Augmenting soft computational machines with the addition of an intelligence prior

Christopher N. Singh,[1,*] Matthew Redell,[1] Mohannad Elhamod,[2] Jie Bu,[2] Anuj Karpatne,[2] and Wei-Cheng Lee[1,†]

[1] *Department of Physics, Applied Physics, and Astronomy, Binghamton University, Binghamton, New York 13902, USA*

[2] *Department of Computer Science, Virginia Tech, Blacksburg, VA 24060, USA*

(Dated: September 18, 2019)

Improving the predictive power of machine learning models is one of the greatest challenges to the science of learning. Here we demonstrate with the simplest of neural networks that the addition of an intelligence prior can drastically improve the learning capabilities. We outline this simple mechanism to decrease the number of exposures, and enhance the predictive power with a number of examples relevant to the study of quantum phase transitions. We find that guided networks are uniquely capable to identify key features of quantum phases where unguided models fail, and that while the mean square error of topologically equivalent models may be commensurate, the structure of the predictions produced by the models is qualitatively very different. In many situations where knowledge of a physical system is available, but direct sampling of the entire phase space is computationally intractable, this approach offers a superior learning alternative.

I. INTRODUCTION

The distillation of knowledge from data defines one of the most fundamental human endeavors – learning, and when viewed as the means by which actionable information is extracted from raw, unfiltered data, learning forms the basis of all intelligence. This fundamental principle has revitalized the science of learning in recent times, with great advancements in both hard and soft intelligence. And even though many of today's most challenging problems can be stated elegantly, consonant solutions are rarely, if ever, available. The consequence of this situation is that finding the best solution is almost always reduced to an optimization problem, where learning is maximized over computation.

Formulation as a minization problem, being unavoidable, affixes serious challenges to computation of expectations, namely, updating proceeds by estimation, with uncertainty inversely proportional to effort. In a world without infinite computational power, one must decide either to have fewer, accurate updates, or many, poorer updates. Computational intelligence, also known as machine learning techniques, have found success in tackling the optimization problem by mimicking biological systems, but still ultimately suffer from this fundamental complication. An exponentially or combinatorically exploding state space, in many ways, remains the final resolute barrier that still compels human mediations.

This work posits a mechanism to enhance the capability of soft intelligence machines' performance in this domain with the addition of an intelligence prior. In this approach, the validity of expectations are gauged with respect not only to the targets, but also required to be consistent with previously accumulated intelligence. We show that this extension pushes soft computational machines' extrapolative power beyond the feature space to which they were exposed, simply because each update is constrained to consistency not only with the targets, but also with a more general behavioural model.

As proof of concept, we apply this approach to identify quantum phase transitions in rings of qubits, as well as localization transitions in disordered, electron-lattice systems. The transverse field Ising chain is a quintessential physical model, thus is goverened by the Schrödinger equation; maintains an exponentially exploding state space, and has an analytical solution. As such, it is a perfect system from which to extract the viability of this technique. Similarly, localization transitions manifest in the simplest of electron models, with the governing parameters being well studied analytically and numerically, and recent applications being suggested in hard computational intelligence, closing an elegant loop between the two fields.

Ultimately, we demonstrate that the ability of the most primitive of soft intelligence machines, the fully connected feedforward neural network, to predict quantum phase transitions can be drastically improved with the simple addition of a physics prior. The mean square error on the testing datasets drops much quicker in the physics guided models than in the black box counterparts, and the predictive power is greatly enhanced. In detecting a quantum phase transition, the physics guided machine can predict qualitative changes in the eigenspectrum that the black box machine entirely overlooks. While identifying a phase transition has been done before with black box models, these models were exposed to training instances in both phases. The guided approach codifies an approach that can predict a phase transition having never been explicitly exposed to it. Therefore, the evolution of physical observables like the magnetization and the fidelity susceptibility accross the transition, when sought by two networks of identical topology, can only be realized with learning guided by an intelligence prior. This result has deep implications for the field of computational intelligence, because it provides a route to augment learning in such a way that predictive power is maximized.

* csingh5@bighamton.edu

† wlee@binghamton.edu

II. METHODS

Because learning the energy whilst having the actual data, combined with the issues of outputs on various domains, does not work and is wasteful with information we already do have, we have decided to formulate the physics loss portion of the cost function in a way that is more transparent, and applicable to a wider range of problems. The condition we were using before is that the coefficients of the ground state wavefunction should satisfy the Schrödinger equation. However, the entire spectrum can be constrained in the exact same way. The following of course will hold for any λ_i if every $\vec{\psi}_i$ is an eigenstate.

$$\sum_i \lambda_i(\hat{H}\vec{\psi}_i - E_i\vec{\psi}_i) = \vec{0} \tag{1}$$

This means we can formulate a matrix representation for the scalar physics cost with the help of two auxiliary definitions for an energy matrix and a Lagrange matrix. The matrix formulation of the equations is important for achieving vectorized operations at the implementation phase. If we define the energy matrix as

$$\hat{E} = \hat{I} \odot (\vec{1}^T \otimes \vec{E}), \tag{2}$$

where $\vec{E}$ is a column vector of the eigenvalues. Likewise we can define for the Lagrange matrix

$$\hat{\lambda} = \hat{I} \odot (\vec{1}^T \otimes \vec{\lambda}). \tag{3}$$

We always stick with Hessian notation, so that the primary vectorial objects are column vectors and matrices are collections of column vectors. In this sense $\vec{1}$ is a column vector of ones and $\vec{0}$ is a column vector of zeros.

Using these definitions, the scalar physics cost function can be defined as

$$C_2 = \frac{1}{2}||\hat{H}\hat{\Psi} - \hat{\Psi}\hat{E}||_F^2. \tag{4}$$

The backpropagation equations require the derivative of the cost function, so here we can take the matrix derivative

$$\frac{\partial C_2}{\partial \hat{\Psi}} = \hat{H}^2\hat{\Psi} - 2\hat{H}\hat{\Psi}\hat{E} + \hat{E}^2\hat{\Psi} \tag{5}$$

In this way, the elements of the result correspond to the gradient of the physics portion of the cost. This can be much more straight forwardly implemented with faster vectorized operations. In the matrix implementation of backpropagation, this expression is evaluated for each instance in the mini batch.

Finally, the method that has shown the most success has been a direct modulation of the error at the output layer.

$$\hat{\delta}^L += \lambda(\hat{H}\hat{\Psi} - \hat{\Psi}\hat{E}) \tag{6}$$

III. RESULTS

IV. DISCUSSION

We have presented two different methods for guided learning of the neural networks, one in which the landscape of the loss function is altered via an additional term C_2, the other in which the error term is directly modulated by an additional guiding function. To illustrate the differences in these methods and determine which method will result in faster learning, we want to consider a simple example. Consider a simple feed forward network with two input layer neurons $\vec{a}^0 = (a_1^0, a^0, 2)^T$, which connect to a single output neuron a^1 via a weight vector $\vec{w} = (w_1, w_2)$ and an activation function $\sigma(z)$ such that

$$a_i^1 = \sigma(z_i) \tag{7}$$

with $z_i = w_1 a_{1,i}^0 + w_2 a_{2,i}^0$ for the ith data point. A schematic diagram of this network can be seen in Fig.**input a figure**.

Now, following the standard backpropagation algorithm, we can calculate the error in the output layer for each input data point i as

$$\delta_i^1 = \frac{\partial C}{\partial a_i^1}\sigma'(z_i) \tag{8}$$

where C is the loss function used and $\sigma'(z)$ is the derivative of the chosen activation function. Furthermore, after we iterate through the training set, we want to update the weight vector in accordance with this error. We do so as

$$\vec{w} = \vec{w} - \frac{\eta}{N}\sum_{i=1}^{N} \delta_i^1 \vec{a}_i^0 \tag{9}$$

where η is the prescribed learning rate and N is the number of training data points. To distinguish which of our methods will learn at a faster rate, let's assume that there exists a set number of training epochs M after which the network will be sufficiently trained. Thus, the initial weight vector $\vec{w}(0)$ will differ from the converged weight vector $\vec{w}(M)$ by a vector

$$\vec{s} = \vec{w}(M) - \vec{w}(0) \tag{10}$$

So, we can write the backpropagation of the error for the mth training epoch as

$$\vec{w}(m+1) = \vec{w}(m) - \frac{\eta}{N}\sum_{i=1}^{N} \delta_i^1(m)\vec{a}_i^0, \tag{11}$$

where $\delta_i^1(m)$ is calculated with respect to the output neuron value of during the mth training epoch $a_i^1(m)$. This means we can characterize the distance that the mth training epoch is from the converged result as

$$\vec{s}(m) = \vec{w}(M) - \vec{w}(m). \tag{12}$$

A simple measure of the speed of training for the network thus could be found from $\Delta_m \vec{s}(m)$, the change in the difference vector as a function of m

$$\begin{aligned}\Delta_m \vec{s}(m) &= \Delta_m(\vec{w}(M) - \vec{w}(m)) \\ &= -\Delta_m \vec{w}(m) \\ &= \frac{\eta}{N}\sum_{i=1}^{N} \delta_i^1(m)\vec{a}_i^0.\end{aligned} \tag{13}$$

This indicates that the most important indicator of convergence speed will be the quality of the error function $\delta_i^1(m)$.

Now, discuss the intracacies of the 3 different methods, BB, C_2, and δ

V. CONCLUSION

We have demonstrated that the addition of an intelligence prior can greatly improve the learning capabilities and predictive power of soft computational intelligence engines. In the most rudimentary of circumstances, the improvements are vast. With the ever increasing importance of precipitating valuable information from effecitvely infinite amounts of chaotic data, succint training routines for neural networks are critical. Although we have demonstrated the efficacy of this approach as applied to quantum phase transitions in model systems, the basic premise is widely applicable to any system that follows known laws of motion in the generalized sense. This strongly suggests the art of machine intelligence should be refined to include the aformentioned precepts, and we suspect that leveraging this technique will find application accross many areas of physics and engineering.

ACKNOWLEDGMENTS

The authors thank Shetab Zaman for helpful comments regarding machine learning.

Appendix: Neuron Dynamics

One way to structure the network for learning physics is to generalize the equations of backpropagation beyond homogeneous neurons. This is necessary in the setting where each of the output neurons is responsible for some portion of a physical variable, whose domain does not necessarily lie on any specific portion of the number line. In the case of the ground state of the Schrödinger equation, one neuron will represent the energy, and the other will track the wavefunction coefficients.

There are four fundamental equations of backpropagation. The first describes the error in each output neuron, effectively how wrong that neuron is. If, in general, we have a neuron dependent firing mechanism in the output layer, as well as a neuron dependent cost function, we should write the error as

$$\delta_j^L \equiv \frac{\partial C}{\partial a_j^L}\frac{\partial}{\partial z_j^L} f_j^L(z_j^L) \tag{A.1}$$

where we use L to indicate this expression is the error only in the output layer. C is the cost function, a_j^L is the activation of the i^{th} neuron, f_j^L is the activation function of the i^{th} neuron and z_j^L is the input to the i^{th} neuron defined by the weighted sum $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$.

The next expression we need is the expression for the error of any neuron in layers that are not the output layer. This is defined as

$$\delta_j^l \equiv \sum_k w_{kj}^{l+1}\delta_k^{l+1}\frac{\partial}{\partial z_j^l} f_j^l(z_j^l) \tag{A.2}$$

where w_{kj}^{l+1} is the synaptic weight matrix connecting the i neurons in the $(l+1)^{th}$ layer to the k neurons in the l^{th} layer. The previous two expressions allow the utility of the following two, the first being

$$\frac{\partial C}{\partial b_j^l} \equiv \delta_j^l, \tag{A.3}$$

and the second being

$$\frac{\partial C}{\partial w_{jk}^l} \equiv a_k^{l-1}\delta_j^l, \tag{A.4}$$

to minimize the cost function by backpropagation of the error.

Now, since we ultimately want to learn the ground state of a quantum system, we will need to have output neurons that fire in specific ways. The ground state of a quantum system has two important quantities, the energy of the state, and the wavefunction. The energy lives on the entire real number line, while the wavefunction is an array of coefficients that live on the interval $[-1, 1]$. This will be important when we define the activation function for the output neurons if they are to represent the ground state of a quantum system.

In order to evaluate these expressions using the backpropagation algorithm, we need to define the constraint imposed by the Schrödinger equation. If we believe that the system under question follows the Schrödinger equation, then the secular equations impose constraints on the values the output neurons can take even just based on the inputs, and the cost of a neuron violating this condition can be defined as

$$C_2 = \lambda\Big[\sum_m \big[\sum_n H_{mn}c_n - E_g c_m\big]^2\Big]. \tag{A.5}$$

C_2 will compose the physical portion of the cost function, but we will retain a portion of the cost function associated with how wrong the output neurons are based on the data they are exposed to. For the canonical cost

function we chose the quadratic cost, a fairly basic choice in the machine learning community, and perhaps a better choice is available. In any case it is defined as

$$C_1 = \frac{1}{2}\sum_j (y_j - a_j)^2. \tag{A.6}$$

Our full cost function is then $C = C_1 + C_2$. From these expressions we can see that the error in the output layer will now be increased when the machine does not predict values consistent with the physical model.

The next task is to define the network topology and neuron properties. For the sake of a canonical example, let us restrict the network topology to three layers for now. As alluded to before, there is something special about the output neurons if they are to output the quantum ground state. The neurons representing the coefficients of the wavefunction, living on the interval of $[-1, 1]$, we choose to be activated as the hyperbolic tangent, this function having the same domain. The energy neuron however, is just a real number, so we choose a linear activation. Therefore,

$$a_N^L = \phi \cdot z_N^L \tag{A.7}$$

and

$$a_j^L = \tanh(z_j^L). \tag{A.8}$$

For the hidden layer, we can choose hyperbolic tangent or sigmoid as canonical choices, and will probably test both.

Before moving on to calculating the gradient of our new cost function, we will change the notation slightly so that C_2 is written in terms of the output layer neurons. Without loss of generality, we assign the computation of the ground state energy to the N^{th} neuron of the output layer, and the rest to the computation of the coefficients. The dimension of the Hilbert space we denote by d, and flatten the Hamiltonian into a row-major array to be used at the input layer.

$$C_2 = \lambda\Big[\sum_m^{N-1}\big[\sum_n^{N-1} a^1_{m\cdot d+n}a^L_n - a^L_N a^L_m\big]^2\Big]. \tag{A.9}$$

We are making good progress sculpting out this problem, and now we need to calculate the derivatives in the expression for the error at the output layer δ_j^L. Care must be taken to recognize that the error is neuron dependent, and depends on the derivative of the cost function with respect to that neuron, as well as the derivative of the activation function of each neuron. So the expression we need to evaluate is

$$\delta_j^L = \frac{\partial}{\partial a_j^L}\Big(\frac{1}{2}\sum_k (y_k - a_k)^2 + \lambda\Big[\sum_m^{N-1}\big[\sum_n^{N-1} a^1_{m\cdot d+n}a^L_n - a^L_N a^L_m\big]^2\Big]\Big)\frac{\partial}{\partial z_j^L} f_j^L(z_j^L). \tag{A.10}$$

It turns out to be easier to focus on first the $j = N$ component representing the energy neuron first, then tackle the others. For the $j = N$ component, the expression is

$$\delta_N^L = \frac{\partial}{\partial a_N^L}\Big(\frac{1}{2}\sum_k (y_k - a_k)^2 + \lambda\Big[\sum_m^{N-1}\big[\sum_n^{N-1} a^1_{m\cdot d+n}a^L_n - a^L_N a^L_m\big]^2\Big]\Big)\frac{\partial}{\partial z_N^L}\phi \cdot z_N^L. \tag{A.11}$$

The first term in only non-zero when $k = N$, and the last derivative can immediately be evaluated. The result is

$$\delta_N^L = \phi\Big(a_N - y_N + 2\lambda a_N^L(1-N)\Big[\sum_m^{N-1}\big[\sum_n^{N-1} a^1_{m\cdot d+n}a^L_n - a^L_N a^L_m\big]\Big]\Big). \tag{A.12}$$

For the other neurons $j < N$, δ_j^L has a part coming from the quadratic cost, a part coming from the physics cost, and a part coming from the activation. The quadratic cost derivative is the same as for the $j = N$ case, and the activation function for these neurons is the hyperbolic tangent, but the physics cost is more formidable. The derivative is only non-zero however in three cases when $m = n = j$, $m = j, n \neq j$, or $n = j, m \neq j$. We will break this into three terms based on the physics cost for the cases respectively. The error is then

$$\delta_j^L = \frac{\partial}{\partial a_j^L}\Big(\frac{1}{2}\sum_k (y_k - a_k)^2 + \lambda\sum_i T_j^{(i)}\Big)\frac{\partial}{\partial z_j^L} f_j^L(z_j^L). \tag{A.13}$$

Lets tackle the case when $m = n = j$ and call this $T_j^{(1)}$. The expression to evaluate is

$$\frac{\partial}{\partial a_j^L}T_j^{(1)} = \frac{\partial}{\partial a_j^L}\big[a^1_{j\cdot d+j}a_j^L - a_N^L a_j^L\big]^2 \tag{A.14}$$

and the result is

$$T_j^{(1)} = 2[a^1_{j\cdot d+j}a^L_j - a^L_N a^L_j][a^1_{j\cdot d+j} - a^L_N] \tag{A.15}$$

$T_j^{(2)}$ is the term for the case when $m = j, n \neq j$. The expression to evaluate is

$$\frac{\partial}{\partial a^L_j}T_j^{(2)} = \frac{\partial}{\partial a^L_j}\Big[\sum_{n\neq j}^{N-1} a^1_{j\cdot d+n}a^L_n - a^L_N a^L_j\Big]^2, \tag{A.16}$$

and the result is

$$T_j^{(2)} = 2a^L_N(2-N)\Big[\sum_{n\neq j}^{N-1} a^1_{j\cdot d+n}a^L_n - a^L_N a^L_j\Big] \tag{A.17}$$

The final one is for $T_j^{(3)}$, the case of $n = j, m \neq j$. The expression to evaluate is

$$\frac{\partial}{\partial a^L_j}T_j^{(3)} = \frac{\partial}{\partial a^L_j}\sum_{m\neq j}^{N-1}\big[a^1_{m\cdot d+j}a^L_j - a^L_N a^L_m\big]^2, \tag{A.18}$$

and the result is

$$T_j^{(3)} = 2\Big[\sum_{m\neq j}^{N-1}(a^1_{m\cdot d+j}a^L_j - a^L_N a^L_m)(a^1_{m\cdot d+j})\Big] \tag{A.19}$$

And the final expression for the error for wavefunction neurons is

$$\delta^L_j = \Big(a^L_j - y_j + \lambda\sum_i T_j^{(i)}\Big)\Big(1-\tanh^2(z^L_j)\Big), \;\; \forall\, j < N \tag{A.20}$$