Homework 5
M1522.001000 Computer Vision (2017 Spring)

Due: June 5 Monday 10:59AM.

In this assignment, you will implement basic Machine Learning algorithms for a simple Computer Vision task - MNIST digit recognition. The main idea of this homework is to get familiar with their objective functions, computing gradients and optimizing the objectives over a set of parameters.

Specifically, you will learn how to implement the objective and gradients for 1) logistic (binary) regression, 2) softmax (multiclsass) regression, and 3) Multi-Layer Perceptron (MLP) algorithm. Understanding these basic tools will form the basis for more sophisticated algorithm that you will develop, such as Convolutionl Neural Network (CNN) and Recurrent Neural Network (RNN), in the future.

0 Submitting Your Assignment

There are 4 questions, 3 coding assignments, and 1 experiment in this homework (100 points in total). Your answer for the question should be written in **English** and stored in 'writeup.pdf' file. Your coding implementation should be written in each designated MATLAB file. Please do **not** attach MATLAB code in the 'writeup.pdf'.

Put your **writeup and code** in a directory called "`(studentid)-(yourname)-HW5`" and pack it into a `tar.gz` (or `zip`) file named "`(studentid)-(yourname)-HW5`" (e.g. `201721234-gildonghong-HW5.tar.gz` or `201721234-gildonghong-HW5.zip`). For example, Your homework folder should be formatted as following:

```
(studentid)-(yourname)-HW5  (floder)
├─ writeup.pdf
└─ matlab  (floder)
    ├─ functions  (floder)
    ├─ library  (floder)
    ├─ HW5a_logisticReg.m
    ├─ HW5b_softmaxReg.m
    └─ HW5c_mlp.m
```

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

1 Logistic Regression

Suppose we want to predict a binary target value $y \in \{0, 1\}$ from a input vector $x \in \mathbb{R}^n$. In logistic regression, we define a hypotheses (or function) h to try to predict the probability that a given example x belongs to the "1" class versus the probability that it belongs to the "0" class. Specifically, we try to learn a function of the form:

$$\begin{aligned} P(y = 1|x) = h_\theta(x) &= \frac{1}{1 + \exp(-\theta^\top x)} = \sigma(\theta^\top x) \\ P(y = 0|x) = 1 - P(y &= 1|x) = 1 - h_\theta(x) \end{aligned} \tag{1}$$

The function $\sigma(z) = 1/(1 + \exp(-z))$ is called "sigmoid" (or "logistic") function - it is an S-shaped function that "squashes" the value of $\theta^\top x$ into the range $[0, 1]$. Hence, we may interpret $h_\theta(x)$ as a probability. Our goal is to search for a weight (or parameter) vector $\theta \in \mathbb{R}^n$ so that the probability $P(y = 1|x) = h_\theta(x)$ is large when x belongs to the "1" class and small when x belong to the "0" class (so that $P(y = 0|x)$ is large). For a set of training example $\{(x^{(i)}, y^{(i)}) : i = 1, ..., m\}$, we can use the following cost function $J(\theta)$ to measure how well a given hypothesis h_θ does this:

$$J(\theta) = -\sum_i \left\{ y^{(i)} \log\big(h_\theta(x^{(i)})\big) + (1 - y^{(i)}) \log\big(1 - h_\theta(x^{(i)})\big) \right\} \tag{2}$$

Note that depending on whether $y^{(i)}$ is 0 or 1, only one of the two terms in (2) is non-zero for each training example. When $y^{(i)} = 1$ minimizing the cost function means we need to make $h_\theta(x^{(i)})$ large, and when $y^{(i)} = 0$ we want to make $1 - h_\theta(x^{(i)})$ large.

We now have a cost function that measures how well the hypothesis h_θ fits our training data. Hence, next step is to actually find the best choice of θ^* that minimizes $J(\theta)$. There exists many algorithms to do this (e.g. stochastic gradient descent, Newton's method, and conjugate gradient, and etc). However, in this homework, we already provided the optimization algorithm `minFunc()` for you. Therefore, all you need to know is that this algorithm requires a computation of the cost function $J(\theta)$ and it's gradient w.r.t θ:

$$\nabla_\theta J(\theta) = \left[\frac{\partial J(\theta)}{\partial \theta_1} \;\; \frac{\partial J(\theta)}{\partial \theta_2} \;\; \cdots \;\; \frac{\partial J(\theta)}{\partial \theta_n} \right]^\top, \text{ where } \frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} x_j^{(i)} (h_\theta(x^{(i)}) - y^{(i)}). \tag{3}$$

If you see the `HW5a_logisticReg.m` file, the optimizer `minFunc()` calls another function named `logistic_regression_cost()`. Your job is to implement the computation of $J(\theta)$ and $\nabla_\theta J(\theta)$ in the `logistic_regression_cost.m` file.

Question 1 Derive the equation (3) from the equation (2). [15 pts]
(*Hint*: think about one training example $\{x^{(1)}, y^{(1)}\}$ at first).

Implementation 1 Implement the computation of $J(\theta)$ and $\nabla_\theta J(\theta)$ in `logistic_regression_cost.m` file. [15 pts]
(*Hint*: be comfortable with vector representation of `X`, `y`, and `weight` at first)

2 Softmax Regression

In softmax regression setting, we are interested in multi-class classification (as opposed logistic regression, which only deals binary label), and so the label y can take on K different values, rather than two. Thus, in our training set $\{(x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)})\}$, we now have that $y^{(i)} \in \{1, 2, ..., K\}$. For example, in the MNIST digit recognition task, we have K=10 different classes.

Given a test input x, we want our hypothesis h to estimate the probability that $P(y = k|x)$ for each value of $k = 1, ..., K$. Thus, our hypothesis will output a K-dimensional vector (whose elements sum to 1) estimating the categorical probabilities. Concretely, our hypothesis $h_{\boldsymbol{\theta}}(x)$ takes the form:

$$h_{\boldsymbol{\theta}}(x) = \begin{bmatrix} P(y=1|x) \\ P(y=2|x) \\ \vdots \\ P(y=K|x) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp\big(\theta^{(1)\top} x\big) \\ \exp\big(\theta^{(2)\top} x\big) \\ \vdots \\ \exp\big(\theta^{(K)\top} x\big) \end{bmatrix} = \text{softmax}(\boldsymbol{\theta}^\top x) \tag{4}$$

Here, $\theta^{(1)}, \theta^{(2)}, ..., \theta^{(K)} \in \mathbb{R}^n$ are the parameters of our model. Notice that the term $1/(\sum_{j=1}^{K} \exp\big(\theta^{(j)\top} x\big))$ normalizes the distribution, so that it sums to one. For convenience, we will also write $\boldsymbol{\theta}$ to denote all the parameters of our model. When you implement softmax regression, it is convenient to represent $\boldsymbol{\theta}$ as a n-by-K matrix obtained by concatenating all $\theta^{(j)}$ into columns, so that $\boldsymbol{\theta} = [\theta^{(1)}, \theta^{(2)}, ..., \theta^{(K)}]$.

Then, the cost function for softmax regression is:

$$J(\boldsymbol{\theta}) = -\Bigg[\sum_{i}^{m} \sum_{k=1}^{K} 1\{y^{(i)} = k\} \log \frac{\exp\big(\theta^{(k)\top} x^{(i)}\big)}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} \Bigg] \tag{5}$$

In the equation above, $1\{\cdot\}$ is the "indicator function": $1\{\text{a true statement}\} = 1$, and $1\{\text{a false statement}\} = 0$. This cost function also has a special name called "cross entropy". The equation (5) is similar to the cost function of logistic regression. In fact, if $K = 2$, it is exactly the same as the equation (2), except the class index is $\{1, 2\}$ instead of $\{0, 1\}$. By minimizing this cost function w.r.t $\boldsymbol{\theta}$. we can find a model $h_{\boldsymbol{\theta}}$ that fits the best of our training examples.

To minimize $J(\boldsymbol{\theta})$ w.r.t $\boldsymbol{\theta}$, again we need it's gradient. The gradient is defined as:

$$\begin{aligned} \nabla_{\theta^{(k)}} J(\boldsymbol{\theta}) &= \begin{bmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1^{(k)}} & \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2^{(k)}} & \cdots & \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_n^{(k)}} \end{bmatrix}^\top, \\ \text{where } \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_l^{(k)}} &= \sum_{i=1}^{m} x_l^{(i)} \Big(\frac{\exp\big(\theta^{(k)\top} x^{(i)}\big)}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})} - 1\{y^{(i)} = k\} \Big). \end{aligned} \tag{6}$$

Hence, we can get n-by-K Jacobian matrix $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = [\nabla_{\theta^{(1)}} J(\boldsymbol{\theta}), \nabla_{\theta^{(2)}} J(\boldsymbol{\theta}), ..., \nabla_{\theta^{(K)}} J(\boldsymbol{\theta})]$. With this formulas on hand, we can now plug it into `minFunc` to get the optimal parameter $\boldsymbol{\theta}$ (recall the parameter is also a n-by-K). If you see the `HW5b_softmaxReg.m` file, the optimizer `minFunc()` calls `softmax_regression_cost()` function. You need to implement the computation of $J(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in the `softmax_regression_cost()` function.

Question 2 Derive the equation (6) from the equation (5). [15 pts]
(*Hint*: think about one training example $\{x^{(1)}, y^{(1)}\}$ at first. Use the chain rule.)

Implementation 2 Implement the computation of $J(\boldsymbol{\theta})$ and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in `softmax_regression_cost.m` file. [15 pts]
(*Hint*: read below)

There are a few tips that you want to know before you actually start implementing the softmax regression. Firstly, let's see the special case when softmax regression reduces to logistic regression. Specifically, when $K = 2$, the hypothesis of softmax regression outputs

$$h_{\boldsymbol{\theta}}(x) = \frac{1}{\exp(\theta^{(1)\top}x) + \exp(\theta^{(2)\top}x)} \begin{bmatrix} \exp\big(\theta^{(1)\top}x\big) \\ \exp\big(\theta^{(2)\top}x\big) \end{bmatrix}. \tag{7}$$

If we multiply $\exp\big(-\theta^{(2)\top}x\big)$ to both the numerator and denominator, we get

$$\begin{aligned} h_{\boldsymbol{\theta}}(x) &= \frac{1}{\exp((\theta^{(1)} - \theta^{(2)})^\top x) + \exp(\mathbf{0}^\top x)} \begin{bmatrix} \exp\big((\theta^{(1)} - \theta^{(2)})^\top x\big) \\ \exp\big(\mathbf{0}^\top x\big) \end{bmatrix} \\ &= \begin{bmatrix} \dfrac{\exp\big((\theta^{(1)} - \theta^{(2)})^\top x\big)}{1 + \exp((\theta^{(1)} - \theta^{(2)})^\top x)} \\ \dfrac{1}{1 + \exp((\theta^{(1)} - \theta^{(2)})^\top x)} \end{bmatrix} = \begin{bmatrix} 1 - \dfrac{1}{1 + \exp((\theta^{(1)} - \theta^{(2)})^\top x)} \\ \dfrac{1}{1 + \exp((\theta^{(1)} - \theta^{(2)})^\top x)} \end{bmatrix}. \end{aligned} \tag{8}$$

Thus, by replacing $\theta^{(2)} - \theta^{(1)}$ with a single parameter vector θ', we can see that softmax regression predicts the probability of one of the classes as $1/(1 + \exp\big(-(\theta')^\top x\big))$, and that of the other class as $1 - 1/(1 + \exp\big(-(\theta')^\top x\big))$, same as logistic regression. Also notice that, we can actually "eliminate" one parameter. If we generalize this, we can always eliminate one parameter by subtracting it from the other parameters, without harming the representational power of our hypothesis. Indeed, rather than optimizing over the $K \times n$ parameters, one can let the K-th parameter disappear and optimize only $(K-1) \times n$ parameters (this is why the `weight` variable in `softmax_regression_cost()` function has one less dimensionality).

Secondly, you might find that the `sub2ind()` function is useful for implementing the indicator function $1\{\cdot\}$ in the cost and the gradient formula. Suppose we have a matrix, A and we want to extract a single element from each row, where the column of the element to be extracted from row i is stored in $y(i)$, where y is a row vector. We can use the `sub2ind()` function like this:

```
Index = sub2ind(size(A), 1:size(A,1),y);
values = A(Idex);
```

Lastly, you may also want to use `bsxfun()` function to speed up the computation of repetitive multiplication or division of a matrix by a vector to the specific direction.

3 Multi-Layer Perceptron

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 $x \in \mathbb{R}^n$ is received as input and is multiplied with a weight matrix $W \in \mathbb{R}^{m,n}$ to produce an output, to which a bias vector $b \in \mathbb{R}^m$ may be added before passing the result through an activation function f (such as sigmoid σ):

$$\text{Input} = x, \quad \text{Output} = f(Wx + b) \tag{9}$$

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 $a_l \in \mathbb{R}^m$ and call it "activation" (meaning output value) of l-th hidden layer. Then, the activation is fed to get the next layer's 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:

$$\begin{aligned}
\text{Input} &= a_0 = x \\
\text{Hidden Layer1 output} &= a_1 = f_1(W_1 a_0 + b_1) \\
\text{Hidden Layer2 output} &= a_2 = f_2(W_2 a_0 + b_2) \\
\text{Output} &= a_3 = f_3(W_3 a_2 + b_3)
\end{aligned} \tag{10}$$

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. $\boldsymbol{\theta} = \{W_1, W_2, W_3, b_1, b_2, b_3\}$). As always, our job is to find the optimal parameter $\boldsymbol{\theta}^*$ so that our model (10) can best describe the training data. And again this could be done via minimizing some cost function $J(\boldsymbol{\theta})$ that we define for specific task. For example, the cost function for our task, MNIST digit recognition, can be defined as:

$$J(\boldsymbol{\theta}) = \text{CrossEntropyCost}(y, \text{softmax}(a_3)) \tag{11}$$

The cost equation (11) is exactly same as the equation (5), except the term $\boldsymbol{\theta}^\top x$ in equation (4) is now substituted with the last output $a_3 = f_3(W_3 a_2 + b_3)$.

To optimize the parameter set $\boldsymbol{\theta}$ in our neural network model, we need the gradient of the cost function $J(\boldsymbol{\theta})$ 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:

$$\begin{aligned} a_0 &= x \\ a_l &= f_l(W_l a_{l-1} + b_l) \end{aligned} \tag{12}$$

Hypothesis and Cost:

$$\begin{aligned} h(x) &= \text{softmax}(a_L) \\ J(\boldsymbol{\theta}) &= \text{CrossEntropyCost}(y, h(x)) \end{aligned} \tag{13}$$

Backward Propagation:

$$\begin{aligned} \delta_L &= \text{softmax}(a_L) - \text{one-hot}(y) \\ \delta_l &= W_{l+1}^\top \delta_{l+1} \circ f_l^{'}(W_l a_{l-1} + b_l) \end{aligned} \tag{14}$$

Gradient of weight and bias:

$$\begin{aligned} \frac{\partial J}{\partial W_l} &= \delta_l x_{l-1}^\top \\ \frac{\partial J}{\partial b_l} &= \delta_l \end{aligned} \tag{15}$$

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 3 Implement the "forward propagation" and and "cost function $J(\boldsymbol{\theta})$" in `mlp_cost.m` file. (*Hint*: be comfortable with `wStack`) [10 pts]

Question 3 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. [10 pts]

Question 4 AlexNet(One of CNN) CNN code is 4096 dimension vector. Explain how to get 4096 dimension vector. Your answer should include explanation of all convolution and pooling layer with dimension calculation. [10 pts]

Experiment 1 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 **writeup.pdf** when you achieve more than 97% test accuracy. What were the important parameters to you? Explain why.

Experiment 2 (Optional) Get more than 98% "test" accuracy. [Bonus 10 pts]
Now, don't hesitate to implement all other techniques you can try. Describe your method and NN model specifically in **writeup.pdf** when you achieve more than 98% test accuracy. Save the model (or functions) so that we can fetch it later.