Homework 5
M1522.001000 Computer Vision (2019 Spring)

Due: May 30 Thursday 01:59PM.

There are 4 theory questions and 2 **MATLAB** programming questions on this assignment, and 130 points in total.

Put your **code and writeup** into a directory called "`(studentid)-(yourname)-HW5`" and pack it into a `tar.gz` or `zip` named "`(studentid)-(yourname)-HW5`". For example, `201912345-gildonghong-HW5.tar.gz` or `201912345-gildonghong-HW5.zip`. Your writeup should be **typed** with **English**. Please do **not** attach your code to writeup.

Email your `tar.gz` or `zip` file ONLY to `ta.cv@vision.snu.ac.kr` with title as "`(studentid)-(yourname)-HW5`". Refer to the web page for the policies regarding collaboration, due dates, extensions, and late days.

Your homework should be formatted as following:

```
(studentid)-(yourname)-HW5
├─ writeup.pdf
└─ matlab
   ├─ k_means.m (modify this!)
   ├─ run_k_means.m
   ├─ run_mlp.m
   ├─ functions
   │  ├─ mlp_cost.m (modify this!)
   │  └─ ...
   ├─ library
   │  └─ ...
   ├─ *-ubyte (dataset, no need to include this)
   └─ 1.jpg 2.jpg (dataset, no need to include this)
```

1 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) 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) 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) 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 : Image Categorization with Bag of Visual Words (20 points)

We have learned that bag of visual words is commonly used in image categorization. The general idea is to represent an image as a set of features and categorize it with learned classifier. As a starting point, we can simply use SIFT feature extractor and use linear SVM for classification.

There are many other design choices to improve its performance and robustness. Explain intuitions and effects of following options: (i) using "dense multi-scale" SIFT feature, (ii) learning visual dictionary (*i.e.* Why not just use SIFT feature directly?), (iii) using "pyramid" spatial histogram, and (iv) applying "kernel trick"" to SVM classifier.

Question 3 : Softmax (20 points)

Prove that softmax is invariant to constant offsets in the input, that is, for any input vector $\mathbf{x}$ and any constant c,

$$\text{softmax}(\mathbf{x}) = \text{softmax}(\mathbf{x} + c) \tag{1}$$

where $\mathbf{x} + c$ means adding the constant c to every dimension of $\mathbf{x}$. Remember that

$$softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2}$$

Note: In practice, we make use of this property and choose $c = \max_i x_i$ *when computing softmax probabilities for numerical stability (i.e. subtracting its maximum element from all elements of* $\mathbf{x}$*).*

Question 4 : Backpropagation and Node Intuitions (20 points)

(a) Derive forward propagation and local gradients of the following graph. You must fill all the blanks in Figure 1 with calculated values and provide its deriviations.

$$a = x + y,\ b = \max(y, z),\ f = ab \tag{3}$$

$$x = 1,\ y = 2,\ z = 0 \tag{4}$$

(b) It is interesting to note that in many cases the backward-flowing gradient can be interpreted on an intuitive level. For example, the three most commonly used gates in neural networks (*add, multiply, max*), which are also used in Figure 1, all have very simple interpretations in terms of how they act during backpropagation.

By using your answer from Question 4(a), describe how we can interprete (*add, multiply, max*) gates on an intuitive level.

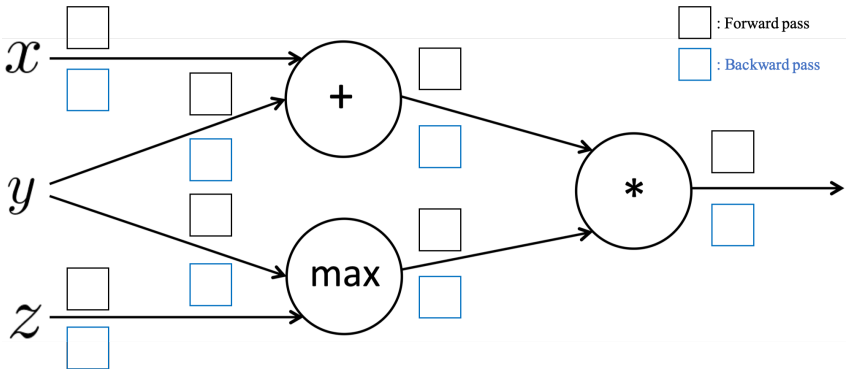

Figure 1: Simple graph with multiple nodes

2 Programming

Question 1 : K-Means Clustering (20 points)

Implement the K-means clustering algorithm in `k_means.m` file, and tweak your K-means algorithm. For example, you can change hyper-parameters (*e.g.* number of centroids, number of iterations and *etc.*) or preprocess input image. You can check segmentation results by running `run_k_means.m` file. Describe your motivation and results for each modifications, and discuss why each modification improved (or weakened) the qualitative performance. Make sure `run_k_means.m` and `k_means.m` contain the final version of your codes.

Question 2 : 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. 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):

$$\text{Input} = x, \text{ Output} = f(Wx + b) \tag{5}$$

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:

$$\text{Input} = a_0 = x \tag{6}$$

$$\text{Hidden Layer1 output} = a_1 = f_1(W_1 a_0 + b_1) \tag{7}$$

$$\text{Hidden Layer2 output} = a_2 = f_2(W_2 a_1 + b_2) \tag{8}$$

$$\text{Output} = a_3 = f_3(W_3 a_2 + b_3) \tag{9}$$

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.* $\theta = \{W_1, W_2, W_3, b_1, b_2, b_3\}$). Our job is to find the optimal parameter θ^* so that our model (9) 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:

$$J(\theta) = CrossEntropyCost(y, softmax(a_3)) \tag{10}$$

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 $W_l(i, j)$ and $b_l(i)$. 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.

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

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

Revision History

Revision	Date	Author(s)	Description
1.0	2019/05/16	TA	HW5 out