In [1]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
from sklearn.decomposition import PCA
import torch.utils.data
import pickle
from tqdm import tqdm
import time
import gc
import collections
```

In [2]:
```
'''
set parameters
'''
base_path = '.'
train_path = base_path + '/train.npy'
test_path = base_path + '/test.npy'
train_labels_path = base_path + '/train_labels.npy'
dev_labels_path = base_path +  '/dev_labels.npy'
dev_path = base_path + '/dev.npy'

pca_available = True
padding_method = 'self'
# padding_method = 'zero'

device = torch.device('cuda:0')
n_labels = 138
n_features = 40
n_epoch = 10
context_num = 18
```

In [3]:
```
def load_train_data():
    t0 = time.time()
    print("Start loading training data...")
    train = np.load(train_path, allow_pickle=True)
    train_labels = np.load(train_labels_path, allow_pickle=True)
    t1 = time.time()
    elapsed_time = t1 - t0
    print("Done loading training data in {0} minutes...".format(elapsed_time/60)

    return train, train_labels
```

In [4]:
```
def load_validation_data():
    t0 = time.time()
    print("Start loading validation data...")
    val = np.load(dev_path, allow_pickle=True)
    val_labels = np.load(dev_labels_path, allow_pickle=True)
    t1 = time.time()
    elapsed_time = t1 - t0
    print("Done loading validation data in {0} minutes...".format(elapsed_time/6

    return val, val_labels
```

In [5]:
```
def load_test_data():
    t0 = time.time()
    print("Start loading test data...")
    test = np.load(test_path, allow_pickle=True)
    t1 = time.time()
    elapsed_time = t1 - t0
    print("Done loading test data in {0} minutes...".format(elapsed_time/60))
    return test
```

In [6]:
```
def load_and_process_data(features, labels, pca, context_num):
    '''
    use the first&last frame of one utterance to pad the empty frame
    '''
    t0 = time.time()
    padding_features = np.concatenate([np.concatenate(( \
                                            np.ones((context_num, pca.n_com
                                            pca.transform(features[i]), \
                                            np.ones((context_num, pca.n_com
                                            for i in range(len(features))])
    padding_features = torch.Tensor(padding_features)
    del features

    '''
    corresponding label for padding frames
    '''
    false_labels = np.array([-1]*context_num)
    padding_labels = np.concatenate([np.concatenate(( \
                                            false_labels, \
                                            labels[i], \
                                            false_labels)) \
                                            for i in range(len(labels))])
    padding_labels = torch.Tensor(padding_labels)
    del labels

    gc.collect()

    return padding_features, padding_labels
```

In [7]:
```
class ContextDataset(Dataset):

    def __init__(self, context_num, features, targets):

        self.context_num = context_num
        self.features = features
        self.targets = targets

    def __len__(self):

        return len(self.targets)

    def __getitem__(self, index):

        if index-self.context_num >= 0 and index+self.context_num+1 <= len(self.t
            '''
            no need for padding
            '''
            X = self.features[index-self.context_num: index+self.context_num+1].
            Y = self.targets[index].long()
        elif index-self.context_num < 0:
            '''
            padding for pre frames, actually doesnt matter since we drop this 'fa
            '''
            X = torch.cat((torch.zeros(self.context_num-index, self.features.shap
            Y = self.targets[index].long()
        else:
            '''
            padding for post frames, same as before
            '''
            X = torch.cat((self.features[index-self.context_num:], torch.zeros(in
            Y = self.targets[index].long()

        return index, X, Y
```

In [8]:
```
#1000,2048,1024,512,256+2,138
class SpeechNet(nn.Module):

    def __init__(self, context_num):

        super(SpeechNet, self).__init__()
        self.relu1 = nn.ReLU()
        self.relu2 = nn.ReLU()
        self.relu3 = nn.ReLU()
        self.relu4 = nn.ReLU()
        self.relu5 = nn.ReLU()
        self.relu6 = nn.ReLU()
        self.relu7 = nn.ReLU()
        self.relu8 = nn.ReLU()

        self.linear1 = nn.Linear((2*context_num+1)*pca.n_components, 2048)
        self.linear2 = nn.Linear(2048, 1024)
        self.linear3 = nn.Linear(1024, 810)
        self.linear4 = nn.Linear(810, 720)
        self.linear5 = nn.Linear(720, 512)
        self.linear6 = nn.Linear(512, 428)
        self.linear7 = nn.Linear(428, 300)
        self.linear8 = nn.Linear(300, 256)
        self.out = nn.Linear(256+2, 138)

        self.batchnorm1 = nn.BatchNorm1d(2048)
        self.batchnorm2 = nn.BatchNorm1d(1024)
        self.batchnorm3 = nn.BatchNorm1d(810)
        self.batchnorm4 = nn.BatchNorm1d(720)
        self.batchnorm5 = nn.BatchNorm1d(512)
        self.batchnorm6 = nn.BatchNorm1d(428)
        self.batchnorm7 = nn.BatchNorm1d(300)
        self.batchnorm8 = nn.BatchNorm1d(256+2)


        self.dropout1 = nn.Dropout(0.1)
        self.dropout2 = nn.Dropout(0.05)
        self.dropout3 = nn.Dropout(0.1)
        self.dropout4 = nn.Dropout(0.05)
        self.dropout5 = nn.Dropout(0.05)
        self.dropout6 = nn.Dropout(0.05)
        self.dropout7 = nn.Dropout(0.05)

    def forward(self, x):

        x = self.linear1(x)
        x = self.batchnorm1(x)
        x = self.relu1(x)
        #2048
        x = self.dropout1(x)
        x = self.linear2(x)
        x = self.batchnorm2(x)
        x = self.relu2(x)
        #1024
        x = self.dropout2(x)
        x = self.linear3(x)
```

```
        x = self.batchnorm3(x)
        x = self.relu3(x)
        #810
        x = self.dropout3(x)
        x = self.linear4(x)
        x = self.batchnorm4(x)
        x = self.relu4(x)

        #512
        x = self.dropout4(x)
        x = self.linear5(x)
        x = self.batchnorm5(x)
        x = self.relu5(x)

        #512
        x = self.dropout5(x)
        x = self.linear6(x)
        x = self.batchnorm6(x)
        x = self.relu6(x)
        #428
        x = self.dropout6(x)
        x = self.linear7(x)
        x = self.batchnorm7(x)
        x = self.relu7(x)
        #300
        x = self.dropout7(x)
        x = self.linear8(x)
        #300

        avg_pool1 = torch.mean(x, 1, keepdims = True)
        max_pool1,_ = torch.max(x, 1, keepdims = True)

        conc = torch.cat((x, avg_pool1, max_pool1), 1)
        conc = self.batchnorm8(conc)
        output = self.out(conc)

        return output
```

In [9]:
```
def generate_dataset(context_num, features, labels):
    t0 = time.time()
    print("It may takes 20 minutes to generate train dataset...")
    context_dataset = ContextDataset(context_num, features, labels)
    dataset = SpeechDataset(context_dataset)
    t1 = time.time()
    print("Dataset generated. Elapsed time: {0}".format((t1-t0)/60))
    return dataset
```

In [10]:
```
def weights_init(m):
    if isinstance(m, nn.Conv2d):
        xavier(m.weight.data)
        xavier(m.bias.data)
```

In [11]:
```
def scale_cos(x):
    start = 5e-3
    end = 1e-5
    return start + (1 + np.cos(np.pi * (1 - x))) * (end - start) / 2
```

In [12]:
```
def second_scale_cos(x):
    start = 1e-4
    end = 1e-8
    return start + (1 + np.cos(np.pi * (1 - x))) * (end - start) / 2
```

In [13]:
```
class ParamScheduler:

    def __init__(self, optimizer, scale_fn, total_steps):

        self.optimizer = optimizer
        self.scale_fn = scale_fn
        self.total_steps = total_steps
        self.current_iteration = 0

    def batch_step(self):
        for param_group in self.optimizer.param_groups:
            param_group['lr'] = self.scale_fn(self.current_iteration/self.total_

        self.current_iteration += 1
```

In [14]:
```
def train_model(train_dataloader, val_dataloader, n_epochs = 10):

    model = SpeechNet(context_num).to(device)
    model.apply(weights_init)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)

    '''
    set scheduler for decaying learning rate
    '''
    parameter_scheduler = ParamScheduler(optimizer, scale_cos, n_epoch*len(train_
    candidate_model = 1
    print('Start training...')
    for i in range(n_epochs):

        t0 = time.time()

        avg_loss_1000_batch = 0
        val_correct = 0
        val_predicted = 0
        model.train()

        for index, (idx, features, labels) in enumerate(train_dataloader):

            optimizer.zero_grad()
            mask = [i for i in range(len(labels)) if labels[i] != torch.Tensor([
            features = features[mask].cuda()
            labels = labels[mask].cuda()
            '''
            forward and backward
            '''
            output = model(features)
            loss = criterion(output, labels.long())
            avg_loss_1000_batch += loss.item()
            loss.backward()

            parameter_scheduler.batch_step()
            optimizer.step()

            if index % 2000 == 0 and index != 0:

                predictions = torch.max(output.data, 1)[1]
                predicted = len(features)
                correct = int(sum(predictions == labels.to(device)).cpu())
                print("Epoch: {0}/{1} Train batch:{2}/{3}   acc: {4}  loss: {5}"
                                                                              n_
                                                                              in
                                                                              le
                                                                              co
                avg_loss_1000_batch = 0

        for index, (idx, val_features, val_labels) in enumerate(val_dataloader):

            mask = [i for i in range(len(val_labels)) if val_labels[i] != torch.
            val_features = val_features[mask].to(device)
            val_labels = val_labels[mask].to(device)
```

```
                model.eval()
                outputs = model(val_features)
                predictions = torch.max(outputs.data, 1)[1]
                val_predicted += len(val_features)
                val_correct += sum(predictions == val_labels.to(device))

            epoch_acc = int(val_correct.cpu())/val_predicted
            if epoch_acc >= 0.68:
                pickle.dump(model, open("candidate_model_{0}.pkl".format(candidate_mo
                print("Save one candidate model.")
                candidate_model += 1

            t1 = time.time()
            print("Validation Accuracy: {0}. Cost time: {1} minutes".format(int(val_c
            print("===================================================")


        return model
```

In [16]:
```
if __name__ == '__main__':


    train_features, train_labels = load_train_data()
    val_features, val_labels = load_validation_data()
    test_features = load_test_data()

    if pca_available:
        '''
        load local pca file
        '''
        pca = pickle.load(open('pca.pkl', 'rb'))
    else:
        '''
        10 features will be enough
        '''
        pca = PCA(10).fit(np.concatenate(train_features))
        pickle.dump(pca, open('pca_{0}_features.pkl'.format(pca.n_components), 'w

    train_features, train_labels = load_and_process_data(train_features, train_la
    val_features, val_labels = load_and_process_data(val_features, val_labels, pc

    train_context_dataset = ContextDataset(context_num, train_features, train_lal
    val_context_dataset = ContextDataset(context_num, val_features, val_labels)

    '''
    It may takes more than 20 minutes to process since it loops over all 15 milli
    But it could speed up later dataloader process
    And I could save this data by pickle
    The drawback is it's not flexible to feature engineering, like change context
    '''
    '''
    train_dataset = SpeechDataset(train_context_dataset)
    val_dataset = SpeechDataset(val_context_dataset)
    '''
    '''
    train_mask = (train_dataset.targets.numpy() != -1)*1
    train_sampler = WeightedRandomSampler(weights=train_mask, num_samples=int(tra
    val_mask = (val_dataset.targets.numpy() != -1)*1
    val_sampler = WeightedRandomSampler(weights=val_mask, num_samples=int(val_mas
    '''
    '''
    train_dataloader = DataLoader(train_dataset,
                                  shuffle = True,
                                  batch_size = 512,
                                  num_workers = 0,
                                  pin_memory = True)

    val_dataloader = DataLoader(val_dataset,
                                  shuffle = True,
                                  batch_size = 512,
                                  num_workers = 0,
                                  pin_memory = True)
    '''
    '''
    Start train model.
```

```
    15 epochs by default.
    Cost about 1 hour.
    '''

    train_dataloader = DataLoader(train_context_dataset,
                                  shuffle = True,
                                  batch_size = 512,
                                  num_workers = 0,
                                  pin_memory = True)

    val_dataloader = DataLoader(val_context_dataset,
                                    shuffle = True,
                                    batch_size = 512,
                                    num_workers = 0,
                                    pin_memory = True)

    model = train_model(train_dataloader, val_dataloader)

    pickle.dump(model, open("submission_model_68.pkl", "wb"))
    #make_submission(model, test_features)
```

```
Start loading training data...
Done loading training data in 0.7987695336341858 minutes...
Start loading validation data...
Done loading validation data in 0.0456453800201416 minutes...
Start loading test data...
Done loading test data in 0.014128851890563964 minutes...
Start training...
Epoch: 1/10 Train batch:2000/31779   acc: 0.45121951219512196  loss: 9.797406
8808835
Epoch: 1/10 Train batch:4000/31779   acc: 0.4979253112033195  loss: 8.1310059
98002365
Epoch: 1/10 Train batch:6000/31779   acc: 0.5040816326530613  loss: 7.6088751
33330002
Epoch: 1/10 Train batch:8000/31779   acc: 0.5708418891170431  loss: 7.2945281
34632856
Epoch: 1/10 Train batch:10000/31779   acc: 0.5343035343035343  loss: 7.054817
696334794
Epoch: 1/10 Train batch:12000/31779   acc: 0.5429769392033543  loss: 6.877835
999010131
Epoch: 1/10 Train batch:14000/31779   acc: 0.5979381443298969  loss: 6.748973
261332139
Epoch: 1/10 Train batch:16000/31779   acc: 0.5637860082304527  loss: 6.626700
401538983
Epoch: 1/10 Train batch:18000/31779   acc: 0.56875  loss: 6.515335530042648
Epoch: 1/10 Train batch:20000/31779   acc: 0.5379876796714579  loss: 6.443903
14957127
Epoch: 1/10 Train batch:22000/31779   acc: 0.5938775510204082  loss: 6.355136
304395273
Epoch: 1/10 Train batch:24000/31779   acc: 0.5967078189300411  loss: 6.303867
116337642
Epoch: 1/10 Train batch:26000/31779   acc: 0.5983263598326359  loss: 6.232847
114559263
Epoch: 1/10 Train batch:28000/31779   acc: 0.5479166666666667  loss: 6.187457
967782393
Epoch: 1/10 Train batch:30000/31779   acc: 0.5860655737704918  loss: 6.145368
884084746
Validation Accuracy: 0.6099275563894199. Cost time: 24.387852080663045 minute
```

```
s
====================================================
Epoch: 2/10 Train batch:2000/31779   acc: 0.6356107660455487  loss: 6.0118769
84499395
Epoch: 2/10 Train batch:4000/31779   acc: 0.5904365904365905  loss: 5.9843146
83126286
Epoch: 2/10 Train batch:6000/31779   acc: 0.6290983606557377  loss: 5.9456812
27371097
Epoch: 2/10 Train batch:8000/31779   acc: 0.5785123966942148  loss: 5.9185198
9692077
Epoch: 2/10 Train batch:10000/31779   acc: 0.6042105263157894  loss: 5.886191
22421369
Epoch: 2/10 Train batch:12000/31779   acc: 0.6125  loss: 5.852636412251741
Epoch: 2/10 Train batch:14000/31779   acc: 0.5871369294605809  loss: 5.832868
226803839
Epoch: 2/10 Train batch:16000/31779   acc: 0.5857142857142857  loss: 5.808117
415057495
Epoch: 2/10 Train batch:18000/31779   acc: 0.5995893223819302  loss: 5.789125
464623794
Epoch: 2/10 Train batch:20000/31779   acc: 0.59375  loss: 5.754060934996232
Epoch: 2/10 Train batch:22000/31779   acc: 0.6470588235294118  loss: 5.732057
94300884
Epoch: 2/10 Train batch:24000/31779   acc: 0.6504065040650406  loss: 5.722947
325091809
Epoch: 2/10 Train batch:26000/31779   acc: 0.6164948453608248  loss: 5.697432
388318703
Epoch: 2/10 Train batch:28000/31779   acc: 0.6244813278008299  loss: 5.680661
244317889
Epoch: 2/10 Train batch:30000/31779   acc: 0.6149068322981367  loss: 5.653364
775236696
Validation Accuracy: 0.6388902041323635. Cost time: 24.138027866681416 minute
s
====================================================
Epoch: 3/10 Train batch:2000/31779   acc: 0.6597938144329897  loss: 5.5778227
88618505
Epoch: 3/10 Train batch:4000/31779   acc: 0.6701244813278008  loss: 5.5390974
03176129
Epoch: 3/10 Train batch:6000/31779   acc: 0.6453608247422681  loss: 5.5392883
79950449
Epoch: 3/10 Train batch:8000/31779   acc: 0.6278118609406953  loss: 5.5180723
93490002
Epoch: 3/10 Train batch:10000/31779   acc: 0.6440329218106996  loss: 5.506670
6200595945
Epoch: 3/10 Train batch:12000/31779   acc: 0.6053169734151329  loss: 5.500768
842641264
Epoch: 3/10 Train batch:14000/31779   acc: 0.6529774127310062  loss: 5.487650
30503273
Epoch: 3/10 Train batch:16000/31779   acc: 0.6459627329192547  loss: 5.484318
618895486
Epoch: 3/10 Train batch:18000/31779   acc: 0.6523517382413088  loss: 5.469916
230067611
Epoch: 3/10 Train batch:20000/31779   acc: 0.6515463917525773  loss: 5.443030
263995752
Epoch: 3/10 Train batch:22000/31779   acc: 0.6040816326530613  loss: 5.427898
246562108
Epoch: 3/10 Train batch:24000/31779   acc: 0.6561181434599156  loss: 5.423127
093818039
```

```
Epoch: 3/10 Train batch:26000/31779   acc: 0.639344262295082  loss: 5.4192846
20322287
Epoch: 3/10 Train batch:28000/31779   acc: 0.6378600823045267  loss: 5.411432
175664231
Epoch: 3/10 Train batch:30000/31779   acc: 0.663135593220339  loss: 5.3958395
22359893
Validation Accuracy: 0.6519155534774709. Cost time: 24.139771298567453 minute
s
====================================================
Epoch: 4/10 Train batch:2000/31779   acc: 0.6257668711656442  loss: 5.3162106
93912581
Epoch: 4/10 Train batch:4000/31779   acc: 0.6548856548856549  loss: 5.2943583
80604535
Epoch: 4/10 Train batch:6000/31779   acc: 0.5907172995780591  loss: 5.2959450
49962029
Epoch: 4/10 Train batch:8000/31779   acc: 0.6321353065539113  loss: 5.2873974
24690425
Epoch: 4/10 Train batch:10000/31779   acc: 0.5917525773195876  loss: 5.273880
596272647
Epoch: 4/10 Train batch:12000/31779   acc: 0.6605691056910569  loss: 5.276953
265769407
Epoch: 4/10 Train batch:14000/31779   acc: 0.6401673640167364  loss: 5.265228
853095323
Epoch: 4/10 Train batch:16000/31779   acc: 0.6384297520661157  loss: 5.254212
558735162
Epoch: 4/10 Train batch:18000/31779   acc: 0.6346555323590815  loss: 5.241534
168599173
Epoch: 4/10 Train batch:20000/31779   acc: 0.675564681724846  loss: 5.2273399
3944712
Epoch: 4/10 Train batch:22000/31779   acc: 0.6567796610169492  loss: 5.215027
284575626
Epoch: 4/10 Train batch:24000/31779   acc: 0.6880165289256198  loss: 5.212054
678238928
Epoch: 4/10 Train batch:26000/31779   acc: 0.656964656964657  loss: 5.2214122
3680228
Epoch: 4/10 Train batch:28000/31779   acc: 0.6869918699186992  loss: 5.199652
779381722
Epoch: 4/10 Train batch:30000/31779   acc: 0.6632860040567952  loss: 5.195865
701651201
Validation Accuracy: 0.6607150255387402. Cost time: 24.175887401898702 minute
s
====================================================
Epoch: 5/10 Train batch:2000/31779   acc: 0.6170212765957447  loss: 5.0965385
02257317
Epoch: 5/10 Train batch:4000/31779   acc: 0.654320987654321  loss: 5.11735978
4897417
Epoch: 5/10 Train batch:6000/31779   acc: 0.6894409937888198  loss: 5.1167313
07236478
Epoch: 5/10 Train batch:8000/31779   acc: 0.7052845528455285  loss: 5.0985987
41926253
Epoch: 5/10 Train batch:10000/31779   acc: 0.65625  loss: 5.090086501324549
Epoch: 5/10 Train batch:12000/31779   acc: 0.6460905349794238  loss: 5.081751
8518306315
Epoch: 5/10 Train batch:14000/31779   acc: 0.696969696969697  loss: 5.0804935
67278609
Epoch: 5/10 Train batch:16000/31779   acc: 0.6275720164609053  loss: 5.084368
4275168926
```

```
Epoch: 5/10 Train batch:18000/31779   acc: 0.6659836065573771  loss: 5.072955
843992531
Epoch: 5/10 Train batch:20000/31779   acc: 0.6511156186612576  loss: 5.063052
929705009
Epoch: 5/10 Train batch:22000/31779   acc: 0.6880165289256198  loss: 5.057854
80258055
Epoch: 5/10 Train batch:24000/31779   acc: 0.694560669456067  loss: 5.0376209
418755025
Epoch: 5/10 Train batch:26000/31779   acc: 0.6912065439672802  loss: 5.037911
129882559
Epoch: 5/10 Train batch:28000/31779   acc: 0.651356993736952  loss: 5.0399528
671987355
Epoch: 5/10 Train batch:30000/31779   acc: 0.6495901639344263  loss: 5.034081
088611856
Validation Accuracy: 0.6682242437514426. Cost time: 24.154389715194704 minute
s
===================================================
Epoch: 6/10 Train batch:2000/31779   acc: 0.609504132231405  loss: 4.94795658
3695486
Epoch: 6/10 Train batch:4000/31779   acc: 0.6456211812627292  loss: 4.9554243
37415025
Epoch: 6/10 Train batch:6000/31779   acc: 0.6715481171548117  loss: 4.9398640
28710872
Epoch: 6/10 Train batch:8000/31779   acc: 0.6659836065573771  loss: 4.9363483
38378593
Epoch: 6/10 Train batch:10000/31779   acc: 0.6743697478991597  loss: 4.935492
197517306
Epoch: 6/10 Train batch:12000/31779   acc: 0.6409185803757829  loss: 4.931298
549287021
Epoch: 6/10 Train batch:14000/31779   acc: 0.6340956340956341  loss: 4.934873
427497223
Epoch: 6/10 Train batch:16000/31779   acc: 0.6268041237113402  loss: 4.929511
033347808
Epoch: 6/10 Train batch:18000/31779   acc: 0.6902286902286903  loss: 4.925154
0408004075
Epoch: 6/10 Train batch:20000/31779   acc: 0.6757322175732218  loss: 4.921367
508592084
Epoch: 6/10 Train batch:22000/31779   acc: 0.689795918367347  loss: 4.9169129
87595424
Epoch: 6/10 Train batch:24000/31779   acc: 0.7276507276507277  loss: 4.916759
182233363
Epoch: 6/10 Train batch:26000/31779   acc: 0.6735966735966736  loss: 4.895018
240902573
Epoch: 6/10 Train batch:28000/31779   acc: 0.6604938271604939  loss: 4.895448
314957321
Epoch: 6/10 Train batch:30000/31779   acc: 0.6997929606625258  loss: 4.893644
623807631
Validation Accuracy: 0.6722858800063921. Cost time: 24.207170498371124 minute
s
===================================================
Epoch: 7/10 Train batch:2000/31779   acc: 0.6473029045643154  loss: 4.8236181
4728938
Epoch: 7/10 Train batch:4000/31779   acc: 0.6871035940803383  loss: 4.8115820
796228945
Epoch: 7/10 Train batch:6000/31779   acc: 0.6911764705882353  loss: 4.8166717
87411906
Epoch: 7/10 Train batch:8000/31779   acc: 0.709278350515464  loss: 4.80841255
```

```
8748387
Epoch: 7/10 Train batch:10000/31779   acc: 0.676954732510288  loss: 4.8102868
042187765
Epoch: 7/10 Train batch:12000/31779   acc: 0.6514522821576764  loss: 4.803506
872034632
Epoch: 7/10 Train batch:14000/31779   acc: 0.65439672801636  loss: 4.79885984
9339351
Epoch: 7/10 Train batch:16000/31779   acc: 0.6570841889117043  loss: 4.791102
161863819
Epoch: 7/10 Train batch:18000/31779   acc: 0.6687116564417178  loss: 4.789656
708016992
Epoch: 7/10 Train batch:20000/31779   acc: 0.6772823779193206  loss: 4.790836
266707629
Epoch: 7/10 Train batch:22000/31779   acc: 0.6940928270042194  loss: 4.789164
402987808
Epoch: 7/10 Train batch:24000/31779   acc: 0.6867219917012448  loss: 4.794918
390456587
Epoch: 7/10 Train batch:26000/31779   acc: 0.6804123711340206  loss: 4.764776
983647607
Epoch: 7/10 Train batch:28000/31779   acc: 0.6804979253112033  loss: 4.777038
602274843
Epoch: 7/10 Train batch:30000/31779   acc: 0.6639175257731958  loss: 4.764546
650694683
Validation Accuracy: 0.6772663782337727. Cost time: 24.20336433649063 minutes
===================================================
Epoch: 8/10 Train batch:2000/31779   acc: 0.6796714579055442  loss: 4.7099605
59150204
Epoch: 8/10 Train batch:4000/31779   acc: 0.6818181818181818  loss: 4.6982629
32128273
Epoch: 8/10 Train batch:6000/31779   acc: 0.675  loss: 4.705215996829793
Epoch: 8/10 Train batch:8000/31779   acc: 0.681912681912682  loss: 4.72149522
7771811
Epoch: 8/10 Train batch:10000/31779   acc: 0.6529774127310062  loss: 4.694871
228188276
Epoch: 8/10 Train batch:12000/31779   acc: 0.6611570247933884  loss: 4.711556
7362634465
Epoch: 8/10 Train batch:14000/31779   acc: 0.6762295081967213  loss: 4.691379
2652776465
Epoch: 8/10 Train batch:16000/31779   acc: 0.7098765432098766  loss: 4.691945
286118425
Epoch: 8/10 Train batch:18000/31779   acc: 0.7037037037037037  loss: 4.689044
41955965
Epoch: 8/10 Train batch:20000/31779   acc: 0.652542372881356  loss: 4.6868194
46622394
Epoch: 8/10 Train batch:22000/31779   acc: 0.6550308008213552  loss: 4.699441
241915338
Epoch: 8/10 Train batch:24000/31779   acc: 0.6735112936344969  loss: 4.680031
765834428
Epoch: 8/10 Train batch:26000/31779   acc: 0.6855345911949685  loss: 4.680239
063454792
Epoch: 8/10 Train batch:28000/31779   acc: 0.6862348178137652  loss: 4.681143
297930248
Epoch: 8/10 Train batch:30000/31779   acc: 0.7018633540372671  loss: 4.674899
903242476
Validation Accuracy: 0.6798705603134488. Cost time: 24.238636338710783 minute
s
===================================================
```

```
Epoch: 9/10 Train batch:2000/31779   acc: 0.6673596673596673  loss: 4.6252376
6933009
Epoch: 9/10 Train batch:4000/31779   acc: 0.6777546777546778  loss: 4.6333181
56236783
Epoch: 9/10 Train batch:6000/31779   acc: 0.6894409937888198  loss: 4.6313127
82526948
Epoch: 9/10 Train batch:8000/31779   acc: 0.6927835051546392  loss: 4.6282717
73690358
Epoch: 9/10 Train batch:10000/31779   acc: 0.7125  loss: 4.638095643138513
Epoch: 9/10 Train batch:12000/31779   acc: 0.6900826446280992  loss: 4.626812
071190216
Epoch: 9/10 Train batch:14000/31779   acc: 0.689795918367347  loss: 4.6344016
23974554
Epoch: 9/10 Train batch:16000/31779   acc: 0.7154639175257732  loss: 4.624055
395019241
Epoch: 9/10 Train batch:18000/31779   acc: 0.6834381551362684  loss: 4.624530
459055677
Epoch: 9/10 Train batch:20000/31779   acc: 0.6923076923076923  loss: 4.618407
785426825
Epoch: 9/10 Train batch:22000/31779   acc: 0.7134146341463414  loss: 4.614892
411511391
Epoch: 9/10 Train batch:24000/31779   acc: 0.6652977412731006  loss: 4.626754
371216521
Epoch: 9/10 Train batch:26000/31779   acc: 0.65625  loss: 4.615499716019258
Epoch: 9/10 Train batch:28000/31779   acc: 0.6956521739130435  loss: 4.628636
922338046
Epoch: 9/10 Train batch:30000/31779   acc: 0.66875  loss: 4.606372268171981
Save one candidate model.
Validation Accuracy: 0.6809092738475014. Cost time: 24.235928801695504 minute
s
===================================================
Epoch: 10/10 Train batch:2000/31779   acc: 0.6987704918032787  loss: 4.599639
82575573
Epoch: 10/10 Train batch:4000/31779   acc: 0.6653061224489796  loss: 4.593702
975544147
Epoch: 10/10 Train batch:6000/31779   acc: 0.6356107660455487  loss: 4.588591
465493664
Epoch: 10/10 Train batch:8000/31779   acc: 0.6604166666666667  loss: 4.586460
459046066
Epoch: 10/10 Train batch:10000/31779   acc: 0.6536082474226804  loss: 4.60019
2621466704
Epoch: 10/10 Train batch:12000/31779   acc: 0.668041237113402  loss: 4.577259
998070076
Epoch: 10/10 Train batch:14000/31779   acc: 0.6939203354297694  loss: 4.59205
89016750455
Epoch: 10/10 Train batch:16000/31779   acc: 0.6831275720164609  loss: 4.59070
4471222125
Epoch: 10/10 Train batch:18000/31779   acc: 0.6708860759493671  loss: 4.58351
8972620368
Epoch: 10/10 Train batch:20000/31779   acc: 0.6694560669456067  loss: 4.59214
6473238245
Epoch: 10/10 Train batch:22000/31779   acc: 0.6975308641975309  loss: 4.59304
5651330613
Epoch: 10/10 Train batch:24000/31779   acc: 0.6548117154811716  loss: 4.58983
1111487001
Epoch: 10/10 Train batch:26000/31779   acc: 0.6851851851851852  loss: 4.58333
218190819
```

```
Epoch: 10/10 Train batch:28000/31779   acc: 0.7063655030800822  loss: 4.60929
1448723525
Epoch: 10/10 Train batch:30000/31779   acc: 0.717479674796748  loss: 4.582990
495022386
Save one candidate model.
Validation Accuracy: 0.6819894175510035. Cost time: 24.225563311576842 minute
s
===================================================
```

In [21]:

```python
def make_submission(test_features, model):

    test_false_labels = np.array([np.zeros((test_features[i].shape[0], 1)).resha
    test_features, test_labels = load_and_process_data(test_features, test_false_
    '''
    generate false labels(just used to skip padding frame)
    '''
    test_context_dataset = ContextDataset(context_num,test_features, test_labels
    test_dataloader = DataLoader(test_context_dataset,
                              shuffle = False,
                              batch_size = 512,
                              num_workers = 0,
                              pin_memory = True,
                              drop_last = False)

    with torch.no_grad():
        model.eval().cuda()
        sub_predictions = []
        for idx, (index, features, labels) in enumerate(test_dataloader):
            mask = [i for i in range(len(labels)) if labels[i] != torch.Tensor([
            features = features[mask].cuda()
            outputs= model(features.cuda())
            predictions = torch.max(outputs.data, 1)[1]
            sub_predictions.append(predictions)
        sub_predictions = torch.cat(sub_predictions).flatten().cpu().numpy()

    submission = pd.DataFrame(sub_predictions, columns = ['label'])
    submission.to_csv("Submission.csv", index_label = 'id')
    print("Submission.csv done.")
```

In [22]:
```
if __name__ == '__main__':

    test_features = load_test_data()
    if pca_available:
        '''
        load local pca file
        '''
        pca = pickle.load(open('pca.pkl', 'rb'))
    else:
        '''
        10 features will be enough
        '''
        pca = PCA(10).fit(np.concatenate(train_features))
        pickle.dump(pca, open('pca_{0}_features.pkl'.format(pca.n_components), 'w
    model = pickle.load(open("submission_model_68.pkl", "rb"))
    make_submission(test_features, model)
```

```
Start loading test data...
Done loading test data in 0.005717750390370687 minutes...
Submission.csv done.
```

In [ ]: