CSC 298/578 Deep Learning and Graphical Models: Homework 1

Term: Spring 2017
Instructor: Dr. Chenliang Xu
TA: Haofu Liao
Due Date: March 6, 2017 Midnight
Version: 1.0 (February 6, 2017)

Constraints: This assignment is to be carried out independently and without consulting online resources.

Problem 1 (15): Independence Properties
These are the first two parts of question 2.7 from KF.
Be sure to consider the definition of conditional independence when proving these properties; note that $P(\mathbf{X}|\mathbf{Z} = \mathbf{z})$ is undefined when $P(\mathbf{Z} = \mathbf{z}) = 0$.

1. (5) Prove that the Weak Union property holds for any probability distribution P.

$$\textbf{Weak Union:} \qquad (\mathbf{X} \perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z}) \Longrightarrow (\mathbf{X} \perp \mathbf{Y} \mid \mathbf{W}, \mathbf{Z}) \tag{1}$$

2. (5) Prove that the Contraction property holds for any probability distribution P.

$$\textbf{Contraction:} \qquad (\mathbf{X} \perp \mathbf{W} \mid \mathbf{Z}, \mathbf{Y}) \& (\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}) \Longrightarrow (\mathbf{X} \perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z}) \tag{2}$$

3. (5) Prove that the Intersection property holds for any probability distribution P. Assume that $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, and $\mathbf{W}$ are mutually disjoint.

$$\textbf{Intersection:} \qquad (\mathbf{X} \perp \mathbf{Y} \mid \mathbf{Z}, \mathbf{W}) \& (\mathbf{X} \perp \mathbf{W} \mid \mathbf{Z}, \mathbf{Y}) \Longrightarrow (\mathbf{X} \perp \mathbf{Y}, \mathbf{W} \mid \mathbf{Z}) \tag{3}$$

Problem 2 (10): Logistic Regression
Recall that logistic regression defines a hypothesis function $h_\theta(x) = g(\theta^T x)$, where $g(z) = \frac{1}{1+e^{-z}}$ is a sigmoid function. Given a training set of m examples, i.e., $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, the goal is to find θ that minimizes the following cost function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} log h_\theta(x^{(i)}) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))] \ . \tag{4}$$

Please show that if we use batch gradient descent, it has the same parameter update rule as linear regression:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \ , \tag{5}$$

where α is a learning rate to be set.

Problem 3 (15): Regularizing 2D Logistic Regression

1. (3) Consider the training data in Figure 1, where we fit the logistic regression model $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$. Suppose we minimize the cost function as defined in Eq. 4 (Problem 2). Sketch a possible decision boundary corresponding to $\theta^* = \operatorname{argmin}_\theta J(\theta)$. (Copy the figure first (a rough sketch is enough), and then superimpose your answer on your copy, since you will need multiple versions of this figure). Is your answer (decision boundary) unique? How many classification errors does your method make on the training set?

2. (4) Now suppose we regularize only the θ_0 parameter, i.e., we minimize

$$J_0(\theta) = J(\theta) + \lambda \theta_0^2 \ . \tag{6}$$

Suppose λ is a very large number, so we regularize θ_0 all the way to 0, but all other parameters are unregularized. Sketch a possible decision boundary. How many classification errors does your method make on the training set?

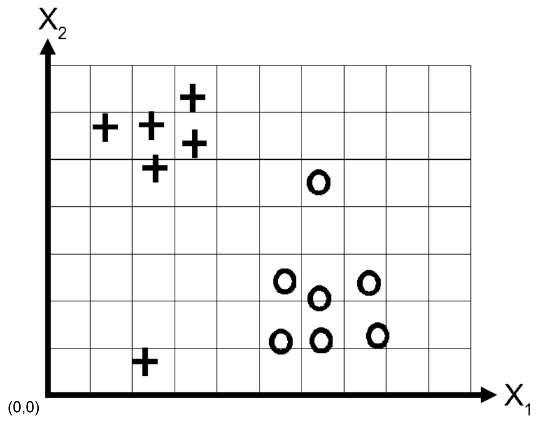

Figure 1: Data for 2D logistic regression question.

3. (4) Now suppose we heavily regularize only the θ_1 parameter, i.e., we minimize

$$J_1(\theta) = J(\theta) + \lambda\theta_1^2 \ . \tag{7}$$

   Sketch a possible decision boundary. How many classification errors does your method make on the training set?

4. (4) Now suppose we heavily regularize only the θ_2 parameter. Sketch a possible decision boundary. How many classification errors does your method make on the training set?

Problem 4 (60): Coding Neural Network and Backpropagation
Source: Haofu Liao

Overview In this assignment you will write your own code to train a neural network (NN) for the task of handwritten digit recognition.

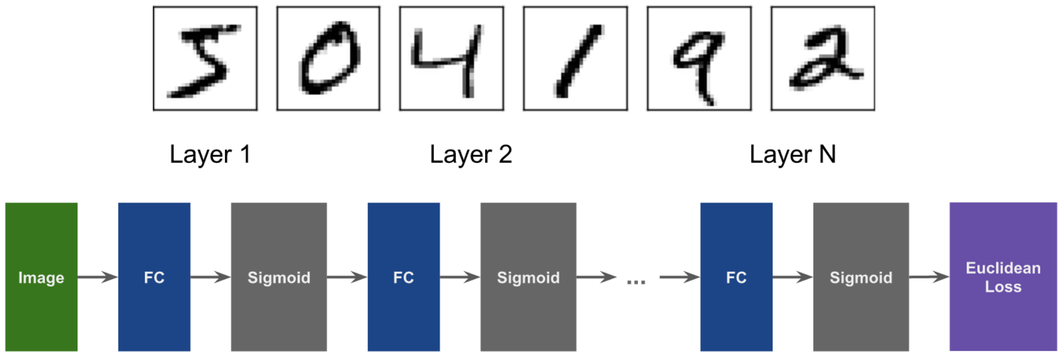

Figure 2: Network Structure

We have laid out a framework and provided initial code to get you started. In general, this homework is composed of two parts: 1) Write the forward and backward code for each layer and implement the backpropagation algorithm through the layer functions. 2) Design and train an NN to classify handwritten digit images from MNIST dataset. Specifically, you will implement a Stochastic Gradient Descent (SGD) algorithm for an NN using backpropagation.

Neural Network The structure of the network is shown in Figure 2. Your network will have N layers with each layer has a "fully connected" layer followed by a "sigmoid" activation function. The loss function you will use is the "euclidean" loss. The computation performed by each layer are:

- Fully Connected layer: $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$

- Sigmoid function: $y = \frac{1}{1+e^{-x}}$
- Euclidean loss: $L = \frac{1}{2}||\mathbf{y}^* - \mathbf{y}||$

You are required to implement both forward and backward computation of each layer.

Files The framework we provide you contains the following files.

- **demo.py:** A demo that demonstrates the usage of the framework. You may use it to test your implementation.
- **graph.py:** This file contains the **Graph** class which is used to hold the entire network structure.
- **layer.py:** Any layer function goes here. Each layer function is implemented as a class with *forward* and *backward* member functions. Only has the **FullyConnected** class and **Sigmoid** class for now. You should implement the *forward* and *backward* functions for theses classes.
- **loss.py:** Any loss function goes here. Each loss function is implemented as a class with *forward* and *backward* member functions. Only has the **Euclidean** class for now. You should implement the *forward* and *backward* function for this class.
- **optimization.py:** Any optimization method goes here. Each optimization method should contain at least a *compute_gradient* method for backpropagation and a *optimize* method for the optimization on the input dataset. Only has the **SGD** class for now. You should implement the *compute_gradient* method and *optimize* method for this class.
- **mnist_loader.py:** Contains the helper functions for loading and reading the mnist dataset.
- **network.py:** This file contains the **Network** class for training and testing a neural network.

Variable Definitions We have structured the code for the layers such that they all follow a similar format. The definition of some variables are as follows.

Parameters:

- `W` ($\mathbf{W}$) - The weights of the layer. An n-by-m matrix where m is the input size and n is the output size. Does not apply to activation functions (sigmoid, RELU, softmax, etc.).
- `b` ($\mathbf{b}$) - The biases of the layer. A n-by-1 vector where n is the output size. Does not apply to activation functions (sigmoid, RELU, softmax, etc.).

Inputs:

- `x` ($\mathbf{x}$) - The input to the layer.
- `dv_y` ($\nabla_y L$) - The partial derivative of the loss with respect to each element in the output matrix.

Outputs:

- `dv_x` ($\nabla_x L$) - The derivative of the loss with respect to the input. It is calculated by taking the derivative of the output with respect to the input $\frac{dy}{dx}$ and multiplying by `dv_y`.
- `dv_W` ($\frac{dL}{dW}$) - The partial derivative of the loss with respect to each element in $\mathbf{W}$. Does not apply to activation functions (sigmoid, RELU, softmax, etc.).
- `dv_b` ($\frac{dL}{db}$) - The partial derivative of the loss with respect to each element in $\mathbf{b}$. Does not apply to activation functions (sigmoid, RELU, softmax, etc.).

Grading Guideline

1. Layers and loss (20): Implement the *forward* and *backward* functions for the **FullyConnected**, **Sigmoid** and **Euclidean** classes.
2. Optimization (20): Implement the *compute_gradient* and *optimize* for the **SGD** class.

3. Training (20): Do some basic quantitative analysis for your training. You may try to figure out what affect the training time and performance. Below are a list of possible options you may try. Include at least two in your report.
    - Learning rate, protocols for lowering the learning rate as you train, i.e., you may try to lower your learning rate after training some epochs.
    - Network depth, the effect of adding or removing layers.
    - Layer width, changing the size of the layers by adding/subtracting neurons.
    - Weight initialization. Currently we are using Gaussian initialization. You may change the initialization method for the parameters by your own.

Submission Process:

Please submit your homework in a *zip* file with the following content.

- code/ - your implementation of the framework we provided.
- report.pdf - the report of coding question; quantitative analysis for your training. You may include any figures that you think best describe your findings.
- homework.pdf - the scanned file of your answers to analytical questions 1-3.
- Do **NOT** include the ”data/” folder of the framework.

Grading and Evaluation: The credit for each problem in this set is given in parentheses at the stated question (sub-question fraction of points is also given at the sub-questions). Partial credit will be given when appropriate.