Download Data

```
# For Google Colab. If not on Colab, make sure kaggle.json is it the right l
from google.colab import files

# upload kaggle.json
uploaded = files.upload()
```

```
Choose Files  no files selected    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
```

```
# move kaggle.json to the right location
!pip install -q kaggle
!ls
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/kaggle.json
```

```
kaggle.json  sample_data
```

```
# download our dataset using the Kaggle api
!kaggle competitions download home-credit-default-risk -p "home-credit-defau
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fi
Warning: Looks like you're using an outdated API Version, please consider upd
Downloading previous_application.csv.zip to home-credit-default-risk
 85% 65.0M/76.3M [00:00<00:00, 106MB/s]
100% 76.3M/76.3M [00:00<00:00, 119MB/s]
Downloading sample_submission.csv to home-credit-default-risk
  0% 0.00/524k [00:00<?, ?B/s]
100% 524k/524k [00:00<00:00, 73.5MB/s]
Downloading bureau.csv.zip to home-credit-default-risk
 95% 35.0M/36.8M [00:00<00:00, 68.1MB/s]
100% 36.8M/36.8M [00:00<00:00, 82.9MB/s]
Downloading POS_CASH_balance.csv.zip to home-credit-default-risk
 89% 97.0M/109M [00:01<00:00, 75.1MB/s]
100% 109M/109M [00:01<00:00, 89.1MB/s]
Downloading HomeCredit_columns_description.csv to home-credit-default-risk
  0% 0.00/36.5k [00:00<?, ?B/s]
100% 36.5k/36.5k [00:00<00:00, 55.2MB/s]
Downloading application_test.csv.zip to home-credit-default-risk
 86% 5.00M/5.81M [00:00<00:00, 41.0MB/s]
100% 5.81M/5.81M [00:00<00:00, 36.8MB/s]
Downloading bureau_balance.csv.zip to home-credit-default-risk
 86% 49.0M/56.8M [00:00<00:00, 61.8MB/s]
100% 56.8M/56.8M [00:00<00:00, 83.2MB/s]
Downloading application_train.csv.zip to home-credit-default-risk
 91% 33.0M/36.1M [00:00<00:00, 72.4MB/s]
100% 36.1M/36.1M [00:00<00:00, 81.0MB/s]
Downloading installments_payments.csv.zip to home-credit-default-risk
 96% 261M/271M [00:02<00:00, 120MB/s]
100% 271M/271M [00:02<00:00, 126MB/s]
Downloading credit_card_balance.csv.zip to home-credit-default-risk
 99% 96.0M/96.7M [00:00<00:00, 119MB/s]
100% 96.7M/96.7M [00:00<00:00, 132MB/s]
```

```
import os
import zipfile
import numpy as np
import pandas as pd

zip_ref = zipfile.ZipFile('home-credit-default-risk/application_train.csv.zip', 'r
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/application_test.csv.zip', 'r'
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/bureau_balance.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/bureau.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/credit_card_balance.csv.zip',
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/installments_payments.csv.zip'
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/POS_CASH_balance.csv.zip', 'r'
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('home-credit-default-risk/previous_application.csv.zip',
zip_ref.extractall('datasets')
zip_ref.close()
```

Load Datasets From Files

```
import numpy as np
import pandas as pd
import os
import zipfile
import warnings
warnings.filterwarnings('ignore')

def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(3))
```

```
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep track of the
DATA_DIR = "datasets" # folder where unzipped files are

ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_nam
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds
```

```
application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OW
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	

3 rows × 122 columns

```
application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALT
0	100001	Cash loans	F	N	Y
1	100005	Cash loans	M	N	Y
2	100013	Cash loans	M	Y	Y

3 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
```

```
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
```

```
# datasets that we have
datasets.keys()
```

```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance
```

Load In Datasets

Previous Applications

```
PA_df = datasets['previous_application'].groupby('SK_ID_CURR').agg({
    "AMT_APPLICATION": "mean",
    "CNT_PAYMENT": "max",
    "DAYS_TERMINATION":"mean",

    "NAME_PORTFOLIO": "max",
    "NAME_GOODS_CATEGORY": "max",
    "NAME_SELLER_INDUSTRY": "max",
})
```

```
PCB_df_copy = datasets['POS_CASH_balance'].groupby('SK_ID_PREV').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
POS_to_PA_df =  datasets['previous_application'].merge(PCB_df_copy, how='left', on=
PCB_df_temp = POS_to_PA_df.groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
PCB_df_temp=PCB_df_temp.rename({"CNT_INSTALMENT":"PREV_CNT_INSTALMENT","CNT_INSTALL
PA_df = pd.concat([PA_df, PCB_df_temp], axis=1)
```

```
IP_df_copy = datasets['installments_payments'].groupby('SK_ID_PREV').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min",
})
IP_df_copy["SUM_MISSED"] = IP_df_copy["AMT_INSTALMENT"] - IP_df_copy["AMT_PAYMENT"
IP_to_PA_df =  datasets['previous_application'].merge(IP_df_copy, how='left', on='
IP_df_temp = IP_to_PA_df.groupby('SK_ID_CURR').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min"
})
IP_df_temp = IP_df_temp.rename({"AMT_INSTALMENT":"PREV_AMT_INSTALMENT", "AMT_PAYMEN
PA_df = pd.concat([PA_df, IP_df_temp], axis=1)
```

```
CCB_df_copy = datasets['credit_card_balance'].groupby('SK_ID_PREV').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count",
})
CCB_to_PA_df =  datasets['previous_application'].merge(CCB_df_copy, how='left', on=
CCB_df_temp = CCB_to_PA_df.groupby('SK_ID_CURR').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count"
})
CCB_df_temp = CCB_df_temp.rename({"AMT_BALANCE":"PREV_AMT_BALANCE", "MONTHS_BALANC
PA_df = pd.concat([PA_df, CCB_df_temp], axis=1)
```

POS Cash Balances

```
PCB_df = datasets['POS_CASH_balance'].groupby('SK_ID_CURR').agg({
    "CNT_INSTALMENT": "count",
    "CNT_INSTALMENT_FUTURE": "mean",
    "MONTHS_BALANCE": "min",
})
```

Instalment Payments

```
IP_df = datasets['installments_payments'].groupby('SK_ID_CURR').agg({
    "AMT_INSTALMENT": "sum",
    "AMT_PAYMENT": "sum",
    "DAYS_INSTALMENT": "min",
    "DAYS_ENTRY_PAYMENT": "min",
})
IP_df["SUM_MISSED"] = IP_df["AMT_INSTALMENT"] - IP_df["AMT_PAYMENT"]
```

Bureau

```
B_df = datasets['bureau'].groupby('SK_ID_CURR').agg({
    "CREDIT_TYPE": "min",
    "CREDIT_ACTIVE": "max",
    "DAYS_CREDIT": "mean",
    "AMT_CREDIT_SUM": "max",
})
```

```
BB_df = datasets['bureau_balance'].groupby('SK_ID_BUREAU').agg({
    "MONTHS_BALANCE": "min",
    "STATUS": ["max", "min", "count"]
})
temp = pd.DataFrame({"MONTHS_BALANCE_MIN": BB_df["MONTHS_BALANCE"]["min"],  "STATU
BB_df = temp
BB_to_B_df =  datasets['bureau'].merge(BB_df, how='left', on='SK_ID_BUREAU')
BB_to_B_df = BB_to_B_df.dropna(subset = ["STATUS_MAX","STATUS_MIN"])
B_df_temp = BB_to_B_df.groupby('SK_ID_CURR').agg({
    "MONTHS_BALANCE_MIN": "min",
    "STATUS_MIN": "min",
    "STATUS_MAX" : 'max',
    "STATUS_COUNT": "count"
})
B_df = pd.concat([B_df, B_df_temp], axis=1)
```

```
CCB_df = datasets['credit_card_balance'].groupby('SK_ID_CURR').agg({
    "AMT_BALANCE": "mean",
    "MONTHS_BALANCE": "min",
    "AMT_CREDIT_LIMIT_ACTUAL": "count",
})
```

```
# initialize results table
results = pd.DataFrame(columns=["ExpID", "ROC AUC Score", "Cross fold train accura
```

Deep Learning Model

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
from time import time

app = datasets["application_train"]
train_x = app.loc[:, app.columns != "TARGET"]
train_y = app["TARGET"]
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
```

```
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import roc_auc_score

# preprocess data
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
  "OCCUPATION_TYPE"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUBLISH"
]

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='most_frequent')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])

scaler = preprocess_pipeline.fit(train_x, train_y)
```

```
train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
# globals
batch_size = 64
num_epochs = 100
num_in = train_x.shape[1]
num_layer_1 = 20
num_output = 2
```

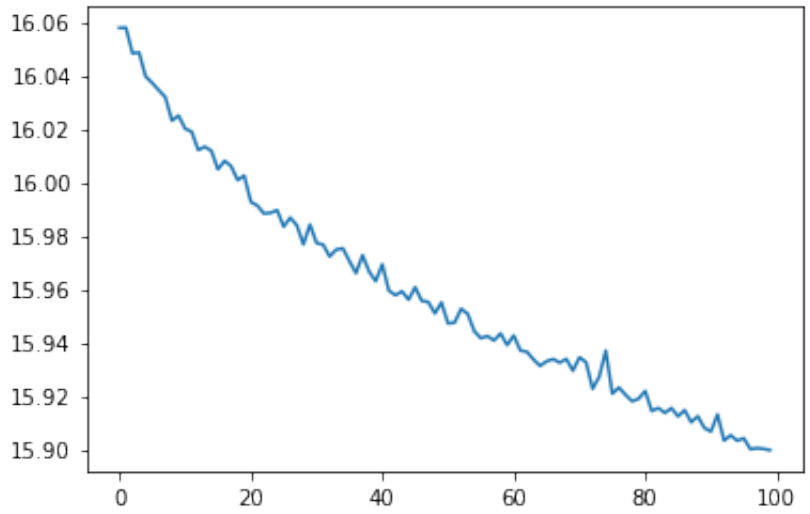
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.SGD(model.parameters(), lr=0.01)
loss_fn = nn.BCELoss()
```

```
from time import time

losses = []
test_losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

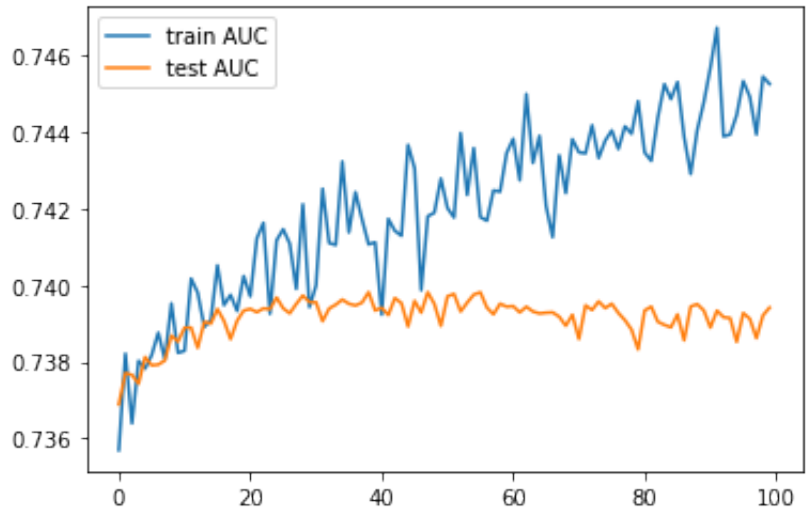
```
import matplotlib.pyplot as plt
plt.plot(range(epochs), losses, label="train loss")
```

```
[<matplotlib.lines.Line2D at 0x7fa9125f2050>]
```




```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa9132746d0>
```




```
from sklearn.metrics import roc_auc_score

start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start
```

```
results.loc[0] = ["Deep Learning", roc, "--", "--",
                  train_time, test_time, "Deep Learning w/ Application Data"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
							Deep Learning

Adding All Features

```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUBLISH",
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG",
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT",

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])



scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
batch_size = 64
num_epochs = 100
num_in = train_x.shape[1]
num_layer_1 = 20
num_output = 2
```

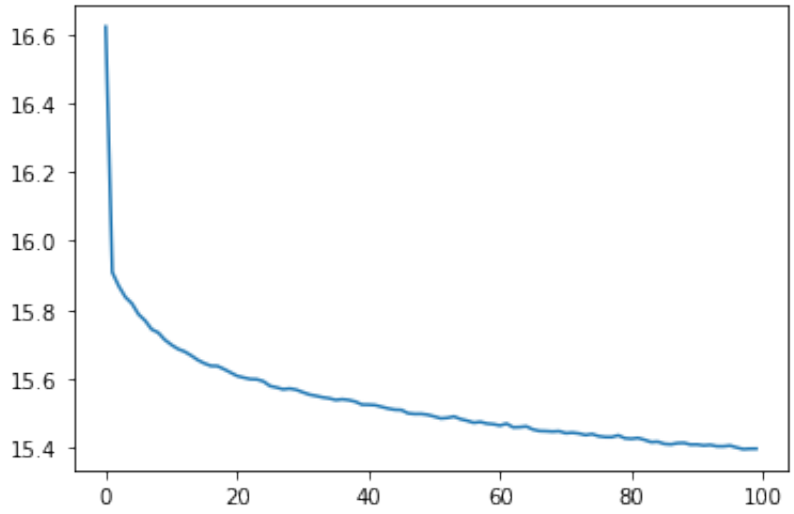
```
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.SGD(model.parameters(), lr=0.01)
loss_fn = nn.BCELoss()
```

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

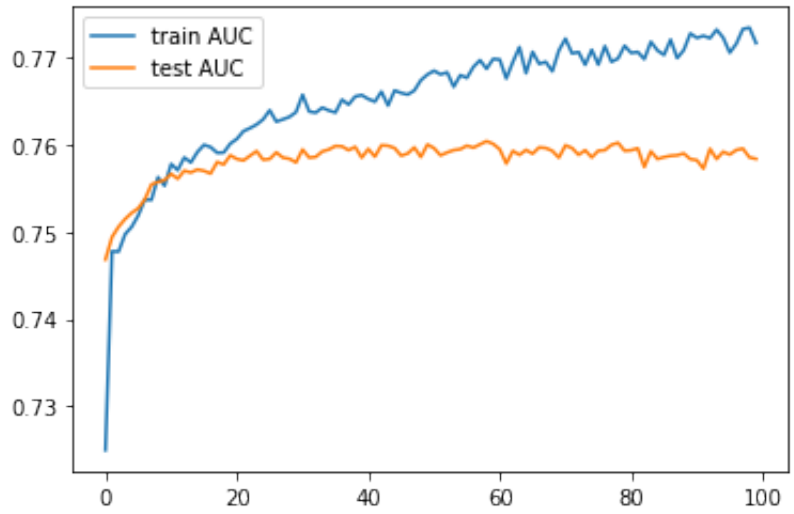
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa9125bf190>]
```




```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa9125816d0>
```




```
from sklearn.metrics import roc_auc_score

start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start
```

```
results.loc[1] = ["Deep Learning", roc, "--", "--",
                  train_time, test_time, "Deep Learning w/ all other data"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.739418	--	--	1075.331878	0.037791	Deep Learning w/ Application Data

Adam Optimizer

```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUBLISH",
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG",
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT",

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])



scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

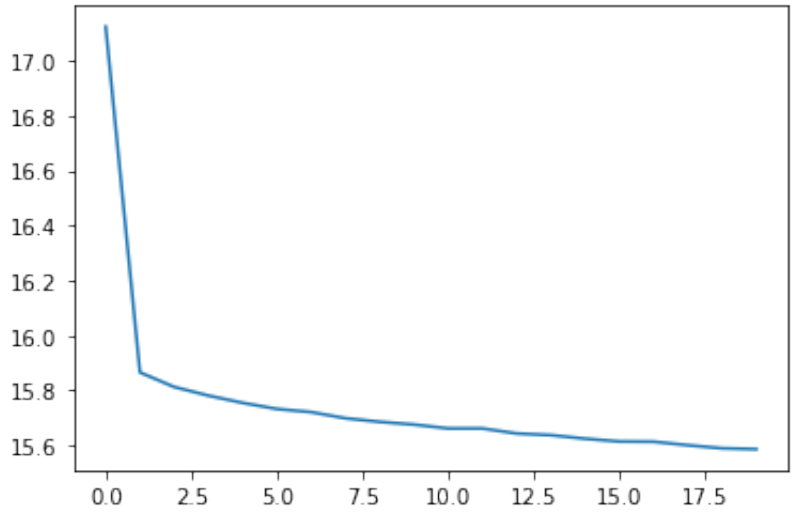
```
# globals
# note: realistically we can only get 20 epochs before overfitting
batch_size = 64
num_epochs = 20
num_layer_1 = 20
num_in = train_x.shape[1]
num_output = 2
```

```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output),
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001, betas=(0.9, 0.999))
loss_fn = nn.BCELoss()
```

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

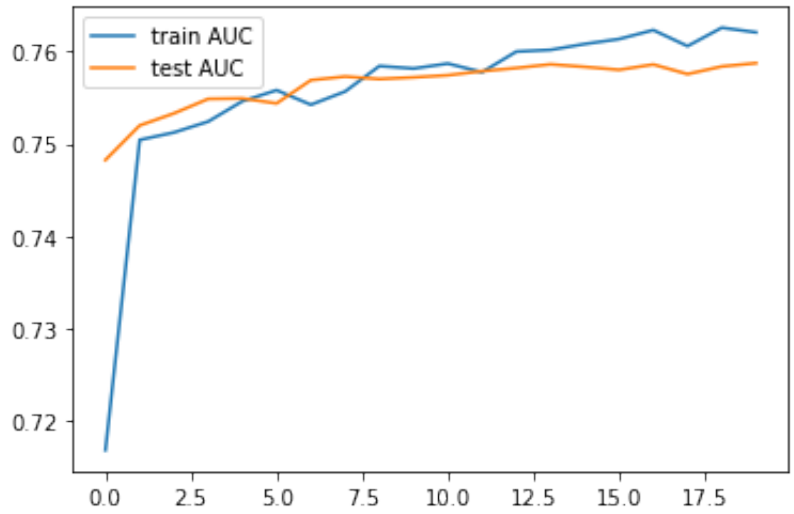
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa912f21950>]
```




```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa9135493d0>
```



```
from sklearn.metrics import roc_auc_score

start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

```
results.loc[2] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "Adam optimizer"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.739418	--	--	1075.331878	0.037791	Deep Learning w/ Application Data
1	Deep Learning	0.758407	--	--	1400.716561	0.048049	Deep Learning w/ all other data
2	Deep Learning	0.758704	--	0.917315	238.759920	0.047533	Adam optimizer

Kaggle Submission

4	Deep Learning	0.732227	--	0.918854	510.319242	1.011763	K-Fold training
5	Deep Learning	0.750459	--	0.917424	264.229233	0.056706	Modifying Layer Sizes

```
test = datasets['application_test']
test = test.merge(PA_df, how='left', on='SK_ID_CURR')
test = test.merge(PCB_df, how='left', on='SK_ID_CURR')
test = test.merge(IP_df, how='left', on='SK_ID_CURR')
test = test.merge(B_df, how='left', on='SK_ID_CURR')
test = test.merge(CCB_df, how='left', on='SK_ID_CURR')

test["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = test['REGION_POPULATION_RELAT
test["AMT_CREDIT/AMT_GOODS_PRICE"] = test['AMT_CREDIT'] / test['AMT_GOODS_PRICE']
test["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = test['DEF_30_CNT_SOCIA
test["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = test['DAYS_BIRTH'] + test['DAYS_LAST_P
test["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = test['DEF_30_CNT_SOCIA
test["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = test['AMT_GOODS_PRICE'] + test['DAYS_EMPLO
test["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = test['REGION_POPULATION_RELAT

test["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = test["DAYS_LAST_PHONE_CHANGE"] + test
test["DAYS_BIRTH+MONTHS_BALANCE"] = test["DAYS_BIRTH"] + test["MONTHS_BALANCE_x"]
test["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = test["DAYS_LAST_PHONE_CHANGE"]
test["DAYS_BIRTH*DAYS_CREDIT"] = test["DAYS_BIRTH"] * test["DAYS_CREDIT"]
```

```
# convert test to tensor
test_numpy = scaler.transform(test)
test_tensor = torch.from_numpy(test_numpy).float()

preds = model(test_tensor)[:, 0].detach().numpy()
submit_df = test[['SK_ID_CURR']]
submit_df['TARGET'] = preds

submit_df.to_csv("submission.csv",index=False)

submit_df.head()
```

	SK_ID_CURR	TARGET
0	100001	0.040534
1	100005	0.196219
2	100013	0.023798
3	100028	0.053782
4	100038	0.170401

```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "NN
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fi
Warning: Looks like you're using an outdated API Version, please consider upd
100% 878k/878k [00:00<00:00, 4.15MB/s]
Successfully submitted to Home Credit Default Risk
```

More Layers

```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUBLISH",
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG",
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT",

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
# globals
# note: realistically we can only get 20 epochs before overfitting
batch_size = 64
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2
```

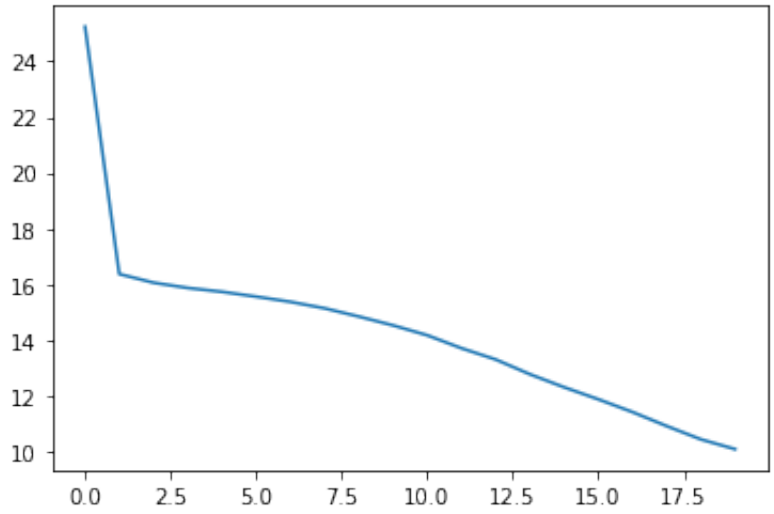
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.BatchNorm1d(128),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.BatchNorm1d(64),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.1),
        nn.Linear(32, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001)
loss_fn = nn.BCELoss()
```

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

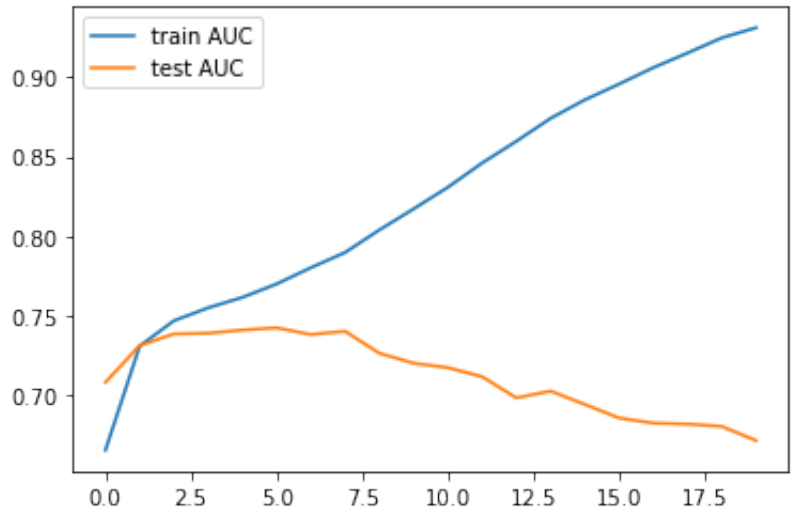
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa912314190>]
```




```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa9188ece50>
```




```
start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

```
results.loc[3] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "More layers"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.739418	--	--	1075.331878	0.037791	Deep Learning w/ Application Data
1	Deep Learning	0.758407	--	--	1400.716561	0.048049	Deep Learning w/ all other data

K-Fold Training

Deep

```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUBLISH",
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG",
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT",

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])



scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
# globals
batch_size = 128
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2

kfold = KFold(n_splits=5, shuffle=True)
indexes_gen = kfold.split(train_x_tensor)
```

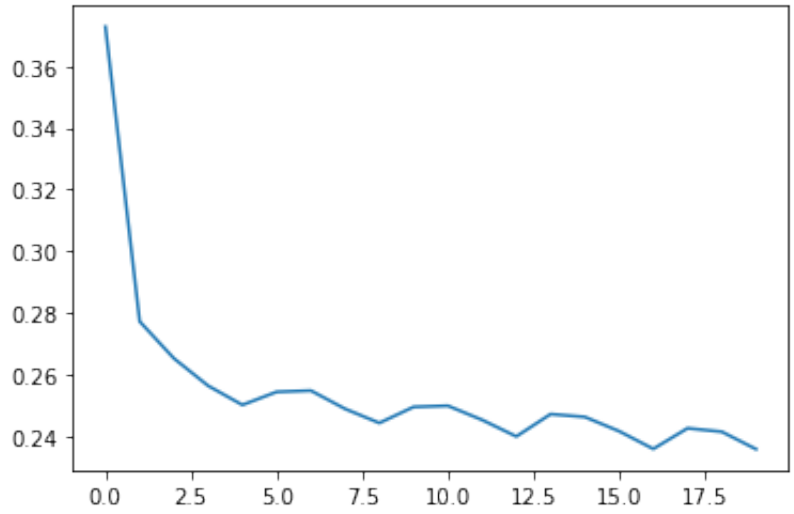
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 256),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.BatchNorm1d(256),
        nn.Linear(256, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.BatchNorm1d(128),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.BatchNorm1d(64),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.2),
        nn.Linear(32, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001)
loss_fn = nn.BCELoss()
```

```
data_loaders = []
test_idxs = []
for train_idx, test_idx in kfold.split(train_x_tensor):
  dataset = torch.utils.data.TensorDataset(train_x_tensor[train_idx], train_y_tens
  data_loaders.append(torch.utils.data.DataLoader(dataset, batch_size=batch_size,
  test_idxs.append(test_idx)
```

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs
curr_fold = 0

start = time()
for epoch in range(epochs):
  # get fold
  if curr_fold+1 >= len(data_loaders):
    curr_fold = 0
  data_loader = data_loaders[curr_fold]
  test_idx = test_idxs[curr_fold]

  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

  # get test-train scores
  preds = model(train_x_tensor[test_idx])[:, 0].detach()
  loss = loss_fn(preds, train_y_tensor[test_idx]).detach()
  try:
    auc = roc_auc_score(train_y_tensor[test_idx], preds.detach().numpy())
  except:
    auc = 0

  losses.append(loss)
  roc_scores.append(auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
  curr_fold += 1
train_time = time() - start
```

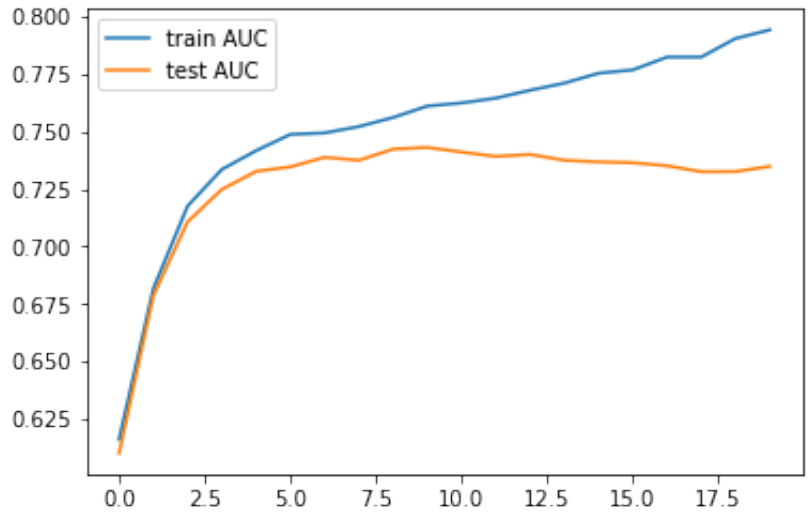
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa91252b110>]
```




```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa913044b10>
```




```
start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

```
results.loc[4] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "K-Fold training"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.739418	--	--	1075.331878	0.037791	Deep Learning w/ Application Data
1	Deep Learning	0.758407	--	--	1400.716561	0.048049	Deep Learning w/ all other data
	Deep						Adam

Other Layer Sizes

```
train = datasets['application_train']
train = train.merge(PA_df, how='left', on='SK_ID_CURR')
train = train.merge(PCB_df, how='left', on='SK_ID_CURR')
train = train.merge(IP_df, how='left', on='SK_ID_CURR')
train = train.merge(B_df, how='left', on='SK_ID_CURR')
train = train.merge(CCB_df, how='left', on='SK_ID_CURR')

train["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = train['REGION_POPULATION_REL
train["AMT_CREDIT/AMT_GOODS_PRICE"] = train['AMT_CREDIT'] / train['AMT_GOODS_PRICE
train["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = train['DAYS_BIRTH'] + train['DAYS_LAS
train["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = train['DEF_30_CNT_SOC
train["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = train['AMT_GOODS_PRICE'] + train['DAYS_EM
train["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = train['REGION_POPULATION_REL

train["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE"] + tr
train["DAYS_BIRTH+MONTHS_BALANCE"] = train["DAYS_BIRTH"] + train["MONTHS_BALANCE_x
train["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = train["DAYS_LAST_PHONE_CHANGE
train["DAYS_BIRTH*DAYS_CREDIT"] = train["DAYS_BIRTH"] * train["DAYS_CREDIT"]
```

```
cat_features = [
  "FLAG_DOCUMENT_3", "REGION_RATING_CLIENT", "REGION_RATING_CLIENT_W_CITY",
  "NAME_INCOME_TYPE", "NAME_EDUCATION_TYPE", "HOUR_APPR_PROCESS_START",
  "OCCUPATION_TYPE", "FLAG_DOCUMENT_4",
  "REG_CITY_NOT_WORK_CITY", "REG_CITY_NOT_LIVE_CITY",

  "NAME_SELLER_INDUSTRY", "NAME_PORTFOLIO", "CREDIT_TYPE", "CREDIT_ACTIVE",

  "STATUS_MIN", "STATUS_MAX"
]

num_features = [
  "EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "FLOORSMAX_AVG",
  "AMT_GOODS_PRICE", "REGION_POPULATION_RELATIVE",
  "ELEVATORS_AVG", "DAYS_LAST_PHONE_CHANGE", "DAYS_BIRTH", "DAYS_ID_PUBLISH",
  "DAYS_EMPLOYED", "FLOORSMIN_AVG", "TOTALAREA_MODE", "APARTMENTS_AVG",
  "LIVINGAPARTMENTS_AVG", "DAYS_REGISTRATION", "OWN_CAR_AGE",
  "DEF_30_CNT_SOCIAL_CIRCLE", "DEF_60_CNT_SOCIAL_CIRCLE",

  "REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH", "AMT_CREDIT/AMT_GOODS_PRICE",
  "DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE",
  "DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE",
  "DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE",
  "AMT_GOODS_PRICE+DAYS_EMPLOYED", "REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE",

  "CNT_INSTALMENT", "MONTHS_BALANCE_x", "DAYS_ENTRY_PAYMENT", "DAYS_INSTALMENT",
  "DAYS_CREDIT", "AMT_BALANCE", "MONTHS_BALANCE_y", "AMT_CREDIT_LIMIT_ACTUAL",

  "DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT", "DAYS_BIRTH+MONTHS_BALANCE",
  "DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT", "DAYS_BIRTH*DAYS_CREDIT",

  "PREV_CNT_INSTALMENT","PREV_CNT_INSTALMENT_FUTURE",
  "PREV_PCB_MONTHS_BALANCE","PREV_AMT_INSTALMENT","PREV_AMT_PAYMENT",
  "PREV_DAYS_INSTALMENT","PREV_DAYS_ENTRY_PAYMENT","PREV_AMT_BALANCE",
  "PREV_CCB_MONTHS_BALANCE","PREV_AMT_CREDIT_LIMIT_ACTUAL",
  "MONTHS_BALANCE_MIN","STATUS_COUNT"
]
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])


scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
# to tensors
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
# globals
batch_size = 64
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2
```

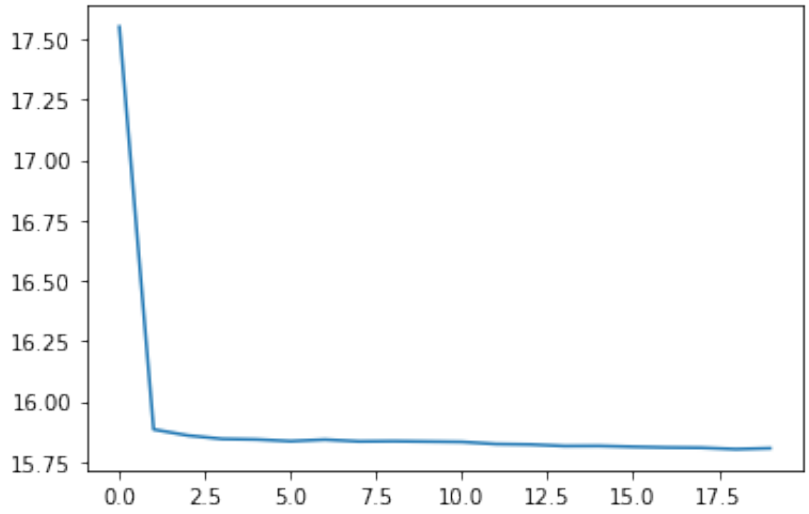
```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 20),
        nn.ReLU(),
        nn.Linear(20, 20),
        nn.ReLU(),
        nn.Linear(20, num_output),
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001, betas=(0.9, 0.9))
loss_fn = nn.BCELoss()
```

```
from time import time

losses = []
roc_scores = []
test_roc_scores = []
epochs = num_epochs

start = time()
for epoch in range(epochs):
  running_loss = 0.0
  running_auc = 0.0
  num_train_auc = 0
  for batch, data in enumerate(data_loader):
    input, labels = data[0], data[1]

    opt.zero_grad()
    pred = model(input)[:, 0]
    loss = loss_fn(pred, labels)
    loss.backward()
    opt.step()

    running_loss += loss.detach()
    try:
      running_auc += roc_auc_score(labels, pred.detach().numpy())
      num_train_auc += 1
    except: pass

  losses.append(running_loss/batch_size)
  roc_scores.append(running_auc/num_train_auc)
  preds = model(test_x_tensor)[:,0].detach().numpy()
  test_roc_scores.append(roc_auc_score(test_y, preds))
train_time = time() - start
```

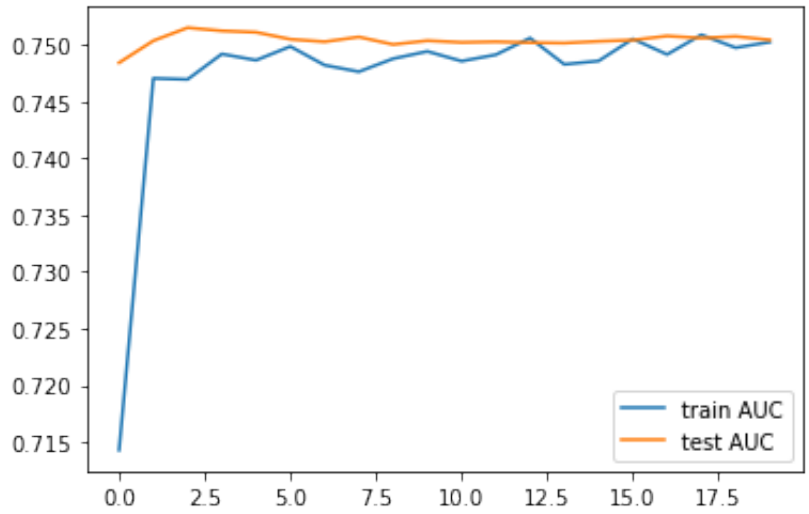
```
import matplotlib.pyplot as plt

plt.plot(range(epochs), losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa91273fbd0>]
```




```
plt.plot(range(epochs), roc_scores, label="train AUC")
plt.plot(range(epochs), test_roc_scores, label="test AUC")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fa91236aed0>
```




```
start = time()
preds = model(test_x_tensor)[:,0].detach().numpy()
roc = roc_auc_score(test_y, preds)
test_time = time() - start

acc = np.sum(np.round(preds) == test_y) / len(test_y)
```

```
results.loc[5] = ["Deep Learning", roc, "--", acc,
                  train_time, test_time, "Modifying Layer Sizes"]

results
```

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Deep Learning	0.739418	--	--	1075.331878	0.037791	Deep Learning w/ Application Data
1	Deep Learning	0.758407	--	--	1400.716561	0.048049	Deep Learning w/ all other data
2	Deep Learning	0.758383	--	0.917359	254.027061	0.046805	Adam optimizer

Kaggle Submission

4	Deep	0.732227	--	0.918854	510.319242	1.011763	K-Fold training

```
test = datasets['application_test']
test = test.merge(PA_df, how='left', on='SK_ID_CURR')
test = test.merge(PCB_df, how='left', on='SK_ID_CURR')
test = test.merge(IP_df, how='left', on='SK_ID_CURR')
test = test.merge(B_df, how='left', on='SK_ID_CURR')
test = test.merge(CCB_df, how='left', on='SK_ID_CURR')

test["REGION_POPULATION_RELATIVE*DAYS_ID_PUBLISH"] = test['REGION_POPULATION_RELAT
test["AMT_CREDIT/AMT_GOODS_PRICE"] = test['AMT_CREDIT'] / test['AMT_GOODS_PRICE']
test["DEF_30_CNT_SOCIAL_CIRCLE/OBS_30_CNT_SOCIAL_CIRCLE"] = test['DEF_30_CNT_SOCIA
test["DAYS_BIRTH+DAYS_LAST_PHONE_CHANGE"] = test['DAYS_BIRTH'] + test['DAYS_LAST_P
test["DEF_30_CNT_SOCIAL_CIRCLE+DEF_60_CNT_SOCIAL_CIRCLE"] = test['DEF_30_CNT_SOCIA
test["AMT_GOODS_PRICE+DAYS_EMPLOYED"] = test['AMT_GOODS_PRICE'] + test['DAYS_EMPLO
test["REGION_POPULATION_RELATIVE*AMT_GOODS_PRICE"] = test['REGION_POPULATION_RELAT

test["DAYS_LAST_PHONE_CHANGE+CNT_PAYMENT"] = test["DAYS_LAST_PHONE_CHANGE"] + test
test["DAYS_BIRTH+MONTHS_BALANCE"] = test["DAYS_BIRTH"] + test["MONTHS_BALANCE_x"]
test["DAYS_LAST_PHONE_CHANGE+DAYS_ENTRY_PAYMENT"] = test["DAYS_LAST_PHONE_CHANGE"]
test["DAYS_BIRTH*DAYS_CREDIT"] = test["DAYS_BIRTH"] * test["DAYS_CREDIT"]
```

```
# convert test to tensor
test_numpy = scaler.transform(test)
test_tensor = torch.from_numpy(test_numpy).float()

preds = model(test_tensor)[:, 0].detach().numpy()
submit_df = test[['SK_ID_CURR']]
submit_df['TARGET'] = preds

submit_df.to_csv("submission.csv",index=False)

submit_df.head()
```

	SK_ID_CURR	TARGET
0	100001	0.060307
1	100005	0.231580
2	100013	0.022034
3	100028	0.028798
4	100038	0.170697

```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "NN
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fi
Warning: Looks like you're using an outdated API Version, please consider upd
100% 877k/877k [00:00<00:00, 3.37MB/s]
Successfully submitted to Home Credit Default Risk
```

Other Stuff

```
#Sequential API
```

```
import torch
import torch.nn as nn
import numpy as np
import torch.optim as optim
from sklearn.model_selection import train_test_split
train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
```

```
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression

# custom layer to get columns we want from DataFrame
class DataFrameSelector(BaseEstimator, TransformerMixin):
  def __init__(self, attribute_names):
    self.attribute_names = attribute_names
  def fit(self, X, y=None):
    return self
  def transform(self, X):
    return X[self.attribute_names].values
def pct(x):
    return round(100*x,1)
num_pipeline = Pipeline([
  ('selector', DataFrameSelector(num_features)),
  ('imputer', SimpleImputer(strategy='median')),
  ('std_scaler', StandardScaler()),
])

cat_pipeline = Pipeline([
  ('selector', DataFrameSelector(cat_features)),
  ('imputer', SimpleImputer(strategy='constant')),
  ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore")),
])

preprocess_pipeline = FeatureUnion(transformer_list=[
  ("num_pipeline", num_pipeline),
  ("cat_pipeline", cat_pipeline),
])



scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x)
```

```
train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()
```

```
batch_size = 128
num_epochs = 20
num_in = train_x.shape[1]
num_output = 2
```

```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, 256),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.BatchNorm1d(256),
        nn.Linear(256, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Linear(256, 128),
        nn.ReLU(),
        nn.Dropout(0.2),
        nn.BatchNorm1d(128),
        nn.Linear(128, 64),
        nn.ReLU(),
        nn.BatchNorm1d(64),
        nn.Linear(64, 32),
        nn.ReLU(),
        nn.BatchNorm1d(32),
        nn.Dropout(0.2),
        nn.Linear(32, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.Adam(model.parameters(), lr=0.0001)
loss_fn = nn.BCELoss()
```

```
# create data loaders
train_set = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
data_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffl
```

```
import torch
import torchvision
import torch.utils.data
import torchvision.transforms as transforms
import torch.nn as nn
import torch.nn.functional as F
```

```
import torch.optim as optim
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
test_size=0.15

losses = []
# is there a GPU availabale. If available use it
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Assuming that we are on a CUDA machine, this should print a CUDA device:
print(device)

train_x = train.loc[:, train.columns != "TARGET"]
train_y = train['TARGET']
train_x, test_x, train_y, test_y = train_test_split(train_x, train_y, test_size=0.
## Scaling

scaler = preprocess_pipeline.fit(train_x, train_y)

train_x = scaler.transform(train_x)
test_x = scaler.transform(test_x) #Transform test set with the same constants

# convert numpy arrays to tensors

train_x_tensor = torch.from_numpy(train_x).float()
test_x_tensor = torch.from_numpy(test_x).float()
train_y_tensor = torch.from_numpy(np.array(train_y)).float()
test_y_tensor = torch.from_numpy(np.array(test_y)).float()



# create TensorDataset in PyTorch
boston_train = torch.utils.data.TensorDataset(train_x_tensor, train_y_tensor)
boston_validation = torch.utils.data.TensorDataset(test_x_tensor, test_y_tensor)
boston_test = torch.utils.data.TensorDataset(test_x_tensor, test_y_tensor)
# create dataloader
batch_size = 128
trainloader_boston = torch.utils.data.DataLoader(boston_train, batch_size=batch_si
validloader_boston = torch.utils.data.DataLoader(boston_validation, batch_size=tes
testloader_boston = torch.utils.data.DataLoader(boston_test, batch_size=test_x.sha

D_in = test_x.shape[1]
```

```
print(D_in)
D_hidden =20
D_out = 2

#optimizer = optim.SGD(model.parameters(), lr=0.0001)

epochs = range(5)
count = 0
running_loss = 0.0
for epoch in epochs:
    running_loss = 0.0
    for batch, data in enumerate(data_loader):
        input, labels = data[0], data[1]

        running_loss = 0.0
        running_auc = 0.0
        num_train_auc = 0
        opt.zero_grad()
        pred = model(input)[:, 0]
        loss = loss_fn(pred, labels)
        loss.backward()
        opt.step()


        # Clear gradient buffers because we don't want any gradient from previous

        # perform gradient update
        #running_loss += loss.item()*input.size(0)
        count += input.size(0)
        running_loss += loss.detach()
        losses.append(running_loss/batch_size)
    print("Epoch {} batch {} BCE Loss is {}".format(epoch,batch,(running_loss/batc
        # print statistics
    #print(f"Epoch {epoch+1}, mini batch loss {batch+1}, MSE loss: {np.round(runni
#print(losses)
print("finished training")



count = 0
running_loss = 0.0
for batch, data in enumerate(testloader_boston):
    input, labels = data[0], data[1]
    # do forward pass
    output = model(input.float())

    # compute loss and gradients
    loss = loss_fn(output, torch.unsqueeze(labels.float(), dim=1))
    # print statistics
```

```
    running_loss += loss.item()*input.size(0)
    count += input.size(0)
    test_size +=batch_size
print(f" TEST  BCE loss: {np.round(running_loss/count, 3)}")

# predict test
output = model(test_x_tensor.float())
# calculate loss via torch
loss = loss_fn(output, torch.unsqueeze(test_y_tensor.float(), dim=1)).detach().num
#print(loss)
```

```
cpu
173
Epoch 0 batch 2722 BCE Loss is 0.0008841883391141891
Epoch 1 batch 2722 BCE Loss is 0.00023666309425607324
Epoch 2 batch 2722 BCE Loss is 0.0010994228068739176
Epoch 3 batch 2722 BCE Loss is 0.0012235455214977264
Epoch 4 batch 2722 BCE Loss is 0.000625741551630199
finished training
 TEST  BCE loss: 0.103
```

```

```