Simulations of state-of-the-art fermionic neural network wave functions with diffusion Monte Carlo

Max Wilson*
Quantum Engineering CDT, Bristol University, Bristol, BS8 1TH, UK
QuAIL, NASA Ames Research Center, Moffett Field, California 94035, USA and
KBR, Inc., 601 Jefferson St., Houston, TX 77002, USA

Nicholas Gao
QuAIL, NASA Ames Research Center, Moffett Field, California 94035, USA
KBR, Inc., 601 Jefferson St., Houston, TX 77002, USA
German Aerospace Center (DLR), Linder Höhe 51147 Köln, Germany and
Technical University of Munich, Boltzmann Str. 3 85748 Garching, Germany

Filip Wudarski†
QuAIL, NASA Ames Research Center, Moffett Field, California 94035, USA and
USRA Research Institute for Advanced Computer Science, Mountain View, California 94043, USA

Eleanor Rieffel and Norm M. Tubman‡
QuAIL, NASA Ames Research Center, Moffett Field, California 94035, USA
(Dated: April 15, 2021)

Recently developed neural network based *ab-initio* methods (Pfau et. al arxiv:1909.02487v2) for finding ground states of fermionic systems can generate state-of-the-art results on a broad class of systems. In this work, we improve the results for this Ansatz with diffusion Monte Carlo. Additionally, we introduce several modifications to the network (Fermi Net) and optimization method (Kronecker-factored approximate curvature) that reduce the number of required resources while maintaining or improving the modelling performance. In terms of the model, we remove redundant computations and alter the way data is handled in the permutation equivariant function. The diffusion Monte Carlo results exceed or match state-of-the-art performance for all systems investigated: atomic systems Be-Ne, and the carbon cation C^+.

I. INTRODUCTION

Neural networks, in recent years, have provided an alternative computational paradigm for solving electronic structure problems including directly solving the time-independent Schrödinger equation to find approximate ground state wave functions of atoms and molecules [1]. Standard quantum chemistry methods [2], such as coupled-cluster [3], full configuration interaction [4], Variational Monte Carlo (VMC) [5], and Diffusion Monte Carlo (DMC) [6] have been used in conjunction with neural network methods in different ways, such as the introduction of new Ansätze [7–12] or providing rich datasets for the prediction of properties of previously untested systems and their dynamics in supervised learning frameworks [13–16].

Notable examples of new Ansätze are neural networks such as SchNet [11], PauliNet [9], Boltzmann machines [12] and Fermi Net [7]. Even though these techniques are still in the early stages of development, machine learning for fermionic systems has been widely studied over the past few years, and some of these techniques are capable of producing state-of-the-art results for electronic structure simulations. We have particular interest in finding good approximations to ground states of fermionic Hamiltonians [9] and the precision and accuracy of Fermi Net [7].

In this paper we modify an existing framework, the Fermi Net [7], by changing how data is handled in the network and removing redundant elements. We apply the wave function optimization algorithm VMC whilst altering some aspects of the Kronecker-Factored Approximate Curvature (KFAC) optimization, and then run the DMC algorithm to improve the wave function further. The use of DMC is standard practice in many Quantum Monte Carlo (QMC) codes, however, the wave functions used in virtually all packages prior to this new wave of neural network approaches are using a wave function consisting of Slater/Jastrow/Multi-determinant/Backflow components, which generally are less accurate or less systematically improveable than the Fermi Net Ansatz.

The QMC we described above is run with standard VMC under the Born-Oppenheimer approximation (nuclei are fixed) and then DMC with the fixed-node approximation in the continuum.

That is to say we solve the Schrödinger equation

$$\hat{H}\psi(X) = E\psi(X), \tag{1}$$

where X defines the system configuration, Figure 1, $\hat{H}$

* aw16952@bristol.ac.uk
† filip.a.wudarski@nasa.gov
‡ norm.m.tubman@nasa.gov

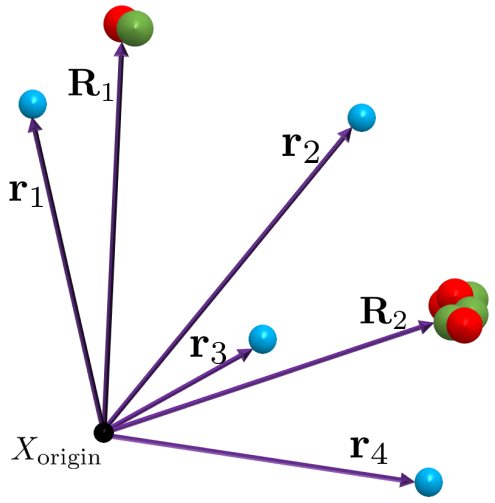


FIG. 1. The system of atoms (protons/neutrons red/green) and electrons (blue). The set of position vectors of a system of atoms, $\mathbf{R}_i$ and electrons $\mathbf{r}_i$. X_{origin} is the origin (black) of the coordinate system. In all cases explored here we considered single atom systems and the origin was set to the nucleus position. However, the choice of origin is completely arbitrary as Fermi Net is invariant to translations.

is the Hamiltonian operator and E is the energy of the eigenfunction $\psi(X)$, the wave function we are attempting to model.

In this work, Section II introduces the problem to be solved (i.e. the time-independent Schrödinger equation for fermionic systems) and all the relevant background including VMC, KFAC, DMC and a sketch of the Fermi Net. Section III describes the particular methods used in this work including detailed descriptions of the algorithms and an Ansatz that is functionally identical to previous work (Fermi Net). We refer to this model and the associated optimization methods as Fermi Net* in order to distinguish from this previous work. The results are described and discussed in Section IV and finally we conclude our findings in Section V.

There are several contributions in this work:

- Introduced changes to the Fermi Net implementation: removing the diagonal elements of the pairwise terms; and changing how the data is handled in the permutation equivariant function. Both changes result in efficiency improvements;
- first (to our knowledge) application of diffusion Monte Carlo with a neural network Ansatz;
- and state-of-the-art results on all systems explored (Be-Ne, C^+)

Throughout the paper we use bold font to denote vectors and regular font to denote vector components (scalars). Both are lower case symbols or letters. Matrices are capitalized symbols or letters and are not written in bold font. Though a batched vector, for example m vectors of dimension n arranged in an $m \times n$, can be represented as a matrix, we notationally treat it as a vector, as is typical for neural network methods. There are some caveats to these, though in general the meaning is obvious from context, for example nuclei coordinates are capitalized. All calculations are performed in atomic units.

II. SOLVING THE SCHRÖDINGER EQUATION FOR FERMIONIC SYSTEMS

A. Variational Monte Carlo

The Schrödinger Equation plays a central role in the description of quantum behavior of chemical systems. Apart from a handful of analytically solvable models, for example the hydrogen atom [17], one mostly needs to incorporate approximate techniques and numerical methods that scale unfavorably with the increasing system size (e.g. number of electrons) [8]. Many techniques and approximations have been introduced to address this problem, and in this work we have a particular focus on real space Monte Carlo approaches.

An atomic or molecular system can be described by the time-independent Schrödinger equation

$$\hat{H}\psi(X) = E\psi(X). \tag{2}$$

Under the Born-Oppenheimer approximation, the position of nuclei are frozen and we have a Hamiltonian for the electronic degrees of freedom,

$$\hat{H} = -\frac{1}{2}\hat{\nabla}^2 + V(X). \tag{3}$$

$\hat{\nabla}^2$, the electron kinetic energy operator of the Hamiltonian, is the multidimensional ($3n_e$ where n_e is the number of electrons) Laplacian of the wave function

$$\hat{\nabla}^2 = \sum_{i=1}^{n_e} \sum_{j=x,y,z} \frac{\partial^2}{\partial r_{i,j}^2}. \tag{4}$$

$r_{i,j}$ and $R_{i,j}$ correspond to coordinate $j = x, y, z$ of the i-th electron and nuclei, respectively. $V(X)$ is the potential energy of some (electron, nuclei) configuration

$$X = (r_{1,x}, r_{1,y}, r_{1,z}, \ldots, r_{n_e,z}; R_{1,x}, \ldots, R_{n_n,z}), \tag{5}$$

given by

$$V(X) = \sum_{i>j}^{n_e} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{i,I}^{n_e,n_n} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \sum_{I>J}^{n_n} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{6}$$

where n_n is the number of nuclei, Z_I is the atomic number of nuclei I, and $\mathbf{r}_i$ and $\mathbf{R}_I$ are the position vectors of the electron i and nuclei I, respectively.

Approximately solving the Schrödinger equation, Equation (2), is a subroutine in finding the minimum energy of the system, i.e. the ground state energy E_0. Since the Hamiltonian is a bounded operator, one may use a variational principle [2]

$$E_0 \leq \frac{\langle\psi|\,\hat{H}\,|\psi\rangle}{\langle\psi|\psi\rangle} = \frac{\int dX \psi^*(X)\hat{H}\psi(X)}{\int dX \psi^*(X)\psi(X)}, \tag{7}$$

detailing that the expectation value of the Hamiltonian $\hat{H}$ with respect to a state $\psi(X)$ is bounded from below by the ground state energy E_0. Finding the best approximation to the ground state $\psi_0(X)$ can be done with a parameterized Ansatz, a so-called trial wave function $\psi(X;\theta)$, which is iteratively optimized until a satisfactory accuracy (in terms of energy) is achieved. Different Ansätze have varying capacities to express wave functions, resulting in different possible minimal energy wave functions. The greater the capacity of an Ansatz to model the true wave function, the better the approximation to the ground state, in general.

A popular class of variational methods - Variational Monte Carlo (VMC) - relies on random sampling of the configuration space in order to estimate expectation value of the Hamiltonian (computing loss function) as

$$\mathcal{L}(\theta) = \frac{\langle\psi(\theta)|\,\hat{H}\,|\psi(\theta)\rangle}{\langle\psi(\theta)|\psi(\theta)\rangle} = \frac{\int dX |\psi(X;\theta)|^2 E_L(X;\theta)}{\int dX |\psi(X;\theta)|^2}, \tag{8}$$

where $E_L(X;\theta) = \psi^{-1}(X;\theta)\hat{H}\psi(X;\theta)$ is local energy, which for molecular/atomic Hamiltonians is convenient to express in log-domain as

$$E_L(X';\theta) = -\frac{1}{2}\Big[\hat{\nabla}^2 \log|\psi(X;\theta)|\big|_{X'} + \big(\hat{\nabla}\log|\psi(X;\theta)|\big|_{X'}\big)^2\Big] + V(X'). \tag{9}$$

$(\hat{\nabla}\cdot)^2$ is the inner product of the nabla operator $\left(\frac{\partial\cdot}{\partial r_{1,x}}, ..., \frac{\partial\cdot}{\partial r_{n_e,z}}\right)$ with itself.

The integral (8) is an expectation value of the sampled configurations X,

$$\int dX |\psi(X;\theta)|^2 E_L(X;\theta) = \mathbb{E}_{X\sim p(X;\theta)}\Big[E_L(X;\theta)\Big] \tag{10}$$

The expectation in Equation (10) is approximated by a Monte-Carlo estimate,

$$\mathbb{E}_{X\sim p(X;\theta)}\Big[E_L(X;\theta)\Big] \approx \frac{1}{N}\sum_{i=1}^{N} E_L(X_i;\theta), \tag{11}$$

where we introduce configuration probability $p(X;\theta) \propto |\psi(X;\theta)|^2$. Samples (also referred to as walkers and are represented by X) are generated from the wave function distribution via the Metropolis Hastings Monte Carlo method. In order to update the parameters θ and improve the wave function, one needs to compute gradients of the loss function with respect to θ denoted $\Delta\mathcal{L}(\theta)$. The parameters θ of the wave function are optimized using some form of gradient descent and computed via

$$\Delta\mathcal{L}(\theta) = \mathbb{E}_X\Big[(E_L(X;\theta) - \mathbb{E}_X[E_L(X;\theta)])\hat{\nabla}\log|\psi(X;\theta)|\Big] \tag{12}$$

and estimated through sampling of the configuration space. This procedure allows us to get close to the ground state $\psi_0(X)$, however it strongly relies on the parameterized Ansatz and ease of computing the gradients $\Delta\mathcal{L}(\theta)$. From now on, we will omit θ parameters where it is clear from the context, and introduce Fermi Net as an Ansatz that provides powerful parameterization.

B. Fermionic Neural Network Ansatz

The Fermionic Neural Network (Fermi Net) is a neural network designed specifically for the task of representing the wave function, in continuous Euclidean space, of the Schrödinger equation for a fermionic Hamiltonian. In this section we describe the original model, found in Reference [7], and in Section III A we describe Fermi Net*, which is functionally identical but better performing (faster) than the original.

At a high level, the Fermi Net consists of

I learnable single electron features (single streams),

II learnable electron-electron interaction features (pairwise streams),

III permutation equivariant operations (EQV),

IV Slater determinant multi-electron orbitals.

To a lesser extent the implementation and optimization details are necessary to the performance and usage. As such they are characteristic of the Fermi Net implementation:

VI KFAC optimization;

VII and stable log-domain computation of the amplitudes, first order derivatives and second order derivatives.

Next, in this background, we give an overview of the Fermi Net model, describing I-IV. Then we detail KFAC, describing VI. Details on VII, including the LogSumExp trick and derivations of the derivatives of the determinant, can be found in the appendix of Reference [7] or understood from other references [18].

Ansatz

There are two sets of streams in the network referred to as the single and pairwise streams. These streams contain the data corresponding to single, $\mathbf{h}_i^{l\alpha}$, and pairwise,

$\mathbf{h}_{ij}^{l\alpha\beta}$, electron features, respectively. These variables are indexed by l, the layer of the network, and i and j, the electron indexes. α is the spin of electron i and β is the spin of electron j.

The inputs to the network indexed by $l = 0$ are computed from the system X:

$$\mathbf{h}_i^{0\alpha} = (\mathbf{r}_i - \mathbf{R}_0, \|\mathbf{r}_i - \mathbf{R}_0\|, \mathbf{r}_i - \mathbf{R}_1, \|\mathbf{r}_i - \mathbf{R}_1\|, ..., \mathbf{r}_i - \mathbf{R}_n, \|\mathbf{r}_i - \mathbf{R}_n\|), \tag{13}$$

$$\mathbf{h}_{ij}^{0\alpha\beta} = (\mathbf{r}_i - \mathbf{r}_j, \|\mathbf{r}_i - \mathbf{r}_j\|). \tag{14}$$

where $\mathbf{r}_i$ and $\mathbf{R}_j$ are the electron and atom position vectors, as shown in Figure 1, and $\|\cdot\|$ is the Euclidean norm.

The data from the streams at each layer are transformed by a permutation equivariant function. These functions ensure that exchanging indexes of the data inputs (the electron positions) results in an equivalent exchange of index on the network outputs (before the Slater determinants). The values of the outputs do not change, but the indexes they correspond do change and are the same as the change in indexes of the inputs. As the wave function must be anti-symmetric (required by fermionic systems) these permutation equivariant functions allow the output of the body of the neural network to be fed into a Slater determinant and have all the properties required for a fermionic Ansatz. The inputs to the single stream layer at layer l, $\mathbf{f}_i^{l\alpha}$, is constructed as

$$\mathbf{f}_i^{l\alpha} = \left(\mathbf{h}^{l\alpha}, \frac{1}{n_\uparrow}\sum_{\beta\neq\downarrow}\mathbf{h}_{ij}^{l\alpha\beta}, \frac{1}{n_\downarrow}\sum_{\beta\neq\uparrow}\mathbf{h}_{ij}^{l\alpha\beta}, \frac{1}{n_\uparrow}\sum_{\alpha\neq\downarrow}\mathbf{h}_i^{l\alpha}, \frac{1}{n_\downarrow}\sum_{\alpha\neq\uparrow}\mathbf{h}_i^{l\alpha}\right) \tag{15}$$

And the updates on the single and pairwise streams at layer l are computed as

$$\begin{aligned}\mathbf{h}_i^{l\alpha} &= \tanh\left(\mathbf{W}_{(l-1)}\mathbf{f}_i^{(l-1)\alpha} + \mathbf{b}_{(l-1)}\right) + \mathbf{h}_i^{(l-1)\alpha} \\ \mathbf{h}_{ij}^{l\alpha\beta} &= \tanh\left(\mathbf{V}_{(l-1)}\mathbf{h}_{ij}^{(l-1)\alpha\beta} + \mathbf{c}_{(l-1)}\right) + \mathbf{h}_{ij}^{(l-1)\alpha\beta}\end{aligned} \tag{16}$$

where $\mathbf{W}_l$, and $\mathbf{V}_l$ are weights, $\mathbf{b}_l$ and $\mathbf{c}_l$ are the biases. Residual connections are added to all layers where $\dim(\mathbf{h}^l) = \dim(\mathbf{h}^{(l-1)})$. Note, that we need to use a twice-differentiable activation function in order to compute the Laplacian as in Equation (4).

There are n_l of these parameterized layers. The outputs $\mathbf{f}_i^{L\alpha}$ are split into spin dependent data blocks. There is a linear transformation to map $\mathbf{h}_i^{L\alpha}$ to a scalars which are coefficients of the exponentials in the orbitals. Multiple determinants are generated in this way, indexed by k, which contribute to the modelling capacity of the network and the elements of the determinants are the product of the coefficient computed by the Fermi Net and the envelopes

$$\phi_{ij}^{\alpha k} = (\mathbf{w}_{Li}^{\alpha k}\mathbf{h}_j^{L\alpha} + d_{Li}^{\alpha k})\times \tag{17}$$

$$\sum_m \pi_{im}^{\alpha k}\exp\left(-|\mathbf{\Sigma}_{im}^{\alpha k}(\mathbf{r}_j^\alpha - \mathbf{R}_m)|\right) \tag{18}$$

$\mathbf{\Sigma}_{im}^{\alpha k}$ control the anisotropic (direction dependent) behaviour of the envelope whereas the inverse exponential ensures the wave function decays to zero when the electrons are large distances from the nuclei.

The determinants are constructed

$$\det\left[\mathbf{\Phi}^{\alpha k}\right] = \begin{vmatrix}\phi_{00}^{\alpha k} & \dots & \phi_{0n}^{\alpha k} \\ \vdots & & \vdots \\ \phi_{n0}^{\alpha k} & \dots & \phi_{nn}^{\alpha k}\end{vmatrix}. \tag{19}$$

and the amplitudes are computed from these

$$\psi(X) = \sum_k \omega_k \det\left[\mathbf{\Phi}^{\uparrow k}\right]\det\left[\mathbf{\Phi}^{\downarrow k}\right]. \tag{20}$$

Equation (20) is a representation of the full determinant as a product of spin up and down determinants. This representation forces off-block-diagonal elements of the full determinant to zero, when

$$\begin{aligned}&(i \in \{1, ..., n_\uparrow\} \wedge j \in \{n_\uparrow + 1, ..., n\}) \\ &\vee (i \in \{n_\uparrow + 1, ..., n\} \wedge j \in \{1, ..., n_\uparrow\}) \\ &= (i \leq n_\uparrow \wedge j > n_\uparrow) \vee (i > n_\uparrow \wedge j \leq n_\uparrow),\end{aligned} \tag{21}$$

where i and j refer to the orbital index. Finally, the sign is split from the the amplitudes and the network outputs the amplitudes in the log-domain for numerical stability

$$\log|\psi(X)| = \log\left|\sum_k \omega_k \det\left[\mathbf{\Phi}^{\uparrow k}\right]\det\left[\mathbf{\Phi}^{\downarrow k}\right]\right| \tag{22}$$

Other work [19], removes the weights ω_k in Equation (20), as they can be absorbed to weights earlier in the network, and replaces the anisotropic decay parameters $\mathbf{\Sigma}_{im}^{\alpha k}$ in Equation (18) with a single parameter, restricting the orbitals to isotropic decay. Surprisingly, the authors note this does not seem to result in a decrease in modelling accuracy but does lead to significant gains in speed. These improvements are further discussed in Section IV.

Kronecker Factored Approximate Curvature

The optimization algorithm used here is a variant approximate natural gradient descent method known as KFAC. Roughly speaking, updates (the approximate natural gradients) $\tilde{\delta}_l$ for layer l are computed as

$$\begin{aligned}\tilde{\delta}_l &= \tilde{F}_l^{-1}\mathrm{vec}(\Delta_l\mathcal{L}) \\ &= \bar{A}_l^{-1}\Delta_l\mathcal{L}\bar{S}_l^{-1},\end{aligned} \tag{23}$$

where $\tilde{F}_l^{-1}$ is the approximate inverse Fisher block, $\Delta_l\mathcal{L}$ are the gradients corresponding to weights in layer l, and $\bar{A}_l$ and $\bar{S}_l$ are the moving covariances of the left and right Fisher factors, respectively, discussed in literature [20–23]. The term Fisher block is used to indicate the elements of the Fisher Information Matrix (FIM) corresponding to a given layer in the network. In Reference [7] the authors use a reduced version of the full KFAC algorithm. They do not use adaptive damping or adaptive learning rates which are somewhat characteristic of the

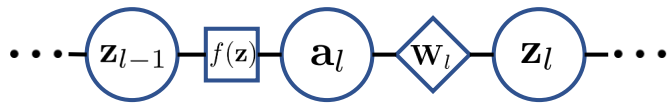


FIG. 2. Sketch of the relationship between $\mathbf{a}$ and $\mathbf{z}$. For some layer l, $\mathbf{z}_l$ are the pre-activations, $f(\mathbf{z})$ is an activation function, $\mathbf{a}_l$ are the activations and $\mathbf{w}_l$ are the weights. Data variables are in circles, functions in square and network parameters in diamond.

original algorithm [24] and considered important in the literature [20].

The KFAC variant used in this work is closer to an adaption named Kronecker Factors for Convolution [21] because the weights in most of the layers are reused. For example, the single stream weights are used n_e times, where n_e is the number of electrons in the system. There are different approaches to approximating the effect this has on the FIM and here we use the simpler and more efficient approximation developed in Reference [22]

$$\tilde{F}_l = |T|^2 \, \mathbb{E}_X \left[\mathbb{E}_i[\mathbf{a}_{li}] \, \mathbb{E}_i[\mathbf{a}_{li}]^T \right] \otimes \mathbb{E}_X \left[\mathbb{E}_i[\mathbf{s}_{li}] \, \mathbb{E}_i[\mathbf{s}_{li}]^T \right] \tag{24}$$

where ^{T} indicates the transpose, $\mathbf{a}_{li}$ are the activations of spatial location (the index of data which are operated on by the same parameters) i at layer l and $\mathbf{s}_{li}$ are the sensitivities of the pre-activations $\mathbf{z}_{li}$ of layer l

$$\mathbf{s}_{li} = \frac{d \log |\psi(X)|}{d\mathbf{z}_{li}}. \tag{25}$$

See Figure 2 for a sketch of how these variables are related. $\mathbb{E}_i$ is the expectation taken over all spatial locations

$$\mathbb{E}_i[\mathbf{a}_{li}] = \frac{1}{|T|} \sum_i \mathbf{a}_{li} \tag{26}$$

$$\mathbb{E}_i[\mathbf{s}_{li}] = \frac{1}{|T|} \sum_i \mathbf{s}_{li} \tag{27}$$

where T is the set of spatial locations and $|T|$ is its cardinality. Further details of KFAC can be found in AppendixB.

C. Diffusion Monte Carlo

Whereas VMC relies on the optimization of the parameters of the Ansatz via the derivatives of the expectation value of the energy, DMC is a projector method relying on repeated application of the imaginary time operator. Given a trial function, which has some non-zero overlap with the ground sate, repeated application of the Hamiltonian (via the power method) will project the wave function toward the ground state. Green's function Monte Carlo is an example of a projector method [25]. Diffusion Monte Carlo is a related but distinct projector method that is equivalent to a Green's function method in the limit of small time steps [6, 26, 27].

Starting with the imaginary time Schrödinger equation

$$-\frac{\partial \psi(X,t)}{\partial t} = (\hat{H} - E_T)\psi(X,t) \tag{28}$$

that can be interpreted as a diffusion equation and solved in the path integral formalism [28] and E_T is some offset. The ground state of the system is a stationary state; there are no time dynamics and the LHS is equal to zero.

Attempting to simulate the imaginary time Schrödinger equation via the application of the approximated imaginary time operator naïvely one would encounter the sign problem [8, 29], resulting from the antisymmetry constraint of exchange of electrons. The most successful approach to avoiding the sign problem in DMC simulations is the fixed node approximation [30]. In this approximation, the nodes (points in space where the wave function changes from positive to negative) are fixed in place. Practically, this is implemented by not allowing walkers to cross nodes during the sampling process.

For some state $\psi_n(X) \neq \psi_0(X)$ with eigenvalue $E_n > E_T$ we can see that the derivative of the excited states amplitudes will also be negative: The amplitudes of the states with eigenvalues $E_n > E_T$ decay as a function of time. It is clearer to see when the wave function is expanded as a linear combination eigenfunctions

$$-\frac{\partial \psi(X,t)}{\partial t} = (\hat{H} - E_T) \sum_i \alpha_i \psi_i(X,t) \tag{29}$$

and the eigenfunctions of this equation are

$$\begin{aligned} \psi(X,t) &= e^{-t(\hat{H}-E_T)} \sum_{i=0}^{\infty} \alpha_i \psi_i(X) \\ &= \sum_{i=0}^{\infty} \alpha_i e^{-t(E_i - E_T)} \psi_i(X) \end{aligned} \tag{30}$$

in the limit of $t \to \infty$ the behaviour of the wave function is dependent on E_T. There are three cases for the behaviour in the asymptotic region:

$$\lim_{t\to\infty} \psi(X,t) = \begin{cases} \infty & \text{for } E_T > E_0 \\ \alpha_0 \psi_0 & \text{for } E_T = E_0 \\ 0 & \text{for } E_T < E_0 \end{cases} \tag{31}$$

The trial energy is not known beforehand. It is set adaptively dependent on the behaviour the evolution. In DMC E_T is varied dynamically to keep the population of walkers finite.

III. METHODS

A. Fermi Net*

We change the network slightly, decreasing the resources (defined as the number of operations) required

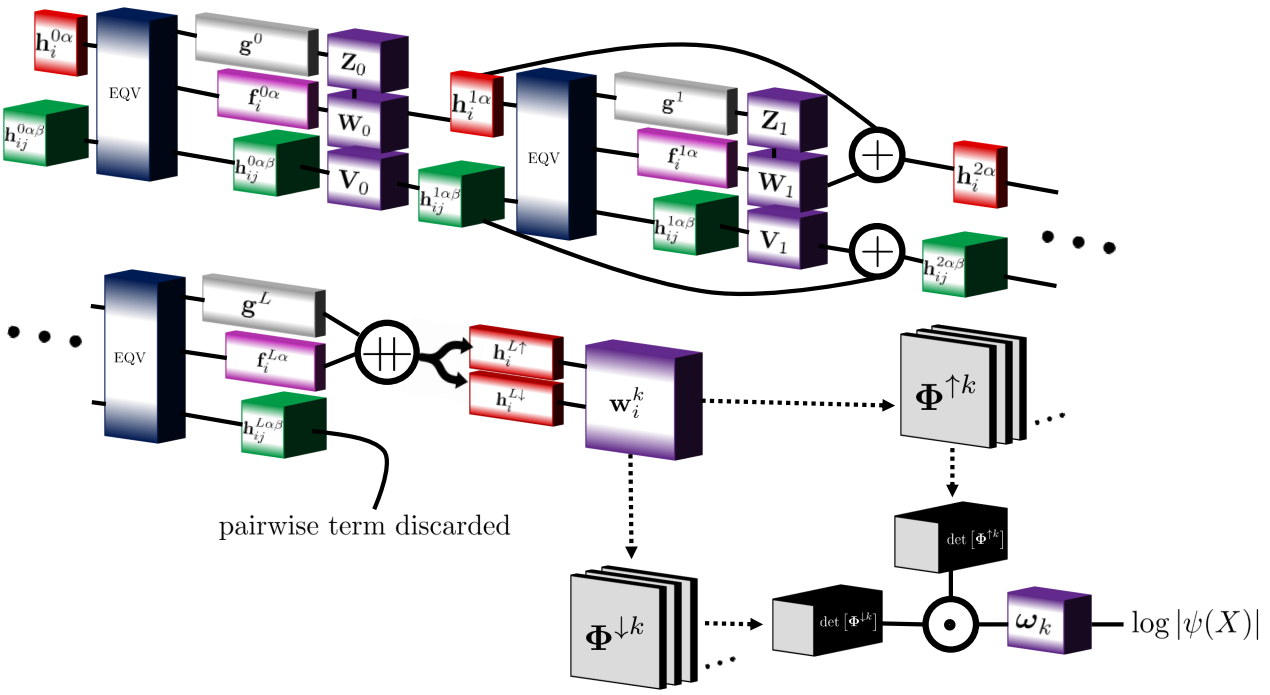


FIG. 3. Overview of Fermi Net. The system description X is used to compute the single stream and pairwise stream input feature tensors $\mathbf{h}_i^{0\alpha}$ and $\mathbf{h}_i^{0\alpha\beta}$, respectively. These are passed to the permutation equivariant function (EQV), Figure 4. Linear layers are applied to the resulting tensors with tanh activations. Outputs of the Split Stream and Single Stream matrix multiplications are combined before a tanh activation in the Single Stream layer. These layers are repeated 4 times. After the final permutation equivariant function, the Split Stream and Single Stream outputs are concatenated ($\plusplus$) and Pairwise Stream data discarded. The concatenated tensor is passed through a final spin dependent linear transformation to spin-up and spin-down determinants. The final layer is a custom computation of the determinants which involves stable first- and second-order derivatives and the LogSumExp trick.

Name	symbol	value
Single Stream hidden units	n_s	256
Pairwise Stream hidden units	n_p	32
Split Stream hidden units	n_{ss}	256
Determinants	n_k	16
Layers	n_l	4

TABLE I. Model hyperparameters.

by the original model. We refer to this implementation of the model (and distinct optimization) as Fermi Net* in order to distinguish from other work, Reference [7], which is referred to as Fermi Net. For both implementations of the model we compute the total number of operations of the implementations and compare the walltime.

The number of operations n_{ops} is computed as

$$n_{\text{ops}} = \sum_l n^l_{\text{uses}} \times d^l_{\text{out}} \times (d^l_{\text{in}} + (d^l_{\text{in}} - 1)) \tag{32}$$

where l is index running over all layers performing matrix multiplications and n^l_{uses}, d^l_{out} and d^l_{in} are the number of times those weights are used, the input dimension and the output dimension, respectively. To clarify, $d_{\text{out}} \times (d_{\text{in}} + (d_{\text{in}} - 1))$ is the cost of one matrix-vector product in terms of the number of operations (multiplication or addition). For the comparison we only compute contributions from layers that change between the two implementations and ignore other operations such as computing the determinant, which is the dominant operation in the complexity for larger systems.

It is possible to completely remove elements from the pairwise streams tensor with no effect to the modelling capacity or performance of the network. The diagonal elements of the pairwise tensor $\mathbf{h}_{ij}^{l\alpha\beta}$ where $i = j$ are redundant. The inputs computed from Equation (14) are zero, have zero contribution to computation of the energy (and therefore zero contribution to the computation of the gradients). Though this only results in a small reduction in resource, Figure 5b, we find a per iteration walltime reduction of $\sim 5 - 10\%$. The effect is more noticeable at larger system sizes. The permutation equivariant function is changed to decrease the number of operations, Figure 4. This new function outputs two

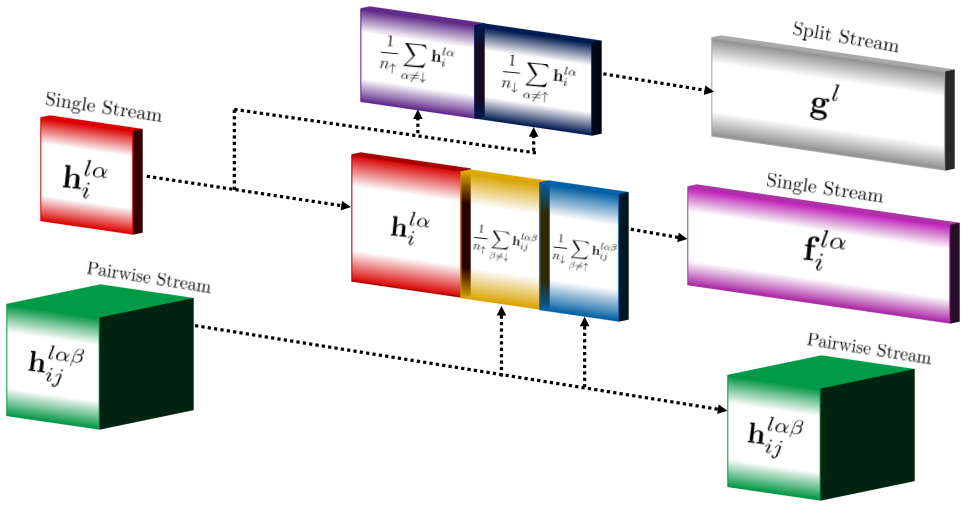


FIG. 4. Data from the single and pairwise streams are combined in this operation via Equations 33 & 34. The pairwise stream data remains unchanged.

streams of data, $\mathbf{f}_i^{l\alpha}$ and $\mathbf{g}^l$:

$$\mathbf{f}_i^{l\alpha} = \left(\mathbf{h}^{l\alpha}, \frac{1}{n_\uparrow}\sum_{\beta\neq\downarrow}\mathbf{h}_{ij}^{l\alpha\beta}, \frac{1}{n_\downarrow}\sum_{\beta\neq\uparrow}\mathbf{h}_{ij}^{l\alpha\beta}\right) \tag{33}$$

$$\mathbf{g}^l = \left(\frac{1}{n_\uparrow}\sum_{\alpha\neq\downarrow}\mathbf{h}_i^{l\alpha}, \frac{1}{n_\downarrow}\sum_{\alpha\neq\uparrow}\mathbf{h}_i^{l\alpha}\right), \tag{34}$$

This implementation requires less resources than the original, as the mean over spin terms in the layer, $\mathbf{g}^l$, are only operated on once, instead of n_e times, in the case that these data are not split and $\mathbf{g}^l$ appears in all single electron streams.

The updates at each layer l are now computed

$$\mathbf{h}_i^{l\alpha} = \tanh\left(\mathbf{W}_{(l-1)}\mathbf{f}_i^{(l-1)\alpha} + \mathbf{Z}_{(l-1)}\mathbf{g}^{(l-1)} + \mathbf{b}_{(l-1)}\right) + \mathbf{h}_i^{(l-1)\alpha} \tag{35}$$

$$\mathbf{h}_{ij}^{l\alpha\beta} = \tanh\left(\mathbf{V}_{(l-1)}\mathbf{h}_{ij}^{(l-1)\alpha\beta} + \mathbf{c}_{(l-1)}\right) + \mathbf{h}_{ij}^{(l-1)\alpha\beta} \tag{36}$$

where $\mathbf{W}_l$, $\mathbf{Z}_l$ and $\mathbf{V}_l$ are weights, $\mathbf{b}_l$ and $\mathbf{c}_l$ are the biases. As before, residual connections are added to all layers where $\dim(\mathbf{h}^l) = \dim(\mathbf{h}^{(l-1)})$. The outputs $\mathbf{f}_i^{L\alpha}$ and $\mathbf{g}^L$ are concatenated, $\mathbf{h}_i^{L\alpha} = (\mathbf{f}_i^{L\alpha}, \mathbf{g}^L)$, and split into two spin dependent data blocks.

This is an alternate but equivalent representation of the permutation equivariant function outlined in Reference [7]. This representation reduces the number of operations required to perform a forward pass in the network, Figures 5a and 5b.

Using this implementation results in a worse approximation to the FIM, discussed further in the next section, see Figure 6a, though we did not observe meaningful differences in performance between the two implementations. The reduction in computational effort (as measured by the per iteration walltime as a proxy) was not as large as the drop in resource requirements, though was significant enough ($\sim 5-10\%$) to warrant the additional complication.

A complete set of the hyperparameters of the Fermi Net* Ansatz used in these experiments are given in Table I.

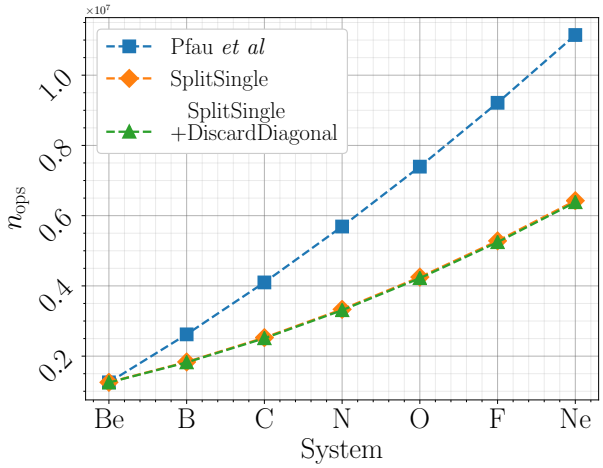
B. Kronecker Factored Approximate Curvature

In terms of the optimization, we found divergent behaviour related to adaptive damping methods in the asymptotic region of the training, most likely due to the relatively large noise on the loss and small precision requirements of this particular optimization, though more rigorous investigation will likely yield interesting and useful insight.

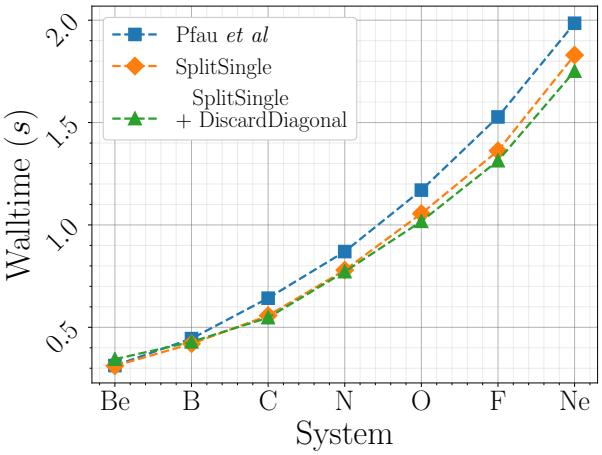
As such, we use a reduced version of KFAC. We decay the learning rate, norm constraint and damping

$$x = \frac{x_0}{1 + t \times 10^{-4}} \tag{37}$$

where x is a dummy variable which stands in for either the damping (λ), learning rate (η) and norm constraint (c) and x_0 for the initial values. t is the iteration of the optimization. The norm constraint is particularly important at the start of training when the quadratic approximation to the local optimization space is inaccurate and the natural gradients are large. In this region the norm constraint plays a role and effectively clips the gradients. Later in the optimization the approximate natural gradients are small and not constrained. The damping / learning rate interplay is an essential ingredient to a functioning KFAC implementation. The algorithm has nice convergence properties, for this problem especially, but high performance is only possible with careful tuning of the damping and learning rate.



(a) Resource comparison of different implementations. The orange line is overlayed by the green line.



(b) Reduction in average walltime resulting from the change of implementation. These values are the average of 1000 sampling steps and 100 energy computations of the model on 1 V100 GPU.

FIG. 5. The blue line corresponds to the network outlined in Reference [7], the orange line to the method of splitting the single streams, Equations 33 and 34, and the green line the resource requirements of splitting the single stream and removing redundant pairwise streams. The resource requirements are measured as the number of required operations, n_{ops}, Equation (32). The walltime comparison of these methods is shown in (b). It is important to note that the computational time of the framework is dominated by the determinant calculation. These improvements will become negligible at much larger systems.

We bundle parameters in the envelope layers by using sparse matrix multiplications. We compute the entire Fisher block for all parameters of the same symbol (i.e. the $\boldsymbol{\pi}$ and $\boldsymbol{\Sigma}$ parameters). This is illustrated in Figure 6b. Although we found this implementation was faster in practice, it performs more operations than maintaining a separation between these layers. It does not scale favorably for larger systems, though relative to the computational cost of the network may be negligible.

In Figures 6a and 6b we show the changes on the left Fisher factor resulting from using an implementation which splits data in the permutation equivariant function, versus an implementation which does not. For atomic systems, the right Fisher factor, $\bar{S}$, is the same in this new formulation. Therefore the changes in $\bar{A}$ are representative of the changes to the entire FIM.

The quality of approximation to the FIM is dependent on the data distribution and the model. In this problem and for a small model, we found that this approximation provided a better representation of the exact FIM than other methods [21] when compared on a small model over an initial optimization run of 1×10^3 iterations. It is impractical to perform the comparison to the exact FIM as the memory requirements scale as $\mathcal{O}(n_{\text{w}}^2)$, where n_{w} is the number of parameters.

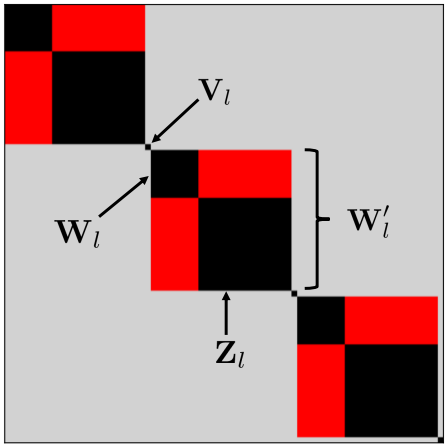
C. Variational Monte Carlo with Kronecker Factored Approximate Curvature

The goal is to create a good enough approximation of the wave function and therefore the nodal structure of the wave function using VMC to be followed up with DMC given the Fermi Net* Ansatz. The VMC requires c_l forward passes (sampling), 1 backward pass (gradient) and 1 energy computations.

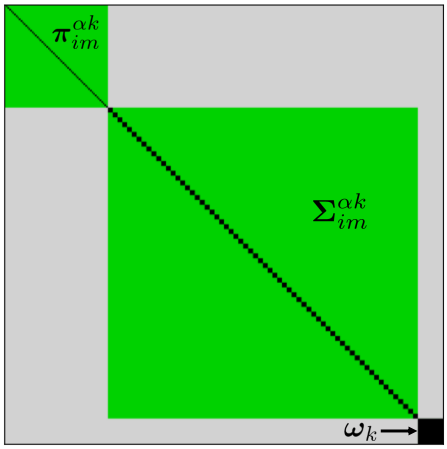
Walkers X, a point in the configuration space that moves around the space by taking random steps, are sampled via Metropolis Hastings. These moves are accepted or rejected dependent on the ratio of in probability of the starting and end points. A description of the algorithm is given in Algorithm 1.

The step size is adaptively changed during training to move the acceptance toward the target sampling acceptance ratio of 0.5. c_l is the correlation length and here was set to 10. The model is pretrained for 1×10^3 iterations using the methods outlined in Appendix A.

The gradients of the parameters are computed from Equation (12) and the centered energies $(E_L(X) - \mathbb{E}_X[E_L(X)])$ in Equation (12) have been clipped 5x from the median value.



(a) Fisher factor A of only the layers containing $\mathbf{Z}_l$, $\mathbf{W}_l$ and $\mathbf{V}_l$ parameters. $\mathbf{W}'_l$ represents the variables for computing the Fisher factor from a layer where the parameters $\mathbf{W}_l$ and $\mathbf{Z}_l$ are not split into two layers.



(b) Fisher factors A of the layers corresponding to the $\boldsymbol{\pi}^{\alpha k}_{im}$, $\boldsymbol{\Sigma}^{\alpha k}_{im}$ and $\boldsymbol{\omega}_k$ parameters.

FIG. 6. Two cartoons of the differences between the approximations to the left Fisher factor A, Equation (23), in this work and in Reference [7]. Regions in gray are not computed in either case. Black are computed in both cases. Red are regions lost in this work and green are regions gained. The variable labels (e.g. $\boldsymbol{\pi}^{\alpha k}_{im}$) are the parameters corresponding to the Fisher factor.

Algorithm 1 Metropolis-Hastings algorithm used here for sampling. c_l is the correlation length. $\mathbf{r}$ and xi are $M \times n_e \times 3$ dimensional tensors. Steps update electron positions simultaneously and acceptance of steps are performed in parallel across all walkers. Line 7 updates walkers where the condition is true. σ is the step size that is adaptively determined to maintain an acceptance ratio of 0.5.

1: **for** c_l **do**
2: $\quad \xi \sim N(0, \sigma)$
3: $\quad \mathbf{r}' \leftarrow \mathbf{r} + \xi$
4: $\quad \alpha \sim U[0, 1]$
5: $\quad P_{\text{move}} \leftarrow \frac{p(\mathbf{r}')}{p(\mathbf{r})}$
6: $\quad$ **if** $P_{\text{move}} > \alpha$ **then**
7: $\quad\quad \mathbf{r} \leftarrow \mathbf{r}'$
8: $\quad$ **end if**
9: **end for**

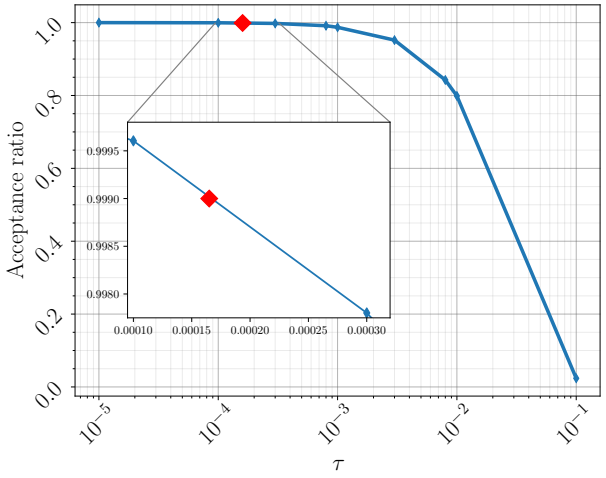


FIG. 7. The Acceptance ratio of Nitrogen as a function of the time-step τ. The red diamond marker indicates the estimated value of τ at an acceptance ratio of 0.999 (99.9%).

The approximate natural gradients for layer l, which are the updates to the Ansatz parameters, are computed via Equation (23). The approximation to the Fisher factors, $\bar{A}$ and $\bar{S}$, are 'warmed-up' by taking 100 steps of stochastic gradient descent with small learning rate ($\nu = 1 \times 10^{-5}$) whilst accumulating statistics. This ensures a smoothed initial approximation to the FIM and is consistent with other work in the literature [20].

For all systems, other than Beryllium, 1×10^5 iterations of VMC were run. The full VMC and KFAC algorithm used in this work is outlined in Algorithm 2 and a full set of hyperparameters and initialization values of variables are given in Table IIIand Table IV, respectively.

D. Diffusion Monte Carlo

The DMC algorithm implemented for this work is a simplified version of the algorithm described in Reference [6]. We make changes by removing transformations to spherical coordinate systems and set $\tau_{\text{eff}} = \tau$. Also, we move all walkers and electrons simultaneously with no change to how the algorithm behaves. We set τ by testing a non-uniform range of values for τ, computing the average acceptance over 100 iterations, and solving

the equation of a line between two points to find the closest value that gives an acceptance ratio (Acceptance) of 0.999, shown in Figure 7. This worked well in practice and all τ acceptances were $\sim 99.9\%$.

The DMC implementation is parallelized over all electrons and walkers. The model weights and walker configurations are converted to 64-bit floats for the DMC. DMC was run for a variable number of iterations, where the minimum was 5×10^4, and this is discussed further in Section V and the full algorithm used in this work is outlined in Algorithm 3 (found in Appendix D)). A complete list of the hyperparameters and variables needed given in Table V and Table VI, respectively.

E. Code and Hardware

Each VMC experiment used 2 V100 GPUs, the DMC runs were distributed over a variable number of CPUs due to constraints on the availability of GPU time.

The code was written in PyTorch 1.5 using CUDA 10.1. The implementation was parallelized such that each GPU held 2 models, or each CPU 1 model. Initial attempts at producing this model were written in Tensorflow 2 with CUDA 10.1 but we experienced significant and sometimes not (easily) diagnosable issues.

Computations of the Hartree Fock orbitals were performed using PySCF [33] and the model was distributed using Ray [34].

IV. RESULTS AND DISCUSSION

Diffusion Monte Carlo

We demonstrate functionality of the introduced computational techniques on the atomic systems from second period (Be-Ne), and the cation C^+. For each system, first we optimize parameters of the Fermi Net*, in order to generate a good trial wave function, that is subsequently improved through DMC method. As it was demonstrated in [7] Fermi Net is capable of outperforming other VMC methods. Our VMC results do not exceed the existing state-of-the-art, the DMC either exceed or match other best results (see Table II).

Table II summarizes the results. All references to Fermi Net* Ansatz indicate the methods outlined in this work. The first column of this Table contains the final energies and errors of the wave function after both VMC and DMC with the Fermi Net* Ansatz and the second column are the corresponding energies only after the VMC. The third through seventh columns (Fermi Net, VMC, DMC, HF and Exact) contain benchmark results from the literature indicated and our simulations in the case of HF. The HF column are the Hartree-Fock energies obtained using the STO-3g basis, the basis used in this work for the pretraining orbitals.

The Fermi Net* VMC energies in column 2 are not directly comparable to the Fermi Net energies. We focused computational resources on the DMC and restricted all but one system (Be) to 1×10^5 iterations, whereas the original Fermi Net work used double this number. We extended one Be to 2×10^5 iterations and found improvements to the energy, Figure 9, computed as -14.66730(1). This is at the top end of the error in the original Fermi Net result, indicating that the performance may be matched with this implementation, if slightly worse. Highlighting this result clearly does not guarantee equivalent performance on larger systems and more extensive trials on larger systems are required. The other Fermi Net* VMC results are consistently poorer than Fermi Net, but better than the referenced benchmark VMC results.

All energies in column 1 are to within less than 0.25% of the respective correlation energies, at best (Be) within 0.03%. Be, B, C^+, and N are all accurate to the exact ground state energy within error bars. Systems Be, B, C^+ and Ne are the most accurate energies. Systems Be, C^+, C and Ne were run with around 2×10^5 iterations, which are explored in more depth in Figure 9, and B, C, N, O with around 5×10^4 iterations of DMC.

The error bars are the standard error $\sigma_{\text{SEM}} = \sigma/\sqrt{m}$ where σ is the standard deviation of the energies of the batches and m is the number of batches evaluated. They are noticeably larger in column 1 of Table II for the systems mentioned where less iterations of DMC are run. DMC consistently improved the wave function, that is the evaluated energy of the wave function was lower, but the error bars are significantly larger in some cases, due to the autocorrelation of the data trace.

Figure 9 shows DMC applied to different trial wave functions. Each DMC point is evolved from the corresponding VMC iteration. The VMC lines in general show a clear trend for improving the energy of the wave function as the iterations increase, and this is mirrored in the energy computed from the DMC. There are several apparent anomalies in the data. In the Neon VMC line there is a clear decrease in the quality of the model (increase in the energy) during the training. We believe this is a result of divergence in training and may indicate that our implementation of KFAC becomes more unstable as the molecules become larger. We cannot extrapolate significant conclusions without more analysis of the optimization. In both the Neon and Carbon DMC lines there is a large kink in the energy achieved by DMC. Further investigation did not reveal divergent behaviour in the DMC, that is large changes in energy after convergence, and repeated runs showed more stable behaviour in line with what would be expected. It is possible on these particular runs the walkers became trapped near the nodes and accumulated anomalous low energy statistics. Again, further analysis and development of this DMC is required to understand if this is a feature of the precise Ansatz or issues with the DMC implementation.

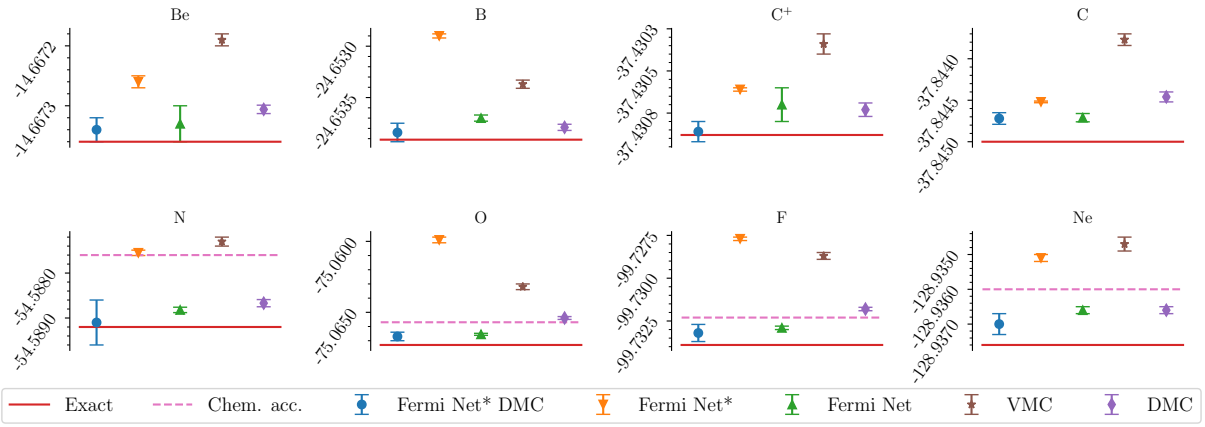


FIG. 8. Graphical representation of data from Table II, see caption for details. A chemical accuracy line (Chem. Acc. - pink dashed line) is plotted where it falls within the range of the plotted data for a system.

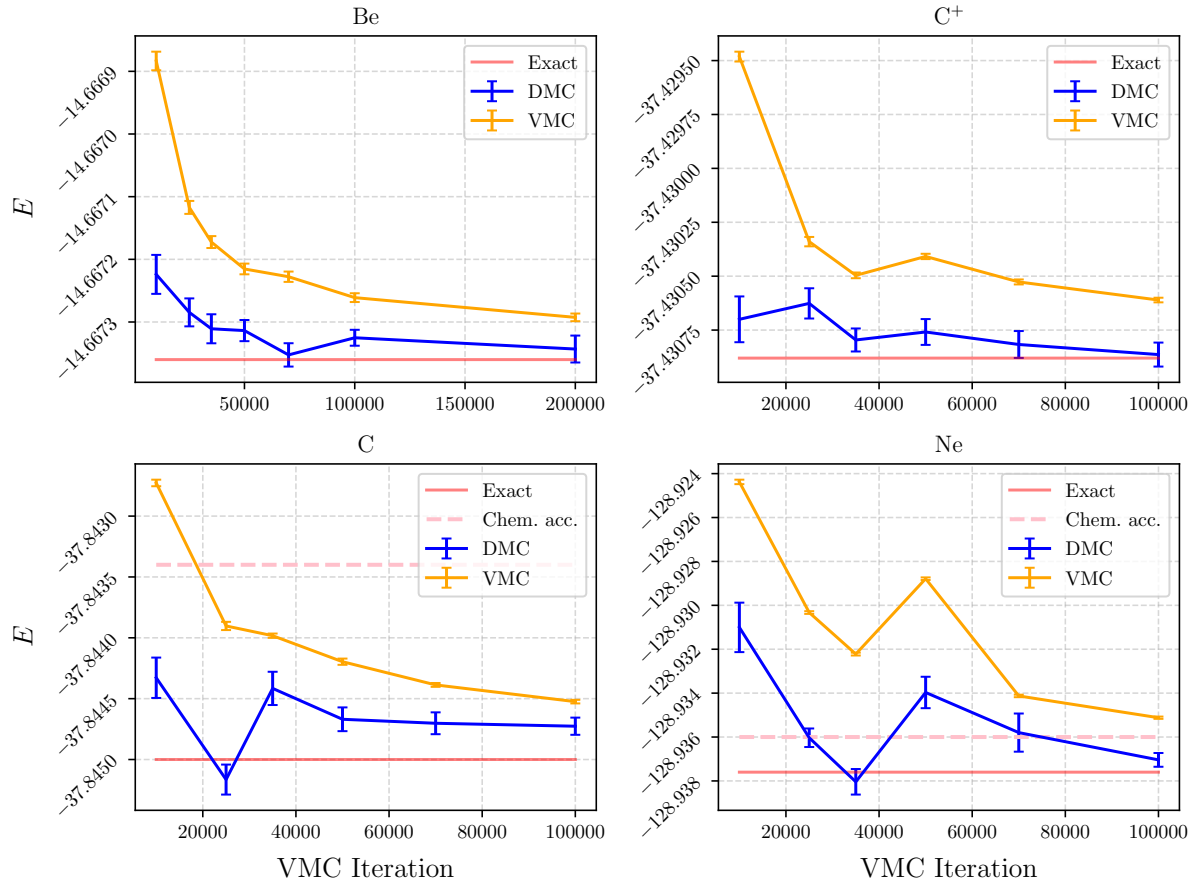


FIG. 9. DMC applied to Ansatz at different stages of VMC optimisation. The red line is the exact energy of the ground state, given in Table II. The orange line is the energy of the wave function at a particular VMC iteration. Note that Be extends to 2×10^5. It is demonstrated here to show that there is still capacity for the wave function to improve, but we were restricted by computational time for the other systems. The blue line is the resulting energy from the application of DMC to that iteration of the wave function. The DMC converged generally after around 5×10^3 iterations, and was run for between $5 \times 10^4 - 2 \times 10^5$ iterations after convergence. The error bars are the standard error $\sigma_{\text{SEM}} = \sigma/\sqrt{m}$ where σ is the standard deviation of the energies of the batches evaluated and m is the number of batches. A chemical accuracy line (pink dashed) is added to the plots where it is within the range of the plotted data.

Atom	Fermi Net* + DMC (This work)	Fermi Net* (This work)	Fermi Net [7]	VMC [31]	DMC [31]	HF	Exact [32]
Be	-14.66734(2)	-14.66726(1)	-14.66733(3)	-14.66719(1)	-14.667306(7)	-14.35188	-14.66736
B	-24.65384(9)	-24.65290(2)	-24.65370(3)	-24.65337(4)	-24.65379(3)	-24.14899	-24.65391
C^+	-37.43086(6)	-37.43061(1)	-37.4307(1)†	-37.43034(6)	-37.43073(4)	-36.87037	-37.43088
C	-37.84472(7)	-37.84452(1)	-37.84471(5)	-37.84377(7)	-37.84446(6)	-37.08959	-37.8450
N	-54.5891(5)	-54.58755(6)	-54.58882(6)	-54.5873(1)	-54.58867(8)	-53.5545	-54.5892
O	-75.0667(3)	-75.0599(2)	-75.06655(7)	-75.0632(2)	-75.0654(1)	-73.6618	-75.0673
F	-99.7332(5)	-99.7277(1)	-99.7329(1)	-99.7287(2)	-99.7318(1)	-97.9865	-99.7339
Ne	-128.9370(3)	-128.9351(1)	-128.9366(1)	-128.9347(2)	-128.9366(1)	-126.6045	-128.9376

TABLE II. Comparison of VMC and DMC results with existing works. Fermi Net* energies are computed from 1×10^4 batches of size 8096 given a model after 1×10^5 of training. The number in the brackets indicates the error on the calculation at the same precision as the value reported. For example, -14.66734(2) indicates a value of -14.66734±0.00002. All errors reported in this work are the standard error on the mean.
† Fermi Net carbon cation result not directly reported. Computed from ionisation energy and carbon energy result and the errors propagated via addition.

A. GPU, CPU and Computational Time

In this work we have used a combination of GPU and CPU compute resources. Typically, research groups may not have access to state-of-the-art compute architectures and may need to exploit alternate compute resources. The GPU methods were significantly faster than the CPU experiments, especially in the cases of larger systems, for example GPU vs. CPU per iteration times for the DMC algorithm on the system Neon were 4s and 24s, respectively, we found that in these small systems a CPU implementation distributed over 3 nodes was enough for reasonable experiment time (< 1 week). However, the scaling makes the CPU implementation impractical for larger systems. CPU implementations are not standard for the machine learning community, but are for the quantum chemistry community. We highlight here that these methods can be used on CPU architectures, though may quickly become impractical for larger systems.

One iteration of DMC is less computationally demanding than VMC, requiring 1 forward pass, 1 energy computation and 1 backward pass, plus some negligible functions. However, we port the Fermi Net weights to 64-bit precision for the DMC phase, finding significantly better performance. This is increases the walltime of 1 iteration of DMC to 1-1.5x 1 iteration of VMC.

V. CONCLUSIONS

In this work we have changed the structure of a neural network Ansatz, the Fermi Net, by removing redundant elements (the diagonal elements of the pairwise streams) and splitting the data in the permutation equivariant function such that it is not reused unnecessarily. These changes increase the performance (as measured by the walltime) of the network. Additionally, we have improved approximations to the ground state, found with Variational Monte Carlo and a Fermi Net* Ansatz, with Diffusion Monte Carlo, matching or exceeding state-of-the-art in all systems.

With respect to the first contribution, although this model is small compared with other state-of-the-art neural networks [35], it contains the expensive determinant computation, which is the dominant term in the complexity of the network scaling as $\mathcal{O}(n_e^3 k)$, where n_e is the number of electrons and k is the number of determinants. Variational Monte Carlo requires computation of the Laplacian, which uses n_e backward passes of the first order derivatives. In total, the estimated complexity of the network is $\mathcal{O}(n_e^4 k)$. Although the changes made here improve the performance, this improvement in speed may be negligible at larger system sizes.

A. Related Work

Other more recent work [19] additionally found significant efficiency gains in altering the neural network with no noticeable reduction in accuracy: replacing the anisotropic decay parameters with isotropic decay parameters in the envelopes; and removing more redundant parameters in the determinant sum. Further, other smaller networks employing a more traditional Jastorw/backflow Ansatz [9] and integrated Hartree-Fock orbitals were significantly less computationally demanding at the cost of notably worse performance. There are clearly still gains to be made in the efficiency of these methods, and a better understanding on the relationship between the problem and the size of the network required will be important pieces of understanding for tuning these methods.

B. Future Work

In this work we alternate between 32-bit and 64-bit computational precision for the Variational Monte Carlo and Diffusion Monte Carlo methods, respectively. It may be possible to exploit this further. One example is mixed

precision algorithms and networks [36]. A suitable and easy first step might be mixed precision schedules. Using low precision weights earlier in training and switching to high precision in the asymptotic region of the optimization might encourage even better results, for example improving the approximation to the Fisher Information Matrix in the region where there are small variations in the optimization landscape. In early tests, reducing the algorithm and model to 16-bit precision proved unstable by generating singular Fisher blocks (that could not be inverted).

In terms of the optimization, KFAC is a powerful algorithm which requires careful tuning. The methods used in this work are relatively simple, notably the schedule of the learning rate and damping, and not representative of the suggested mode of operation. Stable adaptive techniques will be essential to improving the optimization in this domain. Additionally, the efficiency of KFAC can be improved further by using intuitions about the FIM. For example, the FIM will change less in the asymptotic region of the optimization, schedules which exploit this fact and update the Fisher blocks and their inverses less should be used, as outlined in the literature [20].

Finally, there are existing approximations from VMC literature which can be easily integrated into these frameworks, such as pseudopotentials (for scaling to larger systems) [37], and alternate representations of the wave function [38, 39]. Additionally, these new and highly precise techniques can be applied to other systems including solids and other interesting Hamiltonians.

C. Final comment

We conclude that though these results are comparatively good in isolation, the prospects for improvements in these methods and hardware paint a strong future for the application of neural networks in modelling wave functions in the continuum.

ACKNOWLEDGEMENTS

We thank David Pfau for helpful discussions. We are grateful for support from NASA Ames Research Center and from the AFRL Information Directorate under grant F4HBKC4162G001. F.W. was supported by NASA Academic Mission Services, contract number NNA16BD14C.

APPENDICES

Appendix A: Pretraining

The pretraining was performed very similarly to as described in Reference [7]. We outline the methods here for completeness.

The pretraining loss is

$$\mathcal{L}^{\text{pre}} = \int \Bigg[\sum_{\alpha\in\{\uparrow,\downarrow\}} \sum_{ijk} \Big(\phi_{ij}^{k\alpha}(X) - \phi_{i\alpha}^{\text{HF}}(\mathbf{r}^{\beta}) \Big)^2 \Bigg] p^{\text{pre}}(X) dX \tag{A1}$$

where

$$p^{\text{pre}} = \frac{1}{2} \Bigg(\prod_{\alpha\in\{\uparrow,\downarrow\}} \prod_i \phi_{i\alpha}^{\text{HF}}(\mathbf{r}^{\beta}) + |\psi(X)|^2 \Bigg), \tag{A2}$$

with $\phi_{i\alpha}^{\text{HF}}$ being Hartee-Fock orbitals obtained with STO-3g basis evaluated with PySCF python package.

This quantity is approximated by splitting the samples (at random) into two equal length sets and taking one Metropolis Hastings step with $p(X) = \prod_{\alpha\in\{\uparrow,\downarrow\}} \prod_i \phi_{i\alpha}^{\text{HF}}(\mathbf{r}^{\alpha i})$ for the first set and $p(X) = |\psi(X)|^2$ for the second set. The sets are joined and the process repeated. This implementation is slightly different to the original work, and seems to improve the pretraining: The Ansatz (after pretraining) has a lower energy.

As described in Reference [7], this distribution samples where the Hartree Fock (HF) orbitals are large, and also where the wave function is poorly initialized and is incorrectly large. This method of pretraining is a better solution to the problem of guiding the initial wave function. An Ansatz close to the HF orbitals seems to not get stuck in local minima and drastically improves the optimization behaviour. Other solutions include integrating the HF orbitals into the network throughout the whole training, for example in Reference [9]. The optimization was performed using an Adam optimizer with default hyperparameters, notably learning rate 1×10^{-3}.

Appendix B: Kronecker Factored Approximate Curvature

Natural gradient descent [40], is an algorithm for computing the natural gradients of a parameterized function. They are

$$\boldsymbol{\Delta} = F^{-1}\text{vec}(\Delta\mathcal{L}) \tag{B1}$$

where $\boldsymbol{\Delta}$ are the natural gradients, $\Delta\mathcal{L}$ are the gradients of the function parameters, $\text{vec}(M)$ is the vectorization of a matrix M, and F is the FIM

$$F = \mathbb{E}_X \left[\frac{d\log p_\theta(X)}{d\theta}^T \frac{d\log p_\theta(X)}{d\theta} \right], \tag{B2}$$

where $p_\theta(X)$ is some distribution parameterized by θ. KFAC is an algorithm that approximates the natural gradients of a neural network. There are a series of approximations and methods associated with this algorithm, which are often applied in a case dependent way. The main drawback of Natural Gradient Descent is the cost of inverting the FIM, which for an $n \times n$ matrix is $\sim O(n^3)$. This is prohibitive for neural networks with even a few thousand parameters. KFAC reduces this burden whilst still accurately (enough) modelling the Fisher such that useful approximate natural gradients can be found.

Given that the inverse of a Kronecker product is the Kronecker product of the inverses

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{B3}$$

the authors break down the FIM into blocks, extract the most important blocks, and approximate these blocks in a form that allows the identity of inverting Kronecker products, Equation (B3), to be used.

Expressing the FIM as blocks it is natural to use the structure of the neural network to dictate the layout of the blocks:

$$F = \begin{pmatrix} \mathbb{E}[D\theta^{(0)} D\theta^{(0)T}] & \cdots & \mathbb{E}[D\theta^{(0)} D\theta^{(L)T}] \\ \vdots & \ddots & \vdots \\ \mathbb{E}[D\theta^{(0)} D\theta^{(L)T}] & \cdots & \mathbb{E}[D\theta^{(L)} D\theta^{(L)}] \end{pmatrix} \tag{B4}$$

where

$$D\theta^{(i)} = \frac{d\log p_\theta(X)}{d\theta^{(i)}} \tag{B5}$$

the $^{(i)}$ superscript denotes the parameters in layer i of the network.

Removing all but the diagonal blocks we have

$$\tilde{F} = \begin{pmatrix} \mathbb{E}[D\theta^{(0)} D\theta^{(0)T}] & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \mathbb{E}[D\theta^{(L)} D\theta^{(L)}] \end{pmatrix} \tag{B6}$$

Each of these blocks can be reformulated as the Kronecker product of the outer product of the activtations and sensitivities, for example layer 0,

$$\begin{aligned} F^{(0)} &= \mathbb{E}_X[D\theta^{(0)} D\theta^{(0)T}] \\ &= \mathbb{E}_X[(\mathbf{a}_0 \otimes \mathbf{s}_0)(\mathbf{a}_0 \otimes \mathbf{s}_0)^T] \\ &= \mathbb{E}_X[\mathbf{a}_0\mathbf{a}_0^T \otimes \mathbf{s}_0\mathbf{s}_0^T] \end{aligned} \tag{B7}$$

and finally assuming that the expectation of Kronecker product is approximately equal to the Kronecker product of the expectations, which has no name so we simply call the KFAC approximation here, though technically more accurately is independence of the covariances of the activations and covariances of the sensitivities,

$$F^{(l)} \overset{\text{KFAC}}{\approx} \mathbb{E}_X[\mathbf{a}_l^T \mathbf{a}_l] \otimes \mathbb{E}_X[\mathbf{s}_l^T \mathbf{s}_l] \tag{B8}$$

$$= A_l \otimes S_l. \tag{B9}$$

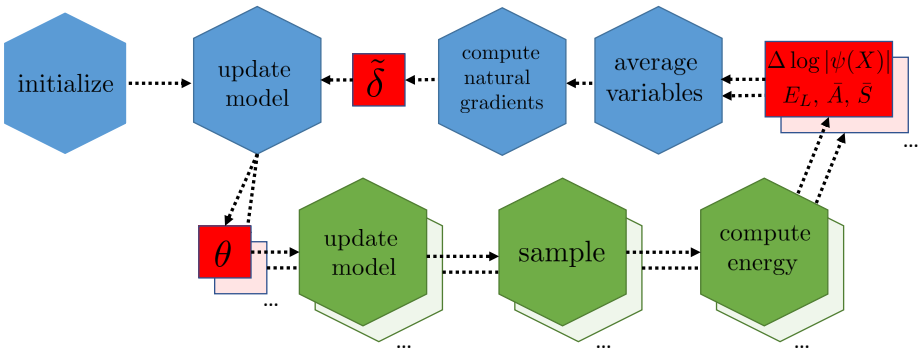


FIG. 10. Flow diagram of distributed KFAC used in this work. The blue hexagons are the head worker, green hexagons workers. There are multiple workers, indicated by the staggered images and ellipses. The red squares are variables computed at one step and used at the next. θ are the parameters of the model, $\tilde{\delta}$ are the approximate natural gradients, $\Delta \log|\psi(X)|$ are the derivatives of the wave function wrt the electron position vectors and E_L are the local energies of the walkers X. These are tensors with M copies, where M is the batch size on a worker. $\bar{A}$ and $\bar{S}$ are the left and right Fisher factors computed during the forward and backward passes. These variables are averaged before being used to compute the approximate natural gradients. Note that the average $\bar{A}$ and $\bar{S}$ are *not* passed back to the workers. This results in a worse approximation to the Fisher, but a difference in performance was not noticed during tests.

In practice, it is common to smooth the approximation to the Fisher in time. At any iteration in the optimization the quality of the approximation is limited by the number of samples in a batch. This can be unstable. Instead, the approximation is smoothed by replacing the left and right Fisher factors with their moving averages. There are different methods for implementing this exponentially decaying average of the history, in this work it is computed

$$\bar{A}_{lt} = (\varkappa_t \bar{A}_{l(t-1)} + \kappa A_{lt})/(\varkappa_t + \kappa) \tag{B10}$$

where

$$\varkappa_t = \begin{cases} 0 & \text{if } t = 0 \\ 1 & \text{otherwise} \end{cases} \tag{B11}$$

and κ is the covariance state decay and is set to 0.95. $\bar{A}_{l0}$ is a matrix of zeros and A_{lt} is the instantaneous covariances computed from Equation (B8). The right Fisher factor is computed in the same way. It is important to be careful here that the contribution to the exponentially decaying average of the moving covariances from the zeroth term is zero, otherwise the following computed averages will be zero biased. The first update step is at $t = 1$.

The approximate natural gradients of each Fisher block can be computed as Equation (B1),

$$\tilde{\delta}_l = \tilde{F}^{(l-1)} \text{vec}(\Delta \mathcal{L}_l) \tag{B12}$$

We use $\tilde{\delta}$ to indicate approximate natural gradients throughout.

1. Damping

This form of KFAC is unstable for two reasons: The approximate natural gradients can be large and the Fisher blocks are potentially singular (thus the inversion can't be performed).

Damping is a tool for shifting the eigenspectrum of a matrix (such that the matrix is non-singular) and also restricting the 'trust region' of the updates. The inverse Fisher described in Equation (B12) is adjusted, typically in Tikhonov damping,

$$\bar{\Delta}_l = (\tilde{F}^{(l)} + \lambda)^{-1} \text{vec}(\Delta \mathcal{L}_l) \tag{B13}$$

This can no longer be decomposed as Equation B3. An adapted technique, Factored Tikhonov Damping described in detail in Reference [20], is used which approximates the true damping

$$(\tilde{F}^{(l)} + \lambda)^{-1} \overset{\text{FT}}{\approx} (A + \sqrt{\pi\lambda}) \otimes (S + \sqrt{\lambda/\pi}) \tag{B14}$$

where

$$\pi_l = \frac{\text{Tr}[A_l] \dim(S_l)}{\text{Tr}[S_l] \dim(A_l)} \tag{B15}$$

This method is derived from minimizing the residual from the expansion of damped Fisher block into Kronecker factors.

2. Centering

All approximations to the Fisher block have thus far been computed via the derivatives of the log-likelihood of the normalized distribution $p(X)$ with respect to the weights. However, we compute $\mathcal{O} = \frac{d \log|\psi(X)|}{d\theta}$ which are the derivatives of the log unnormalized wave function $\psi(X)$. It can be shown, for example Appendix C of

Reference [7] that elements in the FIM can be expressed

$$F_{ij} \propto \mathbb{E}_X \left[(\mathcal{O}_i - \mathbb{E}_X[\mathcal{O}_i])(\mathcal{O}_j - \mathbb{E}_X[\mathcal{O}_j]) \right] \tag{B16}$$

$$= \frac{1}{4} \mathbb{E}_X \left[\frac{d \log p(X)}{d\theta_i} \frac{d \log p(X)}{d\theta_j} \right] \tag{B17}$$

Where the derivatives of the log wave function have been centered by $\mathbb{E}_X[\mathcal{O}]$. However, in tests we found that the performance was more dependent on the hyperparameters used in KFAC than on the centering. There are many approximations affecting the quality of the approximation to the FIM and the interplay between the optimization and this approximation. We ignored the centering to simplify the implementation but outline the method here of approximating this centering for readers who are interested.

$$\begin{aligned} F &\propto \mathbb{E}_X \left[(\mathcal{O} - \mathbb{E}_X[\mathcal{O}])^T (\mathcal{O} - \mathbb{E}_X[\mathcal{O}]) \right] \\ &= \mathbb{E}_X \Big[\mathcal{O}^T \mathcal{O} - \mathbb{E}_X[\mathcal{O}]^T \mathcal{O} - \mathcal{O}^T \mathbb{E}_X[\mathcal{O}] \\ &\quad - \mathbb{E}_X[\mathcal{O}]^T \mathbb{E}_X[\mathcal{O}] \Big] \\ &\overset{\text{LINEAR}}{=} \mathbb{E}_X[\mathcal{O}^T \mathcal{O}] - \mathbb{E}_X[\mathcal{O}]^T \mathbb{E}_X[\mathcal{O}] \end{aligned} \tag{B18}$$

$$\tag{B19}$$

The second term can be approximated by assuming Independent Activations and Sensitivities (IAD):

$$\mathbb{E}_X[\mathcal{O}]^T \mathbb{E}_X[\mathcal{O}] = \mathbb{E}_X[\mathbf{a} \otimes \mathbf{s}]^T \mathbb{E}_X[\mathbf{a} \otimes \mathbf{s}] \tag{B20}$$

$$\overset{\text{IAD}}{\approx} (\mathbb{E}_X[\mathbf{a}] \otimes \mathbb{E}_X[\mathbf{s}])^T (\mathbb{E}_X[\mathbf{a}] \otimes \mathbb{E}_X[\mathbf{s}]) \tag{B21}$$

$$= \mathbb{E}_X[\mathbf{a}]^T \mathbb{E}_X[\mathbf{a}] \otimes \mathbb{E}_X[\mathbf{s}]^T \mathbb{E}_X[\mathbf{s}] \tag{B22}$$

$$= A' \otimes S'. \tag{B23}$$

Overall, the centering outlined in Equation (B18) can be approximated by mean centering the activations and sensitivities in Equation (B8)

$$\bar{\mathbf{a}} = \mathbf{a} - \mathbb{E}_X[\mathbf{a}] \tag{B24}$$

$$\bar{\mathbf{s}} = \mathbf{a} - \mathbb{E}_X[\mathbf{s}] \tag{B25}$$

or alternately maintaining a moving average of A' and S' and subtracting those directly from $\bar{A}$ and $\bar{S}$ in Equation (B10). Both these methods result in the same residual, which can be computed by expanding either expression for the resultant approximate FIM.

Appendix C: Variational Monte Carlo

Table III contains the hyperparameters and Table IV contains the variables, and initial values, for both the VMC and KFAC methods used in this work. The entire algorithm, the update loop for the model, is given in Algorithm 2.

Name	symbol	initial value
number iterations	$k_{\max}^{\text{VMC}}$	1×10^5
learning rate	η	1×10^{-3}
norm constraint	c	1×10^{-4}
covariance state decay	κ	0.95
damping	λ	1×10^{-4}
variable decay	ν	1×10^{-4}
correlation length	c_l	10
sampling step size	σ	0.02^2
batch size	m	4096

TABLE III. Table containing all hyperparameters used in the VMC algorithm using KFAC updates in this work.

Name	symbol	initial value
iteration	k	0
damping factor	π	N/A
instantaneous activations	$\mathbf{a}$	N/A
instantaneous sensitivities	$\mathbf{s}$	N/A
energy gradients	$\Delta\mathcal{L}$	N/A
approx. natural gradients	$\tilde{\delta}$	N/A
spatial locations	i	N/A
decaying average left Kronecker factor	$\bar{A}_l$	$\mathbf{0}$
decaying average right Kronecker factor	$\bar{S}_l$	$\mathbf{0}$
instantaneous left Kronecker factor	A_l	N/A
instantaneous right Kronecker factor	S_l	N/A

TABLE IV. Table containing all variables in the VMC algorithm using KFAC updates in this work including their initial values.

Appendix D: Diffusion Monte Carlo

Table VI contains the hyperparameters and Table VI the variables, and initial values, used in the version of DMC implemented for this work. The entire algorithm is outlined in Algorithm 3.

Name	symbol	initial value
number iterations	$k_{\max}^{\text{DMC}}$	variable
batch size	M	4096

TABLE V. Hyperparameters used in the diffusion Monte Carlo method.

Name	symbol	initial value
Forces	$\mathbf{F}$	N/A
Green's function	$G(\mathbf{r}, \mathbf{r}')$	1.
trial energy	E_T	N/A
acceptance (rejection) probability	p (q)	N/A
weights	$\boldsymbol{\omega}$	$\mathbf{1}$

TABLE VI. Variables in the diffusion Monte Carlo method.

Algorithm 2 KFAC for Fermi Net*. **CholeskyInverse** indicates the Cholesky Inversion, **FactoredTikhonov** is the technique for computing π given by Equation (B15), and **Clip** indicates clipping a batch of values to within 5x of the median value. A high level overview of the distribution of this algorithm across multiple workers is shown in Figure 10

for $k_{\text{max}}^{\text{VMC}}$ **do**

  Update walker coordinates X ▷ Metropolis Hastings
  $\tilde{E}_L = \textbf{Clip}(E_L(X) - \mathbb{E}_X[E_L(X)])$ ▷ energy
  $\Delta\mathcal{L} = \mathbb{E}_X\left[\tilde{E}_L \nabla \log|\psi(X)|\right]$ ▷ backward pass
  Compute $\mathbf{a}$ ▷ forward pass
  Compute $\mathbf{s}$ ▷ backward pass

  for all layers l **do**

    $\bar{\mathbf{a}}_l \leftarrow \mathbb{E}_i[\mathbf{a}_{li}]$
    $A_l \leftarrow \mathbb{E}_X[\bar{\mathbf{a}}_l^T \bar{\mathbf{a}}_l]$
    $\bar{A}_l \leftarrow \kappa \bar{A}_l + (1-\kappa) A_l$

    $\bar{\mathbf{s}}_l \leftarrow \mathbb{E}_i[\mathbf{s}_{li}]$
    $S_l \leftarrow \mathbb{E}_X[\bar{\mathbf{s}}_l^T \bar{\mathbf{s}}_l]$
    $\bar{S}_l \leftarrow \kappa \bar{S}_l + (1-\kappa) S_l$

    $\pi_l \leftarrow \textbf{FactoredTikhonov}(\bar{A}_l, \bar{S}_l)$
    $\lambda_A \leftarrow (\frac{\lambda}{\pi \times |T|^2})^{1/2}$
    $\lambda_S \leftarrow (\frac{\lambda \times \pi}{|T|^2})^{1/2}$

    $\bar{A}_l^{-1} \leftarrow \textbf{CholeskyInverse}(\bar{A}_l + I\lambda_A)$
    $\bar{S}_l^{-1} \leftarrow \textbf{CholeskyInverse}(\bar{S}_l + I\lambda_S)$

    $\tilde{\delta}_l \leftarrow \bar{A}_l^{-1} \frac{\Delta_l \mathcal{L}}{|T|^2} \bar{S}_l^{-1}$

  end for

  $\epsilon \leftarrow \min\left(1, \sqrt{\frac{c}{\sum_l \tilde{\delta}_l \Delta_l \mathcal{L}}}\right)$

  for all layers l **do**

    $\theta_l \leftarrow \theta_l - \epsilon \times \eta \times \tilde{\delta}_l$ ▷ Update the model

  end for

  $\eta \leftarrow \frac{\eta_0/\nu}{1+k}$
  $\lambda \leftarrow \frac{\lambda_0/\nu}{1+k}$
  $c \leftarrow \frac{c_0/\nu}{1+k}$

end for

Algorithm 3 Diffusion Monte Carlo. $\mathbf{r}$, $\boldsymbol{\xi}$ and $\mathbf{F}$ are $M \times n_e \times 3$ dimensional tensors. Steps update electron positions simultaneously and acceptance of steps are performed in parallel across all walkers. Line 7 updates walkers where the condition is true.

1: Compute $\psi(X)$ ▷ forward pass
2: $\mathbf{F} \leftarrow \frac{d \log |\psi(X)|}{d\mathbf{r}}$ ▷ backward pass

3: **for** $k_{\max}^{\mathrm{DMC}}$ **do**

4: $G(\mathbf{r}, \mathbf{r}') = G(\mathbf{r}', \mathbf{r}) = 1$

5: $\boldsymbol{\xi} \sim N(0, \tau)$
6: $\mathbf{r}' \leftarrow \mathbf{r} + \tau\mathbf{F} + \boldsymbol{\xi}$

7: $X' \leftarrow \{\mathbf{R}, \mathbf{r}'\}$
8: Compute $\psi(X')$ ▷ forward pass
9: $\mathbf{F}' \leftarrow \frac{d \log |\psi(X')|}{d\mathbf{r}'}$ ▷ backward pass

10: $G(\mathbf{r}, \mathbf{r}') \leftarrow \prod_i^{n_e} \exp\left((\mathbf{r}_i - \mathbf{r}'_i - \tau\mathbf{F}'_i)^2/(2\tau)\right)$
11: $G(\mathbf{r}', \mathbf{r}) \leftarrow \prod_i^{n_e} \exp\left((\mathbf{r}'_i - \mathbf{r}_i - \tau\mathbf{F}_i)^2/(2\tau)\right)$

12: $p \leftarrow \min\left(1, \frac{|\psi(X')|^2 G(\mathbf{r}, \mathbf{r}')}{|\psi(X)|^2 G(\mathbf{r}', \mathbf{r})}\right)$
13: $q \leftarrow 1 - p$
14: Set $p = 0$ where $\mathrm{sign}(\psi(X)) \neq \mathrm{sign}(\psi(X'))$ ▷ Fixed node approximation

15: $s \leftarrow E_T - E_L(X)$
16: $s' \leftarrow E_T - E_L(X')$
17: $\omega \leftarrow \omega \times \exp\left[\tau\left[\frac{p}{2}(s' + s) + qs\right]\right]$
18: $\alpha \sim U[0, 1]$
19: **if** $p > \alpha$ **then**
20: $\psi(X) \leftarrow \psi(X')$, $\mathbf{r} \leftarrow \mathbf{r}'$, $\mathbf{F} \leftarrow \mathbf{F}'$, etc ▷ Update variables for next iteration
21: **end if**
22: **end for**

[1] O. A. Von Lilienfeld and K. Burke, Retrospective on a decade of machine learning for chemical discovery, Nature communications **11**, 1 (2020).

[2] A. Szabo and N. S. Ostlund, *Modern quantum chemistry: introduction to advanced electronic structure theory* (Courier Corporation, 2012).

[3] H. G. Kümmel, A biography of the coupled cluster method, International Journal of Modern Physics B **17**, 5311 (2003).

[4] I. Ross, Calculations of the energy levels of acetylene by the method of antisymmetric molecular orbitals, including σ-π interaction, Transactions of the Faraday Society **48**, 973 (1952).

[5] W. L. McMillan, Ground state of liquid He 4, Physical Review **138**, A442 (1965).

[6] C. Umrigar, M. Nightingale, and K. Runge, A diffusion Monte Carlo algorithm with very small time-step errors, The Journal of chemical physics **99**, 2865 (1993).

[7] D. Pfau, J. S. Spencer, A. G. d. G. Matthews, and W. M. C. Foulkes, Ab-initio solution of the many-electron Schrödinger equation with deep neural networks, arXiv:1909.02487 (2019).

[8] M. Troyer and U.-J. Wiese, Computational complexity and fundamental limitations to fermionic quantum Monte Carlo simulations, Physical review letters **94**, 170201 (2005).

[9] J. Hermann, Z. Schätzle, and F. Noé, Deep neural network solution of the electronic Schrödinger equation, arXiv:1909.08423 (2019).

[10] K. Schütt, M. Gastegger, A. Tkatchenko, K.-R. Müller, and R. J. Maurer, Unifying machine learning and quantum chemistry with a deep neural network for molecular wavefunctions, Nature communications **10**, 1 (2019).

[11] K. Schütt, P.-J. Kindermans, H. E. S. Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, Schnet: A continuous-filter convolutional neural network for modeling quantum interactions, in *Advances in neural information processing systems* (2017) pp. 991–1001.

[12] G. Carleo and M. Troyer, Solving the quantum many-body problem with artificial neural networks, Science **355**, 602 (2017).

[13] R. M. Balabin and E. I. Lomakina, Neural network approach to quantum-chemistry data: Accurate prediction of density functional theory energies, The journal of chemical physics **131**, 074104 (2009).

[14] G. B. Goh, N. O. Hodas, and A. Vishnu, Deep learning for computational chemistry, Journal of computational chemistry **38**, 1291 (2017).

[15] T. F. Cova and A. A. Pais, Deep learning for deep chemistry: optimizing the prediction of chemical patterns, Frontiers in Chemistry **7** (2019).

[16] W. Yang, L. Peng, Y. Zhu, and L. Hong, When machine learning meets multiscale modeling in chemical reactions, arXiv:2006.00700 (2020).

[17] B. Bransden and C. Joachain, Quantum mechanics,(2000), Prentice Hall..

[18] C. H. Bischof, H. M. Bücker, P. Hovland, U. Naumann, and J. Utke, Advances in automatic differentiation, Springer (2008).

[19] J. S. Spencer, D. Pfau, A. Botev, and W. Foulkes, Better, faster fermionic neural networks, arXiv:2011.07125 (2020).

[20] J. Martens and R. Grosse, Optimizing neural networks with kronecker-factored approximate curvature, in *International conference on machine learning* (2015) pp. 2408–2417.

[21] R. Grosse and J. Martens, A kronecker-factored approximate Fisher matrix for convolution layers, in *International Conference on Machine Learning* (2016) pp. 573–582.

[22] J. Ba, R. Grosse, and J. Martens, Distributed second-order optimization using kronecker-factored approximations, ICLR (2017).

[23] J. Martens, J. Ba, and M. Johnson, Kronecker-factored curvature approximations for recurrent neural networks, in *International Conference on Learning Representations* (2018).

[24] D. Pfau, private communication (2020).

[25] M. A. Lee, K. Schmidt, M. Kalos, and G. Chester, Green's function Monte Carlo method for liquid He 3, Physical Review Letters **46**, 728 (1981).

[26] W. Foulkes, L. Mitas, R. Needs, and G. Rajagopal, Quantum Monte Carlo simulations of solids, Reviews of Modern Physics **73**, 33 (2001).

[27] P. Maldonado, A. Sarsa, E. Buendía, and F. Gálvez, Quantum Monte Carlo ground state energies for the singly charged ions from li through ar, The Journal of chemical physics **133**, 064102 (2010).

[28] I. Kosztin, B. Faber, and K. Schulten, Introduction to the diffusion Monte Carlo method, American Journal of Physics **64**, 633 (1996).

[29] L. Gupta and I. Hen, Elucidating the interplay between non-stoquasticity and the sign problem, Advanced Quantum Technologies **3**, 1900108 (2020).

[30] J. B. Anderson, Quantum chemistry by random walk, The Journal of Chemical Physics **65**, 4121 (1976).

[31] P. Seth, P. L. Ríos, and R. Needs, Quantum Monte Carlo study of the first-row atoms and ions, The Journal of chemical physics **134**, 084105 (2011).

[32] S. J. Chakravorty, S. R. Gwaltney, E. R. Davidson, F. A. Parpia, and C. F. p Fischer, Ground-state correlation energies for atomic ions with 3 to 18 electrons, Physical Review A **47**, 3649 (1993).

[33] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G. K. Chan, Pyscf: the python based simulations of chemistry framework (2017).

[34] Ray (2020), `https://github.com/ray-project/ray`.

[35] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, Language models are few-shot learners, arXiv:2005.14165 (2020).

[36] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, *et al.*, Mixed precision training, arXiv:1710.03740 (2017).

[37] J. J. Quinn and J. J. Quinn, Pseudopotentials, correlations, and hierarchy states in quantum Hall systems: When the composite fermion picture works and why, Solid state communications **140**, 52 (2006).

[38] M. Bajdich, L. Mitas, G. Drobnỳ, L. Wagner, and K. Schmidt, Pfaffian pairing wave functions in electronic-

structure quantum Monte Carlo simulations, Physical review letters **96**, 130201 (2006).
[39] M. Casula and S. Sorella, Geminal wave functions with Jastrow correlation: A first application to atoms, The Journal of Chemical Physics **119**, 6500 (2003).
[40] S. Amari, Neural learning in structured parameter spaces-natural Riemannian gradient, in *Advances in neural information processing systems* (1997) pp. 127–133.