Technical Assessment for Tech Lead – Deep Learning & Algorithm Development

Objective

You are tasked with designing a solution to estimate heart rate (HR) from synthetic PPG signals. The dataset provided simulates real-world conditions with various types of noise. You will solve this problem using two approaches:

1. A classical signal processing method.
2. A deep learning model implemented in PyTorch.

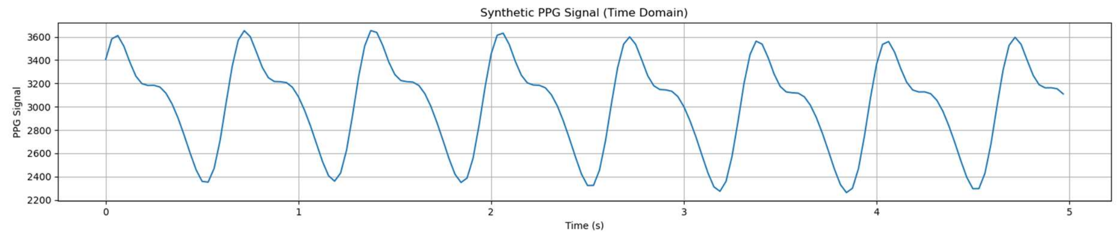
The provided functions will be tested on a separate dataset sampled from the same distribution as the training data.

Background on Photoplethysmography (PPG)

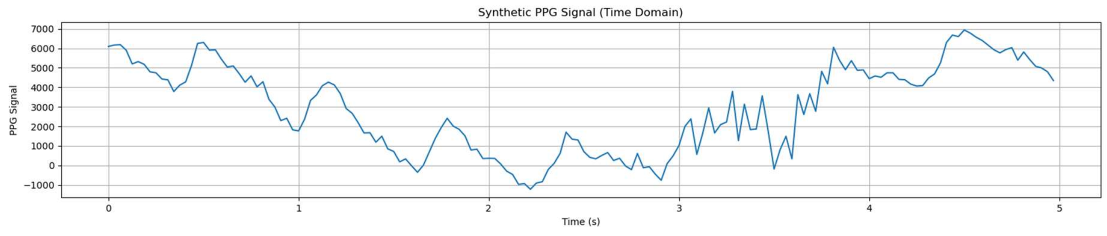
Photoplethysmography (PPG) is a non-invasive optical method used to measure volumetric changes in blood flow. It involves using a light source and a photodetector to detect changes in light absorption due to blood flow variations. PPG signals are widely used in wearable devices for monitoring heart rate (HR), oxygen saturation, and other physiological parameters.

Real-world PPG signals often contain noise caused by motion artifacts, ambient light, or variations in skin tone, which complicates the accurate extraction of physiological information. Thus, robust algorithms are required to handle these challenges effectively.

Below is a sample visualization of a clean PPG signal:



Below is a sample visualization of a "Real-world" PPG signal:



Challenge 1

Dataset

You will receive a file named ppg_samples_with_hr.parquet, which contains the following columns:

- **signal**: An array of 160 float values representing 5 seconds of PPG data sampled at 32 Hz.
- **hr**: The corresponding heart rate (in beats per minute) as a float.

Task Instructions

1. **Classical Method**:
    - Implement a function to estimate HR from PPG signals using a classical approach (either time-domain or frequency-domain analysis).
    - Your solution should achieve an absolute error of less than **6.5 BPM** on the test set.
2. **Deep Learning Method**:
    - Design and train a deep learning model in PyTorch to estimate HR from PPG signals.
    - The model should achieve an absolute error of less than **3.5 BPM** on the test set.
3. **Evaluation and Analysis:**
    - Provide a clear explanation of the methodologies used.
    - Explain how you measured the quality of the model.
    - Describe how you monitored the training process and the types of experiments conducted to improve the models.
4. **Submission Requirements:**
    - Submit Python scripts (.py files), including:
        - task_solution.py:
            - Implements classical_hr_estimation and deep_learning_hr_estimation.
            - Loads the model from models_definition.py.
            - Loads weights from model_weights.pth.
            - Processes input files and outputs HR predictions.
        - Ensure model_definition.py defines a torch.nn.Module class.
        - Include any additional scripts for preprocessing, training, or analysis as needed.
    - Save model weights with torch.save() in model_weights.pth.
    - Provide clear execution instructions.

<u>Challenge 2 (Optional)</u>

Dataset

You will receive a file named ppg_samples_with_hr_v4.parquet, which is generally noisier than the dataset in Challenge 1. It contains:

- **signal:** An array of 160 float values representing 5 seconds of PPG data sampled at 32 Hz.
- **hr:** The corresponding heart rate (in BPM) as a float.
- 20% of the signals are too noisy for reliable HR estimation and should be identified and excluded.

Task Instructions

- Implement challenge_2_hr_estimation using a deep learning model.
- Ensure an absolute error of **< 4 BPM** on the **80%** of usable data.
- Identify and return -1 for the 20% of signals that are too noisy for accurate prediction.

Submission Requirements

- Submit Python scripts (.py files), including:
    - task_solution.py:
        - Implements challenge_2_hr_estimation.
            - Loads the model from models_definition.py.
            - Loads weights from model_weights_c2.pth.
            - Processes input files and outputs HR predictions, marking noisy signals as -1.
    - Ensure model_definition.py defines a torch.nn.Module class if a new architecture is used.
    - Save model weights with torch.save() in model_weights_c2.pth.
- Provide clear execution instructions.

Evaluation Criteria

1. **Understanding:** Ability to clearly explain the code and the approaches during a follow-up interview, including a strong grasp of signal processing and deep learning principles.
2. **Correctness**: The solution meets the accuracy thresholds.
3. **Clarity**: Code is well-documented and easy to follow.
4. **Reproducibility**: Project runs without issues following the provided instructions.
5. **Analysis**: Clear and thorough explanation of the results and methods.
6. **Optimization**: Demonstrates thoughtfulness in improving the model and handling noise.

Submission

Submit your solution as a compressed file containing the codebase and documentation. Ensure all dependencies are specified in a requirements.txt file. You may use any tools or libraries you prefer, including generative AI tools such as ChatGPT, but be prepared to explain all choices made.

We look forward to reviewing your submission. If you have any questions, feel free to reach out.

Good luck!