Implementation MLP Regressor to Predict Housing Price

4105053118 陳謙慶

1 Introduction

There are 506 housing data with 14 features including the house prices. Features include crime rate, average number of rooms per dwelling, percentage of low state of the population, and so on. We have to use the neural network to implement the multilayer regressor to minimize the error between the real house prices and the predicted ones.

2 Data Preprocessing

First, we need to choose the house price feature to be the label y, and another part is going to be the new data X.

Second, we seperate the data into two parts: training set and testing set. The Ratio of testing set and training set is 0.3.

Finally, we do feature scaling using standardization. Then, to standardize training and testing set individually to X and y and finish the data preprocessing.

3 Modeling

At the beginning, we need to initialize the weight and bias between input layer(K), hidden layer(J), and output layer(I). Suppose $a_k^{(in)} \in K$, $a_j^{(h)} \in J$, $a_i^{(out)} \in I$ for some k, j, i. And then, run the forward pass to fix each node in the hidden layer and output layer.

$$
\begin{aligned}
Z^{(h)} &= XW^{(h)} + b^{(h)} \\
A^{(h)} &= \sigma(Z^{(h)}) \\
Z^{(out)} &= A^{(h)}W^{(out)} + b^{(out)} \\
A^{(out)} &= \sigma(Z^{(out)})
\end{aligned}
$$

In the forward pass, we use ReLU as our activative function. The following is Relu and its deveratives:

$$ReLU = max(0, x)$$

$$\frac{\partial ReLU}{\partial x} = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

In the backward pass, we need to use chain rule to compute the formula associated with the gradient of loss function. The loss function is as follows $J(w)$ and we must to minimize it.

$$
\begin{aligned}
J(w) &= \sum_{i=1}^{n} \left\| \hat{y}^{(i)} - y^{(i)} \right\|^2 = \sum_{i=1}^{n} L^{(i)}(w) \\
&= \sum_{i=1}^{n} (\hat{y}^{(i)} - y^{(i)})^T (\hat{y}^{(i)} - y^{(i)}) \\
&= \sum_{i=1}^{n} error^T error
\end{aligned}
$$

To be easier, we just discuss about how the one data work, show as follows:
For output layer:

$$
\begin{aligned}
\frac{\partial L(w)}{\partial w_{i,j}} &= \frac{\partial}{\partial w_{i,j}} (error)^T (error) \\
&= \frac{\partial}{\partial w_{i,j}} [(a_1^{(out)} - y_1)^2 + .. + (a_i^{(out)} - y_i)^2 + ..] \\
&= \frac{\partial}{\partial w_{i,j}} (a_i^{(out)} - y_i)^2 \\
&= 2(a_i^{(out)} - y_i) \frac{\partial}{\partial w_{i,j}} (a_i^{(out)}) \\
&= 2(a_i^{(out)} - y_i) \frac{\partial \sigma(z_i^{(out)})}{\partial z_i^{(out)}} a_j^{(h)} \\
&= a_j^{(h)} \delta_i^{(out)}
\end{aligned}
$$

For hidden layer:

$$\begin{aligned}
\frac{\partial L(w)}{\partial w_{j,k}} &= \frac{\partial}{\partial w_{j,k}}(error)^T(error) \\
&= \frac{\partial}{\partial w_{j,k}}[(a_1^{(out)} - y_1)^2 + .. + (a_i^{(out)} - y_i)^2 + ..] \\
&= \frac{\partial}{\partial w_{j,k}} \sum_{i \in I} (a_i^{(out)} - y_i)^2 \\
&= \sum_{i \in I} 2(a_i^{(out)} - y_i) \frac{\partial}{\partial w_{j,k}}(a_i^{(out)}) \\
&= a_k^{(in)} \frac{\partial \sigma(z_j^{(h)})}{\partial z_j^{(h)}} \sum_{i \in I} 2(a_i^{(out)} - y_i) w_{i,j} \frac{\partial \sigma(z_i^{(out)})}{\partial z_i^{(out)}} \\
&= a_k^{(in)} \frac{\partial \sigma(z_j^{(h)})}{\partial z_j^{(h)}} \sum_{i \in I} w_{i,j} \delta_i^{(out)} \\
&= a_k^{(in)} \delta_j^{(h)}
\end{aligned}$$

Using gradient decent, we can update the weight and bias through each iteration.

$$w_{i,j} = w_{i,j} - \eta \frac{\partial L(w)}{\partial w_{i,j}}$$
$$w_{j,k} = w_{j,k} - \eta \frac{\partial L(w)}{\partial w_{j,k}}$$

Meanwhile, compute the cost again and again.

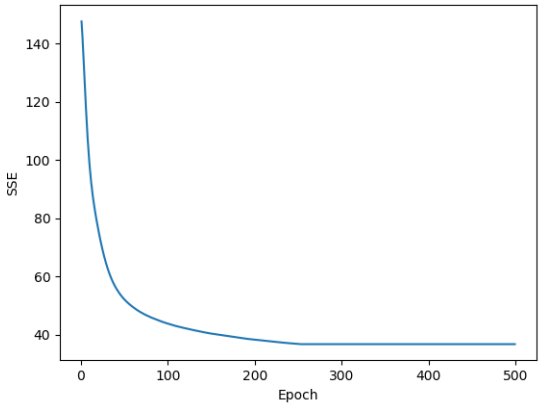
4 Vectorization

Vectorization can be compute every data simultaneously. It is also easy to construct the algorithm to code.

$$\begin{aligned}
\delta^{(out)} &= [\delta_1^{(out)} \delta_2^{(out)} ... \delta_i^{(out)} ...] \\
&= 2 \cdot error \odot \frac{\partial \sigma(z^{(out)})}{\partial z^{(out)}} \\
\delta^{(h)} &= [\delta_1^{(h)} \delta_2^{(h)} ... \delta_j^{(h)} ...] \\
&= \delta^{(out)} W^{T(out)} \odot \frac{\partial \sigma(z^{(h)})}{\partial z^{(h)}} \\
\nabla L^{(out)}(W) &= A^{T(h)} \delta^{(out)} \\
\nabla L^{(h)}(W) &= A^{T(in)} \delta^{(h)} \\
W^{(out)} &= W^{(out)} - \eta \nabla L^{(out)}(W) \\
W^{(h)} &= W^{(h)} - \eta \nabla L^{(h)}(W)
\end{aligned}$$

5 Experiment result

We set 200 units in the hidden layers and 1 unit, which represent the predicted house price, in the output layer. We choose learning rate η to be 0.001. Because we do regression, we do not have accuracy. Every predicted value is continuous, not discrete. Therefore, we compute their sum of square error (SSE) to see how it changes. We do the iteration 500 times. The following picture is the result:



SSE is convergent successfully.