Machine learning-based compression of quantum many body physics: PCA and autoencoder representation of the vertex function

Jiawei Zang
Columbia University
jz3122@columbia.edu

Matija Medvidović
Flatiron Institute, Columbia University
matija.medvidovic@columbia.edu

Dominik Kiese
Flatiron Institute

Domenico Di Sante
University of Bologna

Anirvan M. Sengupta
Flatiron Institute, Rutgers University

Andrew J. Millis
Flatiron Institute, Columbia University

Abstract

The vertex function, a continuous function of three momenta describing particle-particle scattering that is typically obtained by sophisticated calculations, plays a central role in the Feynman diagram approach to quantum many-body physics. Here, we use Principal Component Analysis (PCA) and a deep convolutional autoencoder to derive compact, low-dimensional representations of the vertex functions derived using the functional renormalization group for the two dimensional Hubbard model, a paradigmatic theoretical model of interacting electrons on a lattice. Both methodologies successfully reduced the dimensionality to a mere few dimensions while preserving accuracy. PCA demonstrated superior performance in dimensionality reduction compared to the autoencoder. The results suggest the presence of a fundamental underlying structure in the vertex function and suggest paths to dramatically reducing the computational complexity of quantum many-body calculations.

1 Introduction

Quantum many-body science (understanding the behavior of large systems of many interacting particles) is one of the computational grand challenge problems of the present day. It is naturally formulated in terms of a state in a Hilbert space of dimension that grows exponentially with the number of particles to be described, and quantum entanglement and the fermion sign problem mean that the space cannot be explored via standard Monte Carlo methods. It is believed that a complete solution cannot be obtained in polynomial time, so approximation and compression methods are essential [1].

One important approach to the problem is via *Feynman diagrams*, in essence combinations of correlation functions of operators (often with a physically intuitive meaning) acting on states in the Hilbert space. The study of Feynman diagrams has a rich history stretching back about three quarters of a century and a crucial role is played by the *vertex function*, a function of three momenta conventionally written $\Gamma(k_1, k_2, k_3)$ describing the scattering of two particles in initial momentum states k_1, k_2 into two particles in other momentum states k_3 and $k_4 = k_1 + k_2 - k_3$. Knowledge

Machine Learning and the Physical Sciences Workshop, NeurIPS 2023.

of the vertex function enables insights into the physics, including whether the system is magnetic, superconducting or not ordered and what are the responses to externally applied fields [1–4].

However the vertex function is difficult to compute in general and, although not exponentially large, has a complicated structure that is not well understood. In this paper we address the question: is there a concise representation of this intricate vertex function? We use PCA and deep convolutional autoencoder methods [5] to compress a large dataset of vertex functions, finding that extreme compression is possible without sacrificing accuracy and that PCA methods are markedly superior to autoencoder methods. We introduce metrics for determining the lossiness of the compression and show that the compressed representation leads to interesting physical insight. Our findings forge a path towards refining the computation, storage and use of vertex functions, offering potential advantages in computational efficiency and memory optimization.

2 Physics model

We choose the two dimensional square lattice Hubbard model, a paradigmatic model of interacting electrons on a lattice, as an ideal test bed for our methodology. The model may be written in standard second quantized notation using a mixed momentum (k) and lattice site (i) representation as

$$H = \sum_{ks} \xi_k c^\dagger_{ks} c_{ks} + \mu \sum_{i,s} n_{i,s} + U \sum_i n_{i,\uparrow} n_{i,\downarrow} \tag{1}$$

Here $c^\dagger_{ks}$ creates an electron in a state of momentum k and spin s; the function $\xi_k = -2t(cosk_x + cosk_y) - 4t' cosk_x cosk_y$ describes how electrons move on the lattice (t and t' are respectively the quantum mechanical amplitudes for electrons to move from a site to its first or second neighbor), $n_{is} = c^\dagger_{is} c_{is}$ gives the number of electrons on site i, μ is the chemical potential that controls the electron density, and $U > 0$ is a repulsive interaction that correlates electron motion by disfavoring configurations with two electrons on a site. As t'/t and μ are varied at fixed U/t (for definiteness we focus here on $U = 3t$) the model is tuned between a non ordered "fermi liquid' (FL) regime and antiferromagnetic (AFM), superconducting (SC), and ferromagnetic (FM) ordered states [6].

We obtain vertex functions for the model using the functional renormalization group (fRG) method [7, 8], which has been found to be a valuable computational tool for studying the properties of low dimensional interacting Fermi systems. At the core of the fRG method is a detailed set of coupled nonlinear differential flow equations whose solution determines the vertex functions of the system. If the momentum space in d-dimensions is discretized into N_k^d tiles, the vertex function is specified by N_k^{3d} complex numbers determined by the solution of N_k^{3d} coupled equations. These equations are derived by taking derivatives based on an energy scale, represented by Λ. Details about the generation of the input data are shown in the supplemental material.

The fRG procedure produced vertex functions characterized by a notably high dimensionality of 576^3. To enable a comparison between PCA and the autoencoder without excessive computational costs, we down-sampled the vertex to reduce the dimension to a more manageable 144^3. Once the comparison was complete, the superior model was then applied to compress the original, full-dimension vertex.

We executed PCA using the randomized SVD solver. This linear method identifies the principal components, or axes, that maximize variance, thereby transforming the original data onto a new coordinate system. The significance of each coordinate is ranked by the explained variance. Conversely, the autoencoder is a non-linear method designed to learn efficient data encoding. It compresses the data into a low-dimensional latent space by applying a flexible parametrized transformation (the *encoder*). A second model (the *decoder*) is trained to reconstruct the original dataset as accurately as possible. We used five-layer convolutional neural-networks (CNN, [9]) architecture for both components with layer norms and GeLU activations [10]. In addition to transposed convolutions, we use non-parametric upsampling layers after each layer. Each autoencoder was trained with approximately 4 GPU-hours on a single Nvidia A100 card with a single mean-squared-error cost function.

3 Compression loss: Comparison between PCA and the autoencoder

To assess the quality of the compression we use two error metrics: the direct (pointwise) reconstruction error of the vertex, represented as $\frac{||\hat{\Gamma}-\Gamma||_2}{||\Gamma||_2}$ and the error in the leading eigenvalue [11], a physical

quantity computed from the vertex function that characterizes the tendency towards different forms of order. We took a random 80/20 train/test split of input data generated with t' ranging from 0 to $0.5t$

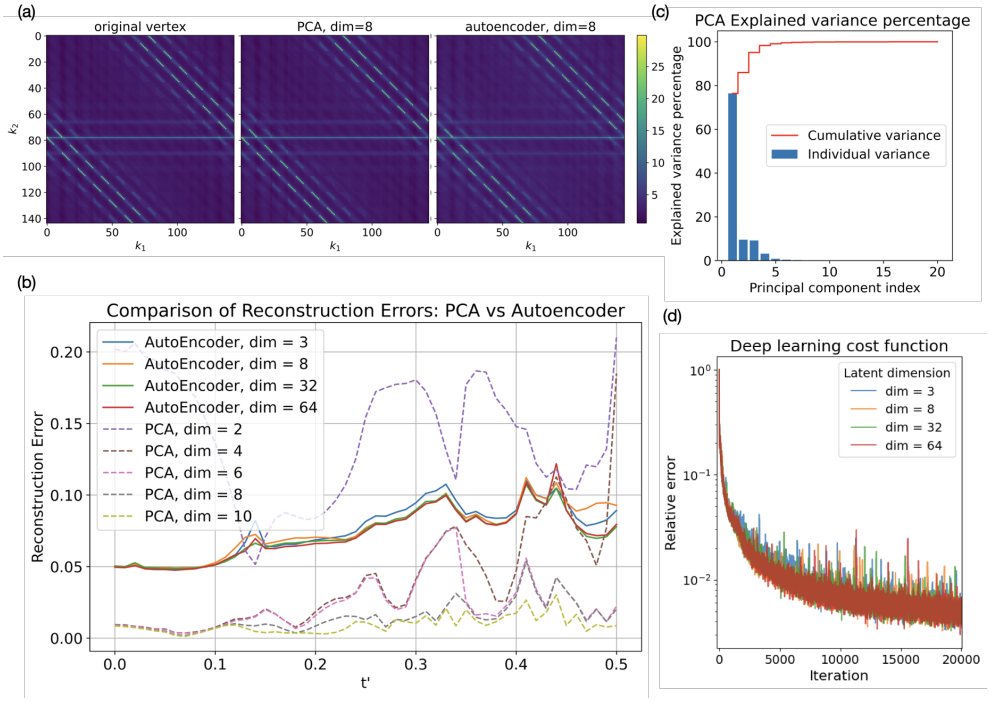

Figure 1: (a) A representative slice of the original vertex function $\Gamma(k_1, k_2, k_3)$ and the corresponding reconstructed ones at $k_3 = 0$ and $t' = 0$. (b) The reconstruction errors, $\frac{||\hat{\Gamma}-\Gamma||_2}{||\Gamma||_2}$, at different t' using two methods, each with varying latent dimensions. (c) PCA explained variance percentage as a function of the principal component index. (d) Deep Learning cost function evolution.

and spaced at intervals of $0.01t$, and subsequently evaluate the error metric utilizing the entire dataset. Fig. 1(a) displays a representative test slice of the original vertex function $\Gamma(k_1, k_2, k_3)$ alongside the corresponding reconstructed ones at $k_3 = 0$ and $t' = 0$. Our findings suggest that a mere few dimensions can encapsulate the majority of the variance. In the case of PCA, approximately 98% of the variance is captured within a 4-dimensional space, as depicted in Fig. 1(c). For the autoencoder, the errors are consistent across different latent spaces, as shown in Fig. 1 (d). For the leading eigenvalue analysis we simply compared the results from the exact and compressed representation of vertices corresponding to particular input parameters; these results are shown in Fig. 2.

In Fig. 1(a), the autoencoder yields errors around 7%, exhibiting consistency across varying latent dimensions. Conversely, PCA displays a continuous decrement in errors as the latent dimension ascends. Notably, at $dim = 2$, PCA underperforms in comparison to the autoencoder; And it surpasses the autoencoder's performance for $dim \geq 6$. Figure 2 shows that Both PCA and the autoencoder exhibit commendable proficiency in replicating the eigenvalues. However, a notable deviation is observed when the latent dimension of PCA is less than 3, even though the direct error of the vertex remains approximately around 20%.

4 PCA Latent space

Because the PCA was found to outperform the autoencoder, we trained the PCA method on the full data set. The reconstruction errors are similar to the ones using the down-sampled data. We now discuss the information encoded in the PCA latent space.

Panel (a) of Fig.3(a) displays $k_3 = 0$ slices of the exact vertices computed for parameters such that the model is in the AF, SC or FM ordered state. We see that the vertices corresponding to AFM and SC states exhibit similar patterns, which markedly differ from those of the FM orders, suggesting a deep

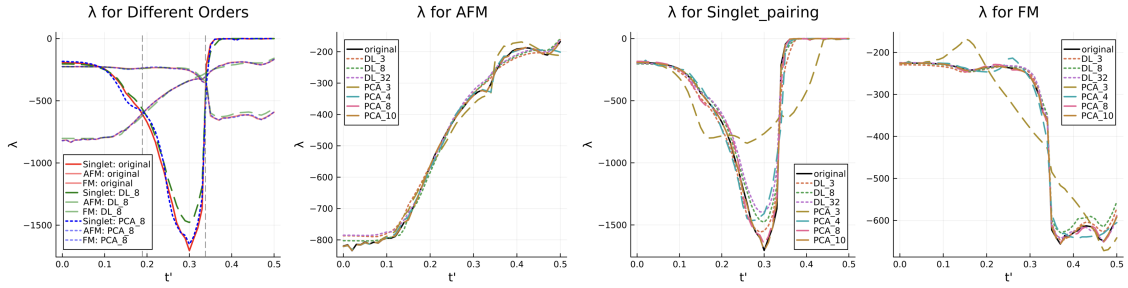

Figure 2: Eigenvalues of the physical orders calculated from the original vertex and the reconstructed ones.

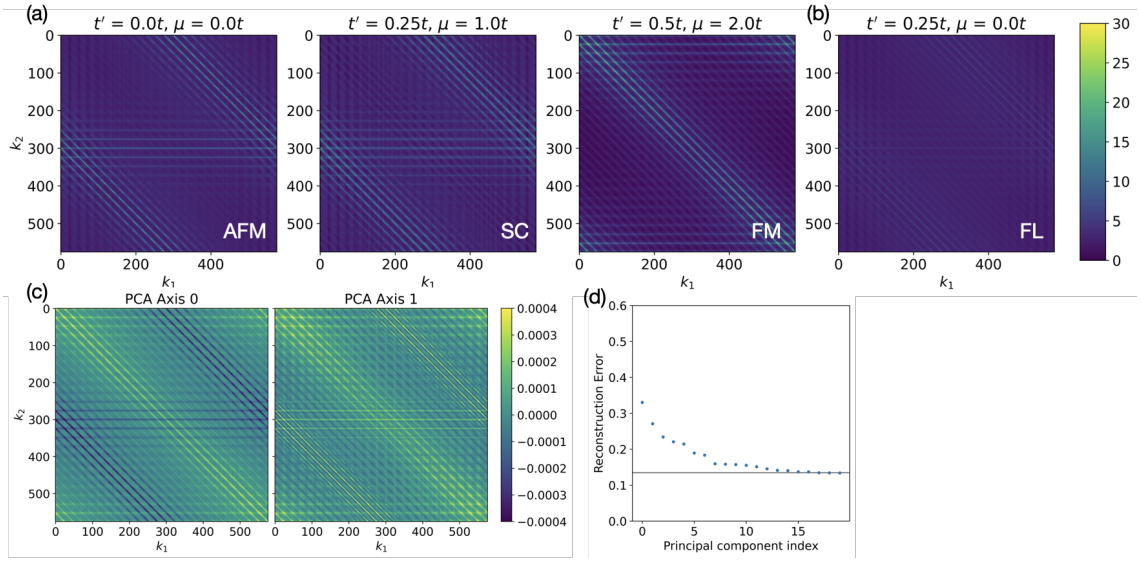

Figure 3: (a) Slice of the vertex functions at $k_3 = 0$ at three different ground states: antiferromagnetic (AFM), superconducting (SC), ferromagnetic (FM), and (b) Fermi Liquid (FL) states. (c) Slice of the first and second PCA axes at $k_3 = 0$. (d) The reconstruction error of the vertex at the FL state.

physical similarity of the AF and SC states. Remarkably, the vertex eigenvector corresponding to the primary PCA axis contains the structures found in all of the ordered states, but with different signs of the diagonal (FM) and side (AFM/SC) lines while the second component shows these features with the same sign; combinations of these two distinguish the two states.

Ordered states are associated with divergent structures in vertices [12]; the non-ordered (Fermi liquid) state has a vertex function with no divergent structures. We show in Fig. 3 (d) how well the vertex calculated in the Fermi liquid regime can be represented in terms to the PCA components learned from the order state regimes. We see that the direct reconstruction error decreases to around 13.5% as more principle component indexes are used, but does not decrease further on adding more components. Thus the ordered state PCA components are significantly present in the FL vertex, but some part of the FL vertex is not captured, because the FL state represents a region distinct from the ordered state.

5 Outlook

The vertex function is an apparently complicated, difficult to calculate function of three momentum coordinates that encodes important information about the physics of quantum many-body systems. We have shown using both PCA and autoencoder methods that for a wide range of parameters encompassing a diversity of physical states, the vertex function of the two dimensional Hubbard

model (the “fruit-fly" of electronic condensed matter physics) can be well represented as a point in a very low (4-12) dimensional latent space. That the apparent complexity of this approach to quantum many body physics can be so strongly compressed suggests that much more computationally efficient calculational methods can be devised. The finding suggests important directions for future research including further investigation into the PCA representation of the FL state, direct application of the methods devised here to the frequency dependent dynamical vertices required in the dynamical mean field approach to correlated systems, and over a longer term, reframing calculations so that one can operate directly on the coefficients of the PCA/autoencoder expansion. Further, our finding that PCA methods provide a substantially more efficient representation than the standard autoencoder methods suggests that research into the optimizations required in the autoencoder method may be beneficial.

Software libraries

Neural networks were trained using JAX [13] for array manipulations, automatic differentiation for sampling and optimization and GPU support, Flax [14] for neural-network construction and manipulation and NumPy [15] and SciPy [16] for CPU array manipulations. Matplotlib [17] was used to produce figures.

Acknowledgments and Disclosure of Funding

The research leading to these results has received funding from the NSF MRSEC program through the Center for Precision-Assembled Quantum Materials (PAQM) - DMR-2011738. MM acknowledges support from the CCQ graduate fellowship in computational quantum physics. The Flatiron Institute is a division of the Simons Foundation.

References

[1] Sénéchal, D., Tremblay, A.-M. & Bourbonnais, C. Theoretical methods for strongly correlated electrons (Springer Science & Business Media, 2006).

[2] Tahvildar-Zadeh, A. N., Freericks, J. K. & Jarrell, M. Magnetic phase diagram of the hubbard model in three dimensions:the second-order local approximation. Phys. Rev. B **55**, 942–946 (1997). URL https://link.aps.org/doi/10.1103/PhysRevB.55.942.

[3] Kuneš, J. Efficient treatment of two-particle vertices in dynamical mean-field theory. Phys. Rev. B **83**, 085102 (2011). URL https://link.aps.org/doi/10.1103/PhysRevB.83.085102.

[4] Rohringer, G., Valli, A. & Toschi, A. Local electronic correlation at the two-particle level. Phys. Rev. B **86**, 125114 (2012). URL https://link.aps.org/doi/10.1103/PhysRevB.86.125114.

[5] Murphy, K. P. Probabilistic Machine Learning: An introduction (MIT Press, 2022). URL probml.ai.

[6] Honerkamp, C. & Salmhofer, M. Magnetic and superconducting instabilities of the hubbard model at the van hove filling. Physical Review Letters **87**, 187004 (2001).

[7] Beyer, J., Hauck, J. B. & Klebl, L. Reference results for the momentum space functional renormalization group. The European Physical Journal B **95** (2022). URL https://doi.org/10.1140%2Fepjb%2Fs10051-022-00323-y.

[8] Honerkamp, C. & Salmhofer, M. Temperature-flow renormalization group and the competition between superconductivity and ferromagnetism. Phys. Rev. B **64**, 184516 (2001). URL https://link.aps.org/doi/10.1103/PhysRevB.64.184516.

[9] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature **521**, 436–44 (2015). URL http://www.ncbi.nlm.nih.gov/pubmed/26017442.

[10] Hendrycks, D. & Gimpel, K. Gaussian error linear units (gelus) (2016). URL http://arxiv.org/abs/1606.08415.

[11] Zhai, H., Wang, F. & Lee, D.-H. Antiferromagnetically driven electronic correlations in iron pnictides and cuprates. Phys. Rev. B **80**, 064517 (2009). URL https://link.aps.org/doi/10.1103/PhysRevB.80.064517.

[12] Metzner, W., Salmhofer, M., Honerkamp, C., Meden, V. & Schönhammer, K. Functional renormalization group approach to correlated fermion systems. Rev. Mod. Phys. **84**, 299–352 (2012). URL https://link.aps.org/doi/10.1103/RevModPhys.84.299.

[13] Bradbury, J. et al. JAX: composable transformations of Python+NumPy programs (2018). URL http://github.com/google/jax.

[14] Heek, J. et al. Flax: A neural network library and ecosystem for JAX (2020). URL http://github.com/google/flax.

[15] Harris, C. R. et al. Array programming with NumPy. Nature **585**, 357–362 (2020). 2006.10256.

[16] Virtanen, P. et al. Scipy 1.0: fundamental algorithms for scientific computing in python. Nature Methods **17**, 261–272 (2020). URL http://www.nature.com/articles/s41592-019-0686-2.

[17] Hunter, J. D. Matplotlib: A 2D graphics environment. Comput. Sci. Eng. **9**, 99–104 (2007).