Survey on Deep Learning Models for Time Series Data

JCS Kadupitiya Kadupitige

Submitted for the PhD Qualifying Exam

07 14 2020

Advisory Committee

Prof. Geoffrey Fox

Prof. Vikram Jadhao

Prof. Minje Kim

CONTENTS

ABSTRACT

Molecular dynamics (MD) simulations accelerated by high-performance computing (HPC) methods are powerful tools for investigating and extracting the microscopic mechanisms characterizing the properties of soft materials such as self-assembled nanoparticles, virus capsids, confined electrolytes, and polymeric fluids. However, despite the employment of optimal parallelization, the scientific simulations can often take hours or days to furnish accurate information, and deep learning (DL) has the potential to address this critical need. On the other hand, due to advances in hardware performances, richer and high-dimensional scientific computation datasets are continued to generate, and the data-driven modeling approaches coupled with machine learning (ML) and DL have opened new pathways to use these high-dimensional datasets to solve some of the hardest problems in soft-matter research and applications. We discuss several broad and effective deep learning models for temporal data in MD simulations and demonstrated their current success in soft-matter simulation applications. While the use of the DL models in MD simulations has many advantages, we highlight that further research is needed to make the approach more suitable for practical applications due to significant challenges such as complex temporal dependency and chaotic behaviors. Furthermore, we provide a brief insight into how some of these issues can be mitigated through technological advances such as innovation in the NN architecture by fusing physical laws inside a custom-tailored design of NN. Even though our focus is on using DL models to study time series data in MD simulations, we note that our research can share insights on several other applicable across different fields.

1. Introduction

Molecular dynamics (MD) simulations are powerful tools for investigating the microscopic origins of the behavior of a wide range of materials, including soft matter [1]. These simulations have enabled the understanding of microscopic mechanisms underlying the assembly of both biological and synthetic soft materials such as virus capsids [2], [3], confined electrolytes [4], polymeric liquids [5], [6], and self-assembled nanostructures (See Figure 1) [7], [8]. The molecular dynamics (MD) method solves Newton's equation of motion for a system of many particles and evolves the positions, velocities, and forces (generally known as configurations) associated with these particles at each time step. At the heart of the method is the integrator that propagates the configuration of the system one small timestep at a time. While MD simulations are generalizable to study a broad range of phenomena in soft matter, they incur high computational costs in several applications where the computational complexity per time step is proportional to the square of the total number of particles in the simulated system.

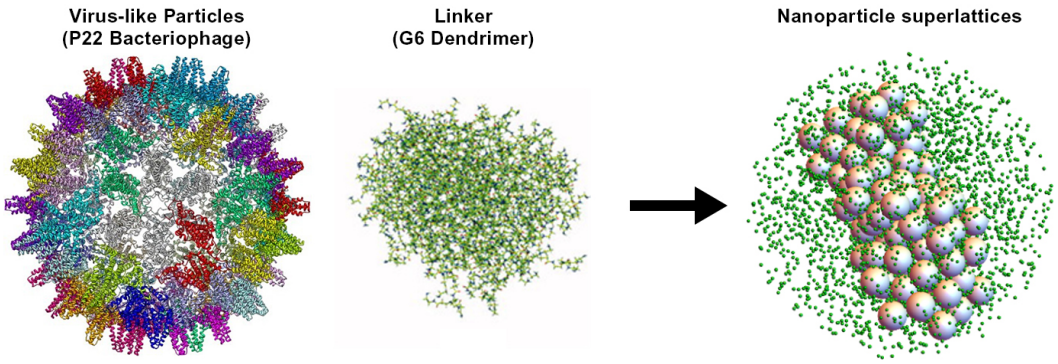

Fig. 1. Example MD simulation model:Functional Nanomaterials, Virus-Linker Superlattices [8]

The high computational costs associated with MD simulations are typically mitigated by employing high-performance computing (HPC) resources and utilizing parallel computing techniques such as OpenMP and MPI. However, despite the employment of the optimal parallelization models suited for the size and complexity of the system, the scientific simulations can often take hours or days to furnish accurate output data and desired information. To expedite MD simulation-driven design of advanced soft materials, it is desirable to rapidly access trends associated with relevant physical quantities that could be learned and predicted with reasonable accuracy based on the history of data generated from earlier simulation runs. Further, in the area of using simulations in education, rapid access to simulation-driven responses to student questions in classroom settings are desirable. In the end, there is a critical need for new approaches to accelerate simulations, leverage past simulations to generate accurate predictions and

expedite the analysis of simulation data to classify material properties. Machine learning (ML) and deep learning (DL) has the potential to address this critical need directly.

On the other hand, as continued advances in hardware performance (Moore's Law), evolution of multicore graphics processing units, and the preponderance of cloud computing, more and more scientific computation data is continued to generate cheaper increasingly, and it is more accessible than ever before. In soft matter simulation applications, the trend towards large data sets is primarily driven by continued advances in hardware performance, while in experiments, it is driven by instrumentation innovations. These large and rich datasets contain essential information that can inform and update the fundamental understanding and guide the engineering and design of advanced soft materials. But these large datasets can be so high-dimensional that the conventional analysis and domain-specific approaches might start to fail. Data-driven modeling approaches coupled with ML and DL have opened new pathways to solve these research and applications, and this is now commonly referred to as data-driven scientific simulations, is under rapid development and attracting many researchers and industries.

Researchers all over the world have attempted to utilize state-of-the-art ML and statistical models on a broad set of tasks that are difficult or even impossible to solve by traditional methods [9]. DL is one of the most effective choices among all employed techniques mainly because of its capacity for multi-level feature learning hierarchically. This has brought lots of significant successes in many areas with either supervised [10], [11], [12], unsupervised [13], [14], [15], semisupervised [9] or reinforcement learning [16], [17], [18] strategies. State of the art deep learning research has solved many difficult problems such as recognizing and distinguishing thousands of human faces at a time [19], [20], understanding, generating and translating human languages almost perfectly and flawlessly [21], [22], [23], and mastering games and beating top human professional players [24] and aiming towards achieving of self-driving cars [25]. Despite the amazing success, DL is improved with proposed new ideas and models updating the state of the art research in this area every year. Moreover, there is plenty of flexible and fast-developing open-source software libraries, such as TensorFlow [26], PyTorch [27], Theano [28], Caffe [29], MXNet [30], Keras [31], and scikit-learn[32] freely available to make it easier for everyone to experience the strength of deep learning in both research and production, leading to a widespread impacts and greater success of deep learning.

Accordingly, there has been a surge in the use of ML to accelerate computational techniques aimed at understanding material phenomena [9]. ML has been used to predict parameters, generate configurations in material simulations, and classify material properties [33], [34], [35], [36], [37], [38], [39], [12], [9], [40], [41]. Motivated by the advancements in ML, DL and by the challenges in employing MD simulations

in research and education, in recent papers [42], [43], we introduced the idea of integrating ML methods with MD simulations to enhance their performance and overall usability. We demonstrated that an artificial neural network (ANN) based regression model, trained on data generated via MD simulations, successfully learns the simulation output features associated with the MD simulation. The ML model instantaneously generated predictions in excellent agreement with results obtained from explicit MD simulations [43]. We also showed that DL models could be an aid to design an adaptive MD-based dynamical optimization framework that updates the simulation timestep and auto-tunes the virtual parameters characterizing the dynamics of ions near polarizable NPs to yield a more stable and efficient simulation [44]. More recently, we have also demonstrated that the design of DL based integrators can propagate the dynamics of an MD simulation with large timesteps [45].

There are many different explanations on why and how deep learning works, either from theoretical means, empirical results, or even thorough some intuitions but the practical reality of deep learning models is not always smooth sailing, and we face many open questions when applying them to MD simulations in the soft matter domain. This problem is especially severe when dealing with temporal data in MD simulations due to complex temporal dependency and chaotic behavior, which imposes several unique and critical challenges that lie in its practical aspects. While the use of the DL approach in MD simulations has many unique benefits, further research is needed to make the approach more suitable for practical applications. One of the significant challenges involves modeling dynamics of systems that exhibit rare event characteristics or involves longer times to yield the low energy configurations due to kinetic bottlenecks that can trap the particles in metastable pathways. In these cases, training a DL model to ”see” a variety of possible distinct phase space explorations in the training dataset or generation of datasets covering a range of physical properties such as interaction potential between particles may prove challenging. Another challenge is scaling these DL approaches to more extensive simulations where the particle population can be varying from a few hundred to thousands of particles.

The DL research in MD simulations of soft materials is primarily application-driven, but innovation is in technology rather than application results [46], [47], [48]. Addressing the issues as mentioned earlier will involve innovation in NN architecture, and smart design of experiments to generate the simulation data. All the new research on NN architecture suggests innovation in the technology is tightly coupled with a domain such as in physics and computer vision [46], [47]. We will investigate the design of DL-based models for predicting the configurations in systems where self-organization into aggregates occurs at longer times, depending on the combination of particle and potential attributes. Explore the training of DL models using configurations produced via simulations in different ensembles (e.g., NVT) might

also be helpful in order to test its accuracy in predicting configuration updates where thermal effects (often stochastic) are incorporated. The NN architecture innovation would be inspired by hierarchical RNNs [49] and alternative DL approaches [50] including transformers [23], physics-informed neural network architectures [46], [47] and its "recurrent" versions. So in contrast, our research plan involves an innovation in NN architecture informed by time-series application and surrogate application, which is driven by Simulation data such as MD simulation, Monte Carlo simulations.

2. Background review

MD simulations serve as essential tools for understanding diverse self-assembly phenomena in nanoscale materials [51], [52], [7], predicting material behavior in practical applications [53], and isolating interesting regions of parameter space for experimental exploration [54]. Recent years have seen an unprecedented rise in the use of machine learning (ML) and deep learning (DL) to enhance the performance of simulations of materials and analyze the data they generate [9], [55], [56], [57], [58], [34], [37], [33], [59], [35], [40], [39], [44], [44], [12], [60]. ML applications in MD simulations of materials include accelerating the sampling of systems with rugged free-energy landscapes [40], generating new configuration updates in latent-space (low-dimensional) variables in ab initio MD simulations of nuclei-electron systems [39], auto-tuning timestep in MD simulations of ions near nanomembranes [44], classifying assembly landscapes employing MD-generated particle trajectories [12], and surrogate modeling and uncertainty quantification [61], [62], [63], [61], [62], [63]. We also note other related work that focuses on using DL to learn the physical laws or properties of simple physical systems and respect conservation laws [46], [47], [64], [65], [66], [67], [68], [69], [70]. Similarly, many classic deep neural networks can be seen as approximations to differential equations and many focused on using DL to learn differential equations and replicate numerical integrators [46], [47], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [70], [74], [75], [76]. So, all the literature of temporal data research in MD simulations can be broadly classified into three main topics, such as Simulation surrogates using machine learning, physics informed neural networks, and neural ordinary differential equations.

- Simulation surrogates using machine learning: The use of DL to derive networks that produce“surrogates” of large-scale simulations, including MD simulations [77], [42], [60], [43], [78]
- Physics Informed Neural Networks: The use of laws of physics to parameterize or constrain the deep neural network architecture to solve many differential equations [46], [47], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [70]
- Neural Ordinary Differential Equations: The use of modern differential equation solvers or integrators to enhance deep neural networks [79], [80], [81], [48], [82], [83], [64], [84].

A. Simulation surrogates using machine learning

In recent years, the ML methods which have been applied to enhance computational techniques can be thought as surrogates to scientific simulations. These research projects aimed at understanding material phenomena; ML has been used to predict parameters, generate configurations in material simulations,

and classify material properties [33], [34], [35], [36], [37], [38], [39], [12], [9], [40], [85], [44], [86], [41], [87], [88].

Machine Learning (ML) abstractions for classification tasks and tuning have been extensively employed in the performance enhancement of scientific simulation frameworks. Denil et al. [89] used artificial neural network (ANN) and convolutional deep learning neural network (NN) to predict the parameters found in image classification tasks with an accuracy of 95%. Yigitbasi et al. [90] employed ML-based auto-tuning for diverse MapReduce applications and cluster configurations in their simulation framework. Their work showed that support vector regression (SVR) exhibits good accuracy while being computationally efficient for performance modelling of MapReduce applications. Fu et al. [35] employed ANN classification model to select efficient updates to accelerate Monte Carlo simulations of classical Ising spin models near the phase transition. More recently, ML has been used to predict specific outcomes (the dissociation timescale of compounds) of ab initio MD simulations by bypassing the time evolution of the particle trajectories [77]. Also, convolutional neural network-based ML "emulators" have been introduced recently to predict output functions (e.g., power spectrum) of simulations in biogeochemistry and other domains [78].

Regression-based prediction schemes have also been employed in different domain areas [91], [92], [93], [94], [95], [96], [97]. Christine et al. [91] used random forest regression algorithm to predict host tropism of influenza A virus proteins with an accuracy above 96%. Similarly, an ensemble of regression trees was employed to perform face alignment for real-time applications (in one millisecond) by [92]. SVR has been used for wind speed prediction by [94]. ANN-based regression has been studied by [96] to yield short term load prediction of electrical power systems based on wind power forecasting. Yadav et al. [97] have employed ANN-based regression for forecasting solar radiation. Matthew et al. [33] applied a simple feedforward ANN to discover interesting areas of parameter space corresponding to crystal formation in the self-assembly of colloidal building blocks. Botu et al. [39] employed kernel ridge regression (KRR) to accelerate the ab initio MD method for nuclei-electron systems by learning the selection of probable configurations in MD simulations which enabled bypassing explicit simulations for several steps. Balachandran et al. [95] have used SVR to create an adaptive ML model to aid the design of new materials with desired elastic properties and enhanced long-term performance using a minimum number of iterations. Recent work has also focused on adaptive ensemble simulations to enhance the computational efficiency of biomolecular simulations [98]. We also note the development of auto-tuning technology for high-performance computing applications to reduce execution time and enhance programmer productivity [99]. Here, auto-tuning relates to the automatic generation of a search space of possible kernels for a computational task to identify the best possible kernel, with recent work involving

the use of ML-based approaches for identifying the search space [100].

However, relative to "hard" condensed matter systems and other scientific simulations, a survey of literature finds far fewer applications of ML in the area of MD simulations of soft materials [9], [44]. JCS et al. have used ML to design an adaptive MD-based dynamical optimization framework that updates the simulation timestep and auto-tunes the virtual parameters characterizing the dynamics of ions near polarizable NPs to yield a more stable and efficient simulation [44], [41]. Related work in the area of adapting timestep in a simulation has involved using analytical approaches to multiple timestep integration [101], [102]. JCS et al. also had introduced a "machine learning surrogate" for MD simulations of soft-matter systems using neural network-based regression model which successfully learns nearly all the interesting features associated with the output ionic density profiles over a broad range of ionic system parameters [42], [43].

B. Physics Informed Neural Networks

We have witnessed a dramatic rise in research for Physics informed neural networks, which illustrates a deep connection between deep neural networks and physical systems explained with differential equations [71], [72], [73], [70], [46], [64], [103], [69], [47]. Researchers have demonstrated that many differential equations (linear, elliptical, non-linear, and even stochastic PDEs) can be solved with the aid of deep neural network with or without closely binding neural network architecture to physical laws, which are described through these differential equations [71], [72], [73], [70].

There are many previous work have sought to furnish deep learning models with better physics priors in specific domains such as molecular dynamics [67], [104], [68], [105], [106], quantum mechanics [107] and robotics [108]. Raissi et al. introduces Physics Informed Neural Networks (PINNs) and demonstrates how to use PINNs to solve several classical PDEs [71]. Liu et al. have addressed the problem of stochastic differential equations (SDEs) but uses traditional generative adversarial neural (GANs) network architecture to solve the SDEs [72]. Yang et al. have conducted a large study that applies PINNs using GANs to solve a set of large scale PDE problems and also provided uncertainty quantification of the models [73]. Though these works have focused on Discovering physical concepts with neural networks but relied on traditional neural networks in a supervised fashion without any architectural or loss function changes and did not explicitly used parameterization techniques with physics laws such as Hamiltonians or Lagrangian [109], [110], [111].

We also note that the related work that focuses on using DL to learn the Hamiltonian of simple physical systems and respect conservation laws in an unsupervised manner [46], [47], [64], [65], [66], [67], [68], [69], [70]. Here, instead of performing explicit time evolution, neural networks observe the positions

and momenta (inputs) and produce the Hamiltonian as output, in some cases leveraging the information that the partial derivatives of the output with respect to inputs are the time derivatives of the inputs [46]. Hamiltonian neural networks [46], Lagrangian neural networks [47], and symplectic RNNs [64] are examples of these DL approaches that have been shown to learn the physics of simple systems such as spring-mass problems in 1D. The Hamiltonian Neural Networks (HNN) is a deep neural network model inspired by Hamiltonian mechanics to train models that learn and respect exact conservation laws (exhibits in a physical system) in an unsupervised manner [46]. Similarly, the Lagrangian Neural Networks (LNN) is a Deep Neural Network model inspired by Lagrangian mechanics to train models that learn and respect exact conservation laws and Lagrangian functions in an unsupervised manner [47].

C. Neural Ordinary Differential Equations

Neural ordinary differential equations illustrate how traditional and modern differential equation solvers or integrators informed the design, accuracy, and memory footprint of deep neural networks and simplify the neural networks[48]. These networks are inspired by the deep residual networks and recurrent neural network decoders which can be seen as an Euler discretization of a continuous transformation as $h_{t+1} = h_t + f(h_t, \theta_t)$ [112], [113]. So the interesting idea is that as we add more layers and take smaller steps (in the limit where $dt \to 0$), we parameterize the continuous dynamics of hidden units using an ordinary differential equation (ODE) specified by a neural network: $\frac{dh(t)}{dt} = f(h(t), t, \theta)$ so that the input layer $h(0)$ and the output layer $h(T)$ becomes the solution to this initial value problem at time T [114], [48]. These ODE nets usually use a modern and accurate differential equation solver (black-box) to set the outputs to the network and compute gradients using the adjoint sensitivity method [48].

Many research papers focused on using DL to learn differential equations and replicate numerical integrators [79], [80], [81], [48], [82], [83], [64], [84]. Generally, these integrators have been derived to operate with the time discretization associated with the baseline integrator timestep dt. Their applications have been demonstrated on relatively simple 1D systems governed by ordinary differential equations. Few studies have probed the DL applications to learn partial differential equations [81]. Few other recent approaches try to improve the accuracy of integrators at a higher dt [115], [80]. Shen et al. [115] have developed an artificial neural network (ANN) based Euler method for numerical integration with high dt by approximating the local truncation error with deep neural networks to obtain accurate solutions with dt up to 200 times larger than the baseline Euler approach. As applications, they have shown the approach to work on 1D systems and relatively simple 2-body problems. For the 1D problems, a 100 fold increase in the timestep (from $2\times$ to $200\times$) led to an increase in the prediction error by over three orders of magnitude. More recently, JCS et al. have demonstrated that RNN-based integrators can limit

the rise in the error with increasing timestep ($> 1000\times$ the baseline) to within an order of magnitude, even for the complex 3D systems [116], [45], [117].

3. DATASETS IN MD SIMULATIONS

The classical MD simulation is one of the most well-known simulation technique which is based on Newton's equations of motion and relies on numerical integrators such as velocity verlet to solve them. These simulations often simulate systems consist of different types of particles [118], [119], [120], [121], [122], [123]. The configuration space of the simulation consists of many physical properties of these particles, such as position, velocity, and force vectors. Using Newton's equations of motion and a numerical integrator, MD simulations propagate the dynamics (configuration space) of the system from one state to the next state in a step-by-step process called verlet integration. A traditional MD simulation will run a different number of computational steps from a few million to billions depending on the system it is simulating. These simulations mainly generate two types of data, such as the time-series data of the configuration space and several statistical quantities derived using the time series data. Now we will describe two such datasets in the following sections.

A. Time series data of a system with many particles interacting via Lennard-Jones (LJ) potential

For this 3D system, the interaction potential energy between any two particles is given by the LJ potential:

$$U(r) = 4\epsilon \left(\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^{6} \right) + 0.0163\epsilon \quad \text{for } r \leq 2.5. \tag{1}$$

For $r > 2.5$, U is 0. We prepared two different types of simulation boxes to generate the datasets: cubic box with periodic boundary conditions and spherical box with reflective boundary. Figure 2 shows snapshots of these MD simulations. In each box, we performed simulations with $N = 3, 8$, and 16 particles. Each of these simulations was created as a separate dataset that yielded six different datasets. The dataset is generated by varying two input parameters: the mass of the particle $m \in [1, 10]$, and the initial position of the particles (x_0, y_0, z_0) with each Cartesian coordinate chosen between -3.0 and 3.0. The well depth was held constant to $\epsilon = 1$ in creating the training dataset. Initial velocities were chosen to be zero. The parameter sweep generated a dataset of 5000 simulations for each of the aforementioned cases (or 30,000 simulations in total). Each simulation in this dataset generated 2 million configurations (position, velocity, and force vectors).

B. Ion density profile data of ions in confinement simulation

This data set was created by varying five input parameters characterizing the ionic system: confinement length h, salt (electrolyte) concentration c, positive ion valency z_p, negative ion valency z_n, and the ion diameter d. All ions are assumed to have the same diameter; in general, oppositely charged ions have

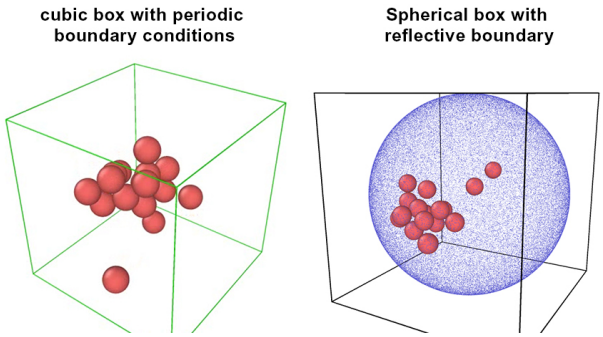

Fig. 2. Simulation snapshots of a MD simulation with 16 particles interacting via Lennard-Jones (LJ).

different sizes. The range of each parameter is selected as follows: $h \in (3.0, 4.0)$ nm, $c \in (0.3, 0.9)$ M, $z_p \in 1, 2, 3$, $z_n \in -1$, and $d \in (0.5, 0.75)$ nm. The salt concentration is defined as the number of negative ions per unit volume [42], [44], [124]. We note that the system temperature is another important input parameter. In this initial application, the temperature is held fixed, and we have employed data generated at room temperature (298 K). The converged distribution for positive ions is selected as the output. This dataset was generated by sweeping over a few discrete values for each of the input parameters to create and run 6,864 MD simulations utilizing HPC resources. On average, each MD simulation was performed in the NVT (canonical) ensemble for over ≈ 5 nanoseconds of ionic dynamics and took 4200 CPU hours (≈ 36 minutes per simulation with MPI/OpenMP parallelization). The dataset creation took approximately 25 days, including the queue wait times on the Indiana University BigRed2 supercomputing cluster. The data associated with the ionic density profiles (dataset relevant to our investigation), was over 2 GB. Each MD simulation produces positive ion distribution characterized by ionic density measured at ≈ 300 positions as output, as shown in Figure 3. For simplicity, using the symmetry of ionic density around the confinement center $z = 0$ (resulting from neutral surfaces), approximately half of the 300 points are selected as the output parameters.

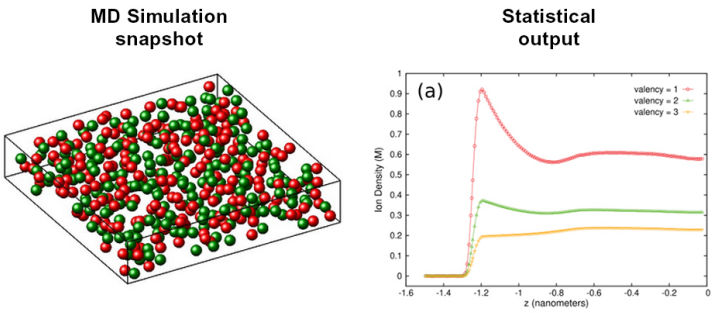

Fig. 3. Simulation snapshot and output of ions in confinement simulation

4. Temporal Data Analytic Models

A. Basic Temporal Data Analytic Models

Temporal data analysis is the process to model and explains data points which have time-dependency. In this section, we review several standard statistical and machine/deep learning models for temporal data related to scientific simulations, which are time series classification, prediction and forecasting.

1) Auto regressive integrated moving average model: The autoregressive integrated moving average (ARIMA) model or often denoted as ARIMA (p, d, q) is a popular model fitting to time series data to understand the data better or to predict future points in the series [125]. It can be seen as the combination of the auto-regression (AR), moving average (MA), and the integrated (I) and is designed for non-stationary data. The p, d, and q in the notation are the parameters for the order of the AR model, the number of differencing parts, and the order of the MA model, respectively. ARIMA attempts to forecast x_t, the value of the time series at time t, from its past value(s). Firstly the AR part indicates that the evolving variable of interest is regressed on its own lagged (i.e., prior) values, and the AR model of order p (i.e., AR(p)) can be written as:

$$x_t = \sum_{i=1}^{p} \Phi_i x_{t-i} + \epsilon_t + c$$

where Φ_i are the parameters, ϵ_t are the error terms, and c is a constant. Secondly the MA part indicates that the regression error is actually a linear combination of error terms whose values occured contemporaneously and at various times in the past. The MA model of order q (i.e., MA(q)) can be written as:

$$x_t = \sum_{i=1}^{p} \Theta_i \epsilon_{t-i} + \epsilon_t + c$$

where Θ_i are the parameters of the MA model. Finally, the I part indicates that the data values have been replaced with the difference between their values and the previous values. By combining all three parts together, the full model of ARIMA(p, d, q) can be written as:

$$\left(1-\sum_{i=1}^{p}\Phi_i\mathbb{L}^i\right)(1-\mathbb{L})^d x_t = c + \left(1+\sum_{i=1}^{q}\Theta_i\mathbb{L}^i\right)\epsilon_t$$

where $\mathbb{L}$ is the lag operator defined as $\mathbb{L}^k x_t = x_{t-k}$. One wat to chose a proper order is to minimize the Akaike information criterion (AIC) or Bayesian information criterion (BIC) [126]. These models can be straightforwardly generalized from univariate to multivariate and lead to the vector autoregression (VAR) models.

2) Kalman filters (KF): Kalman filters (KF) is one kind of state space models for time series data estimation which assumes the linear dynamic properties of the system and Gaussian (white) noise in the temporal data observations [127]. To be more specific, the KF model for states x, control vectors u and observations z can be represented by the following two equations:

$$\begin{aligned} x_t &= Ax_{t-1} + Bu_t + w_k \\ z_t &= Hx_t + v_k \end{aligned} \tag{2}$$

where A, B, H are the state transition model, the control input model, and the emission model, respectively. $w \sim \mathcal{N}(0, Q_t)$ and $v \sim \mathcal{N}(0, R_t)$ are the process noise and the observation noise, respectively and both w and v from a zero mean multivariate (Q_t, R_t) normal distributions. The Kalman filters can be fitted by using five update equations. The first two equations are for the time prediction (update) as follows:

$$\begin{aligned} \hat{x}_t^- &= A\hat{x}_{t-1}^- + Bu_{t-1} \\ P_t^- &= AP_{t-1}A^T + Q \end{aligned} \tag{3}$$

The following three equations are used as the measurement correction steps:

$$\begin{aligned} K_t &= P_t^- H^T (HP_t^- H + R)^{-1} \\ \hat{x}_t &= \hat{x}_t^- + K_t(z_t - H\hat{x}_t^-) \\ P_t &= (I - K_t H)P_t^- \end{aligned} \tag{4}$$

Here, P is the error covariance, K is the Kalman gain, $\star^{-}$ and $\hat{\star}$ denote the prior and estimate of the variable $\star$, respectively. By taking the update equations iteratively, Kalman filters converge to the relevant values after enough N steps.

3) Dynamic time warping (DTW): Calculating similarity between two temporal sequences (which may or may not have different lengths), is a fundamental problem in time series data analysis and serves as a basis requirement for many other tasks. Dynamic time warping (DTW) is one of the most popular similarity measures used in temporal sequence analysis, which aims to target the speed difference between the two time sequences [128]. DTW attempts to find the optimal alignment between two temporal sequences $X = (x_1, \ldots, x_T)$ and $Y = (y_1, \ldots, y_{T'})$ with certain restrictions and rules. First, the two sequences X and Y should be aligned (x_1 and y_1 to x_T and $y_{T'}$). Second, each x_t and $y_{t'}$ must be included in the alignment, and the order of indices should be non-decreasing. If we use $d_{t,t'}$ to indicate the (partial) dynamic time warping distance between $(x_1, \ldots, x_T)$ and $(y_1, \ldots, y_{T'})$, then DTW can be calculated by using dynamic programming as:

$$d_{t,t'} = \min\{d_{t-1,t'}, d_{t,t'-1}, d_{t-1,t'-1}\} + ||x_t - y_{t'}||$$

DTW can also be generalized to multivariate time series in practice by using the euclidean distance as the local distance metric for all (x_t, $y_{t'}$) [129].

B. Deep Learning Models for Temporal Data

Deep learning models, or deep neural networks, have become successful approaches for automated extraction of complex data representations for various applications, including temporal data modeling. Deep learning models consist of layered and hierarchical architectures of neurons for learning and understanding the hidden relationships in data [130]. The hierarchical learning architecture is motivated by artificial intelligence emulating the deep and layered learning process of the primary sensorial areas of the neocortex in the human brain, which automatically extracts features and abstractions from the underlying data [131]. Each neuron in a deep neural network receives one or more inputs and sums them to produce one or many outputs. These neurons in the hidden layers are consists of weights and activation function. The weights are learned from a training process performed on the available data, and the activation function introduces a non-linear capability into the hidden layer. The deep learning models thus can understand the complicated relationship between multidimensional input data and output temporal data through the hierarchical non-linear architecture exhibits in deep neural networks.

1) Deep feed-forward neural network (DNN): Deep feed-forward neural network (DNN) is one of the most basic deep neural models and it is composed with multiple non-linear transformation layers to extract features from the data and possibly one prediction layer on the top to solve either a classification or a regression task [132]. The output from the each layer is fed to the next layer as inputs. For a DNN model with n layers (i.e., n-1 hidden layers and one final output layer), the input vector for the n-th layer is denoted as $x^{[n]} \in \mathbb{R}^{D[n]}$, and the transformation of each layer n can be written as:

$$x^{[n+1]} = f^{[n]}(x^{[n]}) = f^{[n]}(W^{[n]}x^{[n]} + b^{[n]})$$

where $W^{[n]}$ and $b^{[n]}$ are respectively the weight matrix and bias vector of layer n, and $f^{[n]}$ is a nonlinear activation function, which usually takes one of $sigmoid$ ($logistic\ sigmoid$, $\sigma(x) = \frac{1}{1+exp(-x)}$), $tanh$ ($hyperbolic\ tangent$, $\tanh(x) = 2\sigma(2x)-1$), and ReLU ($rectified\ linear\ unit$, $\mathrm{ReLU}(x) = max(0, x)$) [133]. These DNNs are trained using an optimization process called backpropagation that requires a loss function to calculate the error between model prediction and the true value and update the respective weight matrix and bias vector of all the layers. The Maximum likelihood estimation provides the necessary details for choosing a proper loss function based on the particular application and every machine learning model in general. When it come to DNNs, cross-entropy and mean squared error (MSE) are the two main types of loss functions which are being used across varies applications. The application of DNNs can be broadly classified for three different types such as regression problems, binary classification problems, and multi-class classification problems which uses MSE, binary cross-entropy and cross-entropy as the loss functions respectively [132], [134]. Taking the binary classification problem as an example, we can train DNN and learn the weights and biases by optimizing binary cross-entropy loss function during training as following equation:

$$Loss = -\sum_{i=1}^{N} \left(y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right)$$

Where $y^{(i)}$ is the binary label for i-th sample in the training dataset (total size of S samples), and $\hat{y}^{(i)}$) is the output of the DNN model (final layer output after the activation). Note that the ideas of using non-linear activation functions, training by backpropagation, and learning layer-wise features (representations) are not only used for simple DNN models but almost in all other deep learning models. We can directly use DNN for applications with temporal data as discussed the literature review section. Example DNN architecture is shown in Figure 4 [86], [44], [43].

2) Convolutional neural network (CNN): Another popular primary neural network is the convolutional neural network (CNN), which is good at capturing local structure and commonly applied in computer

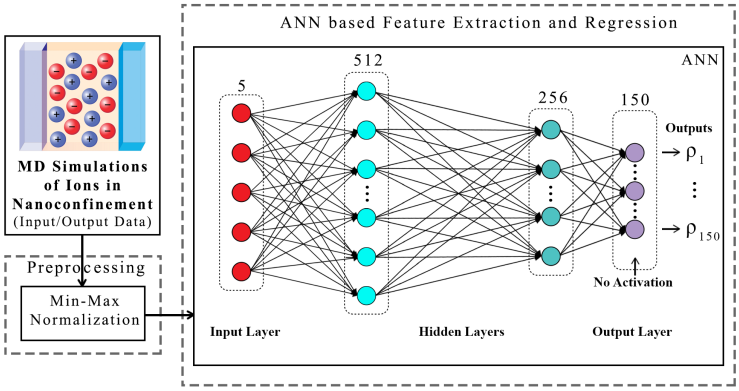

Fig. 4. Architecture of a DNN model

vision applications where the data is either in the form of images or videos [135]. CNN models are build using a combination of three main types of layers: convolutional layer, pooling layer, and fully connected layer. The convectional layer is the core building block of CNN models, which attempts to learn the spatially activated map of features via lots of filters over the inputs with parameter sharing. The pooling layer is used to reduce the size of the feature representations mainly, and the number of parameters learned in the convectional layer. The fully connected layer is just a regular layer which is similar to DNNs and connects to all outputs from its previous layer. A variety of different CNN architectures made up of the mentioned main three types of layers and are proposed to solve different problems. A traditional CNN architecture diagram is shown in Figure 5. There are two ways we can use CNN on temporal data, such as using traditional two-dimensional convolutional operation over the temporal dimension and the input feature per time sample dimensions and using one-dimensional convolutional operation only over the temporal dimension [136], [137].

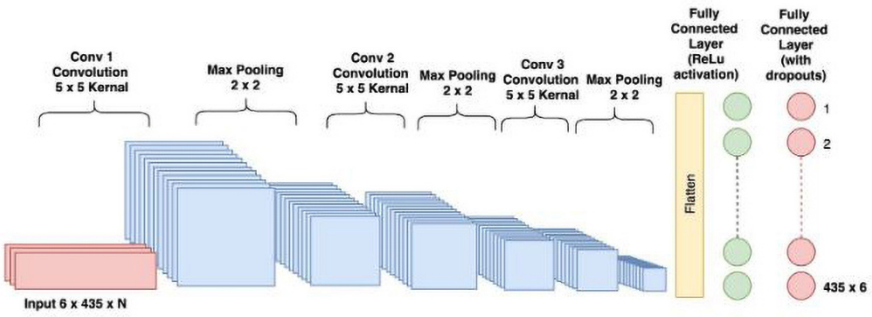

Fig. 5. Architecture of a CNN model

3) Recurrent neural networks (RNN): Recurrent neural networks (RNN) are a class of neural networks that allow previous outputs to be used as inputs while having hidden states. RNNs can use their internal state (memory) to process a variable length sequences of inputs. The most used versions of basic RNNs are described as follows:

a) Vanilla recurrent neural network (RNN): In order to handle sequential or temporal data of arbitrary length and capture temporal information from the data, recurrent neural network (RNN) models are widely used [138]. Unlike DNNs, RNN performs the same operation at each time step of a sequence of inputs, and feed the output to the next time step as part of the input. Because of that, the RNN models can memorize what they have seen before and benefit from shared model weights for all time steps. For example, at time step t, the output of a vanilla RNN layer can be calculated as:

$$h_t = f(W_x x_t + U_h h_{t-1} + b)$$

Where W_x, U_h, and b are the network parameters, $f(.)$ is a non-linear function, and the initial value of the hidden state h_0 is usually set to 0. Like DNN and CNN models, RNN models can also have multiple stacked hidden layers at each time-step. Finally, additional feed-forward layers may also be applied on top of the last output h_T or each output h_t of the final RNN layer. Backpropagation through time algorithm is usually used to train RNN models [139].

b) Long short-term memory (LSTM): In order to capture complex long temporal dependency and avoid vanishing gradient problems, modified RNN models have been proposed with state-of-the-art performance. Long Short-Term Memory (LSTM) is one of the earliest and most commonly used RNN models and there are several architectures of LSTM units [140]. A common architecture is composed of a cell (the memory part of the LSTM unit) and three "regulators", usually called gates, that regulate the flow of information inside the LSTM unit. An input gate (i_t) controls how much new information is added from the present input (x_t) and past hidden state (h_{t-1}) to our present cell state (c_t). A forget gate (f_t) decides what is removed or retained and carried forward to the current cell state (c_t) from the previous cell state (c_{t-1}). An output gate (o_t) decides what to output as the current hidden state (h_t) from the current cell state (c_t). These gated units work as a fully-connected neural network layer that receives the weighted sums of the input x_t and the previous hidden state h_{t-1} as its input. Because of

that, LSTM are also often referred to as FC-LSTM [141]. The LSTM formulation can be expressed as:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c_t} = \sigma_h(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t}$$

$$h_t = o_t \circ \sigma_h(c_t).$$

Here, $x_t \in \mathbf{R}^d$ is the input vector to the LSTM unit, $f_t \in \mathbf{R}^h$ is the forget gate's activation vector, $i_t \in \mathbf{R}^h$ is the input gate's activation vector, $o_t \in \mathbf{R}^h$ is the output gate's activation vector, $h_t \in \mathbf{R}^h$ is the hidden state vector also known as the output vector of the LSTM unit, $c_t \in \mathbf{R}^h$ is the cell state vector, and $\circ$ is the Hadamard product operator. $W \in \mathbf{R}^{h \times d}$ and $U \in \mathbf{R}^{h \times h}$ are the weight matrices and $b \in \mathbf{R}^h$ are the bias vector parameters which need to be learned during training. σ_g and σ_h represent sigmoid function and hyperbolic tangent functions respectively. d and h refer to the number of input features and the number of hidden units respectively. The initial values of h_0 and c_0 are both usually set to be 0. A traditional LSTM architecture diagram used in temporal data predictions, is shown in Figure 6 [116].

c) Extending LSTM with Peephole Connections: Since the invention of LSTM network, a couple of related implementations have been introduced to further improve the performance. One example extends the LSTM by having so called peephole connections [142], [143]. In traditional LSTM each gate receives connections from the input units (x_t) and the outputs of all cells (h_{t-1}), but there is no direct connection from the internal cell state (c_t). When the output gate is closed, there might be a situation of lack of essential information to the gates which may harm network performance. So as a solution the peephole connection from the cell state (c_t) has been introduced to all three gates. These additional connections have the purpose that each gating unit has a direct access to the previous memory cell state C_{t-1}. The motivation for this is to allow the units to learn when to open or close their gates based on the cell state, in order to learn more precise timings. The LSTM with Peephole Connections can be expressed as:

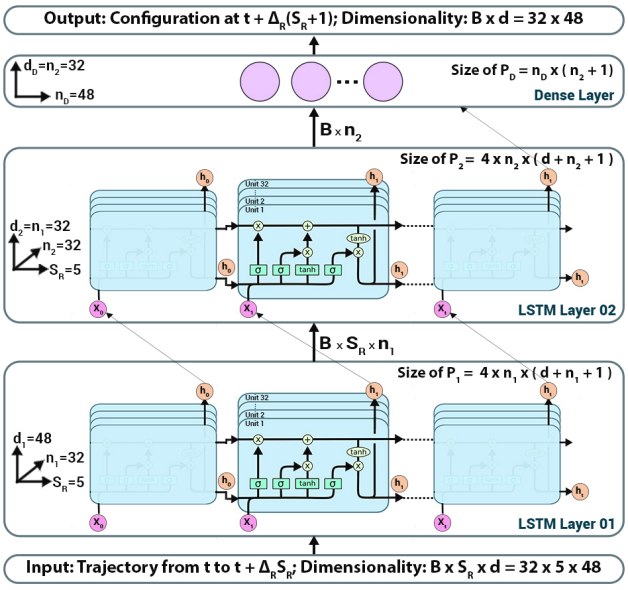

Fig. 6. Architecture of a LSTM model used in temporal data prediction model: LSTM integrator designed to perform a single timestep (Δ_R) evolution of an N particle system characterized by features of size d. Input system and LSTM parameters are shown for 16 particles in 3D.

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + W_{fc} \circ c_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + W_{ic} \circ c_{t-1} + b_i)$$

$$\tilde{c_t} = \sigma_h(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c_t}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + W_{oc} \circ c_t + b_o)$$

$$h_t = o_t \circ \sigma_h(c_t).$$

Here, $W_{*c} \in \mathbf{R}^h$ are the weight vectors of $*$ gate (can be any of the three gates) for the peephole connections which need to be learned during training. All other symbols are defined in the above LSTM section. Note that the output gate takes the current cell state as a peephole connection. An LSTM cell architecture diagram with peephole connections is shown in Figure 7

d) Gated recurrent unit (GRU): Gated recurrent unit (GRU) is another popular RNN model which has a simpler architecture compared to LSTM and has been shown to achieve similar state-of-the-art performance compared to LSTM models for modeling temporal/sequential data [144], [145]. GRU attempts to solve the vanishing gradient problem of a standard RNN, similar to LSTMs but only using

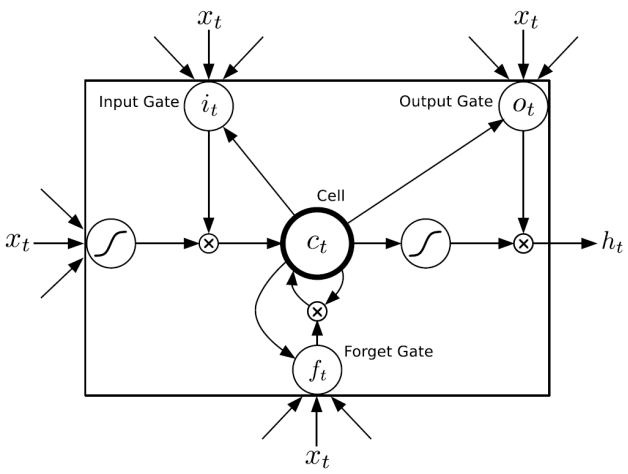

Fig. 7. Architecture of a Long Short-term Memory Cell with Peephole connections [141]

two gates (three gate in LSTM) so-called, update gate and reset gate. These two gates control what information should be passed to the output and what information is irrelevant to the final output. The weight associated with these gates can be trained to keep past information without loosing it through time. The update functions of GRU is shown as:

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$\tilde{h_t} = \sigma_h(W_c x_t + U_c(r_t \circ h_{t-1}) + b_c)$$

$$h_t = z_t \circ \tilde{h_t} + (1 - z_t) \circ h_{t-1}$$

Here, $x_t \in \mathbf{R}^d$ is the input vector to the GRU unit, $z_t \in \mathbf{R}^h$ is the update gate's activation vector, $r_t \in \mathbf{R}^h$ is the reset gate's activation vector, $h_t \in \mathbf{R}^h$ is the hidden state vector also known as the output vector of the GRU unit, $\tilde{h_t} \in \mathbf{R}^h$ is the cell state vector, and $\circ$ is the Hadamard product operator. $W \in \mathbf{R}^{h \times d}$ and $U \in \mathbf{R}^{h \times h}$ are the weight matrices, and $b \in \mathbf{R}^h$ are the bias vector parameters which need to be learned during training. σ_g and σ_h represent sigmoid function and hyperbolic tangent functions respectively. d and h refer to the number of input features and the number of hidden units, respectively. The initial value of h_0 is usually set to be 0.

4) Deep Markov model (DMM): Deep Markov model (DMM) is one of the most influential of models for understanding the hidden, latent variables in temporal data evolution. Unlike other deep learning models introduced above, DMM is a class of generative models which empower Gaussian state-space

models to leverage the representational capacity of deep learning models [146], [147]. Deep Markov models keep the Markov property as hidden Markov models and replace the classic linear emission and transition distributions by multiple deep neural network layers (DNNs). Deep Markov models can be learnt by stochastic gradient ascent on a variational lower bound of the likelihood.

5) Transformers: RNN networks are generally slower as all the temporal input sequence data needs to be passed sequentially or serially one after the other as it needs inputs of the previous state to make any operations on the current state. Such sequential flow does not make use of today's GPUs and other parallelization techniques very well, which are designed for parallel computation. LSTM networks were introduced to capture long temporal dependency and avoid vanishing gradient problems, but they are even slower than vanilla RNN due to architectural complexity. The transformer neural network architecture was introduced in 2017 to address this issue so that the input sequence can be passed in parallel [23]. The network employs an encoder-decoder architecture much like encoder-decoder models with RNNs. The first transformer network is used to perform machine translation from English sentences to French sentences.

The first component of the encoder is an input embedding which converts inputs features to vectors. The next component adds positional encoding to the word vectors, so it produces a vector which has the information about the position in the sequence (in terms of distances). The original architecture uses a sine and cosine function to generate the position encoding vectors, but it could be any reasonable function [23]. Above two layers converts an input English sentence to an embedding to represent the meaning and then added with the positional vectors which capture the context of the words in the sentence. These word vectors then processed inside Multi-head attention layer where it computes how relevant is one word in a sentence (English) relevant to other words in the same sentence. This is done through mapping a query and a set of key-value pairs to an output and the output is computed as a weighted sum (hence, the multi head attention) of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key. The next component of the encoder block is a feed-forward net which is a simple DNN which is applied to every one of the attention vectors and transform the attention vectors into a form that is digestible by the next encoder blocks or decoder blocks. The decoder architecture follows the same output embedding, positional encoding and multi-head attention layers, except that it has a masked out multi-head attention for the French language. While generating the next French word, the network uses all the words from the English sentence; however, it only uses the previous words from the French sentence. This masking allows the network to learn the language relationship between English-French and use that to predict the next french word rather than just learning

to predict the next french word from past french words. Then the output vectors (only the past french word vectors) of the masked out multi-head attention and the output vectors from the encoder block (all English word vectors) is passed to another attention block where the relationship between two languages are encapsulated. Next, the output vectors goes a feed-forward net similar to the encoder following a linear layer which expands the dimensions up to the number of words in the translated language (French in this case). Finally, a soft-max activation layer is applied to get a probability distribution resulting in a final word (corresponding to the highest probability) in the French language. So this process should be applied iteratively until the token for an end of the sentence is generated as a prediction. We can apply the transformer architecture for temporal data prediction by removing the embedding layers and the softmax layer at the end. An architecture diagram of transformer customized for temporal data prediction is shown in Figure 8.

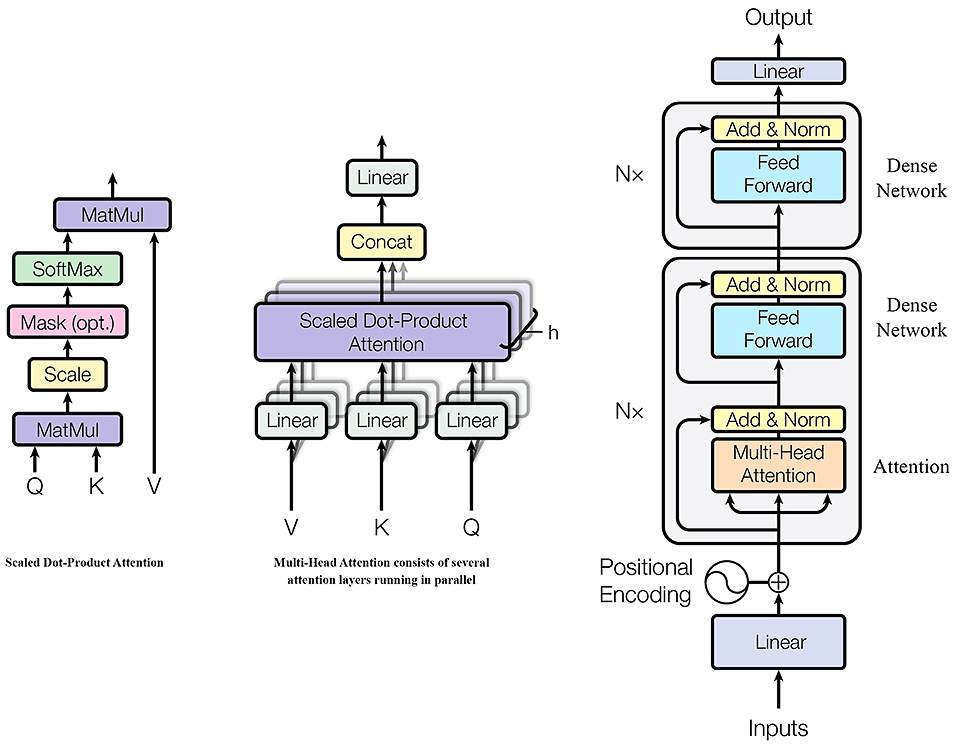

Fig. 8. Architecture of a Transformer network for time-series regression [23]

6) Combination of deep learning models: An essential advantage of using deep learning models to learn temporal data is the flexibility in structure and model architecture with end-to-end training, which allows us to combine different types of layers and networks into one model [148]. For example, we can use a CNN model to look at the structural information, RNN model to take the temporal inputs and a

DNN model to take the non-temporal inputs, and add additional layers that take the output of the outputs from all the combined networks. All components of the entire model (a combination of a few networks) can be trained jointly by backpropagation. One famous example of this type of network is CNN-LSTM, where we first pass all the inputs through a CNN network followed by an LSTM network to reduce the input features per temporal dimension [149]. To learn the temporal features, CNN behaves like a spatial feature extractor and feeds the output to LSTM layers. Another famous architecture is ConvLSTM, where we introduce convolution operation inside the LSTM cell to replace the traditional matrix multiplication calculation of the input [150]. So, ConvLSTM and CNN LSTM full fills the same functionally where ConvLSTM has the convolution embedded in the architecture while CNN-LSTM concatenates the types of networks together externally.

a) Fully-Convolutional LSTM (ConvLSTM): As highlighted earlier one of the major drawback of traditional LSTM (FC-LSTM) is that it doesn't encode any spatial information in its usage of full connections in input-to-state and state-to-state transitions. ConvLSTM address this by converting all the inputs (x_t), hidden states (h_t) and cell outputs (c_t) to 3D tensors ($\mathcal{X}_t$, $\mathcal{H}_t$, $\mathcal{C}_t$). The gates f_t, i_t, o_t of the ConvLSTM are also 3D tensors whose last two dimensions are spatial dimensions (rows and columns) and the inputs and states could be thought as vectors standing on a spatial grid. The ConvLSTM uses convolution operator (instead of matrix multiplication in traditional LSTM) in the state-to-state and input-to-state transitions to determine the future state of a cell in the grid by the inputs and past states of its local neighbors. The ConvLSTM formulation can be expressed as:

$$
\begin{aligned}
f_t &= \sigma_g(W_f * \mathcal{X}_t + U_f * \mathcal{H}_{t-1} + W_{fc} \circ \mathcal{C}_{t-1} + b_f) \\
i_t &= \sigma_g(W_i * \mathcal{X}_t + U_i * \mathcal{H}_{t-1} + W_{ic} \circ \mathcal{C}_{t-1} + b_i) \\
\tilde{\mathcal{C}}_t &= \sigma_h(W_c * \mathcal{X}_t + U_c * \mathcal{H}_{t-1} + b_c) \\
\mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tilde{\mathcal{C}}_t \\
o_t &= \sigma_g(W_o * \mathcal{X}_t + U_o * \mathcal{H}_{t-1} + W_{oc} \circ \mathcal{C}_t + b_o) \\
\mathcal{H}_t &= o_t \circ \sigma_h(\mathcal{C}_t).
\end{aligned}
$$

where, $x_t \in \mathbf{R}^d$ is the input to the LSTM unit, $f_t \in \mathbf{R}^{3h}$ is the forget gate's activation vector, $i_t \in \mathbf{R}^{3h}$ is the input gate's activation vector, $o_t \in \mathbf{R}^{3h}$ is the output gate's activation vector, $h_t \in \mathbf{R}^{4h}$ is the hidden state vector also known as the output tensor of the LSTM unit, $c_t \in \mathbf{R}^{4h}$ is the cell state tensor, '$\circ$' is the Hadamard product operator, and '$*$' denotes the convolution operator. $W_f, W_i, W_o \in \mathbf{R}^{3h \times d}$, $W_c \in \mathbf{R}^{h \times d}$ and $U_f, U_i, U_o, W_{*c} \in \mathbf{R}^{3h \times h}, U_c \in \mathbf{R}^{h \times h}$ are the weight matrices and $b_f, b_i, b_o, \in \mathbf{R}^{3h}, b_c, \in \mathbf{R}^h$ are the bias vector parameters which need to be learned during training. $*$ represents three gates (i, f, o). σ_g

and σ_h represent sigmoid function and hyperbolic tangent functions respectively. d and h refer to the number of input features and the number of hidden units respectively. The initial values of $h_0 \in \mathbf{R}^{4h}$ and $c_0 \in \mathbf{R}^{4h}$ are both usually set to be 0. All other symbols are defined in the above LSTM sections. An ConvLSTM cell architecture diagram with peephole connections is shown in Figure 9. Furthermore, BN represents the batch normalization.

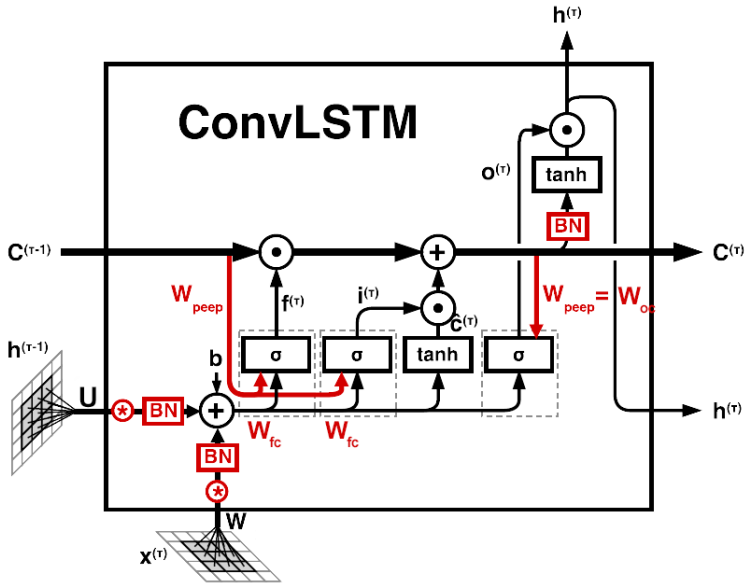

Fig. 9. Architecture of a ConvLSTM cell [150]: The simplified structure of the batch-normalized ConvLSTM cell including peephole connections. The inputs and previous hidden states are convolved to produce 3D tensors that flow through each cell. Changes to standard FC-LSTM are highlighted in red.

7) Physics informed neural networks (PINN): Recent years have seen a dramatic rise in research for Physics informed neural networks which illustrates a deep connection between deep neural networks and physical systems explained with differential equations [71], [72], [73], [70], [46], [64], [103], [69], [47]. So this section will describe some of the important ideas which would change the neural network architecture to fuse in physical laws inside the deep neural models.

a) HNN: Hamiltonian Neural Networks (HNN) is a deep Neural Network model inspired by Hamiltonian mechanics to train models that learn and respect exact conservation laws (exhibits in a physical system) in an unsupervised manner [46]. Hamiltonian mechanics allows us to update a state of a system described with a set of coordinates (q, p) where $q = (q_1, \ldots, q_N)$ represents the positions of a set of objects whereas $p = (p_1, \ldots, p_N)$ denotes their momentum. A scalar function $\mathcal{H}(q, p)$ called Hamiltonian drives the state updates (time evolution of the coordinates) of the system with respect to time as:

$$\frac{dq}{dt} = \frac{\partial \mathcal{H}}{dp}, \quad \frac{dp}{dt} = -\frac{\partial \mathcal{H}}{dq}$$

The scalar quantity Hamilton used in this process is the total energy of the system (q, p). So the vector field $S_{\mathcal{H}} = \left(\frac{\partial \mathcal{H}}{dp}, -\frac{\partial \mathcal{H}}{dq}\right)$ over the inputs of $\mathcal{H}$ describes the moving direction and update of the (q, p) coordinates using:

$$(q_t, p_t) = (q_{t-1}, p_{t-1}) + \int_{t-1}^{t} S_{\mathcal{H}}(q, p)dt$$

The traditional way to train a baseline DNN model to predict the state updates, would be to set the target as state updates $(\frac{dq}{dt}, \frac{dp}{dt})$ and input as current coordinates (q, p) as shown in left side of the Figure 10. The objective would be to minimize the $\mathcal{L}_2$ loss function between actual $(\frac{dq}{dt}, \frac{dp}{dt})$ and network predicted $(\frac{\hat{dq}}{dt}, \frac{\hat{dp}}{dt})$ time derivatives values of (q, p). HNNs are trained in a slightly different method so that the input to the network are position data (p) and momentum data (q) (note that input is similar to baseline DNN) while output is a single scalar "energy-like" quantity $(\mathcal{H}_\theta)$ as shown in right side of the Figure 10. This allows the network to use the properties of Hamilton mechanics to parameterize a deep neural network with parameters θ. Then, before computing the loss, in-graph gradient of the output $(\frac{\partial \mathcal{H}_\theta}{dp}$ and $\frac{\partial \mathcal{H}_\theta}{dq})$ with respect to the inputs (p and q) is calculated. So the objective of this network is to optimize the gradient of the neural network or in other words to minimize the loss function as follows:

$$\mathcal{L}_{HNN} = \left|\left|\frac{\partial \mathcal{H}_\theta}{\partial p} - \frac{\partial q}{\partial t}\right|\right|_2 + \left|\left|\frac{\partial \mathcal{H}_\theta}{\partial q} + \frac{\partial p}{\partial t}\right|\right|_2$$

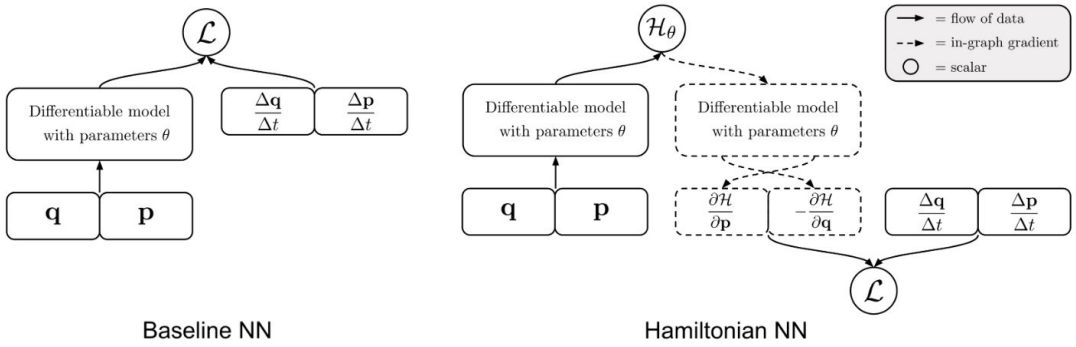

Fig. 10. High level comparison of DNN vs HNN [46]: The forward pass of an HNN is composed of a forward pass through a differentiable model as well as a backpropagation step through the model

b) LNN: Lagrangian Neural Networks (LNN) is a Deep Neural Network model inspired by Lagrangian mechanics to train models that learn and respect exact conservation laws and Lagrangian functions (exhibits in a physical system) in an unsupervised manner [47]. Lets take a physical system which has coordinates $x_t = (q, \dot{q})$ and system state is updated from $x_0 = (q_0, \dot{q}_0)$ to $x_1 = (q_1, \dot{q}_1)$. There are an infinite amount of paths that these coordinates may take as they pass from x_0 to x_1, and these paths are associated with a scalar value S called "the action." Lagrangian mechanics states that the S is related to kinetic (K.E or T) and potential energy (P.E or V) described as:

$$S = \int_{t_0}^{t_1} T(q, \dot{q}) - V(q, \dot{q}) \;\; dt$$

As shown in Figure 11, the system always update its state from x_0 to x_1, by following one path (green path in Figure 11, this is the path which has the least action hence the principle of least action) which gives a stationary value of S.

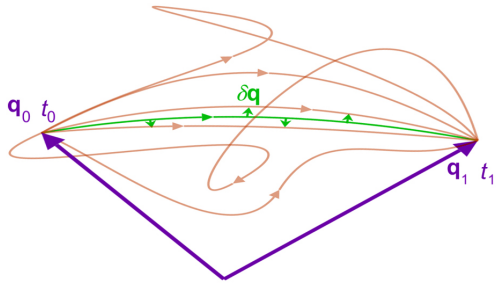

Fig. 11. Possible paths for a physical system from q_0 to q_1 in configuration space[47]: The action is stationary (δS = 0) for small perturbations (δq) to the path that the system actually takes (green).

To drive complete mechanics of solid and fluid bodies, the modern Euler-Lagrange equation is defined as:

$$\mathcal{L} = T(q, \dot{q}) - V(q, \dot{q})$$

with the constrain equation as follows:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}} = \frac{\partial \mathcal{L}}{\partial q}$$

As $\mathcal{L}$ is a function of both q and $\dot{q}$, we can expand the $\frac{d}{dt}$ using chain rule as: $\frac{d}{dt} = \frac{\partial}{\partial t} + \dot{q}\frac{\partial}{\partial q} + \ddot{q}\frac{\partial}{\partial \dot{q}}$ and rewrite the Euler-Lagrange constrain equation as:

$$\dot{q}\frac{\partial^2 \mathcal{L}}{\partial q \partial \dot{q}} + \ddot{q}\frac{\partial^2 \mathcal{L}}{\partial \dot{q}^2} = \frac{\partial \mathcal{L}}{\partial q}$$

We can reformulate the above equation using a matrix inverse to solve for $\ddot{q}$ as:

$$\ddot{q} = \left(\frac{\partial^2 \mathcal{L}}{\partial \dot{q}^2}\right)^{-1}\left(\frac{\partial \mathcal{L}}{\partial q} - \dot{q}\frac{\partial^2 \mathcal{L}}{\partial q \partial \dot{q}}\right)$$

For a given set of coordinates $x_t = (q_t, \dot{q}_t)$, we can calculate $\dot{x}_t = (\dot{q}_t, \ddot{q}_t)$ from the above Lagrangian constrain equation and integrate $\ddot{q}_t$ to obtain the dynamics of the system.

The traditional way to train a baseline DNN model to predict the $\ddot{q}$, would be to set the target as $\ddot{q}$ and input as current coordinates $(q, \dot{q})$ as shown in top (Baseline NN) of the Figure 12. The objective would be to minimize the loss (L) function between actual $(\ddot{q})$ and network predicted $(\hat{\ddot{q}})$. Similar to HNNs, LNNs are trained in a slightly different method so that the input to the network are the current coordinates $(q, \dot{q})$ (note that input is similar to baseline DNN) while output is a single scalar quantity $(\mathcal{L}_\theta)$ as shown in bottom (Lagrangian NN) of the Figure 12. This allows the network to use Lagrangian mechanics properties to parameterize a deep neural network with parameters θ. Then, before computing the loss, in-graph gradient of the output ($\frac{\partial \mathcal{L}_\theta}{dq}$ and $\frac{\partial \mathcal{L}_\theta}{d\dot{q}}$) with respect to the inputs (q and $\dot{q}$) is extracted via modern automatic differentiation. Note that LNN involves both the Hessian and the gradient of a neural network during the forward pass of the LNN. This is not a trivial operation, but modern automatic differentiation makes things surprisingly smooth. So the objective of this network is to minimize the loss function as follows:

$$L_{LNN} = \left|\left|\ddot{q}_{\text{actual}} - \left(\frac{\partial^2 \mathcal{L}}{\partial \dot{q}^2}\right)^{-1}\left(\frac{\partial \mathcal{L}}{\partial q} - \dot{q}\frac{\partial^2 \mathcal{L}}{\partial q \partial \dot{q}}\right)\right|\right|_2$$

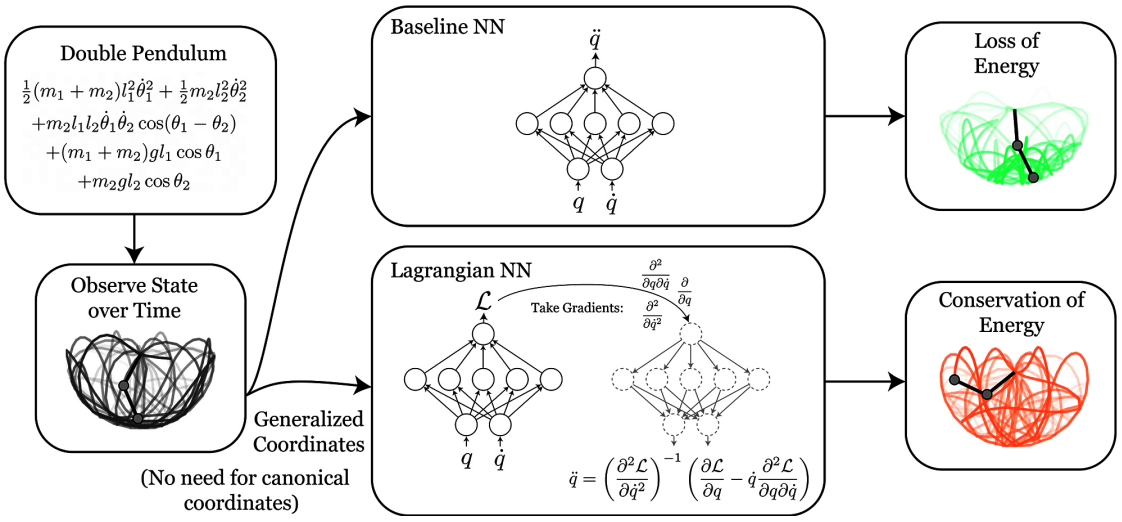

Fig. 12. High level comparison of DNN vs LNN [47]: The forward pass of an LNN is composed of a forward pass through a differentiable model as well as a backpropagation step through the model (Hessian and the gradient)

5. Related Publications

The author has published three papers in the broad area of applied machine learning with focus on applications in nanoscale simulations. Those papers targeted accelerating simulations using machine learning and creating machine learning surrogates to produce simulation outputs. Authored Papers so far:

- Kadupitiya, JCS and Fox, Geoffrey C and Jadhao, Vikram, Machine learning for parameter auto-tuning in molecular dynamics simulations: Efficient dynamics of ions near polarizable nanoparticles, The International Journal of High-Performance Computing Applications, 1094342019899457, 2020, SAGE Publications.
- Kadupitiya, JCS and Fox, Geoffrey C and Jadhao, Vikram, Machine learning for performance enhancement of molecular dynamics simulations, International Conference on Computational Science, pages 116–130, 2019, Springer.
- Kadupitiya, JCS and Sun, Fanbo and Fox, Geoffrey and Jadhao, Vikram, Machine learning surrogates for molecular dynamics simulations of soft materials, Journal of Computational Science, pages-101107, 2020, Elsevier.
- Kadupitiya, JCS and and Fox, Geoffrey and Jadhao, Vikram, Deep Learning Based Integrators for Solving Newton's Equations with Large Timesteps, 2020, (Submitted).

6. Conclusion

While the use of the DL approach in MD simulations has many unique benefits, further research is needed to make the approach more suitable for practical applications. In this review, we have discussed several effective deep learning models for temporal data in MD simulations and demonstrated their success in soft-matter simulation applications. We have also highlighted the main critical challenges with temporal data in MD simulations such as complex temporal dependency and chaotic behaviors, which imposes several unique issues exist in its practical aspects, modeling dynamics of systems that exhibit rare event characteristics or involves longer times to yield the low energy configurations due to kinetic bottlenecks, and scaling these DL approaches to more extensive simulations. We have also provided a brief insight into how some of these issues can be mitigated using innovation in NN architecture informed by time-series application and surrogate applications. Future work will aim to address these issues to make DL models more applicable to practical applications in soft-matter simulations.

REFERENCES

[1] D. Frenkel and B. Smit, Understanding Molecular Simulation, 2nd ed. Academic Press, 2001.

[2] J. Perilla, J. Hadden, B. Goh, C. Mayne, and K. Schulten, "All-atom molecular dynamics of virus capsids as drug targets," The journal of physical chemistry letters, vol. 7, no. 10, pp. 1836–1844, 2016. [Online]. Available: https://pubs.acs.org/doi/10.1021/acs.jpclett.6b00517

[3] J. A. Hadden, J. R. Perilla, C. J. Schlicksup, B. Venkatakrishnan, A. Zlotnick, and K. Schulten, "All-atom molecular dynamics of the hbv capsid reveals insights into biological function and cryo-em resolution limits," Elife, vol. 7, p. e32478, 2018.

[4] R. Allen, J.-P. Hansen, and S. Melchionna, "Electrostatic potential inside ionic solutions confined by dielectrics: a variational approach," Phys. Chem. Chem. Phys., vol. 3, pp. 4177–4186, 2001.

[5] K. Kremer and G. S. Grest, "Dynamics of entangled linear polymer melts: A molecular-dynamics simulation," J. Chem. Phys., vol. 92, pp. 5057–5086, 1990.

[6] V. Jadhao and M. O. Robbins, "Rheological properties of liquids under conditions of elastohydrodynamic lubrication," Tribology Letters, vol. 67, no. 3, p. 66, 2019. [Online]. Available: https://doi.org/10.1007/s11249-019-1178-3

[7] S. C. Glotzer, "Assembly engineering: Materials design for the 21st century (2013 pv danckwerts lecture)," Chemical Engineering Science, vol. 121, pp. 3–9, 2015.

[8] N. E. Brunk, M. Uchida, B. Lee, M. Fukuto, L. Yang, T. Douglas, and V. Jadhao, "Linker-mediated assembly of virus-like particles into ordered arrays via electrostatic control," ACS Applied Bio Mat., vol. 2, no. 5, pp. 2192–2201, 2019.

[9] A. L. Ferguson, "Machine learning and data science in soft materials engineering," Journal of Physics: Condensed Matter, vol. 30, no. 4, p. 043002, 2017.

[10] D. Xue, P. V. Balachandran, J. Hogden, J. Theiler, D. Xue, and T. Lookman, "Accelerated search for materials with targeted properties by adaptive design," Nature communications, vol. 7, no. 1, pp. 1–9, 2016.

[11] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," Neural computation, vol. 8, no. 7, pp. 1341–1390, 1996.

[12] A. W. Long, J. Zhang, S. Granick, and A. L. Ferguson, "Machine learning assembly landscapes from particle tracking data," Soft Matter, vol. 11, no. 41, pp. 8141–8153, 2015.

[13] M. Weigt, R. A. White, H. Szurmant, J. A. Hoch, and T. Hwa, "Identification of direct residue contacts in protein–protein interaction by message passing," Proceedings of the National Academy of Sciences, vol. 106, no. 1, pp. 67–72, 2009.

[14] F. Morcos, A. Pagnani, B. Lunt, A. Bertolino, D. S. Marks, C. Sander, R. Zecchina, J. N. Onuchic, T. Hwa, and M. Weigt, "Direct-coupling analysis of residue coevolution captures native contacts across many protein families," Proceedings of the National Academy of Sciences, vol. 108, no. 49, pp. E1293–E1301, 2011.

[15] P. D'haeseleer, "How does gene expression clustering work?" Nature biotechnology, vol. 23, no. 12, pp. 1499–1501, 2005.

[16] M. A. Bevan, D. M. Ford, M. A. Grover, B. Shapiro, D. Maroudas, Y. Yang, R. Thyagarajan, X. Tang, and R. M. Sehgal, "Controlling assembly of colloidal particles into structured objects: Basic strategy and a case study," Journal of Process Control, vol. 27, pp. 64–75, 2015.

[17] Y. Xue, D. J. Beltran-Villegas, X. Tang, M. A. Bevan, and M. A. Grover, "Optimal design of a colloidal self-assembly process," IEEE Transactions on Control Systems Technology, vol. 22, no. 5, pp. 1956–1963, 2014.

[18] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton et al., "Mastering the game of go without human knowledge," nature, vol. 550, no. 7676, pp. 354–359, 2017.

[19] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," British Machine Vision Association, 2015.

[20] Y. Sun, Y. Chen, X. Wang, and X. Tang, "Deep learning face representation by joint identification-verification," in Advances in neural information processing systems, 2014, pp. 1988–1996.

[21] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv preprint arXiv:1409.0473, 2014.

[22] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3156–3164.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in neural information processing systems, 2017, pp. 5998–6008.

[24] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot et al., "Mastering the game of go with deep neural networks and tree search," nature, vol. 529, no. 7587, pp. 484–489, 2016.

[25] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang et al., "End to end learning for self-driving cars," arXiv preprint arXiv:1604.07316, 2016.

[26] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al., "Tensorflow: a system for large-scale machine learning." in OSDI, vol. 16, 2016, pp. 265–283.

[27] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[28] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky et al., "Theano: A python framework for fast computation of mathematical expressions," arXiv, pp. arXiv–1605, 2016.

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 675–678.

[30] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," arXiv preprint arXiv:1512.01274, 2015.

[31] F. Chollet et al., "Keras," 2015.

[32] L. Buitinck et al., "Api design for machine learning software: experiences from the scikit-learn project," arXiv:1309.0238, 2013.

[33] M. Spellings and S. C. Glotzer, "Machine learning for crystal identification and discovery," AIChE Journal, vol. 64, no. 6, pp. 2198–2206, 2018.

[34] S. S. Schoenholz, "Combining machine learning and physics to understand glassy systems," Journal of Physics: Conference Series, vol. 1036, no. 1, p. 012021, 2018.

[35] J. Liu, Y. Qi, Z. Y. Meng, and L. Fu, "Self-learning monte carlo method," Phys. Rev. B, vol. 95, p. 041101, Jan 2017.

[36] A. Morningstar and R. G. Melko, "Deep learning the ising model near criticality," arXiv preprint arXiv:1708.04622, 2017.

[37] K. Ch'ng, J. Carrasquilla, R. G. Melko, and E. Khatami, "Machine learning phases of strongly correlated fermions," Phys. Rev. X, vol. 7, p. 031038, Aug 2017.

[38] J. Behler, "Perspective: Machine learning potentials for atomistic simulations," The Journal of chemical physics, vol. 145, no. 17, p. 170901, 2016.

[39] V. Botu and R. Ramprasad, "Adaptive machine learning framework to accelerate ab initio molecular dynamics," International Journal of Quantum Chemistry, vol. 115, no. 16, pp. 1074–1083, 2015.

[40] A. Z. Guo, E. Sevgen, H. Sidky, J. K. Whitmer, J. A. Hubbell, and J. J. de Pablo, "Adaptive enhanced sampling by force-biasing using neural networks," The Journal of chemical physics, vol. 148, no. 13, p. 134108, 2018.

[41] J. C. Kadupitige, G. Fox, and V. Jadhao, "Machine learning for auto-tuning of simulation parameters in car-parrinello molecular dynamics," in APS Meeting Abstracts, 2019.

[42] J. Kadupitiya, G. C. Fox, and V. Jadhao, "Machine learning for performance enhancement of molecular dynamics simulations," in International Conference on Computational Science. Springer, 2019, pp. 116–130.

[43] J. Kadupitiya, F. Sun, G. Fox, and V. Jadhao, "Machine learning surrogates for molecular dynamics simulations of soft materials," Journal of Computational Science, p. 101107, 2020.

[44] J. Kadupitiya, G. C. Fox, and V. Jadhao, "Machine learning for parameter auto-tuning in molecular dynamics simulations: Efficient dynamics of ions near polarizable nanoparticles," Indiana University, Nov, 2018.

[45] ——, "Deep learning based integrators for solving newton's equations with large timesteps," 2020.

[46] S. Greydanus, M. Dzamba, and J. Yosinski, "Hamiltonian neural networks," arXiv preprint arXiv:1906.01563, 2019.

[47] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho, "Lagrangian neural networks," arXiv preprint arXiv:2003.04630, 2020.

[48] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," in Advances in neural information processing systems, 2018, pp. 6571–6583.

[49] Y. Du, W. Wang, and L. Wang, "Hierarchical recurrent neural network for skeleton based action recognition," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1110–1118.

[50] D. Eroglu, N. Marwan, M. Stebich, and J. Kurths, "Multiplex recurrence networks," Physical Review E, vol. 97, no. 1, p. 012312, 2018.

[51] V. Jadhao, F. J. Solis, and M. O. De La Cruz, "Simulation of charged systems in heterogeneous dielectric media via a true energy functional," Physical review letters, vol. 109, no. 22, p. 223905, 2012.

[52] V. Jadhao, C. K. Thomas, and M. Olvera de la Cruz, "Electrostatics-driven shape transitions in soft shells," Proceedings of the National Academy of Sciences, vol. 111, no. 35, pp. 12 673–12 678, 2014.

[53] V. Jadhao and M. O. Robbins, "Probing large viscosities in glass-formers with nonequilibrium simulations," Proceedings of the National Academy of Sciences, vol. 114, no. 30, pp. 7952–7957, 2017. [Online]. Available: https://www.pnas.org/content/114/30/7952

[54] N. E. Brunk and V. Jadhao, "Computational studies of shape control of charged deformable nanocontainers," Journal of Materials Chemistry B, vol. 7, p. 6370, 2019. [Online]. Available: http://dx.doi.org/10.1039/C9TB01003C

[55] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," Physics reports, 2019.

[56] K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh, "Machine learning for molecular and materials science," Nature, vol. 559, no. 7715, p. 547, 2018.

[57] G. Fox, J. Glazier, J. Kadupitiya, V. Jadhao, M. Kim, J. Qiu, J. P. Sluka, E. Somogy, M. Marathe, A. Adiga et al., "Learning everywhere: Pervasive machine learning for effective high-performance computation," in 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). IEEE, 2019, pp. 422–429.

[58] Geoffrey Fox, Shantenu Jha, "Learning everywhere: A taxonomy for the integration of machine learning and simulations," in IEEE eScience 2019 Conference, 2019, arXiv:1909.13340. [Online]. Available: https://escience2019.sdsc.edu/

[59] J. Wang, M. A. Gayatri, and A. L. Ferguson, "Mesoscale simulation and machine learning of asphaltene aggregation phase behavior and molecular assembly landscapes," The Journal of Physical Chemistry B, vol. 121, no. 18, pp. 4923–4944, 2017. [Online]. Available: https://pubs.acs.org/doi/abs/10.1021/acs.jpcb.7b02574

[60] A. Moradzadeh and N. R. Aluru, "Molecular dynamics properties without the full trajectory: A denoising autoencoder network for properties of simple liquids," The journal of physical chemistry letters, vol. 10, no. 24, pp. 7568–7576, 2019.

[61] J. Dhamala, H. J. Arevalo, J. Sapp, B. M. Horácek, K. C. Wu, N. A. Trayanova, and L. Wang, "Quantifying the uncertainty in model parameters using gaussian process-based markov chain monte carlo in cardiac electrophysiology," Medical image analysis, vol. 48, pp. 43–57, 2018.

[62] Y. Geifman and R. El-Yaniv, "Deep active learning with a neural architecture search," in Advances in Neural Information Processing Systems, 2019, pp. 5976–5986.

[63] Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification," Journal of Computational Physics, vol. 366, pp. 415–447, 2018.

[64] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou, "Symplectic recurrent neural networks," arXiv preprint arXiv:1909.13334, 2019.

[65] P. Battaglia, R. Pascanu, M. Lai, D. J. Rezende et al., "Interaction networks for learning about objects, relations and physics," in Advances in neural information processing systems, 2016, pp. 4502–4510.

[66] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," arXiv preprint arXiv:1612.00341, 2016.

[67] M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. Von Lilienfeld, "Fast and accurate modeling of molecular atomization energies with machine learning," Physical review letters, vol. 108, no. 5, p. 058301, 2012.

[68] S. Chmiela, A. Tkatchenko, H. E. Sauceda, I. Poltavsky, K. T. Schütt, and K.-R. Müller, "Machine learning of accurate energy-conserving molecular force fields," Science advances, vol. 3, no. 5, p. e1603015, 2017.

[69] A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia, "Hamiltonian graph networks with ode integrators," arXiv preprint arXiv:1909.12790, 2019.

[70] Y. Shin, J. Darbon, and G. E. Karniadakis, "On the convergence and generalization of physics informed neural networks," arXiv preprint arXiv:2004.01806, 2020.

[71] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations," arXiv preprint arXiv:1711.10561, 2017.

[72] L. Yang, D. Zhang, and G. E. Karniadakis, "Physics-informed generative adversarial networks for stochastic differential equations," arXiv preprint arXiv:1811.02033, 2018.

[73] L. Yang, S. Treichler, T. Kurth, K. Fischer, D. Barajas-Solano, J. Romero, V. Churavy, A. Tartakovsky, M. Houston, M. Prabhat et al., "Highly-ccalable, physics-informed gans for learning solutions of stochastic pdes," in 2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS). IEEE, 2019, pp. 1–11.

[74] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, "Learning data-driven discretizations for partial differential equations," Proceedings of the National Academy of Sciences, vol. 116, no. 31, pp. 15 344–15 349, 2019.

[75] P. Shen, X. Zhang, and Y. Fang, "Essential properties of numerical integration for time-optimal path-constrained trajectory planning," IEEE Robotics and Automation Letters, vol. 2, no. 2, pp. 888–895, 2017.

[76] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang, "Predicting disruptive instabilities in controlled fusion plasmas through deep learning," Nature, vol. 568, no. 7753, pp. 526–531, 2019.

[77] F. Häse, I. Fdez. Galván, A. Aspuru-Guzik, R. Lindh, and M. Vacher, "How machine learning can assist the interpretation of ab initio molecular dynamics simulations and conceptual understanding of chemistry," Chem. Sci., vol. 10, pp. 2298–2307, 2019. [Online]. Available: http://dx.doi.org/10.1039/C8SC04516J

[78] M. Kasim, D. Watson-Parris, L. Deaconu, S. Oliver, P. Hatfield, D. Froula, G. Gregori, M. Jarvis, S. Khatiwala, J. Korenaga

et al., "Up to two billion times acceleration of scientific simulations with deep neural architecture search," arXiv preprint arXiv:2001.08055, 2020.

[79] M. Raissi and G. E. Karniadakis, "Hidden physics models: Machine learning of nonlinear partial differential equations," Journal of Computational Physics, vol. 357, pp. 125–141, 2018.

[80] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Multistep neural networks for data-driven discovery of nonlinear dynamical systems," arXiv preprint arXiv:1801.01236, 2018.

[81] Z. Long, Y. Lu, X. Ma, and B. Dong, "Pde-net: Learning pdes from data," arXiv preprint arXiv:1710.09668, 2017.

[82] K. Endo, K. Tomobe, and K. Yasuoka, "Multi-step time series generator for molecular dynamics," in Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[83] P. G. Breen, C. N. Foley, T. Boekholt, and S. P. Zwart, "Newton vs the machine: solving the chaotic three-body problem using deep neural networks," arXiv preprint arXiv:1910.07291, 2019.

[84] F. Tronarp, S. Sarkka, and P. Hennig, "Bayesian ode solvers: The maximum a posteriori estimate," arXiv preprint arXiv:2004.00623, 2020.

[85] L. Ward, A. Agrawal, A. Choudhary, and C. Wolverton, "A general-purpose machine learning framework for predicting properties of inorganic materials," npj Computational Materials, vol. 2, p. 16028, 2016.

[86] J. Kadupitiya, G. C. Fox, and V. Jadhao, "Machine learning for performance enhancement of molecular dynamics simulations," in International Conference on Computational Science. Springer, 2019, pp. 116–130.

[87] G. Fox, J. A. Glazier, J. Kadupitiya, V. Jadhao, M. Kim, J. Qiu, J. P. Sluka, E. Somogyi, M. Marathe, A. Adiga et al., "Learning everywhere: Pervasive machine learning for effective high-performance computation," arXiv preprint arXiv:1902.10810, 2019.

[88] ——, "Learning everywhere: Pervasive machine learning for effective high-performance computation: Application background," Technical report, Indiana University, February 2019. http://dsc. soic . . . , Tech. Rep., 2019.

[89] M. Denil, B. Shakibi, L. Dinh, N. De Freitas et al., "Predicting parameters in deep learning," in Advances in neural information processing systems, 2013, pp. 2148–2156.

[90] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on. IEEE, 2013, pp. 11–20.

[91] C. L. Eng, J. C. Tong, and T. W. Tan, "Predicting host tropism of influenza a virus proteins using random forest," BMC medical genomics, vol. 7, no. 3, p. S1, 2014.

[92] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1867–1874.

[93] V. Cherkassky and Y. Ma, "Practical selection of svm parameters and noise estimation for svm regression," Neural networks, vol. 17, no. 1, pp. 113–126, 2004.

[94] K. Chen and J. Yu, "Short-term wind speed prediction using an unscented kalman filter based state-space support vector regression approach," Applied Energy, vol. 113, pp. 690–705, 2014.

[95] P. V. Balachandran, D. Xue, J. Theiler, J. Hogden, and T. Lookman, "Adaptive strategies for materials design using uncertainties," Scientific reports, vol. 6, p. 19660, 2016.

[96] H. Quan, D. Srinivasan, and A. Khosravi, "Short-term load and wind power forecasting using neural network-based prediction intervals," IEEE transactions on neural networks and learning systems, vol. 25, no. 2, pp. 303–315, 2014.

[97] A. K. Yadav, H. Malik, and S. Chandel, "Selection of most relevant input parameters using weka for artificial neural

network based solar radiation prediction models," Renewable and Sustainable Energy Reviews, vol. 31, pp. 509–519, 2016.

[98] P. M. Kasson and S. Jha, "Adaptive ensemble simulations of biomolecules," Current opinion in structural biology, vol. 52, pp. 87–94, 2018.

[99] R. C. Whaley and J. J. Dongarra, "Automatically tuned linear algebra software," in Supercomputing, 1998. SC98. IEEE/ACM Conference on. IEEE, 1998, pp. 38–38.

[100] P. Balaprakash, J. Dongarra, T. Gamblin, M. Hall, J. K. Hollingsworth, B. Norris, and R. Vuduc, "Autotuning in high-performance computing applications," Proceedings of the IEEE, vol. 106, no. 99, pp. 1–16, 2018.

[101] N. Luehr, T. E. Markland, and T. J. Martínez, "Multiple time step integrators in ab initio molecular dynamics," The Journal of Chemical Physics, vol. 140, no. 8, p. 084116, 2014.

[102] M. Tuckerman, B. J. Berne, and G. J. Martyna, "Reversible multiple time scale molecular dynamics," The Journal of chemical physics, vol. 97, no. 3, pp. 1990–2001, 1992.

[103] P. Toth, D. J. Rezende, A. Jaegle, S. Racanière, A. Botev, and I. Higgins, "Hamiltonian generative networks," arXiv preprint arXiv:1909.13789, 2019.

[104] J. S. Smith, O. Isayev, and A. E. Roitberg, "Ani-1: an extensible neural network potential with dft accuracy at force field computational cost," Chemical science, vol. 8, no. 4, pp. 3192–3203, 2017.

[105] A. Pukrittayakamee, M. Malshe, M. Hagan, L. Raff, R. Narulkar, S. Bukkapatnum, and R. Komanduri, "Simultaneous fitting of a potential-energy surface and its corresponding force fields using feedforward neural networks," The Journal of chemical physics, vol. 130, no. 13, p. 134101, 2009.

[106] J. Wang, S. Olsson, C. Wehmeyer, A. Pérez, N. E. Charron, G. De Fabritiis, F. Noé, and C. Clementi, "Machine learning of coarse-grained molecular dynamics force fields," ACS central science, vol. 5, no. 5, pp. 755–767, 2019.

[107] K. T. Schütt, F. Arbabzadah, S. Chmiela, K. R. Müller, and A. Tkatchenko, "Quantum-chemical insights from deep tensor neural networks," Nature communications, vol. 8, no. 1, pp. 1–8, 2017.

[108] M. Lutter, C. Ritter, and J. Peters, "Deep lagrangian networks: Using physics as model prior for deep learning," arXiv preprint arXiv:1907.04490, 2019.

[109] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," science, vol. 324, no. 5923, pp. 81–85, 2009.

[110] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner, "Discovering physical concepts with neural networks," Physical Review Letters, vol. 124, no. 1, p. 010508, 2020.

[111] R. Bondesan and A. Lamacraft, "Learning symmetries of classical integrable systems," arXiv preprint arXiv:1906.04645, 2019.

[112] Y. Lu, A. Zhong, Q. Li, and B. Dong, "Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations," arXiv preprint arXiv:1710.10121, 2017.

[113] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," Inverse Problems, vol. 34, no. 1, p. 014004, 2017.

[114] L. Ruthotto and E. Haber, "Deep neural networks motivated by partial differential equations," Journal of Mathematical Imaging and Vision, pp. 1–13, 2019.

[115] X. Shen, X. Cheng, and K. Liang, "Deep euler method: solving odes by approximating the local truncation error of the euler method," arXiv preprint arXiv:2003.09573, 2020.

[116] J. Kadupitiya, G. C. Fox, and V. Jadhao, "Simulating molecular dynamics with large timesteps using recurrent neural networks," arXiv preprint arXiv:2004.06493, 2020.

[117] J. Kadupitiya, G. Fox, and V. Jadhao, "Recurrent neural networks based integrators for molecular dynamics simulations," Bulletin of the American Physical Society, vol. 65, 2020.

[118] K. Kadupitiya, S. Marru, G. C. Fox, and V. Jadhao, "Ions in nanoconfinement," Dec 2017. [Online]. Available: https://nanohub.org/resources/nanoconfinement

[119] J. Kadupitiya, N. E. Brunk, and V. Jadhao, "Nanoparticle shape lab," https://nanohub.org/tools/npshapelab, 2020.

[120] N. Brunk, J. Kadupitiya, M. Uchida, T. Douglas, and V. Jadhao, "Nanoparticle assembly lab," https://nanohub.org/tools/npassemblylab, 2019.

[121] J. Kadupitiya, N. Brunk, S. Ali, G. C. Fox, and V. Jadhao, "Nanosphere electrostatics lab," https://nanohub.org/tools/nselectrostatic, 2018.

[122] V. J. Nilsson Lauren, JCS Kadupitiya, "Souffle: Virus capsid assembly lab," 2020.

[123] ——, "Polyvalent nanoparticle binding simulator," 2020.

[124] Y. Jing, V. Jadhao, J. W. Zwanikken, and M. Olvera de la Cruz, "Ionic structure in liquids confined by dielectric interfaces," The Journal of chemical physics, vol. 143, no. 19, p. 194508, 2015.

[125] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, Time series analysis: forecasting and control. John Wiley & Sons, 2015.

[126] G. Claeskens, N. L. Hjort et al., "Model selection and model averaging," Cambridge Books, 2008.

[127] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," Journal of Basic Engineering, vol. 82, no. 1, pp. 35–45, 03 1960. [Online]. Available: https://doi.org/10.1115/1.3662552

[128] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in KDD workshop, vol. 10, no. 16. Seattle, WA, 1994, pp. 359–370.

[129] T. Giorgino et al., "Computing and visualizing dynamic time warping alignments in r: the dtw package," Journal of statistical Software, vol. 31, no. 7, pp. 1–24, 2009.

[130] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," Journal of machine learning research, vol. 10, no. Jan, pp. 1–40, 2009.

[131] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," IEEE transactions on pattern analysis and machine intelligence, vol. 35, no. 8, pp. 1798–1828, 2013.

[132] K. Hornik, M. Stinchcombe, H. White et al., "Multilayer feedforward networks are universal approximators." Neural networks, vol. 2, no. 5, pp. 359–366, 1989.

[133] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 807–814.

[134] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in 2017 International joint conference on neural networks (IJCNN). IEEE, 2017, pp. 1578–1585.

[135] Y. Kim, "Convolutional neural networks for sentence classification," arXiv preprint arXiv:1408.5882, 2014.

[136] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," arXiv preprint arXiv:1709.04875, 2017.

[137] M. Khodayar and J. Wang, "Spatio-temporal graph deep neural network for short-term wind speed forecasting," IEEE Transactions on Sustainable Energy, vol. 10, no. 2, pp. 670–681, 2018.

[138] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure," in Proceedings of International Conference on Neural Networks (ICNN'96), vol. 1. IEEE, 1996, pp. 347–352.

[139] A. Robinson and F. Fallside, The utility driven dynamic error propagation network. University of Cambridge Department of Engineering Cambridge, 1987.

[140] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Neural computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[141] A. Graves, "Generating sequences with recurrent neural networks," arXiv preprint arXiv:1308.0850, 2013.

[142] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium, vol. 3. IEEE, 2000, pp. 189–194.

[143] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," Journal of machine learning research, vol. 3, no. Aug, pp. 115–143, 2002.

[144] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," arXiv preprint arXiv:1409.1259, 2014.

[145] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," arXiv preprint arXiv:1412.3555, 2014.

[146] R. G. Krishnan, U. Shalit, and D. Sontag, "Deep kalman filters.(2015)," arXiv preprint arXiv:1511.05121, 2015.

[147] ——, "Structured inference networks for nonlinear state space models," in Thirty-first aaai conference on artificial intelligence, 2017.

[148] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs," in Proceedings of the ieee conference on computer vision and pattern recognition, 2016, pp. 5308–5317.

[149] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2015, pp. 4580–4584.

[150] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in Advances in neural information processing systems, 2015, pp. 802–810.