In [1]:

```python
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]:

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

In [3]:

```python
# !pip install numpy --upgrade
print(np.__version__)
```

```
1.19.5
```

In [4]:

```python
# 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]:

```python
# 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]:

```python
# 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]:

```python
#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=0.001
epochs=1000
dropout=0.1 #updated
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]:

```python
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]:

```python
# 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 0.40044107547446234
Not enough min models, keep training...
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.42919014598785615
Not enough min models, keep training...
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.4743209454154674
Not enough min models, keep training...
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.535226237786311
Not enough min models, keep training...
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.5805024973197704
Not enough min models, keep training...
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.607624997505133
Not enough min models, keep training...
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6306936542637406
Not enough min models, keep training...
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.649638726964449
Not enough min models, keep training...
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6693156022808536
Not enough min models, keep training...
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6815369441935087
Not enough min models, keep training...
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6926309754596432
```

```
Not enough min models, keep training...
11
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.6977574277705517
Not enough min models, keep training...
12
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7100074470779246
Not enough min models, keep training...
13
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7119550790707277
Not enough min models, keep training...
14
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7133872755894498
Not enough min models, keep training...
15
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7179919667329451
Not enough min models, keep training...
16
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7200369831843685
Not enough min models, keep training...
17
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7192210476801201
Not enough min models, keep training...
18
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7216623266976079
Not enough min models, keep training...
19
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7245716384319516
Not enough min models, keep training...
20
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.727309263533258
Not enough min models, keep training...
21
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7194662485334489
Not enough min models, keep training...
22
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7291571774208957
Not enough min models, keep training...
23
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7329163421946955
Not enough min models, keep training...
24
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7346017637887728
Not enough min models, keep training...
```

```
25
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7300723600342148
Not enough min models, keep training...
26
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7303579988301966
Not enough min models, keep training...
27
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7373464664227899
Not enough min models, keep training...
28
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.733006077290171
Not enough min models, keep training...
29
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.735341360040057
Not enough min models, keep training...
30
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7305524525250401
Not enough min models, keep training...
31
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7361299332332184
Not enough min models, keep training...
32
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7307644079930052
[0.40044107547446234, 0.42919014598785615, 0.4743209454154674, 0.53522623778631
1, 0.5805024973197704, 0.607624997505133, 0.6306936542637406, 0.64963872696444
9, 0.6693156022808536, 0.6815369441935087, 0.6926309754596432, 0.69775742777055
17, 0.7100074470779246, 0.7119550790707277, 0.7133872755894498, 0.7179919667329
451, 0.7200369831843685, 0.7192210476801201, 0.7216623266976079, 0.724571638431
9516, 0.727309263533258, 0.7194662485334489, 0.7291571774208957, 0.732916342194
6955, 0.7346017637887728, 0.7300723600342148, 0.7303579988301966, 0.73734646642
27899, 0.733006077290171, 0.735341360040057, 0.7305524525250401, 0.736129933233
2184, 0.7307644079930052] 27
```

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 0.7652764727339968
Not enough min models, keep training...
1
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7714725229292095
Not enough min models, keep training...
2
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7741935342128585
Not enough min models, keep training...
3
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7751329956586238
Not enough min models, keep training...
4
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7757373970384904
Not enough min models, keep training...
5
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.776089161543747
Not enough min models, keep training...
6
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7746012516482926
Not enough min models, keep training...
7
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7730622041216525
Not enough min models, keep training...
8
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7696481578963932
Not enough min models, keep training...
9
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
validation top- 10 0.7686291212885032
Not enough min models, keep training...
10
XXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

```
validation top- 10 0.7664581430263441
[0.7652764727339968, 0.7714725229292095, 0.7741935342128585, 0.775132995658623
8, 0.7757373970384904, 0.776089161543747, 0.7746012516482926, 0.773062204121652
5, 0.7696481578963932, 0.7686291212885032, 0.7664581430263441] 5
```

In [15]:

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

```
RNN alone:
test loss 0.03665578365325928
top- 10 0.732631782239517
macro AUC 0.8330420034615779
micro AUC 0.9628159803929023
macro F1 0.17132438023078903
micro F1 0.624286659697112
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
KSI+RNN:
test loss 0.03380465880036354
top- 10 0.7729047613463076
macro AUC 0.878538786378303
micro AUC 0.9730488440860957
macro F1 0.2426155271781495
micro F1 0.6432261986171264
```

In [ ]:

```

```