Machine Learning HW2 Report

b02901122 電機四 劉致廷

1. Logistic regression function

   The whole structure is that the weighted sum of every element in the input data will go through a sigmoid function to predict the "1" case or "0" case. If the output is higher than 0.5, we can say that it is the "1"case, vice versa. The three parts below are the function set, Loss function, the Gradient and the Optimizer I used.

   1) Function Set & Cross Entropy (code, where np = numpy)

      Y is the output after sigmoid function, and Y_ is the label answer.

      ```
      Y = 1.0/(1.0+np.exp(-np.dot(X,W)))
      Loss = np.mean(-(Y_*np.log(Y+e)+(1-Y_)*np.log(1-Y+e)))     (e is 1e-20)
      ```

   2) Gradient

      ```
      Grad = -np.dot(X.T,(Y_-Y))/len(X)
      ```

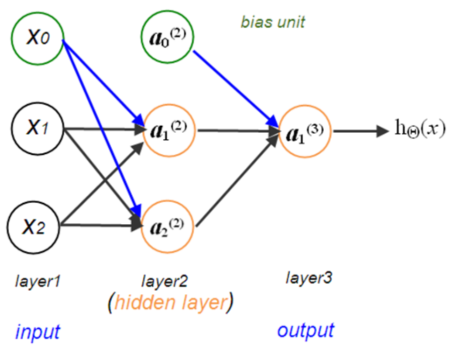
   3) Optimizer: AdamOptimizer (Wil explain next section)

   Best Kaggle Accuracy: 0.92667

2. Describe your another method, and which one is best.

   I use Neural Network for my another method, the model is like the figure below.

   The only one hidden layer neurons are 40, and I combine the bias unit in it, so there are 41 hidden neurons, the input layer neurons are 58, which are the 57 input features and one bias unit, and the output layer is a neuron used to predict two classes, all the activation function I used are sigmoid.



   The optimizer I used was AdamOptimizer, because the downtrend of the loss when I used Gradient Descent is sluggish. Therefore, after I use Adam, the loss would drop steadily. The parameters and the algorithm are as below.

   learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08, g = Gradient

   ```
   m_0 <- 0 (Initialize initial 1st moment vector)
   v_0 <- 0 (Initialize initial 2nd moment vector)
   t <- 0 (Initialize timestep)
   ```

```
t <- t + 1
lr_t <- learning_rate * sqrt(1 - beta2^t) / (1 - beta1^t)

m_t <- beta1 * m_{t-1} + (1 - beta1) * g
v_t <- beta2 * v_{t-1} + (1 - beta2) * g * g
variable <- variable - lr_t * m_t / (sqrt(v_t) + epsilon)
```

Best Kaggle Accuracy: 0.9600, which is much better than logistic regression.

Reference:

1.https://goo.gl/bZpDvK

2. https://goo.gl/fDvXdL

3. How did I choose the nn model ?

First, I try to use the regularization to avoid overfitting, so it just need to add a λw to the gradient. Therefore, I use 3601 training data & 400 validation data to choose my model. After I choose the λ, I try to base on the λ and then choose the number of hidden neurons.

The stop condition when I use training data to train and use validation data to test is that when the output of the network would not change too much(less than 0.004), I would stop the training and start to use validation data to test.

The graph below is the model I chose.

λ = 0.01 (neuron=32)	Validation ACC = 0.9649
λ = 0.1 (neuron=32)	Validation ACC = 0.9549
λ = 10 (neuron=32)	Validation ACC = 0.9525
λ = 100 (neuron=32)	Validation ACC = 0.9605

$\lambda = 0.01$, 16 neurons	Validation ACC = 0.9525
$\lambda = 0.01$, 32 neurons	Validation ACC = 0.9649
$\lambda = 0.01$, 40 neurons	Validation ACC = 0.9675
$\lambda = 0.01$, 64 neurons	Validation ACC = 0.9649

Result1: fix $\lambda = 0.01$, start to tune the hidden neurons,

Result2: the final model: $\lambda = 0.01$, 40 neurons (hidden layer)