International Journal of Research in Engineering, Science and Management
Volume-1, Issue-9, September-2018
www.ijresm.com | ISSN (Online): 2581-5782

Drop: A Simple Way to Prevent Neural Network by Overfitting

Vishal Shirke[1], Ritesh Walika[2], Lalita Tambade[3]
[1,2,3]*Student, Department of Computer Engineering, MGM College, Navi Mumbai, India*

***Abstract*—In the world of neural networks, deep neural nets having a large set of parameters are very powerful machine learning systems. But, such networks comes across a serious problem known as overfitting. With large networks being slow to use, overfitting becomes even more difficult to deal with by combining the predictions of many different large neural nets at a test time. However, there is a technique known as Dropout to address this problem. Dropping the units randomly along with their connections from the network during their training is the key idea behind this technique. What this does is that this prevents the units from co-adapting too much. Dropout samples from an exponential number of different "thinned" networks. The approximation of the effect of averaging the predictions of all these thinned networks is easy at the test time and it's simply done by using a single unthinned network that has smaller weights. And this results in the significant decrease of overfitting giving major improvements over other regularization methods. In short, we will be showing 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.**

***Index Terms*—Neural networks, regularization, model combinations, deep learning**

I. Introduction

Containing non-linear hidden layers makes deep neural networks very expressive models and they can also learn very complicated relationships between their inputs and outputs. But limited training data makes these complicated relationships a result of sampling noise, hence, even if it is drawn from the same distribution, they will exist in the training set but not in real test data. Ultimately, this results in Overfitting and number of methods have been developed for reducing it. Such methods include the stoppage of training as soon as performance on a validation set becomes worsening, introducing various kinds of weight penalties such as L1 and L2 regularization and soft weight training.

The best way to "regularize" a fixed-sized model is to average the predictions of all possible settings of the

Parameters, and also by weighting each setting by the probability of its posterior. However, all this can be approximated quite well sometimes for simple or small models, but we approach Bayesian gold standard's performance using considerably less computation. We propose to do this by taking an approximation of equally weighted geometric mean of all the possible predictions of an exponential number of learned models that share the parameters.

In most of the cases, model combination almost always improves the machine learning method's performance. The obvious idea behind the large neural networks' averaging the outputs of many separately nets being trained is expensive in a prohibitive way. When the individual models are different from each other, combining the several models proves most helpful in order to achieve the neural net models being made different.

Neural Networks are a versatile family of model used to find relationships between enormous volumes of data, such as the one we usually work with. They come in all shapes and sizes. Their accuracy is significantly conditioned by both their structure and size and quality of the data they are trained on.

Building models help data scientists to answer their questions. However, when we use adaptive and deep neural networks models, the risk of overfitting is always present. Thankfully, we can apply a number of procedures and techniques to avoid this overfitting- like pruning when using classification trees, stop criteria in generic algorithms or bagging in a more general context. Some Machine learning methods like the ensemble methods- where many weak learners co-operate smartly combining their predictions- were designed to avoid overfitting. Models following this kind of pattern of many weak learners co-operating often show higher accuracy and more stable results (that is, they generalise better) than other singleton complex models out there.

As we said, the adaptability of feed-forward and deep neural networks is a source of overfitting. Furthermore, the amount of data and computational effort required to train a single neural network grows rapidly as we add hidden layers to its architecture. Thus, separately lots of neural networks in an attempt to mimic ensemble methods is a rather daunting task.

Dropout is a technique that tackles both of these issues by exploiting a simple idea: Dropping some of the neurons and their connections to their counterparts during training.

In every training batch some neurons' connections are temporarily removed, obtaining a simple and lighter version of the compete Neural network. The most generic way to do this is by "dropping" each neuron with probability "p" independent of the others. This means that their weights won't be modified either in the feed-forward or in the back-propagation process, and no output is issued from that neuron.

Once trained, at test time, every weight Wij in the complete Neural network is scaled down, multiplying it by the expected probability of have been used in a given instance of the lighter versions. In the previous case, this just amounts to substitute W_{ij} for its scaled-down value pW_{ij}.

Given that we know a bit about dropout, a question arises—why do we need dropout at all? Why do we need to literally shut-down parts of a neural networks?
The answer to these questions is "to prevent over-fitting".

A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. Now that we know a little bit about dropout and the motivation, let's go into some detail. If you just wanted an overview of dropout in neural networks, the above two sections would be sufficient. In this section, I will touch upon some more technicality.

In machine learning, regularization is way to prevent over-fitting. Regularization reduces over-fitting by adding a penalty to the loss function. By adding this penalty, the model is trained such that it does not learn interdependent set of features weights. Those of you who know Logistic Regression might be familiar with L1 (Laplacian) and L2 (Gaussian) penalties.

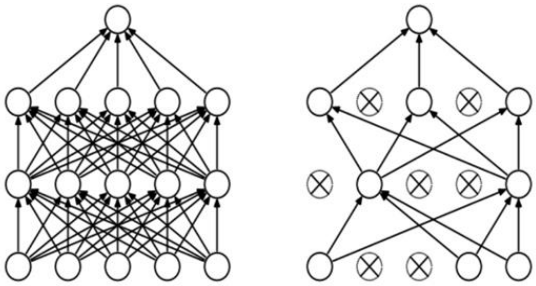
(a) Standard Neural Net (b) After applying dropout.

Fig. 1. Neural networks

Dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.

Training Phase:

For each hidden layer, for each training sample, for each iteration, ignore (zero out) a random fraction, p, of nodes (and corresponding activations).

Testing Phase:

Use all activations, but reduce them by a factor p (to account for the missing activations during training).

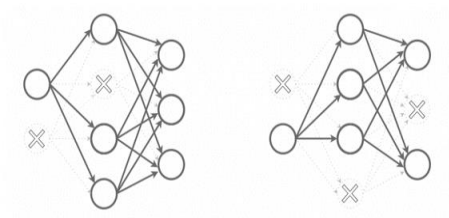

Fig. 2. Dropout approach

II. Related Work

Another question which arises is "Why would I want to cripple my neural network? " So the answer to this would be: As weird as it may sound, cancelling some neurons' ability to learn during training actually aims to obtain better trained neurons and reduce overfitting.

By doing so, we get an approximate result of averaging the simpler trained models, which would otherwise take a lot more time and computational power to be trained by one. But this into the only reason. In fact, training our neurons in such a particular way not only helps them to co-adapt, balancing their weaknesses and strengths, it also ensures that the features they encapsulate work well with randomly chosen subsets of other neurons' learned features. After all, during training time, they couldn't rely on all of their colleagues to do the job as most of the time some of them went missing.

This rests in more demanding neurons that try to move past complicated, tailor-made features – which are prone to generalize poorly, and retain more useful information on their own. In the following figure, we find a comparison of the features learned on MNIST dataset with one hidden layer auto encoder having 256 rectified linear units without dropout (left) and the features learned by the same structure using dropout in its hidden layer with p=0.5 (right). While the former shows unstructured, messy patterns which are impossible to interpret, the latter clearly exhibits purposeful weight distributions that detect strokes edges and spots on their own, breaking their co-dependency with other neurons to carry out the job.

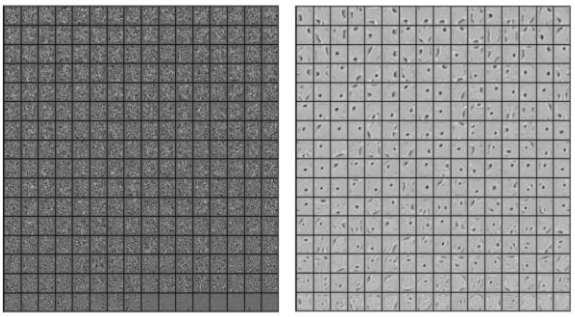
(a) Without dropout (b) Dropout with $p = 0.5$.

Fig. 3. Without dropout and with dropout p=0.5

III. Learning Dropout Nets

This section describes a procedure for training dropout neural nets.

A. Back propagation

Dropout neural networks can be trained using stochastic gradient descent in a manner similar to standard neural nets. The only difference is that for each training case in a mini-batch, we sample a thinned network by dropping out units. Forward and back propagation for that training case are done only on this thinned network. The gradients for each parameter are averaged over the training cases in each mini-batch. Any

training case which does not use a parameter contributes a gradient of zero for that parameter. Many methods have been used to improve stochastic gradient descent such as momentum, annealed learning rates and L2 weight decay. Those were found to be useful for dropout neural networks as well.

B. Unsupervised Pre training

Neural networks can be pre trained using stacks of RBMs (Hinton and Salakhutdinov, 2006), auto encoders (Vincent et al., 2010) or Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009). Pre training is an effective way of making use of unlabelled data. Pre training followed by fine tuning with back propagation has been shown to give

Significant performance boosts over fine tuning from random initializations in certain cases.

IV. Some Observations

1) Dropout forces a neural network to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.
2) Dropout roughly doubles the number of iterations required to converge. However, training time for each epoch is less.
3) With H hidden units, each of which can be dropped, we have 2^H possible models. In testing phase, the entire network is considered and each activation is reduced by a factor p.

V. Experiment in KERAS

Let's try this theory in practice. To see how dropout works, I build a deep net in Keras and tried to validate it on the CIFAR-10 dataset. The deep network is built had three convolution layers of size 64, 128 and 256 followed by two densely connected layers of size 512 and an output layer dense layer of size 10 (number of classes in the CIFAR-10 dataset).

I took ReLU as the activation function for hidden layers and sigmoid for the output layer (these are standards, didn't experiment much on changing these). Also, I used the standard categorical cross-entropy loss.

Finally, I used dropout in all layers and increase the fraction of dropout from 0.0 (no dropout at all) to 0.9 with a step size of 0.1 and ran each of those to 20 epochs. The results look like this:

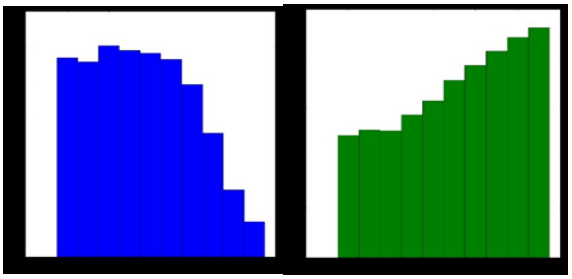

Fig. 4. Accuracy vs. Dropout (blue) and Loss vs. Dropout (green)

From the above graphs we can conclude that with increasing the dropout, there is some increase in validation accuracy and decrease in loss initially before the trend starts to go down.

There could be two reasons for the trend to go down if dropout fraction is 0.2:

1) 0.2 is actual minima for the dataset, network and the set parameters used.
2) More epochs are needed to train the networks

VI. Conclusion and Future Scope

Dropout is a technique which improves neural networks by reducing overfitting. Standard back-propagation learning enables the building of brittle co-adaptations that work for the training data but fails to generalise unseen data. These co-adaptations are broken by random dropouts by making the presence of any particular hidden unit unreliable. Dropout has been found helpful in improving the performance of neural nets in a wide variety of application domains including object classification, digit recognition, speech recognition, document classification and is not specific to any domain. Using this technique, many methods achieve state-of-the-art results on SVHN, I'm agent, CIFAR-100 and MOIST. The performances of standard neural nets on even other data sets has been improved by Dropout.

The main idea which is central to all of this process is to take a large model that overfits easily and repeatedly sample and to train smaller sub-models from it. Such an idea can also be exerted to Restricted Boltzmann Machines and other graphical models and hence we have also developed Dropout RBMs and empirically showed that they have certain desirable properties.

But dropout is far from perfect and one of the major drawbacks of it is that it increases training time. In comparison to a standard neural network or the same architecture, a dropout network typically takes 2-3 times longer. This increase in the training time is majorly caused by the reason that the parameter updates are very noisy. A different random architecture is effectively being tried to trained by each training case effectively. That's why the gradients that are being computed are not gradients of the final architecture that will be used at test time. And hence, training taking a long time comes as no surprise. But stochasticity preventing overfitting is likely however which creates a trade-off between overfitting and training time. But by marginalizing the noise that does the same thing as the dropout procedure (in expectation), is the one way to obtain some of the benefits without stochasticity. We showed this because wanted to tell that for linear regression this regularize risk a modified form of L2 regularization. It is not obvious how to obtain an equivalent regularised for more complicated models. An interesting direction for future work would be to speeding up dropout.

References

[1] D.Maio and D. Maltoni. "Direct gray-scale minutiae detection in fingerprints" IEEE Trans. Pattern Anal. And Machine Intel. 19(1):27-40, 1997.
[2] N. Ratha, S. Chen and A.K. Jain, :Adaptive Flow Orientation Based Feature Extraction in Fingerprint Images", in *Pattern Recognition*, Vol. 28, pp. 1657-1672, November 1995.

[3] Alessandro Farina, Zsolt M.Kovacs-Vajna, Alberto leone, Fingerprint minutiae extraction from skeletonized binary images,in *Pattern Recognition*, vol. 32, no. 4, pp. 877-889, 1999.

[4] M. K. Thakur, R. S. Kumar, M. Kumar, and R. Kumar, "Wireless Fingerprint Based Security System using Zigbee," in *International Journal of Inventive Engineering and Sciences*, vol. 1, no. 5, April 2013.

[5] Mary Lourde R and Dushyant Khosla, "Fingerprint Identification in Biometric Security Systems", *International Journal of Computer and Electrical Engineering*, Vol. 2, no. 5. Pp. 852-855, Oct. 2010.