DO UMAP WHEN YOU'RE TIRED?

EJ MERCER, NATHAN SCHILL, DALLIN SEYFRIED, BRIGG TRENDLER

Abstract. This paper explores the analysis of brainwaves to determine an individual's sleep state. We use the Fourier transform, as well as various dimensionality reduction techniques to clean the raw data. Then, we attempt to use forecasting & classification techniques to accurately model the data. We believe a combination of these methods may be effective in identifying sleep states in real time—which is advantageous for sleep studies and sleep quality analysis.

1. Background/Motivation

A typical sleep cycle consists of various stages of REM (rapid eye movement) and NREM (non-REM) sleep states. These sleep stages can be identified by their relationship to brain activity. Lower brain activity indicates a deeper sleep state, while higher brain activity indicates wakefulness.

Our goal with this project is to find a classifying model using the measured metrics and the known annotated sleep states. Additionally, we seek to explore forecasting future brain activity and sleep states using Reservoir Computing and Kalman Filtering. Successful models in these areas will help with sleep studies and may help create hardware which can identify a user's sleep state in real time. These models can help improve our understanding of the sleep states and their purposes, especially in individuals with sleeping disorders.

1.1. Related Work.

Part of our work focuses on how well Reservoir Computing can forecast brainwaves and their related frequencies and magnitudes. Previous research has been done on this topic using similar techniques with EEG data [7]. Their work evaluates the MSE of its forecasts, whereas we evaluate MSE along with another measurement known as Valid Prediction Time.

Furthermore, in September of 2023, Lee et al. developed a deep learning model that classifies three states—wakefulness, NREM, and REM—of sleep with an overall accuracy of 85.2% [6]. Other studies have employed a variety of deep learning techniques to this problem, including convolutional neural networks, and have achieved similar results [2]. Our project aims to take a different approach and using traditional methods to classify sleep states and

Date: March 30, 2024.

`https://github.com/btrendler/v3_project_winter`

forecast future brain waves. We hope to obtain comparable results, using less computational power.

2. Data Cleaning & Pre-Processing

We obtained our data from a study by Bob Kemp [1], accessed through PhysioNet [4]. Approximately 70 patients participated in this experiment across two nights. During this time, patients wore a cassette-type device which recorded EOG and EEG signals at 100 Hz, as well as the a submental EMG signal, respiration rate, and body temperature at 1 Hz. Each file is around 24 hours long. Although this is not a large sample of nights, this still resulted in a total set of approximately 200,000 samples through our processing, which will be a good foundation for our exploration.

Once this data was recorded, well-trained technicians manually annotated the files to mark various events and each of the sleep states (**W**ake, **R**EM sleep, **1-4**, **M**ovement, and "unknown" (**?**))—thanks to this manual annotation, our data is very reliable based on known sleep state identification techniques. Being a fairly old dataset, it has been cleaned up thoroughly, so we thankfully do not have to worry much about cleaning the data—we only need to prepare it for our modeling techniques.

To begin, we needed to convert the database into a convenient form for use with `scikit-learn` [11]. The data was originally stored in EDF (European Data Format) files; thankfully, we found a library which allowed us to open these files in Python. To simplify the rest of the cleaning, we first began by aggregating all of the files into one giant compressed Numpy [5] archive, storing each patient/night/frequency combination as a separate key in the archive.

One hurdle we ran into was that the polysomnogram (PSG, brainwave recording) files and the sleep state annotations were stored separately, and the sleep states were only stored as a list of time stamps. Since the annotators used a specific state to mean "unknown" (annotated as a **?**), we assumed that the intervals of the markers were intended to overlap perfectly—i.e., there would be one and only one annotated sleep state for every sample in the file. In reality, this was not a perfect assumption, as sometimes there was a gap of a few samples between one marker ending and the subsequent marker starting. However, it was clear enough for our purposes. We mapped the **?** annotation to the value `NaN`, allowing us to account for that later on in the processing, and ignored any annotations that were not sleep states (such as the **M** annotations), mapping the sleep state **W** to 0, **1-4** to 1-4, and **R** to 5.

After processing, this yielded approximately *28.5 GB* of data. Although some of our methods would take advantage of the raw waveforms, we knew that it would be more practical for clustering methods to have the data in frequency-domain format instead of time-domain format, since the dominant frequencies indicate the level of brain activity. To do this, we split each full

day into 30-second intervals. Each interval was then broken down into 0.4-second sections, and we used `SciPy`'s real-valued FFT implementation [9] to decompose these 0.4-second segments into their base frequencies, taking the absolute value to get the magnitudes of each frequency. We then took the average of all 75 segments' frequency profiles to get a profile for the full 30-second interval. Using a 0.4-second segment equated to 40 samples of each waveform, which allowed us to identify 21 unique waves from 0 Hz to 50Hz, inclusive, subsequently helping us identify common dominant kinds of brainwaves. This number was chosen to place a handful of identifiable frequencies in each interval of interest, based on established bands of brain wave frequencies. Notably, the sample rate of our data would not allow us to identify frequencies greater than 50 Hz; however, this is the upper limit of the typically-considered brain waves, anyway. Since the 0 Hz signal just equates to sensor drift, and does not contain any wavelike signals, we discarded this value to identify 20 frequencies for each signal.

On these 30-second intervals, we also averaged any non-wavelike values, such as the respiration rate and body temperature, and used the most common sleep state as the sleep state for that whole interval. We then split the patients into training, validation, and testing sets, making sure that if a patient was monitored over multiple nights, all their data was contained in one set.

A significant advancement came from realizing that our data was being stored as complex 32-bit floating point values, which took up 64 bits of storage for each signal for each sample. This was unnecessary, so we re-exported the data from complex-valued floats to half-precision floats. (We also exported them as full-precision floats, but this file was still about 300 MB, and we did not end up needing the higher precision.) This brought our final data file, containing all the information we needed from the full dataset, down to just **52 MB**. Through this reduction, our dataset became much more computationally tractable.

3. Methods

3.1. **Forecasting.**

3.1.1. *Reservoir Computing.* Our first attempt to model this data was done by using Reservoir Computing (RC), which is a branch off of neural networks, used to make forecasts on time-dependent data. It randomly generates a network adjacency matrix according to some parameters and then trains that network on a continuous part of a time series. After training, the response states can be ran through the trained network to forecast future time values (similar to the procedure for making predictions with a Kalman filter). For a detailed explanation of how this process works, see Section

RC Neural Network permission granted by Dr. Boyd

7.1 in the appendix. Once we have a forecast, we will compute the Mean-Squared Error (MSE) and the Valid Prediction Time (VPT, see Section 7.2; we use $\epsilon = 5$).

Our network for RC was built using an Erdos–Renyi graph to represent the adjacency matrix. We used a grid search using MSE and VPT scoring to determine the best values of the connectivity c and hyperparameters γ, ρ, σ, and α, which adjust training & driving the reservoir network.

3.1.2. *Kalman Filter Sampling.* We also try using a Kalman filter (KF) to forecast sleep states. To do so, we first fit a KF (using `pykalman` [3]) and Softmax classifier (using `scikit-learn`) on several patients. We then filter the first 800 time steps (typically out of about 2700) and sample subsequent hidden states and observations (including the Fourier coefficients) to compare with the true sequences. Unfortunately, this yielded results *worse than guessing* the most common sleep state on subsets of both the train and the validation sets, so we did not attempt training or testing on the full dataset.

3.2. **Classifying.** Another idea we explored was the classification of sleep states using the brainwave signals and body state information. To do this, we treated each 'row' of the data—consisting of the breakdowns of the brainwave frequencies on a particular interval, as well as averaged respiration & temperature values—as a point in 64-dimensional space, and used various methods to categorize these points.

3.2.1. *Classification Metrics.* To determine the accuracy of our classifiers which include clustering algorithms, we realized we have to use classification-specific metrics which ignore labels, instead scoring the similarity of the clusterings; thankfully `scikit-learn` provides many of these, including Fowlkes–Mallows (FM), homogeneity (H), completeness (C), V-measure (V), adjusted Rand index (AR), and mutual information scores (AMI). Each of these scores increases as the classification gets better, maxing out at 1.0; for our grid searches, we will use the adjusted random scoring method.

3.2.2. *Kalman Filter with Softmax Classification on Hidden States.* One approach was to fit a KF on the frequency-domain data, then use Softmax classification on the hidden states of the filter. To do this, we chose a 7-dimensional hidden state space—which hopefully corresponded to the number of sleep states—and then used expectation maximization to fit the KF on the training set. After normalizing the states and applying the classifier, we did not get very good results on the test set; in fact, the accuracy was approximately equivalent to guessing **W** for every state.

Since there exists a lot of variance in people's sleep schedules, we also tried fitting a unique KF on each patient's first night, then used a single classifier trained on the resulting hidden states. Then, to test a patient's new night, we feed in the data to that patient's filter and use the pretrained Softmax classifier. This allows the filtering portion to adjust to each patient, but have the classification all be computed identically.

3.2.3. *Softmax Classification of Fourier Coefficients.* The approach in the above section (3.2.2) may cause the reader to wonder, why use the Kalman filter at all? As mentioned, the Kalman filter would enable forecasting into the future. However, we also try skipping the Kalman filter and just using Softmax classification on the Fourier coefficients (again normalized).

3.2.4. *K-Means & GMM Classification.* Another attempt we made at classification was to do a barebones K-Means classification using `scikit-learn`. After fitting the model to 10% of the data, every metric was abysmally low—the highest score out of the above metrics was still under 0.4. So, we did not explore standalone K-Means further as a method for classifying this data.

To account for potentially different variances, we figured GMM could be a better approach instead. When setting up this model, there was only one parameter to explore—the covariance type, which constrains the shape of the covariance matrices. Performing a grid search using 10% of the data yielded `'tied'`—meaning all states share the same covariance—as the highest scoring parameter when dealing with the high-dimensional data directly. However, this did not produce useful results when tested on the remaining data. One reason for this could be the high dimensionality of the data, since the points lie in 64-dimensional space. To counteract this, we decided to implement some form of dimension reduction and then come back to clustering algorithms.

3.2.5. *Unsupervised Dimension Reduction.* The first dimension reduction we explored was t-distributed Stochastic Neighbor Embedding (TSNE); we began by projecting this data down to just 2 components. This did not yield anything very promising, even after altering the perplexity values—see Figure 1. We also knew that UMAP could be a good option, so we attempted to project down to 2D and 3D using UMAP's default parameters, considering 50 of the nearest neighbors. The results of these projections can be seen in the top row of Figure 2.

Although the TSNE results were not promising, the UMAP results appeared to provide a better starting point which we could feed into some other kind of classification method. So, we hoped that by using a grid search across several parameters, we might be able to find something more accurate—especially if we used some kind of clustering algorithm on the output.

3.2.6. *Supervised Dimension Reduction.* In the process of finding the best UMAP parameters, we discovered that the `umap-learn` package [8] provides a supervised fitting system for UMAP. Immediately, this produced much better results, both in 2D and 3D (seen in the bottom row of Figure 2), and we were very hopeful that we could use this in a pipeline to produce an accurate classifier.

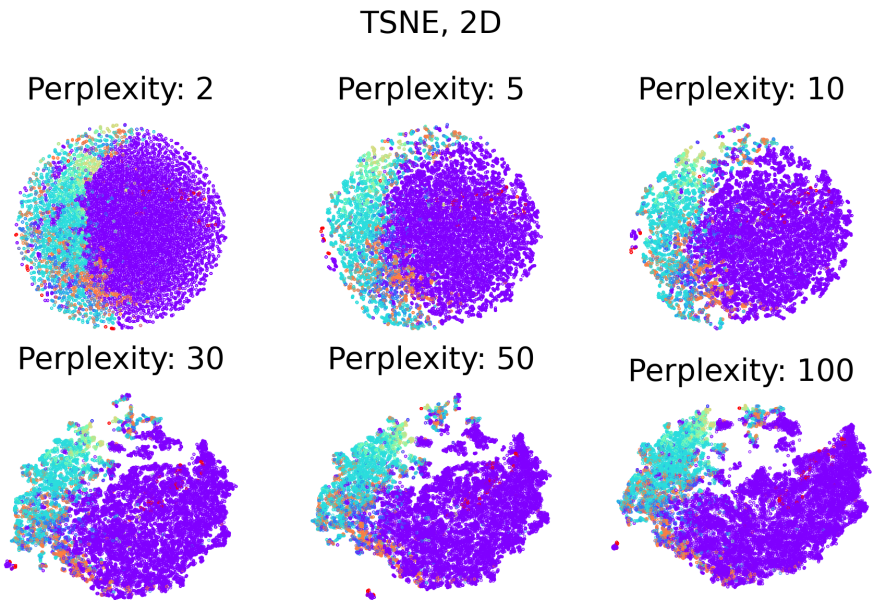

FIGURE 1. Various TSNE projections using 10% of the training data. Each sleep state shows as its own color, ranging from awake (purple) to deepest sleep (orange). **?**-classified points are red.

We tried both GMM and K-Means as the final element in this pipeline (to yield the classification labels), and determined that GMM was a better fit than K-Means would be, due to the different variances and shapes of the visible clusters.

After doing a grid search over 5 different (and increasingly refined) selections of model parameters for the UMAP+GMM pipeline, we determined that the best parameters according to the adjusted random score were a learning rate of 140 with 9 nearest neighbors, and a minimum distance of 1.0. The reduction of the full dataset using these parameters is shown in Figure 3.

3.2.7. *Random Forests.* Upon further inspection of the initial reductions and our modeling decisions, we realized a Random Forest Classifier (RFC) would be a better fit for this problem; this is because sometimes there are multiple clusters of each label—for example, notice the two orange clusters in the bottom left of Figure 2. Immediately with the default parameters projecting into 2 dimensions, UMAP+RFC performed much better. We explored various values for the number of target dimensions, and determined that this number did not influence the accuracy greatly. Higher dimensions

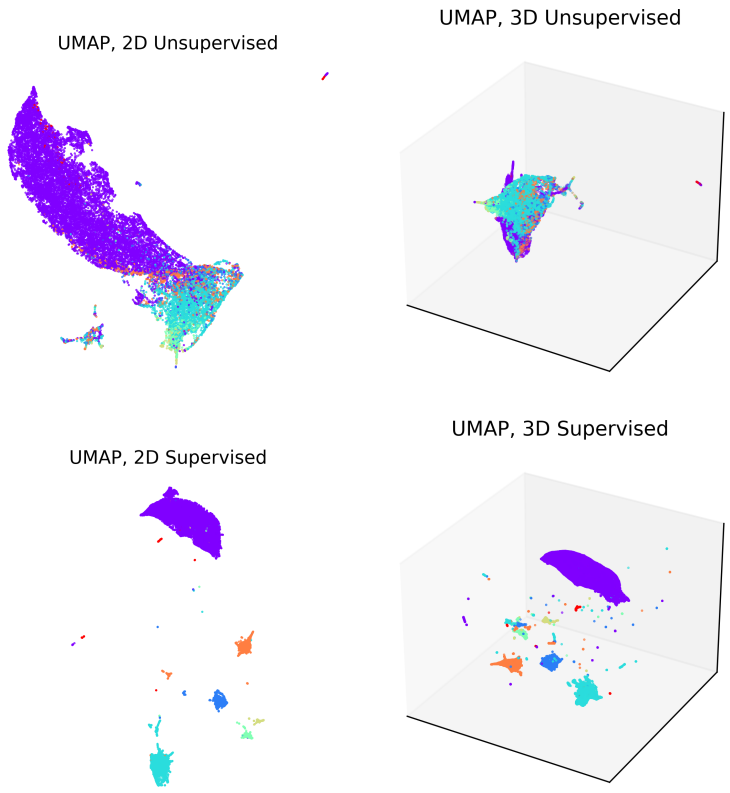

FIGURE 2. Projections into 2D and 3D via unsupervised UMAP (top row) and supervised UMAP (bottom row). These all are done with 10% of the training data, considering 50 of the nearest neighbors, and using the default parameters of UMAP. The coloring is identical to Figure 1.

generally produced slightly higher scores, so 8 dimensions was a reasonable balance between speed and accuracy.

However, in the process of exploring parameters, we noticed that the best accuracy score came from just running a random forest classifier directly on the full-dimensional dataset. Not only was this faster than any of the dimension-reducing configurations, but it generalized to the rest of the test set surprisingly well.

4. Results & Analysis

4.1. **Forecasting.**

4.1.1. *Reservoir Computing.* We performed a grid search on the frequency-domain data, focusing on the Fourier data of just one patient; the results of this grid search and the corresponding forecast can be seen in the left

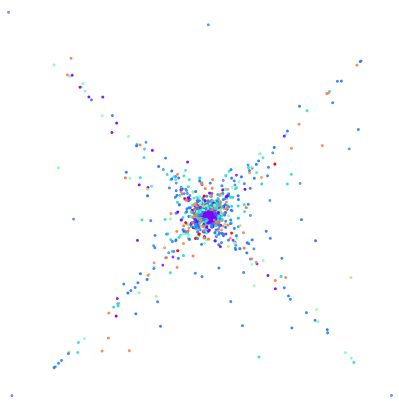

FIGURE 3. UMAP into 2D done with 100% of the training data, using parameters found from a grid search using 5% of the training data. The coloring is identical to Figure 1.

	FM	H	C	V	AR	AMI	Acc.
Kalman+Softmax	.7114	.0008	.0405	.0016	.0038	.0015	.6873
Individual Kalman	.8064	.3583	.3999	.3780	.5772	.3778	.8038
Softmax	.9071	.5448	.6091	.5752	.8056	.5751	.8702
K-Means	.3696	.2859	.2003	.2356	-.010	.2355	.1405
GMM	.4286	.4271	.2379	.3056	.1579	.3055	.2012
UMAP(3)+GMM	.6292	.2650	.2246	.2431	.3415	.2430	.5371
UMAP(8)+RFC	.8887	.5247	.5270	.5258	.7756	.5258	.8443
Plain RFC	.8960	.4949	.6023	.5433	.7741	.5433	.8566

TABLE 1. Various classification metrics of our models being performed on the test set. For all of these, closer to 1 is better; see Section 3.2.1. The best results are highlighted in green and the runner-up in yellow.

column of Figure 4. We also wondered how effectively we could forecast the original PSG data. Since these values appeared to be somewhat noisy, we computed the moving average of a patient's data to help account for this, and trained the network on that as well—the results of this forecast are shown in the right column of Figure 4.

The effectiveness of RC is very restricted, and depended heavily on having smooth/de-noised data to be able to make accurate forecasts. For the Fourier Data, our grid search found that the best parameters were $c = 4, \gamma = 1, \rho = 1, \sigma = 0.14, \alpha = 0.01$, which yielded an MSE score of 445,469 which is absolutely horrendous (see Figure 4 for details) and for the PSG data we found that $c = 3, \gamma = 1, \rho = 5, \sigma = 0.14, \alpha = 0.01$ yielded an MSE score of 223 on average. One of the better results is shown in Figure 4 in the right column.

The errors on both datasets were extraordinarily high and the VPT on both never made it past a few hundredths of a second before the difference became too large. Thus, under these results, RC did not forecast accurately. For more intuition as to why this happens, see section 7.3.

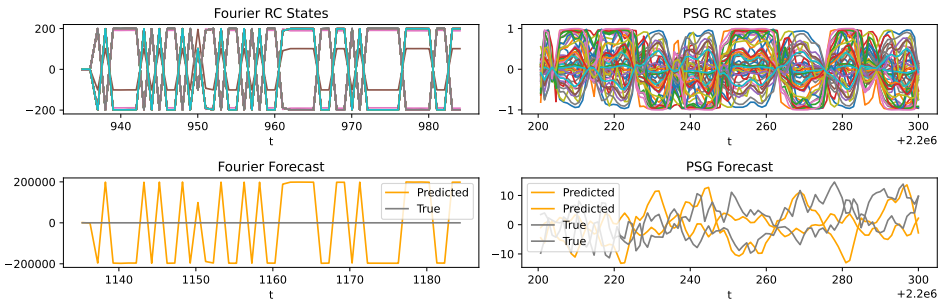

FIGURE 4. Results from a reservoir network of 50 nodes on the frequency-domain data (left) and the original PSG data (right), showing the response rates & forecasts of each model. In RC, bounded 'noisy' response states are *good*, and produce better predictions (hence why PSG is predicting better than Fourier here). See section 7.3 for an extended explanation.

4.2. **Classifying.** Of all the models we tried, Softmax, UMAP+RFC, and a plain random forest classifier were the highest-performing; see the third row and the last two rows of Table 1. If optimizing against completeness, the selection of which model to use should be done based on the efficiency of the Softmax and RFC implementations available. Otherwise, in almost every case, the Softmax is preferable due to its higher scores in every single metric we explored, and it can be run quicker than most of our models as well. This is relatively unexpected, as when we discovered this ability, we had already explored most of the other classification options and figured that RFC was the best option.

We were surprised how effectively the Softmax worked on the test data, scoring an accuracy of about **87%**. This is almost equivalent to what it got on the train & validation sets—on which the accuracy was around 88%—which suggests that this model is fit very accurately and is not at a high risk of being overfit.

5. ETHICAL CONSIDERATIONS

The research which provided the source of our data was done reliably and has been used in many peer-reviewed articles. That being said, the participants of study were exclusively people of Caucasian descent, ages 25 and older—so, there is a lack of diversity in this demographic that must be acknowledged when considering any results from this data. In the future, we

hope that sleep data collection studies will seek to diversify their sampling for more broadly-applicable research use. Furthermore, our analysis does not take into account any mental or physical health data; as such, more robust data collection would be necessary to generalize the results found in this paper. Our results here are fairly benign, as our models only predict future brainwave voltages and estimate sleep state based on those frequencies and amplitudes. Thus, we cannot think of a way the results from our analysis could be maliciously misused or lead to any sort of feedback loop.

6. Conclusion

From our model exploration, we identified several main takeaways. The first was that predicting future brain states based on previous data is very difficult to do accurately, and our work reflects this, both with RC and KF. Although we explored many parameters for RC, there are likely other parameters that could make this model work more effectively; however, finding these parameters would be difficult thanks to the inherent noise of brain signals. Further exploration in RC could focus on using a windowed method for global forecasting onto other patients (see [10]) and limiting ρ. A limited $\rho \in [1, 3]$ introduces nicer properties to the RC algorithm which can be scored using another metric known as consistency. Consistency evaluates how stable the trained network system is given perturbations to the response states (see [12]).

We discovered that a single Kalman filter followed by a Softmax did not classify sleep states well. However, when we fit a KF on each patient's first night and then filtered and Softmax-ed on the second night, the results were significantly better. This suggests—as we would imagine—that each patient's sleep states evolve differently. Future work could search over hyperparameters of the KF, such as the dimension of the hidden state space. This approach would make it possible to annotate a patient's first night by hand and thereafter classify using their personal KF.

The second main takeaway from our project is that taking the FFT and then clustering works very well on a dataset of this type. In this process, we learned a lot about UMAP and how it generalizes. Our theory is that UMAP would have performed better if we had more data—although we had over 200,000 data points when loaded individually, we were trying to find the approximation of a 64-dimensional space. We think supervised UMAP could provide a more effective and practical model if it were given an extremely large dataset and trained ahead of time. This could be optimized as well by precomputing the nearest neighbors for the UMAP algorithm, and then extending that computation to a data point we wish to classify.

However, for now, we must admit defeat with our elaborate models and acknowledge that a well-tuned Softmax classifier produces the most effective clustering results, in a relatively small amount of computation time.

We hereby grant any ACME professor permission to use this in future course material.

References

[1] *B Kemp, AH Zwinderman, B Tuk, HAC Kamphuisen, JJL Oberyé. Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the EEG. IEEE-BME 47(9):1185-1194 (2000).*

[2] *Choi JW, Kim DH, Koo DL, Park Y, Nam H, Lee JH, Kim HJ, Hong SN, Jang G, Lim S, Kim B. Automated Detection of Sleep Apnea-Hypopnea Events Based on 60 GHz Frequency-Modulated Continuous-Wave Radar Using Convolutional Recurrent Neural Networks: A Preliminary Report of a Prospective Cohort Study. Sensors (Basel). 2022 Sep 21;22(19):7177. doi: 10.3390/s22197177. PMID: 36236274; PMCID: PMC9570824.*

[3] *Duckworth, D. (2012). Pykalman [Python]. https://pykalman.github.io/#*

[4] *Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.*

[5] *Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.*

[6] *Lee JH, Nam H, Kim DH, Koo DL, Choi JW, Hong SN, Jeon ET, Lim S, Jang GS, Kim BH. Developing a deep learning model for sleep stage prediction in obstructive sleep apnea cohort using 60 GHz frequency-modulated continuous-wave radar. J Sleep Res. 2024 Feb;33(1):e14050. doi: 10.1111/jsr.14050. Epub 2023 Sep 26. PMID: 37752626.*

[7] *Li-Yu Chen, Yi-Chun Chen, Jason C. Huang, Sophie Sok, Vincent Armbruster, Chii-Chang Chen; Brainwave implanted reservoir computing. AIP Advances 1 January 2024; 14 (1): 015253.* `https://doi.org/10.1063/5.0186854`

[8] *McInnes, L, Healy, J, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, ArXiv e-prints 1802.03426, 2018*

[9] *Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), 261-272.*

[10] *S. Harding, Q. Leishman, W. Lunceford, D. J. Passey, T. Pool, B. Webb; Global forecasts in reservoir computers. Chaos 1 February 2024; 34 (2): 023136.* `https://doi.org/10.1063/5.0181694`

[11] *Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.*

[12] *Thomas Lymburn, Alexander Khor, Thomas Stemler, Débora C. Corrêa, Michael Small, Thomas Jüngling; Consistency in echo-state networks. Chaos 1 February 2019; 29 (2): 023118.* `https://doi.org/10.1063/1.5079686`

7. Appendix

Nota Bene: In the following sections, 'predict' exclusively means 'forecast,' rather than 'classify.'

7.1. **Reservoir Computing, Explained.** With an input signal of brainwaves $\mathbf{u}(t) \in \mathbb{R}^m$ for t in some interval $[0, T]$, our goal is to use a reservoir to predict brainwaves after the final time T. The three steps to Reservoir Computing are as follows:

(1) **Processing:** In the processing step we take nodes of a processing network given by $\mathbf{A} \in \mathbb{R}^{n \times n}$ where $\mathbf{A}_{ij}$ represents the weight of the network connection from node j to node i (like an adjacency matrix). The states then evolve according to this differential equation:

$$\frac{d}{dt}\mathbf{r}(t) = \gamma[-\mathbf{r}(t) + tanh(\rho\mathbf{A}\mathbf{r}(t) + \sigma\mathbf{W}_{in}\mathbf{u}(t))]$$

for $t \in [0, T]$ where $\mathbf{r}(t) \in \mathbf{R}^n$ represents the state of the nodes in the reservoir at time t. $\mathbf{W}_{in} \in \mathbf{R}^{nxm}$ is a constant linear map that sends a linear combination of the m-dimensional training data $\mathbf{u}(t)$ to each of the n nodes in the reservoir. $\mathbf{W}_{in}$ can be thought of as an observation matrix influencing how much each node sees the passed in data and is typically drawn uniformly from $U(-0.5, 0.5)$ for each entry. γ, ρ, and σ are non-negative scalar parameters where ρ is also referred to as the spectral radius since $\mathbf{A}$'s spectral radius has been scaled down to 1. As the input signal $\mathbf{u}(t)$ comes in it drives the nodes of the processing network. The resulting response is given by the differential solution $\mathbf{r}(t)$ for $t \in [0, T]$.

(2) **Aggregation:** We first discretize the time interval $[0, T]$ into equal time-steps given by $\{t_l\}_{l=0}^{L}$. From this we then find the minimizer:

$$\mathbf{W}_{out} = \underset{\mathbf{W} \in \mathbb{R}^{mxn}}{\operatorname{argmin}} \left[\sum_{l=0}^{L} ||\mathbf{W}\mathbf{r}(t_l) - \mathbf{u}(t_l)||_2^2 + \alpha||\mathbf{W}||_2^2 \right]$$

This results in the linear map $\mathbf{W}_{out} \in \mathbb{R}^{mxn}$ which we use to aggregate responses by calculating $\hat{\mathbf{u}}(t) = \mathbf{W}_{out}\mathbf{r}(t) \in \mathbb{R}^m$. The $\alpha > 0$ used above is a ridge alpha we use to prevent overfitting. If α is small then $\hat{\mathbf{u}}(t) \approx \mathbf{u}(t)$ for $t \in [0, T]$.

(3) **Prediction:** The Prediction step begins by substituting $\hat{\mathbf{u}}(t)$ in for $\mathbf{u}(t)$ back up in the processing equation:

$$\frac{d}{dt}\hat{\mathbf{r}}(t) = \gamma[-\hat{\mathbf{r}}(t) + tanh([\rho\mathbf{A} + \sigma\mathbf{W}_{in}\mathbf{W}_{out}]\hat{\mathbf{r}}(t))]$$

Before we were driving the network with our training data $\mathbf{u}(t)$, but now the network builds in $\mathbf{W}_{out}$ so it can accept our predicted states $\hat{\mathbf{r}}(t)$ to forecast a time series. Therefore, we call this the trained

system. This produces a time series $\hat{\mathbf{r}}(t) \in \mathbb{R}^n$ for $t \geq T$ from which we can create the predicted time series $\hat{\mathbf{u}}(t) = \mathbf{W}_{out}\hat{\mathbf{r}}(t) \in \mathbb{R}^m$ to get back to our observation space. See [10] for more details.

7.2. **Valid Prediction Time.** Valid Prediction Time T^* is defined as the time when the following inequality first fails after prediction begins: $||\hat{\mathbf{u}}(t) - \mathbf{u}(t)||_2 < \epsilon$, where ϵ can be tuned to adjust how closely the predictions need to match the true values. This provides a metric for how quickly the predictions break down to a certain level of inaccuracy.

7.3. **Understanding RC Response States.** In Figure 4, we show the results of a reservoir forecast on both the Fourier translated data and on the original PSG data. The upper graphs in this figure represent the response states $\hat{\mathbf{r}}(t)$, overtime and correspond to essentially to the different kinds of combinations the $\mathbf{W}_{out}$ matrix can take to form prediction states and evolve. Somewhat similar in reasoning to the trace plots of MCMC (see Figure 8.6 from the Volume 3 textbook), we hope to see smooth fuzzy caterpillars of the different nodes within a bounded region to approximate the continuing time series. If the region becomes unbounded, such as in the Fourier response states of Figure 4, then the forecast varies wildly out of control. Or in Figure 5, ran with worse parameters for RC, you can see that the flat lines in the response states tend to lead to flat lines in the forecasted time series.

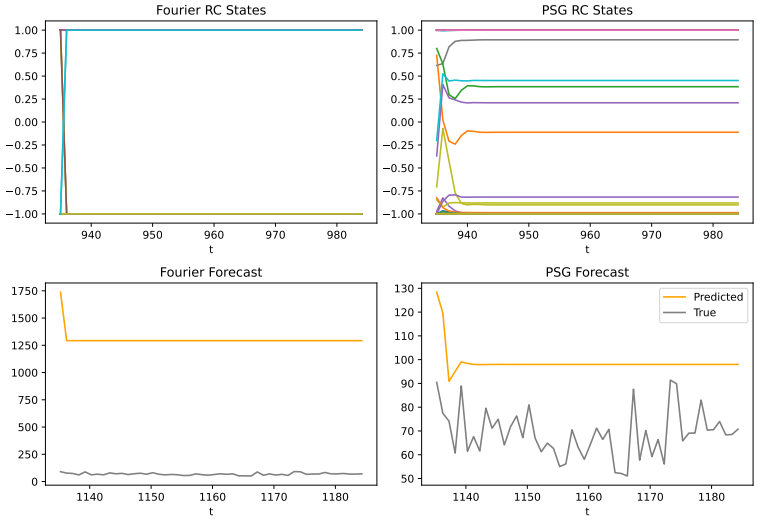

FIGURE 5. A few examples of poor parameter selections for reservoir computing (RC). Notice the correlation between the amplitudes of the response states (top) and the forecasts (bottom).