Paper Notes

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Authors of the Paper: Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov

Compiled By: Aarush Gupta

Discussion held on 15th August 2018.

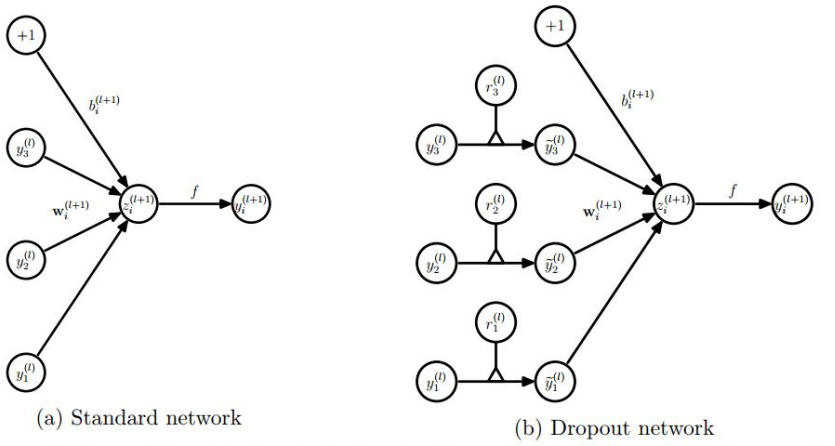
This paper introduces the idea of Dropout for neural networks to prevent units from co-adapting too much by using an ensemble of models. These models are basically constructed by shutting down neurons randomly from the original network. The resulting model is a regularized version of the original model and performs better.

Motivation

1. Training an ensemble model is very time and resource consuming. Also, *combining several neural networks for ensembling is most helpful when the individual models are different from each other, that is, they should either have different architectures or should be trained on different data.*
2. Also, separate regularization methods such as weight constraints, *soft weight sharing*, etc. have to be used with each model in the ensemble to prevent overfitting.
3. Each hidden unit in a neural network trained with dropout must learn to work with a randomly chosen sample of other features on its own without relying on other hidden units to correct its mistakes. The results can be seen as an ensemble of thinned neural networks with extensive weight sharing.

Dropout

1. The outputs from a layer are multiplied element-wise with a 0-1 mask (with number of elements in the mask equal to the number of outputs) sampled from a bernoulli distribution with p as the the probability of sampling **1**.
2. This step is carried out in each layer for each training example in a mini-batch, i.e, for each datapoint in the minibatch, a new thinned sub-network is sampled using the above method on which a forward and backward pass is made.
3. During backpropagation, the gradients of each weight for a mini-batch is the average of the gradients (of each weight) obtained for each training datapoint in that mini-batch. If a weight was dropped out for a training datapoint, then the contribution in the gradient of that weight by that datapoint would be zero.



(a) Standard network (b) Dropout network

Figure : Comparison of the basic operations of a standard and dropout network.

Figure taken from Dropout Paper

4. At test time, each weight is multiplied by p. This ensures that for any hidden unit, the expected output is the same as the actual output at test time. Since test-time performance is critical, it is always preferable to use **inverted dropout**, which performs the **scaling at train time**, leaving the forward pass at test time untouched. This also means that during evaluation the dropout module simply computes an identity function.

Implementation Details

1. The probability for dropout can be deduced by using the validation dataset, or can be safely set to 0.5. This value is close to optimal for a wide range of networks and tasks. Note that for input neurons, the optimal **probability of retention** is close to 1 rather than to 0.5.
2. Dropout can work in tandem with other regularization techniques. In particular, dropout works well with max-norm regularization, which constrains the norm of the incoming weight vector at each hidden units to be upper bounded by a fixed constant c.
3. Large decaying learning rates and high momentum values also provide a significant boost over just using dropout.
4. *Note that the hyperparameter p in most Deep Learning libraries is the probability that sampling from the bernoulli distribution will yield 0.*