In [1]:
```
%matplotlib inline
```

In [2]:
```
import torch
import torch.autograd as autograd
import torch.nn as nn
import torch.optim as optim
import numpy as np
torch.manual_seed(1)
from sklearn.metrics import roc_auc_score
from sklearn.metrics import f1_score
import copy
```

In [3]:

```
label_to_ix=np.load('label_to_ix.npy', allow_pickle=True).item()
ix_to_label=np.load('ix_to_label.npy', allow_pickle=True)
training_data=np.load('training_data.npy', allow_pickle=True)
test_data=np.load('test_data.npy', allow_pickle=True)
val_data=np.load('val_data.npy', allow_pickle=True)
word_to_ix=np.load('word_to_ix.npy', allow_pickle=True).item()
ix_to_word=np.load('ix_to_word.npy', allow_pickle=True)
newwikivec=np.load('newwikivec.npy', allow_pickle=True)
wikivoc=np.load('wikivoc.npy', allow_pickle=True).item()



wikisize=newwikivec.shape[0]
rvocsize=newwikivec.shape[1]
wikivec=autograd.Variable(torch.FloatTensor(newwikivec))

batchsize=32



def preprocessing(data):

    new_data=[]
    for i, note, j in data:
        templabel=[0.0]*len(label_to_ix)
        for jj in j:
            if jj in wikivoc:
                templabel[label_to_ix[jj]]=1.0
        templabel=np.array(templabel,dtype=float)
        new_data.append((i, note, templabel))
    new_data=np.array(new_data)

    lenlist=[]
    for i in new_data:
        lenlist.append(len(i[0]))
    sortlen=sorted(range(len(lenlist)), key=lambda k: lenlist[k])
    new_data=new_data[sortlen]

    batch_data=[]

    for start_ix in range(0, len(new_data)-batchsize+1, batchsize):
        thisblock=new_data[start_ix:start_ix+batchsize]
        mybsize= len(thisblock)
        numword=np.max([len(ii[0]) for ii in thisblock])
        main_matrix = np.zeros((mybsize, numword), dtype= np.int)
        for i in range(main_matrix.shape[0]):
            for j in range(main_matrix.shape[1]):
                try:
                    if thisblock[i][0][j] in word_to_ix:
                        main_matrix[i,j] = word_to_ix[thisblock[i][0][j]]

                except IndexError:
                    pass       # because initialze with 0, so you pad with 0

        xxx2=[]
        yyy=[]
```

```
        for ii in thisblock:
            xxx2.append(ii[1])
            yyy.append(ii[2])

        xxx2=np.array(xxx2)
        yyy=np.array(yyy)
        batch_data.append((autograd.Variable(torch.from_numpy(main_matrix)),
    return batch_data
batchtraining_data=preprocessing(training_data)
batchtest_data=preprocessing(test_data)
batchval_data=preprocessing(val_data)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:31: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray
```

Create the model:

In [4]:

```
Embeddingsize=100
hidden_dim=200
class CNN(nn.Module):

    def __init__(self, batch_size, vocab_size, tagset_size):
        super(CNN, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size+1, Embeddingsize, pad
        self.embed_drop = nn.Dropout(p=0.2)

        self.hidden2tag = nn.Linear(300, tagset_size)


        self.convs1 = nn.Conv1d(Embeddingsize,100,3)
        self.convs2 = nn.Conv1d(Embeddingsize,100,4)
        self.convs3 = nn.Conv1d(Embeddingsize,100,5)


        self.layer2 = nn.Linear(Embeddingsize, 1,bias=False)
        self.embedding=nn.Linear(rvocsize,Embeddingsize)
        self.vattention=nn.Linear(Embeddingsize,Embeddingsize)

        self.sigmoid = nn.Sigmoid()
        self.tanh = nn.Tanh()
        self.dropout = nn.Dropout(0.2)

    def forward(self, vec1, nvec, wiki, simlearning):

        thisembeddings=self.word_embeddings(vec1)
        thisembeddings = self.embed_drop(thisembeddings)
        thisembeddings=thisembeddings.transpose(1,2)

        output1=self.tanh(self.convs1(thisembeddings))
        output1=nn.MaxPool1d(output1.size()[2])(output1)

        output2=self.tanh(self.convs2(thisembeddings))
        output2=nn.MaxPool1d(output2.size()[2])(output2)

        output3=self.tanh(self.convs3(thisembeddings))
        output3=nn.MaxPool1d(output3.size()[2])(output3)

        output4 = torch.cat([output1,output2,output3], 1).squeeze(2)

        if simlearning==1:
            nvec=nvec.view(batchsize,1,-1)
            nvec=nvec.expand(batchsize,wiki.size()[0],-1)
            wiki=wiki.view(1,wiki.size()[0],-1)
            wiki=wiki.expand(nvec.size()[0],wiki.size()[1],-1)
            new=wiki*nvec
            new=self.embedding(new)
            vattention=self.sigmoid(self.vattention(new))
            new=new*vattention
            vec3=self.layer2(new)
            vec3=vec3.view(batchsize,-1)


```

```
        vec2 = self.hidden2tag(output4)
        if simlearning==1:
            tag_scores = self.sigmoid(vec2.detach()+vec3)
        else:
            tag_scores = self.sigmoid(vec2)


        return tag_scores
```

Train the model:

In [6]:

```
topk=10

def trainmodel(model, sim):
    print ('start_training')
    modelsaved=[]
    modelperform=[]
    topk=10


    bestresults=-1
    bestiter=-1
    for epoch in range(1000):
        model.train()

        lossestrain = []
        recall=[]
        for mysentence in batchtraining_data:
            model.zero_grad()

            targets = mysentence[2].cuda()
            tag_scores = model(mysentence[0].cuda(),mysentence[1].cuda(),wik
            loss = loss_function(tag_scores, targets)
            loss.backward()
            optimizer.step()
            lossestrain.append(loss.data.mean())
        print (epoch)
        modelsaved.append(copy.deepcopy(model.state_dict()))
        print ("XXXXXXXXXXXXXXXXXXXXXXXXXXXX")
        model.eval()

        recall=[]
        for inputs in batchval_data:

            targets = inputs[2].cuda()
            tag_scores = model(inputs[0].cuda(),inputs[1].cuda() ,wikivec.cu

            loss = loss_function(tag_scores, targets)

            targets=targets.data.cpu().numpy()
            tag_scores= tag_scores.data.cpu().numpy()


            for iii in range(0,len(tag_scores)):
                temp={}
                for iiii in range(0,len(tag_scores[iii])):
                    temp[iiii]=tag_scores[iii][iiii]
                temp1=[(k, temp[k]) for k in sorted(temp, key=temp.get, reve
                thistop=int(np.sum(targets[iii]))
                hit=0.0
                for ii in temp1[0:max(thistop,topk)]:
                    if targets[iii][ii[0]]==1.0:
                        hit=hit+1
                if thistop!=0:
                    recall.append(hit/thistop)

        print ('validation top-',topk, np.mean(recall))
```

```
57
58
59
60          modelperform.append(np.mean(recall))
61          if modelperform[-1]>bestresults:
62              bestresults=modelperform[-1]
63              bestiter=len(modelperform)-1
64
65          if (len(modelperform)-bestiter)>5:
66              print (modelperform,bestiter)
67              return modelsaved[bestiter]
68
69  model = CNN(batchsize, len(word_to_ix), len(label_to_ix))
70  model.cuda()
71
72  loss_function = nn.BCELoss()
73  optimizer = optim.Adam(model.parameters())
74
75  basemodel= trainmodel(model, 0)
76  torch.save(basemodel, 'CNN_model')
77
78  model = CNN(batchsize, len(word_to_ix), len(label_to_ix))
79  model.cuda()
80  model.load_state_dict(basemodel)
81  loss_function = nn.BCELoss()
82  optimizer = optim.Adam(model.parameters())
83  KSImodel= trainmodel(model, 1)
84  torch.save(KSImodel, 'KSI_CNN_model')
85
86  def testmodel(modelstate, sim):
87      model = CNN(batchsize, len(word_to_ix), len(label_to_ix))
88      model.cuda()
89      model.load_state_dict(modelstate)
90      loss_function = nn.BCELoss()
91      model.eval()
92      recall=[]
93      lossestest = []
94
95      y_true=[]
96      y_scores=[]
97
98
99      for inputs in batchtest_data:
100
101         targets = inputs[2].cuda()
102
103         tag_scores = model(inputs[0].cuda(),inputs[1].cuda() ,wikivec.cuda()
104
105         loss = loss_function(tag_scores, targets)
106
107         targets=targets.data.cpu().numpy()
108         tag_scores= tag_scores.data.cpu().numpy()
109
110
111         lossestest.append(loss.data.mean())
112         y_true.append(targets)
113         y_scores.append(tag_scores)
```

```
114
115         for iii in range(0,len(tag_scores)):
116             temp={}
117             for iiii in range(0,len(tag_scores[iii])):
118                 temp[iiii]=tag_scores[iii][iiii]
119             temp1=[(k, temp[k]) for k in sorted(temp, key=temp.get, reverse=
120             thistop=int(np.sum(targets[iii]))
121             hit=0.0
122 
123             for ii in temp1[0:max(thistop,topk)]:
124                 if targets[iii][ii[0]]==1.0:
125                     hit=hit+1
126             if thistop!=0:
127                 recall.append(hit/thistop)
128     y_true=np.concatenate(y_true,axis=0)
129     y_scores=np.concatenate(y_scores,axis=0)
130     y_true=y_true.T
131     y_scores=y_scores.T
132     temptrue=[]
133     tempscores=[]
134     for  col in range(0,len(y_true)):
135         if np.sum(y_true[col])!=0:
136             temptrue.append(y_true[col])
137             tempscores.append(y_scores[col])
138     temptrue=np.array(temptrue)
139     tempscores=np.array(tempscores)
140     y_true=temptrue.T
141     y_scores=tempscores.T
142     y_pred=(y_scores>0.5).astype(np.int)
143     print ('test loss', torch.stack(lossestest).mean().item())
144     print ('top-',topk, np.mean(recall))
145     print ('macro AUC', roc_auc_score(y_true, y_scores,average='macro'))
146     print ('micro AUC', roc_auc_score(y_true, y_scores,average='micro'))
147     print ('macro F1', f1_score(y_true, y_pred, average='macro')  )
148     print ('micro F1', f1_score(y_true, y_pred, average='micro')  )
149 
150 print ('CNN alone:           ')
151 testmodel(basemodel, 0)
152 print ('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
153 print ('KSI+CNN:           ')
154 testmodel(KSImodel, 1)
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4530655205021126
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.529073778538287
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5752237865553222
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6170548555043338
4
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6467904998800238
5
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6673111386677142
6
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6812441074304616
7
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6933596956867344
8
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7052186240553053
9
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7117489006787834
10
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.722211327818717
11
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7265657968833666
12
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7334972301355923
13
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7345698864388871
14
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7347385604593721
15
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7408095989262027
16
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7420631058486372
17
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7462089367848072
18
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7501318022145766
19
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7528999172460481
20
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7534285951564181
21
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7498198864222271
22
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7532788638686233
23
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7540027411409296
24
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7573006911292467
25
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7541092517584836
26
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7552991371761425
27
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7589011476791745
28
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7577853193291497
29
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7583405388902464
30
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7560631161889517
31
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7577583138492285
32
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7598690505225836
33
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7560855487936969
34
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.756507376786185
35
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7564485341974435
36
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7546294812010665
37
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7564334654936617
[0.4530655205021126, 0.529073778538287, 0.5752237865553222, 0.6170548555043338,
0.6467904998800238, 0.6673111386677142, 0.6812441074304616, 0.6933596956867344,
0.7052186240553053, 0.7117489006787834, 0.722211327818717, 0.7265657968833666,
0.7334972301355923, 0.7345698864388871, 0.7347385604593721, 0.7408095989262027,
0.7420631058486372, 0.7462089367848072, 0.7501318022145766, 0.7528999172460481,
0.7534285951564181, 0.7498198864222271, 0.7532788638686233, 0.7540027411409296,
0.7573006911292467, 0.7541092517584836, 0.7552991371761425, 0.7589011476791745,
0.7577853193291497, 0.7583405388902464, 0.7560631161889517, 0.7577583138492285,
0.7598690505225836, 0.7560855487936969, 0.756507376786185, 0.7564485341974435,
0.7546294812010665, 0.7564334654936617] 32
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
validation top- 10 0.7669781847454861
1
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7734568574207574
2
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7768123484218621
3
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7797745185621022
4
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.780834126290448
5
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7800837381359998
6
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7795087269621294
7
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7794720793975843
8
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7781662562211387
9
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7783857329042567
[0.7669781847454861, 0.7734568574207574, 0.7768123484218621, 0.779774518562102
2, 0.780834126290448, 0.7800837381359998, 0.7795087269621294, 0.779472079397584
3, 0.7781662562211387, 0.7783857329042567] 4
CNN alone:
test loss 0.03949494659900665
top- 10 0.753674492427492
macro AUC 0.8306164538048904
micro AUC 0.9666265882181901
macro F1 0.21344930909581109
micro F1 0.6257646033743762
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+CNN:
test loss 0.03750767931342125
top- 10 0.7747399300742249
macro AUC 0.8609588009174831
micro AUC 0.9723801813466743
macro F1 0.24036851193298106
micro F1 0.6399044546248596
```

In [ ]:
```
1
```