EE 511 Machine Learning, Winter 2018
Homework 3

Due: Wednesday, Feb 14, beginning of class

1 Kernels [50 Points]

In this section we will work with a regression model called a *normalized RBF network*. As usual, we have a dataset (x_i, y_i) with M distinct examples, where $x_i \in \mathbb{R}^N$ and $y_i \in \mathbb{R}$. We will be performing linear regression, and here we will use the RBF kernel to create our basis functions. Recall that the RBF kernel is

$$K(y, z) = \exp\left(\frac{-||y - z||_2^2}{2\sigma^2}\right)$$

where σ is known as the "bandwidth" and determines the "width" of the kernel. As $\sigma \to 0$, $K(y, z)$ tends to 0 when $y \neq z$ and 1 when $y = z$. As $\sigma \to \infty$, $K(y, z)$ tends to the same value for all y, z. We will use the RBF kernel and our data points to define M basis functions

$$\phi_i(x) = K(x, x_i)/Z(x)$$

$$Z(x) = \sum_{j=1}^{M} K(x, x_j)$$

In addition, we define a constant bias term $\phi_0(x) = 1$, and define a linear prediction function with weights $w_0, \ldots, w_M$

$$f(x) = \sum_{i=0}^{M} w_i \phi_i(x)$$

1. *(10 points)* Assume that our loss function is the sum of squared errors on the training data with an L2 penalty on the non-bias weights.

$$L(w) = \frac{1}{2}\left(\sum_{i=1}^{M}(y_i - f(x_i))^2 + \lambda \sum_{i=1}^{M} w_i^2\right)$$

We will also define a helpful matrix

$$Q = \begin{bmatrix} \phi_0(x_1) & \phi_1(x_1) & \ldots & \phi_M(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_M) & \phi_1(x_M) & \ldots & \phi_M(x_M) \end{bmatrix}$$

Using this matrix, give a closed form solution for the optimal weights w^*.

2. *(20 points)* Choosing the bandwidth is very important, as setting it too high or too low can lead to suboptimal results. Give the optimal weights w^* that result as $\sigma \to \infty$. You do not need to give a rigorous proof of optimality, but you should justify your answer reasonably.

   (Hint 1) Consider what happens to $\phi_i(x_j)$ as $\sigma \to \infty$.
   (Hint 2) Let's define $\bar{y} = \sum_i y_i / M$.

3. *(20 points)* Now let's assume that we have no bias term and no $L2$ penalty, so

$$f(x) = \sum_{i=1}^{M} w_i \phi_i(x)$$

$$L(w) = \frac{1}{2} \left(\sum_{i=1}^{M} (y_i - f(x_i))^2 \right)$$

   Give the optimal weights w^* as $\sigma \to 0$. Again, you do not need a rigorous proof, but you should justify your answer.

4. *(EXTRA CREDIT +5 points)* Assume that we add back in the bias term and $L2$ penalty, and choose λ to be a very small (but non-zero) value. What are the optimal weights w^* as $\sigma \to 0$? Justify your answer.

2 Programming Question [150 Points]

2.1 Feedfoward Neural Network in Tensorflow

For this part of programming, we're going to implement feedfoward neural network using Tensorflow (`https://www.tensorflow.org/`). Tensorflow is a popular open-source computational library for machine learning and deep learning applications. With its flexible architecture, Tensorflow allows computation deployment to various platform with heterogeneous computational devices (single or multi CPUs or GPUs on desktop, server, or mobile device).

In this problem, we're going to do the digit recognition task, where we are given an image of a handwritten digit and want to predict what number it represents, from 0 to 9. The digit images are from a very famous dataset MNIST (`http://yann.lecun.com/exdb/mnist/`). Before you try to implement the model in Tensorflow, please take a look at

`data.py`

to see how the data is read in and processed.

First, we're going to implement a 2-hidden-layer feedfoward neural network distinguish images whose labels are 3 or 5, a binary classification problem.

2.1.1 2-Layer Feedforward

First, let's define the 2-layer feedfoward neural network mathematically. Here, we assume all vectors are row vectors. Given a single data point, $\mathbf{x} \in \mathbb{R}^d$, and its label $\mathbf{y} \in \mathbb{R}^C$, where d is the number of pixels and C is the number of digit classes, the output of the 1st layer is

$$\mathbf{z}_1 = f(\mathbf{x}W_1 + \mathbf{b}_1), \tag{1}$$

and the output of the 2nd layer is

$$\mathbf{z}_2 = f(\mathbf{z}_1 W_2 + \mathbf{b}_2), \tag{2}$$

where $\mathbf{z}_1, \mathbf{b}_1 \in \mathbb{R}^{h_1}$, $\mathbf{z}_2, \mathbf{b}_2 \in \mathbb{R}^{h_2}$, $W_1 \in \mathbb{R}^{d \times h_1}$, $W_2 \in \mathbb{R}^{h_1 \times h_2}$ and $f(\cdot)$ is the non-linear transformation. Finally, we perform a linear projection into logit space

$$\mathbf{z} = \mathbf{z}_2 W + \mathbf{b}, \tag{3}$$

where $\mathbf{z} \in \mathbb{R}^C$, $W \in \mathbb{R}^{h_2 \times C}$ and $\mathbf{b} \in \mathbb{R}^C$.

In the same way as logistic regression, we can then use the logit to compute the cross-entropy between the estimated probability and the global labels as we did in the last homework.

2.1.2 Placeholder for inputs

The input interface to Tensorflow model is usually defined as placeholder. The basic function is

```
def placeholder_inputs_feedforward()
```

in `model.py`. The purpose of placeholder is to define the data format that the model is supposes to accept including input types, input shape and data type. In our case, we need the model to read two variables, `image_placeholder, label_placeholder`.

Please first read the API information at `https://www.tensorflow.org/api_docs/python/tf/placeholder` and modify the function accordingly.

2.1.3 Feed Dictionary

After you define the placeholders to the model, the next step is to define how the data you read in from file maps to the corresponding placeholder. This step is carried out in the function

```
def fill_feed_dict()
```

In our case, given a batch of images and labels, map them to the corresponding placeholders you've defined in the previous section.

2.1.4 Build 2-layer Feedforward

For this part, you need to modify the function

```
def feed_forward_net()
```

There are three `with` statements defined `variable_scope` for implementation of Equation (1), (2) and (3). The required coding for the three blocks is almost identical. Therefore, we will only go through for the first coding block for (1). In order to transform the input image into the first hidden layer, we have to first define two variables, the weight matrix W_1 and the bias $\mathbf{b}_1$. In Tensorflow, those variables can be defined using `tf.get_variable()` or `tf.Variable()`. The two methods both require information about variable name, shape and data type.

For the non-linear activation function $f(\cdot)$, we will use the `ReLU` function. Other traditional popular ones are `sigmoid,tanh` and some variants to `ReLU` are `LeakyReLU, PReLU`. Traditionally, more non-linearity activation is preferred for shallow (small number of hidden layers). But now, because deep learning needs to be really **deep**, more linearity activation is shown to be better. If you're now sure how to use those activation functions in Tensorflow, please refer to `https://www.tensorflow.org/versions/r0.12/api_docs/python/nn/activation_functions_`.

2.1.5 Cross-entropy Loss

Here, we will use the cross-entropy loss as the training objective. Actually, the training objective is exactly the same as the objective of logistic regression in the previous assignment. Please update the function

`def compute_loss()`

2.1.6 Put Together

Now, you already know

1. how to feed the data to the model,
2. how to define the forward pass of the model,
3. how to compute the loss.

Altogether, the above steps have defined a computation graph in Tensorflow. In order to drive the training process, we then need to define how to iterate over a sequence batches of data and then minimize the objective along the way. Please see the function

`def train()`

in `train_script.py`. Here, you can modify the number of training iterations `max_iters` in `Config class` to specify how many epochs you want to have over the training data.

2.1.7 Evaluation

Different from the last homework, here we have separate validation set for evaluating your model performance. Since it is a classification task, we're going to use the accuracy as evaluation metric. However, since the model reads in data in batches, we have to compute the number of correct predictions `def eval_prediction()` and then calculate the accuracy by dividing the total number of correct predictions with the data size `def evaluation()`. Here, please update the function

`def evaluation()`

in `model.py` to compute the accuracy properly.

2.1.8 3 VS 5 Binary Classification

Here, we need to tune the model to get the better performance. Let's do some tunning

1. Vary the `max_iters` in 40, 60, 80, and report the trend of the training accuracy and validation accuracy in table.

2. Fix the `max_iters=60`, vary the `hidden1_size=hidden2_size` in 32, 64, 128, and report the change of both training and validation accuracy in table.

Please summarize your findings. Now, you're ready to submit your code for binary classification. Please first make a copy your code into a folder named `binary_code`. Copy the best perform model directory to `binary_best_model`.

2.2 Multi-class Classification

As you might notice that the MNIST dataset actually contains 10 classes of digits, we have only focus on differentiating 3 vs 5 so far. In this part, we want to perform the 10-class digit classification task.

It is quite easy to extend the model we've implemented to support multi-class classification. For that purpose, we only have to modify two essential parts:

1. Modify the function `def data_reader()` in `data.py` to use all images and labels.

2. Modify the attribute `num_class` of `Config class` in `model.py`.

Let's do some tunning

1. Varying the `max_iters` in {300, 400, 500}, report the change of the training accuracy and validation accuracy in table.

2. Fix the `max_iters=400`, varying the `hidden1_size=hidden2_size` in {32, 64, 128}, report the change of both training and validation accuracy in table.

Please summarize your findings. Now, you're ready to submit your code for multi-class classification. Please make a copy your code into a folder named `multi_class_code`. Copy the best perform model directory to `mult_class_best_model`.

2.3 Bonus [+5 Points]

We haven't touched anything about regularization so far. Can you do a quick investigating and try some regularization techniques for neural network. Please describe what you've tried and your findings.

2.4 Submission

Please submit the four folders `binary_class_code`, `binary_best_model`, `multi_class_code`, and `multi_class_best_model` together with your report.