Variational Monte Carlo using a Neural Network Ansatz

Hersh Kumar

In collaboration with Professor Paulo Bedaque and Andy Sheng

University of Maryland

1. The Variational Method
   - Analytic Method
   - Variational Monte Carlo
2. Why Neural Networks?
3. Neural Network VMC
   - Symmetrization
   - Delta Function Sampling
4. Implementation and Results

Basic Principle

Given a Hamiltonian $\hat{H}$, with energy eigenstates $|\phi_n\rangle$, an arbitrary state $|\psi\rangle$ can be decomposed in terms of the eigenstates:

$$|\psi\rangle = \sum_n c_n |\phi_n\rangle$$

And the expectation value of the energy $\left\langle \psi \middle| \hat{H} \middle| \psi \right\rangle$ can be written as

$$\left\langle \psi \middle| \hat{H} \middle| \psi \right\rangle = \sum_n |c_n|^2 E_n$$

Variational Principle

Since $E_0 \leq E_1, E_2, \ldots,$

$$\left\langle \psi \middle| \hat{H} \middle| \psi \right\rangle \geq E_0 \langle \psi | \psi \rangle$$

Application

Given an arbitrary Hamiltonian $\hat{H}$, and some ansatz, $|\psi\rangle$, which is parameterized by some variable a, we can use the Variational Principle to approximate the ground state energy for the Hamiltonian.

We do so by minimizing $\left\langle \psi \middle| \hat{H} \middle| \psi \right\rangle$ with respect to our parameter, a. This minimum will provide the lowest possible energy for the chosen ansatz.

Generalizing to Higher Dimensions

Ansatzes with 1 parameter are inherently limited (only 1 “dimension” of variation). We can generalize this to an ansatz with any number of parameters. Suppose we have an ansatz ψ that depends on a set of parameters $\vec{\theta}$.

The configuration of ψ that produces the lowest energy will be some point in the parameter space, and can be found via optimization.

Generalizing to Higher Dimensions

Ansatzes with 1 parameter are inherently limited (only 1 “dimension” of variation). We can generalize this to an ansatz with any number of parameters. Suppose we have an ansatz ψ that depends on a set of parameters $\vec{\theta}$.

The configuration of ψ that produces the lowest energy will be some point in the parameter space, and can be found via optimization.

However, increasing the number of parameters makes it difficult to locate the minimum manually, and thus **numerical methods are needed**.

Generalizing to Higher Dimensions

Ansatzes with 1 parameter are inherently limited (only 1 “dimension” of variation). We can generalize this to an ansatz with any number of parameters. Suppose we have an ansatz ψ that depends on a set of parameters $\vec{\theta}$.

The configuration of ψ that produces the lowest energy will be some point in the parameter space, and can be found via optimization.

However, increasing the number of parameters makes it difficult to locate the minimum manually, and thus **numerical methods are needed**.

Goal

For a Hamiltonian $\hat{H}$, and a chosen ansatz, ψ, approximate the ground state energy via gradient descent in the space of parameters $\vec{\theta}$.

Leveraging Averages

The average value of the energy can be written as

$$E = \frac{\int \psi(\vec{x})^\dagger \hat{H} \psi(\vec{x})\, d\vec{x}}{\int \psi(\vec{x})^\dagger \psi(\vec{x})}$$

If we assume that the wavefunction represents a bosonic system, we can do a bit of math and write out the energy as a statistical average:

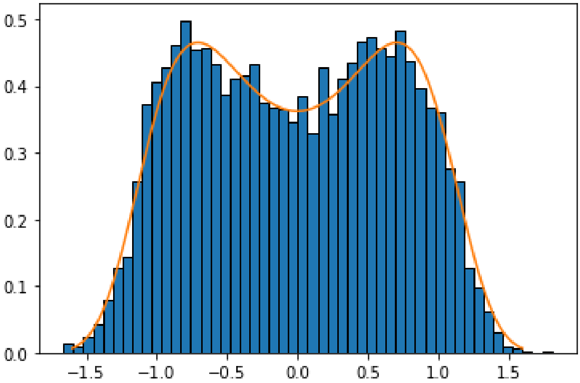
$$E = \left\langle \psi(\vec{x})^{-1} \hat{H} \psi(\vec{x}) \right\rangle_{\psi^2}$$

Doing some similar math for the gradient in parameter space, $\frac{\partial E}{\partial \theta}$:

$$\frac{\partial E}{\partial \theta} = 2 \left\langle \psi^{-1} \frac{\partial \log \psi}{\partial \theta} \hat{H} \psi \right\rangle_{\psi^2} - 2 \left\langle \psi^{-1} \hat{H} \psi \right\rangle_{\psi^2} \left\langle \frac{\partial \log \psi}{\partial \theta} \right\rangle_{\psi^2}$$

Monte Carlo Sampling

Computing these averages can be done via Monte Carlo methods, specifically Metropolis sampling, which lets us sample from a probability distribution.



Metropolis Overview

Given a probability distribution function $P(x)$:

1. Generate a starting sample, $\vec{x}_t$.
2. Generate a random shift, $\vec{r}$ (sampled uniformly within a range).
3. Let $\vec{x}' = \vec{x}_t + \vec{r}$. This is the proposed next sample.
4. Generate random $p \in [0, 1]$. If $p < \frac{P(\vec{x}')}{P(\vec{x})}$, accept the sample, and use it as the next starting sample. Otherwise, reject the sample, and keep the starting sample the same.

This is a Markov Chain algorithm, the next point is based on the previous point.

Intuitively, the algorithm is more likely to accept samples that are from high density regions, and reject samples from low density regions.

Algorithm Overview

For a given Hamiltonian $\hat{H}$:

1. Choose an ansatz $\psi(\vec{x}, \vec{\theta})$

Algorithm Overview

For a given Hamiltonian $\hat{H}$:

1. Choose an ansatz $\psi(\vec{x}, \vec{\theta})$
2. Sample from the distribution $\psi^2(\vec{x}, \vec{\theta})$ via the Metropolis method.

Algorithm Overview

For a given Hamiltonian $\hat{H}$:

1. Choose an ansatz $\psi(\vec{x}, \vec{\theta})$
2. Sample from the distribution $\psi^2(\vec{x}, \vec{\theta})$ via the Metropolis method.
3. Use the samples to compute averages for the energy, as well as the gradient in parameter space.

Algorithm Overview

For a given Hamiltonian $\hat{H}$:

1. Choose an ansatz $\psi(\vec{x}, \vec{\theta})$
2. Sample from the distribution $\psi^2(\vec{x}, \vec{\theta})$ via the Metropolis method.
3. Use the samples to compute averages for the energy, as well as the gradient in parameter space.
4. Use the gradient to update the parameters $\vec{\theta}$.

Algorithm Overview

For a given Hamiltonian $\hat{H}$:

1. Choose an ansatz $\psi(\vec{x}, \vec{\theta})$
2. Sample from the distribution $\psi^2(\vec{x}, \vec{\theta})$ via the Metropolis method.
3. Use the samples to compute averages for the energy, as well as the gradient in parameter space.
4. Use the gradient to update the parameters $\vec{\theta}$.
5. Repeat steps 2 through 4 until the parameters converge.

Algorithm Overview

For a given Hamiltonian $\hat{H}$:

1. Choose an ansatz $\psi(\vec{x}, \vec{\theta})$
2. Sample from the distribution $\psi^2(\vec{x}, \vec{\theta})$ via the Metropolis method.
3. Use the samples to compute averages for the energy, as well as the gradient in parameter space.
4. Use the gradient to update the parameters $\vec{\theta}$.
5. Repeat steps 2 through 4 until the parameters converge.

The end result is a set of parameters $\vec{\theta}$ that give the choice of ψ the lowest $\langle E \rangle$ for $\hat{H}$.

1. The Variational Method
   - Analytic Method
   - Variational Monte Carlo

2. Why Neural Networks?

3. Neural Network VMC
   - Symmetrization
   - Delta Function Sampling

4. Implementation and Results

Why Neural Networks?

Ansatzes are human-generated, based on similar Hamiltonians and their ground state solutions, or based on known properties of the ground state wavefunction:

- Harmonic oscillator potential → inverted quadratic, $\left(x+\frac{a}{2}\right)\left(x-\frac{a}{2}\right)$
- Helium atom potential → Hydrogen atom ground state with e^2s changed to $2e^2$

Why Neural Networks?

Ansatzes are human-generated, based on similar Hamiltonians and their ground state solutions, or based on known properties of the ground state wavefunction:

- Harmonic oscillator potential → inverted quadratic, $\left(x+\frac{a}{2}\right)\left(x-\frac{a}{2}\right)$
- Helium atom potential → Hydrogen atom ground state with e^2s changed to $2e^2$

Main Issue

Ansatz must be chosen by hand, and bad choices of ansatzes give very bad approximations of the ground state energy.

Why Neural Networks?

Ansatzes are human-generated, based on similar Hamiltonians and their ground state solutions, or based on known properties of the ground state wavefunction:

- Harmonic oscillator potential $\to$ inverted quadratic, $\left(x+\frac{a}{2}\right)\left(x-\frac{a}{2}\right)$
- Helium atom potential $\to$ Hydrogen atom ground state with e^2s changed to $2e^2$

Main Issue

Ansatz must be chosen by hand, and bad choices of ansatzes give very bad approximations of the ground state energy.

If some particular ansatz is used, it is unknown whether there exists some *better* ansatz, the variational principle does not give information on how close the optimized value is to E_0.

Why Neural Networks?

How can we use a neural network to get a good ansatz? We want the function to be as **expressive** as possible.

Why Neural Networks?

How can we use a neural network to get a good ansatz? We want the function to be as **expressive** as possible.

It turns out that feedforward neural networks can be used as universal function approximators, and as such, having a neural network as an ansatz provides a generalized wavefunction.

Ansatz

Use the ansatz $\psi(\vec{x}, \vec{\theta}) = e^{-f(\vec{x}, \vec{\theta})} e^{-|\vec{x}|^2}$, where $f(\vec{x}, \vec{\theta})$ is the output of the neural network.

Why Neural Networks?

How can we use a neural network to get a good ansatz? We want the function to be as **expressive** as possible.

It turns out that feedforward neural networks can be used as universal function approximators, and as such, having a neural network as an ansatz provides a generalized wavefunction.

Ansatz

Use the ansatz $\psi(\vec{x},\vec{\theta}) = e^{-f(\vec{x},\vec{\theta})}e^{-|\vec{x}|^2}$, where $f(\vec{x},\vec{\theta})$ is the output of the neural network.

The parameters that are being optimized are the weights of the neural network. Since neural networks can have arbitrary numbers of nodes, this allows for arbitrary “resolution” for our parameter space.

1. The Variational Method
   - Analytic Method
   - Variational Monte Carlo

2. Why Neural Networks?

3. Neural Network VMC
   - Symmetrization
   - Delta Function Sampling

4. Implementation and Results

Overview

Instead of picking an ansatz by hand, we just use the same neural network for all $\hat{H}$, and allow the gradient descent algorithm to converge to the correct functional form of the ground state wavefunction. This removes the need to have any intuition or knowledge about the ground state.

Overview

Instead of picking an ansatz by hand, we just use the same neural network for all $\hat{H}$, and allow the gradient descent algorithm to converge to the correct functional form of the ground state wavefunction. This removes the need to have any intuition or knowledge about the ground state.

However, this method introduces hyperparameters, such as the neural network structure or the choice of initial configuration of weights and biases. There are also two obstacles in the way of this being a practical methodology.

Obstacle 1: Symmetrization

We focus on bosonic systems, which require that the wavefunction be symmetric under particle interchange. For example, in the case of two particles, we require that

$$\psi(x_1, x_2) = \psi(x_2, x_1)$$

When choosing an ansatz by hand, it's simple to enforce symmetry under particle interchange. Enforcing this for a neural network is more difficult.

Obstacle 1: Symmetrization

We focus on bosonic systems, which require that the wavefunction be symmetric under particle interchange. For example, in the case of two particles, we require that

$$\psi(x_1, x_2) = \psi(x_2, x_1)$$

When choosing an ansatz by hand, it's simple to enforce symmetry under particle interchange. Enforcing this for a neural network is more difficult.

Naive idea: sort $\vec{x}$ before passing it into the neural network.

Obstacle 1: Symmetrization

We focus on bosonic systems, which require that the wavefunction be symmetric under particle interchange. For example, in the case of two particles, we require that

$$\psi(x_1, x_2) = \psi(x_2, x_1)$$

When choosing an ansatz by hand, it's simple to enforce symmetry under particle interchange. Enforcing this for a neural network is more difficult.

Naive idea: sort $\vec{x}$ before passing it into the neural network.
Issue: Sorting is not a differentiable operation, and as such the gradient is not computable.

Obstacle 1: Symmetrization

We focus on bosonic systems, which require that the wavefunction be symmetric under particle interchange. For example, in the case of two particles, we require that

$$\psi(x_1, x_2) = \psi(x_2, x_1)$$

When choosing an ansatz by hand, it's simple to enforce symmetry under particle interchange. Enforcing this for a neural network is more difficult.

Naive idea: sort $\vec{x}$ before passing it into the neural network.
Issue: Sorting is not a differentiable operation, and as such the gradient is not computable.

Better idea: Use a transformation of coordinates to convert the input $\vec{x}$ into some $\vec{x}'$ that **is** symmetric under particle interchange.

Symmetric Polynomials

Using the **Newton-Girard Identities**, we can take our N inputs, and pass them through a set of N symmetric polynomials, which will produce N outputs that are independent.

$$\begin{gathered}
x'_0 = x_0^0 + x_1^0 + \cdots + x_N^0 \\
x'_1 = x_0^1 + x_1^1 + \cdots + x_N^1 \\
\vdots \\
x'_N = x_0^N + x_1^N + \cdots + x_N^N
\end{gathered}$$

Intuitively, if we swap any two of the inputs, the outputs will be the same.

Updated Ansatz

Let $S(\vec{x})$ be the symmetrization function. The updated ansatz is now

$$\psi(\vec{x},\vec{\theta}) = e^{-f(S(\vec{x}),\vec{\theta})}e^{-|\vec{x}|^2}$$

Updated Ansatz

Let $S(\vec{x})$ be the symmetrization function. The updated ansatz is now

$$\psi(\vec{x}, \vec{\theta}) = e^{-f(S(\vec{x}),\vec{\theta})} e^{-|\vec{x}|^2}$$

This allows us to enforce symmetrization on the overall ansatz, as the input to the neural network is symmetric, causing the output to be symmetric. Since this transformation has a “nice” mathematical definition (unlike sorting), derivatives of S with respect to $\vec{x}$ are easy to compute.

Obstacle 2: Hamiltonians with Delta Functions

Consider the modified Lieb-Liniger model (N trapped bosons with short and long range interactions)

$$\hat{H} = \sum_{i=1}^{N} \underbrace{\left(-\frac{1}{2m}\frac{\partial^2}{\partial x^2} + \frac{1}{2}m\omega^2 x_i^2 \right)}_{\text{Harmonic potential}} + \sum_{i<j} (\underbrace{g\,\delta(x_i - x_j)}_{\text{Contact potential}} + \underbrace{\sigma|x_i - x_j|}_{\text{Long range pot.}})$$

Where g and σ are interaction strength parameters, and

$$\delta(x_i - x_j) = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

Obstacle 2: Hamiltonians with Delta Functions

Consider the modified Lieb-Liniger model (N trapped bosons with short and long range interactions)

$$\hat{H} = \sum_{i=1}^{N} \underbrace{\left(-\frac{1}{2m}\frac{\partial^2}{\partial x^2} + \frac{1}{2}m\omega^2 x_i^2\right)}_{\text{Harmonic potential}} + \sum_{i<j} \Big(\underbrace{g\,\delta(x_i - x_j)}_{\text{Contact potential}} + \underbrace{\sigma|x_i - x_j|}_{\text{Long range pot.}} \Big)$$

Where g and σ are interaction strength parameters, and

$$\delta(x_i - x_j) = \begin{cases} 1 & \text{if } x_i = x_j \\ 0 & \text{otherwise} \end{cases}$$

How do we get Monte Carlo sampling to deal with delta functions?

Delta Function Sampling

Energy

$$E = \left\langle \sum_i \frac{1}{2m}\left(\frac{\partial^2 A}{\partial x_i^2} - \left(\frac{\partial A}{\partial x_i}\right)^2\right) + \frac{1}{2}m\omega^2 x_i^2 + \sum_{i<j}\left(\sigma\left|x_i - x_j\right|\right) + g\frac{N(N-1)}{2}\frac{e^{-2A(\vec{x'})}}{e^{-2A(\vec{x})}}\mathcal{D}(x_2)\right\rangle_\psi$$

Gradient

$$\frac{\partial E}{\partial \theta} = 2E \cdot \left\langle \frac{\partial A(\vec{x})}{\partial \theta}\right\rangle_\psi - 2\left\langle \frac{\partial A(\vec{x})}{\partial \theta}\left[\sum_i \frac{1}{2m}\left(\frac{\partial^2 A}{\partial x_i^2} - \left(\frac{\partial A}{\partial x_i}\right)^2\right) + \frac{1}{2}m\omega^2 x_i^2 + \sum_{i<j}(\sigma\left|x_i - x_j\right|)\right]\right\rangle_\psi$$
$$- 2g\frac{N(N-1)}{2}\left\langle \frac{\partial A(\vec{x'})}{\partial \theta}\frac{e^{-2A(\vec{x'})}}{e^{-2A(\vec{x})}}\mathcal{D}(x_2)\right\rangle_\psi$$

where

$$\mathcal{D}(x_2) = \frac{1}{\alpha\sqrt{\pi}}e^{-x_2^2/\alpha^2}$$

In short, this works by rewriting the expectation value integrals without the delta function, via the introduction of a new distribution, $\mathcal{D}$, which we *can* sample from.

1. The Variational Method
    - Analytic Method
    - Variational Monte Carlo

2. Why Neural Networks?

3. Neural Network VMC
    - Symmetrization
    - Delta Function Sampling

4. Implementation and Results

Implementation

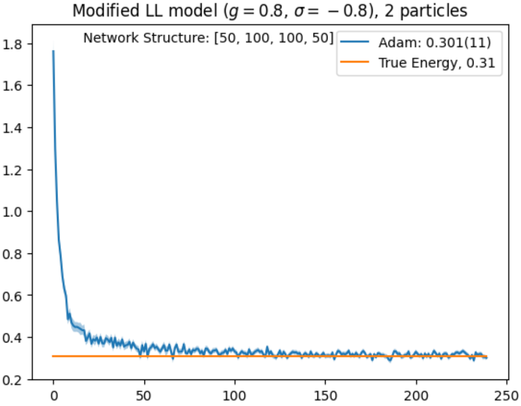
The implementation was done in Python, using JAX for reverse-mode automatic differentiation and just-in-time compilation. Further optimization was done via numpy vectorization methods.

```
dnn_dtheta = jit(grad(A, 1))
vdnn_dtheta = vmap(dnn_dtheta, in_axes=(0, None), out_axes=0)
```

Runtime is largely based on sample count and network size, as well as the machine's CPU.

Example Optimization

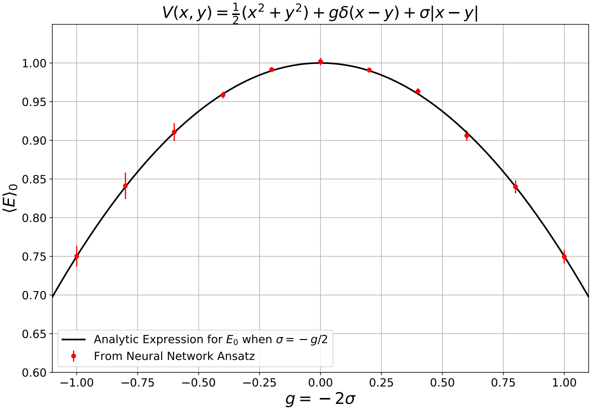
An example gradient descent optimization run for $g = 0.8$ and $\sigma = -0.8$, with around $20,000$ parameters.



The total runtime for this optimization was around 7 minutes.

Comparison to Analytic Results

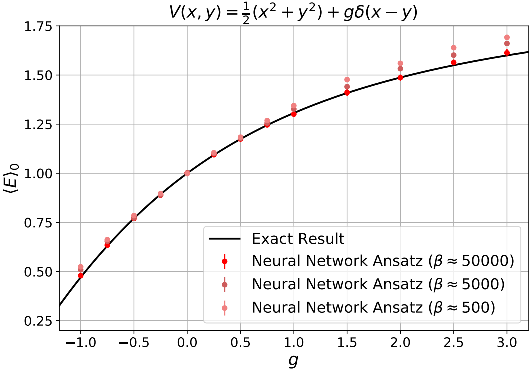
For the modified Lieb Liniger model, which is analytically solvable in the case where $\sigma = -g/2$, the neural network ansatz matches the analytical solution:



For more plots, see arXiv/2309.02352

Influence of Neural Network Structure

Increasing the number of parameters in the optimization, which is the number of weights in the neural network, increases the accuracy of the optimization.



For more plots, see arXiv/2309.02352

Overview

We've shown that a neural network ansatz can obtain approximations of the ground state energy of bosonic Hamiltonians that match the analytic expectations. We also present a method of enforcing bosonic symmetrization and delta function sampling in 1 dimension.

What's next?

- Consider systems in 2D or 3D. Requires new methods for dealing with delta functions in Hamiltonians.
- Benchmarking the ideal network size for Hamiltonians.
- Exploring optimization methods such as sample reuse and GPU parallelization.

Thank You!

Questions?