Lecture #17: Back Propagation

Instructor: Aditya Bhaskara   *Scribe: Tuowen Zhao*

CS 5966/6966: Theory of Machine Learning

March 20th, 2017

Abstract

Introduction of the back propagation algorithm.

1 Introduction

Last week, we have discussed the general structure of a "threshold networks" that have "neurons" in each layer and compute some weighted threshold of the input, where we also assume one output f. We also discussed the *universality property* that a two-layer network can compute any function. But, we also noted that many functions require Exp(#Input) size network, because the VC-dim of the class of m-edges and n-nodes networks is $O((n+m)\log(n+m)$.

The *Fundamental Theorem of Statistical Learning* implies that to learn the best network we only need $O(d/\epsilon^2)$ samples. But, to be able to learn efficiently we also need an efficient algorithm to solve the ERM problem. As we have seen, even for a very simple network, learning is at least as hard as 3-coloring.

Albeit the hardness, there are algorithms that may work in this setting as we will see one such in this lecture.

2 Back propagation algorithm

The idea behind *Back Propagation* is *Gradient Descent*, but in this case the function is not guaranteed convex and may take a very long time to converge. But the algorithm often work fine and find good solutions in practice. We will formulate the algorithm starting with a simple one-layer network and gradually adding layers to it.

One-layer network

Suppose we have a one-layer network such as in Figure 1. For a set of $(x^{(1)}, y_1), \ldots, (x^{(m)}, y_m)$ training examples The training problem is as in Equation 1.

$$(1) \qquad \arg\min_{w,\tau} = \sum_i \mathbb{1}[f(x^{(i)}) \neq y_i]$$

Note that, in the equation, we used the indicator function to calculate the loss while using sign function for calculating the threshold, but these are not suitable for Gradient Descent due to the gradient being not well defined on

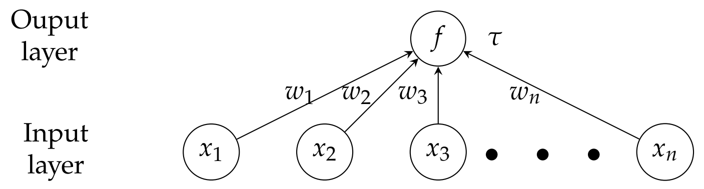

Figure 1: One-layer network

such functions. Other functions can be used for loss function, such as squared loss, l1-norm ... And, functions such as rectified linear and sigmoid $\frac{1}{1+e^{-x}}$ can be used for the activation function instead. Assuming we use loss function ℓ and use some activation function σ.

The gradient of the output in relation to one parameter w_j can be formulated as follows:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_i \ell'(f(x^{(i)}) - y_i)\sigma'(x^{(i)})x_j^{(i)}$$

Using the above we can update the weights in the network using the update formula of gradient descent, $w^{(t+1)} = w^{(t)} - \eta \nabla \ell$. This weight update is related to the input in a way a lot like the *Perceptron* algorithm, where $w^{(t+1)} = w^{(t)} + \eta y_i x^{(i)}$. Also note that for the stochastic gradient descent, instead of summing all the gradient of the examples, we can just pick one example, such that:

For $t = 1 \ldots T$

1. Pick an index $i_t = [N]$
2. $w^{(t+1)} = w^{(t)} - \eta \nabla \ell_1 i_t f(w^{(t)})$

Two-layer network

For a two layer network such as in Figure 2. The training equation becomes that in Equation 2.

$$\underset{w,u,\tau_y,\tau_f}{\arg\min} = \sum_i \ell(f(x^{(i)}), y_i) \tag{2}$$

The gradient to u_j can be calculated by setting the output of y_i being constant, then the formula is the same as in the one-layer network. For w_{jk}, we first can observe that if the absolute value of u_j is small, such as 0, then the gradient of w_{jk} is small as well, because all the influence of w_{jk} to ℓ is through $u_j(y_j)$. We can formulate that intuition in mathematical terms as follows:

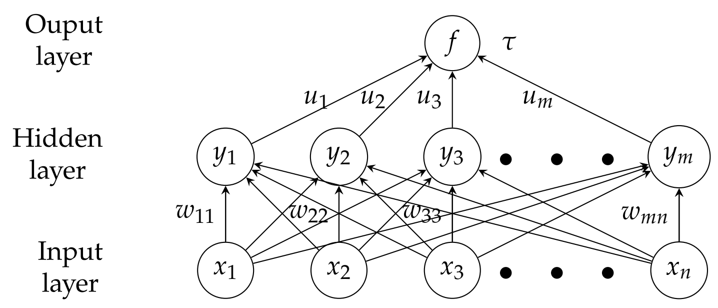

Figure 2: Two-layer network

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_{jk}} &= \sum_i \frac{\partial \ell}{\partial y_j} \frac{\partial y_j}{\partial w_{jk}} \\
&= \sum_i \frac{\partial \ell}{\partial y_j} \sigma'_{y_j} x_k \\
&= \sum_i \ell' \sigma'_f u_j \sigma'_{y_j} x_k
\end{aligned}
$$

If we ignore the summation, the equation becomes:

$$\frac{\partial \ell}{\partial w_{jk}} = \ell' \sigma'_f u_j \sigma'_{y_j} x_k$$

Recall the gradient to u_j:

$$\frac{\partial \ell}{\partial u_j} = \ell' \sigma'_f y_j$$

Let $e = \ell' \sigma'_f$, we have:

$$
\begin{aligned}
\frac{\partial \ell}{\partial w_{jk}} &= e u_j \sigma'_{y_j} x_k \\
\frac{\partial \ell}{\partial u_j} &= e y_j
\end{aligned}
$$

Three-layer network

For a three layer network as in Figure 3. We can repeat the above formulation and come up with a vector relation of the formula. The details will be discussed in the next lecture.

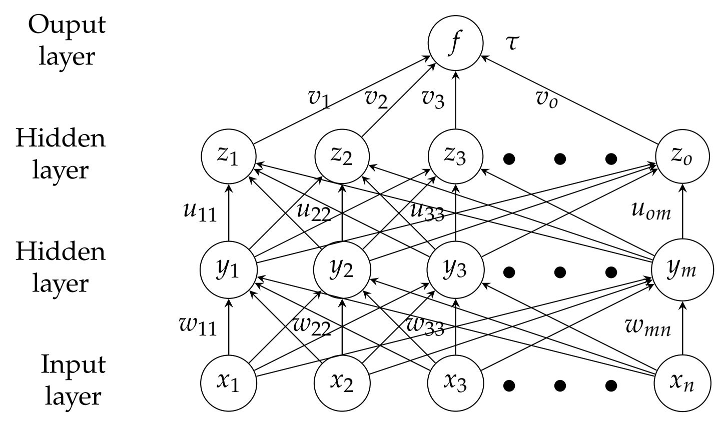

Figure 3: Three-layer network