Journal of Machine Learning Research 18 (2018) 1-28 Submitted 11/16; Revised 8/17; Published 4/18

Surprising properties of dropout in deep networks

David P. Helmbold DPH@SOE.UCSC.EDU
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064, USA

Philip M. Long PLONG@GOOGLE.COM
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA

Editor: Sanjoy Dasgupta

Abstract

We analyze dropout in deep networks with rectified linear units and the quadratic loss. Our results expose surprising differences between the behavior of dropout and more traditional regularizers like weight decay. For example, on some simple data sets 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. We also show that the 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, outputs, and network weights. This last insensitivity implies that there are no isolated local minima of the dropout training criterion. Our work uncovers new properties of dropout, extends our understanding of why dropout succeeds, and lays the foundation for further progress.

Keywords: Dropout, deep neural networks, regularization, learning theory.

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. Dropout training is a variant of stochastic gradient descent (SGD) where, as each example is processed, the network is temporarily perturbed by randomly "dropping out" nodes of the network. The gradient calculation and weight updates are performed on the reduced network, and the dropped out nodes are then restored before the next SGD iteration. Since the ImageNet competition, dropout has been successfully applied to 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; Havaei et al., 2017); for example, it is incorporated into popular packages such as TensorFlow, Torch, and Caffe. It is intriguing that crippling the network during training often leads to such dramatically improved results, and dropout has also sparked substantial

research on related methods (for example, (Goodfellow et al., 2013; Wan et al., 2013; Gal and Ghahramani, 2016)).

In this work, we examine the effect of dropout on the inductive bias of the learning algorithm. A match between dropout's inductive bias and some important applications could explain the success of dropout, and its popularity also motivates the study of its properties.

Weight decay training optimizes the empirical error plus an L_2 regularization term, $\frac{\lambda}{2}||\mathbf{w}||_2^2$, so we call $\frac{\lambda}{2}||\mathbf{w}||_2^2$ the *L_2 penalty* of $\mathbf{w}$ since it is the difference between training criterion evaluated at $\mathbf{w}$ and the empirical loss of $\mathbf{w}$. By analogy, we define the *dropout penalty* of $\mathbf{w}$ to be the difference between the dropout training criterion and the empirical loss of $\mathbf{w}$ (see Section 2). Dropout penalties measure how much dropout discriminates against weight vectors, so they are key to understanding dropout's inductive bias.

Even in one-layer networks, conclusions drawn from (typically quadratic) approximations of the dropout penalty can be misleading (Helmbold and Long, 2015). Therefore we focus on exact formal analysis of dropout in multi-layer networks. Theoretical analysis of deep networks is notoriously difficult, so we might expect that a thorough understanding of dropout in deep networks must be achieved in stages. In this paper we further the process by exposing some of the surprising ways that the inductive bias of dropout differs from L_2 and other standard regularizers. These include the following:

- We show that dropout training can lead to negative weights *even when the output is a positive multiple of the the inputs.* Arguably, such use of negative weights constitutes co-adaptation – this adds a counterpoint to previous analyses showing that dropout discourages co-adaptation (Srivastava et al., 2014; Helmbold and Long, 2015).

- Unlike weight decay and other p-norm regularizers, dropout training is insensitive to the rescaling of input features, and largely insensitive to rescaling of the outputs; this may play a role in dropout's practical success. Dropout is also unaffected if the weights in one layer are scaled up by a constant c, and the weights of another layer are scaled down by c; this implies that dropout training does not have isolated local minima.

- The dropout penalty grows exponentially in the depth of the network in cases where the L_2 regularizer grows linearly. This may enable dropout to penalize the complexity of the network in a way that more meaningfully reflects the richness of the network's behaviors. (The exponential growth with d of the dropout penalty is reminiscent of some regularizers for deep networks studied by Neyshabur et al. (2015).)

- Dropout in deep networks has a variety of other behaviors different from standard regularizers. In particular: the dropout penalty for a set of weights can be negative; the dropout penalty of a set of weights depends on both the training instances and the labels; and although the dropout probability intuitively measures the strength of dropout regularization, the dropout penalties are often non-monotonic in the dropout probability. In contrast, Wager et al. (2013) show that when dropout is applied to generalized linear models, the dropout penalty is always non-negative and does not depend on the labels.

Our analysis is for multilayer neural networks with the square loss at the output node. The hidden layers use the popular rectified linear units (Nair and Hinton, 2010) outputting $\sigma(a) = \max(0, a)$ where a is the node's activation (the weighted sum of its inputs). We study the minimizers of a criterion that may be viewed as the objective function when using dropout. This abstracts away sampling and optimization issues to focus on the inductive bias, as in some previous work (Breiman, 2004; Zhang, 2004; Bartlett et al., 2006; Long and Servedio, 2010; Helmbold and Long, 2015). See Section 2 for a complete explanation.

Related work

A number of possible explanations have been suggested for dropout's success. Hinton et al. (2012) suggest that dropout controls network complexity by restricting the ability to co-adapt weights and illustrate how it appears to learn simpler functions at the second layer. Others (Baldi and Sadowski, 2013; 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 formal analysis of the inductive bias of dropout has concentrated on the single-layer setting, where a single neuron combines the (potentially dropped-out) inputs. Wager et al. (2013) considered the case that the distribution of label y given feature vector $\mathbf{x}$ 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. As we have seen, 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.

The impact of dropout (and its relative dropconnect) on generalization (roughly, how much dropout restricts the search space of the learner) was studied in (Wan et al., 2013).

In the on-line learning with experts setting, Van Erven et al. (2014) showed that applying dropout in on-line trials leads to algorithms that automatically adapt to the input sequence without requiring doubling or other parameter-tuning techniques.

The rest of the paper is organized as follows. Section 2 introduces our notation and formally defines the dropout model. We prove that dropout enjoys several scaling invariances that weight-decay doesn't in Section 3, and that dropout requires negative weights even in very simple situations in Section 4. Section 5 uncovers various properties of the

dropout penalty function. Section 6 describes some simulation experiments. We provide some concluding remarks in Section 7.

2. Preliminaries

Throughout, 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. We assume that n is a positive multiple of K and that K is a perfect square and a power of two to avoid unilluminating floor/ceiling clutter in the analysis. We will call this the *standard architecture*. We use $\mathcal{W}$ to denote a particular setting of the weights and biases in the network and $\mathcal{W}(\mathbf{x})$ to denote the network's output on input $\mathbf{x}$ using $\mathcal{W}$. The hidden nodes are ReLUs, and the output node is linear. $\mathcal{W}$ can be decomposed as $(W_1, \mathbf{b}_1, ..., W_{d-1}, \mathbf{b}_{d-1}, \mathbf{w}, b)$, where each W_j is the matrix of weights on connections from the $j-1$st into the jth hidden layer, each $\mathbf{b}_j$ is the vector of bias inputs into the jth hidden layer, $\mathbf{w}$ are the weights into the output node, and b is the bias into the output node.

An *example distribution* is a joint probability distribution over $(\mathbf{x}, y)$ pairs We focus on square loss, so the loss of $\mathcal{W}$ on example $(\mathbf{x}, y)$ is $(\mathcal{W}(\mathbf{x}) - y)^2$. The *risk* is the expected loss with respect to an example distribution P, we denote the risk of $\mathcal{W}$ as $R_P(\mathcal{W}) \stackrel{\text{def}}{=} \mathbf{E}_{(\mathbf{x},y)\sim P}\left((\mathcal{W}(\mathbf{x}) - y)^2\right)$. The subscript will often be omitted when P is clear from the context.

The goal of L_2 training is to find weights and biases minimizing the *L_2 criterion* with regularization strength λ: $J_2(\mathcal{W}) \stackrel{\text{def}}{=} R(\mathcal{W}) + \frac{\lambda}{2}||\mathcal{W}||^2$. Here and throughout, we use $||\mathcal{W}||^2$ to denote the sum of the squares of the weights of $\mathcal{W}$. (As usual, the biases are not penalized.) We use $\mathcal{W}_{L_2}$ to denote a minimizer of this criterion. The L_2 penalty, $\frac{\lambda}{2}||\mathcal{W}||^2$, is non-negative. This is useful, for example, to bound the risk of a minimizer $\mathcal{W}_{L_2}$ of J_2, since $R(\mathcal{W}) \leq J_2(\mathcal{W})$.

Dropout training independently removes nodes in the network. In our analysis 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 cancelling 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). [1]

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 $\mathbf{x}$, and dropout pattern $\mathcal{R}$, let $\mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R})$ be the output of $\mathcal{W}$ when nodes are dropped out or not following $\mathcal{R}$ (including the $1/p$ rescaling 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:

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

1. Some authors use a similar adjustment where the weights are scaled down at prediction time instead of inflating the kept nodes' outputs at training time.

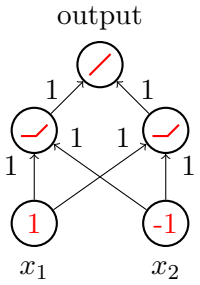

Figure 1: A network where the dropout penalty is negative.

This criterion is equivalent to the expected risk of the dropout-modified network, and we use $\mathcal{W}_D$ to denote a minimizer of it. Since the selection of dropout pattern and example from P are independent, the order of the two expectations can be swapped, yielding

$$J_D(\mathcal{W}) = \mathbf{E}_{(\mathbf{x},y)\sim P}\mathbf{E}_{\mathcal{R}}\left((\mathcal{D}(\mathcal{W},\mathbf{x},\mathcal{R}) - y)^2\right). \tag{1}$$

Equation (1) is a key property of the dropout criterion. It 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.

Consider now the example network in Figure 1. The weight parameters W_1 and $\mathbf{w}$ are all-1's, and all of the biases are 0. $\mathcal{W}(1,-1) = 0$ as each hidden node computes 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. If $\mathcal{R}$ is the dropout pattern where input x_2 is dropped and the other nodes are kept, then the network computes $\mathcal{D}(\mathcal{W},(1,-1),\mathcal{R}) = 8$ (recall that when $p = 1/2$ the values of non-dropped out nodes are doubled). Only three dropout patterns produce a non-zero output, so if P is concentrated on the example $\mathbf{x} = (1,-1), y = 8$ the dropout criterion is:

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

As mentioned in the introduction, the *dropout penalty* of a weight vector for a given example distribution and dropout probability is the amount that the dropout criterion exceeds the risk, $J_D(\mathcal{W}) - R(\mathcal{W})$. Wager et al. (2013) show that for 1-layer generalized linear models, the dropout penalty is non-negative.

Since $\mathcal{W}(1,-1) = 0$, we have $R(\mathcal{W}) = 64$, and 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. In Section 5.2, we give a necessary condition for this variance to be beneficial. As with the dropout criterion, the dropout penalty decomposes into an expectation of penalties over single examples:

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

Definition 1 *Define $P_{(\mathbf{x},y)}$ as the distribution with half of its weight on example $(\mathbf{x},y)$ and half of its weight on example $(\mathbf{0},0)$.*

Unless indicated otherwise, we assume $p = q = 1/2$ for simplicity, although this is not crucial for our results.

3. Scaling inputs, weights and outputs

This section compares the sensitivity of dropout and weight decay on the scale of the training data.

3.1 Dropout is scale-free

Here we prove that dropout regularizes deep networks in a manner that is independent of the scale of the input features. In other words, training under dropout regularization does not penalize the use of large weights when needed to compensate for small input values.

Definition 2 *For any example distribution P, define the* dropout aversion *of P to be the maximum, over minimizers $\mathcal{W}_\mathcal{D}$ of the dropout criterion $J_\mathcal{D}(\mathcal{W}_\mathcal{D})$, of $R_P(\mathcal{W}_\mathcal{D}) - \inf_\mathcal{W} R_P(\mathcal{W})$.*

The dropout aversion of P measures the extent to which P is incompatible with the inductive bias of dropout, measured by the risk gap between the true risk minimizer and the optimizers of the dropout criterion.

Definition 3 *For example distribution P and square matrix A, denote by $A \circ P$ the distribution obtained by sampling $(\mathbf{x}, y)$ from P, and outputting $(A\mathbf{x}, y)$.*

When A is diagonal and has full rank, then $A \circ P$ is a rescaling of the inputs, like changing one input from minutes to seconds and another from feet to meters.

Theorem 4 *For any example distribution P, and any diagonal full-rank $K \times K$ matrix A, the dropout aversion of P equals the dropout aversion of $A \circ P$.*

Proof: Choose a network

$$\mathcal{W} = (W_1, \mathbf{b}_1, ..., W_{d-1}, \mathbf{b}_{d-1}, \mathbf{w}, b).$$

Let

$$\mathcal{W}' = (W_1 A^{-1}, \mathbf{b}_1 ..., W_{d-1}, \mathbf{b}_{d-1}, \mathbf{w}, b).$$

For any $\mathbf{x}$, $\mathcal{W}(\mathbf{x}) = \mathcal{W}'(A\mathbf{x})$, as A^{-1} undoes the effect of A before it gets to the rest of the network, which is unchanged. Furthermore, for any dropout pattern $\mathcal{R}$, we have $\mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R}) = \mathcal{D}(\mathcal{W}', A\mathbf{x}, \mathcal{R})$. 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 $\mathcal{R}$ an a manner paralleling $\mathcal{W}$. Thus, there is bijection between networks $\mathcal{W}$ and networks $\mathcal{W}'$ with $J_\mathcal{D}(\mathcal{W}') = J_\mathcal{D}(\mathcal{W})$ and $R(\mathcal{W}') = R(\mathcal{W})$, yielding the theorem. ■

Theorem 4 indicates that some common normalizations of the input features (for example, to have unit variance) do not affect the quality of the dropout criterion minimizers, but normalization might change the speed of convergence and which minimizer is reached. Centering the features has slightly different properties. Although it is easy to use the biases to define a $\mathcal{W}'$ that "undoes" the centering in the non-dropout computation, different $\mathcal{W}'$ appear to be required for different dropout patterns, breaking the bijection exploited in Theorem 4.

As we will see in Section 3.4, weight decay does not enjoy such scale-free status.

3.2 Dropout's invariance to parameter scaling

Next, we describe an equivalence relation among parameterizations for dropout networks of depth $d \geq 2$. Basically, scaling the parameters at a level creates a corresponding scaling of the output. (A similar observation was made in a somewhat different context by Neyshabur et al. (2015).)

Theorem 5 *For any input* $\mathbf{x}$*, dropout pattern* $\mathcal{R}$*, any network*

$$\mathcal{W} = (W_1, \mathbf{b}_1, ..., W_{d-1}, \mathbf{b}_{d-1}, \mathbf{w}, b),$$

and any positive $c_1, ..., c_d$*, if*

$$\mathcal{W}' = \left(c_1 W_1, c_1 \mathbf{b}_1, c_2 W_2, c_1 c_2 \mathbf{b}_2, ..., c_{d-1} W_{d-1}, \left(\prod_{j=1}^{d-1} c_j \right) \mathbf{b}_{d-1}, c_d \mathbf{w}, \left(\prod_{j=1}^{d} c_j \right) b \right), \quad (2)$$

then $\mathcal{D}(\mathcal{W}', \mathbf{x}, \mathcal{R}) = \left(\prod_{j=1}^{d} c_j \right) \mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R})$*. In particular, if* $\prod_{j=1}^{d} c_j = 1$*, then for any example distribution* P*, networks* $\mathcal{W}$ *and* $\mathcal{W}'$ *have the same dropout criterion, dropout penalty, and expected loss.*

Note that the re-scaling of the biases at layer j depends not only on the rescaling of the connection weights at layer j, but also the re-scalings at lower layers.

Proof: Choose an input $\mathbf{x}$ and a dropout pattern $\mathcal{R}$. Define $\mathcal{W}'$ as in (2). For each hidden layer j, let $(h_{j1}, ..., h_{jn})$ be the jth hidden layer when applying $\mathcal{W}$ to $\mathbf{x}$ with $\mathcal{R}$, and let $(\tilde{h}_{j1}, ..., \tilde{h}_{jn})$ be the jth hidden layer when applying $\mathcal{W}'$ instead. By induction, for all i, $\tilde{h}_{ji} = \left(\prod_{\ell \leq j} c_\ell \right) h_{ji}$; the key step is that the pre-rectified value used to compute $\tilde{h}_{ji}$ has the same sign as for h_{ji}, since rescaling by c_j preserves the sign. Thus the same units are zeroed out in $\mathcal{W}$ and $\mathcal{W}'$ and $\mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R}) = (\prod_j c_j) \mathcal{D}(\mathcal{W}', \mathbf{x}, \mathcal{R})$. When $\prod_j c_j = 1$, this implies $\mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R}) = \mathcal{D}(\mathcal{W}', \mathbf{x}, \mathcal{R})$. Since this is true for all $\mathbf{x}$ and $\mathcal{R}$, we have $J_{\mathcal{D}}(\mathcal{W}) = J_{\mathcal{D}}(\mathcal{W}')$. Since, similarly, $\mathcal{W}(\mathbf{x}) = \mathcal{W}'(\mathbf{x})$ for all $\mathbf{x}$, so $R(\mathcal{W}) = R(\mathcal{W}')$, the dropout penalities for $\mathcal{W}$ and $\mathcal{W}'$ are also the same. ■

Theorem 5 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. It may be possible to exploit the parameterization equivalence of Theorem 5 in training by using canonical forms for the equivalent networks or switching to an equivalent networks whose gradients have better properties. We leave this question for future work.

3.3 Output scaling with dropout

Scaling the output values of an example distribution P does affect the aversion, but in a very simple and natural way.

Theorem 6 *For any example distribution* P*, if* P' *is obtained from* P *by scaling the outputs of* P *by a positive constant* c*, the dropout aversion of* P' *is* c^2 *times the dropout aversion of* P*.*

Proof: If a network $\mathcal{W}$ minimizes the dropout criterion for P, then the network $\mathcal{W}'$ obtained by scaling up the weights and bias for the output unit by c minimizes the dropout criterion for P', and for any $\mathbf{x}, y$, and dropout pattern $\mathcal{R}$, $(\mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R}) - y)^2 = (\mathcal{D}(\mathcal{W}', \mathbf{x}, \mathcal{R}) - cy)^2/c^2$. ■

3.4 Scaling properties of weight decay

Weight decay is not scale-free. Define the *weight-decay aversion* analogously to the dropout aversion: the weight-decay aversion with P is the maximum, over minimizers $\mathcal{W}^*$ of $J_2(\mathcal{W}^*)$, of $R_P(\mathcal{W}^*) - \inf_{\mathcal{W}} R_P(\mathcal{W})$.

We analyze the L_2 criterion for depth-2 networks in Appendix B, resulting in the following theorem. Our proof shows that, despite the non-convexity of the L_2 criterion, it is still possible to identify a closed form for one of its optimizers.

Theorem 7 *Choose an arbitrary number n of hidden nodes, $\lambda > 0$ and $\mathbf{x} \in \mathbf{R}^K$ other than $(0, 0, ..., 0)$.*

The weight-decay aversion of $P_{(\mathbf{x},1)}$ is $\min(\frac{1}{4}, \frac{\lambda^2}{\mathbf{x} \cdot \mathbf{x}})$.

Theorem 7 shows that, unlike dropout, the weight decay aversion *does* depend on the scaling of the input features.

Furthermore, when $2\lambda > \sqrt{\mathbf{x} \cdot \mathbf{x}}$, the weight-decay criterion for $P_{(\mathbf{x},1)}$ has only a single isolated optimum weight setting[2] – all weights set to zero and bias 1/2 at the output node. This means that weight-decay in 2-layer networks can completely regularize away significant signal in the sample *even when λ is finite*, contrasting starkly with weight-decay's behavior in 1-layer networks.

The "vertical" flexibility to rescale weights between layers enjoyed by dropout (Theorem 5) does not hold for L_2: one can always drive the L_2 penalty to infinity by scaling one layer up by a large enough positive c, even while scaling another down by c. On the other hand, the proof of Theorem 7 shows that the L_2 criterion has an alternative "horizontal" flexibility involving the rescaling of weights across nodes on the hidden layer (under the theorem's assumptions). Lemma 31 shows that at the optimizers each hidden node's contributions to the output are a constant (depending on the input) times their contribution to the the L_2 penalty. Shifting the magnitudes of these contributions between hidden nodes leads to alternative weights that compute the same value and have the same weight decay penalty. This is a more general observation than the permutation symmetry between hidden nodes because any portion of a hidden node's contribution can be shifted to another hidden node.

2. Since the output node puts weight 0 on each hidden node and the biases are unregularized, this optimum actually represents a class of networks differing only in the irrelevant biases at the hidden nodes. One can easily construct other cases when weight-decay has isolated minima in this sense, for example when $n = 2$ and there is equal probability on $\mathbf{x}$ and $-\mathbf{x}$, both with label 1.

4. 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.

We first present our theoretical results for many features (Section 4.1) and few features (Section 4.2), and then discuss the implication of these results in Section 4.3.

4.1 The basic case – many features

In this section, we analyze the simple distribution $P_{(\mathbf{1},1)}$ that assigns probability 1/2 to the example $(0,...,0),0$, and probability 1/2 to the example $(1,...,1),1$. The support of $P_{(\mathbf{1},1)}$ is arguably the simplest monotone function. Nevertheless, we prove that dropout uses negative weights to fit this data.

The key intuition is that optimizing the dropout criterion requires controlling the variance. Negative weights at the hidden nodes can be used to control the variance due to dropout at the input layer. When there are enough hidden nodes this becomes so beneficial that every minimizer of the dropout criterion uses such negative weights.

Theorem 8 *For the standard architecture, if $K > 18$ and n is large enough relative to K and d, every optimizer of the dropout criterion for $P_{(\mathbf{1},1)}$ uses at least one negative weight.*

To prove Theorem 8, we first calculate $J_{\mathcal{D}}(\mathcal{W}_{\text{neg}})$ for a network $\mathcal{W}_{\text{neg}}$ that uses negative weights, and then prove a lower bound greater than this value that holds for all networks using only non-negative weights.

All of the biases in $\mathcal{W}_{\text{neg}}$ are 0.

A key building block in the definition of $\mathcal{W}_{\text{neg}}$ is a block of hidden units that we call the *first-one gadget*. Each such block has K hidden nodes, and takes its input from the K input nodes. The ith hidden node in the block takes the value 1 if the ith input node is 1, and all inputs $x_{i'}$ for $i' < i$ are 0; otherwise it takes the value 0. This can be accomplished with a weight vector $\mathbf{w}$ with $w_{i'} = -1$ for $i' < i$, with $w_i = 1$, and with $w_{i'} = 0$ for $i' > i$. The first hidden layer of $\mathcal{W}_{\text{neg}}$ comprises n/K copies of the first-one gadget.

Informally, this construction removes most of the variance in the number of $1's$ in the input, as recorded in the following lemma.

Lemma 9 *On any input $\mathbf{x} \in \{0,1\}^n$ except $(0,0,...,0)$, the sum of the values on the first hidden layer of $\mathcal{W}_{\text{neg}}$ is exactly n/K.*

The weights into the remaining hidden layers of $\mathcal{W}_{\text{neg}}$ are all 1, and all the weights into the output layer take a value $c \stackrel{\text{def}}{=} \frac{K^2}{2n^{d-1}\left(1+\frac{K}{n}\right)\left(1+\frac{1}{n}\right)^{d-2}}$, chosen to minimize the dropout criterion for the network. The following lemma analyzes $\mathcal{W}_{\text{neg}}$.

Lemma 10 $J_{\mathcal{D}}(\mathcal{W}_{\text{neg}}) = \frac{1}{2}\left(1 - \frac{(1-2^{-K})}{\left(1+\frac{K}{n}\right)\left(1+\frac{1}{n}\right)^{d-2}}\right)$.

When n is large relative to K and d, the $\left(1+\frac{K}{n}\right)\left(1+\frac{1}{n}\right)^{d-2}$ denominator in Lemma 10 approaches 1, so $J_{\mathcal{D}}(\mathcal{W}_{\text{neg}})$ approaches $2^{-K}/2$ in this case. Lemma 16 below gives a larger

lower bound for any network with all positive weights. In the concrete case when $d = 2$ and $n = K^3$, then Lemma 10 implies $J_{\mathcal{D}}(\mathcal{W}_{\text{neg}}) < 1/K^2$.

Proof (of Lemma 10): Consider a computation of $\mathcal{W}_{\text{neg}}(1, 1, ..., 1)$ under dropout and let $\hat{y}$ be the (random) output. Let k_0 be the number of input nodes kept, and, for each $j \geq 2$, let k_j be the number of nodes in the jth hidden layer kept. Call the node in each first-one gadget that computes 1 a *key* node, and if no node in the gadget computes 1 because the input is all dropped, arbitrarily make the gadget's first hidden node the key node. This ensures there is exactly one key node per gadget, and every non-key node computes 0. Let k_1 be the number of kept key nodes on the first hidden layer. If $k_0 = 0$, the output $\hat{y}$ of the network is 0. Otherwise, $\hat{y} = c2^d \prod_{j=1}^{d-1} k_j$.

Note that k_0 is zero with probability 2^{-K}. Whenever $k_0 \geq 1$, k_1 is distributed as $B(n/K, 1/2)$. Each other k_j is distributed as $B(n, 1/2)$, and $k_1, k_2, ..., k_{d-1}$ are independent of one another.

$$\begin{aligned}\mathbf{E}(\hat{y}) &= \mathbf{Pr}(k_0 \geq 1) c 2^d \mathbf{E}[k_1 | k_0 \geq 1] \prod_{j=2}^{d-1} \mathbf{E}[k_j] \\ &= (1 - 2^{-K}) c 2^d \left(\frac{n}{2K}\right) \left(\frac{n}{2}\right)^{d-2} = \frac{2c}{K}(1 - 2^{-K}) n^{d-1}.\end{aligned}$$

Using the value of the second moment of the binomial, we get

$$\begin{aligned}\mathbf{E}(\hat{y}^2) &= \mathbf{E}\left[\left(\mathbb{1}_{k_0 \geq 1} c 2^d \prod_{j=1}^{d-1} k_j\right)^2\right] = 4c^2(1 - 2^{-K}) \left(\frac{n}{K}\right) \left(\frac{n}{K} + 1\right) n^{d-2}(n+1)^{d-2} \\ &= \frac{4c^2(1 - 2^{-K})}{K^2} n^{2(d-1)} \left(1 + \frac{K}{n}\right) \left(1 + \frac{1}{n}\right)^{d-2}.\end{aligned}$$

Thus,

$$\begin{aligned}J_{\mathcal{D}}(\mathcal{W}_{\text{neg}}) &= \frac{1}{2}\left(1 - 2\mathbf{E}(\hat{y}) + \mathbf{E}(\hat{y}^2)\right) \\ &= \frac{1}{2}\left(1 - \frac{4c}{K}(1 - 2^{-K}) n^{d-1} + \frac{4c^2(1 - 2^{-K})}{K^2} n^{2(d-1)} \left(1 + \frac{K}{n}\right) \left(1 + \frac{1}{n}\right)^{d-2}\right) \\ &= \frac{1}{2}\left(1 - \frac{(1 - 2^{-K})}{\left(1 + \frac{K}{n}\right)\left(1 + \frac{1}{n}\right)^{d-2}}\right),\end{aligned}$$

since $c = \frac{K}{2n^{d-1}\left(1+\frac{K}{n}\right)\left(1+\frac{1}{n}\right)^{d-2}}$, completing the proof. ■

Next we prove a lower bound on $J_{\mathcal{D}}$ for networks with nonnegative weights. Let $\mathcal{W}$ be an arbitrary such network.

Our lower bound will use a property of the function computed by $\mathcal{W}$ that we now define.

Definition 11 *A function* $\phi : \mathbf{R}^K \to \mathbf{R}$ *is* supermodular *if for all* $\mathbf{x}, \boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \in \mathbf{R}^K$ *where* $\boldsymbol{\delta}_1, \boldsymbol{\delta}_2 \geq \mathbf{0}$

$$\phi(\mathbf{x}) + \phi(\mathbf{x} + \boldsymbol{\delta}_1 + \boldsymbol{\delta}_2) \geq \phi(\mathbf{x} + \boldsymbol{\delta}_1) + \phi(\mathbf{x} + \boldsymbol{\delta}_2),$$

or equivalently:

$$\phi(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)-\phi(\mathbf{x}+\boldsymbol{\delta}_2)\geq\phi(\mathbf{x}+\boldsymbol{\delta}_1)-\phi(\mathbf{x})$$

The latter indicates that adding $\boldsymbol{\delta}_1$ to the bigger input $\mathbf{x}+\boldsymbol{\delta}_2$ has at least as large an effect as adding it to the smaller input $\mathbf{x}$.

Since $\mathcal{W}$ has all non-negative weights, it computes a supermodular function of its inputs. (This fact may be of independent interest.)

Lemma 12 *If a network has non-negative weights and its activation functions $\sigma(\cdot)$ are convex, continuous, non-decreasing, and differentiable except on a finite set, then the network computes a supermodular function of its input $\mathbf{x}$.*

Proof: We will prove by induction over the layers that, for any unit h in the network, if $h(\mathbf{x})$ is the output of unit h when $\mathbf{x}$ is the input to $\mathcal{W}$, then $h(\cdot)$ is a supermodular function of its input.

The base case holds since each input node h outputs the corresponding component of the input, and $(\mathbf{x}+\boldsymbol{\delta}_1)-\mathbf{x}=(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)-(\mathbf{x}+\boldsymbol{\delta}_2)$.

Now, for the inductive step, let $\mathbf{w}$ be the weight vector for node h, let b be its bias, and $\sigma(\cdot)$ be its activation function. Let $I(\mathbf{x})$, $I(\mathbf{x}+\boldsymbol{\delta}_1)$, $I(\mathbf{x}+\boldsymbol{\delta}_2)$, and $I(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)$ be the inputs to node h when the inputs to the network are $\mathbf{x}$, $\mathbf{x}+\boldsymbol{\delta}_1$, $\mathbf{x}+\boldsymbol{\delta}_2$ and $\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2$ respectively.

By induction, these inputs to node h (componentwise) satisfy

$$I(\mathbf{x}+\boldsymbol{\delta}_1)-I(\mathbf{x})\leq I(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)-I(\mathbf{x}+\boldsymbol{\delta}_2).$$

Therefore, since $\mathbf{w}$, $\boldsymbol{\delta}_1$, and $\boldsymbol{\delta}_2$ are non-negative, the interval $[\mathbf{w}\cdot I(\mathbf{x}+\boldsymbol{\delta}_2)+b, \mathbf{w}\cdot I(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)+b]$ is at least as long and starts at least as high as the interval $[\mathbf{w}\cdot I(\mathbf{x})+b, \mathbf{w}\cdot I(\mathbf{x}+\boldsymbol{\delta}_1)+b]$. Since σ is continuous and differentiable except on a finite set, we have

$$\begin{aligned}
h(I(\mathbf{x}+\boldsymbol{\delta}_1))-h(I(\mathbf{x})) &= \int_{\mathbf{w}\cdot I(\mathbf{x})+b}^{\mathbf{w}\cdot I(\mathbf{x}+\boldsymbol{\delta}_1)+b}\sigma'(z)dz \\
&\leq \int_{\mathbf{w}\cdot I(\mathbf{x}+\boldsymbol{\delta}_2)+b}^{\mathbf{w}\cdot I(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2)+b}\sigma'(z)dz \quad \text{(since } \sigma' \text{ is non-decreasing)} \\
&= h(I(\mathbf{x}+\boldsymbol{\delta}_1+\boldsymbol{\delta}_2))-h(I(\mathbf{x}+\boldsymbol{\delta}_2)).
\end{aligned}$$

■

Definition 13 *Let $\mathbf{r}_0\in\{0,1\}^K$ be the dropout pattern concerning the input layer, and let $\mathcal{R}'$ be the dropout pattern concerning the rest of the network, so that the dropout pattern $\mathcal{R}=(\mathbf{r}_0,\mathcal{R}')$.*

For each $\ell\in\{0,...,K\}$, let $\psi_{\mathcal{W}}(\ell)$ be the average output of $\mathcal{W}$ under dropout when ℓ of the inputs are kept: that is,

$$\psi_{\mathcal{W}}(\ell)=\mathbf{E}\left(\mathcal{D}(\mathcal{W},1^K,(\mathbf{r}_0,\mathcal{R}'))\middle|\sum_j r_{0j}=\ell\right).$$

Lemma 14 *For any* $\ell \in \{1, ..., K-1\}$,

$$\psi_{\mathcal{W}}(\ell+1) - \psi_{\mathcal{W}}(\ell) \geq \psi_{\mathcal{W}}(\ell) - \psi_{\mathcal{W}}(\ell-1).$$

Proof: Generate $\mathbf{u}$, i and j randomly by, first choosing $\mathbf{u}$ uniformly at random from among bit vectors with ℓ ones, then choosing i uniformly from the 0-components of $\mathbf{u}$, and j uniformly from the 1-components of $\mathbf{u}$. By Lemma 12,

$$\mathcal{W}(\mathbf{u}+\mathbf{e}_i) - \mathcal{W}(\mathbf{u}) \geq \mathcal{W}(\mathbf{u}-\mathbf{e}_j+\mathbf{e}_i) - \mathcal{W}(\mathbf{u}-\mathbf{e}_j) \tag{3}$$

always holds. Furthermore, $\mathbf{u}+\mathbf{e}_i$ is uniformly distributed among bit vectors with $\ell+1$ ones, $\mathbf{u}-\mathbf{e}_j$ is uniformly distributed among bit vectors with $\ell-1$ ones, and $\mathbf{u}+\mathbf{e}_i-\mathbf{e}_j$ is uniformly distributed among bit vectors with ℓ ones. This is true for $\mathcal{W}$, but it is also true for any network obtained by dropping out some of the hidden nodes of $\mathcal{W}$. Thus

$$\begin{aligned}
&\psi_{\mathcal{W}}(\ell+1) - \psi_{\mathcal{W}}(\ell) \\
&= \mathbf{E}(\mathcal{D}(\mathcal{W}, 1^K, (\mathbf{r}_0, \mathcal{R}')|\sum_j r_{0j} = \ell+1)) - \mathbf{E}(\mathcal{D}(\mathcal{W}, 1^K, (\mathbf{r}_0, \mathcal{R}')|\sum_j r_{0j} = \ell)) \\
&= \mathbf{E}(\mathcal{D}(\mathcal{W}, \mathbf{r}_0, (1^K, \mathcal{R}')|\sum_j r_{0j} = \ell+1)) - \mathbf{E}(\mathcal{D}(\mathcal{W}, \mathbf{r}_0, (1^K, \mathcal{R}')|\sum_j r_{0j} = \ell)) \\
&\geq \mathbf{E}(\mathcal{D}(\mathcal{W}, \mathbf{r}_0, (1^K, \mathcal{R}')|\sum_j r_{0j} = \ell)) - \mathbf{E}(\mathcal{D}(\mathcal{W}, \mathbf{r}_0, (1^K, \mathcal{R}')|\sum_j r_{0j} = \ell-1)),
\end{aligned}$$

by (3)) and the distributions of $\mathbf{u}+\mathbf{e}_i$, $\mathbf{u}-\mathbf{e}_j+\mathbf{e}_i$ and $\mathbf{u}-\mathbf{e}_j$. Since

$$\begin{aligned}
&\mathbf{E}(\mathcal{D}(\mathcal{W}, \mathbf{r}_0, (1^K, \mathcal{R}')|\sum_j r_{0j} = \ell)) - \mathbf{E}(\mathcal{D}(\mathcal{W}, \mathbf{r}_0, (1^K, \mathcal{R}')|\sum_j r_{0j} = \ell-1)) \\
&= \psi_{\mathcal{W}}(\ell) - \psi_{\mathcal{W}}(\ell-1),
\end{aligned}$$

this completes the proof. ■

We will use the following lower bound on the tail of the binomial. (Many similar lower bounds are known.)

Lemma 15 *If* X *is distributed according to* $\mathrm{Bin}(n, 1/2)$, *then*

$$\mathbf{Pr}(X < n/2 - \sqrt{n}/4) = \mathbf{Pr}(X > n/2 + \sqrt{n}/4) \geq 1/4.$$

Proof: Using the fact that, for any i, $\mathbf{Pr}(X = i) \leq 1/\sqrt{n}$, we get $\mathbf{Pr}(|X - n/2| < \sqrt{n}/4) \leq 1/2$, so $\mathbf{Pr}(X < n/2 - \sqrt{n}/4) \geq 1/4$. ■

Now we are ready for the lower bound on $J_{\mathcal{D}}(\mathcal{W})$.

Lemma 16 *If* $K > 18$ *and the weights in* $\mathcal{W}$ *are non-negative, then* $J_{\mathcal{D}}(\mathcal{W}) \geq \frac{1}{36K}$.

Proof: Assume to the contrary that $J_{\mathcal{D}}(\mathcal{W}) < \frac{1}{36K}$. First, note that $\psi_{\mathcal{W}}(0) \leq \sqrt{\frac{1}{18K}}$, or else the contribution to $J_{\mathcal{D}}(\mathcal{W})$ due to the $(0,0),0$ example is at least $\frac{1}{36K}$. Applying Lemma 15, we have

$$\psi_{\mathcal{W}}(K/2 - \sqrt{K}/4) > 1 - \sqrt{\frac{2}{9K}} \text{ and } \psi_{\mathcal{W}}(K/2 + \sqrt{K}/4) < 1 + \sqrt{\frac{2}{9K}} \tag{4}$$

as otherwise the contribution of one of the tails to $J_{\mathcal{D}}(\mathcal{W})$ will be at least $\frac{1}{36K}$ for the $(1,\ldots,1),1$ example. We will contradict this small variation of $\psi_{\mathcal{W}}(\ell)$ around $K/2$. The bounds on $\psi_{\mathcal{W}}(0)$ and $\psi(K/2-\sqrt{K}/4)$ and Lemma 14 imply that $\psi_{\mathcal{W}}(\ell)$ grows rapidly when ℓ is around $K/2$, in particular:

$$\psi(K/2-\sqrt{K}/4+1)-\psi(K/2-\sqrt{K}/4) \geq \frac{1-\sqrt{\frac{2}{9K}}-\sqrt{\frac{1}{18K}}}{K/2-\sqrt{K}/4} > \frac{1}{\sqrt{9/32}K},$$

since $K > 18$. Now using Lemma 14 repeatedly shows that

$$\psi(K/2+\sqrt{K}/4)-\psi(K/2-\sqrt{K}/4) > \frac{\sqrt{K}}{2} \times \frac{1}{\sqrt{9/32}K} = 2\sqrt{\frac{2}{9K}},$$

which contradicts (4), completing the proof. ■

Putting together Lemmas 10 and 16 immediately proves Theorem 8, since for $K > 18$ and large enough n, the criterion for $\mathcal{W}_{\text{neg}}$ must be less than the criterion for any network with all non-negative weights.

4.2 The case when $K = 2$

Theorem 8 uses the assumption that $K > 18$ and n is large enough; is the lower bound on K really necessary? Here we show that it is not, by treating the case that $K = 2$.

Theorem 17 *For the standard architecture, if $K = 2$, for any fixed d and large enough n, every optimizer of the dropout criterion for $P_{(1,1),1}$ uses negative weights.*

Proof: Define $\mathcal{W}_{\text{neg}}$ as in the proof of Lemma 10, except that the output layer has a bias of $1/5$.

We claim that

$$\lim_{n\to\infty} J_{\mathcal{D}}(\mathcal{W}_{\text{neg}}) = 1/10. \tag{5}$$

To see this, consider the joint input/label distribution under dropout:

$$\begin{aligned}
\mathbf{Pr}((0,0),0) &= 1/2\\
\mathbf{Pr}((0,0),1) &= 1/8\\
\mathbf{Pr}((2,0),1) &= 1/8\\
\mathbf{Pr}((0,2),1) &= 1/8\\
\mathbf{Pr}((2,2),1) &= 1/8.
\end{aligned}$$

Due to the bias of $1/5$ on the output, $\mathcal{W}_{\text{neg}}(0,0) = 1/5$. Thus, contribution to $J_{\mathcal{D}}$ from examples with $\mathbf{x} = (0,0)$ in this joint distribution is $1/2 \times (1/5)^2 + 1/8 \times (4/5)^2 = 1/10$.

Now, choose $\mathbf{x} \neq (0,0)$. If, after dropout, the input is $\mathbf{x}$, each node in the hidden layer closest to the input computes 1. Arguing exactly as in the proof of Lemma 10, in such cases,

$$\mathbf{E}((\hat{y}-1)^2) = 1 - \frac{1}{\left(1+\frac{2}{n}\right)\left(1+\frac{1}{n}\right)^{d-2}}.$$

This proves (5).

Now, let $\mathcal{W}$ be an arbitrary network with non-negative weights.

For our distribution,

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

Let $V_{00} = \mathbf{E}(\mathcal{W}(0,0)), V_{22} = \mathbf{E}(\mathcal{W}(2,2)), V_{20} = \mathbf{E}(\mathcal{W}(2,0)), V_{02} = \mathbf{E}(\mathcal{W}(0,2))$ where the expectations are taken with respect to the dropout patterns at the hidden nodes (with no dropout at the inputs). Since each dropout pattern over the hidden nodes defines a particular network, and Lemma 12 holds for all of them, the relationships also hold for the expectations, so

$$V_{22} \geq V_{20} + V_{02} - V_{00}.$$

Using this V notation, handling the dropout at the input explicitly, and the bias-variance decomposition keeping just the bias terms we get:

$$\begin{aligned} J_{\mathcal{D}}(\mathcal{W}) &\geq \frac{(V_{00}-0)^2 + \left((V_{00}-1)^2 + (V_{22}-1)^2 + (V_{20}-1)^2 + (V_{02}-1)^2\right)/4}{2} \\ 8J_{\mathcal{D}}(\mathcal{W}) &\geq 4(V_{00}-0)^2 + (V_{00}-1)^2 + (V_{22}-1)^2 + (V_{20}-1)^2 + (V_{02}-1)^2. \end{aligned}$$

We will continue lower bounding the RHS. We can re-write V_{22} as $V_{20} + V_{02} - V_{00} + \epsilon$ where $\epsilon \geq 0$. This is convex and symmetric in V_{02} and V_{20} so they both take the same value at the minimizer of the RHS, so we proceed using V_{20} for this common minimizing value.

$$8J_{\mathcal{D}}(\mathcal{W}) \geq (2V_{20} - V_{00} + \epsilon - 1)^2 + 2(V_{20}-1)^2 + (V_{00}-1)^2 + 4V_{00}^2.$$

Differentiating with respect to V_{20}, we see that the RHS is minimized when $V_{20} = (2 + V_{00} - \epsilon)/3$, giving

$$8J_{\mathcal{D}}(\mathcal{W}) \geq \frac{(V_{00}-\epsilon-1)^2}{3} + (V_{00}-1)^2 + 4V_{00}^2.$$

If $V_{00} \geq 1$, then $J_{\mathcal{D}} \geq 1/2$ (just from the (0,0),0 example) and when $V_{00} < 1$ the RHS of the above is minimized for non-negative ϵ when $\epsilon = 0$. Using this substitution, the minimizing value of V_{00} is $1/4$ giving

$$\begin{aligned} 8J_{\mathcal{D}}(\mathcal{W}) &\geq 1 \\ J_{\mathcal{D}}(\mathcal{W}) &\geq 1/8. \end{aligned}$$

Combining this with (5) completes the proof. ■

4.3 More general distributions and implications

In the previous sub-section we analyzed the distribution P over the 2-feature examples $(1,1),1$ and $(0,0),0$. However, these two examples can be embedded in a larger feature space by using any fixed vector of additional feature values, creating, for instance, a distribution over $(\underline{0,1},0,0,\underline{0,1/2,1}),0$ and $(\underline{0,1},1,1,\underline{0,1/2,1}),1$ (with the additional features underlined) The results of Section 4.2 still apply to distribution over these extended examples after defining $\mathcal{W}_{\text{neg}}$ network to have zero weight on the additional features, and noticing that any weight on the additional features in the positive-weight network $\mathcal{W}$ can be simulated using the biases at the hidden nodes.

It is particularly interesting when the additional features all take the value 0, we call these *zero-embeddings*. Every network $\mathcal{W}$ with non-negative weights has $J_{\mathcal{D}}(\mathcal{W}) \geq 1/8$ on each of these *zero-embeddings* of P. On the other hand, a single $\mathcal{W}_{\text{neg}}$ network with n/K copies of the K-input first-one gadget has $J_{\mathcal{D}}(\mathcal{W}_{\text{neg}}) \approx 1/10$ simultaneously for all of these zero-embeddings of P (when $n >> K$).

Any source distribution over $\{0,1\}^K \times \{0,1\}$ that puts probability $1/2$ on the the $\mathbf{0},0$ example and distributes the other $1/2$ probability over examples where exactly two inputs are one is a mixture of zero-embeddings of P, and thus $J_{\mathcal{D}}(\mathcal{W}) \geq 1/8$ while $J_{\mathcal{D}}(\mathcal{W}_{\text{neg}}) \approx 1/10$ for this mixture and optimizing the dropout criterion requires negative weights.

In our analysis the negative weights used by dropout are counterintuitive for fitting monotone behavior, but are needed to control the variance due to dropout. This suggests that dropout may be less effective when layers with sparse activation patterns are fed into wider layers, as dropout training can hijack part of the expressiveness of the wide layer to control the artificial variance due to dropout rather than fitting the underlying patterns in the data.

5. Properties of the dropout penalty

This section examines some properties of the dropout penalty applied to deep networks with ReLUs and the quadratic loss, which are often different than the case of networks without hidden layers.

5.1 Growth of the dropout penalty as a function of d

Weight-decay penalizes large weights, while Theorem 5 shows that compensating rescaling of the weights does not affect the dropout penalty or criterion. On the other hand, dropout can be more sensitive to the calculation of large outputs than weight decay, and large outputs can be produced in deep networks using only small weights. We make this observation concrete by exhibiting a family of networks where the depth and desired output are linked while the size of individual weights remains constant. For this family, the dropout penalty grows exponentially in the depth d (as opposed to linearly for weight-decay), suggesting that dropout training is less willing to fit the data in this kind of situation.

Theorem 18 *If* $\mathbf{x} = (1,1,...,1)$ *and* $0 \leq y \leq Kn^{d-1}$*, for* $P_{\mathbf{x},y}$ *there are weights* $\mathcal{W}$ *for the standard architecture with* $R(\mathcal{W}) = 0$ *such that (a) every weight has magnitude at most one, but (b)* $J_{\mathcal{D}}(\mathcal{W}) \geq \frac{y^2}{K}$*, whereas (c)* $J_2(\mathcal{W}) \leq \frac{\lambda y^{2/d}}{2}(Kn + n^2(d-2) + n)$*.*

Proof: Let $\mathcal{W}$ be the network whose weights are all $c = \frac{y^{1/d}}{K^{1/d}n^{(d-1)/d}}$ and biases are all 0, so that the L_2 penalty is the number of weights times $\lambda c^2/2$. It is a simple induction to show that, for these weights and input $(1, 1, ..., 1)$, the value computed at each hidden node on level j is $c^j K n^{j-1}$, so the the network outputs $c^d K n^{d-1}$, and has zero square loss (since $\mathcal{W}(\mathbf{x}) = c^d K n^{d-1} = y$).

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. For each j, let H_j be the value computed by the units in the jth hidden layer.

If k_0 is the number of input nodes kept under dropout, and, for each $j \in \{1, ..., d-1\}$, k_j is the number of hidden nodes kept in layer j, a straightforward induction shows that, for all ℓ, we have $H_\ell = (2c)^\ell \prod_{j=0}^{\ell-1} k_j$, so that the output $\hat{y}$ of the network is $(2c)^d \prod_{j=0}^{d-1} k_j$.

Using a bias-variance decomposition,

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

Since each k_j is binomially distributed, and $k_0, ..., k_{d-1}$ are independent, we have

$$\mathbf{E}(\hat{y}) = (2c)^d (K/2)(n/2)^{d-1} = c^d K n^{d-1} = y,$$

so

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

Since

$$\mathbf{E}(\hat{y}^2) = (2c)^{2d}(K(K+1)/4)(n(n+1)/4)^{d-1} = y^2(1 + 1/K)(1 + 1/n)^{d-1},$$

we have

$$\mathbf{Var}(\hat{y}) = \mathbf{E}(\hat{y}^2) - \mathbf{E}(\hat{y})^2 = y^2((1 + 1/K)(1 + 1/n)^{d-1} - 1) \geq y^2/K,$$

completing the proof. ■

Theorem 18 shows that if $y = \exp(\Theta(d))$, the dropout penalty grows exponentially in d, whereas the L_2 penalty 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 $((1, 1, \ldots, 1), y)$ example has a term that grows exponentially in the depth.

5.2 A necessary condition for negative dropout penalty

Section 2 contains an example where the dropout penalty is negative. The following theorem provides a necessary condition.

Theorem 19 *The dropout penalty can be negative. For all example distributions, a necessary condition for this in rectified linear networks is that either a weight, input, or bias is negative.*

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

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

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. ■

5.3 Multi-layer dropout penalty does depend on labels

In contrast with its behavior 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 L_1 penalty.

Theorem 20 *There are joint distributions P and Q, and weights $\mathcal{W}$ such that, for all dropout probabilities $q \in (0, 1)$, (a) the marginals of P and Q on the input variables are equal, but (b) the dropout penalties of $\mathcal{W}$ with respect to P and Q are different.*

We will prove Theorem 20 by describing a general, somewhat technical, condition that implies that P and Q are witnesses to Theorem 20.

For each input $\mathbf{x}$ and dropout pattern $\mathcal{R}$, let $\underline{\mathcal{H}(\mathcal{W}, \mathbf{x}, \mathcal{R})}$ be the values presented to the output node with dropout. As before, let $\underline{\mathbf{w}} \in \mathbf{R}^n$ be those weights of $\mathcal{W}$ on connections directly into the output node and let b be the bias at the output node. Let $\underline{\mathbf{r}} \in \{0, 1\}^n$ be the indicator variables for whether the various nodes connecting to the output node are kept.

Proof (of Theorem 20): Suppose that P is concentrated on a single $(\mathbf{x}, y)$ pair. We will then get Q by modifying y.

Let $\mathbf{h}$ be the values coming into the output node in the non-dropped out network. Therefore the output of the non-dropout network is $\mathbf{w} \cdot \mathbf{h} + b$ while the output of the network with dropout is $\mathbf{w} \cdot \mathcal{H}(\mathcal{W}, \mathbf{x}, \mathcal{R}) + b$. We now examine the dropout penalty, which is the expected dropout loss minus the non-dropout loss. We will use δ as a shorthand for $\mathbf{w} \cdot (\mathcal{H}(\mathcal{W}, \mathbf{x}, \mathcal{R}) - \mathbf{h})$.

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

which depends on the label y unless $\mathbf{E}(\delta) = 0$.

Typically $\mathbf{E}(\delta) \neq 0$. To prove the theorem, consider the case where

- there are two inputs and one hidden layer,
- $\mathbf{x} = (1, -2)$,
- all weights in the network are 1, and
- all biases are 0.

Without dropout $\mathbf{h} = (0, 0)$, but with dropout the hidden nodes are never negative and compute positive values when only the negative input is dropped, so that the expectation of δ is positive. ■

6. Experiments

To complement our theoretical results we performed two sets of experiments. The first set tests the scale dependence of dropout and weight decay, while the the second set examines its promotion of negative weights even when learning monotone functions. The code is accessible at

`https://www.dropbox.com/sh/6s2lcfrq17zshmp/AAAQ06uDa4gOAuAnw2MAghEMa?dl=0`

6.1 Scale (in)sensitivity

The scale dependence simulations were implemented using Torch. Our experiment used the standard architecture with $K = 5$ inputs, depth $d = 2$, $n = 5$ hidden nodes. We used stochastic gradient using the `optim` package for Torch, with learning rate $\frac{0.01}{1+0.00001t}$ and momentum of 0.5, and a maximum of 100000 iterations.

We performed 10 sets of training runs. In each run:

- Ten training examples were generated uniformly at random from $[-1, 1]^K$.
- Target outputs were assigned using $y = \prod_i \text{sign}(x_i)$.
- Five training sets $S_1, ..., S_5$ 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.
- The weights of a network $\mathcal{W}_{\text{init}}$ were initialized using the default initialization from Torch.
- For each S_i:
  - $\mathcal{W}_{\text{init}}$ was cloned three times to produce $\mathcal{W}_D$, $\mathcal{W}_2$ and $\mathcal{W}_{\text{none}}$ with identical starting parameters.
  - $\mathcal{W}_D$ was trained with dropout probability 1/2 and no weight decay.
  - $\mathcal{W}_2$ was trained with weight decay with $\lambda = 1/2$ and no dropout.
  - $\mathcal{W}_{\text{none}}$ was trained without any regularization.

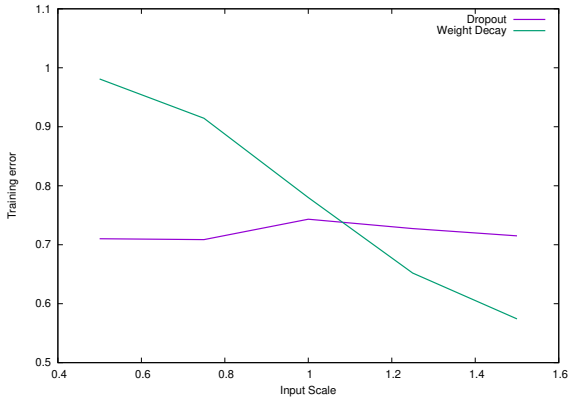

Figure 2: Training error as a function of the scale of the inputs for Dropout and Weight Decay in the experiment of Section 3.

The average training losses of $\mathcal{W}_D$ and $\mathcal{W}_2$, over the 10 runs, are shown in Figure 2. (The average training loss of $\mathcal{W}_{\text{none}}$ was less than 0.05 at all scales.)

The theoretical insensitivity of dropout to the scale of the inputs described in Theorem 4 is also seen here, along with the contrast with weight decay analyzed in Theorem 7.

The scale of the inputs also affects the dynamics of stochastic gradient descent. With very small inputs, convergence is very slow, and with very large inputs, SGD is unstable. The effects of the scale of the inputs on inductive bias analyzed in this paper are visible at the scales where optimization can be done effectively.

6.2 Negative Weights

Here we demonstrate dropout's propensity to use negative weights even when the function to be learned is monotonic. These experiments were implemented with Keras on top of TensorFlow and using SGD optimization with a learning rate of 0.005. Weight decay learning used a parameter of 0.05, and dropout training used a dropout rate of 0.5. We used the standard architecture with six inputs, 12 hidden nodes, and one output. The training set consists of six inputs as follows:

inputs	label	multiplicity
(0,0,0,0,0,0)	0	$\times 3$
(1,1,0,0,0,0)	1	$\times 1$
(0,0,1,1,0,0)	1	$\times 1$
(0,0,0,0,1,1)	1	$\times 1$

A natural network for this data uses all weights set to the same value $w = \sqrt{1/24}$ (with the biases set to 0). This network computes the correct labels for all of the training inputs. We ran each of dropout and weight-decay for 10,000 epochs from 10 different initial weight settings created by by sampling the weights independently from the uniform distribution

Table 1: Summary of weights resulting from noisy "all same" initialization

weights from	num neg	below −1.0	−1.0 to −0.1	−0.1 to −0.001	−0.001 to 0	0 to 0.001	0.001 to 0.1	0.1 to 1.0	above 1.0
initial wts	76	0	0	75	1	1	166	597	0
weight-decay	7	0	0	0	7	1	276	556	0
dropout	82	0	70	12	0	0	146	612	0

Table 2: Summary of weights resulting from noisy "first-one" initialization

weights from	num neg	below −1.0	−1.0 to −0.1	−0.1 to −0.001	−0.001 to 0	0 to 0.001	0.001 to 0.1	0.1 to 1.0	above 1.0
initial wts	204	90	80	34	180	1	47	232	176
weight-decay	111	0	0	57	68	68	298	349	0
dropout	303	91	93	119	5	0	168	189	175

over $[-0.2w, 2.2w]$. This randomization gives a one-in-twelve chance of changing the weight's sign.

Table 1 summarizes the initial weights, weights learned by weight-decay, and weights learned by dropout aggregated over the 10 initializations. This table shows that dropout not only (slightly) increases the number of negative weights, but also tends to greatly increase their magnitude. As expected, weight-decay eliminates almost all negative weights, and the few remaining negative weights have very small magnitudes.

We also ran a similar experiment based on the "first-one" gadget construction of Section 4.1. The hidden nodes are organized into six groups of two, with each group implementing three first-one gadgets – one for each of the input "pairs". Now the uncorrupted weights into the hidden layer are 0, 1, or -1, and those at the output node are $1/6$ so that the network computes the labels in the training set. As before, 10 noisy initial weight settings are chosen by selecting each weight uniformly from $[-0.2w, 2.2w]$, where w is the corresponding uncorrupted weight.

Table 2 summarizes the initial weights, weights learned by weight-decay, and weights learned by dropout aggregated over the 10 initializations. From this initialization, dropout tends to preserve the magnitudes of the negative weights. Although dropout increases the number of small-magnitude negative weights, much of this increase can be explained by half of the 180 weights initialized to exactly zero becoming negative. Again, weight-decay dramatically decreases both the number of the negative weights and their magnitudes.

7. 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 (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 coadaptation 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. We prove that this happens even in very simple situations where nothing in the training data justifies negative weights (Theorem 8). This indicates that the relationship between dropout and co-adaption 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 3 we show 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. Further exploration of these scale invariance properties is warranted to see if they are a contributor to dropout's empirical success or can be exploited to facilitate training. While contrasting dropout to weight-decay in simple situations, we found that a degenerate all-zero network results (Theorem 7) when the L_2 regularization parameter is above a threshold. This is in dramatic contrast to our previous intuition from the 1-layer case.

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 5 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 L_2 regularization.

This paper uncovers a number of dropout's interesting fundamental properties using formal analysis of simple cases. However, the effects of using dropout training in deep networks are subtle and complex, and we hope that this paper lays a foundation to promote further formal analysis of dropout's properties and behavior.

Acknowledgments

We are very grateful to Peter Bartlett, Seshadhri Comandur, and anonymous reviewers for valuable communications. We would like to acknowledge support for this project from the National Science Foundation (NSF grant IIS-9988642) and the Multidisciplinary Research Program of the Department of Defense (MURI N00014-00-1-0637).

Appendix A. Table of Notation

Notation	Meaning
$\mathbb{1}_{\text{set}}$	indicator function for "set"
$(\mathbf{x}, y)$	an example with feature vector $\mathbf{x}$ and label y
$\sigma(\cdot)$	the rectified linear unit computing $\max(0, \cdot)$
$\mathcal{W}$	an arbitrary weight setting for the network
w, v	specific weights, often subscripted
$\mathcal{W}(\mathbf{x})$	the output value produced by weight setting $\mathcal{W}$ on input $\mathbf{x}$
P	an arbitrary source distribution over $(\mathbf{x}, y)$ pairs
$P_{\mathbf{x},y}$	the source distribution concentrated on the single example $(\mathbf{x}, y)$
$R_P(\mathcal{W})$	the risk (expected square loss) of $\mathcal{W}$ under source P
q, p	probabilities that a node is dropped out (q) or kept (p)
$\mathcal{R}$	a dropout pattern, indicates the kept nodes
$\mathbf{r}, \mathbf{s}$	dropout patterns on subsets of the nodes
$\mathcal{D}(\mathcal{W}, \mathbf{x}, \mathcal{R})$	Output with weights $\mathcal{W}$, input $\mathbf{x}$, and dropout pattern $\mathcal{R}$
$J_D(\mathcal{W})$	the dropout criterion
$J_2(\mathcal{W})$	the L_2 criterion
λ	the L_2 regularization strength parameter
$\mathcal{W}_D$	an optimizer of the dropout criterion
$\mathcal{W}_{L_2}$	an optimizer of the L_2 criterion
n, d	the network width and depth
K	the number of input nodes

Appendix B. Proof of Theorem 7

Here we prove Theorem 7, showing that the weight-decay aversion depends on the values of the inputs and the number of input nodes K. Furthermore, unlike the single-layer case, the L_2 regularization strength has a threshold where the minimizer of the L_2 criterion degenerates to the all-zero network.

We will focus on the standard architecture with depth $d = 2$. Recall that we are analyzing the distribution $P_{(\mathbf{x},1)}$ that assigns probability $1/2$ to $(\mathbf{x}, 1)$ and probability $1/2$ to $(\mathbf{0}, 0)$.

Also, recall that, for $P_{(\mathbf{x},1)}$, the weight-decay aversion is the maximum risk incurred by a minimizer of J_2.

The proof of Theorem 7 involves a series of lemmas, in which we show that there is a minimizer of J_2 of a special form, and then relate any hidden node's effect on the output to the regularization penalty on the weights in and out of that node. This will allow us to treat optimizing the L_2 criterion as a one-dimensional problem, whose solution yields the theorem.

For some minimizer $\mathcal{W}_{L_2}$ of J_2, let $\underline{\mathbf{v}_j^*}$ denote the vector of weights into hidden node j and $\underline{w_j^*}$ denote the weight from j to the output node. Let $\underline{a_j^*}$ be the bias for hidden node j and let $\underline{b^*}$ be the bias for the output node. Let $\underline{h_j}$ be the function computed by hidden node j.

Lemma 21 *We may assume without loss of generality that for each hidden node j, there is an input $\tilde{\mathbf{x}} \in \{\mathbf{0}, \mathbf{x}\}$ such that $h_j(\tilde{\mathbf{x}}) = 0$.*

Proof: Suppose neither of $h_j(\mathbf{0})$ or $h_j(\mathbf{x})$ was 0. If $w_j^* = 0$, then replacing both with 0 does not affect the output of $\mathcal{W}_{L_2}$, and does not increase the penalty. If $w_j^* \neq 0$, then subtracting $\min\{h_j(\mathbf{0}), h_j(\mathbf{x})\}$ from a_j^* and adding $w_j^* \cdot \min\{h_j(\mathbf{0}), h_j(\mathbf{x})\}$ to b does not affect the output of $\mathcal{W}_{L_2}$ or the penalty, but, after this transformation, $\min\{h_j(\mathbf{0}), h_j(\mathbf{x})\} = 0$. ■

Lemma 22 *We may assume without loss of generality that for each hidden node j, we have $|\mathbf{v}_j^* \cdot \mathbf{x}| = \max\{h_j(\mathbf{0}), h_j(\mathbf{x})\}$.*

Proof: If $h_j(\mathbf{x}) \geq h_j(\mathbf{0}) = 0$, then bias $a_j^* = 0$, and $h_j(\mathbf{x}) = \mathbf{v}_j^* \cdot \mathbf{x}$.

If $h_j(\mathbf{0}) > h_j(\mathbf{x}) = 0$. Then, $a_j^* > 0$, and $\mathbf{v}_j^* \cdot \mathbf{x} \leq -a_j^*$. If needed, the magnitude of $\mathbf{v}_j^*$ can be decreased to make $\mathbf{v}_j^* \cdot \mathbf{x} = -a_j^*$. This decrease does not affect $\mathcal{W}_{L_2}(\mathbf{x})$ or $\mathcal{W}_{L_2}(\mathbf{0})$, and can only reduce the L_2 penalty. ■

Lemma 23 *We may assume without loss of generality that for each hidden node j, we have $h_j(\mathbf{0}) = 0$.*

Proof: Suppose $h_j(\mathbf{0}) > 0$. Let z be this old value of $h_j(\mathbf{0})$. Then $h_j(\mathbf{x}) = 0$ and $z = h_j(\mathbf{0}) = -\mathbf{v}_j^* \cdot \mathbf{x}$. If we negate $\mathbf{v}^*$ and set $a_j = 0$, then Lemma 22 implies that we swap the values of $h_j(\mathbf{x})$ and $h_j(\mathbf{0})$.

Then, by adding zw_j^* to b^* and negating w_j^*, we correct for this swap at the output node and do not affect the function computed by $\mathcal{W}_{L_2}$ or the penalty. ■

Note that Lemma 23 implies that, without loss of generality, $a_1^* = ... = a_K^* = 0$. We continue with that assumption.

Lemma 24 *For all j, $\mathbf{v}_j^* \cdot \mathbf{x} \geq 0$.*

Proof: Since $a_j^* = 0$, if $\mathbf{v}_j^* \cdot \mathbf{x} < 0$, we could make $\mathcal{W}_{L_2}$ compute the same function with a smaller penalty by replacing $\mathbf{v}_j^*$ with $\mathbf{0}$. ■

Lemma 24 implies that the optimal $\mathcal{W}_{L_2}$ computes the linear function:

$$\mathcal{W}_{L_2}(\tilde{\mathbf{x}}) = (\mathbf{w}^*)^T V^* \tilde{\mathbf{x}} + b^*,$$

when $\tilde{\mathbf{x}}$ is a non-negative multiple of $\mathbf{x}$. Later we will call $(\mathbf{w}^*)^T V^* \tilde{\mathbf{x}}$ the activation at the output node.

Lemma 25 $b^* = \frac{1-(\mathbf{w}^*)^T V^* \mathbf{x}}{2}$.

Proof: Minimize J_2 (wrt distribution $P_{(\mathbf{x},1)}$) as a function of b using Calculus. ■

Now, for $\tilde{\mathbf{x}}$ a non-negative multiple of $\mathbf{x}$,

$$\mathcal{W}_{L_2}(\tilde{\mathbf{x}}) = \frac{1}{2} + (\mathbf{w}^*)^T V^* (\tilde{\mathbf{x}} - \mathbf{x}/2) \tag{6}$$

which immediately implies

$$\mathcal{W}_{L_2}(\mathbf{0}) = 1 - \mathcal{W}_{L_2}(\mathbf{x}). \tag{7}$$

Lemma 26 *Each $\mathbf{v}_j^*$ is a (positive) rescaling of $\mathbf{x}$.*

Proof: Projecting $\mathbf{v}_j^*$ onto the span of $\mathbf{x}$ does not affect h_j, and cannot increase the penalty. ■

Lemma 27 $\mathcal{W}_{L_2}(\mathbf{x}) \leq 1$.

Proof: By (7), if $\mathcal{W}_{L_2}(\mathbf{x}) > 1$ then $\mathcal{W}_{L_2}(\mathbf{0}) < 0$ and the loss and the penalty would both be reduced by scaling down $\mathbf{w}^*$. ■

Lemma 28 *$\mathcal{W}_{L_2}$ maximizes $\mathcal{W}(\mathbf{x})$ over those weight vectors $\mathcal{W}$ that have the same penalty as $\mathcal{W}_{L_2}$ and compute a function of the form $\mathcal{W}(\tilde{x}) = \frac{1}{2} + (\mathbf{w})^T V(\tilde{\mathbf{x}} - \mathbf{x}/2)$ (that is, Equation 6).*

Proof: Let $\mathcal{W}$ maximize $\mathcal{W}(\mathbf{x})$ over the networks considered. If $\mathcal{W}_{L_2}(\mathbf{x}) < \mathcal{W}(\mathbf{x}) \leq 1$, then $\mathcal{W}$ would have the same penalty as $\mathcal{W}_{L_2}$ but smaller error, contradicting the optimality of $\mathcal{W}_{L_2}$.

If $\mathcal{W}(\mathbf{x}) > 1$, then the network $\widetilde{\mathcal{W}}$ obtained by scaling down the weights in the output layer so that $\widetilde{\mathcal{W}}(\mathbf{x}) = 1$ has a smaller penalty than $\mathcal{W}_{L_2}$ and smaller error, again contradicting $\mathcal{W}_{L_2}$'s optimality. ■

Informally, Lemmas 27 and 28 engender a view of the learner straining against the yolk of the L_2 penalty to produce a large enough output on $\mathbf{x}$. This motivates us to ask how large $\mathcal{W}(\mathbf{x})$ can be, for a given value of $||\mathcal{W}||_2^2$ (recall that Lemma 24 allows us to assume that the biases at the hidden nodes are all 0).

Definition 29 *For each hidden node j, let $\underline{\alpha_j}$ be the constant such $\mathbf{v}_j^* = \alpha_j \mathbf{x}$, so that $h_j(\mathbf{x}) = \alpha_j \mathbf{x} \cdot \mathbf{x}$.*

Recall that the *activation* at the output node on input $\mathbf{x}$ is the weighted sum of the hidden-node outputs, $(\mathbf{w}^*)^T V^* \mathbf{x}$.

Definition 30 *The contribution to the activation at the output due to hidden node j is*

$$w_j^* h_j(\mathbf{x}) = w_j^* \alpha_j \mathbf{x} \cdot \mathbf{x}$$

and the contribution to the L_2 penalty from these weights is

$$\frac{\lambda}{2}\left((w_j^*)^2 + \alpha_j^2 \mathbf{x} \cdot \mathbf{x}\right).$$

We now bound the contribution to the activation in terms of the contribution to the L_2 penalty. Note that as the L_2 "budget" increases, so does the the maximum possible contribution to the output node's activation.

Lemma 31 *If B is hidden node j's weight-decay contribution, $(w_j^*)^2 + \alpha_j^2 \mathbf{x}\cdot\mathbf{x}$, then hidden node j's contribution to the output node's activation is maximized when $w_j^* = \sqrt{\frac{B}{2}}$ and $\alpha_j = \sqrt{\frac{B}{2\mathbf{x}\cdot\mathbf{x}}}$, where it achieves the value $B\sqrt{\mathbf{x}\cdot\mathbf{x}}/2$*

Proof: Since $\alpha_j^2 \mathbf{x}\cdot\mathbf{x} + (w_j^*)^2 = B$, we have $w_j^* = \sqrt{B - \alpha_j^2 \mathbf{x}\cdot\mathbf{x}}$, so the contribution to the activation can be re-written as $\alpha_j \mathbf{x}\cdot\mathbf{x}\sqrt{B - \alpha_j^2 \mathbf{x}\cdot\mathbf{x}}$. Taking the derivative with respect to α_j, and solving, we get $\alpha_j = \pm\sqrt{\frac{B}{2\mathbf{x}\cdot\mathbf{x}}}$ and we want the positive solution (otherwise the node outputs 0). When $\alpha_j = \sqrt{\frac{B}{2\mathbf{x}\cdot\mathbf{x}}}$ we have $w_j^* = \sqrt{\frac{B}{2}}$ and thus the node's maximum contribution to the activation is

$$\sqrt{\frac{B}{2}}\sqrt{\frac{B}{2\mathbf{x}\cdot\mathbf{x}}}\,(\mathbf{x}\cdot\mathbf{x}) = \frac{B\sqrt{\mathbf{x}\cdot\mathbf{x}}}{2}.$$

■

Lemma 32 *The minimum sum-squared weights for a network $\mathcal{W}$ (without biases at the hidden nodes) that has an activation A at the output node on input $\mathbf{x}$ is $\frac{2A}{\sqrt{\mathbf{x}\cdot\mathbf{x}}}$.*

Proof: When maximized, the contribution of each hidden node to the activation at the output is $\sqrt{\mathbf{x}\cdot\mathbf{x}}/2$ times the hidden node's contribution to the sum of squared-weights. Since each weight in $\mathcal{W}$ is used in exactly one hidden node's contribution to the output node's activation, this completes the proof. ■

Note that this bound is independent of n, the number of hidden units, but does depend on the input $\mathbf{x}$.

Proof (of Theorem 7): Let $A \geq 0$ be the activation at the output node for $\mathcal{W}_{L_2}$ on input $\mathbf{x}$. From Lemma 25 we get that $b^* = \frac{1-A}{2}$. Combining Lemmas 27 , 28 and 32, we can re-write the J_2 criterion for $\mathcal{W}_{L_2}$ and distribution $P_{(\mathbf{x},1)}$ in terms of A as follows.

$$\begin{aligned} J_2(\mathcal{W}_{L_2}) &= \frac{1}{2}\left(b^* - 0\right)^2 + \frac{1}{2}\left(\mathcal{W}_{L_2}(\mathbf{x}) - 1\right)^2 + \frac{\lambda}{2}||\mathcal{W}||_2^2 \\ &= \frac{1}{2}\left(\left(\frac{1-A}{2}\right)^2 + \left(\frac{1+A}{2} - 1\right)^2 + \lambda\frac{2A}{\sqrt{\mathbf{x}\cdot\mathbf{x}}}\right). \end{aligned} \tag{8}$$

Differentiating with respect to A, we see that the criterion is minimized when

$$\begin{aligned} A &= 1 - \frac{2\lambda}{\sqrt{\mathbf{x}\cdot\mathbf{x}}} && \text{when } \frac{2\lambda}{\sqrt{\mathbf{x}\cdot\mathbf{x}}} \leq 1 \\ A &= 0 && \text{when } \frac{2\lambda}{\sqrt{\mathbf{x}\cdot\mathbf{x}}} > 1 \end{aligned}$$

since we assumed $A \geq 0$; when $A = 0$ then $\mathcal{W}_{L_2}$ has all zero weights with a bias of $1/2$ at the output.

The risk part of (8) simplifies to

$$\frac{(1-A)^2}{4} = \frac{\lambda^2}{\mathbf{x}\cdot\mathbf{x}},$$

so the overall the risk of $\mathcal{W}_{L_2}$, which minimizes J_2, is $\min\left\{\frac{1}{4}, \frac{\lambda^2}{\mathbf{x}\cdot\mathbf{x}}\right\}$. ■

References

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

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

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

P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe. Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156, 2006.

L. Breiman. Some infinity theory for predictor ensembles. *Annals of Statistics*, 32(1):1–11, 2004.

Caffe. Caffe, 2016. http://caffe.berkeleyvision.edu.

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

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

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

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.

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

Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1019–1027, 2016.

Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. Maxout networks. In *ICML*, pages 1319–1327, 2013.

Mohammad Havaei, Axel Davy, David Warde-Farley, Antoine Biard, Aaron Courville, Yoshua Bengio, Chris Pal, Pierre-Marc Jodoin, and Hugo Larochelle. Brain tumor segmentation with deep neural networks. *Medical image analysis*, 35:18–31, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

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

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

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

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

P. M. Long and R. A. Servedio. Random classification noise defeats all convex potential boosters. *Machine Learning*, 78(3):287–304, 2010.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *COLT*, pages 1376–1401, 2015.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

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

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.

TensorFlow. Tensorflow, 2016. https://www.tensorflow.org.

Torch. Torch, 2016. http://torch.ch.

T. Van Erven, W. Kotowski, and M. K. Warmuth. Follow the leader with dropout perturbations. *COLT*, pages 949–974, 2014.

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

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

L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *ICML*, pages 1058–1066, 2013.

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.

T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *Annals of Statistics*, 32(1):56–85, 2004.