Dropout: A Simple Way to Prevent Neural Networks from Overfitting

1 Citation

Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

`http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf`

2 Abstract

Randomly drop half the units during each training batch and cut all weights in half at test-time and you'll get a well regularized model that generalizes well.

3 Introduction

Early stopping, L2 regularization, and ensembling are popular techniques to help models generalize. You can think of dropout as training exponentially many neural nets that share weights. By dropping (i.e. zeroing) half the weights, we are effectively sampling a network. By halving the weights at test time, we are basically computing the expected value of all the networks.

4 Motivation

In animals, genes are created by combining genes from each parent and mutating. This is a bit like dropout. By randomly dropping units, we force units to learn useful things on their own because they can't depend on other units because those units might be dropped.

5 Related Work

Previous work added noise to autoencoder weights to improve generalization. We find dropping 50% of hidden units and 20% of input units helps generalization.

6 Model Description

Let $\boldsymbol{z}^{(l)}$ and $\boldsymbol{y}^{(l)}$ be the input and output, respectively, of layer l in an L-layer neural net. We have, for activation function f, $\boldsymbol{z}_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)}\boldsymbol{y}^{(l)} + b_i^{(l+1)}$ and $\boldsymbol{y}_i^{(l+1)} = f(\boldsymbol{z}_i^{(l+1)})$. With dropout, this becomes (where $*$ is element-wise product)

$$r_j^l = \text{Bernoulli}(p) \tag{1}$$
$$\tilde{\boldsymbol{y}}^{(l)} = \boldsymbol{r}^{(l)} * \boldsymbol{y}^{(l)} \tag{2}$$
$$z_i^{(l+1)} = \boldsymbol{w}_i^{(l+1)} \tilde{\boldsymbol{y}}^{(l)} + b_i^{(l+1)} \tag{3}$$
$$\boldsymbol{y}_i^{(l+1)} = f(\boldsymbol{z}_i^{(l+1)}) \tag{4}$$

At test time, we scale the weights $W_{test}^{(l)} = pW^{(l)}$.

7 Learning Dropout Nets

You can train with usual backpropagation and just sample a different dropout mask for each minibatch. It also helps to constrain the norm of the weights into a unit to be at most some fixed constant (this is called max-norm regularization). Using large decaying learning rates with high momentum and max-norm regularization allows the model to aggressively explore weights without every worrying that they will get too big.

You can also use dropout for pretraining and fine-tuning a Deep Belief Network (i.e. stack of Restricted Boltzmann Machines trained with contrastive divergence). When fine-tuning, scale up weights by $1/p$ to balance out the dropout during pretraining.

8 Experimental Results

We experiment with MNIST (handwritten digit classification), TIMIT (speech recognition), CIFAR-10/CIFAR-100 (image classification), Street View House Numbers (digit classification), ImageNet (image classification), Reuters (document classification), and Alternative splicing (bioinformatics). Dropout helps models on all these datasets.

Deep Belief Networks, feedforward nets, and convnets all benefit from dropout. It works with many different output units (e.g. ReLU, sigmoid, maxout). It works best with maxout units.

Bayesian Neural Networks (these do proper model averaging) are popular for domains where data is scarce. They are slow to train and hard to scale to large nets. Dropout does not face these limitations. The Adaptive splicing dataset (predict occurence of alternative splicing from RNA features) is a dataset where Bayesian Neural Nets do well. They also beat dropout, but dropout is better than all other models. Compared to L2 weight decay, lasso, and KL-sparsity, dropout with max-norm is best.

9 Salient Features

Without dropout, neural nets can have co-adaptation (one unit corrects the mistakes of another). With dropout, a unit cannot depend on other units and thus must learn something useul on its own. Looking at the first-layer features of an autoencoder trained on MNIST, the units are much less noisy with dropout, indicating they are not relying on co-adaptation. Additionally, we find that hidden unit activations are sparser (i.e. few hidden units in a layer have high activations). We vary p and find that picking something between $0.4 \leq p \leq 0.8$ for hidden units works well. Use something larger for input units though. We recommend $p = 0.5$ ($p = 0.8$) for hidden (input) units.

A good regularizer will get good results with a large model on small datasets. If we vary the size of the MNIST dataset, dropout does worse than no-dropout on tiny (not easy to avoid overfitting) datasets, better on medium/large datasets, and the same on huge (really hard to overfit) datasets.

We described how simply scaling the weights at test time approximating the model average. By doing Monte-Carlo sampling of dropout masks at test time (no weight scaling) and averaging the models, we find the result is about the same as our weight scaling heuristic.

10 Dropout Restricted Boltzmann Machines

You can also use dropout on the hidden units of an RBM. We train with CD-1 (contrastive divergence with one reconstruction of visible units). The dropout RBM learnes coarser features and has many fewer dead features. Dropout RBMs also have sparser hidden unit activations.

11 Marginalizing Dropout

The dropout technique we've shown so far has been Monte-Carlo, but you can make it deterministic for simple models. You can explicitly introduce a mask matrix (where the entries are sampled from a Bernoulli distribution) and minimize the expected loss. We derive the cost function for linear regression with dropout. You can also do this for logistic regression. It's hard to do it for deep networks, which is why we use Monte-Carlo dropout.

12 Multiplicative Gaussian Noise

Our dropout mask has entries that are Bernoulli random variables, but you could also sample from zero-mean Gaussians and multiply the activation by $1 + r$ where r is the sampled noise. At test-time, you don't need to do any scaling (the noise has expected value 0).

Also, if you don't want to scale weights at test-time, you can just multiply activations by $1/p$ at train time and leave weights untouched at test time.

13 Conclusion

Dropout improves models by breaking co-adaptation of units. We demonstrated this on several datasets and several neural network types. The problem with dropout is that it takes 2-3 times longer to train than a model without dropout. This is because parameter updates are noisier (we are effectively training many models, one per training step, after all).

14 A Practical Guide for Training Dropout Neural Networks

If there are n hidden units in a layer, using dropout will result in pn effective units (maybe less because we also prevent co-adaptation).

The noise from dropouts causes gradients to cancel each other, so use a high learning rate (10-100 times more than usual) or a high momentum (0.95 or 0.99).

Use max-norm regularization (with max norm $c = 3$ or $c = 4$). Use $p = 0.5$ for hidden units and $p = 0.8$ for input unit, although the number of units n might also be useful for picking the right p.

15 Detailed Description of Experiments and Datasets

For MNIST, we pick hyperparameters with a validation and then combine the validation set into the training set when we picked the hyperparameters. For Street View House Numbers, preprocessing with global contrast normalization and ZCA whitening did not help. We applied dropout on conv layers and hidden layers (conv layers had larger p). For CIFAR-10/CIFAR-100, global contrast normalization and ZCA whitening for preprocessing helped. For TIMIT, we preprocessed with Kaldi and generated an alignment with a monophone system. We use dropout for pretraining and fine-tuning (use a small learning rate of 0.01 for fine-tuning). Dropout did not help much for Reuters, probably because it's a huge dataset. For alternative splicing, our metric is code quality (formula in paper).