In [1]:
```
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 [2]:
```
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()
```

In [3]:
```
wikisize=newwikivec.shape[0]
rvocsize=newwikivec.shape[1]
wikivec=autograd.Variable(torch.FloatTensor(newwikivec))
```

In [4]:
```
batchsize=32
```

In [5]:

```
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:11: 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
  # This is added back by InteractiveShellApp.init_path()
```

Create the model:

In [6]:

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

    def __init__(self, batch_size, vocab_size, tagset_size):
        super(CAML, 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.convs1 = nn.Conv1d(Embeddingsize,300,10,padding=5)
        self.H=nn.Linear(300, tagset_size )
        self.final = nn.Linear(300, tagset_size)

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


        thisembeddings=self.tanh(self.convs1(thisembeddings).transpose(1,2))

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

        m=alpha.matmul(thisembeddings)

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

        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)


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

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


        return tag_scores
```

In [7]:

```
topk=10
```

In [8]:

```
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          modelperform.append(np.mean(recall))
59          if modelperform[-1]>bestresults:
60              bestresults=modelperform[-1]
61              bestiter=len(modelperform)-1
62
63          if (len(modelperform)-bestiter)>5:
64              print (modelperform,bestiter)
65              return modelsaved[bestiter]
```

In [9]:
```
1 model = CAML(batchsize, len(word_to_ix), len(label_to_ix))
2 model.cuda()
```

Out[9]:
```
CAML(
  (word_embeddings): Embedding(47961, 100, padding_idx=0)
  (embed_drop): Dropout(p=0.2, inplace=False)
  (convs1): Conv1d(100, 300, kernel_size=(10,), stride=(1,), padding=(5,))
  (H): Linear(in_features=300, out_features=344, bias=True)
  (final): Linear(in_features=300, out_features=344, bias=True)
  (layer2): Linear(in_features=100, out_features=1, bias=True)
  (embedding): Linear(in_features=12173, out_features=100, bias=False)
  (vattention): Linear(in_features=100, out_features=100, bias=True)
  (sigmoid): Sigmoid()
  (tanh): Tanh()
  (dropout): Dropout(p=0.2, inplace=False)
)
```

In [10]:
```
1 loss_function = nn.BCELoss()
2 optimizer = optim.Adam(model.parameters())
```

In [11]:
```
basemodel= trainmodel(model, 0)
torch.save(basemodel, 'CAML_model')
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6028159410819657
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7314154270403253
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7724270743774396
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7882633693612231
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7974735633097602
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8054670879999204
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8080145665457611
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8089998811603987
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8116785014218055
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8111941689491625
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8129467900513655
11
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8127608618111046
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8126343213339993
13
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8113614322259091
14
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8121865293281534
15
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8125544505383168
[0.6028159410819657, 0.7314154270403253, 0.7724270743774396, 0.788263369361223
1, 0.7974735633097602, 0.8054670879999204, 0.8080145665457611, 0.80899988116039
87, 0.8116785014218055, 0.8111941689491625, 0.8129467900513655, 0.8127608618111
046, 0.8126343213339993, 0.8113614322259091, 0.8121865293281534, 0.812554450538
3168] 10
```

In [12]:
```
model = CAML(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_CAML_model')
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.815991490746274
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8178064030108383
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.820040526535236
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8200844425246018
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8206450266622128
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8189880367972983
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8187936850792691
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8158021972977897
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8129353409313335
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.8130162588698469
[0.815991490746274, 0.8178064030108383, 0.820040526535236, 0.8200844425246018,
0.8206450266622128, 0.8189880367972983, 0.8187936850792691, 0.8158021972977897,
0.8129353409313335, 0.8130162588698469] 4
```

In [13]:

```
def testmodel(modelstate, sim):
    model = CAML(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:

        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')  )
```

In [14]:

```
print ('CAML alone:           ')
testmodel(basemodel, 0)
print ('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
print ('KSI+CAML:           ')
testmodel(KSImodel, 1)
```

```
CAML alone:
test loss 0.03200652822852135
top- 10 0.8082940181977147
macro AUC 0.8528814144184469
micro AUC 0.9780247595956819
macro F1 0.2661915812890757
micro F1 0.6589893901367232
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+CAML:
test loss 0.031134463846683502
top- 10 0.8176190394571022
macro AUC 0.8884265849082379
micro AUC 0.9805001312980839
macro F1 0.2999242538289451
micro F1 0.6662273234847237
```

In [ ]:

```

```