Classifying Protein Conformations Using Deep Learning

Xiaozhou Zhou

Abstract

In this project, we suggest a deep learning method for dimensionality reduction and classification of the MD simulation data. This method combines the RNN and the autoencoder architecture and uses the time sequence of the conformations as the input. It can learn low dimensional features of the sequence in order to be used for classifications of the protein energy states and the transitions. We discuss both the single process version and distributed version of the model, and provide the implementation of the single process version.

1 Introduction

1.1 Protein Conformations and States

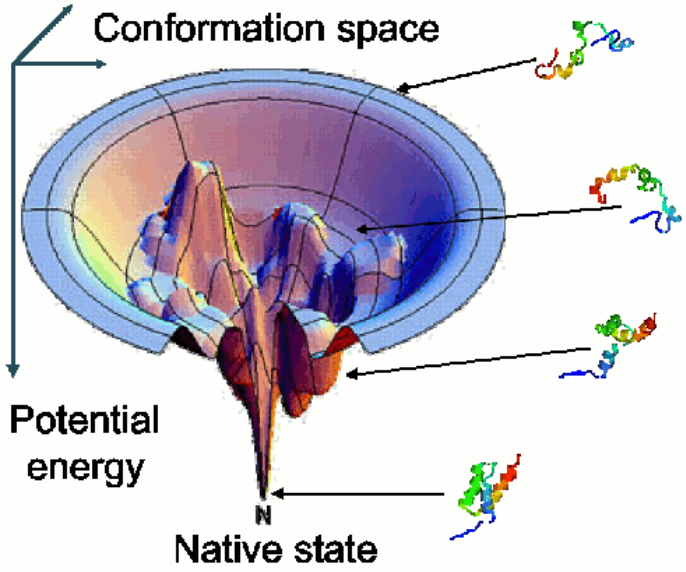
Protein Conformation and Energy States In the study of biophysics, protein is mainly studied by exploring its folding structures and the corresponding functions. The chain of Amino Acids (Polypeptides), the building blocks of proteins, often fold into different structures and provide different functions. The geometric structure of the atoms in the protein molecule is called the conformation of the protein.

The potential energy of the protein molecules are determined by the conformations as shown in Figure 1. The stable conformations of the protein are those in the energy wells of the landscape, which determines the states of the protein conformation. Usually there can be several states for a protein, leading to several different folding structures. Based on these different geometrical structures, proteins could be classified into several functional categories[1].

Because of thermal motion, the protein conformation can transit from one state to another. We can study these temporal behaviors and identify which functional states a protein undergoes during different time intervals.

Molecular Dynamics Simulation One way to study the temporal behaviors of a protein is to use the molecular dynamics(MD) simulation[2]. A simulation starts from a given conformation, i.e., the coordinates of all the atoms in the protein molecule and the water molecules. The coordinates evolve according

Figure 1: Protein Energy Landscape. *source: https://parasol.tamu.edu/*



to differential equations of molecular dynamics with a certain force field model. By solving these equations after each time step, we can generate a time sequence of conformations of the protein. Such conformations can stay in the same energy state or transit to other states. The task of this project is to classify these states by using the given conformations.

1.2 Deep Learning

The coordinates of all the atoms in the protein usually have high dimensionality. When we use machine learning models to classify the conformations, the high dimensionality can lead to suboptimal results and give a poor classification. In this project, we will first use deep learning models to reduce the dimensionality and then do the classification. Deep learning is a branch of methods in machine learning which focuses on using deep neural network to learn representations of the raw features. This section will give a brief introduction to related deep learning models.

Artificial Neural Network A simple artificial neural network has some structure as Figure 2. Each node in the bottom level represents one feature of the given data. Their values are transformed by some non-linear activation functions and sent to the nodes following the arrows. The middle levels are often called the hidden states. The top level is the output of the network, it's the label for classification tasks or numbers to be compared with the some targets for representation learnings. Because the arrows always direct to the next layer and there are no cycles in the graph, this kind of neural network are called the feedforward neural network. A deep neural network is a neural network with

Figure 2: A feed forward neural network. *source: http://cse22-iiith.vlabs.ac.in*

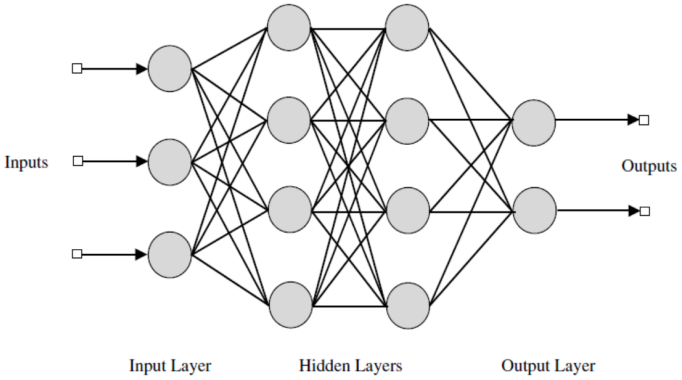
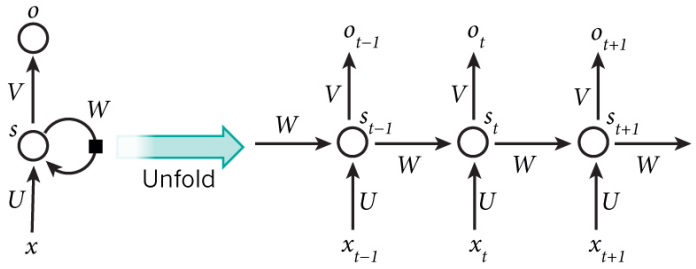


Figure 3: A recurrent neural network and the unfolding in time of the computation. *source: http://www.wildml.com*



multiple hidden layers.

The weights of the activation functions are parameters to be learned by minimizing the difference between the output and the given target, represented by some predefined objective function. The training is done by using a back propagating method that propagates the adjustments of the parameters from the top level to the bottom level. The adjustments are usually calculated by gradient descent methods.

Recurrent Neural Network(RNN) A neural network can be modified to deal with time sequence data, by directing the hidden states to themselves. This is called Recurrent Neural Network(RNN). By using such structures, the model can learn the time dependent relationship among the data.

RNN is one kind of deep neural network, because it is usually trained by "Back-Propagating through Time", i.e., rolling out the time sequence of the hidden states[3]. In principle this sequence should be as long as the whole time sequence of the data, but we usually cut it off within several layers, because the computation is intractable and it will have the vanishing gradient problem when the number of layers gets large. This is problematic when the data has long time dependence for it loses the long time information. [4] The structure of an RNN

and it's unfolding in time of computation is shown in Figure 3.

Sequence to Sequence Model The sequence to sequence model [5] [6] was first introduced in natural language processing(NLP) in order to learn representations of language phrases and deal with the tasks such as translation. It is an RNN combined with memory cells that can encode the time sequence features, and the length of the inputs and outputs can be arbitrary. In this project we will use a modified version of the sequence to sequence model to encode the time sequence of protein conformations.

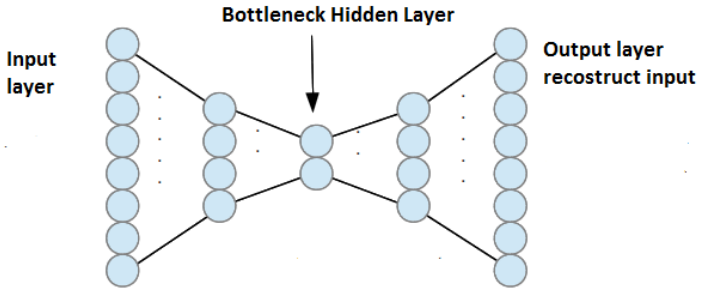
1.3 Dimensionality Reduction

Owing to the recent advances in digital data collection and storage technologies, datasets are becoming so large that traditional information processing methods are inadequate to perform computation on the full data. Datasets are growing both in length and width; we have more data points, in higher dimensions. The earlier is promising as it provides more information about the underlying distribution of the data, while the latter can challenge the learning process, information processing, and inference tasks, which results in the so-called "curse of dimensionality" phenomena[7]. For this reason, many machine learning tasks involve a form of dimensionality reduction or feature extraction, aiming at transforming the data into a lower dimensional feature space, where it is easier to extract useful information from the data. The dimensionality reduction can be useful both from a computation and a statistical viewpoint: less time is required to process the data, and fewer samples are needed to guarantee the convergence of an algorithms.

Kernel PCA Traditional methods of dimensionality reduction including LLE, ISOMAP, Principle Componant Analysis (PCA) and its non-linear variant so-called "kernel PCA" have been extensively studied in the literature. Kernel based approaches provide natural and intuitive means of achieving a nonlinear transformation of the data. Kernel PCA [8] is ubiquitous in data science and machine learning, but it is unscalable because it's non-parametric. So kernel PCA is not suitable for big data problems.

Autoencoder Autoencoders are special types of neural networks which allow dimensionality reduction of large-scale data together with unsupervised learning of features. Stacked Denoising Auto-encoders (SDAEs) are deep neural networks that have proved to be effective in extracting abstract representations from the data [9]. SDAEs and Autoencoder maps an input using a network of neurons to a lower dimension representation. The inverse of the network weights would be then used to reproduce the input at the output layer. Through several iterations of back- and forward-propagation, the weights of the network would be updated. The middle layers of the network (network bottleneck), known as hidden layers, would be then used as the lower dimension representation of the

Figure 4: The structure of an antoencoder



input features [10]. An illustration of the structure of an autoencoder is shown in Figure 4.

In this project, we combine the sequence to sequence model with the autoencoder in order to learn the representations of the sequence data. The second section will give the model details including the structure of the neural network and the algorithms. The third section will talk about the experiments. The forth section is for conclusion and future work.

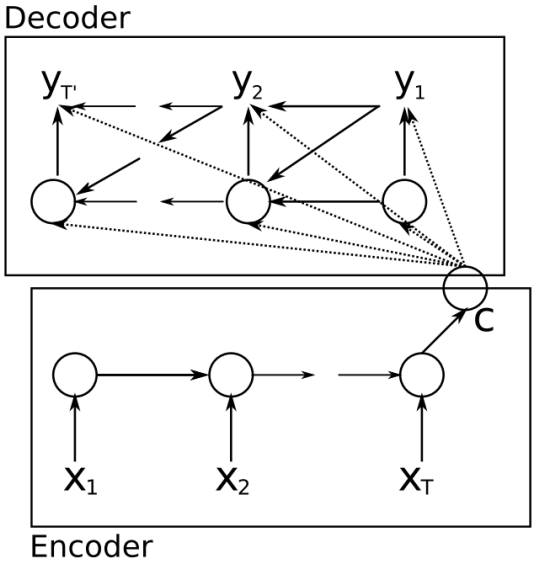
1.4 Molecular Dynamics Analysis - State of the Art

PCA The conventional method of analyzing the conformations is to use PCA to reduce the dimension of the raw data[11]. It was found that most of the positional fluctuations are concentrated in correlated motions in a subspace of only a few (not more than 1%) degrees of freedom. Therefore it is possible to use PCA on the covariance matrix of the coordinate displacement of the atoms to project the data onto a set of low dimensional global variables while remain the most essential motions of the atoms.

Local Scaled Diffusion Map The methods related to PCA are limited in that they consider the number of effective dimensions only as a global property and do not account for the local heterogeneity of MD simulation data. Rohrdanz et al[12] used diffusion map and the determination of local intrinsic dimensionality of large datasets to reduce the high dimensional conformations into a subspace that contains local information. Their method does not require any a priori knowledge about the system (such as prospective reaction coordinates and/or the definition of reactant and product states), and the local heterogeneity of the MD data is accounted for in the construction of global coordinates.

Generative Models Using Random Markov Models Razavian et al[13] constructed generative models known as Markov Random Fields (MRF) to analyze the MD data. They can learn a single model of the data with both the topology and the parameters, or a time-varying model where the topology and parameters of the MRF change smoothly over time. They also suggested

Figure 5: The Encoder-Decoder RNN Model



algorithm learning a Markov Chain over MRFs which can be used to generate new trajectories and study to kinetics.

L1-Regularized Reversible Hidden Markov Models McGibbon et al [14] used L1-regularized, reversible hidden Markov models to analyze the MD simulation data. They presented an EM algorithm for learning and introduced a model selection criteria based on the physical notion of convergence in relaxation timescales. This method captures not only the static information about the proteins but also the temporal behavior which has better physical interpretability.

2 The Model

2.1 Autoencoders for Sequence Data

The traditional autoencoder learns the representations for each single data point. For the sequence data, our autoencoder learns the representations for a sequence of data points with a given length.

We use the structure of the Encoder-Decoder model[6] which is show in Figure 5.

The inputs of the encoder are a sequence of features: $\mathbf{x} = (x_1, \ldots, x_T)$, where T is the length of time steps. At each time step t, the hidden states $\mathbf{h_t}$ are functions of the hidden states in the previous time step and the current input features.

$$\mathbf{h_t} = f(\mathbf{h_{t-1}}, x_t) \tag{1}$$

The function f depends on the structure of the recurrent cells, whose details will be given in Section 2.2.

The learned representations $\mathbf{C}$ of the sequence are a summary of the hidden states at time step T.

The hidden states of the decoder are functions of the summary $\mathbf{C}$, the hidden states and outputs of the previous time step.

$$\mathbf{h_t} = f(\mathbf{h_{t-1}}, y_{t-1}, \mathbf{C}) \tag{2}$$

And the outputs are functions of the current hidden states, the the outputs of the previous time step and the summary $\mathbf{C}$.

$$y_t = g(\mathbf{h_t}, y_{t-1}, \mathbf{C}) \tag{3}$$

The two components of the proposed RNN Encoder-Decoder are jointly trained to minimize the root mean square distance between the output coordinates $\mathbf{y} = (y_1, \ldots, y_{T'})$ and the target coordinates $\hat{\mathbf{y}}$

$$rmsd = \frac{1}{N}|\mathbf{y} - \hat{\mathbf{y}}|^2 \tag{4}$$

Where N is the dimension of the coordinates.

2.2 Memory Cells

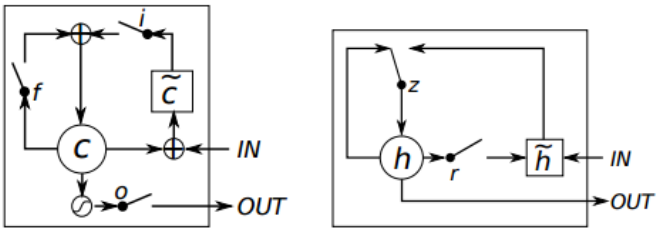
In order to deal with the problem of vanishing gradient and learn the long time dependence of the sequence, "memory cells" are introduced to substitute the simple hidden recurrent units.

LSTM The most popular memory mechanism for RNN is Long Short-Term Memory Units(LSTMs) [15]. As shown in Figure 6(a), the memory cell has some gates which are sigmoid functions σ that control how much information from previous layer to be remembered or forgotten. By using this mechanism we can have long time dependence and get rid of the vanishing gradient problem.

The following are the formula of the elements. i, f, o are called the input, forget and output gates, respectively. h_t are hidden states as in the simple RNN. c_t is the internal memory of the unit. $\tilde{c}_t$ is a candidate hidden state that is computed based on the current input and the previous hidden state. U, W are the corresponding parameters to be learned by training.

$$i = \sigma(x_t U^i + h_{t-1} W^i) \tag{5}$$
$$f = \sigma(x_t U^f + h_{t-1} W^f) \tag{6}$$
$$o = \sigma(x_t U^o + h_{t-1} W^o) \tag{7}$$
$$\tilde{c}_t = tanh(x_t U^{\tilde{c}} + h_{t-1} W^{\tilde{c}}) \tag{8}$$
$$c_t = c_{t-1} \circ f + g \circ \tilde{c} \tag{9}$$
$$h_t = tanh(c_t) \circ o \tag{10}$$

Figure 6: The LSTM and GRU Cells. *source: http://deeplearning4j.org/lstm.html*



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Figure 1: Illustration of (a) LSTM and (b) gated recurrent units. (a) i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content. (b) r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation.

GRU Gated Recurrent Units(GRUs)[6] are a variant and simpler version of LSTM. As shown in Figure 6(b). The reset gate r controls how much of the previous states h_{t-1} is used for building the candidate states, and the update gate z controls how much information from the previous hidden state will carry over to the current hidden state. This is essentially combining the input and forget gates in LSTM into one gate.

$$r = \sigma(x_t U^r + h_{t-1} W^r) \tag{11}$$
$$z = \sigma(x_t U^z + h_{t-1} W^z) \tag{12}$$
$$h_t = z h_{t-1} + (1 - z)\tilde{h}_t \tag{13}$$
$$\tilde{h}_t = tanh(x_t U^{\tilde{h}} + (r \circ h_{t-1}) W^{\tilde{h}}) \tag{14}$$

3 The Algorithms

3.1 Preprocessing

The data sets from MD simulations are trajectories which are time sequence of frames. Each frame contains the coordinates of all the atoms involved in the simulation at a certain time step. A typical time step is chosen by the parameters of the simulaiton, which is often from nanoseconds to microseconds. The conformation of the molecules can be reconstructed by using these coordinates and the topology information of the protein. In order to feed the data into our feature learning model, several steps of preprocessing are needed to transform the data into the proper input. The software package used for preprocessing is MDTraj [16].

- *Load The Trajectory:* Each frame is stored in a "dcd" file. Combined with "pdb" file which provides the topology the molecule, we can load multiple frames into one trajectory object.

- *Superpose the Coordinates:* The coordinates of the atoms of all the frames must be superposed according to the same reference frame in order to compare the movements. Usually the first frame is chosen as the reference frame.
- *Extract the Useful Atoms:* The frames usually contains water atoms which are irrelevant to the analysis. We can extract the protein atoms only in the trajectory. And it's usually helpful to further get rid of the hydrogen atoms in the protein molecules because they contain less information of the conformations than heavier atoms.
- *Smooth the Trajectory:* Because the atoms have thermal fluctuation, it is helpful to take the average of the coordinates in several frames to be the input data, so that the fluctuation is reduced. An often used smoothing window is 10 frames.
- *Rescale the Coordinates:* The molecules in the simulation are confined in a unit cell whose size varies according to the size of the molecule. In order to compare the input numbers with the output numbers of the feature learning model, we need to rescale the coordinates into a unit with all coordinates lying between $[-1, 1]$. The original coordinates can be rescaled using the size of the unit cell, or, if the size of the cell is unknown, for batch learning, they can be rescaled by the difference between the maximum and the minimum coordinate values.
- *Construct the Sequence:* Each data point in our model is a sequence of coordinates, therefore we need to combine multiple frames into sequences. The sequences can be built without sliding, ie., each sequence contains continuous frames without duplications; or with sliding, ie., adjacent sequences may contain the same frames. The length of the sequence determines the depth of the backpropagation, so it is not good to be too large; and it can not be too small because we want to capture the sequence of a transition from one state to another. A proper length is between 5 and 10.

3.2 Training the Sequence Autoencoder

As shown in Figure 5, each input X_i is a set of preprocessed coordinate values in a frame of the sequence. The encoder and decoder parts are trained together. During training, we will compare the output Y_i with the input X_i using the loss function, which can be the $rmsd$ or other predefined objective functions.

The updating of parameters is done by minimizing the loss function using Back-Propagating through Time in a mini batch mode. Using the chain rule of differentiation for $y = g(h(x)) = g(w)$,

$$\frac{dy}{dx} = \frac{dy}{dw}\frac{dw}{dx} \tag{15}$$

if we know the form of all the operation or functions involved in the objective, such as $g(w)$ and $h(x)$, we can compute the back propagation formally with no numerical approximations. This is called the "automatic differentiation"(AD).

Algorithm 1 shows the steps of training the autoencoder network.

Algorithm 1 Training the seq2seq autoencoder

Input:
num_layers: Number of hidden layers.
{*hidden_size*}: Dimensions of the hidden states on each layer, including the size of the bottleneck.
batch_size: number of sequences in a mini-batch
{**X**}: Array of input sequences.

Output:
An object of the model containing all parameters of the neural network.

```
Construct the neural network using the Tensorflow operators and initialize the parameters
repeat
    Extract a random batch with size = batch_size from {X}
    for each sequence X in the mini-batch do
        Input X to the model and get the output Y
        Compute the loss function for the output
        Use backpropagation of the gradient of the loss function to compute the update of parameters
    end for
    Update the model parameters using the average of all the updates in the mini-batch
until converge or reach maximum iteration number
```

Table 1 shows the summary of the preprocessing parameters and their typical values.

3.3 Encoding for Clustering and Classification

To encode the data to low dimensional representations is straightforward. The input is the simulation data with the same preprocessing and the output is the hidden states on the bottleneck of the seq2seq autoencoder.

After obtaining the low dimensional representations for each sequence, we can do clustering or classification by traditional machine learning methods.

3.4 Training the Distributed Model

The size of the MD simulation data can be as large as 10s TB, which makes it impractical to compute with a single process or a single machine. In order to scale up, we need a distributed version of the model. When the model is not large, we

parameter	typical value
smoothing window	10
sequence length	10
number of hidden layers	1
hidden size	100
batch size	10

Table 1: Parameters for preprocessing and model architecture

can use data parallelization and update the parameters in the parameter server asynchronically[18].

The architecture is shown in Figure 7. There is a parameter server which contains all the parameters to be learned. Each replica of the model is a full copy of the neural network, and each one is assigned a random partition of data. During training, each replica is trained independently and they only talk with each other through the parameter server: In each iteration, they fetch the parameter values w from the parameter server, train with the mini-batch of data, and then push the change of parameters computed from its own training to the parameter server, where the parameters are updated. Since the updating is asynchronic, it introduces extra randomization to the optimization, which can help with avoiding local optima and give a good rate of convergence. The algorithm sketch is shown in Algorithm 2.

If the model is large, as indicated in [18], we can also use model parallelization, where each replica of the model is distributed on several nodes. We need to consider the trade off of parallelization and the communication cost. In the case for analyzing the MD simulation data, since the RNN model is a time sequence, the backpropagation of the parameters is almost sequential, so it will not gain much from model parallelization, therefore we will not use this method in our analysis.

3.5 Implementation

The single process version of the model is implemented and the code can be found here: https://github.com/PhiphyZhou/protein. The code is written using Google's Python package Tensorflow[17]. It has built-in AD for all commonly used tensor operators so it can take care of the back propagation automatically.

This implementation uses *rmsd* as the loss function and the optimization method is adagrad. You can choose to use LSTM or GRU as the memory cell. All the modifiable parameters are in the file *config.py*, where you can select which data set to use, define the architecture of the model and set the parameters for optimization. The main module is *encoder_decoder.py*, which can do both training and encoding.

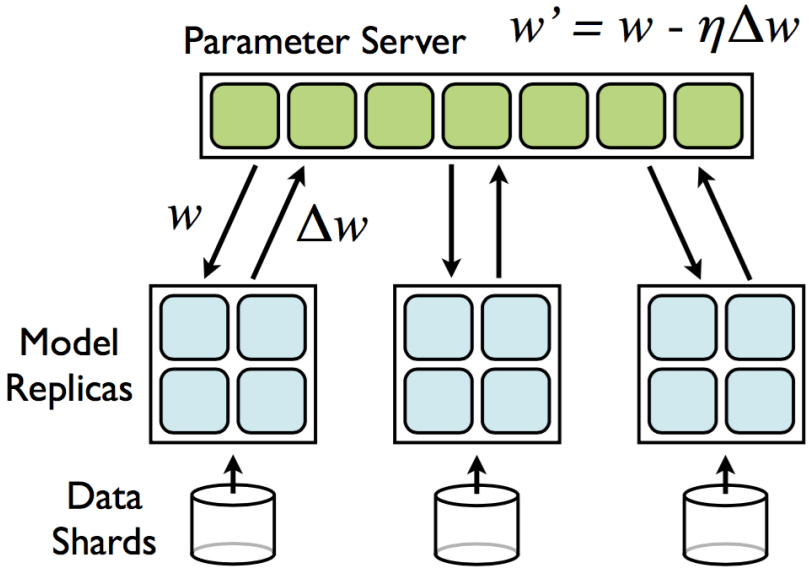
Algorithm 2 Distributed training of the seq2seq autoencoder

Input:
num_rep: Number of model replicas
num_layers: Number of hidden layers
{*hidden_size*}: Dimensions of the hidden states on each layer, including the size of the bottleneck
batch_size: number of sequences in a mini-batch
{**X**}: Array of input sequences

Output:
An object of the model containing all parameters of the neural network.

1: Construct *num_rep* replicas of the neural network using the Tensorflow operators and initialize the parameters
2: Randomly make *num_rep* partitions $\{\mathbf{X_r}\}$ of the data and assign them to each model replica
3: **repeat** for each replica r in parallel
4: Fetch the parameter values from the parameter server
5: Extract a random batch with size = *batch_size* from $\{\mathbf{X_r}\}$
6: **for** each sequence **X** in the mini-batch **do**
7: Input **X** to the model and get the output **Y**
8: Compute the loss function for the output
9: Use backpropagation of the gradient of the loss function to compute the update of parameters
10: **end for**
11: Compute the change of the parameters using the average of all the updates in the mini-batch
12: Push the parameter change to the parameter server
13: The parameter server updates the parameter values
14: **until** converge or reach maximum iteration number

Figure 7: The data parallelization with the parameter server.



4 Conclusion and Future Work

In this project, we discussed a deep learning method for dimensionality reduction and classification of the MD simulation data. This method combines the RNN and the autoencoder architecture to deal with the time sequence of the conformations.

The work remains to be done is to do experiments with the single process model on the small set of MD data such as Alanine Dipeptide which is the simplest molecule that contains enough interesting behaviors of the protein. The accuracy and computational cost of classifying the states using the encoded features can be compared with traditional methods such PCA.

In order to test on the data of more complicated proteins such as BPTI, we need to implement the distributed version since the scale is too large for a single training process. It is worth study how well the asynchronic parallelization can work for achieving an effective optimization of the neural network.

5 Acknowledgments

I thank Dr. Yanif Ahmad and Dr. Tom Woolf for advising on this project. I also thank Dr. Jason Eisner and Ben Ring for useful discussions and suggestions.

References

[1] H. Frauenfelder, F. Parak, and R. Young, "Conformational substates in proteins," *Annu Rev Biophys Biophys Chem*, vol. 17, pp. 451–479, 1988.

[2] M. Karplus and J. McCammon, "Molecular dynamics simulations of biomolecules," *Nat Struct Biol 2002*, vol. 9, pp. 646–652, 2002.

[3] H. Jaeger, "A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach," *GMD Report 159, German National Research Center for Information Technology*, p. 48, 2002.

[4] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies," *A Field Guide to Dynamical Recurrent Networks*, pp. 237–243, 2001.

[5] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, p. 9, 2014.

[6] K. Cho, B. Van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[7] M. Verleysen and D. François, "The Curse of Dimensionality in Data Mining," *Analysis*, vol. 3512, pp. 758 – 770, 2005.

[8] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[9] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *The Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[10] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, *et al.*, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[11] A. Amadei, A. B. M. Linssen, and H. J. C. Berendsen, "Essential dynamics of proteins," *Proteins: Structure, Function, and Bioinformatics*, vol. 17, no. 4, p. 412425, 1993.

[12] M. A. Rohrdanz, W. Zheng, M. Maggioni, and C. Clementi, "Determination of reaction coordinates via locally scaled diffusion map," *Journal of Chemical Physics*, vol. 134, no. 12, pp. 1–11, 2011.

[13] N. S. Razavian, H. Kamisetty, and C. J. Langmead, "Learning generative models of molecular dynamics," *BMC genomics*, vol. 13, no. Suppl 1, p. S5, 2012.

[14] R. T. McGibbon, B. Ramsundar, M. M. Sultan, G. Kiss, and V. S. Pande, "Understanding Protein Dynamics with L1-Regularized Reversible Hidden

Markov Models," *Proceedings of the 31st International Conference on Machine Learning*, vol. 32, pp. 1197–1205, 2014.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1–32, 1997.

[16] R. McGibbon, K. Beauchamp, M. Harrigan, C. Klein, J. Swails, C. Herndez, C. Schwantes, L.-P. Wang, T. Lane, and V. Pande, "Mdtraj: A modern open library for the analysis of molecular dynamics trajectories," *Biophysical Journal*, vol. 109, no. 8, pp. 1528 – 1532, 2015.

[17] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale machine learning on heterogeneous distributed systems (preliminary white paper, november 9, 2015)," Nov. 2015.

[18] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large Scale Distributed Deep Networks," *NIPS 2012: Neural Information Processing Systems*, pp. 1–11, 2012.