In [30]:

```
# попробуем запрограммировать простую рекурентную сеть

import pandas as pd
from string import punctuation
from stop_words import get_stop_words
from pymorphy2 import MorphAnalyzer
import re

import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F

from collections import Counter

from tqdm import tqdm_notebook
```

In [31]:

```
df_train = pd.read_csv("data/train.csv")
df_test = pd.read_csv("data/test.csv")
df_val = pd.read_csv("data/val.csv")
```

In [32]:

```
df_train.shape
```

Out[32]:

```
(181467, 3)
```

In [33]:

```
df_train.head()
```

Out[33]:

	id	text	class
0	0	@alisachachka не уезжаааааааай. :(❤ я тоже не ...	0
1	1	RT @GalyginVadim: Ребята и девчата!\nВсе в кин...	1
2	2	RT @ARTEM_KLYUSHIN: Кто ненавидит пробки ретви...	0
3	3	RT @epupybobv: Хочется котлету по-киевски. Зап...	1
4	4	@KarineKurganova @Yess__Boss босапопа есбоса н...	1

In [69]:

```
en_letters = ['q', 'w','e','r','t','y','u','i','o','p','a'
          ,'s','d','f','g','h','j','k','l','z','x','c','v','b','n','m','1','2','3','4','5',
ru = 'ёйцукенгшщзхъфывапролджэячсмитьбю'
ru_letters = set(ru)
print(len(ru_letters))
print(en_letters)
print('re' not in en_letters)
```

```
33
['q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', 'a', 's', 'd', 'f', 'g',
'h', 'j', 'k', 'l', 'z', 'x', 'c', 'v', 'b', 'n', 'm', '1', '2', '3', '4',
'5', '6', '7', '8', '9', '0', '♥']
True
```

In [35]:

```
df_train.head()
```

Out[35]:

	id	text	class
0	0	@alisachachka не уезжаааааааай. :(♥ я тоже не ...	0
1	1	RT @GalyginVadim: Ребята и девчата!\nВсе в кин...	1
2	2	RT @ARTEM_KLYUSHIN: Кто ненавидит пробки ретви...	0
3	3	RT @epupybobv: Хочется котлету по-киевски. Зап...	1
4	4	@KarineKurganova @Yess__Boss босапопа есбоса н...	1

In [43]:

```
# punctuation.append('♥')
exclude = set(punctuation)
# exclude.append('♥')
print(exclude)
```

```
{'\\', '%', '&', '-', '+', '[', '^', '!', "'", '}', ',', '/', ';', '~', '<',
'(', '?', '#', '"', '*', '>', ':', '|', '`', '_', ')', '$', '{', ']', '@',
'.', '='}
```

очищаю текст от слов несущих малое значение, от английских символов т.к. это ник неймы не несущие никакого смысла, 'не' привязываю к следующему после частицы идущего слова, чтоб не потерять смысл предложения, привожу слова к нормальной форме при помощи библиотеки pymorphy2

In [58]:

```
sw = set(get_stop_words("ru"))
exclude = set(punctuation)
morpher = MorphAnalyzer()

def preprocess_text(txt):
    txt = str(txt)
    txt = "".join(c for c in txt if c not in exclude)
    txt = txt.lower()
    txt = "".join(c for c in txt if c not in letter)
    txt = re.sub("\sне", "не", txt)
    txt = [morpher.parse(word)[0].normal_form for word in txt.split() if word not in sw]
    return " ".join(txt)

df_train['text'] = df_train['text'].apply(preprocess_text)
df_val['text'] = df_val['text'].apply(preprocess_text)
df_test['text'] = df_test['text'].apply(preprocess_text)
```

In [59]:

```
df_train.head(30)
```

Out[59]:

	id	text	class
0	0	уезжаааааааать тожена уезжать	0
1	1	ребята девчата кино любовь завтра вотэтолюбовь	1
2	2	ктоненавидеть пробка ретвит	0
3	3	хотеться котлета покиевск запретный плод	1
4	4	босапоп есбосан бояться мороз	1
5	5	манчестер час играть ян дом	0
6	6	жаба пришол остаться транспорт правда девчёнка...	1
7	7	момент мальчик маньяк ражик гладить рука феликс	1
8	8	простонеудачик поцарапать экран телефон	0
9	9	хахаа запомниться надолго	1
10	10	красава тебен сомневаться	1
11	11	мартовский путёвка дорожать глаз пара оо	0
12	12	прошлый месяц вообщить	0
13	13	привет самый хороший друг извращенобольный фан...	1
14	14	придумать новый	1
15	15	увеличение лимит бесплатный пополнение сделать...	1
16	16	тожить лах завтрана идти	1
17	17	голован прекращать болеть пойти температурка п...	1
18	18	крайне культурный друг разговорнезнакомый кузо...	1
19	19	работать отмена встряхивание	0
20	20	австрия ждееeть последний формальность уладить...	1
21	21	одноклассник наня психанулон плакать потомнить...	1
22	22	парнит понять	1
23	23	оуовать рада ещеть продажа	0
24	24	прогноз погода ит близкий	1
25	25	пара сотрудничество положительный эмоция компа...	1
26	26	красивый ян	1
27	27	пособие применение рука	1
28	28	начальница рассказывать свадьба минута чтоть в...	1
29	29	гггггга момент милаявсё тайлера оч кластный ма...	1

In [43]:

```
# vocab2index = {"":0, "UNK":1}
# words = ["", "UNK"]
```

In [67]:

```
# a = ['red', 'blue', 'red', 'green', 'blue', 'blue']
# counts = Counter()
# counts.update(a)
# # print(len(counts.keys()))
# # print(list(counts.keys())[1])
# # print(counts['red'])
# word = a[0]
# vocab2index[word] = len(words)
# words.append(word)
# print(vocab2index)
# print(words)
# # Counter(a)
```

```
{'': 0, 'UNK': 1, 'red': 24, 'blue': 12, 'green': 23}
['', 'UNK', 'red', 'blue', 'red', 'green', 'blue', 'blue', 'blue', 'blue',
'blue', 'blue', 'blue', 'green', 'green', 'green', 'green', 'green', 'gree
n', 'green', 'green', 'green', 'green', 'green', 'red']
```

In [46]:

```
# len(vocab2index)
```

In [47]:

```
text_corpus_train = df_train['text'].values
text_corpus_valid = df_val['text'].values
text_corpus_test = df_test['text'].values

counts = Counter()
for sequence in text_corpus_train:
    counts.update(sequence.split())

print("num_words before:",len(counts.keys()))
for word in list(counts):
    if counts[word] < 2:
        del counts[word]
print("num_words after:",len(counts.keys()))

vocab2index = {"":0, "UNK":1}
words = ["", "UNK"]
for word in counts:
    vocab2index[word] = len(words)
    words.append(word)
```

```
num_words before: 155836
num_words after: 45689
```

In [55]:

```
```

In [48]:

```
from functools import lru_cache

class TwitterDataset(torch.utils.data.Dataset):

    def __init__(self, txts, labels, w2index, used_length):
        self._txts = txts
        self._labels = labels
        self._length = used_length
        self._w2index = w2index

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

    @lru_cache(50000)
    def encode_sentence(self, txt):
        encoded = np.zeros(self._length, dtype=int)
        enc1 = np.array([self._w2index.get(word, self._w2index["UNK"]) for word in txt.spli
        length = min(self._length, len(enc1))
        encoded[:length] = enc1[:length]
        return encoded, length

    def __getitem__(self, index):
        encoded, length = self.encode_sentence(self._txts[index])
        return torch.from_numpy(encoded.astype(np.int32)), self._labels[index], length
```

In [49]:

```
max([len(i.split()) for i in text_corpus_train])
```

Out[49]:

```
27
```

In [50]:

```
y_train = df_train['class'].values
y_val = df_val['class'].values

train_dataset = TwitterDataset(text_corpus_train, y_train, vocab2index, 27)
valid_dataset = TwitterDataset(text_corpus_valid, y_val, vocab2index, 27)

train_loader = torch.utils.data.DataLoader(train_dataset,
                           batch_size=128,
                           shuffle=True,
                           num_workers=0)
valid_loader = torch.utils.data.DataLoader(valid_dataset,
                           batch_size=128,
                           shuffle=False,
                           num_workers=0)
```

In [51]:

```
class RNNFixedLen(nn.Module) :
    def __init__(self, vocab_size, embedding_dim, hidden_dim) :
        super().__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.rnn = nn.RNN(embedding_dim, hidden_dim, num_layers=2, batch_first=True)
        self.linear = nn.Linear(hidden_dim, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x, l):
        x = self.embeddings(x)
        x = self.dropout(x)
        rnn_out, (ht, ct) = self.rnn(x)
        return self.linear(rnn_out)
```

In [52]:

```
rnn_init = RNNFixedLen(len(vocab2index), 30, 20)
optimizer = torch.optim.Adam(rnn_init.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()
```

RNN

In [60]:

```
epochs = 10
```

In [54]:

```
for epoch in tqdm_notebook(range(epochs)):
    rnn_init.train()
    for i, data in enumerate(train_loader,0):
        inputs, labels, lengths = data[0], data[1], data[2]
        inputs = inputs.long()
        labels = labels.long().view(-1, 1)

        optimizer.zero_grad()

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

    rnn_init.eval()
    loss_accumed = 0
    for X, y, lengths in valid_loader:
        X = X.long()
        y = y.long().view(-1, 1)
        output = rnn_init(X, lengths)
        loss = criterion(output, y)
        loss_accumed += loss
    print("Epoch {} valid_loss {}".format(epoch, loss_accumed))

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

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: TqdmDep
recationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """Entry point for launching an IPython kernel.
```

100% 10/10 [04:57<00:00, 29.69s/it]

```
Epoch 0 valid_loss 114.20539093017578
Epoch 1 valid_loss 113.68492889404297
Epoch 2 valid_loss 113.16081237792969
Epoch 3 valid_loss 115.48741912841797
Epoch 4 valid_loss 117.35490417480469
Epoch 5 valid_loss 117.39289855957031
Epoch 6 valid_loss 116.42377471923828
Epoch 7 valid_loss 120.84660339355469
Epoch 8 valid_loss 121.12635803222656
Epoch 9 valid_loss 122.92884063720703
Training is finished!
```

LSTM

In [61]:

```
class LSTMFixedLen(nn.Module) :
    def __init__(self, vocab_size, embedding_dim, hidden_dim) :
        super().__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, num_layers=2, batch_first=True)
        self.linear = nn.Linear(hidden_dim, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x, l):
        x = self.embeddings(x)
        x = self.dropout(x)
        lstm_out, (ht, ct) = self.lstm(x)
        return self.linear(lstm_out)

lstm_init = LSTMFixedLen(len(vocab2index), 30, 20)
optimizer = torch.optim.Adam(lstm_init.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()
```

In [62]:

```
for epoch in tqdm_notebook(range(epochs)):
    lstm_init.train()
    for i, data in enumerate(train_loader, 0):
        inputs, labels, lengths = data[0], data[1], data[2]
        inputs = inputs.long()
        labels = labels.long().view(-1, 1)

        optimizer.zero_grad()

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

    lstm_init.eval()
    loss_accumed = 0
    for X, y, lengths in valid_loader:
        X = X.long()
        y = y.long().view(-1, 1)
        output = lstm_init(X, lengths)
        loss = criterion(output, y)
        loss_accumed += loss
    print("Epoch {} valid_loss {}".format(epoch, loss_accumed))

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

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: TqdmDep
recationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """Entry point for launching an IPython kernel.
```

100% 10/10 [09:01<00:00, 54.01s/it]

```
Epoch 0 valid_loss 113.22369384765625
Epoch 1 valid_loss 112.22395324707031
Epoch 2 valid_loss 113.3219223022461
Epoch 3 valid_loss 114.02628326416016
Epoch 4 valid_loss 117.17424011230469
Epoch 5 valid_loss 118.502197265625
Epoch 6 valid_loss 119.434814453125
Epoch 7 valid_loss 124.0313720703125
Epoch 8 valid_loss 123.37696838378906
Epoch 9 valid_loss 124.91738891601562
Training is finished!
```

GRU

In [74]:

```
class GRUFixedLen(nn.Module) :
    def __init__(self, vocab_size, embedding_dim, hidden_dim) :
        super().__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)
        self.gru = nn.GRU(embedding_dim, hidden_dim, num_layers=2, batch_first=True)
        self.linear = nn.Linear(hidden_dim, 1)
        self.dropout = nn.Dropout(0.2)

    def forward(self, x, l):
        x = self.embeddings(x)
        x = self.dropout(x)
        gru_out, (ht, ct) = self.gru(x)
        return self.linear(gru_out)

gru_init = GRUFixedLen(len(vocab2index), 30, 20)
optimizer = torch.optim.Adam(gru_init.parameters(), lr=0.01)
criterion = nn.CrossEntropyLoss()
```

In [75]:

```
for epoch in tqdm_notebook(range(epochs)):
    gru_init.train()
    for i, data in enumerate(train_loader, 0):
        inputs, labels, lengths = data[0], data[1], data[2]
        inputs = inputs.long()
        labels = labels.long().view(-1, 1)

        optimizer.zero_grad()

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

    gru_init.eval()
    loss_accumed = 0
    for X, y, lengths in valid_loader:
        X = X.long()
        y = y.long().view(-1, 1)
        output = gru_init(X, lengths)
        loss = criterion(output, y)
        loss_accumed += loss
    print("Epoch {} valid_loss {}".format(epoch, loss_accumed))

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

```
c:\program files\python37\lib\site-packages\ipykernel_launcher.py:1: TqdmDep
recationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
  """Entry point for launching an IPython kernel.
```

100% 10/10 [08:38<00:00, 51.28s/it]

```
Epoch 0 valid_loss 113.19703674316406
Epoch 1 valid_loss 113.18024444580078
Epoch 2 valid_loss 114.15181732177734
Epoch 3 valid_loss 115.62777709960938
Epoch 4 valid_loss 117.05768585205078
Epoch 5 valid_loss 120.30059051513672
Epoch 6 valid_loss 119.49751281738281
Epoch 7 valid_loss 120.0285415649414
Epoch 8 valid_loss 122.36158752441406
Epoch 9 valid_loss 123.08397674560547
Training is finished!
```