Homework Assignment 5

COGS 181: Neural Networks and Deep Learning

Due: November 26, 2017, 11:59pm
Instructions:

1. Please answer the questions below, attach your code, and insert figures to create a pdf file; submit your file to TED (ted.ucsd.edu) by 11:59pm, 11/26/2017. You may search information online but you will need to write code/find solutions to answer the questions yourself.

Late Policy: 5% of the total points will be deducted on the first day past due. Every 10% of the total points will be deducted for every extra day past due.

Grade: ____ out of 100 points

Image Classification on MNIST dataset

In this homework, we are going to train a multilayer perceptron (MLP) and a convolutional neural network (CNN) on MNIST dataset, respectively.

Dataset Description

MNIST handwritten digit dataset contains a training set and a test set. The training set has 10 classes of around 6000 samples each, and each class stands for a handwritten digit from 0 to 9. The test set also has 10 classes of around 1000 samples each. Each data sample is a 28×28 image, and pixel values are 0 to 255. 0 means background (white), 255 means foreground (black)

Download MNIST

The dataset is available to everyone to download, and the link is http://yann.lecun.com/exdb/mnist/. There are 4 data files you need to download from the link. Or use the method from homework 4.

File Name	Data
train-images-idx3-ubyte.gz	training set images
train-labels-idx1-ubyte.gz	training set labels
t10k-images-idx3-ubyte.gz	test set images
t10k-labels-idx1-ubyte.gz	test set labels

Softmax Regression

In hw3 & hw4, we have applied logistic regression to learn a binary classifier. Here, we will introduce the softmax regression with the cross-entropy loss for multiclass classification tasks.
Softmax regression is a generalization of logistic regression for multiple classes. Suppose the dataset contains K classes, and N data samples in total. Each data sample is represented as a vector as $\mathbf{x}_i$, and its corresponding class label is y_i.

$$p(y_i|\mathbf{x}_i) = \frac{\exp\{\mathbf{w}_{y_i}^T \mathbf{x}_i\}}{\sum_{k=1}^{N} \exp\{\mathbf{w}_k^T \mathbf{x}_i\}}. \tag{1}$$

where $p_k^{(i)}$ is the predicted probability that $\mathbf{x}^{(i)}$ is in class k. The weight matrix contains k weight vectors, in which the k^{th} weight vector stands for the k^{th} connection to the output. (For the sake of simplification, we absorb the bias term into the weight matrix $\mathbf{W}$.)

Like what we have done in previous homework, we define the cross-entropy loss function $\mathcal{L}(\mathbf{W})$

$$\mathcal{L}(\mathbf{W}) = -\sum_{i=1}^{N} \ln p(y_i|\mathbf{x}_i) \tag{2}$$

By applying gradient descent algorithm to minimize the loss function $\mathcal{L}(\mathbf{W})$, we are able to find a decent solution for the classification task.

1 Multilayer Perceptron (40 points)

We are going to learn a multilayer perceptron to classify the MNIST dataset. You are supposed to choose one of the deep learning platforms that you prefer, and train a multilayer perceptron on the MNIST dataset. Here are examples:

Tensorflow	Aymeric Damien's Implementation

In this homework you should choose one platform and use it for all your experiments.

1. Build a multilayer perceptron which has 1 input layer, 1 hidden layer, and 1 output layer, and the hidden layer has **300 hidden units**. The activation function we are going to use for each of the hidden units is the sigmoid function. The formulation is shown below:

$$p_k^{(i)} = \frac{\exp\left(\mathbf{w}_k^T \mathbf{q}^{(i)}\right)}{\sum_{k'} \exp\left(\mathbf{w}_{k'}^T \mathbf{q}^{(i)}\right)} \tag{3}$$

$$\mathbf{q}^{(i)} = \sigma(\mathbf{c}^T \mathbf{x}^{(i)}) \tag{4}$$

   where $\sigma(\cdot)$ is the sigmoid function, and it is applied to each entry of the input matrix or vector. $\mathbf{c}$ is a weight matrix that connects the input layer to the hidden layer, and $\mathbf{w}$ is also a weight matrix that connects the hidden layer to the output layer. For simplification, we absorb the bias terms into the associated weight matrices.

2. Train the multilayer perceptron we defined above on the MNIST training set with the stochastic gradient algorithm to minimize the cross-entropy loss function in Eq.(2).

3. Report the hyperparameter setting in your experiments, which includes, at least, batch size and learning rate, and the plot of training accuracy and the test accuracy vs. the number of training iterations, and the final accuracy on the test set.

4. Compose a small training set which is a subset of the original MNIST training set. You need to pick up the first 1000 samples from each class to compose the small training set. Thus, the small training set should contain 10 classes of 1000 samples each.

5. Train the multilayer perceptron on the small training set, and report the plot the training accuracy and the test accuracy vs. the number of training iterations, and the final accuracy on the test set.

6. Compare your plots from 3. and 5., what do you find?

2 Convolutional Neural Networks (40 points)

In this question, we are going to build a convolutional neural network (ConvNet) to classify the MNIST dataset.

Tensorflow	https://www.tensorflow.org/get_started/mnist/pros#build_a_multilayer _convolutional_network

Or you can design your own ConvNet.

1. Build your ConvNet for MNIST dataset. Report your network structure, including the kernel size, stride, number of the filters, etc. (Don't build a too deep ConvNet, it will take you massive time to train on CPU.)

2. Report your hyperparameter settings, including, at least, batch size, learning rate, the strategy for decreasing the learning rate if you have, regularization if you have.

3. Train your ConvNet with **cross-entropy loss function** on the training set, and report the plot of the training loss (not accuracy) and test accuracy vs. the number of the training iterations, and the final accuracy on the test set.

4. Train your ConvNet with **cross-entropy loss function** on the **small** training set we composed in previous question, and report the plot and the final accuracy on test set.

5. Compare your results from 3. and 4., what do you find? (In terms of the rate of convergence, the final accuracy, and overfitting issue.)

3 Recurrent Neural Network (30 points)

In this section, you will reconstruct sine curve using recurrent neural network (RNN). The goal of this question is, given the sequence $A = (a_k, a_{k+1}, ..., a_l)$, we need to predict the "next sequence" $A' = (a_{k+1}, a_{k+2}, ..., a_{l+1})$.

Tensorflow	https://www.tensorflow.org/tutorials/recurrent

Instead of LSTM unit in the tutorial, we will use RNN unit for our experiments (tf.contrib.rnn.BasicRNNCell). Given a sequence $A = (a_1, a_2, ..., a_k)$, RNN takes a contiguous subsequence as its input $I = (a_m, a_{m+1}, ..., a_{m+n})$ where n is the number of time steps in your RNN. The output of $I = (i_1, i_2, i_3, ..., i_n)$ will be $O = (o_1, o_2, ..., o_n)$, where each $o_j = tanh(W_j^T \cdot i_1 + W_{j'}^T \cdot o_{j-1})$. For this homework, on the top of each o_j, there is a fully connected layer W_h with $tanh(*)$. The final output of each state will be $\hat{y}_j = tanh(W_h^T o_j)$. W_h combines all the features from one RNN unit and $tanh(*)$ enforces the output value to be in the range of $[-1, 1]$. Since sine curve values are continuous values, we optimize this model on MSE loss (mean squared error).
The data required for this question is very simple:

```
data = np.array([np.sin(n) for n in range(1000)])
label = np.array([np.sin(n) for n in range(1,1001)])
```

1. Draw the diagram of this RNN (unroll 3 states), including inputs, outputs, loss, fc layer and activation functions.

2. Train your RNN ($tf.nn.static_rnn$) with the following hyper-parameters: $time\_step = 15$, $batch\_size = 40$, $learning\_rate = 0.001$, $iterations = 3000$, $hidden\_units = 16$. Plot out the loss vs. iterations.

3. During testing, you will first feed in a contiguous subsequence T of size n from data to your model as the initial input. For each new output value $\hat{y}_{n+1}$, remove the first element from T and append $\hat{y}_{n+1}$ to T. Feed T to RNN again. Repeat at least n times. After this, pick the best reconstruction you can find in further iterations and paste it in your report.

4. Since at every single point of sine curve, the first derivative can be either positive or negative. Look at the sine curve from two directions might be helpful. Try to adjust your code and repeat 2&3 for bi-directional RNN ($tf.nn.static_bidirectional_rnn$). Hint: you may need more iterations for this model.