Homework 5
M1522.001000 Computer Vision (2018 Spring)
Due: at the beginning of the class in Monday June 11

1 Submitting Your Assignment

Your submission for this assignment consists of answers to two theory questions, the code for your MATLAB implementation and a short writeup describing any experimental results you made or things you did differently while implementing the assignment. You need to zip the following items, name it as `M1522.001000_HW5_(your student ID)#.zip`, and send it to TA's email.

Your submitted zip file should include the files arranged in this layout:

- `theory.txt` or `theory.pdf`
- Folder `result`
  - `experiments.pdf`: the writeup describing your experiments
- Folder `matlab`

Your zip file should be sent before the beginning of the class of the due date. Later than that, you will use one late day.

2 Theory Questions

Question 1: SVM (20 points)

Suppose that training examples are points in 2-D space. The positive examples are $X_+ = \{(1,1),(-1,-1)\}$. The negative examples are $X_- = \{(1,-1),(-1,1)\}$.

(a) [4 pts] Are the positive examples linearly separable from the negative examples? (i.e., Can you draw a line to separate the positive examples from negative examples)?

(b) [8 pts] Consider the feature transformation $\phi(x) = [1, x, y, xy]^T$, where x and y are the first and second coordinates of an example. Write down the transformed coordinates of X_+ and X_- (i.e., $\phi(X_+)$ and $\phi(X_-)$ for all four examples).

(c) [8 pts] Consider the prediction function $y(x) = w^T\phi(x)$. Give the coefficient w of a maximum-margin decision surface separating the positive from the negative examples. (hint: w is $[4 \times 1]$ vector, whose elements are only 0 or 1).

Question 2: MLP and CNN (10 points)

How convolutional neural network (CNN) is different from multi-Layer perceptron (MLP)? Describe more than two different features of CNN and two advantages of CNN. Also discuss if there exists a setting which makes CNN operation equivalent to MLP.

3 Programming

Question 3: K-Means (20 points) Implement the K-means algorithm in `Km.m` file, and tweak your K-means algorithm. For example, you can change hyper-parameters (number of centroids, number of iterations etc.) or preprocess input image. You can check segmentation results by running `demo.m` file. Describe your motivation and results for each modifications, and discuss why each modification improved (or weakened) the qualitative performace in **experiments.pdf**. Make sure `demo.m` and `Km.m` contain the final version of your codes.

Question 4: Mean-Shift (10 points)

Mean-shift algorithm is already implemented in `MeanShiftCluster.m` file. Tweak the given Mean-shift algorithm as you did with the K-means algorithm. You can check segmentation results by running `demo.m` file. Report experimental results and discussions in **experiments.pdf**. Make sure `demo.m` and `MeanShiftCluster.m` contain the final version of your codes.

Question 5: Multi-Layer Perceptron (30 points)

Multi-Layer Perceptron (or Neural Network) gives us a way of defining a complex, non- linear form of hypothesis h. MLP (or NN) is also a regression model that predicts a target value from a given input just like the softmax or logitic regression, except that it is a bit more complex than the others, but much powerful. Any layer of a neural network can be considered as an Affine Transformation followed by application of a non-linear activation function. A vector $x \in R^n$ is received as input and is multiplied with a weight matrix $W \in R^{m,n}$ to produce an output, to which a bias vector $b \in R^m$ may be added before passing the result through an activation function f (such as sigmoid σ):

$$\text{Input} = x, \text{ Output} = f(Wx + b) \tag{1}$$

The activation function f here is an element-wise operation, so the resulting output will be a vector as well. We usually denote this output as $a_l \in R^m$ and call it activation (meaning output value) of l-th layer. Then, the activation is fed to get the next layers output; this repetitive process is called forward propagation.

Thus, a neural network with multiple layers can be easily represented in the form of matrix computation. For example, the forward propagation equations of a simple neural network are as follows:

$$\text{Input} = a_0 = x \tag{2}$$
$$\text{Hidden Layer1 output} = a_1 = f_1(W_1 a_0 + b_1) \tag{3}$$
$$\text{Hidden Layer2 output} = a_2 = f_2(W_2 a_1 + b_2) \tag{4}$$
$$\text{Output} = a_3 = f_3(W_3 a_2 + b_3) \tag{5}$$

The parameters of this neural network model is the set of all the weight matrices W_l and the bias vectors b_l (i.e. $\theta = \{W_1, W_2, W_3, b_1, b_2, b_3\}$). As always, our job is to find the optimal parameter θ^* so that our model (5) can best describe the training

data. And again this could be done via minimizing some cost function $J(\theta)$ that we define for specific task. For example, the cost function for our task, MNIST digit recognition, can be defined as:

$$J(\theta) = CrossEntropyCost(y, softmax(a_3)) \tag{6}$$

To optimize the parameter set θ in our neural network model, we need the gradient of the cost function $J(\theta)$ with respect to each scalar parameters such as $W_l(i,j)$s and $b_l(i)$s. The process of computing the gradients using repetitive chain rule is called back propagation. Once the back propagation is computed, we can update each parameters with their gradient independently.

Derivation of this whole process is really boring. So, we have summarized the forward and backward propagation process here:

Suppose the neural network has L layers. a_0 is the input vector, a_L is the output vector and y is the true vector. The parameters are $\{W_1, W_2, ..., W_L, b_1, b_2, ..., b_L\}$ and the activation functions are $f_1, f_2, ..., f_L$.

Forward Propagation:

$$a_0 = x \tag{7}$$
$$a_l = f_l(W_l a_{l-1} + b_l) \tag{8}$$

Hypothesis and Cost:

$$h(x) = softmax(a_L) \tag{9}$$
$$J(\theta) = CrossEntropyCost(y, h(x)) \tag{10}$$

Backward Propagation:

$$\delta_L = softmax(a_L) - one - hot(y) \tag{11}$$
$$\delta_l = W_{l+1}^T \delta_{l+1} \circ f_l'(W_l a_{l-1} + b_l) \tag{12}$$

Gradient of weight and bias:

$$\frac{\partial J}{\partial W_l} = \delta_l x_l^T \tag{13}$$
$$\frac{\partial J}{\partial b_l} = \delta_l \tag{14}$$

Note that $\circ$ is element-wise multiplication, and one-hot() is a function that returns a vector of a single 1 bit at the position of y value and all the other bits are 0. For a simplicity, the summarization shown above cares only one training example $\{(x, y)\}$ If you see the `HW5c_mlp.m` file, the optimizer `minFunc()` is calling `mlp_cost()`. Your job is to implement the computation of forward propagation and cost function in the `mlp_cost()` function.

Implementation [15 pts]

Implement the forward propagation and cost function $J(\theta)$ in `mlp_cost.m` file. (Hint: be comfortable with wStack)

Experiment [15 pts]

Get more than 97% test accuracy. [10 pts] Tweak (or not) your neural network architecture in `HW5c_mlp.m`. Describe your model (number of layers, activation unit, regularization, etc..) in `experiments.pdf` when you achieve more than 97% test accuracy. What were the important parameters to you? Explain why.