Report on Customer Return Prediction Model

Author: Hakob Hakobyan

Introduction

The goal of this project is to create a neural network model to predict whether customers will return to our website or app based on their actions.

Data

The data used to train the model includes 10 variables that indicate actions of the customers on an online shopping platform and more than 300,000 rows, from which a sample of 17000 observations is selected.

Model Architecture

The best model conceived is a Multi-Layer Perceptron (MLP) with 3 layers and 96 neurons. The architecture of the model is as follows:

In [ ]:
```
class MLP(Module):
    def __init__(self, n_inputs):
        super(MLP, self).__init__()

        self.hidden1 = Linear(n_inputs, 64)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = ReLU()

        self.hidden2 = Linear(64, 20)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = F.leaky_relu

        self.hidden3 = Linear(20, 2)
        xavier_uniform_(self.hidden3.weight)
        self.act3 = Sigmoid()

    def forward(self, X):
        X = self.hidden1(X)
        X = self.act1(X)
        X = self.hidden2(X)
        X = self.act2(X)
        X = self.hidden3(X)
        X = self.act3(X)
        return X

model = MLP(10)
```

In [6]:
```
print(model)
```

```
MLP(
  (hidden1): Linear(in_features=10, out_features=64, bias=True)
  (act1): ReLU()
  (hidden2): Linear(in_features=64, out_features=20, bias=True)
  (hidden3): Linear(in_features=20, out_features=2, bias=True)
```

```
  (act3): Sigmoid()
)
```

Training Process

The model is done using the following training process:

In [7]:
```
def train_model(train_dl, model):
    criterion = CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=0.001)

    for epoch in tqdm_notebook(range(10)):
        for i, (inputs, targets) in enumerate(train_dl):
            optimizer.zero_grad()
            yhat = model(inputs)
            loss = criterion(yhat, targets)
            loss.backward()
            optimizer.step()
```

Discussion

There is a trade-off between increasing the accuracy of the model adding more data and the overall training time. As the results show if we fix the architecture of the network, and add more data, the model accuracy will become from 0.85 to 0.853, but the training tim becomes 1500 sec instead of 96 seconds.

Experiments are done with different architectures and hyper-parameters from simplest to some complicated networks. I tried using 26 neurons and 3 hidden layers, then 46 neurons and 4 hidden layers with dropouts, using Adam instead of SGD etc. I also tried changing the learning rate from 0.01 to 0.001, which shows better results.

In the end, our final model has 96 neurons and 3 hidden layers. It is trained for 10 epochs by SGD optimizer and 0.001 learning rate and achieved an accuracy of **85%**.