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:30: 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 LSTM(nn.Module):

    def __init__(self, batch_size, vocab_size, tagset_size):
        super(LSTM, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size+1, Embeddingsize, pad
        self.lstm = nn.LSTM(Embeddingsize, hidden_dim)
        self.hidden2tag = nn.Linear(hidden_dim, tagset_size)
        self.hidden = self.init_hidden()


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

        self.softmax = nn.Softmax()
        self.sigmoid = nn.Sigmoid()
        self.tanh = nn.Tanh()
        self.embed_drop = nn.Dropout(p=0.2)

    def init_hidden(self):
        return (autograd.Variable(torch.zeros(1, batchsize, self.hidden_dim)
                autograd.Variable(torch.zeros(1, batchsize, self.hidden_dim)


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

        thisembeddings=self.word_embeddings(vec1).transpose(0,1)
        thisembeddings = self.embed_drop(thisembeddings)

        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)



        lstm_out, self.hidden = self.lstm(
            thisembeddings, self.hidden)

        lstm_out=lstm_out.transpose(0,2).transpose(0,1)

        output1=nn.MaxPool1d(lstm_out.size()[2])(lstm_out).view(batchsize,-1

        vec2 = self.hidden2tag(output1)
        if simlearning==1:
            tag_scores = self.sigmoid(vec2.detach()+vec3)
```

```
        else:
            tag_scores = self.sigmoid(vec2)


        return tag_scores
```

Train the model:

In [5]:

```
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()
            model.hidden = model.init_hidden()
            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:
            model.hidden = model.init_hidden()
            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)
```

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

```
        y_true.append(targets)
        y_scores.append(tag_scores)

        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, reverse=
            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)
    y_true=np.concatenate(y_true,axis=0)
    y_scores=np.concatenate(y_scores,axis=0)
    y_true=y_true.T
    y_scores=y_scores.T
    temptrue=[]
    tempscores=[]
    for  col in range(0,len(y_true)):
        if np.sum(y_true[col])!=0:
            temptrue.append(y_true[col])
            tempscores.append(y_scores[col])
    temptrue=np.array(temptrue)
    tempscores=np.array(tempscores)
    y_true=temptrue.T
    y_scores=tempscores.T
    y_pred=(y_scores>0.5).astype(np.int)
    print ('test loss', torch.stack(lossestest).mean().item())
    print ('top-',topk, np.mean(recall))
    print ('macro AUC', roc_auc_score(y_true, y_scores,average='macro'))
    print ('micro AUC', roc_auc_score(y_true, y_scores,average='micro'))
    print ('macro F1', f1_score(y_true, y_pred, average='macro')  )
    print ('micro F1', f1_score(y_true, y_pred, average='micro')  )

print ('LSTM alone:           ')
testmodel(basemodel, 0)
print ('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
print ('KSI+LSTM:           ')
testmodel(KSImodel, 1)
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.42865152425063296
1
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4937347971844057
2
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5270009420629901
3
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5647751687941437
```

```
4
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6070973919648759
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.639208033930959
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6591861390178926
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6720871705039838
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.681262838674693
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6952146312264086
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7048960649029746
11
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.709358782967236
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7133218440095446
13
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7188960280605515
14
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7238117730247063
15
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7336735645029677
16
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7360798024341662
17
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.739616167232288
18
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7447816545167835
19
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7469938538969276
20
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7477263734691333
21
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7520601792137857
22
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7541944230919304
```

```
23
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7524959970805342
24
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7563050451094965
25
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.754255788129518
26
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.757657513173346
27
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7557928129346213
28
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7579869778923267
29
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7589916913320011
30
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7606378455246832
31
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7589054716831615
32
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7592891570709687
33
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7614059946409589
34
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7627144764777577
35
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7627529203510008
36
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7618910381368089
37
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7627580779315861
38
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7659574118049436
39
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7631046816385884
40
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7647194594740986
41
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7645101229076846
```

```
42
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7651831962837605
43
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7651884552127561
[0.42865152425063296, 0.4937347971844057, 0.5270009420629901, 0.5647751687941
437, 0.6070973919648759, 0.639208033930959, 0.6591861390178926, 0.67208717050
39838, 0.681262838674693, 0.6952146312264086, 0.7048960649029746, 0.709358782
967236, 0.7133218440095446, 0.7188960280605515, 0.7238117730247063, 0.7336735
645029677, 0.7360798024341662, 0.739616167232288, 0.7447816545167835, 0.74699
38538969276, 0.7477263734691333, 0.7520601792137857, 0.7541944230919304, 0.75
24959970805342, 0.7563050451094965, 0.754255788129518, 0.757657513173346, 0.7
557928129346213, 0.7579869778923267, 0.7589916913320011, 0.7606378455246832,
0.7589054716831615, 0.7592891570709687, 0.7614059946409589, 0.762714476477757
7, 0.7627529203510008, 0.7618910381368089, 0.7627580779315861, 0.765957411804
9436, 0.7631046816385884, 0.7647194594740986, 0.7645101229076846, 0.765183196
2837605, 0.7651884552127561] 38
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7837404161967941
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7909443516204245
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7909998188906014
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7925701125486476
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7929621953952032
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7905842937402352
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7899778174580904
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7857877460663435
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7835120129886308
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7787964887872976
[0.7837404161967941, 0.7909443516204245, 0.7909998188906014, 0.79257011254864
76, 0.7929621953952032, 0.7905842937402352, 0.7899778174580904, 0.78578774606
63435, 0.7835120129886308, 0.7787964887872976] 4
LSTM alone:
test loss 0.03404521197080612
top- 10 0.7633712690864152
macro AUC 0.8337684970090297
micro AUC 0.9695393405722679
```

```
macro F1 0.1951925657098338
micro F1 0.6436915441878222
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+LSTM:
test loss 0.03159036487340927
top- 10 0.7901006497748037
macro AUC 0.868673097613561
micro AUC 0.976214050352376
macro F1 0.24646961919586005
micro F1 0.6546062107507058
```