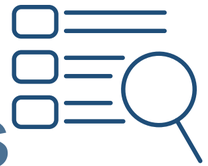
An Introduction to Machine Learning for Solving Quantum Many-Body Problems

ML4Science Group

M.Sc. student: Diego Peña Angeles

Advisor: Huziel E. Sauceda Felix

Contents



- Variationial Monte Carlo
- Neural Networks
- Deep Neural Networks ansatz for VMC
- Examples
- Conclusions

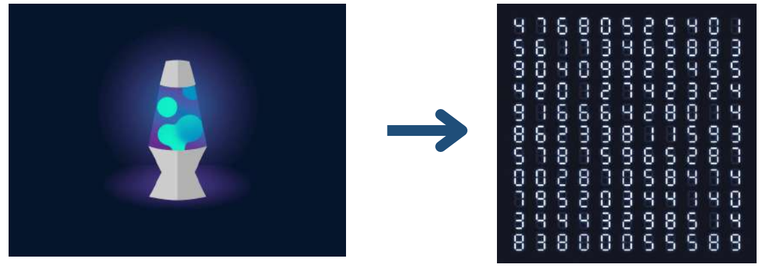
The fundamental laws necessary for the mathematical treatment of a large part of physics and the whole of chemistry are thus completely known, and the difficulty lies only in the fact that application of these laws leads to equations that are too complex to be solved.

Paul Dirac

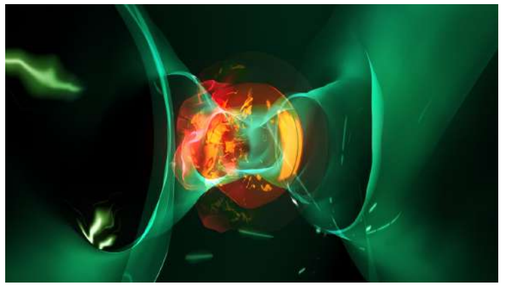
Monte Carlo methods

Method of estimating the value of an unknown quantity with the help of inferential statistics.

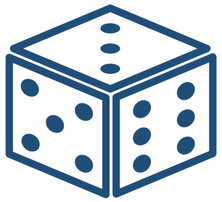
Computational technique used to model and analyze complex systems through of random sampling.



Method in quantum mechanical simulations



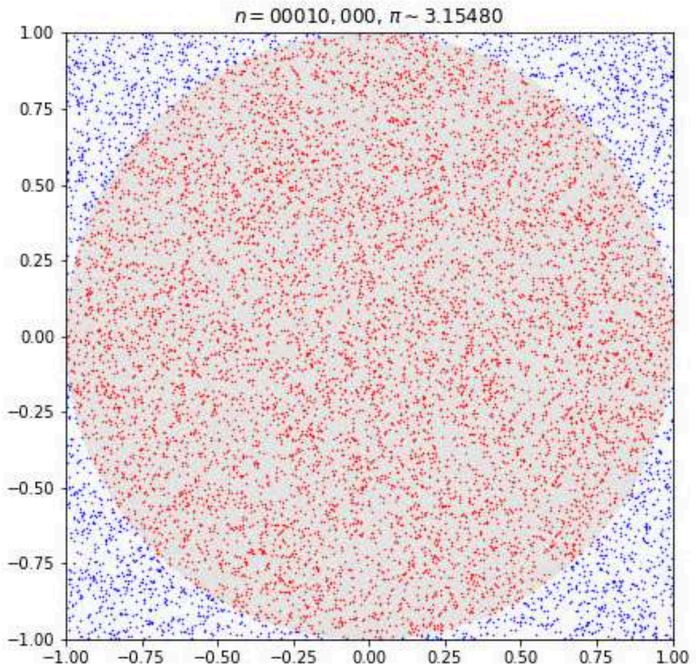
A simple example of Monte Carlo simulation



Basic idea of Monte Carlo through the "dartboard method"

Throw darts which land randomly within the square,

compute A_{circle} and compute π

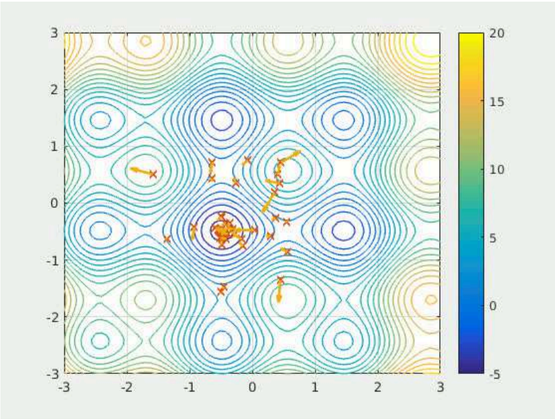


$$\frac{\#\text{hits inside the circle}}{\#\text{ hits inside the square}} = \frac{A_{circle}}{A_{square}} = \frac{\pi}{4}$$

many many hits

Variational Monte Carlo

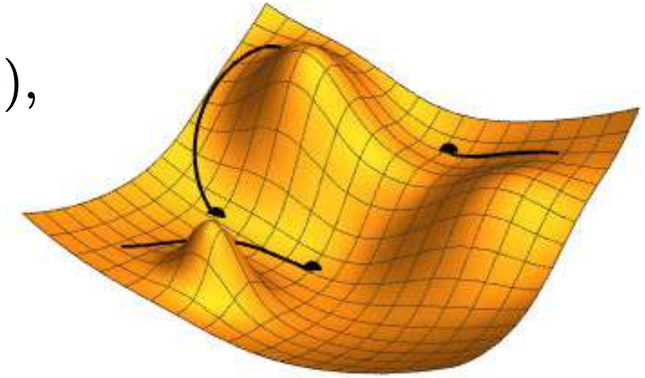
We will use the variational principle where given a hamiltonian:



$$\hat{H} = -\frac{\hbar}{2m}\sum_{i=1}^{N}\hat{\nabla}_i^2 + \sum_{i=1}^{N}\hat{U}_{ext}(\mathbf{r}_i) + \sum_{i<j=1}^{N}\hat{U}_{int}(r_{ij}),$$

$$E_0 \leq E_\alpha$$

$$\langle\psi_0|\hat{H}|\psi_0\rangle \leq \langle\psi_\alpha|\hat{H}|\psi_\alpha\rangle$$



$$\hat{\alpha} = argmin_\alpha E(\alpha)$$

$$\frac{\langle\psi_\alpha|\hat{H}|\psi_\alpha\rangle}{\langle\psi_\alpha|\psi_\alpha\rangle} \geq E_0 \Rightarrow \quad \frac{\int \psi_\alpha H \psi_\alpha}{\int \psi_\alpha^* \psi_\alpha} \geq E_0 \Rightarrow \quad \frac{\int \psi_\alpha^* \psi_\alpha H \frac{\psi_\alpha}{\psi_\alpha}}{\int \psi_\alpha \psi_\alpha} \geq E_0 \Rightarrow \quad \frac{\int |\psi_\alpha|^2 \frac{H\psi_\alpha}{\psi_\alpha}}{\int |\psi_\alpha|^2} \geq E_0 \Rightarrow \quad \int \rho_\alpha E^\alpha_{Local} \geq E_0$$

$$\langle E_\alpha \rangle = \frac{1}{N}\sum_{i=1}^{N} E^\alpha_{Local}$$

The Limitation of VMC

The method relies mostly on giving a really accurate trial wave function.

Why use neural networks?

- **Ability to learn:**

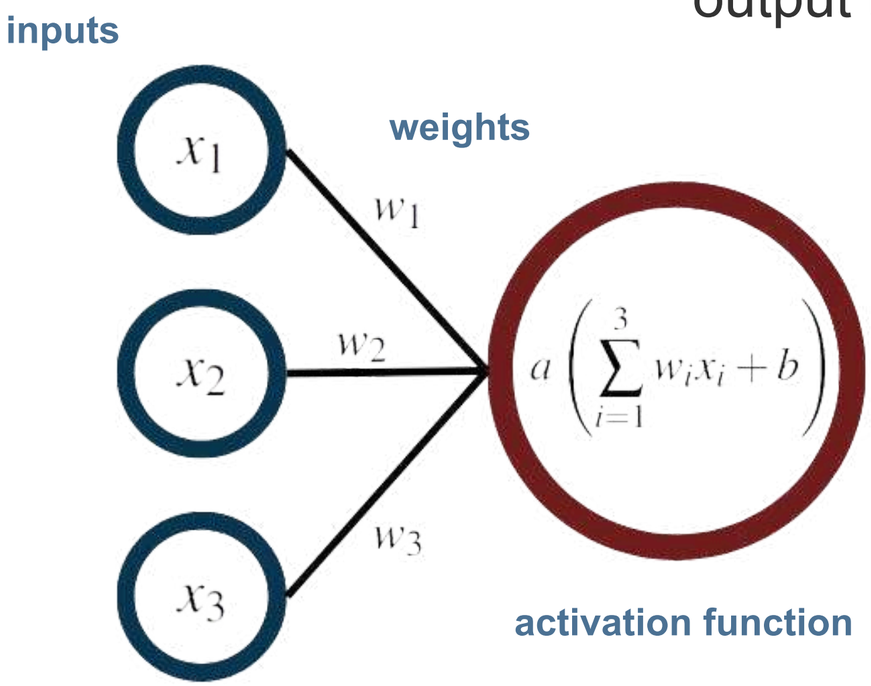
NN’s figure out how to perform their function on theirown.

- **Ability to generalize:**

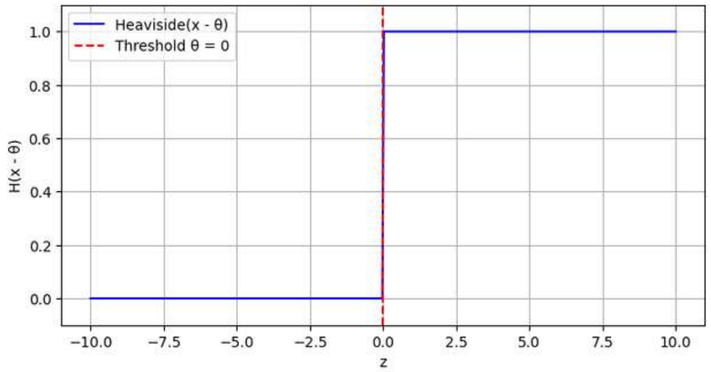
Produce reasonable outputs for inputs it has not beentaught how to deal with.

What is a neuron?

A neuron is a mathematical function that takes in input data, processes it, and passes the output to other neurons in the network.

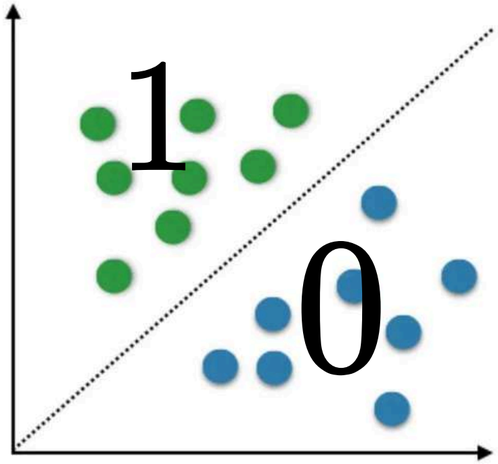


$$a(z)$$

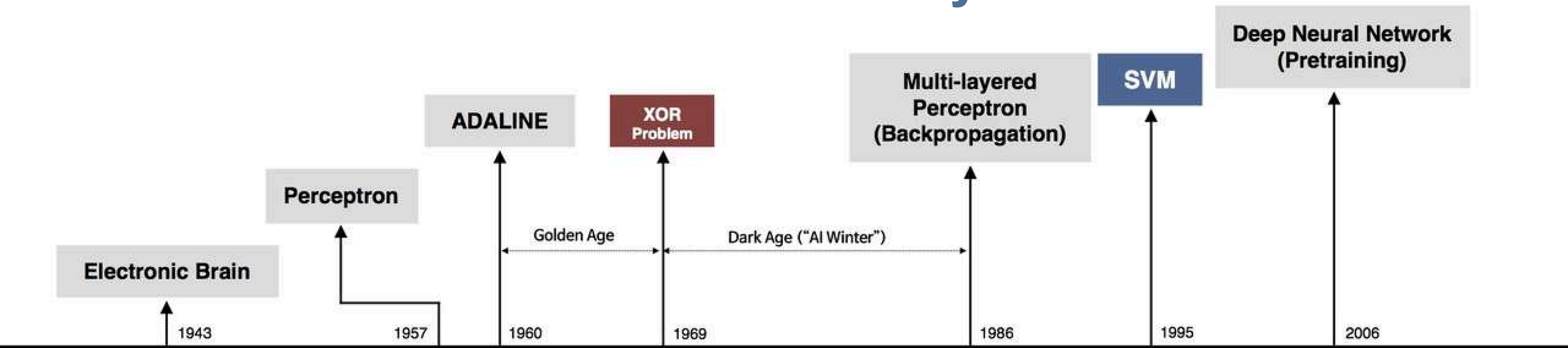


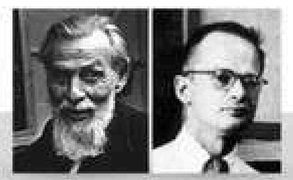
Rosenblatt perceptron

$$z = \sum_i w_i x_i + b > \theta$$



Brief History





S. McCulloch - W. Pitts

F. Rosenblatt

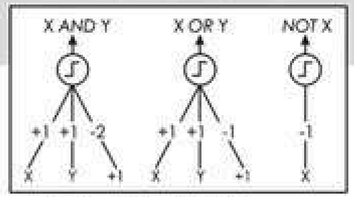
B. Widrow - M. Hoff

M. Minsky - S. Papert

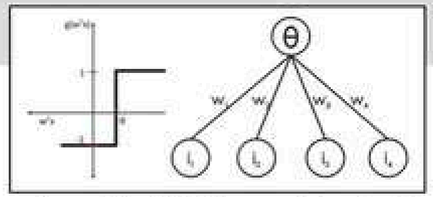
D. Rumelhart - G. Hinton - R. Wiliams

V. Vapnik - C. Cortes

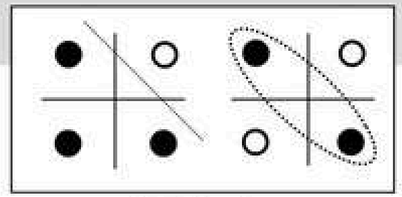
G. Hinton - S. Ruslan



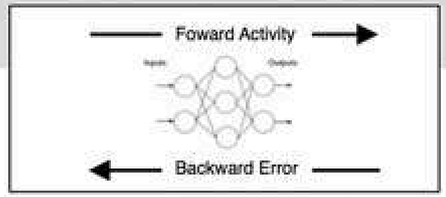
- Adjustable Weights
- Weights are not Learned



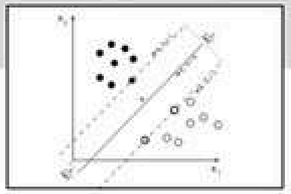
- Learnable Weights and Threshold



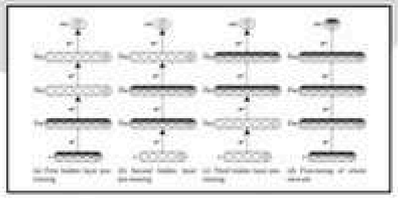
- XOR Problem



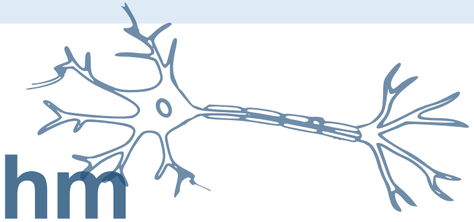
- Solution to nonlinearly separable problems
- Big computation, local optima and overfitting



- Limitations of learning prior knowledge
- Kernel function: Human Intervention



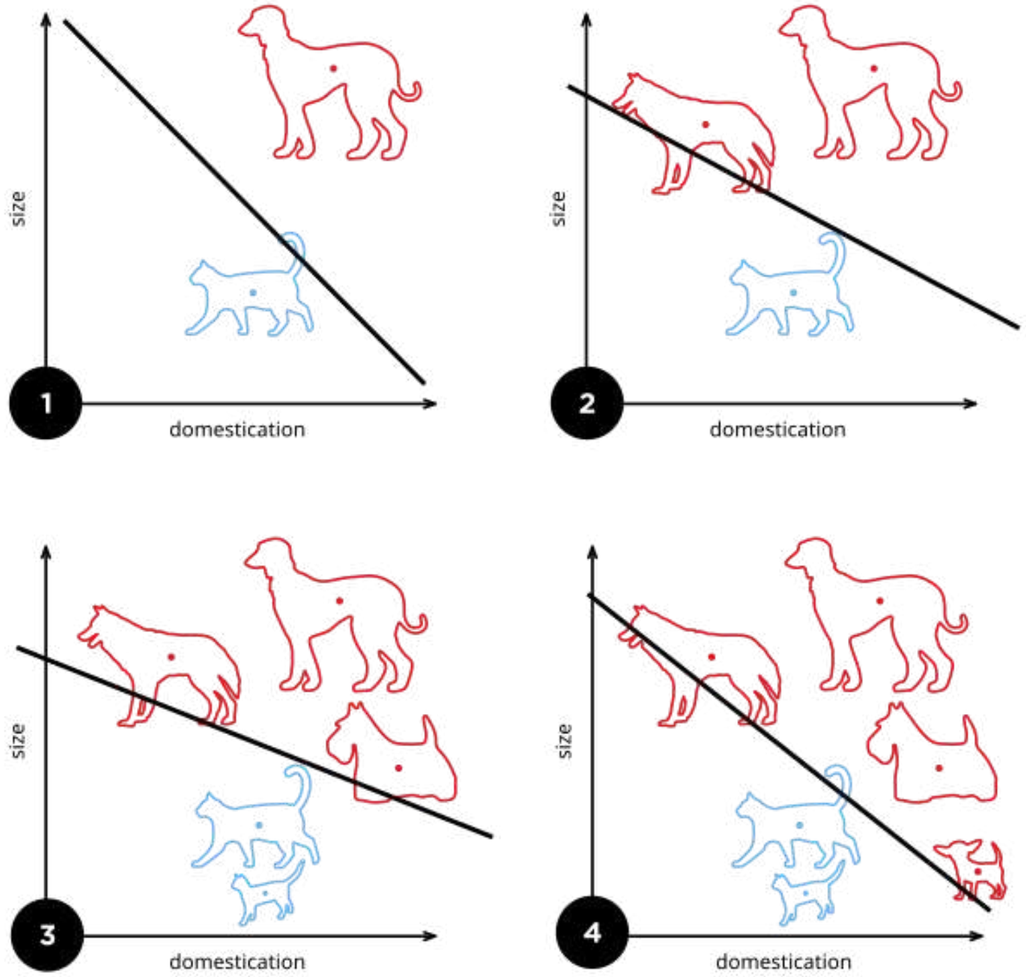
- Hierarchical feature Learning



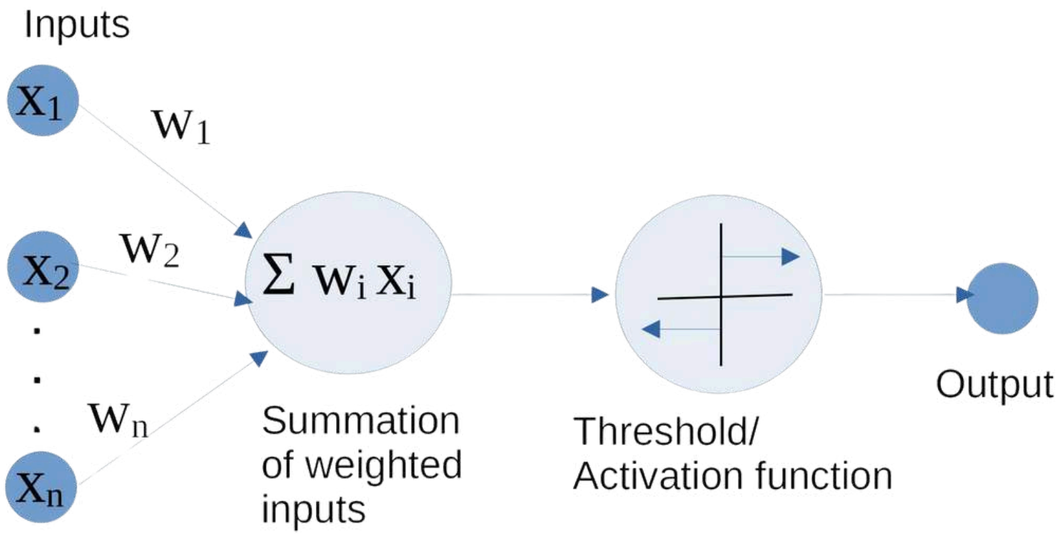
Learning algorithm

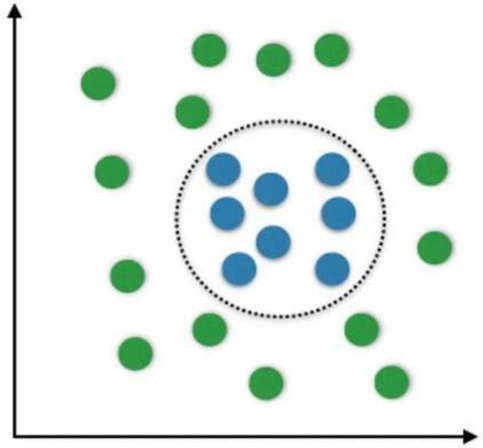
Steps:

- Initialize randomly the weights and bias.
- Calculate the actual output.
- Update the weights given an error metric.

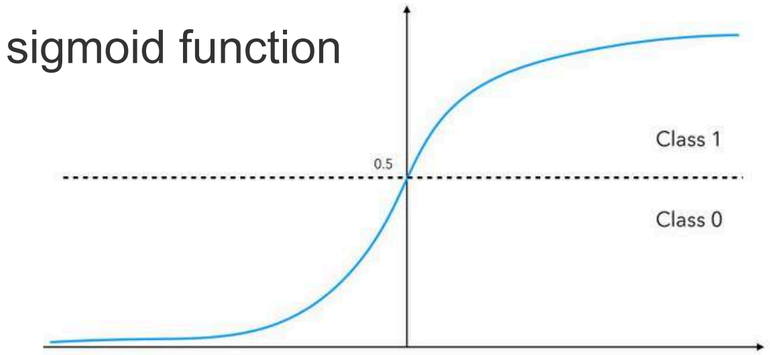


Perceptron





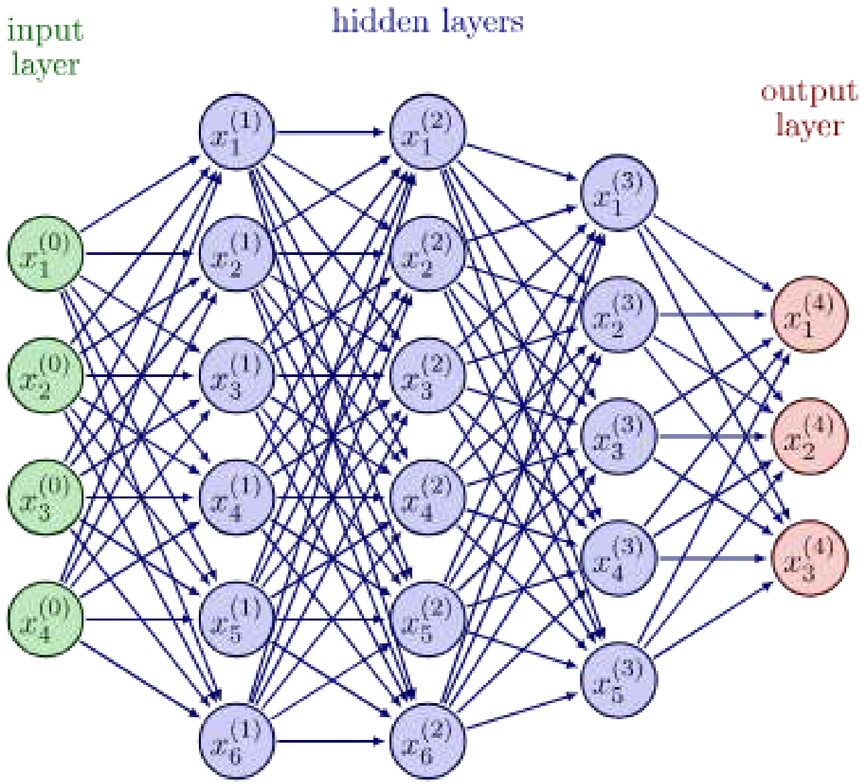
$$\sum_i w_i x_i + b > \theta$$



When the dataset is not linearly separable, then there is no way for a single perceptron to converge.

Multilayer perceptrons

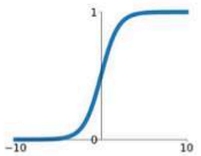
Feedforward artificial neural network, consisting of fully connected neurons with a nonlinear **activation function**.



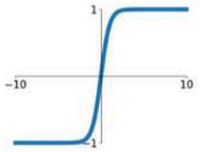
Sigmoidal activation function

$$f(x) \to \begin{cases} 1 & \text{as } x \to +\infty \\ 0 & \text{as } x \to -\infty \end{cases}$$

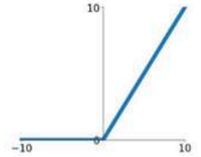
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



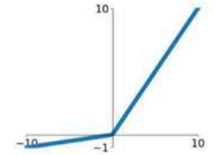
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

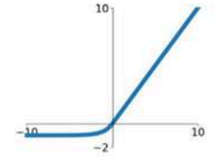


Leaky ReLU
$\max(0.1x, x)$



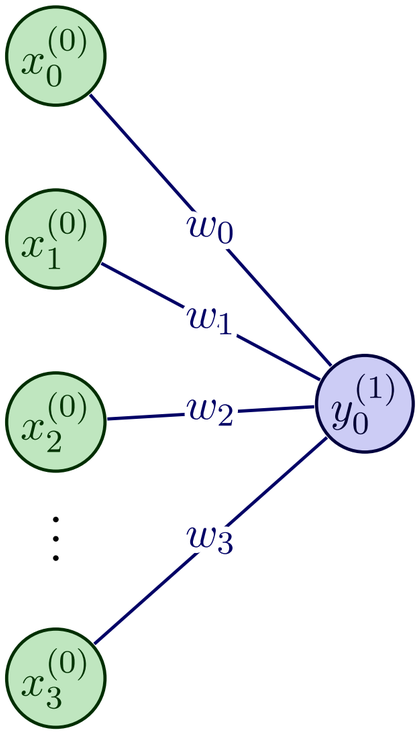
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



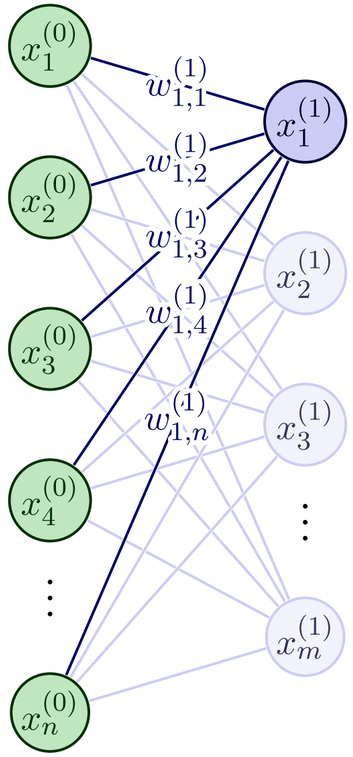
Multilayer perceptrons

We can represent a single perceptron as a functional form $\psi : \mathbb{R}^N \to \mathbb{R}$



$$\psi := \sigma\left(\sum_i^N w_i x_i + b\right)$$

$$\psi := \sigma\left(\mathbf{w}\cdot\mathbf{x} + b\right)$$



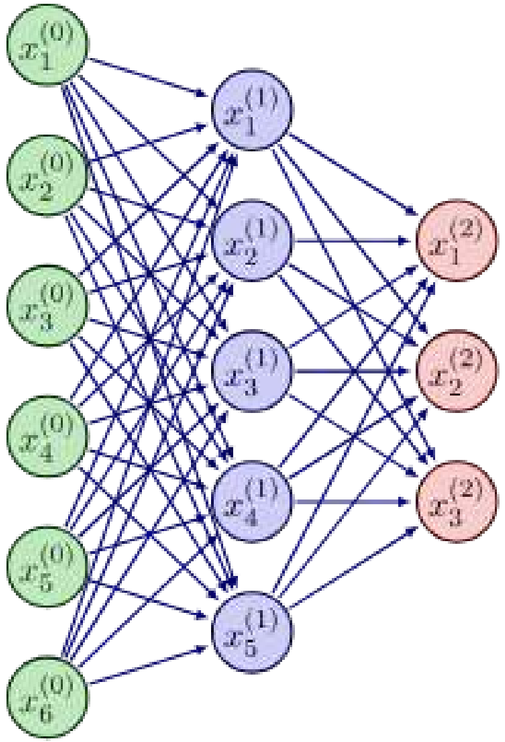
$$x_1^{(1)} := \sigma\left(\sum_i^N w_{1,i}^{(1)} x_i^{(0)} + b^{(1)}\right)$$

$$x_1^{(1)} := \sigma\left(\mathbf{w_1^{(1)}}\cdot\mathbf{x^{(0)}} + b^{(1)}\right)$$

$$\vdots$$

$$x_m^{(1)} := \sigma\left(\mathbf{w_m^{(1)}}\cdot\mathbf{x^{(0)}} + b^{(1)}\right)$$

Perceptron



$$W^{(l)} = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \dots & w_{1,n}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \dots & w_{2,n}^{(l)} \\ \dots & \dots & \ddots & \\ w_{m,1}^{(l)} & w_2^{(l)} & \dots & w_{m,n}^{(l)} \end{pmatrix} \quad b^{(l)} := (b_{x_1^{(l)}}, \dots, b_{x_m^{(l)}})^\top$$

$$\psi^{(l)} := \sigma\left(\mathbf{W^{(l)}} \cdot \mathbf{x}^{(l-1)} + b^{(l)}\right)$$

The big advantage of such a representation in matrix form is that it can be leveraged by parallel computation using a GPU, for instance.

Universal Aproximation Theorem

Let σ be any continuous discriminatory function. Then finite sums of the form

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j^T x + b_j), \quad w_j \in \mathbb{R}^n, \alpha_j, b_j \in \mathbb{R}$$

are dense in $C(I_n)$ Given $\epsilon > 0$ and $f \in C(I_n)$ there is a $G(x)$ such that

$$|G(x) - f(x)| < \epsilon \quad \forall x \in I_n$$

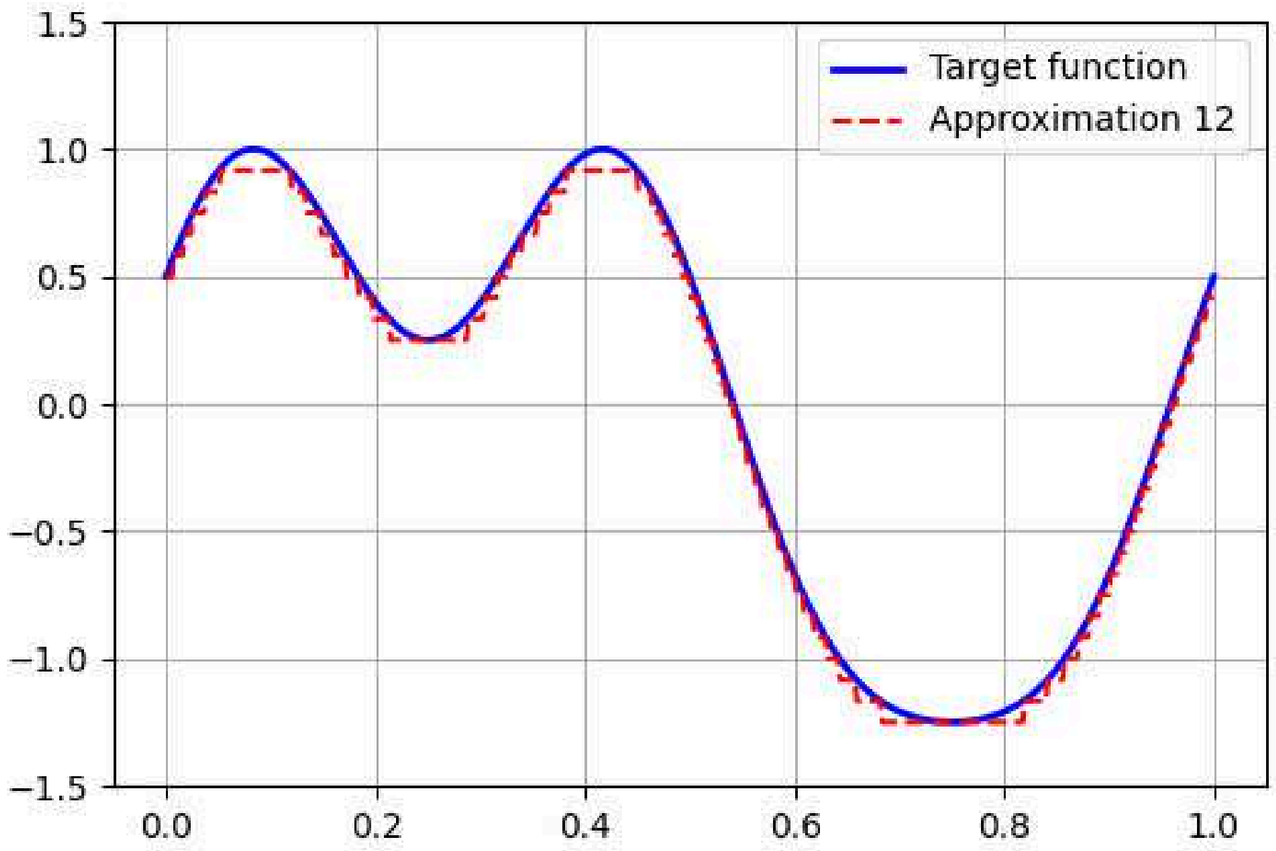
Universal Aproximation Theorem

Let f be a bounded and piecewise continuous function on $\mathbb{R}$

Then

$$f(x) \approx \sum_{i=1}^{N} a_i H(x - x_i)$$

As $N \to \infty$, this approximation converges to $f(x)$ pointwise.



What are neural networks?

A neural network is a mapping between an input vector that is processed through the neurons weights producing an output vector:

$$\boldsymbol{\mathcal{F}} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

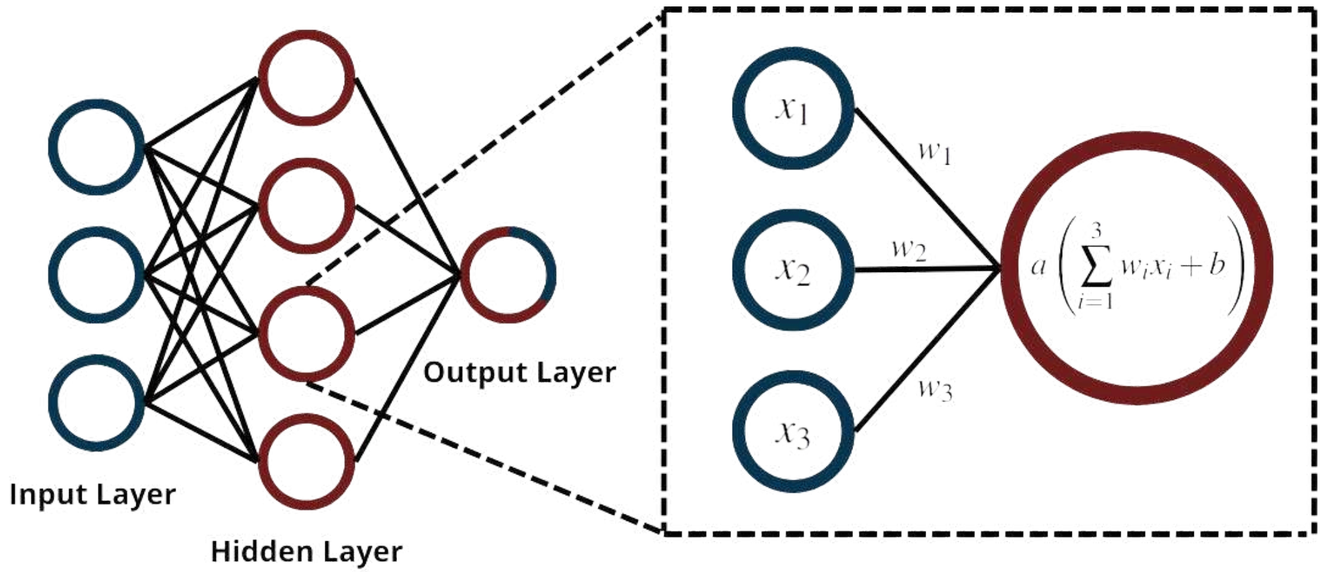
$$\mathcal{F}(x) = g\left(f_L\left(f_{L-1}\left(\ldots f_1(\vec{x})\right)\right)\right)$$

What are neural networks?

Each hidden layer is defined as:

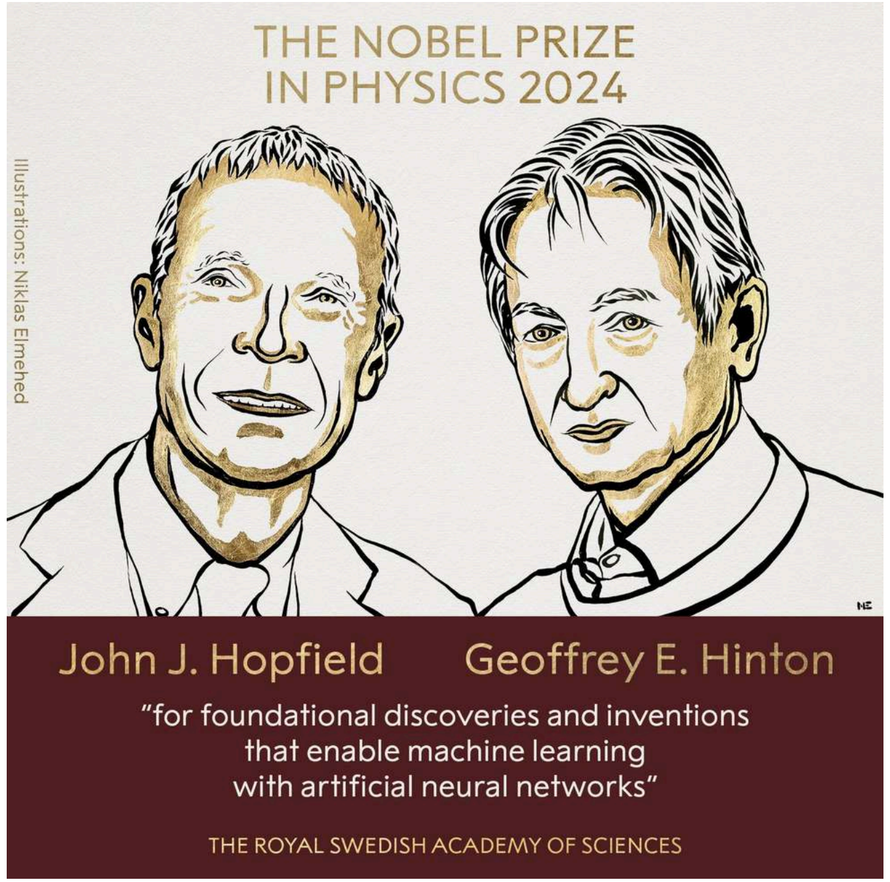
$$f_l(\mathbf{x}) : \mathbb{R}^{N_{l-1}} \rightarrow \mathbb{R}^{N_l}$$

$$f_l(\mathbf{x}^{(l-1)}) = a^{(l)} \left(\sum_{i=1}^{N_{l-1}} w_i^{(l)} x_i^{(l-1)} + b^{(l)} \right)$$
$$= a^{(l)}(\mathbf{W}^{(l)} \cdot \mathbf{x}^{(l-1)} + b^{(l)})$$



where l is the number of layer, a is a nonlinear function, w are the weights and b the bias.

Backpropagation



Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

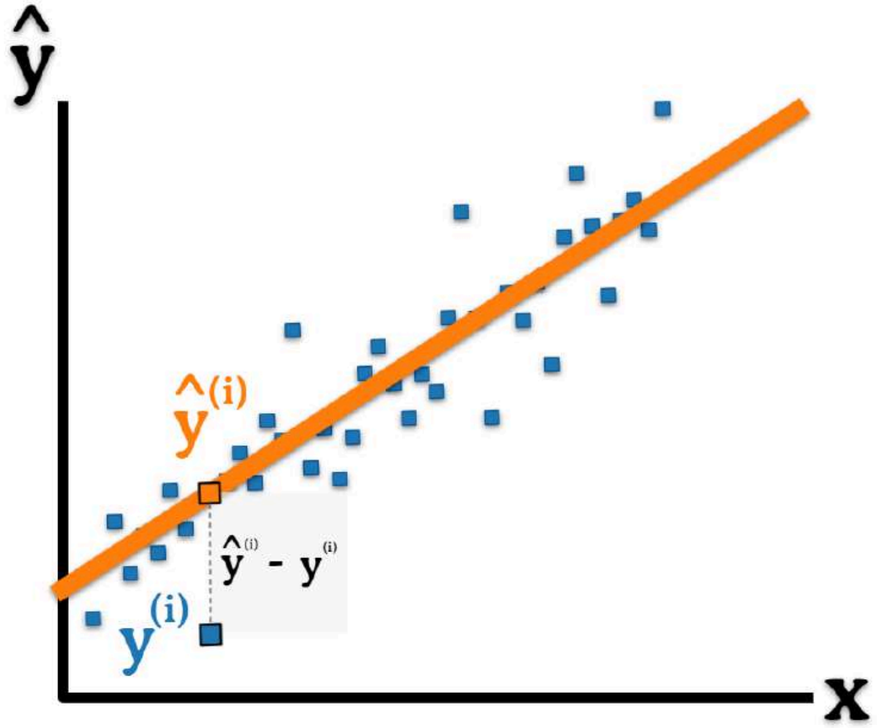
We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Backpropagation

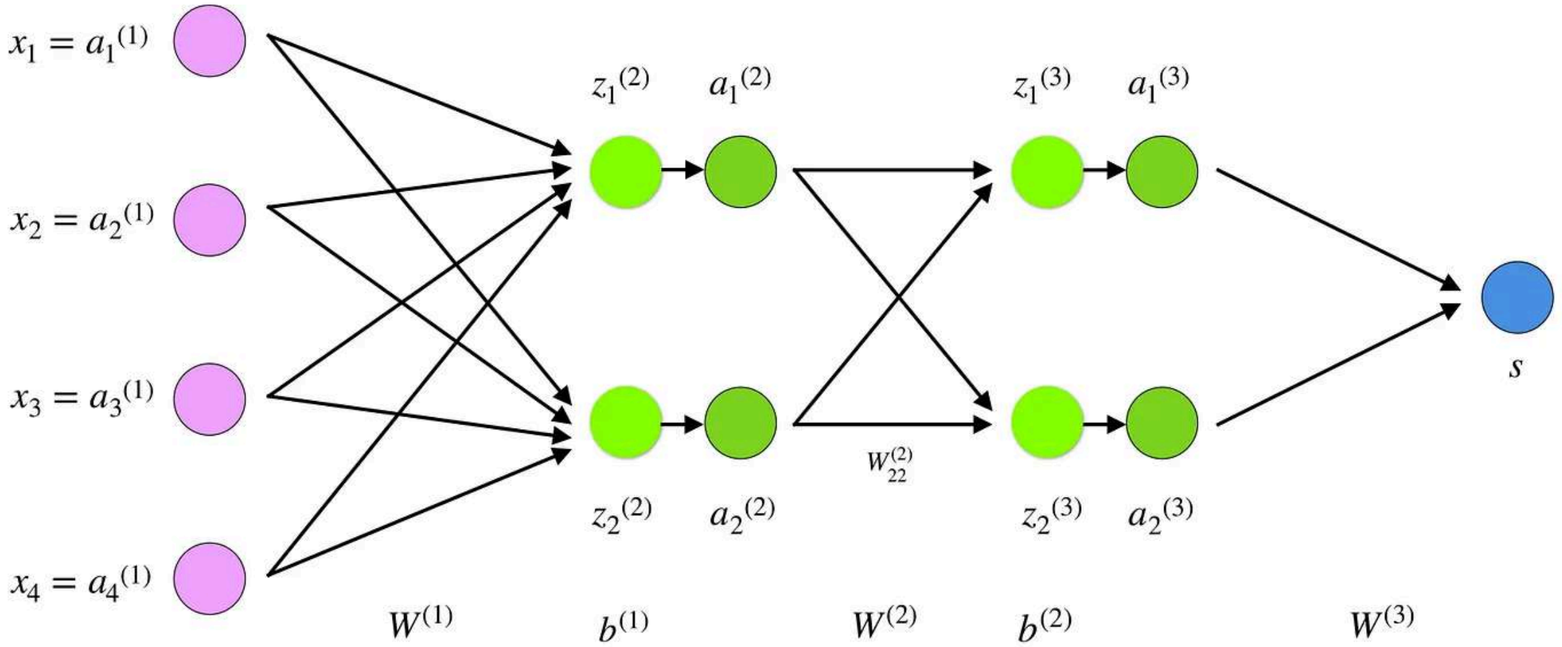
$$J(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left|\hat{y}_{NN}^{(i)} - y^{(i)}\right|$$

$$\nabla J(W,b) = \left(\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_M}, \frac{\partial J}{\partial b_1}, \frac{\partial J}{\partial b_2}, \cdots, \frac{\partial J}{\partial b_N}\right)$$

$$\{W', b'\} = \{W, b\} - \nabla J_{\{W,b\}}$$



Backpropagation



$$x_i = a_i^{(1)}, i \in 1,2,3,4$$

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

$$\hat{y}_{NN} = W^{(3)}a^{(3)}$$

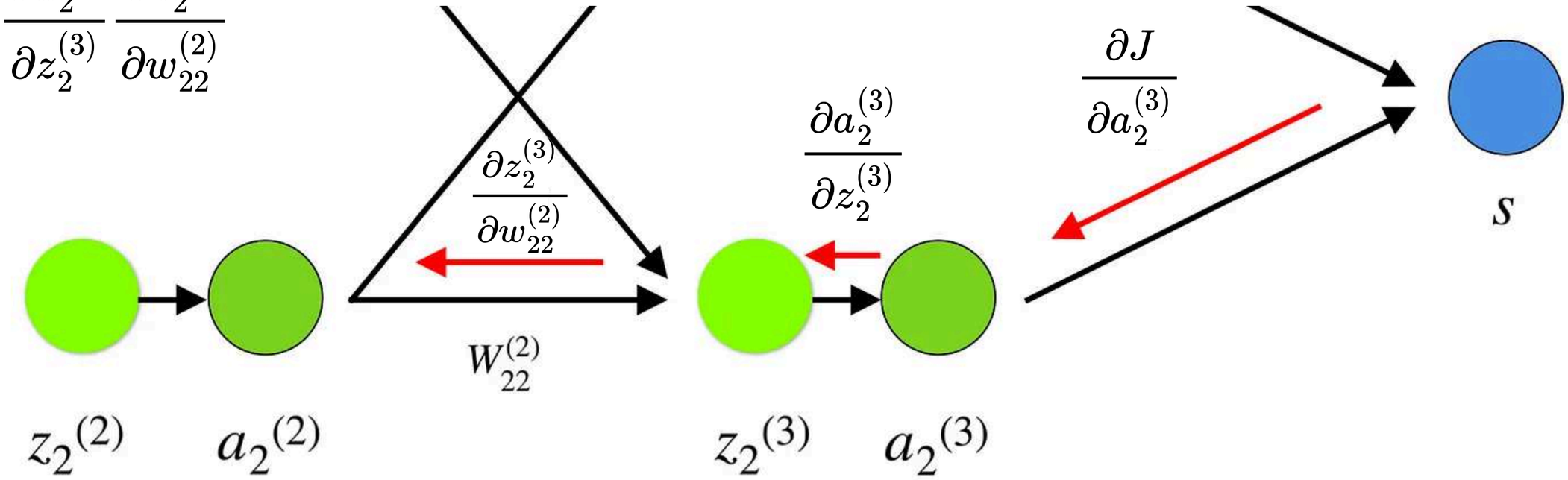
$$J(W,b) = |\hat{y}_{NN} - y|$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} & W_{14}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} & W_{24}^{(1)} \end{bmatrix} \quad b^{(1)} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + W_{14}^{(1)}x_4 \\ W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + W_{24}^{(1)}x_4 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Backpropagation

$$\nabla J(W,b) = \left(\frac{\partial J}{\partial w_{11}^{(1)}}, \frac{\partial J}{\partial w_{12}^{(1)}}, \ldots, \frac{\partial J}{\partial b_1^{(1)}}, \frac{\partial J}{\partial b_2^{(1)}}, \ldots \right)$$

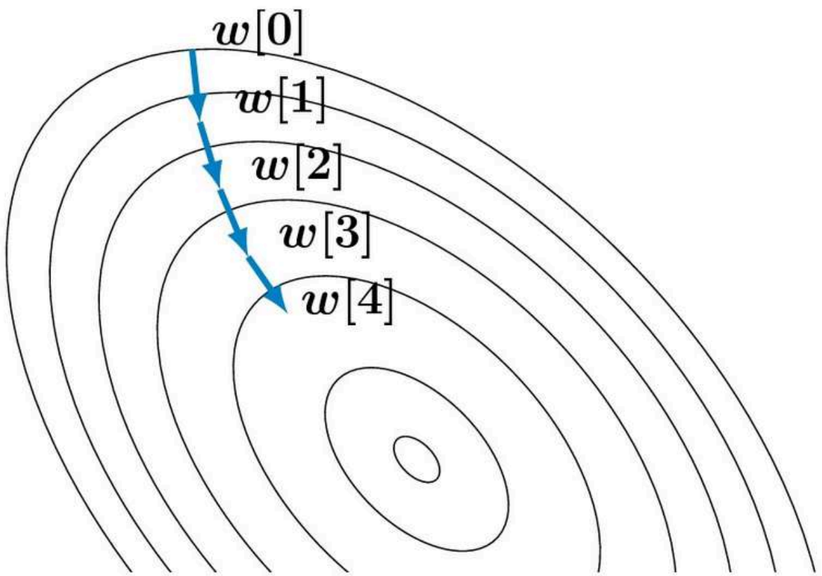
$$\frac{\partial J}{\partial w_{22}^{(2)}} = \frac{\partial J}{\partial a_2^{(3)}} \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$$

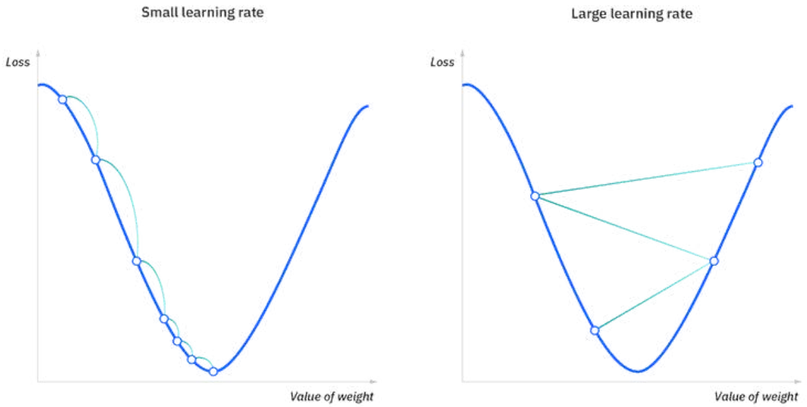


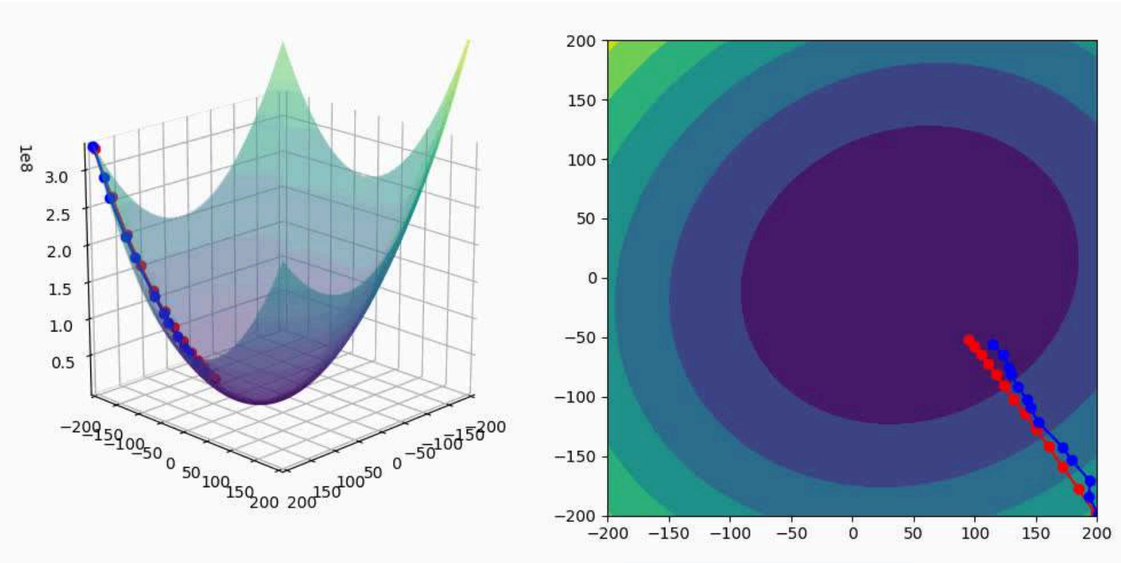
Gradient Descent

$$\{W', b'\} = \{W, b\} - \alpha \nabla J_{\{w,b\}}$$

learning rate

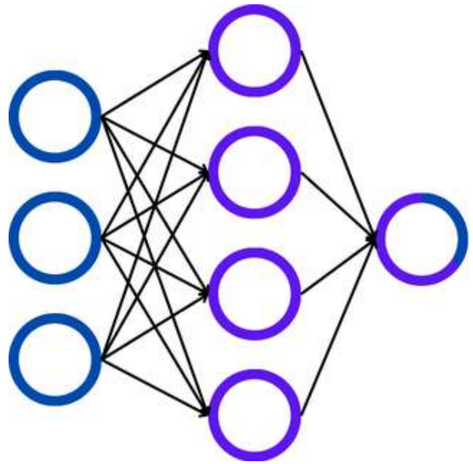






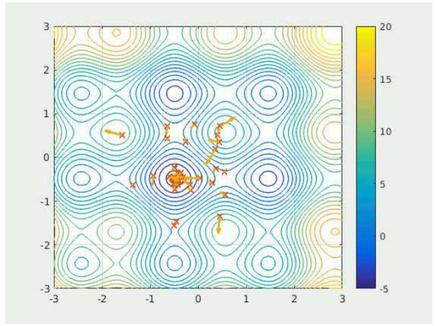
Neural Network ansatz

$$\begin{bmatrix} \vec{x}_1 \\ \vec{x}_2 \\ \vdots \\ \vec{x}_N \end{bmatrix}$$



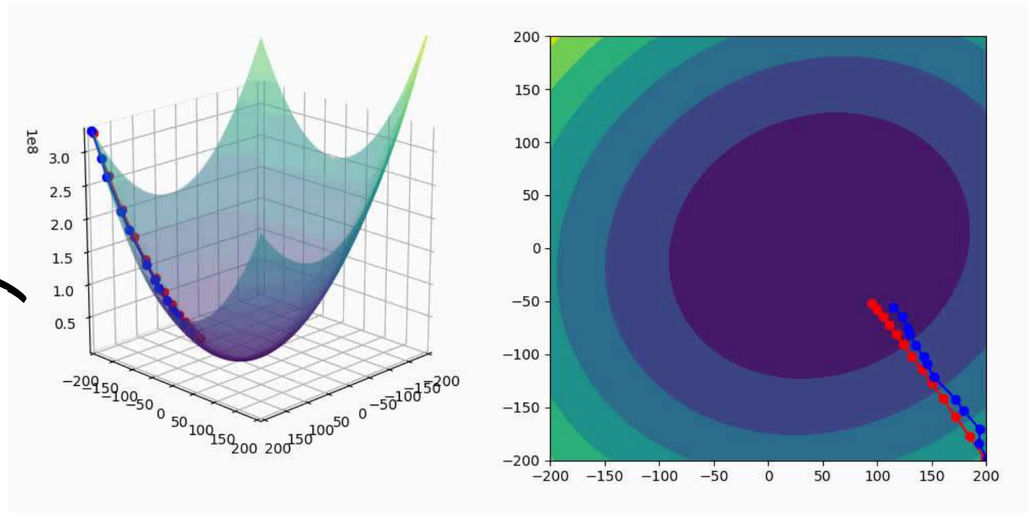
$$\Psi_{\{W,b\}}(\vec{x}_1, \ldots, \vec{x}_N)$$

$$\frac{\langle \Psi_{\{W,b\}} | \hat{H} | \Psi_{\{W,b\}} \rangle}{\langle \Psi_{\{W,b\}} | \Psi_{\{W,b\}} \rangle} = E_{\{W,b\}} \geq E_0$$



$$\{\vec{x}'\} \sim \left|\Psi_{\{w',b'\}}\right|^2$$

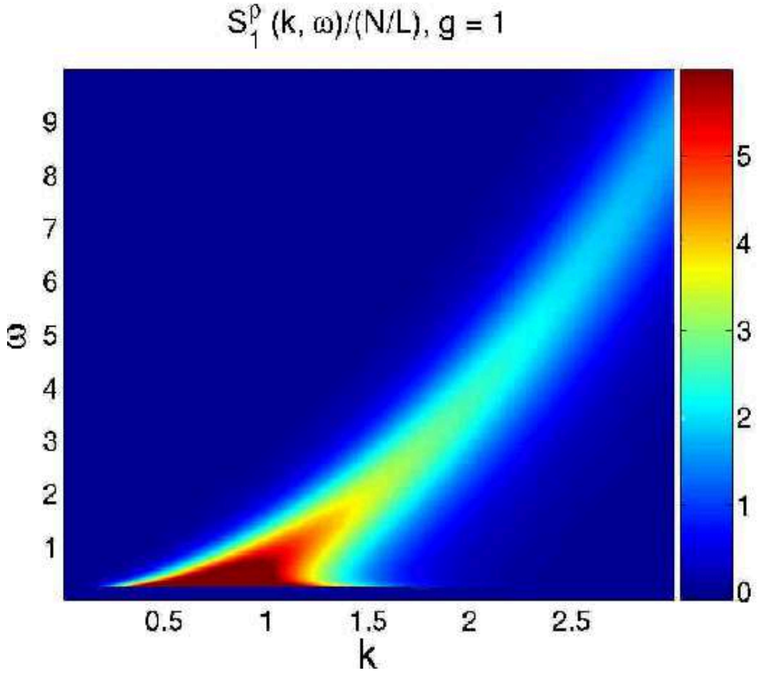
$$\begin{bmatrix} \vec{x}'_1 \\ \vec{x}'_2 \\ \vdots \\ \vec{x}'_N \end{bmatrix}$$



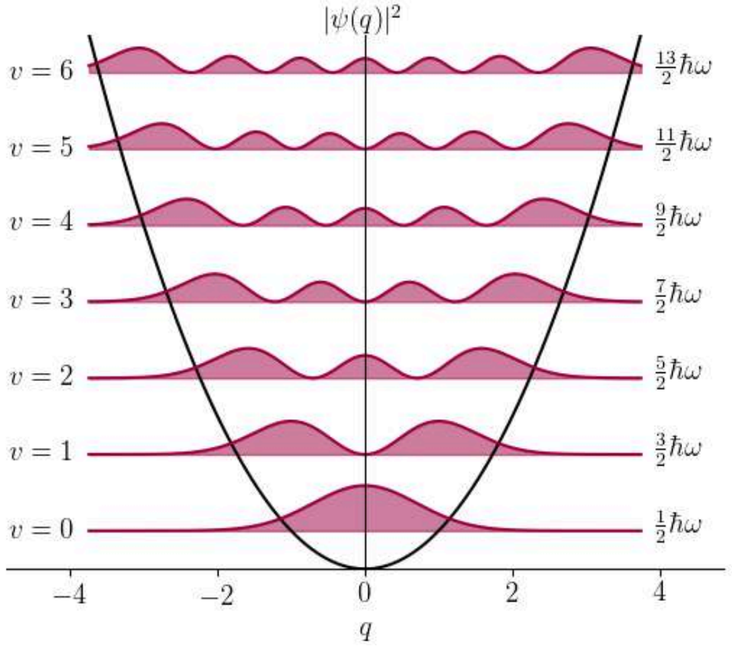
$$\{W', b'\} = \{W, b\} - \nabla E_{\{W,b\}}$$

Toy examples

We will apply the previously described methodology to two quantum system



Lieb Linger model with a linear interaction



Harmonic Oscillator

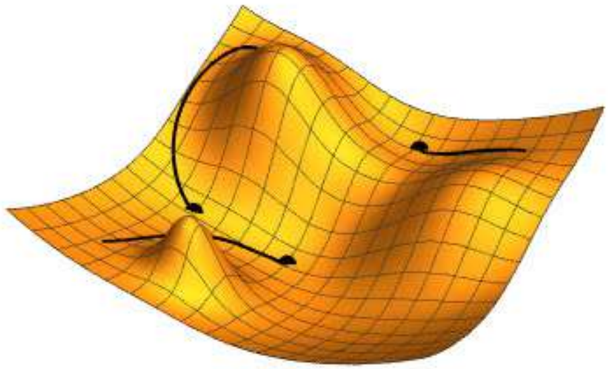
Example: 1D Harmonic Oscillator

$$\hat{H} = \hat{T} + \hat{V} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + \frac{1}{2}m\omega^2 x^2$$

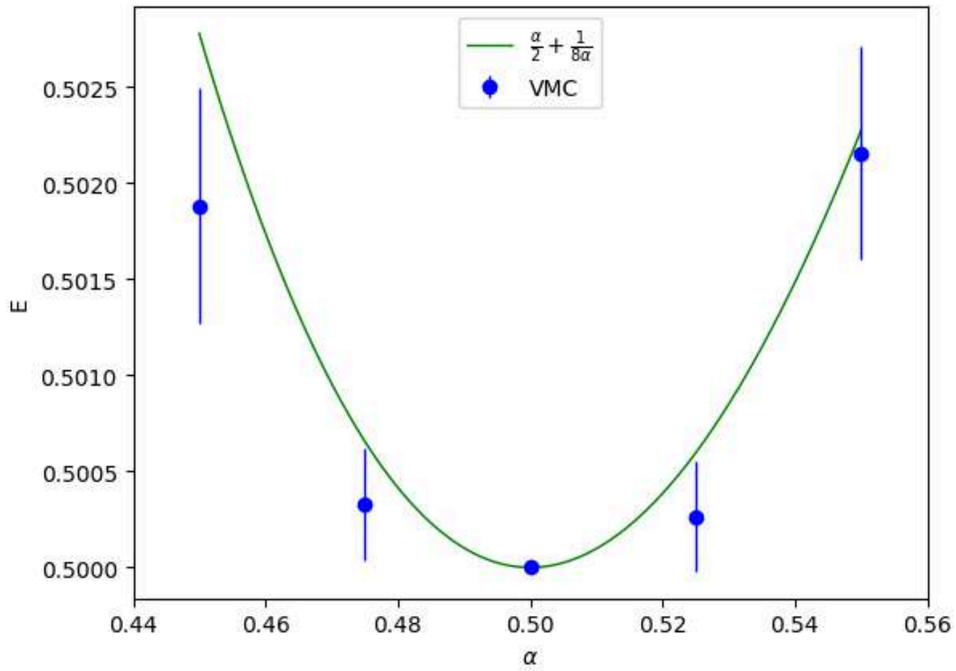
$$\Psi_\alpha(x) = e^{-\alpha x^2}$$

$$E^\alpha(x) = \frac{\alpha}{2} + \frac{1}{8\alpha}$$

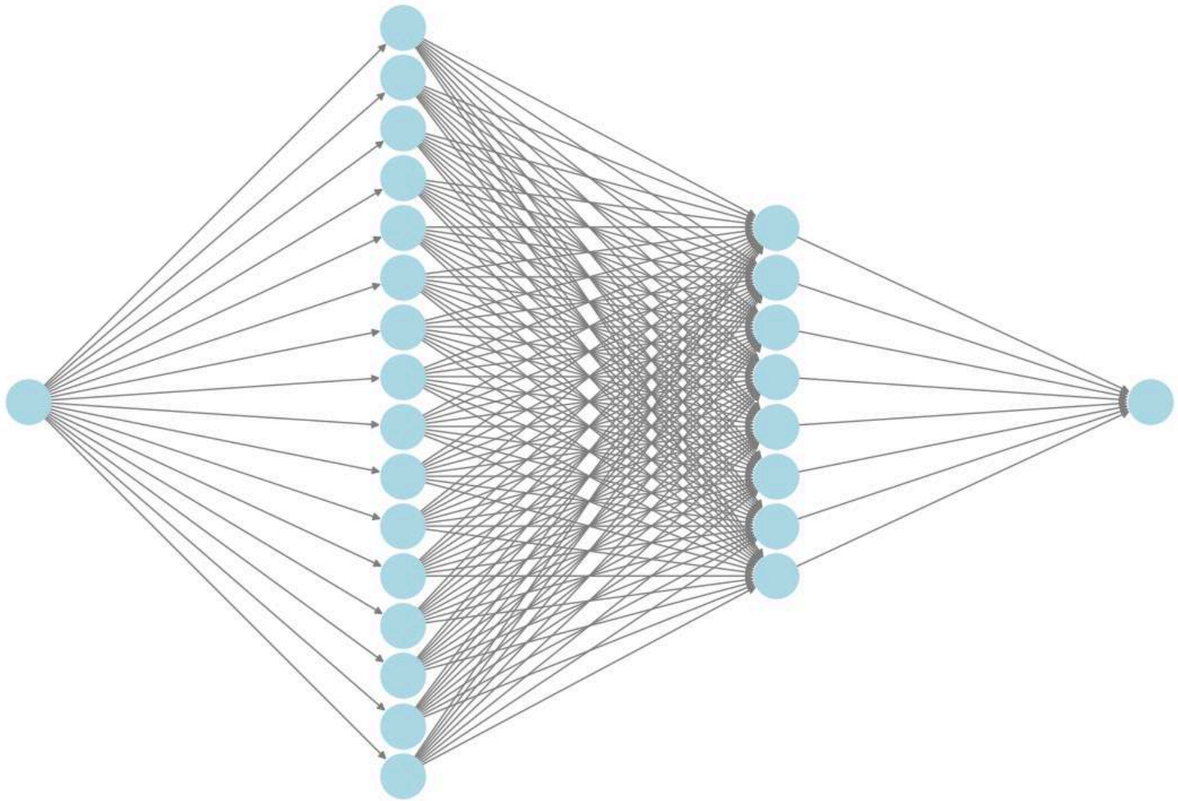
$$\hat{\alpha} = \frac{1}{2}$$



$$\hat{\alpha} = argmin_\alpha E(\alpha)$$



Example: 1D Harmonic Oscillator



Neural Network wavefunction ansatz

Arquitecture: Multilayer perceptrons

Number of layers: 2

Number of parameters: 128

Learning rate: 0.01

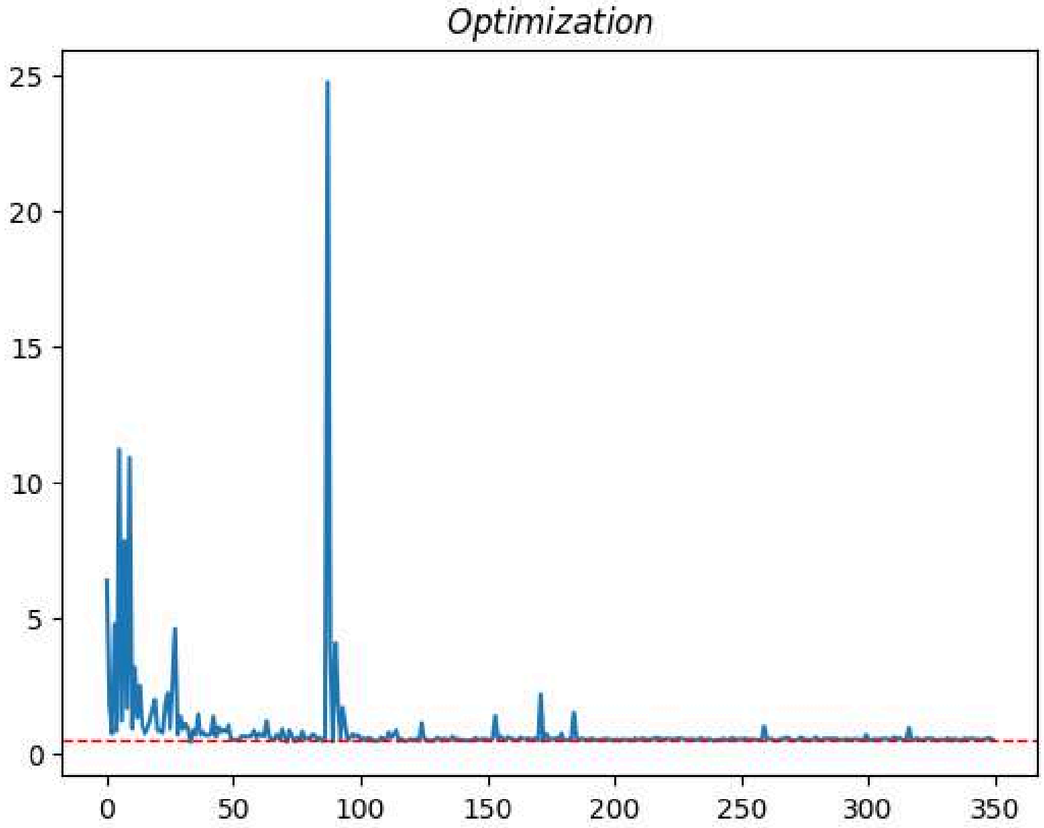
Iterations: 500

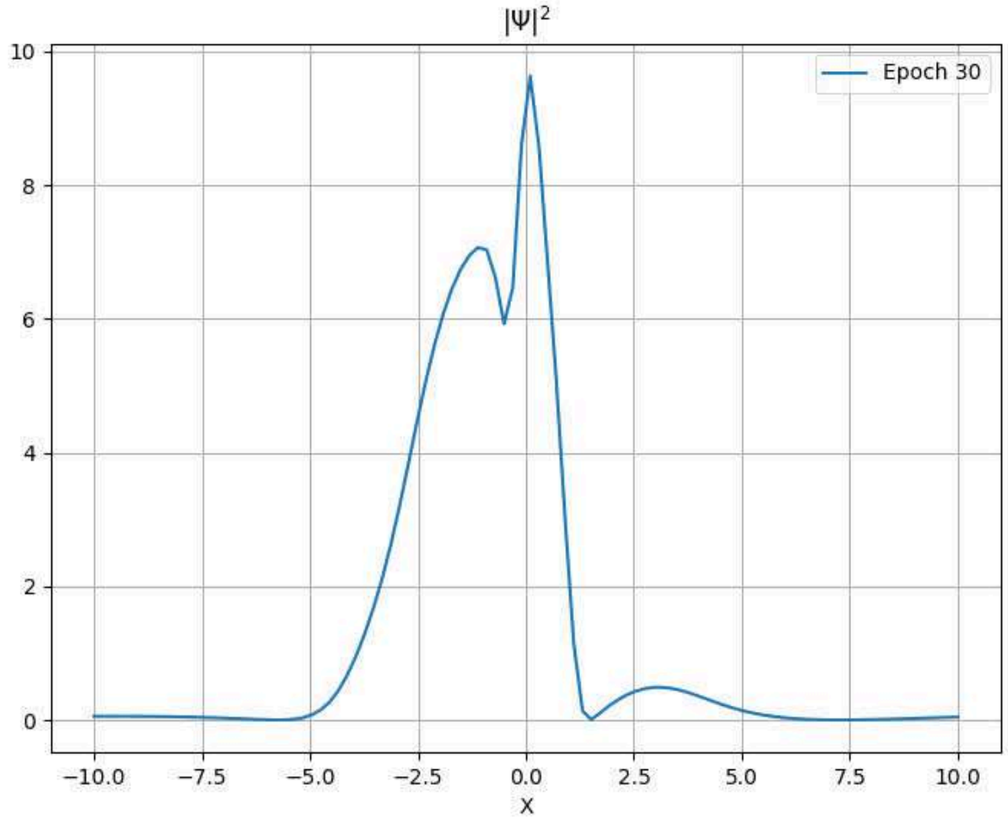
Optimizer: Gradient Descent

Activation function: tanh(x)

Num walkers:100,000

Example: 1D Harmonic Oscillator





Easy implementation

```python
import jax
import jax.numpy as jnp
from jax import grad
from jax import random

# Define a feedforward neural network with two hidden layers and hyperbolic tangent activation.
# The network takes in parameters (weights and biases) and an input vector x.
# Returns the summed output for a single input x.
def neural_network(params, x):
    """
    Feedforward neural network with two hidden layers using hyperbolic tangent activation.

    Parameters:
    - params: Tuple of network parameters (weights and biases).
    - x: Input vector.

    Returns:
    - Summed output from the network.
    """
    w1, b1, w2, b2, w3, b3 = params
    hidden_1 = jnp.tanh(jnp.dot(x, w1) + b1)       # First hidden layer
    hidden_2 = jnp.tanh(jnp.dot(hidden_1, w2) + b2) # Second hidden layer
    output = jnp.dot(hidden_2, w3) + b3             # Output layer
    return jnp.sum(output)
```

Easy implementation

```
# Compute the Laplacian of the neural network output by differentiating twice with respect to position.
gradient_pos = grad(neural_network, argnums=1)
laplacian_pos = grad(gradient_pos, argnums=1)

# Calculate the local energy for a given state, where kinetic energy is computed from the Laplacian.
# Potential energy is defined as the harmonic oscillator potential (1/2 * x^2).
def local_energy(params, x):
    """
    Calculate the local energy for a given configuration of parameters and input.

    Parameters:
    - params: Tuple of network parameters (weights and biases).
    - x: Input vector representing position.

    Returns:
    - Local energy as the sum of kinetic and potential energies.
    """
    kinetic_energy = -0.5 * laplacian_pos(params, x)
    potential_energy = 0.5 * (x**2)  # Harmonic oscillator potential
    return kinetic_energy + potential_energy
```

Example: 1D Bosonic System

Bosons confined in a harmonic trap with short and long-range interaction

$$\hat{H} = \sum_{i=1}^{N} \left(-\frac{1}{2m} \frac{\partial^2}{\partial x_i^2} + \frac{1}{2} m\omega^2 x_i^2 \right) + \sum_{i<j}^{N} \left(g\delta(x_i - x_j) + \sigma|x_i - x_j| \right)$$

Neural Network Solutions of Bosonic Quantum Systems in One Dimension

Paulo F. Bedaque,[1,*] Hersh Kumar,[1,†] and Andy Sheng[1,‡]

[1] *Department of Physics, University of Maryland, College Park, MD 20742*

Neural networks have been proposed as efficient numerical wavefunction ansatze which can be used to variationally search a wide range of functional forms for ground state solutions. These neural network methods are also advantageous in that more variational parameters and system degrees of freedom can be easily added. We benchmark the methodology by using neural networks to study several different integrable bosonic quantum systems in one dimension and compare our results to the exact solutions. While testing the scalability of the procedure to systems with many particles, we also introduce using symmetric function inputs to the neural network to enforce exchange symmetries of indistinguishable particles.

Example: 1D Bosonic System

We need to force symmetry to the neural network

$$\psi(\ldots, x_i, \ldots, x_j, \ldots) = \psi(\ldots, x_j, \ldots, x_i, \ldots)$$

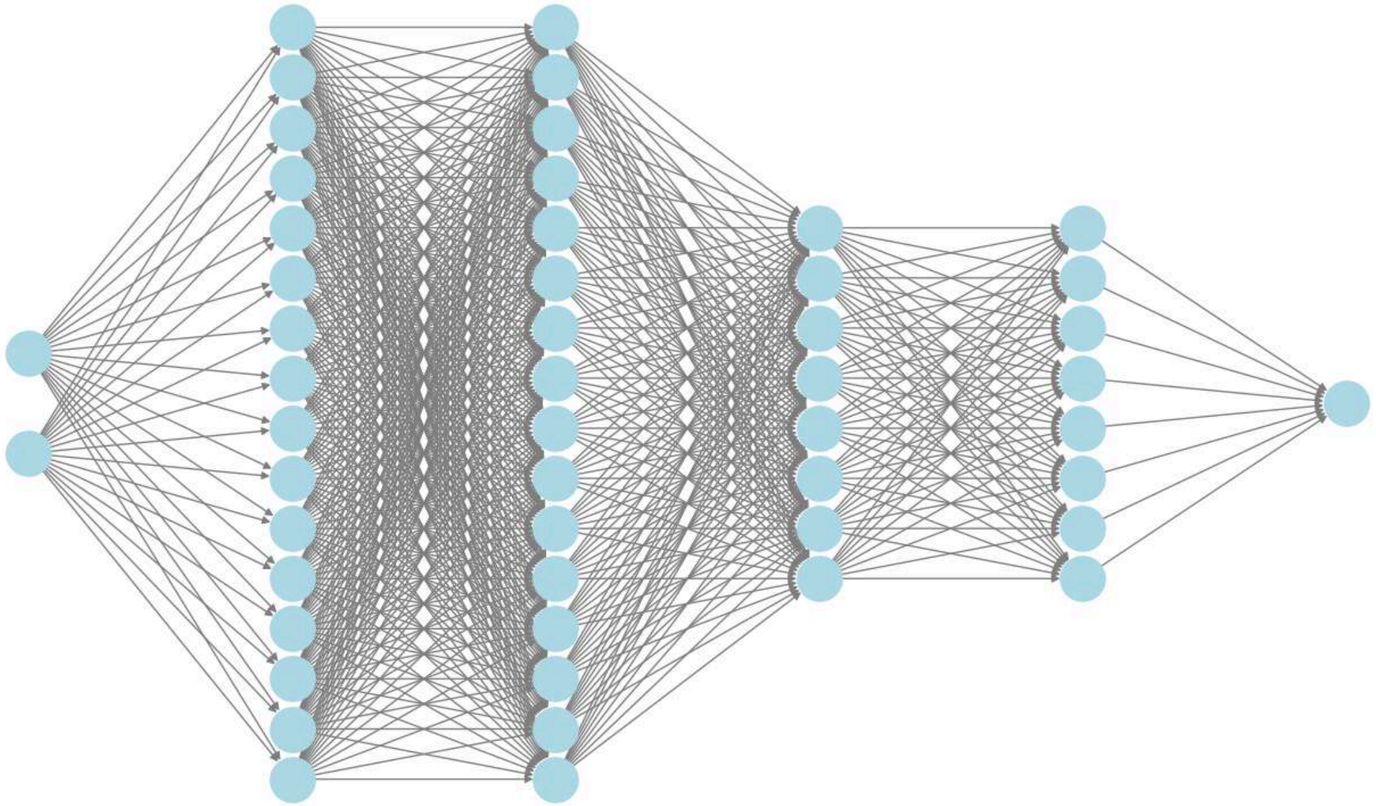
We use the Girard-Newton identities: $u_k = \sum_{i=1}^{N} x_i^k$

$$\{x_k\} \rightarrow \{u_k\}$$

$$\psi(x_1, \ldots, x_N) \approx \mathcal{F}(u_1, \ldots, u_N)$$

We look for bound state: $\psi(x_1, \ldots, x_N) = \mathcal{F}(u_1, \ldots, u_N) \cdot e^{-\Omega \sum_{i=1}^{N} x_i^2}$

Example: 1D Bosonic System



Neural Network wavefunction ansatz

Arquitecture: Multilayer perceptrons

Number of layers: 4

Number of parameters: 537

Learning rate: 0.01

Iterations: 1000

Optimizer: Gradient Descent

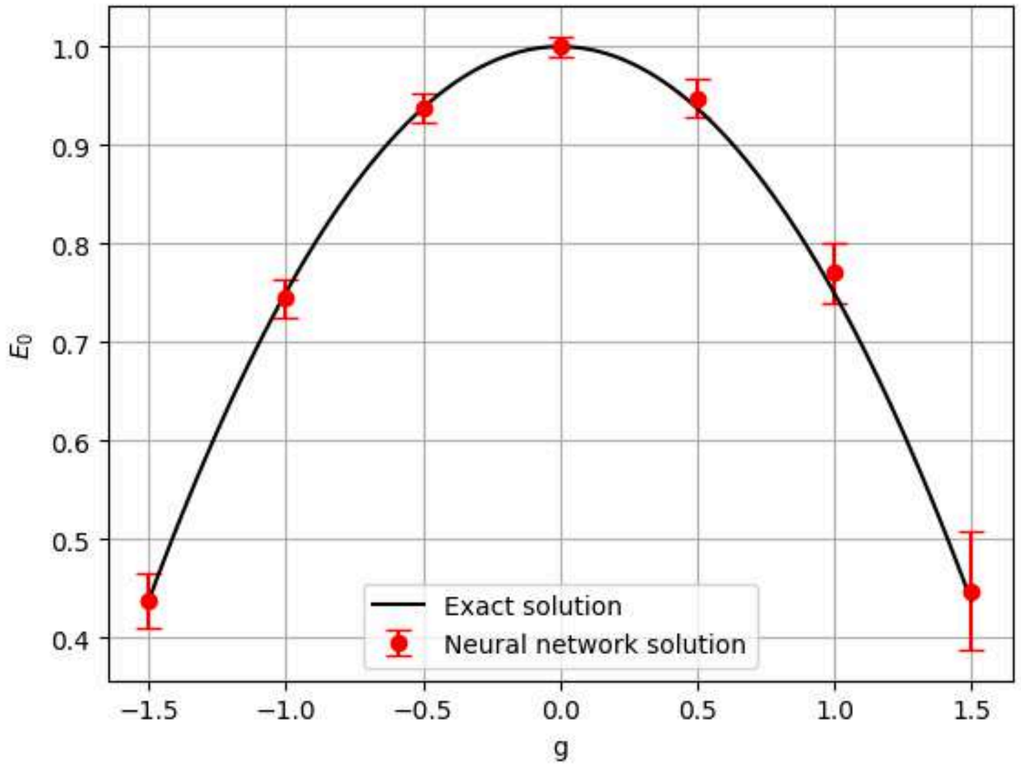
Activation function:

$$\mathrm{CELU}(x) = \begin{cases} x & \text{si } x > 0 \\ e^x - 1 & \text{si } x \leq 0 \end{cases}$$

Num walkers:800,000

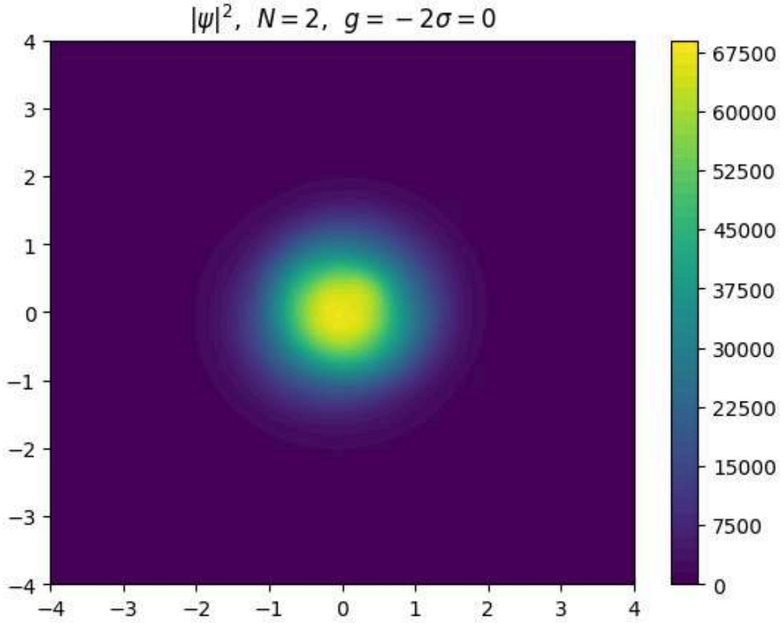
Example: 1D Bosonic System

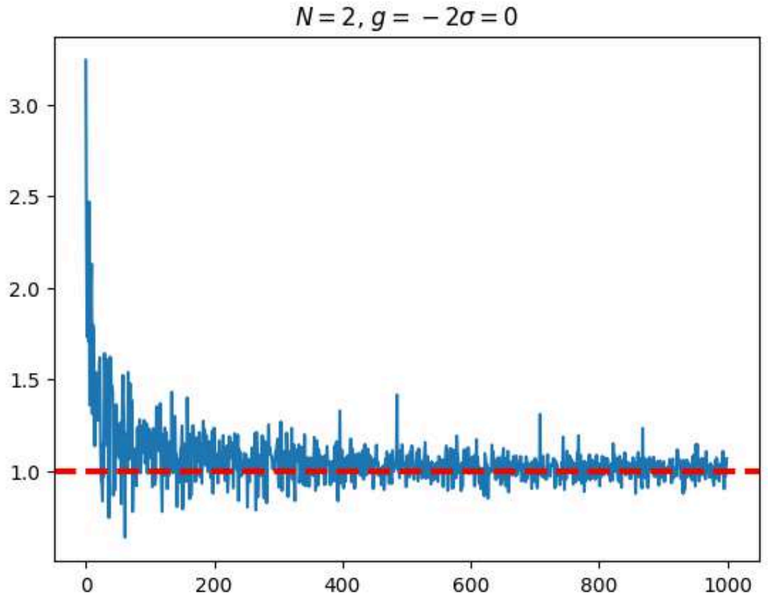
The Hamiltonian has an analytical solution in the regime of $\sigma = -mwg/2$

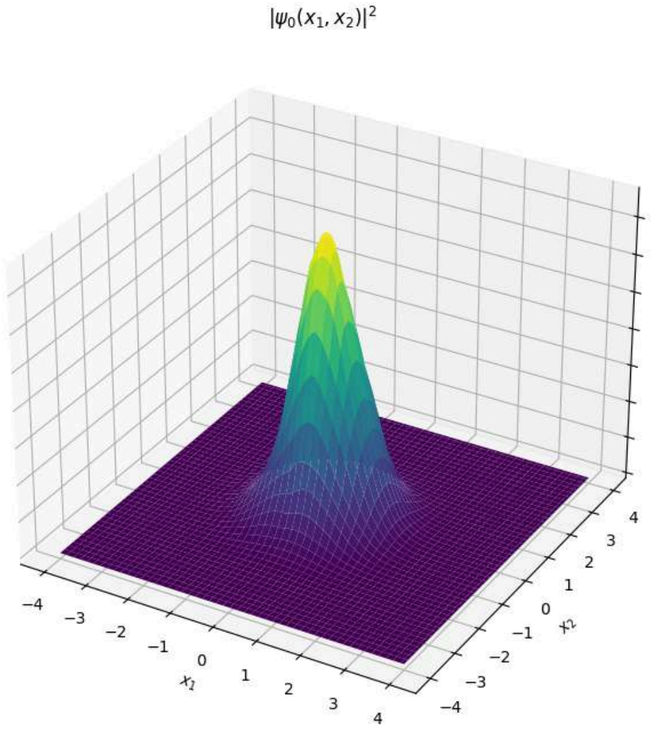


$$E_0 = \frac{N\omega}{2} - mg^2 \frac{N(N^2-1)}{24}$$

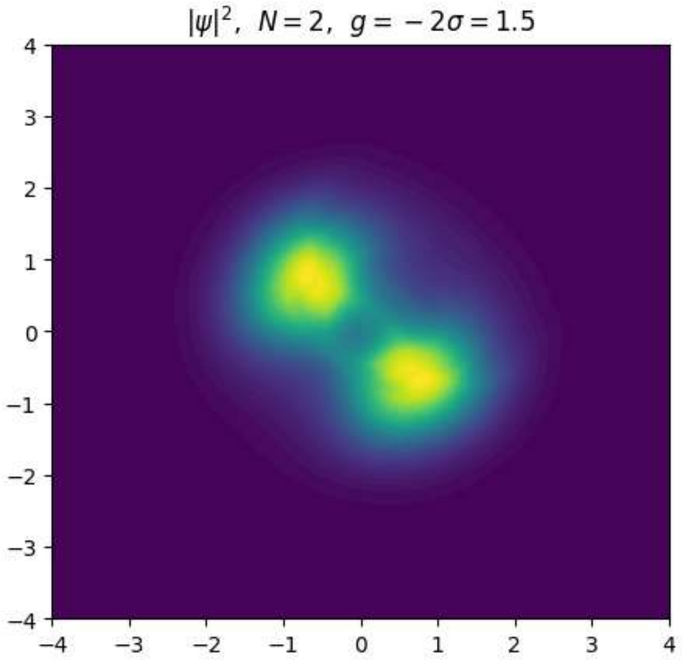
Example: 1D Bosonic System

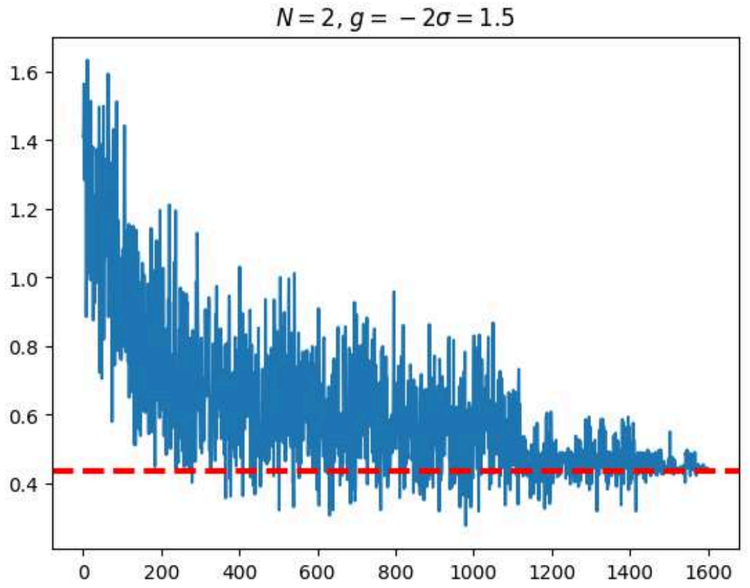


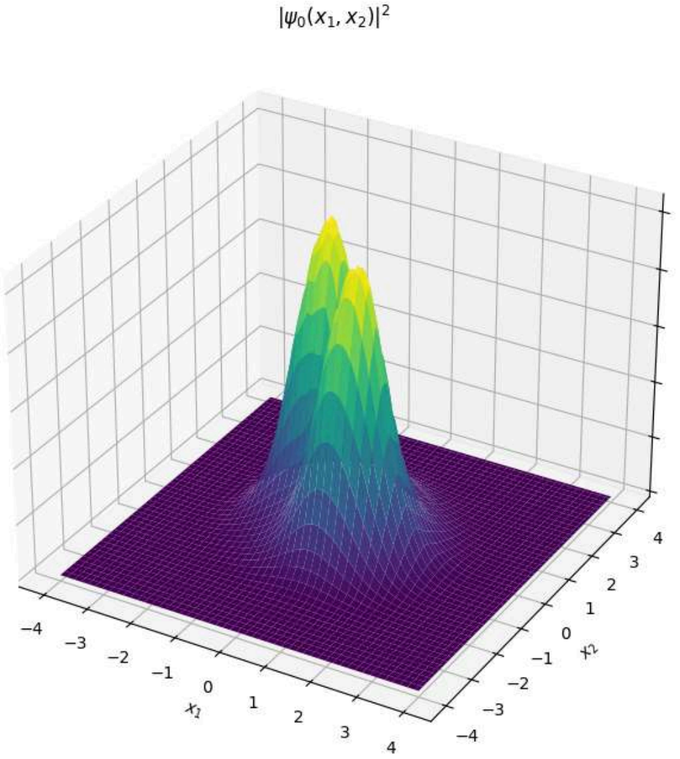




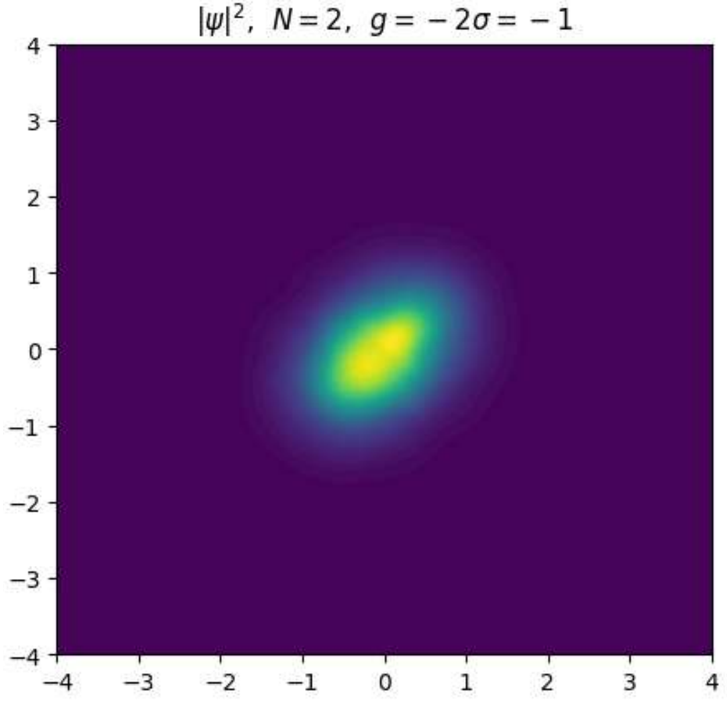
Example: 1D Bosonic System

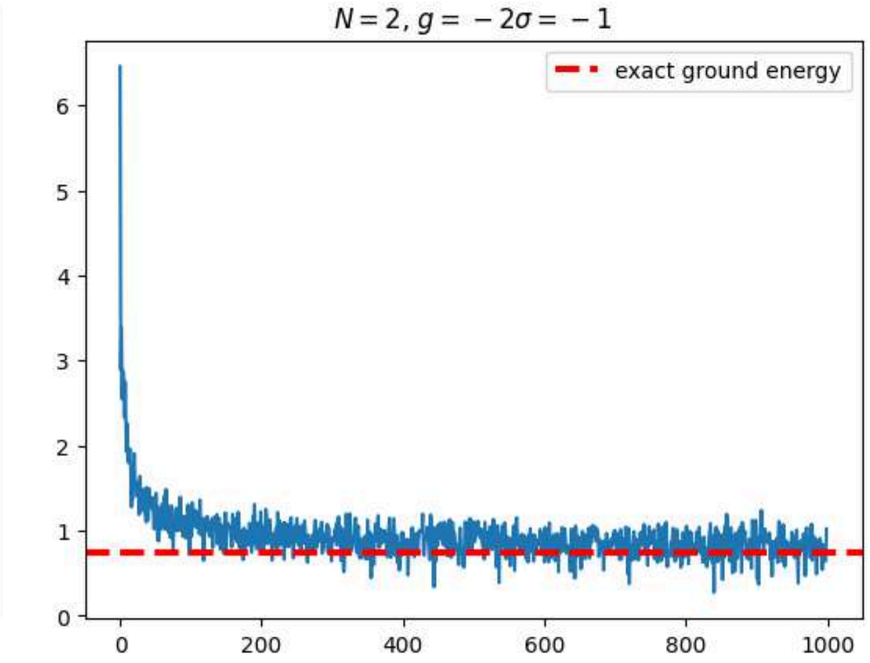


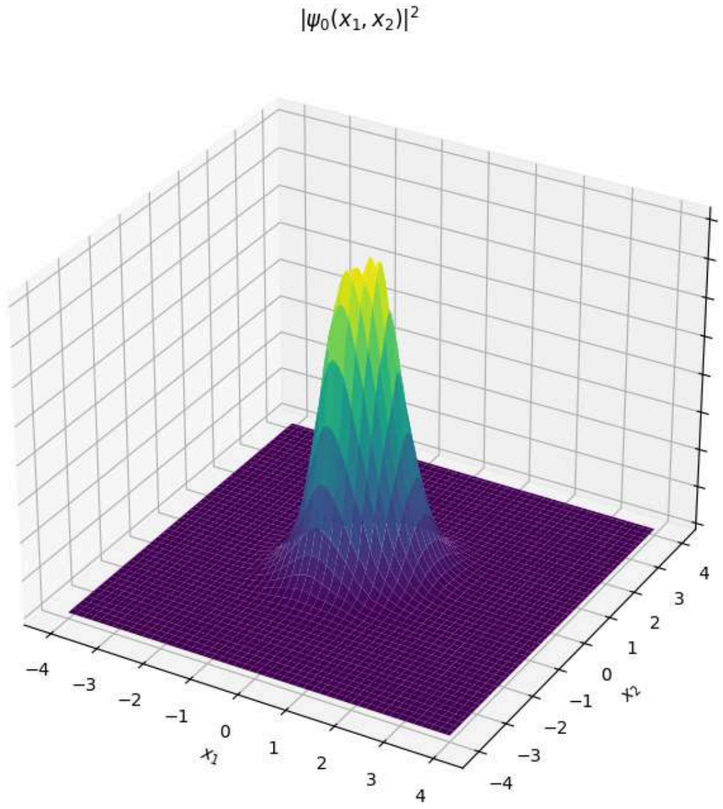




Example: 1D Bosonic System







Conclusions

- NN models provide a robust framework to enhance the VMC method for solving many-body quantum systems.
- A key advantage is its ability to explore a broad space of potential ground state wavefunctions, unrestricted by specific functional forms.
- This flexibility enables a more comprehensive search of the solution space for ground state wavefunctions.
- Neural networks offer convenient scalability, allowing for an easy increase in variational parameters and particle numbers in the system.

Thank you for your time