Machine learning

Lecture 5

Lecturer: Haim Permuter *Scribe: Gal Rattner*

I. Introduction

Throughout this lecture we introduce the *overfitting* problem and regularization methods such as *L-norm* and *dropout*. Parts of this lecture are inspired by the work of Michael Nielsen [1] and T. Cover's book [3]. This lecture assumes you are familiar with the basic probability theory. The notation here are similar to those of the previous lectures.

II. Overfitting and Regularization

Assume we want to fit a polynomial model to a set of training pairs $(x_1, y_1), \ldots, (x_N, y_N)$, s.t. it will generalize in the best way and will give regression estimation for some test set $x_{N+1}, \ldots, x_{N+K}$. It is clear from Figure 1 that the $8th$ order polynomial has smaller error from each one of the training examples, comparing to the $1st$ order polynomial. As the order of the polynomial increases, it consists of more free parameters and therefore it is more likely to fit better to the training set. Though, it is not clear whether or not it generalizes better and will fit better to an unseen test set, as can be seen in this example.

This phenomena is called *overfitting* and it is prevalent in models consist of large number of parameters. Overfitting is one of the major challenges when training deep neural networks, which naturally consists of enormous number of parameters. In order to avoid the overfitting phenomena, there has been suggested several forms of *regularization*, including adding penalty term to the cost function, using dropout or max-norm constraints over the weights inside the net.

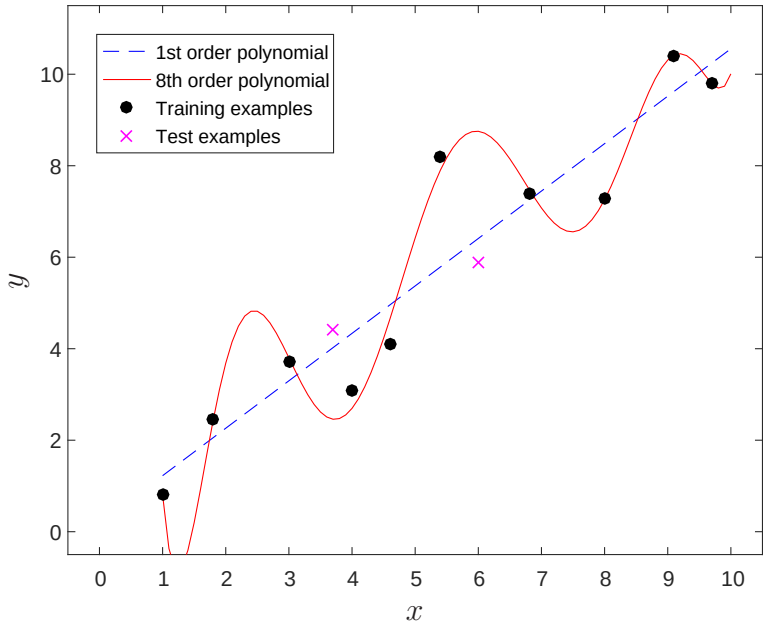

Figure 1. Two different polynomial curves fitted to a set of training examples.

Definition 1 (L-Norm Regularization) Let C_0 be a cost function as presented in the previous lectures, then the term $\lambda \|w\|_L$ is the *L-norm regularization term* added to the cost function. The total cost argument is given by

$$C = C_0 + \lambda \|w\|_L , \tag{1}$$

where λ is the regularization hyper-parameter fixed to determine the influence of the regularization term on the total cost, and $\|w\|_L$ is the *L-norm* expression for the entire set of weights in the model.

Among the most common *L-norm regularization* methods are the *L2* and *L1* regularization terms.

***L2* regularization:** Using the *L2* regularization term, the total cost function is given by

$$C = C_0 + \frac{\lambda}{2n} \|\mathbf{w}\|_2^2$$

$$= C_0 + \frac{\lambda}{2n} \sum_i w_i^2, \tag{2}$$

where C_0 can be the quadratic, cross-entropy or other cost function, w_i are the weights in the net, and the hyper-parameter λ scale the regularization term to have gentle influence on the total cost C, and holds $\lambda > 0$. The influence of the L2 regularization term is clear when observing the partial derivatives of the cost function in equation (2), i.e.

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w, \tag{3}$$

and

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}. \tag{4}$$

The partial derivatives can be computed using backpropagation, as described in lecture 4, and the update rules are given by

$$b \to b - \eta \frac{\partial C_0}{\partial b}, \tag{5}$$

and

$$\begin{aligned} w &\to w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta\lambda}{n} w \\ &= \left(1 - \frac{\eta\lambda}{n}\right) w - \eta \frac{\partial C_0}{\partial w}. \end{aligned} \tag{6}$$

This update rule is the same as the usual gradient descent update rule, except we rescale the weights first by factor $\frac{\eta\lambda}{n}$. This rescaling factor is sometimes referred to as the *weight decay* factor.

***L1* regularization:** Using the *L1* regularization term the total cost argument is given by

$$C = C_0 + \frac{\lambda}{n} \sum_i |w_i|, \tag{7}$$

where $\lambda > 0$.

The partial derivatives of the cost function with respect to w is given by

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} sign(w). \tag{8}$$

The update rule for w is given by

$$w \rightarrow w - \eta \frac{\partial C_0}{\partial w} - \frac{\eta \lambda}{n} sign(w). \tag{9}$$

We notice that both L2 and L1 regularization are penalizing large weights, and causing the weights to decrease toward zero. The difference is in the rate of decrement, where with L1 regularization the decrement rate is constant, with L2 regularization the decrement rate is proportional to the size of the weight w.

III. Dropout

A very common regularization method of deep neural network which have proven its improvement skills is the *dropout*, as presented in [4]. Unlike the regularization methods described above, dropout modify the network's connections rather than the cost function.

The way dropout works is by setting each activation node to zero with some prefixed probability $p \in [0, 1]$, for each training iteration. The hyper-parameter p is generally set before the training session starts. On that way, for an input or hidden layer with N_l nodes, at each training iteration we get an average of only $p \times N_l$ active nodes in the net. Notice that we do not drop any node at the net's output layer. In order to keep the total sum of activations coming out of each layer constant, we multiply any un-dropped activation by factor $\frac{1}{p}$. Repeating this method over and over during the training operation, the deep neural network will learn a set of weights and biases which generalize better than the regular training.

The motivation of using this method can be given in three main points:

1) **Ensemble of networks**: Training a single network using dropout is similar to training an ensemble of networks over the same training dataset. For each training instance, a somewhat different network is trained. When using an ensemble of nets, each started with different random starting point, its final training state will be slightly different. Considering that, combining the results over the ensemble will almost surely reduce the error and give better result. The same assumption stands for a single network trained with dropout.

2) **Reduced number of trained parameters**: The number of trained weights is reduced per training instance, making the number of co-adaptations between different nodes smaller and easier to train.
3) **Avoid overfitting**: Specific weights are prevented from growing too large and push the whole network toward overfitting, because they are dropped and untrained with probability p. The survived nodes at each instance are trained using only part of the total information learnt to that point, making overfitting less plausible.

Visual example for dropout can be seen here were in Figure (2) we see a fully-connected net, and in Figure (3) we see the net after applying dropout on the hidden layer. Notice that the droped nodes are kept blurred, signifying that the nodes are not deleted but only ignored for a specific training iteration.

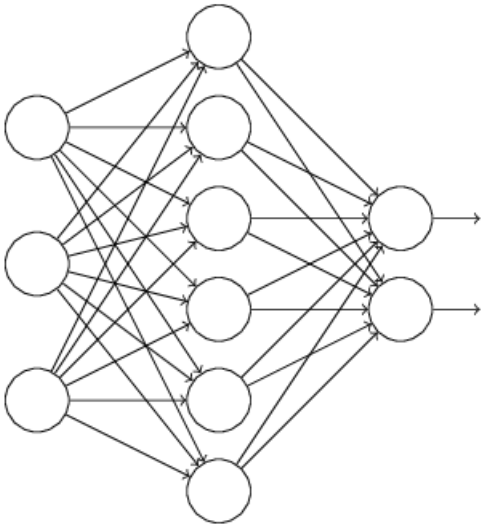

Figure 2. Fully connected neural network.

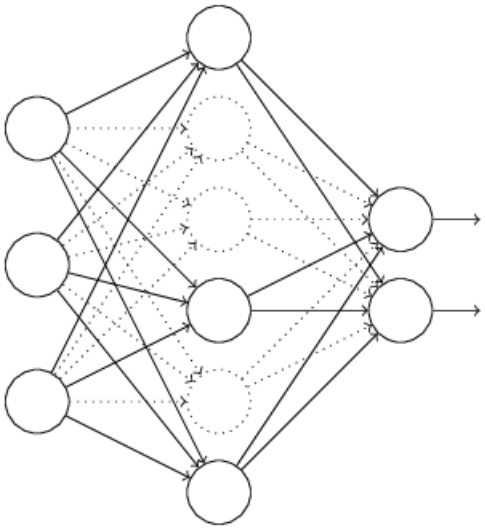

Figure 3. Fully connected neural network after applying dropout on the hidden layer.

REFERENCES

[1] M. Nielsen *Neural Networks and Deep Learning, Chap. 3*. http://neuralnetworksanddeeplearning.com/chap3.html. January 2016.

[2] H. Permuter *Introduction to Information Theory course*. http://www.ee.bgu.ac.il/ haimp/it/index.html

[3] T. M. Cover and J. A. Thomas *Elements of Information Theory, Chap. 1*.

[4] N. srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1), 1929-1958.