CS 6320: Computer Vision Spring 2019

Homework 5

Abhinav Kumar(u1209853)
abhinav.kumar@utah.edu

Due date: 15 April, 2019

1

[**25 points**] Install any deep learning package (Matconvnet, Caffe, Tensorflow, Pytorch) and test the MNIST digit recognition program. You don't have to understand the details of the program, but try to change the number of layers or other parameters and observe the change in the accuracy of the digit recognition. The following websites will be useful:

- `http://www.vlfeat.org/matconvnet/training/`
- `http://caffe.berkeleyvision.org/gathered/examples/mnist.html`
- `https://www.tensorflow.org/get_started/mnist/beginners`
- `https://pytorch.org/`

Show the outputs for 2 different parameter settings.

Architecture	**Model**	**Accuracy**
[784, 10, 10, 10, 10, 10]	1	94.93
[784, 10, 10, 10, 10]	2	94.35

Table 1: Fully connected architectures and their correspondig accuracies.

2

[25 points] Use backpropagation to compute to compute all the gradient $\frac{\partial C}{\partial w^l_{ij}}$ and $\frac{\partial C}{\partial b^l_i}$

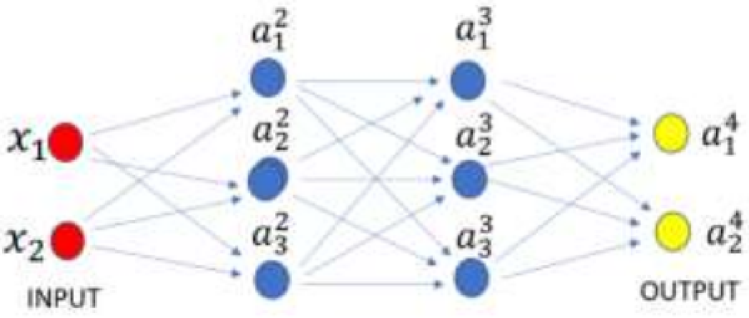

Figure 1: Neural Network

The activation function to be used is ReLU. You can use the following derivatives for ReLU

$$\frac{\partial ReLU'(z)}{\partial z} = ReLU'(z) = \begin{cases} 1, & \text{for } z > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

We use a quadratic cost function as given below

$$C = (y_1 - a^4_1)^2 + (y_2 - a^4_2)^2 \tag{2}$$

The weight matrices are:

$W^2 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$, $W^3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$, $W^4 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

The bias terms are:

$b^2 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $b^3 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $b^4 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

The inputs are $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and the ground truth outputs are $\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

To find the gradients, you will have to do the forward propagation once and then do the back-propagation once.

The feedforward equations are

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)} \tag{3}$$

$z^{(2)} = \begin{bmatrix} 3 \\ 3 \\ 3 \end{bmatrix}$ Since, all are positive, $a^{(2)} = z^{(2)}$

$z^{(3)} = \begin{bmatrix}10\\10\\10\end{bmatrix}$ They are all positive and therefore $a^{(3)} = z^{(3)}$

$z^{(4)} = \begin{bmatrix}31\\31\end{bmatrix}$ They are all positive and therefore $a^{(4)} = z^{(4)}$.

Cost $C = (1-31)^2 + (1-31)^2 = 1800$

Now, let us do the backpropagation one at a time -

$$\frac{\partial C}{\partial \mathbf{a}^{(4)}} = \begin{bmatrix}\dfrac{\partial C}{\partial a_1^{(4)}}\\ \dfrac{\partial C}{\partial a_2^{(4)}}\end{bmatrix} = \begin{bmatrix}2(a_1^{(4)}-y_1)\\2(a_2^{(4)}-y_2)\end{bmatrix} = \begin{bmatrix}60\\60\end{bmatrix}.$$ As $z^{(4)} > 0$, so we have $\frac{\partial C}{\partial \mathbf{z}^{(4)}} = \begin{bmatrix}60\\60\end{bmatrix} = 60\begin{bmatrix}1\\1\end{bmatrix}$

Now, $\boxed{\frac{\partial C}{\partial W^{(4)}} = \frac{\partial C}{\partial \mathbf{z}^{(4)}}\begin{bmatrix}a_1^{(3)} a_2^{(3)} a_3^{(3)}\end{bmatrix} = 600\begin{bmatrix}1&1&1\\1&1&1\end{bmatrix}}$ and $\boxed{\frac{\partial C}{\partial \mathbf{b}^{(4)}} = \frac{\partial C}{\partial \mathbf{z}^{(4)}} = 60\begin{bmatrix}1\\1\end{bmatrix}}$

Again, $\frac{\partial C}{\partial \mathbf{a}^{(3)}} = W^{(4)T}\frac{\partial C}{\partial \mathbf{z}^{(4)}} = 120\begin{bmatrix}1\\1\\1\end{bmatrix}$ Since, $z^{(3)} > 0$, $\frac{\partial C}{\partial \mathbf{z}^{(3)}} = 120\begin{bmatrix}1\\1\\1\end{bmatrix}$

Now, $\boxed{\frac{\partial C}{\partial W^{(3)}} = \frac{\partial C}{\partial \mathbf{z}^{(3)}}\begin{bmatrix}a_1^{(2)} a_2^{(2)} a_3^{(2)}\end{bmatrix} = 360\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}}$ and $\boxed{\frac{\partial C}{\partial \mathbf{b}^{(3)}} = \frac{\partial C}{\partial \mathbf{z}^{(3)}} = 120\begin{bmatrix}1\\1\\1\end{bmatrix}}$

Again, $\frac{\partial C}{\partial \mathbf{a}^{(2)}} = W^{(3)T}\frac{\partial C}{\partial \mathbf{z}^{(3)}} = 360\begin{bmatrix}1\\1\\1\end{bmatrix}$ Since, $z^{(2)} > 0$, $\frac{\partial C}{\partial \mathbf{z}^{(2)}} = 360\begin{bmatrix}1\\1\\1\end{bmatrix}$

Now, $\boxed{\frac{\partial C}{\partial W^{(2)}} = \frac{\partial C}{\partial \mathbf{z}^{(2)}}\begin{bmatrix}a_1^{(1)} a_2^{(1)}\end{bmatrix} = 360\begin{bmatrix}1&1\\1&1\\1&1\end{bmatrix}}$ and $\boxed{\frac{\partial C}{\partial \mathbf{b}^{(2)}} = \frac{\partial C}{\partial \mathbf{z}^{(2)}} = 360\begin{bmatrix}1\\1\\1\end{bmatrix}}$

3

[**25 points**] Consider a linear threshold unit T. Let $\{x_1, x_2, ..., x_n\}$ be the real inputs to the linear threshold unit and $\{w_1, w_2, ..., w_n\}$ be the learned real weights and let b be the bias term. Then the output from T will be 1 if $\mathbf{w}^T\mathbf{x} + b > 0$ and 0 otherwise.

Consider a function $f(x, y)$ that takes two real inputs $\{x, y\}$ and gives Boolean output 1 or 0 as shown in Fig 2. Use threshold activation functions to implement the following function $f(x, y)$ as shown below

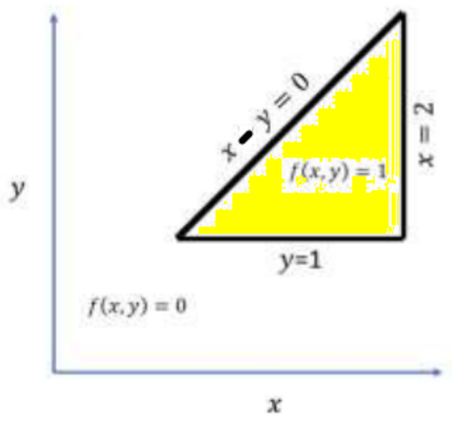

Figure 2: A simple function $f(x, y)$ that takes the value 1 inside the triangle and 0 otherwise.

Build the function using linear threshold units. You are free to use as many linear threshold units as you need. Manually come up with weights and biases for each linear threshold unit.

The shaded region is an AND of three lines. The hidden nodes in the violet simulate the three conditions while the output node is an AND gate between the three conditions. Figure 3 shows the architecture using linear threshold units.

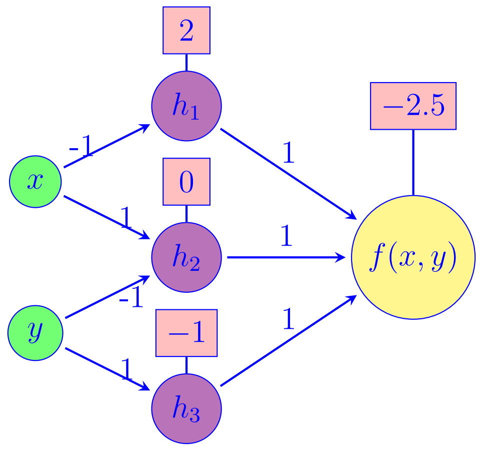

Figure 3: Neural Network for the same. The weights are shown on the links and the biases are shown in red squared boxes.

4

[25 points]
We are given a deep neural network DNN for alphabet classification. The first layer is the input layer with 784 neurons coming from 28×28 image pixels. There is only one channel in the input. The second layer is a convolution layer with 3 filters, each of size 7×7 with stride 2. Assume that the convolution layer uses bias terms. We do not use any padding for convolution. The output from the second layer passes through RELU activation unit (RELU(z) = max(z,0)). The third layer is a 2×2 max-pooling layer with no padding and stride 2. The fourth layer is a fully connected layer with 26 neurons for classifying the 26 alphabets. The final fifth layer is a softmax layer. The softmax layer produces 26 outputs, where each output could vary from 0 to 1. After training, an input of alphabet a would produce an output vector close to $([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...])$ for most of the time.

1. Show the dimensions of the layers 2 and 3.

   Layer 2: 7×7 convolution with stride 2.
   Since there is no padding in the second layer, the output image dimension is going to shrink. There will also be sub-sampling because of stride > 1. The dimensions of the second layer will be $11 \times 11 \times 3$.

   Layer 3: 2×2 maxpool with stride 2.
   The dimensions of the third layer will be $5 \times 5 \times 3$.

2. What is the total number of parameters (weights and biases) in the network?
   The number of weights in 2nd layer is $\#filters \times 7 \times 7 = 147$. The number of weights in the fourth layer is 75×26. Max-pool and softmax do not have any trainable parameters with them.

Layer	**Weights**	**Bias**
2	147	3
3	0	0
4	1950	26
5	0	0
Total	2097	29

Table 2: Fully connected architectures and their corresponding accuracies.

   From table 2, we have a total of 2126 parameters in the network.

3. Can you think of having fewer than 26 output neurons to classify 26 alphabets? In that case, would you need a softmax layer?

   Softmax is needed when we want to ensure exclusivity of labels in the output layer. In other words, when only one of the label should be on.

   If the problem is casted in the form of a classification problem, we will have $log_2 26$ (less than 26) output neurons (eg if we use binary encoding or gray encoding but then the outputs might not be one-hot but will be a multi-hot vector). So, then in that case, we do not need a softmax layer but can train as a multi-label problem with BCEWithLogits loss in pytorch.

If the problem is casted in the form of regression problem, we can have a single neuron whose different values correspond to different classifications. e.g.: level $[0, 0.1]$ can correspond to class 0, $[0.1, 0.2]$ can correspond to class 1 and so on. We do not need any softmax layer in this case again.