Kushagra Agrawal

Summary :

A Simple Way to Prevent Neural Networks from Overfitting

Abstract

Deep neural nets with a large 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 the predictions of 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 the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single unthinned network that has smaller weights. This significantly reduces overfitting and gives major improvements over other regularization methods. We show that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology, obtaining state-of-the-art results on many benchmark data sets.

Impressions

Primarily, the paper is looking to evaluate Dropout as a method of reducing overfitting and improving model generalization.

Dropout is a regularization technique — a family of techniques for reducing overfitting (thereby improving generalization) by making the decision boundary or fitted model smoother.

The most widely used implementation of Dropout is as a *stochastic* regularization technique, and that is the implementation primarily tested in this paper.

In addition, the authors find that there are additional improvements whereby neuron co-adaptation is reduced and feature (representational) sparsity is improved.

Takeaways: Dropout and its relation to architecture and training

Appendix A is titled “A Practical Guide for Training Dropout Networks” and makes a few recommendations:

- Dropout rate: On real-valued input layers (images, speech), drop out 20%. For internal hidden layers, drop out 50%.
- Layer size: If you think a layer size of N is appropriate for your problem (e.g. an internal hidden layer of N=128 nodes) and want to us dropout rate P (say p=0.5 for 50% dropout), resize that layer to N/P (e.g. double it to 128/0.5=256 nodes) to keep the same representational power.
- Optimizer: When using dropout, use 10–100 times the learning rate. Also bump up your momentum from 0.9 (typical rate without dropout) to 0.95–0.99. (Sec A.2)
- Regularization: Adding max-norm regularization works well, since you are increasing momentum and learning rate.

Takeaways: Using Dropout with CNNs and RNNs

Why do we not see dropout in modern CNN networks — e.g. fully convolutional residual networks with batch normalization?

Takeaways: Using Dropout to measure uncertainty

There is very interesting research by Yarin Gal that builds on dropout to produce uncertainty measures (error bars) from neural networks. Very loosely, at prediction time you use Monte Carlo dropout (predict from many thinned networks), take the average of all the predictions as your prediction and the *variance* of all the predictions as the uncertainty.

Prior methods of reducing overfitting:

- Early stopping
- Regularization: L2 weight penalization, lasso, KL-sparsity, max-norm
- Adding noise during the training process to make the predictor more robust — notably denoising autoencoders (DAEs).

Ensembling:

It is appealing to make an analogy between dropout and ensembling. It seems that Monte Carlo dropout is truly ensembling, and the weight scaling approximation described is an approximation to this.

Methods: Sparsity📓 Results

Results: Generalization

For generalization, the authors compared networks with and without dropout on the following datasets, observing that “dropout improved generalization performance on all data sets compared to neural networks that did not use dropout”:

MNIST : A standard toy data set of handwritten digits.

TIMIT : A standard speech benchmark for clean speech recognition.

CIFAR-10 and CIFAR-100 : Tiny natural images (Krizhevsky, 2009).

Street View House Numbers data set (SVHN) : Images of house numbers collected by Google Street View (Netzer et al., 2011).

ImageNet : A large collection of natural images.

Reuters-RCV1 : A collection of Reuters newswire articles.

Alternative Splicing data set: RNA features for predicting alternative gene splicing (Xiong et al., 2011).

Results: Sparsity

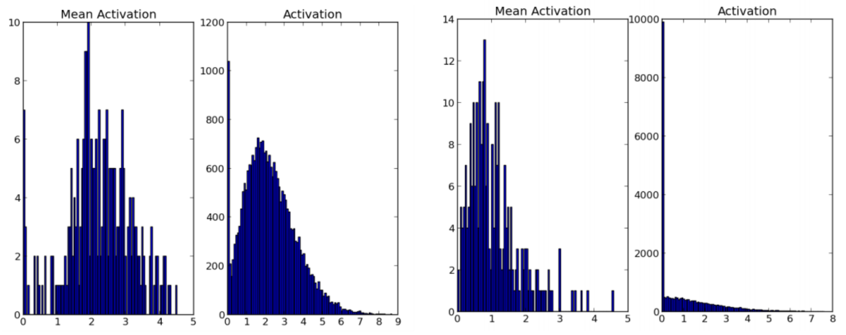
A sparse feature is a feature that that has mostly zero values like a one-hot encoding of categorization or a TFIDF encoding. Think of them in contrast to dense features — some examples of which are image data, audio data, and word embedding vectors. Sparse features *should* be more interpretable, as individual neurons will be activated (or individual dimensions given a high value) which correspond to concepts. Think of this like NLP networks with a “sentiment neuron”, or the output of VGG where each softmax dimension is a distinct class.

The authors assert that:

In a good sparse model, there should only be a few highly activated units for any data case. Moreover, the average activation of any unit across data cases should be low.

The authors produce a histogram of activation weights as well as a histogram of the mean activation weights across data samples.

For sparsity, indeed, we can see that dropout pushes the distribution of activations to skew heavily toward zeroes with very few units showing high activation. The mean activations are lowered from 2.0 to 0.7:



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

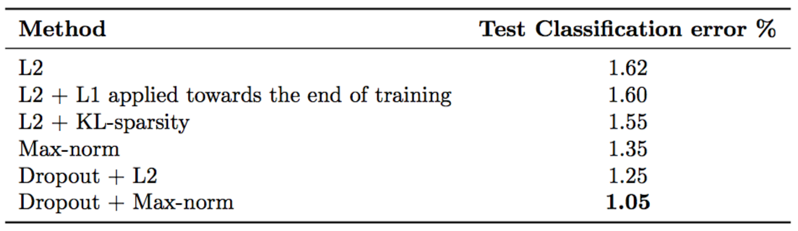
Figure 8: Effect of dropout on sparsity. ReLUs were used for both models. **Left**: The histogram of mean activations shows that most units have a mean activation of about 2.0. The histogram of activations shows a huge mode away from zero. Clearly, a large fraction of units have high activation. **Right**: The histogram of mean activations shows that most units have a smaller mean mean activation of about 0.7. The histogram of activations shows a sharp peak at zero. Very few units have high activation.

From the paper:

[Examining] features learned by an autoencoder on MNIST (...) each hidden unit [without dropout] on its own does not seem to be detecting a meaningful feature. On the other hand, [with dropout], the hidden units seem to detect edges, strokes and spots in different parts of the image. This shows that dropout does break up co-adaptations, which is probably the main reason why it leads to lower generalization errors.

Results: Comparison to other regularization methods

Here is Table 9 from the paper with results:



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

Table 9: Comparison of different regularization methods on MNIST.

Results: Prediction: Weight scaling and Monte Carlo sampling

The authors find that the weight scaling approximation method is a good approximation of the true model

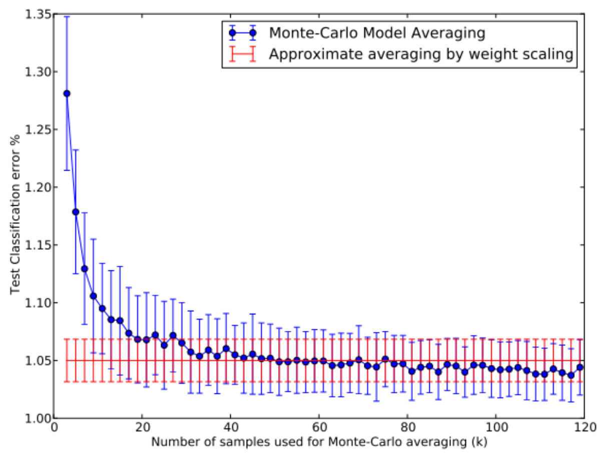


Figure 11: Monte-Carlo model averaging vs. weight scaling.

average:

We (...) do classification by averaging the predictions of k randomly sampled neural networks. (...) 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.

Results: Comparison to Bayesian Neural Networks

It is interesting to me that the authors find that Bayesian Neural Networks yield beter performance than dropout networks at reducing overfitting in small dataset regimes, at the cost of additional computation. I was not previously aware of Bayesian Neural Networks and this does pique my interest in learning more:

Xiong et al. (2011) used Bayesian neural nets for this task. As expected, we found that Bayesian neural nets perform better than dropout. However, we see that dropout improves significantly upon the performance of standard neural nets and outperforms all other methods. The challenge in this data set is to prevent overfitting since the size of the training set is small.

The scaling approximation not as good as BNN, but for the one problem they examine (predicting the occurrence of alternative splicing based on RNA features), it's better than other approaches (early stopping, regression with PCA, SVM with PCA).

Reference-
Paper Review: Dropout: A Simple Way to Prevent Neural Networks from Overfitting | by Jason Morrison | Paper Club | Medium
srivastava14a.pdf (toronto.edu)