INDIVIDUAL LAB REPORT 3
Progress Review
MRSD Project

Name: Keerthana P G
Andrew ID: kgopalak

Team D

27th October 2017

Teamwork

Ritwik, Luxing: They linked Resnet, LSTM and speechnet, and debugged by training to see if loss decreases
Luka: speech and text pre-processing
Keerthana: Text parsing and embedding, bidirectional LSTM for text encoding, dual attention network for speech and text

Individual Responsibility

I was responsible for generating word vectors for verbal modality from speech passed as strings. I also implemented a bidirectional LSTM for text encoding and a dual attention network for speech and text modalities.

Text-preprocessing

The dialogues of a user between time intervals are parsed as strings from Luka's function and passed into the wordembed.py function. This function first parses through the sentence using regular expressions, separating words and removing punctuations, digits, trash characters, etc. It then uses Google's pretrained word2vec[1] model to generate vector embeddings for this array of words. Word vectors thus generated are real-valued representations of words in 300-D space such that words that share semantic relatedness in the corpus are located in close proximity to one another in this vector space. These real values may even permit arithmetic operations on words, such as, king-man+woman=queen. The word2vec model that we are using is pre-trained on Google News corpus.

Stage of Implementation
Working fine. Descriptions of actions given in parenthesis have to be removed.

Example output of code:

```
Roses are red and violets are blue
['Roses', 'red', 'violets', 'blue']
[[-0.3203125   0.18847656 -0.33007812 ..., -0.18359375  0.08544922
   0.32226562]
 [ 0.09716797 -0.08496094  0.27148438 ...,  0.04614258  0.14746094
   0.14355469]
 [ 0.05151367  0.31835938  0.03466797 ..., -0.02233887  0.2890625
   0.1953125 ]
 [ 0.0390625   0.08642578  0.22363281 ...,  0.04663086  0.02258301
  -0.15722656]]
```

Bidirectional-LSTM for Text Encoding

For text processing, we have decided to use a bidirectional LSTM as detailed in [2]

Given word embedding of T input words $\{w_1, \cdots, w_T\}$, the vectors are fed into the bi-LSTM.

$h(f)_t = LSTM(f)(x_t, h(f)_{t-1})$
$h(b)_t = LSTM(b)(x_t, h(b)_{t+1})$,
where $h(f)_t$ and $h(b)_t$ represent the hidden states at time t from the forward and backward LSTMs, respectively.

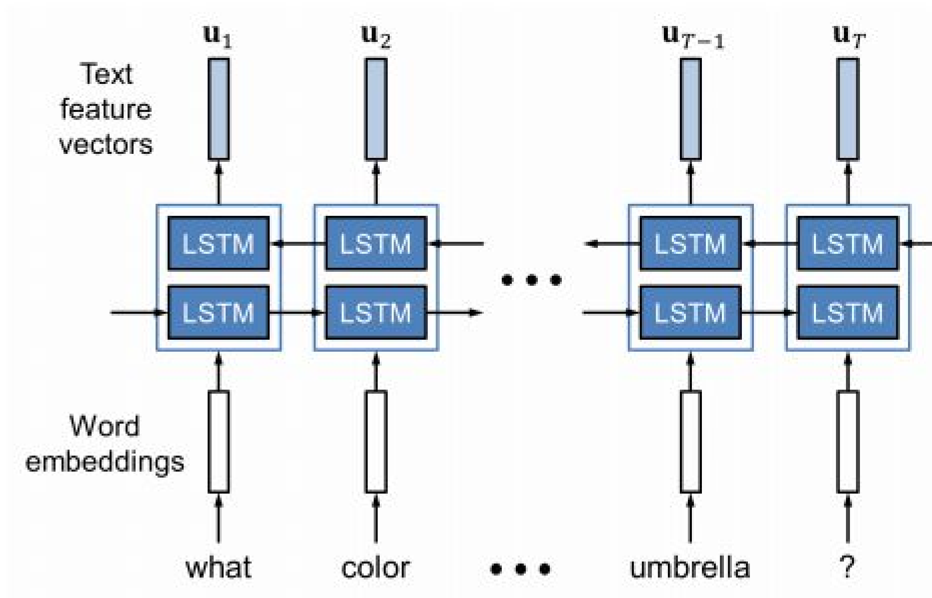

Fig1: Bidirectional LSTM for text encoding. Source:[2]

Then we add the two hidden states at each time step
$u_t = h(f)_t + h(b)_t$
To construct a set of feature vectors $\{u_1, \cdots, u_T\}$ where u_t encodes the semantics of the t-th word in the context of the entire sentence. These context vectors are then passed to the dual attention network.

Stage of Implementation

Currently, we've coded the layers for the bi-LSTM. However, it has to be linked with wordembed.py and attention.py and training modules have to be written. We have not finalised the number of hidden layers.

Dual Attention Network

The dual attention network measures semantic similarity between vocal and verbal inputs by learning a joint space where the two feature vectors from the two modes are directly comparable. This will help the network converge faster by encoding shared concepts that co-occur together. Additionally, the text and vocal vectors aren't paired while training, while maintaining separate memory vectors, making it possible to directly compare arbitrary verbal and vocal vectors. In our implementation, cross-modal similarity is calculated as an inner product of the context vectors after each layer of training.

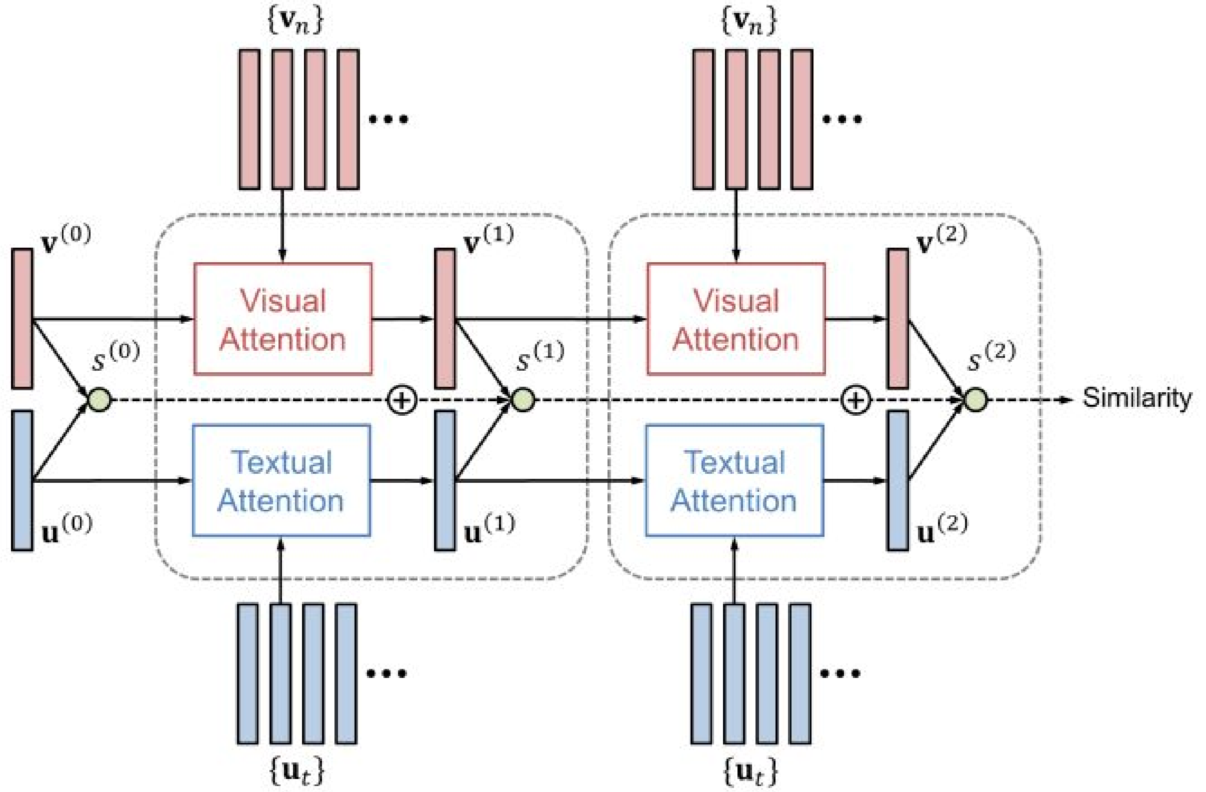

Fig 2: Dual attention network Source: modified from [2]

Unimodal attention:

Here, the context vectors from each model is acted upon by a two-layer feed forward neural network to obtain attention weights by focusing on specific parts of the input at each time timestep.

$u^{(k)} = \text{UnimodalAtt}(\{u_t\}^T_{t=1}, m^{(k-1)}_u)$,

where $m^{(k-1)}_u$ is a memory vector, t is time step, and k is layer number.

The architecture of each layer of the network is as follows:

$h^{(k)}_{u,t} = \tanh(W^{(k)}_u u_t) \odot \tanh(W^{(k)}_{u,m} m^{(k-1)}_u)$

$\alpha^{(k)}_{u,t} = \text{softmax}(W^{(k)}_{u,h} h^{(k)}_{u,t})$

$u(k) = \sum_t \alpha^{(k)}_{u,t} u_t$.

Here, $W^{(k)}_u$, $W^{(k)}_{u,m}$ and $W^{(k)}_{u,h}$ are the network parameters and $h^{(k)}_{u,t}$ is a hidden state.

The above network is replicated for the other mode and trained parallelly. After each layer, the two context vectors outputted by the layers are used to find the similarity:

$s^{(k)} = v^{(k)} \odot u^{(k)}$

After performing K(here K=2) steps of the dual attention and memory update, the final similarity S between the given vocal vector and sentence is calculated as:
$S = \sum_{k=0}^{K} s^{(k)}$

The loss function for this network is a bidirectional max-margin ranking loss. To this end, we sample an additional negative vocal vector v − and negative verbal vector u − for each correct pair of vocal vector and verbal vector (v, u), to construct two negative pairs (v −, u) and (v, u −). Then, the loss function is evaluated as:
$L = \sum_{(v,u)} ma(0, m - S(v, u) + S(v -, u)) + max(0, m - S(v, u) + S(v, u -))$
where m is a margin constraint(here initialised to 100).
By minimizing this function, the network is trained to focus on the common semantics that only appears in correct text-vocal pairs.

Stage of Implementation and Challenge
All parts of this network except one have been tested on random initial values. The remaining one is the max-margin loss function, which, since we are defining on our own rather than using inbuilt loss functions, gives a technical error in the pytorch framework. This has to be ironed out.

Challenges

1. Pytorch framework gives an error due to manual specification and calculation of loss function in attention network, as explained above. We plan to debug the code to remove this.
2. We still have to figure out how to factor in similarity of text and vocal in the larger framework of multimodal learning. For this, the attention net will have to be properly linked with the LSTM at the end.
3. The script includes verbal descriptions of feelings and actions of the speaker apart from what they actually speak. This will have to be weeded out lest they interfere with training
4. After designing the hardware in CAD, Luka figured that we had not previously considered how heavy the camera actually is, due to which we will have to shift to heavire and sturdier motors and links than previously planned. After finalising part models and specifications, coherent with design and mechanics, we will place the order.

Scope of Improvement

1. Currently, the wordembed.py implementation ignores words not found in vocabulary. An improvement can be achieved if we associate a small prior to these words. The details of how to do it have to be figured out. However, this may entail extensive post-training.

Future Plans

1. Linking and training of dual attention networks
2. Implementation of tri-modal network with above attention models incorporated.
3. Drawing conclusions from training of bi-modal network

References

1. Mikolov T, Chen K, Corrado G, Dean J, “Efficient Estimation of Word Representations in Vector Space”, arXiv:1301.3781, https://arxiv.org/pdf/1301.3781.pdf
2. Nam H, Ha JW, Kim j, “Dual Attention Networks for Multimodal Reasoning and Matching”, arXiv:1611.00471, https://arxiv.org/abs/1611.00471

Codes

Attention.py

```
#this will have text processing, a bi-LSTM for text representation
import torch.nn as nn
from torch.autograd import Variable
import torch
import torchvision.models as models

#hyperparameters
u_verbal_size=1280
u_vocal_size=1280


class verbalattention(nn.Module):
    def __init__(self,u_verbal_size):
        super(verbalattention, self).__init__()
        self.Wu = nn.Linear(in_features=u_verbal_size, out_features=u_verbal_size)#512 is the
number of verbal vectors which is input size
        self.Wum = nn.Linear(in_features=u_verbal_size, out_features=u_verbal_size) #1792 is
the size of concatenated
        self.Wuh = nn.Linear(in_features=u_verbal_size, out_features=u_verbal_size)
        self.memory=Variable(torch.zeros(u_verbal_size))


    def forward(self, context):
        self.u=Variable(torch.zeros(u_verbal_size))
        x1=torch.tanh(self.Wu(context))
        x2=torch.tanh(self.Wum(self.memory))
        hut=Variable(torch.zeros(u_verbal_size))
        hut=torch.addcmul(hut,x1,x2)
        m=self.Wuh(hut)
        m=m.view(1,m.size(0))
        t=nn.Softmax()
        m=t(m)
        aut=m.view(-1)
        self.u = torch.addcmul(self.u, 1.0,aut, context)
        return self.u

    def memory_update(context):
        self.memory+=context
```

```
class vocalattention(nn.Module):
    def __init__(self, u_vocal_size):
        super(vocalattention, self).__init__()
        self.Wu = nn.Linear(in_features=u_vocal_size, out_features=u_vocal_size)#512 is the
number of verbal vectors which is input size
        self.Wum = nn.Linear(in_features=u_vocal_size, out_features=u_vocal_size) #1792 is the
size of concatenated
        self.Wuh = nn.Linear(in_features=u_vocal_size, out_features=u_vocal_size)

    def forward(self, context):
        self.memory=Variable(torch.zeros(u_vocal_size))
        self.u=Variable(torch.zeros(u_vocal_size))
        x1=torch.tanh(self.Wu(context))
        x2=torch.tanh(self.Wum(self.memory))
        hut=Variable(torch.zeros(u_vocal_size))
        hut=torch.addcmul(hut,x1,x2)
        m=self.Wuh(hut)
        m=m.view(1,m.size(0))
        t=nn.Softmax()
        m=t(m)
        aut=m.view(-1)
        self.u = torch.addcmul(self.u, 1.0,aut, context)
        return self.u

    def memory_update(context):
        self.memory+=context

def calcloss(S,margin):
    loss=(2*torch.max(Variable(torch.zeros(u_vocal_size)),margin-S))
    print(loss)
    return loss

def attentionnet(context_verbal, context_vocal):
    S=Variable(torch.zeros(u_vocal_size))

    output_verb = verbAtt1(context_verbal)
    output_voc= vocAtt1(context_vocal)
    S = torch.addcmul(S,output_voc, output_verb)

    output_verb = verbAtt2(output_verb)
    output_voc= vocAtt2(output_voc)
    S = torch.addcmul(S,output_voc, output_verb)
```

```
    return(S)


context_vocal = Variable(torch.randn(1280)) #currently size 1280
context_verbal = Variable(torch.randn(1280)) #currently size 512
margin=Variable(torch.FloatTensor([100]))
verbAtt1 = verbalattention(u_verbal_size)
vocAtt1 = vocalattention(u_vocal_size)
verbAtt2 = verbalattention(u_verbal_size)
vocAtt2 = vocalattention(u_vocal_size)
params = list(verbAtt1.parameters()) + list(vocAtt1.parameters())+list(verbAtt2.parameters())
+list(vocAtt2.parameters())
optimizer = torch.optim.Adam(params, 0.01)
`


for epoch in range(10000):
    print('Epoch [%d]'
          % (epoch+1))
    verbAtt1.zero_grad()
    verbAtt2.zero_grad()
    vocAtt1.zero_grad()
    vocAtt2.zero_grad()

    S=attentionnet(context_verbal, context_vocal)
    #loss=calcloss(S,margin)
    loss= 2*torch.max(Variable(torch.zeros(u_vocal_size)),margin-S)

    loss.backward()
    optimizer.step()

    print('Epoch [%d], Loss: %.4f, Perplexity: %5.4f'
          % (epoch, loss.data[0], np.exp(loss.data[0])))
```

Bilstm.py

```
#this will have text processing, a bi-LSTM for text represenation
import torch.nn as nn
import torch.nn.functional as F

class TextNet(nn.Module):
    def __init__(self, batch_size):
        super(TextNet, self).__init__()
        self.bilstm=nn.LSTM(input_size=10, hidden_size=20, num_layers=2, bidirectional=true)
        h0 = Variable(torch.randn(2, 3, 20))
```

```
        c0 = Variable(torch.randn(2, 3, 20))


    def forward(self, textinput):
            output,hn=F.relu(self.bilstm(textinput, (self.h0,self.c0)))
        print(output)
        #x = x.view(1280, -1)
        return output

input = Variable(torch.randn(5, 3, 10))
Textnet.forward(input)
```

Wordembed.py

```
import gensim
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
import re
#nltk.download('stopwords') #run once
#nltk.download('punkt') #run this once on system to download dependency

# Load Google's pre-trained Word2Vec model.
w2v_model =
gensim.models.KeyedVectors.load_word2vec_format('./GoogleNews-vectors-negative300.bin',
binary=True)
tokenizer = RegexpTokenizer(r'\w+')

stop_words = set(stopwords.words('english'))



def wordembed(sentence):
        sentence = re.sub(r'\S*\d\S*', '', sentence).strip() #remove numbers or words containing
numbers
        sentence = re.sub(r'\(\S*\)','', sentence) #removes things within parenthesis
        sentence = re.sub(r'[^A-Za-z ]', '', sentence)#removes non alphabet characters
        sentence = re.sub(r' +',' ',sentence) # removes multiple spaces
        print(sentence)
        tokens = tokenizer.tokenize(sentence)
        filtered_tokens = [w for w in tokens if not w in stop_words and w in
w2v_model.wv.vocab]#remove stop words
```

```
    print(filtered_tokens)
    return(w2v_model.wv[filtered_tokens])


sentence='Angad and Yoga are looking at our project (So excited! 999)'
vectors=wordembed(sentence)
print(vectors)
```