Open-Source Fermionic Neural Networks with Ionic Charge Initialization

Shai Pranesh[1], **Shang Zhu**[2], **Venkat Viswanathan**[3], **Bharath Ramsundar**[1]

[1]Deep Forest Sciences, California, USA
[2]Department of Mechanical Engineering, University of Michigan, Michigan,USA
[3]Department of Aerospace Engineering, University of Michigan, Michigan,USA
shai@deepforestsci.com, shangzhu@umich.edu, venkvis@umich.edu, bharath@deepforestsci.com

Abstract

Finding accurate solutions to the electronic Schrödinger equation plays an important role in discovering important molecular and material energies and characteristics. Consequently, solving systems with large numbers of electrons has become increasingly important. Variational Monte Carlo (VMC) methods, especially those approximated through deep neural networks, are promising in this regard. In this paper, we aim to integrate one such model called the FermiNet, a post-Hartree-Fock (HF) Deep Neural Network (DNN) model, into a standard and widely used open source library, DeepChem. We also propose novel initialization techniques to overcome the difficulties associated with the assignment of excess or lack of electrons for ions.

Introduction

The Variational Monte Carlo (VMC) technique is based on the variational principle in quantum mechanics, which states that the expected value of the energy of a trial wave function is always greater than or equal to the ground state energy of the molecule system. The expected energy is the average energy of the sampled electrons.

$$\mathbf{E}_{\text{ground}} \leq \frac{\langle \psi \mid \hat{H} \mid \psi \rangle}{\langle \psi \mid \psi \rangle} \approx \mathbf{E}_{\text{expected}} = \frac{1}{N} \sum E(\theta) \quad (1)$$

where the energy is calculated by the equation

$$\mathbf{E}(\theta) = \mathbf{E}_{\text{kinetic}} + \mathbf{E}_{\text{el-el}} + \mathbf{E}_{\text{nuc-nuc}} + \mathbf{E}_{\text{nuc-el}} \quad (2)$$

and where kinetic energy is calculated as follows:

$$\mathbf{E}_{\text{kinetic}} = -\frac{1}{2} \sum_i \left[\left(\frac{\partial \log(|\psi_\theta|)}{\partial r_i} \right)^2 + \frac{\partial^2 \log(|\psi_\theta|)}{\partial x^2} \right] \quad (3)$$

Here r_i is the electron coordinates in the i-th domain (x, y, z) coordinates, and ψ_θ is the trial wavefunction depending on parameters θ. $\mathbf{E}_{\text{nuc-el}}$, $\mathbf{E}_{\text{nuc-nuc}}$, $\mathbf{E}_{\text{el-el}}$ correspond to the total potential energy between each nucleon with each electron, each nucleon with other nucleons, each electron with other electrons in the molecule system, respectively.

Traditional VMC techniques initialize and sample the electron's coordinates via the Monte Carlo algorithm and move toward values that minimize the expected energy. The general steps in a VMC algorithm are as follows:

- Initialize suitable parameters θ in a chosen trial wavefunction ψ_θ and initialize electron coordinates.
- Compute new trial coordinates of the electrons using a suitable distribution.
- Employ the Metropolis algorithm to determine whether to accept or reject the proposed new move, using the square of the magnitude of the wavefunction as the electron's probability.
- If the step is accepted, update parameters after a set number of steps.
- Conclude the calculation and compute the final averages for expected energy calculation.

The results heavily depend on the quality of the trial wavefunction ψ_θ. In this regard, DNN-based approaches can help to approximate and optimize the trial wave functions while running the VMC simultaneously. Certain physics-based elements are built into the ansatz for better convergence, with the training cost as to minimize the expected energy function as low as possible.

DeepChem and Differentiable Physics

DeepChem (Ramsundar et al. 2021) is an open source Python library for scientific machine learning and deep learning on molecular and quantum datasets. DeepChem offers a framework for tackling challenging scientific problems in domains such as drug discovery, energy, and biotechnology. This is achieved by standard workflows constructed from fundamental components, including data loaders, featurizers, data splitters, learned models, metrics, and hyperparameter tuners. This systematic design has allowed DeepChem to be part of a wide variety of applications, like large-scale benchmarking for molecular machine learning through the MoleculeNet benchmark suite (Wu et al. 2018), protein-ligand interaction modeling (Gomes et al. 2017), generative modeling of molecules (Frey, Gadepally, and Ramsundar 2023), and more.

DeepChem aims to support open-source differentiable physics (Ramsundar, Krishnamurthy, and Viswanathan 2021) infrastructure for scientific machine learning. Differentiable density functional theory infrastructure has been previously integrated into DeepChem for the same purpose

(Vidhyadhiraja et al. 2023). This work integrates FermiNet (Pfau et al. 2020) as part of the ongoing efforts to facilitate differentiable physics support and enable rapid physical calculations via neural network approximations.

Implementation

The FermiNet model was split and implemented as the following major components, each dealing with a specific task:

- Electron Sampler
- Hartree-Fock Baselines
- Ionic Charge Initialization
- FermiNet Model (neural network layers)

Electron Sampler

The electron sampler was implemented into the DeepChem utils subpackage using Numpy. The electron sampler can initialize electron positions centered on each nucleon. The number of electrons initialized depends on the atomic number of each nucleon, and in the case of ions, electrons are either added or deleted according to Mulliken population analysis (discussed briefly in the subsequent subsection).

After initialization, the trial electron coordinates are sampled using Gaussian distribution centered around the present electron's coordinates, which are then accepted/rejected via the Metropolis-Hastings algorithm using the log probability of the wavefunction ($2 \cdot \log(|\psi|)$) calculated at the electron's position. The difference between the log probability of trial and original electrons gives the probability of the jump or acceptance probability(A). A random value $u \sim \log(\text{Uniform}(0, 1))$ is then generated, and if $u \leq A$, the trial coordinates are accepted ($r_{t+1} = r'$); otherwise, it is rejected, and the Markov chain stays at the current state ($r_{t+1} = r_t$). This process is repeatedly done for a set number of iterations, usually until the model gives converged results.

Hartree-Fock Baselines

At each epoch of the initial pretraining for the FermiNet model, the HF baseline is computed using the STO-6G basis function. This calculation is performed using the sampled electron's postion for that particular epoch, aiming to align the models's calculated orbitals with the HF baseline. This aids in accelerating convergence during the actual training of the model layers. The conventional HF calculation ignores the electron-correlation term, which will be integrated into the model training process itself. These calculations were performed using the PySCF library (Sun et al. 2020).

Ionic Charge Initialization

In an ion, one or more of the nucleons has either an excess or lack of electrons to accommodate the ionic charge. There is no present initialization policy of ionic charges in DeepMind's implementation of the model (James S. Spencer and Contributors 2020). We propose a novel initialization policy based on Mulliken population analysis (Mulliken 1955).

The Mulliken population charge matrix is calculated during the HF calculations, which gives the partial electron charges of each atom in a molecule based on the overlap of atomic orbitals. This charge matrix can act as a heuristic to add/remove electrons as per the ionic charge of the molecule system. The nucleon with the most negative partial charge is the most ideal candidate for excess electrons and the one with the most positive partial charge will have a lack of electrons.

Algorithm 1: Charge Initialization Policy

Input: Partial charge matrix from Mulliken population analysis.
Output: Electron number for each nucleon in the ion

Let $charge$ be the partial charge matrix.
Let $electron$ be the array initialized with the atomic number of each nucleon.
Let $t =$ ionic charge of the system.
while $t \neq 0$ **do**
  if $t < 0$ **then**
    $index =$ argmin(*charge*)
    $electron[index]$ += 1
    t += 1
    $charge[index]$ += 1
  else
    $index =$ argmax(*charge*)
    $electron[index]$ -= 1
    t -= 1
    $charge[index]$ -= 1
  end if
end while
return electron

The exact algorithm for the charge initialization policy can be seen in Algorithm 1. The input for this algorithm will be a matrix of partial charges computed via Mulliken Population analysis. For example, in a LiH+ ion, the partial charge matrix will be [0.88694, 0.11306], where the first term corresponds to the Li partial charge and the other term to the partial charge of H in the ionic system. We also use another matrix for the calculation called the electron matrix which consists of the electrons of each nucleon in an uncharged system. For example in LiH+, the electron matrix would be [3, 1] as Li has 3 electrons and H 1 electron in its uncharged form.

FermiNet Model

The FermiNet architecture consists of two major feature layers: 1-electron and 2-electron feature layers which act as the embedding layer for electron distance features to be used to model the wavefunction ansatz. The 2-electron layer featurizes each of the electron's distance from the other electrons whereas the 1-electron layer featurizes each electron's distance from each of the nucleons, using the value of 2-electron features in the process.

The ansatz consists of an "envelope layer" which uses the 1-electron features to enforce the boundary condition of the probability of the electron's position tending to 0 when the electron moves towards infinity from the nucleus. The envelope layer of each electron j is modeled as:

$$\sum_{m} \pi_{k,\alpha}^{i,m} . \exp(-\sigma_{k,\alpha}^{i,m} |\mathrm{R}_{j,m}|) \tag{4}$$

where σ is a learnable 3×3 tensor parameter and π is a learnable scalar parameter. There are a total of $ikm\alpha$ copies of the σ and π parameters. m, k, i, j, and α correspond respectively to the number of nucleons, the number of determinants to be used in the ansatz, the number of orbitals, the number of electrons, and the spin of the electron taken. The term $\mathrm{R}_{j,m}$ corresponds to the 1-electron-distance vector of the j^{th} electron and m^{th} nucleon.

The envelope layer along with the computed 1-electron feature is used to model the ansatz's orbital evaluated at a specific electron. Using the slater-type determinants of these orbitals, the wavefunction's value can be computed at the input electron's position.

FermiNet Training FermiNet has 2 parts to the training: supervised pretraining and unsupervised training. During pretraining, orbital values calculated from the Hartree-Fock method are used as labels, with FermiNet's orbital values trained to match the Hartree-Fock baseline. The loss function is:

$$\mathcal{L}_{pretrain} = \frac{1}{N} \sum_{kij} \left(\Phi_{kij} - \Phi_{kij}^{HF} \right)^2 \tag{5}$$

where Φ and Φ^{HF} correspond to the calculated orbital values from the FermiNet model and the HF baseline respectively. k, i, and j correspond to the number of determinants, number of orbitals, and number of electrons respectively.

Next in the training phase, the model's parameters are tuned according to the modified gradients using the expected energy. The parameters in the network are updated according to the following gradient

$$\nabla_\theta \mathcal{L}_{train} = 2\mathbb{E}_p \left[\left(E_p - \mathbb{E}_p(E_p) \right) \nabla_\theta \log |\psi| \right] \tag{6}$$

where $\mathbb{E}_p$ is the expected mean over the sampled electrons, E_p is the energy of sample p across all batches, and $\nabla_\theta \mathcal{L}_{train}$ is the modified gradient with respect to the model parameters θ, while $\nabla_\theta \log |\psi|$ is the gradients with respect to the $\log |\psi|$ term.

The gradients of the network are updated using the PyTorch hooks module. For the kinetic energy calculation, functorch (Horace He 2021) (available from Pytorch 2.0.0) is used to calculate the Hessian and the Jacobian.

Results

In Fig 1, we plot the ground state energies for Hydrogen molecules with different internuclear distances and compare them against the CCSD values calculated by PySCF. The FermiNet's ground state energy curve closely tracks the CCSD curve. We also tested the model to estimate the LiH molecule's ionization potential, and compared them with the CCSD and HF calculation values obtained from the NIST database (National Institute of Standards and Technology n.d.). The result can be found in Table 1. We used a batch size of 8 and 10 steps between MCMC updates to get the results. Using a larger batch size, more iterations, and more MCMC steps will decrease the noise in the results at the cost of increased training time.

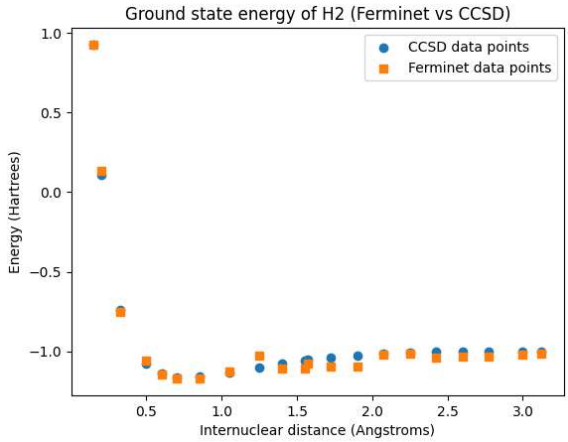

Figure 1: H2 ground state energy - FermiNet vs CCSD. Plotted using 21 different data points. The values obtained from FermiNet can be seen to closely match that of the values calculated via CCSD. Running the model with more MCMC steps and iterations can give more reliable results.

FermiNet	CCSD	HF
0.2481	0.2637	0.2387

Table 1: The ionization potential for LiH in hartrees obtained from different methods, as stated, is given. The FermiNet's ionization potential can be seen to have improved upon the values obtained from Hartree-Fock calculations and is closer to the energy obtained using CCSD. This ionization potential was calculated using the ionic charge initialization policy that we proposed (and is not present in DeepMind's implementation.)

Conclusion

From Figure 1 and Table 1, we observe that the model's results closely match that of the CCSD method and improve on top of the HF baseline for LiH ionization potential calculation. We anticipate integrating FermiNet into DeepChem will lead to improved infrastructure for rapid experimentation with the FermiNet model for calculating accurate ground state energies of molecule/ion systems.

We aim to further improve the model by using Pytorch's JIT functionality to match the capabilities of DeepMind's JAX implementation (Spencer et al. 2020). Also, we aim to integrate features from PauliNet(Schätzle et al. 2023), another DNN-based VMC model. PauliNet has the advantage of employing fewer parameters and achieving quicker training compared to FermiNet but with slightly lower accuracy. This is because PauliNet has "physical conditions" built into the ansatz whereas FermiNet aims to train and learn those

features. Thus, incorporating physical priors into FermiNet can lead to faster convergence of the model with accurate results. The standardized DeepChem implementation we have introduced will help facilitate such experimentation in future work.

References

Frey, N. C.; Gadepally, V.; and Ramsundar, B. 2023. Fastflows: Flow-based models for molecular graph generation. https://arxiv.org/abs/2201.12419. arXiv:2201.12419.

Gomes, J.; Ramsundar, B.; Feinberg, E. N.; and Pande, V. S. 2017. Atomic Convolutional Networks for Predicting Protein-Ligand Binding Affinity. arXiv:1703.10603.

Horace He, R. Z. 2021. functorch: JAX-like composable function transforms for PyTorch. https://github.com/pytorch/functorch.

James S. Spencer, D. P.; and Contributors, F. 2020. FermiNet.

Mulliken, R. S. 1955. Electronic population analysis on LCAO–Mo molecular wave functions. I. *The Journal of Chemical Physics*, 23(10): 1833–1840.

National Institute of Standards and Technology. n.d. Computational Chemistry Comparison and Benchmark DataBase (CCCBDB).

Pfau, D.; Spencer, J.; de G. Matthews, A.; and Foulkes, W. 2020. Ab-Initio Solution of the Many-Electron Schrödinger Equation with Deep Neural Networks. *Phys. Rev. Research*, 2: 033429.

Ramsundar, B.; Eastman, P.; MacBride, A.; and Trang, N. V. 2021. Making DeepChem a Better Framework for AI-Driven Science.

Ramsundar, B.; Krishnamurthy, D.; and Viswanathan, V. 2021. Differentiable Physics: A Position Piece.

Schätzle, Z.; Szabó, P. B.; Mezera, M.; Hermann, J.; and Noé, F. 2023. DeepQMC: An open-source software suite for variational optimization of deep-learning molecular wave functions. *The Journal of Chemical Physics*, 159(9): 094108.

Spencer, J. S.; Pfau, D.; Botev, A.; and Foulkes, W. M. C. 2020. Better, Faster Fermionic Neural Networks. arXiv:2011.07125.

Sun, Q.; Zhang, X.; Banerjee, S.; Bao, P.; Barbry, M.; Blunt, N. S.; Bogdanov, N. A.; Booth, G. H.; Chen, J.; Cui, Z.-H.; and et al. 2020. Recent developments in the PySCF program package. *The Journal of Chemical Physics*, 153(2).

Vidhyadhiraja, A.; Thiagarajan, A. P.; Zhu, S.; Viswanathan, V.; and Ramsundar, B. 2023. Open Source Infrastructure for Differentiable Density Functional Theory. In *1st Workshop on the Synergy of Scientific and Machine Learning Modeling @ ICML2023*.

Wu, Z.; Ramsundar, B.; Feinberg, E.; Gomes, J.; Geniesse, C.; Pappu, A. S.; Leswing, K.; and Pande, V. 2018. MoleculeNet: A Benchmark for Molecular Machine Learning. *Chem. Sci.*, 9: 513–530.