# Neural Networks for Physicists

From one neuron to attention

*Mu-Yang Chen*

August 3, 2026

# Where we are going

## 1 — Foundations

- What machine learning actually is
- Basic architecture of a network
- Why it can fit anything
- How it is trained
- Examples: a pendulum, then Hopfield and Boltzmann machines

## 2 — Attention

- Why sequences need something new
- Query, key, value
- Single head, then many
- The bridge: attention is built to learn *relations* — and correlation is a relation

## 3 — In physics

- Neural-network wavefunction ansätze, two schools
- Variational Monte Carlo, and why quantum geometry enters
- Reduced density matrices without the wavefunction

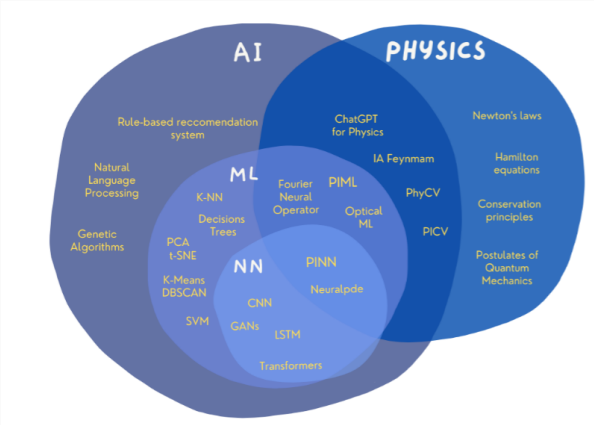
**One sentence**

A neural network is a very flexible fitting function. Everything else is detail — but the details are where the physics enters.

# Foundations

# Why a physicist should care

- **2024 Nobel Prize in Physics** — Hopfield & Hinton, for neural networks. The energy function of a Hopfield network *is* the spin-glass energy.
- Machine learning now sits inside DFT, lattice QCD, gravitational-wave searches, phase classification.
- Two directions:
  - Put physics in the network to *improve* the fit
  - Use the network to *find* physics




Miranda *et al.*, arXiv:2505.13042, Fig. 1

# Basic idea of training

## A physicist's fit

1. Write down the form: $\theta(t) = \theta_0 \cos \omega t$
2. One unknown: $\omega$
3. Minimise $\sum_i (\theta_i - \theta(t_i))^2$

## Machine learning

1. Write down a generic form with $10^2$–$10^{12}$ parameters
2. The "basis functions" adapt during the fit
3. Minimise the same kind of thing

## Training needs exactly four ingredients

1. A parametrised function $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \Theta)$, $\Theta = \{\boldsymbol{W}^{[\ell]}, \boldsymbol{b}^{[\ell]}\}$ — anywhere from $10^2$ to $10^{12}$ numbers
2. A training set $\mathcal{D} = \{(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha)\}_{\alpha=1}^{M}$ — the input/output pairs you want reproduced
3. A loss function $\mathcal{L}(\Theta)$ — one number saying how wrong $f$ currently is
4. An optimiser — the rule that changes $\Theta$ to make $\mathcal{L}$ smaller

Ingredient 2 defines the paradigm: **supervised** (pairs $\boldsymbol{x} \to \boldsymbol{y}$ — everything today), **unsupervised** (only $\boldsymbol{x}$), **reinforcement** (act, get reward).

# Ingredients 3 and 4: the loss, and how to minimise it

## 3. The loss function

One number measuring how far $f(\boldsymbol{x}; \Theta)$ sits from the data.

**Regression** — mean squared error

$$\mathcal{L} = \frac{1}{M} \sum_{\alpha=1}^{M} \left(y_\alpha - \hat{y}_\alpha\right)^2$$

Linear regression is just the special case $\hat{y} = wx + b$. Penalises big misses quadratically.

**Classification** — cross entropy

$$\mathcal{L} = -\frac{1}{M} \sum_{\alpha} \sum_{c} y_{\alpha c} \ln \hat{y}_{\alpha c}$$

$\hat{y}_{\alpha c}$ = predicted probability of class $c$ (a softmax output); $y_{\alpha c} \in \{0, 1\}$ is the true label. Squared error would barely punish a confident wrong answer — ln punishes it infinitely.

## 4. The optimiser

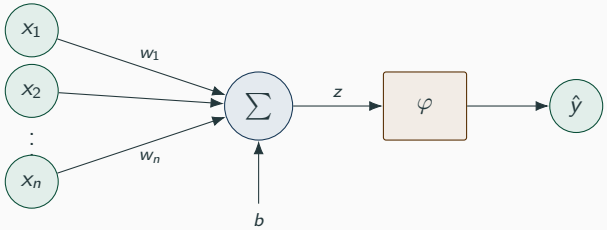
The rule that walks $\Theta$ downhill:

$$\Theta \leftarrow \Theta - \eta \, \nabla_\Theta \mathcal{L}$$

$\eta$ = learning rate, fixed *by hand* before training.

- GD — whole dataset, one step
- SGD — one batch at a time; the noise helps escape saddles
- Momentum — give $\Theta$ inertia
- Adam — per-parameter adaptive step; what everyone actually uses

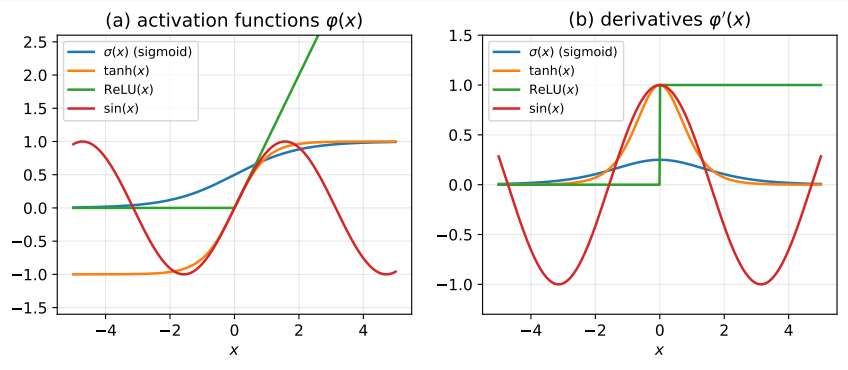
# The perceptron: one neuron




$$\hat{y} = \varphi\Big(\underbrace{\textstyle\sum_i w_i x_i + b}_{\text{affine — learned}}\Big), \qquad \varphi = \text{fixed nonlinearity}$$

$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ is a hyperplane: $\boldsymbol{w}$ is its normal, $b$ shifts it off the origin. The neuron reports which side you are on.

# Activation functions




**Why any nonlinearity at all?**

If $\varphi = \mathrm{id}$, composing layers gives
$\boldsymbol{W}^{[L]}(\cdots \boldsymbol{W}^{[2]}\boldsymbol{x}\cdots) = \boldsymbol{W}_{\text{eff}}\boldsymbol{x} + \boldsymbol{b}_{\text{eff}}$.

A hundred linear layers = one linear layer.

**Why ReLU won**

$\sigma'$ and tanh$'$ die for $|x| \gtrsim 4$ — a saturated neuron stops learning (*vanishing gradients*). ReLU has $\varphi' = 1$ on the whole positive axis.

# Aside: the sigmoid is the Fermi function

**A physicist's reading**

Write $x = -\beta(\varepsilon - \mu)$:

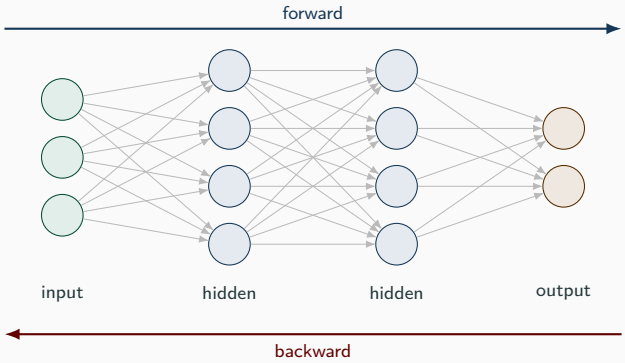
$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{1}{1 + e^{\beta(\varepsilon - \mu)}}$$

A sigmoid neuron is one two-state level at finite temperature.

- $\beta \to \infty$: step function, hard yes/no decision
- finite $\beta$: soft decision — and *differentiable*, which is the only reason we can train it

Not a loose analogy: Boltzmann machines are built by running exactly this limit backwards, replacing a deterministic threshold by a stochastic update with probability $\sigma(2\beta a_i)$. *[Hohm 2026, §4.2]*

# Stack them: the multilayer perceptron(MLP)




$$\boldsymbol{z}^{[\ell]} = \boldsymbol{W}^{[\ell]}\boldsymbol{a}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}, \qquad \boldsymbol{a}^{[\ell]} = \varphi\big(\boldsymbol{z}^{[\ell]}\big), \qquad \boldsymbol{a}^{[1]} = \boldsymbol{x}$$

# Why can this fit anything?

**Kolmogorov–Arnold 1957** — exact

$$f(x_1, \ldots, x_n) = \sum_{q=0}^{2n} \Phi_q \left( \sum_{p=1}^{n} \phi_{q,p}(x_p) \right)$$

$f \in C\left([0,1]^n\right)$, $\quad \Phi_q,\ \phi_{q,p} : \mathbb{R} \to \mathbb{R}$ continuous

**Cybenko 1989, Hornik 1991** — approximate

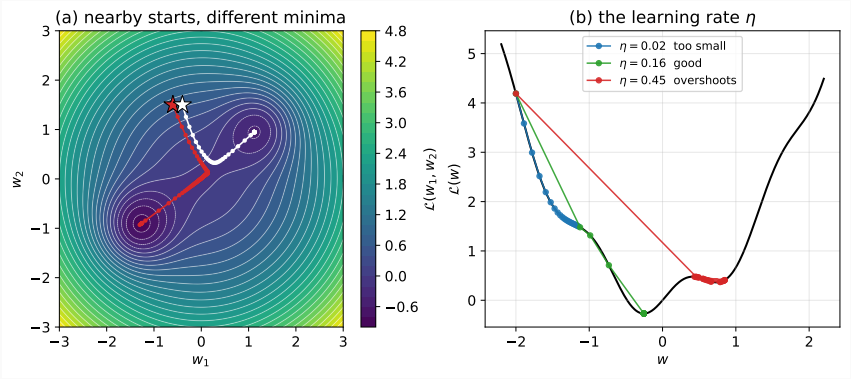
$$\sup_{\boldsymbol{x} \in K} \left| \sum_{k=1}^{N} c_k \, \varphi(\boldsymbol{w}_k \cdot \boldsymbol{x} + b_k) - g(\boldsymbol{x}) \right| < \varepsilon$$

$g \in C(K)$, $K$ compact, $\quad \varphi : \mathbb{R} \to \mathbb{R}$ $\quad$ (one hidden layer)

| | **Kolmogorov–Arnold** | **Cybenko–Hornik** |
|---|---|---|
| result | exact, $=$ | approximate, $< \varepsilon$ |
| number of terms | fixed, $2n+1$ | unbounded $N$ |
| one-variable functions | nowhere differentiable | any non-polynomial |
| fixed before seeing $f$ | the inner $\phi_{q,p}$ | nothing |

Row 3 is forced — Vitushkin (1954): smooth superpositions cannot exist. Only the right column licenses an MLP.

# Training: downhill




$$\Theta \leftarrow \Theta - \eta \nabla_\Theta C, \qquad C = \frac{1}{m} \sum_\alpha \tfrac{1}{2} \left\| \boldsymbol{y}_\alpha - \mathrm{NN}(\boldsymbol{x}_\alpha; \Theta) \right\|^2$$

epoch = one sweep through the data | batch = the subset used for one step | Adam = the optimiser everyone uses

# Backpropagation is the chain rule with bookkeeping

From the forward pass:

$$\boldsymbol{z}^{[\ell]} = \boldsymbol{W}^{[\ell]}\boldsymbol{a}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}, \qquad \boldsymbol{a}^{[\ell]} = \varphi\big(\boldsymbol{z}^{[\ell]}\big)$$

- $\boldsymbol{z}^{[\ell]}$ — pre-activation: what goes *into* the nonlinearity at layer $\ell$
- $\boldsymbol{a}^{[\ell]}$ — activation: what comes *out*
- $\boldsymbol{a}^{[L]} = \hat{\boldsymbol{y}}$ — the prediction of the last layer $L$
- $\boldsymbol{y}$ — the target from the training set
- $C$ — the cost; $\varphi'$ — slope of the activation

$$\Delta^{[L]} = (\boldsymbol{a}^{[L]} - \boldsymbol{y}) \odot \varphi'(\boldsymbol{z}^{[L]})$$

$$\Delta^{[\ell]} = \big[(\boldsymbol{W}^{[\ell+1]})^{\top}\Delta^{[\ell+1]}\big] \odot \varphi'(\boldsymbol{z}^{[\ell]})$$

The one new object:

$$\Delta_i^{[\ell]} = \frac{\partial C}{\partial z_i^{[\ell]}}$$

How much the cost moves when neuron $i$ of layer $\ell$ is nudged — the error assigned to that neuron.

So $(\boldsymbol{a}^{[L]} - \boldsymbol{y})$ below is simply prediction − truth: backpropagation starts at the output error and walks it backwards.

$$\frac{\partial C}{\partial b_i^{[\ell]}} = \Delta_i^{[\ell]}$$

$$\frac{\partial C}{\partial w_{ij}^{[\ell]}} = \Delta_i^{[\ell]}\, a_j^{[\ell-1]}$$

**Notation: $\odot$**

Elementwise (Hadamard) product, $(\boldsymbol{u} \odot \boldsymbol{v})_i = u_i v_i$ — not a matrix product: $\begin{pmatrix}1\\2\end{pmatrix} \odot \begin{pmatrix}3\\4\end{pmatrix} = \begin{pmatrix}3\\8\end{pmatrix}$. It is there because $\partial a_i^{[\ell]} / \partial z_j^{[\ell]} = \varphi'(z_i^{[\ell]})\,\delta_{ij}$ is diagonal.

# A pendulum, two ways

# The same system, two very different networks

$$\ddot{\theta} + \frac{g}{\ell}\sin\theta = 0 \quad \xrightarrow[\text{small } \theta]{} \quad \ddot{\theta} + \omega_0^2\theta = 0, \qquad \theta(t) = \theta_0 \cos\omega_0 t$$

## 1. Learn a constant

Network outputs $\omega'$; the known solution sits in the loss:

$$\mathcal{L} = \tfrac{1}{M}\sum_i \left[\theta_i - \theta_0\cos(\omega' t_i)\right]^2$$

4 parameters, 300 data points.

## 2. Solve the ODE (PINN)

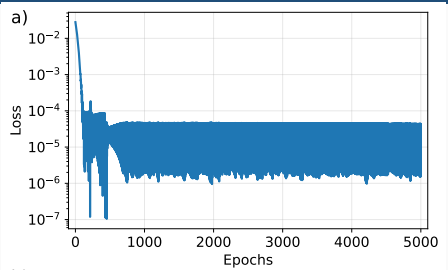
Network is $\theta(t)$; the equation is the loss — and there is no data at all:

$$\mathcal{L} = \tfrac{1}{M}\sum_i \left[\tfrac{\mathrm{d}^2}{\mathrm{d}t^2}\mathrm{NN}(t_i) + \omega_0^2\mathrm{NN}(t_i)\right]^2 + \mathcal{L}_{\mathrm{IC}}$$

141 parameters, 50 collocation points.

Same optimiser, same 5000 epochs. Total runtime for both: $\approx$ 10 seconds on a laptop CPU.

# Result 1: recovering $g$



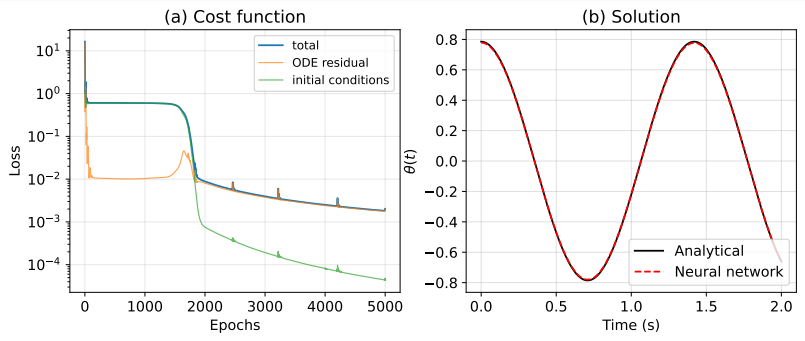

- Truth: $g = 9.81$, $\ell = 10$ m
- Recovered: $g' = 9.808954$
- Error $1.0 \times 10^{-3}$, i.e. 0.011%
- 2.1 s of training

**Two honest caveats**

The "network" is degenerate — the code forces its output to be constant in $t$, so 4 parameters represent 1 number.

And at $\eta = 10^{-4}$ the loss curve looks *smoother* but gives $g' = 9.924$ — 100× worse.

# Result 2: the PINN — and what the loss hides




**Read the plateau**

Epochs 100–1700: the ODE residual is *low* while the initial-condition term sits at 0.6. The network has found $\theta(t) \equiv 0$ — which satisfies $\ddot{\theta} + \omega_0^2 \theta = 0$ exactly. Only the initial conditions eventually push it off. Plotting one lumped loss curve hides this completely.

# Hopfield network: a spin glass that remembers

$N$ neurons $\sigma_i = \pm 1$, symmetric weights $w_{ij} = w_{ji}$, $w_{ii} = 0$:

$$E(\sigma) = -\frac{1}{2}\sum_{i \neq j} w_{ij}\sigma_i\sigma_j - \sum_i b_i\sigma_i$$

Local field $a_i = \sum_j w_{ij}\sigma_j + b_i$. Update one neuron at a time, deterministically:

$$\sigma_i \leftarrow \mathrm{sign}(a_i)$$

Each flip can only lower $E \Rightarrow$ the dynamics runs downhill into a local minimum and stops.

## Storing memories — Hebb's rule

For $P$ patterns $\xi^A$, set $w_{ij} = \frac{1}{N}\sum_A \xi_i^A \xi_j^A$. Each $\xi^A$ becomes a local minimum.

Feed in a corrupted pattern; the dynamics relaxes to the nearest stored one. Associative memory — addressed by content, not by location.

Capacity $P_{\max} \approx 0.138\, N$; beyond it the minima merge into spin-glass states and recall fails.

(Amit–Gutfreund–Sompolinsky 1985)

## A physicist's reading

$E$ is exactly the spin-glass energy — Ising spins renamed neurons, couplings renamed weights. Nothing is analogy here. Hopfield's 2024 Nobel was for noticing that a memory is an attractor. *[Hohm 2026, §4.1]*

# Boltzmann machine: switch on the temperature

Same energy, same weights — but the update is now stochastic:

$$\sigma_i = +1 \text{ with probability } \sigma(2\beta a_i)$$

The sigmoid from three slides ago, at inverse temperature $\beta$.

$\beta \to \infty$ recovers Hopfield.

At equilibrium the network samples

$$P(\sigma) = \frac{e^{-\beta E(\sigma)}}{Z}$$

It no longer computes a function — it generates configurations.

A model of a distribution, not a map.

**Learning = matching correlations**

$$\Delta w_{ij} = \eta\Big(\langle \sigma_i \sigma_j \rangle_{\text{data}} - \langle \sigma_i \sigma_j \rangle_{\text{model}}\Big)$$

Adjust couplings until the model's two-point function reproduces the data's. Stop when they agree.

**Restricted BM**: visible and hidden neurons, no couplings *within* either layer. $\langle \cdot \rangle_{\text{data}}$ becomes tractable, and integrating out the hidden layer is literally a renormalisation-group step. Stacking such layers is what led to deep learning. *[Hohm 2026, §4.2–4.3]*

# Attention

# Sequences need something new

An MLP takes a fixed-size vector. A sentence, a time series, a trajectory, a set of particles — these are sequences of $T$ tokens, with $T$ varying, where what matters is the relation *between* elements. We want to replace each $\boldsymbol{x}_i$ by something informed by the whole sequence:

$$\boldsymbol{o}_i = \sum_{j=1}^{T} \alpha_{ij}\, \boldsymbol{x}_j, \qquad \sum_j \alpha_{ij} = 1$$

**The entire design question: where do the $\alpha_{ij}$ come from?**

1. $\alpha_{ij} = 1/T$ — a plain average. Throws everything away.
2. $\alpha_{ij} = f(i-j)$ — fixed by *separation*. This is a convolution (a CNN). Same kernel whatever the content; limited reach.
3. $\alpha_{ij} = \alpha(\boldsymbol{x}_i, \boldsymbol{x}_j)$ — computed from the *content*, afresh every time. This is attention.

# Attention is a soft dictionary lookup

A lookup table — a `dict`, a library catalogue — stores (key, value) pairs. You arrive with a query, match it against the keys, get back a value.

**Library catalogue**

- key = the subject heading — what makes a book *findable*
- value = the book itself — what you actually *take home*
- query = your request, in your own words

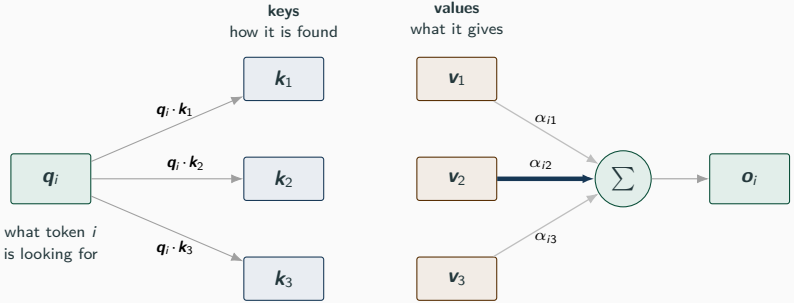
**The problem**

An exact match is all-or-nothing ⇒ not differentiable ⇒ no gradient, nothing to train.

**The fix**

Score every key against the query, softmax the scores, return the weighted blend of all the values. Hard retrieval becomes smooth interpolation — and now it is differentiable.

# One attention operation




$$q_i = W_Q x_i, \qquad k_j = W_K x_j, \qquad v_j = W_V x_j$$

$W_Q, W_K, W_V$ are the only trainable objects. The weights $\alpha_{ij}$ are recomputed for every input.

# Why *three* different matrices?

**key $\neq$ value**

What makes a token findable need not be what it contributes. You search on the subject heading; you take home the book. Tying $\boldsymbol{W}_K = \boldsymbol{W}_V$ would force a token to be retrieved by exactly the information it passes on.

**query $\neq$ key**

Relevance is asymmetric. An adjective looking for its noun is a different relation from that noun looking back at the adjective. $\boldsymbol{W}_Q = \boldsymbol{W}_K$ would make the score matrix symmetric in $i \leftrightarrow j$ — and destroy the direction, which is most of the grammar.

# The formula, and a worked example

$$\alpha_{ij} = \frac{\exp(\boldsymbol{q}_i \cdot \boldsymbol{k}_j / \sqrt{d_k})}{\sum_{j'} \exp(\boldsymbol{q}_i \cdot \boldsymbol{k}_{j'} / \sqrt{d_k})}, \qquad \boldsymbol{o}_i = \sum_j \alpha_{ij} \boldsymbol{v}_j \qquad \Longleftrightarrow \qquad \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

"the ball bounced", embedding axes = (noun-ness, verb-ness); $\boldsymbol{W}_Q$ turns verb-ness into "I want a noun", $\boldsymbol{W}_K$ turns noun-ness into "I am a noun".

| $j$ | $\boldsymbol{q} \cdot \boldsymbol{k}_j$ | $e^{(\cdot)/\sqrt{2}}$ | $\alpha_{\text{bounced},j}$ |
|---|---|---|---|
| the | 0.8 | 1.76 | 0.090 |
| ball | 4.0 | 16.92 | **0.860** |
| bounced | 0.0 | 1.00 | 0.051 |

The verb puts 86% of its attention on ball — it has found its subject.

Cost: the score matrix is $T \times T$, so attention is $O(T^2)$. That is why context length is expensive.

# The two formulas, collected

## Single head

$$Q = XW_Q^\top, \quad K = XW_K^\top, \quad V = XW_V^\top, \qquad \mathrm{Attn}(Q,K,V) = \mathrm{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}} + M\Big)V$$

$X \in \mathbb{R}^{T\times d}$ stacks the $T$ tokens as rows; $Q, K \in \mathbb{R}^{T\times d_k}$, $V \in \mathbb{R}^{T\times d_v}$, output $\in \mathbb{R}^{T\times d_v}$. Mask $M_{ij} = -\infty$ for $j > i$ (causal), 0 otherwise. Softmax acts on each row.

## Multi-head — $H$ of them in parallel

$$\mathrm{head}_h = \mathrm{Attn}\big(XW_Q^{(h)\top}, XW_K^{(h)\top}, XW_V^{(h)\top}\big), \qquad \mathrm{MHA}(X) = \big[\,\mathrm{head}_1 \,\|\, \cdots \,\|\, \mathrm{head}_H\,\big]\, W_O$$

$W_O \in \mathbb{R}^{Hd_v\times d}$ mixes the heads back to width $d$. Taking $d_k = d_v = d/H$ makes $H$ heads cost the same as one full-width head — $4d^2$ parameters either way.

Why more than one head? A single softmax concentrates on *one* criterion of relevance at a time. $H$ heads run $H$ criteria at once — syntax, position, long-range reference — and $W_O$ decides how to combine them.

# Neural networks in physics

# Two places to put the network

The many-body problem: $\dim \mathcal{H}$ grows exponentially. Two ways to spend a network on it.

## Route 1 — learn the wavefunction

$\Psi_\theta(R)$ *is* the network. Optimise by energy minimisation:

$$E_\theta = \frac{\langle \Psi_\theta | \hat{H} | \Psi_\theta \rangle}{\langle \Psi_\theta | \Psi_\theta \rangle} \geq E_0$$

Variational $\Rightarrow$ lower energy is strictly better. No training data — the Hamiltonian is the supervisor, exactly as in the PINN.

Fu (attention orbitals) and Zhang (Slater–Jastrow–backflow)

## Route 2 — skip $\Psi$, learn the RDM

Most observables need only

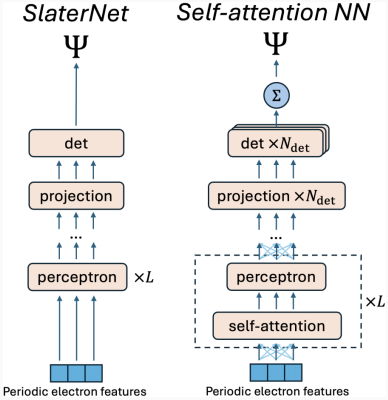
$$O = \langle c^\dagger_{k_1\alpha_1} \cdots c_{k'_1\alpha'_1} \rangle$$

1-RDM gives every symmetry-breaking order parameter; the 2-RDM gives the energy of any two-body $\hat{H}$. The full $\Psi$ is more than you need.

Yu (interpolating RDMs across momentum meshes)

All three use the machinery of Part 1 and Part 2 — nothing new is needed beyond attention, MLPs, and gradient descent.

# Fu group: attention builds *correlated* orbitals




Geier *et al.*, PRB **112**, 045119 (2025)

**SlaterNet** — MLP only

$$\phi_j(\boldsymbol{r}_i) \longrightarrow \Psi = \det_{ij} \phi_j(\boldsymbol{r}_i)$$

Each electron passes through the *same* MLP independently. One Slater determinant ⇒ this is exactly unrestricted Hartree–Fock, found variationally.

**Self-attention NN** — add the coupling

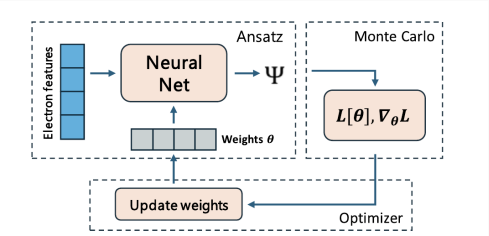
$$\Psi(R) = \sum_{m=1}^{N_{\text{det}}} \det_{ij} \; \phi_j^m\big(\boldsymbol{r}_i; \{\boldsymbol{r}_{/i}\}\big)$$

Orbital of electron $i$ now depends on where all the others are — produced by self-attention over the $N$ electron streams: the $Q, K, V$ of Part 2, with electrons in place of tokens.

Attention is permutation-*equivariant*, so the determinant stays antisymmetric — Fermi statistics survive by construction.

Parameters scale as $\sim N^2$, against $e^{\sqrt{N}}$ for tensor networks.

# How you actually vary it: neural-network VMC




Geier *et al.*, PRB **112**, 045119 (2025), Fig. 2

The integral is $3N$-dimensional. Rewrite it as an average:

$$E_\theta = \frac{\int dR\, |\Psi_\theta|^2\, E_{\mathrm{loc}}}{\int dR\, |\Psi_\theta|^2} = \mathbb{E}_{R\sim|\Psi_\theta|^2}\left[E_{\mathrm{loc}}(R)\right]$$

with the local energy $E_{\mathrm{loc}} = \Psi_\theta^{-1}\hat{H}\Psi_\theta$.

- Sample $R = (\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N)$ by Metropolis — no normalisation needed
- Average $E_{\mathrm{loc}}$ over the batch $\Rightarrow$ energy *and* its gradient $\nabla_\theta E$
- Update $\theta$; resample; repeat
- Zero-variance property: in an exact eigenstate $E_{\mathrm{loc}}$ is constant, so the error bar vanishes as $\Psi_\theta \to \Psi_0$

It is the training loop of Part 1 with two substitutions: the training set becomes Monte Carlo samples redrawn every step, and the loss becomes the energy itself.

# Why the quantum geometric tensor enters the optimiser

Plain gradient descent takes the steepest step in parameter space. But $\theta$ is arbitrary — reparametrise and the "steepest" direction changes.

What is physical is distance between states:

$$\|d\theta\|^2 = 1 - |\langle \Psi_{\theta+d\theta} | \Psi_\theta \rangle|^2 = \sum_{nm} g_{nm}\, d\theta_n d\theta_m$$

the quantum geometric tensor (Fubini–Study metric on projective Hilbert space)

$$g_{nm} = \langle \partial_n \Psi | \big(1 - |\Psi\rangle\langle\Psi|\big) | \partial_m \Psi \rangle$$

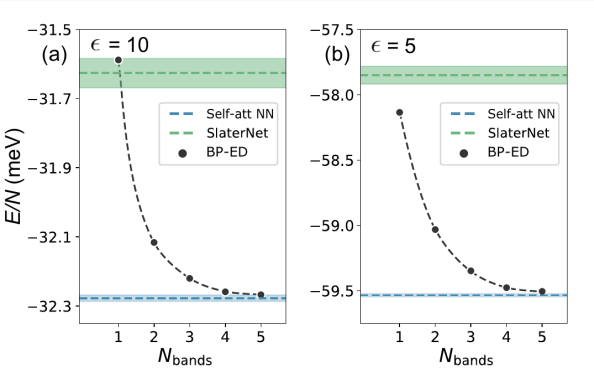
**Natural gradient descent**

$$d\theta = -\eta\, g^{-1} \nabla_\theta E$$

Steepest descent *on the manifold of wavefunctions*, not on the coordinate chart. Invariant under reparametrisation.

- $g$ is $N_{\mathrm{par}} \times N_{\mathrm{par}}$ — inverting it directly is hopeless for $10^5$ parameters
- KFAC (Fu): approximate $g^{-1}$ layer by layer
- MinSR (Zhang): work in the $\leq N_{\mathrm{samples}}$-dimensional subspace instead
- Real $g \Rightarrow$ quantum metric; imaginary part $\Rightarrow$ Berry curvature. The same tensor you use for band geometry

# Does it work? Benchmark against ED and Hartree–Fock




Geier *et al.*, PRB **112**, 045119 (2025), Fig. 5 — $WSe_2/WS_2$ moiré, $3 \times 3$ supercell, $\nu = 2/3$

- Green — SlaterNet = Hartree–Fock. Misses correlation entirely.
- Black dots — band-projected ED, as more bands are kept
- Blue — self-attention NN

ED converges *towards* the attention result from above as $N_{\text{bands}}$ grows, and never gets below it. The gap to SlaterNet is the correlation energy — about 2% here, far larger than in molecules.

**The honest reading**

This is a variational upper bound, not a proof of exactness. What it shows is that a general-purpose architecture, given no physics beyond $\hat{H}$, beats a method that must truncate the Hilbert space by hand.

# Zhang group: keep the physics form, learn the corrections

Valenti *et al.*, arXiv:2512.07947 — $\alpha$-jellium, does the anomalous Hall crystal survive quantum fluctuations?

**The (MP)$^2$NQS ansatz — Slater–Jastrow–backflow**

$$\Psi(R,S) = \underbrace{e^{-U(R,S)}}_{\text{Jastrow}} \det_{ij}\Big[\varphi_j\big(\underbrace{\boldsymbol{r}_i + \boldsymbol{\mathcal{N}}_i(R)}_{\text{backflow}},\, \sigma_i\big)\Big]$$

$$\underbrace{\varphi_j(\boldsymbol{r},\sigma) = \sum_{\boldsymbol{k}} c_\sigma^{(j\boldsymbol{k})}\, e^{i\boldsymbol{k}\cdot\boldsymbol{r}}}_{\text{orbitals: a truncated plane-wave basis, coefficients optimised}} \qquad U = \underbrace{\sum_{i\neq j} u_{\sigma_i\sigma_j}\big(|\boldsymbol{r}_i - \boldsymbol{r}_j|\big)}_{\text{2-body, B-spline}} + \underbrace{U_{\mathrm{N.N.}}(R)}_{\text{MLP}}$$

**Four learnable objects, each an old idea:**

- $c_\sigma^{(j\boldsymbol{k})}$ — plane-wave coefficients. The orbitals never leave the free-electron basis; only the mixing is learned
- $\boldsymbol{\mathcal{N}}_i$ — backflow shift, a message-passing GNN
- $u_{\sigma\sigma'}$ — two-body Jastrow, B-splines
- $U_{\mathrm{N.N.}}$ — many-body Jastrow, an MLP

$\sigma_i = \pm$ is the spinor index — it carries the Berry curvature.

**Contrast with Fu:** there the orbitals *are* network outputs, no fixed basis at all. Here the plane-wave expansion is written down first, and the networks supply only backflow and Jastrow corrections.

**Result:** the AHC survives beyond mean field, and quantum geometry enhances crystallisation — stable up to an order of magnitude above the $\alpha = 0$ critical density.

HF $\to$ SJ $\to$ SJBF $\to$ NQS moves the liquid–crystal competition from $10^{-2}$ to $10^{-4}$ Ry — not an artefact of the ansatz.

# Two philosophies for the same object

| | **Fu** (attention) | **Zhang** ((MP)$^2$NQS) |
|---|---|---|
| starting form | bare Slater determinant | Slater–Jastrow–backflow |
| orbital basis | none — orbitals are network outputs | truncated plane waves $\sum_{\boldsymbol{k}} c_\sigma^{(j\boldsymbol{k})} e^{i\boldsymbol{k}\cdot\boldsymbol{r}}$ |
| where correlation lives | correlated orbitals $\phi_j(\boldsymbol{r}_i; \{\boldsymbol{r}_{/i}\})$ | Jastrow $\times$ backflow shift |
| network | self-attention over electrons | message-passing GNN + MLP |
| architectural bias | none — general purpose | the known physics of a good trial state |
| optimiser | KFAC | MinSR |

## Learn everything

No prior structure. Fu's group finds chiral $p_x + ip_y$ pairing without ever telling the network about pairing (Li *et al.*, arXiv:2509.03683).

## Learn the correction

Start from a state that already knows about screening and exchange; the network only supplies what is missing. Fewer parameters, faster convergence.

This is the trade-off from the pendulum, one level up: how much physics you build in sets how much the network has to discover.

# Yu group: don't learn $\Psi$ at all — learn the RDM

Azam, Zhao & Yu, arXiv:2511.07367

## The observation

For a gapped state, the $n$-RDM is a smooth function over the Brillouin zone.

Smooth functions are exactly what networks interpolate well.

So: train on a coarse momentum mesh where ED or HF is affordable, then evaluate on a fine mesh you could never diagonalise.

Two architectures:

- SIREN — an MLP with sin activations mapping momentum $\boldsymbol{k} \mapsto$ RDM value. A coordinate network, not a data-fitter.
- Self-attention NN — maps a random RDM onto a physical one

## Why this is a different game

Routes 1 and 2 both minimise energy. This is interpolation: the physics input is not a Hamiltonian but the smoothness of a gapped ground state.

The sin activation is the same trick as the PINN in Part 1 — choose a basis that already resembles the answer. Here the target lives on a torus, so sinusoids are the natural choice.

Compare Part 1: universal approximation guarantees nothing outside the training domain. Here the training domain is the whole Brillouin zone — it is only ever asked to fill in *between* known points.

# What interpolation buys

## Prediction — pair-pair correlation function

Richardson model of superconductivity:

- SIREN trained on a $6 \times 6$ mesh predicts the $18 \times 18$ pair-pair correlation to 94.3%
- Trained instead on 4 tilted meshes of 12 points each: 93.8%

## Acceleration — Hartree–Fock initial guess

- $6 \times 6, 8 \times 8 \rightarrow$ initial guess for $50 \times 50$ translation-invariant HF: $-91.6\%$ iterations
- $\rightarrow 30 \times 30$ translation-breaking HF: $-92.8\%$ iterations

## And with representability built in

Hart *et al.*, arXiv:2605.20326 — twisted bilayer $MoTe_2$, FCI at $\nu = 2/3$. Impose $N$-representability conditions in the architecture *and* the loss, then optimise the 2-RDM variationally: energy 0.104 meV below ED on a $6 \times 6$ mesh, using $< 1/20$ the parameters of semidefinite programming.

Notable: among six architectures tested, the winner was a plain residual MLP — a Kolmogorov–Arnold network lost. The non-smoothness obstruction from Part 1 has not gone away.

The catch: nothing here is variational. An interpolated RDM carries no energy bound, and representability must be imposed by hand — the full condition is NP-complete.

# What to take away

## Foundations

- A network is an adaptive fitting function: affine map, nonlinearity, repeat
- It approximates anything — but only on the domain you trained on
- Backpropagation makes the gradient cheap; autodiff makes it automatic
- Hopfield and Boltzmann machines are spin glasses

## Attention

- Weights computed from content, recomputed every input
- $\mathrm{softmax}(QK^\top/\sqrt{d_k})V$ — a Boltzmann average with $\beta = 1/\sqrt{d_k}$
- Multi-head = several relevance criteria at once, at no extra cost
- Built to learn relations — which is what correlation is

## In physics

- Attention orbitals find chiral $p$-wave pairing unprompted
- Or keep Jastrow–backflow and learn only the correction
- Or skip $\Psi$ and interpolate the RDM
- Optimisation needs the quantum geometric tensor — the same object as band geometry

**The one question to ask of any of this**

How much physics is built in — and therefore, how much is the network actually being asked to discover?

# Questions?

| | |
|---|---|
| **Notes** | `note/nn_for_physicists.pdf` |
| **Code** | `scripts/` |
| **Background** | Miranda *et al.* arXiv:2505.13042 |
| | Hohm arXiv:2605.06394 |
| **Physics** | Geier *et al.* PRB 112, 045119 (2025) |
| | Li *et al.* arXiv:2509.03683 |
| | Valenti *et al.* arXiv:2512.07947 |
| | Azam *et al.* arXiv:2511.07367 |
| | Hart *et al.* arXiv:2605.20326 |