In [1]:

```
%matplotlib inline
```

In [2]:

```
import torch
import torch.autograd as autograd
import torch.nn as nn
import torch.nn.functional as F
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:29: VisibleDepreca
tionWarning: 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 de
precated. If you meant to do this, you must specify 'dtype=object' when creatin
g the ndarray
```

Create the model:

In [4]:

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

    def __init__(self, batch_size, vocab_size, tagset_size):
        super(LSTMattn, 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.hidden = self.init_hidden()

        self.H=nn.Linear(hidden_dim, tagset_size )
        self.final = nn.Linear(hidden_dim, tagset_size)

        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.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(0,1)

        alpha=self.H.weight.matmul(lstm_out.transpose(1,2))
        alpha=F.softmax(alpha, dim=2)

        m=alpha.matmul(lstm_out)

        myfinal=self.final.weight.mul(m).sum(dim=2).add(self.final.bias)


        if simlearning==1:
            tag_scores = self.sigmoid(myfinal.detach()+vec3)
        else:
            tag_scores = self.sigmoid(myfinal)


        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)

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



        modelperform.append(np.mean(recall))
        if modelperform[-1]>bestresults:
            bestresults=modelperform[-1]
            bestiter=len(modelperform)-1

        if (len(modelperform)-bestiter)>5:
            print (modelperform,bestiter)
            return modelsaved[bestiter]

model = LSTMattn(batchsize, len(word_to_ix), len(label_to_ix))
model.cuda()
loss_function = nn.BCELoss()
optimizer = optim.Adam(model.parameters())

basemodel= trainmodel(model, 0)
torch.save(basemodel, 'LSTMattn_model')

model = LSTMattn(batchsize, len(word_to_ix), len(label_to_ix))
model.cuda()
model.load_state_dict(basemodel)
loss_function = nn.BCELoss()
optimizer = optim.Adam(model.parameters())
KSImodel= trainmodel(model, 1)
torch.save(KSImodel, 'KSI_LSTMattn_model')

def testmodel(modelstate, sim):
    model = LSTMattn(batchsize, len(word_to_ix), len(label_to_ix))
    model.cuda()
    model.load_state_dict(modelstate)
    loss_function = nn.BCELoss()
    model.eval()
    recall=[]
    lossestest = []

    y_true=[]
    y_scores=[]


    for inputs in batchtest_data:
        model.hidden = model.init_hidden()
        targets = inputs[2].cuda()

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

        loss = loss_function(tag_scores, targets)

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


        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 ('LSTMattn alone:           ')
testmodel(basemodel, 0)
print ('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
print ('KSI+LSTMattn:           ')
testmodel(KSImodel, 1)
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.37988820337654555
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5104482603359948
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6266978811026137
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6595165288696112
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
validation top- 10 0.6859925783877213
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7074581327207593
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7279008211626297
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7452480736993783
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.759016435522099
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7637212570536341
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7705378830020023
11
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7759446183764552
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7818910104256545
13
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7873337243834775
14
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7897661533039431
15
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7896431993888611
16
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7931866940027331
17
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7941931676870783
18
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7928390644189163
19
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7963826490295235
20
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7961022934012223
21
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7951516205182724
22
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7973918889129703
23
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
validation top- 10 0.7948993508422044
24
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7938015279237891
25
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7935765049264573
26
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7928097294806639
27
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7928827429214348
[0.37988820337654555, 0.5104482603359948, 0.6266978811026137, 0.6595165288696
112, 0.6859925783877213, 0.7074581327207593, 0.7279008211626297, 0.7452480736
993783, 0.759016435522099, 0.7637212570536341, 0.7705378830020023, 0.77594461
83764552, 0.7818910104256545, 0.7873337243834775, 0.7897661533039431, 0.78964
31993888611, 0.7931866940027331, 0.7941931676870783, 0.7928390644189163, 0.79
63826490295235, 0.7961022934012223, 0.7951516205182724, 0.7973918889129703,
0.7948993508422044, 0.7938015279237891, 0.7935765049264573, 0.792809729480663
9, 0.7928827429214348] 22
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8044314961620841
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8067662100662395
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8087511717170007
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8093848491817769
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8091355231047173
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8092732886573508
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8074876173077044
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8054115864744192
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8041572170163643
[0.8044314961620841, 0.8067662100662395, 0.8087511717170007, 0.80938484918177
69, 0.8091355231047173, 0.8092732886573508, 0.8074876173077044, 0.80541158647
44192, 0.8041572170163643] 3
LSTMattn alone:
test loss 0.03530406951904297
top- 10 0.7900533901928264
macro AUC 0.8469555433080055
micro AUC 0.974202738572247
```

```
macro F1 0.253806770132122
micro F1 0.6438496873279481
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+LSTMattn:
test loss 0.03251442313194275
top- 10 0.804811802206705
macro AUC 0.8848782941414216
micro AUC 0.9778077616901655
macro F1 0.3016705252258659
micro F1 0.6616050763935742
```

Type *Markdown* and LaTeX: α^2