Dropout Training

Wantee Wang

2015-03-03 15:18:26 +0800

Contents

Dropout is a regularisation technique for reducing over-fitting in large neural nets. Hinton proposes the method in this paper. Most materials are from Srivastava's page.

It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term *dropout* refers to dropping out units (hidden and visible) in a neural network.

1 The Method

There are 2 key points for dropout learning:

- a) Dropping units while training;
- b) Scaling output to be matched between training and testing.

As shown in following figure (Figure 1), where p is the dropout retention.

Units to be dropped is chosen in a random way. Note that dropping a unit out means temporarily removing it from the network, along with all its incoming and outgoing connections. Therefore we have to deal with it both during forward pass and backpropagation.

Applying dropout to a neural network amounts to sampling a *thinned* network from it. A neural net with n units, can be seen as a collection of 2^n possible thinned neural networks. For each presentation of each training case, a new thinned network is sampled and trained.

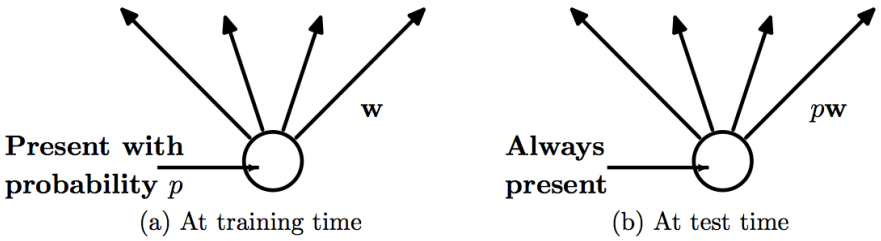

Figure 1: Dropout

At test time, the ideal way is to explicitly average the predictions from exponentially many thinned models, which is obviously not feasible. The intuitive way is using a single neural net without dropout at test time, however this needs some approximation.

The goal is that for any hidden unit the expected output (under the distribution used to drop units at training time) is the same as the actual output at test time.

Let $\mathbb{M}$ be the set of all thinned networks, and $\mathcal{M}$ be the network without dropout used in test time, i.e. the network containing all units. Note that, weights for all networks in $\mathbb{M}$ are shared and are equal to the ones in $\mathcal{M}$. Thus the expected output of a unit j is

$$\mathbb{E}[\mathbf{y}_j] = \sum_{M \in \mathbb{M}} Pr(M)\mathbf{y}_j^M$$

Where, $\mathbf{y}^M$ is the output of thinned network M.

Let $\mathbb{M}^*$ be the set of networks in which unit j is active, then $|\mathbb{M}^*| = p|\mathbb{M}|$. If we assume that the probability of Ms are equal, i.e. $Pr(M) = \frac{1}{|\mathbb{M}|}$, and assume $\mathbf{y}^M = \mathbf{y}^{\mathcal{M}}$, we get,

$$\mathbb{E}[\mathbf{y}_j] = p|\mathbb{M}|\frac{1}{|\mathbb{M}|}\mathbf{y}_j^{\mathcal{M}} = p\mathbf{y}_j^{\mathcal{M}}$$

At this point, there are two method to match the training output and testing output. First one, by scaling down the weight used at test time, i.e. $\mathbf{w}'_{ji} = p\mathbf{w}_{ji}$, we can achieve the goal. This is the way used in the above paper and shown in the figure.

The second way is to scale up the output at training time to the same magnitude as test time, i.e. $\mathbf{y}'_j = \frac{1}{p}\mathbf{y}_j$.

2 Implementation in Kaldi

Both Karel's and Dan's implementation have the Dropout codes, with some differences.

Karel's code(`src/nnet/nnet-activation.h:Dorpout`) uses the scale-up method to get the expected output. Dropping out is implemented during forward pass and by storing the dropped out units using a 0/1 vector, the back-propagated derivative can be set properly.

Dan's code(`src/nnet2/net-component.cc:DropoutComponent`) use a clever way to avoid storing the dropping units. While backpropagation, we can get the input error $\mathbf{e_i}$ from output error $\mathbf{e_i}$ by

$$\mathbf{e_i} = \frac{\mathbf{a_o}}{\mathbf{a_i}}\mathbf{e_o}$$

where $\mathbf{a_i}$ and $\mathbf{a_o}$ is the activation of input and output for Dropout component. Elements in $\mathbf{a_o}$ is the equal to the corresponding scaled value in $\mathbf{a_i}$, which maybe zero if it is the dropping ones.

Dan's code applies a more general form of scaling. Instead of set the output of dropping unit to zero, we can just scale the output value by a factor α. To get a proper scaled version of output, we'd like to scale all the units besides the dropping ones and make it satisfy that the expected scale factor should be 1, i.e.,

$$q\alpha + (1-q)\beta = 1$$

where, $q = 1 - p$ is the dropout proportion. Therefore, we can get the factor of other units $\beta = \frac{1-q\alpha}{1-q}$. If we set $\alpha = 0$, then $\beta = \frac{1}{1-q} = \frac{1}{p}$, which is equal to the scale-up factor.

Online version at http://wantee.github.io/2015/03/03/dropout-training/