CSCE 5063-001: Assignment 3

Due 11:59pm Friday, March 22, 2019

1 Neural Networks

In this problem, you will implement a neural network to recognize handwritten digits. Your goal is to implement the forward propagation algorithm for prediction, and also to implement the backpropagation algorithm for learning the weights.

Important: Please carefully follow each step as it is easy to get bugs in the implementation.

1.1 Dataset

You are given a dataset that contains 5000 training examples of handwritten digits, where each training example is a 20 pixel by 20 pixel grayscale image of the digit. Figure 1 shows 100 randomly selected digits from the dataset. The 20 by 20 grid of pixels is "unrolled" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000×400 matrix X where every row is a training example for a handwritten digit image, contained in file `X.csv`.

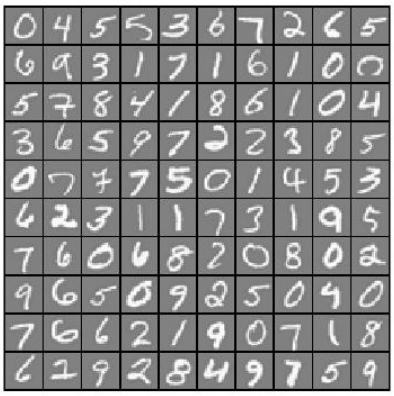

Figure 1: Handwritten digits.

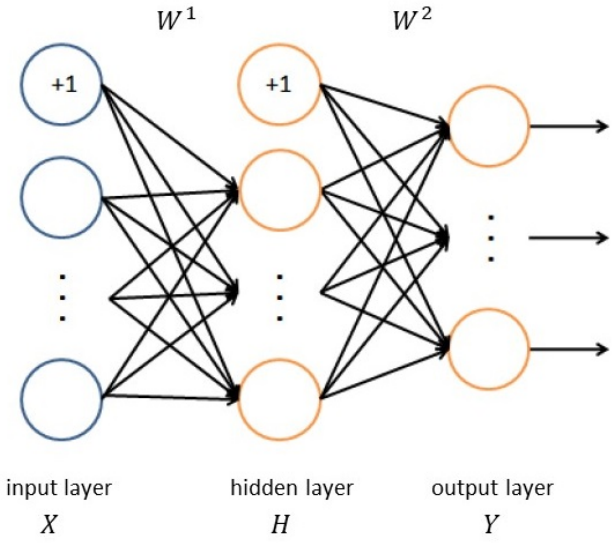

Figure 2: Neural network structure

The second part of the training set is a 5000-dimensional vector Y that contains labels for the training set. Note that a "0" digit is labeled as "10", while the digits "1" to "9" are labeled as "1" to "9" in their natural order. This vector is contained in file `Y.csv`.

Important: For debugging purpose, do not shuffle the data.

1.2 Multi-class classification

For multi-class classification, you need to first transform each label to a 10-dimensional vector, where the corresponding element is 1 and all other elements are 0. We use the first element in the vector to denote label "1", the second element in the vector to denote label "2", etc., and use the 10th element to denote label "0". For example, label "3" should be transformed to vector $[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$. After the transformation, Y becomes a 5000×10 matrix.

Important: For debugging purpose, do not change the order of representation in the vector.

1.3 Neural network structure

The structure of the neural network is shown in Figure 2. It has 1 hidden layer, in addition to the input layer and the output layer. Since the images are of size 20×20, this gives us 400 input nodes, plus one extra bias node which always outputs +1. There are 25 hidden nodes, plus one extra bias node which always outputs +1. There are 10 output nodes, corresponding

to the 10 digit classes. As a result, the weights of the hidden layer W^1 is a 25×401 matrix, and the weights of the output layer W^2 is a 10×26 matrix.

1.4 (Batch) gradient descent algorithm

You will use the (batch) gradient descent for learning the weights of the neural network. It will be good to first implement the framework of the gradient descent algorithm (or adapt from your framework in previous assignments), and then implement each component in the framework. The framework is shown in Algorithm 1.

Algorithm 1: Batch Gradient Descent

```
Initialize weights;
while convergence criteria not reached do
    Perform the forward propagation for all training examples to compute the
     outputs of each layer and the loss function;
    Perform the backpropagation for all training examples to compute the gradient
     of the loss function;
    Update the weights using the gradient;
```

Important: For the convergence criteria, we execute a fixed number of 500 iterations.

1.5 Initialize weights

For debugging purpose, I give you a fixed set of initialized weights, contained in files `initial_W1.csv` and `initial_W2.csv`. Load the files for initialization.

1.6 Forward propagation and prediction

Now you will implement forward propagation for the neural network and make the prediction. Denote the vector of input layer as x, the vector of hidden layer as h, the vector of output layer as y. Recall that the forward propagation asks you to do the following for each training example $x = X(i,:)^T$ (the transpose of the ith row in matrix X)

$$z^1 = W^1 x^{(i)}, \quad h = \sigma(z^1), \quad z^2 = W^2 h, \quad \hat{y} = \sigma(z^2),$$

where we use the logistic function for the activation function σ, i.e., $\sigma(z) = \frac{1}{1+e^{-z}}$.

Important: Remember to insert an extra 1 ahead of the first element of x before it is used to compute z^1; and also insert an extra 1 ahead of the first element of h before it is used to compute z^2.

Important: For above forward propagation, it would be much simpler if we use the matrix form, as shown below.

$$X = [1, X], \quad Z^1 = X \times (W^1)^T, \quad H = \sigma(Z^1), \quad H = [1, H], \quad Z^2 = H \times (W^2)^T, \quad \hat{Y} = \sigma(Z^2),$$

where Z^1, H, Z^2 are the matrices of z^1, h, z^2 for all 5000 training examples.

For prediction, simply output the index of the maximum element in $\hat{y}$. Remember that you will output "10" if the predicted digit is "0".

For debugging purposes, I give you a set of well-trained weights contained in files `W1.csv` and `W2.csv`, where `W1.csv` contains a 25×401 matrix for weights W^1, and `W2.csv` contains a 10×26 matrix for weights W^2. Load these weights to your program and make the prediction using your implemented forward propagation. Compare the prediction with real labels. You should see that the accuracy, i.e., the fraction of correct prediction, is 97.52%.

1.7 Loss function

The next step is to implement the loss function, which is given below.

$$J = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{10} \left[-y_k^{(i)} \log \hat{y}_k^{(i)} - (1 - y_k^{(i)}) \log(1 - \hat{y}_k^{(i)}) \right] + \frac{\lambda}{2m} \left[\sum_{j=1}^{25} \sum_{k=2}^{401} (w_{j,k}^1)^2 + \sum_{j=1}^{10} \sum_{k=2}^{26} (w_{j,k}^2)^2 \right],$$

where $m = 5000$, $\lambda = 3$, $y_k^{(i)}$ is the kth element of the label vector of the ith training example (i.e., $Y(i, k)$), $\hat{y}_k^{(i)}$ is the kth element of the output vector for the ith training example (i.e., $\hat{Y}(i, k)$), $w_{j,k}^1$ is one element in W^1, and $w_{j,k}^2$ is one element in W^2.

Important: In the regularization term, we don't consider weights $w_{j,1}^1$ and $w_{j,1}^2$ for all j, as they are associated with the bias nodes.

For debugging purposes, perform the forward propagation and compute the loss function using the weights given by files `W1.csv` and `W2.csv`. If you implement the loss function correctly, you will obtain a value of 0.576051, where the first term of the loss function gives you 0.287629, and the second term gives you 0.288422.

1.8 Logistic gradient

We will need to compute the gradient of logistic function in backpropagation. The gradient for the logistic function can be computed as

$$\sigma'(z) = \sigma(z)(1 - \sigma(z)).$$

For correctness checking, $\sigma'(0) = 0.25$.

1.9 Backpropagation

Now you will implement the backpropagation to compute the gradient of the loss function. In our case, the gradient is computed as

$$\frac{\partial J}{\partial w_{j,i}^l} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial \mathcal{L}}{\partial w_{j,i}^l} + \frac{\lambda}{m}\begin{cases} w_{j,i}^l & \text{if } i \neq 1 \\ 0 & \text{if } i = 1 \end{cases}$$

where

$$\frac{\partial \mathcal{L}}{\partial w_{j,i}^l} = \beta_j^l \cdot h_i^{l-1}$$

$$\beta_k^2 = \hat{y}_k - y_k,$$

and

$$\beta_j^1 = \left(\sum_{k=1}^{10}\beta_k^2 \cdot w_{k,j+1}^2\right) \cdot \sigma'(z_j^1)$$

Important: It would be much simpler if we use the matrix form, as shown below.

$$\nabla_{W^2} J = \frac{1}{m}\sum_{i=1}^{m}\nabla_{W^2} J^{(i)} + \frac{\lambda}{m}\left[0, W^2(:, 2:\text{end})\right]$$

$$\nabla_{W^1} J = \frac{1}{m}\sum_{i=1}^{m}\nabla_{W^1} J^{(i)} + \frac{\lambda}{m}\left[0, W^1(:, 2:\text{end})\right]$$

where $[0, W^2(:, 2:\text{end})]$ simply means that replacing the first column of W^2 with all zeros, and similarly for $[0, W^1(:, 2:\text{end})]$.

To compute $\nabla_{W^2} J^{(i)}$ and $\nabla_{W^1} J^{(i)}$ for each training example i, we follow 4 steps:

1. $\beta^2 = \hat{Y}(i, :) - Y(i, :)$
2. $\beta^1 = (\beta^2 \times W^2(:, 2:\text{end})) \cdot \sigma'(Z^1(i, :))$
3. $\nabla_{W^2} J^{(i)} = (\beta^2)^T \times H(i, :)$
4. $\nabla_{W^1} J^{(i)} = (\beta^1)^T \times X(i, :)$

where $W^2(:, 2:\text{end})$ means matrix W^2 after removing the first column.

For debugging purposes, I provide you the gradient of W^2 and W^1 for the first 3 iterations, contained in files `W2_grad.xlsx` and `W1_grad.xlsx`. Check whether you implement backpropagation correctly.

1.10 Update weights

For each iteration, after the backpropagation, update the weights as $W^2 = W^2 - \eta \nabla_{W^2} J$ and $W^1 = W^1 - \eta \nabla_{W^1} J$, where learning rate $\eta = 0.2$.

1.11 Prediction

Finally, after learning 500 iterations, make the prediction and examine the accuracy using the learned weights. Your accuracy should be 85.82%.

2 What to submit

1. The source code.
2. Plot of loss function J vs. the number of iterations.