Week 05 Quiz: Backpropagation Solutions

Name SID

November 29, 2020

Exercise 1

Last week in 16ML, you have seen the role of sigmoid function in logistic regression. Sigmoind function, and the more general softmax function, are important transformations in classification problem. Neural network can be used to solve classification problem, so you will see different applications of these functions in the later course materials. In this question, we want to model sigmoid function as a simple neural network that you can perform backprogation. We want you to get familar with what backprogagation is actually doing and make sure you understand the intuition behind it.

Consider a classification model given by parameter $w \in R^3$, and the input feature $x \in R^2$ with an intercept term:

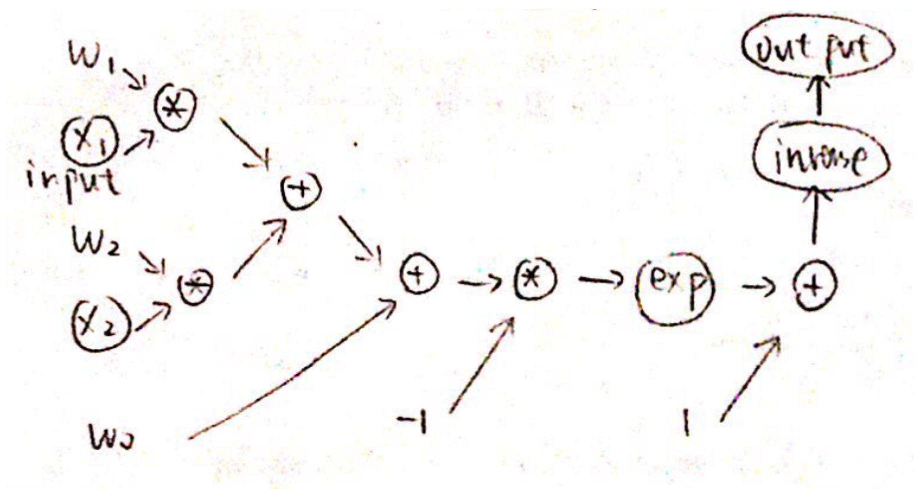
$$f(w, x) = \frac{1}{1 + \exp\{-(w_0 + w_1x_1 + w_2x_2)\}}$$

As you remember from logistic regression last week in 16ML, there's no close form solution so we usually perform gradient descent updates to iteratively approximate optimal parameter. Now we want to use backpropagation to speed up this process.

1. Draw computation graph of $f(w, x)$. Explicitly label where each weight value comes from.

   Hint: each neuron should correspond to either an algebraic step or an intermediate value.

   Solution:



2. Use chain rule in calculus to compute derivative of f with respect to w_0, w_1, w_2.

   Solution:

   This model we have intermediate value $-(w_0 + w_1x_1 + w_2x_2)$. We then use chain rule to compute

$$\frac{\partial f}{\partial w_j} = \frac{\partial f}{\partial(-(w_0 + w_1x_1 + w_2x_2))} \cdot \frac{\partial(-(w_0 + w_1x_1 + w_2x_2))}{\partial w_j)}$$

$$\begin{aligned}
\frac{\partial f}{\partial w_0} &= \frac{\exp\{-(w_0 + w_1x_1 + w_2x_2)\}}{(1 + \exp\{-(w_0 + w_1x_1 + w_2x_2)\})^2} \\
\frac{\partial f}{\partial w_1} &= \frac{x_1 \cdot \exp\{-(w_0 + w_1x_1 + w_2x_2)\}}{(1 + \exp\{-(w_0 + w_1x_1 + w_2x_2)\})^2} \\
\frac{\partial f}{\partial w_2} &= \frac{x_2 \cdot \exp\{-(w_0 + w_1x_1 + w_2x_2)\}}{(1 + \exp\{-(w_0 + w_1x_1 + w_2x_2)\})^2}
\end{aligned}$$

3. Suppose our loss function is square loss. Given data point (x, y) and prediction $\hat{y} = f(w, x)$,

$$L(y, \hat{y}) = (y - \hat{y})^2$$

   Compute derivative of loss function with respect to model parameters w_0, w_1, w_2.

   Solution:

$$\frac{\partial L}{\partial f} = -2(y - f(w, x))$$

   Use the chain rule, we plug in what we have computed in the previous section. Here, every term are already evaluated above, so for simplicity we don't need to copy and paste here.

$$\begin{aligned}
\frac{\partial L}{\partial w_0} &= \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial w_0} \\
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial w_1} \\
\frac{\partial L}{\partial w_2} &= \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial w_2}
\end{aligned}$$

   Note that data point (x, y) is known so out unknow parameters are w.

4. Given step size η, write down one step gradient descent update of w_0, w_1, w_2, to minimize loss function above. You may reuse what you computed in previous parts without re-deriving all of them.

   Solution:

$$w_0^{(t+1)} = w_0^{(t)} - \eta \cdot \frac{\partial L}{\partial w_0^{(t)}}$$

$$w_1^{(t+1)} = w_1^{(t)} - \eta \cdot \frac{\partial L}{\partial w_1^{(t)}}$$

$$w_2^{(t+1)} = w_2^{(t)} - \eta \cdot \frac{\partial L}{\partial w_2^{(t)}}$$

Exercise 2

Now you should be familiar with how to derive backpropagation in either simple or mode complicated neural network. In this question, we consider a neural network with two hidden layers.

Input layer: $x \in R^4$ and $x = [x_1, x_2, x_3, x_4]^T$. We don't add bias element to neural network in the input layer.

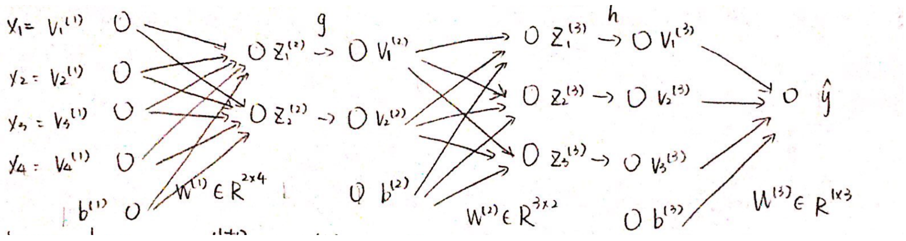
Hidden layer 1: $Z^{(2)} \in R^2$ and $Z^{(2)} = [Z_1^{(2)}, Z_2^{(2)}]^T$, with activation function g. We add a bias $b^{(2)} \in R$ to hidden layer 1, after applying activation function.

Hidden layer 2: $Z^{(3)} \in R^3$ and $Z^{(3)} = [Z_1^{(3)}, Z_2^{(3)}, Z_3^{(3)}]^T$, with activation function h. We add a bias $b^{(3)} \in R$ to hidden layer 2, after applying activation function.

Output layer: $y \in R$ is a real number

1. Draw the computation graph corresponding to the described neural network as above. Label weights and bias between each layer as $W^{(1)}$, $W^{(2)}$, $W^{(3)}$, $b^{(2)}$, $b^{(3)}$. Recall we don't add bias term to the input layer.

   Solution:

   Between layers we have relationship

   $$V^{(l+1)} = \sigma^{(l+1)}(Z^{(l+1)}) = \sigma^{(l+1)}(W^{(l)}V^{(l)} + b^{(l)})$$

   Given value at layer l $V^{(l)}$, compute $W^{(l)}V^{(l)} + b^{(l)}$ as $Z^{(l+1)}$. Apply activation function at layer $l+1$ $\sigma^{(l+1)}$ to get value at layer $l+1$.

2. Write down relationship between different layers. That is, how neural network model updates value from x to $\hat{y}$ step by step and layer by layer.

   Solution:

   We can read directly from the computation graph by following exactly what each step and each weight is doing.

   $$Z^{(2)} = W^{(1)}x + b^{(1)}$$

   $$V^{(2)} = g(Z^{(2)}) = g(W^{(1)}x + b^{(1)})$$

   $$Z^{(3)} = W^{(2)}V^{(2)} + b^{(2)}$$

   $$V^{(3)} = h(Z^{(3)}) = h(W^{(2)}V^{(2)} + b^{(2)})$$

   $$\hat{y} = W^{(3)}V^{(3)} + b^{(3)}$$

3. Suppose we use loss function $C(y, \hat{y})$, where $\hat{y}$ is prediction output from neural network. Assume loss function is differentiable and we know derivative of C.

   Compute derivative of C with respect to parameters in the last hidden layer, i.e. $W_1^{(3)}$, $W_2^{(3)}$, $W_3^{(3)}$, $b^{(3)}$.

   Solution:

   From structure of neural network, prediction $\hat{y}$ is given by

$$\hat{y} = \sum_{j=1}^{3} W_j^{(3)} V_j^{(3)} + b^{(3)}$$

   Take derivative with respect to model parameter $W_j^{(3)}$, we have

$$\frac{\partial \hat{y}}{\partial W_j^{(3)}} = V_j^{(3)}$$

   and

$$\frac{\partial \hat{y}}{\partial b^{(3)}} = 1$$

   Since loss function C only depends on output prediction $\hat{y}$ as variable, we use chain rule to regard $\hat{y}$ as an intermediate variable.

$$\frac{\partial C}{\partial W_j^{(3)}} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W_j^{(3)}} = \frac{\partial C}{\partial \hat{y}} V_j^{(3)} = \frac{\partial C}{\partial \hat{y}} h(Z_j^{(3)})$$

   for $j = 1, 2, 3$.

$$\frac{\partial C}{\partial b^{(3)}} = \frac{\partial C}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b^{(3)}} = \frac{\partial C}{\partial \hat{y}}$$

4. Then use your results from the previous part to compute derivative of C with respect to value in the hidden layer 2, i.e. $Z_1^{(3)}$, $Z_2^{(3)}$, $Z_3^{(3)}$.

   Solution:

   Use chain rule in calculus and regard $V_j^{(3)}$ as an intermediate variable.

$$\frac{\partial C}{\partial Z_j^{(3)}} = \frac{\partial C}{\partial V_j^{(3)}} \cdot \frac{\partial V_j^{(3)}}{\partial Z_j^{(3)}} = \frac{\partial C}{\partial V_j^{(3)}} \cdot h'(Z_j^{(3)}) = \frac{\partial C}{\partial \hat{y}} \cdot h'(Z_j^{(3)})$$

   for $j = 1, 2, 3$.

5. Use same process as question 3 and 4, compute derivative of C with respect to $W_{11}^{(2)}$, $W_{12}^{(2)}$, $W_{21}^{(2)}$, $W_{22}^{(2)}$, $W_{31}^{(2)}$, $W_{32}^{(2)}$, $b^{(2)}$, $Z_1^{(2)}$, $Z_2^{(2)}$.

   Solution:

   Basically, the big picture is same as we move from output layer to hidden payer 2 and to hidden layer 1, and finally to input layer. We will implement chain rule recursively on the neural network in order to find derivative of loss function on every single model parameter. You can check your work by repeating your calculation, but the following can be a suggestion of how this process should work.

$$\frac{\partial C}{\partial W_{ij}^{(2)}} = \frac{\partial C}{\partial Z_i^{(3)}} \cdot \frac{\partial Z_i^{(3)}}{\partial W_{ij}^{(2)}} = \frac{\partial C}{\partial Z_i^{(3)}} \cdot V_j^{(2)} = \frac{\partial C}{\partial Z_i^{(3)}} \cdot g(Z_j^{(2)})$$

for $i = 1, 2, 3$ and $j = 1, 2$.

$$\frac{\partial C}{\partial b_j^{(2)}} = \frac{\partial C}{\partial Z_j^{(3)}} \cdot \frac{\partial Z_j^{(3)}}{\partial b_j^{(2)}} = \frac{\partial C}{\partial Z_j^{(3)}}$$

for $j = 1, 2$

Now we should observe from what we get on the above expression, that we have common term $\frac{\partial C}{\partial Z_i^{(3)}}$ in expression of partial derivative of loss function C with respect to all neural network parameter in the same layer, that is with same superscript. To compute exact value, we just need to plug in what we have from previous part to the above expression. This will speed up our computation time complexity.

$$\frac{\partial C}{\partial Z_j^{(2)}} = \frac{\partial C}{\partial V_j^{(2)}} \cdot \frac{\partial V_j^{(2)}}{\partial Z_j^{(2)}} = \left(\sum_{k=1}^{3} \frac{\partial C}{\partial Z_k^{(3)}} \cdot \frac{\partial Z_k^{(3)}}{\partial V_j^{(2)}}\right) \cdot g'(Z_j^{(2)}) = \left(\sum_{k=1}^{3} \frac{\partial C}{\partial Z_k^{(3)}} \cdot W_{jk}^{2}\right) \cdot g'(Z_j^{(2)})$$

for $j = 1, 2$.

6. Finally, compute derivative of C with respect to model parameters in input layer and hidden layer 1, i.e. $W^{(1)}$ and $b^{(1)}$.

Solution:

We have numerical relationship between layers as

$$Z_i^{(2)} = \sum_{j=1}^{4} W_{ij}^{(1)} x_j + b_i^{(1)}$$

for $i = 1, 2$ Take derivative with respect to $W_{ij}^{(1)}$ on both sides we have

$$\frac{\partial Z_i^{(2)}}{\partial W_{ij}^{(1)}} = x_j$$

and

$$\frac{\partial Z_i^{(2)}}{\partial b_i^{(1)}} = 1$$

We then reuse values of $\frac{\partial C}{\partial Z_j^{(2)}}$ we computed above to find exact value of derivatives here.

$$\frac{\partial C}{\partial W_{ij}^{(1)}} = \frac{\partial C}{\partial Z_i^{(2)}} \cdot \frac{\partial Z_i^{(2)}}{\partial W_{ij}^{(1)}} = \frac{\partial C}{\partial Z_i^{(2)}} \cdot x_j$$

for $i = 1, 2$ and $j = 1, 2, 3, 4$

and

$$\frac{\partial C}{\partial b_j^{(1)}} = \frac{\partial C}{\partial Z_j^{(2)}} \cdot \frac{\partial Z_j^{(2)}}{\partial b_j^{(1)}} = \frac{\partial C}{\partial Z_j^{(2)}}$$

for $j = 1, 2, 3, 4$.

7. Now review what you did throughout the whole process above. When you manually write down values of derivative in each layer, do you encounter many repetitions? Based on your experience, how do you think backpropagation will speed up your work, when you are trying to compute derivative of loss function with respect to parameters $W_{ij}^{(l)}$ and $b_j^{(l)}$ in neural network?

   Solution:

   So far we only discuss multi-layer perceptron as structure of neural network. Our computation graph can generate certain copological sorting. In this case, we can recoganize contribution of each weight and bias parameter to the loss function as a simple formula. we also discover than local gradient of loss function at each layer will be fixed and reused throughout the later computation, when we want to find derivative with respect to previous layers that have contributions to this certain local neuron. So we only need to compute local gradient once for each neuron. And back propagation gives us linear time complexity in terms of number of neurons.