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=0.01  #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 0.40101608663973637
Not enough min models, keep training...
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.38618976115425224
Not enough min models, keep training...
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.38970947758109376
Not enough min models, keep training...
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4504764719365307
Not enough min models, keep training...
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4564919365005066
Not enough min models, keep training...
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4660167262287978
Not enough min models, keep training...
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.46967451319739933
Not enough min models, keep training...
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.455825257555359
Not enough min models, keep training...
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4620362717945726
Not enough min models, keep training...
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.46059115258629557
Not enough min models, keep training...
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.47923415371216876
```

```
Not enough min models, keep training...
11
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.47959522091145806
Not enough min models, keep training...
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4516464749271538
Not enough min models, keep training...
13
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5100073226221283
Not enough min models, keep training...
14
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5121169906786013
Not enough min models, keep training...
15
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5237021301347901
Not enough min models, keep training...
16
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5205453972325008
Not enough min models, keep training...
17
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5219308745393592
Not enough min models, keep training...
18
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5286124162093939
Not enough min models, keep training...
19
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5241472545949427
Not enough min models, keep training...
20
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5295816630662274
Not enough min models, keep training...
21
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5341732020650285
Not enough min models, keep training...
22
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5366185058931356
Not enough min models, keep training...
23
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5119711288506761
Not enough min models, keep training...
24
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5162232759963225
Not enough min models, keep training...
```

```
25
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5134678804632867
Not enough min models, keep training...
26
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.523015572021539
Not enough min models, keep training...
27
XXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5020314007431665
[0.40101608663973637, 0.38618976115425224, 0.38970947758109376, 0.4504764719365
307, 0.4564919365005066, 0.4660167262287978, 0.46967451319739933, 0.45582525755
5359, 0.4620362717945726, 0.46059115258629557, 0.47923415371216876, 0.479595220
91145806, 0.4516464749271538, 0.5100073226221283, 0.5121169906786013, 0.5237021
301347901, 0.5205453972325008, 0.5219308745393592, 0.5286124162093939, 0.524147
2545949427, 0.5295816630662274, 0.5341732020650285, 0.5366185058931356, 0.51197
11288506761, 0.5162232759963225, 0.5134678804632867, 0.523015572021539, 0.50203
14007431665] 22
```

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 0.6862041917202336
Not enough min models, keep training...
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7036455748105339
Not enough min models, keep training...
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6926186362212684
Not enough min models, keep training...
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6931495612398194
Not enough min models, keep training...
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6868922438566547
Not enough min models, keep training...
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6939204668987508
Not enough min models, keep training...
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.683987716591683
[0.6862041917202336, 0.7036455748105339, 0.6926186362212684, 0.693149561239819
4, 0.6868922438566547, 0.6939204668987508, 0.683987716591683] 1
```

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 0.05026585981249809
top- 10 0.5301316443299815
macro AUC 0.7298834993966542
micro AUC 0.9351245553636647
macro F1 0.03837643252372856
micro F1 0.28125
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+RNN:
test loss 0.03798191621899605
top- 10 0.6998947962814728
macro AUC 0.8281500377577606
micro AUC 0.9660602134574177
macro F1 0.10300033466534458
micro F1 0.44059065934065933
```

In [ ]:
```

```