In [1]:
```
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
import sys
from utils import preprocessing #using the same preprocessing method from ht
```

In [2]:
```
# Authors: Haocheng Zhang and Kehang (Fred) Chang
# portion of codes came from authors in https://github.com/tiantiantu/KSI
```

In [3]:
```
# !pip install numpy --upgrade
print(np.__version__)
```

```
1.19.5
```

In [4]:
```
# modify the default parameters of np.load
np_load_old = np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)
```

In [5]:
```
# choose CPU if GPU is not available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

In [6]:
```
# For consistency, import the data like other modals.
label_to_ix=np.load('label_to_ix.npy').item()
ix_to_label=np.load('ix_to_label.npy')
training_data=np.load('training_data.npy')
test_data=np.load('test_data.npy')
val_data=np.load('val_data.npy')
word_to_ix=np.load('word_to_ix.npy').item()
ix_to_word=np.load('ix_to_word.npy')
newwikivec=np.load('newwikivec.npy')
wikivoc=np.load('wikivoc.npy').item()
```

In [7]:
```
#init global vars
wikisize=newwikivec.shape[0]
rvocsize=newwikivec.shape[1]
wikivec=autograd.Variable(torch.FloatTensor(newwikivec))
```

In [8]:
```
# Use the same hyper params
batchsize=32
Embeddingsize=100
topk=10
padding_idx=0
lr=0.0005  #updated lr
epochs=1000
dropout=0.2
hidden_dim=200
min_good_models=5
```

In [9]:
```
# Use the same preprocessing methods to get training, test and val dataset
batchtraining_data=preprocessing(training_data, label_to_ix, word_to_ix, wik
batchtest_data=preprocessing(test_data, label_to_ix, word_to_ix, wikivoc, ba
batchval_data=preprocessing(val_data, label_to_ix, word_to_ix, wikivoc, batc
```

```
/home/hzhan147/utils.py:18: VisibleDeprecationWarning: Creating an ndarray from
ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarray
s with different lengths or shapes) is deprecated. If you meant to do this, you
must specify 'dtype=object' when creating the ndarray
  new_data=np.array(new_data)
```

Create the model:

In [10]:

```
class RNN(nn.Module):

    def __init__(self, batch_size, vocab_size, tagset_size, padding_idx=0):
        super(RNN, self).__init__()
        self.hidden_dim = hidden_dim
        self.word_embeddings = nn.Embedding(vocab_size+1, Embeddingsize, pad
        self.rnn = nn.GRU(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.sigmoid = nn.Sigmoid()
        self.tanh = nn.Tanh()
        self.embed_drop = nn.Dropout(p=dropout)

    #init hidden layers and encapsulate it to a method, so that we can re-in
    def init_hidden(self):
        return 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)

        #to match what authors' research, we use the SAME KSI algo.
        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)

        #Super simple RNN architecture: Sigmoid -> Linear -> MaxPool1d -> ta
        rnn_out, self.hidden = self.rnn(thisembeddings, self.hidden)
        rnn_out = self.tanh(rnn_out)
        rnn_out=rnn_out.transpose(0,2).transpose(0,1)
        output1=nn.MaxPool1d(rnn_out.size()[2])(rnn_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
```

In [11]:

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


    bestresults=-1
    bestiter=-1
    for epoch in range(epochs):

        model.train()

        lossestrain = []
        recall=[]
        for mysentence in batchtraining_data:
            model.zero_grad()
            #re-init hidden layers on each train
            model.hidden = model.init_hidden()
            targets = mysentence[2].cuda()
            # train model
            tag_scores = model(mysentence[0].cuda(),mysentence[1].cuda(),wik
            # calc loss
            loss = loss_function(tag_scores, targets)
            # backprob
            loss.backward()
            # update params
            optimizer.step()
            # record loss for later calc
            lossestrain.append(loss.data.mean())
        print (epoch)

        # save model since we are tracking model improvements... If no impro
        modelsaved.append(copy.deepcopy(model.state_dict()))
        print ("XXXXXXXXXXXXXXXXXXXXXXXXXXXX")
        model.eval()

        recall=[]
        for inputs in batchval_data:
            #re-init hidden layers on each eval
            model.hidden = model.init_hidden()
            targets = inputs[2].cuda()
            # eval model
            tag_scores = model(inputs[0].cuda(),inputs[1].cuda() ,wikivec.cu

            #calc loss
            loss = loss_function(tag_scores, targets)

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

            #calc recall based on top-K scores
            for idx in range(0,len(tag_scores)):
                temp={}
                for score_idx in range(0,len(tag_scores[idx])):
                    temp[score_idx]=tag_scores[idx][score_idx]
                temp1=[(k, temp[k]) for k in sorted(temp, key=temp.get, reve
```

```
                thistop=int(np.sum(targets[idx]))
                hit=0.0
                for ii in temp1[0:max(thistop,topk)]:
                    if targets[idx][ii[0]]==1.0:
                        hit=hit+1
                if thistop!=0:
                    recall.append(hit/thistop)

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


        #track model performances here based on recalls mean.
        #if current one is better, update best recalls mean and set best idx
        modelperform.append(np.mean(recall))
        if modelperform[-1]>bestresults:
            bestresults=modelperform[-1]
            bestiter=len(modelperform)-1

        #use the best idx (bestiter) to track if we have minimum models afte
        if (len(modelperform)-bestiter)>min_good_models:
            print (modelperform,bestiter)
            return modelsaved[bestiter]
        else:
            print('Not enough min models, keep training...')
```

In [12]:

```
def testmodel(modelstate, sim):
    #--------reload model static params-----#
    model = RNN(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:
        #re-init hidden layers on each test
        model.hidden = model.init_hidden()
        targets = inputs[2].cuda()

        #test model
        tag_scores = model(inputs[0].cuda(),inputs[1].cuda() ,wikivec.cuda()
        #calc loss
        loss = loss_function(tag_scores, targets)

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

        #tracking loss
        lossestest.append(loss.data.mean())
        y_true.append(targets)
        y_scores.append(tag_scores)

        #calc recall based on top-K scores
        for idx in range(0,len(tag_scores)):
            temp={}
            for score_idx in range(0,len(tag_scores[idx])):
                temp[score_idx]=tag_scores[idx][score_idx]
            temp1=[(k, temp[k]) for k in sorted(temp, key=temp.get, reverse=
            thistop=int(np.sum(targets[idx]))
            hit=0.0
            for ii in temp1[0:max(thistop,topk)]:
                if targets[idx][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=[]

    #prepare trues and scores for later performance calc
    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

    #extract predictions
    y_pred=(y_scores>0.5).astype(np.int)

    #print all the metrics
    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 [13]:
```
# START all the training here
model = RNN(batchsize, len(word_to_ix), len(label_to_ix), padding_idx)
model.cuda()

#use BCE loss as loss function
loss_function = nn.BCELoss()
#use Adam optimizer with lr
optimizer = optim.Adam(model.parameters(), lr=lr)
#train model with mode 0 (base RNN)
basemodel= trainmodel(model, 0)
#save base RNN model as file named 'RNN_model'
torch.save(basemodel, 'RNN_model')
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4483715580570907
Not enough min models, keep training...
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4458119639700801
Not enough min models, keep training...
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4460457586480414
Not enough min models, keep training...
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4614050691514662
Not enough min models, keep training...
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4819205638533535
Not enough min models, keep training...
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.509091708262604
Not enough min models, keep training...
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5496401117183958
Not enough min models, keep training...
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5684018860349116
Not enough min models, keep training...
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5935941706047034
Not enough min models, keep training...
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6147655928879384
Not enough min models, keep training...
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6287266606850695
```

```
Not enough min models, keep training...
11
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6421605161182421
Not enough min models, keep training...
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6511728496271545
Not enough min models, keep training...
13
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6559660162744072
Not enough min models, keep training...
14
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6696371134358617
Not enough min models, keep training...
15
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6776025417453773
Not enough min models, keep training...
16
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6892413450700745
Not enough min models, keep training...
17
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6958064249336613
Not enough min models, keep training...
18
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7067978244862239
Not enough min models, keep training...
19
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.711381906679991
Not enough min models, keep training...
20
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7182009251901419
Not enough min models, keep training...
21
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7218711507846677
Not enough min models, keep training...
22
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7265431833859578
Not enough min models, keep training...
23
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7299585941977802
Not enough min models, keep training...
24
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7337447628353596
Not enough min models, keep training...
```

```
25
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.737012946043798
Not enough min models, keep training...
26
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7403975053498824
Not enough min models, keep training...
27
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7412651047783829
Not enough min models, keep training...
28
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7455333014288898
Not enough min models, keep training...
29
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7483247389264142
Not enough min models, keep training...
30
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7460568998036989
Not enough min models, keep training...
31
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7485800345636758
Not enough min models, keep training...
32
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7502302323167985
Not enough min models, keep training...
33
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7491145505224464
Not enough min models, keep training...
34
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7500042913734006
Not enough min models, keep training...
35
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7521250214736818
Not enough min models, keep training...
36
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7517171296634486
Not enough min models, keep training...
37
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7523244623725422
Not enough min models, keep training...
38
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.751265716624483
Not enough min models, keep training...
39
```

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7550207085473756
Not enough min models, keep training...
40
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7515034227957493
Not enough min models, keep training...
41
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7545105804504323
Not enough min models, keep training...
42
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.754942342162871
Not enough min models, keep training...
43
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7571533495483376
Not enough min models, keep training...
44
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7545641884107579
Not enough min models, keep training...
45
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7537581912324638
Not enough min models, keep training...
46
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7565219518309251
Not enough min models, keep training...
47
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7579280871563207
Not enough min models, keep training...
48
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7564415563385456
Not enough min models, keep training...
49
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7583020416963663
Not enough min models, keep training...
50
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7584645431821616
Not enough min models, keep training...
51
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7554702867636138
Not enough min models, keep training...
52
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7549601297807188
Not enough min models, keep training...
53
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
validation top- 10 0.7593632777569843
Not enough min models, keep training...
54
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7576188492770408
Not enough min models, keep training...
55
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7556951557374955
Not enough min models, keep training...
56
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7592367335558762
Not enough min models, keep training...
57
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7557997759379115
Not enough min models, keep training...
58
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7590446353736293
[0.4483715580570907, 0.4458119639700801, 0.4460457586480414, 0.461405069151466
2, 0.4819205638533535, 0.509091708262604, 0.5496401117183958, 0.568401886034911
6, 0.5935941706047034, 0.6147655928879384, 0.6287266606850695, 0.64216051611824
21, 0.6511728496271545, 0.6559660162744072, 0.6696371134358617, 0.6776025417453
773, 0.6892413450700745, 0.6958064249336613, 0.7067978244862239, 0.711381906679
991, 0.7182009251901419, 0.7218711507846677, 0.7265431833859578, 0.729958594197
7802, 0.7337447628353596, 0.737012946043798, 0.7403975053498824, 0.741265104778
3829, 0.7455333014288898, 0.7483247389264142, 0.7460568998036989, 0.74858003456
36758, 0.7502302323167985, 0.7491145505224464, 0.7500042913734006, 0.7521250214
736818, 0.7517171296634486, 0.7523244623725422, 0.751265716624483, 0.7550207085
473756, 0.7515034227957493, 0.7545105804504323, 0.754942342162871, 0.7571533495
483376, 0.7545641884107579, 0.7537581912324638, 0.7565219518309251, 0.757928087
1563207, 0.7564415563385456, 0.7583020416963663, 0.7584645431821616, 0.75547028
67636138, 0.7549601297807188, 0.7593632777569843, 0.7576188492770408, 0.7556951
557374955, 0.7592367335558762, 0.7557997759379115, 0.7590446353736293] 53
```

In [14]:
```
#START all the KSI training here
model = RNN(batchsize, len(word_to_ix), len(label_to_ix), padding_idx)
model.cuda()
model.load_state_dict(basemodel)

#use BCE loss as loss function
loss_function = nn.BCELoss()
#use Adam optimizer with lr
optimizer = optim.Adam(model.parameters(), lr=lr)
#train model with mode 1 (KSI RNN)
KSImodel= trainmodel(model, 1)
#save KSI RNN model as file named 'KSI_RNN_model'
torch.save(KSImodel, 'KSI_RNN_model')
```

```
start_training
0
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7823335916239801
Not enough min models, keep training...
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7880826539572561
Not enough min models, keep training...
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7900280329816457
Not enough min models, keep training...
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7923945783389855
Not enough min models, keep training...
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7917662909436379
Not enough min models, keep training...
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7914105819027177
Not enough min models, keep training...
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7910124287028835
Not enough min models, keep training...
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7915992793147493
Not enough min models, keep training...
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7909345023266783
[0.7823335916239801, 0.7880826539572561, 0.7900280329816457, 0.792394578338985
5, 0.7917662909436379, 0.7914105819027177, 0.7910124287028835, 0.79159927931474
93, 0.7909345023266783] 3
```

In [15]:
```
#print separater between two models' performances for better readability
print ('RNN alone:           ')
testmodel(basemodel, 0)
print ('XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX')
print ('KSI+RNN:           ')
testmodel(KSImodel, 1)
```

```
RNN alone:
test loss 0.03485935553908348
top- 10 0.7581499152853416
macro AUC 0.8502987703322176
micro AUC 0.9680343408219535
macro F1 0.1896937545034506
micro F1 0.6428402769251111
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+RNN:
test loss 0.03179658204317093
top- 10 0.7904457823561494
macro AUC 0.8904576854381114
micro AUC 0.9756648154017689
macro F1 0.2532724014682206
micro F1 0.6597866178648579
```

In [ ]:
```

```