В **этом ноутбуке** пробуется модель wavenet. Большинство кода взято отсюда https://www.kaggle.com/mobassir/understanding-ion-switching-with-modeling (https://www.kaggle.com/mobassir/understanding-ion-switching-with-modeling) - просто отчаянная попытка хоть как-то повысить скор за счет ноутбуков на каггле.

In [0]:
```
import numpy as np
import pandas as pd
import os
import torch
import torch.nn as nn
import time
import copy
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
from sklearn.metrics import f1_score
from sklearn.model_selection import KFold, GroupKFold
device = torch.device("cuda:0") if torch.cuda.is_available() else torch.devi
from torch.optim.lr_scheduler import ReduceLROnPlateau
import gc
import torchcontrib
from tqdm import tqdm
from torchcontrib.optim import SWA
import torchcontrib
```

1. Параметры

In [0]:
```
EPOCHS = 80 #150
NNBATCHSIZE = 16
GROUP_BATCH_SIZE = 4000
SEED = 321
LR = 0.001
SPLITS = 5

outdir = 'wavenet_models'
flip = False
noise = False


if not os.path.exists(outdir):
    os.makedirs(outdir)



def seed_everything(seed):
    random.seed(seed)
    np.random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    tf.random.set_seed(seed)
```

2. Данные

In [5]:

```
1 from google.colab import drive
2 drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:

```
def read_data():
    train = pd.read_csv('/content/gdrive/My Drive/data/train_clean_kalman.cs
    test  = pd.read_csv('/content/gdrive/My Drive/data/test_clean_kalman.csv
    sub  = pd.read_csv('/content/gdrive/My Drive/data/sample_submission.csv'
    return train, test, sub

def batching(df, batch_size):
    #print(df)
    df['group'] = df.groupby(df.index//batch_size, sort=False)['signal'].agg
    df['group'] = df['group'].astype(np.uint16)
    return df

# normalize the data (standard scaler). We can also try other scalers for a
def normalize(train, test):
    train_input_mean = train.signal.mean()
    train_input_sigma = train.signal.std()
    train['signal'] = (train.signal - train_input_mean) / train_input_sigma
    test['signal'] = (test.signal - train_input_mean) / train_input_sigma
    return train, test

# get lead and lags features
def lag_with_pct_change(df, windows):
    for window in windows:
        df['signal_shift_pos_' + str(window)] = df.groupby('group')['signal'
        df['signal_shift_neg_' + str(window)] = df.groupby('group')['signal'
    return df

# main module to run feature engineering. Here you may want to try and add o
def run_feat_engineering(df, batch_size):
    # create batches
    df = batching(df, batch_size = batch_size)
    # create leads and lags (1, 2, 3 making them 6 features)
    df = lag_with_pct_change(df, [1, 2, 3])
    # create signal ** 2 (this is the new feature)
    df['signal_2'] = df['signal'] ** 2
    return df

# fillna with the mean and select features for training
def feature_selection(train, test):
    features = [col for col in train.columns if col not in ['index', 'group'
    train = train.replace([np.inf, -np.inf], np.nan)
    test = test.replace([np.inf, -np.inf], np.nan)
    for feature in features:
        feature_mean = pd.concat([train[feature], test[feature]], axis = 0).
        train[feature] = train[feature].fillna(feature_mean)
        test[feature] = test[feature].fillna(feature_mean)
    return train, test, features


def split(GROUP_BATCH_SIZE=4000, SPLITS=5):
    print('Reading Data Started...')
    train, test, sample_submission = read_data()
    train, test = normalize(train, test)
    print('Reading and Normalizing Data Completed')
    print('Creating Features')
    print('Feature Engineering Started...')
```

```
    train = run_feat_engineering(train, batch_size=GROUP_BATCH_SIZE)
    test = run_feat_engineering(test, batch_size=GROUP_BATCH_SIZE)
    train, test, features = feature_selection(train, test)
    print(train.head())
    print('Feature Engineering Completed...')

    target = ['open_channels']
    group = train['group']
    kf = GroupKFold(n_splits=SPLITS)
    splits = [x for x in kf.split(train, train[target], group)]
    new_splits = []
    for sp in splits:
        new_split = []
        new_split.append(np.unique(group[sp[0]]))
        new_split.append(np.unique(group[sp[1]]))
        new_split.append(sp[1])
        new_splits.append(new_split)
    target_cols = ['open_channels']
    print(train.head(), train.shape)
    train_tr = np.array(list(train.groupby('group').apply(lambda x: x[target
    train = np.array(list(train.groupby('group').apply(lambda x: x[features]
    test = np.array(list(test.groupby('group').apply(lambda x: x[features].v
    print(train.shape, test.shape, train_tr.shape)
    return train, test, train_tr, new_splits
```

3. Модель

In [0]:

```
class Wave_Block(nn.Module):

    def __init__(self,in_channels,out_channels,dilation_rates):
        super(Wave_Block,self).__init__()
        self.num_rates = dilation_rates
        self.convs = nn.ModuleList()
        self.filter_convs = nn.ModuleList()
        self.gate_convs = nn.ModuleList()

        self.convs.append(nn.Conv1d(in_channels,out_channels,kernel_size=1))
        dilation_rates = [2**i for i in range(dilation_rates)]
        for dilation_rate in dilation_rates:
            self.filter_convs.append(nn.Conv1d(out_channels,out_channels,ker
            self.gate_convs.append(nn.Conv1d(out_channels,out_channels,kerne
            self.convs.append(nn.Conv1d(out_channels,out_channels,kernel_siz

    def forward(self,x):
        x = self.convs[0](x)
        res = x
        for i in range(self.num_rates):
            x = F.tanh(self.filter_convs[i](x))*F.sigmoid(self.gate_convs[i]
            x = self.convs[i+1](x)
            x += res
        return x




class Classifier(nn.Module):
    def __init__(self):
        super().__init__()
        input_size = 128

        self.LSTM = nn.GRU(input_size=input_size,hidden_size=64,num_layers=2

        self.wave_block1 = Wave_Block(8,16,12)
        self.wave_block2 = Wave_Block(16,32,8)
        self.wave_block3 = Wave_Block(32,64,4)
        self.wave_block4 = Wave_Block(64, 128, 1)
        self.fc = nn.Linear(128, 11)

    def forward(self,x):
        x = x.permute(0, 2, 1)

        x = self.wave_block1(x)
        x = self.wave_block2(x)
        x = self.wave_block3(x)

        #x,_ = self.LSTM(x)
        x = self.wave_block4(x)
        x = x.permute(0, 2, 1)
        x,_ = self.LSTM(x)
        #x = self.conv1(x)
        #x = self.attention(x)
        x = self.fc(x)
        return x
```

```
57
58
59
60  class EarlyStopping:
61      def __init__(self, patience=5, delta=0, checkpoint_path='checkpoint.pt',
62          self.patience, self.delta, self.checkpoint_path = patience, delta, c
63          self.counter, self.best_score = 0, None
64          self.is_maximize = is_maximize
65
66
67      def load_best_weights(self, model):
68          model.load_state_dict(torch.load(self.checkpoint_path))
69
70      def __call__(self, score, model):
71          if self.best_score is None or \
72                  (score > self.best_score + self.delta if self.is_maximize el
73              torch.save(model.state_dict(), self.checkpoint_path)
74              self.best_score, self.counter = score, 0
75              return 1
76          else:
77              self.counter += 1
78              if self.counter >= self.patience:
79                  return 2
80          return 0
```

4. Даталоадеры

In [0]:

```
1   from torch.utils.data import Dataset, DataLoader
2   class IronDataset(Dataset):
3       def __init__(self, data, labels, training=True, transform=None, seq_len=
4           self.data = data
5           self.labels = labels
6           self.transform = transform
7           self.training = training
8           self.flip = flip
9           self.noise_level = noise_level
10          self.class_split = class_split
11          self.seq_len = seq_len
12
13      def __len__(self):
14          return len(self.data)
15
16      def __getitem__(self, idx):
17          if torch.is_tensor(idx):
18              idx = idx.tolist()
19
20          data = self.data[idx]
21          labels = self.labels[idx]
22
23          return [data.astype(np.float32), labels.astype(int)]
```

In [9]:
```
1 train, test, train_tr, new_splits = split()
```

```
Reading Data Started...
Reading and Normalizing Data Completed
Creating Features
Feature Engineering Started...
     time    signal  ...  signal_shift_neg_3  signal_2
0  0.0001 -1.148772  ...           -1.298012  1.319677
1  0.0002 -1.184075  ...           -1.303999  1.402034
2  0.0003 -1.012891  ...           -1.104036  1.025949
3  0.0004 -1.298012  ...           -1.123085  1.684836
4  0.0005 -1.303999  ...           -1.082287  1.700413

[5 rows x 11 columns]
Feature Engineering Completed...
     time    signal  ...  signal_shift_neg_3  signal_2
0  0.0001 -1.148772  ...           -1.298012  1.319677
1  0.0002 -1.184075  ...           -1.303999  1.402034
2  0.0003 -1.012891  ...           -1.104036  1.025949
3  0.0004 -1.298012  ...           -1.123085  1.684836
4  0.0005 -1.303999  ...           -1.082287  1.700413

[5 rows x 11 columns] (5000000, 11)
(1250, 4000, 8) (500, 4000, 8) (1250, 4000, 1)
```

5. Обучение

In [10]:
```
1 model = Classifier()
2 model
```

Out[10]:
```
Classifier(
  (LSTM): GRU(128, 64, num_layers=2, batch_first=True, bidirectional=True)
  (wave_block1): Wave_Block(
    (convs): ModuleList(
      (0): Conv1d(8, 16, kernel_size=(1,), stride=(1,))
      (1): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (2): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (3): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (4): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (5): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (6): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (7): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (8): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (9): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (10): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (11): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
      (12): Conv1d(16, 16, kernel_size=(1,), stride=(1,))
    )
    (filter_convs): ModuleList(
```

In [11]:

```
%%time
test_y = np.zeros([int(2000000/GROUP_BATCH_SIZE), GROUP_BATCH_SIZE, 1])
test_dataset = IronDataset(test, test_y, flip=False)
test_dataloader = DataLoader(test_dataset, NNBATCHSIZE, shuffle=False, num_v
test_preds_all = np.zeros((2000000, 11))


oof_score = []
for index, (train_index, val_index, _) in enumerate(new_splits[0:], start=0)
    print("Fold : {}".format(index))
    train_dataset = IronDataset(train[train_index], train_tr[train_index], s
    train_dataloader = DataLoader(train_dataset, NNBATCHSIZE, shuffle=True,

    valid_dataset = IronDataset(train[val_index], train_tr[val_index], seq_l
    valid_dataloader = DataLoader(valid_dataset, NNBATCHSIZE, shuffle=False,

    it = 0
    model = Classifier()
    model = model.cuda()

    early_stopping = EarlyStopping(patience=40, is_maximize=True,
                                   checkpoint_path=os.path.join(outdir, "gru


    weight = None#cal_weights()
    criterion = nn.CrossEntropyLoss(weight=weight)
    optimizer = torch.optim.Adam(model.parameters(), lr=LR)

    optimizer = torchcontrib.optim.SWA(optimizer, swa_start=10, swa_freq=5,


    schedular = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='
    avg_train_losses, avg_valid_losses = [], []



    for epoch in range(EPOCHS):
        print('**********************************')
        print("Folder : {} Epoch : {}".format(index, epoch))
        print("Curr learning_rate: {:0.9f}".format(optimizer.param_groups[0]
        train_losses, valid_losses = [], []
        tr_loss_cls_item, val_loss_cls_item = [], []

        model.train()  # prep model for training
        train_preds, train_true = torch.Tensor([]).cuda(), torch.LongTensor(

        for x, y in tqdm(train_dataloader):
            x = x.cuda()
            y = y.cuda()
            #print(x.shape)


            optimizer.zero_grad()
            #loss_fn(model(input), target).backward()
```

```
57
58              #optimizer.zero_grad()
59              predictions = model(x)
60
61              predictions_ = predictions.view(-1, predictions.shape[-1])
62              y_ = y.view(-1)
63
64              loss = criterion(predictions_, y_)
65
66              # backward pass: compute gradient of the loss with respect to mo
67              loss.backward()
68              # perform a single optimization step (parameter update)
69              optimizer.step()
70
71              #schedular.step()
72              # record training lossa
73              train_losses.append(loss.item())
74              train_true = torch.cat([train_true, y_], 0)
75              train_preds = torch.cat([train_preds, predictions_], 0)
76
77          #model.eval()  # prep model for evaluation
78          optimizer.swap_swa_sgd()
79          val_preds, val_true = torch.Tensor([]).cuda(), torch.LongTensor([]).
80          print('EVALUATION')
81          with torch.no_grad():
82              for x, y in tqdm(valid_dataloader):
83                  x = x.cuda()#.to(device)
84                  y = y.cuda()#..to(device)
85
86                  predictions = model(x)
87                  predictions_ = predictions.view(-1, predictions.shape[-1])
88                  y_ = y.view(-1)
89
90                  loss = criterion(predictions_, y_)
91
92                  valid_losses.append(loss.item())
93
94
95                  val_true = torch.cat([val_true, y_], 0)
96                  val_preds = torch.cat([val_preds, predictions_], 0)
97
98          # calculate average loss over an epoch
99          train_loss = np.average(train_losses)
100         valid_loss = np.average(valid_losses)
101         avg_train_losses.append(train_loss)
102         avg_valid_losses.append(valid_loss)
103         print("train_loss: {:0.6f}, valid_loss: {:0.6f}".format(train_loss,
104
105         train_score = f1_score(train_true.cpu().detach().numpy(), train_pred
106                                labels=list(range(11)), average='macro')
107
108         val_score = f1_score(val_true.cpu().detach().numpy(), val_preds.cpu(
109                              labels=list(range(11)), average='macro')
110
111         schedular.step(val_score)
112         print("train_f1: {:0.6f}, valid_f1: {:0.6f}".format(train_score, val
113         res = early_stopping(val_score, model)
```

```
        #print('fres:', res)
        if  res == 2:
            print("Early Stopping")
            print('folder %d global best val max f1 model score %f' % (index
            break
        elif res == 1:
            print('save folder %d global val max f1 model score %f' % (index
    print('Folder {} finally best global max f1 score is {}'.format(index, e
    oof_score.append(round(early_stopping.best_score, 6))

    model.eval()
    pred_list = []
    with torch.no_grad():
        for x, y in tqdm(test_dataloader):

            x = x.cuda()
            y = y.cuda()

            predictions = model(x)
            predictions_ = predictions.view(-1, predictions.shape[-1]) # sha
            #print(predictions.shape, F.softmax(predictions_, dim=1).cpu().r
            pred_list.append(F.softmax(predictions_, dim=1).cpu().numpy()) #
            #a = input()
        test_preds = np.vstack(pred_list) # shape [2000000, 11]
        test_preds_all += test_preds
```

```
100%|██████████| 63/63 [00:37<00:00,  1.67it/s]
  0%|          | 0/16 [00:00<?, ?it/s]

EVALUATION

100%|██████████| 16/16 [00:04<00:00,  3.97it/s]

train_loss: 0.079804, valid_loss: 0.087705

  0%|          | 0/32 [00:00<?, ?it/s]

train_f1: 0.938050, valid_f1: 0.937883
Folder 4 finally best global max f1 score is 0.9384072723611167

100%|██████████| 32/32 [00:08<00:00,  3.92it/s]

CPU times: user 4h 9min 29s, sys: 30min 1s, total: 4h 39min 30s
Wall time: 4h 41min 49s
```

In [12]:

```
print('all folder score is:%s'%str(oof_score))
print('OOF mean score is: %f'% (sum(oof_score)/len(oof_score)))
print('Generate submission.............')
submission_csv_path = '/content/gdrive/My Drive/data/sample_submission.csv'
ss = pd.read_csv(submission_csv_path, dtype={'time': str})
test_preds_all = test_preds_all / np.sum(test_preds_all, axis=1)[:, None]
test_pred_frame = pd.DataFrame({'time': ss['time'].astype(str),
                                'open_channels': np.argmax(test_preds_all, a
test_pred_frame.to_csv("/content/gdrive/My Drive/data/gru_submission.csv", i
print('over')
```

```
all folder score is:[0.938465, 0.939936, 0.93835, 0.93827, 0.938407]
OOF mean score is: 0.938686
Generate submission.............
over
```

Результат: 0.941 на public lb