CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 18: Backpropagation

Lecturer: C. V. Jawahar *Date: DATE*

18.111 High Level Picture

Let us consider an MLP as a sequence/chain of computational blocks. (see figure 18.18). In practice, these blocks have a matrix multiplication and an activation function of the form $\mathbf{x}_{n+1} = \phi(\mathbf{W_n}\mathbf{x}_n)$. Here, $\mathbf{x}_n$ corresponds to the number of neurons in the n th layer. Note that the number of neurons in each layer may be different. This implies that $\mathbf{W}_n$ matrix need not be square. Needless to say, they could be different for each layer.

There is a loss layer at the end of the chain. This module computes the discrepancy of the last output ($\mathbf{x}_{p+1}$) with the expected value ($\mathbf{y}$) and compute a scalar loss measure.

The objective of learning is to find the parameters (i.e., $\mathbf{W}$ matrices) that minimize the loss.

Assuming that there are p layers, we have matrices $\mathbf{W}_1, \ldots \mathbf{W}_p$ that parameterize the neural network. In otherwise, we have that many parameters to learn.

Our gradient descent learning rule will allow us to learn in the form of

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^k - \eta \frac{\partial L}{\partial \mathbf{W}^k} \tag{18.40}$$

As in the previous gradient descent schemes, we can start with a random (or preferably a smart) initialization of the weight matrices in the zero iteration (initialization) and update it with every iteration k, until some convergence criteria is met.

However, the problem is not simple. The loss depends only on the $\mathbf{x}_{p+1}$ and the true prediction $\mathbf{y}$. Then how can the partial derivatives in equation 18.40 be nonzero?. On a closer look, we realize that $\mathbf{x}_{p+1}$ depends on the previous weight matrix $\mathbf{W}_p$, and also $\mathbf{x}_p$.

18.112 Derivatives

Computation of the partial derivatives is not that complex, if we use our familiar chain rule. We make an assumption at this stage:

1. **criteria -A** For each block, we know how to compute the partial derivative of the output with respect to that of input.
$$i.e., \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}$$

2. **criteria -B** For each block we know how to compute the partial derivative of the output with respect to that of the learnable parameters (say $\mathbf{W}$).
$$i.e., \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{W_n}}$$
Some of the blocks may also have learnable parameters θ other than weights. In such case, we also should know the
$$i.e., \frac{\partial \mathbf{x}_{n+1}}{\partial \theta}$$

We can compute the partial derivative of the loss with respect to any of the learnable parameters using the chain rule. This allows us to learn the weights using the gradient update rule in equation 18.40.

18.112.1 Loss Layer

Consider the final loss layer which computes loss from $\mathbf{x}_{p+1}$ and $\mathbf{y}$ for each sample. An example of the loss is

$$\mathcal{L} = \sum_{i=1}^{N} ||\mathbf{x}_{p+1} - \mathbf{y}||^2 = \sum_{i=1}^{N} [\mathbf{x}_{p+1} - \mathbf{y}]^T [\mathbf{x}_{p+1} - \mathbf{y}] \tag{18.41}$$

In this case $\frac{\partial \mathcal{L}}{\mathbf{x_p}}$ is easily computable. Note that $\mathbf{y}$ is constant/fixed. It is part of the data or gound truth.

Q: Do compute $\frac{\partial L}{\mathbf{x_p}}$ for three popular loss functions. (or loss functions that you think are meaningful.)

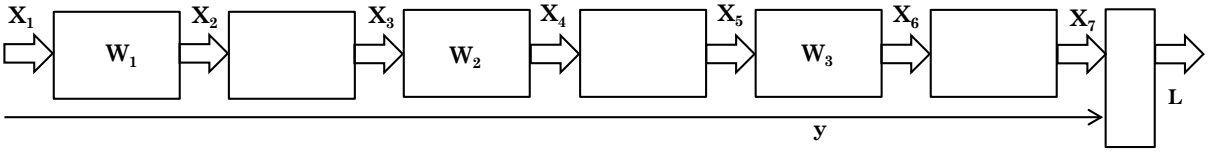

Figure 18.18: MLP as a sequence of computational blocks. Input $\mathbf{x}$ is same as $\mathbf{x_1}$. Output $\mathbf{x}_{p+1}$ is compared to the true output in the loss layer.

18.112.2 Typical Fully Connected Layer

A typical fully connected later (i.e., all neurons in layer k is connected to all the neurons in layer $k+1$) can be represented as

$$\mathbf{x}_{n+1} = \phi(\mathbf{W_n}\mathbf{x}_n)$$

To make the equations simpler, let us define a temporary variable $\mathbf{t}$, and rewrite the above as two steps.

$$\mathbf{t} = \mathbf{W_n}\mathbf{x}_n \tag{18.42}$$

$$\mathbf{x}_{n+1} = \phi(\mathbf{t}) \tag{18.43}$$

We need to compute $\frac{\partial \mathbf{x}_{n+1}}{\mathbf{W_n}}$ and $\frac{\partial \mathbf{x}_{n+1}}{\mathbf{x_n}}$.
These two are nothing but:

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{W_n}} = \phi'(\mathbf{t}) \cdot \frac{\partial \mathbf{t}}{\partial \mathbf{W_n}} \tag{18.44}$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x_n}} = \phi'(\mathbf{t}) \cdot \frac{\partial \mathbf{t}}{\partial \mathbf{x_n}} \tag{18.45}$$

All the partial derivates on the right side are straightforward to compute.

$\phi'(\mathbf{t})$ is a vector of $\phi'(t_i)$. i.e., element-wise evaluation of the derivative.

Note: (i) Look some where else for how the matrix vector derivates are computed. Tom Minka's notes shared in the past is worth. (ii) See discussions somewhere else for How to compute $\phi'()$ for some of the popular activation functions.

18.113 Forward and Backward Passes

We are given $(\mathbf{x_i}, \mathbf{y_i})$ $i = 1, \ldots, N$. Our objective is to learn the weight matrices $\mathbf{w}_i$.

18.113.1 Forward Pass

For each sample in the training data we can give $\mathbf{x}_i$ as input and go through it through a series of matrix multiplications and activations. Finally the network predicts $\mathbf{x}_{p+1}$. We compute the loss per sample using equation 18.41 or similar other loss equations. Finally the total loss $\mathcal{L}$ is computed as the sum of loss over all the samples.

Forward pass is straightforward. It involves many matrix multiplications. This leaves scope for parallelization and running on dedicated hardware at high speed.

18.113.2 Backward Pass

Example 1 Let us consider the situation, we want to update the weight matrix of the last block as

$$\mathbf{w}_p^{k+1} \leftarrow \mathbf{W}_p^k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$$

How do we compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$?

Chain rule helps us to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$ as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{W}_p}$$

The first term is available (see the text next to equation 18.41) and the next term is available with the definition of the block (see criteria B).

Example 2 Now let us try updating the weights in the last but one block.

$$\mathbf{w}_{p-1}^{k+1} \leftarrow \mathbf{W}_{p-1}^k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{p-1}}$$

Similar to the previous case, we can compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{p-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{x}_{p-1}} \cdot \frac{\partial \mathbf{x}_{p-1}}{\partial \mathbf{W}_{p-1}}$$

We already know the availability of the first and last term (from the previous example of updating $\mathbf{W}_p$). We also know that the middle term is available from our criteria-A.

Example 3 Now we can compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{x}_{p-1}} \cdot \ldots \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{W}_1}$$

The point to note in the backward computation is that the partial derivatives required for the computation is available already, if we have updated the weights backwards.

18.114 Backpropagation

The error backpropagation or backpropagation can be summarized as:

1. Initialize the network with random weights.
2. For all the samples, compute the output of the neural network $\mathbf{x}_{p+1}$
3. Compute the loss for the full batch of N samples as

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{x}_{p+1}^{i}, \mathbf{y}_i)$$

4. Adjust all the parameters (such as weight matrices) as

$$\theta^{k+1} \leftarrow \theta^{k} - \Delta\theta$$

or

$$\theta^{k+1} \leftarrow \theta^{k} - \eta\frac{\partial L}{\partial \theta}$$

5. Repeat steps 2-4 until convergence.

18.114.1 Refinements over backpropagation algorithm

Over years, backpropagation algorithm has been refined significantly with many minor but critical innovations. What all can change in the simple version we saw early?

1. **step 1:** Initialization can be smarter.
2. **step 2:** Computing loss over a full batch and updating it once is not the best.
3. **step 3:** Loss function can be different. There re many other loss functions available beyond MSE.
4. **step 4:** Update rule can be different. What we saw here is too simple.

18.115 Closer Look at the Derivatives (*)

Now that we had seen the larger picture of backpropagation and the chain rule for computing the derivatives, let us have a closer look at the equations 18.44 and 18.45.

$$t_n = W_n\mathbf{x}_n \quad ; \quad \mathbf{x}_n = \phi(\mathbf{t}_n)$$

$$\mathbf{x}_{n+1} : q \times 1 \quad ; \quad \mathbf{x}_n : p \times 1 \quad ; \quad W_n : q \times p \quad ; \quad \mathbf{t}_n : q \times 1$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n} = \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{t}_n}\frac{\partial \mathbf{t}_n}{\partial \mathbf{x}_n} \tag{18.46}$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial W_n} = \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{t}_n}\frac{\partial \mathbf{t}_n}{\partial W_n} \tag{18.47}$$

It should be noted that in equation 18.47, even though $\mathbf{t}_n$ is of dimension $q \times 1$ and W_n is dimension $q \times p$, the derivative $\frac{\partial \mathbf{t}_n}{\partial W_n}$ has a maximum of $q \times p$ non-zero values. This is because, the i^{th} element of $\mathbf{t}_n$ depends only on the i^{th} row of W_n. As such, for the purpose of equation 18.47, we assume that the derivative $\frac{\partial \mathbf{t}_n}{\partial W_n}$ is represented by a $q \times p$ matrix with the i^{th} row containing the derivative of the i^{th} element of $\mathbf{t}_n$ with respect to the i^{th} row of W_n. Also to be noted here is that the derivative $\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{t}_n}$ would be a $q \times q$ dimensional diagonal matrix, because the i^{th} element of $\mathbf{x}_{n+1}$ depends only on the i^{th} element of $\mathbf{t}_n$.

18.116 Training Process

18.117 More on Backpropagation

We had seen the back-propagation algorithm as one that iteratively minimizing the loss/error over the samples. We discussed the algorithm as two steps:

1. **Forward Pass:** Do a forward pass for all the samples and compute the loss over the training set.
2. **Backward Pass:** Do refine all the learnable parameters (weights) iteratively as:

$$\theta^{k+1} \leftarrow \theta^{k} - \eta\frac{\partial J}{\partial \theta}$$

To make the discussions and notations simple, we assume that all the learnable parameters are part of θ, and the loss/objective is $J(\theta)$. We use this notation throughout.

18.118 Stochasticity

In practice we do not compute the loss over all the samples and then update the parameters in one go. We do this over a randomly selected subset of the samples. This leads to stochastic mini batch backpropagation algorithm.

Note that the batch mode of the BP computes loss over all the samples. This is actually an approximation of the "true" gradient which we are not able to compute, since we do not know the analytic function form. If the true gradient can be approximated with sum of gradients over

a number of samples, thus approximation can be computed from a subset of the samples also. However, computing the gradient from a smaller set may have larger error than that computed from all the samples. However, this is much more efficient. Therefore, we can have BP implemented as batch, single sample and mini-batch. Minibatch is preferred.

The convergence and properties of stochastic gradient descent methods have been analyzed in both convex minimization and stochastic approximation. In many related areas, it has been shown that the stochastic gradient descent will converge to the batch (not stochastic) estimates in many practical situations.

18.118.1 Epochs and Iterations

In neural network literature, it is common to use the word epoch. In the neural network terminology, one epoch consists of one forward pass and one backward pass of all the training examples. However, as we discussed above, batch size (i.e., the number of training examples in one forward/backward pass) could be much smaller than the entire data. Though it is possible to randomly create the batch size every time, it becomes computationally efficient to create random batches once (in the beginning of the training) and use the same batches throughout. When the batch size is large, the memory requirement of the training could increase.

18.119 Sub-gradients

Another issue is the sub-gradient. many functions that we use in the modern deep neural networks are not truely differentiable. Eg. ReLU. How do we handle these?

Eg1: ReLu A popular activation function is ReLU

$$\phi(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Eg2: Hinge Loss Hinge loss has been a key component in the success of SVMs.

$$max(0, 1 - t \cdot y)$$

Though in these two cases, the function is not continuous, we can compute the derivatives at each point, except the point of discontinuity. This is done with the help of "subgradients". In mathematics, the subgradient, generalizes the derivative to convex functions which are not necessarily differentiable.

18.120 Homeworks

1. Consider a simple neural network with two input two hidden and one output neurons. Hidden nodes have tanh activation and output has sigmoid activation.

   There are four learnable parameters (weights). Let the first four be represented as $w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1$ and the second set of two be w_{11}^2 and w_{12}^2. Loss is MSE loss.

   Derive the gradient update rule for each of these six weights as:

   $$w_{ij}^k \leftarrow w_{ij}^k + ...$$

2. Create a set of 100 each samples in 2D in two classes (assume multivariate Gaussian; Data is not linearly separable).

   Randomly sample 80% for training and 20% for testing.

   - implement the above learning process (i.e., in Q1). Does it give satisfactory results?
   - improve your implementation by adding bias. Does it give satisfactory results?
   - Try out a popular implementation (in your favorite library). How does this result compare with the above? Discuss.
   - Show learning graphs in all the above cases.

3. Use MLP to classify 1 vs 2 in MNIST data that you have been using. Input is $28 \times 28 = 784$. Two hidden layer of the size 1000 and final output of size 1.

   Discuss why or whether this is effective.