Improving Neural Networks with Dropout

by

Nitish Srivastava

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Improving Neural Networks with Dropout

Nitish Srivastava
Master of Science
Graduate Department of Computer Science
University of Toronto
2013

Deep neural nets with a huge number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining many different large neural nets at test time. Dropout is a technique for addressing this problem. The key idea is to randomly drop units (along with their connections) from a neural network during training. This prevents the units from co-adapting too much. Dropping units creates thinned networks during training. The number of possible thinned networks is exponential in the number of units in the network. At test time all possible thinned networks are combined using an approximate model averaging procedure. Dropout training followed by this approximate model combination significantly reduces overfitting and gives major improvements over other regularization methods. In this work, we describe models that improve the performance of neural networks using dropout, often obtaining state-of-the-art results on benchmark datasets.

Contents

Chapter 1

Introduction

Neural networks are powerful computational models that are being used extensively for solving problems in vision, speech, natural language processing and many other areas.

In spite of many successes, neural networks still suffer from a major weakness. The presence of non-linear hidden layers makes deep networks very expressive models which are therefore prone to severe overfitting. A typical neural net training procedure involves early stopping to prevent this. Several regularization schemes have also been proposed to prevent overfitting. These methods combined with large datasets have made it possible to apply neural networks for solving machine learning problems in several domains. However, overfitting still remains a major challenge to overcome when it comes to training extremely large neural networks or working in domains which offer very small amounts of data.

Model combination typically improves the performance of machine learning models. Averaging the predictions of several models is most helpful when the individual models are different from each other and each model is fast to train and use at test time. However, large neural networks are hard to train and slow to use at test time. In order to make them different they must either have different hyperparameters or be trained on different data. This often makes it impractical to train many large networks and average all their predictions at test time.

"Dropout" is a technique that aims to address both these concerns. The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, we mean removing it from the network, along with all its incoming and outgoing edges. The choice of which units to drop is random. In the simplest case, each unit is retained with a fixed probability p, where p can be chosen based on the particular problem by a validation set (a typical value is $p = 0.5$). Dropping out is done independently for each hidden unit *and* for each training case. Thus, applying dropout to a neural network amounts to sub-sampling a "thinned" neural network from it. 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 large n, each time a training case is presented, it is likely to use a new thinned network. So training a neural network with dropout can be seen as training a collection of 2^n thinned networks with massive weight sharing, where each thinned network gets trained very rarely, if at all.

When the model is being used at test time, it is not feasible to explicitly average the predictions from exponentially many thinned models. However, a very simple approximate averaging method works well. The idea is to use a single neural net at test time without dropout. The weights of this test network are scaled versions of the weights of the thinned networks used during training. The weights are scaled such that for any given input to a hidden unit the *expected* output (under the distribution used to drop units at training time) is the same as the output at test time. So, if a unit is retained with probability p, this amounts to multiplying the outgoing weights of that unit by p. With this approximate averaging method, 2^n networks with shared weights can be combined into a single neural network to be used at test time. Training a network with dropout and using the approximate averaging method at test time leads to significantly lower generalization error on a wide variety of classification problems.

Dropout can also be interpreted as a way of regularizing a neural network by adding noise to its hidden units. This idea has previously been used in the context of Denoising Autoencoders [26, 27] where noise is added to the inputs of an autoencoder and the target is kept noise-free. Our work extends this idea by dropping units in the hidden layers too and performing appropriate weight scaling at test time. Compared to the 5% noise that typically works best for DAEs, it is usually optimal to drop out 20% of input units and 50% of the hidden units to obtain the most benefit from dropout.

A motivation for this method comes from a theory of role of sex in evolution [10]. Sexual reproduction involves taking half the genes of one parent and half of the other and combining them to produce an offspring. The asexual alternative involves creating an offspring with a copy of the parent's genes. It seems plausible that asexual reproduction is a better optimizer of fitness which is widely believed to be the criterion for natural selection, i.e., successful organisms would be able to create more copies of successful organisms. Sexual reproduction seems to be downgrading the genes by pairing up two randomly chosen halves. However, sexual reproduction is the way most advanced organisms evolved. One explanation is that the criteria for natural selection may not be individual fitness but rather mix-ability of genes. The ability of genes to be able to work well with another random set of genes makes them more robust. Since a gene cannot rely on an exact partner to be present at all times, it must learn to do something useful on its own without relying on a partner to make up for its shortcomings. Similarly, hidden units in a neural network trained with dropout must learn to work with a randomly chosen sample of other units. This makes each hidden unit more robust and drives it towards creating useful features on its own without relying on other hidden units to correct its mistakes. Preventing co-adaptation in this manner improves neural networks.

The idea of dropout is not limited to feed forward neural nets. It can be more generally applied to graphical models such as Boltzmann Machines. The chapters that follow explore different aspects of dropout in detail, apply it to different problems and compare it with other forms of regularization and model combination.

Chapter 2

Dropout with feed forward neural nets

This chapter describes training and testing methods to be used when dropout is applied to feed forward neural nets.

2.1 Model Description

This section describes the dropout neural network model. Consider a neural network with L hidden layers. Let $l \in \{1, \ldots, L\}$ index the hidden layers of the network. Let $\mathbf{z}^{(l)}$ denote the vector of inputs into layer l, $\mathbf{y}^{(l)}$ denote the vector of outputs from layer l ($\mathbf{y}^{(0)} = \mathbf{x}$ is the input). $W^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases at layer l. The feed forward operation of a neural network can be described as (for $l \in \{0, \ldots, L-1\}$)-

$$\begin{aligned} \mathbf{z}^{(l+1)} &= W^{(l+1)}\mathbf{y}^{l} + \mathbf{b}^{(l+1)} && (2.1) \\ \mathbf{y}^{(l+1)} &= f(\mathbf{z}^{(l+1)}) && (2.2) \end{aligned}$$

where f is any activation function. With dropout, the feed forward operation becomes -

$$\begin{aligned} r_i^{(l)} &\sim \text{Bernoulli}(p) && (2.3) \\ \widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)} && (2.4) \\ \mathbf{z}^{(l+1)} &= W^{(l+1)}\widetilde{\mathbf{y}}^{l} + \mathbf{b}^{(l+1)} && (2.5) \\ \mathbf{y}^{(l+1)} &= f(\mathbf{z}^{(l+1)}) && (2.6) \end{aligned}$$

Here $\mathbf{r}^{(l)}$ is a vector of Bernoulli random variables each of which has probability p of being 1. This vector is sampled for each layer and multiplied element-wise with the outputs of that layer, $\mathbf{y}^{(l)}$, to create the thinned outputs $\widetilde{\mathbf{y}}^{(l)}$. The thinned outputs are then used as input to the next layer. For learning, the derivatives of the loss function are backpropagated through the thinned network.

At test time, the weights are scaled as $W_{test}^{(l)} = pW^{(l)}$. The resulting neural network is run without dropout.

2.2 Learning dropout nets

Dropout neural networks can be trained with stochastic gradient descent. Dropout is done separately for each training case in every minibatch. Dropout can be used with any activation function and our experiments with logistic, tanh and rectified linear units yielded similar results though requiring different amounts of training time (rectified linear units were fastest to train). Several methods that have been used to improve stochastic gradient descent with standard neural networks such as momentum, decaying learning rates and L2 weight decay are useful for dropout neural networks as well.

One particular form of regularization was found to be especially useful for dropout - constraining the norm of the incoming weight vector at each hidden unit to be upper bounded by a fixed constant c. In other words, if $\mathbf{w}_i$ represents the vector of weights incident on hidden unit i, the neural network was optimized under the constraint $||\mathbf{w_i}||_2 \leq c$. This constraint was imposed during optimization by scaling $\mathbf{w_i}$ to lie on a ball of radius c, if it ever violated the constraint. This is kind of regularization is also called max-norm regularization and has been previously used in the context of collaborative filtering [20].

The constant c is a tuneable hyperparameter, which can be determined using a validation set. Although dropout alone gives significant improvements, optimizing under this constraint, coupled with a large decaying learning rates and high momentum provides a significant boost over just using dropout. One explanation of this fact is that constraining the weight vector to lie inside a ball of fixed radius makes it possible to use a huge learning rate without the possibility of weights blowing up. The noise provided by dropout then allows the optimization process to explore different regions of the weight space that it would have otherwise not encountered. As the learning rate decays, the optimization takes shorter steps and gradually trades off exploration with exploitation and finally settles into a minimum.

2.3 Pretraining dropout nets

Neural networks can be pretrained using stacks of RBMs [6], autoencoders [27] or Deep Boltzmann Machines [17]. This pretraining followed by finetuning with backpropagation has been shown to give significant performance boosts over finetuning from random initializations in certain cases. Pretraining is also an effective way of making use of unlabeled data.

Dropout nets can also be pretrained using these techniques. The procedure is identical to standard pretraining [6] except with a small modification - the weights obtained from pretraining should be scaled up by a factor of $1/p$. The reason is similar to that for scaling down the weights by a factor of p when testing (maintaining the same expected output at each unit). Compared to learning from random initializations, finetuning from pretrained weights typically requires a smaller learning rate so that the information in the pretrained weights is not entirely lost.

2.4 Classification Results

The above training and test procedure was applied to several datasets. The best results were consistently obtained when dropout was used. These datasets range over a variety of domains and tasks -

- MNIST is a standard toy dataset of handwritten digits.
- TIMIT is a standard speech benchmark for clean speech recognition.

- SVHN consists of images of house numbers collected by Google Street View.
- Reuters-RCV1 is a collection of Reuters newswire articles.
- Flickr-1M consists of multimodal data (1 million images and tags).
- Alternate Splicing dataset consists of biochemistry data for genes.

For completeness, we also report results obtained by other researchers who have used dropout. These include CIFAR-10 [7] and ImageNet-1K [9]. This section describes the results along with the neural net architectures used to obtain these results. All datasets are publicly available and all except TIMIT and WSJ are free. The code for reproducing these results can be obtained from `http://www.cs.toronto.edu/~nitish/dropout`. The implementation is GPU-based and uses cudamat [11]. Convolutional neural net implementation is based on kernels from cuda-convnet used for obtaining results in [9].

2.4.1 Results on MNIST

Method	**Unit Type**	**Error %**
2 layer NN [19]	Logistic	1.60
SVM gaussian kernel	-	1.4
Dropout	ReLU	1.25
Dropout + weight norm constraint	ReLU	1.05
DBN + finetuning	Logistic	1.18
DBN + dropout finetuning	Logistic	0.92
DBM + finetuning	Logistic	0.96
DBM + dropout finetuning	Logistic	**0.79**

Figure 2.1: Comparison of training methods

.

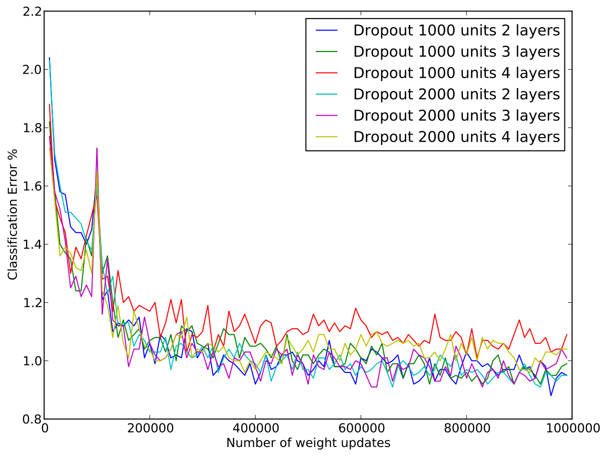


Figure 2.2: Test error for different architectures

.

MNIST is a collection of 28×28 pixel handwritten digit images. There are 60,000 training and 10,000 test images. A validation set consisting of 10,000 images was held out from the training set. No input preprocessing was done. No spatial information or input distortions was used.

Classification experiments were done with networks of many different architectures. Fig. 2.2 shows the test error curves obtained for some of these. All of these used rectified linear units.

Fig 2.1 compares the test classification results obtained by several different methods and their extensions using dropout. The pretrained dropout nets use logistic units and all other networks use rectified linear units. The best performance without unsupervised pretraining for the permutation invariant setting using neural nets is 1.60% [19]. Adding dropout reduced the error to 1.25% and adding weight norm constraints further reduced that to 1.05%. Pretrained dropout nets also improved the performance for Deep Belief Nets and Deep Boltzmann Machines. DBM pretrained dropout nets achieve a test error of 0.79% which is state-of-the-art for the permutation invariant setting.

2.4.2 Results on SVHN

The Street View House Numbers (SVHN) Dataset [14] consists of real-world images of house numbers obtained from Google Street View. The part of the dataset that we use in our experiments consists of 32×32 pixel color images centered on a digit in a house number. Fig. 2.3 shows some examples of images from this dataset. The task is to identify the digit in the center of the image.

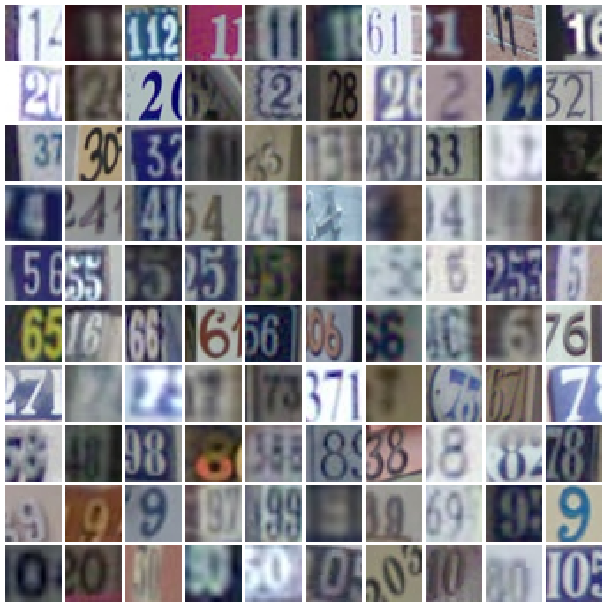


Figure 2.3: Samples of images from the Street View House Numbers (SVHN) dataset.

For this dataset, dropout was applied in convolutional neural networks. The network consists of three convolutional layers each followed by a max-pooling layer. The convolutional layers have 64, 64 and 128 filters respectively. Each convolutional layer has a 5×5 receptive field applied with a stride of 1 pixel. The max pooling layers pool a 3×3 region and are applied at strides of 2 pixels. The convolutional layers are followed by two fully connected hidden layers having 3072 and 2048 units respectively. All units use the rectified linear activation function. Dropout was applied to all the layers of the network with the probability of retaining the unit being $p = (0.9, 0.9, 0.9, 0.5, 0.5, 0.5)$ for the different layers of the network (going from input to convolutional layers to fully connected layers). These hyperparameters were tuned using a validation set. In addition, the weight norm constraint was used for hidden units in the fully-connected layers. Besides the test set, the SVHN dataset consists of a standard labelled training set and another set of labelled examples that are easy. The validation set was constructed by taking examples from both the sets. Two-thirds of it were taken from the standard set (400 per class) and one-third from the extra set (200 per class), a total of 6000 samples. This same process is used in [18]. The inputs were RGB pixels normalized to have zero mean and unit variance.

Table. 2.1 compares the results obtained by using dropout with other methods. Dropout leads to a more than 35% relative improvement over the best previously published results. It bridges the distance to human-level performance by more than half. The additional gain in performance obtained by adding dropout in the convolutional layers besides doing dropout in the fully connected layers suggests that the utility of dropout is not limited to densely connected neural networks but can be more generally applied to other specialized architectures.

Table 2.1: Results on the Street View House Numbers dataset.

Method	**Error %**
Binary Features (WDCH) [14]	36.7
HOG [14]	15.0
Stacked Sparse Autoencoders [14]	10.3
KMeans [14]	9.4
Multi-stage Conv Net with average pooling [18]	9.06
Multi-stage Conv Net + L2 pooling [18]	5.36
Multi-stage Conv Net + L4 pooling + padding [18]	4.90
Conv Net + max-pooling	3.95
Conv Net + max pooling + dropout in fully connected layers	3.02
Conv Net + max pooling + dropout in all layers	2.78
Conv Net + max pooling + dropout in all layers + input translations	**2.68**
Human Performance	2.0

2.4.3 Results on TIMIT

TIMIT is a speech dataset with recordings from 680 speakers covering 8 major dialects of American English reading ten phonetically-rich sentences in a controlled noise-free environment. It has been used to benchmark many speech recognition systems. Table. 2.2 compares dropout neural nets against some of them. The open source Kaldi toolkit [16] was used to preprocess the data into log-filter banks and to get labels for speech frames. Dropout neural networks were trained on windows of 21 frames to predict the label of the central frame. No speaker dependent operations were performed. A 6-layer dropout net gives a phone error rate of 23.4%. This is already a very good performance on this dataset. Dropped further improves it to 21.8%. Similarly, a 4-layer pretrained dropout net improves the phone error rate from 22.7% to 19.7%.

Table 2.2: Phone error rate on the TIMIT core test set.

Method	**Phone Error Rate%**
Neural Net (6 layers) [12]	23.4
Dropout Neural Net (6 layers)	21.8
DBN-pretrained Neural Net (4 layers)	22.7
DBN-pretrained Neural Net (6 layers) [12]	22.4
DBN-pretrained Neural Net (8 layers) [12]	20.7
mcRBM-DBN-pretrained Neural Net (5 layers) [2]	20.5
DBN-pretrained Neural Net (4 layers) + dropout	**19.7**
DBN-pretrained Neural Net (8 layers) + dropout	**19.7**

2.4.4 Results on Reuters-RCV1

Reuters-RCV1 is a collection of newswire articles from Reuters. We created a subset of this dataset consisting of 402,738 articles and a vocabulary of 2000 most commonly used words after removing stop words. The subset was created so that the articles belong to 50 disjoint categories. The task is to identify the category that a document belongs to. The data was split into equal sized training and test sets.

A neural net with 2 hidden layers of 2000 units each obtained an error rate of 31.05%. Adding dropout reduced the error marginally to 29.62%.

2.4.5 Results on Flickr-1M

Often real-world data consists of multiple modalities - photographs on the web (images and text), videos (images and sound), sensory perception (images, sound, touch, internal feedbacks). Multimodal data raises interesting machine learning problems such as fusing multiple modalities into a joint representation and inferring missing modalities conditioned on observed ones. Recent efforts have been made in computer vision [4] and deep learning [15, 22, 21].

The Flickr-1M dataset [8] consists of 1 million pairs of images and tags (text attributed to the images by users) obtained from the social photography website Flickr. 25,000 pairs are labelled into 38 overlapping topics. The other 975,000 image-text pairs are unlabeled. The task is to identify the topics to which the labelled pairs belongs. Applying dropout to this dataset seeks to demonstrate two ideas. Firstly, the use of unlabeled data to pretrain dropout neural networks and secondly, to show the applicability of dropout to the much less studied domain of multimodal data.

Table 2.3: Results on the Flickr-1M dataset.

Method	**Mean Average Precision %**	**Precision at 50**
LDA [8]	0.492	0.754
SVM [8]	0.475	0.758
DBN [22]	0.599	0.867
Autoencoder (based on [15])	0.600	0.875
DBM [22]	0.609	0.873
Multiple Kernel Learning SVMs [4]	0.623	-
DBN with dropout finetuning	0.628	0.891
DBM with dropout finetuning	**0.632**	**0.895**

Table. 2.3 compares the pretrained dropout neural networks with other models. The evaluation metrics are Mean Average Precision and Precision at 50. Mean Average Precision is the mean over all 38 topics of the recall-weighted precision for each topic. Precision at 50 is the mean over all 38 topics of the precision at a recall of 50 data points. The labelled set was split as 10K-5K-10K for training, validation and testing respectively. The unlabeled data was used for training DBN and DBM models as described in [22]. The discriminative model pretrained by a DBN has more than 10 million parameters. The DBM model, after being unrolled as described in [17] has around 16 million parameters. However, the training set is only 10,000 in size. This makes it hard to discriminatively finetune the models without causing overfitting. However, when dropout is applied, overfitting is drastically reduced. Dropout with pretrained models achieves state-of-the-art results, outperforming the best previously published results on this dataset that were obtained with an Multiple Kernel Learning based SVM model [4]. It is also interesting to note that the MKL model used over 30,000 standard computer vision features while our model used 3857 features only.

2.4.6 Results on ImageNet

ImageNet-1K is a collection of over 1 million images categorized into 1000 labels. The system that was used to obtain state-of-the-art results on this dataset in the ILSVRC-2012 competition [9] used convolutional neural networks trained with dropout. The model achieved a top-5 error rate of 15.3% and won the competition by a massive margin (The second best entry stood at 26.2%).

2.5 Comparison with Bayesian methods.

Dropout can be seen as a way of doing an approximate equally-weighted averaging of exponentially many models. On the other hand, Bayesian neural networks [13] are the proper way of doing model averaging over a continuum of neural network models with appropriate weights. Unfortunately, Bayesian neural nets are slow to train and difficult to scale to very large neural nets. It is also expensive to get predictions from many large nets at test time. On the other hand, dropout neural nets are much faster to train and use at test time. However, Bayesian neural nets are extremely useful for solving problems in domains where data is scarce such as medical diagnosis, genetics, drug discovery and other bio-chemical applications. In this section we report experiments that compare Bayesian neural nets with dropout neural nets for small datasets where Bayesian neural networks are known to perform well and obtain state-of-the-art results. These datasets are mostly characterized by having a large number of dimensions relative to the number of examples.

2.5.1 Predicting tissue-regulated alternative splicing

Alternative splicing is a significant cause of cellular diversity in mammalian tissues. Predicting the occurrence of alternate splicing in certain tissues under different conditions is important for understanding many human diseases. The alternative splicing dataset consists of data for 3665 cassette exons, 1014 RNA features and 4 tissue types derived from 27 mouse tissues. Given the RNA features, the task is to predict the probability of three splicing related events that biologists care about. See [29] for a full exposition. The evaluation metric is Code Quality which is a measure of the negative KL divergence between the target and predicted probability distributions (Higher is better).

Table 2.4: Results on the Alternative Splicing Dataset.

Method	**Code Quality (bits)**
Neural Network (early stopping) [29]	440
Regression, PCA [29]	463
SVM, PCA [29]	487
Neural Network (dropout)	567
Bayesian Neural Network [29]	**623**

A two layer network with 1024 units in each layer was trained on this dataset. A value of $p = 0.5$ was used for the hidden layer and $p = 0.7$ for the input layer. Results were averaged across the same 5 folds used in [29]. Table. 2.4 compares dropout neural nets with other models trained on this data. This experiment suggests that dropout improves the performance of neural networks significantly but not enough to match the performance of Bayesian neural networks. The dropout neural networks outperform SVMs and standard neural nets trained with early stopping. It is interesting to note that the dropout nets are very large (1000s of hidden units) compared to a few tens of units in the Bayesian network.

2.6 Comparison with standard regularizers.

Several regularization methods have been proposed for preventing overfitting in neural networks. These include L2 weight decay (more generally Tikhonov regularization [24]), lasso [23] and KL-sparsity regularization which minimizes the KL-divergence between the distribution of hidden unit activations and a target Bernoulli distribution. Another regularization involves putting an upper bound on the norm of the incoming weight vector at each hidden unit. Dropout can be seen as another way of regularizing neural networks. In this section we compare dropout with some of these regularization methods.

The MNIST dataset is used to compare these regularizers. The same network architecture (784-1024-1024-2048-10) was used for all the methods. Table. 2.5 shows the results. The KL-sparsity method used a target sparsity of 0.1 at each layer of the network. It is easy to see that dropout leads to less generalization error. An important observation is that weight norm regularization significantly improves the results obtained by dropout alone.

Table 2.5: Comparison of different regularization methods on MNIST

Method	**MNIST Classification error %**
L2	1.62
L1 (towards the end of training)	1.60
KL-sparsity	1.55
Max-norm	1.35
Dropout	1.25
Dropout + Max-norm	**1.05**

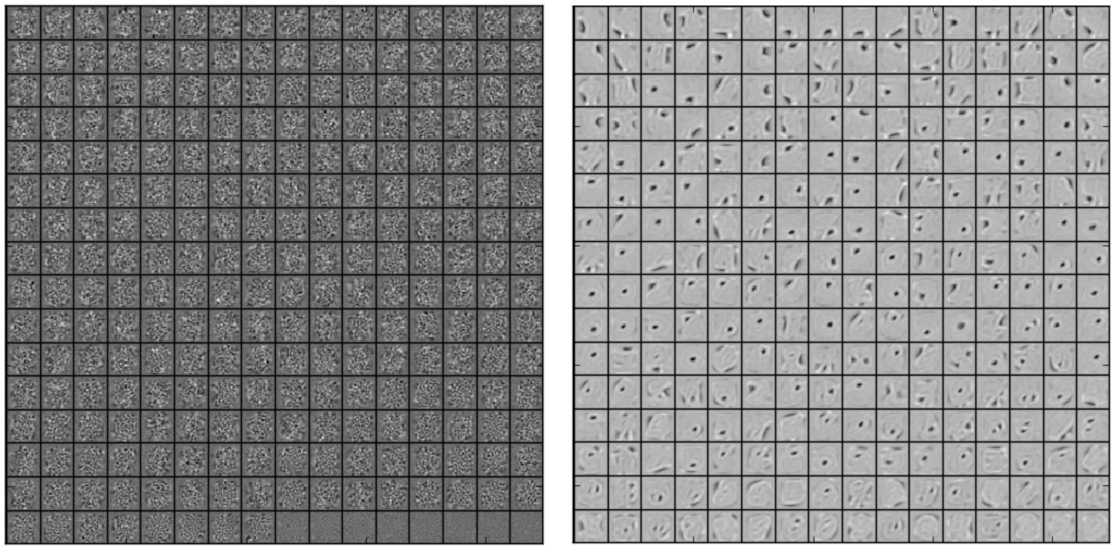
2.7 Effect on features.

In a standard neural network, each parameter individually tries to change so that it reduces the final loss function, *given* what all other units are doing. This conditioning may lead to complex co-adaptations which cause overfitting since these co-adaptations do not generalize. We hypothesize that for each hidden unit, dropout prevents co-adaptation by making the presence of other hidden units unreliable. Therefore, no hidden unit can rely on other units to correct its mistakes and must perform well in a wide variety of different contexts provided by the other hidden units. The experimental results discussed in previous sections lend credence to this hypothesis. To observe this effect directly, we look at the features learned by neural networks trained on visual tasks with and without dropout.

Fig. 2.4a shows features learned by an autoencoder with a single hidden layer of 256 rectified linear units without dropout. Fig. 2.4b shows the features learned by an identical autoencoder which used dropout in the hidden layer with $p = 0.5$. It is apparent that the features shown in Fig. 2.4a have co-adapted in order to produce good reconstructions. Each hidden unit on its own does not seem to be detecting a meaningful feature. On the other hand, in Fig. 2.4b, the features seem to detect edges and spots in different parts of the image.

2.8 Effect on sparsity.

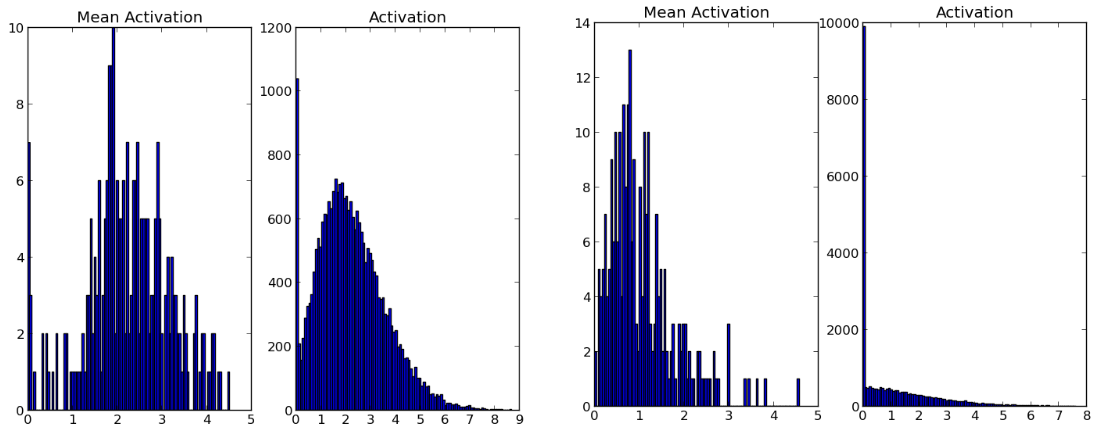
A curious side-effect of doing dropout training is that the activations of the hidden units become sparse, even when no sparsity inducing regularizers are present. Thus, dropout leads to sparser representations. To observe this effect, we take the autoencoders trained in the previous section and look at the histogram of hidden unit activations on a random mini-batch taken from the test set. We also look at the histogram of mean hidden unit activations over the minibatch. Fig. 2.5a and Fig. 2.5b show the histograms for the two models. For the dropout autoencoder, we do not scale down the weights since that would obviously



(a) Without dropout (b) Dropout with $p = 0.5$.

Figure 2.4: Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

increase the sparsity by making the weights smaller. To ensure a fair comparison, the weights used to obtain the histogram were the same as the ones learned during training.



(a) Without dropout (b) Dropout with $p = 0.5$.

Figure 2.5: Effect of dropout on sparsity: In each panel, the figure on the left shows a histogram of the **mean** activation of hidden units in a randomly chosen test minibatch. The figure on the right shows a histogram of the activations on the same minibatch.

In Fig. 2.5a, there are many more hidden units that are in a non-zero state compared to those in Fig. 2.5b, as seen by the significant mass away from zero. The mean activation of hidden units is close to 2.0 for the autoencoder without dropout but drops to around 0.5 when dropout is used.

2.9 Effect of dropout rate.

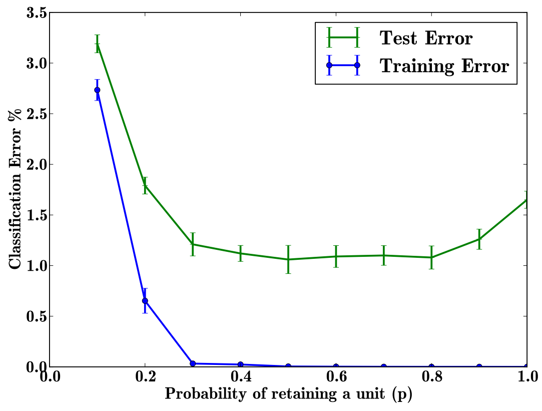
Dropout has a tune-able hyperparameter p (the probability of retaining a hidden unit in the network). In this section, the effect of varying this hyperparameter is explored. The comparison is done in two situations -

1. The number of hidden units is held constant.

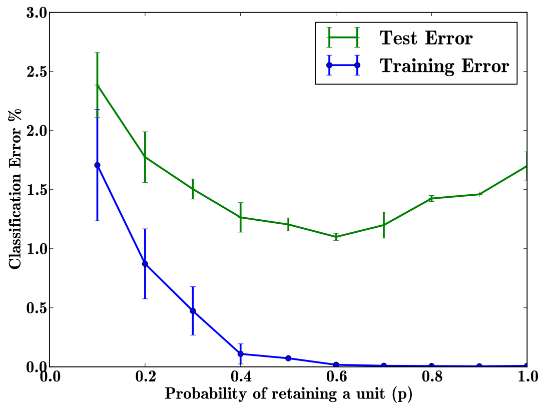
2. The expected number of hidden units that will be retained is held constant.

In the first case, all the nets have the same architecture at test time but they are trained with different amounts of dropout. In our experiment we use a 784-2048-2048-2048-10 architecture. The inputs were not thinned. Fig. 2.6a shows the test error obtained as a function of p. It can be observed that the performance is insensitive to the value of p if $0.4 \leq p \leq 0.8$, but rises sharply for small value of p. This is to be expected because for the same number of hidden units, having a small p means very few units will turn on during training. It can be seen that this has lead to underfitting since the training error is also high.

Therefore, a more fair comparison is the second case in which the quantity pn is held constant where n is the number of hidden units in any particular layer. This means that networks that have small p will have larger number of hidden units. This ensures that the expected number of units that will be present after dropout is same. However, the test networks will be of different sizes. In our experiments, $pn = 256$ for the first two hidden layers and $pn = 512$ for the last hidden layer. Fig. 2.6b shows the test error obtained as a function of p. We notice that the magnitude of errors for small values of p has reduced compared to Fig. 2.6a. Values of p that are close to 0.6 seem to perform best for this choice of pn but our usual default value of 0.5 is close to optimal.



(a) Keeping n fixed.



(b) Keeping pn fixed.

Figure 2.6: Effect of changing dropout rates on MNIST.

2.10 Effect of data set size.

One test of a good regularizer is that it should make it possible to train models with a large number of parameters even on small datasets. This section explores the effect of changing the dataset size when dropout is used with feed forward networks. Huge neural networks trained in the standard way overfit massively on small datasets. To see if dropout can help, we run classification experiments on MNIST and vary the amount of data given to the network.

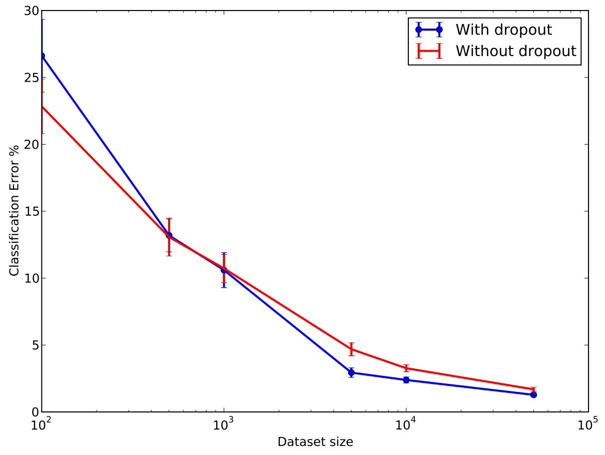


Figure 2.7: Effect of varying dataset size.

The results of these experiments are shown in Fig. 2.7. The network was given datasets of size 100, 500, 1K, 5K, 10K and 50K randomly sampled without replacement from the MNIST training set. The same network architecture (784-1024-1024-2048-10) was used for all datasets. Dropout with $p = 0.5$ was performed at all the hidden layers and $p = 0.8$ at the input layer. It can be observed that for extremely small datasets (100, 500) dropout does not give any improvements. The model has enough parameters that it can overfit on the training data, even with all the noise coming from dropout. As the size of the dataset is increased, the gain from doing dropout increases up to a point and then declines. This suggests that for any given architecture and dropout rate, there is a "sweet spot" corresponding to some amount of data that is large enough to not be memorized in spite of the noise but not so large that overfitting is not a problem anyways.

2.11 Monte-Carlo model averaging vs. weight scaling.

The test time procedure that was proposed is to do an approximate model combination by scaling down the weights of the trained neural network. Another expensive but reasonable way of averaging the models is to sample k neural nets using dropout for each test case and average their predictions. As $k \to \infty$, this Monte-Carlo model average gets close to the true model average. Finite values of k are also expected to give reasonable results. It is interesting to compare the performance of this method with the weight scaling method that has been used till now.

We again use the MNIST dataset and do classification by averaging the predictions of k randomly sampled neural networks. Fig. 2.8 shows the test error rate obtained for different values of k. This is compared with the error obtained using the weight scaling method (shown as a horizontal line). It can be seen that around $k = 50$, the Monte-Carlo method becomes as good as the approximate method. Thereafter, the Monte-Carlo method is slightly better than the approximate method but well within one standard deviation of it. This suggests that the weight scaling method is a fairly good approximation of the true model average.

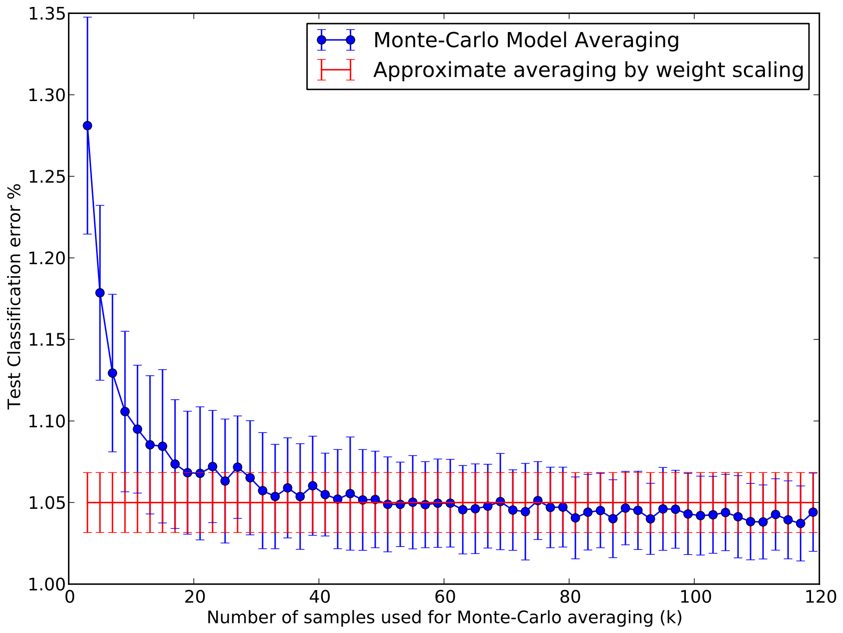


Figure 2.8: Monte-Carlo model averaging vs. weight scaling.

Chapter 3

Dropout with Boltzmann Machines

The core idea behind dropout is to sample smaller sub-models from a large model, train them and then combine them at test time. This idea can be generalized beyond feed forward networks. In this chapter, we explore dropout when applied to Restricted Boltzmann Machines. For clarity of exposition, we describe dropout for hidden units only. Extending dropout to visible units is straightforward.

3.1 Dropout RBMs

Consider an RBM with visible units $\mathbf{v} \in \{0,1\}^D$ and hidden units $\mathbf{h} \in \{0,1\}^F$. It defines the following probability distribution

$$P(\mathbf{h}, \mathbf{v}; \theta) = \frac{1}{\mathcal{Z}(\theta)} \exp(\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v})$$

Where $\theta = (W, \mathbf{a}, \mathbf{b})$ represents the model parameters and $\mathcal{Z}$ is the partition function.

Dropout RBMs are RBMs augmented with a vector of binary random variables $\mathbf{r} \in \{0,1\}^F$. Each random variable r_j takes the value 1 with probability p, independent of others. If r_j takes the value 1, the hidden unit h_j is retained, otherwise it is dropped from the model. The joint distribution defined by a Dropout RBM can be expressed as-

$$\begin{aligned} P(\mathbf{r}, \mathbf{h}, \mathbf{v}; p, \theta) &= P(\mathbf{r}; p) P(\mathbf{h}, \mathbf{v} | \mathbf{r}; \theta) \qquad (3.1) \\ P(\mathbf{r}; p) &= \prod_{j=1}^{F} p^{r_j} (1-p)^{1-r_j} \\ P(\mathbf{h}, \mathbf{v} | \mathbf{r}; \theta) &= \frac{1}{\mathcal{Z}'(\theta, \mathbf{r})} \exp(\mathbf{v}^\top W \mathbf{h} + \mathbf{a}^\top \mathbf{h} + \mathbf{b}^\top \mathbf{v}) \prod_{j=1}^{F} g(h_j, r_j) \\ g(h_j, r_j) &= \mathbb{1}(r_j = 1) + \mathbb{1}(r_j = 0) \mathbb{1}(h_j = 0) \end{aligned}$$

$\mathcal{Z}'(\theta, \mathbf{r})$ is the normalization constant. $g(h_j, r_j)$ imposes the constraint that if $r_j = 0$, h_j must be 0.

The distribution over **h**, conditioned on **v** and **r** is factorial

$$\begin{aligned} P(\mathbf{h}|\mathbf{r},\mathbf{v}) &= \prod_{j=1}^{F} P(h_j|r_j,\mathbf{v}) \\ P(h_j=1|r_j,\mathbf{v}) &= \mathbb{1}(r_j=1)\sigma\left(b_j+\sum_i W_{ij}v_i\right) \end{aligned}$$

The distribution over **v** conditioned on **h** is same as that of an RBM-

$$\begin{aligned} P(\mathbf{v}|\mathbf{h}) &= \prod_{i=1}^{D} P(v_i|\mathbf{h}) \\ P(v_i=1|\mathbf{h}) &= \sigma\left(a_i+\sum_j W_{ij}h_j\right) \end{aligned}$$

Conditioned on **r**, the distribution over $\{\mathbf{v},\mathbf{h}\}$ is same as the distribution that an RBM would impose, except that the units for which $r_j = 0$ are dropped from **h**. Therefore, the Dropout RBM model can be seen as a mixture of exponentially many RBMs with shared weights each using a different subset of **h**.

3.2 Learning Dropout RBMs

Learning algorithms developed for RBMs such as Contrastive Divergence [5] can be directly applied for learning Dropout RBMs. The only difference is that **r** is first sampled and only the hidden units that are retained are used for training. Similar to dropout neural networks, a different **r** is sampled for each training case in every minibatch. In our experiments, we use CD-1 for training dropout RBMs.

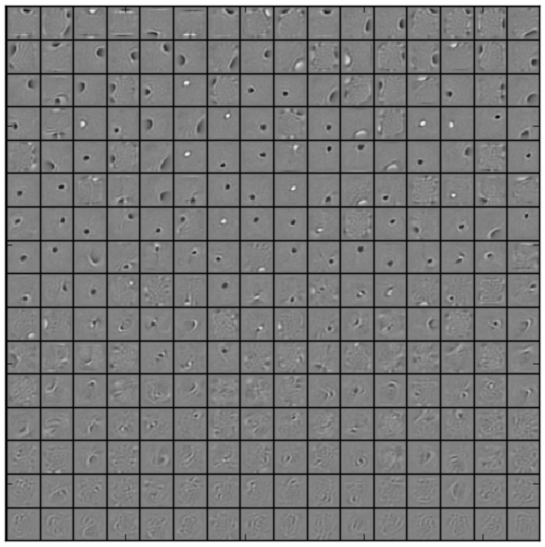
3.3 Effect on features

Dropout in feed forward networks improved the quality of features by reducing co-adaptations. This section explores whether this effect transfers to Dropout RBMs as well.

Fig. 3.1a shows features learned by a binary RBM with 256 hidden units. Fig. 3.1b shows features learned by a dropout RBM with the same number of hidden units. Features learned by the dropout RBM appear qualitatively different in the sense that they seem to capture features that are coarser compared to the sharply defined stroke-like features in the standard RBM. There seem to be very few dead units in the dropout RBM relative to the standard RBM.

3.4 Effect on sparsity

Next, we investigate the effect of dropout RBM training on sparsity of the hidden unit activations. Fig. 3.2a shows the histograms of hidden unit activations and their means on a test mini-batch after training an RBM. Fig. 3.2b shows the same for dropout RBMs. The histograms clearly indicate that the dropout RBMs learn much sparser representations than standard RBMs even when no additional sparsity inducing regularizer is present.


(a) Without dropout

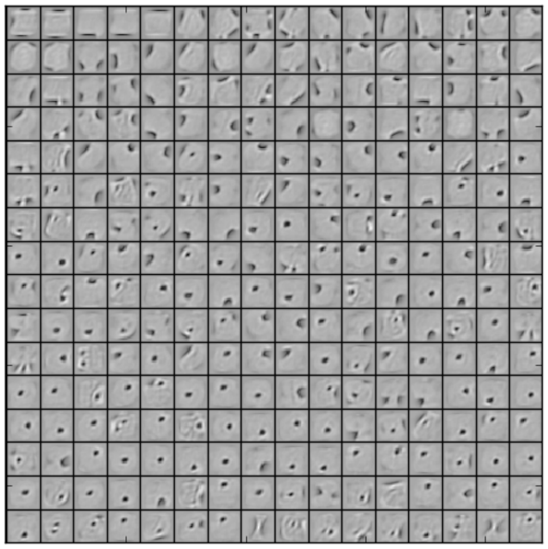
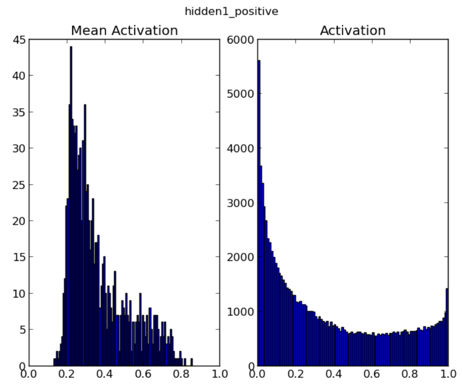
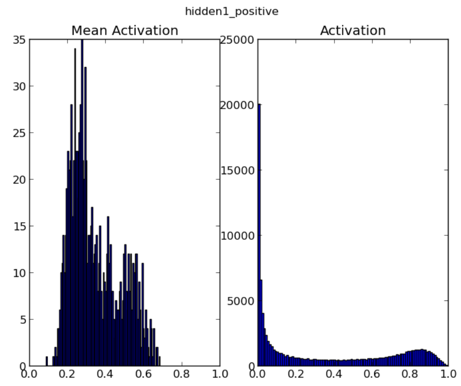

(b) Dropout with $p = 0.5$.

Figure 3.1: Features learned on MNIST by 256 hidden unit RBMs.



(a) Without dropout



(b) Dropout with $p = 0.5$.

Figure 3.2: Effect of dropout on sparsity: In each panel, the figure on the left shows a histogram of the **mean** activation of hidden units in a randomly chosen test minibatch. The figure on the right shows a histogram of the activations on the same minibatch.

Chapter 4

Marginalizing dropout

Dropout can be seen as a way of adding noise to the states of hidden units in a neural network. In this chapter, we explore the class of models that arise as a result of marginalizing this noise. These models can be seen as deterministic versions of dropout. In contrast to regular ("Monte-Carlo") dropout, these models do not need random bits and it is possible to get gradients for the marginalized loss functions. In this chapter, we briefly explore these models.

Marginalization in the context of denoising autoencoders has been explored previously [1, 25]. Deterministic algorithms have been proposed that try to learn models that are robust to feature deletion at test time [3].

4.1 Linear Regression

First we explore a very simple case of applying dropout to the classical problem of linear regression. Let $X \in \mathbb{R}^{N \times D}$ be a data matrix of N data points. $\mathbf{y} \in \mathbb{R}^N$ be a vector of targets. Linear regression tries to find a $\mathbf{w} \in \mathbb{R}^D$ that minimizes

$$||\mathbf{y} - X\mathbf{w}||^2$$

When the input X is dropped out such that any input dimension is retained with probability p, the input can be expressed as $R * X$ where $R \in \{0,1\}^{N \times D}$ is a random matrix with $R_{ij} \sim \text{Bernoulli}(p)$ and $*$ denotes an element-wise product. Marginalizing the noise, the objective function becomes

$$\underset{\mathbf{w}}{\text{minimize}} \quad \mathbb{E}_{R \sim \text{Bernoulli(p)}} \left[||\mathbf{y} - (R * X)\mathbf{w}||^2 \right]$$

This reduces to

$$\underset{\mathbf{w}}{\text{minimize}} \quad ||\mathbf{y} - pX\mathbf{w}||^2 + p(1-p)||\Gamma\mathbf{w}||^2$$

where $\Gamma = (\text{diag}(X^\top X))^{1/2}$. Therefore, dropout with linear regression is equivalent, in expectation, to ridge regression with a particular form for Γ. This form of Γ essentially scales the weight cost for weight w_i by the standard deviation of the ith dimension of the data.

Another interesting way to look at this objective is to absorb the factor of p into $\mathbf{w}$. This leads to the following form

$$\underset{\mathbf{w}}{\text{minimize}} \quad ||\mathbf{y} - X\widetilde{\mathbf{w}}||^2 + \frac{1-p}{p}||\Gamma\widetilde{\mathbf{w}}||^2$$

Where $\widetilde{\mathbf{w}} = p\mathbf{w}$. This makes the dependence of the regularization constant on p explicit. For p close to 1, all the inputs are retained and the regularization constant is small. As more dropout is done (by decreasing p), the regularization constant grows larger.

4.2 Logistic regression and deep networks

For logistic regression and deep neural nets, it is hard to obtain a closed form marginalized model. However, Wang [28] showed that in the context of dropout applied to logistic regression, the corresponding marginalized model can be trained approximately. Under reasonable assumptions, the distributions over the inputs to the logistic unit and over the gradients of the marginalized model are Gaussian. Their means and variances can be computed efficiently. This approximate marginalization outperforms Monte-Carlo dropout in terms of training time and generalization performance.

However, the assumptions involved in this technique become successively weaker as more layers are added and it would be interesting to see if this same technique can be directly extended to deeper networks.

Chapter 5

Conclusions

Dropout is a technique for improving neural networks by reducing overfitting. The main idea is to prevent co-adaptation of hidden units. Dropout improves performance of neural nets in a wide variety of application domains including object classification, digit recognition, speech recognition, document classification and analysis of bio-medical data. This suggests that dropout as a technique is quite general and not specific to any domain. It has been used in models that achieve state-of-the-art results on ImageNet and SVHN.

The central idea of dropout is to take a large model that overfits easily and repeatedly sample and train smaller sub-models from it. Since all the sub-models share parameters with the large model, this process trains the large model which is then used at test time. We demonstrated that this idea works in the context of feed forward neural networks. This idea can be extended to Restricted Boltzmann Machines and other graphical models which can be seen as composed of exponentially many sub-models with shared weights.

Marginalized versions of dropout models may offer some of the benefits of dropout training without having to deal with noise. These models are an interesting direction for future work.

Bibliography

[1] Minmin Chen, Zhixiang Xu, Kilian Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. In John Langford and Joelle Pineau, editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, ICML '12, pages 767–774. ACM, New York, NY, USA, July 2012.

[2] G.E. Dahl, M. Ranzato, A. Mohamed, and GE Hinton. Phone recognition with the mean-covariance restricted boltzmann machine. *Advances in Neural Information Processing Systems*, 23:469–477, 2010.

[3] A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning*, pages 353–360. ACM, 2006.

[4] M. Guillaumin, J. Verbeek, and C. Schmid. Multimodal semi-supervised learning for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 902 –909, june 2010.

[5] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[6] Geoffrey Hinton and Ruslan Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504 – 507, 2006.

[7] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[8] Mark J. Huiskes, Bart Thomee, and Michael S. Lew. New trends and ideas in visual concept detection: the MIR flickr retrieval evaluation initiative. In *Multimedia Information Retrieval*, pages 527–536, 2010.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1106–1114. 2012.

[10] Adi Livnat, Christos Papadimitriou, Nicholas Pippenger, and Marcus W. Feldman. Sex, mixability, and modularity. *Proceedings of the National Academy of Sciences*, 107(4):1452–1457, 2010.

[11] Volodymyr Mnih. Cudamat: a CUDA-based matrix class for python. Technical Report UTML TR 2009-004, Department of Computer Science, University of Toronto, November 2009.

[12] A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, (99):1–1, 2010.

[13] Radford M. Neal. *Bayesian Learning for Neural Networks.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[14] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[15] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y. Ng. Multimodal deep learning. In *International Conference on Machine Learning (ICML)*, Bellevue, USA, June 2011.

[16] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, December 2011. IEEE Catalog No.: CFP11SRW-USB.

[17] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455, 2009.

[18] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *International Conference on Pattern Recognition (ICPR 2012)*, 2012.

[19] P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, volume 2, pages 958–962, 2003.

[20] Nathan Srebro and Adi Shraibman. Rank, trace-norm and max-norm. In *Proceedings of the 18th annual conference on Learning Theory*, COLT'05, pages 545–560, Berlin, Heidelberg, 2005. Springer-Verlag.

[21] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep belief nets. *ICML 2012 Representation Learning Workshop*, 2012.

[22] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. In P. Bartlett, F.C.N. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2231–2239. 2012.

[23] Robert Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, 58(1):267–288, 1996.

[24] Andrey Nikolayevich Tikhonov. On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39(5):195–198, 1943.

[25] Laurens van der Maaten, M. Chen, S. Tyree, and Kilian Weinberger. Learning with marginalized corrupted features. In *Proceedings of the International Conference on Machine Learning*, In Press.

[26] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[27] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, December 2010.

[28] Sida Wang. Fast dropout training for logistic regression. In *NIPS workshop on log-linear models*, 2012.

[29] Hui Yuan Xiong, Yoseph Barash, and Brendan J. Frey. Bayesian prediction of tissue-regulated splicing using rna sequence and cellular context. *Bioinformatics*, 27(18):2554–2562, 2011.