Percieving Circadian Gene Expression through Artificial Neural Networks

Damian Sowinski,[1, *] Alexander Crowell,[2, †] Jack Holland,[3, ‡] and Rawan Al Ghofaili[3, §]

[1]*Center for Cosmic Origins*
Department of Physics and Astronomy, Dartmouth College
[2]*Department of Genetics, Geisel School of Medicine, Dartmouth College*
[3]*Department of Computer Science, Dartmouth College,Hanover, NH 03755, USA*
(Dated: Last Updated: Sunday 16th November, 2014, 16:32)

Science is done!

INTRODUCTION

CIRCADIAN RHYTHMS

Circadian Clocks are an adaptive response to the evolutionary necessity of anticipating day night cycles. They occur widely in higher eukaryotes and sparsely in fungi and cyanobacteria. Because of the widespread and long-standing evolutionary history of organisms specializing their activity to a specific phase of the day night cycle, circadian clocks, where present, have been found to regulate almost every biological process imaginable. Notable among these are the cell cycle, DNA damage repair and metabolism, processes central to the majority of human disease [1]. Perturbation of the clock upsets these processes as well and has been shown to contribute to disease [2]. One of the most important recent trends in circadian research has been the interaction of the circadian cycle with disease treatment, specifically the timed delivery of medicine [3]. Additionally the circadian clock's modulation of such a wide variety of gene and protein expression along with the diverse processes employed to achieve that modulation provides an excellent opportunity for the study of gene regulation at every stage from transcription to degradation [4, 5].

The current state of the art in the identification of Circadian gene expression is the JTK_CYCLE algorithm (JTK) [6]. JTK uses a nonparametric test for monotonic orderings in the data over all possible period lengths and phases and for high (2hr) resolution data it provides high sensitivity, specifity and computational efficiency. JTK and earlier approaches including curve fitting [7], autocorrelation [8], Fourier analysis [? ?] and Fisher's G Test [?] all fall short however as the resolution of data decreases. Due to technical challenges in the maintenance and synchronization of cells as well as sequencing costs, dataset resolution is currently a major limiting factor in the identification of circadian genes. Therefore an improved computational technique for identifying circadian genes in low temporal resolution data could rapidly provide knew knowledge from publicly available pre-existing datasets

ARTIFICIAL NEURAL NETWORKS

In this section we go over the theoretical background of how to build an ANN, develop the formalism behind the backpropogation algorithm, and discuss possible candidates for tunable hyperparameters/functions used. We will develop both the feed forward and back prop algorithms for completeness.

ANNs are a supervised machine learning algorithm used for classification problems. Given some labeled set of data $\mathcal{D} = \{(\vec{\mathbf{x}}, \vec{\mathbf{y}})_i\}_{i=1}^D$, where to each multidimensional datum $\vec{\mathbf{x}}$ there is an associated multidimensional label $\vec{\mathbf{y}}$, the ANN learns from $\mathcal{D}$ how to place labels on new data. Processing a datum by the network is accomplished via the *feed forward* algorithm, while training is done through the *back propogation* algorithm.

The basic building block of an ANN is the perceptron. N signals go into the perceptron, are weighted, summed, and then put through some activation function to create an output signal:

$$\text{output} = \theta(\sum_{i=0}^N w_i \times \text{input}_i) \qquad (1)$$

The form of the **activation function**, θ is typcally chosen to be one that gives a value of 1 for arguments larger than some threshhold, and 0 otherwise. The backpropogation algorithm requires this function to be both differentiable and invertible, so typical choices include the logistic, arctan, and hyperbolic tangent functions. Note also that the sum begins at 0, leading to $N + 1$ inputs. The 0th input is always taken to be 1, and its corresponding weight is known as the **bias**.

We can stack perceptrons into layers, and in turn stack those together so that the inputs of the lth layer are the outputs coming from the $(l - 1)^{th}$ layer. This is the basic form of an ANN. We need only specify the architecture of the ANN (number of layers/{Input dimensionality,number of perceptrons in each layer}), which will be $L/\{N_0, N_1, \ldots, N_L\}$. Note that N_0 represents the dimensionality of the input signal, and N_L is the dimensionality of the output layer. Between each pair of layers there is a corresponding weight matrix, $\mathbf{w}^{(l)} : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$, the components of which are $w_{ij}^{(l)}$, where l is the target layer, i is the index of the *to* perceptron in the target layer, and

j is the *from* perceptron in the domain layer. We denote the set of all these matrices $\mathbf{W}$, and refer to this set as the **architecture** of our ANN. With these definitions in hand, we can write the output of the ith perceptron in the lth layer in compact form:

$$x_i^{(l)} = \theta(\sum_{j=0}^{N_{l-1}} w_{ij}^{(l)} x_j^{(l-1)}) \qquad (2)$$

where once again,, the $w_{i0}^{(l)}$'s represent biases.

With the architecture in place, we can now easily state the feed forward algorithm, 1. It is nothing more than allowing an input signal propagate through the network, and eventually pop out at the end.

This is all fine and dandy, but what we want is to be able to train the network based on the known labels. We need to compare the output of the network with the known label, define an error based on that comparison, and then alter the weights in the network so as to decrease that error. Furthermore, when the weights are changed, we do not want them to take on any values (one can imagine one weight becoming so high that it dominates the flow of signals in the ANN), so we must also regularize the way in which the error function is causing the weights to change. These statements can be made more precise.

Initialize an ANN with randomized weights. Take a datum $(\vec{\mathbf{x}}^{(0)}, \vec{\mathbf{y}})$ and have the ANN process the input to get an output $(\vec{\mathbf{x}}^{(L)}, \vec{\mathbf{y}})$. We define the cost of the function as an explicit function of the output, the **error**, and an explicit function of the weights, the **regularizer**:

$$C(\vec{\mathbf{x}}^{(L)}, \vec{\mathbf{y}}, \mathbf{W}) = E(\vec{\mathbf{x}}^{(L)}, \vec{\mathbf{y}}) + R(\mathbf{W}) \qquad (3)$$

This factorization of the arguments is a bit deceptive, since the output is an implicit function of the weights of the network. This fact will be crucial as we start exploring weight-space and will need to take gradients of the cost with respect to the weights. Our goal will be to decrease the cost of our ANN at each iteration by changing the weights of the network. Over the course of many epochs, the network will learn the patterns in the data, and we will have minimized the cost. So, how do we choose the change in weights to achieve this at each iteration?

We want:

$$C(\vec{\mathbf{x}}^{(L)}, \vec{\mathbf{y}}, \mathbf{W} + \delta\mathbf{W}) \leq C(\vec{\mathbf{x}}^{(L)}, \vec{\mathbf{y}}, \mathbf{W})$$

Expanding the RHS to first order in $\delta\mathbf{W}$ we have:

$$\nabla C \cdot \delta\mathbf{W} \leq 0$$

where the gradient is taken with respect to the weights.

We can easily satisfy this equation if we choose:

$$\delta\mathbf{W} = -\eta\nabla C$$
$$\Downarrow$$
$$\delta w_{ij}^{(l)} = -\eta \frac{dC}{dw_{ij}^{(l)}} \qquad (4)$$

This is our **learning rule**. The hyperparameter η is called the **learning rate**. Taking the gradient of the regularizer is simple, since it is an explicit function of the weights. To take the gradient of the error, however, we will have to employ some mathematical trickery. We have to use the fact that the output depends on the weights, making the error an implicit function of the weights.

First off we will break up the reweighing of the ANN layer by layer. So we will start at the output layer, and move backward. To proceed we need one result on how to differentiate an output neuron with respect to weight:

$$\frac{dx_n^{(l)}}{dw_{ij}^{(l)}} = \frac{d}{dw_{ij}^{(l)}}\theta(\sum_{k=0}^{N_{l-1}} w_{nk}^{(l)} x_k^{(l-1)})$$
$$= \theta' \circ \theta^{-1}(x_n^{(l)})\delta_{ni} x_j^{(l-1)} \qquad (5)$$

if we define $\phi = \theta' \circ \theta^{-1}$, the gradient of the error can then be written:

$$\frac{dE}{dw_{ij}^{(L)}} = \sum_{k=1}^{N_L} \frac{\partial E}{\partial x_k^{(L)}} \frac{dx_k^{(l)}}{dw_{ij}^{(l)}}$$
$$= \frac{\partial E}{\partial x_i^{(L)}} \phi(x_i^{(L)}) x_j^{(L-1)} \qquad (6)$$

We can generalize this for the gradient w.r.t. other layer weights by defining:

$$\delta_i^{(l)} = \begin{cases} \phi(x_i^{(l)})\frac{\partial E}{\partial x_i^{(l)}} & l = L \\ \phi(x_i^{(l)})\sum_j \delta_j^{(l+1)} w_{ji}^{(l+1)} & l < L \end{cases} \qquad (7)$$

This is known as the δ-rule, and is used to write the learning rate in its compact form:

$$\delta w_{ij}^{(l)} = -\eta\left(\frac{dR}{dw_{ij}^{(l)}} + \delta_i^{(l)} x_j^{(l-1)}\right) \qquad (8)$$

This is all we need to define the back propogation algorithm, 2.

To go any farther we need to discuss the possible choices for the functions that define our network: the activation, error, and regularizer. We will now discuss each of these in turn.

Function Choices

Activation Function

The only constraints that we have on our choice is that the function be both differentiable and invertible. Typ-

ical choices are therefore any smooth strictly monotonic function. We should also specify the domain of the inputs. The list below has been normalized to the range $(0, 1)$, and we have included the backprop functions for completeness:

- Sigmoid (blue)

$$\theta(x) = \frac{1}{1 + e^{-x}} \tag{9}$$
$$\phi(x) = x(1 - x)$$

- Hyperbolic Tangent (magenta)

$$\theta(x) = \frac{1}{2}(1 + \tanh x) \tag{10}$$
$$\phi(x) = 2x(1 - x)$$

- Arctangent (gold)

$$\theta(x) = \frac{1}{2}(1 + \frac{2}{\pi}\arctan x) \tag{11}$$
$$\phi(x) = \frac{1}{\pi}\cos^2\frac{\pi}{2}(2x - 1)$$

The activations are plotted in figure **??**. The qualitative differences are mostly with the arctan function, which requires incredibly high absolute value inputs to generate output close to 0 or 1. The corresponding backprop functions are plotted in figure.

Algorithms

Algorithm 1 Feed Forward

procedure FEEDFORWARD($\vec{\mathbf{x}}^{(0)}$, $\mathbf{W}$)

$L \leftarrow$ number of layers in $\mathbf{W}$
$N_l \leftarrow$ number of perceptrons in $\mathbf{w}^{(l)} \in \mathbf{W}$
$N_0 \leftarrow \dim(\mathrm{domain}((\vec{\mathbf{x}}^{(0)}))$

for $l = 1, 2, \ldots, L$
$\quad x_0^{(l-1)} \leftarrow 1$
$\quad$**for** $i = 1, 2, \ldots, N_l$
$\quad\quad x_i^{(l)} \leftarrow \theta(\sum_{j=0}^{N_{l-1}} w_{ij}^{(l)} x_j^{(l-1)})$ $\qquad \triangleright$ and store
$\quad$**end**
end

return $\mathbf{X} = (\vec{\mathbf{x}}^{(0)}, \vec{\mathbf{x}}^{(1)}, \ldots, \vec{\mathbf{x}}^{(L)})$

end procedure

Algorithm 2 Back Propogation

procedure BACKPROP($\mathbf{X}$, $\vec{y}$, $\mathbf{W}$)

$L \leftarrow$ number of layers in $\mathbf{W}$
$N_l \leftarrow$ dimensionality of each $\vec{\mathbf{x}}^{(l)} \in \mathbf{X}$

for $l = L, L-1, \ldots, 2, 1$
$\quad$**for** $i = 1, 2, \cdots, N_l$
$\quad\quad$**if** $l = L$
$\quad\quad\quad \delta_i^{(l)} \leftarrow \phi(x_i^{(l)})\nabla_i E$
$\quad\quad$**else**
$\quad\quad\quad \delta_i^{(l)} \leftarrow \phi(x_i^{(l)}) \sum_{j=0}^{N_{l+1}} \delta_j^{(l+1)} w_{ji}^{(l+1)}$
$\quad\quad$**end**
$\quad\quad$**for** $j = 1, 2, \ldots, N_{l-1}$
$\quad\quad\quad w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\left(\nabla_{ij}^{(l)} R + \delta_i^{(l)} x_j^{(l-1)}\right)$
$\quad\quad$**end**
$\quad$**end**
end

return $\mathbf{W}$

end procedure

Algorithm 3 Train Network

procedure TRAIN($\mathcal{D}$, $\mathbf{W}$, numEpochs)

for $i = 1, 2, \ldots,$ numEpochs
$\quad$**randomize**($\mathcal{D}$)
$\quad$**for** $d \in \mathcal{D}$
$\quad\quad \mathbf{X} \leftarrow$ FeedForward(d_1, $\mathbf{W}$)
$\quad\quad \mathbf{W} \leftarrow$ BackProp($\mathbf{X}$, d_2, $\mathbf{W}$)
$\quad$**end**
end

return $\mathbf{W}$

end procedure

Algorithm 4 Cross Validation

procedure CROSSVAL($\mathcal{D}$, $\mathbf{W}$, numEpochs, N)

randomize($\mathcal{D}$)
partition $\mathcal{D} = \mathcal{D}^1 \sqcup \mathcal{D}^2 \sqcup \cdots \sqcup \mathcal{D}^N$
for $i = 1, 2, \ldots, N$
$\quad \mathbf{W} \leftarrow$ Train($\mathcal{D}\backslash\mathcal{D}^i$, $\mathbf{W}$, numEpochs)
$\quad \mathrm{MSE}_i \leftarrow \|\mathcal{D}_2^i - \mathrm{FeedForward}(\mathcal{D}_1^i, \mathbf{W})\|_2$
end
CVerror $\leftarrow$ mean($\{MSE_i\}$)

return CVerror

end procedure

DATA

The training dataset employed in this project was derived from Neurospora Crassa a filamentous fungus which is one of the primary model organisms in the study of
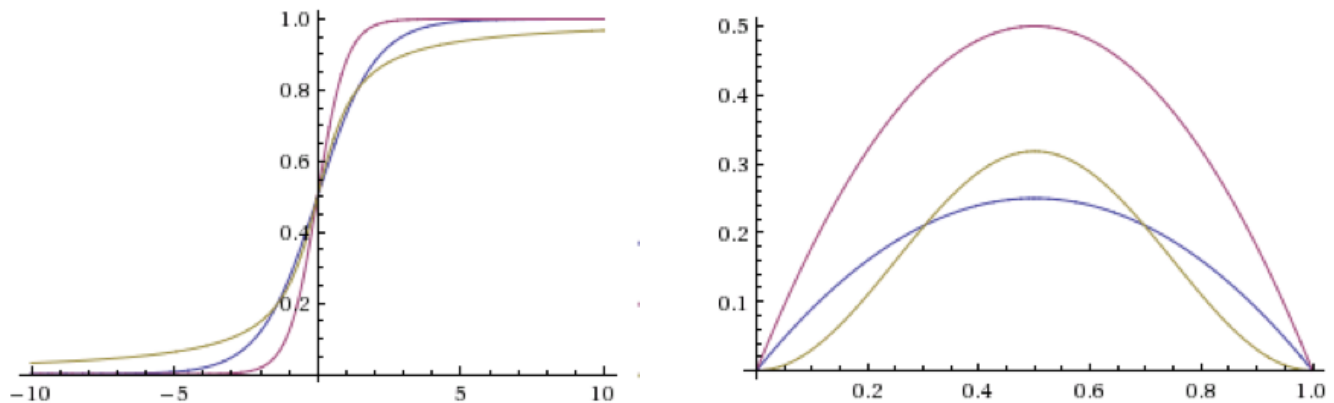
FIG. 1: On the left we have the activation functions plotted, and the back propagated functions on the right. The sigmoid, hyperbolic tangent, and arctangent are blue, magenta, and gold, respectively. Not that the main difference

circadian clocks due to it's genetic tractability, ease of culture and possession of a semantically similar clock to higher eukaryotes. All samples were synchronized with a light to dark transition and three biological replicates were collected every two hours for 48 hours. These samples were then sequenced and aligned to the neurospora genome using tophat. Feature counting was performed with HTSeq count and normalization performed with the DESeq bioconductor package. The resulting gene counts were then processed using JTK for Circadian classification. The training data is comprised of a combination of the normalized gene counts at each timepoint and a classifier of circadian, non-circadian or unclassified based on the adjusted P value output by JTK. The cutoffs for these classifiers were determined by examining the P and Q value distributions for all genes along with hierarchically clustered heatmaps for each set.

EXPERIMENT

CONCLUSION

* Electronic address: Damian.Sowinski.GR@dartmouth.edu
† Electronic address: Alexander.M.Crowell.GR@dartmouth.edu
‡ Electronic address: Jack.E.Holland.GR@dartmouth.edu
§ Electronic address: Rawan.al.Ghofaili.GR@dartmouth.edu

[1] Dunlap JC. Molecular bases for circadian clocks. *Cell*, 96(2):271–290, 1999.
[2] Bass J and Takahashi JS. Circadian integration of metabolism and energetics. *Science*, 330(6009):1349–1354, 2010.
[3] Zhang R, Lahens NF, Ballance HI, Hughes ME, and Hogenesch JB. A circadian gene expression atlas in mammals: Implications for biology and medicine. *PNAS*, 111(45):16219–16224, 2014.
[4] Mehra A, Baker CL, Loros JJ, and Dunlap JC. Post-translational modifications in circadian rhythms. *Trends in Biochemical Sciences*, 34(10):483–490, 2009.
[5] Menet JS, Pescatore S, and Rosbash M. Clock:bmal1 is a pioneer-like transcription factor. *Genes Development*, 28(1):8–13, 2014.
[6] Hughes ME, Hogenesch JB, and Kornacker K. JTK_CYCLE: An efficient nonparametric algorithm for detecting rhythmic components in genome-scale data sets. *Journal of Biological Rhythms*, 25(5):372–380, 2010.
[7] Straume M. Dna microarray time series analysis: automated statistical assessment of circadian rhythms in gene expression patterning. *Meth Enzymol*.
[8] Levine JD, Funes P, Dowse HB, and Hall JC. Signal analysis of behavioral and molecular cycles. *BMC Neuroscience*.