcal{R}) = \mathcal{D}(\mathcal{W}', A\boldsymbol{x}, \mathcal{R})$. Once again A^{-1} undoes the effects of A on kept nodes (since A is diagonal), and the rest of the network $\mathcal{W}'$ is modified by R in the same way as $\mathcal{W}$. Thus, $J_D(\mathcal{W}') = J_D(\mathcal{W})$ and $\mathcal{R}(\mathcal{W}') = \mathcal{R}(\mathcal{W})$ proving that dropout is scale-free. As explained in experiments section, weight decay does not enjoy such scale-free status.

4.2 Dropout is invariant to parameter scaling

For a network $\mathcal{W} = (W_1, \boldsymbol{b}_1, \cdots, W_{d-1}, \boldsymbol{b}_{d-1}, \boldsymbol{w}, b)$, let's define

$$\mathcal{W}' = (c_1 W_1, c_1 \boldsymbol{b}_1, c_2 W_2, c_1 c_2 \boldsymbol{b}_2 \cdots, c_{d-1} W_{d-1}, c_1 c_2 \cdots c_{d-1} \boldsymbol{b}_{d-1}, c_d \boldsymbol{w}, c_1 c_2 \cdots c_d b)$$

where all c_i's are positive. For each hidden layer j, let $(h_{j1}, \cdots, h_{jn})$ be the jth hidden layer when applying $\mathcal{W}$ to $\boldsymbol{x}$ with dropout pattern $\mathcal{R}$, and let $(\widetilde{h}_{j1}, \cdots, \widetilde{h}_{jn})$ be the jth hidden layer when applying $\mathcal{W}'$ instead. Then for all i, we have $\widetilde{h}_{ji} = c_1 c_2 \cdots c_j h_{ji}$. Since, all c_i's are positive, the same units are zeroed out in $\mathcal{W}$ and $\mathcal{W}'$. Therefore,

$$\mathcal{D}(\mathcal{W}', \boldsymbol{x}, \mathcal{R}) = (\prod_j c_j)\mathcal{D}(\mathcal{W}, \boldsymbol{x}, \mathcal{R})$$

When $\prod_j c_j = 1$, this implies $\mathcal{D}(\mathcal{W}, \boldsymbol{x}, \mathcal{R}) = \mathcal{D}(\mathcal{W}', \boldsymbol{x}, \mathcal{R}), \mathcal{W}'(\boldsymbol{x}) = \mathcal{W}(\boldsymbol{x}), J_D(\mathcal{W}) = J_D(\mathcal{W}')$. Hence the dropout penalties for $\mathcal{W}$ and $\mathcal{W}'$ are same.

This implies that the dropout criterion never has isolated minimizers, since one can continuously up-scale the weights on one layer with a compensating down-scaling at another layer to get a contiguous family of networks computing the same function and having the same dropout criterion.

4.3 Dropout penalty could be negative

Let's consider a simple example of 2-layer network shown in fig. 4. All the weights are equal to 1 and biases equal to 0. $\mathcal{W}(1, -1)$ = 0 since both the hidden nodes compute to 0. Each dropout pattern indicates the subset of the four lower nodes to be kept, and when $q = p = 1/2$ each subset is equally likely to be kept. Out of the 16 dropout patterns, only three dropout patterns produce a non-zero output (since hidden layer are ReLUs), so if P is concentrated on the example, $\boldsymbol{x} = (1, -1)$ and $y = 8$, the dropout criterion is

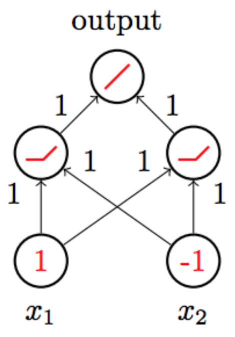

Figure 4: A simple network with negative dropout penalty.

$$J_D(\mathcal{W}) = \frac{1}{16}(8-8)^2 + \frac{2}{16}(4-8)^2 + \frac{13}{16}(0-8)^2 = 54$$

As mentioned in the introduction, the dropout penalty for a given example distribution and dropout probability is the amount that the dropout criterion exceeds the risk, $J_D(\mathcal{W}) - R(\mathcal{W})$. This dropout penalty decomposes into an expectation of penalties over single examples:

$$J_D(\mathcal{W}) - R(\mathcal{W}) = \mathbf{E}_{(\boldsymbol{x},y)\sim P}(\mathbf{E}_{\mathcal{R}}((\mathcal{D}(\mathcal{W},\boldsymbol{x},\mathcal{R}) - y)^2) - (\mathcal{W}(\boldsymbol{x}) - y)^2)$$

For the above example, we have $R(\mathcal{W}) = (0-8)^2 = 64$, which leads to a dropout penalty of -10.

Wager et al. (2013) show that for 1-layer generalized linear models, the dropout penalty is non-negative but the dropout penalty is negative in our example. This is because the variance in the output due to dropout causes the network to better fit the data (on average) than the network's non-dropout evaluation.

Baldi and Sadowski (2014) show that for networks of linear units (as opposed to the non-linear rectified linear units discussed here) the network's output without dropout equals the expected output over dropout patterns, $\mathcal{W}(\boldsymbol{x}) = \mathbf{E}_{\mathcal{R}}\mathcal{D}(\mathcal{W},\boldsymbol{x},\mathcal{R})$. Assume for the moment that the network consists of linear units and the example distribution is concentrated on the single example $(\boldsymbol{x}, y)$. Using the bias-variance decomposition for square loss and this property of linear networks,

$$J_D(\mathcal{W}) = \mathbf{E}_{\mathcal{R}}((\mathcal{D}(\mathcal{W},\boldsymbol{x},\mathcal{R}) - y)^2) = (\mathbf{E}_{\mathcal{R}}(\mathcal{D}(\mathcal{W},\boldsymbol{x},\mathcal{R}) - y))^2 + \mathrm{Var}_{\mathcal{R}}(\mathcal{D}(\mathcal{W},\boldsymbol{x},\mathcal{R}) \geq (\mathcal{W}(\boldsymbol{x}) - y)^2$$

and the dropout penalty is again non-negative. Since the same calculations go through when averaging over multiple examples, we see that the dropout penalty is always non-negative for networks of linear nodes. When all the weights, biases and inputs in a network of rectified linear units are positive, then the rectified linear units behave as linear units, so the dropout penalty will again be non-negative. Therefore, for all example distributions, a necessary condition for dropout penalty in rectified linear networks is that either a weight, input, or bias is negative.

4.4 Multi-layer dropout penalty does depend on labels

For each input $\boldsymbol{x}$ and dropout pattern $\mathcal{R}$, let $\mathcal{H}(\mathcal{W},\boldsymbol{x},\mathcal{R})$ be the values presented to the output node with dropout. As before, let $\boldsymbol{w}$ be the weights on connections directly into the output node and let b be the bias at the output node. Let $\boldsymbol{h}$ be the values coming into the output node in the non-dropped out network.

$$\begin{aligned} \text{Dropout penalty} &= \mathbf{E}((\boldsymbol{w}\cdot\mathcal{H}(\mathcal{W},\boldsymbol{x},\mathcal{R}) + b - y)^2) - (\boldsymbol{w}\cdot\boldsymbol{h} + b - y)^2 \\ &= \mathbf{E}((\boldsymbol{w}\cdot\mathcal{H}(\mathcal{W},\boldsymbol{x},\mathcal{R}) + b - \boldsymbol{w}\cdot\boldsymbol{h} + \boldsymbol{w}\cdot\boldsymbol{h} - y)^2) - (\boldsymbol{w}\cdot\boldsymbol{h} + b - y)^2 \\ &= \mathbf{E}(\delta^2) + 2(\boldsymbol{w}\cdot\boldsymbol{h} + b - y)\mathbf{E}(\delta) \end{aligned}$$

where $\delta = \boldsymbol{w}\cdot(\mathcal{H}(\mathcal{W},\boldsymbol{x},\mathcal{R}) - \boldsymbol{h})$. Hence, dropout penalty depends on the label y unless $\mathbf{E}(\delta) = 0$. Typically $\mathbf{E}(\delta) \neq 0$, consider the case where $\boldsymbol{x} = (1, -2)$ and there are two hidden nodes with weights 1 on their connections to the output node. The value at the hidden nodes without dropout is 0, but with dropout the hidden nodes are never negative and computes positive values when only the negative input is dropped, so the expectation of δ is positive.

This adds a counterpoint to the behavior of dropout on a variety of linear models including logistic regression (Wager et al., 2013), the dropout penalty can depend on the value of the response variable in deep networks with ReLUs and the quadratic loss. Thus in a fundamental and important respect, dropout differs from traditional regularizers like weight-decay or an L1 penalty.

4.5 Growth of dropout penalty with number of hidden layers

Let's consider the example $\boldsymbol{x} = (1, 1, \cdots 1), y = c^d K n^{d-1}$ and let $\mathcal{W}$ (K inputs, d layers, n hidden units in each layer) be a network whose weights are all c and biases are all zero. For this network, the output is equal to $c^d K n^{d-1}$ and square loss is zero. Consider now dropout on this network. This is equivalent to changing all of the weights from c to $2c$ and, independently with probability 1/2, replacing the value of each node with 0. For a fixed dropout pattern, each node on a given layer has the same weights, and receives the same (kept) inputs. Thus, the value computed at every node on the same layer is the same. If k_0 is the number of input nodes kept under dropout, and, for each $j \in \{1, \cdots, d-1\}$, k_j is the number of hidden nodes kept in layer j, then the output

$\hat{y} = (2c)^d \prod_{j=0}^{d-1} k_j$.

Using a bias-variance decomposition,

$$\mathbf{E}(\hat{y} - y)^2 = (\mathbf{E}\hat{y} - y)^2 + \mathbf{Var}(\hat{y})$$

Since k_j's are independent binomial variables, we have

$$\mathbf{E}\hat{y} = (2c)^d \mathbf{E}(k_0) \prod_{j=1}^{d-1} \mathbf{E}(k_j) = (2c)^d \frac{K}{2} (\frac{n}{2})^{d-1} = c^d K n^{d-1} = y$$

$$\mathbf{E}(\hat{y} - y)^2 = (\mathbf{E}\hat{y} - y)^2 + \mathbf{Var}(\hat{y}) = \mathbf{Var}(\hat{y}) = \mathbf{E}(\hat{y}^2) - (\mathbf{E}\hat{y})^2$$

$$\mathbf{E}(\hat{y}^2) = (2c)^{2d} \mathbf{E}(k_0^2) \prod_{j=1}^{d-1} \mathbf{E}(k_j^2) = (2c)^{2d} \frac{K(K+1)}{4} (\frac{n(n+1)}{4})^{d-1}$$

$$J_D(\mathcal{W}) = \mathbf{E}(\hat{y} - y)^2 = y^2((1 + \frac{1}{K})(1 + \frac{1}{n})^{d-1} - 1) \geq y^2/K$$

This shows that if $y = \exp(\Theta(d))$, the dropout penalty grows exponentially in d. On the other hands L_2 penalty is the number of weights time $\lambda c^2/2$, so it grows polynomially. Since the dropout penalty of a distribution decomposes into the expectation over single examples, the dropout penalty for any distribution P that puts positive probability on this $((1, 1, \cdots, 1), y)$ example has a term that grows exponentially in the depth. This may enable dropout to penalize the complexity of the network.

4.6 Dropout uses negative weights for monotone functions

If the weights of a unit are non-negative, then the unit computes a monotone function, in the sense that increasing any input while keeping the others fixed increases the output. The bias does not affect a node's monotonicity. A network of monotone units is also monotone. Let's consider a simple 2-layer network with joint input distribution under dropout given as, $P((0,0),0) = 1/2, P((0,0),1) = 1/8, P((0,1),1) = 1/8, P((1,0),1) = 1/8, P((1,1),1) = 1/8$. Let's define $\mathcal{W}_{\text{neg}}$ a network that uses negative weights such that weight vector for each hidden unit in 1st hidden layer is defined as $w_{i'} = -1$ for $i' < i$ and $w_i = 1$ and $w_{i'} = -1$ for $i' > i$ for each block of 2. In this way, the sum of the values on the first hidden layer of $\mathcal{W}_{\text{neg}}$ is exactly $n/2$ since each block of 2 has single 1 for non-zero input. Let c be the weights from 1st hidden layer to output and output bias equal to 1/5. Due to the bias, $\mathcal{W}_{\text{neg}}$(0,0) = 1/5. Thus, contribution to J_D from examples with $\boldsymbol{x} = (0,0)$ in this joint distribution is $1/2 * (1/5)^2 + 1/8 * (1/5 - 1)^2 = 1/10$. Now for $\boldsymbol{x} \neq (0,0)$, $\mathbf{E}(\hat{y}) = c * 2^2 * \frac{n}{4} = nc, \mathbf{E}(\hat{y}^2) = c^2 * 2^4 * \left[(\frac{n}{4})^2 + \frac{n}{8}\right] = 2nc^2\left[\frac{n}{2} + 1\right]$.

$$J_D(\mathcal{W}_{\text{neg}}) = \mathbf{E}(\hat{y} - 1)^2 = 1 - 2\mathbf{E}(\hat{y}) + \mathbf{E}(\hat{y}^2) = 1 - \frac{n}{n+2}$$

where $c = \frac{1}{n+2}$ minimizes the above expression. Hence, for pretty large n, $J_D(\mathcal{W}_{\text{neg}}) = \frac{1}{10}$. Now, let $\mathcal{W}$ be an arbitrary network with non-negative weights. For the given distribution, dropout criterion can be written as,

$$J_D(\mathcal{W}) = \frac{\mathbf{E}_{\mathcal{R}}((\mathcal{D}(\mathcal{W}, (0,0), \mathcal{R}) - 0)^2) + \mathbf{E}_{\mathcal{R}}((\mathcal{D}(\mathcal{W}, (1,1), \mathcal{R}) - 1)^2)}{2}$$

Handling the dropout at the input explicitly, and using the bias-variance decomposition keeping just the bias terms,

$$J_D(\mathcal{W}) \geq \frac{(V_{00} - 0)^2 + ((V_{00} - 1)^2 + (V_{02} - 1)^2 + (V_{20} - 1)^2 + (V_{11} - 1)^2)/4}{2}$$

where $V_{00} = \mathcal{W}(0,0), V_{02} = \mathcal{W}(0,2), V_{20} = \mathcal{W}(2,0), V_{22} = \mathcal{W}(2,2)$. Since, network has non-negative weights and its activation functions are non-decreasing, $V_{22} + V_{00} \geq V_{20} + V_{02}$.

$$8J_D(\mathcal{W}) \geq 4(V_{00} - 0)^2 + (V_{00} - 1)^2 + (V_{02} - 1)^2 + (V_{20} - 1)^2 + (V_{11} - 1)^2$$

This is convex and symmetric in V_{02} and V_{20} so they both take the same value at the minimizer of the RHS. Substituting $V_{02} = V_{20}, V_{22} = V_{20} + V_{02} - V_{00} + \epsilon$ for $\epsilon \geq 0$ and minimizing the RHS leads to

$$J_D(\mathcal{W}) \geq \frac{1}{8}$$

Thus, $J_D(\mathcal{W}_{\text{neg}}) < J_D(\mathcal{W})$. Hence, the dropout criterion uses negative weights even for monotonic functions which is not so intuitive. Such use of negative weights constitutes co-adaptation and this adds a counterpoint to previous analyses showing that dropout discourages co-adaptation (Srivastava et al., 2014; Helmbold and Long, 2015).

5 Experiments

To verify the scale-free properties of dropout and compare it with the weight decay, a simple 2-layer network was implemented with 5 input units and 5 hidden units. Ten training examples were generated uniformly at random from $[-1, 1]^5$ and target outputs were assigned using $y = \prod_i \text{sign}(x_i)$. Five training sets $S1, \cdots, S5$ with ten examples each were obtained by rescaling the inputs by $0.5, 0.75, 1, 1.25, 1.5$ and leaving the outputs unchanged. $\mathcal{W}_D$ was trained with dropout probability 1/2 and $\mathcal{W}_2$ was trained with weight decay with $\lambda = 1/2$. The average training losses of $\mathcal{W}_D$ and $\mathcal{W}_2$, over the 10 runs, are shown in Figure 5. The theoretical insensitivity of dropout to the scale of the inputs described in section 4.1 is also seen here, along with the contrast with weight decay which decreases with increase in scaling factor.

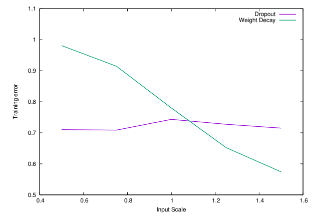

Figure 5: Training error as a function of the scale of the inputs for Dropout and Weight Decay

6 Conclusions

The reasons behind dropout's surprisingly good performance in training deep networks across a variety of applications are somewhat mysterious and there is relatively little existing formal analysis. A variety of explanations have been offered (e.g. (Bachman et al., 2014; Baldi and Sadowski, 2014, 2013; Gal and Ghahramani, 2015; Wager et al., 2014)), including the possibility that dropout reduces the amount of co-adaptation in a network's weights (Hinton et al., 2012).

The dropout criterion is an expected loss over dropout patterns, and the variance in the output values over dropout patterns contributes to this expected loss. Therefore dropout may co-adapt weights in order to reduce this (artificial) variance. This happens even in very simple situations where nothing in the training data justifies negative weights. This indicates that the relationship between dropout and co-adaptation is not a simple one.

The effects of dropout in deep neural networks are rather complicated, and approximations can be misleading since the dropout penalty is very non-convex even in 1-layer networks (Helmbold and Long, 2015). In section 4, it has been shown that dropout does enjoy several scale-invariance properties that are not shared by weight-decay. A perhaps surprising consequence of these invariances is that there are never isolated local minima when learning a deep network with dropout.

In (Wager et al., 2013), dropout was viewed as a regularization method, adding a data dependent penalty to the empirical loss of (presumably) undesirable solutions. Section 4 shows that, unlike the generalized linear models case analyzed there, the dropout penalty in deeper networks can be negative and depends on the labels in the training data, and thus behaves unlike most regularizers. On the other hand, the dropout penalty can grow exponentially in the depth of the network, and thus may better reflect the complexity of the underlying model space than L2 regularization.

References

[1] G. E. Hinton. Dropout: a simple and effective way to improve neural networks, 2012. videolectures.net.

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting.*JMLR*, 15:1929–1958, 2014.

[3] P. Baldi and P. Sadowski. The dropout learning algorithm. *Artificial intelligence*, 210: 78–122, 2014.

[4] Pierre Baldi and Peter J Sadowski. Understanding dropout. *In Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

[5] S. Wager, W. Fithian, S. Wang, and P. S. Liang. Altitude training: Strong bounds for single-layer dropout. *NIPS*, 2014.

[6] S. Wager, S. Wang, and P. Liang. Dropout training as adaptive regularization. *NIPS*, 2013.

[7] G. E. Dahl. Deep learning how I did it: Merck 1st place interview, 2012. http://blog.kaggle.com.

[8] G. E. Dahl, T. N. Sainath, and G. E. Hinton. Improving deep neural networks for LVCSR using rectified linear units and dropout. *ICASSP*, 2013.

[9] L. Deng, J. Li, J. Huang, K. Yao, D. Yu, F. Seide, M. L. Seltzer, G. Zweig, X. He, J. Williams, Y. Gong, and A. Acero. Recent advances in deep learning for speech research at microsoft. *ICASSP*, 2013.

[10] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *ACL*, pages 655–665, 2014.

[11] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[12] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CVPR*, 2015.

[14] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29, 2016.

[15] TensorFlow. Tensorflow, 2016. https://www.tensorflow.org.

[16] Torch. Torch, 2016. http://torch.ch.

[17] Caffe. Caffe, 2016. http://caffe.berkeleyvision.edu.

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. Arxiv, arXiv:1207.0580v1, 2012.

[19] P. Bachman, O. Alsharif, and D. Precup. Learning with pseudo-ensembles. *NIPS*, 2014.

[20] D. P. Helmbold and P. M. Long. On the inductive bias of dropout. *JMLR*, 16:3403–3454, 2015.

[21] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. arXiv:1506.02142, 2015.