Title: Transformers

Speakers: Mohamed Hibat Allah

Collection: Navigating Quantum and AI Career Trajectories: A Beginnerâ€™s Mini-Course on Computational Methods and their Applications

Date: May 24, 2024 - 9:30 AM

URL: https://pirsa.org/24050087

Lecture 3: Transformers

Energy-based model (Intractable)

Last time:

"Generative modeling"

Restricted Boltzmann Machines (RBM)

Recurrent Neural Networks (RNN) } Auto… (Trac…

Transformers

Other architectures (Not covered in this mini-cour…

Outline:

1. Attention mechanism
2. Transformer architecture
3. Transformer Wave function

① Attention mechanism:

Most famous paper:

"Attention is All you need"

Vaswani et. al (2017)

* Why it's interesting?

→ Enabled state of the art results in Natural Language processing (NLP)

→ Transformer is the basic building block of ChatGPT, GPT4, GPT4o

GPT = Generative Pre-trained Transformer.

based on the "attention" mechanism

→ Other architectures (Not covered in this ...)

$$\vec{X} = (x_1, x_2, \ldots, x_N)$$

$x_i \in 0, 1$

Query: $[d_k]$ $\vec{q}_i \equiv \vec{x}_i \, W^{(Q)}$

Key: $\vec{K}_i \equiv \vec{x}_i \, W^{(K)}$ $\rightarrow [d_e \times d_k]$

Value: $\vec{V}_i \equiv \vec{x}_i \, W^{(V)}$

Embedding of x_i: $\vec{x}_i = \begin{pmatrix} \vdots \end{pmatrix} \Big\} d_e$

"↓" $\rightarrow \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

"↑" $\rightarrow \begin{pmatrix} 0 \\ 1 \end{pmatrix}$

Example of Youtube

↳ Query: Text on search bar.

↳ Key: Videos, titles & descriptions

↳ Value: Best video matches.

→ We can use $\vec{q}_i, \vec{K}_i, \vec{v}_i$ to compute self-attention.

$$\vec{A}_i = \sum_{j=1}^{N} \underbrace{\alpha\left(\vec{q}_i , \vec{K}_j^{\dagger}\right)}_{\text{“Correlation between site “}i\text{” and “}j\text{””}} \cdot \vec{v}_j$$

Solves the vanishing gradient problem in RNNs

Attention that has information about the most relevant « x_j »$_{j=1}^{N}$ on “x_i”

“Attention of x_i on x_j”

d_e

attention mechanism

« I live in France, hence I speak French »

i

$$\alpha(\vec{q}_i, K_j^T) = \mathrm{Softmax}\left(\frac{\vec{q}_i \cdot \vec{K}^T}{\sqrt{d_k}}\right)_j$$

$$\begin{pmatrix} \vec{K}_1 \\ \vec{K}_2 \\ \vdots \\ \vec{K}_N \end{pmatrix}$$

For normalization purposes

$$\sum_{j=1}^{N} \alpha(\vec{q}_i, \vec{K}_j^T) = 1$$

$$\vec{q}_i \cdot \vec{K}_j \in \mathbb{R}$$

$$\vec{q}_i \cdot K = \begin{pmatrix} \vdots \\ \vdots \end{pmatrix} \updownarrow N$$

mechanism

“I live in France, hence I speak French”

i

$$\sum_{j=1}^{N} \alpha(\vec{q}_i, \vec{K}_j^{\dagger}) = 1$$

$$\begin{pmatrix} \vec{K}_1 \\ \vec{K}_2 \\ \vdots \\ \vec{K}_N \end{pmatrix}$$

$$\vec{q}_i \cdot \vec{K}_j \in \mathbb{R}$$

$$\alpha(\vec{q}_i, K_j^{\dagger}) = \mathrm{Softmax}\left(\frac{\vec{q}_i \cdot \vec{K}^{\dagger}}{\sqrt{d_K}}\right)_j$$

$$\vec{q}_i \cdot \vec{K} = \begin{pmatrix} \vdots \\ \vdots \end{pmatrix} \Big\updownarrow N$$

For normalization purposes.

For normalization purposes

$$\vec{A}_i = \sum_{j=1}^{N} \mathrm{Softmax}\left(\frac{\vec{q}_i \cdot K^T}{\sqrt{d_k}}\right)_j \vec{v}_j$$

$[d_k]$

$$A = \underbrace{\mathrm{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)}_{N \times N} \cdot V$$

$[N \times d_k]$

Takes advantage of GPU parallelization power

↳ Value : Best video matches.

* Positional encoding

$$\vec{X} = \overset{x_1}{\text{I}}\text{'}\overset{x_2}{\text{m}}\ \overset{x_3}{\text{not}}\ \overset{x_4}{\text{sad}},\ \overset{x_5}{\text{I}}\text{'m}\ \overset{x_6}{\text{happy}}.$$

$$\vec{X} = \text{I'm not happy, I'm sad}$$

Idea $\quad \vec{x}_i + \underbrace{\vec{PE}(i)}_{[d_e]} \longrightarrow \vec{x}_i'$

on "x_i"

$$\vec{PE}(i)_{2j} = \sin\left(\frac{i}{10000^{2j/d_e}}\right)$$

$$\vec{PE}(i)_{2j+1} = \cos\left(\frac{i}{\underbrace{10000}_{\text{Max sequence length}}{}^{2j/d_e}}\right)$$

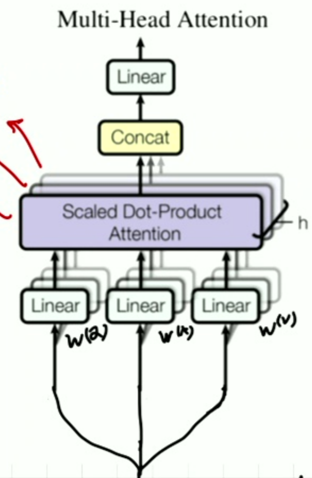
↳ Nearby positions have similar PE at particular component "k"

↳ Each component of the PE has a different wavelength → Short - long range dependencies.

* **Multi-head Attention:** allows to learn attention about different features

A^μ
Q^μ
K^μ
V^μ
← New index $\mu = 1 \to h$

A^1 A^2 A^3



different self-attentions

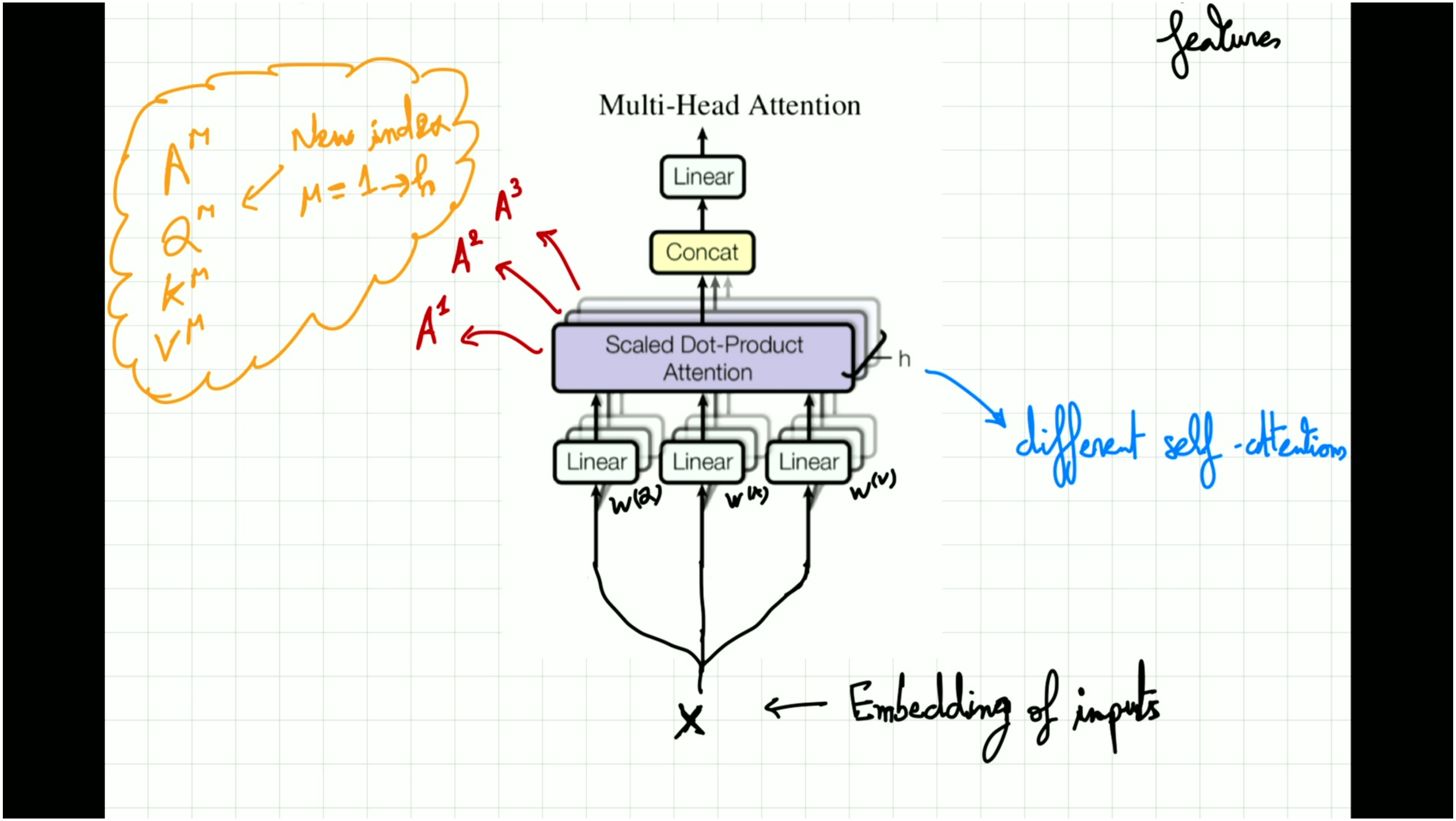
X ← Embedding of inputs

Example: Group of people ⟶ Character, Job, Age, Gender...

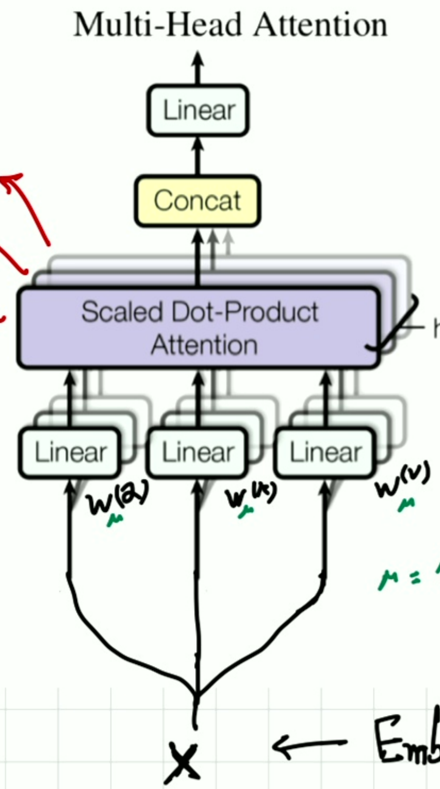
Spins ⟶ Correlation, magnetization, ...

In equation:

$$\text{Multi head Attention} = \text{Concat}(A^1, A^2, \dots, A^h)\, W^{(O)},$$


features
Multi-Head Attention
Linear
Concat
Scaled Dot-Product Attention
h
Linear
Linear
Linear
$W^{(Q)}$
$W^{(K)}$
$W^{(V)}$
A^1
A^2
A^3
A^μ
Q^μ
K^μ
V^μ
New index
$\mu = 1 \to h$
different self-attentions
X
Embedding of inputs

A^μ, Q^μ, K^μ, V^μ ← New index $\mu = 1 \to h$



A^1, A^2, A^3

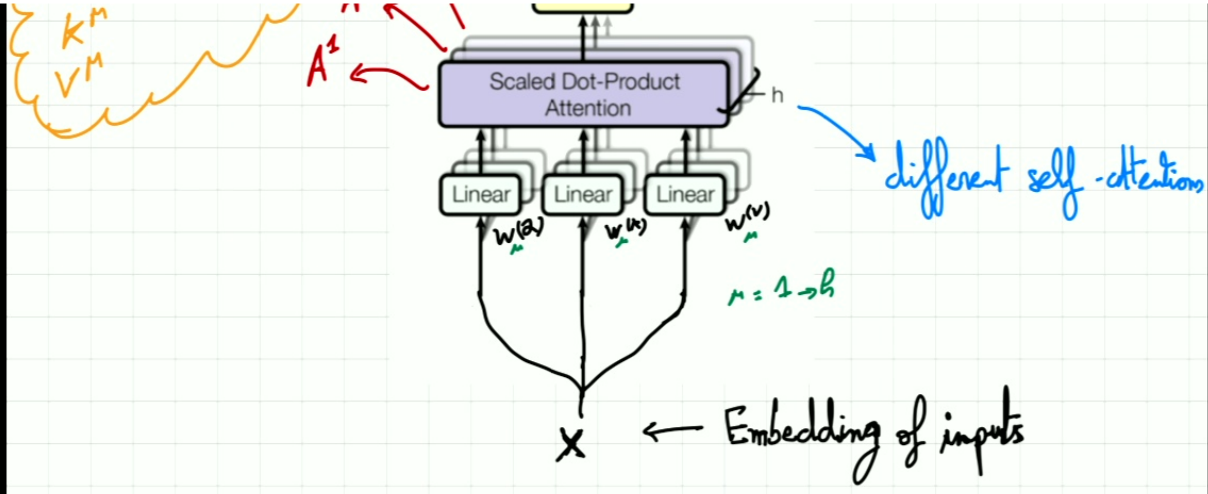
different self-attentions

$W^{(Q)}_\mu$ $W^{(K)}_\mu$ $W^{(V)}_\mu$

$\mu = 1 \to h$

X ← Embedding of inputs

Example: Group of people ⟶ Character, Job, Age, Gender...



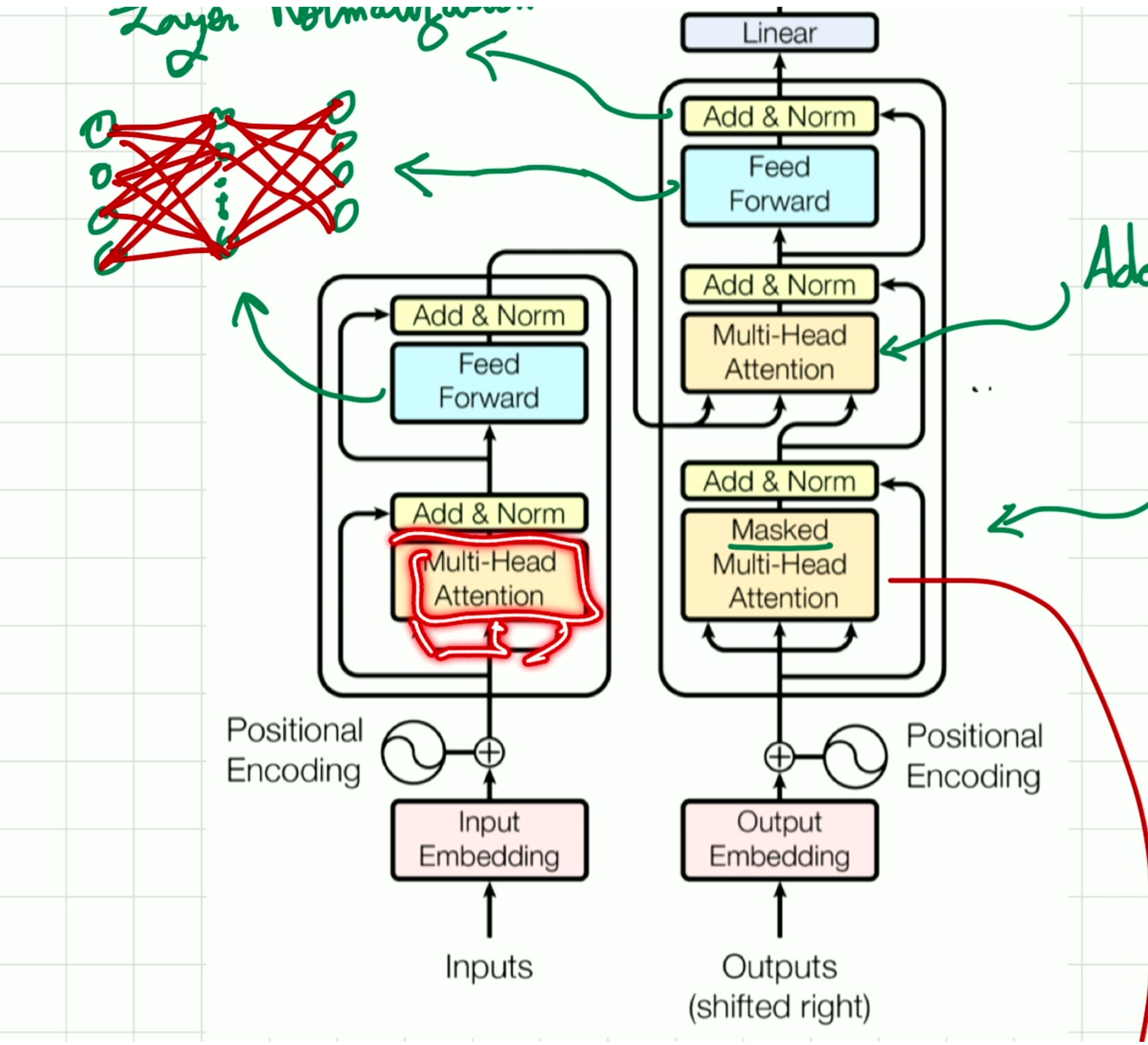
Example: Group of people ⟶ Character, Job, Age, Gender...

Spins ⟶ Correlation, magnetization, ...

In equation:

$$\text{Multi head Attention} = \text{Concat}\left(A^1, A^2, \ldots, A^h\right) W^{(0)}$$

Learnable weights



probability

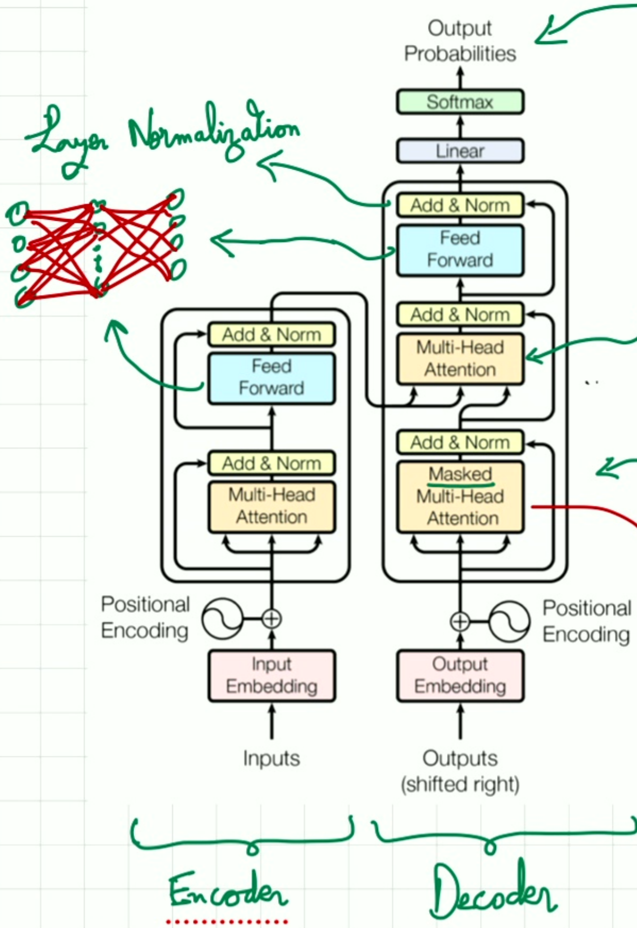
Adding Attention on inputs Y

At step i

Masking

$x_1, x_2, \ldots, x_i, \cancel{x_{i+1}}, \ldots, \cancel{x_N}$

$$\vec{A}_i = \sum_{j<i} \alpha\left(\vec{q}_i, \vec{k}_j^{\,t}\right) \cdot \vec{v}_j$$



Conditional probability

Input

Adding Attention on inputs Y

At step i

Masking

$$x_1, x_2, \dots, x_i, \cancel{x_{i+1}}, \dots, \cancel{x_N}$$

$$\vec{A}_i = \sum_{j<i} \alpha\left(\vec{q}_i, \vec{k}_j^{\,t}\right) \cdot \vec{v}_j$$

|GPT-3|

175 B parameters

$W^{(Q)}, W^{(K)}, W^{(V)}$

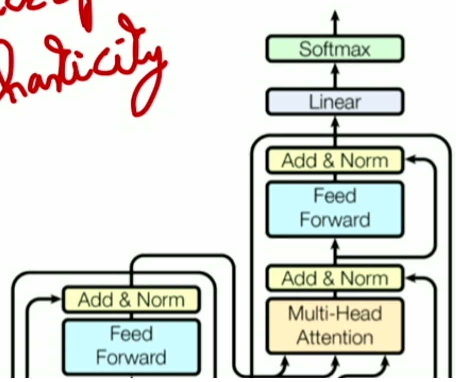
$W^{(O)}, \ldots$

96 Transformer layers

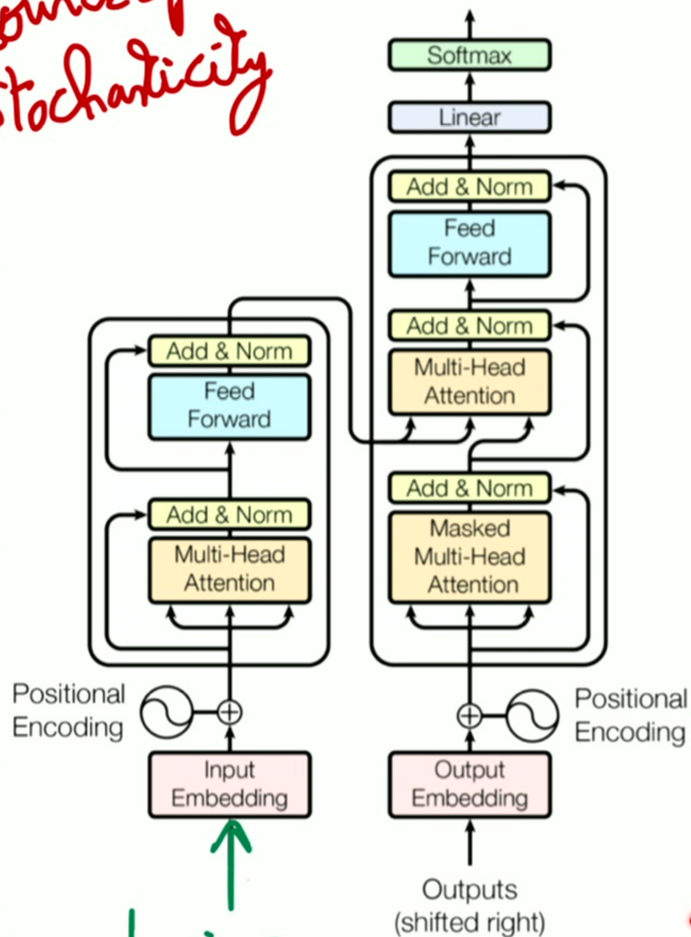
Training using massive Dataset D ~ 300 B words

Human feedback (Reward) to fine tune the parameters

Source of Stochasticity ← P(Next word | X)



Source of Stochasticity



Sentence beginning

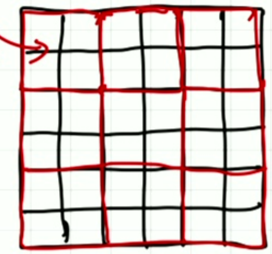
Human feedback (Reward) to fine tune the parameters

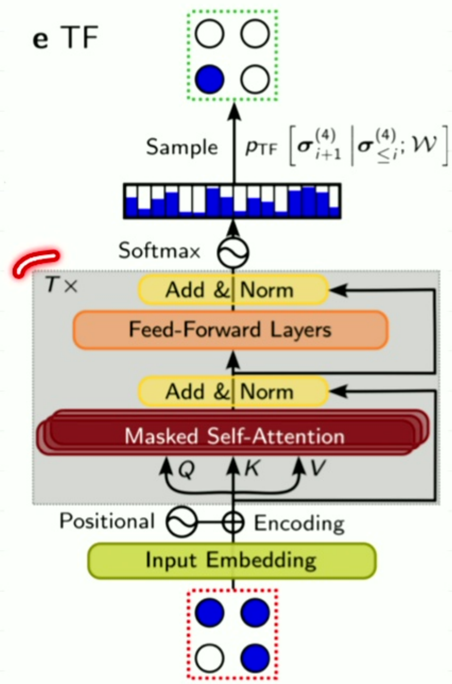
⚠ Training cost: ~ 4.6 million $ to train.

550 tons of CO_2 → Climate Change

(3) Transformer Wavefunctions

Patch
eg: 4 spins





$$\Psi_{TF}(\sigma_1, \sigma_2, \dots, \sigma_N)$$

$$= \sqrt{P_{TF}(\sigma_1, \sigma_2, \dots, \sigma_N)}$$

↳ For Stoquastic Hamiltonians

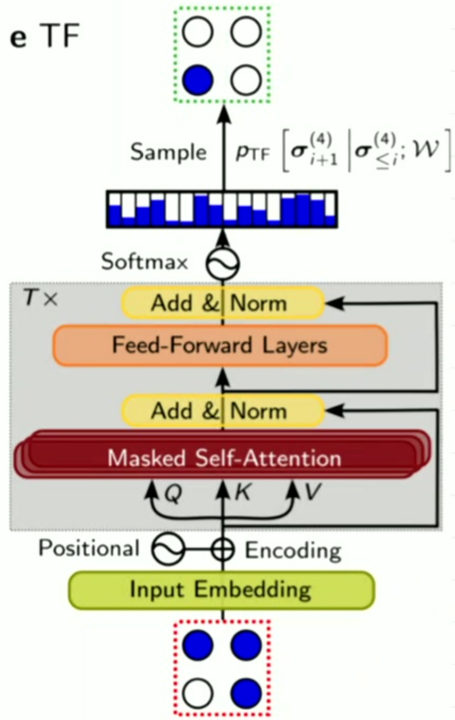
Cost function:

$$E = \langle \Psi_{TF} | \hat{H} | \Psi_{TF} \rangle$$

K. Sprague & S. Czischek, Comms Phys 2024

(3) Transformer Wavefunctions

Patch
eg: 4 spins



$$\Psi_{TF}(\sigma_1, \sigma_2, \dots, \sigma_N)$$

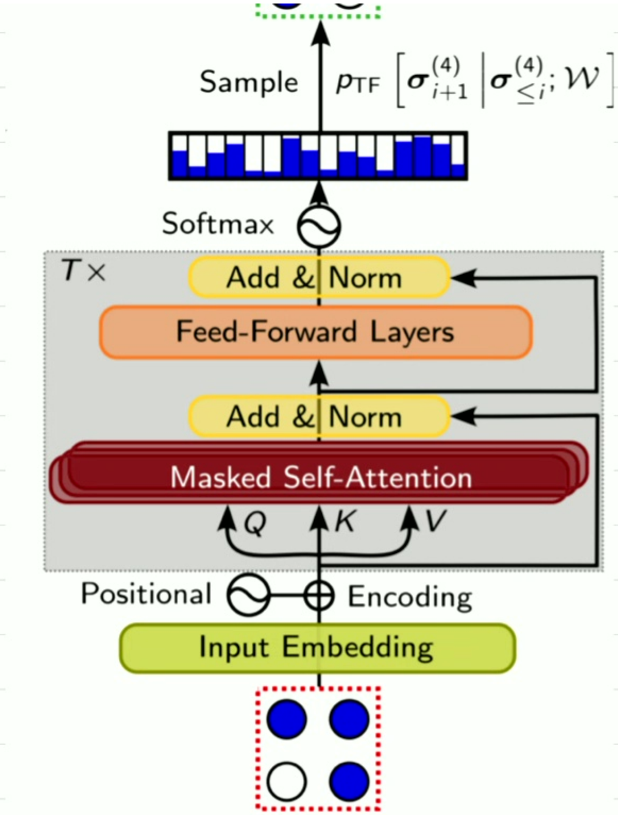
$$= \sqrt{P_{TF}(\sigma_1, \sigma_2, \dots, \sigma_N)}$$

For Stoquastic Hamiltonians

Cost Function:

$$E = \langle \Psi_{TF} | \hat{H} | \Psi_{TF} \rangle$$

K. Sprague & S. Czischek, Comms Phys 2024



$$\Psi_{TF}(\sigma_1, \sigma_2, \dots, \sigma_N) = \sqrt{P_{TF}(\sigma_1, \sigma_2, \dots, \sigma_N)}$$

For Stoquastic Hamiltonians

Cost function:

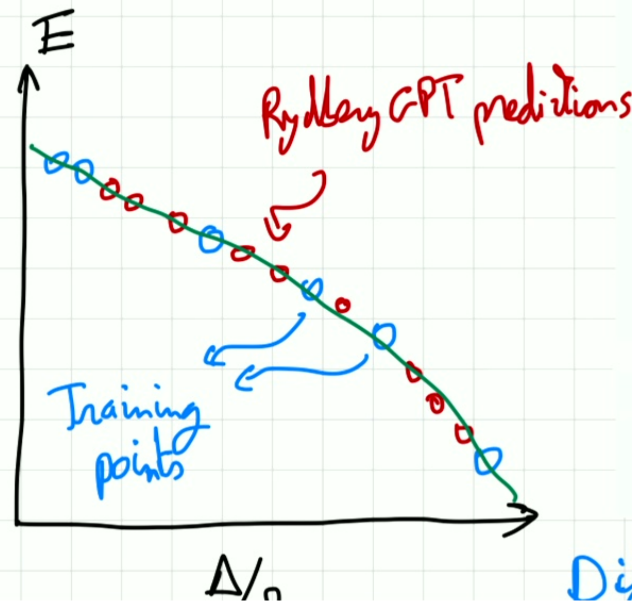
$$E = \langle \Psi_{TF} | \hat{H} | \Psi_{TF} \rangle$$

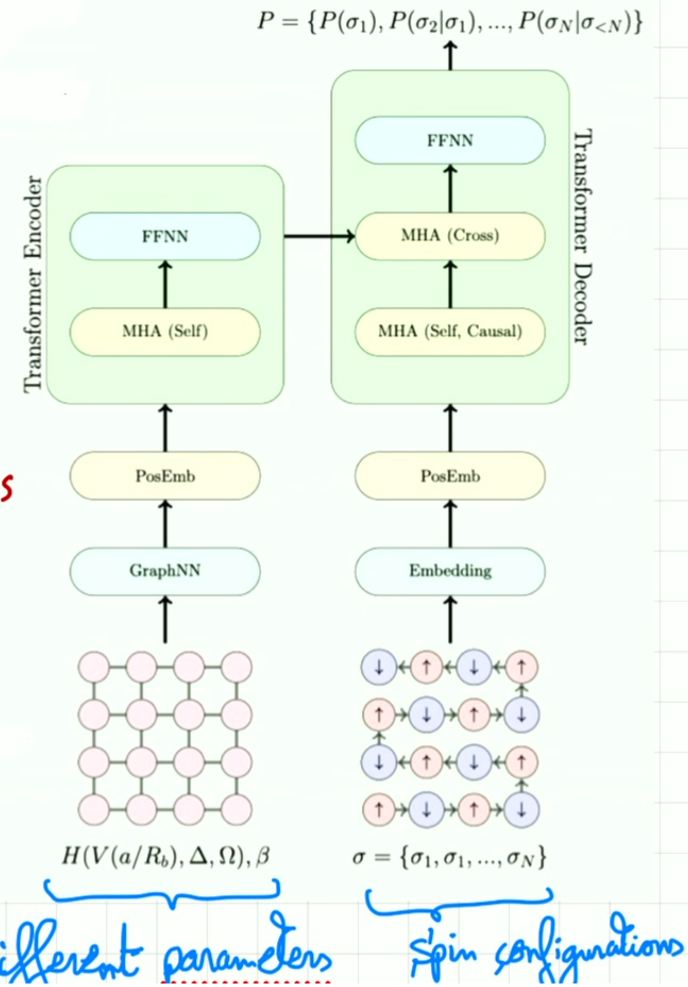
K. Sprague & S. Czischek, Comms Phys 2024

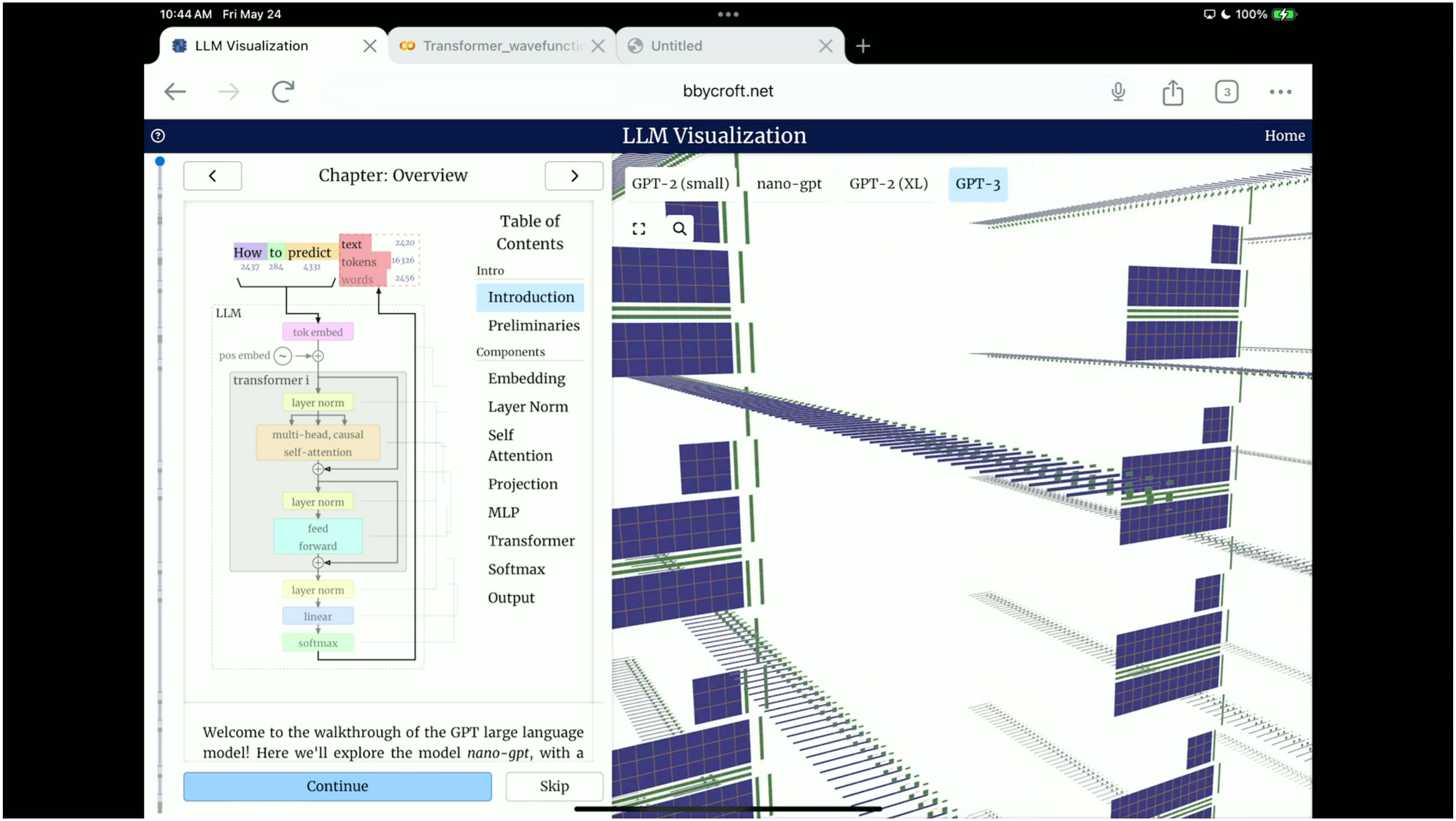
Another paper: "ArXiv: 2310.05715"

*Rydberg atoms Hamiltonian:

$$H = \Omega \sum_i \sigma_i^x - \Delta \sum_i n_i + \sum_{i<j} V_{ij} n_i n_j$$







10:44 AM Fri May 24
100%
LLM Visualization
Transformer_wavefunctic
Untitled
bbycroft.net
LLM Visualization
Home
Chapter: Overview
How to predict
2437 284 4331
text 2420
tokens 16326
words 2456
LLM
tok embed
pos embed
transformer i
layer norm
multi-head, causal self-attention
layer norm
feed forward
layer norm
linear
softmax
Table of Contents
Intro
Introduction
Preliminaries
Components
Embedding
Layer Norm
Self Attention
Projection
MLP
Transformer
Softmax
Output
Welcome to the walkthrough of the GPT large language model! Here we'll explore the model *nano-gpt*, with a
Continue
Skip
GPT-2 (small)
nano-gpt
GPT-2 (XL)
GPT-3

10:45 AM Fri May 24

LLM Visualization | Transformer_wavefuncti | GitHub - APRIQuOt/VMC

colab.research.google.com

Transformer_wavefunctions_on_Rydberg_Hamiltonian_on_square_lattice.ipynb

Transformer wave functions

This notebook is based on code implemented on this repository: https://github.com/APRIQuOt/VMC_with_LPTFs based on this paper: https://www.nature.com/articles/s42005-024-01584-y

Cloning github repository:

```
!git clone https://github.com/mhibatallah/VMC_with_TF.git
```

```
Cloning into 'VMC_with_TF'...
remote: Enumerating objects: 96, done.
remote: Counting objects: 100% (96/96), done.
remote: Compressing objects: 100% (94/94), done.
remote: Total 96 (delta 45), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (96/96), 47.67 KiB | 185.00 KiB/s, done.
Resolving deltas: 100% (45/45), done.
```

Training Arguments:

```
L         (int)     -- Total lattice size (8x8 would be L=64).

    Q         (int)     -- Number of minibatches per batch.

    K         (int)     -- size of each minibatch.

    B         (int)     -- Total batch size (should be Q*K).

    NLOOPS    (int)     -- Number of loops within the off_diag_labels function. Higher values save ram and
                           generally makes the code run faster (up to 2x). Note, you can only set this
```

10:46 AM Fri May 24

LLM Visualization | Transformer_wavefuncti | GitHub - APRIQuOt/VMC

colab.research.google.com

roximate the gradients, more efficient but approximate.

he end of the output directory (inside a subfolder).

```
n L=16 steps=1000 --ptf patch=2x2 Nh=32 nhead=1 --rydberg V=7 delta=

tf', 'patch=2x2', 'Nh=32', 'nhead=1', '--rydberg', 'V=7', 'delta=1',
s/torch/nn/modules/transformer.py:306: UserWarning: enable_nested_te
 is True, but self.use_nested_tensor is False because {why_not_spars
          {'L': 16, 'Q': 1, 'K': 256, 'B': 256, 'NLOOPS': 1, 'steps'
          {'L': 16, 'patch': '2x2', 'Nh': 32, 'dropout': 0.0, 'num_l
          {'Lx': 4, 'Ly': 4, 'V': 7, 'Omega': 1, 'delta': 1, 'L': 16

077 , Transformer energy variance= 471.61472
```

10:48 AM Fri May 24 ••• 100%

LLM Visualization | Transformer_wavefuncti | GitHub - APRIQuOt/VMC

colab.research.google.com

Transformer_wavefunctions_on_Rydberg_Hamiltonian_on_square_lattice.ipynb

```
1 !python VMC_with_TF/train.py --train L=16 steps=1000 --ptf patch=2x2 Nh=32 nhead=8 --rydberg V=7 delta=
```

```
['--train', 'L=16', 'steps=1000', '--ptf', 'patch=2x2', 'Nh=32', 'nhead=8', '--rydberg', 'V=7', 'delta=1', 'Omega=1']
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/transformer.py:306: UserWarning: enable_nested_tensor is True, but self.use_
  warnings.warn(f"enable_nested_tensor is True, but self.use_nested_tensor is False because {why_not_sparsity_fast_path}")
train                                          {'L': 16, 'Q': 1, 'K': 256, 'B': 256, 'NLOOPS': 1, 'steps': 1000, 'dir': 'PTF', 'lr':
model                                          {'L': 16, 'patch': '2x2', 'Nh': 32, 'dropout': 0.0, 'num_layers': 2, 'nhead': 8, 'rep
hamiltonian                                    {'Lx': 4, 'Ly': 4, 'V': 7, 'Omega': 1, 'delta': 1, 'L': 16, 'name': 'RYDBERG'}

Output folder path established
Step : 0 , Transformer energy = 31.8925 , Transformer energy variance= 543.7189

Step : 50 , Transformer energy = 7.497976 , Transformer energy variance= 179.69516

Step : 100 , Transformer energy = -2.9836862 , Transformer energy variance= 35.55628

Step : 150 , Transformer energy = -5.044218 , Transformer energy variance= 19.876806

Step : 200 , Transformer energy = -6.4999714 , Transformer energy variance= 2.637269

Step : 250 , Transformer energy = -6.633377 , Transformer energy variance= 3.255067

Step : 300 , Transformer energy = -6.476509 , Transformer energy variance= 8.979716

Step : 350 , Transformer energy = -6.746655 , Transformer energy variance= 1.1118166

Step : 400 , Transformer energy = -6.842925 , Transformer energy variance= 1.4557284
```

```
[ ] 1 !python VMC_with_TF/train.py --train L=16 steps=1000 --ptf patch=2x2 Nh=128 nhead=8 --rydberg V=7 delta=1 Omega=1
```

Final Remarks:

* Complexity:

Computationally more expensive

* Regular Transformer: $\mathcal{O}(N^3)$ sampling, $\mathcal{O}(N^2)$ inference

* RNN: $\mathcal{O}(N)$ for sampling and inference

Linear transformers: $\mathcal{O}(N)$

One example: « Transformers are RNNs »

* The intersection of $\langle \text{NLP} \,|\, \text{Many body physics} \rangle$ is very promising.

Modern applications of machine learning in quantum sciences

Anna Dawid[1,2,3⋆], Julian Arnold[4†], Borja Requena[2†], Alexander Gresch[5,6†], Marcin Płodzień[2], Kaelan Donatella[7], Kim A. Nicoli[8,9], Paolo Stornati[2], Rouven Koch[10], Miriam Büttner[11], Robert Okuła[12,13], Gorka Muñoz–Gil[14], Rodrigo A. Vargas–Hernández[15,16,17], Alba Cervera-Lierta[18], Juan Carrasquilla[16], Vedran Dunjko[19], Marylou Gabrié[20], Patrick Huembeli[21,22], Evert van Nieuwenburg[19,23], Filippo Vicentini[21,24], Lei Wang[25,26], Sebastian J. Wetzel[27], Giuseppe Carleo[21], Eliška Greplová[28], Roman Krems[29], Florian Marquardt[30,31], Michał Tomza[1], Maciej Lewenstein[2,32] and Alexandre Dauphin[2,33⋆]