INDIVIDUAL LAB REPORT
Progress Review
MRSD Project

Name: Keerthana P G
Andrew ID: kgopalak

Team D

20th October 2017

Teamwork

Ritwik: Visual data pre-processing and resnet
Luka: speech pre-processing
Keerthana: LSTM and speechnet, PCB design, dataset review
Luxing: Speechnet, Website update

Details

After consulting with Emotech, Prof Jeff Cohn and Prof Louis Philip Morency, and performing literature review on our own, we realised that extracting affective information from text is a hard task in itself and involves a larger delay due to high processing required. Also, there aren't many well supported APIs that convert speech to text reliably without significant delay. Due to this, we decided to start working on bimodal detection using visual and vocal modes to get started. We also edited the requirements to suit this change in focus.

Value Proposition

Upon discussion and literature review we understand that a lot of work has been done on unimodal emotion recognition. To this effect, we'd be focusing on the multimodal part, of combining different modes of input, and enhancing predictions by weighing the importance of and accuracy from different modes.

Dataset Review

After an extensive search for datasets and trade study we decided to go ahead with IEMOCAP, SEMAINE and Microsoft MISC.
Parameters[1] on which datasets were compared involve number and diversity of subjects, size of data, whether they were annotated for emotions, number of raters, dimension of emotions, whether script was available, presence of all three modes, number of subjects in conversation simultaneously, permission to use, etc. Details on trade study performed were submitted in CoDR.

Literature Review

After extensive literature review, we decided to implement a hybrid network for bimodal emotion recognition as detailed in [2] to get started. Once we have a trained model, we'll tune parameters, modify architectures, introduce the third mode and make other changes to improve the overall network and accuracy.

Data Pre-processing

1. Speech: The speech input to the network as detailed in paper required a 16kHz sampling rate and therefore a 96000 long input vector every 150ms. However, both

SEMAINE and MISC has a different input rate. Luka handled this part by selective sampling.

2. On dyadic datasets: Since SEMAINE has more than one subject in conversation, it is required to separate the speech of two different speakers and feed them into network separately.
3. Image: Cropped images, after resizing to input size are fed to the resnet. Ritwik is doing this part using dlib. Some datasets such as MISC only contain coordinates of landmark features and using this data in our network that expects face images would be a challenge.

Network Architecture

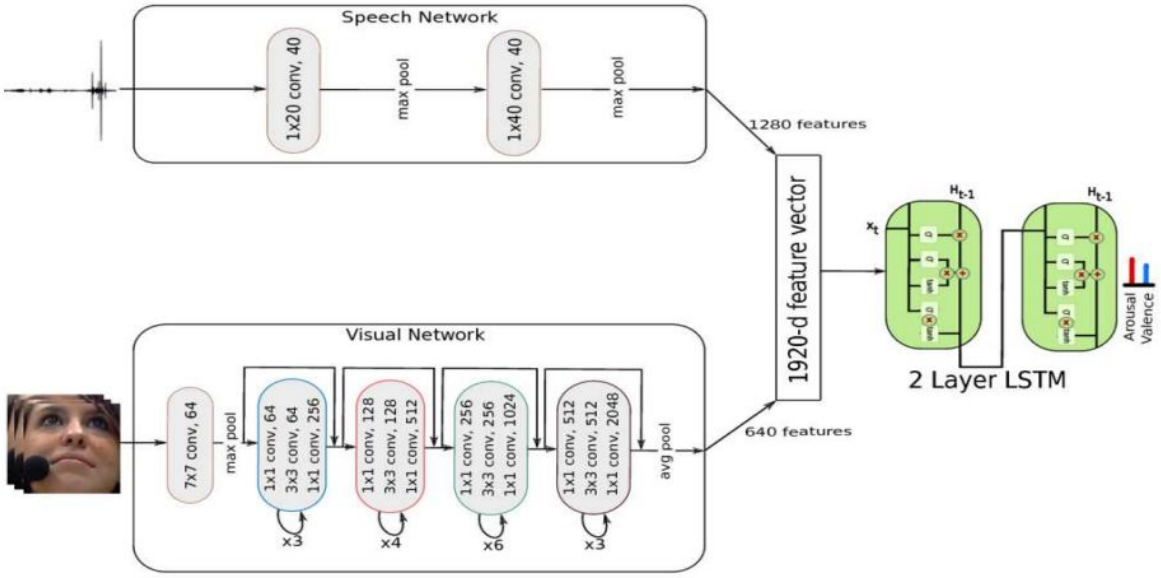

Figure 1: Network Architecture[2]

Individual Responsibility

I was responsible for implementing speechnet(along with Luxing), implementing LSTM and figuring out manipulator mechanism and PCB design for mechanical component.

The codes were made modular by using specific classes and combining all functions of a particular net to the respective class. A main.py file was introduced to interface between different parts of the network and call the modules.

Speechnet

The speech network takes an input vector of length 96000 and has 2 convolutional layers, 2 ReLU layers, 2 maxpool layers and a dropout layer. The first convolution layer has a kernel size of 20 and 40 channels while the second has a kernel size of 40 and 40 channels. Padding and

stride for both are 0 and 1 respectively. Each convolutional layer is followed by a ReLU layer and a maxpool layer. Kernel sizes of the two maxpool layers are 2 and 10 respectively. At the end, the dropout layer acts with a probability of 0.5. The output of the speechnet is feature vector of length 1280 that is concatenated with the output of the ResNet and fed to the LSTM.

I wrote the layers using torch.nn and integrated them with Luxing's class based modular structure.

Multimodal LSTM

The LSTM network has 2 layers with a hidden size of 250 each. The internal workings of an LSTM are summarised the following equations[3]

Input gate : $i_t = \sigma(W^{(i)}x_t+U^{(i)}h_{t-1}+b^{(i)})$
Forget gate : $f_t = \sigma(W^{(f)}x_t+U^{(f)}h_{t-1}+b^{(f)})$
Output gate: $f_t = \sigma(W^{(o)}x_t+U^{(o)}h_{t-1}+b^{(o)})$
Update gate: $u_t = \sigma(W^{(u)}x_t+U^{(u)}h_{t-1}+b^{(u)})$
Memory cell: $c_t = i_t \bullet u_t + f_t \bullet c_{t-1}$
Hidden cell: $h_t = o_t \bullet \tanh(c_t)$
Here t is step number.
I implemented the LSTM layers using torch.nn.

Mechanism and PCB design

Our objective was to design a manipulator mechanism that can track and align itself with the face of the user. The details of design and rationale behind selecting components have been detailed in the PCB conceptual design submitted yesterday. I was single handedly responsible for this part.

Challenges

1. Communicating with the entire team, making sure everybody is on the same page, peacefully resolving conflicts, working with people having different technical and language expertise and time commitments(owing to midterms), dealing with unhealthy team playing attitudes, distributing and compartmentalising work has been our biggest challenge so far. We are still working on it.

Things to ponder

1. Currently we are using a ReLU nonlinearity after every convolution layer in the speech network in an attempt to make the generalisation function non linear. However, since a number of input values are negative, the ReLU nonlinearity ignores them and they do not contribute to variations in the feature vector. This may not be an optimal situation as we may lose valuable information. We will discuss with mentors on if this is a good idea, whether we can get rid of the nonlinear layer or we should use some other nonlinear function like sigmoid that acknowledges negative values.

2. Upon discussion with Emotech we understand that feeding pixel values than landmark features gives better results as the network is able to capture subtleties in the image that contribute to affective expression, information that would be passed over if the network is only fed landmarks. However, currently, the MISC dataset we possess has only coordinates of landmarks and procuring full videos require further permissions. The ways to factor in the variations in data available across datasets is something we should consider.

Future Plans

1. Implement backpropagation in LSTM and speechnet.
2. Explore the use of attention models[4]
3. Connect output of speechnet and resnet to input of LSTM
4. Ensure that speech/image pre processing are coherent to the input requirements
5. Perform preliminary training
6. Procure hardware for mechanism design

References

1. http://emotion-research.net/wiki/Databases
2. https://arxiv.org/abs/1704.08619
3. https://nlp.stanford.edu/pubs/tai-socher-manning-acl2015.pdf
4. https://arxiv.org/abs/1409.0473

Codes

Speechnet.py

```
import torch
import torch.nn as nn
from torch.autograd import Variable
import numpy as np


class SpeechNet(nn.Module):
    def __init__(self):
        super(SpeechNet, self).__init__()

        # first convolution layer
        self.convNet1 = nn.Conv1d(in_channels=1, out_channels=40, kernel_size=20, stride=1,
padding=0, dilation=1, groups=1, bias=True)

        #first pooling layer
        self.pool1 = nn.MaxPool1d(kernel_size=2, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False) #paper states kernel size is 2, rest Keerthana defined

        #second convolution layer
        self.convNet2 = nn.Conv1d(in_channels=40, out_channels=40, kernel_size=40, stride=1,
padding=0, dilation=1, groups=1, bias=True)

        #second pooling layer
        self.pool2 = nn.MaxPool1d(kernel_size=10, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False) #pool size = 10 as given in paper.

        #dropout layer
        self.dropout = nn.Dropout(p=0.5, inplace=False, training=self.training)


    def forward(self, speechinput):
        output=nn.ReLU(self.convNet1(speechinput))#why are we using ReLU non linearity and
not signoid  or something else? wont we lose -ve values
        output=self.pool1(output)
        output=nn.ReLU(self.convNet2(output))
        output=self.pool1(output)
        output = self.dropout(output)
        output = output.view(1, 1280)
```

```
        return output


model = SpeechNet()
input_vector = Variable(torch.Tensor(np.random.randn(1, 1, 96000)))#need to change this.


def train(input):
    model.train()
    output = model(input)
    print(output)

train(input_vector)
```

Multimodallstm.py

```
import torch.nn as nn
from torch.autograd import Variable
import torch

# To be put together with speech CNN inside one Net
input_size = 1280 + 512
hidden_size = 256
seq_len = 150
batch_size = 1
num_layers = 2
rnn = nn.LSTM(input_size=input_size, hidden_size=hidden_size, num_layers=num_layers)
input = Variable(torch.randn(seq_len, batch_size, input_size))
h0 = Variable(torch.randn(2, batch_size, hidden_size))
c0 = Variable(torch.randn(2, batch_size, hidden_size))
output, (hn, cn) = rnn(input, (h0, c0))
# print(output)
# print(cn)
```

Main.py

```
import torch.nn

from __future__ import print_function
import argparse
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable
```

```
# Training settings
parser = argparse.ArgumentParser(description='PyTorch MNIST Example')
parser.add_argument('--batch-size', type=int, default=64, metavar='N',
                    help='input batch size for training (default: 64)')
parser.add_argument('--test-batch-size', type=int, default=1000, metavar='N',
                    help='input batch size for testing (default: 1000)')
parser.add_argument('--epochs', type=int, default=10, metavar='N',
                    help='number of epochs to train (default: 10)')
parser.add_argument('--lr', type=float, default=0.01, metavar='LR',
                    help='learning rate (default: 0.01)')
parser.add_argument('--momentum', type=float, default=0.5, metavar='M',
                    help='SGD momentum (default: 0.5)')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
parser.add_argument('--log-interval', type=int, default=10, metavar='N',
                    help='how many batches to wait before logging training status')
args = parser.parse_args()
args.cuda = not args.no_cuda and torch.cuda.is_available()

torch.manual_seed(args.seed)
if args.cuda:
    torch.cuda.manual_seed(args.seed)

def emotionrec():
        #load input from speech
        #do speech processing
        #call speech network
        model = speechnet()
        if args.cuda:
            model.cuda()
        #parallelly, do image processing
        #after both done, call multimodallstm.py
```