Будем практиковаться на датасете: https://www.kaggle.com/c/avito-demand-prediction (https://www.kaggle.com/c/avito-demand-prediction)

Ваша задача:

Создать Dataset для загрузки данных (используем только числовые данные) Обернуть его в Dataloader

Написать архитектуру сети, которая предсказывает число показов на основании числовых данных (вы всегда можете нагенерить дополнительных факторов). Сеть должна включать BatchNorm слои и Dropout (или НЕ включать, но нужно обосновать)

Учить будем на функцию потерь с кагла (log RMSE) - нужно её реализовать Сравните сходимость Adam, RMSProp и SGD, сделайте вывод по качеству работы модели

train-test разделение нужно сделать с помощью sklearn random_state=13, test_size = 0.25

In [105]:

```
import math
import torch
import torch.nn.functional as F
import torch.nn as nn
import torch.utils.data as data_utils
from torch.utils.data import TensorDataset, DataLoader

from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [61]:

```
class RangeDataset(torch.utils.data.Dataset):
    def __init__(self, start, end, step=1):
        self.start = start
        self.end = end
        self.step = step

    def __len__(self):
        return math.ceil((self.end - self.start) / self.step)

    def __getitem__(self, index):
        if index > len(self):
          raise RuntimeError
        value = self.start + index * self.step
        return np.array([value, value])
```

In [93]:

```
RANDOM_STATE=13
```

In [63]:

```
dataset = RangeDataset(0, 29)
data_loader = torch.utils.data.DataLoader( dataset, batch_size=6, shuffle=True, num_workers

for i, batch in enumerate(data_loader):
    print(i, batch)
```

```
0 tensor([[ 6,  6],
        [ 0,  0],
        [27, 27],
        [17, 17],
        [21, 21],
        [ 5,  5]], dtype=torch.int32)
1 tensor([[14, 14],
        [15, 15],
        [25, 25],
        [18, 18],
        [ 4,  4],
        [12, 12]], dtype=torch.int32)
2 tensor([[23, 23],
        [ 7,  7],
        [19, 19],
        [ 9,  9],
        [24, 24],
        [11, 11]], dtype=torch.int32)
3 tensor([[26, 26],
        [10, 10],
        [ 2,  2],
        [16, 16],
        [13, 13],
        [20, 20]], dtype=torch.int32)
```

item_id - Идентификатор объявления.
user_id - Идентификатор пользователя.
region - Рекламный регион.
city - Рекламный город.
parent_category_name - Категория объявлений верхнего уровня, классифицированная по рекламной модели Avito.
category_name - Категория мелкозернистых объявлений, классифицированная по рекламной модели Avito.
param_1 - Необязательный параметр из рекламной модели Avito.
param_2 - Необязательный параметр из рекламной модели Avito.
param_3 - Необязательный параметр из рекламной модели Avito.
title - Название объявления.
description - Описание объявления.
price - Цена объявления.
item_seq_number - Порядковый номер объявления для пользователя.
activation_date- Дата размещения объявления.
user_type - Тип пользователя.
image - Идентификационный код изображения. Привязка к файлу jpg в train_jpg. Не в каждой рекламе есть изображение.
image_top_1 - Классификационный код Авито для изображения.

deal_probability - Целевая переменная. Это вероятность того, что объявление действительно что-то продало. Невозможно с уверенностью проверить каждую транзакцию, поэтому значение этого столбца может быть любым с плавающей точкой от нуля до единицы.

In [64]:

```
data = pd.read_csv("train.csv")
```

In [65]:

```
data = data.head(300000)
```

In [66]:

```
data.shape
```

Out[66]:

```
(300000, 18)
```

In [67]:

```
data.head()
```

Out[67]:

	item_id	user_id	region	city	parent_category_name	category_n
0	b912c3c6a6ad	e00f8ff2eaf9	Свердловская область	Екатеринбург	Личные вещи	Товары дет игру
1	2dac0150717d	39aeb48f0017	Самарская область	Самара	Для дома и дачи	Мебе интер
2	ba83aefab5dc	91e2f88dd6e3	Ростовская область	Ростов-на-Дону	Бытовая электроника	Аудио и ви
3	02996f1dd2ea	bf5cccea572d	Татарстан	Набережные Челны	Личные вещи	Товары дет игру
4	7c90be56d2ab	ef50846afc0b	Волгоградская область	Волгоград	Транспорт	Автомоб

In [68]:

```
data.isna().sum()
```

Out[68]:

```
item_id                     0
user_id                     0
region                      0
city                        0
parent_category_name        0
category_name               0
param_1                 11940
param_2                130893
param_3                172370
title                       0
description             23084
price                   17048
item_seq_number             0
activation_date             0
user_type                   0
image                   22544
image_top_1             22544
deal_probability            0
dtype: int64
```

In [69]:

```
data.drop(['param_1', 'param_2', 'param_3', 'description', 'image', 'image_top_1'], 1, inpl
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning: In a future version of pandas all arguments of DataFrame.drop except
for the argument 'labels' will be keyword-only
  """Entry point for launching an IPython kernel.
```

In [70]:

```
data.drop(['item_id', 'user_id'], 1, inplace= True)
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning: In a future version of pandas all arguments of DataFrame.drop except
for the argument 'labels' will be keyword-only
  """Entry point for launching an IPython kernel.
```

In [71]:

```
data['price'] = data['price'].fillna(data['price'].median())
```

In [72]:

```
columns = data.columns
```

In [73]:

```
for i in columns:
    print(i,   len(data[i].unique()), data[i].dtype)
# data[[columns]].unique()
```

```
region 28 object
city 1511 object
parent_category_name 9 object
category_name 47 object
title 186560 object
price 6570 float64
item_seq_number 10670 int64
activation_date 18 object
user_type 3 object
deal_probability 9599 float64
```

In [74]:

```
data.drop('title', 1, inplace= True)
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning: In a future version of pandas all arguments of DataFrame.drop except
for the argument 'labels' will be keyword-only
  """Entry point for launching an IPython kernel.
```

In [75]:

```
data = pd.get_dummies(data)
```

In [76]:

```
data.shape
```

Out[76]:

```
(300000, 1619)
```

In [77]:

```
X = data.drop('deal_probability', 1)
```

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: FutureW
arning: In a future version of pandas all arguments of DataFrame.drop except
for the argument 'labels' will be keyword-only
  """Entry point for launching an IPython kernel.
```

In [78]:

```
y = data['deal_probability']
```

In [80]:

```
from sklearn.preprocessing import StandardScaler
```

In [81]:

```
scaler = StandardScaler()
X_features = scaler.fit_transform(X)
```

In [82]:

```
from sklearn.decomposition import PCA
```

In [83]:

```
pca = PCA(n_components=15)
pca.fit(X)
```

Out[83]:

```
PCA(n_components=15)
```

In [84]:

```
x_pca = pca.transform(X)
```

In [85]:

```
x_pca.shape
```

Out[85]:

```
(300000, 15)
```

In [87]:

```
xx = pd.DataFrame(x_pca)
```

In [88]:

```
xx[:5]
```

Out[88]:

	0	1	2	3	4	5	6	7
0	-242238.663885	-720.835603	0.603805	-0.016663	0.015591	-0.027468	0.058606	-0.231576
1	-239638.662100	-704.110856	-0.307752	-0.727026	-0.038502	0.921677	0.311521	-0.086605
2	-238638.663165	-714.216716	-0.311985	-0.637023	-0.024123	-0.664669	0.485251	-0.115250
3	-240438.633829	-437.026155	-0.232047	1.107202	0.047632	0.066343	0.111090	-0.099740
4	-202638.664001	-724.028028	-0.219064	-0.624202	-0.011004	-0.086540	-0.101278	-0.089342

In [89]:

```
xx.to_csv('X_new.csv')
```

In [112]:

```
X = pd.read_csv('X_new.csv', index_col=0)
```

In [113]:

```
X.shape
```

Out[113]:

```
(300000, 15)
```

In [114]:

```
X[:5]
```

Out[114]:

	0	1	2	3	4	5	6	7
0	-242238.663885	-720.835603	0.603805	-0.016663	0.015591	-0.027468	0.058606	-0.231576
1	-239638.662100	-704.110856	-0.307752	-0.727026	-0.038502	0.921677	0.311521	-0.086605
2	-238638.663165	-714.216716	-0.311985	-0.637023	-0.024123	-0.664669	0.485251	-0.115250
3	-240438.633829	-437.026155	-0.232047	1.107202	0.047632	0.066343	0.111090	-0.099740
4	-202638.664001	-724.028028	-0.219064	-0.624202	-0.011004	-0.086540	-0.101278	-0.089342

In [115]:

```
dim = X.shape[1]
dim
```

Out[115]:

```
15
```

In [155]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state=RA
```

In [163]:

```
class Perceptron(nn.Module):
    def __init__(self, input_dim, output_dim, activation="relu"):
        super(Perceptron, self).__init__()
        self.fc = nn.Linear(input_dim, output_dim)
        self.activation = activation

    def forward(self, x):
        x = self.fc(x)
        if self.activation=="relu":
            return F.relu(x)
        if self.activation=="sigmoid":
            return F.sigmoid(x)
        raise RuntimeError


class FeedForward(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(FeedForward, self).__init__()
        self.bn1 = nn.BatchNorm1d(input_dim)
        self.fc1 = Perceptron(input_dim, hidden_dim)
        self.bn2 = nn.BatchNorm1d(hidden_dim)
        self.dp = nn.Dropout(0.25)
        self.fc2 = Perceptron(hidden_dim, 10, "relu")

    def forward(self, x):
        x = self.bn1(x)
        x = self.fc1(x)
        x = self.dp(x)
        x = self.fc2(x)
        return x
```

In [164]:

```
net = FeedForward(dim, 8)

optimizer = torch.optim.Adam(net.parameters(), lr = 0.01)
criterion =  nn.MSELoss()
```

In [158]:

```
# torch_tensor = torch.tensor(data['deal_probability'].values)
```

In [159]:

```
# X_train, X_test, y_train, y_test

X_train = torch.DoubleTensor(X_train.values)
X_test = torch.DoubleTensor(X_test.values)
y_train = torch.DoubleTensor(y_train.values)
y_test = torch.DoubleTensor(y_test.values)
```

In [167]:

```
# torch.set_default_dtype(torch.float64)
num_epochs = 10
# target = pd.DataFrame(y)
train = data_utils.TensorDataset(X_train, y_train)
train_loader = data_utils.DataLoader(train, batch_size=10, shuffle=True, drop_last=True)
```

In [168]:

```
from tqdm import tqdm
```

In [169]:

```
for epoch in tqdm(range(num_epochs)):
    running_loss = 0.0
    for i, data in enumerate(train_loader, 0):
        inputs, labels = data[0], data[1]

        # обнуляем градиент
        optimizer.zero_grad()

        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # выводим статистику о процессе обучения
        running_loss += loss.item()
        if i % 300 == 0:    # печатаем каждые 300 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 300))
            running_loss = 0.0

print('Training is finished!')
```

```
  0%|
| 0/10 [00:00<?, ?it/s]
```

In [ ]:

```
```