## MULTIPLE SCALE MODELING FOR PREDICTIVE MATERIAL DEFORMATION ANALYSIS

**Motivation.** Material deformation and stress-strain is an active area of mathematical modeling relevant to industrial and research-oriented materials science. It is important to take variations in material properties into account in these models. Multi-scale models that incorporate inhomogeneity were studied and modeling frameworks that address this need were created and tested. Incorporating variations in material properties at the micro scale resulted in significantly different predictions of material deformation under similar loading. Variations in material properties were accounted for through averaging over stresses in representative volume elements (RVEs). This project was a collaborative effort by students, **Rachel Aronow**, **Aaron da Silva**, **Rose Dennis**, **Abdel Kader Geraldo**, **Dean Katsaros**, **Melissa Sych**, and **Richard Touret**, under the guidance of Professor **Qian-Yong Chen**.

**Background.** In material science, the deformation of materials under tension is an important problem with widespread application. Using the stress/strain equations as the basis for the model, material properties are encoded, and the model is solved to predict behaviors of real-life materials. The material properties are encoded via the material's Poisson ratio, and its young's modulus. Young's modulus is defined mathematically as the ratio of the tensile stress to the strain. Poisson's ratio is the ratio of the transverse strain to the axial strain. Simply, Young's modulus is the measure of material stiffness, and Poisson's ratio the tendency of the material to expand perpendicular to the direction of compression.

It is important to note that a material may not have the same Poisson ratio or Young's modulus in different parts of the material. We say that this is an inhomogeneous material. Commonly, deformation models assume homogeneity in the material being modeled. The homogenous version of these stress/strain equations is much easier to solve. However, this assumption leaves important characteristics of the material out of the model. This motivates taking a multi-scale approach to modeling the material.

The multi-scale approach incorporates the inhomogeneity of a material via combining a macro and micro-scale set of equations. Both equations assume homogeneity, but the micro-scale equations have different property coefficients corresponding to their location. Representative Volume Elements (RVEs) is the name given to the micro-scale units. This project investigated the differences in modeling results when a multi-scale approach is taken.

**Tissue Mechanical Testing.** To investigate the effects of multiscale modeling, we chose to focus on the problem of a ligament under uniaxial tension testing, as performed by Chokandre et al. in 2015. In particular, ligaments are an inhomogeneous material. The uniaxial tension test performed in this study is described as follows: a 1 mm x 4.46 mm dumbbell-shaped sample of tissue from an MCL (0.53 mm thick) is clamped at both ends and stretched slowly, such that the internal forces in the tissue remain balanced (quasi-static equilibrium). At each time step, the force applied to the tissue and the tissue's resulting displacement is recorded.

To model this experiment, we used the plane stress/strain equations. These equations are given by:

$$\nabla\cdot(\sigma) = F \qquad \sigma = D\epsilon \qquad \epsilon = \nabla\cdot U$$

where $U=[u,v]$ is displacement, $\epsilon=[\epsilon_x,\epsilon_y,\tau_{xy}]$ is strain, $\sigma=[\sigma_x,\sigma_y,\tau_{xy}]$ is stress, $F=[F_x,F_y]$ is the internal body force (set to zero), and $D$ is a coefficient matrix describing the elasticity of the material. $D$ is dependent on $E$ and $\nu$, and is easy to calculate for a homogenous material. These equations are derived from the following assumptions:

1. The problem is two-dimensional
2. The material is homogenous and its elasticity can be described fully by two parameters: Young's Modulus, $E$ and Poisson's Ratio, $\nu$
3. The material is in equilibrium (stresses are balanced throughout the material)

Using MATLAB and the open source toolbox FEATOOL (https://www.featool.com/), we defined a 1 mm x 5 mm rectangular region as our ligament. With the data from Chokandre et al. (2015) in hand, we calculated an average Young's Modulus (slope of the stress vs. strain curve) of the MCL ($E$=36 MPa) and used this to describe the elasticity of our simulated material. For a Poisson's ratio, we adopted a typical value for an MCL ($\nu$=0.02) from Sweigart & Athanasiou (2005). Assuming quasi-static equilibrium, we set the initial displacement of the material to be 0 and applied a forcing of 1.3 N in the positive y-direction to the top edge (Figure 1). To simplify the model, we assumed symmetry and set the bottom edge to be held fixed. These conditions were enforced via boundary conditions. Solving the plane stress equations via the finite element method results in

the displacements displayed in Figure 2. We observe that the top edge of the region is displaced 0.27 mm, compared to a displacement of ≈0.45 mm observed in the data set.

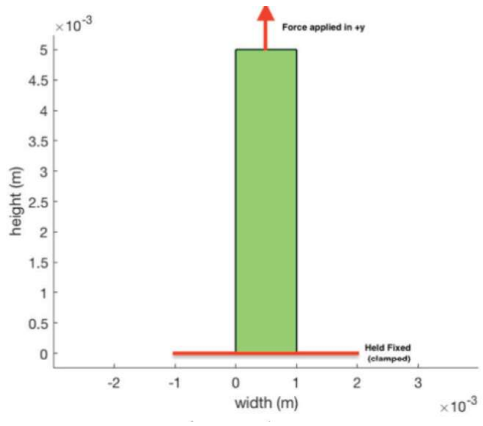


Figure 1.

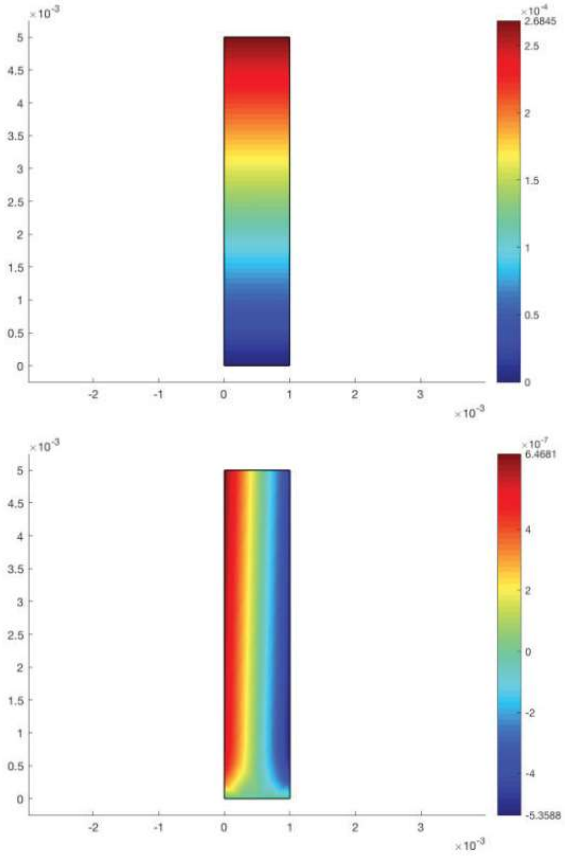

Figure 2. (y displacement left, x displacement right)

**Multiscale Modeling Approach.** In order to solve the force balance equations, we had to make the assumption that the entire ligament is homogenous. However, some of the most exciting potential applications of biomechanics apply to problems involving in-homogenous materials. For example, we consider a ligament with an "injury" point, in order to incorporate a multi-scale approach to the problem.

**One Macro-Region.** For a benchmark, we first solved the problem for the simple homogenous case. As in Section 2, we used the average values of $E$ and $v$ across the whole body. Again, the initial displacement and internal forcing (body force) were set to 0, and a 1 N upwards forcing was applied to the top edge while the bottom edge was held fixed. Then the plane stress/strain equations were solved using the finite element method. These steps were repeated until the material stabilized its shape. The solution was then interpolated for the entirety of the region using MATALB's scatteredInterpolate function. The resulting deformation of the material is shown in Figure 3. Qualitatively, the material behaves as expected under tensile stress.

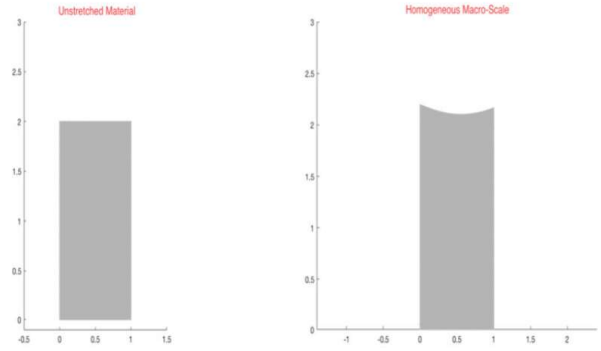


Figure 3.

For our second model, we introduced four RVEs, located at the Gauss points of the rectangular macro-region. The RVE in the top right corner was given a lower Young's Modulus and greater Poisson's ratio than the rest of the material, to indicate a point of poor stability (see insert in Figure 4). We then applied a multi-scale approach based off Barocas (2007), and summarized as follows: We first solved one iteration of the macro-problem as described in Section 3.1. We then solved the plane stress equations in each homogenous RVE for the stress resulting from the external loading. For this step, we used boundary conditions interpolated from the closest macro-region boundary to the RVE. Next, we calculated $Q$, as introduced in Barocas (2007). The $Q$ factor is defined by the volume average of $Q_{RVE(k)}$ where $k$ refers to the indexing of the RVEs and $Q_{RVE(k)} = \frac{1}{Vol_{micro}} \int (stress_{RVE(k)} - AvgStress_{macro}) d\mu(x)$. We then resolved the plane strain equations in the macro-region, but with the internal forcing set to $Q$ instead of 0. This captures the effect of inhomogeneity on the internal forcing of the material, and thus how it responds

to an external forcing. We continued to switch between calculating $Q$ in the micro-scale problem and balancing forces in the macro-scale problem until the solution stabilized. The result of this simulation is shown in Figure 4.

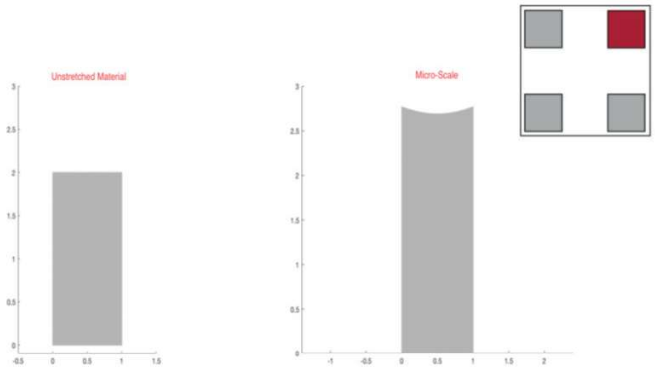


Figure 4.

**Multiple Macro-Units with a Microscale**. To include finer detail in the model, we introduced a grid of four macro-regions, each containing four RVEs at their Gauss points. We therefore have sixteen locations to introduce inhomogeneity. The top and bottom edges of the ligament, where the tissue is anchored to the bone, were set to be slightly less elastic than the overall material, and the injury point was set to be more elastic, representing an instability point in the fiber alignment. Starting with the top left macro-unit, we complete one iteration of the method described in Section 3.2. We then repeated the process for the top right macro-unit, with the additional boundary condition of the stress along the right edge of the top-left macro-unit being equivalent to the stress felt by the left edge of the top-right macro-unit. Continuing clockwise, we then solved the problem for each of the bottom units in a similar manner. This process was then repeated until the solution stabilized.

**Results & Discussion.** A comparison of the three simulations is shown in Figure 5. Incorporating the microscale in the second simulation affects the magnitude of the deformation, but not its shape. This is an issue inherent to our method. The microscale is represented by the $Q$ factor, which behaves like a weighted average of the relative stress felt in each RVE. This term was then used as an internal forcing term, and so a positive $Q$ value forces the body in the same direction as the external force, meaning the same magnitude external force can displace the material further. Therefore, this model does not capture any information about the location of the fine details, just the existence of a more flexible region within the tissue.

The model with multiple macro-units however, does demonstrate a change in both the magnitude and shape of the deformation. The lopsided tissue shows signs of the location of the injury point. Although the effects of the micro-units are still averaged together to find $Q$, there is now a $Q$ factor for each macro-region, therefore holding on to structural details within each macro-unit. Thus, the introduction of multiple macro-units is better at encoding fine details, without losing all the details to averaging.

One of the challenges to multi-scale modeling is the lack of a rigorous benchmarking method. Our results show that incorporating a micro-scale changes the shape of the deformation, but we do not possess the data sets to determine which model is "best."

Furthermore, the way the microscale stresses are incorporated into the macro scale should be examined in future work. The $Q$ factor is a counterintuitive way to do this, as it is an averaging process. The degree to which the microscale is detailed can become computationally impractical very quickly, so some sort of approximation is necessary. However, it is not clear whether there is a better approximation than the $Q$ factor.

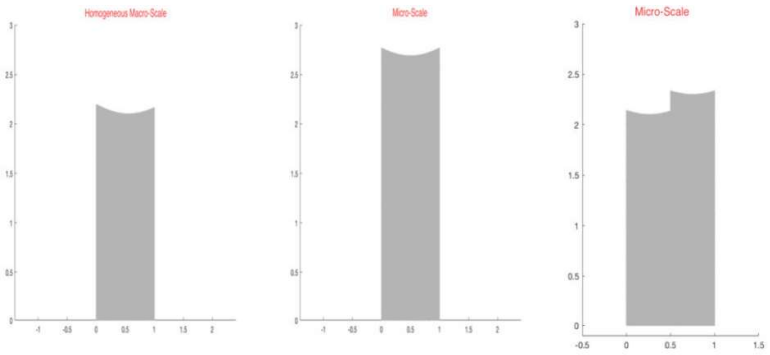


Figure 5.

**Conclusion & Future Work.** Incorporation of the microscale causes different deformation behavior. This reflects the importance of taking into account the heterogeneous nature of many materials. For more comprehensive studies to be productive, the benchmarking of these models needs to be more rigorous. Benchmarking should not necessarily be reliant on imaging, as comparison of real images to experimental images can be ill-defined. The model could be made more robust in future studies by using fiber equations instead of stress-strain modeling. Here, microscale stresses would be calculated at cross-linking sites in the micro-scale, and these stresses would be factored into the plane stress at macro-scale, a computationally expensive approach.

# ELECTROMYOGRAPHY CLASSIFICATION USING RECURRENT SYSTEMS

## I. MOTIVATION

Classifying Electromyography (EMG) data is important for many application domains, such as prosthetics or the remote control of robots, but the current state of the art is stuck in a tug of war between interpretability, accuracy, and generalization to new data sets. This Applied Math Master's Project compared methods typically used to achieve state-of-the-art performance on this EMG classification task with newer methods which might better exploit the time-series nature of the data, possibly at the expense of interpretability. The project was a collaborative effort by students **Connor Amorin**, **Gabriel P. Andrade, Chris Brissette**, **Matthew Gagnon**, **Brandon Iles**, **Jimmy Smith**, **and Lance Wrobel**, under the guidance of Professor **Qian-Yong Chen**.

## II. EMG DATA

EMG is a technique for recording the electrical activity produced by the neurons in skeletal muscles. The signals produced by EMG are collected from electrodes either placed on the surface of the skin or inserted directly into the muscle. Though measurements are less reliable due to noise introduced from a myriad sources, surface EMG data is more readily available since it is far less invasive. For these reasons, in what follows, we will focus on these surface EMG recordings; an example of a raw signal is shown in Figure 1.

EMG data has a number of issues of which we must always be mindful. These factors force us to place significant trust in whomever collects the data, making it very hard to assess how best to work with the data. As was already mentioned, the data is necessarily noisy. Furthermore, the subject's recorded response to stimulus is understood to be encoded in changes of amplitude and also of frequency[1]. When these changes occur, and how they change, however, depends on the stimulus itself,

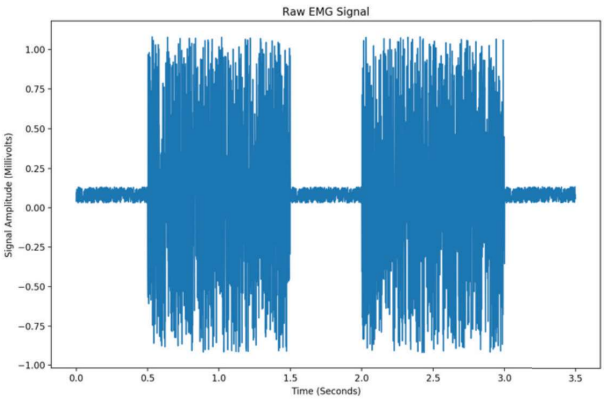


Figure 1: An example of an unfiltered EMG signal.

the "intensity" of the action, the state(s) of the subject prior to taking the action, and so forth. Further complications arise due to measurements being highly dependent on electrode placement and differing stimulus response from individual subjects.

## III. THE DATASETS

Our goal is to compare methods for classifying EMG with accuracy, ease of generalizing to a new data set, and interpretability of the method in mind. Therefore we will be discussing:

1. The University of California Irvine Machine Learning Repository's hand movement data set

2. Dr. Rami Khushaba's (from the University of Technology Sydney) finger movement data set

The first data set (hand movement) contains signals from 5 subjects (3 female, 2 male) performing 30 trials for each of 6 specific hand movements (6-second-long trials). The 6 hand movements used in this experiment can be seen in Figure 2.

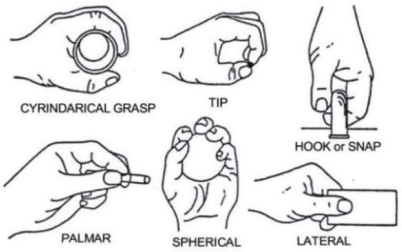


Figure 2: Different hand motions used for classification.

The second data set (finger movement) contains EMG signals from 8 subjects (6 male, 2 female) performing 3 trials for each of 14 specific finger movements (20-second-long trials). The finger movements used in this experiment can be seen in Figure 3.

Both of these datasets are popular benchmarks in the literature and differ quite drastically with regards to data collection (e.g. machines used, sampling rate). By using these data sets in our evaluation of EMG

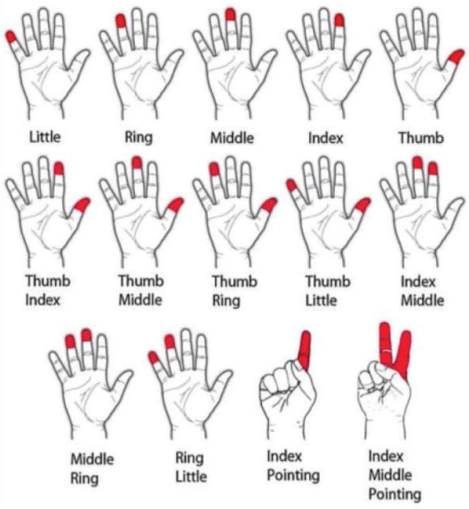


Figure 3: Different finger motions used for classification.

classification techniques, we were able to test the three factors we are interested in simultaneously.

## IV. CLASSICAL MACHINE LEARNING

To strike a balance between accuracy, interpretability, and generalization, a bulk of the literature on EMG classification focuses on properly preparing the data and then using this to train classification models we understand well. Preparation consists of preprocessing and feature extraction which ideally make the information received from our learning algorithm easier to interpret. Unfortunately, this type of approach is fundamentally limited by the method that one chooses for preparation. We review common preprocessing and feature-extraction approaches used in the literature along with several classification techniques. In the interest of space, we do not include everything we did using these methods, but instead describe models which best give the basic idea of the general class of classifiers we experimented with[2].

### 1. Data Processing

Due to noise from electronics and external sources, EMG signals should undergo preprocessing steps to try and strip away "abnormalities". A common first step is to filter the EMG signal with the Butterworth Filter $H = 1/\sqrt{1+\left(\frac{w}{w_c}\right)^{2n}}$ where $w$ represents the angular frequency, $w_c$ is the cutoff frequency represented as an angular value, and $n$ is the number of elements in the filter. Next is rectification of the EMG signal to remove the negative amplitudes, either by half rectification, or by full rectification. This is followed by smoothing of the signal in some way (we used a moving average of subsets of the signal). Finally, we partition this "cleaned" form of the EMG signal into subsets called windows used for feature extraction or classification. This process is demonstrated in Figure 4.

We experimented with multiple window sizes on our data sets and ultimately concluded that, with feature extraction[3], (i) windows of 512*ms* with a 256*ms* overlap between windows worked best on the hand data and (ii) windows of 128*ms* with a 64*ms* overlap between windows worked best on the digit data.

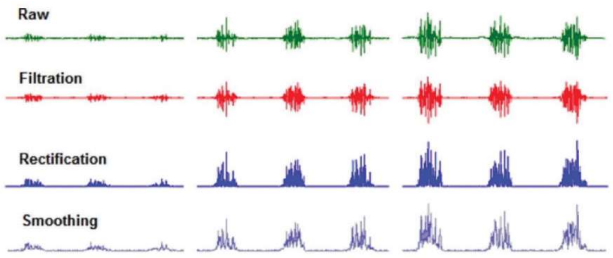


Figure 4: Four preprocessing steps EMG data goes through prior to classification.

| Name | Domain | Formula |
|---|---|---|
| Root Mean Square | Time | $RMS = \sqrt{\frac{\sum_{i-1}^{n} x_i^2}{n}}$ |
| Mean Absolute Value | Time | $MAV = \frac{1}{n}\sum_{i=1}^{n}\lvert x_i\rvert$ |
| AR Coefficients $(\beta_1, \ldots, \beta_p)$ | Frequency | $X_t = c + \sum_{i=1}^{p}\beta_i X_{t-i} + \epsilon_t$ |
| Waveform Length | Time | $WL = \frac{1}{n}\sum_{i=1}^{n-1}\lvert x_{i+1} - x_i\rvert$ |
| Wilson Amplitude | Time | $WAMP = \sum_{i=1}^{n-1} I(\lvert x_{i+1} - x_i\rvert \geq h)$ |
| Sign-Slope Changes | Time | $SSC = \sum_{i=2}^{n-1} I((x_i - x_{i-1}) \cdot (x_i - x_{i+1})) \geq h)$ |
| Zero Crossing | Time | $ZC = \sum_{i=1}^{n} sgn(-x_i \cdot x_{i+1})$ |

Figure 5. Features used for Classification. In this table, $I$ refers to an indicator function: either 1 when its argument is true or 0 otherwise; $h$ is some arbitrary threshold value.

### 2. Feature Extraction

Though preprocessing certainly "cleans" the signal and relieves a large amount of uncertainty surrounding it, we are still left with a long non-stationary time series that may contain inconspicuous redundant information. We use feature extraction methods to attempt uncovering those redundancies in a more succinct form while abstracting away properties that emerge due to time. Most classic machine-learning models are not designed with ordered data in mind, and they struggle with high-dimensional input; feature extraction simultaneously acts to reduce dimensionality of the data while also packaging data in a form more suitable for training.

Using the same windows described above, we calculate features of a preprocessed signal and append the results to get a "feature vector" that describes multiple aspects of the signal. In this project, multiple combinations of time-domain features were used along with autoregressive model coefficients. Figure 6 contains details about the most useful features employed.

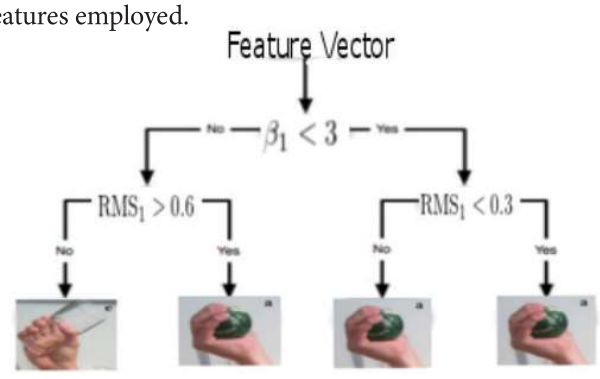


Figure 6. A pictorial representation of a simple tree for hand data classification.

### 3. Support Vector Machine

A popular and well understood classifier used for EMG is the support vector machine (svm). This model attempts to split an $m$-dimensional feature space of training data with ($m$-1)-dimensional hyperplanes. These hyperplanes carve out regions in the feature space that we associate with classes by maximizing the margin between the two closest points on either side of the hyperplane during training[4]. This creates the maximal separation between classes seen in training and allows for the most "flexibility" assuming the test data behave similarly.

More formally, if the hyperplane is defined by

$$(n \cdot x)\text{-}b = 0$$

with $x$ as our training data, then we have the optimization problem of minimizing $\| n \|$ subject to

$$y_i(n \cdot x_i\text{-}b \geq 1), \forall i \in [m]$$

where $y_i \in \{-1, 1\}$ is the class of training sample $x_i$. The $n$ and $b$ solving this problem determines the maximum-margin hyperplane.

### 4. Classification Tree

Another common – and certainly easy to interpret – classifier used in the literature is the classification tree. These models essentially step through each element of the input vector, making a simple decision at each element, and ultimately settling on a class it assigns to said input. We trained these using the popular greedy algorithm known as CART, which splits the feature space recursively using training data in order to create a decision tree for classifying new data points.

This algorithm starts at the root of the tree (i.e. some specific element of the input vector) and, for each feature in the samples' feature vectors, chooses a "best" splitting point for defining a class from the range that said feature exists in; we define this as the point that minimizes the sum of the *Gini impurities*[5] across the two children nodes. Once the algorithm has done this with the entire training set, it then chooses the best split point at each feature by using the "purest" split among those points it found. This simple process is repeated until some stopping criterion is met (e.g. the existence of a node with zero impurity).

A major problem with classification trees is that they tend to overfit the training data it is given. To deal with this problem as well as reduce variance, we can use what is known as bagging, since the computational burden of producing a tree is minimal. Tree bagging (also known as bootstrap aggregating) takes $n$ subsets of training samples with replacement from the complete set of training data and trains a classification tree using these subsets. When we have input we hope to classify, we simply run it through every tree and then use the class predicted by the majority of the trees. Of course, this method is rather naïve and there tends to be correlation between trees due to certain features having higher predictive power for certain classes. We again rely on the simplicity of this model to tackle the problem by creating a "random forest" of trees created from random subsets of the features themselves (with bagging done on each subset of features).

## V. NEURAL NETWORKS

Neural networks are a popular form of machine learning based loosely on connections in the brain. They have gained traction recently due to their success in areas such as computer vision, natural language processing, and time series classification and forecasting. Neural networks consist of layers of nonlinear functions composed with trainable linear transforms and as with classic machine learning, they ultimately aim to approximate statistical distributions hidden in underlying data. These models have next to no interpretability, but they generalize trivially to new data sets and can drastically increase accuracy when enough data are available.

### 1. Neurons and Connections

Neural networks are composed of two main components neurons and synapses. Synapses are weighted connections between neurons that together form the linear transforms between nonlinear-neuron layers.

We can think of a neural network as a series of matrix multiplications followed by nonlinear transforms on vectors. That is to say, our input vector $X = (x_1, x_2, ..., x_n)$ is operated on by the synapses $a_{ij}$ composing a matrix of weights

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

and then each of the resultant vector's elements is fed into some nonlinear function. We repeat this process through a predetermined number of layers until we have a vector of the same dimension as we have classes. It is very important that we note $m = n$ in general, as neuron layers can contain different amounts of neurons.

### 2. Training

Training of the neural network hinges on altering the weights in between layers. A training set consists of data

for which we know the expected outputs. To train we alter weights between layers using a method of computational gradient descent known as back-propagation. The goal of backpropagation is to recreate expected input-output pairs on our training set. Back-propagation consists of two steps: the forward pass and the backward pass. The forward pass requires calculating outputs of our network over the training set, and the backward pass calculates gradient descent over our loss function via the chain rule through our network. This first requires a notion of "distance" which is provided by our loss function. A loss function tells us how close we are to our expected output on the training set.

The specific loss function we use for our neural networks is cross entropy. Cross entropy is a value from information theory which measures the average amount of information needed to identify an event from an underlying set. It takes two input probability distributions, $p$ and $q$, and is calculated as:

$$H(p,q) = -\sum_{p} p(x)\log\left(q(x)\right)$$

Naturally, this makes sense as a loss function since we are looking to determine an event based on the smallest amount of information possible. The value is closely related to the Kullback-Leibler Divergence,

$$D_{KL}(P||Q) = \sum_{i} P(i)\log\frac{P(i)}{Q(i)}$$

which, while not symmetric, and thus not a true metric, determines the "distance" between probability distributions.

The actual calculation of the gradient descent can be done in a few ways; here we focus on stochastic gradient descent, which is what was used. The method consists of taking portions of the training set, calculating loss for them, averaging them, and then calculating gradient descent for this average. This is opposed to standard gradient descent which does the same for full expectations of the training set. With stochastic gradient descent we can, in many situations, avoid settling into a local, as opposed to a global, minimum. This motivates our use of the method.

### 3. Recurrenty Neural Networks

What we have described is the theory around the "feed-forward" neural network. This system has a large shortcoming when it comes to time-dependent data. Feed-forward networks are poor at finding correlations between disparate events in time. To remedy this, we give our network a notion of memory: this constitutes the recurrence. Recurrent neural networks (RNNs) are feed-forward networks where the output from the last hidden layer is included in the input during the next iteration of the network. To train, we take time-series data and partition it. These partitions are fed in to the network successively and the hidden layer output is fed into the network again for the successive calculation. In a sense this can be seen as a very deep feed-forward network, but in time.

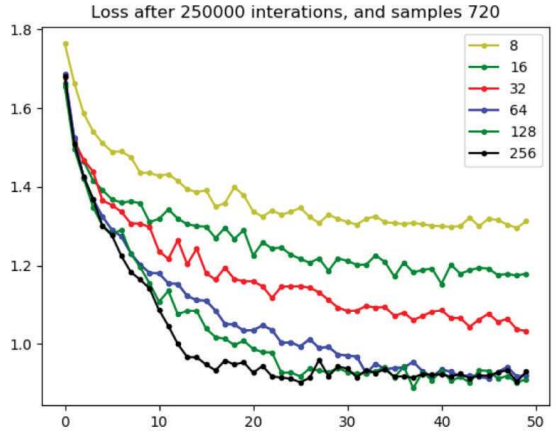


Figure 7. Each color represents different numbers of neurons in the hidden layer. The floor of the loss seems to change predictably with the decreasing size of the hidden layers. Once the hidden layer size is 64, we seem to have found the minimum of this loss function that is possible by only changing this parameter.

In fact, training is done the same way training a feed-forward network is done. The main difference here is that during gradient descent partial derivatives are in time. These networks are particularly deep, though, and as such are very susceptible to both vanishing gradient and gradient blow up. Respectively, these entail our gradient all but vanishing to zero, and our gradient becoming too large and changing our loss drastically in a single iteration. Both of these are important to avoid, and make training RNN's far harder than feed-forward networks.

The recurrent neural network used in our project was based off a simple single layer network, and was programmed in the Python package PyTorch. A set of autoregressive coefficients were set as the input layer for the recurrent network. Usually, an important aspect of the architecture of a recurrent network is the ability to take in variable input sizes, but with this feature input system, it is not entirely necessary.

We used cross entropy loss in conjunction with stochastic gradient descent for training. A total of 250, 000 iterations of training were done, which equates to roughly 250 passes through randomly generated training sample sets. In order to decide an optimal hidden layer size for classification, we used a variety of different sizes. Using the set of values [8, 16, 32, 64, 128, 256] for hidden layer size, the recurrent network was trained and tested on the same data in order to compare accuracy between sets. Overall, there was large improvement

on loss and accuracies for all increases in hidden layer size, with an exception of hidden layers of size 64, 128, and 256 which were comparable in loss and test results.

## VI. RESERVOIR COMPUTING

Reservoir computing is a paradigm from computational neuroscience that has been gaining traction in recent years due to the computational efficiency and accuracy of the models it produces. The basic idea is to use an RNN without trained weights as a nonlinear, time-dependant filter for the input signal and to use the new time-series of states from some predetermined subset of neurons in this filter as the input to a simple classification model (typically a regression). Due to how recently these models were discovered, there is no hard consensus on what exactly qualifies as a reservoir computer, but it is generally accepted in the literature that any reservoir computing model has input neurons, a reservoir containing many recurrently connected neurons, and a set of readout neurons whose states are used for classification. An example of this is found in Figure 8. These models are arguably the least interpretable of any of them, but generally require much less data than RNNs.

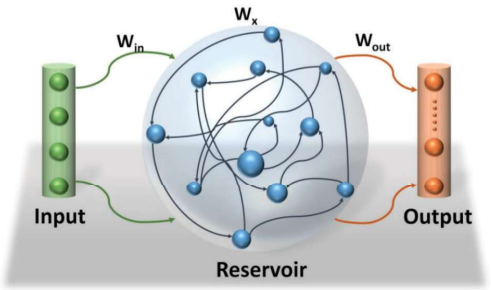


Figure 8. The basic structure of reservoir computing models. Only $W_{out}$ is trained corresponds to training the classifier being used.

As opposed to an neural network, a reservoir computer does not change internal weights and instead trains only the classifier itself. Furthermore, individual points from the signal are fed into the reservoir sequentially. By constructing a model in this way, we rely on the fact that a reservoir is a highly coupled dynamical system with unique solutions that essentially behave like a preprocessing step (input datum act like perturbations which drive the high dimensional dynamics). Not much is known about what makes a reservoir "good," but they tend to perform best when they allow the effects of previous signal points to echo without amplifying themselves. The time series of states of the readout neurons are bundled together and used to train a classifier in the hopes that the dynamics induced by new data are comparable.

In this project, we developed a modular code base using PyTorch for quickly specifying reservoir computing models (we could quickly change topology, weight sampling distribution, size, etc.). Using this we experimented with multiple reservoir network structures, but focused on two simple topologies since they emphasize extremes of modern reservoir design found in the literature:

(i) a fully connected graph of nodes and with weights sampled from a normal distribution and (ii) a cyclic graph of nodes with fully connected input, fully connected output, and weights sampled from a simple uniform distribution. Both use simple *tanh* neurons which transform their input as follows:

$$\tanh\left(I(t) + \sum_{v \in N(x)} v w_{v,x}\right)$$

Here $N(x)$ is the open neighborhood of node $x$, $w_{v,x}$ is the weight from node $v$ to $x$, and $I(t)$ is the input data at time $t$. We should note that since noise should not present itself in a consistent way, Reservoirs should intuitively be good at removing the effects of noise, as such we do not feature extraction nor preprocessing on the experiments involving reservoirs[6].

| Model | Features | Accuracy |
|---|---|---|
| True Random | N/A | 16.6% |
| FC Reservoir | None | 37.7% |
| SVM | AR(6)+RMS | 40.1% |
| Ring Reservoir | None | 43.3% |
| RNN | ARMA(1,1) | 54.5% |
| Classification Trees | AR(6) + RMS | 91% |

Figure 9. Comparing classification methods for hand data

| Model | Features | Accuracy |
|---|---|---|
| True Random | N/A | 6.6% |
| SVM | AR(11)+RMS | 22.9% |
| Classification Trees | AR(6)+MAV+WAMP | 55% |
| LSTM | None | 60.7% |
| Ring Reservoir | None | 70.3% |
| FC Reservoir | None | 82.5% |

Figure 10. Comparing classification methods for digit data

## VII. RESULTS & CONCLUSIONS

For classification of hand data, classical machine learning achieved by far the best results, while the reservoirs struggled. A comparison of the methods described above can be found in Figure 9. In the classification of the digit data, we had drastically different different performances between our models. On this dataset, the reservoirs performed the best while the classic machine learning methods yielded poor results despite careful feature engineering.

This work suggests that feature extraction methods in conjunction with simple machine learning models are good when computational resources are limited and

interpretability is key. By contrast RNNs are good when there isn't a human capable of tuning a model's features to the dataset in question or when data becomes abundant. Reservoir computing models share some of the pros and cons from both of these approaches since they do not required a skilled human nor an abundance of data, but they require some intuition about dynamic systems and luck. As one would expect, there is no "best" approach for balancing interpretability, accuracy, and generalization to new data sets.

[1]It is rather obvious in Figure 1 that there are windows of time where the signal is in phases which correspond to the subject being recorded at rest vs. while performing an action.

[2]Detailed explanations and all of our classification results can be provided upon inquiry to Qian-Yong Chen: qchen@math.umass.edu

[3]We also experimented without feature extraction, but the results were not far from "random" classification as expected.

[4]For this reason they are aptly named maximum-margin hyperplanes.

[5]For $N$ classes and $p_i$ describing the fraction of items labeled with class $i$ on the node $p$, we define a Gini impurity by $\sum_{i=1}^{N}(p_i(1-p_i)) = 1 - \sum_{i=1}^{N} p_i^2$

[6]Interestingly we found that, for essentially every network topology we tried, preprocessing the data tended to have a negative effect on the Reservoirs' ability to generalize.

## HACKING LECTURES AT 'OLYMPICS OF MATHEMATICS' IN RIO DE JANEIRO

Associate Professor **Paul Hacking**, with his collaborator Sean Keel from the University of Texas at Austin, was invited to deliver a lecture at the 2018 International Congress of Mathematicians (ICM) held August 1-9 in Rio de Janeiro. With Keel unable to attend, Hacking delivered the lecture to members of the Algebraic and Complex Geometry section on 7 August.

The ICM, held roughly every four years since 1897, is one of the premier forums for presenting and discussing significant mathematical discoveries. Some call it the Olympics of mathematics; the gold Fields Medal that is awarded there Hacking describes as "the Nobel Prize of mathematics.

"It's an unusual honor to be invited to give a speech at this international meeting. I very much appreciate the recognition of my peers. I'm looking forward to it, but it is a bit daunting. Many of my colleagues will be there and I look forward to catching up with them." The last time UMass Amherst was represented by a speaker at the ICM was 32 years ago in Berkeley, where **Bill Meeks**, now a distinguished professor emeritus, had the honor.

Hacking's research area is algebraic geometry, one of two primary methods scientists use to study and define shapes. He explains, "Differential geometry uses the tools of calculus to solve geometric problems, whereas in algebraic geometry we use abstract algebra instead."

"Geometry is often intuitive, and it's easy for us to visualize the difference between the surface of a donut and a sphere," he adds. "But to absolutely pin it down, you need to develop a language to rigorously describe these objects and how a ball is different than a donut. Mathematical language will nail it down precisely."

Hacking has worked with two main colleagues, Mark Gross at Cambridge, U.K., and Keel at UT Austin, plus Maxim Kontsevich of the Institut des Hautes Études Scientifiques, near Paris, to produce several papers and a survey of this research area over the past five years.

One field where algebraic geometry is useful is theoretical physics, Hacking says. String theory, which seeks to describe the fundamental forces of nature and how the universe operates, asserts that rather than the three dimensions plus time we are familiar with, there are instead 10 dimensions, and six of them are very, very small, in the quantum arena and not visible to the naked eye.

"Imagine a garden hose seen from a long distance away that appears to be one-dimensional, but as you get closer you see another dimension," he explains. "String theory says if you were able to look at smaller scales you'd see extra dimensions. At very small scales, quantum mechanics plays a role in physics and weird phenomena will be explained by studying these six extra dimensions. There is a geometry of the very small object that would explain quantum behavior. This six-dimensional object is called a Calabi-Yau (C-Y) manifold, and it's one of the objects we study in algebraic geometry."

Understanding quantum physics requires precisely describing the shapes of these manifolds, Hacking says. "The other key idea is that the elementary particles of quantum physics are not points but little loops of string. This smooths out the interactions and makes the mathematics possible. Particle interaction, then, rather than an instantaneous collision, is gradual." This leads on to mirror symmetry, which refers to paired C-Y manifolds, related in physics but mathematically very different, he adds.

Hacking, who came to campus in 2009 from the University of Washington, earned his undergraduate and advanced degrees at the University of Cambridge and completed a postdoctoral fellowship at the University of Michigan.