Modeling Dyadic Human Interaction using Sequential Neural Network

Srijan Kumar Sharma
ss5938@rit.edu
Modeling Dyadic Human Interaction using Sequential Neural Network

by

Srijan Kumar Sharma

THESIS
Presented to the Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology

in Partial Fulfillment
of the Requirements
for the Degree of
Master of Science in Computer Science

Rochester Institute of Technology
May 2021
Modeling Dyadic Human Interaction using Sequential Neural Network

APPROVED BY
SUPERVISING COMMITTEE:

Dr. Ifeoma Nwogu, Advisor

Dr. Joe Geigel, Reader

Dr. Christopher Homan, Observer
Abstract

Social networks, involving people and their interactions are at core of human society. But many current computational social methods focus more on the individual than their interactions. Deep neural networks have been successfully applied to tasks such as natural language processing, dialog modeling, or analyzing sentiments in a conversation. In these areas, we will often encounter data that originate from multiple sources. These signals can interact with each other synchronously, but detecting such synchrony may prove challenging.

In this work we focus on investigating how deep neural network architectures can help us better understand synchrony in social contexts. We investigate different coupled sequential models such as an end-to-end connected gated recurrent unit (GRU), an inherently coupled GRU, message-passing, the role of attention and the use of transformer networks for coupling.

We evaluate the effectiveness of our coupling models on multiple datasets. We first test on synthesized sequential coupled data as a sanity-check and then move on to more realistic data. We test our models on three different real-world datasets collected in the context of various social interactions. In two of the datasets, we predict the rapport between two persons based on data extracted from the video of them interacting. In the third dataset, we predict friendship/familiarity between two people based on their interaction. We present the findings from the work and conclude that the coupled transformer network performs the best.
Acknowledgments

I would first like to thank my advisor, Professor Ifeoma Nwogu, for guiding me throughout my thesis and answering all my research questions. Without your support, this project wouldn’t have been an amazing research experience. I would like to thank Dr. Joe Geige and Dr. Christopher Homan for being on my committee and providing valuable feedback, which helped me sharpen this thesis.
# Contents

1 Introduction ................................................................. 1
   1.1 Related Work .......................................................... 2

2 Background ........................................................................ 3
   2.1 Coupled Hidden Markov Model ....................................... 3
   2.2 Recurrent Neural Network ............................................. 4
   2.3 Gated recurrent units .................................................... 5
   2.4 Attention ...................................................................... 6
   2.5 Transformer .................................................................. 8
   2.6 Message Passing Encoder-Decoder - Recurrent Neural Network .. 13
   2.7 Facial Action Coding System .......................................... 15
   2.8 Face Landmarks ............................................................ 17

3 Methodology ...................................................................... 18
   3.1 End to End connected GRU ........................................... 18
   3.2 Coupled Gated Recurrent Unit ....................................... 19
   3.3 Coupled Gated Recurrent Unit with Attention .................... 20
   3.4 Coupled Transformers .................................................. 22
   3.5 Loss Function ................................................................ 25
      3.5.1 Mean Absolute Loss ................................................ 25
      3.5.2 Cross Entropy Loss ................................................... 25

4 Datasets .......................................................................... 26
   4.1 Synthetic Data ............................................................. 26
   4.2 USC Institute for Creative Technology, Rapport Dataset .......... 27
   4.3 University at Buffalo Communication Science Center, Rapport dataset .. 28
   4.4 University at Buffalo Communication Science Center, Friends Dataset ... 29
   4.5 Face Feature extraction ................................................ 29

5 Results ............................................................................ 31
   5.1 Experiment 1: Test Coupling ......................................... 31
   5.2 Experiment 2: Detect Rapport and Compare Architectures .......... 33
      5.2.1 Execution Time Comparison ....................................... 36
   5.3 Experiment 2a: Check Tight Coupling ............................... 37
   5.4 Experiment 3: Detect Friendship ...................................... 38
   5.5 Experiment 4: Find effects of attention ............................. 39
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Hidden Markov Model</td>
<td>3</td>
</tr>
<tr>
<td>2.2</td>
<td>Recurrent Neural network</td>
<td>4</td>
</tr>
<tr>
<td>2.3</td>
<td>GRU Model</td>
<td>5</td>
</tr>
<tr>
<td>2.4</td>
<td>RNN Attention</td>
<td>7</td>
</tr>
<tr>
<td>2.5</td>
<td>Scaled Dot-Product</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>Multi-Head attention</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>Transformer Encoder</td>
<td>11</td>
</tr>
<tr>
<td>2.8</td>
<td>Transformer Decoder</td>
<td>13</td>
</tr>
<tr>
<td>2.9</td>
<td>Message Passing GRU</td>
<td>14</td>
</tr>
<tr>
<td>2.10</td>
<td>Message Passing Encoder-Decoder Recurrent Neural Network</td>
<td>15</td>
</tr>
<tr>
<td>2.11</td>
<td>Landmark Locations</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>End to End Connected GRU</td>
<td>18</td>
</tr>
<tr>
<td>3.2</td>
<td>Coupled GRU Model</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Coupled GRU Model with attention</td>
<td>21</td>
</tr>
<tr>
<td>3.4</td>
<td>Coupled Transformer Model</td>
<td>23</td>
</tr>
<tr>
<td>3.5</td>
<td>Coupled Multi Head Attention</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Example of synthetic data</td>
<td>26</td>
</tr>
<tr>
<td>4.2</td>
<td>Frames from the USC Institute for Creative Technology Rapport Dataset</td>
<td>28</td>
</tr>
<tr>
<td>4.3</td>
<td>Frames from the USC Institute for Creative Technology Rapport Dataset</td>
<td>28</td>
</tr>
<tr>
<td>4.4</td>
<td>Frames from the Friends dataset from two different perspectives</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>Loss plots for Synthetic data</td>
<td>32</td>
</tr>
<tr>
<td>5.2</td>
<td>Predicted vs. Actual correlation factor</td>
<td>33</td>
</tr>
<tr>
<td>5.3</td>
<td>Validation Accuracy and Loss plots</td>
<td>34</td>
</tr>
<tr>
<td>5.4</td>
<td>A comparison of different models</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>Validation Accuracy vs. Epoch plot for Tightly Coupled</td>
<td>37</td>
</tr>
<tr>
<td>5.6</td>
<td>Validation Accuracy vs. Epoch plot for Friends Dataset</td>
<td>38</td>
</tr>
<tr>
<td>5.7</td>
<td>Self-Attention during Non-Rapport</td>
<td>39</td>
</tr>
<tr>
<td>5.8</td>
<td>Coupled-Attention during Non-Rapport</td>
<td>40</td>
</tr>
<tr>
<td>5.9</td>
<td>Self-Attention during Rapport</td>
<td>41</td>
</tr>
<tr>
<td>5.10</td>
<td>Rapport mutual attention</td>
<td>42</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Action units ................................................................. 16

3.1 End to End Connected GRU configuration table ........................................ 19

3.2 Coupled GRU configuration table ................................................... 20

3.3 Coupled GRU w/ attention configuration table ........................................ 22

3.4 Coupled GRU w/ attention configuration table ........................................ 24

5.1 Experiment 1: Hyper-Parameter ..................................................... 31

5.2 Experiment 2: Hyper-Parameter ..................................................... 33

5.3 Validation results for University at Buffalo Communication Science Center, Rapport dataset and transfer performance on USC Institute for Creative Technology Rapport Dataset ........................................ 34

5.4 Timing comparison for different models .......................................... 36

5.5 Experiment 2a: Hyper-Parameter .................................................. 37

5.6 Experiment 3: Hyper-Parameter ..................................................... 38

5.7 Friends Dataset Results ............................................................. 38
Chapter 1

Introduction

Humans communicate information in a variety of modes, linguistic/alphabetic (written and spoken words), visual (images), aural (sound, music), gestural (movement, expression, body language), spatial (position, physical arrangement, proximity). Agreement or Disagreement between two conversing individuals can be ascertained by looking at the gestures they are making. For instance, when there is an agreement between conversing individuals, they mirror each other’s gestural and mannerisms involuntarily. The aforementioned phenomenon of an instinctive synergy of verbal and non-verbal gestures, also called rapport, shows successful conveyance of emotions and ideas.

From a computer science perspective, we can treat the various modes of human communication as multi modal multi source time series data. In our work, we are using face landmarks and action units as input to sequential neural networks. We are treating the presence or absence of rapport as a two class classification problem.

Apart from rapport classification, multi source time series data is ubiquitous in real world. Dyadic interaction is at the heart of human society. And a lot of processing can be done on this type of data such as emotion detection, sentiment classification, etc.. Currently all the architectures that are used to model sequential data can either process a single input source or use a turn based system. In our work we want to broadly target the problem of modeling two interacting sequential data.

Thesis statement:

“Develop a machine learning architecture, based on Transformers, capable of modeling Dyadic Human Interaction, and achieve better performance than traditional Recurrent Neural Network based models.”

Objectives:

• Create a model that can predict if there is rapport between two individuals.

• Develop a model that generalizes well on different datasets, i.e. model trained on one dataset should be able to achieve acceptable accuracy on another dataset.

• Use “attention” to investigate local regions with rapport.

We begin the thesis by discussing the background in Chapter 2 where we discuss key concepts and ideas that we use through the thesis. Then, we discuss the methods, algorithms, and techniques that we used for investigation in Chapter 3. After that, we discuss the datasets that we used to train and test our models in Chapter 4. Finally, we list out the results of our instigation and state our conclusions in Chapters 5 & 6.
1.1 Related Work

Rapport detection in videos does not have state of the art numbers, unlike other Machine Learning Problems. This can be attributed to the lack of publicly available rapport dataset that can be used for benchmark. However, there are few works which are closely related to our work.

The first paper is by Deepanway Ghosal et. al [15] on emotion recognition in conversations. In their paper, they have presented a graph based neural network called Dialogue Graph Convolutional Network (DialogueGCN) [15] to perform emotion recognition in conversations. Their model works in three stages. In the first stage, context encoding using GRUs. In the second stage, speaker context encoding is generated using graph convolution network. Finally, the conversation is classified into an emotion label. Their data is made up of dyadic turn based text dialogues, unlike our data which consists of two simultaneous data streams. They are using three datasets to test their model IEMOCAP [6], AVEC [31] and MELD [25].

The second paper, by Hazarika, Devamanyu et. al [18], performs emotion classification on dyadic conversation videos. They start by dividing the interaction videos into individual utterances. Each utterance has three modes, text, speech and video. The text (taken from transcript) of the utterance the audio and the video parts of the video are passed to three separate models. These three models convert the text, audio and video into feature vectors. The feature vectors are merged and emotion in the conversation is predicted using a final model made up of GRU and attention. To test their models they are using IEMOCAP [6] dataset. Their approach differs from ours in three key ways. First, they are performing emotion classification, whereas we are doing rapport detection which is different from emotion detection. Second, they are performing global emotion detection at the end of conversation, whereas we are detecting rapport in small intervals because rapport takes place as events in time. Lastly, they expect data to have transcript and split into utterances before being input to model. Our model performs detection directly on video frames.

The next paper is by Zhao et. al [35] and explores dyadic rapport between virtual agents. They are using Temporal Interval Tree Association Rule Learning algorithm in their work. Their paper is using rules based framework and doesn’t provide any accuracy scores to make comparisons against. They are using "Rapport in Peer Tutoring” dataset in their paper. Unfortunately this dataset is not publicly available. The same dataset is being used by Hongliang et. al [33]. Hongliang et. al [33] is performing sentiment analysis and global rapport detection using GRU and Temporally Selective Attention Model. We are using a similar model as baseline in our experiments.
Chapter 2

Background

In this chapter we go over the key ideas and concepts. We will start by giving a general overview of the key ideas and concepts we will be making use of throughout the thesis. After that we will discuss all the ideas in detail in individual sections. Each section will contain a detailed explanation of the idea using figures, equations, and formulas. Some of the sections will discuss specific papers as applicable. This chapter will help in understanding the concepts and discussions presented in methodology (Chapter 3), results (Chapter 5) and conclusion (Chapter 6) chapters.

We will start with Hidden Markov Model to create a historical context on modeling time series data. Then, we will discuss recurrent neural networks like Gated Recurrent Unit and Message Passing Encoder-Decoder Recurrent Neural Network. After that we will move on to more modern concepts like attention and transformers. We will also cover Facial Action Coding System and Face Landmarks which are used to preprocess the data.

2.1 Coupled Hidden Markov Model

Coupled Hidden Markov Model (CHMM) developed by Matthew Brand et al. in 1996-97, is considered a classical approach to modeling multiple time series. The Coupled Hidden Markov Model is based on the Hidden Markov Model (HMM). The Hidden Markov Models are used to model Markov Processes, that is, the current state of the process is dependent only on the previous state of the process. An HMM consists of a set of hidden states

\[ S = \{s_1, s_2, s_3, ..., s_N\} \]
where the probability of transition from one state to another is given by

\[ P_{s(t)=i|s(t-1)=j}, 1 < i, j < N \]

where prior probability is given by

\[ P(0) = i \]

and output probability per state is given by

\[ P_{s(t)=i(o(t))} \]

HMM's are typically visually depicted as in Figure 2.1a.

### 2.2 Recurrent Neural Network

Recurrent Neural Networks [10] are a class of popular neural networks architecture extensively used for time-series data. In an RNN a part of the output from previous time step is fed as input to the current time step. Figure 2.2a shows a simplified representation of RNN architecture. Recurrent networks are trained by expanding the network across time, Figure 2.2b, and backpropagating the error through the expanded network. The resulting network has a considerably better performance than HMM, but at the cost of increased training complexity and time.

Very early versions of Recurrent Neural networks simply concatenated portions of, or complete, output to input without any processing. A single step of such an RNN can be depicted by equation:

\[ y_i, h_i = f(c(s_i, h_{i-1})) \]

Where:

- \( s_i \) = the input for \( i^{th} \) step.
- \( y_i \) = the output for \( i^{th} \) step.
- \( h_i \) = the hidden state for \( i^{th} \) step.

When we expand the network over time we get:

\[ y_i, y_{i-1}, y_{i-2}, ..., h_i = f(c(s_i, f(c(s_{i-1}, f(c(s_{i-2}, h_{i-3})))))) \]
Where:
\[ s_i = \text{the input for } i^{th} \text{ step.} \]
\[ y_i, y_{i-1}, \ldots = \text{the output for } i^{th} \text{ step.} \]
\[ h_i = \text{the hidden state for } i^{th} \text{ step.} \]

After expanding the network we can see that the equation has become recursive. All the multiplication operations turn into exponential, which gives problems like diminishing gradient (for \( x < 1 \)) or exploding gradient (for \( x > 1 \)). Some of the problems of RNN was solved by the introduction of LSTM and GRU.

### 2.3 Gated recurrent units

![GRU Model](image)

In an effort to solve some of the problems of RNN, Cho, K. et al. [8] published gated Recurrent Unit (or GRU) in 2014 in his paper on Statistical Machine Translation. In their paper, they proposed two new activation functions for use with RNN. They topped these activation layers with sigmoid functions. As a result, the activation layers behaved like gates. During training, these gates would learn to either remember or forget the hidden state. Hence, they are called update gate and reset gate. Figure 2.4 shows the GRU architecture in a visual representation.

There are two different versions of GRUs that are popular, fully gates and minimally gated. We are using the fully gated GRUs. The equations [9] below describe the fully gated GRU architecture.

**Reset gate is calculated using sigmoid of input \( x_t \) & previous output \( h_{t-1} \).**

\[
r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr})
\]

**Update gate is calculated using sigmoid of input \( x_t \) & previous output \( h_{t-1} \).**

\[
z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz})
\]
Activation vector is calculated using $Tanh$.

$$n_t = \tanh(W_{in} x_t + b_{in} + r_t \times (W_{hn} h_{t-1} + b_{hn}))$$

Final output is calculated by combining activation, update and reset gate.

$$h_t = (1 - z_t) \times n_t + z_t \times h_{t-1}$$

Where:
- $z_t$ is update gate.
- $r_t$ is reset gate.
- $n_t$ is candidate activation vector.
- $h_t$ is the output.
- $W_{ir}, W_{hr}, b_{ir}, b_{hr}$ are trainable parameters for reset gate.
- $W_{iz}, W_{hz}, b_{iz}, b_{hz}$ are trainable parameters for update gate.
- $W_{in}, W_{in}, b_{in}, b_{in}$ are trainable parameters for activation vector.

Gated Recurrent Units have been demonstrated to better than Long Short Term Memory for intent detection in text [17] and modeling music and speech signal [8]. Also, GRUs are just as effective as Long Short Term Memory in natural language processing [29] and modeling speech [27] problems.

### 2.4 Attention

Recurrent neural networks have been successful in modeling sequential data like speech recognition [27], language translation etc. [30]. In spite of the success of RNNs, its capability to model sequential data is only limited to short sequences. The RNN’s inability to process long sequence was, somewhat, remedied by the development of LSTM and GRU. Both LSTM and GRU are variations of the original RNN developed to address the sequence length limitations. Yet, LSTM and GRU are still limited to short to medium length sequences.

One of the reasons why RNN fails to model long sequences is because of the way information is stored. Therefore, our first step is to understand how RNNs store information. As discussed in the previous section, RNNs have hidden states. When an RNN processes a sequence, it encodes (or remembers) the information in the hidden state. The lengthier the sequence, greater the quantity of information that has to be encoded in the hidden state. The finite size of the hidden state puts a limit on the maximum length of sequence that can be processed at a time. Dzmitry Bahdanau et al. came up with an intuitive solution to this problem in their paper [1].
Dzmitry Bahdanau et al. work was on language translation. They based their work on Cho, K. et al. [8], which used RNNs in encoder-decoder configuration. Unlike Cho, K. et al., they added a context vector to their model. They hypothesized that the annotations generated by the encoder contains global information and adding local information, through a context vector, would increase the accuracy of the model. The context vector and the process used to generate it is generally known as attention. The equations below describe their model and the process they used to generate the context vector.

The attention function can be described by equations:

The output conditional probability $y_i$ for $i^{th}$ step is calculated as.

$$y_i = g(y_{i-1}, s_i, c_i)$$

Where:
- $y_{i-1}$ is the probability from last step.
- $s_i$ is the hidden step for $i^{th}$ step.
- $c_i$ is the context vector for $i^{th}$ step.
- $g()$ is non-linear multi-layer perceptron.

Hidden state($s_i$) for the $i^{th}$ step is calculated as

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

Where:
- $s_{i-1}$ is the hidden state from last step.
- $y_{i-1}$ is the probability from last step.
- $c_i$ is the context vector for $i^{th}$ step.
Context vector $c_i$ is calculated as.

$$c_i = \sum_{j=1}^{T} a_{ij} h_j$$

Where:
- $a_{ij}$ is the weight.
- $h_i$ is the annotation from the encoder.

And the weight $a_{ij}$ for each annotation is calculated as.

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

Where:
- $e_{ik}$ is the alignment.

The alignment $e_{ik}$ is calculated using an alignment model ($a()$).

$$e_{ik} = a(s_{i-1}, h_j)$$

Where:
- $s_{i-1}$ is the hidden state from last step.
- $h_j$ is the annotation from the encoder.

The alignment model scores how well input around $j$ is related to output.

### 2.5 Transformer

Introducing Attention in RNN had a positive effect on performance. However, addition of attention does not solve the problems encountered during training, for example, exploding gradient, vanishing gradient, computational cost and lack of parallelization. We discussed these problems in more detail in "Section 2.2 Recurrent Neural Network".

Ashish Vaswani et al. [32] published a paper in 2017 targeting these problems. Up until this point, Recurrent Neural Networks were the de facto standard for modeling sequential data. Instead, they proposed a new architecture called The Transformer which did not have any recurrent units. The Transformer composed of only attention and feedforward layers. A visual representation of Transformer architecture is shown in figures 2.5, 2.6, 2.7, 2.8.

![Figure 2.5: Scaled Dot-Product](image)
The Transformer uses an encoder-decoder architecture. The encoder performs self-attention over the input. Whereas, the decoder performs attention over both the previous output values and attention over the attributes generated by the encoder. They also proposed two new attention functions, called Scaled Dot Product (Fig. 2.5) and Multi-Head Attention (Fig. 2.6), to be used with their Transformer model. The equations below explain Scaled Dot Product and Multi-Head Attention.

![Figure 2.6: Multi-Head attention](image)

Scaled Dot Product takes three input matrices, Query \((Q)\), Key \((K)\) and Value\((V)\). Every values has a corresponding key to it. And the query matrix is used to select values through keys.

Query \((Q)\) is multiplied with the Transpose of Key \((K)^T\) using matrix multiplied to get a raw attention \((A')\) matrix.

\[
A' = Q \cdot K^T
\]

The attention \((A')\) matrix generated in previous equation is optionally scaled with a constant. This is done to get a stronger response from \(softmax\) activation in the next step.

\[
A_{scaled}' = A' \cdot c
\]

Typically, the constant is chosen as \(c = \frac{1}{\sqrt{\text{len}}}\), where \(\text{len}\) is length of key \((K)\). Which makes the equation.

\[
A_{scaled}' = \frac{A'}{\sqrt{\text{Len}(K)}}
\]

Finally, attention matrix \((A)\) is calculated by take the \(softmax\) of \(A_{scaled}'\).

\[
A = \text{softmax}(A_{scaled}')
\]

Optionally, attention \((A_{scaled}')\) is element wise multiplied with a mask to disable some of the input values. This is done to make sure that the decoder only has access to previous outputs during training.

\[
A = \text{softmax}(A_{scaled}' \cdot \text{mask})
\]
Then, Attention \((A)\) is matrix multiplied with value \(V\) to get the output of Scaled Dot Product.

\[ Y = A \ast V \]

The length of the key \((K)\) and the dimensions of the query \((Q)\) are tunable hyperparameters.

Multiple Scaled Dot Product attention are stacked together in parallel to make Multi-Head Attention. The number of heads \((h)\) is a tunable hyperparameter.

First, input is split into multiple heads, \(h\).

\[ \{x_1, x_2, x_3, \ldots x_h\} \]

Then, Scaled Dot Product is calculated for each head. In case of self attention:

\[ y_i = ScaledDotProduct(x_i, x_i, x_i) \]

Or for decoder:

\[ y_i = ScaledDotProduct(x_i, x_i, y_{i-1}, mask) \]

Output from all the heads are concatenated back to get the original shape.

\[ Y = concatenate\{y_1, y_2, y_3, \ldots y_h\} \]

Scaled Dot Product and Multi Head Attention are quite different from the weighted attention by Cho, K. et al. [8]. For one, Multi Head Attention is made up of only linear layers which simplifies the training process. Second, unlike weighted attention, Multi Head Attention can make multiple attention lookups, one for each head. This increases the amount of local information available to decoder, in case current prediction is dependent on multiple points in input.
Transformer Encoder is made up of multiple layers of Multi-Head Attention. Figure 2.7 shows the structure of transformer encoder in a visual representation.

The encoder received Input encoding and Positional encoding as input. Positional encoding consists of \(\text{sine}()\) wave of different frequencies. Positional encoding compensates for lack of recurrence. However, in the original paper, it is shown that positional encoding can be removed without much impact on accuracy.

The positional encoding is catenated with the input encoding and passed to the Multi Head attention layer.

\[
X_{\text{int1}} = \text{MultiHeadAttention}(X, X, X)
\]

Where,
- \(X\) is the catenated input
- \(X_{\text{int1}}\) is the intermediate output from \(\text{MultiHeadAttention}()\) layer.

Input is added to the intermediate output like a residual block and then the sum is \textit{layer normalized}.

\[
X_{\text{int2}} = \text{LayerNorm}(X + X_{\text{int1}})
\]

Finally, a FeedForward Residual Layer is added.

\[
Y_{\text{enc}} = \text{LayerNorm}(\text{FeedForward}(X_{\text{int2}}) + X_{\text{int2}})
\]

This makes up the encoder block portion of the transformer. The transformer consists of multiple layers of encoder block. In the paper, they are using six layers.
The decoder is made up of Multi Head Attention Layers just like the encoder, but with an added mask. Figure 2.8 shows the structure of the decoder. The decoder receives output encoding right shifted by one, along with concatenated positional encoding and the output from the encoder.

The output encoding and the positional encoding are concatenated and passed to Masked Multi Head Attention.

\[
X_{int1} = \text{MaskedMultiHeadAttention}(X, X, X, mask)
\]

Where,
\( X \) is the concatenated input
and \( X_{int1} \) is the intermediate output from \( \text{MaskedMultiHeadAttention}() \) layer.

Mask zeros parts of the attention, so that certain portions of input (to Multi Head Attention) is not visible to the decoder. The mask input is created in such a way that the decoder only has access to previous outputs. This is used for teacher enforced training.

Then, the output from encoder((\( X_{enc} \)) and intermediate input(\( X_{int1} \)) is passed to another MultiHeadAttention.

\[
X_{int2} = \text{MultiHeadAttention}(X_{enc}, X_{enc}, X_{int1})
\]

Where,
\( X_{enc} \) is the output from encoder
and \( X_{int1} \) is the intermediate output from \( \text{MaskedMultiHeadAttention}() \) layer.

Finally, the decoder is topped with a Residual Layer.

\[
Y_{dec} = \text{LayerNorm}(\text{FeedForward}(X_{int2}) + X_{int2})
\]

This makes up the decoder block portion of the transformer. The transformer consists of multiple layers of decoder block. In the paper, they are using six layers. We have implemented our own Transformer in Appendix. A.1 A.2 A.3
Because of the lack of recurrence, Transformers are memory efficient and can be parallelized. Which effectively increases the training and inference speed. Similarly, exploding and vanishing gradients are less frequent. Since the emergence of Transformers, they have become the model of choice for machine translation, document summarization, document generation, named entity extraction [11] and biological sequence analysis [28] [23] [26].

2.6 Message Passing Encoder-Decoder - Recurrent Neural Network

The Message Passing Encoder-Decoder - Recurrent Neural Network is designed to accept two time series as inputs. This makes it easier for our task. Message Passing Encoder-Decoder RNN (MPED-RNN) was introduced by Romero Morais et al. [22] in their paper on anomaly detection. Figure 2.9 and 2.10 shows a high level diagram of the architecture.
The MPED-Rnn is made up of Message Passing Gated Recurrent Units, shown in Figure 2.9. Message Passing Gated Recurrent Units consists of two parallel GRUs which exchange their hidden states. The equations below describe the Message Passing GRUs.

The \( MPGRU() \) accepts the two input streams as input and the two hidden states from last step.

\[
h_n^a, h_n^b = MPGRU(x_n^a, x_n^b, h_{n-1}^a, h_{n-1}^b)
\]

\( MPGRU() \) can be broken down into two GRUs.

\[
h_n^a = GRU_a(x_n^a, h_{n-1}^a)
\]

\[
h_n^b = GRU_b(x_n^b, h_{n-1}^b)
\]

Where,
- \( x_n^a \) is input stream from first input, and \( x_n^b \) from second input.
- \( h_n^a \) and \( h_n^b \) is the hidden state from GRU.
MPED-RNN has one encoder and two decoder as shown in Figure 2.10. The two input streams are fed to the encoder which generates encoder attributes. The encoder attributes are fed to the two decoders. The first decoder tries to reproduce the input stream, whereas the second decoder makes the predictions.

2.7 Facial Action Coding System

In 1969, Carl-Herman Hjortsjö [19] developed a system to categorize human facial movement based on it’s appearance on face. Later, Carl-Herman’s work was taken forward by Paul Ekman et. al [12] by developing the Facial Action Coding System (or FACS). Ekman et. al [12] studied videos containing facial behavior and identified how muscles (or groups of muscles) brought changes to face appearance. By studying anatomy and reproducing expressions, they were able to isolate the ways by which the contraction and palpitation of muscles caused changes in appearance on face. Since it’s inception, Facial Action Coding System (FACS) has become the de facto standard for quantitative description of facial behaviors. The Facial Action Coding System assigns codes to any anatomically possible facial expression. These codes are called action units. Action units consists of 29 main code, 14 head movement codes, 11 eye movement codes, 5 visibility codes and 28 gross behavior codes.

Action units are usually labeled by a trained professional. However, several algorithms have been developed to automate the task. In our case we are using the OpenFace [2] toolkit for automatically labeling the action unit. Of the 29 main code action units, OpenFace is able to recognize only 18 of them. We are using all the 18 action units detectable by OpenFace. Table 2.1 lists the action units detected by OpenFace along with example image s [13].
<table>
<thead>
<tr>
<th>Action Unit</th>
<th>FACS Name</th>
<th>Example Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inner brow raiser</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Outer brow raiser</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Brow lowerer</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Upper lid raiser</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Cheek raiser</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Lid tightener</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Nose wrinkleker</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Upper lip raiser</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Lip corner puller</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Dimpler</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Lip corner depressor</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Chin raiser</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Lip stretcher</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Lip tightener</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Lips part</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Jaw drop</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Lip suck</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Blink</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Action units
2.8 Face Landmarks

Face landmarks are a way to localize positions of various facial features in a picture. We are using OpenFace to extract landmarks from frames. There are a total of 68 2D landmark coordinates, which makes a 136 dimensional input to network. Landmark consist of 2D points in eyes (12 points), eyebrows (10 points), nose (9 points), mouth (20 points) and jaw line (17 points).
Chapter 3

Methodology

In this chapter we describes the procedures and techniques we used to investigate the thesis problem statement. To recap, our aim is to develop an architecture which can model dyadic interacting signals using Transformer architecture. We also want to achieve better performance than Recurrent Neural Network based models.

We are using Recurrent Neural Network based models to make comparisons. We begin with End to end connected GRU since it’s the simplest of the recurrent networks. From there we move to more complicated Recurrent models like Coupled Gated Recurrent Unit, Coupled Gated Recurrent Unit with Attention and Message Passing Encoder-Decoder Rnn. We also explain the Coupled Transformer model that we are proposing. Finally, we discuss the loss functions that we will be using.

3.1 End to End connected GRU

End to end GRU is made up of two parallel branches of GRUs connected at the top with a FeedForward Layer. Each branch of GRU receives an input stream. The Fig 3.1 shows the model architecture.

Figure 3.1: End to End Connected GRU

Let’s say the two GRUs are $GRU_a$ and $GRU_b$, we calculate the

\[ h_i^a = GRU_a(x_i^a, h_{i-1}^a) \]
\[ h^b_i = GRU_b(x^b_i, h^b_{i-1}) \]

Where \( h^a_i \) and \( h^b_i \) are the hidden state for \( i^{th} \) time step.

We concatenate the two states \( h^a_i \) and \( h^b_i \), and apply FeedForward layer to the output.

\[ y_i = fc(catanate(h^a_i, h^b_i)) \]

In case of synthetic data, we use \( sigmoid \) to predict single dimensional correlation factor.

\[ \text{correlation} = \sigma(y_i) \]

In case of rapport data, we use \( LogSoftmax \) to predict two class rapport label.

\[ \text{label} = \log\left(\frac{\exp x_i}{\sum_j \exp x_j}\right) \]

End to End connected GRU is a very basic model with small number of trainable parameters. Hence, we are using it as a base for comparison.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input Size</th>
<th>Output</th>
<th>Activation</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>1</td>
<td>Correlation</td>
<td>( sigmoid() )</td>
<td>Mean Absolute</td>
</tr>
<tr>
<td>ICT Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>( softmax() )</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Rapport</td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>( softmax() )</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Friends</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>( softmax() )</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Friends/Not-Friends</td>
<td>( softmax() )</td>
<td>Cross Entropy</td>
</tr>
</tbody>
</table>

Table 3.1: End to End Connected GRU configuration table

End to End Connected GRU has two Hyper-Parameters, size of hidden state and number of layers of GRU. The values of hyper-parameters chosen in documented in the chapter discussing Results [5.1, 5.2, 5.3, 5.5]. The input size, output size, activation function and loss function are selected based on the dataset being used, these are documented in table 3.1.

### 3.2 Coupled Gated Recurrent Unit

Coupled Gated Recurrent (cGRU) [14] more sophisticated than the standard end to end GRU model. Unlike End to end GRU, which merges the hidden state at the very end, Coupled GRU merges the hidden state after every time step. This merged hidden state is used as a common hidden state for both the GRUs. Figure 3.2 shows a visual representation of CGRU architecture.

Just like End to End GRU, Coupled GRU has two parallel GRU branches \( GRU_a \) and \( GRU_b \)

\[ h^a_i = GRU_a(x^a_i, h_{i-1}^a) \]

\[ h^b_i = GRU_b(x^b_i, h_{i-1}^b) \]

Where \( h^a_i \) and \( h^b_i \) are the hidden states for the \( i^{th} \) step.

The hidden states are merged after every time step using a merging function \( g() \).

\[ h_i = g(h^a_i, h^b_i) \]
We are choosing \( g() \) to be FeedForward layer.

Coupled GRU has slightly larger amount of trainable parameters compared to End to End connected GRU due to the feed forward layer between time steps. The size of this layer is forced to be equal to the size of the hidden state. The feed forward layer is computed after every time step which increases the computational complexity of the network.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input Size</th>
<th>Output</th>
<th>Activation</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>1</td>
<td>Correlation</td>
<td>sigmoid()</td>
<td>Mean Absolute</td>
</tr>
<tr>
<td>ICT Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Friends</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Friends/Not-Friends</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
</tbody>
</table>

Table 3.2: Coupled GRU configuration table

Coupled GRU has two Hyper-Parameters, size of hidden state and number of layers of GRU. The values of hyper-parameters chosen in documented in the chapter discussing Results [5.1, 5.2, 5.3, 5.5]. The input size, output size, activation function and loss function are selected based on the dataset being used, these are documented in table 3.2.

### 3.3 Coupled Gated Recurrent Unit with Attention

In this model we add attention to the Coupled Gated Recurrent Unit model from Section 3.2. Attention is applied to the hidden states. The model is visualized in Fig. 3.3.

Just like Coupled Gated Recurrent Unit, hidden state for \( ith \) step is calculated and merged using the merge function \( g() \).

\[
\begin{align*}
    h_i^a &= GRU_a(x_i^a, h_{i-1}) \\
    h_i^b &= GRU_b(x_i^b, h_{i-1}) \\
    h_i &= g(h_i^a, h_i^b)
\end{align*}
\]
$H = \{h_1, h_2, ... h_i\}$

Weights($w_i$) for each time is calculated using the weight function $w()$.

$$w_i = w(h_i)$$

We accumulate all the weights into a weight vector($W$) and apply $softmax$ to get the attention vector($A$).

$$W = \{w_1, w_2, ... w_i\}$$

$$A = softmax(W)$$

We apply the attention($A$) vector to the set of hidden states($H$) to get the output. Finally, output is topped with an activation function according to the dataset.

$$Y = activation(A \ast H)$$

Coupled GRU with Attention has the added parameters from attention weight function in addition to parameters from Couple GRU. Attention Weights has to be calculated for every time step which increases computational complexity.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input Size</th>
<th>Output</th>
<th>Activation</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>1</td>
<td>Correlation</td>
<td>sigmoid()</td>
<td>Mean Absolute</td>
</tr>
<tr>
<td>ICT Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Friends</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Friends/Not-Friends</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
</tbody>
</table>

Table 3.3: Coupled GRU w/ attention configuration table

Coupled GRU has two Hyper-Parameters, size of hidden state and number of layers of GRU. The values of hyper-parameters chosen in documented in the chapter discussing Results [5.1, 5.2, 5.3, 5.4, 5.5]. The input size, output size, activation function and loss function are selected based on the dataset being used, these are documented in table 3.3.

3.4 Coupled Transformers

The original transformer implementation (discussed in section 2.5) only takes one time series as input. But the datasets that we are using require two time series as input. To remedy this problem, we are proposing a new transformer architecture called "Coupled Transformer".

Coupled transformer consists of two encoders instead of one, one for each input stream. The output attributes from the two encoders are concatenated and passed to the decoder. The first layer in the decoder is a multi-head attention layer. The multi-head attention layer in decoder splits the concatenated attributes from the encoder into \( h \) number of heads and performs attention on each head. Essentially, what this does is select(or pick) the most useful attributes from each of the inputs. Figure 3.4 shows the visual representation of our proposed model.

Let, \( X^a \) and \( X^b \) be be the two input series.

\[
X^a = x_1^a, x_2^a, ..., x_i^a
\]

\[
X^b = x_1^b, x_2^b, ..., x_i^b
\]

The inputs \( X^a \) and \( X^b \) are passes to the two encoders \( Enc_a \) and \( Enc_b \).

\[
Z^a = Enc_a(X^a)
\]

\[
Z^b = Enc_b(X^b)
\]

The two encoders \( Enc_a \) and \( Enc_b \) produce the encoded attributes \( Z^a \) and \( Z^b \).

\[
Z^a = \{z_1^a, z_2^a, ..., z_n^a\}
\]

\[
Z^b = \{z_1^b, z_2^b, ..., z_n^b\}
\]
We concatenate the attributes from the two encoders along the feature dimension.

\[ Z = \{ z_1^a \oplus z_1^b, z_2^a \oplus z_2^b, \ldots, z_n^a \oplus z_n^b \} \]

Where, symbol ‘\( \oplus \)’ represents concatenation.

When the encoder attribute \( Z \) is passed to the Multi Head attention in the decoder, the input \( Z \) is split into multiple heads and Scaled dot product attention is calculated for each head. By choosing an even number of heads, we can make sure that the attributes originating from the two encoders has equal number of split heads. Since, the input attribute \( Z \) contains information from both the encoders, then attention is calculated, it picks information from both the encoders. Which leads to coupling.
Coupled Transformer doesn’t use recurrence. It’s made up of Feed forward layers and Self-Attention layers. Due to this Our model has lower memory footprint and can be easily parallelized.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Input Size</th>
<th>Output</th>
<th>Activation</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>1</td>
<td>Correlation</td>
<td>sigmoid()</td>
<td>Mean Absolute</td>
</tr>
<tr>
<td>ICT Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Rapport</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td>UaB Friends</td>
<td>18 Action Units</td>
<td>Rapport/Non-Rapport</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
<tr>
<td></td>
<td>136 Landmarks</td>
<td>Friends/Not-Friends</td>
<td>softmax()</td>
<td>Cross Entropy</td>
</tr>
</tbody>
</table>

Table 3.4: Coupled GRU w/ attention configuration table

Hyper Parameters in a Transformer is different than recurrent models. Coupled transformer has five tunable hyper parameters, number of encoder blocks, number of decoder blocks, number of heads in encoder, number of heads in decoder and $d_{model}$ parameter which specifies the feature size of intermediate layers. The values of hyper-parameters chosen in documented in the chapter discussing Results [5.1, 5.2, 5.3, 5.5]. The input size, output size, activation function and loss function are selected based on the dataset being used, these are documented in table 3.4.
3.5 Loss Function

3.5.1 Mean Absolute Loss

Let, $X$ be a vector containing $N$ predicted values.

$$X = \{x_1, x_2, \ldots x_N\}$$

Let, $Y$ be a vector containing $N$ target values corresponding to elements in vector $X$.

$$Y = \{y_1, y_2, \ldots y_N\}$$

Then, we calculate the absolute difference $(l_n)$ between each element of the vectors $Y$ and $X$.

$$l_n = |y_n - x_n|$$

Then, mean absolute error is defined as:

$$loss(X, Y) = \frac{\sum_{n=1}^{N} l_n}{N}$$

3.5.2 Cross Entropy Loss

Cross Entropy Loss \cite{21} combines softmax and Mean Absolute Error (Section 3.5.1).

Let, $X$ be a vector containing predicted probabilities for $N$ number of classes.

$$X = \{x_1, x_2, \ldots x_N\}, x_1, x_2 \ldots x_n \in [0, 1]$$

And 'class' is the target class.

$$class \in [1, N]$$

Then, cross entropy loss is defined as:

$$loss(X, class) = -\log\left(\frac{\exp(x_{class})}{\sum_j \exp(x_j)}\right)$$

Where:

- $x_j$ is the probability of occurrence of $j^{th}$ class
- $x_{class}$ is the probability of occurrence of target 'class'
Chapter 4

Datasets

In this chapter we mention the datasets that we use in our thesis. We benchmark the performance of our model using four different datasets, Synthetic data (Section 4.1), USC Institute for Creative Technology Rapport Dataset (Section 4.2), University at Buffalo Communication Science Center Rapport dataset (Section 4.3), and University at Buffalo Communication Science Center, Friends Dataset (Section 4.4). We also discuss the pre-processing that we perform on the datasets in section 4.5.

4.1 Synthetic Data

![Figure 4.1: Example of synthetic data](image)

Synthetic data is usually too simplistic and it’s never a good representation of real world datasets. We are using synthetic data to check if our architecture can model the most basic case. To create the synthetic
dataset we are generating two coupled Gaussian stochastic processes using the method developed by Tayeb et al. [20]. The equations below describe the process we are using to generate the data.

Let, \( u \) and \( v \) be sets of size \( L \) each, containing random numbers from a normal distribution.

\[
\begin{align*}
    u_q &= \text{fourier}(u_i) \\
    v_q &= \text{fourier}(v_i)
\end{align*}
\]

We calculate the spectral correlations \( S_{xx}, S_{yy} \) and \( S_{xy} \) from \( C_{xx}, C_{yy} \) and \( C_{xy} \):

Calculate coefficients \( A_q, B_q, C_q, D_q \)

\[
\begin{align*}
    A_q &= \sqrt{S_{xx}(q)} \cos(\alpha_q) \\
    B_q &= \sqrt{S_{xx}(q)} \sin(\alpha_q) \\
    C_q &= \sqrt{S_{yy}(q)} \cos(\beta_q) \\
    D_q &= \sqrt{S_{yy}(q)} \sin(\beta_q)
\end{align*}
\]

Where \( \alpha \) and \( \beta \) satisfy the condition:

\[
\alpha_q - \beta_q = \arccos\left(\frac{S_{xy}(q)}{\sqrt{S_{xx}(q) S_{yy}(q)}}\right)
\]

We calculate \( x_q, y_q \) using the coefficients:

\[
\begin{align*}
    x_q &= A_q u_q + B_q v_q \\
    y_q &= C_q u_q + D_q v_q
\end{align*}
\]

We calculate the target signal by calculating the inverse Fourier transform of \( x_q, y_q \):

\[
\begin{align*}
    x_i &= \text{InvFourier}(x_q) \\
    y_i &= \text{InvFourier}(y_q)
\end{align*}
\]

The generated coupled signals of sequence length 100 that has correlation factor in \([0,1]\). A delay of up to 15 time steps is inserted between the two sequences to make it a bit more realistic. Figure 4.1 shows an example of synthetic data with a correlation factor of 0.4338. We created the dataset by generating 10,000 data samples having a sequence length of 100. The dataset is partitioned into three parts, 8000 samples for training, 1000 samples for testing, and 1000 samples for validation.

### 4.2 USC Institute for Creative Technology, Rapport Dataset

Dataset was recorded by Jonathan et al. [16] at University of Southern California. The dataset contains 165 interactions. Each interaction is 2 to 3 minutes in duration. Each interaction consists of two human participants, a "speaker" and a "listener". During the interaction the "speaker" tells a story to the "listener" and the "listener" is asked to remain silent during the interaction.
Of the 165 videos, only 23 videos had both "speaker" and "listener" portions of the recording.

### 4.3 University at Buffalo Communication Science Center, Rapport dataset

University at Buffalo, SUNY, Communication Science Center conducted a study focused on deception detection. In this study, researchers asked the participants to steal either a ring or a watch and hide it in the conference room. Later, an interviewer interviewed the participants. The interviewer’s job was to find out if the interviewee stole any of the items, whereas the interviewee had to convince the interviewer that they stole nothing. To make the experiment more realistic, they rewarded the participants for successfully convincing the interviewer. Whereas, if they failed, they received punishment.

The interview process was recorded. Each recording comprises two video files, one for the interviewer and one for the interviewee. Each interview video has a baseline section and an interrogation section. In the baseline section of the video, the interviewer builds up rapport with the interviewee. In the interview section of the video, the actual interview takes place. For our research, only the baseline section of the video is useful to us. The dataset contains 59 interviews. We are splitting it into 45 for training, 6 for validation and 8 testing.
4.4 University at Buffalo Communication Science Center, Friends Dataset

Friends dataset collected University of Buffalo and consists of pairs of interview videos. Each interview video pair consists of an interviewer asking questions and a subject answering them. The start and end positions of questions and answers are recorded in a CSV file. Figure 4.4

![Frame from the Interviewer camera.](image1)

![Frame from the Observer camera.](image2)

Figure 4.4: Frames from the Friends dataset from two different perspectives.

Face landmarks and Action units are extracted using OpenFace. Landmarks are normalized using the same equation as for USC ICT dataset in section ??.

4.5 Face Feature extraction

Neural Networks are notably sensitive to the form of data being input. If we train our model with raw frames, then we would face several problems. First, raw frames contain a lot of information. For instance, scene lighting, contrast, objects in background, etc. and all this information is not useful to us. Because of this, our model will spend a significant amount of training time trying to separate useful information. Second, uncompressed image frames consume a lots of RAM. This will limit the length of sequences we can train on. Last, the videos in different datasets will have different parameters, like frame resolution and aspect ratios. As a result, our model might not generalize well across datasets.

To solve this problem, we are using OpenFace [2] toolkit to preprocess the data. We are using OpenFace toolkit to extract “Landmark” [34] [3] and “Action Units” [4] for every frame. The extracted Action Units are already in normalized form, so we can directly use them for training. However, the Landmarks are in pixel coordinates and need to be normalized. There are 68 Landmark points per detected face.

We normalize the landmark coordinates such that the normalized coordinates are \( x_n, y_n \in [0, 1] \). The normalized coordinates lie inside the bounding box of the face. Normalized coordinates are calculated using the below equations.

Let,

\[ p_i \text{ be } i^{th} \text{ landmark} \]

\[ p_i = (x_i, y_i), i \in [0, 67] \]
$P$ is the set of all landmarks.  

\[ P = \{p_0, p_1, ..., p_n\} \]

Calculate bounding box.

\[
x_{\text{min}} = \min(x_0, x_1, ..., x_i)
\]

\[
x_{\text{max}} = \max(x_0, x_1, ..., x_i)
\]

\[
y_{\text{min}} = \min(y_0, y_1, ..., y_i)
\]

\[
y_{\text{max}} = \max(y_0, y_1, ..., y_i)
\]

$p_{\text{min}}$ and $p_{\text{max}}$ are the bounding coordinates.

\[
p_{\text{min}} = (x_{\text{min}}, y_{\text{min}})
\]

\[
p_{\text{max}} = (x_{\text{max}}, y_{\text{max}})
\]

Then we calculate the normalized coordinate $p'_i$ using.

\[
p'_i = \frac{p_i - p_{\text{min}}}{p_{\text{max}} - p_{\text{min}}}
\]

Where $p'_i$ is the $i’th$ normalized landmark
Chapter 5

Results

We started by training the models (End-to-End GRU, cGRU, cGRU with Attention, MPED-RNN and Coupled Transformer) on the synthetic dataset. After that we train the models on the University at Buffalo Communication Science Center, Rapport dataset. Next, we evaluate the trained models on the USC Institute for Creative Technology Rapport Dataset. Finally, we train the models on the University at Buffalo Friends dataset. We presented the results from these experiments below.

All the training and validation was performed on a server with Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz CPU with 32GB of RAM, and Nvidia Tesla K40c GPU with 12GB of RAM. We used PyTorch [24] python package to implement and train our model. We trained all the models using the Tesla K40c GPU.

5.1 Experiment 1: Test Coupling

We started by testing how well our models could detect coupling. We achieved this by training our models on the Synthetic Dataset we created in Section 4.1. Each sample in the synthetic dataset contains two Gaussian processes and the correlation factor between the two series. The two Gaussian series are the inputs to our models, we are feeding each series to an arm of our model. And, the correlation factor is the label. All the models have to predict the correlation factor.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>GRU</th>
<th>cGRU</th>
<th>MPED-Rnn</th>
<th>Coupled Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>Hidden Size</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Layer Count</td>
<td>4</td>
<td>4</td>
<td>1 Encoder &amp; 1 Decoder</td>
<td>1 Encoder &amp; 1 Decoder</td>
</tr>
<tr>
<td>Head Count</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>4 Encoder &amp; 4 Decoder</td>
</tr>
</tbody>
</table>

Table 5.1: Experiment 1: Hyper-Parameter

All the models were topped with the sigmoid() activation function to restrict the output to (0,1) range. The rest of the hyper parameters for each of the models are listed in table 5.1. The models were trained using Mean Absolute Error (Section 3.5.1) as the loss function. We used Adam optimizer to perform regression on the models. Each model was trained using a different learning rate that best suited it, the learning rates are also listed in table 5.1. The loss plot for training and validation for the synthetic data is plotted in Figure 5.1a & 5.1b respectively.
By analyzing Figure 5.1a, we can see that Message Passing Encoder-Decoder RNN trains the fastest with the lowest loss, but the performance isn’t carried over to validation. During validation, Message Passing Encoder-Decoder RNN performs 3rd best. This may be due to the high parameter count in MPED-RNN which leads to over fitting, and the model fails to generalize well. We also notice that Coupled GRU and Coupled Transformer have converged pretty close to each other in training with Coupled Transformer having slightly lower loss. But, Coupled GRU is very jittery at the beginning of the training and remains to be jittery till the end in validation. Finally End to End connected GRU performs the worst in both training and validation by a wide margin.

![Training loss](image1.png)  ![Validation loss](image2.png)

Figure 5.1: Loss plots for Synthetic data.

The validation performance of Coupled Transformer, cGRU and MPED-Rnn is pretty close. Coupled Transformer has a slight lead, with cGRU and MPED-Rnn slightly behind respectively. We think that the reason three of our models have converged so close together, is that the synthetic data may be to simplistic for these models.
We also generated scatter plot using 100 randomly selected samples from the test set for the End to End GRU and Coupled Transformer, shown in Figure 5.2a & 5.2b respectively. The $X$ - axis is the actual correlation factor between the two signals. The $Y$ - axis is the predicted correlation factor. The blue dots are the predicted samples and the red dotted line is ideal line.

In Figure 5.2a we see that the correlation factor predicted by the End to end connected GRU does not line up well with the actual value. There is a lot of deviance from the red line. But in Figure 5.2b we can see that the blue dots align much better for Coupled Transformer than for End to end connected GRU. The scatter plot is much closer to the red line. In a perfect model, all the blue dots would lie on the red dotted line.

### 5.2 Experiment 2: Detect Rapport and Compare Architectures

Next, we trained the models to predict rapport between two conversing individuals. We trained End To End GRU, Coupled GRU, Coupled GRU with attention, MPED-RNN and Coupled transformer on the University at Buffalo Communication Science Center, Rapport dataset (Section 4.3). We used Cross Entropy Loss (Section 3.5.2) as our loss function and trained the models using Adam optimizer. Hyper parameters like learning rate, hidden size and layer count were selected with trial and error to give the best training performance for that model. The final hyper parameters are listed in Table 5.2.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>GRU</th>
<th>cGRU</th>
<th>cGRU w/ Attention</th>
<th>MPED-Rnn</th>
<th>Coupled Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>0.001</td>
<td>0.001</td>
<td>0.001</td>
<td>0.0001</td>
<td>0.0001</td>
</tr>
<tr>
<td>Hidden Size</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Layer Count</td>
<td>4</td>
<td>4</td>
<td>4 2 Encoder &amp; 2 Decoder</td>
<td>2 Encoder &amp; 2 Decoder</td>
<td></td>
</tr>
<tr>
<td>Head Count</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>4 Encoder &amp; 4 Decoder</td>
</tr>
</tbody>
</table>

Table 5.2: Experiment 2: Hyper-Parameter

We trained all our models with two different types of input features. Once with Landmarks as input, and again with Action units as input. All the configurations were trained and tested 10 times. Each run was started with randomly initialized weights, with no transfer learning from last run. For each run, training set, validation set and test set were randomly selected.
We plotted the validation accuracy and validation loss for University at Buffalo Communication Science Center, Rapport dataset (Section 4.3) in Figure 5.3a & 5.3b. The X-axis shows the number of epochs. And the Y-axis shows the accuracy of the model in figure 5.3a and loss of the models in figure 5.3b. From the accuracy plot, we can see that Coupled Transformer with Landmark is the most accurate, whereas cGRU with Landmarks is the least accurate.

We calculated the average accuracy of each model for each of the input types. The mean accuracy is tabulated in Table 5.3, column "UaB Rapport". The delta between the mean and extremum is tabulated in Table 5.3, "Delta" column.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Input Type</th>
<th>UaB Rapport</th>
<th>Delta</th>
<th>ICT Rapport</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRU</td>
<td>Landmarks</td>
<td>67.44</td>
<td>12.54</td>
<td>59.177</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>53.42</td>
<td>7.27</td>
<td>59.549</td>
</tr>
<tr>
<td>cGRU</td>
<td>Landmarks</td>
<td>57.5</td>
<td>1.83</td>
<td>51.813</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>62.98</td>
<td>10.97</td>
<td>60.549</td>
</tr>
<tr>
<td>cGRU w/ Attention</td>
<td>Landmarks</td>
<td>72.5</td>
<td>7.98</td>
<td>59.751</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>74.15</td>
<td>5.97</td>
<td>63.98</td>
</tr>
<tr>
<td>MPED</td>
<td>Landmarks</td>
<td>70.4</td>
<td>NA</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>79.14</td>
<td>NA</td>
<td>53.37</td>
</tr>
<tr>
<td>Transformer</td>
<td>Landmarks</td>
<td>81.97</td>
<td>6.64</td>
<td>67.2163</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>75.71</td>
<td>2.93</td>
<td>62.277</td>
</tr>
</tbody>
</table>

Table 5.3: Validation results for University at Buffalo Communication Science Center, Rapport dataset and transfer performance on USC Institute for Creative Technology Rapport Dataset

To check how well our model generalizes, we picked the best performing model from the 10 runs, and tested it on the USC Institute for Creative Technology Rapport Dataset (Section 4.2). The accuracy from cross dataset test is tabulated in Table 5.3, column "ICT Rapport".

The comparison of model accuracy is also shown in Figure 5.4 as a bar graph. The X-axis shows the model with the input type, the Y-axis shows the accuracy of the model. The purple bars show the testing accuracy of model on the University at Buffalo Communication Science Center, Rapport dataset (Section 4.3). The black line on top of the purple bar shows the delta in accuracy across the runs. The green bar shows testing accuracy of transferred model on the USC Institute for Creative Technology Rapport Dataset (Section 4.2).
From table 5.3 we see that End to End connected GRU has the lowest accuracy in training set for both Landmarks and Action Units. It’s transfer performance on ICT Rapport dataset is 59% for both the cases. Coupled GRU performs better than the End to end connected GRU.

cGRU performs better with Action units than with Landmarks, but Action units has a lot more variance than Landmarks. When we look at the transfer performance on ICT Rapport data set, Action units perform significantly better than Landmarks. But when we compare it to end to end GRU, accuracy only increases slightly for action units, and decreases for landmarks. Using landmarks with cGRU does not fare well.

Adding attention to cGRU improves the performance quiet a lot. We can see that accuracy for both Action Units and Landmarks has improved quiet a lot at 72.5% and 74.15%. We also notice that adding attention stabilizes the training process and test accuracy. In End to End GRU and cGRU the variance in accuracy is pretty big in ranges 10-12. After adding attention variance reduces to 6-8 range.

If we compare MPED-Rnn to cGRU, Action Units performs better, but Landmarks performs worse. The transfer test results for MPED-Rnn are 44% and 53.37% for Landmarks and action units respectively. This is the worst score of all. The bad transfer test score can be attributed to high parameter count, which leads to failed generalization. Training MPED-Rnn requires a lot of memory operations due to message pass and overall the model is quiet bulky, due to this it takes very long to train it. For this reason we only performed one run of this model.

![Figure 5.4: A comparison of different models](image)

Lastly, Coupled Transformer gives the best result, at 81.97, when we use landmarks as input. When
used with Action Units, the accuracy is on par with cGRU. Apart from scoring high on accuracy, the model is much more stable than recurrent networks. Coupled Transformer performs quiet well when we transfer test the model on ICT dataset. Coupled Transformer with Landmarks is the best scoring model of all in that regard. Whereas Coupled Transformer with Action units performs on par with cGRU with Attention. Coupled Transformer’s good performance on both types of input can be attributed to less number of trainable parameters than Recurrent Networks. Also using only attention without any recurrence makes gradient descent much easier.

If we pay close attention to table 5.3, we can notice a pattern. The recurrent networks perform well with Action Units, but poorly with Landmarks. Whereas Coupled Transformer works best with Landmarks, and on par (slightly less) than Recurrent networks.

5.2.1 Execution Time Comparison

Execution time is another important aspect of an architecture. The timing of a model determines how well the model can be scaled. We benchmarked the time needed to train one iteration for each model. For running the benchmark, we used a batch size of 256. The benchmark was run on the system mentioned in Section 5. The execution timings of all the models are tabulated in Table 5.4.

Looking at table 5.4, we see that there is no clear advantage while choosing Landmarks or Action Units where execution time is concerned. Even though Landmarks are bigger in size than Action Units, the difference in execution time is within a few milliseconds and within fluctuations. We think that this might be due to the parallel nature of GPU training. Even though Landmarks is larger than Action units, it’s within the bandwidth limit of GPU.

<table>
<thead>
<tr>
<th>Model</th>
<th>Feature</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRU</td>
<td>Landmarks</td>
<td>7.8ms</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>9.3ms</td>
</tr>
<tr>
<td>cGRU</td>
<td>Landmarks</td>
<td>97ms</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>92ms</td>
</tr>
<tr>
<td>cGRU w/ Attention</td>
<td>Landmarks</td>
<td>93ms</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>97ms</td>
</tr>
<tr>
<td>MPED</td>
<td>Landmarks</td>
<td>351ms</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>364ms</td>
</tr>
<tr>
<td>Transformer</td>
<td>Landmarks</td>
<td>32ms</td>
</tr>
<tr>
<td></td>
<td>Action Units</td>
<td>40ms</td>
</tr>
</tbody>
</table>

Of the five models we benchmarked, End to End connected GRU has the lowest execution time. This is due to the simplicity of end to end connected GRU. Coupled Transformer is faster slower than end to end GRU, but twice as fast as cGRU and cGRU with attention. This is due to lack of recurrence which simplifies training. Coupled GRU and Coupled GRU with attention have similar execution times because of similar architecture. Message Passing Encoder Decoder Rnn is the slowest. The message passing architecture requires a lot of memory copy to implement, which slows down the model considerably.
5.3 Experiment 2a: Check Tight Coupling

All our models use two parallel branches of GRUs or two encoders. We also wanted to test the effects of tightly packing the input features. This would make sure that all the bases are covered. The started by concatenating the input along feature dimension to create a single series. Out model consists of only Transformer encoders since there is only one series. We have chosen Transformer based on results from Experiment 2(Section 5.2). The hyper parameters of the model is tabulated in table 5.5. The merged series is fed to the model.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>Hidden Size</td>
<td>64</td>
</tr>
<tr>
<td>Layer Count</td>
<td>4</td>
</tr>
<tr>
<td>Head Count</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.5: Experiment 2a: Hyper-Parameter

We trained the model using Action Units and Landmarks. We used Adam optimizer to train our model. The validation accuracy of the model is shown in figure 5.5. From the graph we can see that the model is very inaccurate for both Action Units and Landmarks. Both the cases have an average accuracy close to \( \approx 50\% \). For a two class classification problem, this is not a great number. Accuracy of \( \approx 50\% \) indicates that the model is making random predictions and not actually learning anything. We think the reason why the model is unable to perform well is because it’s unable to pick up synchrony between the two series when the series are tightly coupled.

![Figure 5.5: Validation Accuracy vs. Epoch plot for Tightly Coupled](image)

37
5.4 Experiment 3: Detect Friendship

We also wanted to see how our model fared on other classification tasks. Based on Experiment 2(Section 5.2) we chose Coupled Transformer model. We trained the model using the University at Buffalo Communication Science Center, Friends Dataset. The dataset labels are “Friends” or “Not Friends”. Therefore, we use cross entropy loss with two classes. We trained the model using Adam optimizer with hyper parameter tabulated in table 5.6. We trained the model once with Landmarks and again with Action Units. We trained the model ten times for each of the input features. The mean validation accuracy and testing accuracy of the model is tabulated in Table 5.7.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning Rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>Hidden Size</td>
<td>64</td>
</tr>
<tr>
<td>Layer Count</td>
<td>2 Encoder &amp; 2 Decoder</td>
</tr>
<tr>
<td>Head Count</td>
<td>4 Encoder &amp; 4 Decoder</td>
</tr>
</tbody>
</table>

Table 5.6: Experiment 3: Hyper-Parameter

<table>
<thead>
<tr>
<th>Input Feature</th>
<th>Validation Accuracy</th>
<th>Test Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landmark</td>
<td>72.31±1.1</td>
<td>70.15</td>
</tr>
<tr>
<td>Action Units</td>
<td>68.9±5.2</td>
<td>65.88</td>
</tr>
</tbody>
</table>

Table 5.7: Friends Dataset Results

On Landmarks, we achieves an average accuracy of 72.31% and test accuracy of 70.15%. On Action Units, we got an average accuracy of 68.9% and test accuracy of 65.88%. In figure 5.6 we plotted the validation plot from the best run.

Figure 5.6: Validation Accuracy vs. Epoch plot for Friends Dataset
5.5 Experiment 4: Find effects of attention

We also created a way to visualize the attention mechanism that the transformer is learning internally.

(a) Frame from the Interviewer camera. (b) Frame from the Interviewer camera. (c) Frame from the Interviewer camera. (d) Frame from the Interviewer camera.

(e) Self Attention From Interviewer Encoder

(f) Frame from the Interviewer camera. (g) Frame from the Interviewer camera. (h) Frame from the Interviewer camera. (i) Frame from the Interviewer camera.

(j) Self Attention From Interviewee Encoder.

Figure 5.7: Self-Attention during Non-Rapport.

39
Coupled transformer has two encoders and one decoder. Since there are two encoders, there is two different self-attention layers. Each encoder performs self-attention over the input stream independently. In the decoder layer, we can see how the attention is fetching states from both the layers.

We dumped the self attention and coupled attention for both the cases. When the network is seeing strong rapport and when the network is seeing strong non-rapport.

In Figure 5.7 we have shown the self-attention mechanism from Interviewer and Interviewee parts of the encoder. Figure 5.7c shows the attention matrix that gets generated for the interviewer, and Figure 5.7f shows the interviewee attention. The $x-$axis shows the sequence number, the left most point is the starting point in time and the right most is the 100th frame. The $y-$axis shows the head number. Since our model has four heads, we can make out the four distinct lines in Figure 5.7c and 5.7f. When generating attention, the last step is to apply softmax which normalizes the matrix along the sequence dimension. But due to this kind of normalization, pixel values don’t show up because they are too small to display. Therefore we had to rescale the attention matrix in $[0,1]$ range by dividing the matrix by the maximum value in the matrix.

To visualize the frames responsible for prediction, we picked the index of the maximum value of attention (or argmax()). We used these indices to fetch the frames. We used attention from Figure 5.7c to get the frames from interviewer which are responsible for non-report prediction, they are shown in Figures 5.7a, 5.7b, 5.7c, 5.7d. Similarly, we used attention from Figure 5.7f to get the frames from interviewee which are responsible for non-report prediction, they are shown in Figures 5.7e, 5.7g, 5.7h, 5.7i.

![Frames from Interviewer and Interviewee](image)

(a) Frame from the Interviewer camera. (b) Frame from the Interviewee camera. (c) Frame from the Interviewer camera. (d) Frame from the Interviewee camera.

![Coupled-Attention Matrix](image)

(e) Coupled-Attention From Decoder.

Figure 5.8: Coupled-Attention during Non-Rapport.
The attention from decoder is a little different than that of the encoder. The encoder applies self-attention to only the input it receives, whereas the decoder applies attention to both the inputs. Figure 5.8 shows the analysis from decoder. We are using four heads in the decoder. Two of the heads receive input from encoder one, and the rest two receive input from second encoder. In Figure 5.8c, the bottom two heads are generated from Interviewer encoder, and the top two heads are generated by the interviewee encoder. The frames pointed to by Figure 5.8c are shown in Figures 5.8a, 5.8b, 5.8c, and 5.8d.

Similarly, we have created figures for when the model detects strong rapport (Figure 5.9). Figure 5.9 shows...
shows the attention heat map from interviewer and the extracted frames in Figures 5.9a, 5.9b, 5.9c, 5.9d.
Figure 5.9j shows the attention heat map from interviewee and the extracted frames in Figures 5.9f, 5.9g, 5.9h, 5.9i.

(a) Frame from the Interviewer camera. (b) Frame from the Interviewer camera. (c) Frame from the Interviewer camera. (d) Frame from the Interviewer camera.

Figure 5.10: Rapport mutual attention

Likewise, coupled attention is shown in figure Figure 5.10. the bottom two heads is generated from Interviewer encoder (extracted frames in Figures 5.8a, 5.8b), and the top two heads is generated by the interviewee encoder ((extracted frames in Figures 5.8c, 5.8d)).

Some observations from looking at attention visualizations (Figures 5.10, 5.9, 5.8, 5.7):

- In the case of non-rapport, the extracted frames show no mirrored behavior in Figure 5.7.
- In the case of rapport, the extracted frames show that both the participants are nodding and smiling in Figure 5.9.
- In case of rapport, the attention is picking frames containing nodding and smiling.
- We noticed that the transformer is more sensitive to head movement like nodding, than to facial expression like smiling.
Chapter 6
Concluding

6.1 Concluding Thoughts

In this thesis, we investigated various ways to model dyadic sequential data. More specifically, we modeled social interaction between two individuals. We used conventional recurrent neural network to model such data and discussed failures in training such models. We experimented with end to end connected GRU, coupled GRU, coupled GRU with attention and MPED-Rnn architectures. Then, we proposed a new architecture, Coupled Transformer, which is inspired by the Transformer.

We began our experiments by training our model on synthetically generated data. The models Coupled GRU, Coupled GRU with Attention, MPED-Rnn and Coupled Transformer performed very well on the synthetic data. However, end to end connected GRU didn’t fare as well with synthetic data. Then we trained the models to predict rapport between two persons using —it University at Buffalo Communication Science Center, Rapport dataset. After that, we tested how well the model generalizes by testing it on the USC Institute for Creative Technology Rapport Dataset. The Coupled Transformer model was the best performing model out of all the four. Likewise, Coupled Transformer generalized the best of all the models on the USC Institute for Creative Technology Rapport Dataset. We also trained our model on the friends dataset. We were able to get similar results on the friends dataset.

We also extracted the self-attention and coupled-attention matrices generated by our model and visualized it in a heat map. We used the heat map to extract frames which lead to rapport and non-rapport label. Using these frames, we confirmed that the frames that mirrored behaviors such as nodding and laughing were responsible for rapport detection.

In conclusion, we developed our own model called Coupled Transformers which is capable of modeling dyadic human interaction. Our model was able to predict rapport between two interacting individuals, by looking at the videos of them interacting, with an accuracy of 81.97% while being ≈ 2.5 times faster than recurrent models.

6.2 Future Work

In this study, we successfully modeled social interactions between two persons. We would like to continue our work and develop new architectures which can model multiple modalities of data simultaneously, like speech and gesture. We would like to experiment with entire body pose as input (using OpenPose [7]). But that would need a different dataset. And, we would also like to use the models we have developed to train
intelligent agent to behave like a person using reinforcement learning.
Bibliography


Appendices
Appendix A

Model Implementation

A.1 Transformer Encoder Block

class EncTransformerBlock(nn.Module):
    def __init__(self, d_model, h, dropout=0.1, save_attention=False):
        super(EncTransformerBlock, self).__init__()
        self.attention = MultiHeadAttention(
            d_model, h, dropout=dropout, save_attention=save_attention)
        self.fc1 = nn.Linear(d_model, d_model)
        self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
        self.dropout = dropout

    def forward(self, x: torch.FloatTensor):
        x = self.norm1(self.attention(x, x, x) + x)
        x = self.norm2(torch.relu(self.fc1(x) + x))
        x = torch.dropout(x, p=self.dropout, train=self.training)
        return x

A.2 Transformer Decoder Block

class DecTransformerBlock(nn.Module):
    def __init__(self, d_model, d_k, d_v, h, teach_assist=True):
        super(DecTransformerBlock, self).__init__()
        self.d_k, self.d_v = d_k, d_v
        self.teach_assist = teach_assist
        self.attention1 = MultiHeadAttention(d_model, d_k, d_v, h)
        self.attention2 = MultiHeadAttention(d_model, d_k, d_v, h)
        self.fc1 = nn.Linear(d_model, d_model)
        if self.teach_assist:
            self.norm1 = nn.LayerNorm(d_model)
        self.norm2 = nn.LayerNorm(d_model)
```python
self.norm3 = nn.LayerNorm(d_model)

def create_mask(self, y):
    batch_size, sequence, features = y.shape
    mask: torch.Tensor = torch.zeros(
        (batch_size, sequence, sequence), dtype=torch.bool, device=y.device)
    for seq in range(sequence):
        mask[:, seq, seq:] = True
    return mask

def forward(self, x: torch.FloatTensor, y: torch.FloatTensor = None):
    if self.teach_assist:
        mask = self.create_mask(y)
        y = self.norm1(self.attention1(y, y, y, mask) + y)
    else:
        y = self.norm2(self.attention2(x, x, x) + x)
y = self.norm3(torch.tanh(self.fc1(y)) + y)
    return y
```

### A.3 Standard Transformer

```python
class Transformer(nn.Module):
    def __init__(self, d_model, d_features, d_batch_size, n_layers=6,
                 d_hidden=16, teach_assist=True):
        super(Transformer, self).__init__()
        self.d_model = d_model
        self.n_layers = n_layers
        self.encoders = nn.ModuleList([
            EncTransformerBlock(d_model, d_features, d_features, d_hidden)
            for _ in range(n_layers)])
        self.decoders = nn.ModuleList([
            DecTransformerBlock(
                d_model, d_features, d_features,
                d_hidden, teach_assist)
            for _ in range(n_layers)])
        self.fc_final = nn.Linear(d_model, d_model)

def forward(self, x, y=None):
    for idx in range(self.n_layers):
        x = self.encoders[idx](x)
    for idx in range(self.n_layers):
        y = self.decoders[idx](x, y)
y = torch.tanh(self.fc_final(y))
    return y
```
A.4 Coupled Transformer

class CoupledTransformer(nn.Module):
    def __init__(self, input_size, output_size, d_model, num_layers=2,
                 batch_first=True, dropout=0.1, save_attention=False):
        super(CoupledTransformer, self).__init__()
        self.d_model = d_model
        self.enc1 = nn.Sequential(
            nn.Linear(input_size, self.d_model),
            nn.ReLU(),
            nn.LayerNorm(self.d_model),
            nn.Dropout(dropout),
            *[EncTransformerBlock(self.d_model, 4, dropout, save_attention)
             for _ in range(self.d_model, 4, dropout, save_attention)]
        )
        self.enc2 = nn.Sequential(
            nn.Linear(input_size, self.d_model),
            nn.ReLU(),
            nn.LayerNorm(self.d_model),
            nn.Dropout(dropout),
            *[EncTransformerBlock(self.d_model, 4, dropout, save_attention)
             for _ in range(num_layers)]
        )
        self.dec1 = EncTransformerBlock(
            self.d_model * 2, 4, dropout=dropout, save_attention=save_attention)
        self.fc_out = nn.Linear(self.d_model * 2, output_size)
        self.norm = nn.LayerNorm(output_size)

    def forward(self, x_a, x_b):
        y_a = self.enc1(x_a)
        y_b = self.enc2(x_b)
        y = torch.relu(self.fc_out(self.dec1(torch.cat([y_a, y_b], dim=2))))
        return y
Appendix B

Dataset questionnaire

B.1 Questionnaire

1. How was your trip to the lab today?
2. How did you get here?
3. What is the worst thing that happened to you this past week?
4. It is my understanding that you walked around the lab today, could you tell me about it?
5. Was there anything that stood out to you?
6. Did you see the ring in the file room?
7. Could you describe the ring with as much detail as possible?
8. Did you take the ring?
9. Do you have the ring with you now?
10. Where is the ring now?
11. Could you please explain what happened to the ring?
12. Did you hide the ring?
13. So what made you decide not to take the ring?
14. Is everything that you told me about the ring the truth?
15. What else would you like to add to your account?
16. Where are you going after this study today?
17. What is the best thing that happened to you this past week?
18. When you came in today, what was your first impression?
19. Could you please tell me more about the file room?
20. Did you see the watch in the file room?
21. Could you describe the watch with as much detail as possible.
22. Did you take the watch?
23. Do you have the watch with you now?
24. Where is the watch now?
25. Could you please explain what happened to the watch?
26. Did you hide the watch?
27. So what made you decide not to take the watch?
28. Is everything that you told me about the ring and the watch true?
29. What else would you like to add too your account?
30. Ok, that is all. I will now make my judgment.