DISTINGUISHING BEHAVIOR FROM HIGHLY VARIABLE NEURAL RECORDINGS USING MACHINE LEARNING

by

JONATHAN P. SASSE

Submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Biology

CASE WESTERN RESERVE UNIVERSITY

May, 2018
We hereby approve the thesis of

Jonathan P. Sasse

...candidate for the degree of Master of Science...*

Committee Chair

Dr. Sarah Diamond

Committee Member

Dr. Hillel Chiel

Committee Member

Dr. Karen Abbott

Committee Member

Dr. Peter Thomas

Data of Defense

March 23rd, 2018

*We also certify that written approval has been obtained for any proprietary material contained therein.
# Table of Contents

Abstract ......................................................................................................................................................... 4  
Introduction .................................................................................................................................................. 5  
  Background on *Aplysia* .......................................................................................................................... 7  
  *Aplysia*’s use as a model organism ........................................................................................................ 8  
  Brief history of Artificial Neural Networks .............................................................................................. 14  
  Standard Feed-Forward Neural Network ................................................................................................. 17  
  Long Short-Term Memory Networks ....................................................................................................... 18  
  Entropy .................................................................................................................................................... 23  
  Backpropagation ..................................................................................................................................... 25  
Methods ...................................................................................................................................................... 31  
Results ......................................................................................................................................................... 35  
Discussion ................................................................................................................................................... 40  
Appendix ..................................................................................................................................................... 43  
References .................................................................................................................................................. 47
List of Tables .................................................................................................................................................................
Table 1 .......................................................................................................................................................................... 19
Table 2 .......................................................................................................................................................................... 38
Distinguishing Behavior from Highly Variable Neural Recordings with Machine Learning

Abstract

By

JONATHAN P. SASSE

Flexible and robust neural pathways are ubiquitous in animals. Previous work has demonstrated that variability in feeding behavior in the marine mollusk *Aplysia californica* can be useful to the animal—in general, motor components relevant to feeding show higher variability within animals, even as they vary less across different animals. (Cullins et al. Current Biology 2015). This variability, though, makes interpreting neural recordings challenging, especially in an automated context. In this research, we explore the ability for a combination of artificial neural network architectures (Long Short-Term Memory [LSTM] and Dense Fully Connected) to not only classify behaviors but to distinguish behaviors prior to any observable cue. The examined four channel recordings came from the key protractor muscle (I2) and three motor nerves that control the feeding apparatus of *Aplysia californica* during feeding behaviors. Each channel of the recordings had an LSTM dedicated to learning how to discern bites from swallows from white noise. The output from these four LSTMs were then passed to a dense, fully connected layer for a final classification using context from all channels. Surprisingly, the overall architecture appears able to discriminate bites from swallows (at an accuracy between 97 and 99%) at least half a second before the classic marker (I2 firing frequency exceeding 10hz) occurs. These results suggest that previously disregarded sub-threshold activity may contain high (or at least sufficient) levels of contextual information for behavioral classification which raises exciting questions about possible implications for closed circuit controllers and medical technology. *TensorFlow* was used with a Python interface to implement the networks.
Introduction

Analysis of neural signals is difficult. The dynamic nature of neurons, including neuromodulation, temporal encoding, sensory feedback, complex architectures leading to dynamic cross-talk amongst neurons, and many more qualities, contribute to this difficulty. Currently, the most common methods for analyzing signals are built around quantification of firing frequencies and “spike-sorting” (Lu et al., 2013; Erguzel et al., 2014; Rey et al., 2015).

While these techniques provide a framework for identification of the contributions of individual neurons and analysis of pre-recorded signals, they struggle to transition this success into more dynamic, on-line scenarios (Rey et al., 2015). One potential reason for this struggle is that, for the most part, spike shape and firing frequency are static measures of highly dynamic systems with complex features that unfold on multiple spatiotemporal scales (Morgan et al., 2002; Proekt et al., 2008; Akhmedov et al., 2013; Bruno et al., 2015; Dacks et al., 2013; Lu et al., 2015). Use of these static

Figure 1 - A: Sample recording from four separate extracellular electrodes, recording simultaneously within an intact, behaving animal (Cullins et al. 2010). The technique uses a multi-electrode system to record from the I2 muscle, radular nerve (RN), buccal nerve 2 (BN2) and buccal nerve 3 (BN3). Though these are the nerves and muscles recorded from, the technique picks up information from many neurons “in the neighborhood”. Custom Mathematica software was developed to automatically identify the activity of these individual neurons and muscles present in this recording. The blue segment highlights activity of the I2 muscle, which is used in protraction. The red section highlights activity of the B8a and B8b neurons that directly impact grasper closure and the green segment highlights B6/B9, the longest-acting neurons contained within the retraction pattern. Neuron B43 (orange segment embedded within the green) is of great importance as when it stops firing, the behavior, in most cases, is finished. B: Schematic representation of the processes of biting and swallowing as performed by the buccal mass of Aplysia californica with approximate positioning of the three major biomechanical events from the recording indicated. An enlarged version of this figure can be found in the appendix.
measurements to quantify behavior naturally complicates matters.

In an attempt to address this potential disconnect between available methods and the need for extremely reliable on-line classification techniques, the possible use of the recent advances in artificial neural networks on neural signal classification was investigated. Four channel recordings (electromyograms and electroencephalograms, Figure 1) from the marine mollusk *Aplysia californica* that had been made and curated during the inquiries leading to Cullins et al., 2015 *Current Biology* publication were analyzed using a combination of recurrent and fully connected neural networks. Channel 1 within the recording corresponds to an EMG from the I2 muscle, the primary muscle involved in protraction of the grasper. Channel 2 is an ENG from the radular nerve (RN); the radula is analogous to a human mouth in that it opens to allow food to enter the buccal cavity and closes to grip such food. Lastly, channels 3 and 4 correspond to ENGs from buccal nerve 2 (BN2) and buccal nerve 3 (BN3) respectively. BN2 contains projections from a large number of neurons involved in retraction of the grasper, which retraction, by necessity, signals the end of the behavior. Patterns of overlap and timing of activity of RN and BN2 are some of the clearest indicators of ingestive vs. egestive motor patterns (Morton and Chiel 1993). BN3 activity, most often associated with the B4/B5 retraction-associated motorneurons, typically marks the transition from protraction to retraction. B4/B5 are also known to have a somewhat inhibitory interaction with B8, the main radular closing motorneuron.

During the curation process, the full recordings made had been segmented into hand-classified sections (similar to what is seen in Figure 1) that corresponded with observed feeding behaviors in *Aplysia* (specifically, biting, i.e., an attempt to grasp food; and swallowing, i.e.,
pulling grasped food into the buccal cavity). Through use of these classified segments, the neural networks were trained to identify bites from swallows from noise. Before proceeding to a discussion of the full methodology and neural network architecture used, however, it will be helpful to more thoroughly discuss some of the biomechanics of *Aplysia* and how neural networks work toward learning any particular task.

**Background for *Aplysia***

General anatomical studies of *Aplysia* can be traced back to the early 1800’s, although specific targeting of the neural anatomy associated with feeding did not begin until the early 1970’s. Specifically, Daniel Gardner (1971) explored the bilateral symmetry of the two buccal ganglia and their associated interneurons. Detailed studies of feeding behaviors displayed by *Aplysia* followed shortly thereafter from Irving Kupfermann in 1974. Kupfermann demonstrated that *Aplysia* are able to adjust their biting or swallowing frequency depending upon their internal state (satiety, previous actions) and external stimuli. Kupfermann observed that *Aplysia*’s initial biting response is highly repetitive, with movement of the head and repeated protraction of the grasper in an attempt to locate food. Once contact with food has been made, the system switches from biting to swallowing and behaviors become much more variable. He noted that many transitional feeding behaviors exist in *Aplysia*, including “bite/swallows” and swallows shifting into rejections (egestion of partially swallowed food/inedible objects).

Studies of Eric Kandel and his associates established the value of *Aplysia* as a model organism, for which Kandel won the Nobel Prize in 2000. On multiple occasions, Kandel has stated that *Aplysia* has many characteristics that make it an appealing choice for neuroscience,
i.e.: well-honed feeding and escape responses, relatively few (on the order of 20,000) neurons that can be quite large (up to 1 mm in diameter), a simple body plan (as a muscular hydrostat, a large portion of Aplysia’s size is due to liquid as opposed to viscera), and extremely robust tissues.

**Aplysia’s use as a model organism**

Some of Dr. Kandel’s more notable findings include the identification of varying concentrations of cAMP (cyclic adenosine monophosphate) in the presence of repeated activations of certain synapses. This work was one of the first demonstrations of the role of cAMP as a signaling molecule associated with plasticity (Cedar and Kandel, 1972). In addition to studying cAMP, Kandel identified the role of covalent modification in short-term memory and the role of transcriptional modification in long-term memory (Goelet et al., 1986).

One of the premiere Neurology textbooks, Dr. Kandel’s *Principles of Neural Science 5th Edition* lays out much of the work he conducted, and oversaw, on *Aplysia*. The work that led to identification of cAMP as a key second-messenger molecule associated with memory looked at repeated stimulation of the gill and siphon withdrawal reflex in *Aplysia*. Short-term memory is thought to take on many

![Figure 2](https://commons.wikimedia.org/wiki/Category:Aplysia_californica)

Figure 2 - Stock image of Aplysia californica. The raised ridge to which the arrow points is the gill. When the gill opens, the siphon is exposed. When the gill retracts, one side folds over the other as the animal contracts, apparently in an attempt to become as small as possible. Image from commons.wikimedia.org
forms, two of which, sensitization and habituation, served as the basis for the discovery that morphological changes are involved in memory (Pinsker et al., 1970; Castelluci et al., 1978; Bailey and Chen, 1983). These two forms of memory are both considered to be forms of implicit memory, in other words, they govern processes and reflexes that do not require conscious attention on the part of the organism within which the memory resides. Experimenters gently brushed the edge of the gill (Figure 2), which caused an immediate retraction of the gill and siphon through contraction of the muscle that resides deep to the skin (Castelluci et al., 1978). Repeated stimulations caused the effectiveness of sensory neuron activation to diminish (it was also found that the connection between excitatory interneurons and their respective motor neurons also was diminished) – quantal analysis showed a downregulation in the amount of glutamate released by the sensory neurons (Castelluci and Kandel 1974). This type of habituation, i.e. habituation that takes place entirely within the pathway for the behavior without concomitant input from external sources, is called homosynaptic depression.

When the experimental regimen only is carried out once, the resultant depression lasts just a few minutes, however investigators found that four sessions of 10 stimuli separated by variable time intervals (ranging from a few hours to a day) produced long-term habituation that lasted up to three weeks. While short-term habituation was mediated by a drop in quantal content, long-term habituation showed significant anatomical changes, with a drop from 90% to 30% of identified sensory neurons projected onto known motor neurons. During this study of sensitization, a similar, though opposite, change was noticed when new synaptic connections were formed in response to aversive stimuli (tail shocks) (Martin et al., 1997). This investigation
led to the discovery of the key roles that cAMP and covalent modification play in synaptic plasticity.

One group of interneurons involved in sensitization in *Aplysia* was found to be serotonergic and Kandel observed this group’s synaptic release of serotonin caused increased activity in adenylyl cyclase in the postsynaptic neuron – which, in some instances, was the presynaptic terminals of the sensory cells involved in the gill and siphon withdrawal reflex (Guan et al., 2002). This increased activity led directly to an increased level of cAMP in the sensory cells which, in turn, through multiple pathways, increased the levels of protein kinase A (PKA) and protein kinase C (PKC) (Huang and Kandel, 1998). The phosphorylation mediated by PKA and PKC (re: covalent modification) directly led to an increase in neurotransmitter release (increased quantal content) by the sensory neurons.

In addition to the findings on short-term memory, Andrea Casadio, a member of Dr. Kandel’s lab, found that the cAMP-PKA pathway additionally is crucial for long-term memory. In order for changes in long-term memory to occur, however, the levels of cAMP must be elevated long enough for a sufficient amount of PKA’s catalytic subunit to translocate into the nucleus. Once in the nucleus, the subunit activates cAMP response element binding protein 1 (CREB-1), which alters gene transcription levels by binding to transcription promoters (Casadio et al., 1999; Kandel 2001). Interestingly, these two processes (short and long-term memory) appear to be highly conserved as the general pathways for both have been confirmed in vertebrate models (i.e.: cAMP->PKA->CREB-1 might have additional intermediaries, but these still are the key elements). The full mechanism (as it is currently known) can be found in chapter 66 of *Principles of Neural Science 5th Edition*. 
Aplysia’s relatively low number of neurons enables rich analysis of the role of central pattern generators (CPGs) and neuromodulation in complex behavior networks. Two widely studied CPGs in Aplysia are those associated with locomotion and feeding. Bruno et al., 2015 took advantage of the small number of neurons associated with Aplysia locomotion and derived some very interesting results. Through the use of fluorescent imaging, the researchers were able to identify discrete locations within the pedal ganglia whose activities highly correlated with segments of locomotive movement. In essence, a correlation between discrete groupings of neurons within the pedal ganglia and discrete portions of the movement process was demonstrated. Mapping of this sort is a remarkable technique; it not only appears to show evidence of spatial specialization within neurons, it (potentially) demonstrates a highly effective method discovered by evolution to handle the architecture of a dynamic system.

Feeding within Aplysia also is governed by an oft-studied CPG. Friedman et al., 2009 explored the role of the internal state of the CPG in the determination of the efficacy of generated behaviors. Given what is known about plasticity and the dependence of synaptic activity on downstream neuronal outputs, it is not surprising that a state dependence on the output generated by the CPG was identified. The primary metric used to quantify the state dependence was the activity of B48 and B44 motorneurons associated with opening of the radula. They found that not only does network state influence any generated outputs, it also influences the way inputs are processed. This was a remarkable finding as it is direct evidence of the fact that short-term, covalent modification plays a role in priming not only individual synapses but entire dynamic networks.
More recently but along similar lines, Lu et al., 2015 found that this sort of priming is not only present but may be essential to the dynamic system’s ability to effectively perform its physiological function. Lu et al. also took this finding a step further and showed that certain neurons (B3, B6 and B9) involved in the feeding process are activated during biting in order to prime the system to swallow. Even though the relatively “simple” nature of Aplysia’s nervous system is the feature that attracted neuroscience researchers to it decades ago, the dynamics this system displays are incredibly rich. Given that Aplysia is known to have feeding patterns that exist on a continuum between biting and swallowing, this hard-coded systemic priming that functions to facilitate the shift between biting and swallowing makes sense. One could ask if this facilitation serves to leave the system in a perpetual state of “anticipation” during biting, allowing for rapid behavior shifting as opposed to the need to react and generate a behavior “from scratch”.

Behavior initiation, however, starts with the higher order neurons, that is, cerebral-to-buccal interneurons (CBI) 2 and 3. Morgan et al. 2001 investigated the role these two command-like neurons play in ingestive vs. egestive motor patterns. Activation of only CBI-2 was indicative of egestive motor patterns while combinatorial activation of CBI-2 and CBI-3 through electrical coupling initiated ingestive motor patterns. Given that these higher order neurons control ingestion vs. egestion, it is logical that the decision for biting or swallowing must indeed happen at a “lower” level. In fact, Nargeot et al. showed in 2009 that isolated buccal ganglia (CBI 2 and 3 are located in the cerebral ganglia, and are thereby removed from the isolated buccal ganglia preparation) are capable of operant conditioning and, therefore, of learning. A system that seems to rest in a state of anticipation during feeding behaviors and has
been shown responsive to conditioning raises the question of exactly how much of a slug’s “thought processes” could be contained within recordings of the feeding apparatus.

One set of (bilateral) interneurons that have been identified as “a decision point” of whether or not to feed are B31 and B32 (Susswein and Chiel, 2012). These interneurons reside within the buccal ganglia of *Aplysia* and are a critical portion of the feeding CPG as well as a direct synapse onto the I2 muscle; B31 and B32 are a decision point because of this duality. As an integral portion of the feeding CPG, activity of B31 and B32 indicates the animal’s intent to engage in a feeding behavior and the fact that B31 and B32 synapse directly onto the I2 muscle means that it is quite possible that B31/B32 activity is the first event to appear on I2 EMGs, prior even to behavior induction.

The possibility that an unknown level of previously ignored information is contained within extensively studied EMG and ENG recordings motivates the need for more advanced techniques of analysis. Given the ability of Long Short-Term Memory networks to learn time dependencies amongst data, those networks appear to be a natural choice for examination of this dataset. Use of LSTMs, though, introduces a rather interesting paradox. Advanced neural network architectures such as LSTMs have fairly concretely demonstrated their ability to excel in signal recognition tasks (Chojnowski et al., 2012; Yang et al., 2012; Schmidhuber 2013; Adjemov et al., 2015). However, as of yet, their effectiveness largely remains a black box. In other words, their ability to discriminate feeding patterns suggests hidden information but does very little to identify what that information might be.
Brief History of Artificial Neural Networks

Artificial Neural Networks have existed at least since the 1940’s, and certain parties argue that they date as far back as the 1800’s and the development of linear regression algorithms (Schmidhuber 2015). Schmidhuber’s 2015 review of the history of artificial neural networks is extremely detailed. Instead of repeating the information contained in his review, I have taken selections that illustrate some of the major advancements and achievements in neural networks and arranged those selections in a timeline form, which is available on the next page. Of direct relevance to this work are the Perceptron (a binary classifier), introduced by Rosenblatt in 1957 and the original Long Short-Term Memory network (specialized for time series analysis) introduced in 1997 by Hochreiter and Schmidhuber (Figure 4).

Rosenblatt’s Perceptron was the first artificial neural network to generate national attention as his public statements regarding its potential were greeted simultaneously with fear and disbelief (particularly his contention that it would lead to the first autonomous robot AI). In reality, though, using current terminology, the Perceptron is a linear classifier with no ability to learn context. It is defined as a single vector of weights mapping an input to a binary output (for a complete description of how weights are used in neural networks, see the worked example of backpropagation in the section on backpropagation). Formally, the Perceptron algorithm can be defined as follows: $f(\sum \text{input} \ast \text{weight})$ for $f(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{all other cases} \end{cases}$ where input $\ast$ weight is the dot product of the input (matrix) with the weight (matrix). It is possible to generalize the Perceptron to multiclass classification, but this generalization requires a change in $f(x)$. In this instance, the single output node shown in Figure 3 must be changed such that
there is one output node for each desired class. This change is necessary as $f(x)$ is being changed to $g(x)$, what is commonly referred to as the “argmax” function. The argmax function takes the vector of all output nodes and selects the node with the highest value, which in this setting is equivalent to selecting the class associated with that node.

At their core, all artificial neural networks are predicated on the idea that a matrix of weights can be obtained through back propagation of some form of optimization (i.e.: gradient descent; Ruder, 2016 provides a review of many optimization algorithms) that, when matrix multiplied with a vector of incoming data, produce a desired output. That is to say, mathematically, any neural network that consists of at least one hidden layer and a non-linear activation function can be shown to be Turing complete, or, stated another way, as a universal function approximator (Siegelmann and Sontag, 1995). Before continuing, it is important to stress that though this is formally mathematically true, in practice for many problems it can be extremely (potentially infinitely) difficult to actually reach the proper N-dimensional vector of weights to construct the appropriate transformation.

![Figure 3 - Schematic representation of a Perceptron. Image from commons.wikimedia.org](commons.wikimedia.org)
1943 - McCulloch and Pitts publish one of the earliest NN architectures - however it was incapable of learning and more akin to linear regression.

1949 - Hebb publishes precursors to what today is considered unsupervised learning

1957 - Rosenblatt creates the Perceptron algorithm, stirs controversy by contending it will become the first self-aware robot/Al

1961 - Early work done on Perceptrons with a single hidden layer - termed Multilayer Perceptron, goal is to enable the learning of non-linearities

1961 - Bryson publishes work on the steepest descent within a weight space, a la gradient descent, appears, earliest precursor to current backpropagation methods

1962 - Dreyfus publishes a backpropagation derivation using only the chain rule

1965-1971 Ivakhenko publishes many times on Group Method of Data Handling, possibly the first attempt at a Feed Forward Multilayer Perceptron; employs problem-based architecture decisions and his 1971 publication used 8 layers, firmly placing it as an early example of "deep learning"

1970 - Linmainmaa publishes what seems to be the first formal, explicit error backpropagation in actual neural network style systems in his master's thesis

1973 - Dreyfus uses efficient backpropagation to minimize cost functions by optimizing control parameters

1979 - Fukushima publishes work on the Neocognitron - the first example of "deep" learning other than Ivakenko's GDMH networks. He uses inspirations from the cat's visual cortex to introduce the idea of convolutional networks and subsampling.

1981 - Werbos publishes a use of backpropagation to optimize a neural network

1982 - Hopfield publishes his 'Hopfield Net', one of the earliest examples of recurrent neural networks

1980s (mid to late) - Backpropagation quickly falls out of favor. Although the theory indicates it should greatly improve error propagation in arbitrarily deep networks, it seems to only work with single layer networks.

1987 - Rumelhart and McClelland publish Parallel Distributed Processing: Explorations in the Microstructure of Cognition which explored the human brain's unrivaled ability to process data in parallel and the advantage that parallelism imparts over computer based processing schemes.

1991 - Hochreiter publishes a thesis with the first formal findings explaining why backpropagation was failing. He had identified the exploding/vanishing gradient problem associated with arbitrary numbers of multiplications through application of the chain rule.

1997 - Hochreiter and Schmidhuber publish their formulation of the Long Short-Term Memory network. The current LSTM is based off of this architecture with a few alterations.

2009 - With rapidly advancing graphics processing technology allowing neural networks to perform deep calculations quickly, they win their first international pattern recognition competition and new ANN based architectures continue to be state of the art.

2012 - Hinton achieves historical image recognition results with a deep convolutional neural network. This achievement is commonly credited with sparking the current artificial neural network interest.

Figure 4 - Timeline snapshot of some of the more influential advances in Artificial Neural Networks over the past eighty years, based on the review of Schmidhuber, 2015.
**Standard Feed-Forward Neural Network**

Figure 5 shows a standard feed forward neural network with one hidden layer. Each input unit ($X_i$) is connected to each hidden unit ($a_i$) by a weight vector.

During training, this network would be presented with a 1 x 3 feature vector (numeric representation of some aspect of a dataset), with each input unit X receiving a value for one feature. If the weight vectors were to be stacked into a weight matrix, in this example it would be a 3 x 3 matrix, with each row corresponding to all of the weights for a particular input node (i.e.: row one would be all of the weights for $X_1$). Similarly, there is a weight matrix which connects the hidden layer to the output layer. Each node in the hidden layer is defined as: 

$$a_i = f\left(\sum_{j=1}^{3} \sum_{i=1}^{3} X_j w_{ji}^{2.1}\right).$$

Here, "f" can be one of many non-linear functions with domains of -1 to 1 or 0 to 1. The most common non-linear function is the sigmoid, or, logistic, function defined as: 

$$\frac{1}{1 + e^{-x}}$$

which also is the function used within this work. “X” represents the input nodes, and “$w_{ji}^{2.1}$” represents a value within the 3 x 3 weight matrix that maps values onto layer 2 from layer 1 with “j” and “i” interpreted as the standard “row, column” coordinate system. Similarly, the output layer is defined as 

$$O = f\left(\sum_{j=1}^{3} a_j w_j^{3.2}\right).$$

However, as there is only a single vector of weights, there is only one positional argument for the weight matrix. In many standard neural networks, there are more than one output nodes and, in the case of the network being used for classification, most often the softmax function (definition below) is used and the output node with the largest
numeric value is chosen. Because of the definition of the softmax function, selection of the largest value is equivalent to selection of the output with the highest probability of being true. Formally, the definition of the softmax function is:

\[ \frac{e^{a_{T_w^k}}}{\sum_k e^{a_{T_w^k}}} \text{ for } k = 1, 2, \ldots, i \]

In practice, the softmax function takes a vector of arbitrary length and maps its values to a range of 0 to 1 and scales the values such that they add to 1. Interpretation of this function as a probability distribution arises from the link of its derivation to classic binary logistic regression and the assumption that logistic regression variables are Bernoulli distributed. Extending Bernoulli distributions to multivariate instances gives the Categorical distribution, from which the softmax function is derived. As a result of this formulation, the softmax function is extremely attractive for classification problems and often is used as a way to interpret the output of more complex networks.

**Long Short-Term Memory Networks**

As previously mentioned, a large portion of the work contained herein depends on the Long Short-Term Memory architecture developed by Hochreiter and Schmidhuber. Two key aspects distinguish LSTM networks from standard feed-forward networks: 1) the idea of a cell memory state, and 2) a recurrent connection permitting input to interact with the memory of prior network activity. Figure 6, below, demonstrates the manner in which an LSTM network would process the first half second of a recording for the I2 channel in Figure 1 if that half second were subdivided into segments of 0.125ms or, equivalently, 1/8 of a second. The signal displayed in Figure 6 is the average of the first half second for all bites (left) and all swallows (right) analyzed in this work. Schematically, Figure 6 demonstrates the manner in which “c” (cell
memory) and “h” (hidden state) are passed from time step to time step and updated as new information is gained. It can be seen that information is passed from “h” to “c” after being operated on by functions symbolized as “f”, “i”, and “o,” which stand for “forget”, “input”, and “output”, respectively. Given that these functions control the flow of information from input to memory, the functions sometimes are referred to as gating information, or more simply as gates, which is how they will be referenced from here on. The equations for these gates and the cell memory and hidden state are as follows (shown for a vectorized implementation, i.e.: more than neuron):

\[
\begin{align*}
\tilde{f}_t &= \sigma(W_f \tilde{x}_t + U_f \tilde{h}_{t-1} + b_f), & \text{Eq. 1} \\
\tilde{i}_t &= \sigma(W_i \tilde{x}_t + U_i \tilde{h}_{t-1} + b_i), & \text{Eq. 2} \\
\tilde{o}_t &= \sigma(W_o \tilde{x}_t + U_o \tilde{h}_{t-1} + b_o), & \text{Eq. 3} \\
\tilde{c}_t &= \tilde{f}_t \circ \tilde{c}_{t-1} + \tilde{i}_t \circ \tanh(W_c \tilde{x}_t + U_c \tilde{h}_{t-1} + b_c), & \text{Eq. 4} \\
\tilde{h}_t &= \tilde{o}_t \circ \tanh(\tilde{c}_t). & \text{Eq. 5}
\end{align*}
\]

**Table 1 - Notation, parameters and parameter description to accompany the LSTM defining equations.**

<table>
<thead>
<tr>
<th>Notation</th>
<th>Parameters</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sigma) - the sigmoid function, (\frac{1}{1 + e^{-x}}), applied elementwise</td>
<td>(W_{f,i,o,c})</td>
<td>Weight matrices that interact with incoming data; typically randomly initiated from a normal distribution</td>
</tr>
<tr>
<td>+ - simple elementwise addition</td>
<td>(U_{f,i,o,c})</td>
<td>Weight matrices that interact with the previous time step’s hidden state; typically randomly initiated from a normal distribution</td>
</tr>
<tr>
<td>(\circ) - elementwise multiplication</td>
<td>(b_{f,i,o,c})</td>
<td>Bias term that is added to allow the network to learn offsets for certain time steps; single scalar values, typically initiated to a value of 1</td>
</tr>
<tr>
<td>(W_f \tilde{x}_t) – implied matrix multiplication</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
At face value, even with equations and images, it can be difficult to envisage what exactly is happening within an LSTM and, therefore, why LSTMs seem to work so well. Let us step through what is occurring in Figure 6 from the vantage point of the averaged bite and an LSTM with a single unit. We will consider time step 0 to be that of parameter initialization, which is an obligate precursor to working with a neural network. So, at time step 0 we weight matrices $W_{f,i,o,c}$ and $U_{f,i,o,c}$ are initialized from a normal distribution with mean = 0 and standard deviation = 1. Sampling from a statistical distribution as opposed to initializing all
values to a constant number is extremely important. During training, the presence of symmetric weights can be extremely detrimental and often-times lead to gradients of 0 and the inability of a network to learn. For similar reasons, bias terms \((b_{f,i,o,c})\) are added to help networks learn offsets for each time step and avoid becoming stuck in non-minima valleys within their search space. As they do not directly interact with data and are erased during derivative calculations, bias terms are usually initialized to 1 so that there is a pre-defined number for a gradient to work (as opposed to 0), but their initial value is unimportant. Finally, during time step 0, the cell and hidden state vectors are initialized to all 0’s so that new writes to them do not encounter interference with random initialization noise.

As the current discussion concerns initialization of a network, it is worth taking a moment to address what is meant by “units” or “neurons” within an LSTM. Figure 4 helps to illustrate that in feed-forward networks, a unit or neuron is a relatively straight-forward concept. Each circle is a unit and each unit is associated with a single value in a matrix. In an LSTM, the entire top portion of Figure 6, all three gates, the cell state and the hidden state make up a single unit. While such construction enables far richer dynamics, especially with the built-in non-linearities, in principal the only thing that has changed is that a single value in a single matrix has transitioned into a single value in four matrices and two single value vectors (the cell state and the hidden state). If it is desirable to add \(n + 1\) more neurons, the matrices and vectors simply have \(n + 1\) columns added to them.

One final note on terminology before continuing with the inner workings of the LSTM: when “time-step” is used, it is not in the traditional sense that one with a scientific or mathematical background might infer. In the case of the example data used here, the averaged
bite is from a 5 kilohertz (Khz) recording that was subsequently downsampled to 1Khz. This operation would seem to imply that one “time-step” is equivalent to 1 millisecond (ms). However, the time consumed in training an LSTM on long time-series datasets for a single value at a time is extremely high. To evade this obstruction, within the world of LSTMs “one time-step” is redefined as one step into a dataset; instead of the example using 500 time-steps with one dimensional input, we instead use four time-steps with 125 dimensional input.

At the start of step 1, a 1 x 125 vector ($\mathbf{x}_t$) is presented to the network and immediately concatenates to or, equivalently, passes in tandem with, the network’s hidden state from the previous time-step, $h_0$, which, in this case, was 0. This concatenation is depicted in the lower left of the schematic diagram at the top of Figure 6. Subsequent to the concatenation, the result is immediately passed to four upcoming operations: the forget gate (equation 1), the input gate (equation 2), the output gate (equation 3) and the hyperbolic tangent portion of the cell state definition (equation 4). Once these four operations are completed, updates to the cell state can occur. Note that from the definition of equation 4, the output of the forget gate for time step 1 is completely nullified by the zero value initialization of the cell state. This can be reasoned in the sense that, the forget gate output, which is bound between 0 and 1, tells the network how strongly to weight prior experience. In this instance, though, there is no prior experience, so its output has no meaning. Once the cell state is defined, the new hidden state, which doubles as the network’s output for any given time step, can be calculated according to equation 5. Now, all that is left is to pass the cell state and hidden state for time step 1, $c_1$ and $h_1$, respectively, to the next time step and repeat the process.
Highlighting a few aspects of this process is important as the workflow of an LSTM is simultaneously hard to follow and deceptively simple. The cell state update and hidden state concatenation are the two truly masterful aspects of the LSTM. Concatenating the hidden state ensures that what the network “thought” during the last time step is propagated forward as the network moves through a dataset and the fact that the cell state is both impacted by the previous “thoughts” and impacting the next “thought” means that both short term dependencies (hidden state) and long term dependencies (cell state) can be learned. As mentioned previously, it is common practice to use a single-layer feed-forward network to map LSTM output (hidden states) into a readily interpretable probability space using the softmax function for classification purposes.

In the supervised learning context such as the one used in this work, the output of the softmax function is compared to a training label and the network is penalized using a cost function for incorrect answers. Gradient descent and backpropagation, then, are used to minimize the cost function in an effort to tune the weights for each step and operation within the LSTM and softmax layer so that forward inference (the process walked through in the previous paragraph) yields “correct” results (i.e., results that are the same as the training labels). Cost functions are the tool that allows a network to quantify its performance and thereby have something to improve upon. For the work done in this research, the cost function used is known as cross-entropy.
Entropy

Claude Shannon introduced the idea of informational entropy in his 1948 paper *A Mathematical Theory of Communication*, where he refers to the ability of a communication system to encode messages using bits of information (where bit is defined as the classic binary 1 or 0). If one knows the actual distribution of the information within a message, one can compute the optimal number of bits to encode that message *on average*. This optimal number is given by the formula \( \log \frac{1}{y_i} \) where \( y_i \) is the probability associated with the “i’th” piece of information in a chosen embedding\(^1\). An entire system to be encoded, then, has an optimal entropy described by the sum of the entropy associated with each piece of information it contains. Explicitly, this is given by the formula \( -\sum_i y_i \log y_i \). It follows, then, that cross-entropy is the number of bits needed to encode a message using a non-optimal encoding. Cross-entropy, then, is given by the sum \( -\sum_i y_i \log \hat{y}_i \) where \( \hat{y}_i \) is the incorrect or non-optimal encoding. Given that softmax returns a probability distribution for labeled classes and the correct label distribution is known by the training algorithm, this is a very straight-forward application for cross-entropy. It can be observed, then, that in the case of perfect training, a network’s cross-entropy would approach its true entropy.

\(^1\) Here, embedding refers to using a vector space to represent an idea or symbol. For example, if one needed to create a numeric representation of the letters “A”, “B”, and “C”, one method would be “embedding” them in a three dimensional vector space with the following mapping: 

\[
A \rightarrow 1, 0, 0, \quad B \rightarrow 0, 1, 0, \quad C \rightarrow 0, 0, 1
\]

This method is known as “One-Hot Encoding” as the embedding vectors contain a single “1” with all other entries being 0. It is also worth noting here that embeddings such as these should not be considered simply as conventions to mediate neural network learning. It’s long been accepted that memory makes use of covalent and transcriptional encoding methods to mediate learning in animals (Kandel, *Principles of Neural Science 5th Edition*).
While not of immediate importance to this work, a concept related to the notion of minimization of cross-entropy and approaching the true entropy of a system is the KL (Kullback-Leibler) divergence (Kullback and Leibler, 1951). The KL divergence, sometimes referred to as relative entropy, is the difference between the cross-entropy and the entropy of a system. Intuitively, using the terminology from the discussion on entropy, the divergence can be thought of as the number of extra bits required to encode information when using a non-optimal set of embeddings. It is rather straightforward to see that a minimization of the cross-entropy is equivalent to a minimization of the KL divergence: \[
\lim_{\tilde{y} \to y} \text{cross entropy} \to \text{entropy} \Rightarrow \quad KL \text{ divergence } (\text{cross entropy} - \text{entropy}) \to 0.
\]

KL divergence becomes very important if one wishes to introduce a more probability-based view of network weights, a la Bayesian inference, as cross entropy and entropy can be viewed as the prior distribution and posterior distribution respectively (Burnham and Anderson, 2002).

**Backpropagation**

Backpropagation allows the knowledge gained by cross-entropy to influence the long-term behavior of the network. Conceptually, backpropagation of error had existed prior to Stuart Dreyfus’ 1962 paper; however, his paper was the first to provide concise definitions by way of the chain rule, and is the formulation most widely used today. At an intuitive level, backpropagation allows gradients to be calculated based on the cost function and for these gradients to flow backwards through a network’s dependency chain (the arrows in Figure 3 with the reverse direction) to nudge all weights and biases towards optimal values.
A complete discussion of backpropagation would require a paper unto itself; however, a practical example using the simple network introduced in Figure 5 will be worked through to help solidify the intuition behind how this method works. In order to have numbers to work from, the network is shown again, Figure 7 below, with weights arbitrarily initialized as multiples of 0.05. The problem that this network will be trained on is, given inputs of 3, 2 and 1, produce an output of 0. Prior to working through backpropagation, though, the forward pass must first be computed. Recalling the definition for the hidden layer nodes from earlier, we have: $a_i = \sigma(\sum_{j=1}^{3} X_j w_{j,i}^2)$. So,

<table>
<thead>
<tr>
<th>$X_i$</th>
<th>$w_{1,i}$</th>
<th>$w_{2,i}$</th>
<th>$w_{3,i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.10</td>
<td>0.15</td>
<td>0.20</td>
</tr>
<tr>
<td>0.10</td>
<td>0.20</td>
<td>0.25</td>
<td>0.30</td>
</tr>
<tr>
<td>0.15</td>
<td>0.25</td>
<td>0.35</td>
<td>0.40</td>
</tr>
<tr>
<td>0.20</td>
<td>0.30</td>
<td>0.40</td>
<td>0.45</td>
</tr>
</tbody>
</table>

$net_{a_1} = 3 \times 0.05 + 2 \times 0.2 + 1 \times 0.35 = 0.9$,

$a_1 = \frac{1}{1 + e^{-0.9}} = 0.71094,$

$net_{a_2} = 3 \times 0.1 + 2 \times 0.25 + 1 \times 0.4 = 1.2$,

$a_2 = \frac{1}{1 + e^{-1.8}} = 0.76852,$

$net_{a_3} = 3 \times 0.15 + 2 \times 0.3 + 1 \times 0.45 = 1.5$,

$a_3 = \frac{1}{1 + e^{-2.1}} = 0.81757.$

Now that the hidden layer activations have been calculated, the final output can be calculated as well. $net_O = 0.71094 \times 0.5 + 0.76852 \times 0.55 + 0.81757 \times 0.6 = 1.2687$ and finally, $O = \frac{1}{1 + e^{-1.2687}} = 0.78052.$

Here, for simplicity, we will be using the squared error, or: $Error = \frac{1}{2} \sum (target - prediction)^2$; however, in this case, as there is only a single prediction/output, $O$, the
summation is moot. Therefore, the calculated error is: 

$$\text{Error} = \frac{1}{2} (0 - 0.78052)^2 = 0.30439$$

and with that the forward pass is complete. Now backpropagation, whose goal is to adjust every weight in the network such that upon the next forward pass the overall error of the network decreases, is initiated. As mentioned previously, this task is accomplished through use of the chain rule and computation of the gradient of the error function with respect to each weight in the network. As a reminder, here are the weight matrices: 

$$\begin{bmatrix}
.05 & .1 & .15 \\
.2 & .25 & .3 \\
.35 & .4 & .45
\end{bmatrix}$$

and 

$$\begin{bmatrix}
.5 & .55 & .6
\end{bmatrix}$$

For notations sake, I will refer to them in the following manner:

$$\begin{bmatrix}
W_1 & W_2 & W_3 \\
W_4 & W_5 & W_6 \\
W_7 & W_8 & W_9
\end{bmatrix} \text{ and } \begin{bmatrix}
W_{10} & W_{11} & W_{12}
\end{bmatrix}$$

As a first step, we apply the chain rule to the weights closest to the output node, which would be $$[W_{10} \ W_{11} \ W_{12}]$$, starting with $$W_{10}$$,

$$\frac{\partial \text{error}}{\partial W_{10}} = \frac{\partial \text{error}}{\partial \text{prediction}} \ast \frac{\partial \text{prediction}}{\partial \text{net}_O} \ast \frac{\partial \text{net}_O}{\partial W_{10}}$$

Now we need to figure out exactly how much the error changes with respect to the prediction:

$$\frac{\partial \text{error}}{\partial \text{prediction}} = -1 \ast 2 \ast \frac{1}{2} (\text{target} - \text{prediction})^1 = -(\text{target} - \text{prediction}) = 0.78052.$$

Next, we need to know how much the prediction changes with respect to the output layer\(^2\):

$$\frac{\partial \text{prediction}}{\partial \text{net}_O} = 0.78052 \ast (1 - 0.78052) = 0.17131.$$ Finally, we must calculate how much the output layer changes due to $$W_{10}$$, 

$$\text{net}_O = a_1 \ast W_{10} + a_2 \ast W_{11} + a_3 \ast W_{12}$$

so $$\frac{\partial \text{net}_O}{\partial W_{10}} = a_1 = 0.71094$$. Plugging in values for all of the partials, we get:

$$\frac{\partial \text{error}}{\partial W_{10}} = 0.78052 \ast 0.17131 \ast 0.71094 = 0.09506.$$ Application of this number to a network is accomplished by simply

\(^{\text{2}}\) See the appendix for the worked derivative of the sigmoid function. It ends up being: $$\text{sigmoid} \ast (1 - \text{sigmoid})$$
subtracting it from the weight in question, $W_{10}$. In practice, most networks are implemented
with a learning rate, which is a number that scales the gradient so that it makes the network
less likely to jump over minima. To see it in action, we will use a learning rate of 0.3. So, after
one update from backpropagation, $W_{10}$ becomes $0.5 - 0.3 \times 0.09506 = 0.47148$. When we
calculate the rest of the updates, we use the non-updated values for everything and only
substitute the new values after the entire backwards pass is complete.

Given the symmetric nature of weights 11 and 12 with respect to 10, we can easily
calculate the updates:

$$
\frac{\text{error}}{\partial W_{11}} = -(\text{target} - \text{prediction}) \times \text{prediction} \times (1 - \text{prediction}) \times \frac{\partial \text{error}}{\partial W_{11}} = (\text{target} - \text{prediction}) \times \text{prediction} \times (1 - \text{prediction}) \times a_3 = 0.78052 \times 0.17131 \times 0.81757 = 0.10932.
$$

Identifying the errors for the weights that feed into the hidden layer is much the same; one simply traces
dependencies back to the equation of which the weight is a part. So, for $W_1$, the symbolic
equation would be:

$$
\frac{\partial e}{\partial W_1} = \frac{\partial e}{\partial \text{prediction}} \times \frac{\partial \text{prediction}}{\partial \text{net}_O} \times \frac{\partial \text{net}_O}{\partial a_1} \times \frac{\partial a_1}{\partial \text{net}_{a1}} \times \frac{\partial \text{net}_{a1}}{\partial W_1}.
$$

Notice that we already calculated the first two values, so we only need to find the last three partials. Recall
that in symbolic form, $\text{net}_O = a_1 \times W_{10} + a_2 \times W_{11} + a_3 \times W_{12}$, through which by inspection
one can see that $\frac{\partial \text{net}_O}{\partial a_1}$ is simply $W_{10}$ which equals 0.5. Again, recalling that the derivative of the
sigmoid function is simply $\text{sigmoid} \times (1 - \text{sigmoid})$, we see that $\frac{\partial a_1}{\partial \text{net}_{a1}} = a_1 \times (1 - a_1) =
0.71094 \times (1 - 0.71094) = 0.20550$. Finally, $\text{net}_{a1} = X_1 W_1 + X_2 W_4 + X_3 W_7$ so $\frac{\partial \text{net}_{a1}}{\partial W_1} =
X_1 = 3. So: \frac{\partial e}{\partial W_1} = \frac{\partial e}{\partial \text{prediction}} \times \frac{\partial \text{prediction}}{\partial \text{net}_O} \times \frac{\partial \text{net}_O}{\partial a_1} \times \frac{\partial a_1}{\partial \text{net}_{a1}} \times \frac{\partial \text{net}_{a1}}{\partial W_1} = -(\text{target} - \text{prediction}) \times \text{prediction} \times (1 - \text{prediction}) \times W_{10} \times a_1 \times (1 - a_1) \times X_1 = 0.78052 \times \ldots
$$
\[ 0.17131 \times 0.5 \times 0.20550 \times 3 = 0.04121 \]. Again, we can exploit symmetry for \( W_4 \) and \( W_7 \) as they are part of the same sub-net, all we need to do is switch to \( X_2 \) and \( X_3 \) respectively. This process yields: \( \frac{\partial \text{error}}{\partial W_4} = 0.02748 \) and \( \frac{\partial \text{error}}{\partial W_7} = 0.01374 \).

Most of the work is already done for the rest of the weights as well. For \( W_2 \) we have

\[
\frac{\partial \text{error}}{\partial W_2} = \frac{\partial \text{error}}{\partial \text{prediction}} \cdot \frac{\partial \text{prediction}}{\partial \text{net}_0} \cdot \frac{\partial \text{net}_0}{\partial a_2} \cdot \frac{\partial a_2}{\partial \text{net}_{a_2}} \cdot \frac{\partial \text{net}_{a_2}}{\partial W_2}
\]

and we simply swap \( a_2 \) for \( a_1 \) and \( W_{11} \) for \( W_{10} \) in our expression for \( \frac{\partial \text{error}}{\partial W_1} = -(\text{target} - \text{prediction}) \cdot \text{prediction} \cdot (1 - \text{prediction}) \cdot W_{10} \cdot a_1 \cdot (1 - a_1) \cdot X_1 \). Doing so, we arrive at \( \frac{\partial \text{error}}{\partial W_2} = 0.78052 \times 0.17131 \times 0.55 \times 0.17790 \times 3 = 0.03924 \). Using symmetry again, we simply need to replace the \( X \) values to obtain the updates for weights 5 and 8. This process yields: \( \frac{\partial \text{error}}{\partial W_5} = 0.02616 \) and \( \frac{\partial \text{error}}{\partial W_8} = 0.01308 \).

For our last three weights, again we simply need to replace \( a_2 \) with \( a_3 \) and \( W_{11} \) with \( W_{12} \). So, \( \frac{\partial \text{error}}{\partial W_3} = -(\text{target} - \text{prediction}) \cdot \text{prediction} \cdot (1 - \text{prediction}) \cdot W_{12} \cdot a_3 \cdot (1 - a_3) \cdot X_1 = 0.78052 \times 0.17131 \times 0.6 \times 0.14915 \times 3 = 0.03590 \). Again swapping \( X \) values to obtain the updates for weights 6 and 9 we get: \( \frac{\partial \text{error}}{\partial W_6} = 0.02393 \) and \( \frac{\partial \text{error}}{\partial W_9} = 0.01197 \).

Doing a mass update, our new weight matrices are:

\[
\begin{bmatrix}
0.05 & 0.1 & 0.15 \\
0.35 & 0.4 & 0.45
\end{bmatrix}
\begin{bmatrix}
0.04121 & 0.03924 & 0.03590 \\
0.02748 & 0.02616 & 0.02393 \\
0.01374 & 0.01308 & 0.01197
\end{bmatrix}
= 
\begin{bmatrix}
0.03764 & 0.08823 & 0.13923 \\
0.19176 & 0.24215 & 0.29282 \\
0.34588 & 0.39608 & 0.44641
\end{bmatrix}
\begin{bmatrix}
0.5 \\
0.55 \\
0.6
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.09506 \\
0.10276 \\
0.10932
\end{bmatrix}
= 
\begin{bmatrix}
0.47148 \\
0.51917 \\
0.56720
\end{bmatrix}
\]

Now, with the completely updated weights, we can perform one
more forward pass to verify that the error did indeed get smaller: $net_{a_1} = 3 \times .03764 + 2 \times .19176 + 1 \times .34588 = 0.84232$, $net_{a_2} = 3 \times 0.08823 + 2 \times 0.24215 + 1 \times 0.39608 = 1.1451$, and $net_{a_3} = 3 \times 0.13923 + 2 \times 0.29282 + 1 \times 0.44641 = 1.4497$. The hidden layer activations become: $\sigma \begin{bmatrix} 0.84232 \\ 1.1451 \\ 1.4497 \end{bmatrix} = \begin{bmatrix} 0.69895 \\ 0.75861 \\ 0.81000 \end{bmatrix}$, which leads to $net_o = 0.69895 \times 0.47148 + 0.75861 \times 0.51917 + 0.81000 \times 0.5672 = 1.1828$ and finally $\sigma(1.3559) = 0.76545$ which is clearly closer to zero than 0.78052.

After observation of one application of backpropagation in a very small and simple network, it begins to become clear why neural networks, especially the more advanced ones, did not become tractable until the more recent advances in parallel computing. It feels very natural, though, that when combining the ideas of feed-forward neural nets, softmax, LSTMs, cross-entropy and backpropagation, that the united networks could do a very good job of classifying biological time series data or, at the very least, their ability to do this job should be investigated. This work begins to examine the question of whether a formulation of these networks can indeed excel at this task and, if so, the implications that this success might have.
Methods

As already stated, the dataset used for this investigation came from the Cullins et al. 2015a,b publications. In order to arrange the data in a suitable state to present to a neural network, custom Python code was written that organized all four channel bite (599) and swallow (618) recordings into a single dataframe. To avoid the possible interpretation of noise as an actual behavior, white noise was generated from Gaussian distributions with mean and standard deviations tailored to each of the four channels (Figure 8) and then appended to the same dataframe for a total of 1817 patterns (599 bites, 618 swallows, 600 white noise). Each recording was sub-divided into segments of length 125, corresponding to 1/8 of a second, or 125ms. In order to allow a neural network to classify, labels were generated following the one-hot encoding scheme (see footnote on page 24 for an example) for bites, swallows and noise for the first 32 segments of each recording leading to a 96 dimensional label space for each channel. One-hot encoding, depending on the nature of the task and network, can be used to either hand-embed concepts in a vector space or to create a vector space of labels that a network learns to map a dataset to.

Figure 8 – Plots of the average voltages for the I2, RN, BN2 and BN3 channels during bites and swallows (rows 1 and 2) and the simulated noise used for each channel during training (row 3).
Sub-dividing the data in this way accomplished two things: first, it allowed for classification on a small time-scale which enables the investigator to isolate a relatively small interval where the recording became identifiable. Secondly, it lowers the impact of having so few recordings as the long-term dependencies that are learned are fewer in number (dependencies from 32 time steps as opposed to 4000 time steps). In order to avoid the vanishing or exploding gradient problem and to make calculations more tractable, data was processed in batches of 30 patterns at a time.

In order to take advantage of the fact that there is information coming from four separate sources about the same event, each channel had an LSTM and a softmax layer. Formulating the network this way ensured that the context specific to each channel and, therefore, each set of higher order (or, at least, upstream) neurons was (theoretically) accounted for. After each LSTM/softmax layer made a prediction as to what it was seeing, that prediction was then passed to a multilayer feed-forward network in order for an overall classification considering all four channels every 0.125 seconds.

Figure 8 (below) demonstrates the dataflow and overall network architecture. Each LSTM was layered, which can be accomplished in much the same way as layering a feed forward neural network with the output from LSTM layer 1 serving as the input to the next LSTM layer. Two sizes of networks were used, initially a three layer LSTM with 24 hidden units was used. Upon observation of unusual findings for the first half second of recordings, the number of units was dropped to 4, though the layers were kept at 3, and the number of data subsets also was restricted to the first 4 sections of 0.125ms (Figure 6). All neural networks were implemented using Tensorflow’s Python API.
During training, optimization was performed by minimizing a cross-entropy cost function through use of a form of gradient descent called RMSprop (proposed but not published by Geoffrey Hinton, Ruder 2016) with a learning rate of 0.003. In short, RMSprop adaptively changes the rate at which optimization traverses a gradient or search space to minimize the chances of overshooting some minima while accelerating movement away from maxima. The data was partitioned into a training set, validation set and testing set (60%/20%/20%) with the patterns within each set randomized prior to every trial. 

Training was run for 150 epochs with the order of presentation of training data randomized between epochs to avoid

Figure 9 - Schematic representation of the overall logic flow of the network used in this work. Recorded data from each individual channel is passed into their respective LSTM in 125ms segments. The LSTM networks then pass their context-dependent output to a softmax layer to map the output into a pre-defined label space. The decisions from all four softmax layers are then passed to the three-layer feed forward network for a final decision of whether the network is looking at a bite or a swallow using information from all four channels.
learning of any false long-term dependencies. Summary data was collected by use of the Tensorboard portion of the Tensorflow API.

Recording data was aligned such that each recording had half a second of padding prior to behavior initiation (I2 firing frequency > 10 Hz). In order to examine if this alignment had any impact on results, training was performed with 25, 50 and 75ms offsets from time 0, in addition to beginning at time 0 for the first 500ms. Similarly, to examine the ability of networks to discriminate behaviors and ensure that the process is robust to changes in data partitioning, the bin size was varied from 50, 75, 100, 125 and 150ms with a bin size of 125ms being shown in Figure 6 for the first 500ms.
Results

In terms of accuracy for all 32 recording subdivisions, the RN, BN2 and BN3 underlying
LSTMs are each able to achieve an accuracy of approximately 99% on their training and
validation data (n = 100 runs). When used on the test dataset, accuracy is between 95-97%
(Figure 10 shows the output of one run on a testing set). I2’s LSTM accuracy is more variable
with accuracy ranging from 90% to 99% and the average tending towards 93-95%
on training data. When applied to test
datasets, it performs similarly to its
average training performance, usually
giving accuracy around 92-94%, the feed-forward overlay achieved an accuracy > 99%. Figure
11 shows graphs of the network accuracy and loss during the 150 training epochs. While the
accuracy and loss mirror each other, it’s important to remember that accuracy and cross-
entropy are related but not the same. Accuracy relates to the argmax of the softmax function
matching the assigned label while the cross-entropy is an indication of the efficiency of the
network’s representative embedding. All channels showed susceptibility to overtraining if the
number of epochs was increased significantly. The I2 channel showed the most sensitivity to
over-training with error rising to 98% and never improving if the network is run for between
160-200 epochs (the differing sensitivity to overtraining can be accounted for by the random
initialization of all weights). Additionally, when smaller networks were explored for the 32
subset dataset (as opposed to the 4 subset dataset) the results were extremely numerically unstable with accuracy jumping between 20 and 30 percentage points between epochs.

All recordings used contain half a second of what was assumed to be noise prior to the start of the behavior (I2 firing frequency ≥ 10hz, Lu et al., 2013), however, these accuracy levels implied the ability to accurately identify that half second as coming from bites, swallows or noise. Figure 12 shows the result of running the smaller 3 layer (4 units per layer) network on only the first half second of the I2 channel. During training and testing, similar levels of accuracy are seen as are observed on the 32 subset trials. The network’s performance was severely
degraded by shuffling of the labels and the patterns with which they were associated, with training and testing bracketing 50% accuracy, a rate which one might expect if a network were simply “guessing” as to whether data belonged to a bite or a swallow. It’s important to note that one should not expect an accuracy of 0% as the labels were shuffled at random between 599 bites and 618 swallows, so that it is entirely possible (and should be expected) that some bites and some swallows were accurately labeled. With this in mind, the fact that the accuracy bracketed 50% in a binary system is entirely plausible.

Given that the most unexpected results came from the first half-second, that is where the investigation of robustness was performed, again using the smaller network. Table 2 (below) shows the results after n = 50 trials for all cases other than no offset with 125ms bins which has n = 100; all accuracies were rounded to the nearest percentage point. Discrimination
success falls off rapidly beginning at bins of 75ms and continuing to be no better than chance at bins of 50ms. No qualitatively significant change in success is seen for bins of 100, 125 or 150ms nor for start time offsets.

Identifying what the networks are actually learning, though, is much more difficult. Several papers were published in early 2018 detailing work that is being done to try to unravel what exactly is being learned within deep neural networks (Olah et al., 2018; Strobelt et al., 2018). Strobelt et al. propose methodology for analyzing LSTM networks which look extremely promising; however, at this time, those methods have not been implemented on this network. For the network in question, Figure 13 (above) and Figures S1, S2 and S3 (see appendix) show the weights from the three layers of all four LSTMs for the input, output and forget gates. One can see in Figure 13 (y-axis in multiples of 10) that the network weights hone in on certain aspects of the dataset fairly quickly, but others are still being fine-tuned after 120 epochs. The most extreme values for the input and output weights seem to be in layer 3, while the more extreme forget weights appear in layer 2. Guesses can be made as to what this means; however, no concrete information is available at this time.
Figure 13 - Color maps for the 24 units in layers 1, 2 and 3 for the input, forget and output gates within the I2 LSTM. The y-axis is in multiples of 10. Though one can observe that the most extreme values in the input and output gates appear in layer 3 and the most extreme values for the forget gates appear in layer 2, at this time a more intuitive understanding of what they are identifying is not clear.
Discussion

The results obtained from this exploratory work seem to invite two statements. First, they seem to indicate that LSTM’s have strong potential for use to analyze biological signals and possibly to identify information-rich regions within signals that were previously thought to be noise. Specifically, the ability of this network to identify bites and swallows from a period of I2 activity previously thought to simply be noise is an extremely surprising result. There is a possibility that data that was previously written off as noise actually is identifiable sub-threshold signals coming from B31/32 (interneurons that directly synapse onto I2), which, if true, opens up large avenues of investigation that previously were not thought to exist.

Second, as it currently stands, the performance of this network yields no concrete information without a deeper analysis of the exact focus of the weights. Implementations of tools similar to LSTMVis (Strobelt et al., 2018) would help to offer insight into what is occurring. Tracing the portions of the input signals that are affected by the different weights in the vein of the worked example of backpropagation, and actually pairing the weights and outputs with the signals they are multiplying will permit one to piece together what the network deems to be important.

It is possible at this time, however, to make a few generalized observations about the nature of the weights. For instance, the weights for the first and second layers of all four networks take on very few extreme values. This observation meshes well with the idea that in deep network architectures, one of the things accomplished is that the networks learn their own internal representations for objects on which they are trained. As the majority of weights that do not reside in the output layer have intermediate values, no (or very little) information
from the sequences is immediately discarded. Weight values begin to take on extremes only when the network organizes its output.

Table 2 shows an interesting result in that dropping the bin size below 100ms appears to degrade the network’s ability to discriminate behaviors. Prior to trying to make this observation into a concrete statement, though, the same settings (50ms bin size, arbitrary offset) need to be explored with a larger network. In conjunction with the prior work done on the half-second padding, these analyses were carried out with a 4 units per layer. It is quite possible that should the number of units per layer be increased, the discriminating power would return, and this must be investigated.

An interesting corollary can be made here, as during memory transition in the dentate gyrus, it is thought that representations of objects are ordered then converted into sparse representations prior to being localized to different regions of cortex. One can describe the way the three layer LSTM handles data as analogous to the process of interpreting, ordering and then creating a sparse representation (many layer 3 “forget” weights near 0, meaning they are wiping out a lot of cell memory) before passing the information out to the softmax layer.

This work serves as an exciting first step, but more work is needed to identify what is happening, and in order for it to truly have a significant impact, the idea must be translatable into an online context. Simple translation of the idea into an online context, though, won’t be sufficient as currently we see high levels of accuracy, but we don’t know what confidence level to associate with those predictions. Real time decisions should not be made on a network prediction if the underlying distribution is 51 to 49 in favor of the “right” answer. One potential
way to deal with this question is to apply Bayesian reasoning to the weights in the network and construct prior distributions for all of the weights, allowing for multiple runs of inference on any signal with each run sampling from the weight distributions. This procedure would permit construction of a posterior probability of the prediction correctness, permitting a more reasonable decision to be made as to whether or not a prediction should be heeded.
Appendix

Derivative of the Sigmoid Function – Demonstration that the derivative of the sigmoid (sigmoid) function is sigmoid * (1 – sigmoid)

\[
\frac{d}{dx} \left( \frac{1}{1 + e^{-x}} \right) = \frac{d}{dx} (1 + e^{-x})^{-1} \\
= -1 * (1 + e^{-x})^{-2} * (-e^{-x}) \\
= \frac{e^{-x}}{(1 + e^{-x})^{-2}} \\
= \frac{e^{-x}}{1 + e^{-x}} * \frac{1}{1 + e^{-x}} \\
= \left( \frac{1 + e^{-x} - \frac{1}{1 + e^{-x}}}{1 + e^{-x}} \right) * \frac{1}{1 + e^{-x}} \\
= \frac{1}{1 + e^{-x}} * (1 - \frac{1}{1 + e^{-x}}) \\
= sigmoid * (1 - sigmoid)
\]
Figure S1 - Color maps for the 24 units in layers 1, 2 and 3 for the input, forget and output gates within the RN LSTM. The y-axis is in multiples of 10. Though one can observe that the most extreme values in the output gates appear in layer 3 and the most extreme values for the forget gates appear in layer 2, at this time a more intuitive understanding of what they are identifying is not available.
Figure S2 - Color maps for the 24 units in layers 1, 2 and 3 for the input, forget and output gates within the BN2 LSTM. The y-axis is in multiples of 10. Though one can observe that the most extreme values in the input and output gates appear in layer 3 and the most extreme values for the forget gates appear in layer 2, at this time a more intuitive understanding of what they are identifying is not available.
Figure S3 - Color maps for the 24 units in layers 1, 2 and 3 for the input, forget and output gates within the BN3 LSTM. The y-axis is in multiples of 10. Though one can observe that the most extreme values in the input and output gates appear in layer 3 and the most extreme values for the forget gates appear in layer 2, at this time a more intuitive understanding of what they are identifying is not available.
References


Ruder S. An overview of gradient descent optimization algorithms. *arXiv*.


