# Фреймворк PyTorch для разработки искусственных нейронных сетей

# Урок 3. Dataset, Dataloader, BatchNorm, Dropout, Оптимизация

## Практическое задание

## Попрактикуемся с тем, что изучили

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

Ваша задача:

1. Создать Dataset для загрузки данных (используем только числовые данные)
2. Обернуть его в Dataloader
3. 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)
4. Учить будем на функцию потерь с кагла (log RMSE) - нужно её реализовать
5. 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

Вопросы? в личку @Kinetikm

Выполнил ***Соковнин ИЛ***

## Как загрузить данные на Colaboratory

https://www.youtube.com/watch?v=Ve5oW1qqbZg
https://colab.research.google.com/drive/1e3bbNmpQ1Wremp4O5NiNOgqTV-_WG5wL#scrollTo=1_lwnzf4Bhlu

## Шаг 0. Подготовка данных

Загружаем данные с Google Drive, затем распаковываем их в папку /content/drive/my_drive/data

```
!ls -la
```

```
total 16
drwxr-xr-x 1 root root 4096 Mar 23 14:22 .
drwxr-xr-x 1 root root 4096 Mar 30 08:53 ..
drwxr-xr-x 4 root root 4096 Mar 23 14:21 .config
drwxr-xr-x 1 root root 4096 Mar 23 14:22 sample_data
```

Создаём дирректорию для данных

```
!mkdir -p /content/drive/my_drive/data
```

## Использование Google Drive

Копирование больших файлов просходит значительно быстрее, если использовать Google Drive.

Подключение Google Drive к виртуальной машине:

```
from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)
```

```
Mounted at /content/gdrive
```

```
!ls /content/gdrive/"My Drive" -la
```

```
total 2332713
drwx------ 2 root root       4096 Dec  9  2020 'Colab Notebooks'
-rw------- 1 root root 2388667849 Dec  4 16:43  features.csv.zip
drwx------ 2 root root       4096 Jan 21 08:43  nn
drwx------ 2 root root       4096 Mar 27 08:53  PyTorch
-rw------- 1 root root      15020 Oct 29  2020  task_1.drawio
-rw------- 1 root root       1767 Oct 29  2020 'Untitled Diagram.drawio'
-rw------- 1 root root        143 Apr  2  2021  Резюме.gdoc
```

```
!ls /content/gdrive/"My Drive"/PyTorch/lesson3/data -la
```

```
total 443208
-rw------- 1 root root   4144048 Mar 27 08:34 sample_submission.csv.zip
-rw------- 1 root root 116236318 Mar 28 08:46 test.csv.zip
-rw------- 1 root root 333463334 Mar 28 08:48 train.csv.zip
```

Просматриваем подключенные диски

```
!df -h
```

```
Filesystem      Size  Used Avail Use% Mounted on
overlay         108G   40G   69G  37% /
tmpfs            64M     0   64M   0% /dev
shm             5.8G     0  5.8G   0% /dev/shm
/dev/root       2.0G  1.2G  817M  59% /sbin/docker-init
tmpfs           6.4G   32K  6.4G   1% /var/colab
/dev/sda1        81G   44G   38G  54% /etc/hosts
tmpfs           6.4G     0  6.4G   0% /proc/acpi
tmpfs           6.4G     0  6.4G   0% /proc/scsi
tmpfs           6.4G     0  6.4G   0% /sys/firmware
drive            15G  3.3G   12G  22% /content/gdrive
```

```
!pwd
```

```
/content
```

Просматриваем содержимое диска

```
!ls /content/gdrive/
```

```
MyDrive
```

```
!ls /content/gdrive/"My Drive"/PyTorch/lesson3/data -la
```

```
total 443208
-rw------- 1 root root   4144048 Mar 27 08:34 sample_submission.csv.zip
-rw------- 1 root root 116236318 Mar 28 08:46 test.csv.zip
-rw------- 1 root root 333463334 Mar 28 08:48 train.csv.zip
```

Копирование данных с Google Drive на локальный диск виртуальной машины

```
# !cp /content/gdrive/'My Drive'/PyTorch/lesson3/data/*.* .
!cp /content/gdrive/'My Drive'/PyTorch/lesson3/data/train.csv.zip .
```

```
!ls -la
```

```
total 325676
drwxr-xr-x 1 root root      4096 Mar 30 08:55 .
drwxr-xr-x 1 root root      4096 Mar 30 08:53 ..
drwxr-xr-x 4 root root      4096 Mar 23 14:21 .config
drwxr-xr-x 3 root root      4096 Mar 30 08:54 drive
drwx------ 5 root root      4096 Mar 30 08:55 gdrive
drwxr-xr-x 1 root root      4096 Mar 23 14:22 sample_data
-rw------- 1 root root 333463334 Mar 30 08:55 train.csv.zip
```

```
!mv train.csv.zip /content/drive/my_drive/data/train.csv.zip
# !mv test.csv.zip /content/drive/my_drive/data/test.csv.zip
# !mv sample_submission.csv.zip /content/drive/my_drive/data/sample_submission.csv.zip
```

```
!ls -la
```

```
total 24
drwxr-xr-x 1 root root 4096 Mar 30 08:55 .
```

```
drwxr-xr-x 1 root root 4096 Mar 30 08:53 ..
drwxr-xr-x 4 root root 4096 Mar 23 14:21 .config
drwxr-xr-x 3 root root 4096 Mar 30 08:54 drive
drwx------ 5 root root 4096 Mar 30 08:55 gdrive
drwxr-xr-x 1 root root 4096 Mar 23 14:22 sample_data
```

Распаковка

```
!ls -la /content/drive/my_drive/data
```

```
total 325660
drwxr-xr-x 2 root root      4096 Mar 30 08:55 .
drwxr-xr-x 3 root root      4096 Mar 30 08:54 ..
-rw------- 1 root root 333463334 Mar 30 08:55 train.csv.zip
```

```
!unzip '/content/drive/my_drive/data/train.csv.zip' -d '/content/drive/my_drive/data'
# !unzip '/content/drive/my_drive/data/test.csv.zip' -d '/content/drive/my_drive/data'
# !unzip '/content/drive/my_drive/data/sample_submission.csv.zip' -d '/content/drive/my_drive/data'
```

```
Archive:  /content/drive/my_drive/data/train.csv.zip
  inflating: /content/drive/my_drive/data/train.csv
```

# Import libs

```
import os, sys
import numpy as np
import pandas as pd

import torch
from torch import nn
from torch import optim

import torchvision
import torchvision.transforms as transforms

from tqdm import tqdm  # means "progress" in Arabic (taqadum)

# import matplotlib.pyplot as plt ### воспользуемся для отображения изображения
```

```
!pwd
```

```
/content
```

## *File and column descriptions*

**train.csv** - Train data.
**item_id** - Ad id.
**user_id** - User id.
**region** - Ad region.
**city** - Ad city.
**parent_category_name** - Top level ad category as classified by Avito's ad model.
**category_name** - Fine grain ad category as classified by Avito's ad model.
**param_1** - Optional parameter from Avito's ad model.
**param_2** - Optional parameter from Avito's ad model.
**param_3** - Optional parameter from Avito's ad model.
**title** - Ad title.
**description** - Ad description.
**price** - Ad price.
**item_seq_number** - Ad sequential number for user.
**activation_date** - Date ad was placed.
**user_type** - User type.
**image** - Id code of image. Ties to a jpg file in train_jpg. Not every ad has an image.
**image_top_1** - Avito's classification code for the image.
**deal_probability** - The target variable.

This is the likelihood that an ad actually sold something. It's not possible to verify every transaction with certainty, so this column's value can be any float from zero to one.

# 1 Data loading

```
PATH_DATA = '/content/drive/my_drive/data/'
```

## 1.1 Train data

```
data_train = pd.read_csv(PATH_DATA + "train.csv")
# train = pd.read_csv(PATH_DATA + "train.csv")
# data_train = train.head(3000)
```

```
data_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1503424 entries, 0 to 1503423
Data columns (total 18 columns):
 #   Column                Non-Null Count    Dtype  
---  ------                --------------    -----  
 0   item_id               1503424 non-null  object 
 1   user_id               1503424 non-null  object 
 2   region                1503424 non-null  object 
 3   city                  1503424 non-null  object 
 4   parent_category_name  1503424 non-null  object 
 5   category_name         1503424 non-null  object 
 6   param_1               1441848 non-null  object 
 7   param_2               848882 non-null   object 
 8   param_3               640859 non-null   object 
 9   title                 1503424 non-null  object 
 10  description           1387148 non-null  object 
 11  price                 1418062 non-null  float64
 12  item_seq_number       1503424 non-null  int64  
 13  activation_date       1503424 non-null  object 
 14  user_type             1503424 non-null  object 
 15  image                 1390836 non-null  object 
 16  image_top_1           1390836 non-null  float64
 17  deal_probability      1503424 non-null  float64
dtypes: float64(3), int64(1), object(14)
memory usage: 206.5+ MB
```

```
data_train.head(3)
```

```
# data_train.dtypes
```

```
data_train['activation_date_int'] = pd.to_numeric(data_train['activation_date'].str.replace("-", "")).astype(int)
data_train['activation_date'] = pd.to_datetime(data_train.activation_date)
data_train['day_of_month'] = data_train.activation_date.apply(lambda x: x.day)
data_train['day_of_week'] = data_train.activation_date.apply(lambda x: x.weekday())
data_train['month'] = data_train.activation_date.apply(lambda x: x.month)
# data_train['month'] = data_train['activation_date'].dt.month
```

```
data_train[['activation_date', 'activation_date_int', 'day_of_month', 'day_of_week', 'month']].head()
```

```
group = data_train.groupby('region')['deal_probability']
# group.all()
group.count()
```

```
region
Алтайский край              41520
Башкортостан                68291
Белгородская область        28868
Владимирская область        26741
Волгоградская область       48998
Воронежская область         44116
Иркутская область           44030
Калининградская область     32756
Кемеровская область         44635
Краснодарский край         141416
Красноярский край           53442
Нижегородская область       73643
Новосибирская область       62486
Омская область              42939
Оренбургская область        29303
Пермский край               62704
Ростовская область          89995
Самарская область           73407
Саратовская область         49645
Свердловская область        94475
Ставропольский край         39187
Татарстан                   81284
Тульская область            25733
Тюменская область           35411
Удмуртия                    28537
Ханты-Мансийский АО         28709
Челябинская область         78339
Ярославская область         32814
Name: deal_probability, dtype: int64
```

```
group.describe()
```

```
from tqdm import notebook  # IPython/Jupyter Notebook progressbar decorator for iterators.

cols = ['region', 'city', 'category_name','user_type']

for col in notebook.tqdm(cols):
    group = data_train.groupby(col)['deal_probability']  # deal_probability - вероятность сделки
    mean = group.mean()
    data_train[col + '__deal_probability_avg'] = data_train[col].map(mean)  # convert the data to the desired format
data_train['_deal_probability'] = data_train['deal_probability']
```

```
data_train = data_train.drop([
'item_id',
'user_id',
'region',
'city',
'parent_category_name',
'category_name',
'param_1',
'param_2',
'param_3',
'title',
'description',
# 'price',
# 'item_seq_number',
'activation_date',
'user_type',
'image',
# 'image_top_1',
'deal_probability'
], axis=1)

cols
```

```
     ['region', 'city', 'category_name', 'user_type']
```

```
for col in data_train.columns:
    if data_train[col].isna().sum() > 0:
        data_train[col].fillna(data_train[col].median(), inplace=True)

# В итоге получаем
data_train.head(3)
```

## 1. **Создать Dataset для загрузки данных** (используем только числовые данные)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

```
# data_train = data_train.head(10000)
```

```
# разделим данные на train/test
# train-test разделение нужно сделать с помощью sklearn random_state=13, test_size = 0.25
X_train, X_test = train_test_split(data_train, test_size =0.25, random_state=13)

X_train.head(3)
```

```
idx=0
y_0 = X_train.iloc[idx, -1:]
print(y_0)
x_0 = X_train.iloc[idx, 0:-1]
print(x_0)
```

```
     _deal_probability    0.0
     Name: 31723, dtype: float64
     price                                1.300000e+03
     item_seq_number                      6.000000e+00
     image_top_1                          1.056000e+03
     activation_date_int                  2.017032e+07
     day_of_month                         2.300000e+01
     day_of_week                          3.000000e+00
     month                                3.000000e+00
     region__deal_probability_avg         1.470655e-01
     city__deal_probability_avg           1.890690e-01
     category_name__deal_probability_avg  1.784801e-01
     user_type__deal_probability_avg      1.495572e-01
     Name: 31723, dtype: float64
```

```
from torch.utils.data import Dataset

class AvitoDataset(Dataset):
    def __init__(self, init_dataset, normalize=False, transform=None):
      self._base_dataset = init_dataset
      self.normalize = normalize
      self.transform = transform

    def __len__(self):
      return len(self._base_dataset)

    def __getitem__(self, idx):
      """ Основной метод-точка входа для наборов данных PyTorch

      Аргументы:
          idx (int): индекс данных
      Возвращает:
          словарь признаков (x_data) и метки (y_target) данных
      """
      y = self._base_dataset.iloc[idx, -1:]
      # y = torch.tensor(y.values, dtype=torch.float)
      y = torch.FloatTensor(y.values)
      row = torch.FloatTensor(self._base_dataset.iloc[idx, 0:-1].values)

      return row, y


df_train, df_test = AvitoDataset(X_train), AvitoDataset(X_test)
```

## 2. Обернуть его в Dataloader

Чтобы изменить содержимое ячейки, дважды нажмите на нее (или выберите "Ввод")

```
BATCH_SIZE = 1024
train_loader = torch.utils.data.DataLoader(df_train,
                                           batch_size=BATCH_SIZE,
                                           shuffle=True,
                                           num_workers=1)

test_loader = torch.utils.data.DataLoader(df_test,
                                          batch_size=BATCH_SIZE,
                                          shuffle=True,
                                          num_workers=1)
```

**3. 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)

```
# import torch.nn.functional as F

# class FeedForwardNN(nn.Module):
#     def __init__(self, input_dim, hidden_dim):
#         ''' В init() - делаем необходимые настройки.

#         Аргументы:
#           input_dim - размерность вектора входных данных
#           hidden_dim - параметр для настройки скрытых слоёв
#         '''
#         super(FeedForwardNN, self).__init__()

#         self.bn1 = nn.BatchNorm1d(input_dim)
#         self.fc1 = nn.Linear(input_dim, 4*hidden_dim)
#         self.dp1 = nn.Dropout(0.25)

#         self.bn2 = nn.BatchNorm1d(4*hidden_dim)
#         self.fc2 = nn.Linear(4*hidden_dim, 3*hidden_dim)
#         self.dp2 = nn.Dropout(0.15)

#         self.bn3 = nn.BatchNorm1d(3*hidden_dim)
#         self.fc3 = nn.Linear(3*hidden_dim, 2*hidden_dim)
#         self.dp3 = nn.Dropout(0.15)

#         self.bn4 = nn.BatchNorm1d(2*hidden_dim)
#         self.fc4 = nn.Linear(2*hidden_dim, 1)

#     def forward(self, x):
#         x = self.bn1(x)
#         x = self.fc1(x)
#         # Задаём функцию потерь Leaky_relu
#         # x = F.leaky_relu(x, 0.05)
#         x = torch.tanh(x)
#         x = self.dp1(x)

#         x = self.bn2(x)
#         x = self.fc2(x)
#         # Задаём функцию потерь Leaky_relu
#         # x = F.leaky_relu(x, 0.05)
#         x = torch.tanh(x)
#         x = self.dp2(x)

#         x = self.bn3(x)
#         x = self.fc3(x)
#         # Задаём функцию потерь Leaky_relu
#         # x = F.leaky_relu(x, 0.05)
#         x = torch.tanh(x)
#         x = self.dp3(x)
```

```
#         x = self.bn4(x)
#         x = self.fc4(x)
#         x = torch.sigmoid(x)

#         return x

    # def predict(self, x):
    #     x = self.forward(x)
    #     x = F.softmax(x)
    #     return x

# net = FeedForwardNN(11, 4)
# print(net)

class FeedForwardNN(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(FeedForwardNN, self).__init__()

        self.bn1 = nn.BatchNorm1d(input_dim)
        self.fc1 = nn.Linear(input_dim, 8*hidden_dim)
        self.dp1 = nn.Dropout(0.25)

        self.bn2 = nn.BatchNorm1d(8*hidden_dim)
        self.fc2 = nn.Linear(8*hidden_dim, 4*hidden_dim)
        self.dp2 = nn.Dropout(0.15)

        self.bn3 = nn.BatchNorm1d(4*hidden_dim)
        self.fc3 = nn.Linear(4*hidden_dim, 2*hidden_dim)
        self.dp3 = nn.Dropout(0.15)

        self.bn4 = nn.BatchNorm1d(2*hidden_dim)
        self.fc4 = nn.Linear(2*hidden_dim, 1)

    def forward(self, x):
        x = self.bn1(x)
        x = self.fc1(x)
        x = torch.tanh(x)
        x = self.dp1(x)

        x = self.bn2(x)
        x = self.fc2(x)
        x = torch.tanh(x)
        x = self.dp2(x)

        x = self.bn3(x)
        x = self.fc3(x)
        x = torch.tanh(x)
        x = self.dp3(x)

        x = self.bn4(x)
        x = self.fc4(x)
        x = torch.sigmoid(x)

        return x
```

## 4. Учить будем на функцию потерь с кагла (log RMSE) - нужно её реализовать

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log y_i - \log \hat{y}_i)^2},$$

```
def log_rmse(y_pred, y_true):
    loss = torch.sqrt(torch.mean(
        (torch.log(y_pred+1)-torch.log(y_true+1))**2)
    )
    return loss


# def log_rmse(y_pred, y_true):
#       RMSLE = torch.sqrt(torch.mean(
#           (torch.log(y_pred)-torch.log(y_true))**2
#       ))

#       return RMSLE

# def log_rmse(y_true, y_pred):
```

```
#     d_i = (K.log(y_pred) -  K.log(y_true))
#     loss1 = K.mean(K.square(d_i))
#     loss2 = K.square(K.sum(K.flatten(d_i),axis=-1))/(K.cast_to_floatx(2) * K.square(K.cast_to_floatx(K.int_shape(K.flatten(d_i))[0]
#     loss = loss1 - loss2
#     return loss
```

# 5. Сравните сходимость (Оптимизаторы) 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

```
epochs = 5
lr = 0.01
for batch, label in train_loader:
    print(batch.shape)
    break
```

```
torch.Size([1024, 11])
```

```
def tr_loss(test_loader, model):
    batches = len(test_loader)
    loss = 0

    with torch.no_grad():
        for X, y in test_loader:
            y_pred = model(X)
            loss += log_rmse(y_pred, y)

    loss /= batches

    return loss


def train_proc(train_loader, test_loader, model, optimizer):

    for i, data in enumerate(train_loader):
        inputs, labels = data[0], data[1]

        y_pred = model(inputs)
        loss = log_rmse(y_pred, labels)

        # обнуляем градиент
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    return tr_loss(test_loader, model)
```

## ADAM

```
model = FeedForwardNN(11, 8)
print(model)
```

```
FeedForwardNN(
  (bn1): BatchNorm1d(11, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=11, out_features=64, bias=True)
  (dp1): Dropout(p=0.25, inplace=False)
  (bn2): BatchNorm1d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc2): Linear(in_features=64, out_features=32, bias=True)
  (dp2): Dropout(p=0.15, inplace=False)
  (bn3): BatchNorm1d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc3): Linear(in_features=32, out_features=16, bias=True)
  (dp3): Dropout(p=0.15, inplace=False)
  (bn4): BatchNorm1d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc4): Linear(in_features=16, out_features=1, bias=True)
)
```

```
optimizer = torch.optim.RMSprop(model.parameters(), lr=lr)
print(f'ADAM:\n')
for epoch in tqdm(range(epochs)):
    result = train_proc(train_loader,
```

```
                            test_loader,
                            model,
                            optimizer
                            )

    print(f'Test_loss={result}')
```

```
 ADAM:

  20%|██          | 1/5 [11:24<45:36, 684.04s/it]Test_loss=0.18010510504245758
  40%|████        | 2/5 [22:52<34:20, 686.85s/it]Test_loss=0.1796482801437378
  60%|██████      | 3/5 [34:31<23:04, 692.17s/it]Test_loss=0.1796303391456604
  80%|████████    | 4/5 [46:23<11:40, 700.17s/it]Test_loss=0.17977136373519897
 100%|██████████| 5/5 [58:27<00:00, 701.51s/it]Test_loss=0.18032890558242798
```

## RMSProp

```
model = FeedForwardNN(11, 8)
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
print(f'RMSprop:\n')

for epoch in tqdm(range(epochs)):
    result = train_proc(train_loader,
                            test_loader,
                            model,
                            optimizer
                            )

    print(f'Test_loss={result}')
```

```
 RMSprop:

  20%|██          | 1/5 [11:52<47:28, 712.18s/it]Test_loss=0.1798865646123886
  40%|████        | 2/5 [23:56<35:58, 719.39s/it]Test_loss=0.17945596575737
  60%|██████      | 3/5 [35:57<24:00, 720.25s/it]Test_loss=0.17960643768310547
  80%|████████    | 4/5 [47:46<11:55, 715.83s/it]Test_loss=0.17945215106010437
 100%|██████████| 5/5 [59:31<00:00, 714.30s/it]Test_loss=0.17936037480831146
```

## SGD

```
model = FeedForwardNN(11, 8)
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
print(f'SGD:\n')

for epoch in tqdm(range(epochs)):
    result = train_proc(train_loader,
                            test_loader,
                            model,
                            optimizer
                            )
    print(f'Test_loss: {result}')
```

```
 SGD:

  20%|██          | 1/5 [11:40<46:42, 700.52s/it]Test_loss: 0.19872649013996124
  40%|████        | 2/5 [23:29<35:16, 705.33s/it]Test_loss: 0.18643750250339508
  60%|██████      | 3/5 [35:04<23:21, 700.80s/it]Test_loss: 0.1849292516708374
  80%|████████    | 4/5 [46:41<11:39, 699.26s/it]Test_loss: 0.18434157967567444
 100%|██████████| 5/5 [58:12<00:00, 698.50s/it]Test_loss: 0.1837015599012375
```

## Вывод:

Наилучшую сходимость и лучший результат наблюдаем при использовании оптимизатора RMSProp

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

#         # обнуляем градиент
#         optimizer.zero_grad()
```

```
#         outputs = model(inputs)
#         loss = log_rmse(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!')
```

Root Mean Squared Error (RMSE) Submissions are scored on the root mean squared error. RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

where y hat is the predicted value and y is the original value.