Layered ϵ-Machines

G. Mikaberidze*
Department of Physics, University of California, Davis, CA, 95616, USA
(Dated: June 16, 2017)

ϵ-Machines are optimal predictors of minimal size. This means nothing can beat them at optimal prediction. The goal of this project is to explore a new structure: Layered ϵ-Machine, and compare it with standard ϵ-Machine when the optimal prediction is impossible for practical reasons. I use stock market prices for different goods as input data. One part of the data sequence is used to infer the structure and parameters for both ϵ-Machine and Layered ϵ-Machine, the rest of it is used to check how well each predictor works.

I. INTRODUCTION

Mathematical structures that can learn are already an important part of our every day life. Virtual Personal Assistants (Siri, Google Now, Cortana etc.), Purchase Predictions (Target, Amazon etc.), Spam Detection, Online Customer Supports, Smart Home Devices, all of these are either continuously learning and becoming better or at least have been trained using real world data. These algorithms are already serving millions of people and are becoming more and more important.

How can mathematical structures learn? How do even living organisms learn? If we find the answer to the first question, it will also explain the second one as brain can simulate mathematical processes. Thus if mathematical structures can be trained, than surely brains can be trained too.

Suppose we have some stationary process that generates information: sequence of symbols. We are observing the information, but the process is hidden from us. Turns out that there are ways to extract knowledge about the source by looking at the symbols only [1]. The goal is to have a smallest structure that would still capture all the available information about the source. Such structures exist and they are called ϵ-Machines. ϵ-Machines are the smallest optimal predictors.

If ϵ-Machines are the smallest optimal predictors, why do not we use them for example for face recognition purposes? One could convert images into strings of symbols, after each such string there would be a 1 if it contained a face and 0 otherwise. One could use a sequence of such strings to train an ϵ-Machine and then use it to predict next symbol after inputting any image. This digit would be the best possible guess about a face being in the picture. As a side effect this machine would also be able to complete a partially seen image to the best of theoretical possibility. It is clear from this description that this kind of structure would have to be much larger than what our smart-phones can handle.

Therefore in such cases we have to forget about optimal prediction and build structures, that are much smaller and still predict well. Obviously, for any such structure there is a corresponding ϵ-Machine which will do the same job and is not larger. The problem is finding such machines, because there are too many possibilities.

Now let us look at this problem from the other point of view. Let us discuss music as an example. The alphabet is chords and it is a discrete sequence. If we try to use Bayesian Inference [2] to find the best suiting epsilon machine of limited size, we will have to limit our self to very small ϵ-Machines: only several states. This means, that we will not be able to capture long scale behaviour. But in case of even simple music, it is obviously important to keep track of major / minor, repeating theme and so on. In other words, it's possible, that for practical purposes one would prefer to capture some long range behavior first and leave out some short scale details. And this is the main idea behind Layered ϵ-Machines: infer structure from symbol sequence in the giving priority to long range behavior.

According to Christopher C. Strelioff and James P. Crutchfield [2], inferring structure from data series is also an integral to many fields of science ranging from bioinformatics [3, 4], dynamical systems [5–8], and linguistics [9, 10] to single-molecule spectroscopy [11, 12], neuroscience [13, 14], and crystallography [15, 16].

II. BACKGROUND

ϵ-Machines are directed graphs with each node representing an internal state of the source and each edge representing a possible transition with a corresponding output symbol and probability. It is unifilar: if we know current state and see a transition symbol, there can be no ambiguity regarding the next state. The process can be stochastic, and at each point the future has different possibilities with corresponding probabilities (called future morph). Saying that system is in a given state means, it has a given future morph.

To infer ϵ-Machine for a given data string, one can use Bayesian inference. This algorithm takes data string and possible ϵ-Machines (without transition probabilities). It finds the best suiting machine and tunes transition probabilities. One can extract the most probable machine or one can sample machines according to their likelihoods.

* mikaberidze@ucdavis.edu

III. DYNAMICAL SYSTEM

For this project I chose stock market prices for different goods as a dynamical system. The main reason for this was that the state number is probably immensely large. Also the alphabet is just simple numbers and it can be simplified even more by reducing it to binary alphabet: 1 - price went up; 0 - price went down. It is also a good option to use ternary alphabet: if the price didn't change more than some given threshold, we assign 1; if it went higher than threshold, assign 2; if it went down lower than threshold, assign 0. The process is clarely non stationary and this could cause some problems, but it could cause same kind of problems for both ϵ-Machine and Layered ϵ-Machine, and the only thing that I am interested in for this project is the comparison of these two.

The stock markets have changed very much in 21^{st} century, thus to make the comparison less random, I will cut off prices for last 17 years.

IV. METHODS

A. Constructing Layered ϵ-Machines

Although long scale behavior can be important, ϵ-Machines do not allow for capturing it before all the shorter scale behavior is captured. The problem is, that we do not have enough computational power to deal with behavior on all the scales completely. That is why in Layered ϵ-Machines I am constructing separate ϵ-Machines to deal with different scale behaviors.

The idea is simple: we have to pick some beaning size n and some beaning function f, such that f maps n symbols from original alphabet to 1 symbol (alphabet of this new symbol might be different). Now we have to split the original data into sub-strings of length n and map each sub-string to a new symbol using f.

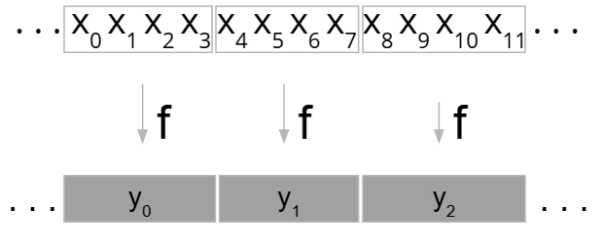

FIG. 1. Beaning example with $n = 4$.

If we had enough computational power to do optimal predictions, there would not be any need of making a Layered ϵ-Machine in the first place, consequently f is supposed to be lossy, many-to-one function. It is supposed to capture some important aspect of long scale behavior and get rid of some details. For example, in case of stock market prices, a good beaning function should be obtained just by comparing first and last symbols in a sub-string: $f(substring) = 1$ if price went up; $f(substring) = 0$ if price went down.

Once we have the new sequence, we can build an ϵ-Machine for it using Bayesian inference. This machine will learn long scale behavior of the system. The hope is, that short scale behavior of the process depends on the long scale state. A good example is music here: the short scale behavior is heavily affected by long scale state being "major" or "minor". Therefore, for each state of long scale machine we can build a new, short scale machine.

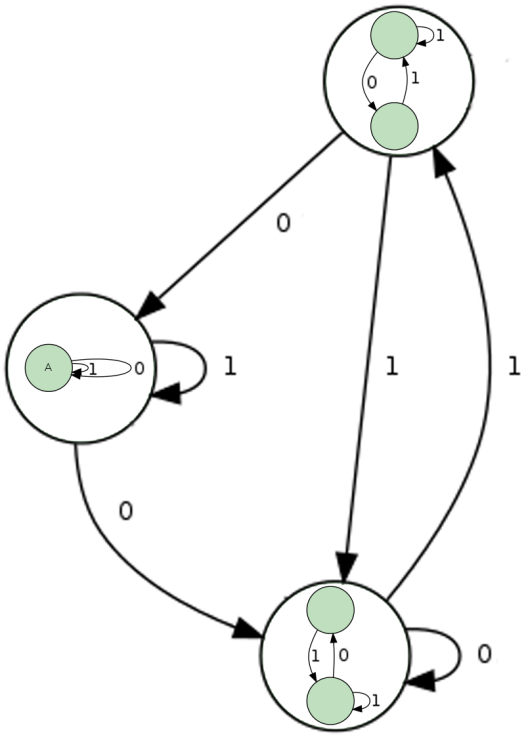

FIG. 2. Visualization of Layered ϵ-Machine.

The number of short scale machines is the same as the number of states in long scale machine. We have to train each of them separately using Bayesian Inference. To infer short scale machine, we should use the sub-strings of original data that were produced while the long scale machine was in this specific state.

Bayesian Inference takes in a single string of data symbols, but for short scale machine we have a number of short sub-strings. One straightforward solution to this problem would be to concatenate the sequences and use it as a single string. This approach has two problems: one is, that the sequence contains a lot of unreal transitions that never happened, so we would be training an ϵ-Machine with wrong data; another is that we would not have a good starting state for short scale machine. Bayesian Inference would give the best starting node but it would be tested only once at the very beginning of the concatenated string. We need the best starting state to

use every time when large scale machine activates this short scale one.

A good solution for the problem described above is to make use of a new resetting symbol: instead of just concatenating the sequences, now we will sandwich the resetting symbol between them, and from every candidate machine we will require, that this symbol is only used for transferring any state to the starting state. Therefore the number of candidate machines and computation time stays unchanged. Now, every time the machine meets this new symbol, it resets and continues training with the new sub-string from the start state.

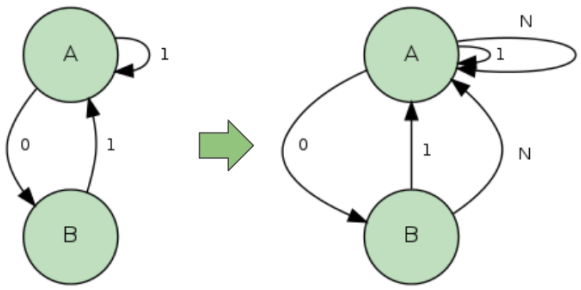

FIG. 3. Adding a reset transitions with reset symbol N.

Other than training data, Bayesian Inference also takes in a set of candidate ϵ-Machines. Standard set is topological ϵ-Machines. This is the set of machines that have different future morph topologies for each state, i.e. the future morphs for each state have to differ not only by transition probabilities, but also by allowed symbol sequences. In case of stock market prices, anything can happen, therefore all sequences are possible from any state. This means, that future morphs have to have the same full topology for every state (full support). The only topological machine with full support is the single state ϵ-Machine. Therefore we need to construct our own set of candidate machines.

For our purposes we need only full support ϵ-Machines. To construct all such machines of given size k, we need to start with a graph of k nodes and no transitions. Full support machine automatically means that every node has outgoing edges for any symbol from the alphabet. Thus we have to go over all states and assign A outgoing edges to each, where A is the size of alphabet. I build the possible graphs recursively: each step of recursion picks the first node that does not have all the outgoing edges and adds a new edge to it. But this edge can be going to any other node, therefore it copies the graph and for each copy this new edge goes to a different node. Finally it calls the recursion for each of the new copies.

If the algorithm does just this, we will have a lot of duplicate machines which will differ by only the names of nodes. We do not want this to happen, as training extra machines will take unnecessary computational time and memory. To avoid this, we can keep track of all the states that have not been connected to anything else yet. These states are completely symmetric to each other. Therefore, at every stage of recursion, we need to connect the new edge to all the nodes that are not in this set and to only one node from this set.

finally, we will have to eliminate the disconnected graphs, add the reset transitions and use Bayesian Inference to select the best one and tune it.

One obvious generalization of the structure described above is to allow more than one layer. We can have as many layers as our computational power allows. The algorithm generalizes simply: Instead of one beaning number and one function we now have $l-1$ beaning numbers and functions, where l is layer number. First we bean down the data sequence step-by-step for all the layers, after that we start by constructing standard ϵ-Machine for lowest layer, for each state of this machine we construct a new machine on the next layer. For each state of each machine on this layer we construct a new ϵ-Machine on the next layer and so on.

B. Checking how well does a Layered ϵ-Machine predict

Once the Layered ϵ-Machine is trained with one part of available data, I use the rest of it to test how well the Layered ϵ-Machine predicts the future prices.

At each moment Layered ϵ-Machines make predictions on every layer. For each layer there is only one active machine, and for each machine there is only one active node. These nodes have outgoing edges with corresponding symbols and probabilities. Each node is bound to have reset transition with it's probability. These transitions are artifacts of construction, therefore we need to ignore them and normalize the other probabilities. Once normalized for each layer we have corresponding predictions of symbol probabilities. These are different scale behavior predictions.

To test how well the Layered ϵ-Machine predicts, I look at the predictions of the top layer. This predictions are probabilities of daily prices going up or down.

Once the machine is trained and synchronized, I run a simple trading algorithm using the daily predictions. The algorithm should not try to maximize mathematical expectation for daily income. Otherwise it would bet everything on even 51% chance and would loose all the resources very soon. Instead the algorithm starts off with some amount of money and every day it buys or sells goods according to the predictions. Amount of goods it buys or sells depends on how sure we are that price will grow or fall and how much money we currently have. So daily gain is $money * (newPrice - oldPrice) * (upProb - downProb)$.

This is not a good trading algorithm, but it does not have to be. It is important, that if the predictions are good, we will end up with increased money, and the better the predictions are, the more the gain will be.

V. RESULTS

Once the algorithm is implemented, if we choose to have only 1 layer, it ends up being a standard ϵ-Machine. As I already mentioned, I am only interested in comparison of ϵ-Machine and Layered ϵ-Machine. Thus the only thing that needs to be done is to run the same trader algorithm for both of them.

I have daily stock market prices for different goods: Crude Oil; Gold; Copper; Natural Gas; Silver; U.S. Dollar Index. The longest one starts from 1967, the shortest one starts in 1985. For each of these sequences I am using roughly 2/3 of the data to train Layered ϵ-Machine and ϵ-Machine and I use the rest of data to trade.

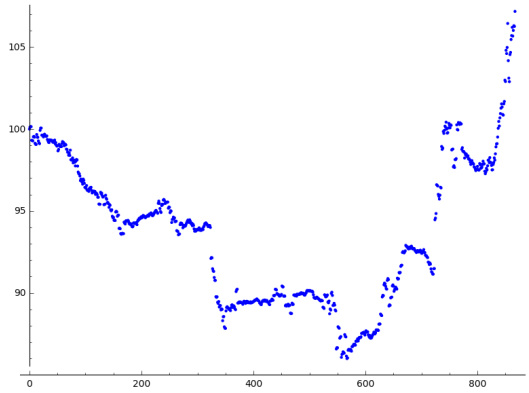

FIG. 4. Total balance VS trading day. Trading silver. Layered ϵ-Machine parameters: 1 layer; beaning size 50; candidate ϵ-Machine sizes [1,2,3]

Because I use 6 different kinds of goods, I can calculate total gain for each of them, then look at the average and standard deviation. I can do this for different beaning sizes, candidate ϵ-Machine sizes and layer numbers (including $l = 1$ corresponding to ϵ-Machine).

Layers	Beaning num	Mach. sizes	avg gain	std
2	50	[1,2,3]	0.9	74
2	50	[1,2]	10.4	29
2	20	[1,2,3]	-30.2	40
2	20	[1,2]	10.5	29
1		[1,2,3]	-15.9	32
1		[2,3]	-13.6	33
1		[1,2]	-15.9	32

TABLE I. Parameters of trading machines and their results

In TABLE I one can see Layered ϵ-Machines and ϵ-Machines with different parameters and different trading results. Column "Layers" contains number of layers for each machine. Column "Beaning num" is the beaning number, if there is only one layer, it is undefined. "Mach. sizes" is the sizes of candidate ϵ-Machines. "Avg gain" is the average gain and "std" is its standard deviation.

VI. CONCLUSIONS

According to TABLE I, standard deviations are larger then gain differences between. Thus I can not conclude rigorously that any of the Layered ϵ-Machine of ϵ-Machine performed better then the other. Neither FIG. 4 looks very promising. It is an average example of a winning strategy and one can see that it is very unstable. And thus it is totally unsuitable for actual trading.

This all said, I still can not disregard the fact that average gains are on the average better for the Layered ϵ-Machines compared to the ϵ-Machines. This comparison obviously still needs to be carried out for some other dynamical systems, which hopefully would be more stationary and give smoother results.

[1] Crutchfield, James P. "Between order and chaos." *Nature Physics* 8, no. 1 (2012): 17-24.

[2] Strelioff, Christopher C., and James P. Crutchfield. "Bayesian structural inference for hidden processes." *Physical Review E* 89, no. 4 (2014): 042119.

[3] Yoon, Byung-Jun. "Hidden Markov models and their applications in biological sequence analysis." *Current genomics* 10, no. 6 (2009): 402-415.

[4] Narlikar, Leelavati, Nidhi Mehta, Sanjeev Galande, and Mihir Arjunwadkar. "One size does not fit all: on how Markov model order dictates performance of genomic sequence analyses." *Nucleic acids research* 41, no. 3 (2012): 1416-1424.

[5] Davidchack, Ruslan L., Ying-Cheng Lai, Erik M. Bollt, and Mukeshwar Dhamala. "Estimating generating partitions of chaotic systems by unstable periodic orbits." *Physical Review E* 61, no. 2 (2000): 1353.

[6] C. Stuart, Charles Edward Andrew Finney, and Eugene R. Tracy. "A review of symbolic analysis of experimental data." *Review of Scientific instruments* 74, no. 2 (2003): 915-930.

[7] Kennel, Matthew B., and Michael Buhl. "Estimating good discrete partitions from observed data: Symbolic false nearest neighbors." *Physical Review Letters* 91, no. 8 (2003): 084102.

[8] Strelioff, Christopher C., and James P. Crutchfield. "Optimal instruments and models for noisy chaos." Chaos: *An Interdisciplinary Journal of Nonlinear Science* 17, no. 4 (2007): 043127.

[9] Rao, Rajesh PN, Nisha Yadav, Mayank N. Vahia, Hrishikesh Joglekar, R. Adhikari, and Iravatham Mahadevan. "A Markov model of the Indus script." *Proceedings of the National Academy of Sciences* 106, no. 33 (2009): 13685-13690.

[10] Lee, Rob, Philip Jonathan, and Pauline Ziman. "Pictish symbols revealed as a written language through application of Shannon entropy." In *Proceedings of the Royal Society of London A: Mathematical, Physical and En-*

gineering Sciences, p. rspa20100041. The Royal Society, 2010.

[11] Kelly, David, Mark Dillingham, Andrew Hudson, and Karoline Wiesner. "A new method for inferring hidden Markov models from noisy time sequences." *PloS one* 7, no. 1 (2012): e29703.

[12] Li, Chun-Biu, Haw Yang, and Tamiki Komatsuzaki. "Multiscale complex network of protein conformational fluctuations in single-molecule time series." *Proceedings of the National Academy of Sciences* 105, no. 2 (2008): 536-541.

[13] beim Graben, Peter, J. Douglas Saddy, Matthias Schlesewsky, and Jrgen Kurths. "Symbolic dynamics of event-related brain potentials." *Physical Review E* 62, no. 4 (2000): 5518.

[14] Haslinger, Robert, Kristina Lisa Klinkner, and Cosma Rohilla Shalizi. "The computational structure of spike trains." *Neural computation* 22, no. 1 (2010): 121-157

[15] Varn, Dowman P., Geoffrey S. Canright, and James P. Crutchfield. "Inferring planar disorder in close-packed structures via-machine spectral reconstruction theory: structure and intrinsic computation in zinc sulfide." *Acta Crystallographica Section B: Structural Science* 63, no. 2 (2007): 169-182.

[16] Varn, D. P., G. S. Canright, and J. P. Crutchfield. "-Machine spectral reconstruction theory: a direct method for inferring planar disorder and structure from X-ray diffraction studies." *Acta Crystallographica Section A: Foundations of Crystallography* 69, no. 2 (2013): 197-206.