Machine Learning Course - CS-433

Neural Nets – Training: SGD and Backpropagation

Dec 8, 2016

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

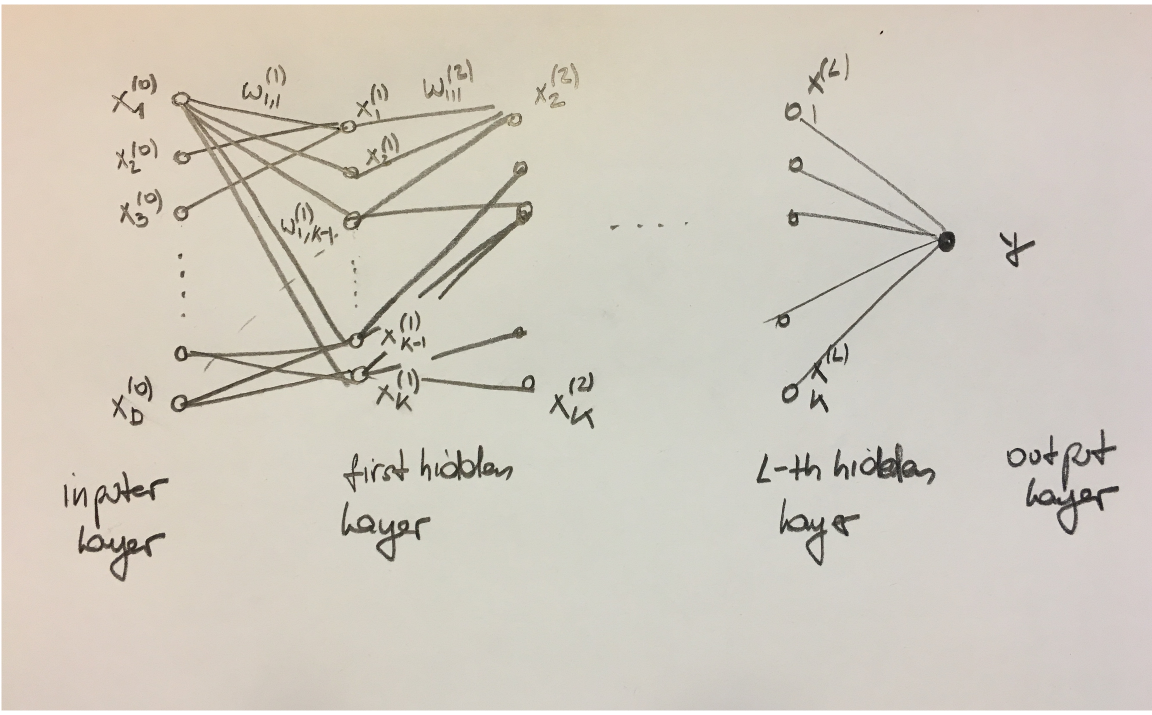

Figure 1: A neural network with one input layer, L hidden layers, and one output layer.

Motivation

Recall the structure of a neural network. For your convenience it is shown again in Figure 1. Assume that our task is regression. I.e., we have a training set $S_t = \{(y_n, \mathbf{x}_n)\}$. Let $f(\mathbf{x})$ be the function that is represented by the nn (including the last layer). I.e., $f(\mathbf{x})$ is the output of the nn.
Let us assume that we use our standard cost function

$$\mathcal{L} = \frac{1}{N}\sum_{n=1}^{n}(y_n - f(\mathbf{x}_n))^2,$$

where $(\mathbf{w}, \mathbf{b})$ is the vector of all parameters. In practice, we might want to add a regularization term to avoid overfitting, but this term is trivial to compute and to take into account. Therefore, we omit such a regularization term from our future discussion.

Given our training set S_t, our task is to train the network to minimize the cost function. Our go-to technique for training models is stochastic gradient descent. We have seen how it works for simple linear regression models but also for the matrix factorization problem. It is also a natural candidate for training neural nets and the current state-of-the art.
Recall that our set of parameters consists of all the weights of the net as well as the bias terms. As always when dealing with gradient descent we compute the gradient of the cost function for a particular input sample and then take a small step in the direction opposite to this gradient (assuming that we want to minimize the cost function).
As we will see, computing the derivative with respect a particular parameter is really just applying the *chain rule* of calculus. But since in general there are many parameters it would not be efficient to do this for each parameter individually. We therefore discuss how to compute all the derivatives jointly more efficiently. The algorithm for doing so is very natural and it is called *back propagation.*

Compact Description of Output

Let us start by writing down the output as a function of the input explicitly in compact form. It is natural and convenient describe the function that is implemented by each layer of the network separately at first. The overall function is then the composition of these functions.
Let $\mathbf{W}^{(l)}$ denote the *weight* matrix that connects layer $l-1$ to layer l. The matrix $\mathbf{W}^{(1)}$ is of dimension $D \times K$, the matrices $\mathbf{W}^{(l)}$, $2 \leq l \leq L$, are of dimension $K \times K$, and the

matrix $\mathbf{W}^{(L+1)}$ is of dimension $K \times 1$. The entries of each matrix are given by

$$\mathbf{W}_{i,j}^{(l)} = w_{i,j}^{(l)},$$

where we recall that $w_{i,j}^{(l)}$ is the weight on the edge that connects node i on layer $l-1$ to node j on layer l.
Further, let us introduce the *bias* vectors $\mathbf{b}^{(l)}$, $1 \leq i \leq L+1$, that collect all the bias terms. All these vectors are of length K, except the term $\mathbf{b}^{(L+1)}$, that is a scalar.
With this notation we can describe the function that is implemented by each layer in the form

$$\mathbf{x}^{(l)} = f^{(l)}(\mathbf{x}^{(l-1)}) = \phi((\mathbf{W}^{(l)})^\top \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}), \qquad (1)$$

where the (generic) activation function is applied pointwise to the vector.
The overall function $y = f(\mathbf{x}^{(0)})$ can then be written in terms of these functions as the composition

$$f(\mathbf{x}^{(0)}) = f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x}^{(0)}).$$

Cost Function

The cost function can be written as

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{n} (y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x}_n^{(0)}))^2.$$

Note that this cost function is a function of all weight matrices and bias vectors and that it is a composition of all the functions describing the transformation at each layer.

The Backpropagation Algorithm

In SGD we compute the gradient of this function with respect to one single sample. Therefore, we start with the function

$$\mathcal{L}_n = (y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x}_n^{(0)}))^2.$$

Recall that our aim is to compute

$$\frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}}, l = 1, \cdots, L+1,$$
$$\frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}}, l = 1, \cdots, L+1.$$

It will be convenient to first compute two preliminary quantities. The desired derivatives are then easily expressed in terms of those quantities.
Let

$$\mathbf{z}^{(l)} = (\mathbf{W}^{(l)})^\top \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)},$$

where $\mathbf{x}^{(0)} = \mathbf{x}_n$ and $\mathbf{x}^{(l)} = \phi(\mathbf{z}^{(l)})$, see (1). In words, $\mathbf{z}^{(l)}$ is the total input computed at the l-th layer before applying the activation function. These quantities are easy to compute by a *forward* pass in the network.
Further, let

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}_n}{\partial z_i^{(l)}}.$$

Let $\delta^{(l)}$ be the corresponding vector at level l. Whereas the quantities $\mathbf{z}^{(l)}$ where easy to compute by a forward pass, the quantities $\delta^{(l)}$ are easily computed by a *backwards* pass:

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}$$
$$= \sum_k \delta_k^{(l+1)} \mathbf{W}_{j,k}^{(l+1)} \phi'(\mathbf{z}^{(l)}).$$

In vector form we can write this as

$$\delta^{(l)} = (\mathbf{W}^{(l+1)})^\top \delta^{(l+1)} \odot \phi'(\mathbf{z}^{(l)}).$$

Now where we have both $\mathbf{z}^{(l)}$ and $\delta^{(l)}$ let us get back to our initial goal.
Note that

$$\frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{i,j}^{(l)}} = \underbrace{\frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}}}_{\delta_j^{(l)}} \underbrace{\frac{\partial z_j^{(l)}}{\partial w_{i,j}^{(l)}}}_{\mathbf{x}_i^{(l-1)}} = \delta_j^{(l)} \mathbf{x}_i^{(l-1)}.$$

In a similar manner,

$$\frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}} = \sum_k \frac{\partial \mathcal{L}_n}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial b_j^{(l)}} = \underbrace{\frac{\partial \mathcal{L}_n}{\partial z_j^{(l)}}}_{\delta_j^{(l)}} \underbrace{\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}}}_{1} = \delta_j^{(l)} \cdot 1 = \delta_j^{(l)}.$$

Let us now summarize the whole procedure.

Summary of Backpropagation Algorithm for Computing the Derivatives

We are given a nn with L hidden layers. All weight matrices $\mathbf{W}^{(l)}$ and bias vectors $\mathbf{b}^{(l)}$, $l = 1, \cdots, L+1$, are fixed. We are given in addition a sample $(y_n, \mathbf{x}_n)$. We want to compute the derivatives

$$\frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}}, \quad \frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}}, l = 1, \cdots, L+1,$$

where

$$\mathcal{L}_n = (y_n - f^{(L+1)} \circ \cdots \circ f^{(2)} \circ f^{(1)}(\mathbf{x}_n^{(0)}))^2.$$

Forward Pass: Set $\mathbf{x}^{(0)} = \mathbf{x}_n$. Compute for $l = 1, \cdots, L{+}1$,

$$\mathbf{z}^{(l)} = (\mathbf{W}^{(l)})^\top \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{x}^{(l)} = \phi(\mathbf{z}^{(l)}).$$

Backward Pass: Set $\delta^{(L+1)} = 2(y_n - \mathbf{x}^{(L+1)})\phi'(\mathbf{z}^{(l+1)})$. Compute for $l = L, \cdots 1$,

$$\delta^{(l)} = (\mathbf{W}^{(l+1)})^\top \delta^{(l+1)} \odot \phi'(\mathbf{z}^{(l)}).$$

Final Computation: For all parameters compute

$$\frac{\partial \mathcal{L}_n}{\partial w_{i,j}^{(l)}} = \delta_j^{(l)} \mathbf{x}_i^{(l-1)}, \frac{\partial \mathcal{L}_n}{\partial b_j^{(l)}} = \delta_j^{(l)}.$$

Now where we have the gradient with respect to all parameters, the SGD algorithm makes a small step in the direction opposite to the gradient, then picks a new sample $(y_n, \mathbf{x}_n)$, and repeats.