Report on Rapport model

[Please, also see the README.md for a brief overview of the rapport model files.]

Contents

Dataset

All rapport models were developed using the Mimicry DB [https://mahnob-db.eu/mimicry/]. We used the FaceFar2 videos (since they are centered and seem to capture the face better than FaceNear) from 53 out of 54 conversation sessions (sessionid 39 was skipped as it was missing the FaceFar2 videos). This resulted in a total of 106 single-person videos, covering 46 unique subjects.

We extracted vision and audio features, determined voice activity, and extended the dataset with automatically generated binary per-frame annotations of: head nods, head shakes, head tilts, smile, gaze away, voice activity, and turn taking.

The whole data preprocessing pipeline can be illustrated as follows (the filenames are relative to the corresponding readme file deep-virtual-rapport-agent/datasets_scripts/mimicry/README.md).

extract_vision_features.sh --> rename_featuresfiles.ipynb --> (0)

(0) --> Annotate vision features with nod, shake, and tilt head gestures using the developed Head Gesture Detector (deep-virtual-rapport-agent/head_gesture_detector/hgd_annotate_frames.ipynb) --> (1)

(0) --> separate_audio_channels.ipynb --> downsample_to_*kHz.sh --> extract_audio_features.sh --> (2)
--> voice_activity_detection --> (3)

(1) & (3) --> smile_gaze_va_turn_annotate_frames.ipynb --> (4)
--> fully_annotate_videos.py

(2) & (3) & (4) --> generate_dataset.ipynb

[Details:] For parsing the dataset xml files, we had to fix several session.xml files, changing each '&' to '&' in xml attributes. All dataset preprocessing scripts were originally in the folder dvra_datasets/mimicry. Then, they were copied into the project repository folder deep-virtual-rapport-agent/datasets_scripts. From now onwards we refer to dataset scripts/files relative to this folder. The folder access contains login details to access the Mimicry dataset.

Vision features extraction

We used OpenFace to get per-frame pose, gaze and AU features (command flags -2Dfp -3Dfp -pdmparams -pose -aus -gaze). Details on which of these vision features were finally used are provided in the section *Generating sequences/segments*.

[Details:] Vision features extraction script: extract_vision_features.sh.
Time logs (containing filepath,sessid,filename,time) from vision feature extraction were merged into one file (vision_feature_extraction_time_log.txt), where for the first few videos the full filepath is missing but it was never needed. The script rename_featuresfiles.ipynb ensures the following naming convention for the extracted features: *sessid_{session id}_P{subject position either 1 or 2}_sid_{subject id}.csv*

Audio/speech features extraction

First, we separated the audio channels of the two subjects from each stereo audio recording (e.g. 2010.11.15.10.33.56_TrimmedAudio_c1_c2.wav, using the script separate_audio_channels.ipynb). The resulting mono audio files follow the same naming convention as the vision features: *sessid_{session id}_P{subject position either 1 or 2}_sid_{subject id}.csv*

Next, we downsampled the mono audio files to 16kHz (downsample_to_16kHz.sh).

We then used OpenSMILE to extract 2 types of audio/speech features, named ‘emobase’ and ‘mfcc’:

- emobase feature set
  - See the config ./opensmile_configs/emobase_csv.conf
  - We used the LLD and delta LLD features directly to get an audio feature vector for each audio frame, instead of the functionals of LLD and delta LLD as was the case of the original config.
  - We changed the output to csv instead of arff files.
  - The emobase feature set contains the following 52 features:
    - 'pcm_intensity_sma', 'pcm_loudness_sma', 'mfcc_sma[1]',
    - 'mfcc_sma[2]', 'mfcc_sma[3]', 'mfcc_sma[4]', 'mfcc_sma[5]',
    - 'mfcc_sma[6]', 'mfcc_sma[7]', 'mfcc_sma[8]', 'mfcc_sma[9]',
    - 'mfcc_sma[10]', 'mfcc_sma[11]', 'mfcc_sma[12]', 'lspFreq_sma[0]',
    - 'lspFreq_sma[1]', 'lspFreq_sma[2]', 'lspFreq_sma[3]', 'lspFreq_sma[4]',
    - 'lspFreq_sma[5]', 'lspFreq_sma[6]', 'lspFreq_sma[7]', 'pcm_zcr_sma',
    - 'voiceProb_sma', 'F0_sma', 'F0env_sma', 'pcm_intensity_sma_de',
    - 'pcm_loudness_sma_de', 'mfcc_sma_de[1]', 'mfcc_sma_de[2]',
    - 'mfcc_sma_de[3]', 'mfcc_sma_de[4]', 'mfcc_sma_de[5]', 'mfcc_sma_de[6]',
    - 'mfcc_sma_de[7]', 'mfcc_sma_de[8]', 'mfcc_sma_de[9]', 'mfcc_sma_de[10]',
    - 'mfcc_sma_de[11]', 'mfcc_sma_de[12]', 'lspFreq_sma_de[0]',
    - 'lspFreq_sma_de[1]', 'lspFreq_sma_de[2]', 'lspFreq_sma_de[3]',
    - 'lspFreq_sma_de[4]', 'lspFreq_sma_de[5]', 'lspFreq_sma_de[6]',
    - 'lspFreq_sma_de[7]', 'pcm_zcr_sma_de', 'voiceProb_sma_de', 'F0_sma_de',
    - 'F0env_sma_de'
- mfcc feature set
  - See the config ./opensmile_configs/MFCC12_0_D_A_extra.conf
  - We extended the MFCC12_0_D_A.conf as follows:
    - Prepended: Probability of voicing, Pitch (F0), F0 envelope, intensity, loudness, log energy, 0^{th} MFCC (in the original config MFCC12_0_D_A.conf the 0^{th} MFCC is replaced by log energy)
    - Took 1^{st} and 2^{nd} deltas of all these (6 + 13) features => 57-d feature vector.
    - We changed the output to csv instead of htk files.
  - The mfcc feature set contains the following 57 features:
    - 'voiceProb', 'F0', 'F0env', 'pcm_intensity', 'pcm_loudness', 'pcm_LOGenergy',
    - 'pcm_fftMag_mfcc[0]', 'pcm_fftMag_mfcc[1]', 'pcm_fftMag_mfcc[2]',

- 'pcm_fftMag_mfcc[3]', 'pcm_fftMag_mfcc[4]', 'pcm_fftMag_mfcc[5]',
- 'pcm_fftMag_mfcc[6]', 'pcm_fftMag_mfcc[7]', 'pcm_fftMag_mfcc[8]',
- 'pcm_fftMag_mfcc[9]', 'pcm_fftMag_mfcc[10]', 'pcm_fftMag_mfcc[11]',
- 'pcm_fftMag_mfcc[12]',
- 'voiceProb_de', 'F0_de', 'F0env_de', 'pcm_intensity_de', 'pcm_loudness_de',
- 'pcm_LOGenergy_de',
- 'pcm_fftMag_mfcc_de[0]', 'pcm_fftMag_mfcc_de[1]', 'pcm_fftMag_mfcc_de[2]',
- 'pcm_fftMag_mfcc_de[3]', 'pcm_fftMag_mfcc_de[4]', 'pcm_fftMag_mfcc_de[5]',
- 'pcm_fftMag_mfcc_de[6]', 'pcm_fftMag_mfcc_de[7]', 'pcm_fftMag_mfcc_de[8]',
- 'pcm_fftMag_mfcc_de[9]', 'pcm_fftMag_mfcc_de[10]',
- 'pcm_fftMag_mfcc_de[11]', 'pcm_fftMag_mfcc_de[12]',
- 'voiceProb_de_de', 'F0_de_de', 'F0env_de_de', 'pcm_intensity_de_de',
- 'pcm_loudness_de_de', 'pcm_LOGenergy_de_de',
- 'pcm_fftMag_mfcc_de_de[0]', 'pcm_fftMag_mfcc_de_de[1]',
- 'pcm_fftMag_mfcc_de_de[2]', 'pcm_fftMag_mfcc_de_de[3]',
- 'pcm_fftMag_mfcc_de_de[4]', 'pcm_fftMag_mfcc_de_de[5]',
- 'pcm_fftMag_mfcc_de_de[6]', 'pcm_fftMag_mfcc_de_de[7]',
- 'pcm_fftMag_mfcc_de_de[8]', 'pcm_fftMag_mfcc_de_de[9]',
- 'pcm_fftMag_mfcc_de_de[10]', 'pcm_fftMag_mfcc_de_de[11]',
- 'pcm_fftMag_mfcc_de_de[12]'

Further details about the OpenSMILE configs:

- Smoothing of all LLD using the moving average filter is performed in the emobase config, but it is not done for mfcc.
- In our configs, we unified the attribute “frameCenterSpecial = left” for all frames generated, so that the output csvs start with the same timestamp 0.0.

We also prepared the audio feature extraction using AudioSet+VGGish, but it was not further used.

- Downloaded the latest version & installed from the link:
- https://github.com/tensorflow/models/tree/master/research/audioset/vggish
- Created the script ./audio_features_extraction_vggish/extract_audio_features.py based on kalins_extract_audio_features.py and vggish_inference_demo.py
- Set the hop size to 200 ms => audio features generated at 5 Hz.
- 128-D audio features
- Finally, not used since for the real-time embedding into the Multisense system OpenSMILE might be better.

Voice activity detection

Voice activity detection (VAD) was necessary to determine time intervals when a subject is speaking, so that we could use the data from these time intervals to learn the mapping from speaker’s audio-visual features to listener’s behavior.

For each mono audio, there is a main speaker whose voice-active intervals we wish to determine. Since, the other subject might still be audible in the recording, we also employed speaker diarization where possible (namely, the methods 3, 4, and 5 below). In the case of the first 2 methods (OpenSMILE and WebRTC VAD) we had to assume that all the voice activity is by the main speaker, which explains why the latter methods give better results.

We experimented with several methods for VAD:

1. OpenSMILE VAD

   We ran ./voice_activity_detection/vad_opensmile/run_vad_opensmile.sh to get raw voice activity (in the range -1 to 1) for each timestamp and saved the results as csv files. We then threshold the raw voice activity based on its

median to get a binary voice activity and generate plots that show how the detected raw/binary voice activity aligns with the audio waveform (./voice_activity_detection/vad_opensmile/plot_threshold_filter_vad_opensmile.ipynb).
=> The detected voice activity is very inaccurate (see the plots in the Jupyter notebook).

2. WebRTC VAD (by Google)

Obtained from https://cmusphinx.github.io/wiki/asr/vad/ and https://github.com/wiseman/py-webrtcvad/ to create the script ./voice_activity_detection/vad_webrtc/run_vad_webrtc.py
WebRTC VAD only accepts 16-bit mono PCM audio, sampled at 8000, 16000, 32000 or 48000 Hz. A frame must be either 10, 20, or 30 ms in duration.
Experimented with several aggressiveness factors (0, 1, 2, 3). The aggressiveness factor 3 gave the best results when looking at the plots of the obtained binary voice activity and audio waveform (./voice_activity_detection/vad_webrtc/plot_vad_webrtc.ipynb).
=> Seems better than the OpenSMILE VAD.

3. Google ASR speech-to-text (STT)

Scripts based on Leili Tavabi's ./voice_activity_detection/vad_google_stt/from_leili.
First, we converted the audio to 16 kHz FLAC format (./voice_activity_detection/vad_google_stt/preprocess_audio.sh).
We ran (./voice_activity_detection/vad_google_stt/run_google_stt.py) the Google ASR STT service with speaker diarization and en-GB model (a special video model was not available for en-GB).
Setup notes:

- on Google Cloud STT API download credentials (json)
- make sure you have roles: [under IAM] Owner, Storage Admin, Storage Object Admin
- in Google Storage: create bucket and upload files
- in Google Console
  - pip install --upgrade google-cloud-speech
  - upload credentials (json) to home folder
  - export GOOGLE_APPLICATION_CREDENTIALS="./Rapport-2f4c2bc0eb53.json"
  - upload your script and run
  - https://cloud.google.com/storage/docs/quickstart-gsutil
    - list files: https://cloud.google.com/storage/docs/listing-objects
  - data input from Google Cloud Storage – blob, bucket
  - data output to home folder on Google Cloud Console [to save to Google Cloud Storage directly: https://stackoverflow.com/questions/56596951/save-pandas-data-frame-to-google-cloud-bucket/56597035]

We saved all results for words and sentences, and also the raw dumps as JSON.
However, the speaker tags were not assigned for all recordings:

- Both speaker tags 1 and 2 present
  - 84 recordings
  - => correct diarization => use timings of words only from the last result (result is a field name in the JSON dump)
- Speaker tag 1 only
  - 0 recordings
- Speaker tag 2 only
  - 4 recordings
  - => only one speaker identified (assume it is the main speaker we wish to identify) => use timings of words from the last result only
- No speaker tags
  - 18 recordings
  - => speaker diarization failed => take words from all results and assume they are all from the main speaker we wish to identify
  - in plots denoted by speaker tag 1 [assigned by us]

Plots of voice activity for each speaker based on timings of individual words were generated and saved by ./voice_activity_detection/vad_google_stt/plot_google_stt.ipynb
=> Better than OpenSMILE or WebRTC VADs.
=> Speaker diarization failed on some recordings completely.

4. IBM Watson speech-to-text (STT)

   As suggested by the IBM Watson website for our case (https://cloud.ibm.com/docs/services/speech-to-text?topic=speech-to-text-audio-formats&_ga=2.112000070.31787808.1569539847-2082697583.1569539847#frequency), we used 8kHz audio (downsample_to_8kHz.sh), en-GB NarrowbandModel and the version ibm-watson>=3.1.1 (./voice_activity_detection/vad_ibm_watson_stt/run_ibm_watson_stt.py).
   For 1 recording (sessid_36_P2_sid_40) 3 speakers were identified => we proceeded as in the case of 2 identified speakers.
   For 4 recordings (sessid_06_P2_sid_21, sessid_16_P1_sid_24, sessid_26_P2_sid_32, sessid_53_P2_sid_08) only 1 speaker was identified => assume this is the main speaker.
   Plots of voice activity for each speaker based on timings of individual words were generated and saved by ./voice_activity_detection/vad_ibm_watson_stt/plot_ibm_watson_stt.ipynb (cell 1)

   To map audio filename to the speaker tag of the main speaker, we manually created speaker diarization csvs (./voice_activity_detection/vad_google_stt/speaker_diarization_google.csv and ./voice_activity_detection/vad_ibm_watson_stt/speaker_diarization_ibm_watson.csv, for Google STT and IBM STT respectively). This was done by looking at the plot of audio waveform and binary voice activity of all speaker tags from STT.

   Overall comparison with IBM Watson STT results:
   => Better than OpenSMILE or WebRTC VADs.
   => Compared to Google ASR STT, the binary voice activity is not smoothed but does not seem to fail in cases where Google does.

   We thus performed smoothing of the binary voice activity (./voice_activity_detection/vad_ibm_watson_stt/plot_ibm_watson_stt.ipynb (cell 2)) as follows:
   - if 2 consecutive voice-active (VA) intervals are less than 200 ms apart, join them
     - based on http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.6135&rep=rep1&type=pdf
     - i.e. maximum pause duration to join voice-active intervals is 200 ms
     - => Looks better

   Finally, having the correct identities of the main speaker (by speaker diarization csvs above), we saved the new plots of smoothed voice activity, and also the time intervals of the smoothed voice activity for each audio file in a csv (./vad_ibm_watson_stt/plot_ibm_watson_stt.ipynb (cell 2)). These time intervals were later used for correct segmentation of the data when creating sequences.
   We also plotted the distributions of durations of the resulting VA intervals (./vad_ibm_watson_stt/plot_ibm_watson_stt.ipynb (cell 3)).

5. There is also Amazon Transcribe

   We found they provide the following services, but we have not tried them.
   - Channel identification (input is stereo audio)
     - In our case, this could be directly applied on stereo audio files.
   - Speaker identification (input is mono audio)

To double-check the correct separation of audio channels that the content of audio file correctly matches the P1/P2 (the position and thus also the correct subject id of the speaker), we manually looked at the speaker activity in preview videos (where both subjects are present) and audio waveforms in the above-obtained speaker diarization plots.

Annotations

The Mimicry database comes only with the annotations of mimicry which is not of much use for our project where we need per-subject annotations. Therefore, we annotated the Mimicry dataset in an automatic way with the following binary per-frame labels:

- head nod
- head shake
- head tilt
- smile
- gaze away
- voice active
- take turn

Figure 1 shows the whole annotation procedure for rapport model data.

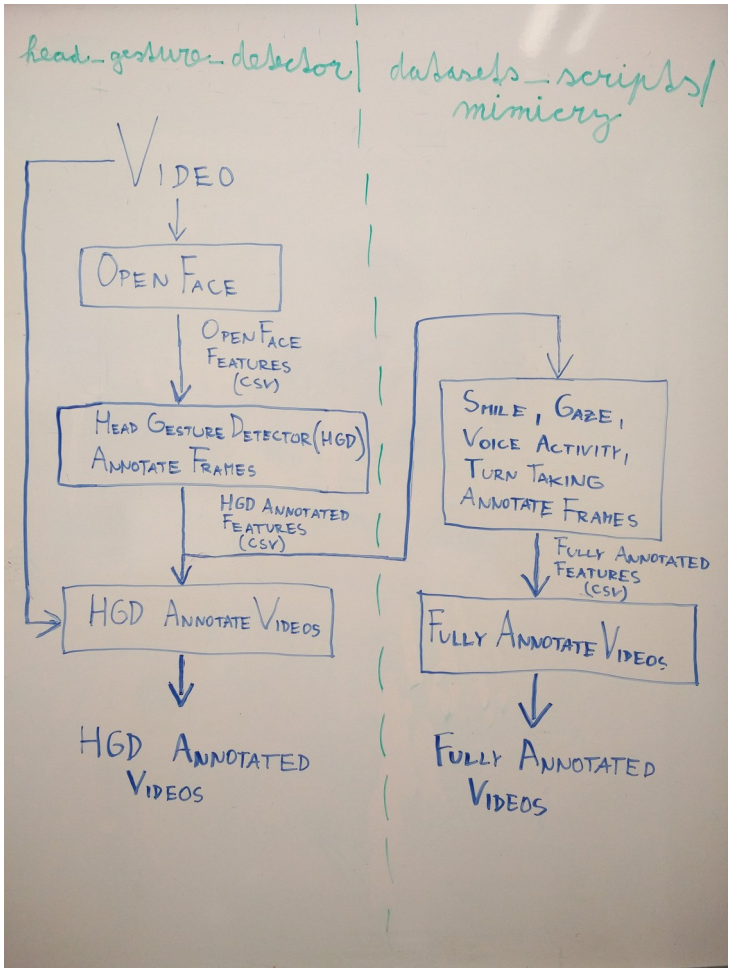

Figure 1: Annotation procedure for rapport model data.

For head gestures (nod, shake, tilt), we used our own Head Gesture Detector (deep-virtual-rapport-agent/head_gesture_detector/hgd_annotate_frames.ipynb) to extend the vision features csvs with these labels.

The remaining labels (smile, gaze away, voice active, take turn) were also added to the vision features csvs (in-place) using the script deep-virtual-rapport-agent/datasets_scripts/mimicry/smile_gaze_va_turn_annotate_frames.ipynb, utilizing the following rules:

- Smile
  - If the continuous AU intensities ‘ AU06_r’ and ‘ AU12_r’ (by OpenFace) are both >= 1.0, then the frame is labeled with 1 and with 0 otherwise.
  - For this, we considered sincere and involuntary (Duchenne) smiles, based on [9] from https://en.wikipedia.org/wiki/Facial_Action_Coding_System
  - We smoothed the obtained smile labels using the median filter with the kernel size of 15 frames (500ms), as [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5826373/] states that the total duration of a genuine smile can range from 500 to 4,000 ms.
- Gaze away
  - We used the gaze angles (gaze_angle_x, gaze_angle_y) by OpenFace and defined thresholds based on their statistics (see the plots of distributions in mimicry_stats). In particular, for each video we calculated the mean and standard deviation of the distribution of gaze_angle_x and gaze_angle_y. We then labeled frames as gaze away if its gaze_angle_x or gaze_angle_y was outside of the corresponding mean +/- standard deviation. This, of course, assumes that most (~68%) of the time people were looking at each other, which can be supported by “… researchers have found considerable variability in [both how often speakers gaze at a listener (20–65% of the time) and] **how often listeners gaze at a speaker (30–80%)** ...” [Kendon A. Some functions of gaze-direction in social interaction. Acta Psychol 1967. 10.1016/0001-6918(67)90005-4]
  - After viewing the original videos with these gaze annotations, we noticed that many frames are incorrectly labeled due to eye blinking. We thus extended the rule to check if eyes are closed. Specifically, the gaze away annotation was assigned only if the eyes were open (intensity of AU45 < 1) and the above-described condition on gaze angles was satisfied.
  - We also smoothed the gaze labels using the median filter with the kernel size of 59 frames (~2s), based on previous works from [http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.9028&rep=rep1&type=pdf]
    - “*Most studies report a wide variability in the duration of individual glances. Argyle & Ingham [3] report gaze durations anywhere between 2 and 7 seconds, with an average of about 3.0 s. Rutter et al. [13] report durations between 2.6 s and 6.0 s with an average of 4.4 s. In their experiment, Garau et al. [9] used mean gaze durations between 1.6 and 2.5 s. Colburn et al. [6] used a mean gaze duration between 4.0 and 6.0 s. (2.0 s during mutual gaze).*”
  - The distributions of gaze angles (gaze_angle_x, gaze_angle_y) extracted (using OpenFace) from the Mimicry dataset can be found in deep-virtual-rapport-agent/data_analysis/mimicry_stats.ipynb
- Voice active
  - We annotated the voice activity based on the time intervals recorded in deep-virtual-rapport-agent/datasets/mimicry/voice_activity_detection/vad_ibm_watson/voice_activity_ibm_watson.csv
- Take turn
  - We set 1 in the first frame after each voice active interval, and 0 otherwise.
  - These take turn annotations were changed when generating the final dataset of sequences (see the section *Generating sequences/segments*).

Annotated videos

- Using the per-frame annotations of vision features csvs and the original videos, we created annotated videos with in-video annotations of listener behavior, namely, head nod, head shake, head tilt, gaze away, voice activity, and turn taking. See the script deep-virtual-rapport-agent/datasets_scripts/mimicry/fully_annotate_videos.py
- This allowed us to qualitatively evaluate the performance of the employed automatic annotation methods.

Generating sequences/segments

Having vision features, audio features, voice activity, and annotations for each listener and each speaker, we generated a final dataset of sequences/segments (of length WINDOW_SIZE) of listener features and speaker labels.
Using the script deep-virtual-rapport-agent/datasets_scripts/mimicry/generate_dataset.ipynb, we generated the dataset as follows:

- From each of the 53 sessions, we extracted the data twice (both ways): first, treating the subject at position P1 as listener and the subject at position P2 as speaker, and second, treating the subject at position P2 as listener and the subject at position P1 as speaker.
- We downsampled (grouped the nearby frames and averaged) all the data to a common data rate of 5 Hz (so that the model is trained on data of the same rate as the online prediction is expected to be clocked at). After this kind of averaging, we had to restore the binary annotations in the vision features dataframes by simple rounding.
- We also made new take turn annotations (as compared to the procedure described in the section *Annotations*), annotating all (WINDOW_SIZE – 1) frames after a voice-active region with 1 (fewer frames if there was another voice-active region).
- We then slid a window of WINDOW_SIZE frames over the data. For each speaker's voice-active region, we extracted a sequence of audio and video features only if the target prediction frame of the sequence was within the voice-active region, or at least one frame of the sequence was within the voice-active region and the prediction frame of the sequence was not within the next voice-active region.
- We only generated sequences that didn't require padding, which discarded only a small fraction (about 118/238,077) of sequences but eliminated the need for padding and masking within PyTorch. Thus, all the generated sequences were of the same length (WINDOW_SIZE).
- We saved the sequences of features as npy files (separate for audio and vision), and the sequence metadata and labels were saved in one large metadata+labels csv file (i.e., dataset file).
- We made several versions of the dataset:
  - v0 (11 vision features, 52 audio features, 7 binary targets)
    - As the speaker's vision features, we used the following 11 features:
      - # First-order differences of head translations by OpenFace
      - 'diff_ pose_Tx',
      - 'diff_ pose_Ty',
      - 'diff_ pose_Tz',
      - # First-order differences of head rotations by OpenFace
      - 'diff_ pose_Rx',
      - 'diff_ pose_Ry',
      - 'diff_ pose_Rz',
      - # Mean normalized (per-recording) head rotations by OpenFace (as a proxy for gaze away prediction)
      - 'unorm_ pose_Rx',
      - 'unorm_ pose_Ry',
      - # Mean normalized (per-recording) gaze angles by OpenFace
      - 'unorm_ gaze_angle_x',
      - 'unorm_ gaze_angle_y',
      - # Smile binary annotation assigned as described in the section *Annotations*
      - 'smile'
    - As the speaker's audio/speech features, we used all 52 emobase features described in the section *Audio/speech feature extraction* (mfcc features were not further used yet).
    - As the target labels we used 6 annotations (nod, shake, tilt, gaze away, voice active) from the listener's vision dataframes and take-turn annotations from the speaker's vision dataframe (since the speaker's take-turn annotation indicates when it is a good time for a listener to start talking).
      - Predicting the binary label of voice activity of the listener can be used for paraverbal generation. For example, at inference time, one can randomly choose to generate one from a predefined set of paraverbals.
    - (For instance, it took about 1 hour to generate 238,077 sequences (1.2 GB) for WINDOW_SIZE = 8.)

    - This was done for WINDOW_SIZE = 8, 16, 32, resulting in 237 959, 272 264, 312 820 sequences, respectively.
- v1 (12 vision features, 53 audio features, 7 binary targets)
    - We replaced the binary ‘smile’ vision feature with 2 continuous smile indicators, namely, ‘ AU06_r’ and ‘ AU12_r’.
    - We also added a speaking time audio feature that explicitly expresses the time the speaker is active for in the current voice-active interval.
    - This was done for WINDOW_SIZE = 8, 16, 32.
- v2 (27 vision features, 53 audio features, 7 binary targets)
    - We added 15 more continuous AU vision features: ' AU01_r', ' AU02_r', ' AU04_r', ' AU05_r', ' AU07_r', ' AU09_r', ' AU10_r', ' AU14_r', ' AU15_r', ' AU17_r', ' AU20_r', ' AU23_r', ' AU25_r', ' AU26_r', ' AU45_r'.
    - This was done for WINDOW_SIZE = 8, 16, 32.
- v3 (27 vision features, 53 audio features, 7 binary targets)
    - We kept the common data rate at 30 Hz, so that no averaging of vision features was necessary.
    - This was done for WINDOW_SIZE = 32, resulting in 1,367,904 sequences.

The generated sequences and the metadata+labels file (i.e., dataset file) are then loaded by the PyTorch dataset loaders defined in deep-virtual-rapport-agent/rapport_model/data.py

Figure 2 illustrates the inputs and outputs of rapport models. It also shows the sliding window approach used to extract sequences/segments.

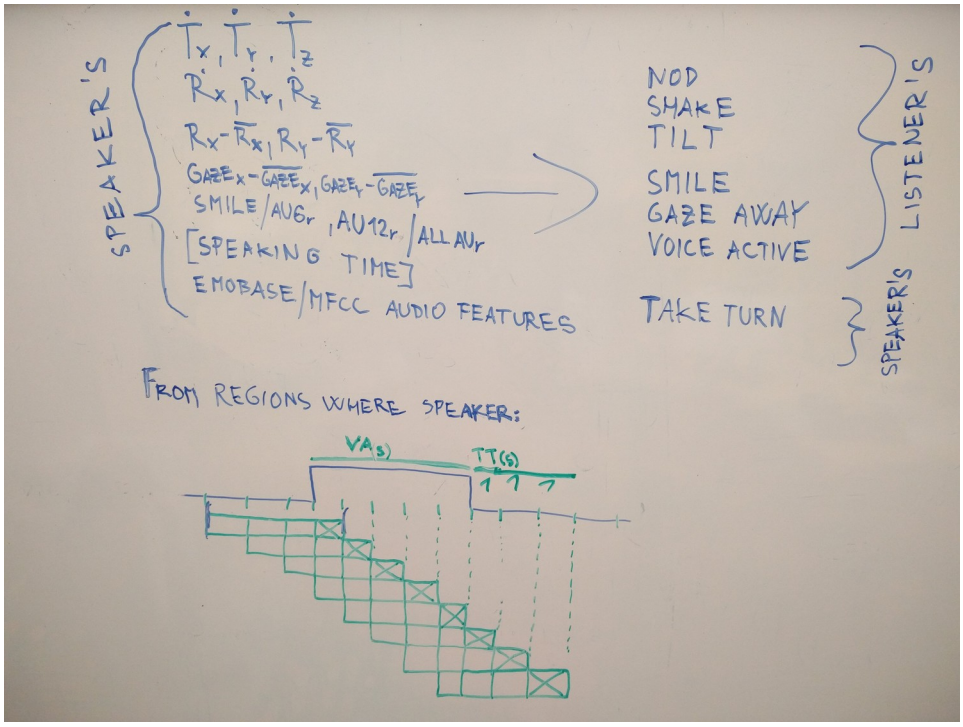

Figure 2: Inputs and outputs of rapport models, and the sliding window approach used to extract sequences/segments. VA(s) denotes frames with voice activity of the speaker and TT(s) denotes frames of turn-taking.

Models

Using the PyTorch library, we implemented various machine learning models to learn the mapping from the speaker's vision and/or audio feature sequence to the listener's 1 or up to 7 binary label(s) at a target frame (the last frame in the sequence). Please, see the script deep-virtual-rapport-agent/rapport_model/model.py

There are 2 model types to process the fixed-length sequences:
- GRU-based base classifier
- Temporal Convolutional Network (TCN) classifier (https://github.com/locuslab/TCN)

For each model type, there are unimodal (vision or speech) as well as multimodal (vision and speech) sequence model variants:
- Unimodal model
  - Inputs a batch of either vision or speech sequences
  - Consists of:
    - Sequence layer GRU/TCN
    - Fully-connected layer(s)
  - Outputs a batch of target labels (1 or up to 7 per frame)

- Multimodal model
  - Consists of:
    - 2 submodels (one for each modality)
      - Inputs a batch of vision/speech sequences
      - Sequence layer GRU/TCN
    - Concatenation of the outputs from the submodels
    - Fully-connected layer(s)
  - Outputs a batch of target labels (1 or up to 7 per frame)

Training and testing

The rapport models can be trained and tested using a subject-dependent or subject-independent (leave-one-subject-out (LOSO)) cross-validation. In the case of subject-dependent cross-validation, the sequences from all subjects are shuffled and then split to create train and test partitions for each fold. By default, we used 10 folds. In the case of subject-independent LOSO cross-validation, the sequences are grouped by subjects (namely, by the listener subject id, since the listener is the target subject) and in each fold the data of one subject form a test partition and the remaining data form the train partition. In both cross-validation variants, within each fold we used a fixed fraction (default value of 0.2) of the training data for validation.

We used the following default training settings:
- Batch size: 32
- Number of training epochs: 100
- Loss: binary cross-entropy (with logits)
  - We weight the loss according to class weights, for each class separately.
  - When training and predicting multiple outputs (e.g. nod and shake), we compute the loss and all the evaluation metrics for each class separately and also combined, averaging over all classes (the key 'overall' in the metrics dictionary). This way the overall loss is computed and optimized.
  - For correct (manual) loss normalization in PyTorch when weighting the loss, see https://discuss.pytorch.org/t/passing-the-weights-to-crossentropyloss-correctly/14731/10
    - *"if your loss function uses reduction='mean', the loss will be normalized by the sum of the corresponding weights for each element. If you are using reduction='none', you would have to take care of the normalization yourself."*
- Learning rate: 0.0001
- Weight decay (L2 norm): 0.0

- We monitored either the validation loss or the validation balanced accuracy metric to decide whether to update the best model checkpoint.

The training and testing scripts can be found at deep-virtual-rapport-agent/rapport_model/train.py and deep-virtual-rapport-agent/rapport_model/test.py respectively.

To visualize the training, we used tensorboard as follows:

- Install tensorboard from: https://pytorch.org/docs/stable/tensorboard.html
- Run the command: tensorboard --logdir logs
- In your browser, go to: http://cronos:6006/
- Filter plots by tag names, e.g.:
  - *fold_01/train/nod/bacc* shows the training balanced accuracy of nod class/label for fold 1
  - *fold_.*/val/nod/loss* shows the validation loss of nod class/label for all folds

Final training

After choosing the best data representation, model architecture and parameters, a final training is performed on the whole dataset. In this case, no hold-out test set is created and no cross-validation is carried out. However, we still perform a single-split validation - either subject-dependent or subject-independent. The validation set size is given by a fixed fraction 'validation_set_size' (default value of 0.2). In the case of subject-dependent validation, the sequences from all subjects are shuffled and one train-validation split is made. In the case of subject-independent validation, the sequences are grouped by subjects (namely, by the listener subject id, since the listener is the target subject) and the number of subjects in the validation set is determined by the validation set size, so that the resulting actual validation set size will be at greater than or equal to the validation set size requested/specified by 'validation_set_size'. The resulting model is then ready to be used for online prediction.

The script for final training can be found at deep-virtual-rapport-agent/rapport_model/final_train.py

Online prediction

(Currently, just a skeleton code deep-virtual-rapport-agent/rapport_model/predict_online.py)
During online prediction a camera and microphone will capture the speaker. This audio-visual data will be fed into OpenFace and OpenSMILE to extract vision and audio features respectively. These features will be further preprocessed to comply with the model input format. The final model (from final training) will then input audio and video feature vectors from the most recent timestep and make a prediction. The predicted labels will be used to drive the behavior of a virtual human by sending BML commands to Smartbody.

It is necessary to perform a calibration recording (using the script, currently, just a skeleton code deep-virtual-rapport-agent/rapport_model/record_normalization_params.py) prior to online prediction in order to record normalization parameters, namely, 2 mean head rotations (from ' pose_Rx' and ' pose_Ry' by OpenFace) and 2 mean gaze angles (from ' gaze_angle_x' and ' gaze_angle_y' by OpenFace) while the speaker is looking directly at the virtual human. These means are necessary to preprocess the vision features from OpenFace for the model.

For online prediction the batch size is set to 1 and the GRU-based models are treated as statefull (internal states of GRUs are not reset between batches which allows us to feed one timestep of data at a time). For more details on statefull/less RNNs see:

- https://discuss.pytorch.org/t/stateful-rnn-example/10912
- https://discuss.pytorch.org/t/solved-training-a-simple-rnn/9055
- Functions in the PyTorch model
  - forward() - used for training, state is reset for each sequence
  - predict_online() - used for online prediction, state is preserved between batches and requires batch_size = 1

However, the TCN models need to input whole sequences, the same way as during the training.

Experiments

First, we tried to train a single model to predict all 7 target classes/labels. However, learning such a complex multi-task problem without much insight into training of models for individual tasks is a difficult starting point. Therefore, we trained a separate model for each task (target class).

For training configurations in the following experiments, please, see deep-virtual-rapport-agent/rapport_model/run.sh

1. Comparing unimodal speech, unimodal vision and multimodal models
   - Using the dataset v0
   - Comparison based on train/val balanced accuracy (bacc) [approximate values]
   - Train curves all increasing; validation curves not very stable
   - nod
     - speech => 0.70 train — val: early overfit around 0.56
     - vision => 0.61 train — val: overfit around 0.56
     - multi => 0.73 train — val: early overfit around 0.57
   - shake
     - speech => 0.83 train — val: overfit around 0.66
     - vision => 0.66 train — val: overfit around 0.59
     - multi => 0.84 train — val: early overfit around 0.66
   - tilt
     - speech => 0.77 train — val: overfit around 0.62
     - vision => 0.62 train — val: overfit/plateau around 0.57
     - multi => 0.79 train — val: overfit around 0.62
   - smile
     - speech => 0.74 train — val: early overfit around 0.58
     - vision => 0.67 train — val: overfit around 0.65
     - multi => 0.83 train — val: early overfit around 0.64
   - gaze_away
     - speech => 0.74 train — val: early overfit around 0.53
     - vision => 0.57 train — val: increasing, so far 0.52
     - multi => 0.76 train — val: early overfit around 0.53
   - voice_active
     - speech => 0.87 train — val: overfit around 0.77
     - vision => 0.70 train — val: overfit around 0.65
     - multi => 0.89 train — val: overfit around 0.78
   - take_turn
     - speech => 0.92 train — val: overfit around 0.88
     - vision => 0.67 train — val: overfit/plateau around 0.64
     - multi => 0.92 train — val: overfit around 0.89

   - We can conclude that only speech-based voice activity prediction and speech-based turn taking prediction perform reasonably well.

2. Experiments with unimodal vision nod [comparing bacc and loss using tensorboard]
   - Using the dataset v0
   - batch_size=128 not better than the original 32
   - Leaving out one hidden layer after RNN (i.e, reducing the network capacity) did not help
     - Having 2 instead of 1 layer after RNN gave more stable training
   - Increasing network capacity to 64 GRU units => seemed to help
   - Window size 16 => better
   - Window size 4 => worse
   - 128 GRU units => higher val bacc
   - 3 fully-connected layers after GRU => not better than 2

- There were minimal differences between the various settings, but the following seemed the best:
  - Window size 32 (> 6 sec)
  - 128 GRU
  - 2 fully-connected layers
  - Overfit on validation set in about 15 epochs

3. Comparing the dataset v0 vs v1
   - smile based on vision
     - Bigger difference is caused by changing the window size than by the dataset version v0/v1 (window_size = 32 might be the best).
     - Validation: overfit immediately and loss just increases.
     - v1 can get slightly lower train loss => continuous smile features slightly help.
   - gaze based on audio
     - Bigger difference is caused by changing the window size than by the dataset version v0/v1 (window_size = 32 might be the best).
     - Validation: overfit immediately and loss just increases.
     - v1 slightly better on validation => speaking time audio feature slightly helps.
   - nod based on audio
     - Bigger difference is caused by changing the window size than by the dataset version v0/v1 (window_size = 16 might be the best).
     - v1 slightly better on train (not very convincing) => speaking time audio feature slightly helps.
   - Overall, it is not very clear if v1 is better than v0, but v1 certainly did not degrade the performance. The large optimal window size of 32 frames (> 6 seconds) does not seem right.

4. Comparing the dataset v1 vs v2
   - Compare v2 vs v1: nod-vision, smile-vision, gaze-vision; smaller network (64u, 2fc); looking at the first 3 folds
   - nod based on vision
     - v2: train bacc and loss much better (still the best is window_size = 32).
     - Validation: not very clear, v2 might be slightly better, in terms of bacc.
     - val loss only increases
   - smile based on vision
     - v2: train bacc and loss much better (still the best is window_size = 32).
     - Validation: not very clear, v1 might be slightly better, in terms of bacc and loss.
     - val loss only increases
   - gaze based on vision
     - v2: train bacc and loss much better (still the best is window_size = 32).
     - Validation: not very clear what is better.
     - val loss only increases.
   - Overall, it is not very clear if v2 is better than v1, but v2 certainly did not degrade the performance. The large optimal window size of 32 frames (> 6 seconds) does not seem right.

5. Comparing smaller network sizes for nod prediction based on vision
   - 32/16 window size, 32/16 GRU units and 2 fully-connected layers
     - 32 vs 16 GRU units: 32 GRU units give better train loss & bacc.
     - Not much difference in train between 32/16 window size.
     - Overfit in about 20 steps, below 60% validation bacc.
     - => Not clear what is the best configuration.

6. Comparing TCN vs GRU
   - 16/32 window size, 32 GRU/TCN units in 1 layer
   - nod based on vision
     - GRU has better train bacc and loss
     - TCN val loss does not explode so much, but does not optimize that well.

- nod based on speech
  - TCN does not train faster than GRU (which is interesting, since the TCN should train faster)
  - same characteristics as for vision:
    - GRU has better train bacc and loss
    - TCN val loss does not explode so much, but does not optimize that well.

7. Comparing various numbers of TCN layers
   - Tried TCN with 2/3 layers of 32 units, for nod based on vision and nod based on speech, and for 16/32 window size.

8. Comparing the dataset v2 vs v3
   - Overall, it is not very clear if v3 is better than v2, but v3 certainly did not degrade the performance.

Future work & ideas to try

- Data
  - Use the smaller CCDb dataset instead of the large Mimicry DB and predict only the head nod, shake and tilt labels that are provided in CCDb (this avoids the need for noisy automatic annotation as was the case with Mimicry data).
  - Alternative automatic gaze annotation based on geometry
    - Calculate mean head translations of each head over recording (from Tx,Ty,Tz by OpenFace) and use these mean translations to estimate angles (a_x, a_y) between the camera and the head. The angles (a_x, a_y) will be then subtracted from gaze angles and the resulting delta angles will be checked if they are within an average face/head size (https://en.wikipedia.org/wiki/Human_head#/media/File:AvgHeadSizes.png) at the particular distance (given by mean head translation).
    - See "*Modeling human visual attention in multiparty open world dialogs*" on how to normalize gaze.
  - Use mel spectrograms for audio features (see PyTorchAudio).

- Model
  - Try generative models such as CVAE (for a recurrent version see *"Anticipating many futures: Online human motion prediction and synthesis for human-robot collaboration"*).
  - Also can try Quasi-Recurrent Neural Network (QRNN)
    - Alternative to TCN (might be better)
    - PyTorch code: https://github.com/salesforce/pytorch-qrnn
  - Some PyTorch model code that might be helpful: https://github.com/rasbt/deeplearning-models
  - Place each model class into a separate file.
  - Try multi-task learning
    - For overview, see "An Overview of Multi-Task Learningin Deep Neural Networks" https://arxiv.org/pdf/1706.05098.pdf
    - Also, see the related work "Learning Off-line vs. On-line Models of Interactive Multimodal Behaviors with Recurrent Neural Networks" https://hal.archives-ouvertes.fr/hal-01609535/file/dan_PRL2017_R2_v2.pdf
      - jointly model speech, gaze and gestures of two subjects involved in a collaborative task
      - predict the instructor's co-verbal gestures and region of interest fixated by the instructor's gaze given his verbal activity and the interlocutor's gestures
      - use multi-task LSTM
      - perform both on-line and off-line prediction using LSTM and BiLSTM models respectively
      - (for evaluation) perform coordination histogram to capture global coordination patterns between different modalities given synchronous streams of discrete events

- Training
  - Plot weight gradients in tensorboard.

- Smartbody
  - download from https://sourceforge.net/projects/smartbody/
  - read the docs http://smartbody.ict.usc.edu/HTML/documentation/SB/Running-SmartBody-as-a-Standalone-Application_10027085.html