nature astronomy

SUPPLEMENTARY INFORMATION

https://doi.org/10.1038/s41550-017-0321-z

In the format provided by the authors and unedited.

A recurrent neural network for classification of unevenly sampled variable stars

Brett Naul[1]*, Joshua S. Bloom[1], Fernando Pérez[2,3,4] and Stéfan van der Walt[3]

[1]Department of Astronomy, University of California, Berkeley, CA, USA. [2]Department of Statistics, University of California, Berkeley, CA, USA. [3]Berkeley Institute for Data Science, University of California, Berkeley, CA, USA. [4]Department of Data Science and Technology, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA. *e-mail: bnaul@berkeley.edu

Supplementary information: A recurrent neural network for classification of unevenly sampled variable stars

Brett Naul, Joshua S. Bloom, Fernando Pérez, Stéfan van der Walt

October 22, 2017

1 Background on neural networks for time series data

Machine learning techniques for time series data typically map each sequence to a set of scalar-valued features that attempt to capture various distinguishing properties; these may range from simple summary statistics like median or maximum to parameters of complicated model fits. Features are then used as inputs to train models which map fixed-length feature vectors to the relevant labels. The process of building and selecting features can be difficult, requiring extensive expert knowledge and iterative fine-tuning, and feature generation itself may be computationally expensive. One distinct advantage of neural networks is the elimination of the need to manually create, compute, and select effective features [**?**]. Nearly all applications of neural networks to time series data rely on the assumption that samples are evenly spaced in time.

Artificial neural networks have previously been applied to many time series tasks, including classification and forecasting. The simplest approaches make use of fully connected networks (see, e.g., [1]), which treats each sequence as consisting of independent observations and thereby ignores the local temporal structure. Convolutional neural networks have also been applied to time series data with considerable success [2], in particular for the problem of speech recognition [3]. The enormous popularity of convolutional networks for image analysis has led to efficient optimization techniques and highly optimized software implementations, so convolutional networks tend to be faster and easier to train than other approaches, including recurrent networks (which our technique employs). As opposed to fully connected networks which ignore the local structure of sequences, convolutional networks are well-suited for recognizing local, short-term patterns in time series data. However, convolutional networks are less ideally suited for handling irregular sequences of unequal length. For this reason we focus on recurrent networks, which are designed to handle sequences of arbitrary length.

Recurrent networks have been found to be more difficult to train than convolutional networks due to their repeating structure, which makes them more susceptible to problems of exploding or vanishing gradients during training [4]. Another drawback of the recursive structure of recurrent networks is that recomputing the internal state for each input can lead to rapid loss of information over time and difficulty in tracking long-term dependencies; these difficulties led to the development of the popular long short-term memory (LSTM) [5] and gated recurrent unit (GRU) [6] architectures, which help information be retained over longer periods of time. Some sequence processing tasks for which recurrent neural networks have been successfully applied include recognition [7], machine translation [8], and natural language processing [9].

2 Frequency-domain characteristics of unevenly sampled time series

One important class of features that is often used in time series inference problems is frequency-domain properties, including estimates of period(s) or power spectra over some range of frequencies. By far the most common approach for transforming time series data between the time and frequency domains is the (discrete) Fourier transform (DFT). The default formulation of the DFT assumes that the data are uniformly sampled. If this assumption isn't met, more complex approaches are required that interpolate the data onto a uniform grid [10] or project it onto spaces with fixed, predefined bandwidth. A frequently used version of the latter approach is the Lomb-Scargle periodogram [11, 12], which estimates the frequency properties of a signal using a least squares fit of sinusoids. Once the power spectrum of a signal has been estimated using one of the above methods, features such as the dominant frequencies/periods can be extracted and used in machine learning tasks such as classification.

In the next section, we demonstrate a neural network architecture which is able to infer periodic behavior directly from an irregularly sampled time series without constructing an explicit estimate of the full power spectrum.

3 Simulation study: sinusoidal data

In order to evaluate the effect of depth, hidden layer size, and other architecture choices, we first carry out a number of experiments using simulated time series data generated from a know distribution. In particular, we construct periodic functions of the form

$$f_i(t) := A_i \sin(2\pi\omega_i t + \phi_i) + b_i \tag{1}$$

for randomly selected parameter values ω_i (frequency; we will also write $T_i = \omega_i^{-1}$ to denote period), A_i (amplitude), ϕ_i (phase), and b_i (offset). The parameter values are chosen as independent identically-distributed (i.i.d.) samples from the probability distributions

$$A_i \sim \mathcal{U}(0.5, 2), \tag{2}$$

$$\omega_i^{-1} = T_i \sim \mathcal{U}(1, 10), \tag{3}$$

$$\phi_i \sim \mathcal{U}(-\pi, \pi), \tag{4}$$

$$b_i \sim \mathcal{N}(0, 1), \tag{5}$$

where $\mathcal{U}(a, b)$ is the uniform distribution on $[a, b]$, and $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2. For each periodic function, we generate a set of $n_T = 200$ sampling times $(t_i^{(1)}, \ldots, t_i^{(n_T)})$ by sampling the differences from a heavy-tailed power law distribution $t_i^{(j+i)} - t_i^{(j)} \sim \text{Lomax}(0.05, 2)$, where the $\text{Lomax}(\lambda, \alpha)$ distribution has probability density function

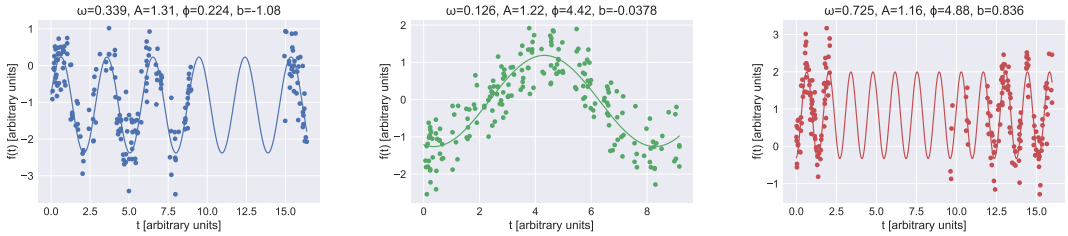
$$f(x; \lambda, \alpha) = \frac{\alpha}{\lambda}\left(1 + \frac{x}{\lambda}\right)^{-(\alpha+1)}.$$

The resulting distribution of times is such that the mean total time is $E[t_{n_T}] = 10$ but the sequence $(t_i^{(1)}, \ldots, t_i^{(n_T)})$ is highly unevenly sampled and may contain many large gaps between consecutive samples.

Our training dataset consists of $N_{\text{train}} = 50,000$ periodic functions and sampling time sequences generated using the above procedures; we also add white Gaussian random noise $\boldsymbol{\epsilon}_i^{(j)} \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.5$ to each measurement, so the final input data has the form

$$\boldsymbol{t}_i = (t_i^{(1)}, \ldots, t_i^{(n_T)}),\ \boldsymbol{y}_i = (f_i(t_i^{(1)}) + \epsilon_i^{(1)}, \ldots, f_i(t_i^{(n_T)}) + \epsilon_i^{(1)}). \tag{6}$$

Supplementary Figure 1 shows examples of randomly generated periodic functions and randomly selected points with added noise.



(a) (b) (c)

Supplementary Figure 1: **Examples of randomly generated periodic functions.** Panels a), b) and c) each contain values of a sinusoidal function with random parameters evaluated at random times. The distributions of the parameters are given in (2)–(6).

The accuracies and losses for each model are evaluated using a separate dataset of $N_{\text{valid}} = 10,000$ sequences generated using the same methodology.

3.1 Estimating period, phase, amplitude, offset

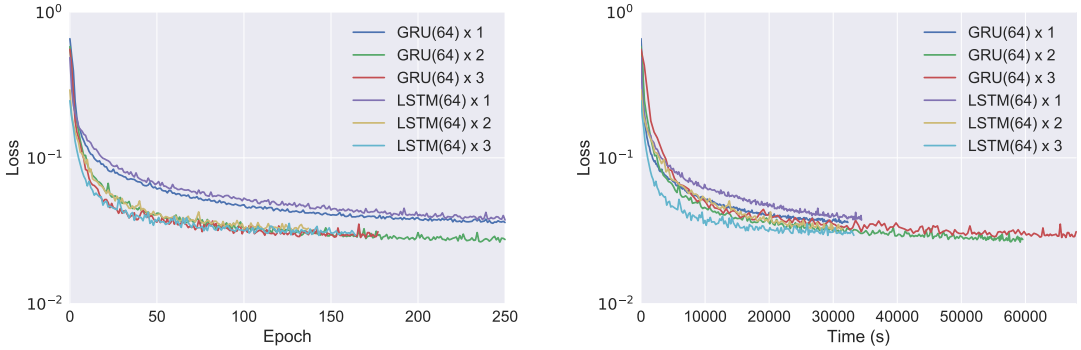
As our initial experiment, we implement a network which solves a supervised regression problem of estimating the parameters $\boldsymbol{\theta}_i = (\omega_i, A_i, \phi_i, b_i)$ [as defined in (1)] from the generated times and measurement values. The corresponding network consists of the encoder portion of the network shown in Supplementary Figure 1 of the main text with an output size of four. The network is trained to minimize the mean squared error

$$\text{MSE} = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \|\theta_i - \hat{\theta}_i\|_2^2 \tag{7}$$

for the given training data; the target values are first preprocessed by re-parametrizing $\boldsymbol{\theta}_i$ as $(T_i, A_i \cos\phi_i, A_i \sin\phi_i, b_i)$ (which leads to faster convergence) and then centering and scaling each target variable to have mean 0 and standard deviation 1. Our network is trained via the Adam optimization method [13] with standard parameter values $\beta_1 = 0.9, \beta_2 = 0.999$ using a learning rate of $\eta = 5 \times 10^{-4}$ and a batch size of 500. We also impose 25% dropout [14] between recurrent layers for regularization, which does not seem to be necessary for this simple task but is beneficial for the astronomical classification tasks discussed in the main text. All models were implemented in Python using the Keras library [15] and trained using an NVIDIA Tesla K80 GPU.

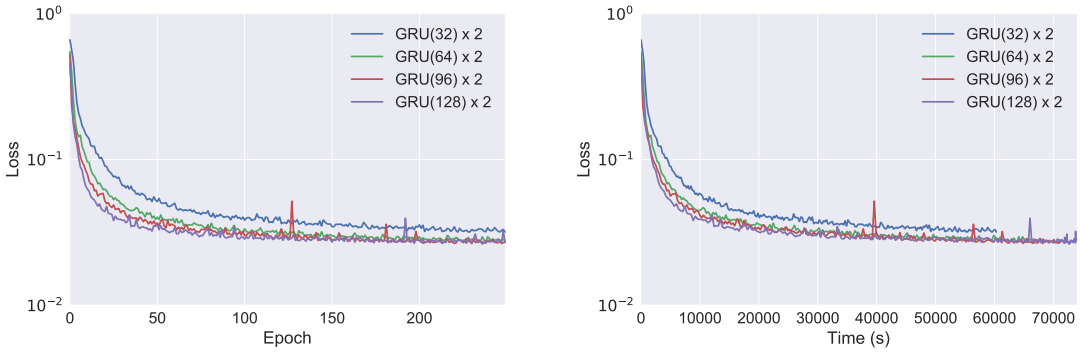
As seen in Supplementary Figures 2 and 3, the maximum accuracy achieved by the one layer networks is somewhat inferior to that of the multi-layer networks; nevertheless, it is clear that all of the architectures considered are able to accurately recover information about the periodic characteristics of the unknown functions, even in the presence of noise and sampling irregularity.

Supplementary Figure 11 depicts three realizations of the light curve resampling process described above. Supplementary Figure 4 compares the accuracy of an RNN estimator with two 96-unit GRU layers (ignoring the other estimated parameters besides the period) with that of a Lomb-Scargle estimate of the period, which has well-understood statistical convergence properties [16]. The accuracy of the two models is comparable, with the Lomb-Scargle estimate performing slightly better for this particular case. In the next we estimate the same periodic parameters using a semi-supervised approach based on autoencoder features.



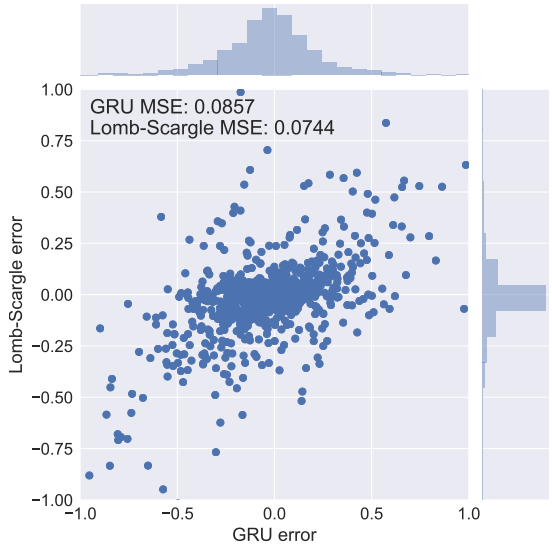
(a) By epoch (b) By time

Supplementary Figure 2: **Comparison of validation losses during training for various numbers and types of layers on the periodic parameters $\boldsymbol{\theta}_i$.** The loss plotted is the mean squared error over the validation set of the parameters being estimated as a function of a) training epoch and b) training wall time. Each colored line corresponds to a specific network architecture (layer type, layer size, and network depth) as shown in the legend: for example, "GRU(64) $\times$ 2" denotes two decoder GRU layers each composed of 64 hidden units.



(a) By time (b) By epoch

Supplementary Figure 3: **Comparison of validation losses during training for various sizes of layers on the periodic parameters $\boldsymbol{\theta}_i$.** The loss plotted is the mean squared error over the validation set of the parameters being estimated as a function of a) training epoch and b) training wall time. Each colored line corresponds to a specific network architecture (layer type, layer size, and network depth) as shown in the legend: for example, "GRU(64) $\times$ 2" denotes two decoder GRU layers each composed of 64 hidden units.



Supplementary Figure 4: **Comparison of period estimate residuals $T_i - \hat{T}_i$ for GRU and Lomb-Scargle models.** Each point on the scatter plot represents the error of the predicted period from each model; the histograms on the top and right show the overall distributions of errors across all samples for the GRU and Lomb-Scargle models, respectively.

3.2 Inferring periodic parameters from autoencoding features

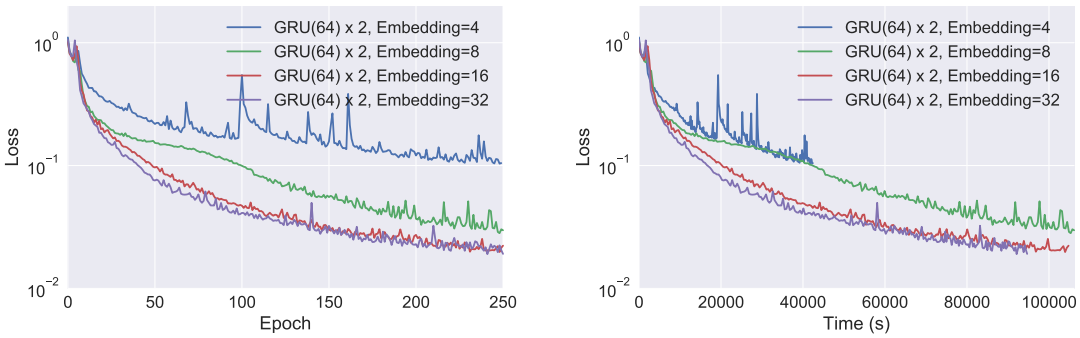
For the same simulated N_{train} periodic sequences generated above, we now train a full encoder-decoder network as depicted in Supplementary Figure 1 of the main text and attempt to recover the parameters ω_i, A_i, ϕ_i, and b_i from the encoding $\boldsymbol{v}_i$. Instead of minimizing a loss function for the parameters themselves, we now train the network to minimize the mean squared reconstruction error

$$\text{MSE} = \frac{1}{N_{\text{train}} n_T} \sum_{i=1}^{N_{\text{train}}} \sum_{j=1}^{n_T} \left(f_i(t_i^{(j)}) - \hat{y}_i^{(j)} \right)^2 . \tag{8}$$

Note that reconstruction target is the original (de-noised) periodic function, not to the raw measurement values.

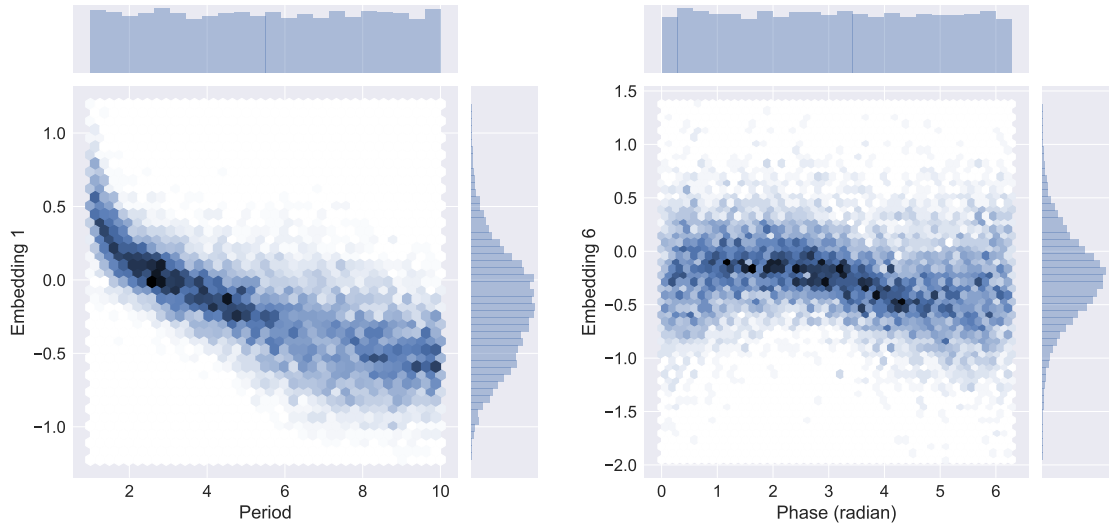
The autoencoder model has an additional hyperparameter (compared to the encoder for directly estimating the periodic parameters) of the size of embedding to use for the encoding layer. Supplementary Figure 5 shows the reconstruction accuracy for a fixed encoder and decoder architecture, with each consisting of two bidirectional GRU layers of size 64, for a range of embedding sizes. Although each target function can be fully characterized in terms of four parameters, it is clear that increasing the embedding size does improve the accuracy of the reconstruction up to a point. However, given a longer training time or different choice of learning rate, it is conceivable that a model with only four embedding values could achieve the same level of accuracy as those with higher-dimensional representations.

Although the inputs to the autoencoder are only the raw time series measurements and not the periodic parameters, the encoding layer does represents some alternative fixed-dimensional representation of the entire sequence, so it is reasonable to inquire if these values can be used to recover the original parameters $\boldsymbol{\theta}_i$. Supplementary Figure 6 demonstrates that strong (non-linear) relationships exist between some of the encoding values and the period and phase of the input sequences. While there is not any obvious



(a) By epoch (b) By time

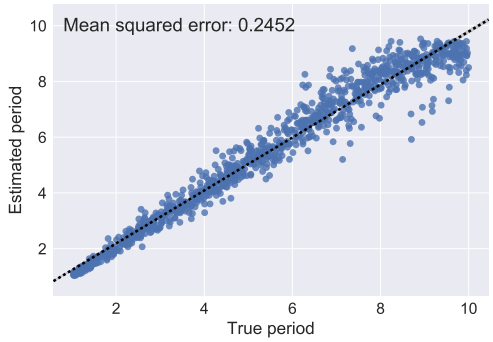
Supplementary Figure 5: **Comparison of GRU autoencoder validation losses during training for various embedding sizes.** The loss plotted is the mean squared error over the validation set of the reconstructed sequences as a function of a) training epoch and b) training wall time. Each colored line corresponds to a different dimensionality of embedding: for example, "GRU(64) × 2, Embedding=8" denotes two encoder and two decoder GRU layers each composed of 64 hidden units, with an embedding layer of dimension 8 inbetween.



(a) Autoencoder feature vs. period (b) Autoencoder feature vs. phase

Supplementary Figure 6: **Visual depiction of the relationships between 8-dimensional autoencoding time series features and period.** We observe clear nonlinear relationships between some of the learned autoencoder features and the a) period and b) phase of the input signals. The darkness of each point represents the relative frequency of that region of pairs of values, and the histograms on the top and right show the overall distributions of the estimated parameters and relevant autoencoder features, respectively.

closed-form relationship that should exist between a single autoencoding feature and any of the elements of $\boldsymbol{\theta}_i$, we can attempt to train a separate model mapping the encoding vectors to the parameters of interest. Since the relationships between the autoencoding features appear to be somewhat non-linear, we train a random forest regressor [17] with 1000 trees using `scikit-learn` [18] to estimate the period, phase, amplitude, and offset of each sequence using the autoencoding features. Supplementary Figure 7 shows the accuracy of our model for estimating period from the encoding values for various representation lengths; the random forest model is trained on the $N_{\text{train}} = 50,000$ training sequences and evaluated on the $N_{\text{valid}} = 10,000$ validation sequences. The accuracy of the model depends on the size of the embedding used, as well as indirectly on many other factors: the size, number, and type of recurrent layers used, the choice of optimization method and stopping criterion, and on the particular distribution chosen for the simulated data. Nevertheless, it is clear from this exercise that useful aggregate properties of time series can be inferred from features automatically generated by an autoencoder. In the next section we explore how these features can be used within the context of a real-world supervised classification problem.



Supplementary Figure 7: **True period vs. estimated period for random forest regressor model.** The dotted black line represents an exact match $T_i = \hat{T}_i$.

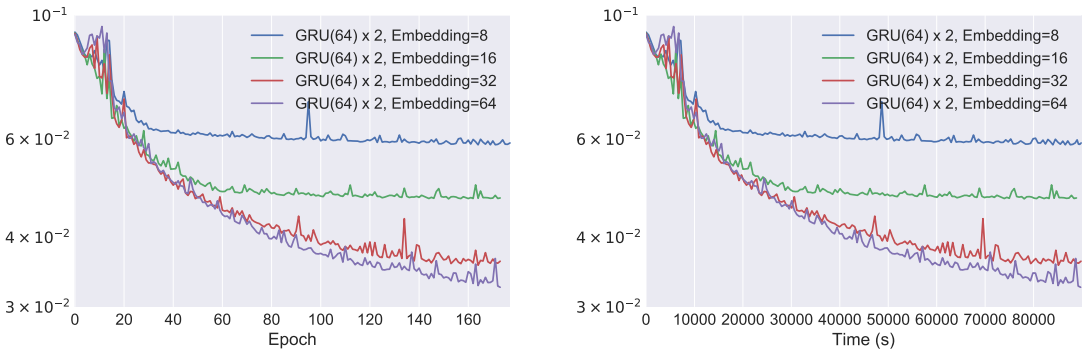
4 Application to astronomical light curves of variable stars

4.1 Validation accuracy

The distinction between training, test, and validation data is a subtle one in our problem. In order to avoid overly optimistic results, it is crucial to avoid making use of the test data in any way during the training process. Our experiment, however, is intended to simulate the case where some labeled and some unlabeled light curves are available: in this case, the practitioner is free to train the autoencoder using only the labeled light curves, or using the full dataset. We initially withheld some validation light curves in order to better understand the difference between reconstruction accuracy for light curves that are seen during training versus new light curves; the validation losses reported in Supplementary Figures 8 and 9 are computed using 20% of the available light curves which were withheld during training. After confirming that the reconstruction performance for training and validation light curves was comparable, we re-trained our autoencoder using all the available data before training our random forest classifiers.

Supplementary Figure 8 shows the validation loss over time for various embedding sizes. As expected, the embedding size required for an accurate reconstruction is larger than in the case of simulated purely sinusoidal data, and the average reconstruction error

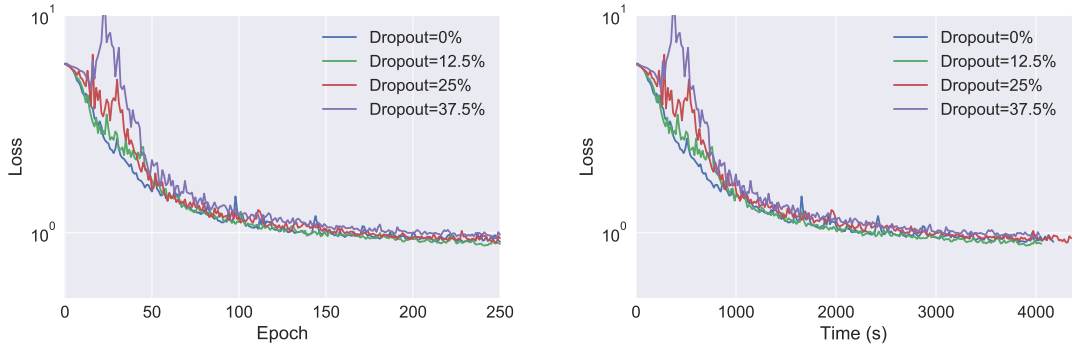
at convergence is somewhat higher. Nevertheless, we find that our autoencoder is able to accurately model light curves using a relatively parsimonious feature representation.



(a) By epoch (b) By time

Supplementary Figure 8: **Comparison of period-folded light curve autoencoder validation losses during training for various embedding sizes.** The loss plotted is the mean squared error over the validation set of the reconstructed sequences as a function of a) training epoch and b) training wall time. Each colored line corresponds to a different dimensionality of embedding: for example, "GRU(64) × 2, Embedding=8" denotes two encoder and two decoder GRU layers each composed of 64 hidden units, with an embedding layer of dimension 8 inbetween.

The above experiments using simulated data show that autoencoders can be used to model periodic sequence that has been irregularly sampled, and that the resulting encodings contain information about the frequency domain properties and other global characteristics of the input time series.



(a) By epoch (b) By time

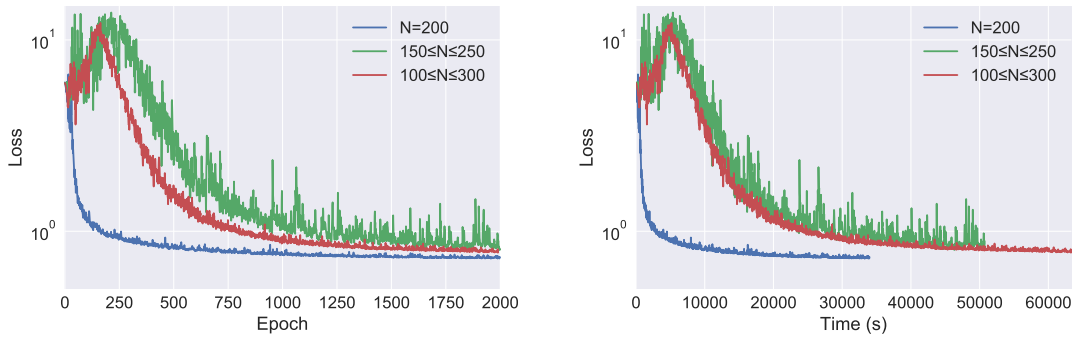
Supplementary Figure 9: **Comparison of period-folded light curve autoencoder validation losses during training for various levels of dropout.** The loss plotted is the mean squared error over the validation set of the reconstructed sequences as a function of a) training epoch and b) training wall time. Each colored line corresponds to a different dimensionality of embedding: for example, "GRU(64) × 2, Embedding=8" denotes two encoder and two decoder GRU layers each composed of 64 hidden units, with an embedding layer of dimension 8 inbetween.

As described in Supplementary Figure 1 of the main text, in our experiments we imposed 25% dropout [14] between recurrent layers in order to avoid overfitting. We tested the effect of dropout using the LINEAR dataset since its smaller size makes overfitting a

bigger concern. Supplementary Figure 9 shows how varying the dropout parameter affects the resulting reconstruction validation error; we find that the effect is minimal for this problem.

4.2 Sequence length

As described in the main text, before training our autoencoder we split the available light curves into subsequences of length $n_T = 200$. This step is purely optional, since our RNN architecture can accommodate input and output sequences of arbitrary length. However, the use of constant-length sequences is advantageous in training: it eliminates the need to pad input sequences and/or mask output sequences when computing the loss function, and empirically it seems to improve training speed (as measured in both iterations and minutes).



(a) By epoch

(b) By time

Supplementary Figure 10: **Comparison of period-folded light curve autoencoder validation losses during training for various sequence lengths.** The loss plotted is the mean squared error over the validation set of the reconstructed sequences as a function of a) training epoch and b) training wall time. Each colored line corresponds to a different dimensionality of embedding: for example, "GRU(64) × 2, Embedding=8" denotes two encoder and two decoder GRU layers each composed of 64 hidden units, with an embedding layer of dimension 8 inbetween.

Supplementary Figure 10 depicts the autoencoder training process for different lengths of sequence: first, for constant-size sequences of length $n_T = 200$; second, for variable-length sequences between $150 \leq n_T \leq 250$; and finally for $100 \leq n_t \leq 300$. In each case light curves from the LINEAR dataset were used for training, but for the variable-length sequences the length n_T was chosen at random for each light curve subsequence. Although the required training time is significantly longer for the variable-length problem, roughly the same quality of reconstruction is ultimately achieved. However, it is clear that at some point the difference in sequence length would become problematic, and another training approach might need to be explored (for example, grouping different lengths of light curves together and training in batches, or training different models for different sizes of data).

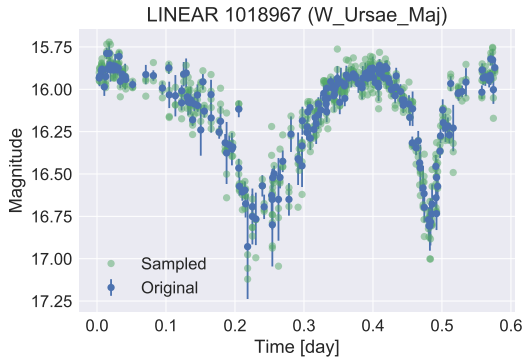
4.3 Data augmentation using noise properties

Data augmentation is commonly applied in the context of image processing tasks in order to increase the effective volumes of available training data (see, e.g., [19]). Unlike in the case of image data, for astronomical surveys the noise properties of the measurements are often known, which provides a natural way to generate synthetic training examples that mimic additional samples from the same survey for a given source. In particular, we

attempted to augment our light curve datasets by generating new training samples from existing light curves by adding Gaussian noise corresponding to the estimated measurement error at each time step:

$$\tilde{y}_i = \left(y_i^{(1)} + \epsilon_i^{(1)}, \ldots, y_i^{(n_T)} + \epsilon_i^{(n_T)}\right), \; \epsilon_i^{(j)} \sim \mathcal{N}(0, \sigma_i^{(j)}). \tag{9}$$

Supplementary Figure 11 depicts one realization of the light curve resampling process



Supplementary Figure 11: **Examples of generated LINEAR light curves using noise properties.** Original measurements (with error bars denoting measurement uncertainties) are shown in blue, and sampled values from the noise distributions in green.

described above.

Training our autoencoder on the LINEAR dataset for the same number of epochs but using the additional training samples leads to an increase in validation accuracy from 72.6% to 78.1% for the non-period folded model. However, for the period-folded model, the use of additional training examples did not improve performance, and in fact slightly diminished the validation accuracy (from 97.1% to 96.7%); this is likely because the autoencoder model is already able to accurately reconstruct the period-folded light curves, whereas the raw light curves are not well-modeled and therefore the additional training data is useful. We also attempted applying the same type of noise resampling for the previous example using the ASAS dataset, but the change in accuracy for the resulting classifier was negligible.

4.4 Direct supervised classification

The architecture we have described can easily be modified for other tasks, including direct supervised classification of unevenly sampled time series: in this case, the decoder module in Figure 1 would be replaced by one or more fully-connected layers, and the network would be trained to minimize the multi-class cross entropy classification loss. Such an approach trained on the period-folded data and sampling times also leads to high classification accuracy, achieving 98.6% average validation accuracy for the ASAS dataset and 96.7% for the LINEAR dataset. However, the fully supervised approach requires a large amount of labeled training data, whereas the autoencoder method can make use of either labeled or unlabeled data; the use of unsupervised methods is common in such cases where limited labeled data is available [20].

4.5 Transfer learning

In the experiments in the main text, data from a single survey is used to train an autoencoder, which is then used to produce features used for classification of labeled training examples from the same survey. Because the autoencoder and random forest models are decoupled, it is straightforward to incorporate additional or completely different training

data for the unsupervised portion of the model. In the ASAS example, we make use of both labeled and unlabeled examples when training the autoencoder; however, we could just as easily train the autoencoder using only unlabeled data, or data from an entirely different source. As a simple example, we found that features from the autoencoder trained on ASAS or LINEAR data yielded nearly the same level of classification accuracy for light curves from the other survey (less than a 1% reduction in each case).

References

[1] Lapedes, A. & Farber, R. Nonlinear signal processing using neural networks: prediction and system modeling. *Los Alamos National Laboratory Technical Report LA-UR-87-2662* (1987).

[2] LeCun, Y., Bengio, Y. *et al.* Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks* **3361**, 1995 (1995).

[3] Hinton, G. *et al.* Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine* **29**, 82–97 (2012).

[4] Pascanu, R., Mikolov, T. & Bengio, Y. On the difficulty of training recurrent neural networks. *ICML (3)* **28**, 1310–1318 (2013).

[5] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural computation* **9**, 1735–1780 (1997).

[6] Cho, K. *et al.* Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[7] Graves, A., Mohamed, A.-r. & Hinton, G. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, 6645–6649 (IEEE, 2013).

[8] Bahdanau, D., Cho, K. & Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[9] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J. & Khudanpur, S. Recurrent neural network based language model. In *Interspeech*, vol. 2, 3 (2010).

[10] Beylkin, G. On the fast fourier transform of functions with singularities. *Applied and Computational Harmonic Analysis* **2**, 363–381 (1995).

[11] Lomb, N. R. Least-squares frequency analysis of unequally spaced data. *Astrophysics and space science* **39**, 447–462 (1976).

[12] Scargle, J. D. Studies in astronomical time series analysis. ii-statistical aspects of spectral analysis of unevenly spaced data. *Astrophys. J.* **263**, 835–853 (1982).

[13] Kingma, D. & Adam, J. B. A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* **15**, 1929–1958 (2014).

[15] Chollet, F. Keras. https://github.com/fchollet/keras (2015).

[16] Schwarzenberg-Czerny, A. Accuracy of period determination. *Mon. Not. R. Astron. Soc.* **253**, 198–206 (1991).

[17] Breiman, L. Random forests. *Machine learning* **45**, 5–32 (2001).

[18] Pedregosa, F. *et al.* Scikit-learn: Machine learning in python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011).

[19] Simard, P. Y., Steinkraus, D., Platt, J. C. *et al.* Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, vol. 3, 958–962 (2003).

[20] Glorot, X., Bordes, A. & Bengio, Y. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, 513–520 (2011).