Title: PSI 2019/2020 - Machine Learning for Many-Body Physics - Lecture 2

Speakers: Mohamed Hibat Allah

Collection: PSI 2019/2020 - Machine Learning for Many-Body Physics

Date: February 26, 2020 - 9:30 AM

URL: http://pirsa.org/20020068

Lecture 2: Recurrent Network Wavefunctions

Last time: $\hat{H}|\psi_G\rangle = E_G|\psi_G\rangle \longrightarrow E_G \simeq \min_{\lambda} \langle\psi_\lambda|\hat{H}|\psi_\lambda\rangle$.

Eigenvalue problem.

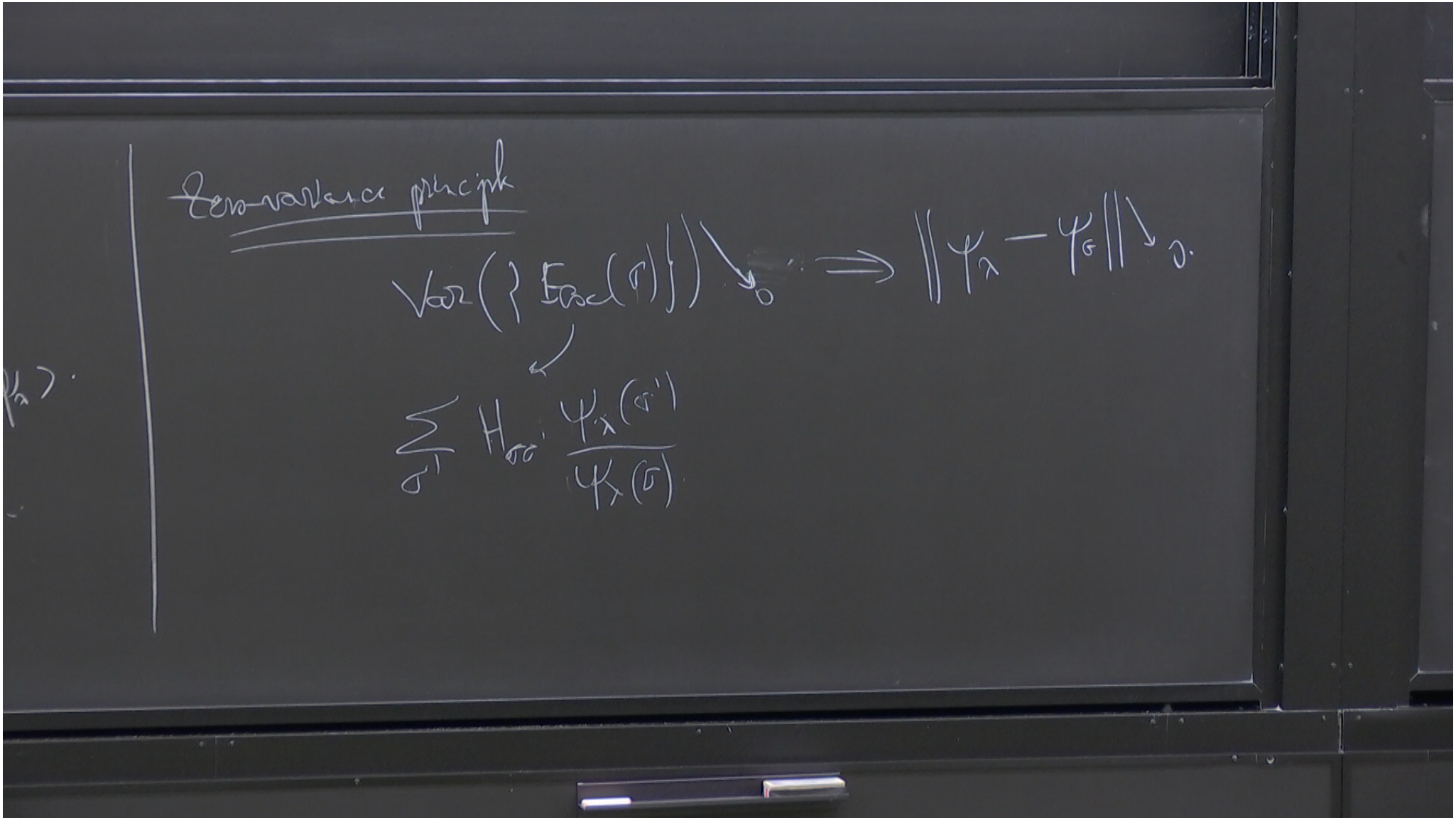
$O(2^{2N})$.

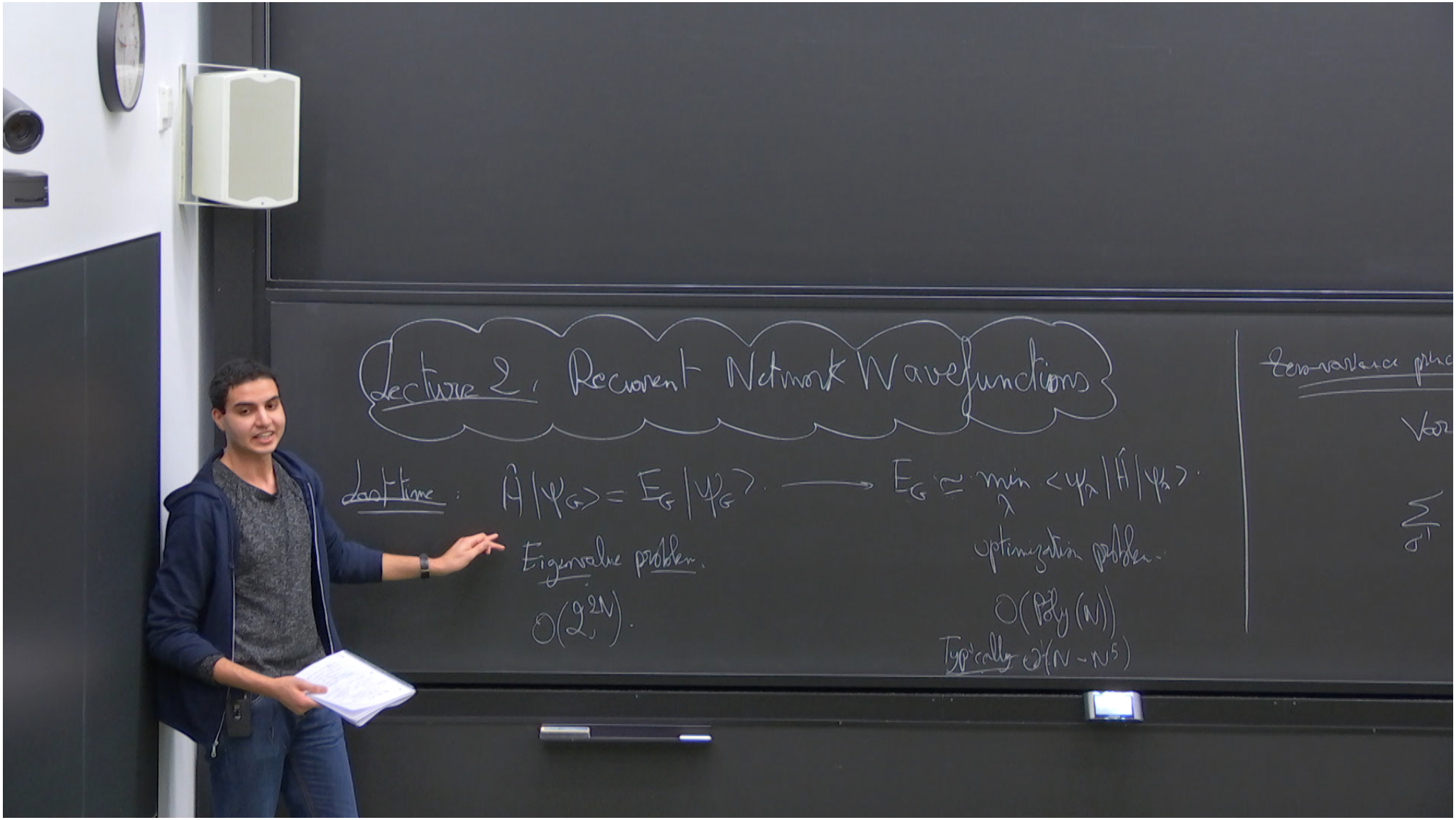
optimization problem.

$O(\text{Poly}(N))$

Typically $O(N - N^5)$

Tens


Low-variance principle
$\mathrm{Var}\left(\{E_{\mathrm{loc}}(\sigma)\}\right) \to 0 \Rightarrow \|\psi_\lambda - \psi_G\| \to 0.$
$\sum_{\sigma'} H_{\sigma\sigma'} \frac{\psi_\lambda(\sigma')}{\psi_\lambda(\sigma)}$


Lecture 2: Recurrent Network Wavefunctions
Last time: $\hat{H}|\psi_G\rangle = E_G|\psi_G\rangle$ ⟶ $E_G \simeq \min_\lambda \langle\psi_\lambda|\hat{H}|\psi_\lambda\rangle$
Eigenvalue problem.
$O(2^{2N})$.
optimization problem.
$O(\text{Poly}(N))$
Typically $O(N - N^5)$


Lecture 2: Recurrent Network Wavefunctions

Lecture 2: Recurrent Network Wavefunctions

Last time: $\hat{A}|\psi_G\rangle = E_G|\psi_G\rangle$ ⟶ $E_G \simeq \min_{\lambda} \langle \psi_\lambda | \hat{H} | \psi_\lambda \rangle$

Ansatz

Eigenvalue problem.

$O(2^{2N})$.

gradient descent

optimization problem.

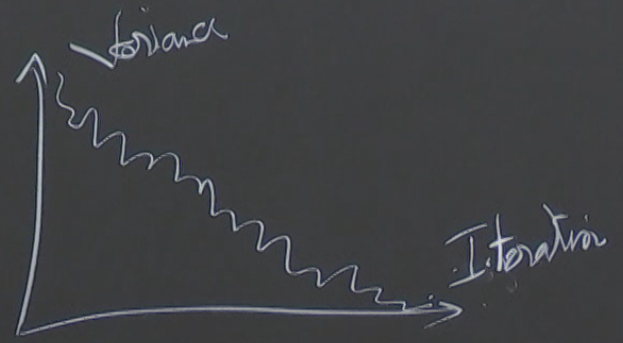
$O(\text{Poly}(N))$

Typically $O(N - N^5)$

Zero-variance principle

$$\mathrm{Var}\left(\{E_{\mathrm{loc}}(\sigma)\}\right) \to 0 \implies \|\psi_\lambda - \psi_G\| \to 0.$$

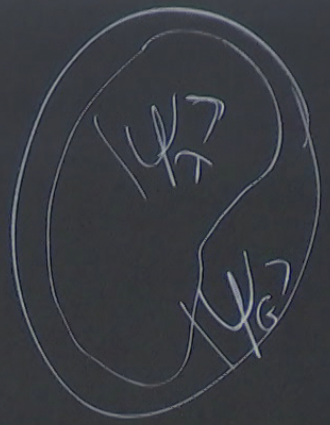
$$\sum_{\sigma'} H_{\sigma\sigma'} \frac{\psi_\lambda(\sigma')}{\psi_\lambda(\sigma)}$$

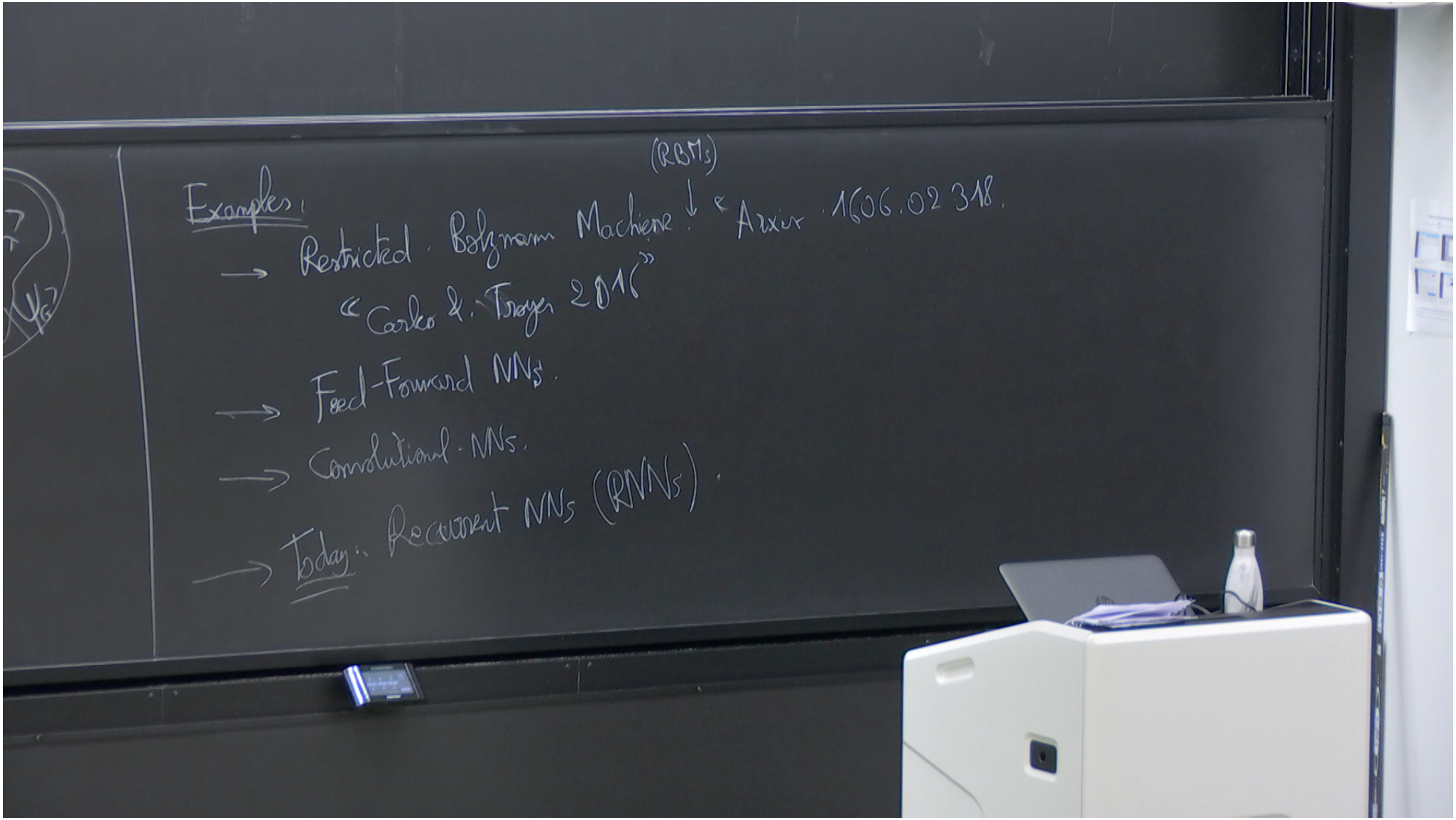


* How to choose "$|\psi_\lambda\rangle$"?

→ $|\psi_\lambda\rangle$ should be expressive enough.

↳ Neural Network




Examples:
(RBMs)
→ Restricted Boltzmann Machine ↓ « Arxiv 1606.02318.
« Carleo & Troyer 2017 »
→ Feed-Forward NNs.
→ Convolutional NNs.
→ Today: Recurrent NNs (RNNs).

Outline

(1) . Recurrent Neural Networks RNNs

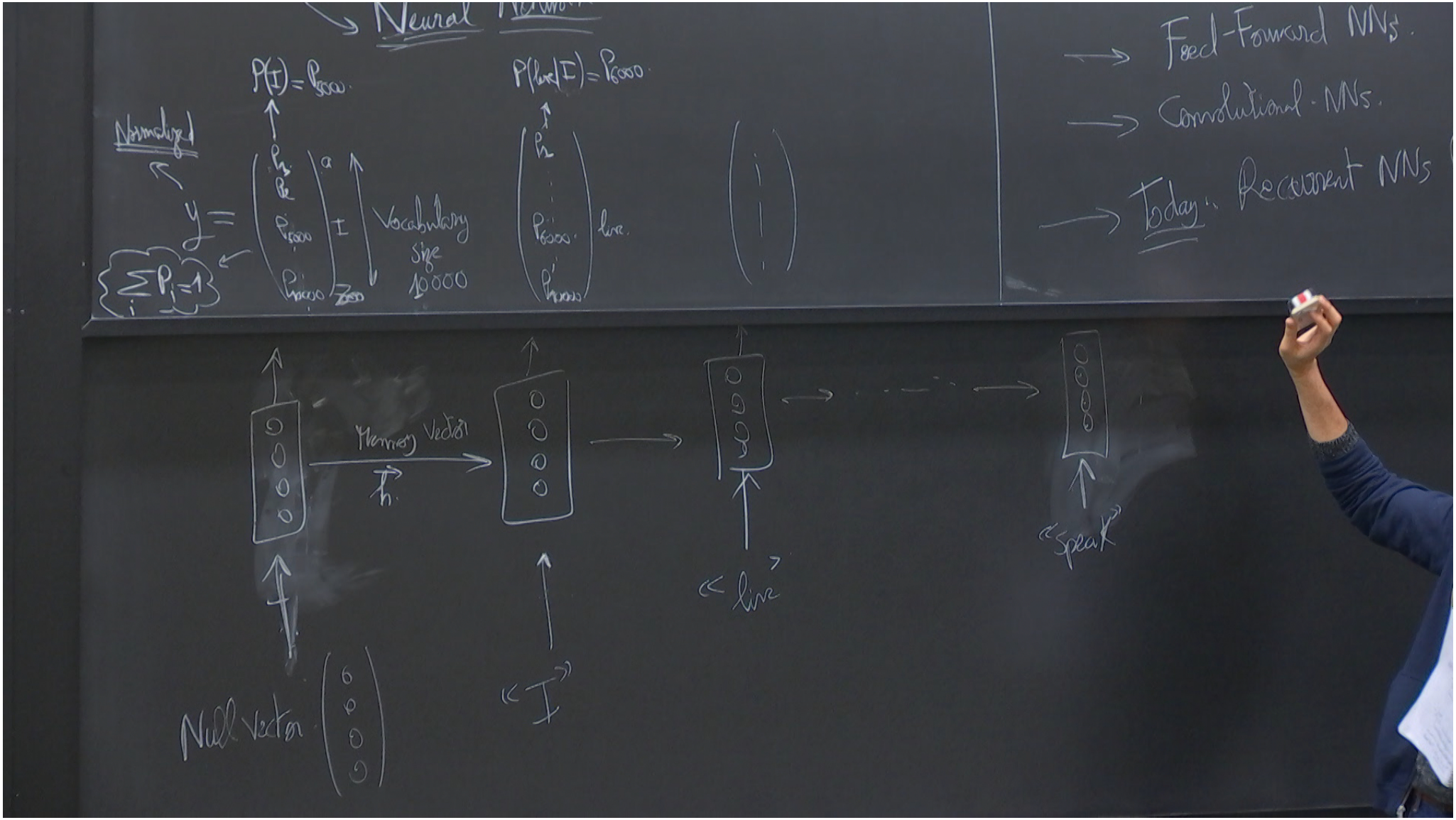
(2) $|\Psi_{\lambda}\rangle = |RNN\rangle$.

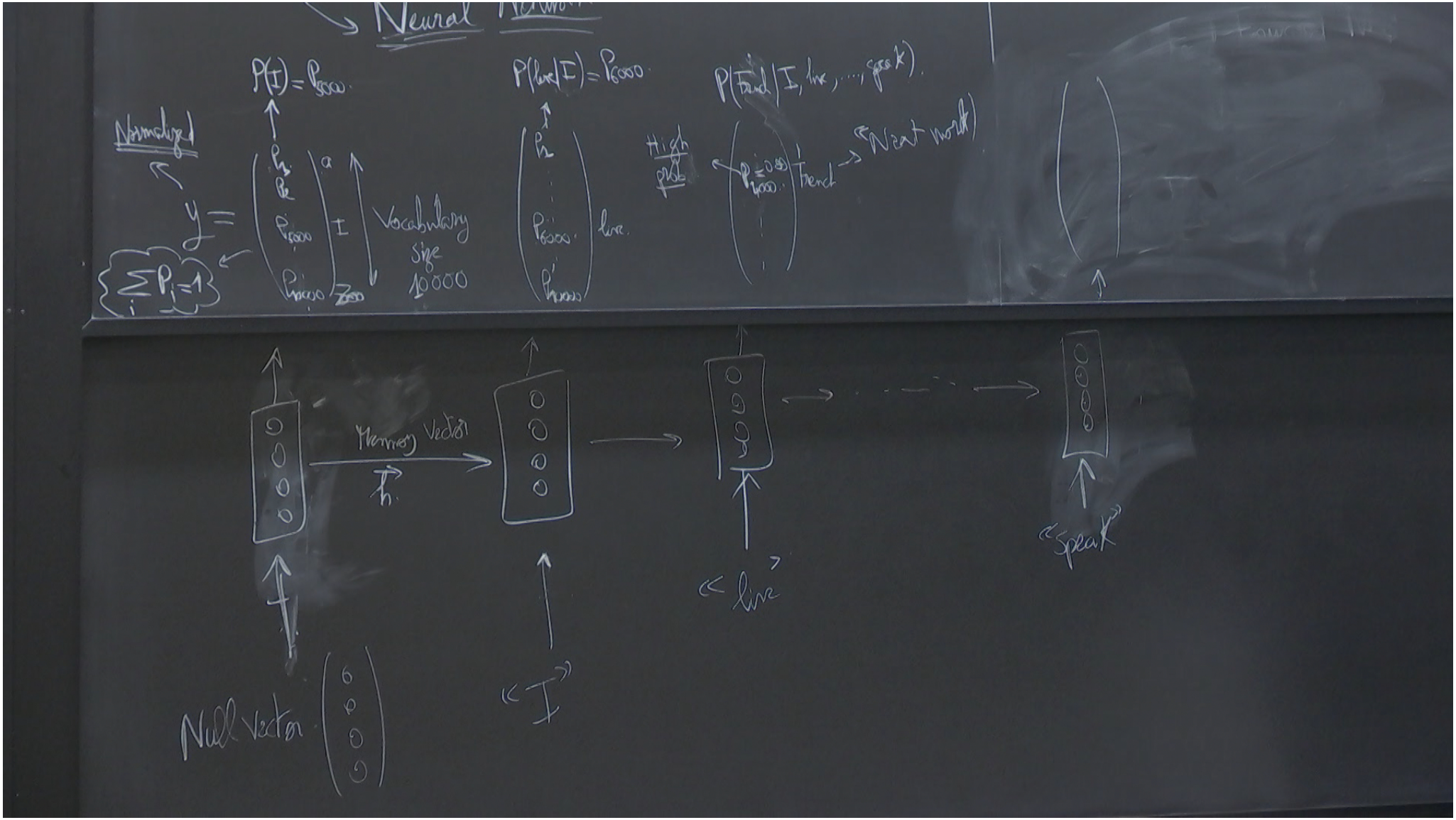
2_1 RNNs:

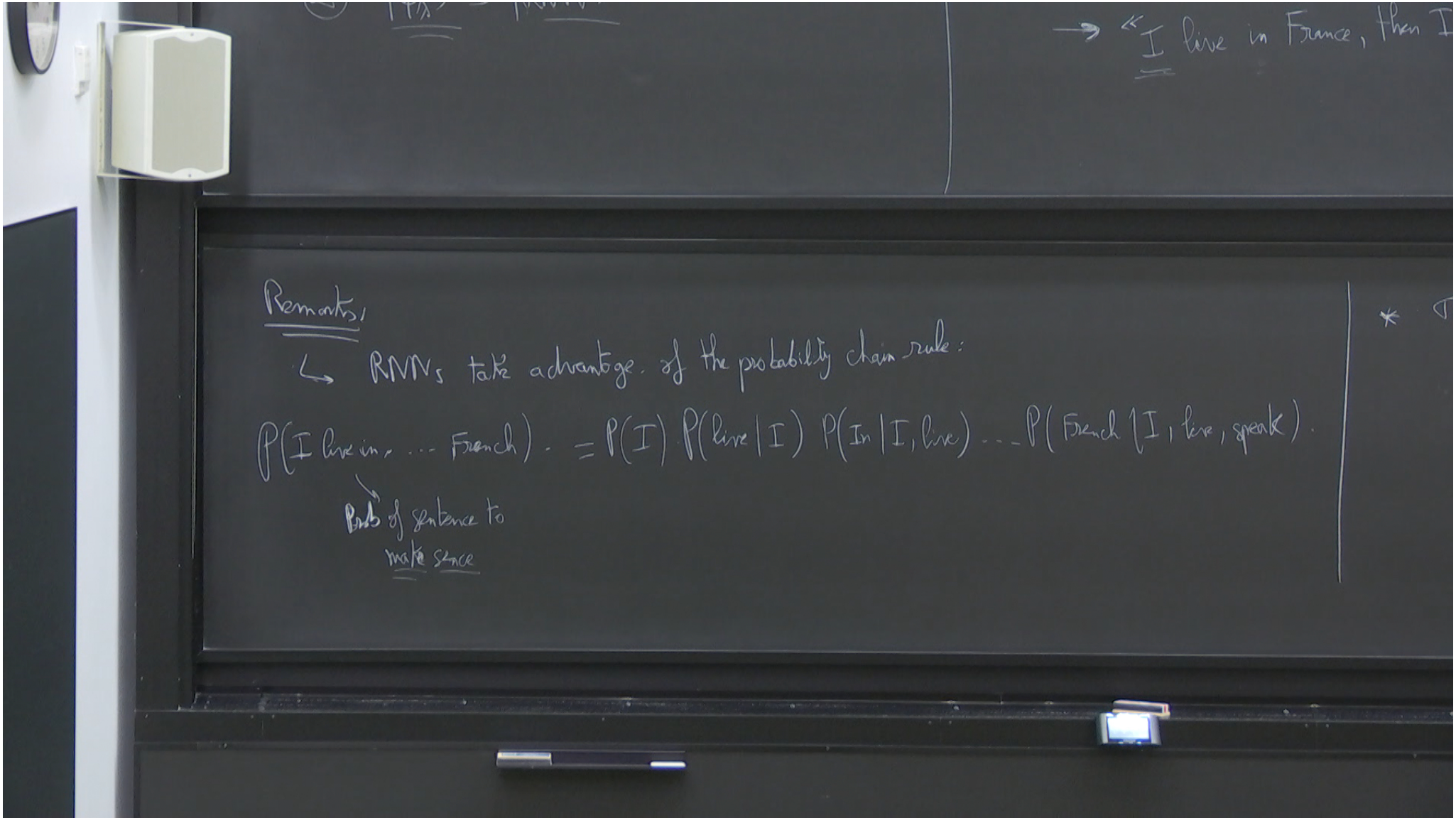
↳ Originally built for language processing

Ex. Machine translation, speech recognition, Music g

→ "I live in France, then I speak ..." → French


Neural
Feed-Forward NNs.
Convolutional NNs.
Today: Recurrent NNs
Normalized
Vocabulary size 10000
Memory Vector
Null vector
«I»
«live»
«speak»


Neural
P(I) = P5000
P(love|I) = P5000
Normalized
Vocabulary size 10000
high prob
Thermo Vectors
«I»
«love»
«speak»
Null Vector

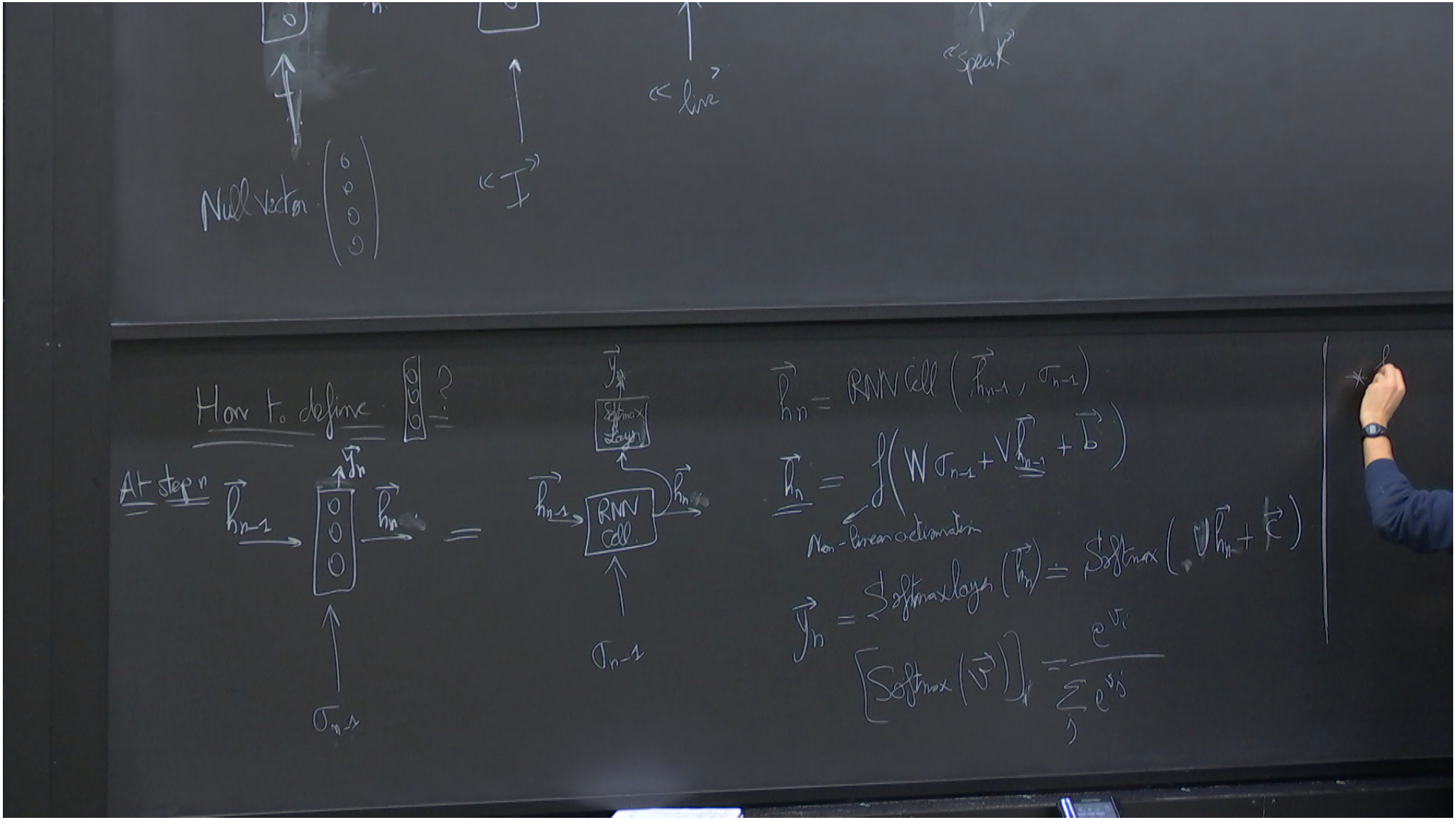

→ "I live in France, then I
Remarks:
↳ RNNs take advantage of the probability chain rule:
P(I live in ... French) = P(I) P(live | I) P(In | I, live) ... P(French | I, live, speak)
Prob of sentence to
make sense

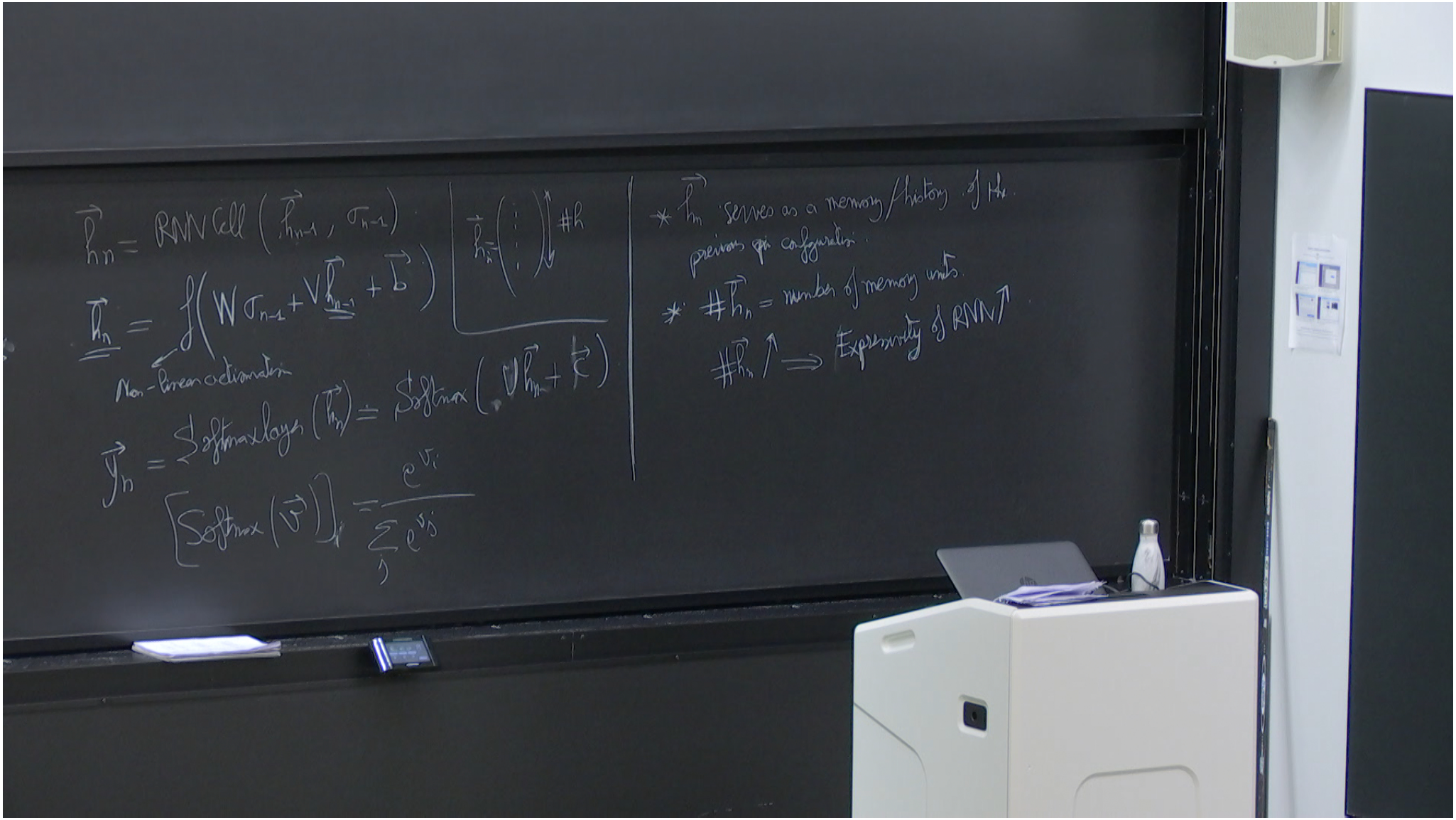
* $\sigma = (\sigma_1, \sigma_2, \cdots, \sigma_N)$. $\longleftrightarrow$ Sentence.

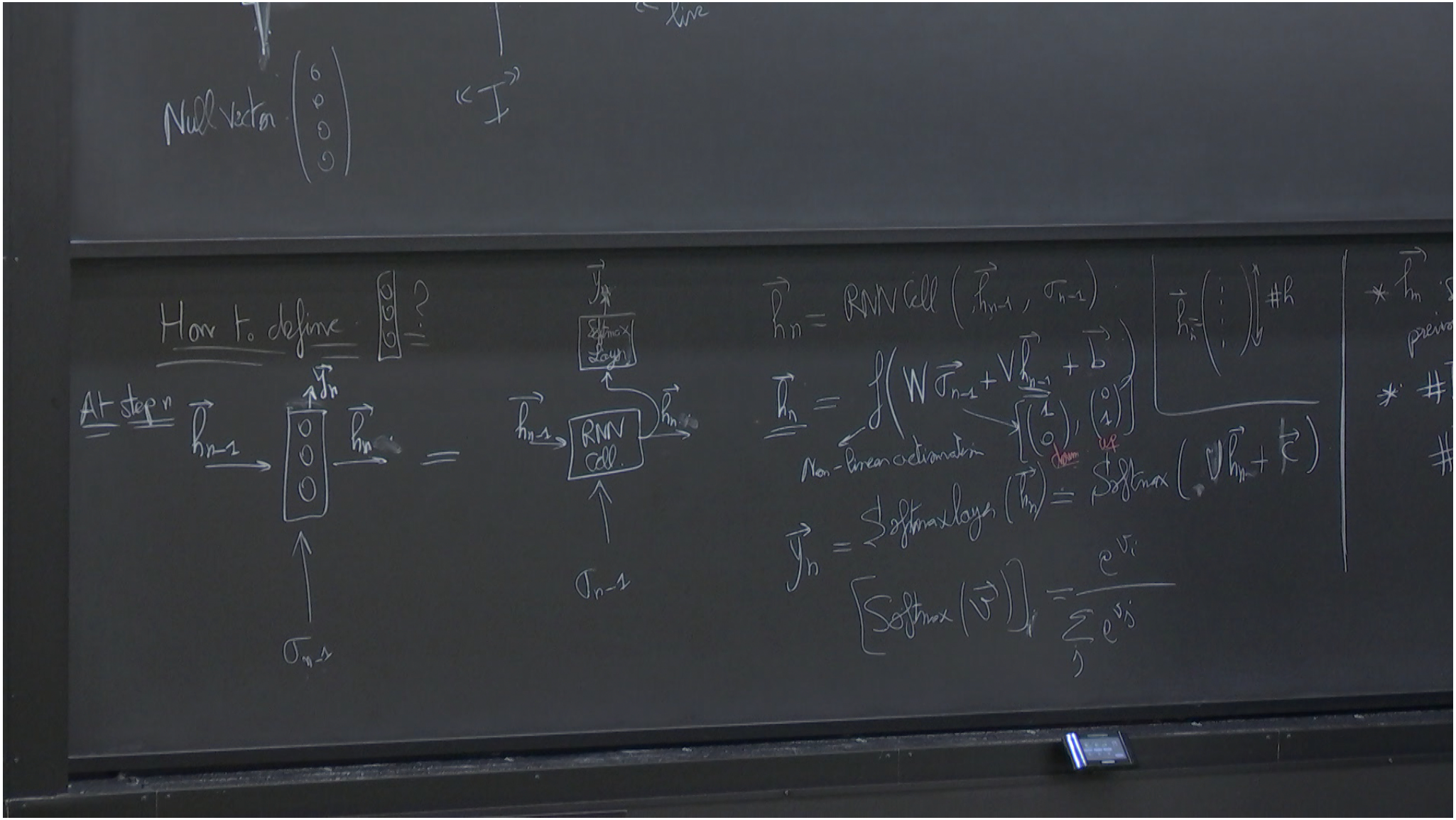
$\sigma_i =$ "$\uparrow$" or "$\downarrow$" $\Longrightarrow$ "Spin vocabulary size" $= 2$

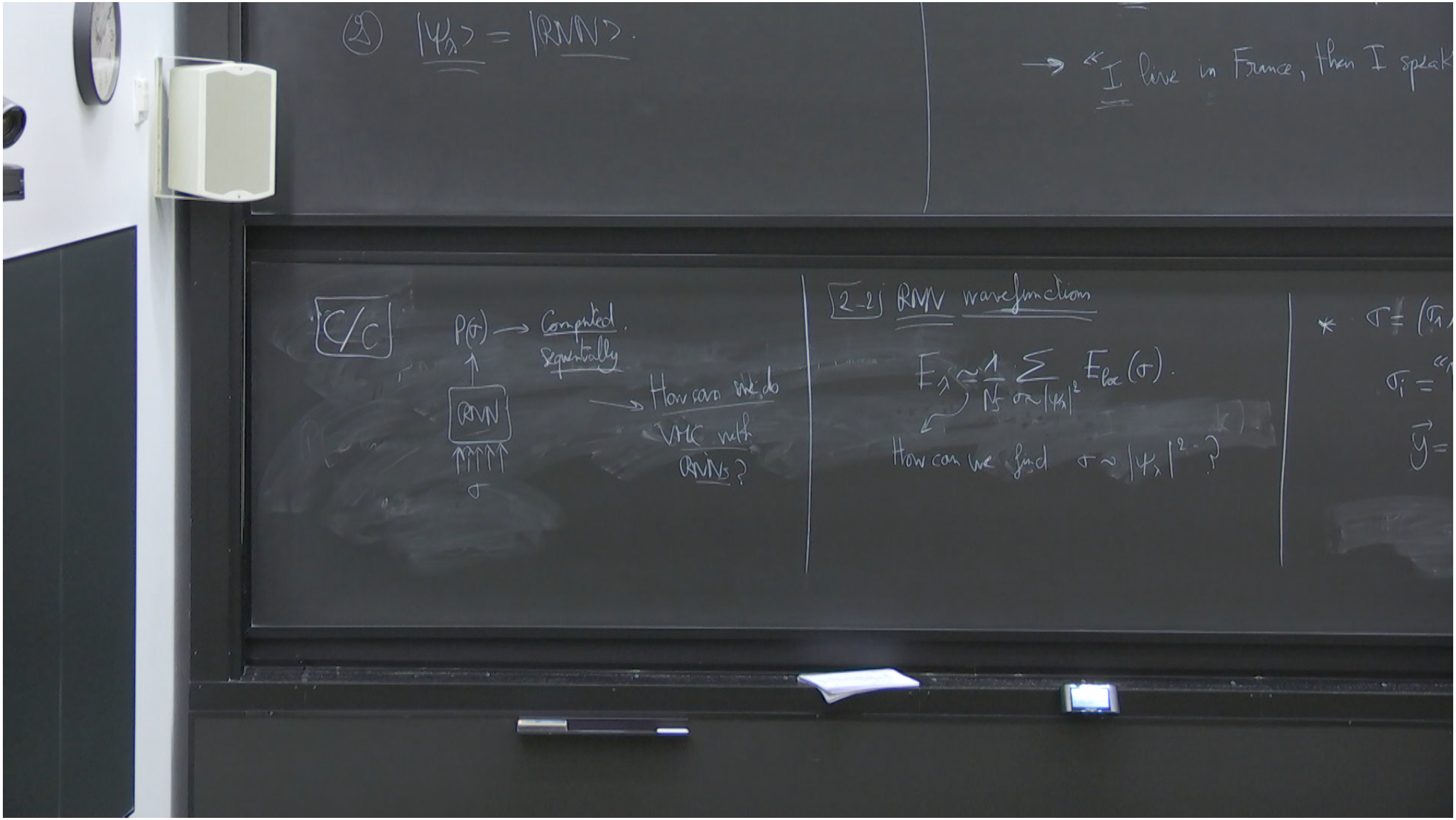
$$\vec{y} = \begin{pmatrix} p_\downarrow \\ p_\uparrow \end{pmatrix}$$

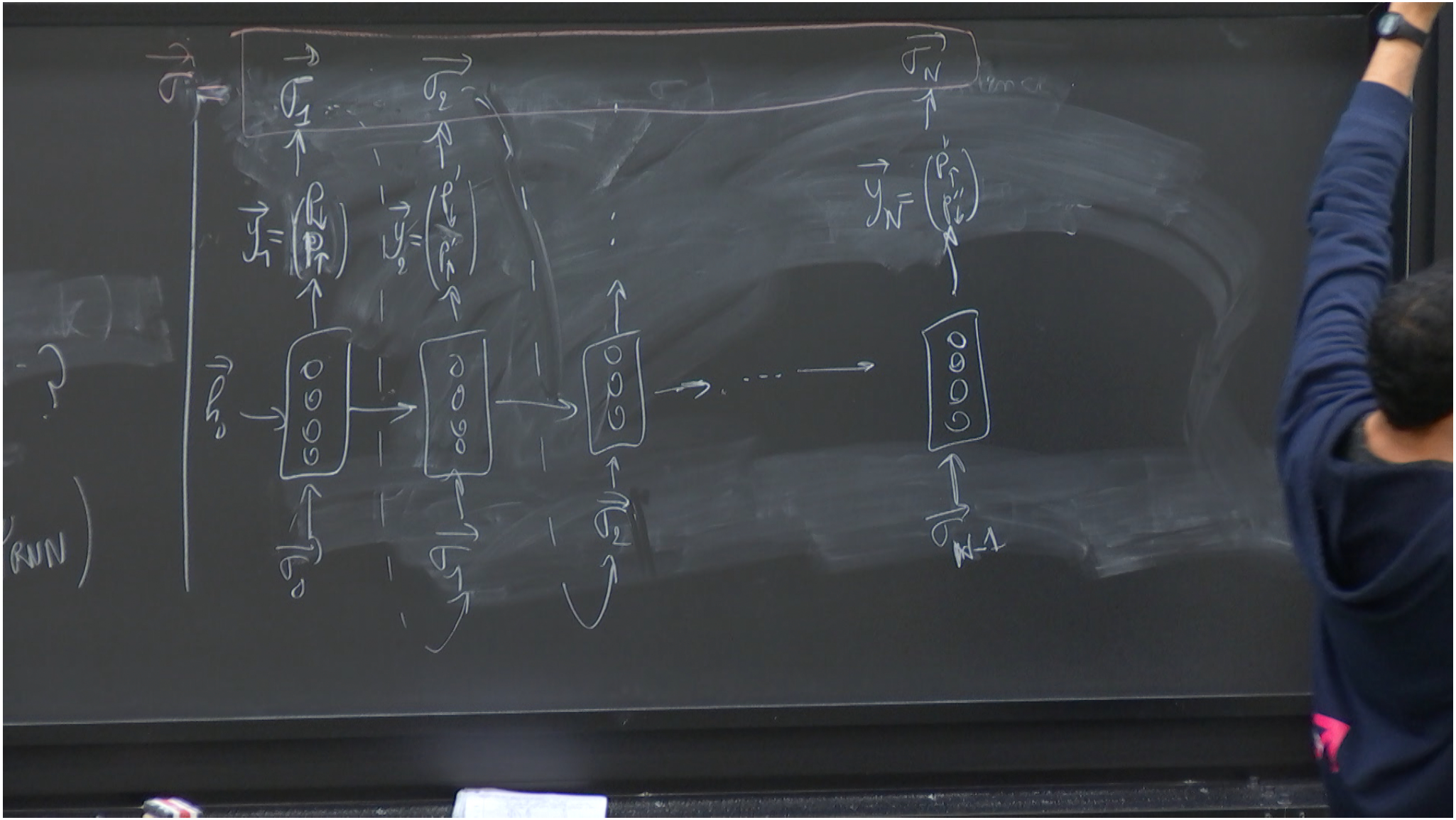
$$P(\sigma_1, \sigma_2, \cdots, \sigma_N) = P(\sigma_1)\, P(\sigma_2 | \sigma_1) \cdots P(\sigma_N | \sigma_1, \cdots, \sigma_{N-1}) .$$

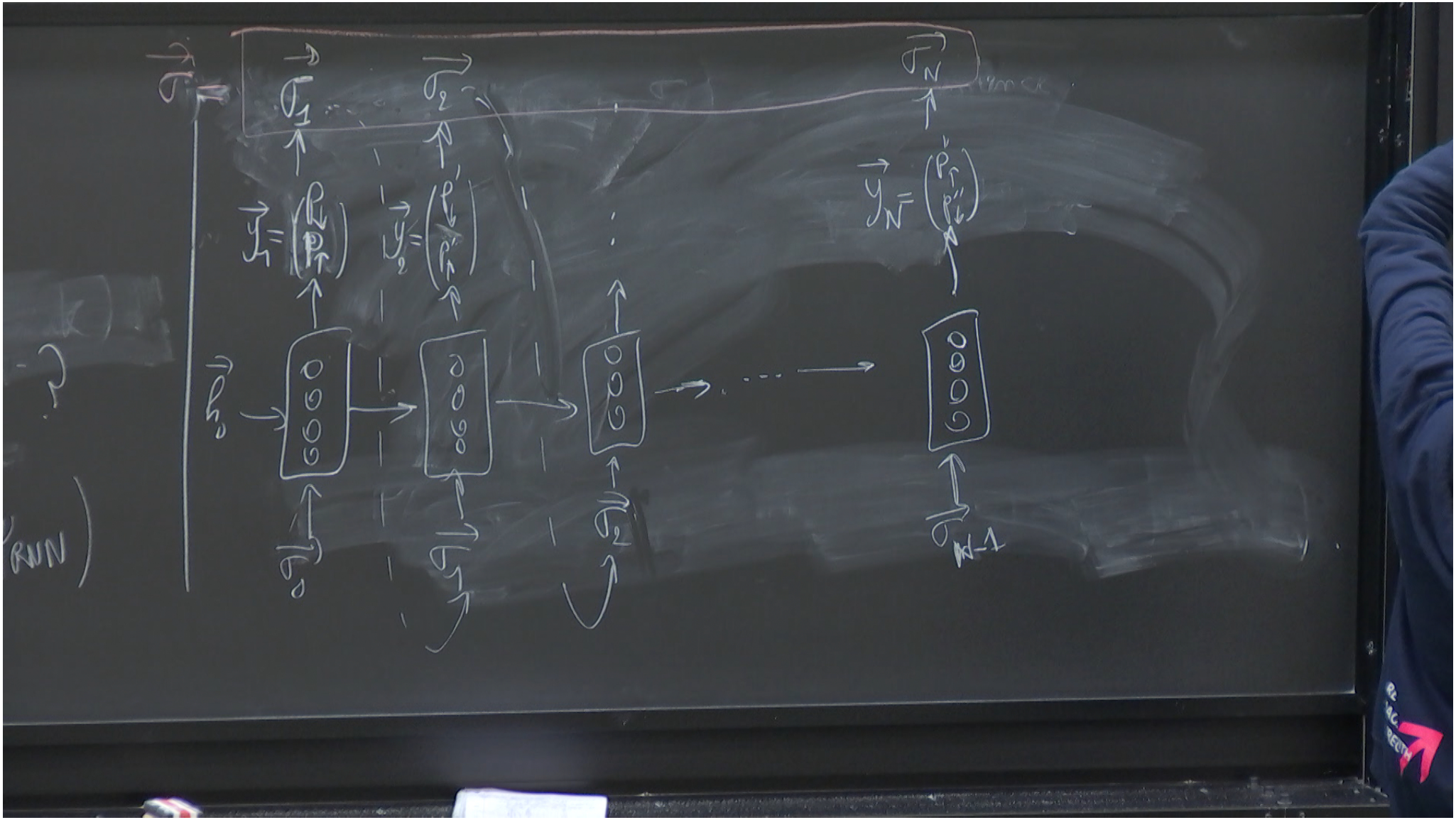

Null vector
« I »
« live »
« speak »
How to define ?
At step n
RNN cell
Softmax Layer
Non-linear activation

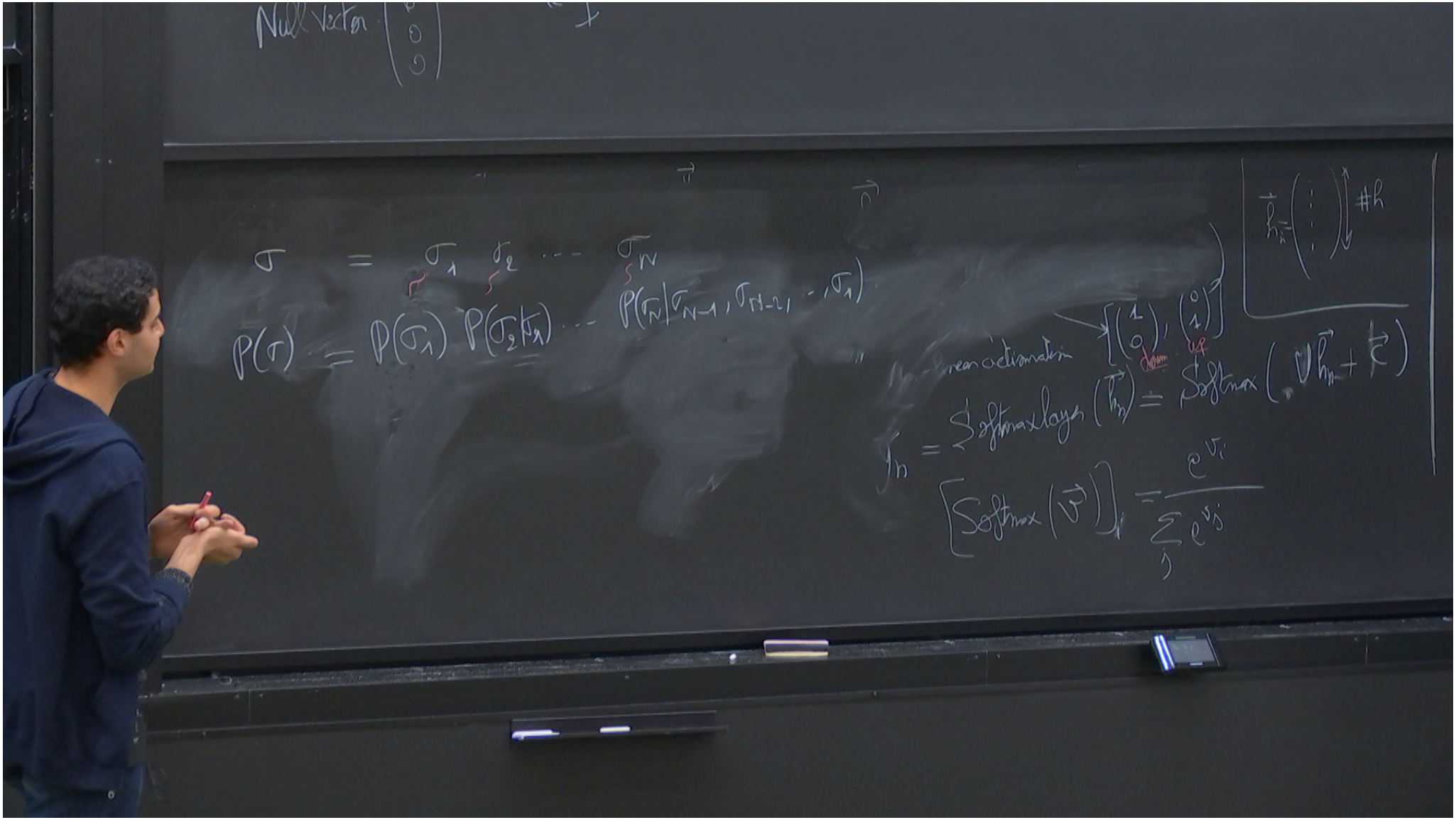

$\vec{h}_n = \text{RNN cell}(\vec{h}_{n-1}, \sigma_{n-1})$
$\vec{h}_n = f(W\sigma_{n-1} + V\vec{h}_{n-1} + \vec{b})$
Non-linear activation
$\vec{h}_n = \begin{pmatrix} \vdots \end{pmatrix} \} \#h$
$\vec{y}_n = \text{Softmax layer}(\vec{h}_n) = \text{Softmax}(U\vec{h}_n + \vec{c})$
$[\text{Softmax}(\vec{v})]_i = \frac{e^{v_i}}{\sum_j e^{v_j}}$
* $\vec{h}_n$ serves as a memory/history of the previous qm configuration.
* $\#\vec{h}_n$ = number of memory units.
$\#\vec{h}_n \nearrow \Longrightarrow$ Expressivity of RNN $\nearrow$

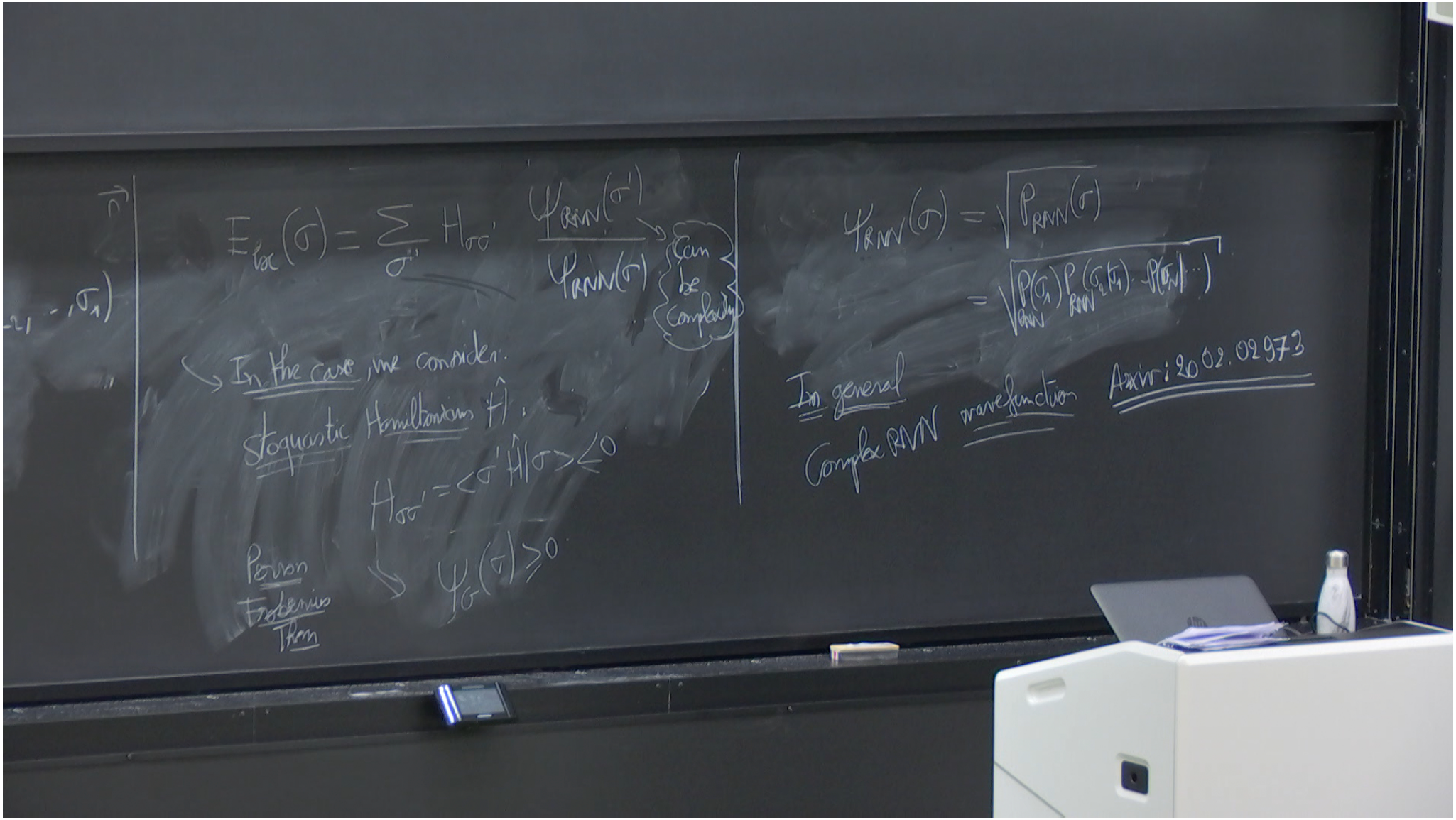

Null vector
"I"
How to define ?
At step n
RNN Cell
Softmax Layer
Non-linear activation
Softmax layer
Softmax

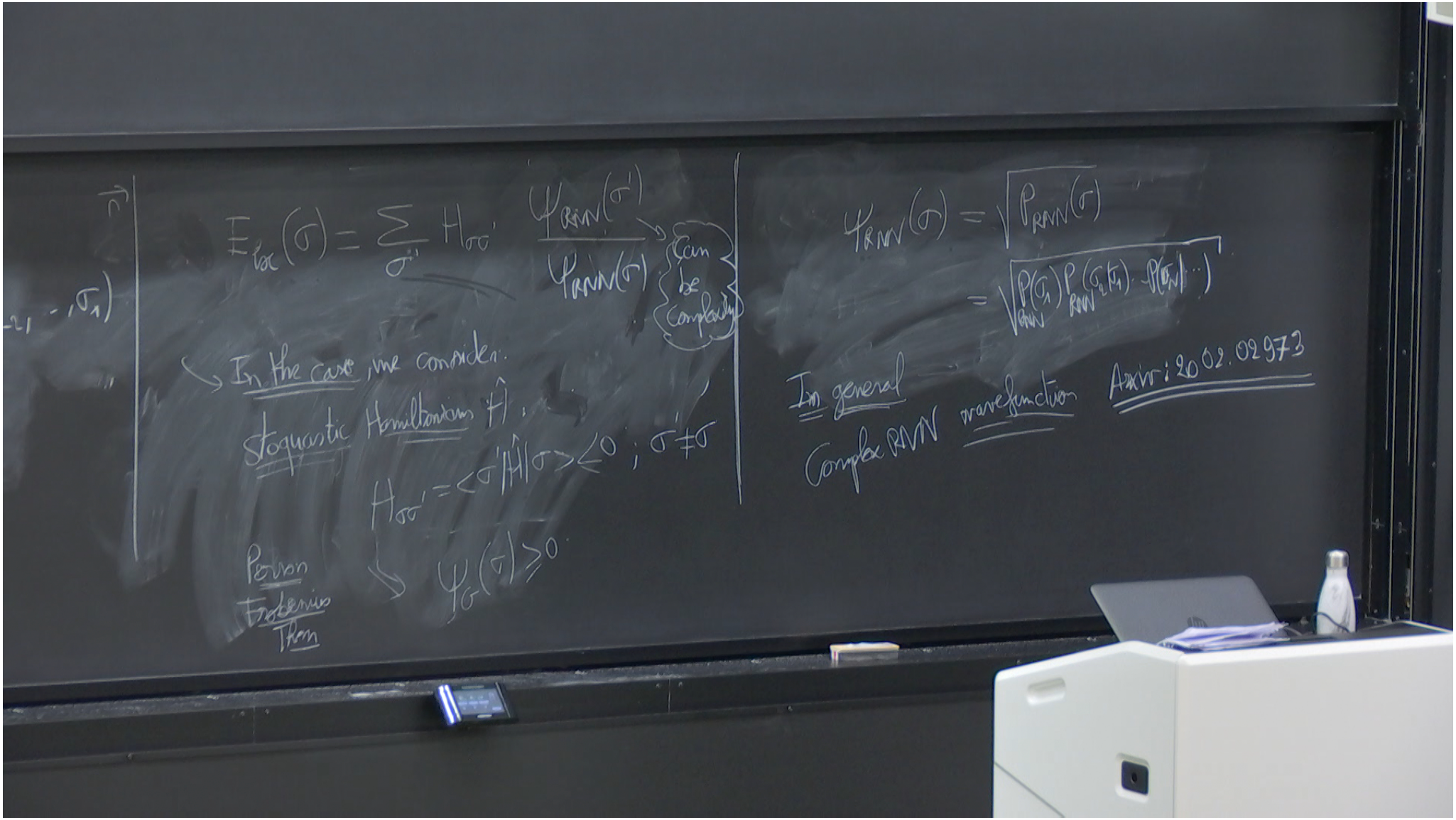

C/C
P(σ) → Computed sequentially
RNN
How can we do VMC with RNNs ?
2-2 RNN wavefunctions
How can we find σ ~ |ψ_λ|² ?
"I live in France, then I speak






Null vector
$\sigma = \sigma_1 \sigma_2 \cdots \sigma_N$
$P(\sigma) = P(\sigma_1) P(\sigma_2|\sigma_1) \cdots P(\sigma_N|\sigma_{N-1}, \sigma_{N-2}, \ldots, \sigma_1)$
Softmax layer
$\#h$


$E_{loc}(\sigma) = \sum_{\sigma'} H_{\sigma\sigma'} \frac{\psi_{RNN}(\sigma')}{\psi_{RNN}(\sigma)}$
Can be complex
In the case, we consider
Stoquastic Hamiltonians $\hat{H}$:
$H_{\sigma\sigma'} = \langle\sigma'|\hat{H}|\sigma\rangle \leq 0$
Perron Frobenius Thm
$\psi_\sigma(\sigma) \geq 0$
$\psi_{RNN}(\sigma) = \sqrt{P_{RNN}(\sigma)}$
$= \sqrt{P(\sigma_1) P(\sigma_2|\sigma_1) \cdots}$
In general
Complex RNN wavefunction
Arxiv: 2002.02973


$E_{loc}(\sigma) = \sum_{\sigma'} H_{\sigma\sigma'} \frac{\psi_{RNN}(\sigma')}{\psi_{RNN}(\sigma)}$
Can be complex
In the case, we consider
Stoquastic Hamiltonians $\hat{H}$:
$H_{\sigma\sigma'} = \langle\sigma'|\hat{H}|\sigma\rangle \leq 0 \; ; \; \sigma' \neq \sigma$
Perron Frobenius Thm
$\psi_\sigma(\sigma) \geq 0$
$\psi_{RNN}(\sigma) = \sqrt{P_{RNN}(\sigma)}$
$= \sqrt{P(\sigma_1) P(\sigma_2|\sigma_1) \cdots}$
In general
Complex RNN wavefunction
Arxiv: 2002.02973

1DTFIM - Google Drive | Demonstration_RNNWavefunc

https://colab.research.google.com/drive/1CV2IGWhIdOuFhAijYXB7 80%

Demonstration_RNNWavefunctions.ipynb

File Edit View Insert Runtime Tools Help

Comment Share M

+ Code + Text

RAM Disk Editing

```
[2]  8 rcParams['font.serif']      = ['Computer Modern']
     9 rcParams['font.size']       = 10
    10 rcParams['legend.fontsize'] = 20
    11 rcParams['xtick.labelsize'] = 20
    12 rcParams['ytick.labelsize'] = 20
```

1 - Calculating the ground state energy of 1DTFIM using Exact Diagonalization (ED)

Here, we attempt to calculate the ground state and the ground state energy of the 1D Transverse-field Ising Model with open boundary conditions using Exact Diagonalization (ED).

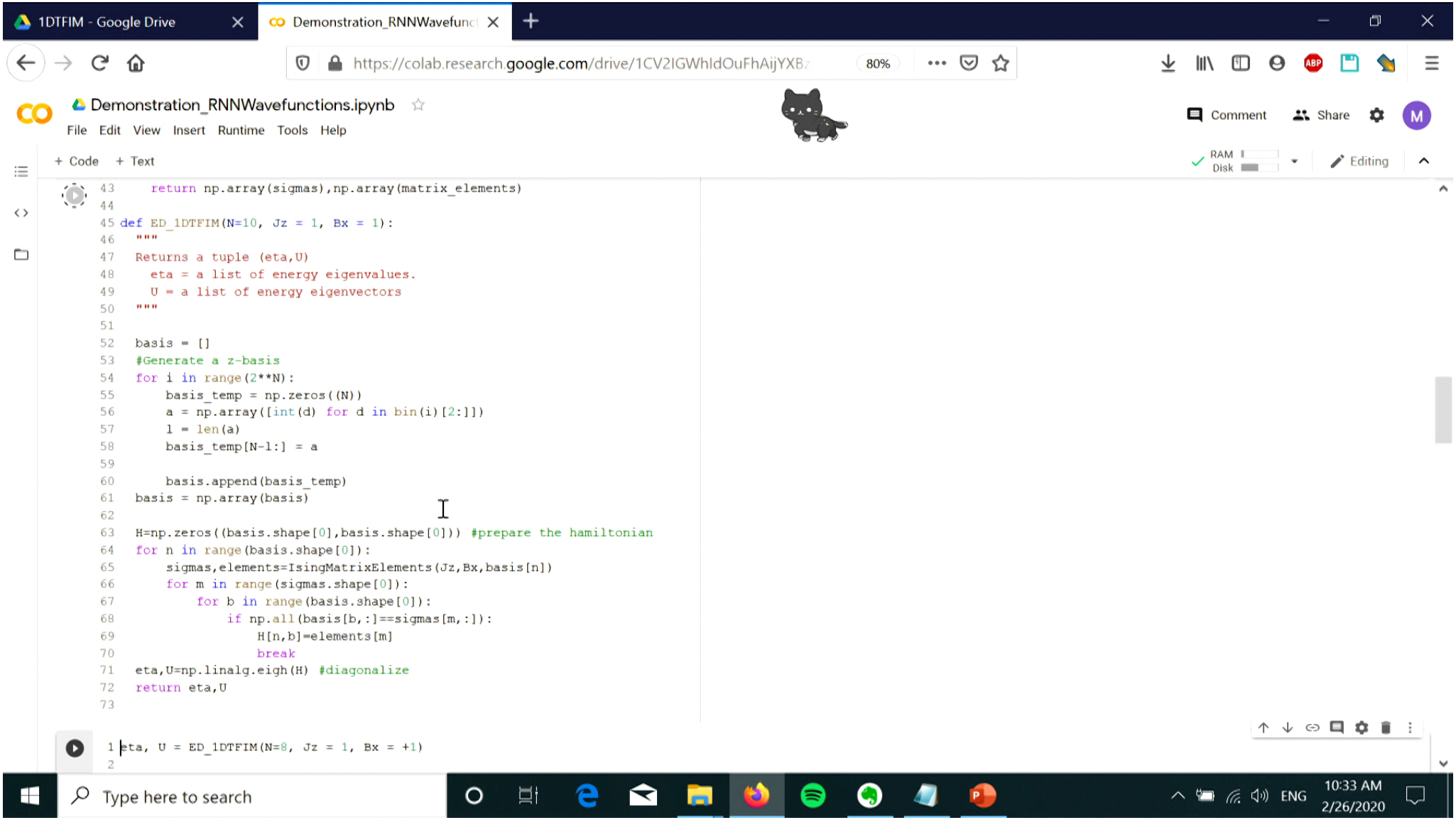
The Hamiltonian is given as follows:

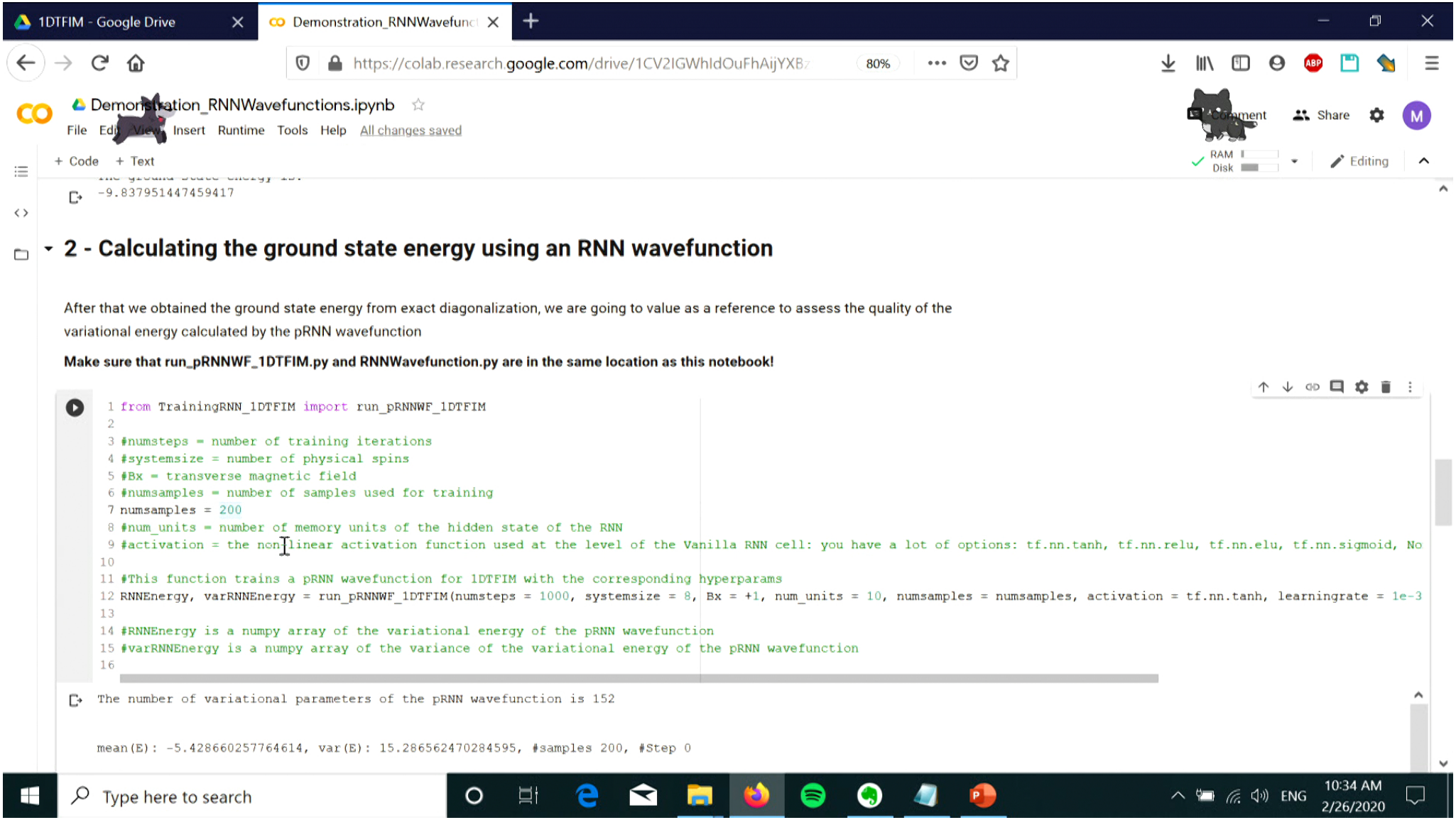
$$\hat{H}_{\text{TFIM}} = -J_z \sum_{\langle i,j \rangle} \hat{\sigma}_i^z \hat{\sigma}_j^z - B_x \sum_i \hat{\sigma}_i^x$$

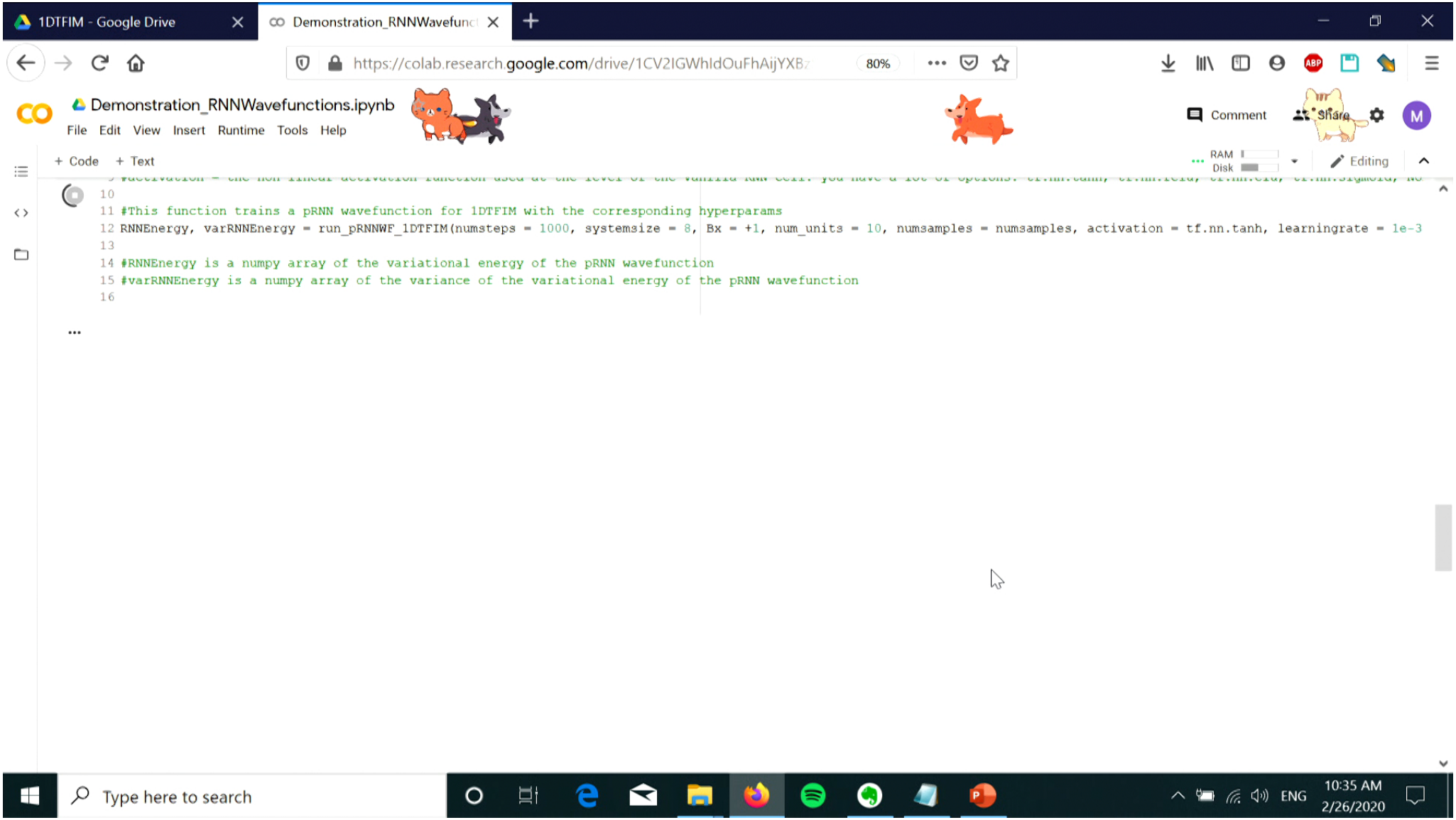
where σ_i are pauli matrices. Here, $\langle i, j \rangle$ denote nearest neighbor pairs.

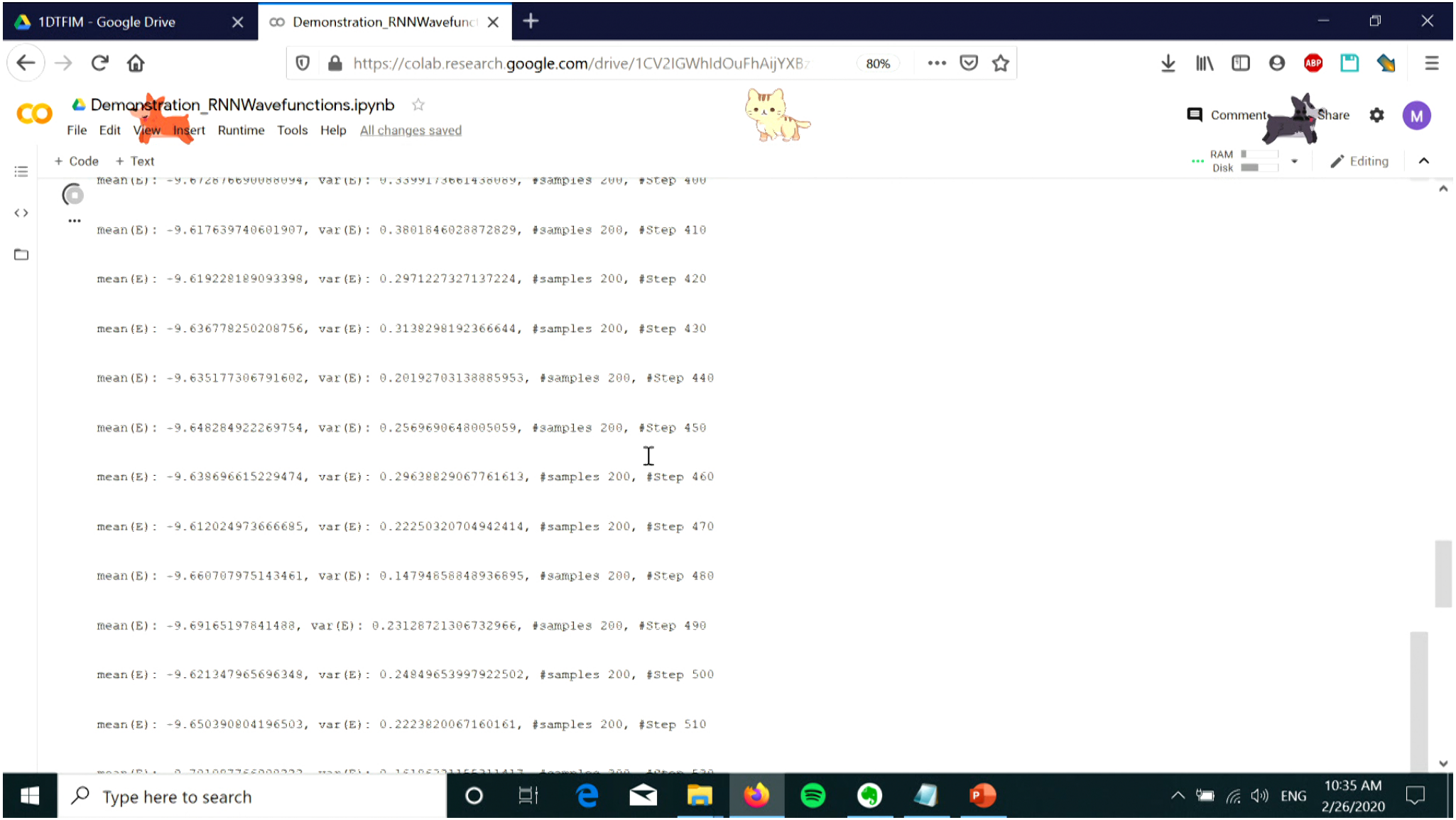
```
[ ]  1 def IsingMatrixElements(Jz,Bx,sigmap):
     2     """
     3     computes the matrix element of the open Ising Hamiltonian for a given state sigmap
     4     ---------------------------------------------------------------------------
     5     Parameters:
     6     Jz: np.ndarray of shape (N), respectively, and dtype=float:
     7                 Ising parameters
     8     sigmap:     np.ndarrray of dtype=int and shape (N)
     9                 spin-state, integer encoded (using 0 for down spin and 1 for up spin)
    10                 A sample of spins can be fed here.
    11     Bx: Transvers magnetic field (N)
    12     ---------------------------------------------------------------------------
    13     Returns: 2-tuple of type (np.ndarray,np.ndarray)
```

Type here to search ENG 10:33 AM 2/26/2020


1DTFIM - Google Drive
Demonstration_RNNWavefunc
https://colab.research.google.com/drive/1CV2IGWhIdOuFhAijYXB
80%
Demonstration_RNNWavefunctions.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
M
+ Code + Text
RAM
Disk
Editing
43 return np.array(sigmas),np.array(matrix_elements)
44
45 def ED_1DTFIM(N=10, Jz = 1, Bx = 1):
46 """
47 Returns a tuple (eta,U)
48 eta = a list of energy eigenvalues.
49 U = a list of energy eigenvectors
50 """
51
52 basis = []
53 #Generate a z-basis
54 for i in range(2**N):
55 basis_temp = np.zeros((N))
56 a = np.array([int(d) for d in bin(i)[2:]])
57 l = len(a)
58 basis_temp[N-l:] = a
59
60 basis.append(basis_temp)
61 basis = np.array(basis)
62
63 H=np.zeros((basis.shape[0],basis.shape[0])) #prepare the hamiltonian
64 for n in range(basis.shape[0]):
65 sigmas,elements=IsingMatrixElements(Jz,Bx,basis[n])
66 for m in range(sigmas.shape[0]):
67 for b in range(basis.shape[0]):
68 if np.all(basis[b,:]==sigmas[m,:]):
69 H[n,b]=elements[m]
70 break
71 eta,U=np.linalg.eigh(H) #diagonalize
72 return eta,U
73
1 eta, U = ED_1DTFIM(N=8, Jz = 1, Bx = +1)
2
Type here to search
ENG
10:33 AM
2/26/2020


1DTFIM - Google Drive
Demonstration_RNNWavefunc
https://colab.research.google.com/drive/1CV2IGWhIdOuFhAijYXB
80%
Demonstration_RNNWavefunctions.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
M
+ Code + Text
RAM
Disk
Editing
-9.837951447459417
2 - Calculating the ground state energy using an RNN wavefunction
After that we obtained the ground state energy from exact diagonalization, we are going to value as a reference to assess the quality of the variational energy calculated by the pRNN wavefunction
Make sure that run_pRNNWF_1DTFIM.py and RNNWavefunction.py are in the same location as this notebook!
1 from TrainingRNN_1DTFIM import run_pRNNWF_1DTFIM
2
3 #numsteps = number of training iterations
4 #systemsize = number of physical spins
5 #Bx = transverse magnetic field
6 #numsamples = number of samples used for training
7 numsamples = 200
8 #num_units = number of memory units of the hidden state of the RNN
9 #activation = the non-linear activation function used at the level of the Vanilla RNN cell: you have a lot of options: tf.nn.tanh, tf.nn.relu, tf.nn.elu, tf.nn.sigmoid, No
10
11 #This function trains a pRNN wavefunction for 1DTFIM with the corresponding hyperparams
12 RNNEnergy, varRNNEnergy = run_pRNNWF_1DTFIM(numsteps = 1000, systemsize = 8, Bx = +1, num_units = 10, numsamples = numsamples, activation = tf.nn.tanh, learningrate = 1e-3
13
14 #RNNEnergy is a numpy array of the variational energy of the pRNN wavefunction
15 #varRNNEnergy is a numpy array of the variance of the variational energy of the pRNN wavefunction
16
The number of variational parameters of the pRNN wavefunction is 152
mean(E): -5.428660257764614, var(E): 15.286562470284595, #samples 200, #Step 0
Type here to search
ENG
10:34 AM
2/26/2020


1DTFIM - Google Drive
Demonstration_RNNWavefunc
https://colab.research.google.com/drive/1CV2IGWhIdOuFhAijYXB
80%
Demonstration_RNNWavefunctions.ipynb
File Edit View Insert Runtime Tools Help
Comment
Share
M
+ Code + Text
RAM
Disk
Editing
#activation = the non linear activation function used at the level of the vanilla RNN cell. you have a lot of options: tf.nn.tanh, tf.nn.relu, tf.nn.sigmoid, etc
#This function trains a pRNN wavefunction for 1DTFIM with the corresponding hyperparams
RNNEnergy, varRNNEnergy = run_pRNNWF_1DTFIM(numsteps = 1000, systemsize = 8, Bx = +1, num_units = 10, numsamples = numsamples, activation = tf.nn.tanh, learningrate = 1e-3
#RNNEnergy is a numpy array of the variational energy of the pRNN wavefunction
#varRNNEnergy is a numpy array of the variance of the variational energy of the pRNN wavefunction
Type here to search
ENG
10:35 AM
2/26/2020


1DTFIM - Google Drive
Demonstration_RNNWavefunc
https://colab.research.google.com/drive/1CV2IGWhIdOuFhAijYXB
80%
Demonstration_RNNWavefunctions.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
M
+ Code + Text
RAM
Disk
Editing
mean(E): -9.672676690088094, var(E): 0.33991736614360689, #samples 200, #Step 400
mean(E): -9.617639740601907, var(E): 0.3801846028872829, #samples 200, #Step 410
mean(E): -9.619228189093398, var(E): 0.2971227327137224, #samples 200, #Step 420
mean(E): -9.636778250208756, var(E): 0.3138298192366644, #samples 200, #Step 430
mean(E): -9.635177306791602, var(E): 0.20192703138885953, #samples 200, #Step 440
mean(E): -9.648284922269754, var(E): 0.2569690648005059, #samples 200, #Step 450
mean(E): -9.638696615229474, var(E): 0.296308290677761613, #samples 200, #Step 460
mean(E): -9.612024973666685, var(E): 0.22250320704942414, #samples 200, #Step 470
mean(E): -9.660707975143461, var(E): 0.14794858848936895, #samples 200, #Step 480
mean(E): -9.69165197841488, var(E): 0.23128721306732966, #samples 200, #Step 490
mean(E): -9.621247965696348, var(E): 0.24849653997922502, #samples 200, #Step 500
mean(E): -9.650390804196503, var(E): 0.2223820067160161, #samples 200, #Step 510
Type here to search
ENG
10:35 AM
2/26/2020

1DTFIM - Google Drive | Demonstration_RNNWavefunc

https://colab.research.google.com/drive/1CV2IGWhIdOuFhAijYXB7 80%

Demonstration_RNNWavefunctions.ipynb

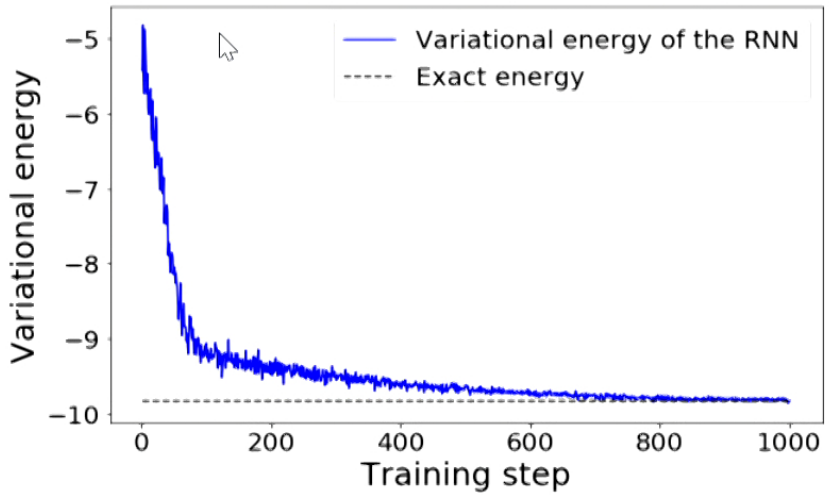
File Edit View Insert Runtime Tools Help All changes saved

Comment Share M

RAM Disk Editing

+ Code + Text

```
[6]  2
     3 ax.plot(np.arange(1, len(RNNEnergy)+1), RNNEnergy, "b-", label="Variational energy of the RNN")
     4 ax.plot(np.arange(1, len(RNNEnergy)+1), [E_exact]*len(RNNEnergy), "k--", label="Exact energy")
     5
     6 ax.set_xlabel(r'Training step', fontsize = 25)
     7 ax.set_ylabel('Variational energy', fontsize = 25)
     8 plt.legend()
     9
    10 plt.show()
```



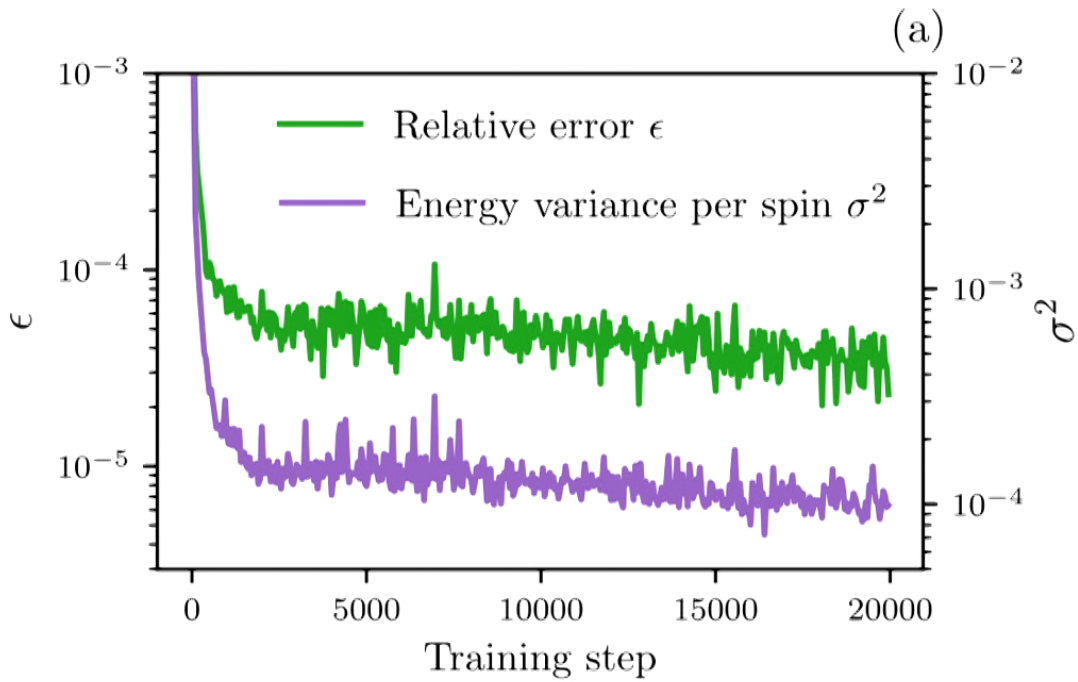

```
1
```

Type here to search ENG 10:36 AM 2/26/2020

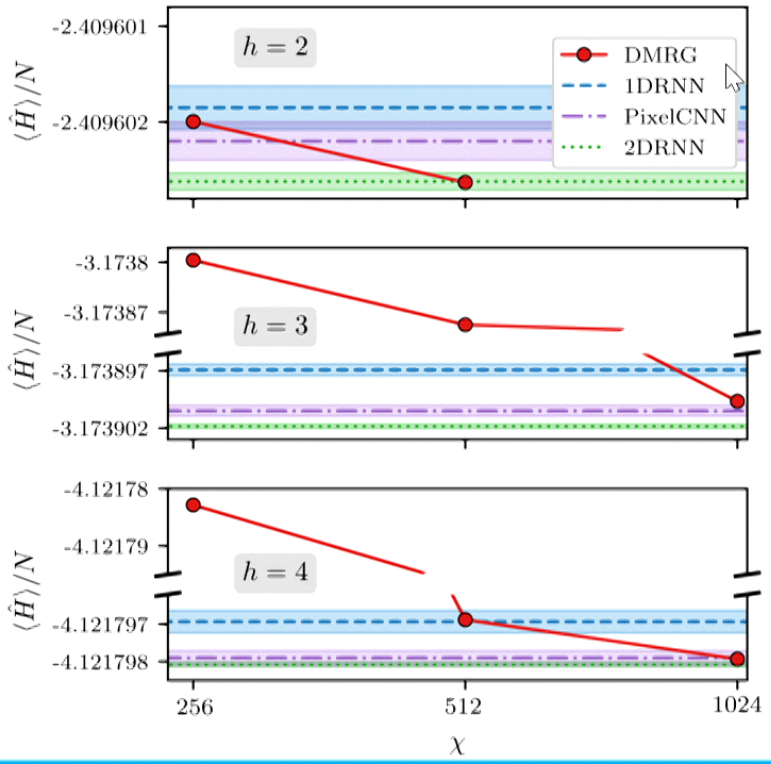
1D TFIM at the critical point (N = 1000 spins)

$$\epsilon = \left| \frac{E_{RNN} - E_{Exact}}{E_{Exact}} \right|$$

$$\sigma^2 = \frac{var(E_{loc})}{N}$$



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



TFIM in 2D

$$\hat{H}_{\mathrm{TFIM}} = -\sum_{\langle i,j \rangle} \hat{\sigma}_i^z \hat{\sigma}_j^z - h \sum_i \hat{\sigma}_i^x$$

Future directions

- Ground state of Frustrated Systems in 2D.
- Ground state of Molecular Systems.
- First excited states.
- Using more advanced architectures (LSTMs, Transformers,...).
- Solving optimization problems.
- ...