Development of a Transformed based architecture to solve the Time Independent Schrodinger Equation for many bodies

Jorge Alvaro Munoz Laredo[1]

[1]Faculty of Physics, Universidad Nacional de Ingeniería, Lima, Peru, jorge.munoz.l@uni.pe

November 22, 2025

Abstract

With accurate solutions to the many-electron Schrodinger equation all the chemistry could be derived from first principles, but analytical treatment is intractable due the intrinsic strong electron-electron correlations, anti symmetry and cusp behavior. Recently, due to its high flexibility deep learning approaches had been applied for this problem, neural wave functions models such as FermiNet and PauliNet have advanced accuracy, yet computational cost and error typically grows steeply with system size, limiting applicability to larger molecules. They also lack of strong architectures designed to capture long-range electronic correlations with scalable attention. In this work I develop the Psiformer a transformer-based ansatz that couples scalable attention with physics-aware structure. Training is formulated within Variational Monte Carlo (VMC), evaluation will be do it by comparing ground state energy against another traditional methods, I also outline design questions for further improvement, including sparsified/global attention and optimizer choices inspired by recent transformer advances.

1 Introduction

The electronic structure problem remains challenging: the wave functions which fully describes the system lives in a $3N$-dimensional space,N the number of electrons, each one lives on the 3 dimensional space. Additionally, it must satisfy specific properties due to physical laws and live on the complex space.

Although the governing laws have been known for almost a century [1], obtaining practical approximations to the quantum many-body wavefunction remains difficult. Established approaches such as density-functional theory [2], Born Oppenheimer [3], and structured variational ansätze [4] trade generality for tractability by imposing specific functional forms or approximations to correlation. These choices are effective within their regimes but can struggle for strongly correlated systems or scale poorly with electron count cite. This is where modern learning-based methods enter: instead of fixing the functional form, we learn it, while enforcing essential physics (antisymmetry, cusp behavior, permutation symmetry), here is where deep learning methods shines. This field has reshaped several scientific domains, from protein structure prediction [5] to vision modeling [6] and PDE surrogates [7]. Motivated by these successes, the community has explored neural approaches for quantum many-body problems ,seeking accurate and scalable approximations to the many-electron wavefunction [8, 9].

Thus, neural wavefunction models have emerged as a promising alternative. Architectures such as **FermiNet** [10] and **PauliNet** [11] combine flexible function approximators with determinant structures to respect antisymmetry, improving variational expressivity. However, two practical limitations persist. First, error or compute cost often scales unfavorably with the number of electrons, restricting applicability to larger molecules. Second, mechanisms for **long-range electronic correlation** central to Coulomb and exchange effects are typically implicit or expensive to capture, leading to optimization difficulty and brittle generalization.

Transformers offer an appealing direction. Self-attention provides direct, many-to-many interactions among tokens in a single layer, is highly parallelizable, and has empirically favorable scaling behavior in other domains (Natural Language Processing [12]. For electronic structure, where any electron can interact with any other and electron indices are exchangeable, attention aligns naturally with the physics: it enables global coupling without imposing an arbitrary ordering. The challenge is to embed the right **inductive biases** (distance awareness, spin structure, cusp handling) and to maintain **fermionic antisymmetry** while keeping computational cost under control.

I develop **Psiformer**, a transformer-based variational ansatz for many-electron systems [13]. Psiformer uses self-attention to construct rich per-electron features informed by electron–electron and electron–nucleus descriptors, then imposes antisymmetry explicitly via determinant-based heads. Physics-aware priors (e.g., distance/radial encodings and cusp-motivated embeddings) are incorporated to reduce sample complexity and stabilize training. The optimization is formulated within **Variational Monte Carlo (VMC)** [4] by minimizing the variational energy.

Contributions.

- A transformer-based neural wavefunction

(**Psiformer**) [13] implemented on Pytorch [14] that separates correlation modeling (attention) from fermionic symmetry (determinant heads) while embedding Coulomb-aware priors.

- A VMC training recipe with practical choices for stability (feature design, and optional natural-gradient preconditioning).

The objectives are the follow:

2 Objectives

- Obtain a model which is able to approximate the ground state energy for specific atoms and molecules.
- Compare our model against other state of the art methods to solve the many electrons Schrodinger equation.
- Look for future improvements when try to tackle larger molecules.

3 Overview

This work is structured as follow: The theoretical framework section 4 introduces the foundations of quantum many-body theory, the structure of the Schrodinger equation for many-bodies like also foundational concepts of Deep Learning that are going to be used in the development of this specific problem, section 5 introduce **Psi Former** a transformer based architecture built upon **Fermi Net**, section 6 specify the specific tools and environments used to implement this work.

4 Theoretical Framework

In this section, we develop the physical and mathematical foundation needed to understand the problem we study. We begin with the Schrödinger wave equation, which is central to non-relativistic quantum mechanics, later

4.1 The wave function and the governing physical laws

4.1.1 The Schrodinger Equation

The Schrodinger equation was presented in a series of publications made it by Erwin Schrodinger in the year 1926 [1]. There we search the complex function ψ which lives on a Hilbert space $\mathcal{H}$ called **wave function**, for a non relativistic spinless single particle this function depends on the position of the particle $\tilde{\mathbf{r}}$ and time t ($\psi(\tilde{\mathbf{r}}, t)$), the quantity $|\psi(\mathbf{r}, t)|^2$ is the **probability density** to find the particle near $\mathbf{r}$ at time t [15].

Guided by de Broglie's discover [16] of the wave particle duality and a very smart intuition Schrodinger proposed the time dependent equation (TDSE):

$$i\hbar\frac{\partial\psi}{\partial t} = \hat{H}\psi \tag{1}$$

Where i is the complex unit, $\hbar$ is the reduced Planck constant approximate to $1.054571817\cdots \times 10^{-34} J \cdot s$ and $\hat{H}$ is a hermitian linear operator called the **Hamiltonian** which represents the total energy of the system, for a single non relativistic (low energy) particle of mass m in a scalar potential $V(\mathbf{r}, t)$, it takes the form:

$$\hat{H} = \frac{\hat{\mathbf{p}}^2}{2m} + V(\mathbf{r}, t) \tag{2}$$

Where $\hat{\mathbf{p}}$ is the momentum operator and in the **position representation** it takes the form of [17]:

$$\hat{\mathbf{p}} = -i\hbar\nabla \tag{3}$$

Where ∇ is the Laplacian operator, thus the TDSE is explicitly:

$$i\hbar\,\frac{\partial\psi}{\partial t} = \left[-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r}, t)\right]\psi \tag{4}$$

The **time independent form** (TISE) could be derived from equation ,when the wave function ψ could be written like the product of two functions R and T, where R depends uniquely on the spatial term ($\mathbf{r}$) and T uniquely on the temporal (t), this is:

$$\psi(\mathbf{r}, t) = R(\mathbf{r})T(t) \tag{5}$$

Plugin this form on equation 1, you can derive that [17]:

$$T(t) = e^{-iEt/\hbar}$$

Where E, the energy of the system, is a constant. You also obtain that the spatial part is conditioned by:

$$\hat{H}R(\mathbf{r}) = ER(\mathbf{r}) \tag{6}$$

We are going to represent the spatial function R as ψ. And in this work we are going to only focus in the TDSE, when we treat with constant energy, the electron are almost always found near the lowest energy state, known as the ground state. Solutions with higher energy known as excited states are relevant to photochemistry , but in this work we will restrict our attention to ground states.

4.1.2 The many electron Schrodinger Equation

When we are considering more than one single particle we consider the spin (σ) and the interaction between particles. Thus, in its time-independent form the Schrodinger equation can be written as an eigen value problem:

$$\hat{H}\psi(\mathbf{x}_0, \ldots, \mathbf{x}_n) = E\psi(\mathbf{x}_1, \ldots, \mathbf{x}_n) \tag{7}$$

Where $\mathbf{x}_i = \{\mathbf{r}_i, \sigma\}$, $\mathbf{r}_i \in \mathbb{R}^3$ is the position of each electron and $\sigma \in \{\uparrow . \downarrow\}$ is the so called spin. It's possible model the potential energy of a many body

system (e.g atoms, molecules), we first have to consider the potential given the repulsion between electrons:

$$V_{ij} = \frac{e^2}{4\pi\varepsilon_0}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{8}$$

Here $e = 1.602\,176\,634 \times 10^{-19}$ C is the elementary charge, $\varepsilon_0 = 8.854\,187\,8128 \times 10^{-12}$ F m^{-1} is the electrical permittivity of vacuum, $\mathbf{r}_i$ is the position vector of electron i in the chosen reference frame. The potential given the attraction between the proton I and the electron i is given by:

$$V_{iI} = -\frac{1}{4\pi\varepsilon_0}\frac{eZ_I}{|\mathbf{r}_i - \mathbf{R}_I|} \tag{9}$$

Where Z_I is the atomic number of nucleus I (for instance, in a Helium atom $Z = 2$) and $\mathbf{R}_I$ is the position of that nucleus from a chosen reference frame.
The reference frame is usually taken at the **center of mass** or at the **center of the molecule**. The potential given the repulsion between the nuclei I and J (protons) is:

$$V_{IJ} = \frac{1}{4\pi\varepsilon_0}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{10}$$

To avoid writing these constants every time, quantum chemistry commonly uses **atomic units (a.u.)**. In this system, the unit of length is the **Bohr radius** a_0,
and the unit of energy is the **Hartree** E_h [18]:

$$E_h = \frac{e^2}{4\pi\varepsilon_0 a_0} \tag{11}$$

Under atomic units, $e = 1$, $4\pi\varepsilon_0 = 1$,$\hbar = 1$, and $m_e = 1$. Thus, the potential energy of a system with N_e electron and N_n nucleus can be written compactly as:

$$V = -\sum_{i=1}^{N_e}\sum_{I=1}^{N_n}\frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \sum_{i>j}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{I>J}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{12}$$

where i, j are index electrons, I, J are index nuclei.The first term represents **electron–nucleus attraction**, the second term is **electron–electron repulsion**, the third term is **nucleus–nucleus repulsion**. For the kinetic term on the Hamiltonian we need to consider two expressions: ∇_i^2 acts on the **electron** coordinates $\mathbf{r}_i$ (fast, light particles) and ∇_I^2 acts on the **nuclear** coordinates $\mathbf{R}_I$ (slow, heavy particles). Let N electrons at $\mathbf{r}_i$ and M nucleous at $\mathbf{R}_I$ with charges Z_I and masses M_I (in units of m_e) then the **Hamiltonian** can be written as:

$$\hat{H} = -\sum_{i=1}^{N}\frac{1}{2}\nabla_i^2 - \sum_{I=1}^{M}\frac{1}{2M_I}\nabla_I^2 - \sum_{i=1}^{N}\sum_{I=1}^{M}\frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \sum_{1\le i<j\le N}\frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{1\le I<J\le M}\frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{13}$$

4.1.3 Conditions of the solution

As with any differential equation, where one is searching one solution, is important to consider initial conditions (IC) and boundary conditions (BC), here we have to fulfill certain conditions that comes from physical laws.

Bosons (e.g., photons) follow **Bose–Einstein** statistics [19, 20], whereas fermions (e.g., electrons, protons) follow **Fermi Dirac** statistics [21, 22], because at the microscopic level identical particles are indistinguishable then the wave function should be equal but Pauli Exclusion [23] don't like that , the many-particle wave function must be **anti symmetric** under the exchange of any two fermions. This implies:

$$\psi(\dots, \mathbf{x}_i, \dots, \mathbf{x}_j, \dots) = -\psi(\dots, \mathbf{x}_j, \dots, \mathbf{x}_i, \dots) \tag{14}$$

We can enforce **antisymmetry** using a $N \times N$ Slater determinant [24], which involves one-particle states only (a wave function with a single input) . An interchange of any pair of particles corresponds to an interchange of two columns of the determinant; this interchange introduces a change in the sign of the determinant. For even permutations we have $(-1)^P = 1$, and for odd permutations we have $(-1)^P = -1$.

$$\Psi(\mathbf{x}_1, \dots, \mathbf{x}_N) \propto \begin{vmatrix} \phi_1(\mathbf{x}_1) & \phi_2(\mathbf{x}_1) & \cdots & \phi_N(\mathbf{x}_1) \\ \phi_1(\mathbf{x}_2) & \phi_2(\mathbf{x}_2) & \cdots & \phi_N(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(\mathbf{x}_N) & \phi_2(\mathbf{x}_N) & \cdots & \phi_N(\mathbf{x}_N) \end{vmatrix} \tag{15}$$

Where ϕ_k are orthonormal (by construction) spin orbitals. This is, for instance ($N = 2$):

$$\Psi(\mathbf{x}_1, \mathbf{x}_2) \propto \Big[\phi_a(\mathbf{x}_1)\phi_b(\mathbf{x}_2) - \phi_a(\mathbf{x}_2)\phi_b(\mathbf{x}_1)\Big]. \tag{16}$$

The potential energy becomes infinite, when two particles overlap, which places strict constraints on the form of the wave function at these points, these constraints are known as the **Kato Cusp Conditions** [25]. The cusp conditions states that the wave function must be non-differentiable at these points, and give exact values for the average derivatives at the cusps. More precisely the cusp are: Electron-nucleus cusp (electron with charge -1, nucleus charge $+Z$, reduced mass $\mu \approx 1$)

$$\lim_{riI\to 0}\left(\frac{\partial\psi}{\partial r_{iI}}\right) = -Z\psi(r_{iI} = 0)$$

Electron-electron cusp, opposite spins (charges -1, reduced mass $\mu = \frac{1}{2}$)

$$\lim_{r_{ij}\to 0}\left(\frac{\partial\psi}{\partial r_{ij}}\right) = \frac{1}{2}\psi(r_{ij} = 0)$$

Where $r_{iI}(r_{ij})$ is an electron-nuclear (electron-electron) distance, Z_I is the nuclear charge of the I-th nucleous and ave implies a spherical averaging over all directions.

This conditions can be obtained if we multiply to the ansatz by multiplying by a Jastrow factor $\mathcal{J}$which satisfies these conditions analytically [26].

So that is the problem we need to find a ψ such that it satisfies all those conditions.

4.1.4 Approximations to the problem

Is clear that find analytical solutions is practically impossible, so what people have been doing is first apply good approximations. **Born Oppenheimer approximation** [3] makes it possible to separate the motion of the nuclei and the motion of the electrons, neglects the motion of the atomic nuclei when describing the electrons in a molecule. The physical basis for the Born-Oppenheimer approximation is the fact that the mass of an atomic nucleus in a molecule is much larger than the mass of an electron (more than 1000 times) [27]. Because of this difference, the nuclei move much more slowly than the electrons. In addition, due to their opposite charges, there is a mutual attractive force of: acting on an atomic nucleus and an electron. This force causes both particles to be accelerated. Since the magnitude of the acceleration is inversely proportional to the mass, $a = f/m$ [28], the acceleration of the electrons is large and the acceleration of the atomic nuclei is small; the difference is a factor of more than 1000. Consequently, the electrons are moving and responding to forces very quickly, and the nuclei are not, then we fix the nucleus, $\mathbf{R}_I$ becomes a constant, thus the kinetic term for nucleus becomes zero. And the potential energy for the repulsion between nucleus becomes a constant.

$$\hat{H}_{\text{el}} = -\sum_{i=1}^{N} \frac{1}{2}\nabla_i^2 - \sum_{i=1}^{N}\sum_{I=1}^{M} \frac{Z_I}{|\mathbf{r}_i - \mathbf{R}_I|} + \sum_{1\leq i<j\leq N} \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} + \sum_{I<J} \frac{Z_I Z_J}{|\mathbf{R}_I - \mathbf{R}_J|} \tag{17}$$

This is the form that we are going to work with; a second useful approach to the problem is use an **Ansatz**, which is a guess solution guided by intuition, this normally depend on certain number of parameters, then the problem becomes on optimize this **Ansatz**.

4.1.5 Rayleigh Quotient

We need a way to measure how well our ansatz ψ_θ is doing, if our ansatz is "bad" then don't reflect the truly form of the system, most deep learning approaches use data sets to train their ansatz (model initialized on randoms parameters) in this case we don't need any external but just physical principles. Lets first introduce the **Rayleigh quotient**. If A is an operator and x is a state, the number:

$$R_A(x) = \frac{\langle x|A|x\rangle}{\langle x\,|\,x\rangle} \tag{18}$$

is the **expectation value** of that operator in that state. The important for us is that if ψ is a wave function and $\hat{H}$ the **Hamiltonian**, then the Rayleigh quotient:

$$R_{\hat{H}}(\psi) = \frac{\langle\psi|\hat{H}|\psi\rangle}{\langle\psi|\psi\rangle} \tag{19}$$

is the average (expected) energy of the system when it is in the state ψ [29].

4.1.6 Variational Principle

The variational principle for electronic systems states that the expectation value for the binding energy obtained using an approximate wave function and the exact Hamiltonian operator will be higher than or equal to the true energy for the system [**?**]. This idea is really powerful. When implemented it permits us to find the best approximate wavefunction from a given wavefunction that contains one or more adjustable parameters, called a trial wavefunction [30] . A mathematical statement of the variational principle is.

$$R_{\hat{H}}(\psi_{\text{ansatz}}) \geq R_{\hat{H}}(\psi_{\text{true}}) \tag{20}$$

The true ground-state wave function ψ_{true} is the one that minimizes the rayleigh quotient.

$$\psi_0 = \underset{\psi}{\operatorname{argmin}}(R_{\hat{H}}(\psi)) \tag{21}$$

So if we minimize the rayleigh quotient of our ansatz we are going to be more near of the true wave function.

4.1.7 Variational Monte Carlo

To the process that we are going to use for optimize our ansatz is called Variational Monte Carlo (VMC)[4, 31]. Now we can see to the rayleigh quotient as a loss function $\mathcal{L}$ with the form of.

$$\mathcal{L}(\Psi_\theta) = \frac{\langle\Psi_\theta|\,\hat{H}\,|\Psi_\theta\rangle}{\langle\Psi_\theta|\Psi_\theta\rangle} = \frac{\int d\mathbf{r}\Psi^*(\mathbf{r})\hat{H}\Psi(\mathbf{r})}{\int d\mathbf{r}\Psi^*(\mathbf{r})\Psi(\mathbf{r})} \tag{22}$$

Evaluate that integral is hard, another smart approach is the follow, define a probability distribution p_θ with the follow form:

$$p_\theta(x) = \frac{|\Psi_\theta(x)|^2}{\int dx'\Psi_\theta^2(x')} \tag{23}$$

Realize that for compute $p_\theta(x)$ for a specific x is complicated due the integral that appears, this is going to be important later. Defining the local energy E_L with:

$$E_L(x) = \Psi_\theta^{-1}(x)\hat{H}\Psi_\theta(x) \tag{24}$$

Then the loss becomes:

$$\mathcal{L}(\Psi_\theta) = \int \frac{\hat{H}\Psi_\theta(x)}{\Psi_\theta(x)} p_\theta(x)dx \tag{25}$$

Which is the expected value of the local energy:

$$\mathcal{L}(\Psi_\theta) = \mathbb{E}_{x\sim p_\theta}[E_L(x)] \tag{26}$$

To optimize our wave function we know by deep learning that we need evaluate that expectation and obtain the derivative for back propagation, how we make that? We are going to use the the Monte Carlo estimator [32]:

$$\mathcal{L}_\theta = \mathbb{E}_{x\sim p_\theta}[E_L(x)] \approx \frac{1}{M}\sum_{i=1}^{M} E_L(\mathbf{R}_k) \tag{27}$$

Whit $\mathbf{R}_k$ are samples from the p_θ probability distribution. This is

$$\mathbf{R}_1, \ldots, \mathbf{R}_M \sim p_\theta(\mathbf{R})$$

We obtain E_L using:

$$E_L(\mathbf{R}_k) = \frac{\hat{H}\psi(\mathbf{R}_k)}{\psi(\mathbf{R}_k)} = -\frac{1}{2}\frac{\nabla^2\psi(\mathbf{R_k})}{\psi(\mathbf{R}_k)} + V(\mathbf{R}_k) \quad (28)$$

Calculus tell us that for a any derivable function f.

$$\frac{\nabla^2 f}{f} = [\nabla^2 \log f + (\nabla f)^2] \quad (29)$$

In practice is more numerically stable work using that form, thus:

$$E_L(\mathbf{R}_k) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{3}\left[\frac{\partial^2 \log|\Psi(x)|}{\partial r_{ij}^2} + \left(\frac{\partial \log|\Psi(x)|}{\partial r_{ij}}\right)^2\right] + V(\mathbf{R_k}) \quad (30)$$

The gradient of the energy respect to the parameters by a parameterized wave functions is:

$$\nabla_\theta \mathcal{L} = 2\mathbb{E}_{x\sim\Psi^2}[(E_L(x) - \mathbb{E}_{x'\sim\Psi^2}[E_L(x')])\nabla \log|\Psi(x)|] \quad (31)$$

4.1.8 Metropolis Hastings Algorithm

To obtain samples from the probability distribution p_θ we are going to use the Metropolis Hasting algorithm (MH) [33] which is a Markov Chain Monte Carlo (MCMC) method used to obtain a sequence of random samples from a probability distribution. The reason to use this method over another well knows methods (e.g. example) is that MH don't suffer of the *Curse of Dimensionality* this is, it remains strong while increase the dimension of the problem, and since we are going to be working on dimension around ten is efficient use this method. The algorithm works like follow:

1. Take a initial configuration $\mathbf{X}_0 \in E$ arbitrary:
2. Propose $\mathbf{X}' = \mathbf{X}_0 + \eta$,where $\eta \sim q(\eta)$, q is a probability density on E called **proposal kernel**. In our case we are going to a symmetric Gaussian.
3. Compute the quantity:

   $$A(\mathbf{X_0}, \mathbf{X}') = \min\left(1, \frac{\rho(\mathbf{X}')}{\rho(\mathbf{X}_0)}\frac{q(\mathbf{X}' - \mathbf{X}_0)}{q(\mathbf{X}_0 - \mathbf{X}')}\right)$$

   Where ρ is the target distribution where we want sample, In the case where q is symmetric, this simplifies to:

   $$A(\mathbf{X}_0, \mathbf{X}') = \min\left(1, \frac{\rho(\mathbf{X})}{\rho(\mathbf{X}_0)}\right)$$

   Note that in our case ρ is equal to p_θ, we said that compute the integral factor remains a challenge, but in this case we have it.

   $$\frac{p_\theta(\mathbf{X}')}{p_\theta(\mathbf{X}_0)} = \frac{|\psi_\theta(\mathbf{X}')|^2/\int|\psi_\theta|^2 dx}{|\psi_\theta(\mathbf{X}_0)|^2/\int|\psi_\theta|^2 dx} = \frac{|\psi_\theta(\mathbf{X}')|^2}{|\psi_\theta(\mathbf{X}_0)|^2}$$

4. Generate a uniform number $U \in [0, 1]$
5. If: $U < A(\mathbf{X}_0 \to \mathbf{X'}_l)$ then $\mathbf{X_1} = \mathbf{X}'$, otherwise try another $\mathbf{X}'$. Accept or decline.
6. Repeat until obtain N_{eq} accepted sample, the changes stabilizes (we reach a stationary distribution) this phase is called **burn in**.
7. From $\mathbf{X}_{N_{\text{eq}}}$ generate M samples until reach the sample $\mathbf{X}_{N_{\text{eq}}+M+1}$. In each sample generates $E_L(\mathbf{R}_k)$ then average to obtain $\mathbb{E}(E_L)$ and with it and the derivative from equation we are able to begin the back propagation step.

4.2 Deep Learning Fundamentals

This subsection introduces the core concepts of Deep Learning that are going to be applied in this work.

4.2.1 Multi Layer Perceptron

A multi layer perceptron (MLP) is a nonlinear function $\mathcal{F} : \mathbb{R}^{\text{in}} \to \mathbb{R}^{\text{out}}$ [34] , it actually is the composition of L layers, the first layer is called the input layer, the last output layer and the intermediates hidden layers. In each layer we find a arbitrary number of neurons although is a good practice always choose number which are powers of two and an affine map $\mathbf{z}^{(l)}, l \in \{L, L-1, \ldots, 2\}$ ($l = 1$ is the input layer) of the follow form.

$$\mathbf{z}^{(l)} = \mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

Where $\mathbf{W}^{(l)}$ called weight matrix and $\mathbf{b}^{(l)}$ the bias vector of the l layer. We use a non-linear function $\sigma^{(l)}$ in the l layer (typically Softmax, ReLu, Tanh), thus the output of each layer is:

$$f^{(l)} = \sigma^{(l)} \circ \mathbf{z}^{(l)}$$

Where $\circ$ means composition. A MLP is the composition of all the layers.

$$\mathcal{F} = f^{(L)} \circ f^{(L-1)} \circ \cdots \circ f^{(1)}$$

We call parameters to the set of all the weights and bias of each layer. And represented it with the symbol θ.

$$\{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=2}^{L} = \theta$$

You typically train a MLP, using a training data set, a loss function (e.g Mean Square Error, Mean Absolute Error, Cross entropy) and an optimizer (e.g GD, SGD, ADAM). Additionally you can use regularization techniques such as dropout to improve the generalization of the Net.

4.2.2 Natural gradient Descent

As we mentioned, there are many ways to update the parameters of a neural network: Gradient Descent, Stochastic Gradient Descent, Adaptive Moment Estimation (ADAM), etc. All of them implicitly assume that the parameter space $\Theta \subset \mathbb{R}^d$ is equipped with

the standard Euclidean metric, so that "length" and "steepest descent" are measured with respect to $\|\Delta\theta\|_2$.

In our case the loss $\mathcal{L}(\theta)$ depends on a probability distribution p_θ, not just on θ directly. For example, in variational Monte Carlo we take

$$p_\theta(x) = \frac{|\psi_\theta(x)|^2}{\displaystyle\int |\psi_\theta(x')|^2 \, dx'},$$

so θ parametrizes an entire family of probability densities over configurations x. It is therefore more natural to measure distances between *distributions* p_θ and $p_{\theta+\Delta\theta}$, rather than between the parameter vectors themselves.

A canonical way to measure the distance between nearby probability distributions is the Kullback–Leibler (KL) divergence

$$\mathrm{KL}\big(p_\theta \,\|\, p_{\theta+\Delta\theta}\big) = \mathbb{E}_{x\sim p_\theta}\left[\log \frac{p_\theta(x)}{p_{\theta+\Delta\theta}(x)}\right].$$

For small steps $\Delta\theta$ one can show that a second–order Taylor expansion of the KL gives

$$\mathrm{KL}\big(p_\theta \,\|\, p_{\theta+\Delta\theta}\big) = \tfrac{1}{2}\,\Delta\theta^\top \mathcal{F}(\theta)\,\Delta\theta + \mathcal{O}(\|\Delta\theta\|^3),$$

where $\mathcal{F}(\theta)$ is the Fisher Information Matrix (FIM) [35]. To define it, introduce the **score function**

$$s_\theta(x) = \nabla_\theta \log p(x \mid \theta) \in \mathbb{R}^d$$

then the FIM is:

$$\mathcal{F}(\theta) = \mathbb{E}_{x\sim p(\cdot|\theta)}\big[\, s_\theta(x)\, s_\theta(x)^\mathsf{T}\big].$$

The set of distributions

$$\mathcal{M} = \{\, p_\theta(z) \mid \theta \in \Theta \subset \mathbb{R}^d \,\}$$

can be viewed as a differentiable manifold, and $\mathcal{F}(\theta)$ defines a Riemannian metric on its tangent space. Concretely, for tangent vectors $u, v \in \mathbb{R}^d$ at θ we define the inner product

$$\langle u, v\rangle_\theta = u^\mathsf{T}\,\mathcal{F}(\theta)\,v.$$

This metric says: two parameter directions are "close" if they induce similar infinitesimal changes in the *distribution* p_θ.

Now ask the usual steepest–descent question, but with this non-Euclidean metric:

Find the direction $\Delta\theta$ that decreases $\mathcal{L}(\theta)$ the fastest, among all directions with fixed "length" $\|\Delta\theta\|_\theta^2 = \Delta\theta^\top\mathcal{F}(\theta)\,\Delta\theta$.

Solving this constrained optimization problem (e.g. with Lagrange multipliers) yields the **natural gradient** direction [36]

$$\Delta\theta_{\mathrm{nat}} \;\propto\; -\,\mathcal{F}(\theta)^{-1}\,\nabla_\theta\mathcal{L}(\theta).$$

Thus the natural gradient descent update is

$$\Delta\theta_{\mathrm{nat}} = -\,\eta\,\mathcal{F}(\theta)^{-1}\,\nabla_\theta\mathcal{L}(\theta),$$

where $\eta > 0$ is a step size. Compared with the usual gradient $\nabla_\theta\mathcal{L}$, the factor $\mathcal{F}^{-1}$ "preconditions" the gradient by the local geometry of the model's probability distribution: directions that barely change p_θ are amplified, directions that change it a lot are damped.

Natural gradient descent is therefore meaningful exactly in the situation we care about: when the loss depends on the parameters *through* a probability model p_θ (e.g. likelihood, cross-entropy, KL, variational objectives, variational Monte Carlo energy, etc.).

4.2.3 Kronecker Factored Approximate Curvature

Directly computing and inverting the full Fisher matrix $\mathcal{F}(\theta)$ is infeasible for modern neural networks, since θ can have millions of components. Kronecker Factored Approximate Curvature (KFAC) [37] is an efficient approximation that makes natural gradient updates practical for layered networks.

We sketch the construction for a fully connected layer ℓ with weight matrix W_ℓ and (for simplicity) no bias. Bias terms can be included by augmenting the activations with a constant 1; we comment on this below.

Forward definition of $\mathbf{a}_\ell$

Consider a standard MLP. For a single input sample x, the forward pass at layer ℓ is

- **Input (activation) to layer ℓ**:

  $$\mathbf{a}_\ell \in \mathbb{R}^{n_\ell}$$

  This is the column vector of activations coming into layer ℓ. For the first hidden layer, $\mathbf{a}_1$ is just the (possibly preprocessed) input. For deeper layers it is the nonlinearity output from the previous layer.

- **Pre-activation at layer ℓ**:

  $$\mathbf{h}_\ell = W_\ell\,\mathbf{a}_\ell,$$

  where $W_\ell \in \mathbb{R}^{m_\ell\times n_\ell}$.

- **Output activation of layer ℓ**:

  $$\tilde{\mathbf{a}}_\ell = \phi(\mathbf{h}_\ell),$$

  where ϕ is applied element-wise. In many notations $\tilde{\mathbf{a}}_\ell$ would become the input to the next layer, but to keep notation consistent with the Fisher block for W_ℓ, we explicitly distinguish:

  - $\mathbf{a}_\ell$: input to W_ℓ,
  - $\mathbf{h}_\ell$: pre-activation,
  - $\tilde{\mathbf{a}}_\ell$: output activation of layer ℓ.

In KFAC, when we talk about $\mathbf{a}_\ell$ for the Fisher block of W_ℓ, we always mean "the vector that W_ℓ multiplies on the right".

Backward definition of $\mathbf{e}_\ell$

Let the loss for a single sample be $\mathcal{L}(\theta)$ (for example, negative log-likelihood or negative log of the wavefunction probability). Define the **backward sensitivity** (or error signal) at layer ℓ as

$$\mathbf{e}_\ell = \frac{\partial\mathcal{L}}{\partial\mathbf{h}_\ell} \in \mathbb{R}^{m_\ell}.$$

This is computed via backpropagation: - At the output layer L:

$$\mathbf{e}_L = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_L} = \left(\frac{\partial \mathcal{L}}{\partial \tilde{\mathbf{a}}_L}\right) \odot \phi'(\mathbf{h}_L),$$

where $\odot$ is the element-wise product or also Hadamard product. - For hidden layers $\ell < L$:

$$\mathbf{e}_\ell = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_\ell} = \left(W_{\ell+1}^\mathsf{T} \mathbf{e}_{\ell+1}\right) \odot \phi'(\mathbf{h}_\ell).$$

In the context of natural gradient for probabilistic models, $\mathcal{L}$ is often chosen as $-\log p(X \mid \theta)$, so up to a sign we can also think of $\mathbf{e}_\ell$ as

$$\mathbf{e}_\ell = \frac{\partial \log p(X \mid \theta)}{\partial \mathbf{h}_\ell}.$$

Gradient w.r.t. W_ℓ and the form $\mathbf{a}_\ell \otimes \mathbf{e}_\ell$
For a single sample, using the chain rule,

$$\frac{\partial \mathcal{L}}{\partial W_\ell} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_\ell} \frac{\partial \mathbf{h}_\ell}{\partial W_\ell} = \mathbf{e}_\ell \, \mathbf{a}_\ell^\mathsf{T}.$$

If instead of $\mathcal{L}$ we use $\log p(X \mid \theta)$ (as in the Fisher definition), we get

$$\frac{\partial \log p(X \mid \theta)}{\partial W_\ell} = \mathbf{e}_\ell \, \mathbf{a}_\ell^\mathsf{T},$$

with $\mathbf{e}_\ell = \partial \log p / \partial \mathbf{h}_\ell$.

Now vectorize the gradient. Using the standard identity

$$\mathrm{vec}(uv^\mathsf{T}) = v \otimes u,$$

with $u = \mathbf{e}_\ell$ and $v = \mathbf{a}_\ell$, we obtain

$$\frac{\partial \log p(X \mid \theta)}{\partial \mathrm{vec}(W_\ell)} = \mathrm{vec}\left(\frac{\partial \log p}{\partial W_\ell}\right) = \mathrm{vec}(\mathbf{e}_\ell \, \mathbf{a}_\ell^\mathsf{T}) = \mathbf{a}_\ell \otimes \mathbf{e}_\ell.$$

This gives the key structural form used by KFAC.

Fisher block for a single layer

The Fisher block associated with the parameters W_ℓ is

$$\mathcal{F}_\ell = \mathbb{E}_{p(\mathbf{X})}\left[\frac{\partial \log p(X \mid \theta)}{\partial \mathrm{vec}(W_\ell)} \frac{\partial \log p(X \mid \theta)}{\partial \mathrm{vec}(W_\ell)}^\mathsf{T}\right].$$

Plugging in the expression above,

$$\mathcal{F}_\ell = \mathbb{E}_{p(\mathbf{X})}\big[(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)^\mathsf{T}\big].$$

Here $p(\mathbf{X})$ denotes the distribution over inputs and labels (or configurations, in the VMC case). In practice this expectation is approximated by averaging over a mini-batch of samples X and the corresponding forward/backward passes that produce $\mathbf{a}_\ell$ and $\mathbf{e}_\ell$.

Computing and inverting $\mathcal{F}_\ell$ directly is still expensive, because its dimension is

$$(\dim(\mathbf{a}_\ell)\dim(\mathbf{e}_\ell)) \times (\dim(\mathbf{a}_\ell)\dim(\mathbf{e}_\ell)).$$

KFAC makes two key approximations to make this tractable.

1. **Block–diagonal across layers.**
   Off–diagonal blocks $\mathcal{F}_{ij}$ are assumed negligible when θ_i and θ_j belong to different layers. This makes the Fisher approximately block–diagonal, with one block per layer.

2. **Kronecker factorization within each layer.**
   Inside a layer, KFAC assumes that the correlation between activations and errors factorizes:

$$\begin{aligned}
\mathcal{F}_\ell &= \mathbb{E}_{p(\mathbf{X})}\big[(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)^\mathsf{T}\big] && (32)\\
&= \mathbb{E}_{p(\mathbf{X})}\big[(\mathbf{a}_\ell \mathbf{a}_\ell^\mathsf{T}) \otimes (\mathbf{e}_\ell \mathbf{e}_\ell^\mathsf{T})\big] && (33)\\
&\approx \mathbb{E}_{p(\mathbf{X})}[\mathbf{a}_\ell \mathbf{a}_\ell^\mathsf{T}] \otimes \mathbb{E}_{p(\mathbf{X})}[\mathbf{e}_\ell \mathbf{e}_\ell^\mathsf{T}] && (34)
\end{aligned}$$

Define the *activation covariance* and *error covariance*:

$$A_\ell = \mathbb{E}_{p(\mathbf{X})}[\mathbf{a}_\ell \mathbf{a}_\ell^\mathsf{T}], \qquad S_\ell = \mathbb{E}_{p(\mathbf{X})}[\mathbf{e}_\ell \mathbf{e}_\ell^\mathsf{T}].$$

In practice these expectations are updated as running averages over mini-batches:

$$A_\ell \approx \frac{1}{B}\sum_{b=1}^{B} \mathbf{a}_\ell^{(b)} \mathbf{a}_\ell^{(b)\mathsf{T}}, \qquad S_\ell \approx \frac{1}{B}\sum_{b=1}^{B} \mathbf{e}_\ell^{(b)} \mathbf{e}_\ell^{(b)\mathsf{T}},$$

where b indexes samples in the batch and $\mathbf{a}_\ell^{(b)}, \mathbf{e}_\ell^{(b)}$ are obtained by a standard forward and backward pass for that sample. With this approximation we have

$$\mathcal{F}_\ell \approx A_\ell \otimes S_\ell.$$

The crucial property of the Kronecker product is that

$$(A_\ell \otimes S_\ell)^{-1} = A_\ell^{-1} \otimes S_\ell^{-1},$$

so the inverse of the (huge) layer–Fisher block can be obtained by inverting the much smaller matrices A_ℓ and S_ℓ. Thus the natural gradient update for the weights of layer ℓ becomes

$$\Delta\theta_{\mathrm{nat},\ell} \approx -\eta \left(A_\ell^{-1} \otimes S_\ell^{-1}\right) \nabla_{\mathrm{vec}(W_\ell)} \mathcal{L}.$$

In summary, KFAC replaces the intractable inverse

$$\mathbb{E}_{p(\mathbf{X})}\big[(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)^\mathsf{T}\big]^{-1}$$

by the efficiently computable:

$$\begin{aligned}
\mathcal{F}_\ell^{-1} &= \mathbb{E}_{p(\mathbf{X})}\big[(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)(\mathbf{a}_\ell \otimes \mathbf{e}_\ell)^\mathsf{T}\big]^{-1} && (35)\\
&\approx \mathbb{E}_{p(\mathbf{X})}[\mathbf{a}_\ell \mathbf{a}_\ell^\mathsf{T}]^{-1} \otimes \mathbb{E}_{p(\mathbf{X})}[\mathbf{e}_\ell \mathbf{e}_\ell^\mathsf{T}]^{-1} && (36)
\end{aligned}$$

which captures the dominant curvature structure while keeping the cost of natural gradient descent comparable to standard first–order methods. We have ignored biases above for clarity. In practice one can either (i) augment $\mathbf{a}_\ell$ with a constant 1 to absorb biases into W_ℓ, or (ii) maintain separate smaller KFAC factors for biases; both approaches preserve the same Kronecker structure.

4.2.4 Self-Attention and Multi-Head Self-Attention

Attention Mechanisms The idea of an *attention mechanism* was introduced in neural machine translation by Bahdanau et al. @bahdanau2014neural Instead of compressing an entire input sequence into a single fixed-size vector, the model learns to **focus** on different parts of the input when generating each output token.

Given a query vector $\mathbf{q} \in \mathbb{R}^{d_h}$ and a set of key–value pairs $\{(\mathbf{k}_j, \mathbf{v}_j)\}_{j=1}^{T}$ with $\mathbf{k}_j, \mathbf{v}_j \in \mathbb{R}^{d_h}$, the (scaled dot–product) attention mechanism computes:

1. **Compatibility scores**

$$e_j \;=\; \frac{\mathbf{q}^\top \mathbf{k}_j}{\sqrt{d_h}}, \qquad j = 1, \dots, T,$$

2. **Normalized attention weights**

$$\alpha_j \;=\; \frac{\exp(e_j)}{\sum_{m=1}^{T} \exp(e_m)} \;=\; \mathrm{Softmax}_j\!\left(\frac{\mathbf{q}^\top \mathbf{k}_j}{\sqrt{d_h}}\right),$$

3. **Weighted sum of values**

$$\mathbf{o} \;=\; \sum_{j=1}^{T} \alpha_j \mathbf{v}_j.$$

Intuitively, the query $\mathbf{q}$ asks: *"which elements of the set are relevant to me now?"* The keys $\mathbf{k}_j$ encode *what each element offers*, and the values $\mathbf{v}_j$ encode *what we take from each element once we decide to pay attention to it.*

In **self-attention**, the queries, keys, and values are all obtained from the **same** set of input vectors. Consider a sequence of input embeddings

$$\mathbf{x}_1, \dots, \mathbf{x}_T \in \mathbb{R}^d,$$

and stack them into a matrix

$$\mathbf{X} \in \mathbb{R}^{T \times d}, \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_T^\top \end{bmatrix}.$$

To build one attention **head** of dimension d_h, we introduce three learnable matrices:

$$\mathbf{W}^Q \in \mathbb{R}^{d \times d_h}, \quad \mathbf{W}^K \in \mathbb{R}^{d \times d_h}, \quad \mathbf{W}^V \in \mathbb{R}^{d \times d_h}.$$

We then compute queries, keys, and values:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^Q \in \mathbb{R}^{T \times d_h} \tag{37}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}^K \in \mathbb{R}^{T \times d_h} \tag{38}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}^V \in \mathbb{R}^{T \times d_h} \tag{39}$$

The **scaled dot-product self-attention** for this head is:

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{Softmax}\!\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_h}}\right)\mathbf{V},$$

where the softmax is applied row-wise. Element-wise, the output at position t is

$$\mathbf{o}_t = \sum_{j=1}^{T} \alpha_{tj} \mathbf{v}_j \text{ with } \alpha_{tj} = \frac{\exp\!\big(\mathbf{q}_t^\top \mathbf{k}_j / \sqrt{d_h}\big)}{\sum_{m=1}^{T} \exp\!\big(\mathbf{q}_t^\top \mathbf{k}_m / \sqrt{d_h}\big)}.$$

You can think of this as: *each position t in the sequence "looks at" every other position j and decides how much to care about it.*

Multi-Head Self-Attention

A single head can only look at interactions in one "representation subspace" of dimension d_h.

Multi-head attention uses several heads in parallel, each with its own projection matrices, so that different types of relationships can be captured simultaneously. Let n_h be the number of heads. For head $i = 1, \dots, n_h$ we have

$$\mathbf{W}_i^Q,\, \mathbf{W}_i^K,\, \mathbf{W}_i^V \in \mathbb{R}^{d \times d_h}.$$

Head i computes:

$$\mathrm{head}_i(\mathbf{X}) = \mathrm{Attention}(\mathbf{X}\mathbf{W}_i^Q,\, \mathbf{X}\mathbf{W}_i^K,\, \mathbf{X}\mathbf{W}_i^V) \in \mathbb{R}^{T \times d_h}.$$

The outputs of all heads are concatenated along the feature dimension and then linearly mixed:

$$\mathbf{U} = [\mathrm{head}_1(\mathbf{X})\,;\,\dots\,;\,\mathrm{head}_{n_h}(\mathbf{X})] \in \mathbb{R}^{T \times (n_h d_h)},$$

$$\mathbf{O} = \mathbf{U}\mathbf{W}^O, \qquad \mathbf{W}^O \in \mathbb{R}^{(n_h d_h) \times d}.$$

If we focus on one time step t and head i, we can write the per-head output as

$$\mathbf{o}_{t,i} = \sum_{j=1}^{T} \mathrm{Softmax}_j\!\left(\frac{\mathbf{q}_{t,i}^\top \mathbf{k}_{j,i}}{\sqrt{d_h}}\right)\mathbf{v}_{j,i},$$

and the final vector at time t after concatenation and output projection as

$$\mathbf{u}_t = \mathbf{W}^O \begin{bmatrix} \mathbf{o}_{t,1} \\ \vdots \\ \mathbf{o}_{t,n_h} \end{bmatrix}.$$

From a physics point of view, you can read multi-head attention as **several different "channels" of interaction**: one head might focus on short-range relations, another on long-range ones, another on some specific pattern (e.g. symmetry, local structure), and so on.

4.2.5 Transformer Architecture

The **Transformer** was introduced with the slogan *"Attention Is All You Need."* [12]. Its core building block is a **layer** that combines: 1. **Multi-head self-attention** 2. **Position-wise feed-forward network (FFN)** Both sublayers use **residual connections** and **layer normalization**. For an input sequence $\mathbf{X} \in \mathbb{R}^{T \times d}$ (already including positional information), one Transformer layer performs: 3. **Multi-head self-attention sublayer**

$$\mathbf{H} = \mathrm{MHA}(\mathbf{X}), \qquad \mathbf{X}^{(1)} = \mathrm{LayerNorm}(\mathbf{X} + \mathbf{H})\,.$$

4. **Feed-forward sublayer** (applied independently at each position)

$$\mathrm{FFN}(\mathbf{x}) = \sigma(\mathbf{x}\mathbf{W}_1 + \mathbf{b}_1)\,\mathbf{W}_2 + \mathbf{b}_2,$$

typically with σ a nonlinearity such as ReLU or GELU and an intermediate width $d_{\text{ff}} > d$. At the sequence level:

$$\mathbf{Z} = \mathrm{FFN}(\mathbf{X}^{(1)}), \qquad \mathbf{X}^{\text{out}} = \mathrm{LayerNorm}\Big(\mathbf{X}^{(1)} + \mathbf{Z}\Big).$$

Stacking several such layers yields a deep architecture where, at each layer, every position can interact with every other position through self-attention.

In the original formulation [12], **positional encodings** (sinusoidal or learned) are added to the embeddings so that the model can distinguish different positions in the sequence:

$$\mathbf{X}_0 = \mathbf{E} + \mathbf{P},$$

where $\mathbf{E}$ are token embeddings and $\mathbf{P}$ are positional encodings.

4.2.6 Why Transformers Instead of RNNs or LSTMs?

Recurrent Neural Networks (RNNs) [cite] and Long Short-Term Memory (LSTM) [cite] networks process the sequence **sequentially** this is each new state depends on the previous one. This has two important consequences: 1. Information must flow through many time steps, which can lead to vanishing or exploding gradients and makes it hard to model very long-range interactions. 2. Poor parallelization because each step depends on the previous one, you cannot compute all time steps in parallel. Training and inference are inherently sequential. Transformers address both issues: - Global interactions in one step, self-attention allows every position to directly interact with every other position in a *single* layer, which is ideal when we care about *all-to-all* correlations (as in many-electron systems, where each electron "feels" all the others). - Full parallelism over sequence length. Given $\mathbf{X}$, the matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ and the attention outputs for all time steps are computed via matrix multiplications. This is extremely efficient on modern accelerators (GPUs/TPUs). For a many-electron Schrödinger equation, the wave function depends on the joint configuration of all particles. A Transformer-based ansatz naturally provides a way for each electron's representation to **look at all other electrons** and the nuclei, capturing complex correlation patterns through attention, while remaining highly parallelizable.

A very important work for us is FermiNet (Pfau et al. 2020). It uses deep neural networks to represent **orbitals** and then combines them into a sum of Slater determinants. At the top level, the ansatz is a linear combination of K determinant products

$$\psi(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{k=1}^{K} \omega_k \, \det[\Phi^k],$$

where ω_k are learnable coefficients and Φ^k is a matrix of single-particle orbitals. For a system without explicit spin separation one can write

$$\det[\Phi^k] = \begin{vmatrix} \phi_1^k(\mathbf{x}_1) & \ldots & \phi_1^k(\mathbf{x}_n) \\ \vdots & & \vdots \\ \phi_n^k(\mathbf{x}_1) & \ldots & \phi_n^k(\mathbf{x}_n) \end{vmatrix} = \det[\phi_i^k(\mathbf{x}_j)].$$

Here ϕ_i^k is the i-th orbital in determinant k, and we evaluate it on the coordinates of electron j.

However, in FermiNet we are dealing with electrons with spin, so things are slightly more structured, and the orbitals depend on **all** electron coordinates, not only on the one being "plugged in". That is why we write the orbitals as

$$\phi_i^{k\alpha}\big(\mathbf{r}_j^\alpha; \{\mathbf{r}_{/j}^\alpha\}; \{\mathbf{r}^{\bar{\alpha}}\}\big),$$

where: - $\alpha \in \{\uparrow, \downarrow\}$ is the spin sector, - $\mathbf{r}_j^\alpha$ is the position of electron j with spin α, - $\{\mathbf{r}_{/j}^\alpha\}$ denotes the positions of all **other** electrons with spin α, - $\{\mathbf{r}^{\bar{\alpha}}\}$ denotes the positions of electrons with the opposite spin.

So the orbital evaluated on electron j "knows" about all other electrons. The indices: - i = orbital index (row of the determinant), - j = electron index (column of the determinant), - α, β = spin labels ($\uparrow$ or $\downarrow$), - k = determinant index in the sum.

5 Psi Former Model

5.1 Fermi Net

A very important work for us is FermiNet (Pfau et al. 2020). It uses deep neural networks to represent **orbitals** and then combines them into a sum of Slater determinants. At the top level, the ansatz is a linear combination of K determinant products

$$\psi(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{k=1}^{K} \omega_k \, \det[\Phi^k],$$

where ω_k are learnable coefficients and Φ^k is a matrix of single-particle orbitals. For a system without explicit spin separation one can write

$$\det[\Phi^k] = \begin{vmatrix} \phi_1^k(\mathbf{x}_1) & \ldots & \phi_1^k(\mathbf{x}_n) \\ \vdots & & \vdots \\ \phi_n^k(\mathbf{x}_1) & \ldots & \phi_n^k(\mathbf{x}_n) \end{vmatrix} = \det[\phi_i^k(\mathbf{x}_j)].$$

Here ϕ_i^k is the i-th orbital in determinant k, and we evaluate it on the coordinates of electron j.

However, in FermiNet we are dealing with electrons with spin, so things are slightly more structured, and the orbitals depend on **all** electron coordinates, not only on the one being "plugged in". That is why we write the orbitals as

$$\phi_i^{k\alpha}\big(\mathbf{r}_j^\alpha; \{\mathbf{r}_{/j}^\alpha\}; \{\mathbf{r}^{\bar{\alpha}}\}\big),$$

where: - $\alpha \in \{\uparrow, \downarrow\}$ is the spin sector, - $\mathbf{r}_j^\alpha$ is the position of electron j with spin α, - $\{\mathbf{r}_{/j}^\alpha\}$ denotes the positions

of all **other** electrons with spin α, - $\{\mathbf{r}^{\bar{\alpha}}\}$ denotes the positions of electrons with the opposite spin.

So the orbital evaluated on electron j "knows" about all other electrons. The indices: - i = orbital index (row of the determinant), - j = electron index (column of the determinant), - α, β = spin labels ($\uparrow$ or $\downarrow$), - k = determinant index in the sum.

5.1.1 Input coordinates and features

We denote by - $\mathbf{r}_1^{\uparrow}, \ldots, \mathbf{r}_{n^{\uparrow}}^{\uparrow}$ the coordinates of spin-up electrons, - $\mathbf{r}_1^{\downarrow}, \ldots, \mathbf{r}_{n^{\downarrow}}^{\downarrow}$ the coordinates of spin-down electrons, - $\mathbf{R}_I$ the positions of nuclei, $I = 1, \ldots, N_{\text{nuc}}$.

The network builds two types of features:

1. **Electron–nucleus features** for each electron i with spin α:

$$\mathbf{h}_i^{0,\alpha} = \text{concatenate}\Big(\mathbf{r}_i^{\alpha} - \mathbf{R}_I,\ \big|\mathbf{r}_i^{\alpha} - \mathbf{R}_I\big|\ \forall\, I\Big).$$

   This produces a feature vector that contains, for electron (i, α), all its relative position vectors to each nucleus, plus their distances.

2. **Electron–electron features** for each pair of electrons (i, α) and (j, β):

$$\mathbf{h}_{ij}^{0,\alpha\beta} = \text{concatenate}\Big(\mathbf{r}_i^{\alpha} - \mathbf{r}_j^{\beta},\ \big|\mathbf{r}_i^{\alpha} - \mathbf{r}_j^{\beta}\big|\ \forall\, j, \beta\Big).$$

   For fixed (i, α), we build such features for all other electrons (j, β), capturing their relative positions and distances.

The superscript 0 indicates that these are the features at layer $\ell = 0$ (input to the deep network). At deeper layers we will keep updating - $\mathbf{h}_i^{\ell\alpha}$ (single-electron features), - $\mathbf{h}_{ij}^{\ell\alpha\beta}$ (pairwise features), for $\ell = 0, 1, \ldots, L-1$.

5.1.2 Mixing and updating features across layers

At each hidden layer ℓ, we want each electron's features to depend on *all* other electrons, in a permutation-symmetric way. To do this, we form **averages** over electrons of the same or opposite spin.

First, define global spin-averaged single-electron features

$$\mathbf{g}^{\ell\uparrow} = \frac{1}{n^{\uparrow}} \sum_{j=1}^{n^{\uparrow}} \mathbf{h}_j^{\ell\uparrow}, \qquad \mathbf{g}^{\ell\downarrow} = \frac{1}{n^{\downarrow}} \sum_{j=1}^{n^{\downarrow}} \mathbf{h}_j^{\ell\downarrow}.$$

Next, for each electron (i, α), define averaged pairwise features:

$$\mathbf{g}_i^{\ell\alpha\uparrow} = \frac{1}{n^{\uparrow}} \sum_{j=1}^{n^{\uparrow}} \mathbf{h}_{ij}^{\ell\alpha\uparrow}, \qquad \mathbf{g}_i^{\ell\alpha\downarrow} = \frac{1}{n^{\downarrow}} \sum_{j=1}^{n^{\downarrow}} \mathbf{h}_{ij}^{\ell\alpha\downarrow}.$$

Now we *concatenate* all this information into a single feature vector for electron (i, α):

$$\Big(\mathbf{h}_i^{\ell\alpha}, \frac{1}{n^{\uparrow}} \sum_{j=1}^{n^{\uparrow}} \mathbf{h}_j^{\ell\uparrow}, \frac{1}{n^{\downarrow}} \sum_{j=1}^{n^{\downarrow}} \mathbf{h}_j^{\ell\downarrow}, \frac{1}{n^{\uparrow}} \sum_{j=1}^{n^{\uparrow}} \mathbf{h}_{ij}^{\ell\alpha\uparrow}, \frac{1}{n^{\downarrow}} \sum_{j=1}^{n^{\downarrow}} \mathbf{h}_{ij}^{\ell\alpha\downarrow}\Big) \tag{40}$$

$$= \Big(\mathbf{h}_i^{\ell\alpha}, \mathbf{g}^{\ell\uparrow}, \mathbf{g}^{\ell\downarrow}, \mathbf{g}_i^{\ell\alpha\uparrow}, \mathbf{g}_i^{\ell\alpha\downarrow}\Big) = \mathbf{f}_i^{\ell\alpha}. \tag{41}$$

This $\mathbf{f}_i^{\ell\alpha}$ is what enters the **single-electron MLP** at layer ℓ. The update is

$$\mathbf{h}_i^{\ell+1,\alpha} = \tanh\big(\mathbf{V}^{\ell}\mathbf{f}_i^{\ell\alpha} + \mathbf{b}^{\ell}\big) + \mathbf{h}_i^{\ell\alpha},$$

where $\mathbf{V}^{\ell}$ and $\mathbf{b}^{\ell}$ are learnable weights and biases, shared between electrons (for the given spin sector). The residual connection $+\mathbf{h}_i^{\ell\alpha}$ stabilizes training.

In parallel, the pairwise features are updated with a **pairwise MLP**:

$$\mathbf{h}_{ij}^{\ell+1,\alpha\beta} = \tanh\big(\mathbf{W}^{\ell}\mathbf{h}_{ij}^{\ell\alpha\beta} + \mathbf{c}^{\ell}\big) + \mathbf{h}_{ij}^{\ell\alpha\beta},$$

with weights $\mathbf{W}^{\ell}$ and biases $\mathbf{c}^{\ell}$, again shared over all pairs (i, j, α, β).

By repeating these updates for $\ell = 0, \ldots, L-1$, we eventually obtain **final single-electron features**

$$\mathbf{h}_j^{L\alpha} \quad \text{for each electron } j \text{ of spin } \alpha.$$

Notice how the indices work: - ℓ runs over layers and disappears at the end, - i or j always refer to a specific electron within a spin sector, - α, β tell you which spin sector that electron belongs to.

5.1.3 From final features to orbitals

The final orbitals are built as a function of the last-layer features $\mathbf{h}_j^{L\alpha}$, plus some additional "envelope" factors that handle the long-range decay and cusp conditions. For each determinant index k, spin α, orbital index i, and electron j we define

$$\phi_i^{k\alpha}\big(\mathbf{r}_j^{\alpha}; \{\mathbf{r}_{/j}^{\alpha}\}; \{\mathbf{r}^{\bar{\alpha}}\}\big) = \big(\mathbf{w}_i^{k\alpha} \cdot \mathbf{h}_j^{L\alpha} + g_i^{k\alpha}\big) \times \sum_m \pi_{im}^{k\alpha} \exp\Big(-\big|\mathbf{\Sigma}_{im}^{k\alpha}\big(\mathbf{r}_j^{\alpha} - \mathbf{R}_m\big)\big|\Big)$$

Here: - $\mathbf{w}_i^{k\alpha}$ and $g_i^{k\alpha}$ are learnable linear parameters for the "MLP part" of the orbital, - the sum over m is an "envelope" over nuclei (or centers), - $\pi_{im}^{k\alpha}$ and $\mathbf{\Sigma}_{im}^{k\alpha}$ are learnable coefficients and matrices controlling the exponential decay around nucleus m.

All these parameters depend on the indices: - k selects which determinant in the sum, - i selects which orbital (row in the determinant), - α selects the spin sector, - m selects which nuclear center in the envelope.

The dependence on all other electrons is hidden inside $\mathbf{h}_j^{L\alpha}$, which was built from the full set of positions $\{\mathbf{r}^{\uparrow}\}, \{\mathbf{r}^{\downarrow}\}$ through the deep network.

5.1.4 Assembling the spin-separated determinants

For each determinant index k and spin sector $\alpha \in \{\uparrow, \downarrow\}$, we build a matrix

$$D_{ij}^{k\alpha} = \phi_i^{k\alpha}\big(\mathbf{r}_j^{\alpha}; \{\mathbf{r}_{/j}^{\alpha}\}; \{\mathbf{r}^{\bar{\alpha}}\}\big),$$

with: - rows indexed by the orbital label $i = 1, \ldots, n^{\alpha}$, - columns indexed by the electron label $j = 1, \ldots, n^{\alpha}$ (with that spin).

Taking the determinant gives a properly antisymmetric function of the positions of electrons **with that spin**:

$$\det\left[D^{k\alpha}\right] = \det\left[\phi_i^{k\alpha}(\mathbf{r}_j^{\alpha}; \{\mathbf{r}_{/j}^{\alpha}\}; \{\mathbf{r}^{\bar{\alpha}}\})\right].$$

For the full wavefunction, we combine spin-up and spin-down blocks:

$$\psi(\mathbf{r}_1^{\uparrow}, \ldots, \mathbf{r}_{n^{\uparrow}}^{\uparrow}, \mathbf{r}_1^{\downarrow}, \ldots, \mathbf{r}_{n^{\downarrow}}^{\downarrow}) = \sum_k \omega_k det[D] det[D] \quad (42)$$

We have don't explained why we can write the determinant as that product. In electronic structure, when we separate spin and spatial parts using spin-orbitals, the full Slater determinant over all electrons factorizes into the product of: one determinant involving only spin-up electrons, another determinant involving only spin-down electrons.

Each of these determinants is antisymmetric under exchange of two electrons **with the same spin**. The overall wavefunction constructed as the product of a spin-up determinant and a spin-down determinant is antisymmetric under exchange of any two electrons (when you take into account the spin labels). FermiNet keeps this structure and lets each block be represented by a powerful neural network ansatz for the orbitals. Up to this point the building blocks are just MLP layers (with residual connections and special feature mixing), but the careful indexing - (i, α) for "which electron/spin", - j for summation over electrons, - ℓ for layers, - k for determinant index, is what guarantees that the final object has the correct permutation symmetry and antisymmetry required for a fermionic wavefunction.

5.2 Applying Attention to Fermi Net (Psiformer-style)

5.2.1 Jastrow Factor for Psi Former

[cite self attention]

The Psiformer wavefunction has the usual Slater-Jastrow Ansatz [30]:

$$\Psi_\theta(\mathbf{x}) = \exp\left(\mathcal{J}_\theta(\mathbf{x})\right) \sum_{k=1}^{N_{\text{det}}} \det[\mathbf{\Phi}_\theta^k(\mathbf{x})],$$

where $\mathbf{x} = (x_1, \ldots, x_N)$ is the collection of all N electron states

$$x_i = (\mathbf{r}_i, \sigma_i), \qquad \mathbf{r}_i \in \mathbb{R}^3,\ \sigma_i \in \{\uparrow, \downarrow\}.$$

- $\mathcal{J}_\theta : (\mathbb{R}^3 \times \{\uparrow, \downarrow\})^N \to \mathbb{R}$ is the **Jastrow factor**, encoding (here) only electron–electron cusp information.
- $\mathbf{\Phi}_\theta^k$ is the matrix of (spin-)orbitals for determinant k.

In Psiformer, the Jastrow factor is *very* simple: it has only two learnable parameters, one for parallel-spin pairs and one for antiparallel-spin pairs:

$$\mathcal{J}_\theta(\mathbf{x}) = \sum_{i<j;\, \sigma_i=\sigma_j} -\frac{1}{4}\frac{\alpha_{\text{par}}^2}{\alpha_{\text{par}} + |\mathbf{r}_i - \mathbf{r}_j|} + \quad (43)$$

$$\sum_{i,j;\, \sigma_i\neq\sigma_j} -\frac{1}{2}\frac{\alpha_{\text{anti}}^2}{\alpha_{\text{anti}} + |\mathbf{r}_i - \mathbf{r}_j|}. \quad (44)$$

- α_{par} controls the strength of the Jastrow for **same-spin** electron pairs.
- α_{anti} does the same for **opposite-spin** pairs.

This Jastrow is responsible for enforcing the electron–electron cusp conditions. The neural network itself (the Psiformer) only sees **electron–nucleus** information in its attention stream; all explicit $|\mathbf{r}_i - \mathbf{r}_j|$ dependence lives in $\mathcal{J}_\theta$.

Conceptually, Psiformer is "FermiNet with the two-electron stream replaced by self-attention", we can see it clearly doing.

- FermiNet: separate one-electron and two-electron feature streams, then mix.
- Psiformer: a **single stream** of self-attention layers on electron–nuclear features only; electron–electron features enter only via the Jastrow.

We now explain the indices and equations carefully.
Indices
We will use:

- $i, j = 1, \ldots, N$: electron indices.
- $I = 1, \ldots, N_{\text{nuc}}$: nucleus index.
- $\sigma_i \in \{\uparrow, \downarrow\}$: spin of electron i.
- $\ell = 0, \ldots, L-1$: layer index in the Psiformer.
- $h = 1, \ldots, H$: attention head index.
- $k = 1, \ldots, N_{\text{det}}$: determinant index.
- d: hidden dimension of the per-electron feature vectors. So at each layer ℓ, each electron i carries a feature (hidden state)

$$\mathbf{h}_i^\ell \in \mathbb{R}^d.$$

Input features and initial hidden states
Psiformer only uses **electron–nuclear** features (plus spin) as input to the attention stack. For each electron i: 1. Let $\mathbf{R}_I$ be nuclear positions. 2. Build raw features by concatenating for all I: - some function of $\mathbf{r}_i - \mathbf{R}_I$ (relative position), - $|\mathbf{r}_i - \mathbf{R}_I|$ (distance), - and the spin as a scalar (e.g. $\sigma_i = +1$ for $\uparrow$, -1 for $\downarrow$).

In the paper they rescale the electron–nucleus vectors so that large distances grow only logarithmically, but at the level of notation we can just write

$$\mathbf{f}_i^0 \in \mathbb{R}^{d_{\text{in}}} \quad \text{(electron–nucleus features + spin)}.$$

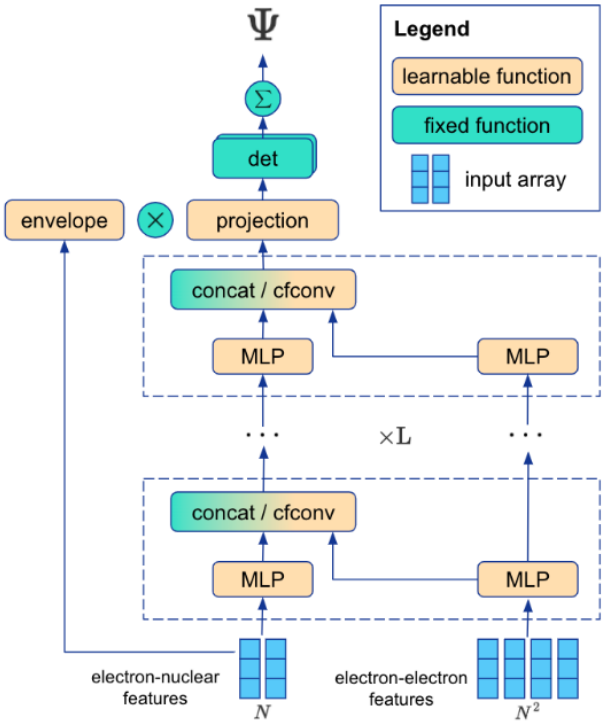

Figure 1: Architecture of FermiNet .Image source Dfau et al

These are then mapped into the model hidden dimension by a linear layer

$$\mathbf{h}_i^0 = \mathbf{W}^0\,\mathbf{f}_i^0,$$

where $\mathbf{W}^0 \in \mathbb{R}^{d\times d_{\text{in}}}$ is learned. So: - index i is "which electron", - superscript 0 means "before any attention layers."

One self-attention layer

At layer ℓ, we have all electron hidden states

$$\mathbf{h}_1^\ell, \ldots, \mathbf{h}_N^\ell.$$

For each **head** h and electron i, we compute:

- Query:
$$\mathbf{q}_i^{\ell h} = \mathbf{W}_q^{\ell h}\mathbf{h}_i^\ell$$
- Key:
$$\mathbf{k}_i^{\ell h} = \mathbf{W}_k^{\ell h}\mathbf{h}_i^\ell$$
- Value:
$$\mathbf{v}_i^{\ell h} = \mathbf{W}_v^{\ell h}\mathbf{h}_i^\ell$$

Here each $\mathbf{W}_q^{\ell h}, \mathbf{W}_k^{\ell h}, \mathbf{W}_v^{\ell h}$ is a learned matrix, shared across all electrons i, but specific to layer ℓ and head h.

Then the **self-attention output for electron i, head h** is

$$\mathbf{A}_i^{\ell h} = \sum_{j=1}^{N} \underbrace{\frac{\exp\left((\mathbf{q}_i^{\ell h})^\intercal \mathbf{k}_j^{\ell h}/\sqrt{d_k}\right)}{\sum_{j'=1}^{N}\exp\left((\mathbf{q}_i^{\ell h})^\intercal \mathbf{k}_{j'}^{\ell h}/\sqrt{d_k}\right)}}_{\text{attention weight from } i \text{ to } j} \mathbf{v}_j^{\ell h}.$$

- j runs over "all other electrons," so electron i "looks" at all others via attention.
- d_k is the key/query dimension (usually $d_k = d/H$ or similar).

This is exactly your

$$A_h^\ell = [\text{SelfAttn}(\mathbf{h}_1^\ell, \ldots, \mathbf{h}_N^\ell; \mathbf{W}_q^{\ell h}, \mathbf{W}_k^{\ell h}, \mathbf{W}_v^{\ell h})],$$

but now written explicitly with indices i and j.

Next, we **concatenate over heads** for each electron:

$$\mathbf{A}_i^\ell = \text{concat}_{h=1}^{H}\left[\mathbf{A}_i^{\ell h}\right] \in \mathbb{R}^{Hd_v},$$

where d_v is the value dimension of each head.

This is your

$$A^\ell = \text{concat}_h[A_h],$$

but again with the electron index i made explicit.

5.2.2 Residual projection and MLP

We then map the concatenated attention output back to the hidden dimension and add a residual connection:

$$\mathbf{f}_i^{\ell+1} = \mathbf{h}_i^\ell + \mathbf{W}_o^\ell\,\mathbf{A}_i^\ell,$$

where $\mathbf{W}_o^\ell$ is a learned matrix.

Then we pass this through a small MLP (just a linear + tanh here), again with a residual:

$$\mathbf{h}_i^{\ell+1} = \mathbf{f}_i^{\ell+1} + \tanh\left(\mathbf{W}^{\ell+1}\mathbf{f}_i^{\ell+1} + \mathbf{b}^{\ell+1}\right).$$

So a full Psiformer layer ℓ is:

1. Self-attention: $\{\mathbf{h}_i^\ell\} \to \{\mathbf{A}_i^\ell\}$.
2. Linear + residual: $\{\mathbf{A}_i^\ell\} \to \{\mathbf{f}_i^{\ell+1}\}$.
3. MLP + residual: $\{\mathbf{f}_i^{\ell+1}\} \to \{\mathbf{h}_i^{\ell+1}\}$.

Repeat this for $\ell = 0, \ldots, L-1$ and you get **final hidden states**

$$\mathbf{h}_j^L \quad \text{for each electron } j.$$

Here: - L = number of layers in the Psiformer. - For each layer, i indexes the electron the output belongs to, j indexes electrons we attend over. - h indexes different heads in multi-head attention.

5.2.3 From hidden states to orbitals and determinants

From the final hidden states $\mathbf{h}_j^L$, we build the spin-orbital matrix for each determinant k.

For each determinant index k and orbital index i, define a **linear "orbital head"**:

$$\tilde{\phi}_i^k(x_j) = \mathbf{w}_i^k \cdot \mathbf{h}_j^L + g_i^k,$$

where $\mathbf{w}_i^k$ and g_i^k are learned. The dependence on spin σ_j and all other electrons is implicit in $\mathbf{h}_j^L$: the self-attention layers have already mixed that information in.

Then we multiply by an **envelope** to enforce the correct asymptotic decay:

$$\Omega_{ij}^k = \sum_m \pi_{im}^k \exp\big(-\big|\mathbf{\Sigma}_{im}^k(\mathbf{r}_j - \mathbf{R}_m)\big|\big),$$

where - m indexes nuclei (or "envelope centers"), - π_{im}^k and $\mathbf{\Sigma}_{im}^k$ are learned parameters.

The final spin-orbital entries are

$$\Phi_{ij}^k = \Omega_{ij}^k \, \tilde{\phi}_i^k(x_j).$$

Collecting these into the matrix

$$\mathbf{\Phi}^k(\mathbf{x}) = \big[\Phi_{ij}^k\big]_{i,j=1}^N,$$

we form the determinant

$$\det[\mathbf{\Phi}^k(\mathbf{x})] = \det\big[\Phi_{ij}^k\big] = \det\big[\phi_i^k(x_j)\big],$$

and finally the full Psiformer wavefunction

$$\Psi_\theta(\mathbf{x}) = \exp(\mathcal{J}_\theta(\mathbf{x})) \sum_{k=1}^{N_{\text{det}}} \det[\mathbf{\Phi}_\theta^k(\mathbf{x})].$$

So the story in terms of indices is:

- i = which **orbital** (row of the determinant).
- j = which **electron** the orbital is evaluated on (column of the determinant).
- k = which **determinant** in the sum.
- ℓ = which **layer** of the attention/MLP stack produced the hidden states.
- h = which **attention head** participated in mixing information across electrons.
- I, m = which **nucleus/center** is used for the envelope.

The self-attention layers are what let $\mathbf{h}_j^L$ depend on all other electrons in a flexible, learned way, while the determinant over i, j and the Jastrow over i, j enforce fermionic antisymmetry and cusp conditions.

6 Methodology

To implement the code, the choose of the library is important. The three options to implement this kind of matter are JAX, Tensor Flow and pytorch, each one with his advantages and disadvantages.

6.1 Environment

For this project we are going to be using Pytorch due his user-friendly and support. Python, and several libraries as transformers from hugging face, a library who implements KFCA and like guide the implement of Fermi Net by google deepmind which is made it on TensorFlow.

Project manager with UV.

6.2 Training

Due the high computational power needed we are going to using GPUS mention by the authors. Several weeks to train molecules

and of course CUDA. Is clear that we are going to use virtual GPUS, for that matter we have two option or well use a GPU via SSH or directly using services like Azure , Colab, or anothers matters. For simplicity we are going to use Colab services or RunPod. The election of the GPU is not trivial. use TPUS are not a bad idea.

References

[1] Erwin Schrödinger. An undulatory theory of the mechanics of atoms and molecules. 28(6):1049–1070, 1926.

[2] W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Physical Review*, 140(4A):A1133–A1138, 1965.

[3] M. Born and R. Oppenheimer. Zur quantentheorie der molekeln. 84(20):457–484, 1927.

[4] W. L. McMillan. Ground state of liquid he^4. *Physical Review*, 138(A442–A447):A442–A447, 1965.

[5] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Anna Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. 596(7873):583–589, 2021.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

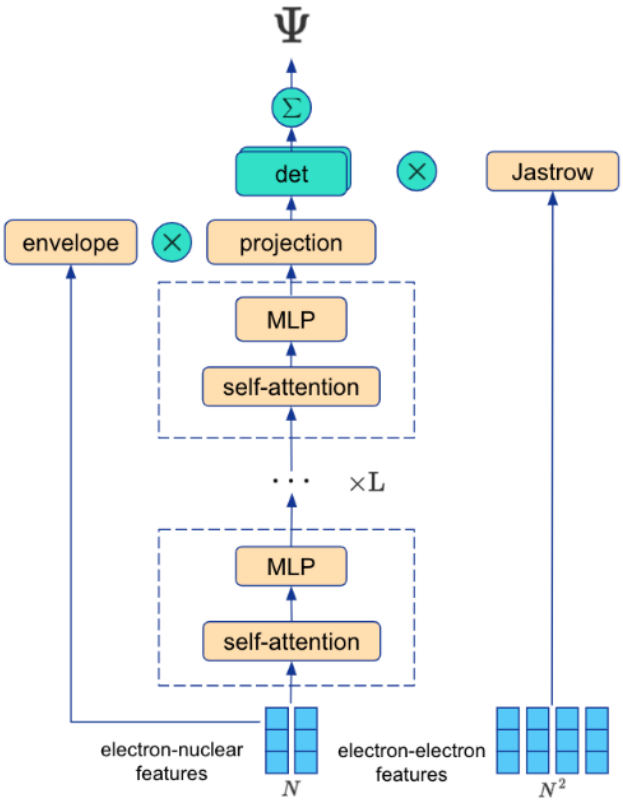

Figure 2: Architecture of Psi Former, Source: Pfau et all

[7] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. 378:686–707, 2019.

[8] Di Luo and Bryan K. Clark. Backflow transformations via neural networks for quantum many-body wave functions. *Physical Review Letters*, 122(22), 2019.

[9] Zhuoran Qiao, Matthew Welborn, Animashree Anandkumar, Frederick R. Manby, and Thomas F. Miller. OrbNet: Deep learning for quantum chemistry using symmetry-adapted atomic-orbital features. 153(12), 2020-09.

[10] David Pfau, James S. Spencer, Alexander G. D. G. Matthews, and W. M. C. Foulkes. Ab initio solution of the many-electron Schrödinger equation with deep neural networks. 2(3), 2020-09.

[11] Jan Hermann, Zeno Schätzle, and Frank Noé. Deep-neural-network solution of the electronic schrödinger equation. *Nature Chemistry*, 12(10):891–897, 2020.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 30, 2017.

[13] James S. Glehn, Ingrid von Spencer and David Pfau. A self-attention ansatz for ab-initio quantum chemistry, 2023.

[14] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. arXiv preprint, 2019.

[15] Niels Bohr. The quantum postulate and the recent development of atomic theory. *Nature*, 121(3050):580–590, 1928.

[16] Louis de Broglie. *Recherches Sur La Théorie Des Quanta*. PhD thesis, Université de Paris, 1924.

[17] Nouredine Zettili. *Quantum Mechanics: Concepts and Applications*. John Wiley & Sons, Chichester, UK, 2 edition, 2009.

[18] IUPAC. Atomic units. IUPAC Compendium of Chemical Terminology (Gold Book), 1997 / updated. See IUPAC Gold Book entry A00504; DOI: 10.1351/goldbook.A00504.

[19] S. N. Bose. Planck's law and the light quantum hypothesis. *Zeitschrift fr Physik*, 26:178–181, 1924.

[20] Albert Einstein. Quantentheorie des einatomigen idealen gases. *Sitzungsberichte der Preuischen Akademie der Wissenschaften, Physikalisch-mathematische Klasse*, pages 261–267, 1924.

[21] Enrico Fermi. Zur quantelung des idealen einatomigen gases. *Zeitschrift für Physik*, 36:902–912, 1926.

[22] Paul Adrien Dirac. On the theory of quantum mechanics. *Proceedings of the Royal Society of London. Series A*, 112:661–677, 1926.

[23] Wolfgang Pauli. Über den zusammenhang des abschlusses der elektronengruppen im atom mit der komplexstruktur der spektren. *Zeitschrift für Physik*, 31(1):765–783, 1925.

[24] John C. Slater. The theory of complex spectra. *Physical Review*, 34:1293–1322, 1929.

[25] T. Kato. On the eigenfunctions of many-particle systems in quantum mechanics. *Communications on Pure and Applied Mathematics*, 10(2):151–177, 1957.

[26] Robert Jastrow. Many-body problem with strong forces. *Physical Review*, 98(5):1479–1484, 1955.

[27] Particle Data Group. Review of particle physics. *Progress of Theoretical and Experimental Physics*, 2(8):083C01, 2024.

[28] Isaac Newton. *Philosophiae Naturalis Principia Mathematica*. Joseph Streater, London, 1687.

[29] Walther Ritz. Über eine neue methode zur lösung gewisser variationsprobleme der mathematischen physik. *Journal für die reine und angewandte Mathematik*, 135:1–61, 1909.

[30] W. M. C. Foulkes, L. Mitas, R. J. Needs, and G. Rajagopal. Quantum monte carlo simulations of solids. *Rev. Modern Phys.*, 73:33, 2001.

[31] M. Bajdich, L. Mitas, G. Drobný, L. K. Wagner, and K. E. Schmidt. Pfaffian pairing wave functions in electronic-structure quantum monte carlo simulations. *Phys. Rev. Lett.*, 96:130201, 2006.

[32] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949.

[33] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[34] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[35] Ronald A. Fisher. On the mathematical foundations of theoretical statistics. *Philosophical Transactions of the Royal Society of London. Series A*, 222:309–368, 1922.

[36] Shun-ichi Amari. Natural gradient works efficiently in learning. 10:251–276, 1998.

[37] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37 of *Proceedings of Machine Learning Research*, pages 2408–2417, Lille, France, July 2015. PMLR.