Title: Lecture - Machine Learning, PHYS 777

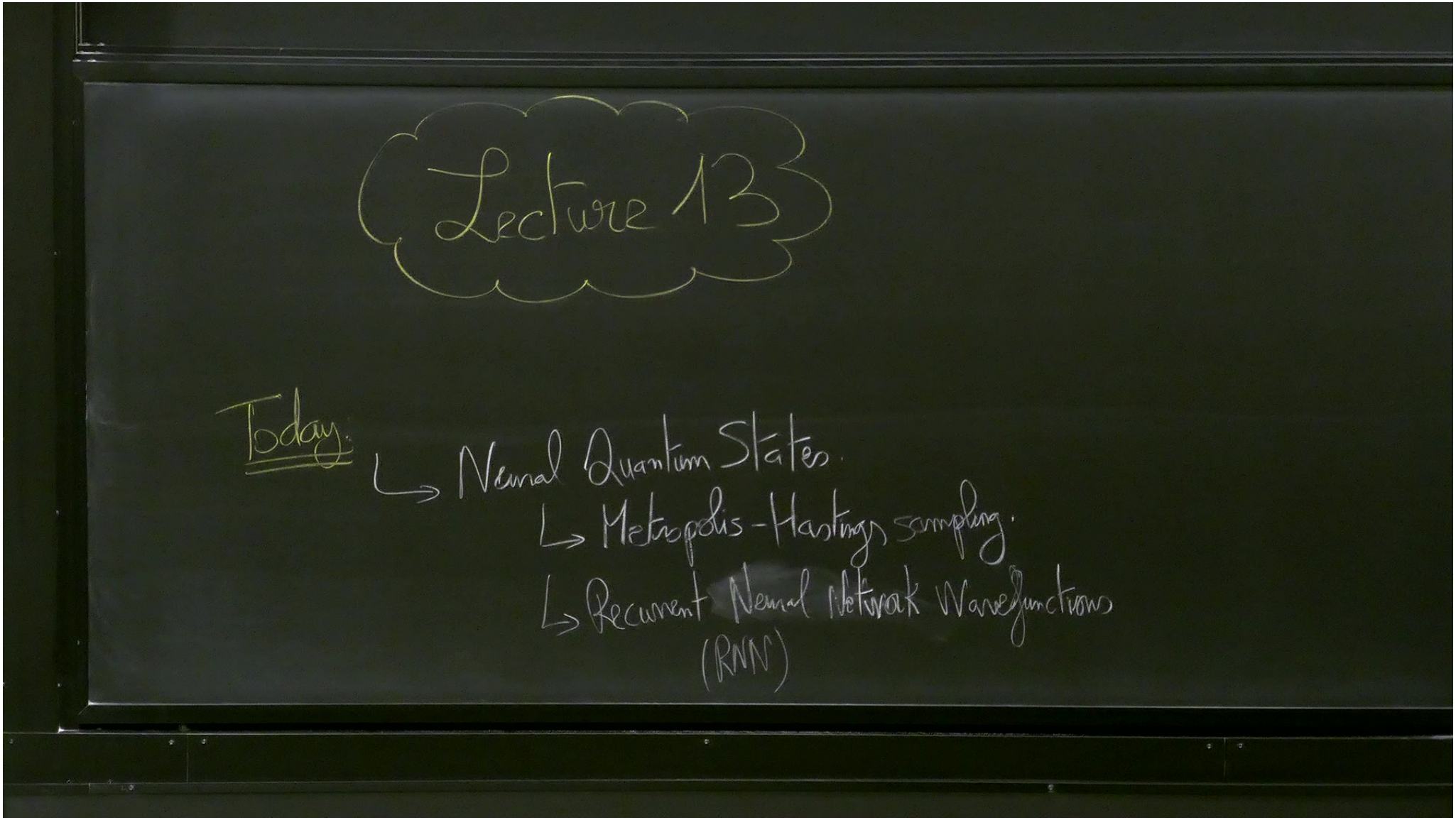
Speakers: Mohamed Hibat Allah

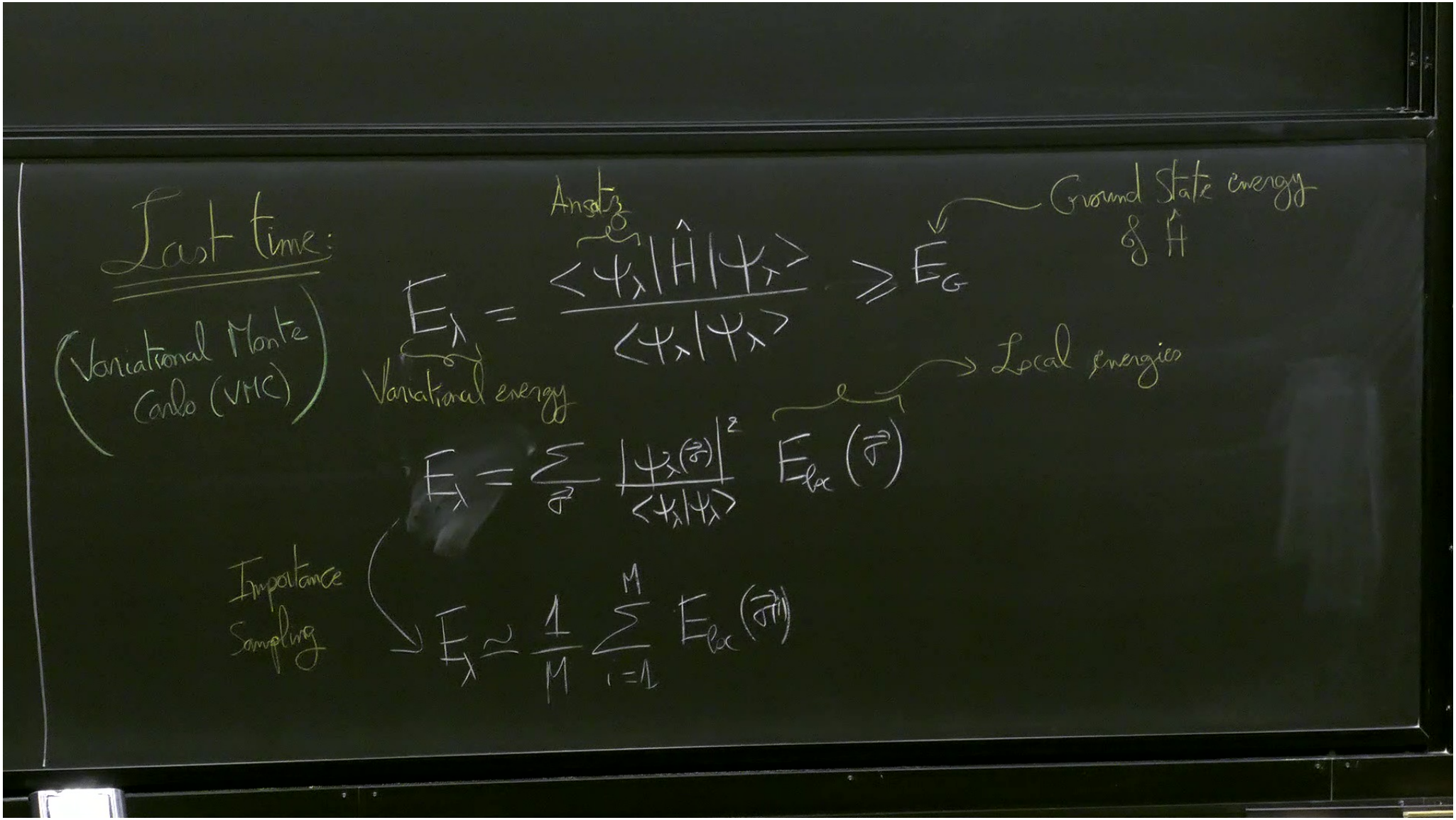
Collection/Series: Machine Learning (Elective), PHYS 777, February 24 - March 28, 2025

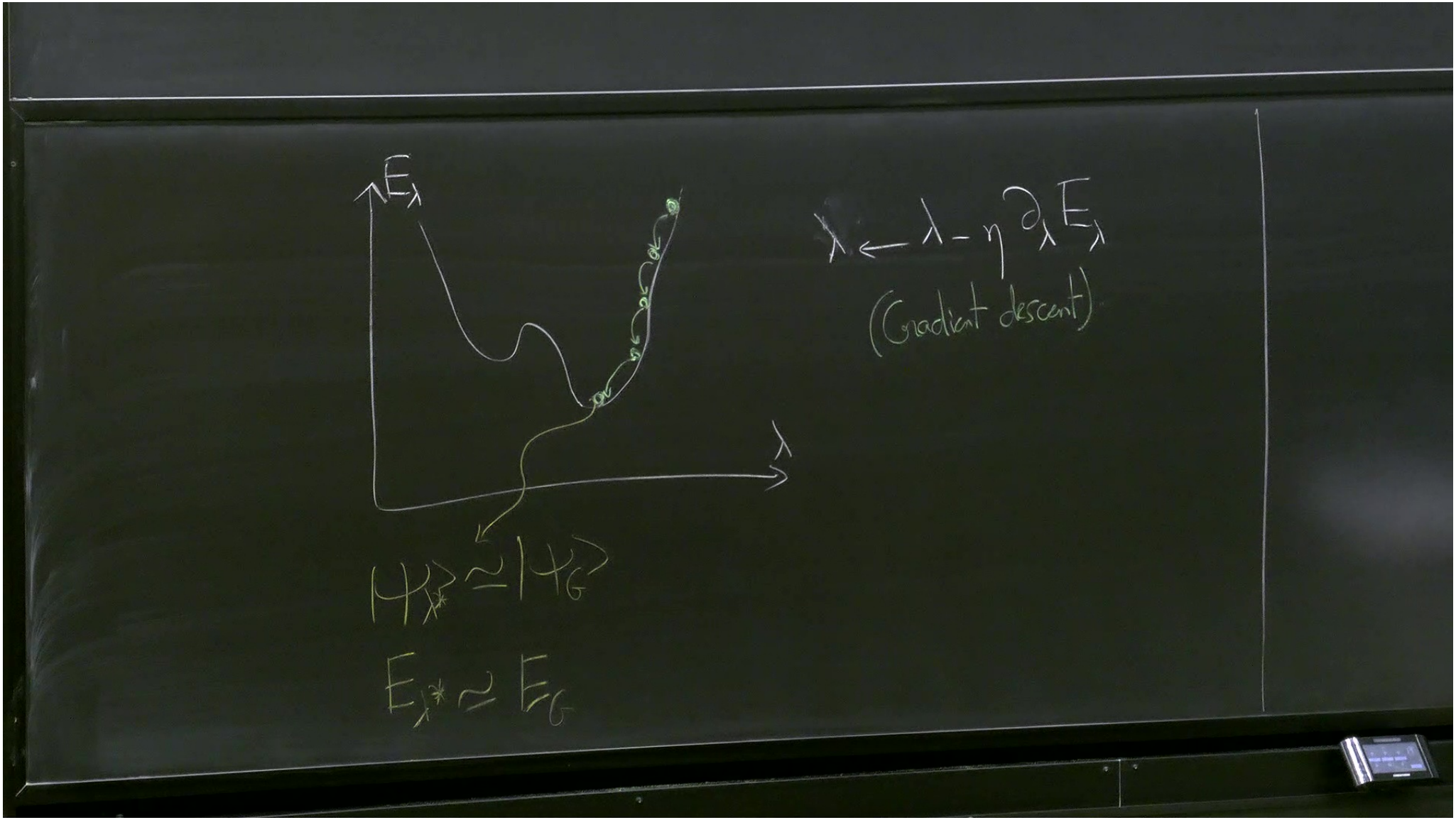
Subject: Condensed Matter, Other

Date: March 28, 2025 - 9:00 AM

URL: https://pirsa.org/25030044


Lecture 13
Today:
↳ Neural Quantum States.
↳ Metropolis-Hastings sampling.
↳ Recurrent Neural Network Wavefunctions
(RNN)

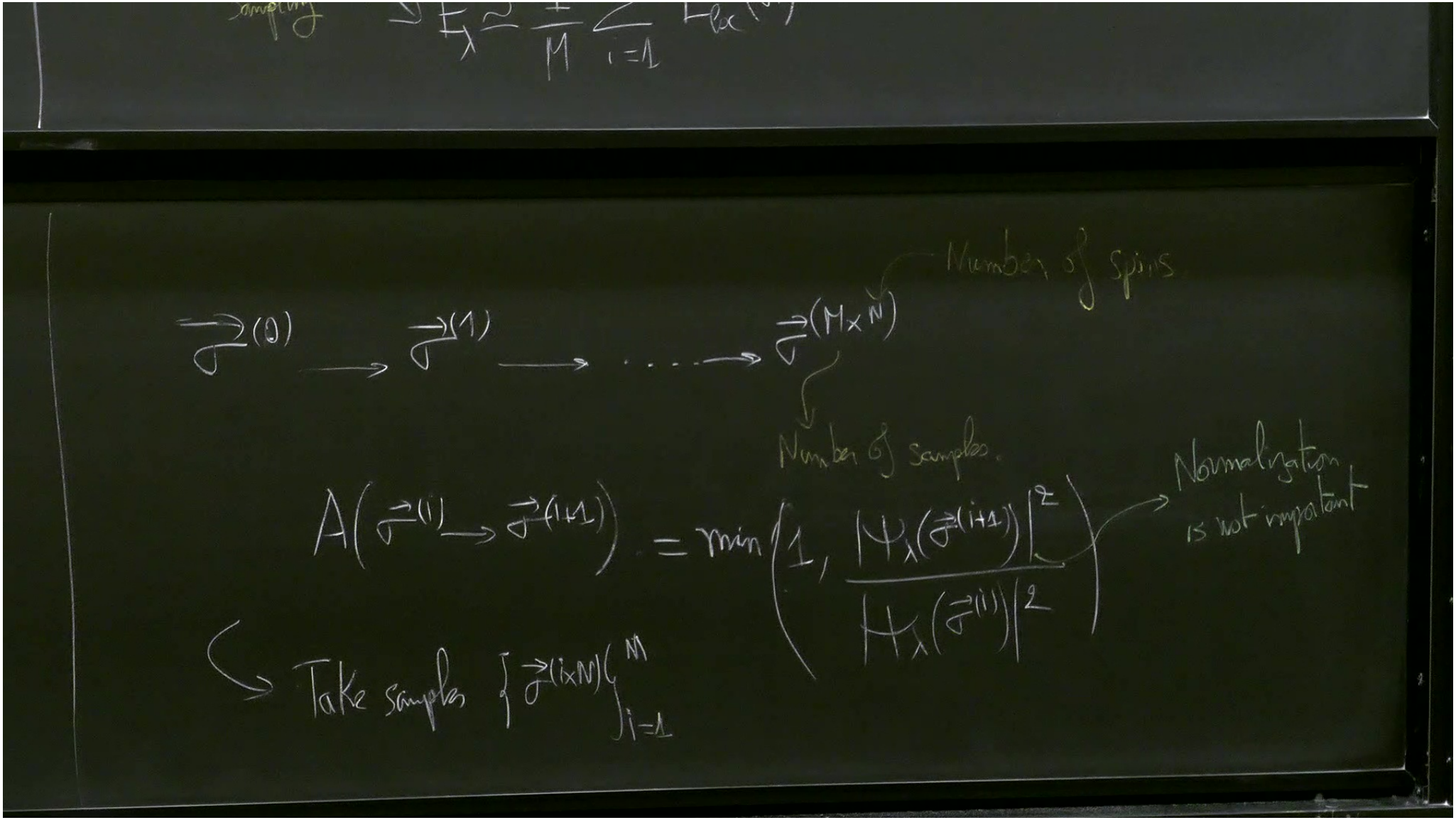

Last time:
(Variational Monte Carlo (VMC))
Ansatz
$E_\lambda = \frac{\langle \psi_\lambda | \hat{H} | \psi_\lambda \rangle}{\langle \psi_\lambda | \psi_\lambda \rangle} \geq E_G$
Ground State energy of $\hat{H}$
Variational energy
$E_\lambda = \sum_{\vec{\sigma}} \frac{|\psi_\lambda(\vec{\sigma})|^2}{\langle \psi_\lambda | \psi_\lambda \rangle} E_{loc}(\vec{\sigma})$
Local energies
Importance Sampling
$E_\lambda \simeq \frac{1}{M} \sum_{i=1}^{M} E_{loc}(\vec{\sigma}^i)$

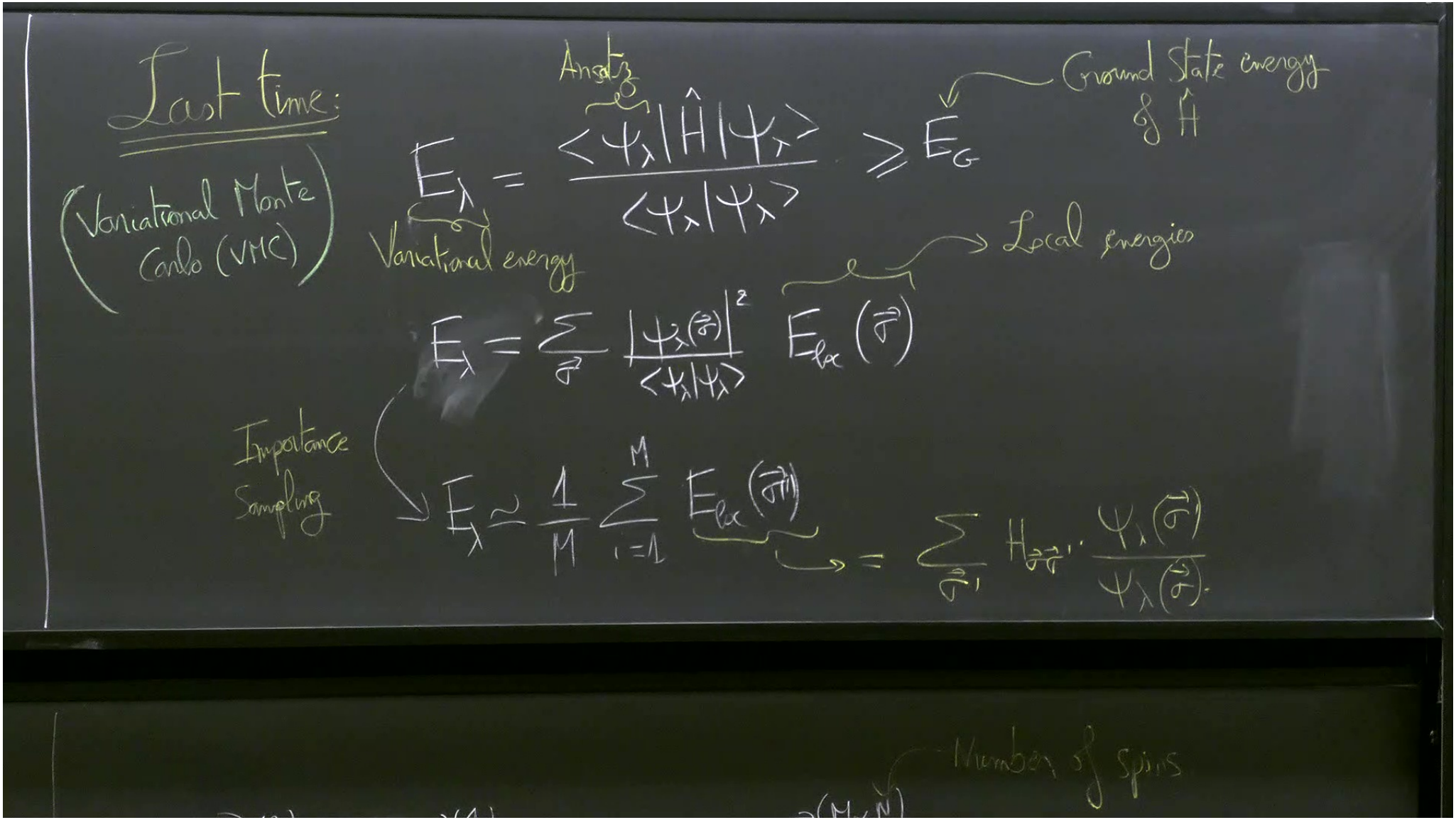

E_λ
λ
$\lambda \leftarrow \lambda - \eta \partial_\lambda E_\lambda$
(Gradient descent)
$|\psi_{\lambda^*}\rangle \simeq |\psi_G\rangle$
$E_{\lambda^*} \simeq E_G$

(RNN)

Metropolis-Hastings sampling:

$$\vec{r} \sim \frac{|\Psi_\lambda(\vec{r})|^2}{\langle \Psi_\lambda | \Psi_\lambda \rangle}$$


sampling
$E_\lambda \simeq \frac{1}{M} \sum_{i=1}$
$\vec{\sigma}^{(0)} \longrightarrow \vec{\sigma}^{(1)} \longrightarrow \cdots \longrightarrow \vec{\sigma}^{(M \times N)}$
Number of spins
Number of samples.
$A(\vec{\sigma}^{(i)} \to \vec{\sigma}^{(i+1)}) = \min\left(1, \frac{|\Psi_\lambda(\vec{\sigma}^{(i+1)})|^2}{|\Psi_\lambda(\vec{\sigma}^{(i)})|^2}\right)$
Normalization is not important
Take samples $\{\vec{\sigma}^{(i \times N)}\}_{i=1}^{M}$


Last time:
(Variational Monte Carlo (VMC))
Ansatz
Ground State energy of $\hat{H}$
$E_\lambda = \frac{\langle \psi_\lambda | \hat{H} | \psi_\lambda \rangle}{\langle \psi_\lambda | \psi_\lambda \rangle} \geq E_G$
Variational energy
Local energies
$E_\lambda = \sum_{\vec{\sigma}} \frac{|\psi_\lambda(\vec{\sigma})|^2}{\langle \psi_\lambda | \psi_\lambda \rangle} E_{loc}(\vec{\sigma})$
Importance Sampling
$E_\lambda \simeq \frac{1}{M} \sum_{i=1}^{M} E_{loc}(\vec{\sigma_i})$
$= \sum_{\vec{\sigma}'} H_{\vec{\sigma},\vec{\sigma}'} \cdot \frac{\psi_\lambda(\vec{\sigma}')}{\psi_\lambda(\vec{\sigma})}$
Number of spins

Firefox File Edit View History Bookmarks Tools Window Help Fri Mar 28 9:11 AM

Hydrogen_Atom_Demo | Ground-State: Heisenberg mod | gs-heisenberg.ipynb - Colab

localhost:8888/notebooks/Hydrogen_Atom_Demo.ipynb 150%

jupyter Hydrogen_Atom_Demo Last Checkpoint: 2 years ago

File Edit View Run Kernel Settings Help Trusted

Markdown JupyterLab Python 3 (ipykernel)

Based on Giuseppe Carleo Tutorials. Edited by Mohamed Hibat-Allah.

[4]:
```
from optimization import *
```

Hamiltonian of the Hydrogen *Atom*

$$\hat{H} = -\frac{1}{2}\frac{\partial^2}{\partial^2 r} + \frac{l(l+1)}{2r^2} - \frac{1}{r}$$

We consider $n = 0$ (principal quantum number), $l = 0$ (angular momentum) to get the ground state. Physical constants are absorbed in `r`. ¶

- Ground state energy: $E_G = -\frac{1}{2}$

- Ground state: $\psi_G(r) = r\exp(-r)$

Firefox File Edit View History Bookmarks Tools Window Help — Fri Mar 28 9:12 AM

Hydrogen_Atom_Demo | Ground-State: Heisenberg mod | gs-heisenberg.ipynb - Colab

localhost:8888/notebooks/Hydrogen_Atom_Demo.ipynb 150%

jupyter Hydrogen_Atom_Demo Last Checkpoint: 2 years ago

File Edit View Run Kernel Settings Help — Trusted

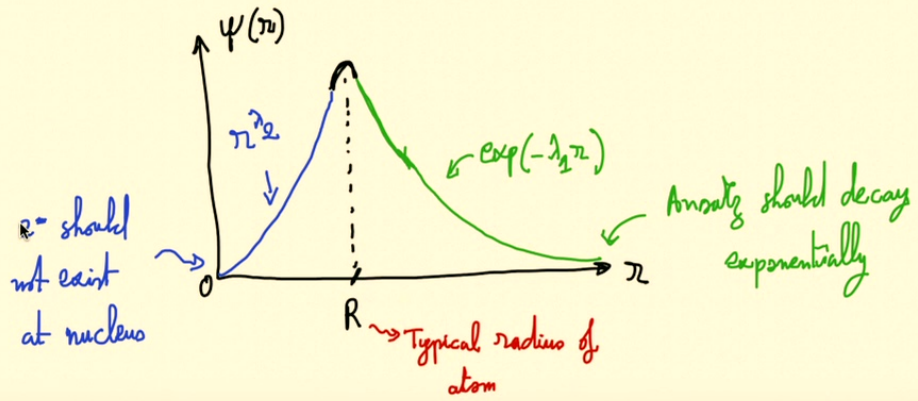
Markdown — JupyterLab — Python 3 (ipykernel)

We consider $n = 0$ (principal quantum number), $l = 0$ (angular momentum) to get the ground state. Physical constants are absorbed in `r`.

- Ground state energy: $E_G = -\frac{1}{2}$

- Ground state: $\psi_G(r) = r\exp(-r)$

1. Define the variational wave function (ansatz)



Firefox File Edit View History Bookmarks Tools Window Help — Fri Mar 28 9:14 AM

Hydrogen_Atom_Demo | Ground-State: Heisenberg mod | gs-heisenberg.ipynb - Colab

localhost:8888/notebooks/Hydrogen_Atom_Demo.ipynb

jupyter Hydrogen_Atom_Demo Last Checkpoint: 2 years ago

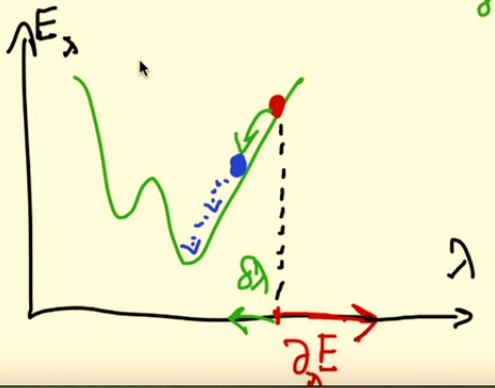
File Edit View Run Kernel Settings Help — Trusted

Markdown — JupyterLab — Python 3 (ipykernel)

[5]:
```
psi_ansatz=State([8.1, 6.2]) #Define the variational wavefunction
```

2. Running the optimization with gradient descent

$$\min_{\lambda} E_\lambda \rightsquigarrow \lambda \longrightarrow \lambda + \underbrace{\left(-\varepsilon\, \partial_\lambda E_\lambda\right)}_{\delta\lambda}$$



Firefox File Edit View History Bookmarks Tools Window Help — Fri Mar 28 9:14 AM

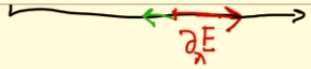
Hydrogen_Atom_Demo | Ground-State: Heisenberg mod | gs-heisenberg.ipynb - Colab

localhost:8888/notebooks/Hydrogen_Atom_Demo.ipynb

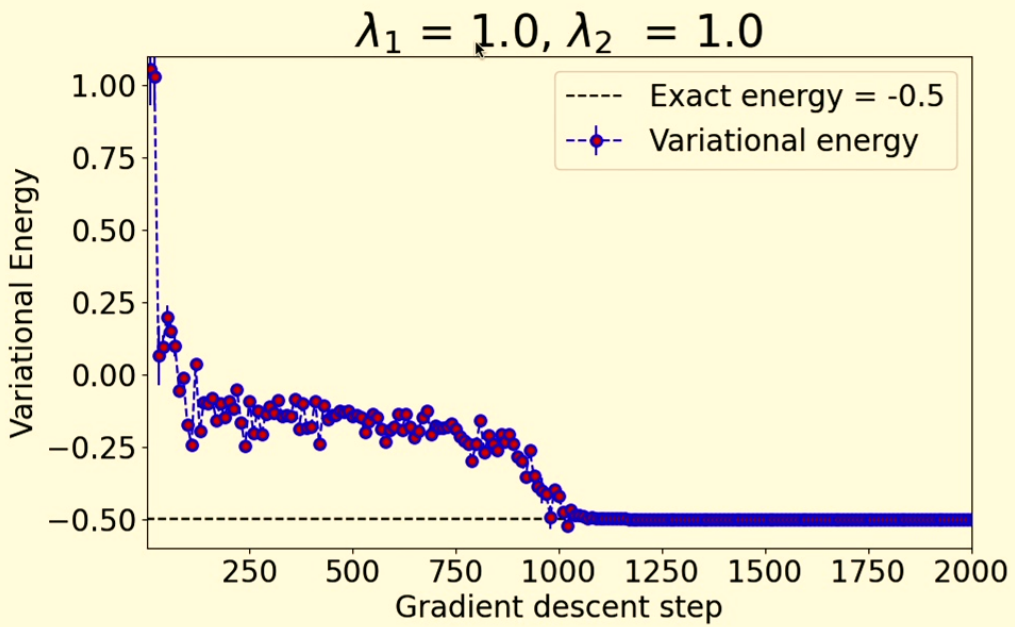
jupyter Hydrogen_Atom_Demo Last Checkpoint: 2 years ago

File Edit View Run Kernel Settings Help — Trusted

Code — JupyterLab — Python 3 (ipykernel)



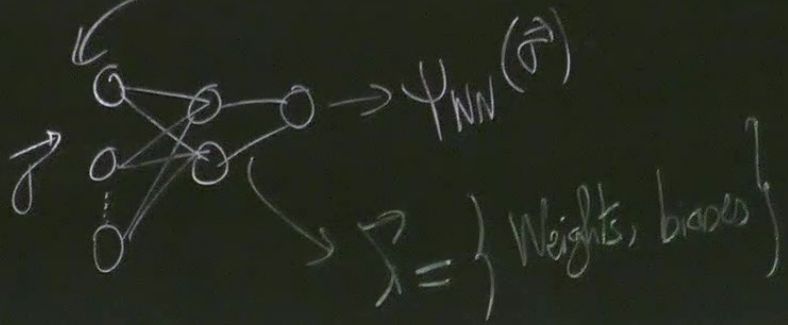
[9]:
```
run_optimization(psi_ansatz, epsilon=0.4)
```

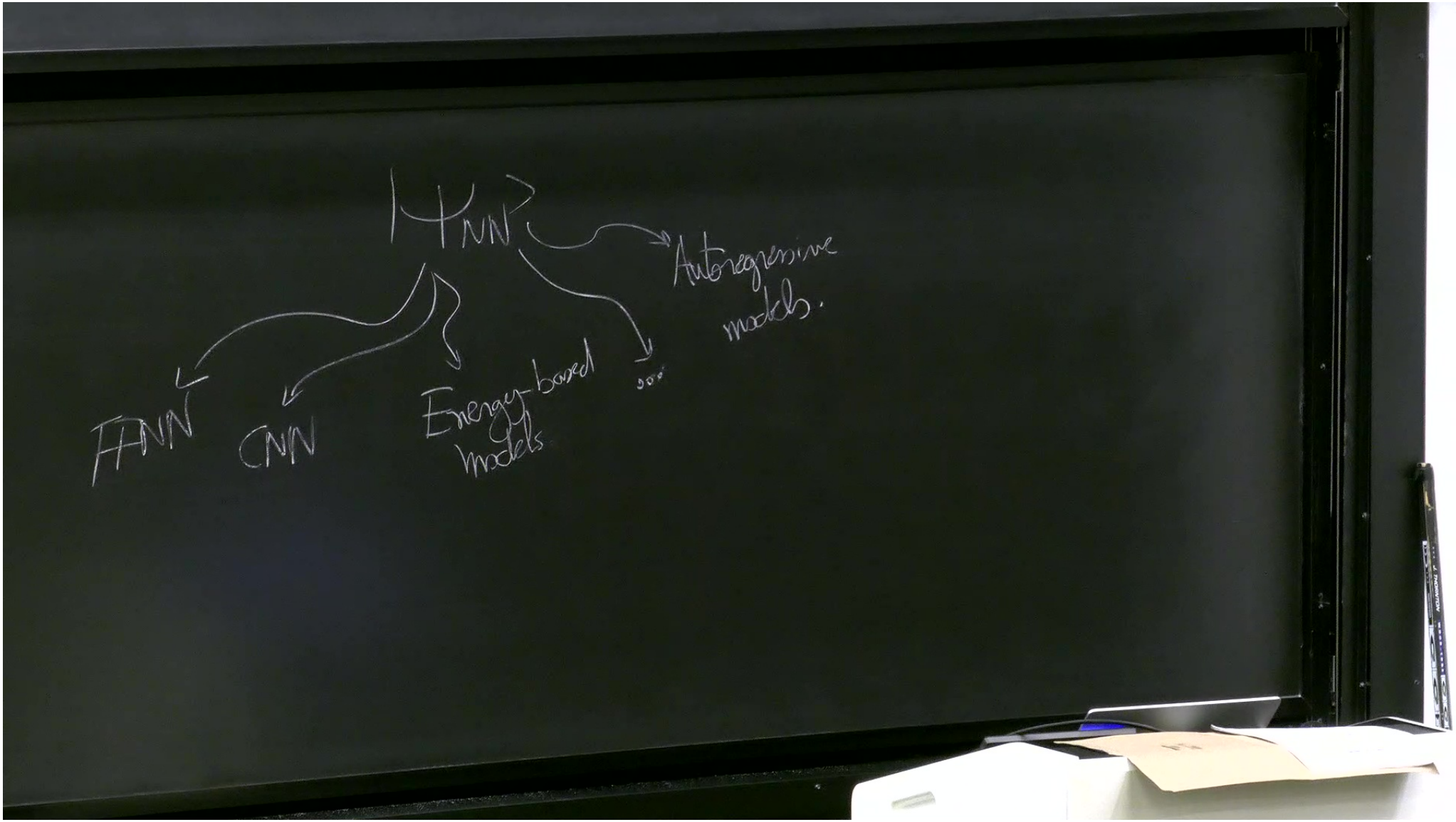



```
step 2000
lambda1  =  1.0000000000000004 , lambda2  =  1.0000000000000004
energy =  -0.5
```

Neural Quantum States:

$$|\Psi_{NN}\rangle = \sum_{\vec{\sigma}} \Psi_{NN}(\vec{\sigma}) |\vec{\sigma}\rangle$$




NN
FFNN
CNN
Energy-based models
...
Autoregressive models.

Variational principle is stat mechanics

$$F_\rho = \langle \hat{H} \rangle_\rho - T S_\rho$$

$$\langle \hat{H} \rangle_\rho = \sum_{\vec{\sigma}} \rho(\vec{\sigma}) E(\vec{\sigma})$$

$$S_\rho = \sum_{\vec{\sigma}} -\rho(\vec{\sigma}) \log(\rho(\vec{\sigma})) = \langle -\log(\rho(\vec{\sigma})) \rangle_\rho$$

Take samples $\{\vec{\sigma}^{(i\times N)}\}_{i=1}^{M}$

$\forall \rho: \quad F_\rho \geq F_{P_T}$ → Boltzmann prob.

Use $\rho = P_\lambda$

$$P_T(\vec{\sigma}) = \frac{\exp(-\beta E(\vec{\sigma}))}{Z}$$

$$F_\lambda = \langle H \rangle_{P_\lambda} + T \langle \log(P_\lambda(\vec{\sigma})) \rangle_{P_\lambda(\vec{\sigma})}$$

F_λ: Variational free energy (Cost function).

$\simeq P_T$ (mean convergence).

$Z = \sum_{\vec{\sigma}} \exp(-\beta E(\vec{\sigma}))$

qN terms $(\sigma_1, \sigma_2, \ldots, \sigma_N)$

Firefox File Edit View History Bookmarks Tools Window Help — Fri Mar 28 9:29 AM

Hydrogen_Atom_Demo | Ground-State: Heisenberg mod | gs-heisenberg.ipynb - Colab

https://colab.research.google.com/drive/1t7388MOTclWilSJqx7h4klxXJ1PBLEln#scrollTo=DUsOlOjHBMxU&uniqifier=1

gs-heisenberg.ipynb

File Edit View Insert Runtime Tools Help

Share | Gemini

Commands | + Code | + Text | Reconnect T4

Ground-State: Heisenberg model

Author: Giuseppe Carleo and Filippo Vicentini (EPFL-CQSL)

The goal of this tutorial is to review various neural network architectures available in NetKet, in order to learn the ground-state of a paradigmatic spin model: the spin-$1/2$ Heisenberg antiferromagnetic chain.

The Hamiltonian we will consider for this tutorial is the following

$$H = \sum_{i=1}^{L} \vec{\sigma}_i \cdot \vec{\sigma}_{i+1}.$$

L is the length of the chain, and we will use both open and periodic boundary conditions. $\vec{\sigma} = (\sigma^x, \sigma^y, \sigma^z)$ denotes the vector of Pauli matrices. Please note that there is a factor of 2 between Pauli-matrices and spin-1/2 operators (thus a factor of 4 in H).

We will consider in this tutorial 5 possible ways of determining the ground-state of this model.

0. Installing Netket

If you are executing this notebook on Colab, you will need to install netket. You can do so by running the following cell:

```
[ ]  1 %pip install --quiet netket
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 695.1/695.1 kB 14.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 177.1/177.1 kB 10.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.1/3.1 MB 58.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 kB 1.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 17.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.2/55.2 kB 3.1 MB/s eta 0:00:00
```

✓ 1m 18s completed at 10:56 PM

Ground-State: Heisenberg model

Author: Giuseppe Carleo and Filippo Vicentini (EPFL-CQSL)

The goal of this tutorial is to review various neural network architectures available in NetKet, in order to learn the ground-state of a paradigmatic spin model: the spin-$1/2$ Heisenberg antiferromagnetic chain.

The Hamiltonian we will consider for this tutorial is the following

$$H = \sum_{i=1}^{L} \vec{\sigma}_i \cdot \vec{\sigma}_{i+1}$$

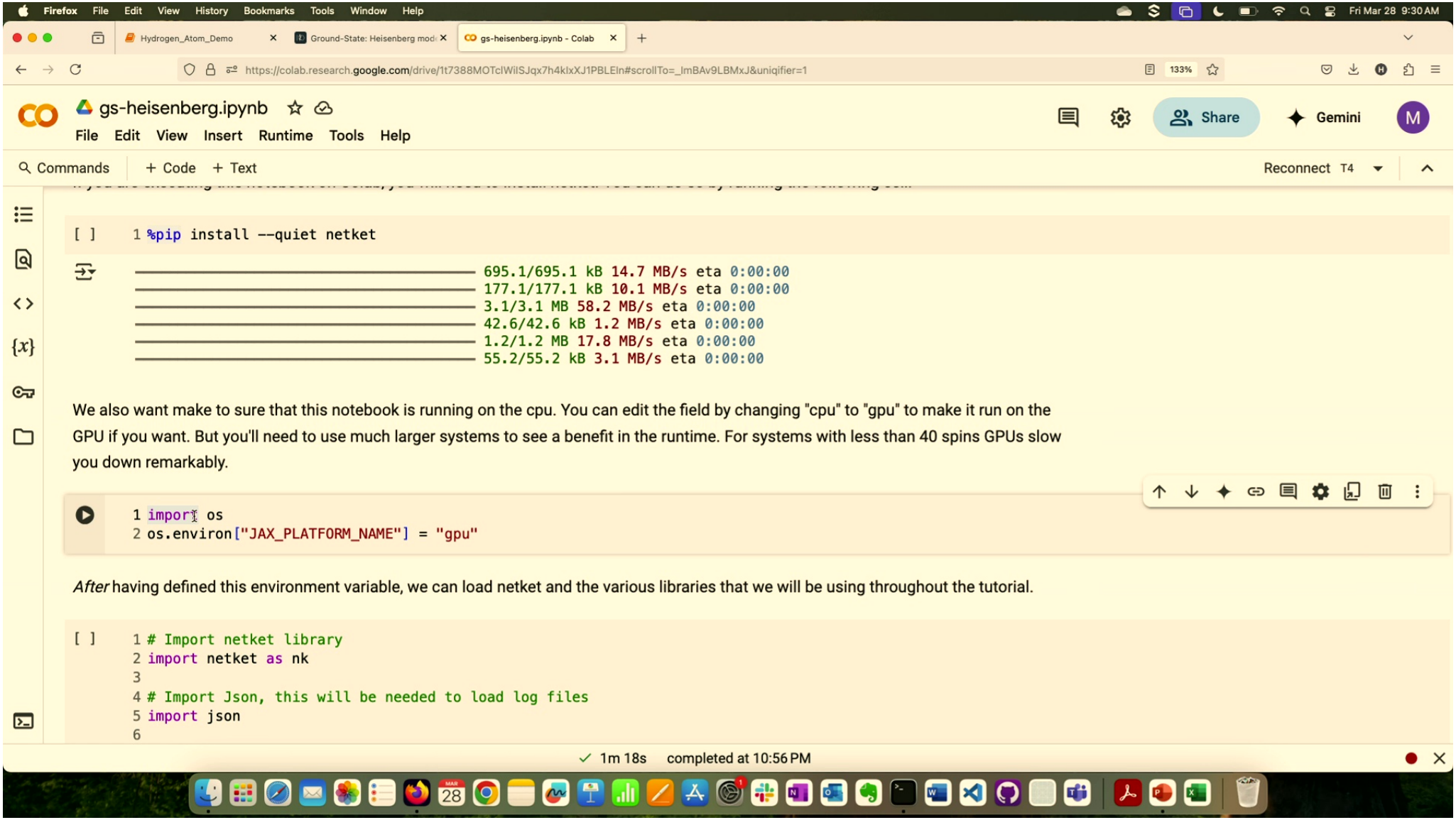
L is the length of the chain, and we will use both open and periodic boundary conditions. $\vec{\sigma} = (\sigma^x, \sigma^y, \sigma^z)$ denotes the vector of Pauli matrices. Please note that there is a factor of 2 between Pauli-matrices and spin-1/2 operators (thus a factor of 4 in H).

We will consider in this tutorial 5 possible ways of determining the ground-state of this model.

0. Installing Netket

If you are executing this notebook on Colab, you will need to install netket. You can do so by running the following cell:

```
[ ]  1 %pip install --quiet netket
```


Firefox File Edit View History Bookmarks Tools Window Help
Fri Mar 28 9:30 AM
Hydrogen_Atom_Demo
Ground-State: Heisenberg mod
gs-heisenberg.ipynb - Colab
https://colab.research.google.com/drive/1t7388MOTclWilSJqx7h4klxXJ1PBLEln#scrollTo=_lmBAv9LBMxJ&uniqifier=1
133%
gs-heisenberg.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Reconnect T4
[ ] 1 %pip install --quiet netket
695.1/695.1 kB 14.7 MB/s eta 0:00:00
177.1/177.1 kB 10.1 MB/s eta 0:00:00
3.1/3.1 MB 58.2 MB/s eta 0:00:00
42.6/42.6 kB 1.2 MB/s eta 0:00:00
1.2/1.2 MB 17.8 MB/s eta 0:00:00
55.2/55.2 kB 3.1 MB/s eta 0:00:00
We also want make to sure that this notebook is running on the cpu. You can edit the field by changing "cpu" to "gpu" to make it run on the GPU if you want. But you'll need to use much larger systems to see a benefit in the runtime. For systems with less than 40 spins GPUs slow you down remarkably.
1 import os
2 os.environ["JAX_PLATFORM_NAME"] = "gpu"
After having defined this environment variable, we can load netket and the various libraries that we will be using throughout the tutorial.
[ ] 1 # Import netket library
2 import netket as nk
3
4 # Import Json, this will be needed to load log files
5 import json
6
1m 18s completed at 10:56 PM

```
# We impose to have a fixed total magnetization of zero
hi = nk.hilbert.Spin(s=0.5, total_sz=0, N=g.n_nodes)
```

The final element of the triptych is of course the Hamiltonian acting in this Hilbert space, which in our case in already defined in NetKet. Note that the NetKet Hamiltonian uses Pauli Matrices (if you prefer to work with spin-$1/2$ operators, it's pretty trivial to define your own custom Hamiltonian, as covered in another tutorial)

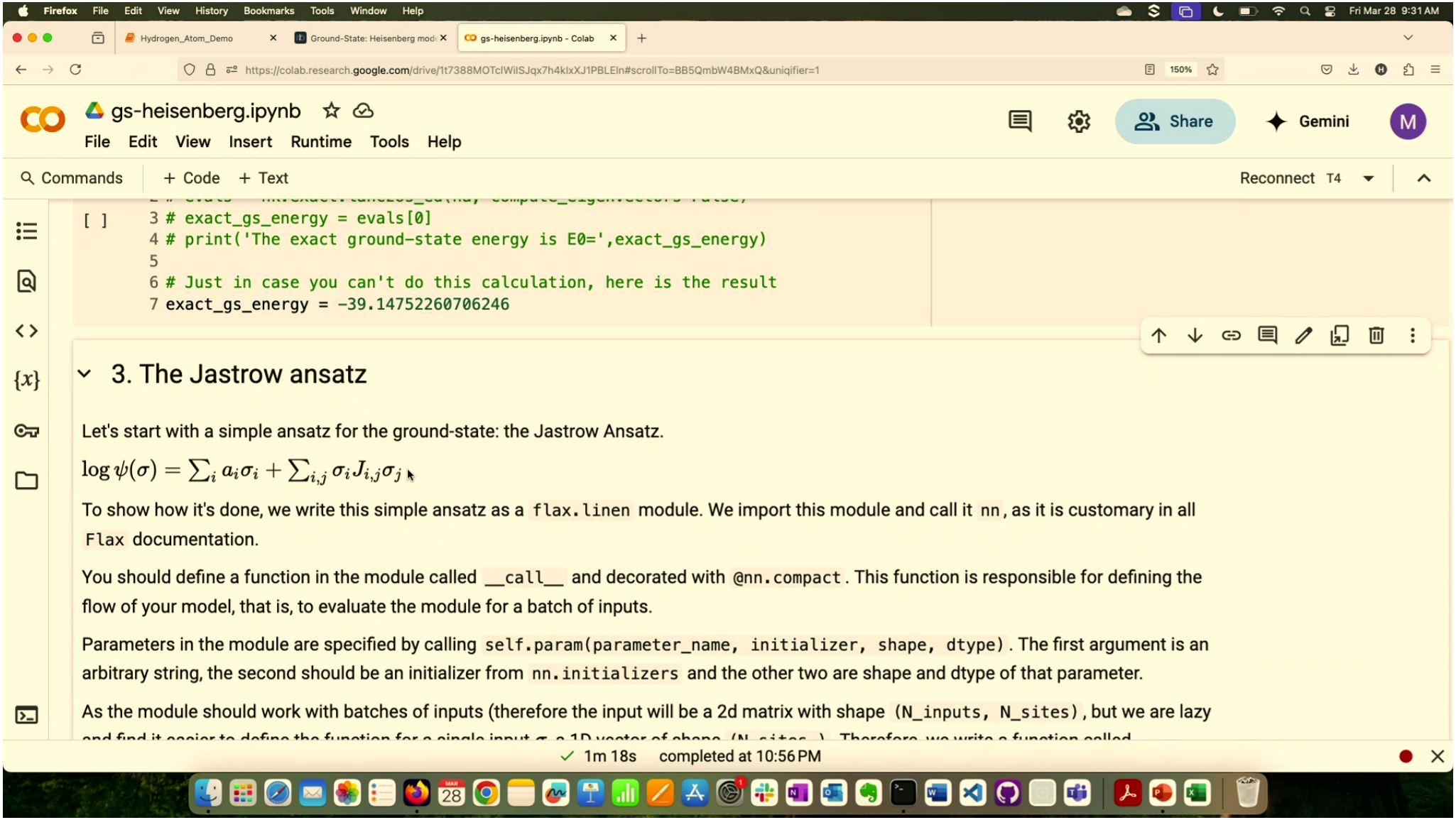
```
# calling the Heisenberg Hamiltonian
ha = nk.operator.Heisenberg(hilbert=hi, graph=g)
```

2. Exact Diagonalization (as a testbed)

Just as a matter of future comparison, let's compute the exact ground-state energy (since this is still possible for $L = 22$ using brute-force exact diagonalization). NetKet provides wrappers to the Lanczos algorithm which we now use:

```
# compute the ground-state energy (here we only need the lowest energy, and do not need the eigenstate)
# evals = nk.exact.lanczos_ed(ha, compute_eigenvectors=False)
# exact_gs_energy = evals[0]
# print('The exact ground-state energy is E0=',exact_gs_energy)

# Just in case you can't do this calculation, here is the result
exact_gs_energy = -39.14752260706246
```


Firefox File Edit View History Bookmarks Tools Window Help
Fri Mar 28 9:31 AM
Hydrogen_Atom_Demo
Ground-State: Heisenberg mod
gs-heisenberg.ipynb - Colab
https://colab.research.google.com/drive/1t7388MOTclWilSJqx7h4klxXJ1PBLEln#scrollTo=BB5QmbW4BMxQ&uniqifier=1
150%
gs-heisenberg.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Reconnect T4
[ ]
# exact_gs_energy = evals[0]
# print('The exact ground-state energy is E0=',exact_gs_energy)
# Just in case you can't do this calculation, here is the result
exact_gs_energy = -39.14752260706246
3. The Jastrow ansatz
Let's start with a simple ansatz for the ground-state: the Jastrow Ansatz.
$\log \psi(\sigma) = \sum_i a_i \sigma_i + \sum_{i,j} \sigma_i J_{i,j} \sigma_j$
To show how it's done, we write this simple ansatz as a flax.linen module. We import this module and call it nn, as it is customary in all Flax documentation.
You should define a function in the module called __call__ and decorated with @nn.compact. This function is responsible for defining the flow of your model, that is, to evaluate the module for a batch of inputs.
Parameters in the module are specified by calling self.param(parameter_name, initializer, shape, dtype). The first argument is an arbitrary string, the second should be an initializer from nn.initializers and the other two are shape and dtype of that parameter.
As the module should work with batches of inputs (therefore the input will be a 2d matrix with shape (N_inputs, N_sites), but we are lazy
1m 18s completed at 10:56 PM

Well that's not too bad for a simple ansatz. But we can do better, can't we?

3. Learning with a Restricted Boltzmann Machine (RBM)

We will now consider another celebrated ansatz, the Restricted Boltzmann Machine (RBM). It simply consists of two layers: a visible one representing the L spin 1/2 degrees of freedom, and an hidden one which contains a different number M of hidden units. There are connections between all visible and hidden nodes. The ansatz is the following

$$\Psi_{\mathrm{RBM}}(\sigma_1^z, \sigma_2^z, \dots, \sigma_L^z) = \exp\left(\sum_{i=1}^{L} a_i \sigma_i^z\right) \prod_{i=1}^{M} \cosh\left(b_i + \sum_j W_{ij} \sigma_j^z\right)$$

a_i (resp. b_i) are the visible (resp. hidden) bias. Together with the weights W_{ij}, they are variational parameters that we (or rather NetKet) will optimize to minimize the energy. Netket gives you the control on the important parameters in this ansatz, such as M and the fact that you want to use or not the biases. The full explanation is here.

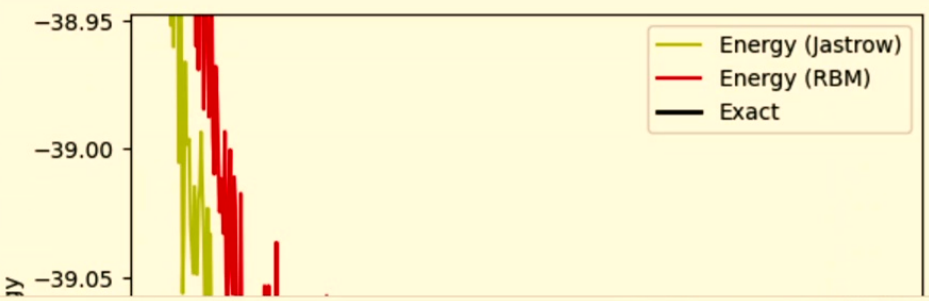
More conveniently (especially if you want to try another L in this tutorial), let's define the hidden unit density $\alpha = M/L$, and invoke the RBM ansatz in NetKet with as many hidden as visible units.

```
[ ]  1 # RBM ansatz with alpha=1
     2 ma = nk.models.RBM(alpha=1)
```

✓ 1m 18s completed at 10:56 PM

```
# Extract the relevant information
iters_RBM = data["Energy"]["iters"]
energy_RBM = data["Energy"]["Mean"]

fig, ax1 = plt.subplots()
ax1.plot(iters_Jastrow, energy_Jastrow, color='C8', label='Energy (Jastrow)')
ax1.plot(iters_RBM, energy_RBM, color='red', label='Energy (RBM)')
ax1.set_ylabel('Energy')
ax1.set_xlabel('Iteration')
plt.axis([0,iters_RBM[-1],exact_gs_energy-0.03,exact_gs_energy+0.2])
plt.axhline(y=exact_gs_energy, xmin=0,
            xmax=iters_RBM[-1], linewidth=2, color='k', label='Exact')
ax1.legend()
plt.show()
```



✓ 1m 18s completed at 10:56 PM

Firefox File Edit View History Bookmarks Tools Window Help — Fri Mar 28 9:33 AM

Hydrogen_Atom_Demo | Ground-State: Heisenberg mod | gs-heisenberg.ipynb - Colab

https://colab.research.google.com/drive/1t7388MOTclWilSJqx7h4klxXJ1PBLEln#scrollTo=z2nXmYsiBMxR&uniqifier=1

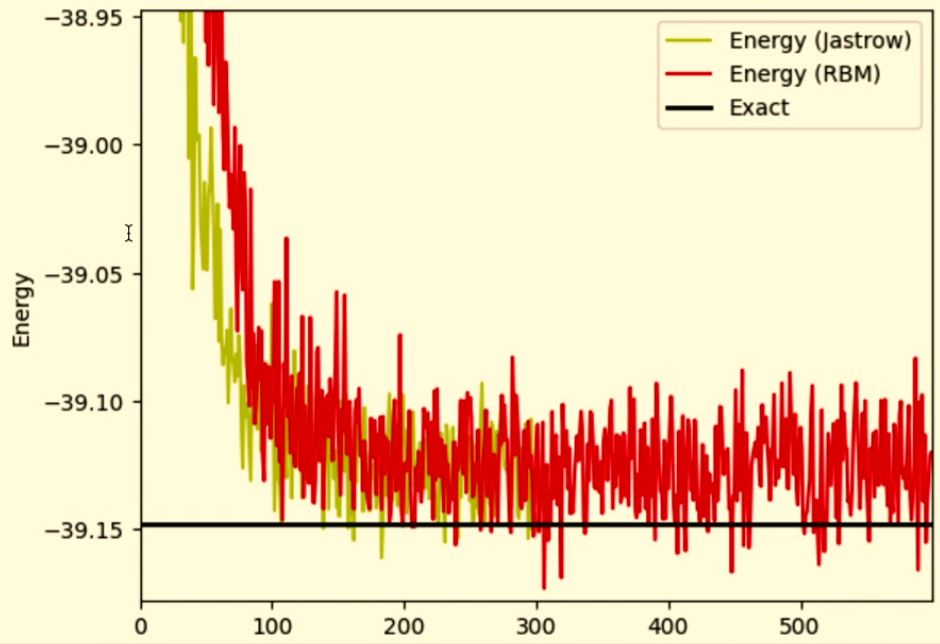
gs-heisenberg.ipynb

File Edit View Insert Runtime Tools Help

Share | Gemini | M

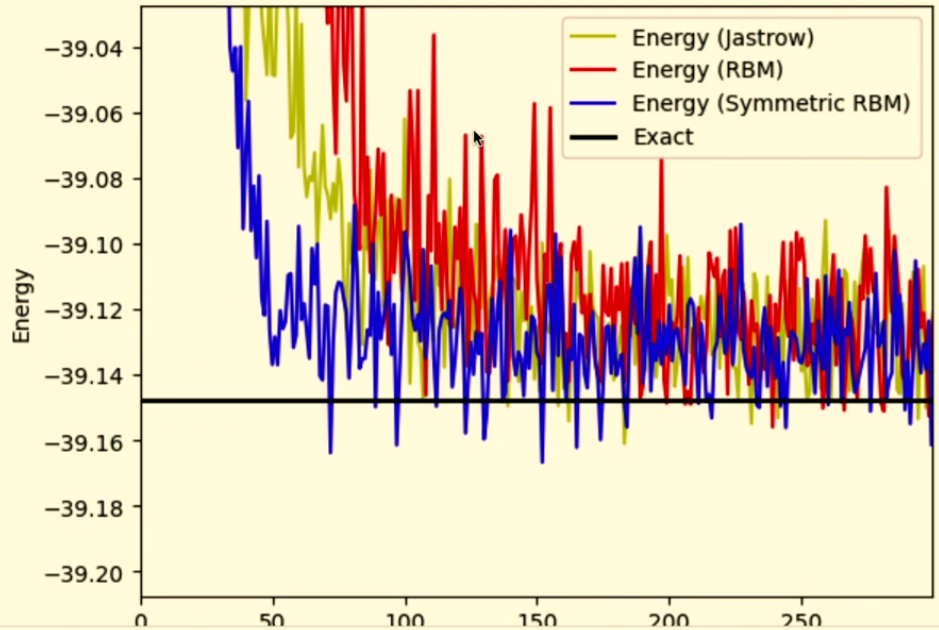
Commands | + Code | + Text | Reconnect T4

```
15                 xmax=iters_RBM[-1], linewidth=2, color='k', label='Exact')
16 ax1.legend()
17 plt.show()
```



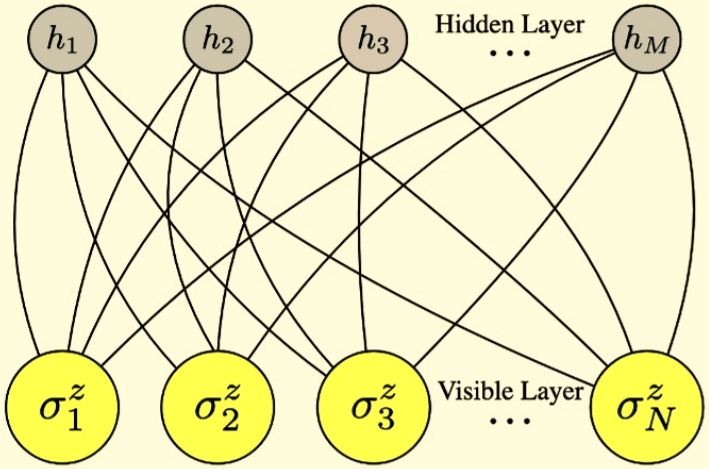
✓ 1m 18s completed at 10:56 PM

```
18                     xmax=iters_RBM[-1], linewidth=2, color='k', label='Exact')
19 ax1.legend()
20 plt.show()
```



✓ 1m 18s completed at 10:56 PM

Neural Quantum States



Restricted Bolzmann Machine (RBM)

$$\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2 \ldots \mathcal{S}_N)$$

$$\Psi_M(\mathcal{S}; \mathcal{W}) = \sum_{\{h_i\}} e^{\sum_j a_j \sigma_j^z + \sum_i b_i h_i + \sum_{ij} W_{ij} h_i \sigma_j^z}$$

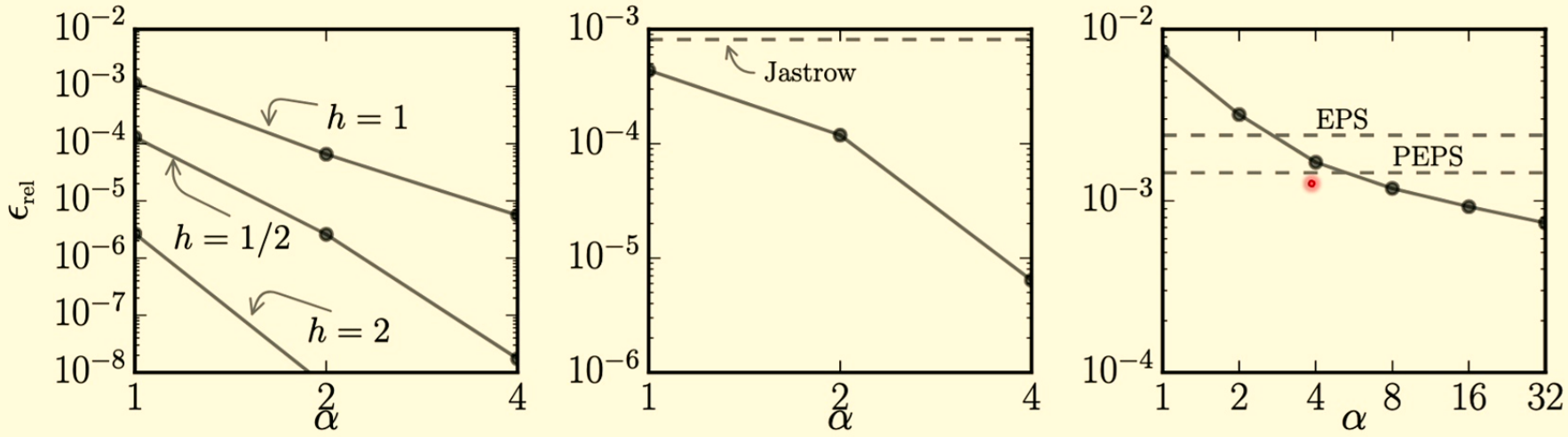
$$\Psi(\mathcal{S}; \mathcal{W}) = e^{\sum_i a_i \sigma_i^z} \times \Pi_{i=1}^{M} F_i(\mathcal{S})$$

$$F_i(\mathcal{S}) = 2\cosh\left[b_i + \sum_j W_{ij} \sigma_j^z\right]$$

Carleo, Troyer, Science, 2017.

Ground States

$\alpha = M/N$



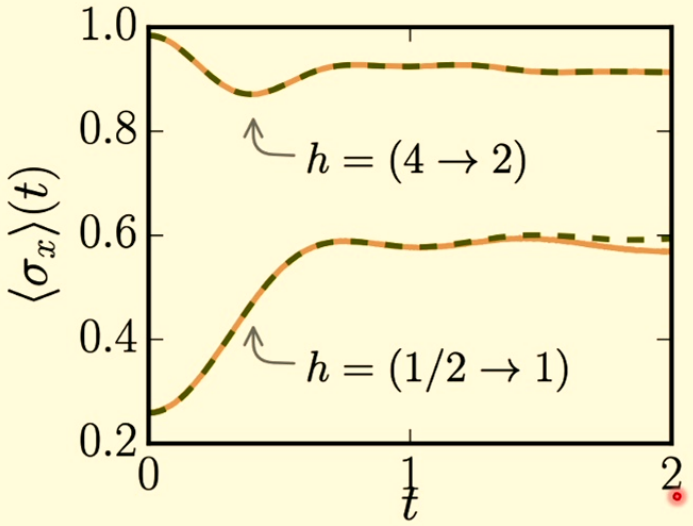
$$\mathcal{H}_{\text{TFI}} = -h \sum_i \sigma_i^x - \sum_{\langle i,j \rangle} \sigma_i^z \sigma_j^z$$

$$\mathcal{H}_{\text{AFH}} = \sum_{\langle i,j \rangle} \sigma_i^x \sigma_j^x + \sigma_i^y \sigma_j^y + \sigma_i^z \sigma_j^z$$

Carleo, Troyer, Science, 2017.

Time evolution

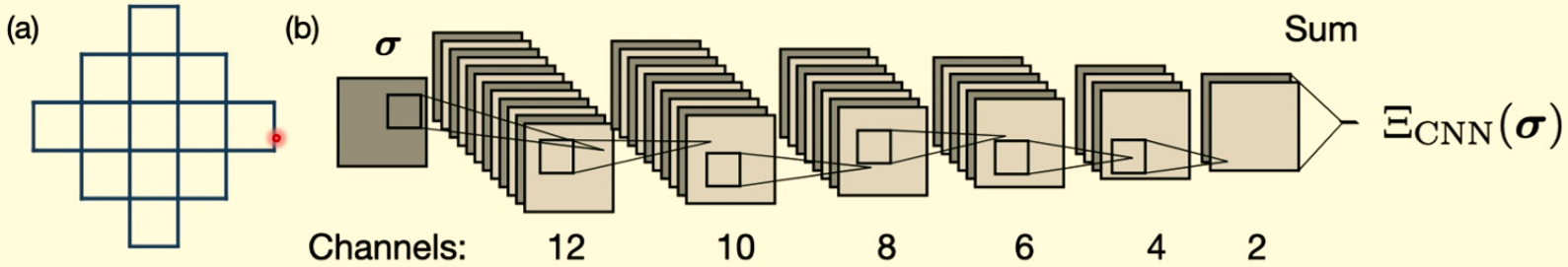
$$R(t;\dot{\mathcal{W}}(t)) = \mathrm{dist}(\partial_t \Psi(\mathcal{W}(t)), -i\mathcal{H}\Psi)$$

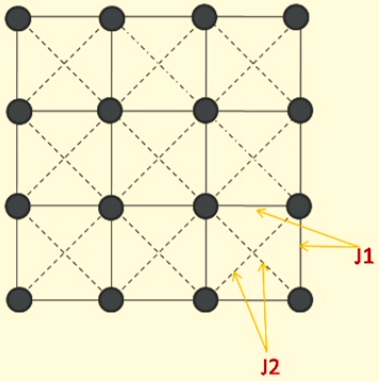


$$\mathcal{H}_{\mathrm{TFI}} = -h\sum_i \sigma_i^x - \sum_{\langle i,j\rangle} \sigma_i^z \sigma_j^z$$

Carleo, Troyer, Science, 2017.

Convolutional Neural Network Quantum States

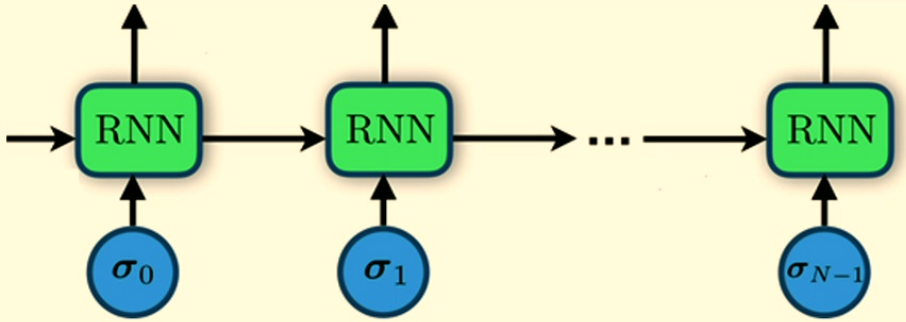




$$\hat{H} = J_1 \sum_{\langle ij \rangle} \hat{\boldsymbol{S}}_i \cdot \hat{\boldsymbol{S}}_j + J_2 \sum_{\langle\langle ij \rangle\rangle} \hat{\boldsymbol{S}}_i \cdot \hat{\boldsymbol{S}}_j$$

Choo, Neupert, Carleo, PRB, 2019.

Recurrent Neural Networks (RNNs)



Very powerful at generating sequential data:

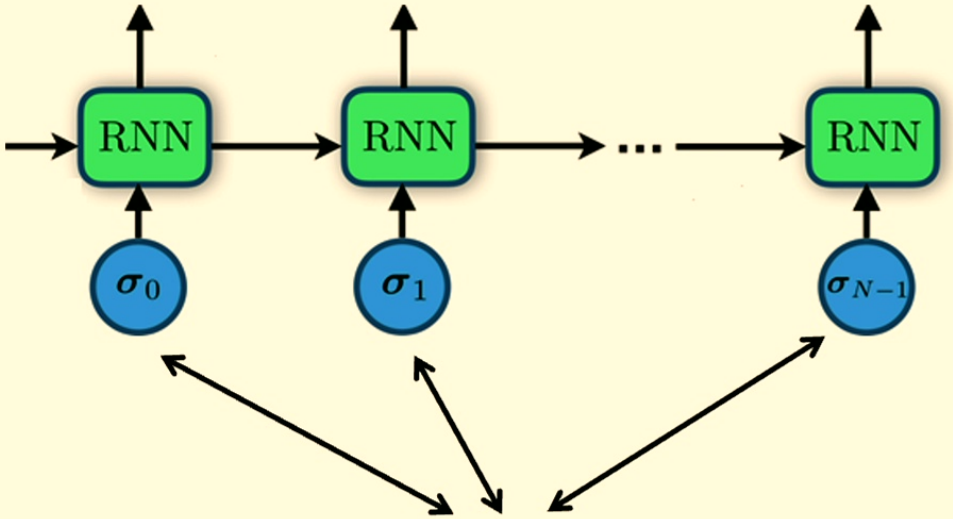
- Speech recognition, machine translation,…

An RNN is a Universal Turing Machine: can be seen as a general-purpose computer that can perform any classical computation.

Siegelmann and Sontag, ACM, 1982

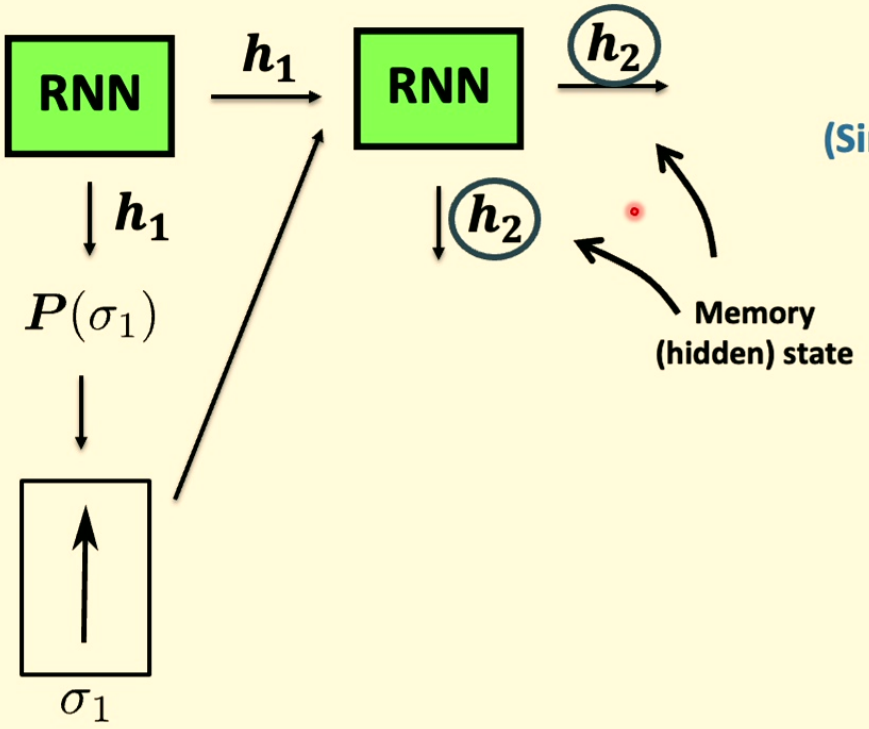
Chung and Siegelmann, NeurIPS 2021

RNNs can be also used in many-body Physics



"Spins" instead of "Words"

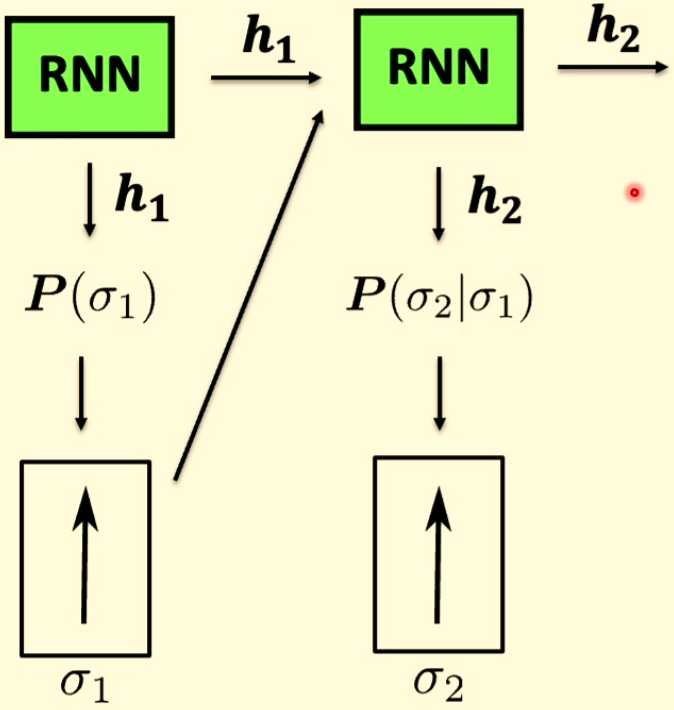
Autoregressive sampling



(Simplest) Recursion relation:

$$h_n = f(W[h_{n-1}; \sigma_{n-1}] + b)$$

Autoregressive sampling



(Simplest) Recursion relation:

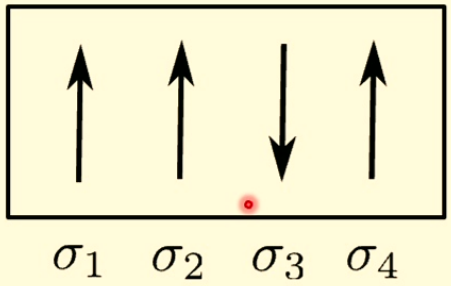
$$h_n = f(W[h_{n-1}; \sigma_{n-1}] + b)$$

Conditional probability:

$$P(\sigma_i|\sigma_{<i}) = \mathbf{Softmax}(Uh_n + c) \cdot \sigma_n$$

W, U, b **and** ***c*** **are the parameters of the RNN.**

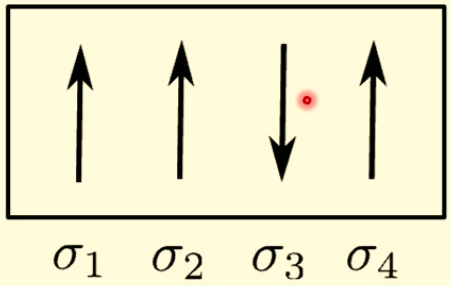
RNN wave functions



$$|\Psi_{\mathbf{RNN}}(\sigma_1,\sigma_2,\sigma_3,\sigma_4)|^2 = \boldsymbol{P}(\sigma_1)\boldsymbol{P}(\sigma_2|\sigma_1)\boldsymbol{P}(\sigma_3|\sigma_2,\sigma_1)\boldsymbol{P}(\sigma_4|\sigma_3,\sigma_2,\sigma_1)$$

RNN
wave function
$\Psi_{\mathbf{RNN}}$

Autoregressive sampling

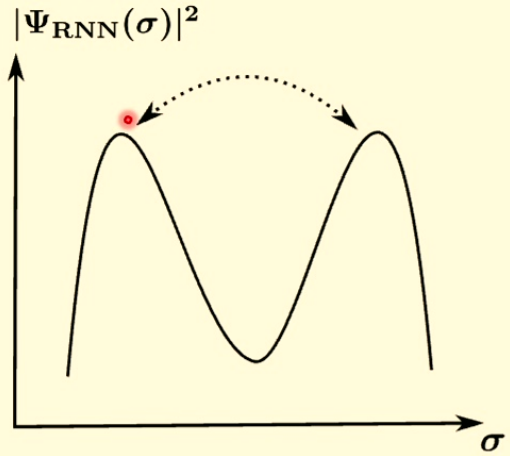


$$|\Psi_{\mathbf{RNN}}(\sigma_1,\sigma_2,\sigma_3,\sigma_4)|^2 = \boldsymbol{P}(\sigma_1)\boldsymbol{P}(\sigma_2|\sigma_1)\boldsymbol{P}(\sigma_3|\sigma_2,\sigma_1)\boldsymbol{P}(\sigma_4|\sigma_3,\sigma_2,\sigma_1)$$

The samples are independents and can be generated in parallel $\{\boldsymbol{\sigma}^{(j)}\}_{i=1}^{N_s}$

Importance sampling: $$\boldsymbol{E}_{\mathbf{RNN}} \equiv \Psi^{\dagger}_{\mathbf{RNN}}\, \boldsymbol{H}\, \Psi_{\mathbf{RNN}} \approx \frac{1}{N_s}\sum_{i=1}^{N_s} \boldsymbol{E}_{\mathbf{loc}}(\boldsymbol{\sigma}^{(j)})$$

Autoregressive sampling

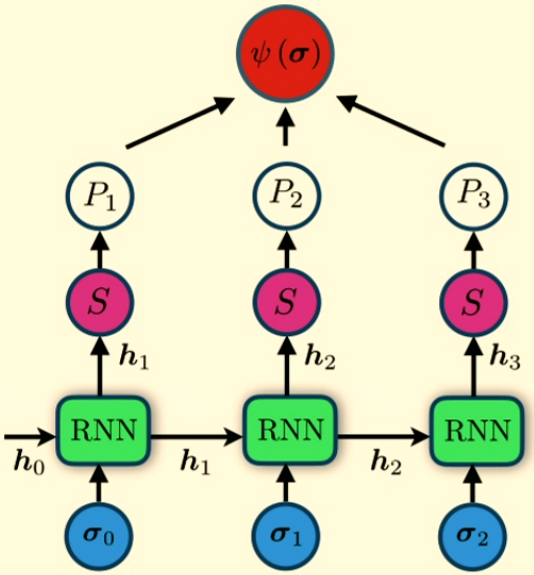


The RNN can sample from multiple modes

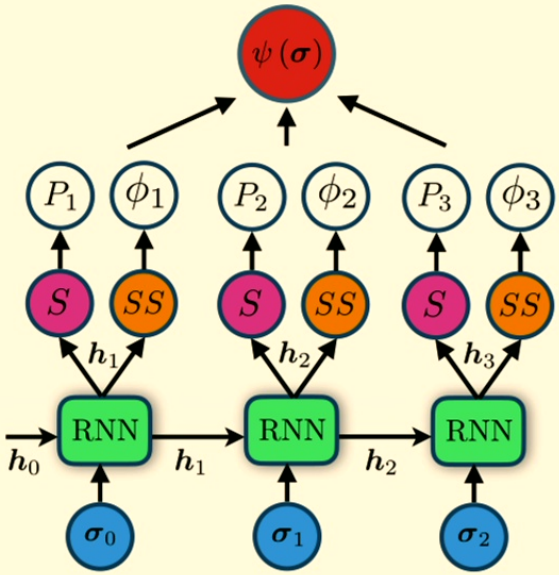
$$|\Psi_{\mathbf{RNN}}(\sigma_1,\sigma_2,\sigma_3,\sigma_4)|^2 = \boldsymbol{P}(\sigma_1)\boldsymbol{P}(\sigma_2|\sigma_1)\boldsymbol{P}(\sigma_3|\sigma_2,\sigma_1)\boldsymbol{P}(\sigma_4|\sigma_3,\sigma_2,\sigma_1)$$

RNN wave functions

Positive RNN wave function



Complex RNN wave function



$$P(\boldsymbol{\sigma}) \equiv \Pi_{n=1}^{N} P_n$$

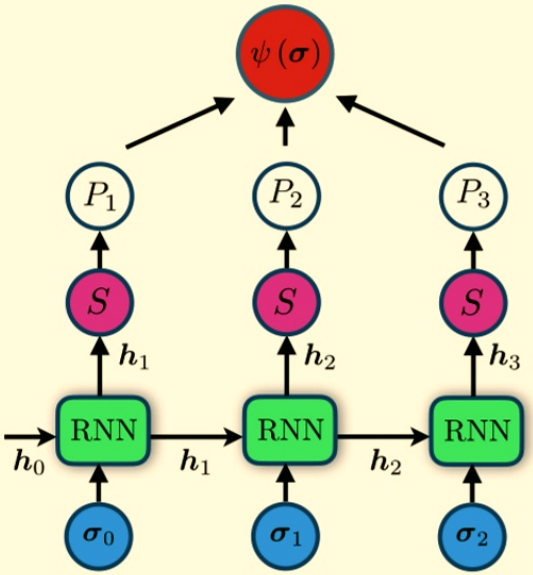
$$\phi(\boldsymbol{\sigma}) \equiv \sum_{n=1}^{N} \phi_n$$

$$\Psi_{\mathrm{RNN}}(\boldsymbol{\sigma}) = \sqrt{P(\boldsymbol{\sigma})}\exp(\mathrm{i}\phi(\boldsymbol{\sigma}))$$

M.H, M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, PRReasearch, 2020.

RNN wave functions

Positive RNN wave function



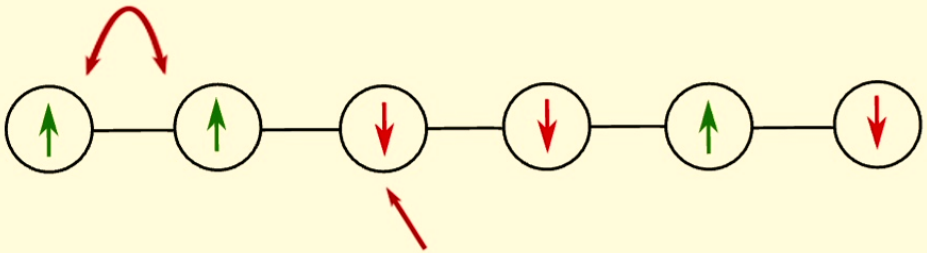
$$P(\boldsymbol{\sigma}) \equiv \Pi_{n=1}^{N} P_n$$

$$\Psi_{\mathbf{RNN}}(\sigma) = \sqrt{P(\sigma)}$$

M.H, M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, PRReasearch, 2020.

Transverse-field Ising model in 1D

Ferromagnetic coupling (order)



Spin flip coupling (disorder)

Hamiltonian:

$$\hat{H} = -\sum_{i=1}^{N-1} \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h \sum_{i=1}^{N} \hat{\sigma}_i^x$$

Order

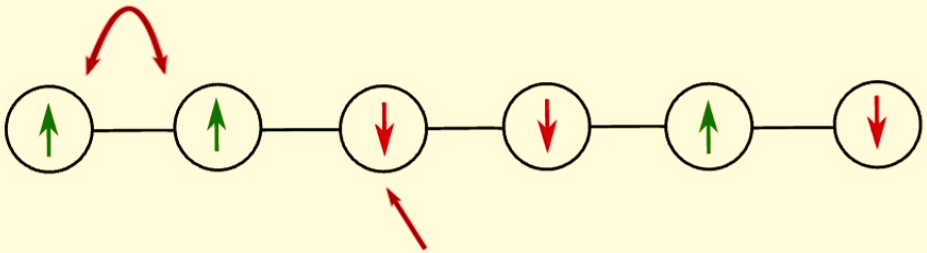
Disorder

Pauli matrices:

$$\hat{\sigma}^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad \hat{\sigma}^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Transverse-field Ising model in 1D

Ferromagnetic coupling (order)



Spin flip coupling (disorder)

Hamiltonian:

$$\hat{H} = -\sum_{i=1}^{N-1} \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h \sum_{i=1}^{N} \hat{\sigma}_i^x$$

Order

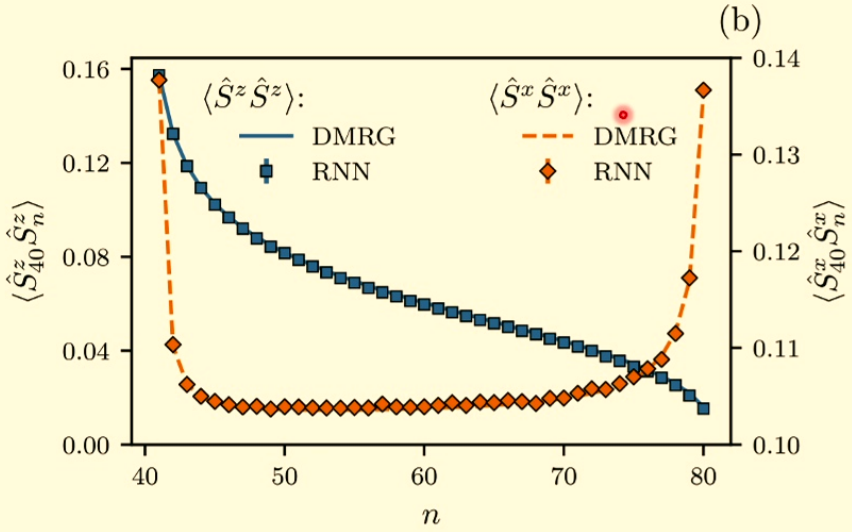
Disorder

$$\mathbf{H} = \begin{pmatrix} * & \cdots & * \\ \vdots & \ddots & \vdots \\ * & \cdots & * \end{pmatrix}$$

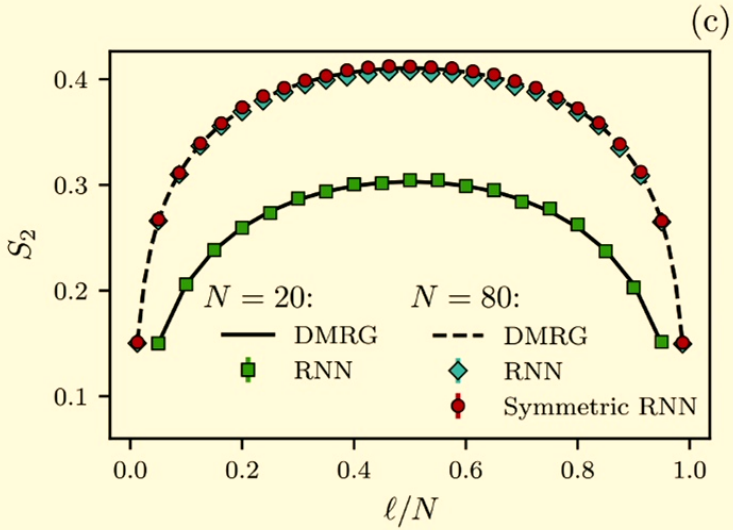
2^N (columns)

2^N (rows)

RNNs can estimate observables and entanglement



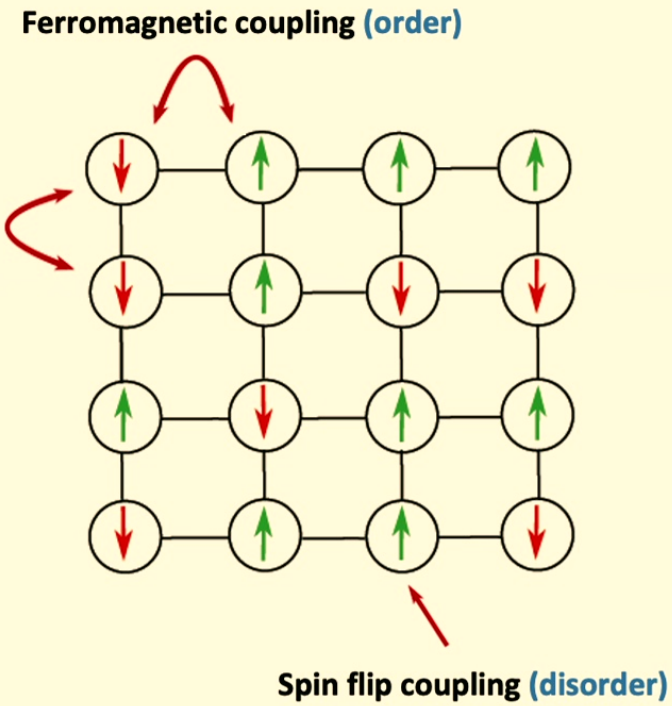
Correlations functions



Entanglement entropies

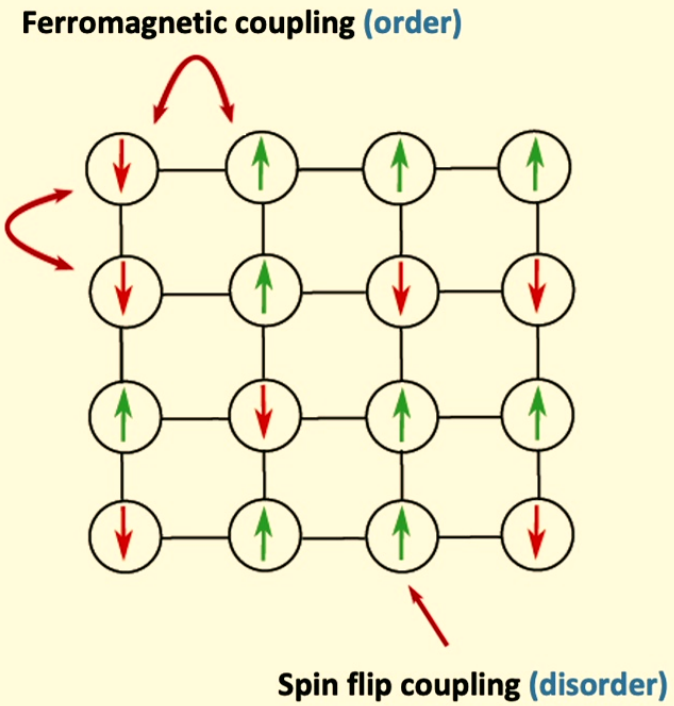
M.H, M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, PRReasearch, 2020.

Transverse-field Ising model in 2D



- In 2D, the exact ground state energy is not known theoretically.

Transverse-field Ising model in 2D



- Hamiltonian:

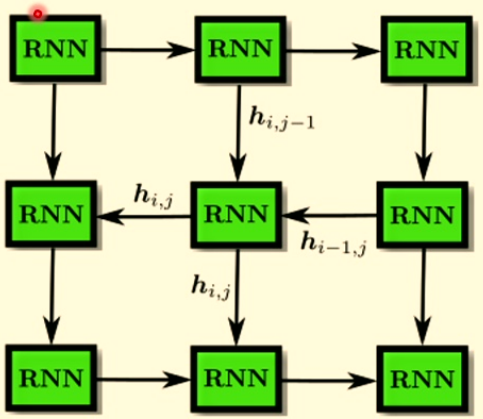
$$\hat{H} = -\sum_{\langle i,j \rangle} \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h \sum_{i=1}^{N} \hat{\sigma}_i^x$$

Order

Disorder

$$\mathbf{H} = \begin{pmatrix} * & \cdots & * \\ \vdots & \ddots & \vdots \\ * & \cdots & * \end{pmatrix} \begin{matrix} \\ \\ \\ \end{matrix} \quad \begin{matrix} 2^N \\ \\ \\ 2^N \end{matrix}$$

2D recurrent neural networks



M.H, M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, RNN Wave functions, PRReasearch, 2020.

A. Grave et al., Multi-dimensional recurrent neural networks, 2007.

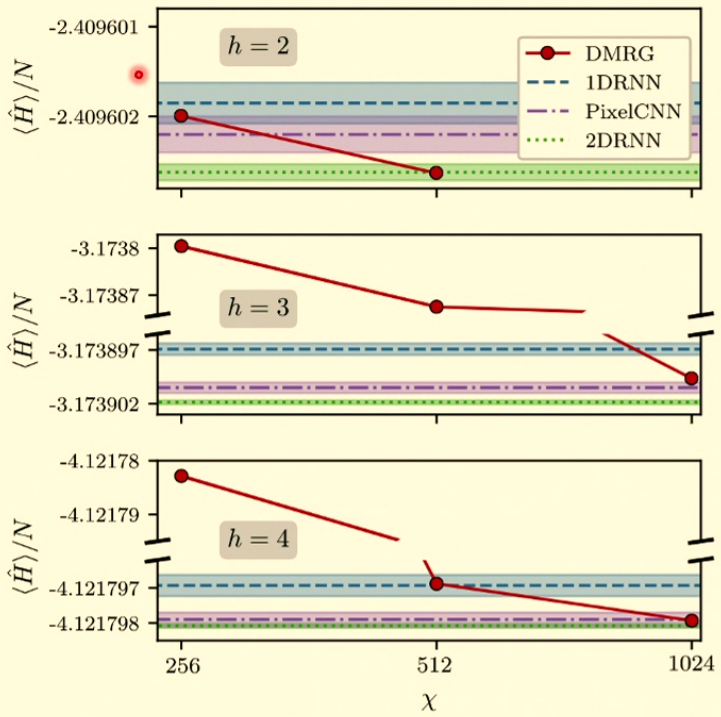
1D recursion relation:

$$\boldsymbol{h}_n = f(W[\boldsymbol{h}_{n-1}; \boldsymbol{\sigma}_{n-1}] + \boldsymbol{b})$$

2D recursion relation:

$$\boldsymbol{h}_{i,j} = f\left(W^{(h)}[\boldsymbol{h}_{i-1,j}; \boldsymbol{\sigma}_{i-1,j}] + W^{(v)}[\boldsymbol{h}_{i,j-1}; \boldsymbol{\sigma}_{i,j-1}] + \boldsymbol{b}\right)$$

Comparison with the state-of-the art in 2D



- Hamiltonian (N = 12x12):

$$\hat{H} = -\sum_{\langle i,j \rangle} \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h \sum_{i=1}^{N} \hat{\sigma}_i^x$$

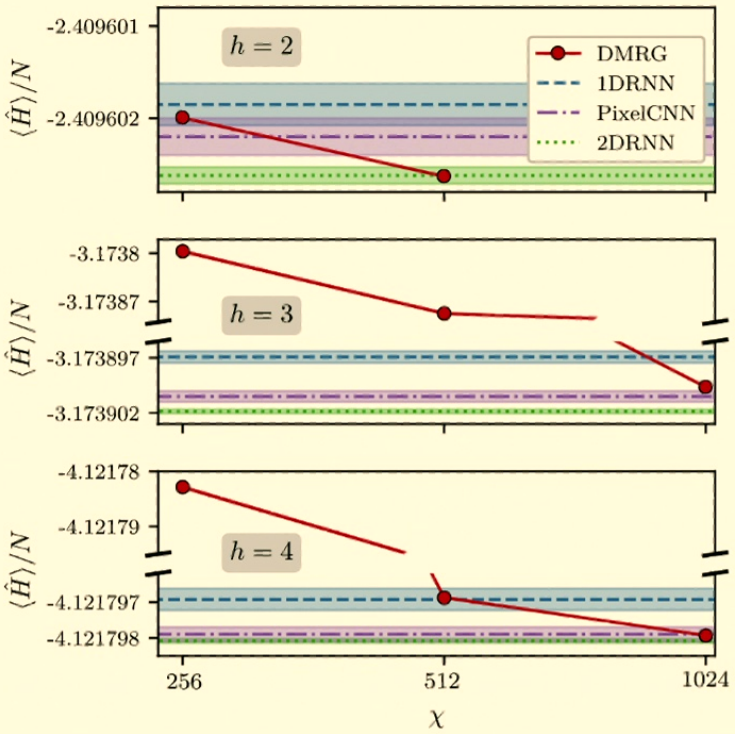
DMRG: Density Matrix Renormalization Group.

2DRNN and PixelCNN: Neural Network architectures.

- The lower the energy the better!

2DRNN: M.H, M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, PRReasearch, 2020.
Pixel CNN: Sharir et al. 2020, PRL, 2020.

Comparison with the state-of-the art in 2D



- **Hamiltonian (N = 12x12):**

$$\hat{H} = -\sum_{\langle i,j \rangle} \hat{\sigma}_i^z \hat{\sigma}_{i+1}^z - h \sum_{i=1}^{N} \hat{\sigma}_i^x$$

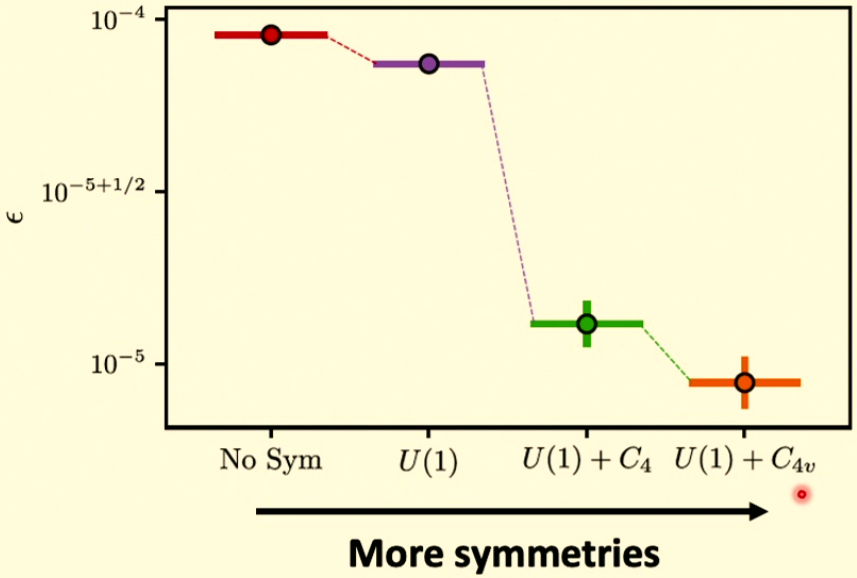
- **RNNs are more compact!**

0.03% of the variational parameters of **DMRG**.

2.6% of the variational parameters of **PixelCNN.**

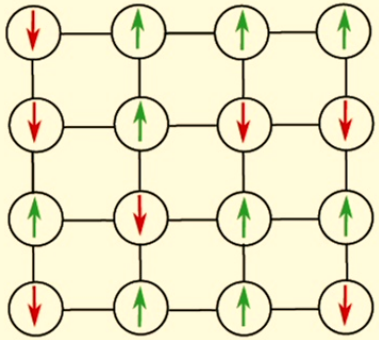
2DRNN: M.H, M. Ganahl, L. Hayward, R. Melko, J. Carrasquilla, PRReasearch, 2020
Pixel CNN: Sharir et al. 2020, PRL, 2020.

Embedding physical symmetries into 2D RNNs



2D Heisenberg model on square lattice

$$\hat{H} = \frac{1}{4}\sum_{\langle i,j\rangle}(\hat{\sigma}_i^x\hat{\sigma}_j^x + \hat{\sigma}_i^y\hat{\sigma}_j^y + \hat{\sigma}_i^z\hat{\sigma}_j^z)$$



- Relative error :

$$\epsilon = |E_{\mathrm{RNN}} - E_{\mathrm{Exact}}|/|E_{\mathrm{Exact}}|$$

M.H, R. Melko, J. Carrasquilla, ML for Physical Sciences workshop, NeurIPS 2021.

What is next?

https://doi.org/10.1038/s42005-024-01584-y

Variational Monte Carlo with large patched transformers

Kyle Sprague & Stefanie Czischek

Large language models, like transformers, have recently demonstrated immense powers in text and image generation. This success is driven by the ability to capture long-range correlations between elements in a sequence. The same feature makes the transformer a powerful wavefunction ansatz that addresses the challenge of describing correlations in simulations of qubit systems. Here we consider two-dimensional Rydberg atom arrays to demonstrate that transformers reach higher accuracies than conventional recurrent neural networks for variational ground state searches. We further introduce large, patched transformer models, which consider a sequence of large atom patches, and show that this architecture significantly accelerates the simulations. The proposed architectures reconstruct ground states with accuracies beyond state-of-the-art quantum Monte Carlo methods, allowing for the study of large Rydberg systems in different phases of matter and at phase transitions. Our high-accuracy ground state representations at reasonable computational costs promise new insights into general large-scale quantum many-body systems.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

What is next?

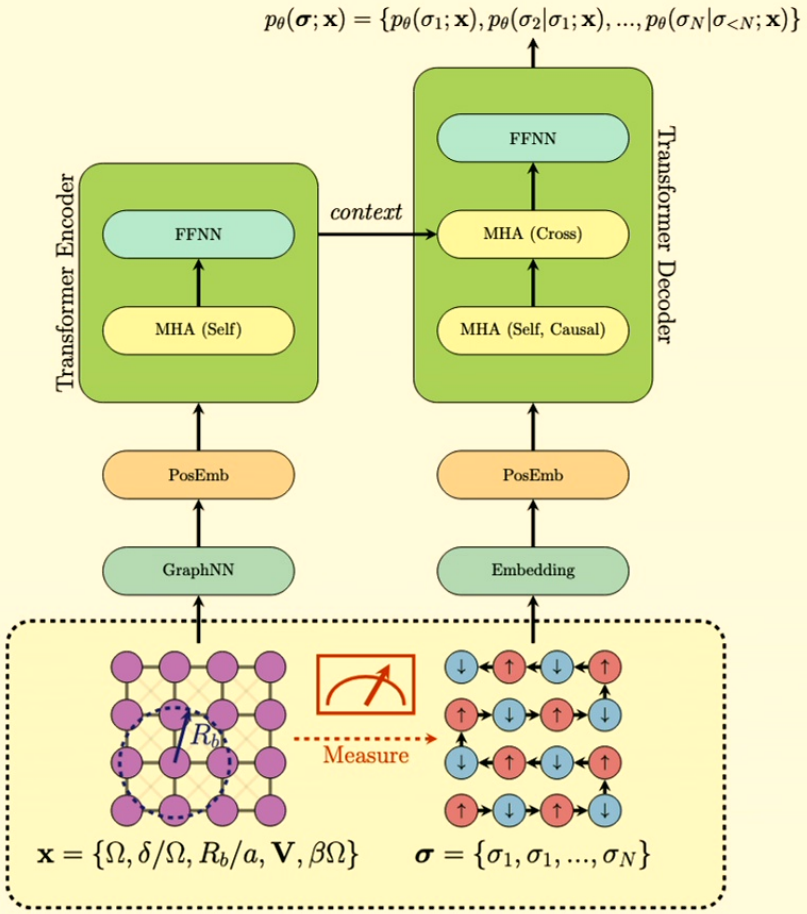
RydbergGPT

David Fitzek,[1,2] Yi Hong Teoh,[3] Hin Pok Fung,[3] Gebremedhin A. Dagnew,[3]
Ejaaz Merali,[3] M. Schuyler Moss,[3] Benjamin MacLellan,[3] and Roger G. Melko[3,4]

[1] *Department of Microtechnology and Nanoscience, Chalmers University of Technology, 412 96 Gothenburg, Sweden*
[2] *Volvo Group Trucks Technology, 405 08 Gothenburg, Sweden*
[3] *Department of Physics and Astronomy, University of Waterloo, 200 University Ave. West, Waterloo, Ontario N2L 3G1, Canada*
[4] *Perimeter Institute for Theoretical Physics, Waterloo, Ontario N2L 2Y5, Canada*

(Dated: June 3, 2024)

We introduce a generative pretained transformer (GPT) designed to learn the measurement outcomes of a neutral atom array quantum computer. Based on a vanilla transformer, our encoder-decoder architecture takes as input the interacting Hamiltonian, and outputs an autoregressive sequence of qubit measurement probabilities. Its performance is studied in the vicinity of a quantum phase transition in Rydberg atoms in a square lattice array. We explore the ability of the architecture to generalize, by producing groundstate measurements for Hamiltonian parameters not seen in the training set. We focus on examples of physical observables obtained from inference on three different models, trained in fixed compute time on a single NVIDIA A100 GPU. These can act as benchmarks for the scaling of larger RydbergGPT models in the future. Finally, we provide RydbergGPT open source, to aid in the development of foundation models based off of a wide variety of quantum computer interactions and data sets in the future.



Thank you!

1. Natural Language Processing meets many-body physics.

2. RNN wave functions are competitive with the state-of-the-art.

3. The future of < Machine Learning | many-body physics > is promising!