Predicting Sleepiness from Raw PPG Signals Using Recurrent Neural Networks: A Comparative Study of RNN, LSTM, and GRU Architectures

Yoan Tutunarov
(Dated: December 11, 2025)

Predicting human fatigue from physiological signals is an important but challenging task, especially when relying on noisy raw data such as photoplethysmography (PPG). Traditional approaches depend on handcrafted features like heart rate variability, yet it remains unclear how effectively modern recurrent neural networks can learn predictive structure directly from the waveform. In this project, I evaluate RNN, LSTM, and GRU architectures for classifying self–reported sleepiness levels (1–7) from raw ear–PPG recordings in the 5–Gamer dataset. A unified PyTorch pipeline is used to compare architectures, hidden sizes, and sequence lengths under consistent training and preprocessing conditions. The GRU achieved the best test accuracy (≈ 0.59), outperforming both LSTMs and vanilla RNNs. Increasing hidden size improved performance (up to ≈ 0.58 for $H = 128$), while shorter temporal windows yielded better generalization. All models struggled with rare high–sleepiness classes due to class imbalance. Overall, the results demonstrate that recurrent networks can extract meaningful temporal information directly from raw PPG signals, and the findings provide practical guidance for end–to–end physiological modelling in applications such as fatigue or stress prediction.

I. INTRODUCTION

Estimating human fatigue and cognitive state from physiological signals is an increasingly important challenge in domains such as transportation safety, occupational health, and human–computer interaction. Photoplethysmography (PPG), a simple optical measurement of blood-volume changes, is especially attractive for continuous monitoring because it can be collected unobtrusively from wearable devices. However, raw PPG signals are highly noisy, subject to motion artefacts, and vary significantly across individuals, making the extraction of reliable fatigue-related markers a nontrivial task.

Traditional approaches in fatigue research rely on handcrafted physiological features, most prominently heart rate (HR) and heart rate variability (HRV), which require accurate peak detection and careful signal processing. Although these methods are well established, they depend heavily on domain-specific preprocessing and may fail when signal quality is low. With the growing availability of large datasets and deep learning techniques, an alternative direction has emerged: learning temporal representations directly from raw biosignals using recurrent neural networks (RNNs) and related sequence models [1, 2].

Despite the promise of end-to-end learning, questions remain about how effectively different recurrent architectures - such as vanilla RNNs, Long Short-Term Memory networks (LSTMs), and Gated Recurrent Units (GRUs) [3] - can capture meaningful structure from raw PPG, and how hyperparameters such as hidden-state dimensionality and sequence length influence performance. This is particularly relevant for datasets where labels, such as self-reported sleepiness scores, are sparse, subjective, and imbalanced.

In this project, I investigate whether recurrent neural networks can predict sleepiness levels directly from raw PPG signals collected in the 5 - Gamer dataset. The goal is not to reproduce physiological features such as HRV, but to evaluate the ability of modern sequence models to learn discriminative patterns end-to-end. I implement a unified PyTorch framework and systematically compare model architectures (RNN, LSTM, GRU), hidden sizes, and temporal window lengths.

The remainder of this report is organized as follows. Section II describes the data, preprocessing pipeline, and recurrent architectures. Section IV presents empirical results, including learning curves, confusion matrices, and comparisons across model variants. Section V summarizes the main findings and discusses limitations and directions for future work.

All code developed for this report is available via a GitHub repository.[1]

II. METHODS

A. Problem formulation

The goal of this project is to predict a subjective *sleepiness score* from raw photoplethysmography (PPG) signals using recurrent neural networks (RNNs). The data consist of continuous PPG recordings from five gamers during an extended gaming session, together with self–reported sleepiness scores on a discrete scale from 1 (fully alert) to 7 (very sleepy). After preprocessing (Section II C) each training example is a short window of normalized PPG samples

$$\mathbf{x}^{(n)} = (x_1^{(n)}, \ldots, x_T^{(n)}) \in \mathbb{R}^T,$$

[1] https://github.com/komandoyoko/Yoan_FYS_STK4155/tree/main

with fixed sequence length T, and an associated class label $y^{(n)} \in \{1, \ldots, 7\}$. The learning task is therefore a multiclass sequence–classification problem: we seek a function $f_\theta : \mathbb{R}^T \to \{1, \ldots, 7\}$ parameterized by weights θ (an RNN, LSTM or GRU network), trained to approximate the conditional distribution $p_\theta(y \mid \mathbf{x})$.

B. Relation to the dataset and methodological choice

The 5-Gamer dataset was originally collected for research on cognitive fatigue, with an emphasis on deriving physiological indicators such as heart rate (HR), heart rate variability (HRV), and respiration rate from the raw PPG signal. The data creators propose a multi-step signal–processing pipeline involving heartbeat peak detection, HRV feature extraction, respiration estimation, trend analysis, and ultimately the prediction of a "too tired to work" epoch. This approach relies heavily on classical physiological modelling and requires substantial signal-processing expertise. In practice, reliably detecting PPG peaks and computing HRV metrics from continuous, noisy data is a technically challenging task and would constitute a full project on its own.

In this project, however, I follow the alternative course option of applying neural-network methods to a dataset of our own choosing. Instead of reproducing the full physiological pipeline, I focus on the machine-learning component, which is the central topic of the course [1, 4]. I aim to adopt a modern end-to-end deep learning approach: short windows of raw, normalized PPG samples are used directly as input to RNN, GRU and LSTM models, which are trained to classify the subject's Stanford Sleepiness Scale score (1–7). This reformulates the problem as a supervised sequence-classification task and allows me to concentrate on comparing recurrent neural architectures, their training behaviour, and their ability to model temporal patterns.

Although my method differs from the feature-engineering approach envisioned by the dataset creators, it is well aligned with the course aims and the chosen project scope. The objective here is therefore not to reconstruct physiological markers such as HRV, but to investigate whether recurrent neural networks can learn predictive structure directly from the raw PPG waveform and to evaluate their relative performance on this task.

C. Data preprocessing and windowing

For each gamer, the raw PPG signal (red channel) was loaded and merged with the corresponding annotation file. Timestamps were converted to a common time axis and samples with missing values were discarded. To make the signal scale–invariant across subjects, the PPG amplitude was standardized using a per–gamer z–score normalization

$$\tilde{x} = \frac{x - \mu}{\sigma},$$

where μ and σ are the mean and standard deviation of the PPG signal for that gamer.

Sleepiness labels are provided at irregular time points. For each annotation at time t_{ann} with a valid sleepiness score in $\{1, \ldots, 7\}$, a fixed-length segment of the preceding signal was extracted:

$$[t_{\text{ann}} - \Delta t,\ t_{\text{ann}}],$$

with $\Delta t = 10$ minutes. Within this interval, overlapping windows of length T samples were generated with a stride of $T/2$ (50% overlap). Each window inherits the sleepiness score of the corresponding annotation. Windows were discarded if the segment did not contain enough samples to fill a complete sequence.

All windows from all five gamers were concatenated and then randomly shuffled. The total dataset was split into training, validation and test sets using fractions 70%, 15% and 15%, respectively. This split is done at the window level, so that all three sets contain data from all subjects, but no individual window appears in more than one split.

D. Recurrent neural network architectures

Recurrent neural networks are well suited for sequential data because they maintain a hidden state that can summarize information about previous time steps [1, 2]. In a standard ("vanilla") RNN, the hidden state $\mathbf{h}_t \in \mathbb{R}^H$ at time t is updated as

$$\mathbf{h}_t = \phi(W_{xh}\mathbf{x}_t + W_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)\,, \tag{1}$$

where $\mathbf{x}_t \in \mathbb{R}^d$ is the input at time t, $W_{xh} \in \mathbb{R}^{H\times d}$ and $W_{hh} \in \mathbb{R}^{H\times H}$ are weight matrices, $\mathbf{b}_h$ is a bias, and $\phi(\cdot)$ is a non–linear activation function (here tanh). For sequence classification we use the final hidden state as a summary of the whole input sequence,

$$\mathbf{o} = W_{yh}\mathbf{h}_T + \mathbf{b}_y, \tag{2}$$

and obtain class probabilities via a softmax,

$$p_\theta(y = k \mid \mathbf{x}) = \text{softmax}(\mathbf{o})_k. \tag{3}$$

Although simple RNNs can, in principle, represent long–range temporal dependencies, they are notoriously difficult to train due to vanishing and exploding gradients. To address this, the project also employs Long Short–Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures, which introduce gating mechanisms that regulate information flow through time.

a. Long Short–Term Memory (LSTM). An LSTM maintains both a hidden state $\mathbf{h}_t$ and a cell state $\mathbf{c}_t$. At each time step it computes a set of gates:

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \quad \text{(forget gate)}, \tag{4}$$

$$\mathbf{i}_t = \sigma(W_i \mathbf{x}_t + U_i \mathbf{h}_{t-1} + \mathbf{b}_i) \quad \text{(input gate)}, \tag{5}$$

$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c) \quad \text{(candidate cell)}, \tag{6}$$

$$\mathbf{o}_t = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o) \quad \text{(output gate)}. \tag{7}$$

The cell and hidden states are then updated as

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{8}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t), \tag{9}$$

where $\odot$ denotes element–wise multiplication and $\sigma(\cdot)$ is the logistic sigmoid. This architecture allows the LSTM to preserve information in the cell state over many time steps while selectively incorporating new information.

b. Gated Recurrent Unit (GRU). The GRU is a related architecture introduced by Cho et al. [3] that combines the cell and hidden state into a single vector and uses two gates instead of three.

For each time step,

$$\mathbf{z}_t = \sigma(W_z \mathbf{x}_t + U_z \mathbf{h}_{t-1} + \mathbf{b}_z) \quad \text{(update gate)}, \tag{10}$$

$$\mathbf{r}_t = \sigma(W_r \mathbf{x}_t + U_r \mathbf{h}_{t-1} + \mathbf{b}_r) \quad \text{(reset gate)}, \tag{11}$$

$$\tilde{\mathbf{h}}_t = \tanh(W_h \mathbf{x}_t + U_h(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h)\,, \tag{12}$$

and the hidden state update is

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \tag{13}$$

In practice GRUs are often computationally cheaper than LSTMs, while retaining the ability to model long–range dependencies.

E. Model configurations

All models were implemented in PyTorch using a common architecture template. The input is a univariate PPG sequence ($d = 1$), and the recurrent layer (RNN, LSTM or GRU) is followed by a fully connected layer that maps the final hidden state to the seven output logits. Unless otherwise stated, the default configuration uses a single recurrent layer with hidden size $H = 64$ and dropout of 0.1 between the recurrent layer and the output layer.

To study the effect of model capacity and temporal context, the following configurations were evaluated:

- **Architecture comparison:** vanilla RNN, GRU and LSTM, all with hidden size $H = 64$ and sequence length $T = 200$ samples.
- **Hidden size:** LSTM models with $H \in \{32, 64, 128\}$ and fixed $T = 200$.
- **Sequence length:** LSTM models with $T \in \{100, 200, 400\}$ samples and fixed $H = 64$.

All other hyperparameters (optimizer, learning rate, batch size and early stopping settings) were kept identical across experiments to ensure a fair comparison.

F. Training procedure

Given a batch of input sequences and labels $\{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{B}$, the model outputs logits $\mathbf{o}^{(n)}$ for each sequence. The training objective is the multiclass cross–entropy loss

$$\mathcal{L}(\theta) = -\frac{1}{B} \sum_{n=1}^{B} \log p_\theta\Big(y^{(n)} \mid \mathbf{x}^{(n)}\Big)\,, \tag{14}$$

where $p_\theta(y^{(n)} \mid \mathbf{x}^{(n)})$ is obtained by applying a softmax to the logits.

Models were trained using the Adam optimizer with a learning rate of 10^{-3} and mini–batches of size 128. Each experiment was run for up to 40 epochs. After every epoch the loss on the validation set was computed, and the model parameters with the lowest validation loss were saved. Early stopping was applied with a patience of six epochs: if the validation loss did not improve for six consecutive epochs, training was terminated and the best checkpoint was retained. All experiments were performed on CPU.

G. Evaluation metrics

For each trained model, performance was reported on the held–out test set using the best validation checkpoint. The primary metric is overall classification accuracy

$$\mathrm{Acc} = \frac{1}{N_{\text{test}}} \sum_{n=1}^{N_{\text{test}}} \mathbb{I}\Big[\hat{y}^{(n)} = y^{(n)}\Big]\,, \tag{15}$$

where $\hat{y}^{(n)}$ is the predicted sleepiness class and $\mathbb{I}$ is the indicator function. To obtain a more detailed view of how the models behave across classes, a confusion matrix and per–class accuracies were also computed. These metrics are particularly important here because the sleepiness scores are imbalanced, with lower scores occurring much more frequently than very high sleepiness levels.

III. IMPLEMENTATION AND VALIDATION

The implementation is structured around a modular PyTorch pipeline that handles data preprocessing, model definition, training, and evaluation. All components are configured through a common configuration file to ensure reproducibility and to allow systematic comparison between different recurrent architectures and hyperparameters.

A. Preprocessing and dataset construction.

Each gamer's raw PPG signal and annotation file were first loaded and merged using their timestamp fields. Missing values were removed, and the PPG amplitude was normalized using per-gamer z-score normalization to account for differences in baseline signal level. For every annotated sleepiness score, a fixed 10-minute segment of preceding PPG data was extracted and subdivided into overlapping windows of fixed sequence length T. The windows overlap by 50% (stride $T/2$), and each inherits the corresponding Stanford Sleepiness Scale score (1–7). The resulting collection of PPG windows and labels from all five participants was shuffled and split into training, validation, and test sets using proportions of 70%, 15%, and 15%, respectively.

B. Network architecture.

All models share a simple and comparable architecture implemented in PyTorch. The input to the network is a univariate PPG sequence of length T, represented as a tensor of shape (batch size, T, 1). The core of the network is a single recurrent layer, which is instantiated as one of:

- a vanilla RNN with tanh activation,
- a GRU (Gated Recurrent Unit), or
- an LSTM (Long Short–Term Memory unit).

The recurrent layer has hidden state dimension H and produces a sequence of hidden states $(\mathbf{h}_1, \ldots, \mathbf{h}_T)$. For sequence classification, only the final hidden state $\mathbf{h}_T$ is used as a summary of the entire PPG window. This vector is passed through a fully connected output layer

$$\mathbf{o} = W\mathbf{h}_T + \mathbf{b},$$

producing seven logits, one for each sleepiness class. A softmax is applied to obtain class probabilities. Dropout with rate 0.1 is applied between the recurrent layer and the output layer to reduce overfitting. Apart from the choice of recurrent unit and the values of H and T, the architecture is kept identical across all experiments to ensure a fair comparison.

C. Experimental configurations.

Three families of experiments were conducted to study the effect of architecture, model capacity and temporal context:

- **Architecture comparison:** vanilla RNN, GRU and LSTM models, all with a single recurrent layer, hidden size $H = 64$ and sequence length $T = 200$ samples.
- **Hidden-size comparison:** LSTM models with hidden sizes $H \in \{32, 64, 128\}$ and fixed sequence length $T = 200$ samples. This isolates the effect of model capacity while keeping the temporal context constant.
- **Sequence-length comparison:** LSTM models with sequence lengths $T \in \{100, 200, 400\}$ samples and hidden size $H = 64$. This investigates how much temporal context is beneficial when predicting sleepiness from PPG.

In all cases, the remainder of the architecture (one recurrent layer followed by a linear output layer with seven units) and the training procedure (described below) were held constant.

D. Training procedure.

Training was performed using the Adam optimizer with a learning rate of 10^{-3} and mini-batches of size 128. The loss function is the multiclass cross-entropy, computed between the predicted class probabilities and the true sleepiness labels. Each model was trained for up to 40 epochs. After every epoch, the loss on the validation set was computed, and the model parameters corresponding to the lowest validation loss were saved as a checkpoint. Early stopping with a patience of six epochs was used: if the validation loss did not improve for six consecutive epochs, training was terminated and the best checkpoint was retained. All experiments were run on CPU, and the same random seed was used to reduce variability between runs.

E. Validation and test evaluation.

During training, model performance was monitored on the validation set to guide early stopping and model selection. Once training converged, the best checkpoint for each configuration (architecture, hidden size or sequence length) was evaluated on the test set, which was not used in training or model selection. The primary evaluation metric is overall classification accuracy on the test set. In addition, confusion matrices, per-class accuracies and learning curves (training and validation loss as a function of epoch) were generated to provide a more detailed picture of model behaviour. The numerical test losses and accuracies for all configurations are summarized in a results table in Section IV, and the corresponding plots are used to support the discussion of the models' strengths and limitations.

IV. RESULTS AND DISCUSSION

A. Label distribution

Figure 1 shows the empirical distribution of the Stanford Sleepiness Scale (1–7) across all five gamers. The dataset is markedly imbalanced: the majority of annotations fall in the range 1–3, while higher scores (5–7) occur relatively rarely. This reflects the natural progression of the gaming sessions, where subjects begin alert and only develop strong fatigue later into the 22-hour trial.

This imbalance has direct implications for model performance. Since high sleepiness levels are underrepresented, models trained purely with cross-entropy loss may struggle to learn accurate decision boundaries for these classes. This effect is later visible in the confusion matrix (Figure 4) and the per-class accuracy plot, where the accuracy for classes 6 and 7 is substantially lower than for the more frequent classes.

Overall, this plot confirms that the supervised learning task is nontrivial and that class imbalance is an important factor to consider when interpreting the results.

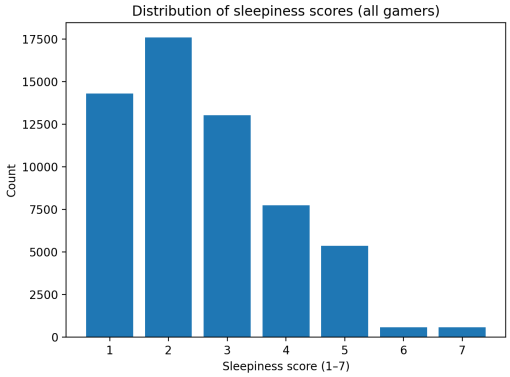

FIG. 1. Distribution of Stanford Sleepiness Scale labels across all five gamers. Lower sleepiness scores (1–3) dominate the dataset, while higher levels are comparatively rare.

B. Example PPG windows per class

Figure 2 shows representative normalized PPG segments for each sleepiness class. These windows illustrate several important characteristics of the data:

- The raw PPG waveform contains considerable noise and variability, both within and between subjects.
- There is no immediately visible distinction between the waveform morphology of different sleepiness levels, confirming that this is a subtle learning task.
- The amplitude and local oscillatory patterns vary naturally with subjects and recording conditions, which motivates the use of a model capable of temporal feature extraction rather than relying on simple amplitude thresholds.

These examples justify the choice of an end-to-end sequence model: manually engineered features would likely require substantial signal-processing sophistication, whereas RNN-based models can in principle learn discriminative temporal patterns directly from raw data.

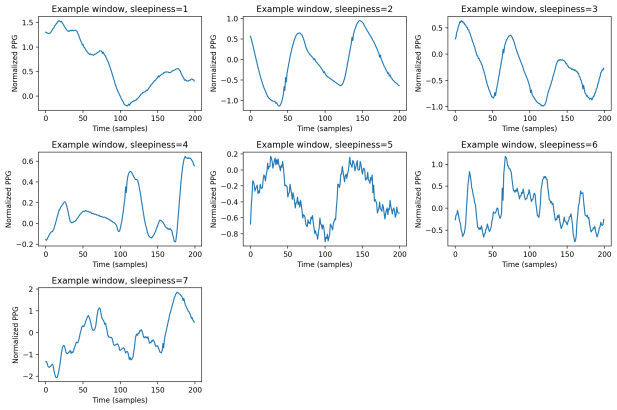

FIG. 2. Example normalized PPG windows for each sleepiness class. The waveform shows substantial variability and noise, and no visually obvious separation between classes, motivating the use of recurrent neural networks to learn temporal patterns directly from raw data.

C. Learning curves

Figure 3 shows the training and validation loss for the baseline LSTM model ($H = 64$, $T = 200$). Both curves decrease smoothly over the first 30 epochs, after which the validation loss stabilizes. The gap between training and validation loss remains modest throughout training, indicating that the model does not suffer from severe overfitting.

The learning curves also illustrate the effectiveness of early stopping: although training was allowed up to 40 epochs, the best validation performance was achieved near epoch 39, and the final model used for testing corresponds to this checkpoint. The stable convergence and absence of exploding or vanishing gradients confirm that the chosen architecture and preprocessing pipeline are appropriate for this dataset.

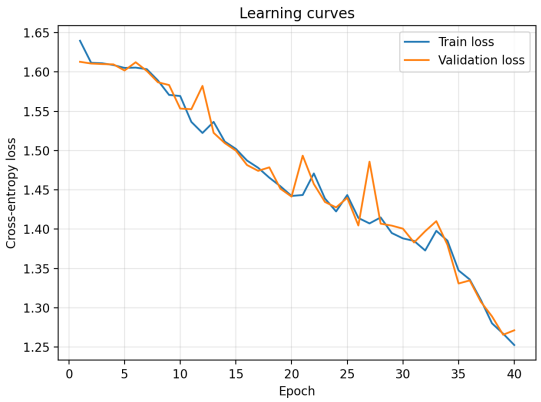

FIG. 3. Training and validation loss curves for the baseline LSTM model. Both losses decrease smoothly, and the validation curve stabilizes without diverging, indicating stable training and limited overfitting. The best model checkpoint is selected based on the minimum validation loss.

D. Confusion matrix and accuracy per class

The confusion matrix in Figure 4 shows that the model predicts the most frequent classes (1–3) reasonably well, while higher sleepiness levels are often confused with adjacent classes. This pattern reflects the class imbalance in the dataset and the difficulty of distinguishing rare high-fatigue states from raw PPG alone.

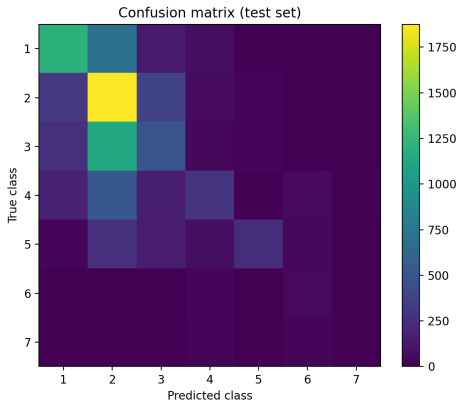

FIG. 4. Confusion matrix for the best-performing model. Most predictions fall along the diagonal for the frequent classes (1–3), while higher sleepiness levels are more often misclassified due to class imbalance.

Figure 5 reports accuracy for each sleepiness class. The model performs best on the frequent labels 1–3, while accuracy decreases substantially for the rarer high-sleepiness classes (5–7). This pattern is consistent with the class imbalance observed in the dataset.

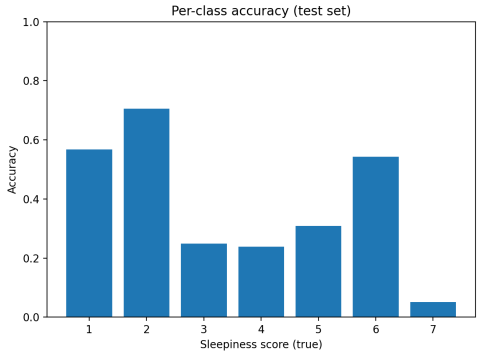

FIG. 5. Per-class accuracy for the best-performing model. Accuracy is highest for common classes and lowest for rare high-sleepiness levels.

E. True vs predicted sleepiness levels

Figure 6 compares the true and predicted sleepiness levels across the test set. The model follows the general trend of the target signal but shows systematic underestimation at high sleepiness levels, again reflecting the scarcity of these classes in the training data.

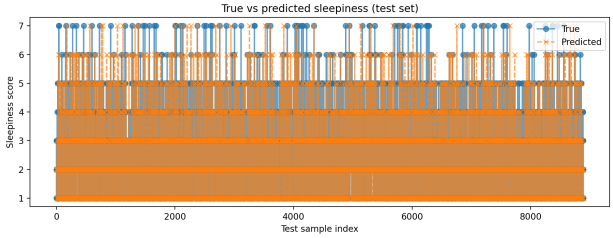

FIG. 6. True vs. predicted sleepiness levels over the test index. The model captures the overall trend but tends to underestimate higher sleepiness scores.

F. Model comparison across architectures, hidden sizes, and sequence lengths

This section summarizes how architectural choice, model capacity, and temporal context influence predictive performance.

Figure 7 compares the three recurrent architectures (RNN, GRU, LSTM) with identical hyperparameters. The GRU achieves the highest accuracy, while the vanilla RNN performs notably worse, indicating that gating mechanisms are important for modelling noisy physiological time-series data.

Figure 8 shows the effect of varying the hidden size for the LSTM model. Increasing the hidden dimension improves accuracy, with the best performance obtained for $H = 128$, suggesting that additional model capacity helps capture subtle temporal dependencies in the PPG

signal.

Figure 9 evaluates different sequence lengths. A shorter context window ($T = 100$) performs slightly better than longer windows, likely because very long sequences introduce additional noise and reduce the number of training samples due to fewer possible windows.

Overall, these comparisons show that model choice and temporal windowing have a meaningful impact on performance, and that GRUs with moderate-to-large hidden sizes deliver the strongest results on this dataset.

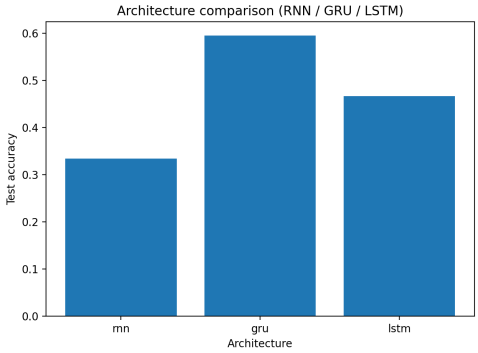

FIG. 7. Architecture comparison between vanilla RNN, GRU, and LSTM with identical hyperparameters.

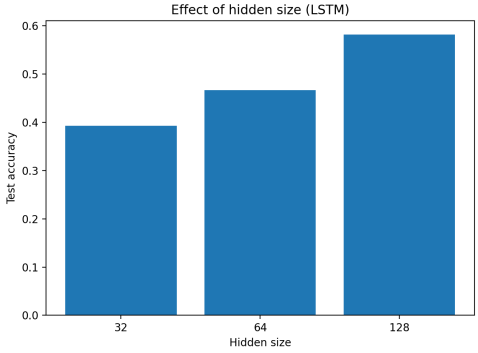

FIG. 8. Effect of hidden size on LSTM performance. Larger models achieve higher accuracy.

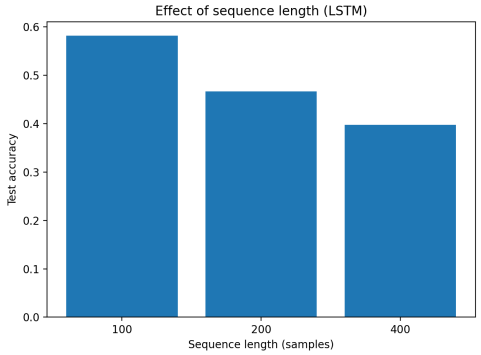

FIG. 9. Effect of sequence length on LSTM performance. Shorter windows yield slightly better accuracy.

V. CONCLUSION

This project investigated whether recurrent neural networks can predict subjective sleepiness levels directly from raw photoplethysmography (PPG) signals, without relying on engineered physiological features such as heart rate variability. By framing the task as a multiclass sequence–classification problem and systematically comparing RNN, GRU and LSTM architectures, the study provides insight into how temporal models behave on noisy physiological time-series data.

Across all experiments, gated architectures (GRU and LSTM) consistently outperformed the vanilla RNN. The GRU achieved the highest overall test accuracy, confirming that its update and reset gates offer an effective balance between expressiveness and computational efficiency when modelling subtle temporal dependencies. Increasing the hidden size improved performance for LSTMs, demonstrating that additional model capacity helps extract structure from the highly variable and noise-dominated PPG waveform. Sequence-length experiments showed that shorter windows ($T = 100$) slightly outperform longer ones, likely because long sequences introduce more noise and reduce the number of training examples.

Despite these gains, all models struggled to correctly classify high sleepiness levels (scores 5–7), a consequence of both class imbalance and the inherent difficulty of predicting subjective fatigue states from raw PPG alone. The confusion matrix and per-class accuracy plots confirm that the models primarily learn to distinguish among the frequent classes (1–3), while underestimating very high fatigue levels. This limitation suggests that additional physiological channels (ECG, respiration), engineered features (HRV metrics), or subject-specific calibration may be required for reliable prediction in the upper range of the sleepiness scale.

Overall, the results demonstrate that recurrent neural networks can extract meaningful temporal patterns from raw PPG signals and provide reasonable approximations of self-reported sleepiness, even under challeng-

ing conditions. The project highlights both the potential and the limitations of end-to-end deep learning approaches in physiological time-series analysis. Future work could integrate classical signal-processing features with learned representations, explore attention-based architectures such as Transformers, or employ imbalance-aware training strategies (e.g., focal loss or class weighting) to improve performance on rare fatigue states.

AI DISCLAIMER

This project has also made limited use of AI tools, such as ChatGPT, as a support mechanism for improving clarity, refining explanations, and troubleshooting code. The primary analysis, problem solving, and discussion were carried out independently by myself. AI assistance was used only to enhance language in the report, verify understanding, and used for polishing figure names, ensuring that I fully comprehend the underlying concepts and methods.

Acknowledgments

I would like to thank our lecturer, Morten Hjorth-Jensen, for his valuable guidance and insights throughout the course. I also extend my gratitude to our group teachers for their helpful feedback and discussions during the project.

[1] M. Hjorth-Jensen, Fys-stk4155 machine learning and data analysis: Week 47 lecture notes, https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week47.html (2024), accessed: 2025-01-10.

[2] S. Raschka, Y. Liu, and V. Mirjalili, Machine learning with pytorch and scikit-learn, chapter 15: Modeling sequential data using recurrent neural networks, https://github.com/rasbt/machine-learning-book/tree/main/ch15 (2022), accessed: 2025-01-10.

[3] K. Cho, B. v. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, Learning phrase representations using rnn encoder–decoder for statistical machine translation, arXiv preprint arXiv:1406.1078 (2014).

[4] M. Hjorth-Jensen, Fys-stk4155 machine learning and data analysis: Week 46 lecture notes, https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week46.html (2024), accessed: 2025-01-10.