```
import torch.nn as nn
import torch.nn.functional as F
import torch
from torch.utils.data import IterableDataset, DataLoader
import pandas as pd
import numpy as np
from tqdm import tqdm
import matplotlib.pyplot as plt
import gc
from sklearn.metrics import roc_auc_score, accuracy_score, recall_score, precision_score
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split
import math
from torch.optim.lr_scheduler import ReduceLROnPlateau,StepLR
from google.colab import drive
```

```
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tr
```

```
df = pd.read_csv(r'drive/My Drive/filtered_train.csv')
```

```
df['hr'] = df['hour'].astype(str).str.slice(6, 8).astype(int)
df = df.iloc[: , 1:]
df = df.drop('id', axis = 1)
```

```
df['hr-app_category'] = df['hr'].astype(str) + df['app_category']
df['hr-site_category'] = df['hr'].astype(str) + df['site_category']
df['hr-device_type'] = df['hr'].astype(str) + df['device_type'].astype(str)
df['banner_pos-device_type'] = df['banner_pos'].astype(str) + df['device_type'].astype(str)
df['device_type-app_category'] = df['device_type'].astype(str) + df['app_category']
df['device_type-site_category'] = df['device_type'].astype(str) + df['site_category']
```

```
df.columns
```

```
Index(['click', 'hour', 'C1', 'banner_pos', 'site_id', 'site_domain',
       'site_category', 'app_id', 'app_domain', 'app_category', 'device_id',
       'device_ip', 'device_model', 'device_type', 'device_conn_type', 'C14',
       'C15', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21', 'hr',
       'hr-app_category', 'hr-site_category', 'hr-device_type',
       'banner_pos-device_type', 'device_type-app_category',
       'device_type-site_category'],
      dtype='object')
```

```
df.tail()
```

	click	hour	C1	banner_pos	site_id	site_domain	site_category	a
404285	1	14103023	1005	0	85f751fd	c4e18dd6	50e219e0	9c
404286	0	14103020	1005	0	85f751fd	c4e18dd6	50e219e0	9c
404287	0	14103021	1002	0	c545a354	c4e18dd6	50e219e0	ec
404288	0	14103021	1005	0	85f751fd	c4e18dd6	50e219e0	e2
404289	1	14103022	1005	0	5b08c53b	7687a86e	3e814130	ec

5 rows × 30 columns

```
df.nunique()
```

```
click                           2
hour                          240
C1                              7
banner_pos                      7
site_id                      2225
site_domain                  2188
site_category                  22
app_id                       2241
```

```
app_domain                    143
app_category                   27
device_id                   64742
device_ip                  261706
device_model                 4380
device_type                     4
device_conn_type                4
C14                          2088
C15                             8
C16                             9
C17                           411
C18                             4
C19                            65
C20                           161
C21                            60
hr                             24
hr-app_category               408
hr-site_category              377
hr-device_type                 96
banner_pos-device_type         12
device_type-app_category       48
device_type-site_category      26
dtype: int64
```

```
df = df.drop(columns=(['device_ip','device_id']))
```

```
device = 'cpu'
use_cuda = True
if use_cuda and torch.cuda.is_available():
    print('cuda ready...')
    device = 'cuda'
print(device)
```

```
cpu
```

```
obj_features = list(df.select_dtypes(['object']).columns)
int_features = list(df.select_dtypes(['int64']).columns)
int_features.remove('click')

df[obj_features] = df[obj_features].fillna('-1', )
df[int_features] = df[int_features].fillna(0, )

for feat in obj_features:
    lbe = LabelEncoder()
    df[feat] = lbe.fit_transform(df[feat])
```

```
df.tail()
```

	click	hour	C1	banner_pos	site_id	site_domain	site_category	app
404285	1	14103023	1005	0	1150	1694	5	1
404286	0	14103020	1005	0	1150	1694	5	1
404287	0	14103021	1002	0	1701	1694	5	2
404288	0	14103021	1005	0	1150	1694	5	1
404289	1	14103022	1005	0	811	1034	3	2

5 rows × 28 columns

```
df.nunique()
```

```
click                        2
hour                       240
C1                           7
banner_pos                   7
site_id                   2225
site_domain               2188
site_category               22
app_id                    2241
app_domain                 143
app_category                27
device_model              4380
device_type                  4
```

```
device_conn_type               4
C14                         2088
C15                            8
C16                            9
C17                          411
C18                            4
C19                           65
C20                          161
C21                           60
hr                            24
hr-app_category              408
hr-site_category             377
hr-device_type                96
banner_pos-device_type        12
device_type-app_category      48
device_type-site_category     26
dtype: int64
```

```
CTR = df

sparse_features = CTR.loc[:, CTR.nunique() <= 100].columns.tolist()
sparse_features.remove('click')

for feat in sparse_features:
  CTR_temp = pd.get_dummies(CTR[feat], prefix=[feat])
  CTR = CTR.drop(feat, axis=1)
  CTR = pd.concat([CTR, CTR_temp], axis=1)


train, test = train_test_split(CTR, test_size=0.2, random_state=2022)


EMBEDDING_INPUTS = [
           'device_modelSEP4380SEP256',
           'app_idSEP2241SEP256',
           'site_idSEP2225SEP256',
           'site_domainSEP2188SEP256',
           'app_domainSEP143SEP128',
          ]
EMBEDDING_INPUTS1 = [
           'device_model',
           'app_id',
           'site_id',
           'site_domain',
           'app_domain',
          ]
WIDE_DIM = 423
```

```
class Model(nn.Module):
    def __init__(self, wide_dim, embedding_inputs, hidden_layers, dropout_p=0.7):
        super().__init__()
        self.wide_dim = wide_dim
        self.embedding_inputs = embedding_inputs
        self.deep_feature_dim = 0
        self.hidden_layers = hidden_layers

        for embedding_input in self.embedding_inputs:
            col_name, vocab_size, embed_dim = embedding_input.split('SEP')
            print(col_name, vocab_size, embed_dim)
            setattr(self, col_name+'_emb_layer', nn.Embedding(int(vocab_size), int(embed_dim)))
            self.deep_feature_dim += int(embed_dim)

        self.linear_layer_1 = nn.Linear(self.deep_feature_dim, self.hidden_layers[0])
        self.bn_1 = nn.BatchNorm1d(self.hidden_layers[0])
        for i, hidden_layer in enumerate(self.hidden_layers[1:]):
            setattr(self, f'linear_layer_{i+2}', nn.Linear(self.hidden_layers[i], hidden_layer))

        self.dropout = nn.Dropout(p=0.7)
        self.fc = nn.Linear(self.wide_dim+self.hidden_layers[-1], 1)


    def forward(self, X_w, X_d):
        embeddings = [getattr(self, col_name+'_emb_layer')(X_d[:, i].long())
                      for i, embedding_input in enumerate(self.embedding_inputs)
                      for col_name in embedding_input.split('SEP')
                      if not col_name.isdigit()
                     ]

        deep_out = torch.cat(embeddings, dim=-1)

        for i, _ in enumerate(self.hidden_layers):
            deep_out = F.relu(getattr(self, f'linear_layer_{i+1}')(deep_out))

        X_w = self.dropout(X_w)
        fc_input = torch.cat([X_w, deep_out], dim=-1)
        out = self.fc(fc_input)

        return out


model = Model(wide_dim=WIDE_DIM, embedding_inputs=EMBEDDING_INPUTS, hidden_layers=[512, 256, 128], dropout_p=0.7)
model.to(device)

optimizer = torch.optim.Adam(model.parameters(),lr=1e-3)
loss_fn = nn.BCEWithLogitsLoss()
scaler = torch.cuda.amp.GradScaler()

best_val_loss = float('inf')
train_losses = []
test_losses = []

X_w_train = torch.tensor(train.iloc[:, 12:].values.astype(np.float32))
X_w_train = X_w_train.squeeze().to(device)
X_d_train = torch.tensor(train.iloc[:][EMBEDDING_INPUTS1].values.astype(np.float32))
X_d_train = X_d_train.squeeze().to(device)
label_train = torch.tensor(train['click'].values.astype(np.float32)).to(device)
label_train = label_train.squeeze().unsqueeze(1).to(device)

X_w_test = torch.tensor(test.iloc[:, 12:].values.astype(np.float32))
X_w_test = X_w_test.squeeze().to(device)
X_d_test = torch.tensor(test.iloc[:][EMBEDDING_INPUTS1].values.astype(np.float32))
X_d_test = X_d_test.squeeze().to(device)
label_test = torch.tensor(test['click'].values.astype(np.float32)).to(device)
label_test = label_test.squeeze().unsqueeze(1).to(device)

for epoch in range(20):
    y_pred_train = model(X_w_train,X_d_train)
    loss_train = loss_fn(y_pred_train, label_train)

    optimizer.zero_grad()
    scaler.scale(loss_train).backward()
    scaler.step(optimizer)
    scaler.update()

    print(f'=============== Epoch {epoch} ===============')
    print('epoch:', epoch, 'Training Loss:', loss_train.item())
```

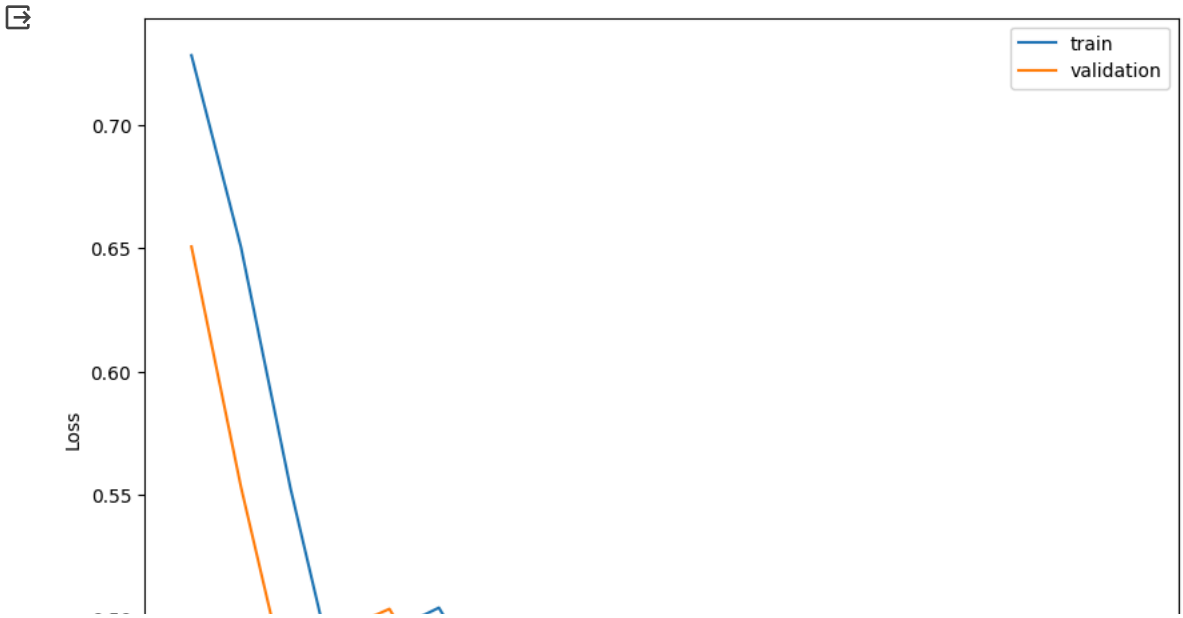
```
    y_pred_test = model(X_w_test,X_d_test)
    loss_test = loss_fn(y_pred_test, label_test)
    print('epoch:', epoch, 'testing Loss:', loss_test.item())

    train_losses.append(loss_train.item())
    test_losses.append(loss_test.item())

    if test_losses[-1] < best_val_loss:
        best_val_loss = test_losses[-1]
        print('Best model saved.\n')
        torch.save(model.state_dict(), './saved_model.pt')
```

```
device_model 4380 256
app_id 2241 256
site_id 2225 256
site_domain 2188 256
app_domain 143 128
/usr/local/lib/python3.10/dist-packages/torch/cuda/amp/grad_scaler.py:126: UserWarning: torch.cuda.amp.GradScaler is enal
  warnings.warn(
=============== Epoch 0 ===============
epoch: 0 Training Loss: 0.7282196283340454
epoch: 0 testing Loss: 0.6506996750831604
=============== Epoch 1 ===============
epoch: 1 Training Loss: 0.6505413055419922
epoch: 1 testing Loss: 0.5533163547515869
=============== Epoch 2 ===============
epoch: 2 Training Loss: 0.5528595447540283
epoch: 2 testing Loss: 0.46768632531166077
=============== Epoch 3 ===============
epoch: 3 Training Loss: 0.46757665276527405
epoch: 3 testing Loss: 0.4948439598083496
=============== Epoch 4 ===============
epoch: 4 Training Loss: 0.4945472478866577
epoch: 4 testing Loss: 0.5038530826568604
=============== Epoch 5 ===============
epoch: 5 Training Loss: 0.5043344497680664
epoch: 5 testing Loss: 0.47016605734825134
=============== Epoch 6 ===============
epoch: 6 Training Loss: 0.47076544165611267
epoch: 6 testing Loss: 0.4474058747291565
=============== Epoch 7 ===============
epoch: 7 Training Loss: 0.44816458225250244
epoch: 7 testing Loss: 0.44602450728416443
=============== Epoch 8 ===============
epoch: 8 Training Loss: 0.4466046988964081
epoch: 8 testing Loss: 0.45257213711738586
=============== Epoch 9 ===============
epoch: 9 Training Loss: 0.4529997706413269
epoch: 9 testing Loss: 0.4565974771976471
=============== Epoch 10 ===============
epoch: 10 Training Loss: 0.45665520429611206
epoch: 10 testing Loss: 0.4541183114051819
=============== Epoch 11 ===============
epoch: 11 Training Loss: 0.4536987245082855
epoch: 11 testing Loss: 0.4457624554634094
=============== Epoch 12 ===============
epoch: 12 Training Loss: 0.4455466568470001
epoch: 12 testing Loss: 0.43647173047065735
=============== Epoch 13 ===============
epoch: 13 Training Loss: 0.435831755399704
epoch: 13 testing Loss: 0.430876225233078
=============== Epoch 14 ===============
epoch: 14 Training Loss: 0.42974162101745605
epoch: 14 testing Loss: 0.43226373195648193
=============== Epoch 15 ===============
epoch: 15 Training Loss: 0.4309704303741455
epoch: 15 testing Loss: 0.43704840540885925
=============== Epoch 16 ===============
epoch: 16 Training Loss: 0.4353370666503906
```

```
plt.figure(figsize=(10, 8))
plt.plot(train_losses, label='train')
plt.plot(test_losses, label='validation')
plt.legend()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```


train
validation
0.70
0.65
0.60
0.55
Loss