Deep Learning Beyond Loss Curves

A tutorial on statistical rigour and informed design in ML

Timothy Heightman
Quantum Information Theory and Quantum Optics Theory Groups ICFO

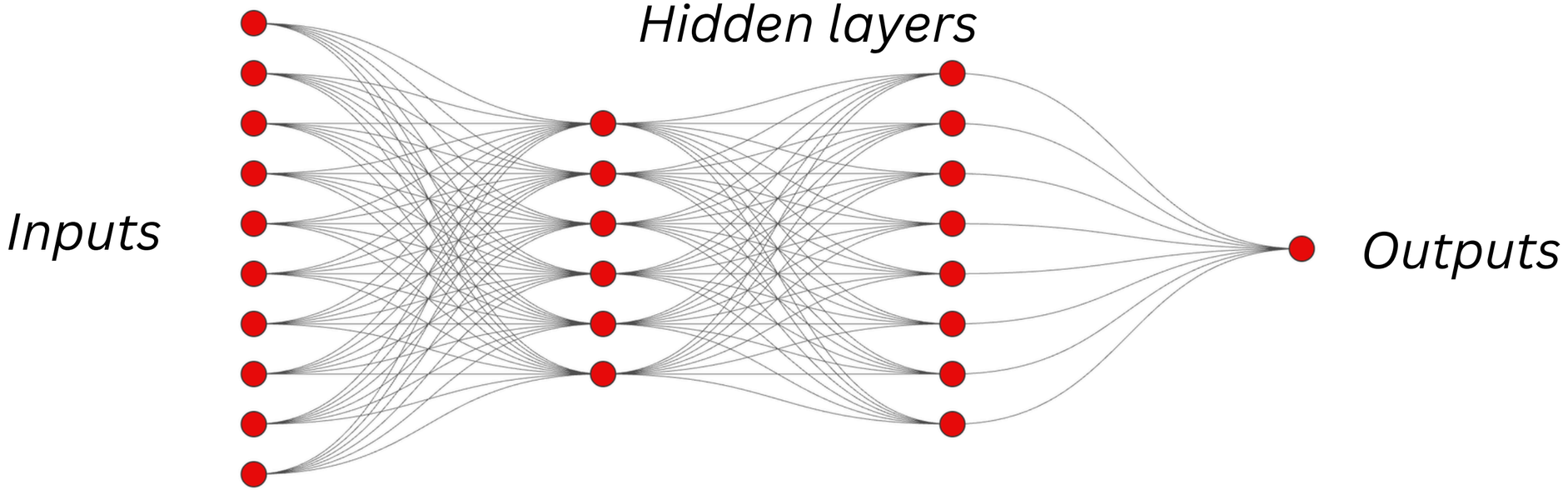
2nd Machine Learning Methods for Complex and Quantum Systems (ML2026)

CONTENT

CONTENT

Neural Network Anatomy

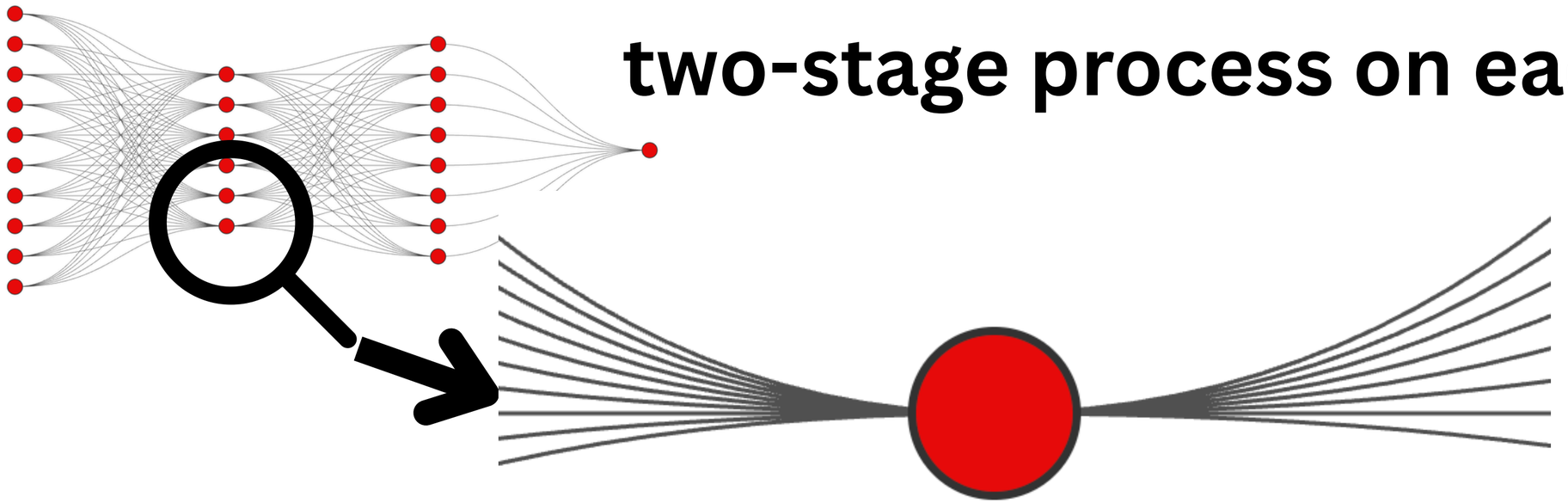
- A neural network is just a tunable non-linear function:



- Read from left to right tells us information flow

- Within a given neuron, there is is a non-linear “*activation function*”

Neuron Anatomy



two-stage process on each layer

1. Linear transformation $z \leftarrow \mathbf{W}z + b$

2. Non-linear activation e.g. *sigmoid* $\sigma(z) = \frac{1}{1 + e^{z}}$

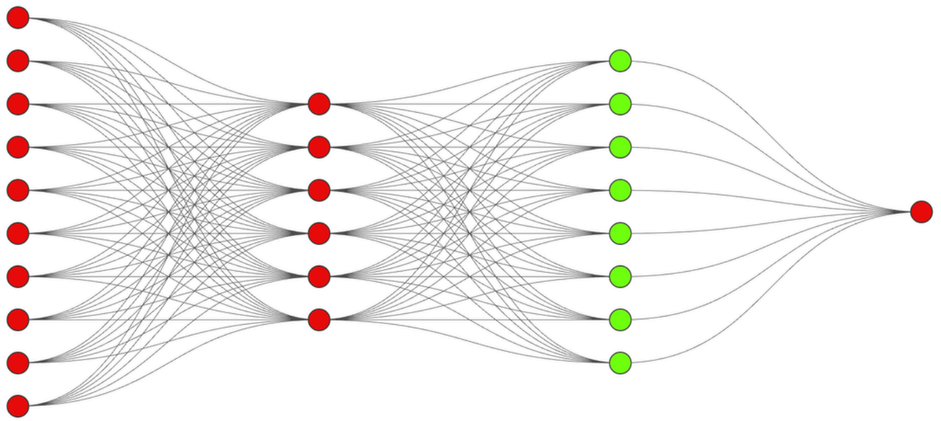
- W: Weight matrix
- b: bias
- z: latent vector

Neural Network Anatomy

- A full layer's transoformation is

$$h^{(l)}(\mathbf{z}) = \sigma\Big(\sum_{i=1}^{N_l}\sum_{j=1}^{N_{l-1}} \mathbf{W}_{i,j}^{(l)}\mathbf{z}_j + \sum_{i=1}^{N_l} \mathbf{b}_i^{(l)}\Big).$$

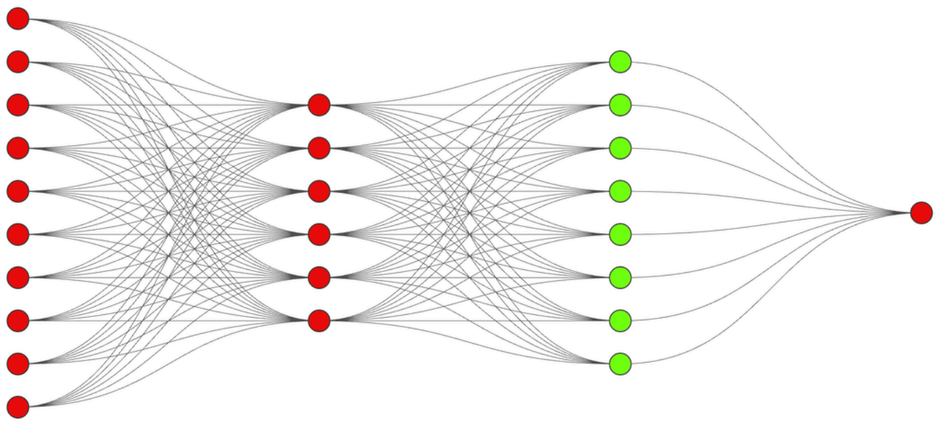
- e.g. layer two into three has 6 x 8 weight matrix



Neural Network Anatomy

- Variational paramters are any paramters we can tune

$$\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)} | l = 1, \ldots, L\}.$$

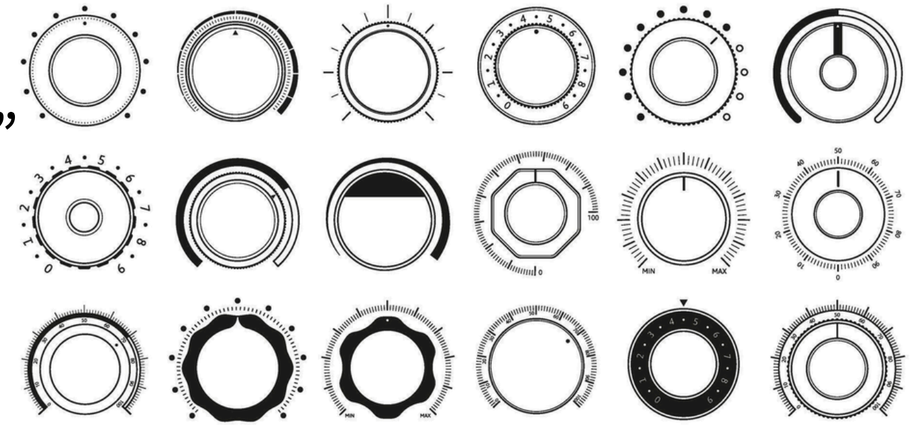


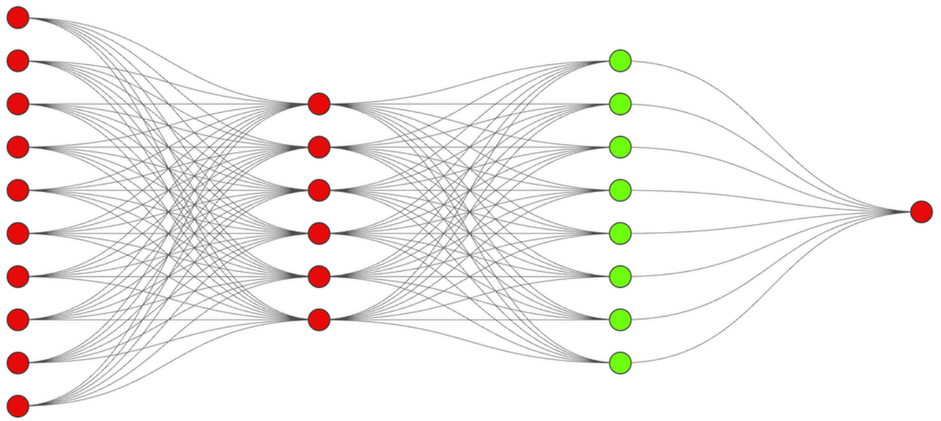
Neural Network Anatomy

- Variational paramters are any paramters we can tune

$$\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)} | l = 1, \ldots, L\}.$$

“bunch of dials and knobs we can tune”

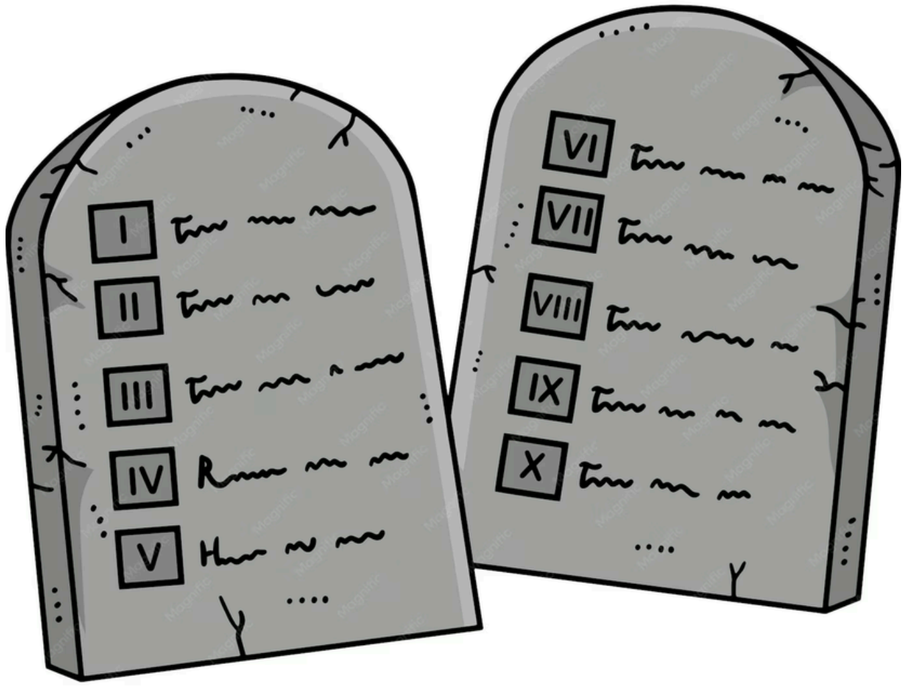




Universal Approximation Theorem

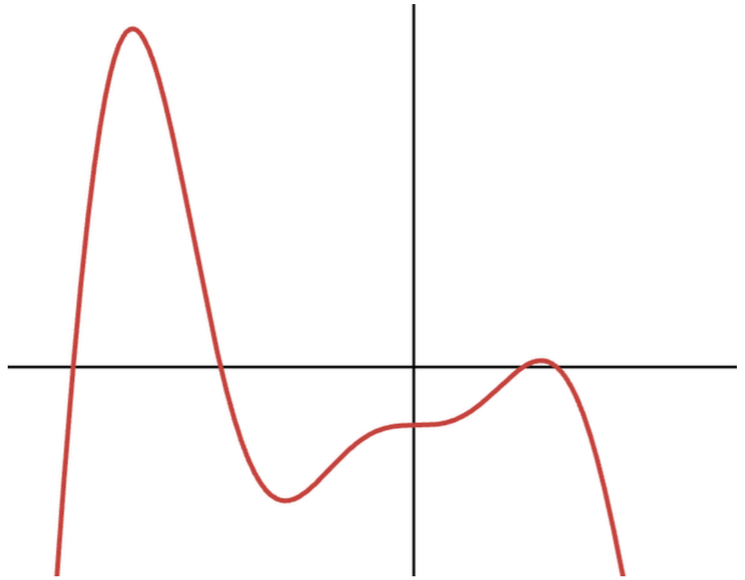
- Any continuous function on a compact domain can be approximated to an arbitrary level of accuracy with a sufficiently deep neural network.

- Central idea of deep learning!!



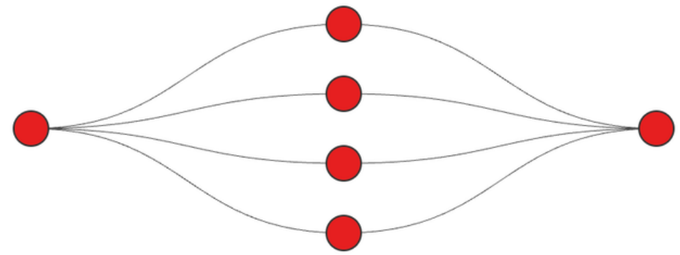
A First Example

$$g(x) = x^2 \sin(\pi x) - \frac{1}{2 + \cos(x)}$$



- Our goal is to find a set of *optimal* parameters $\boldsymbol{\theta}^* = \{\boldsymbol{W}^{(1)*}, \boldsymbol{b}^*, \boldsymbol{W}^{(2)*}\},$
- When we find them, we have a *good* approximation

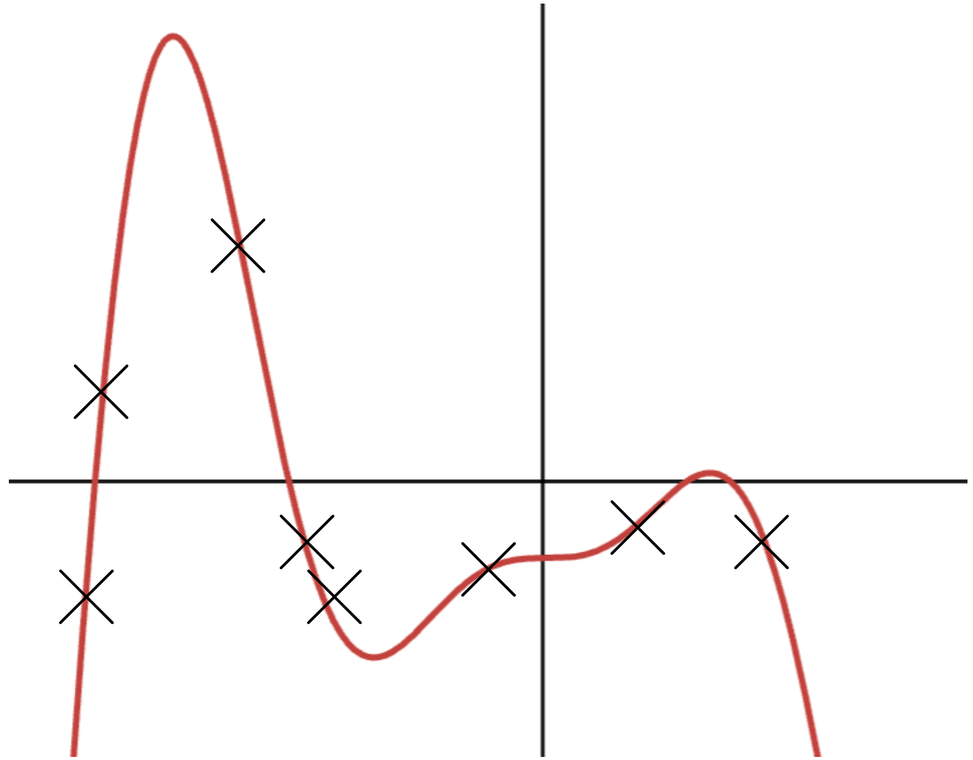
$$|f_\theta(x) - g(x)| \to 0$$



A First Example

- To tune parameters, we need a *dataset*

$$\mathcal{D} = \{x_i, g(x_i)\}_{i=1}^{K}$$



- We also need a measure of approximation quality - “*Loss function*”

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{i=1}^{K} |g(x_i) - f(x_i;\boldsymbol{\theta})|^2$$

A First Example

- We vary parameters when *training* the network with gradient information:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{K} \sum_{i=1}^{K} |g(x_i) - f(x_i; \boldsymbol{\theta})|^2$$

  - Backpropagation (the chain rule) lets us update parameters accross the whole network

A First Example

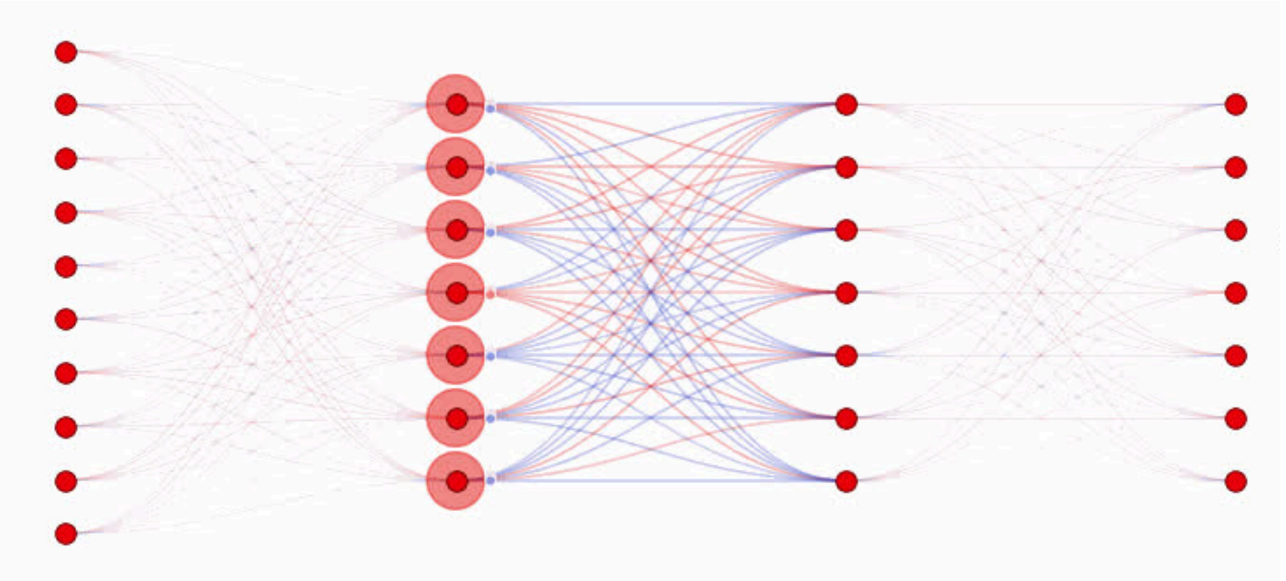
- We vary parameters when *training* the network with gradient information:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L$$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{K} \sum_{i=1}^{K} |g(x_i) - f(x_i; \boldsymbol{\theta})|^2$$

- Backpropagation (the chain rule) lets us update parameters accross the whole network

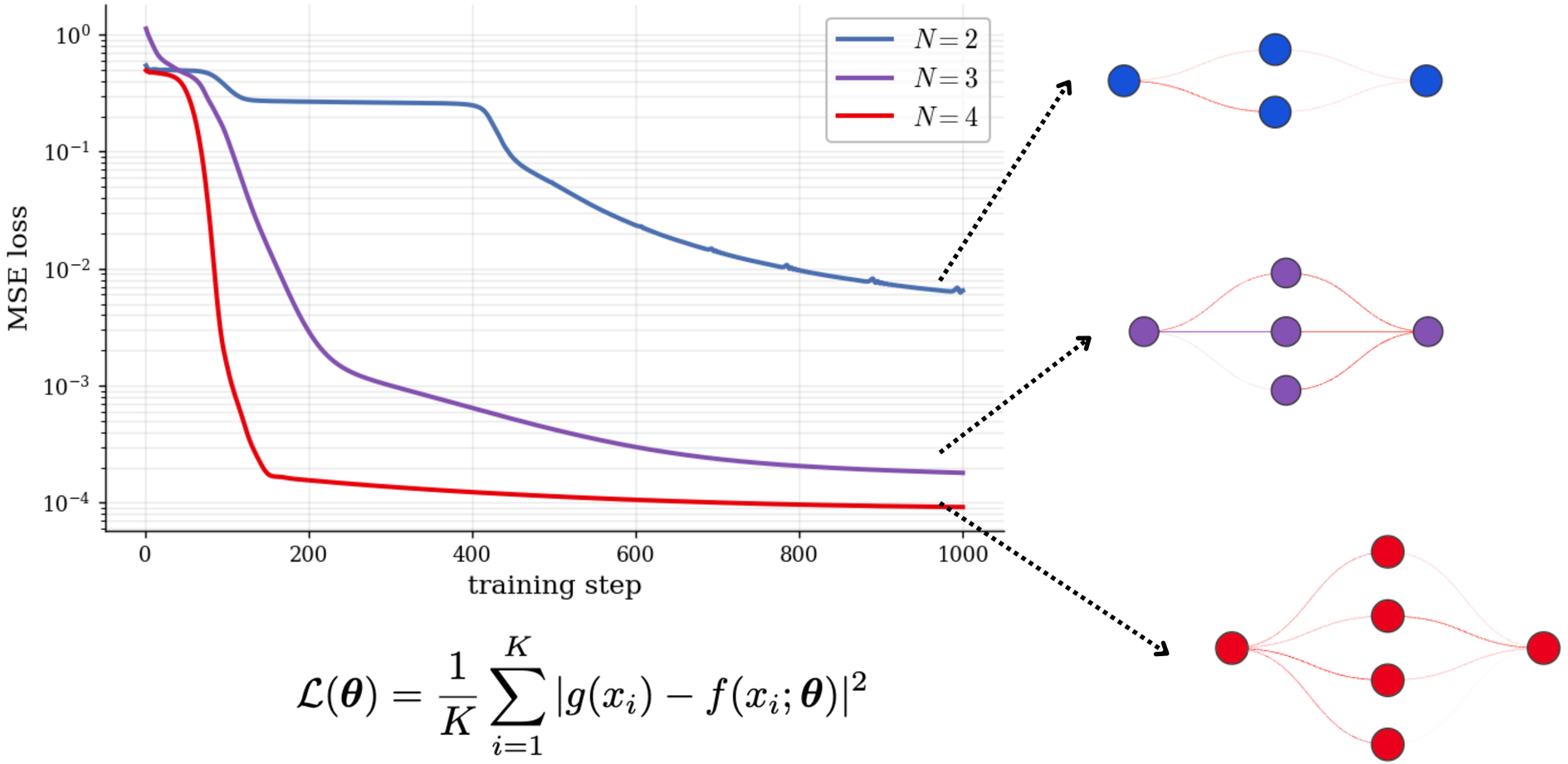
Input



$$\mathcal{L}(\theta)$$

A First Example

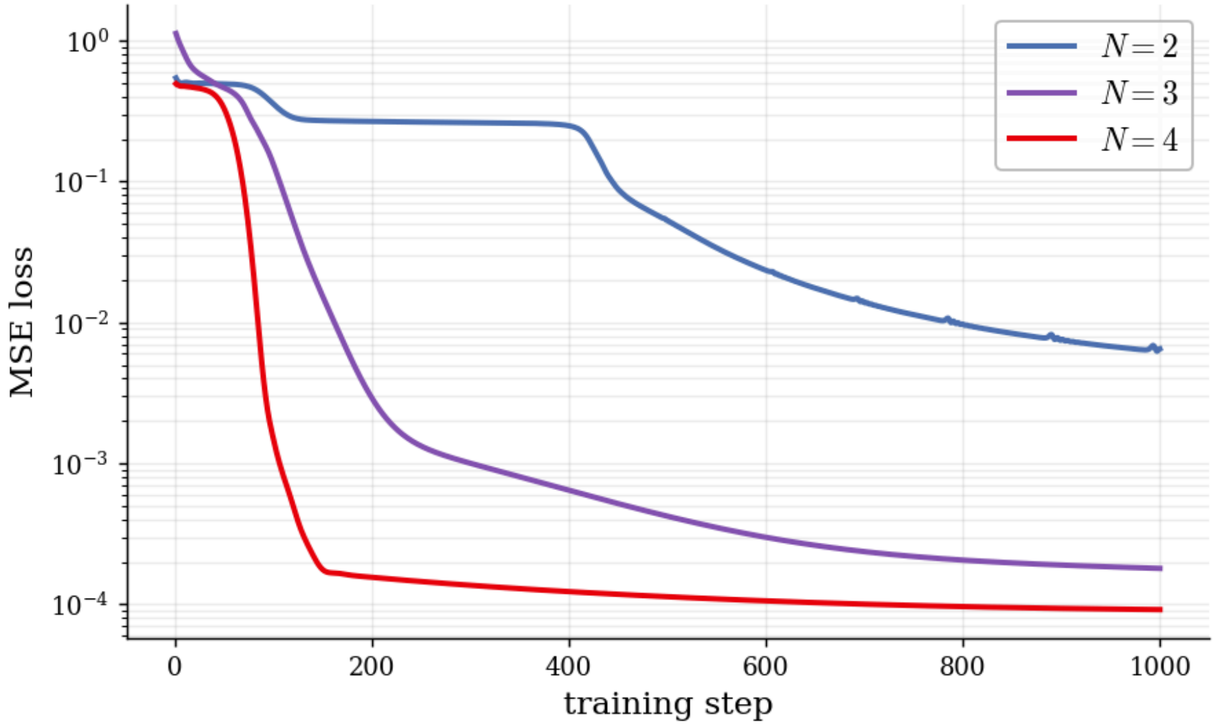
- After training

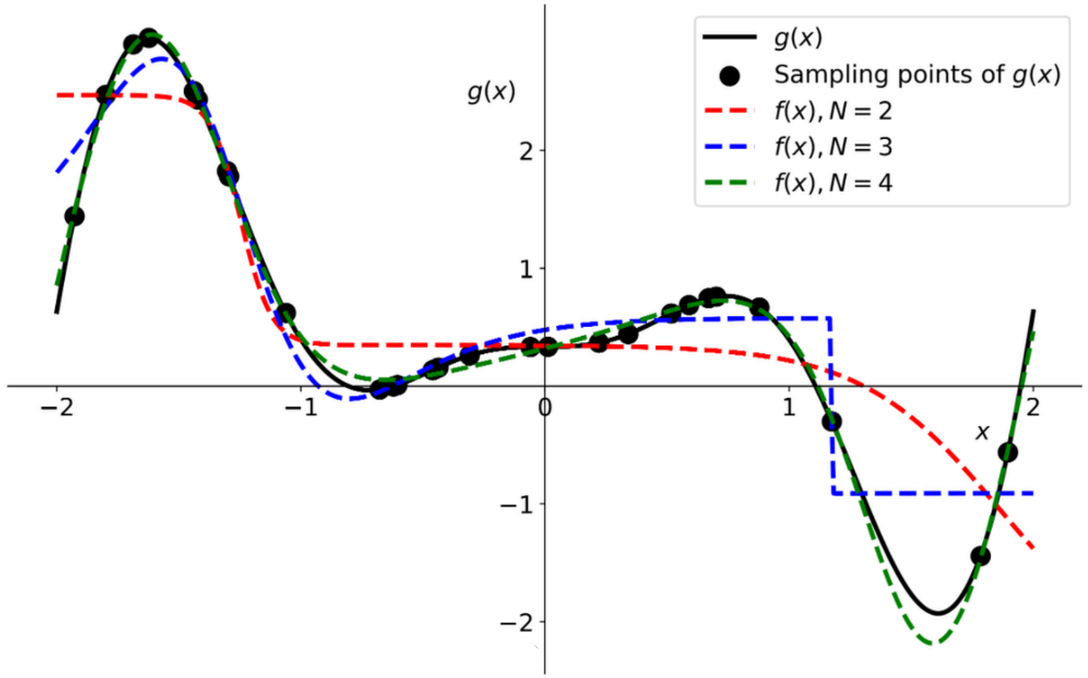


$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{i=1}^{K} |g(x_i) - f(x_i;\boldsymbol{\theta})|^2$$

A First Example

- After training





$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{K}\sum_{i=1}^{K} |g(x_i) - f(x_i;\boldsymbol{\theta})|^2$$

$$g(x) = x^2\sin(\pi x) - \frac{1}{2+\cos(x)}$$

Neural Network Anatomy and Training

1. **Forward Pass**

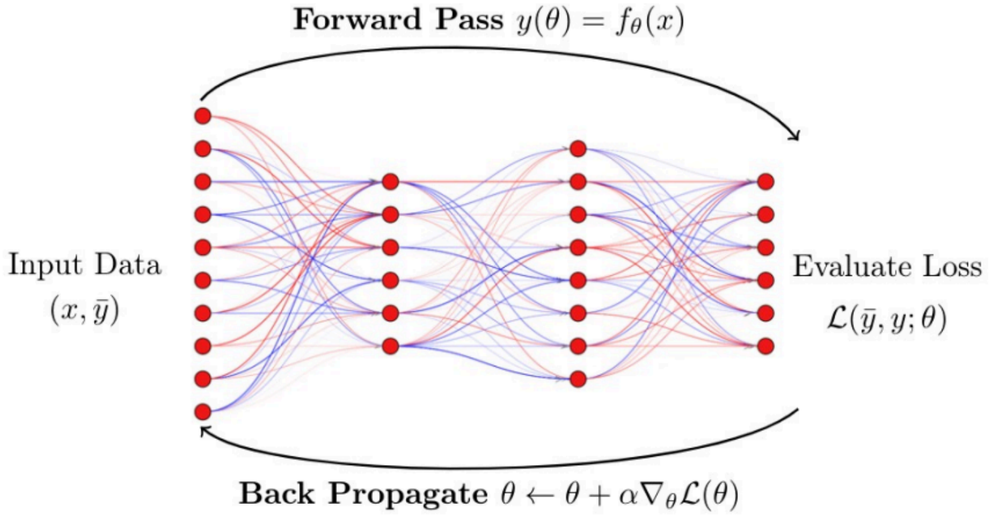
$$\bar{\vec{y}}_i^{(j)} = f_{\vec{\theta}_t}(\vec{x}_i^{(j)})$$

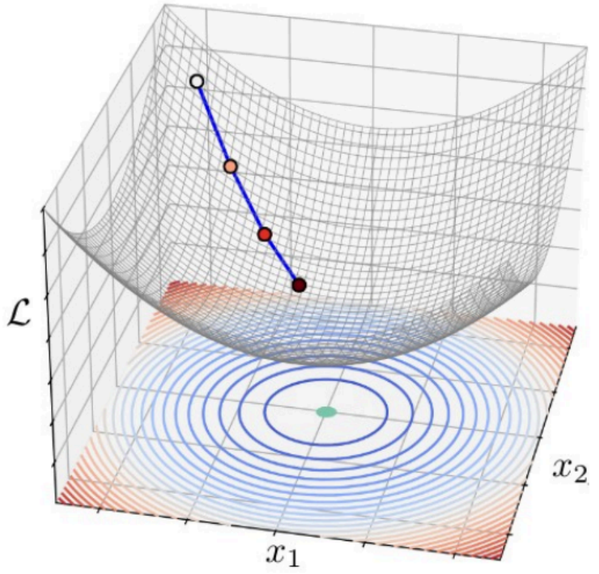
2. **Loss calculation**

$$\mathcal{L}^{(j)}(\vec{\theta}_t) = \frac{1}{N_{\mathcal{D}^{(j)}}} \sum_{i=1}^{N_{\mathcal{D}^{(j)}}} \mathcal{L}_i(\vec{y}_i^{(j)}, \bar{\vec{y}}_i^{(j)})$$

3. **Backpropagation**

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t - \eta \frac{\partial \mathcal{L}^{(j)}(\vec{\theta}_t)}{\partial \vec{\theta}_t}.$$





Seems simple enough right?...

- What about on more complicated functions?

$$g : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

- Or say.... wavefunctions...?

$$|\psi\rangle \in \mathcal{H}$$

- Can we say a network is training well and learning a function correctly based on loss function alone?

Beyond the loss curve

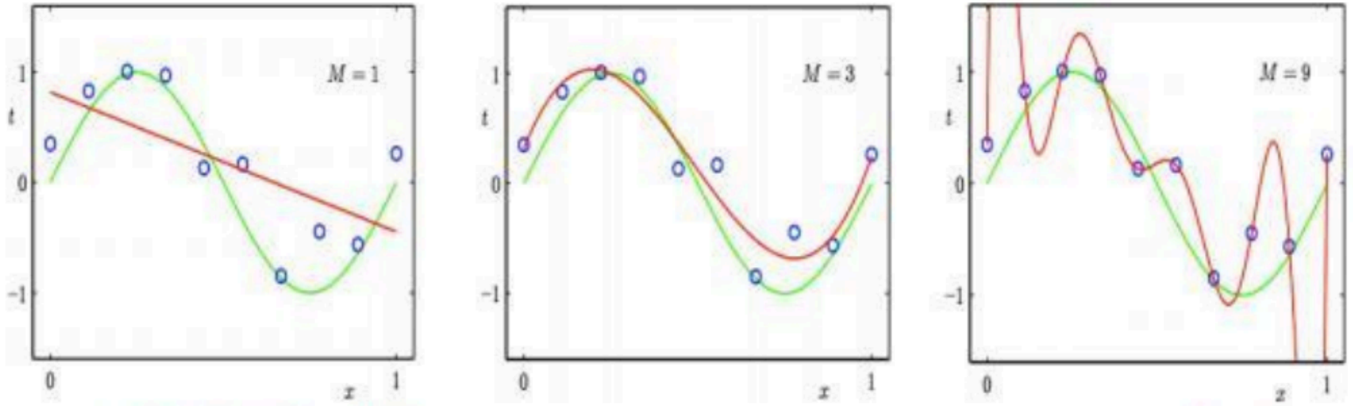
- Loss is going down, is my model actually “good”?
- I “tweaked it” and loss improved… did it really, or did I just get lucky?
- Which network should I use?

CONTENT

CONTENT

Overfitting Problem

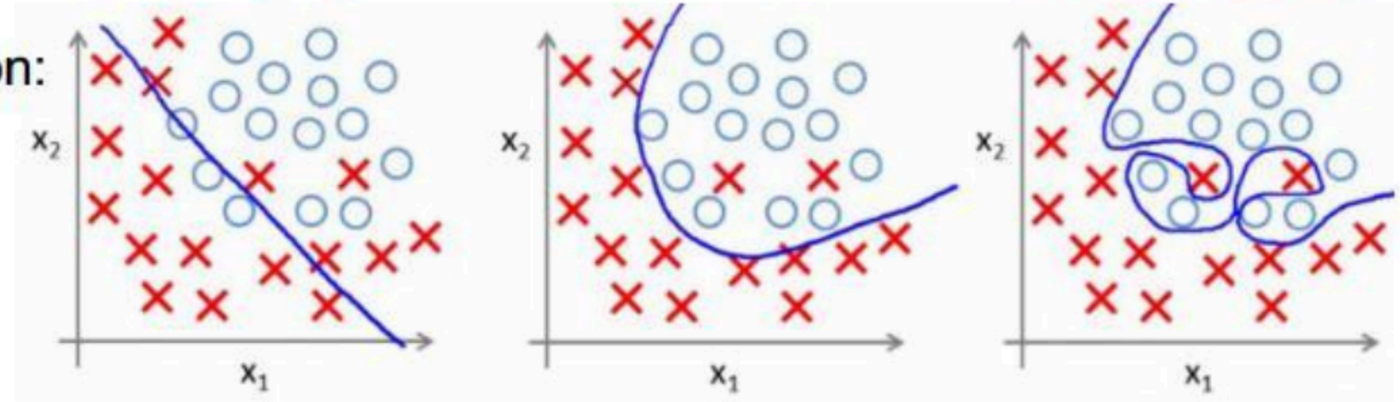
Regression:



predictor too inflexible:
cannot capture pattern

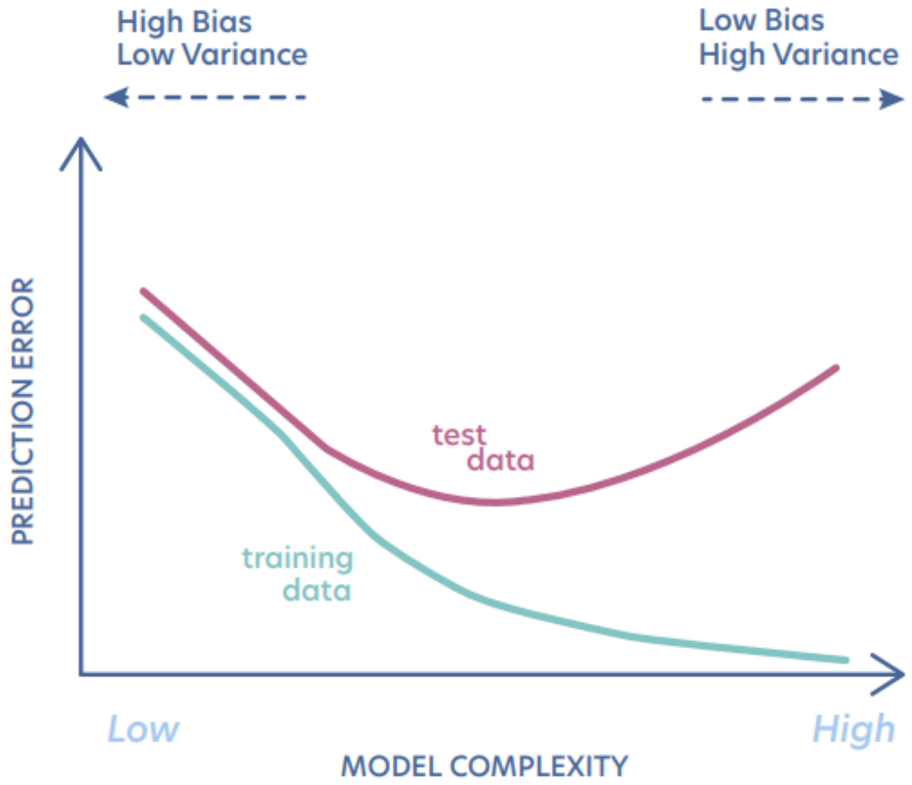
predictor too flexible:
fits noise in the data

Classification:



Overfitting Problem

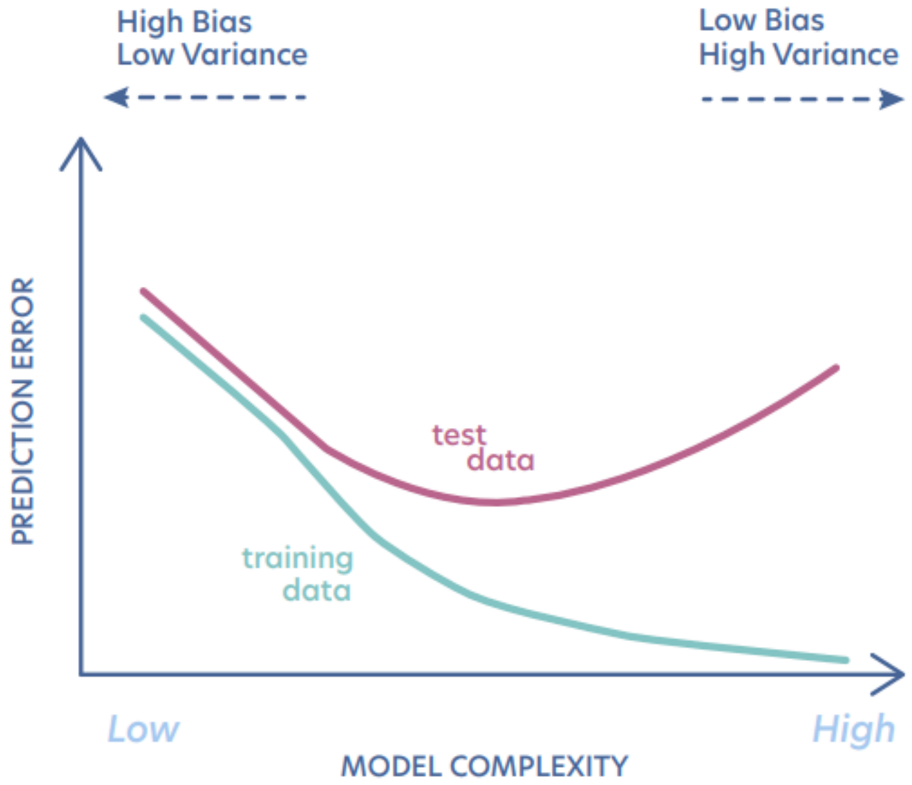
- Prevent overfitting by splitting data into training, validation and testing sets



- Detect it by seeing error decrease on train set *only*...

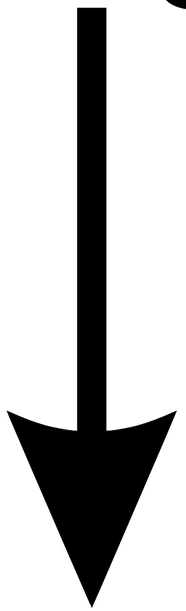
Overfitting Problem

- Prevent overfitting by splitting data into training, validation and testing sets



- Detect it by seeing error decrease on train set *only*...
- This happens because networks can learn the trend *and the noise*... *effectively memorising training data*

Overfitting Problem



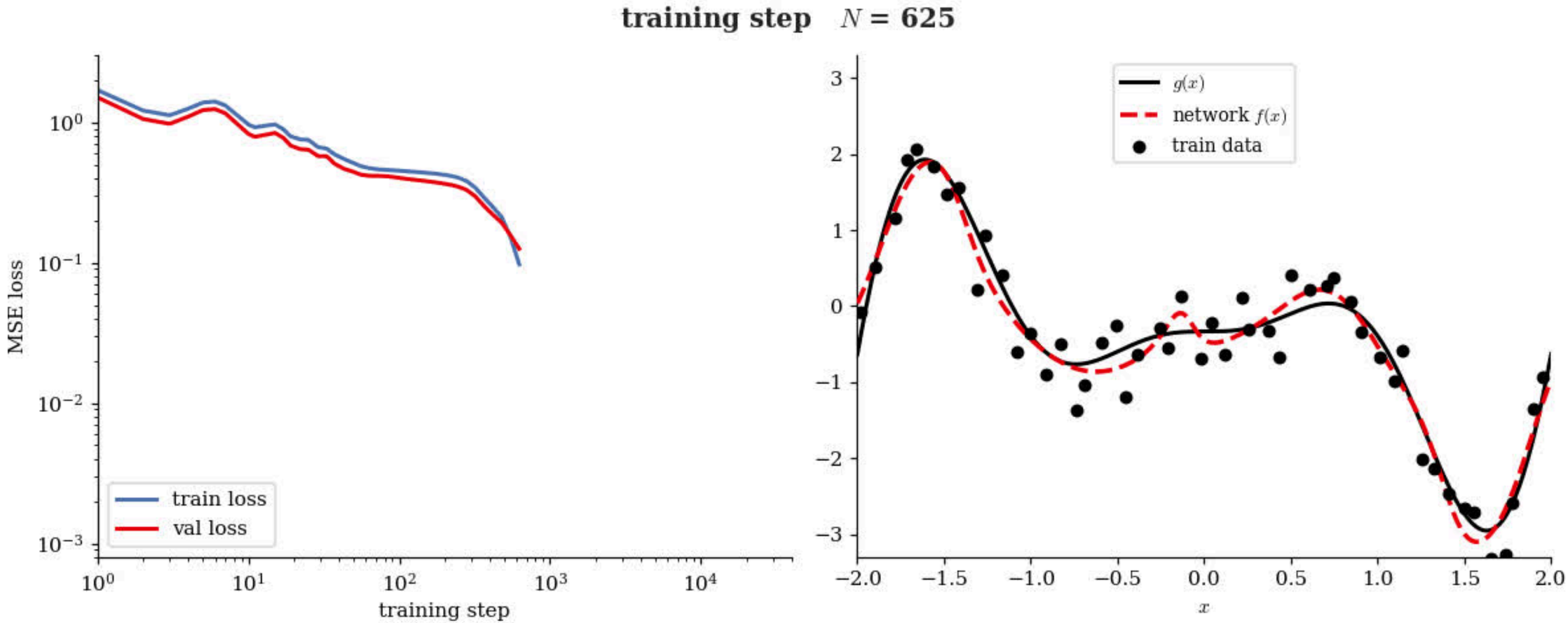
3 solutions...

1 Regularisation

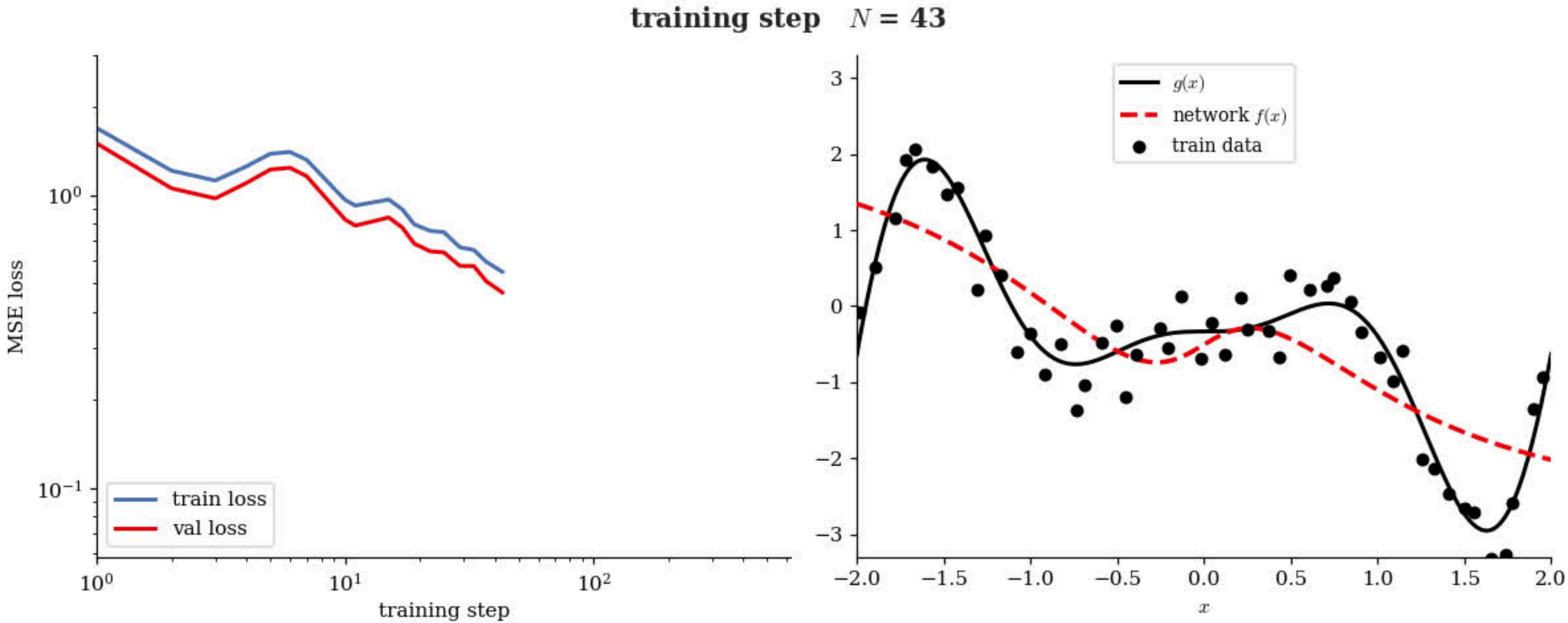
- Any term that penalises complexity, so driving the loss down no longer rewards memorising... $R = |\theta|$
- Creates composite loss functions...

$$\mathcal{L} = \mathcal{L}_{data} + \lambda R$$

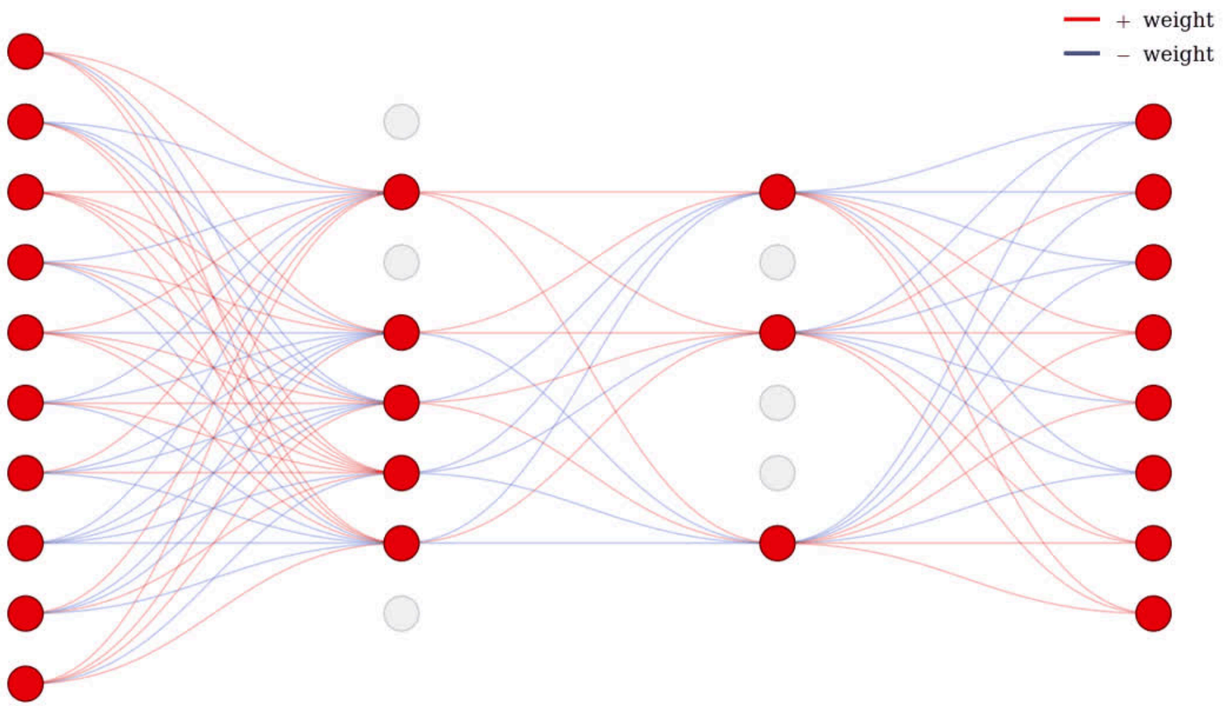
2 Early stopping



2 Early stopping



3 Dropout

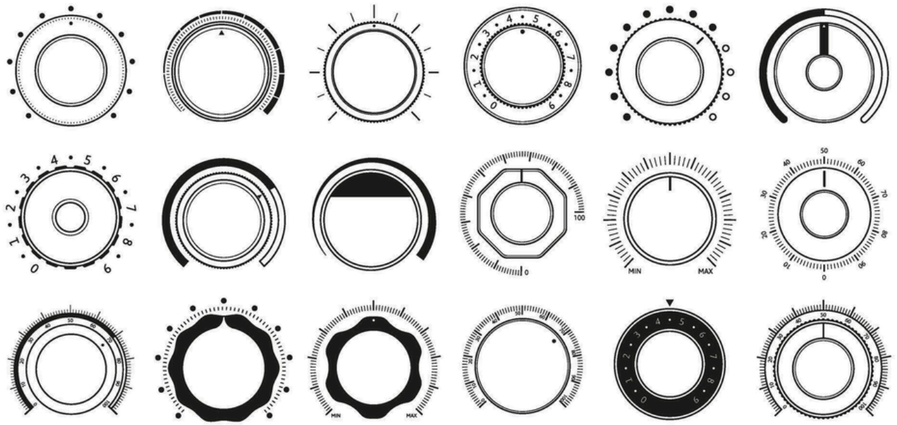


39%| | 233/600

- Randomly switch off sets of parameters during training

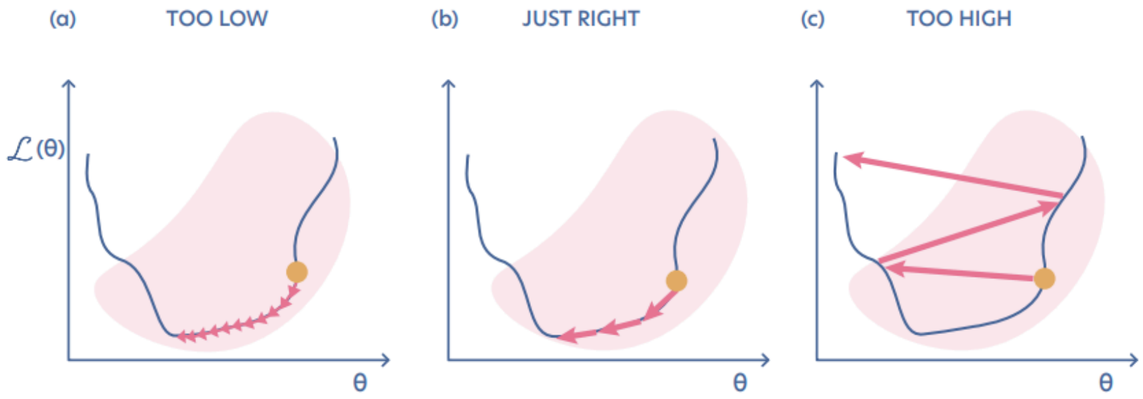
Hyperparameters...

- A hyperparameter is any part of a model we can tweak that is not touched by backpropagation
    - Learning rate
    - Network shape
    - Choice of activation function
    - Encoding of input data
    - Choice of Optimiser



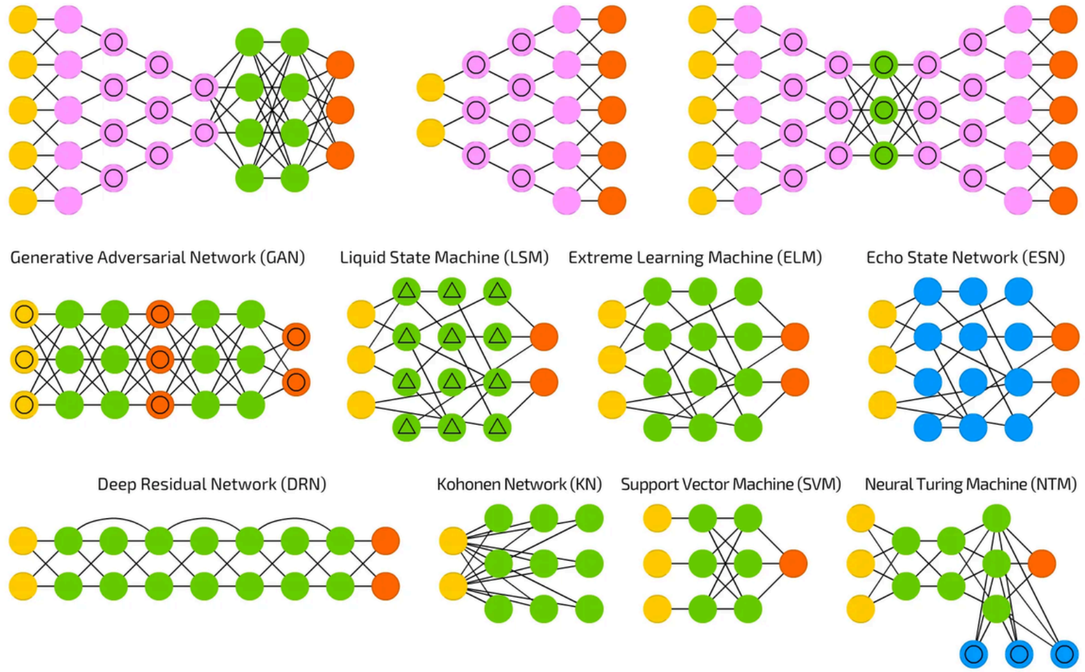
Hyperparameters...

- A hyperparameter is any part of a model we can tweak that is not touched by backpropagation
    - **Learning rate**
    - Network shape
    - Choice of activation function
    - Choice of Optimiser



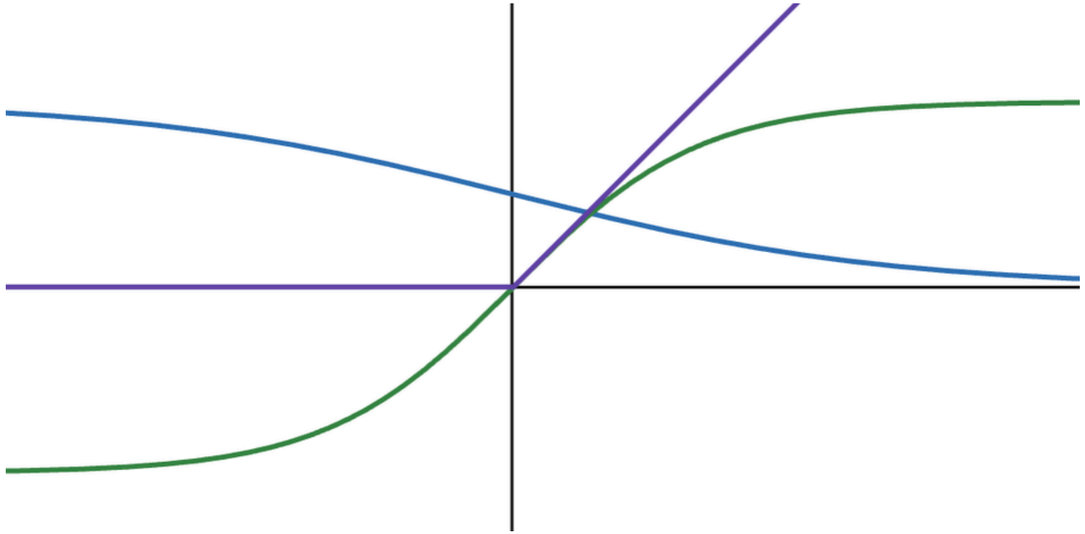
Hyperparameters...

- A hyperparameter is any part of a model we can tweak that is not touched by backpropagation
    - Learning rate
    - **Network shape**
    - Choice of activation function
    - Choice of Optimiser



Hyperparameters...

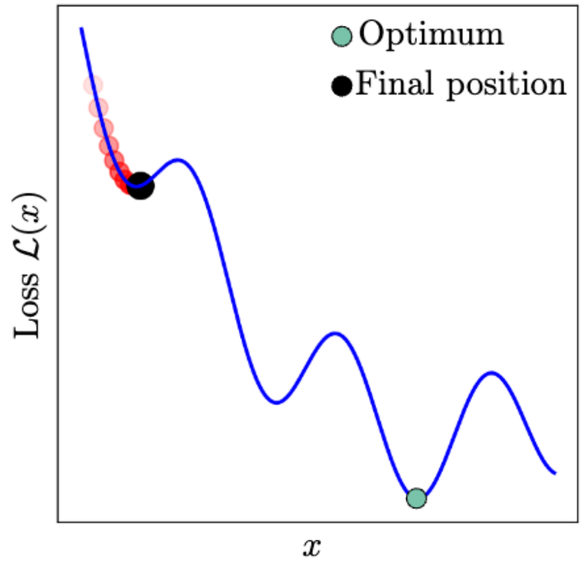
- A hyperparameter is any part of a model we can tweak that is not touched by backpropagation
    - Learning rate
    - Network shape
    - **Choice of activation function**
    - Choice of Optimiser

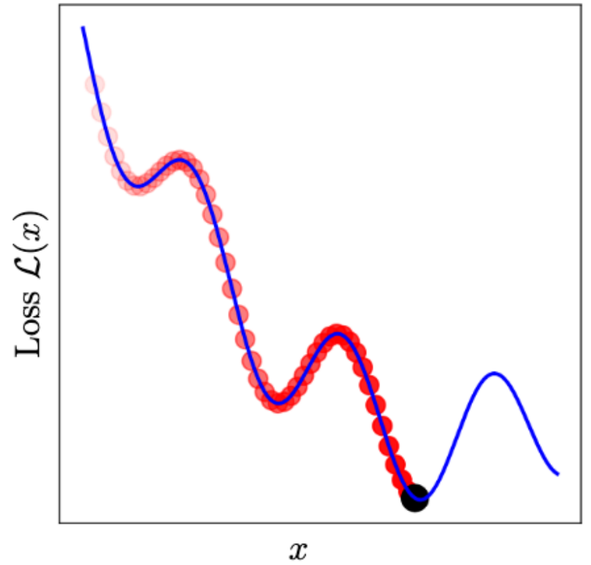


Hyperparameters...

- A hyperparameter is any part of a model we can tweak that is not touched by backpropagation
    - Learning rate
    - Network shape
    - Choice of activation function
    - **Choice of Optimiser**

    (vanilla, momentum, ADAM, SPFS, Kron, Spring, KFAC...)



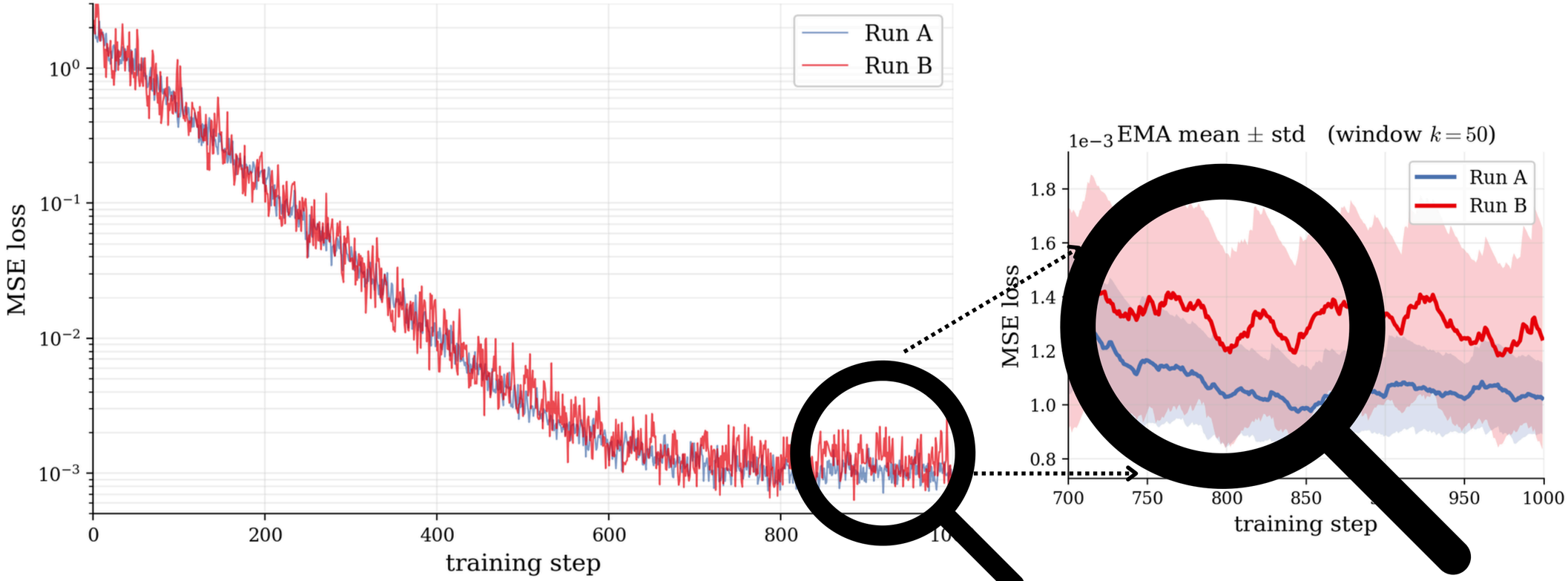


Data-driven Network Design

- Suppose we have two reasonable looking loss curves, A and B, both with reasonable loss curves, which is better?

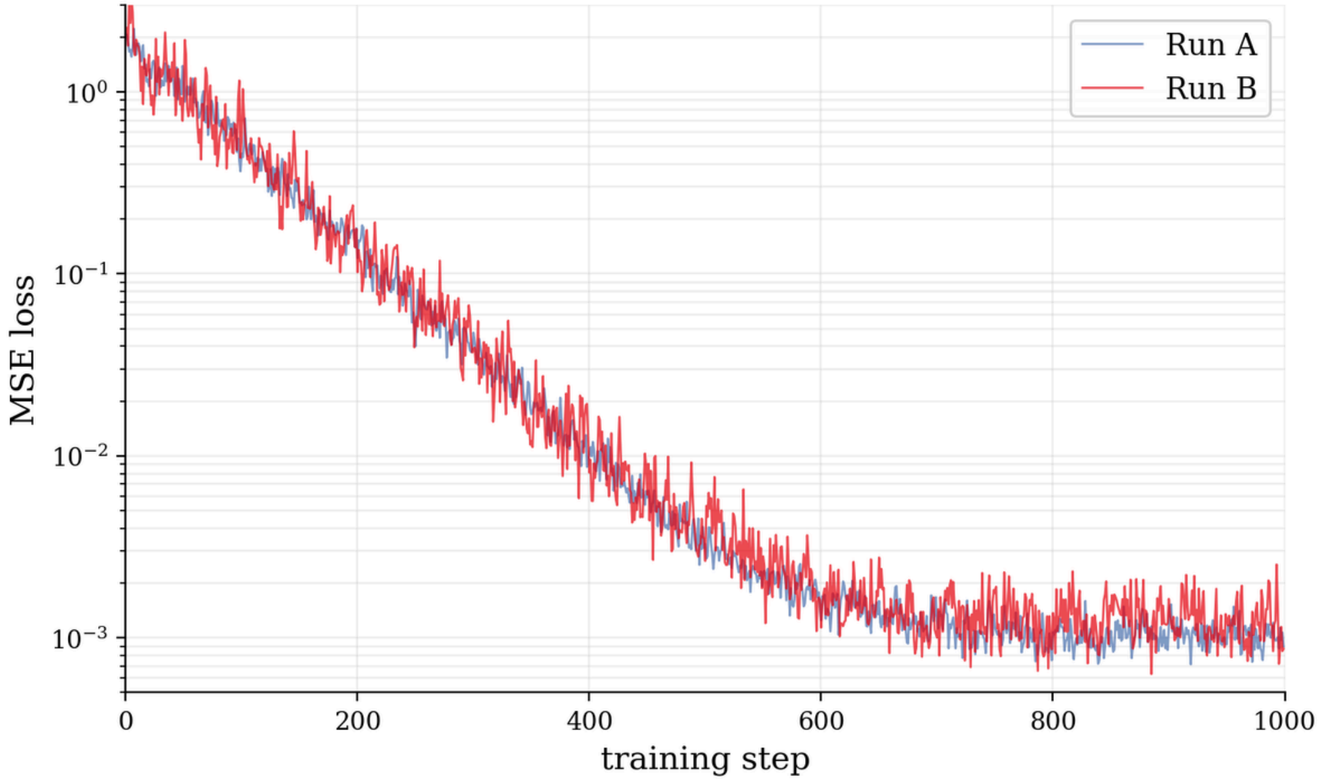
Data-driven Network Design

- Suppose we have two reasonable looking loss curves, A and B, both with reasonable loss curves, which is better?
  - Avoid comparing two runs by their final-step loss due to minibatch noise

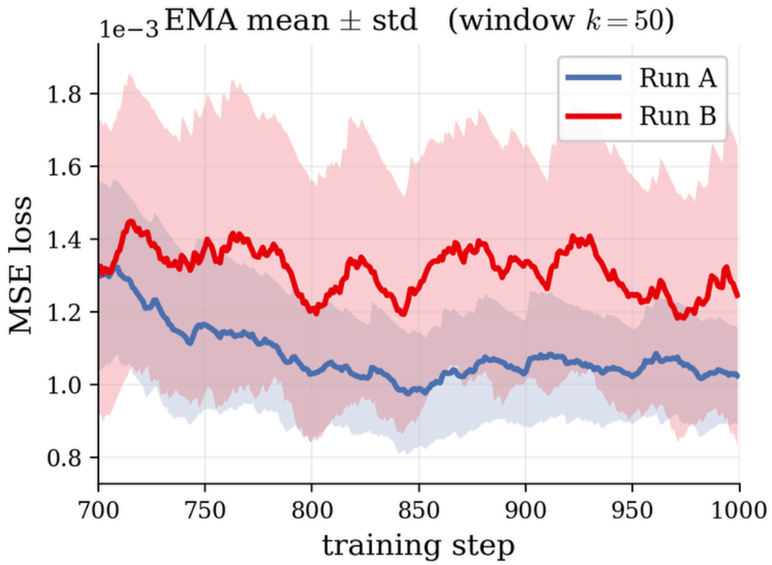


Data-driven Network Design

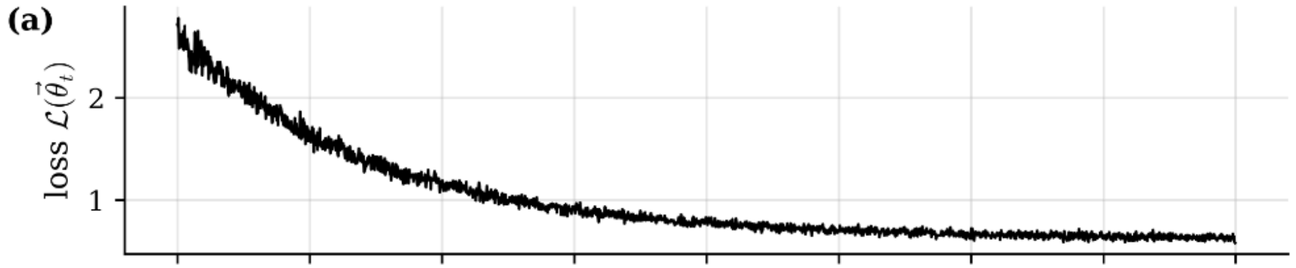
- Suppose we have two reasonable looking loss curves, A and B, both with reasonable loss curves, which is better?
  - Better to compute the **average loss over a window** of K later steps



$$\hat{L} = \frac{1}{K} \sum_{t=T-K+1}^{T} \mathcal{L}(\boldsymbol{\theta}_t), \qquad T_0 \leq T - K + 1,$$

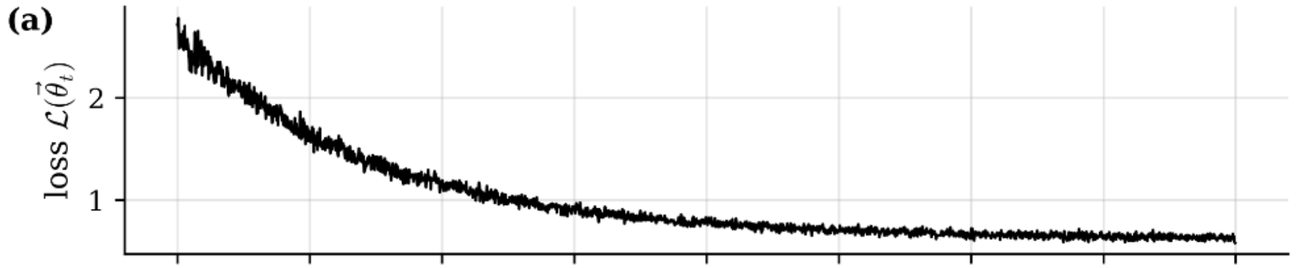


Data-driven Network Design

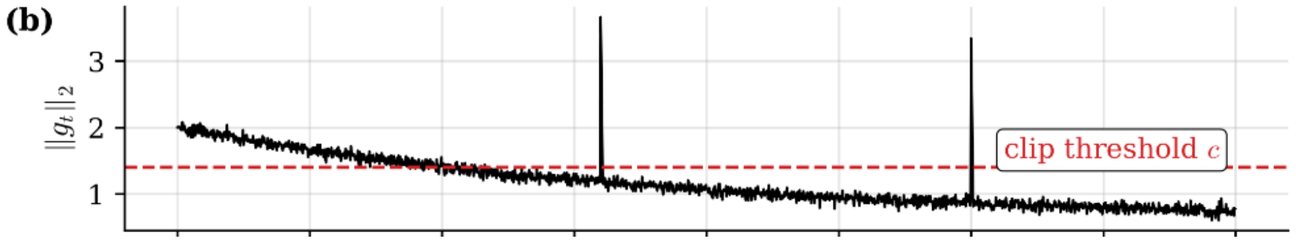


- Good idea to monitor the gradient *norm* during training steps $g_t = ||\nabla_\theta||_2$

Data-driven Network Design

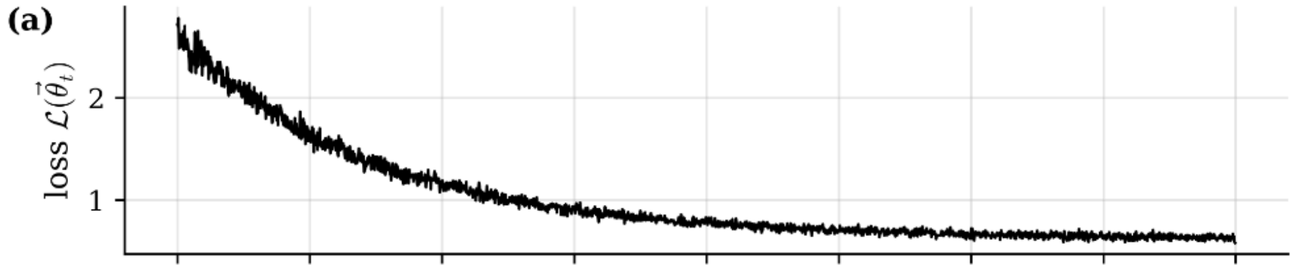


- Good idea to monitor the gradient *norm* during training steps $g_t = ||\nabla_\theta||_2$

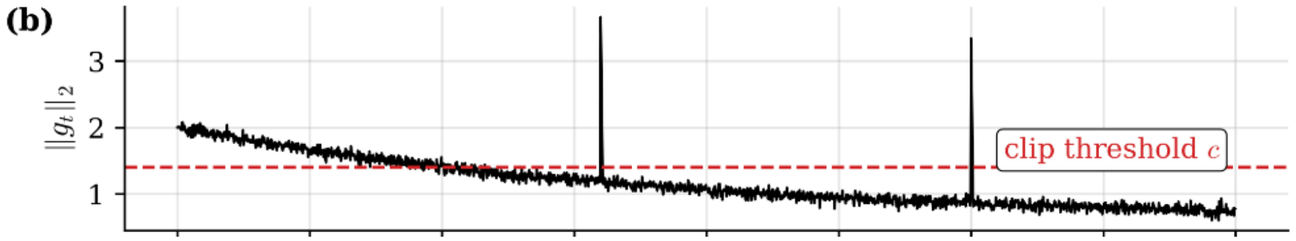


- We sometimes get spikes…

Data-driven Network Design



- Good idea to monitor the gradient *norm* during training steps $g_t = ||\nabla_\theta||_2$

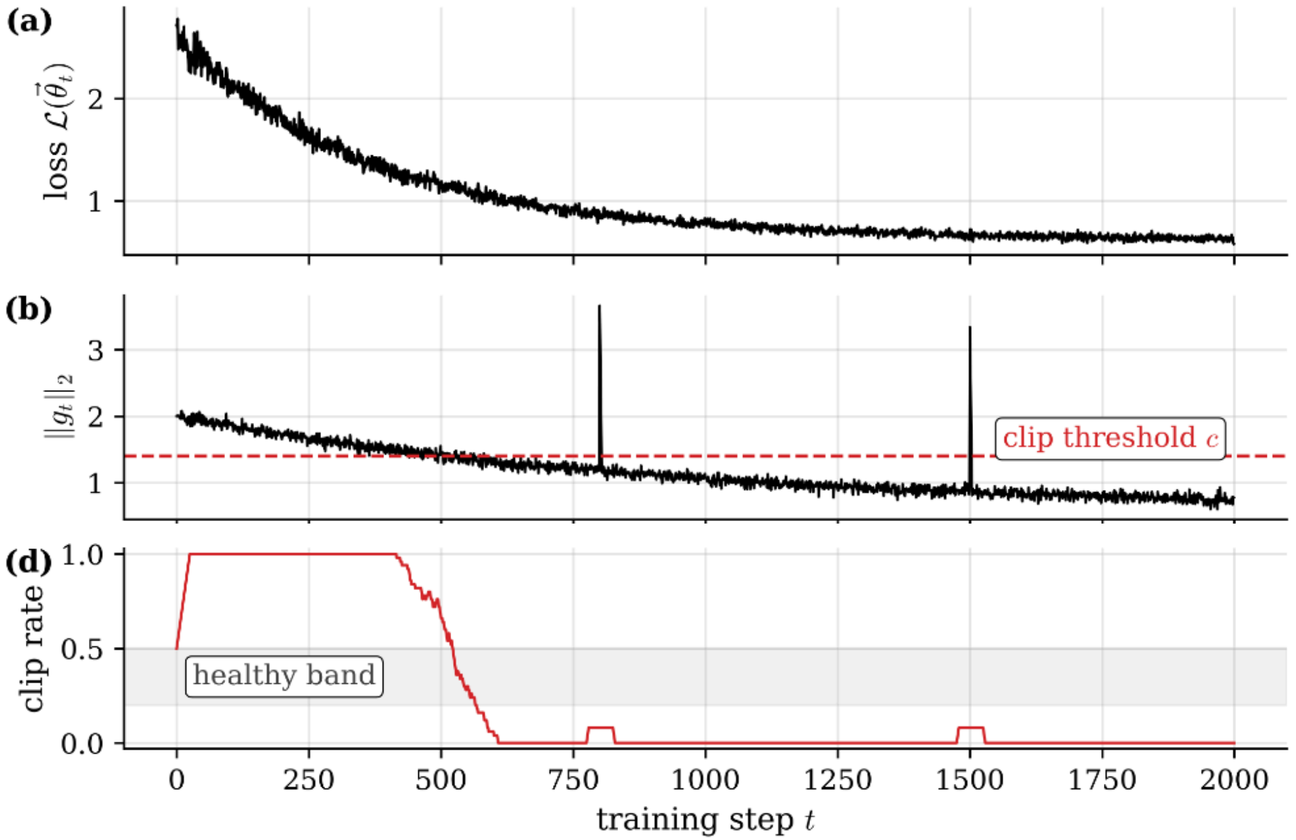


- We sometimes get spikes...

- These can affect update steps $\theta \leftarrow \theta - \alpha \nabla_\theta L$

Data-driven Network Design

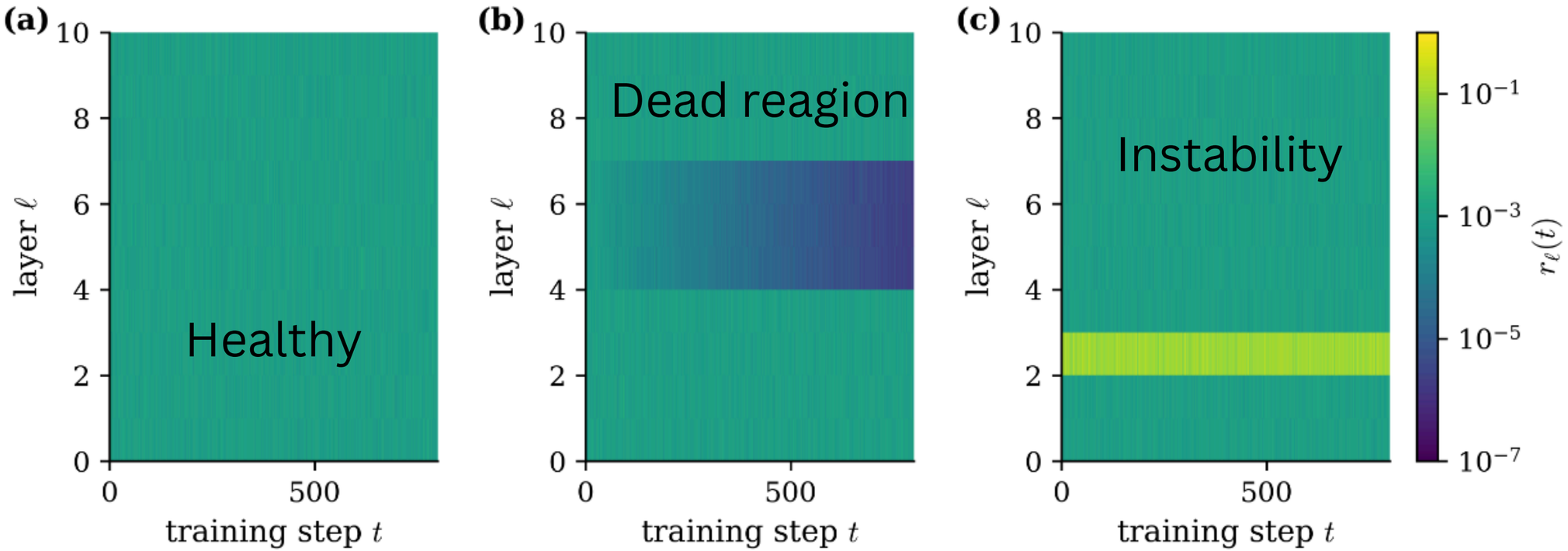
- **Gradient Clipping** can regularise *step-size*



- Careful not to under- or over-clip!

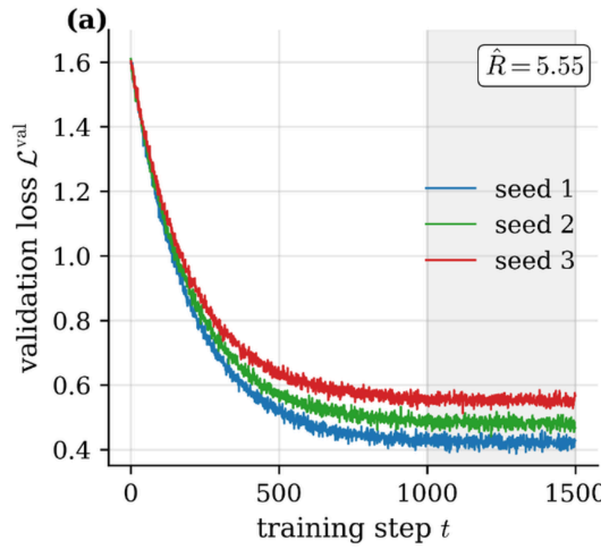
Data-driven Network Design

Per-layer update-to-weight ratio



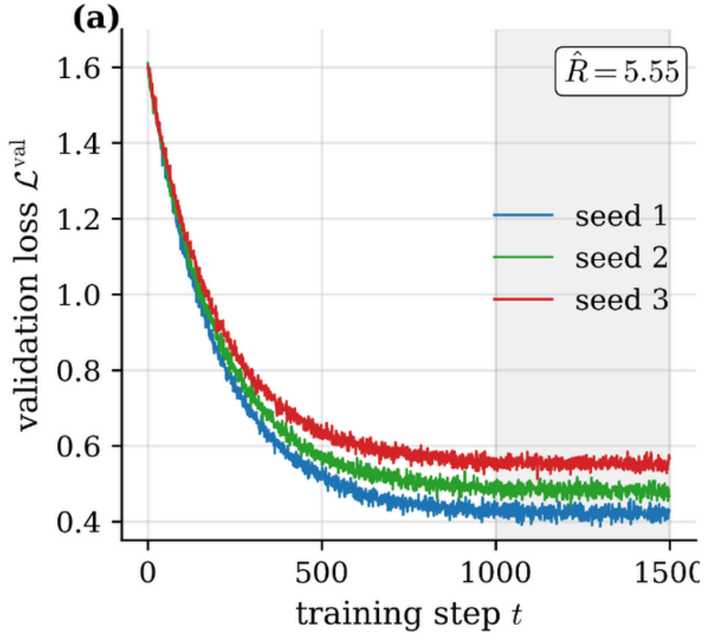
Luck vs Progress

- To claim one *architecture* is better than another, we want to see it outperforming on many random seeds.
- How can we tell if one model is more or less responsive to seed-luck?
  - Using the tail window of length M from each seed



Luck vs Progress

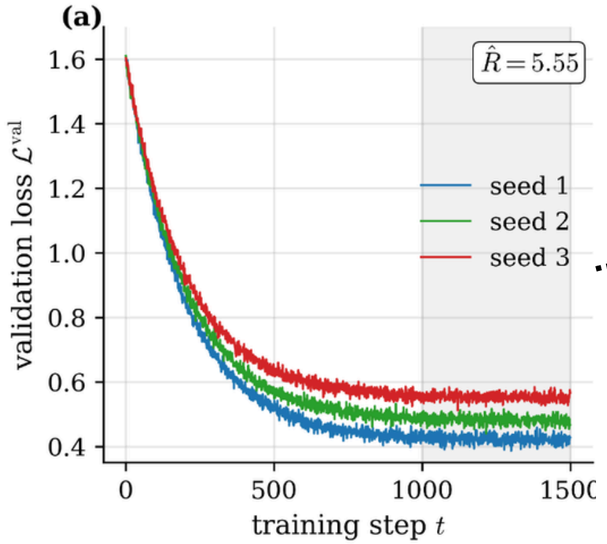
- To claim one *architecture* is better than another, we want to see it outperforming on many random seeds.
- How can we tell if one model is more or less responsive to seed-luck?
  - Using the tail window of length M from each seed



- Compute Gelman-Rubin Metric
  - (we will do this in the practical!)

Luck vs Progress

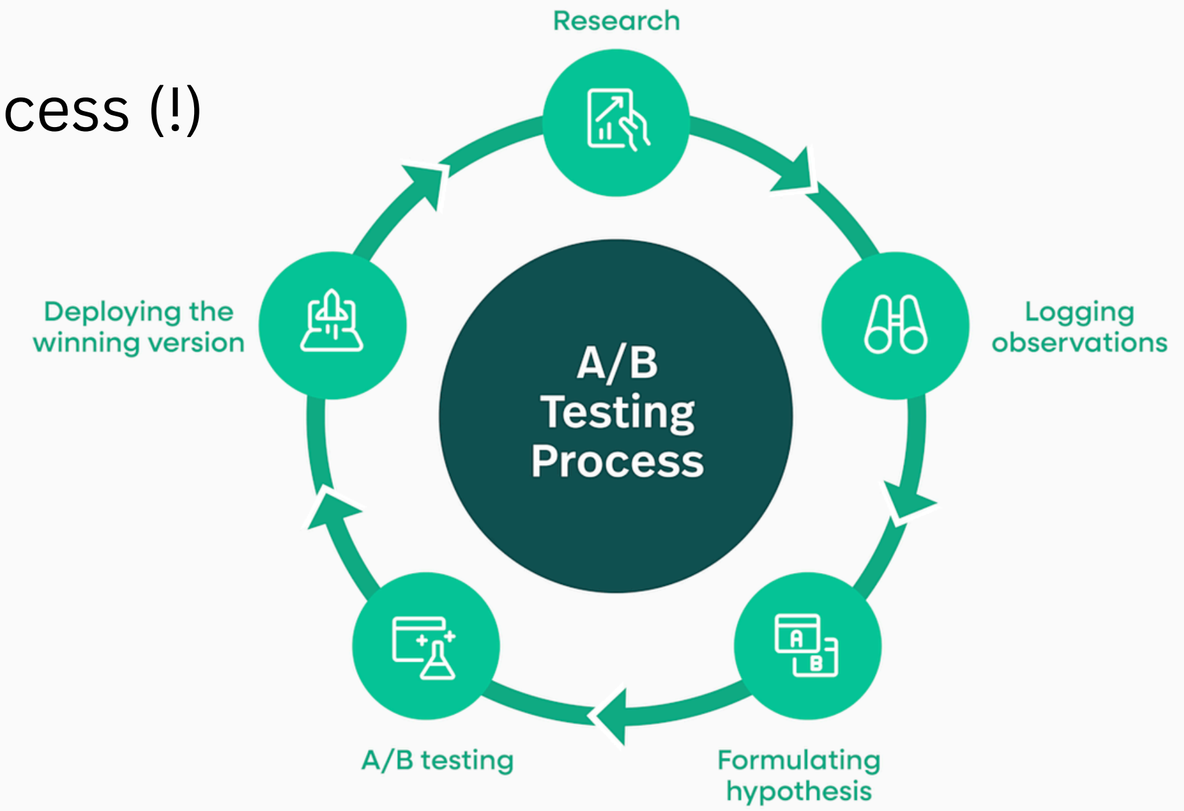
- To claim one *architecture* is better than another, we want to see it outperforming on many random seeds.
- How can we tell if one model is more or less responsive to seed-luck?
  - Using the tail window of length M from each seed



$$\hat{L}_m \qquad s_m^2 \qquad W = \frac{1}{M} \sum_{m=1}^{M} s_m^2,$$

Agentic Architecture Search

- Tests are good gates for coding agents to find architectures
- This can automate the tuning process (!)



Data-driven Network Design

- Learning rate
- EMA-windows
- Gradient clipping
- Update ratios
- Seed safety

} Shown (very!) briefly today

- Batch norm
- Data encoding
- Layer norms
- Types of optimizer
- Layer eigen-spectra
- Effective step-sizes

} Not shown but worth understanding (!)

Data-driven Network Design

- Learning rate
- EMA-windows
- Gradient clipping
- Update ratios
- Seed safety

} Shown (very!) briefly today

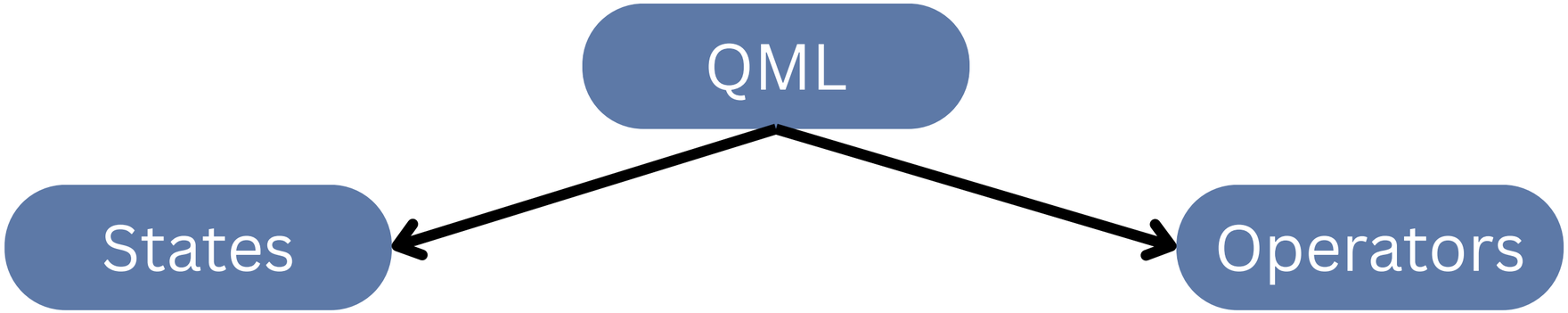
- Batch norm
- Data encoding
- Layer norms
- Types of optimizer
- Layer eigen-spectra
- Effective step-sizes

} Not shown but worth understanding (!)

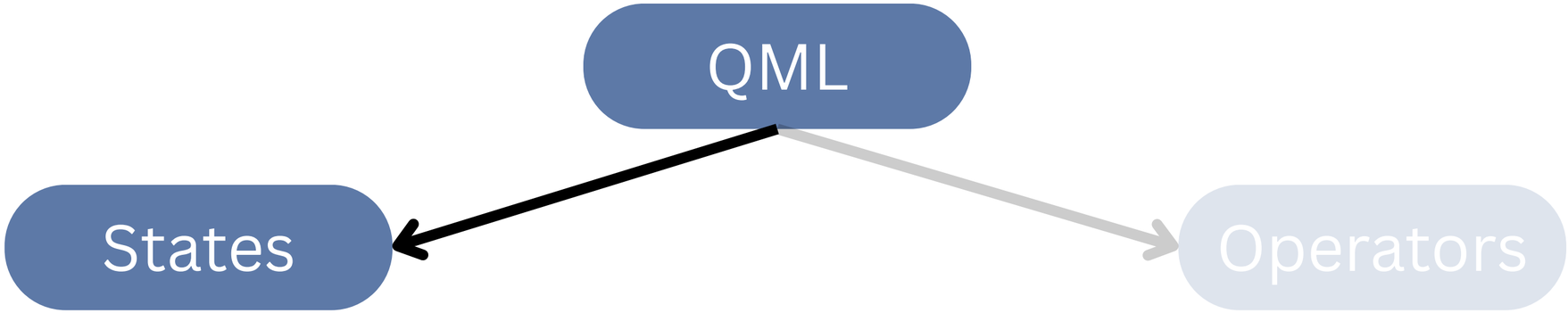
CONTENT

CONTENT

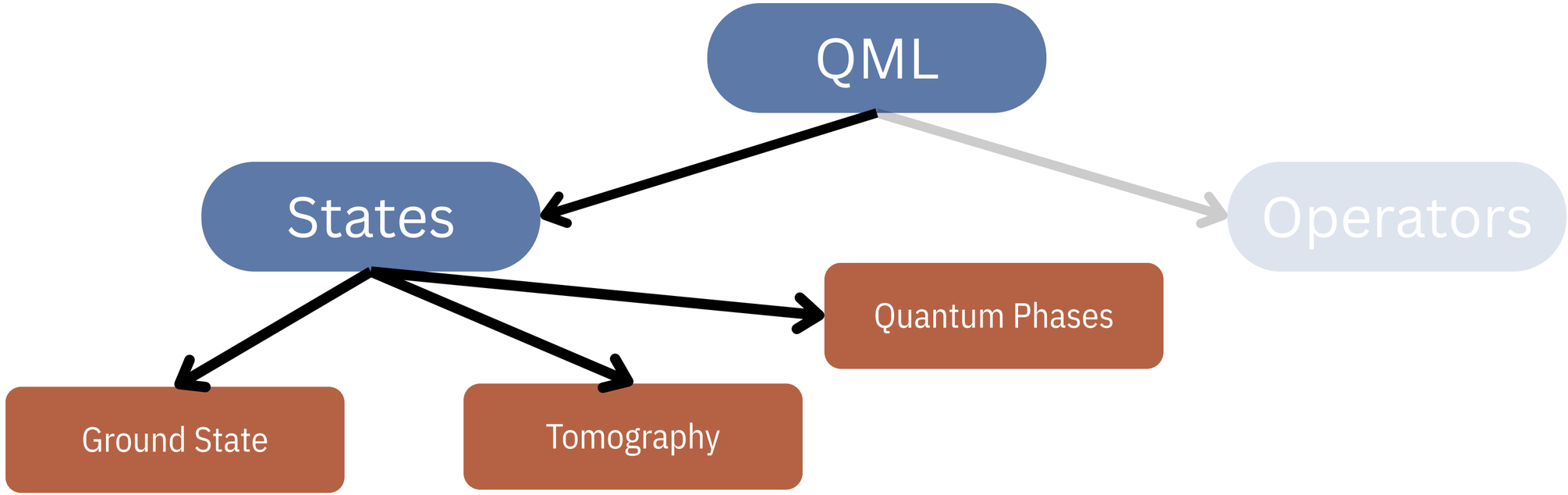
Quantum Machine Learning - Top Down View



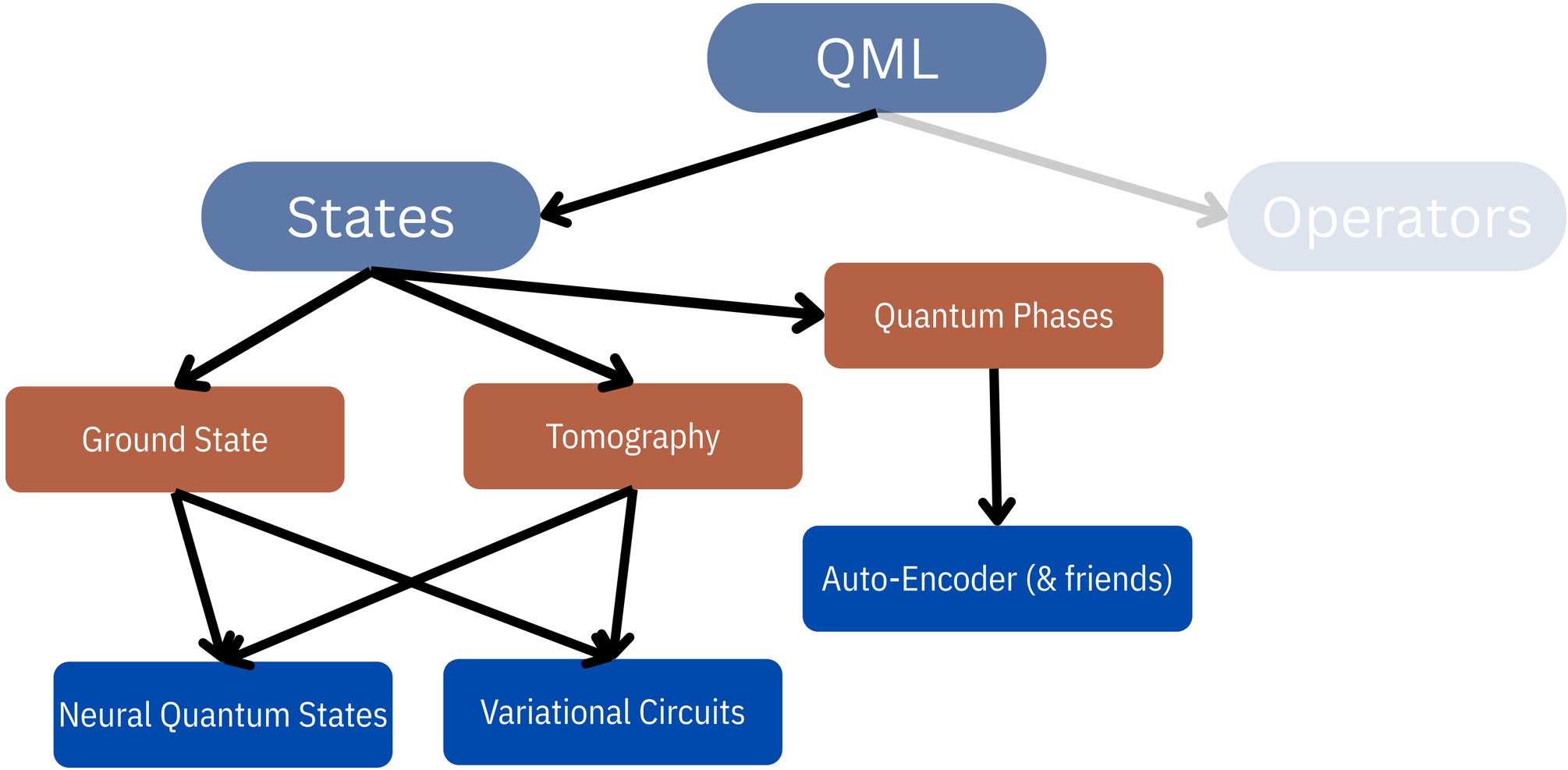
Quantum Machine Learning - Top Down View



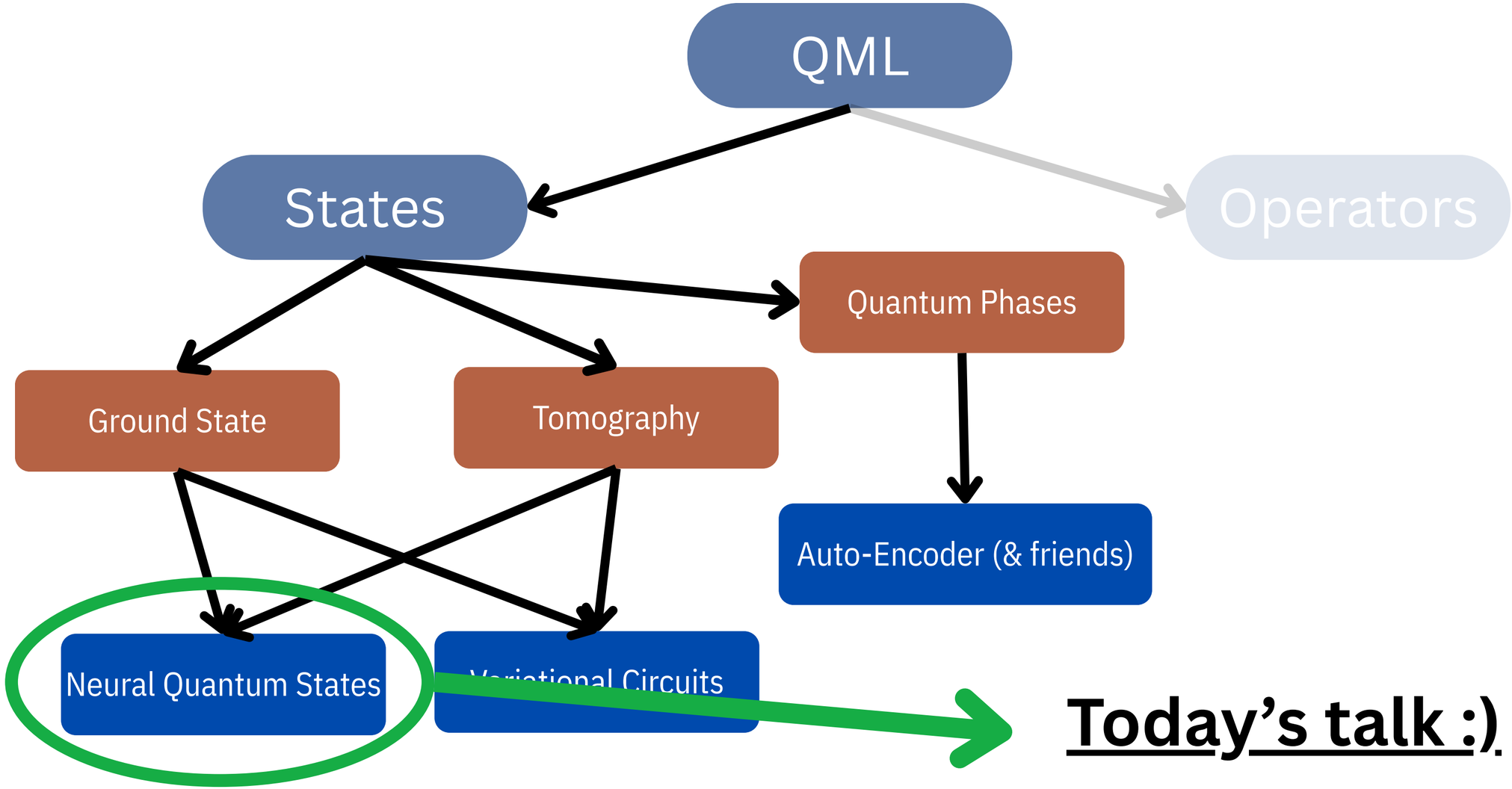
Quantum Machine Learning - Top Down View



Quantum Machine Learning - Top Down View

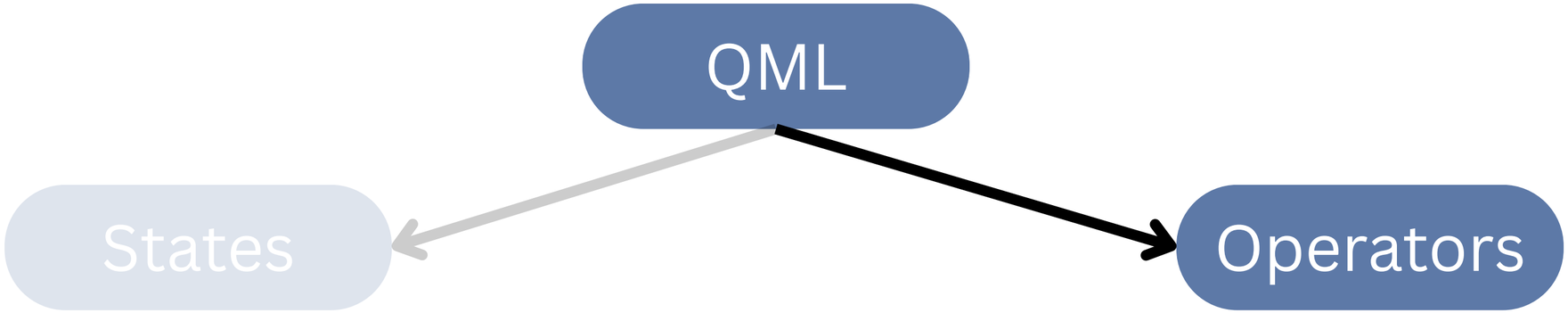


Quantum Machine Learning - Top Down View

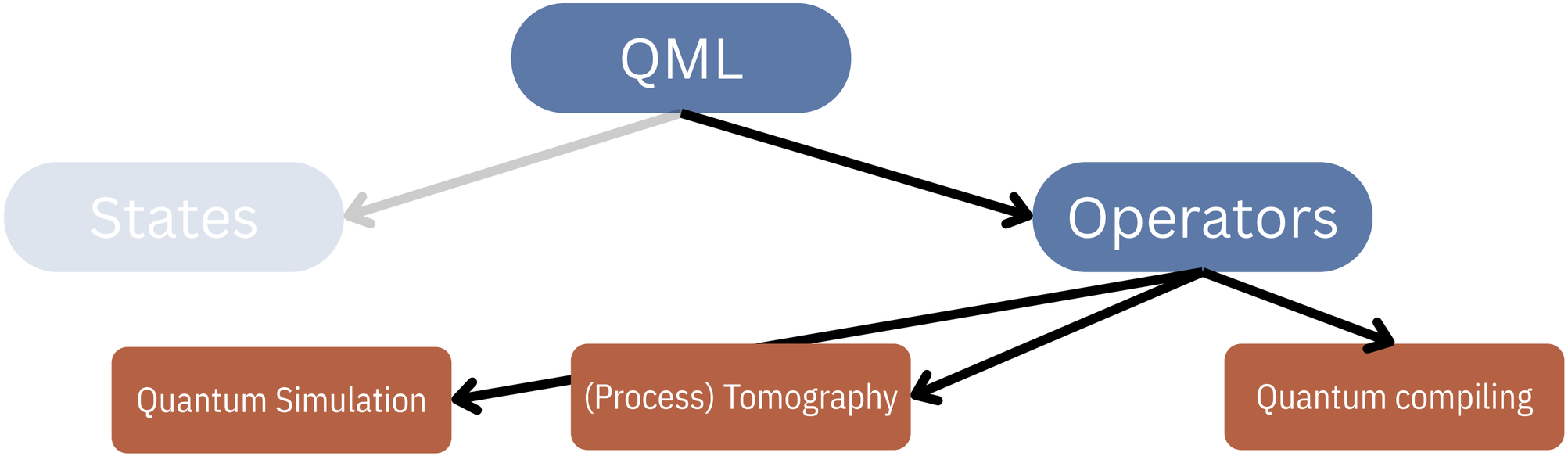


Today's talk :)

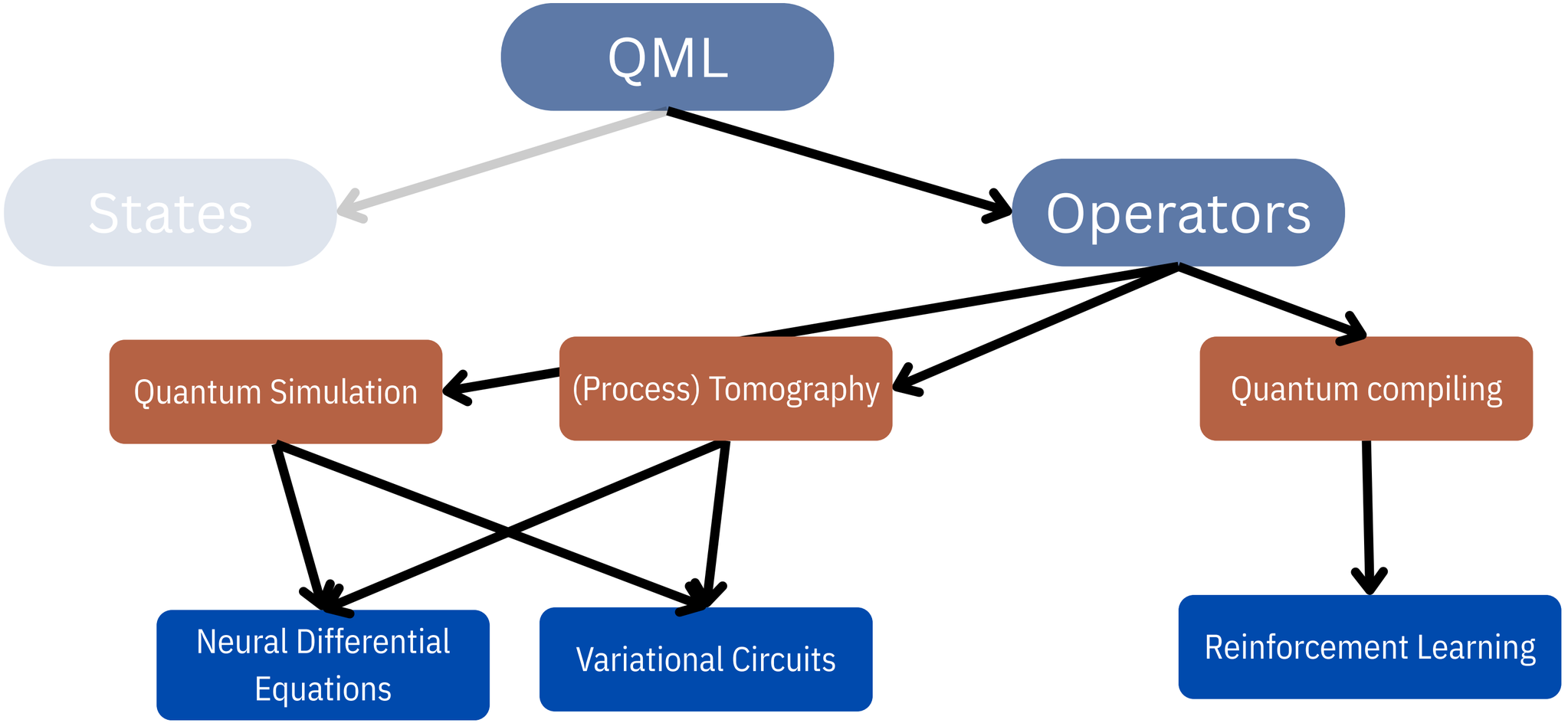
Quantum Machine Learning - Top Down View



Quantum Machine Learning - Top Down View



Quantum Machine Learning - Top Down View



NQS PART I

- Definitions
- Computing observables

WHAT DOES A WAVEFUNCTION DO ANYWAYS?

The Quantum Many-Body Problem

- Consider an *N*-qubit quantum state

$$|\psi(t)\rangle = \sum_{b_1,\ldots,b_N=0}^{1} c_{b_1,\ldots,b_N}(t)|b_1,\ldots,b_N\rangle$$

- Exponential number of coefficients to keep track of here

- Representing states and dynamics quickly becomes intractible with N...

Coefficients can themselves be functions…

- Consider an *N*-qubit quantum state

$$|\psi(t)\rangle = \sum_{b_1,\ldots,b_N=0}^{1} c_{b_1,\ldots,b_N}(t)|b_1,\ldots,b_N\rangle$$

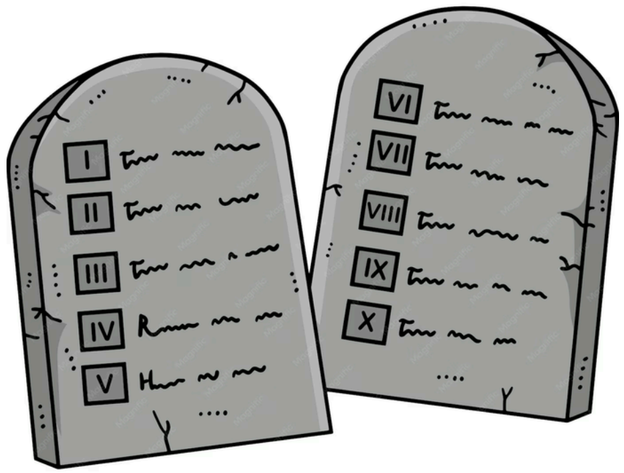
- Coefficient "function" $c : \{0,1\}^N \to \mathbb{C}$ receives a binary vector $\mathbf{b} \in \{0,1\}^N$ as input and spits out a complex number.

$$c(\mathbf{b}) \in \mathbb{C}$$

Neural Quantum States

- **Key idea:** use neural networks to approximate this function

$$\psi_\theta(\mathbf{b}) = \sqrt{p_\theta(\mathbf{b})} e^{i\phi_\theta(\mathbf{b})} \approx c(\mathbf{b})$$



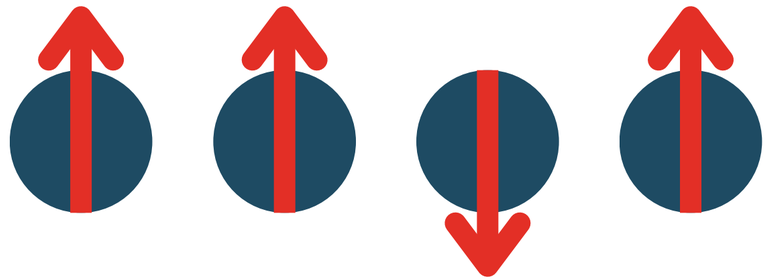
This works again thanks to the Universal Approximation Theorem - which says a large enough neural network can approximate *any* compact function given enough training and expressivity.

How does this work in practise?

Neural Quantum States

e.g. 4 quantum spin-1/2 chain (qubits)

$$|\psi\rangle = \sum_{j=\{0,1\}^4} c_{j_1,\ldots,l_4} |j_1, \ldots, j_4\rangle$$

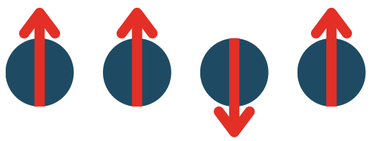


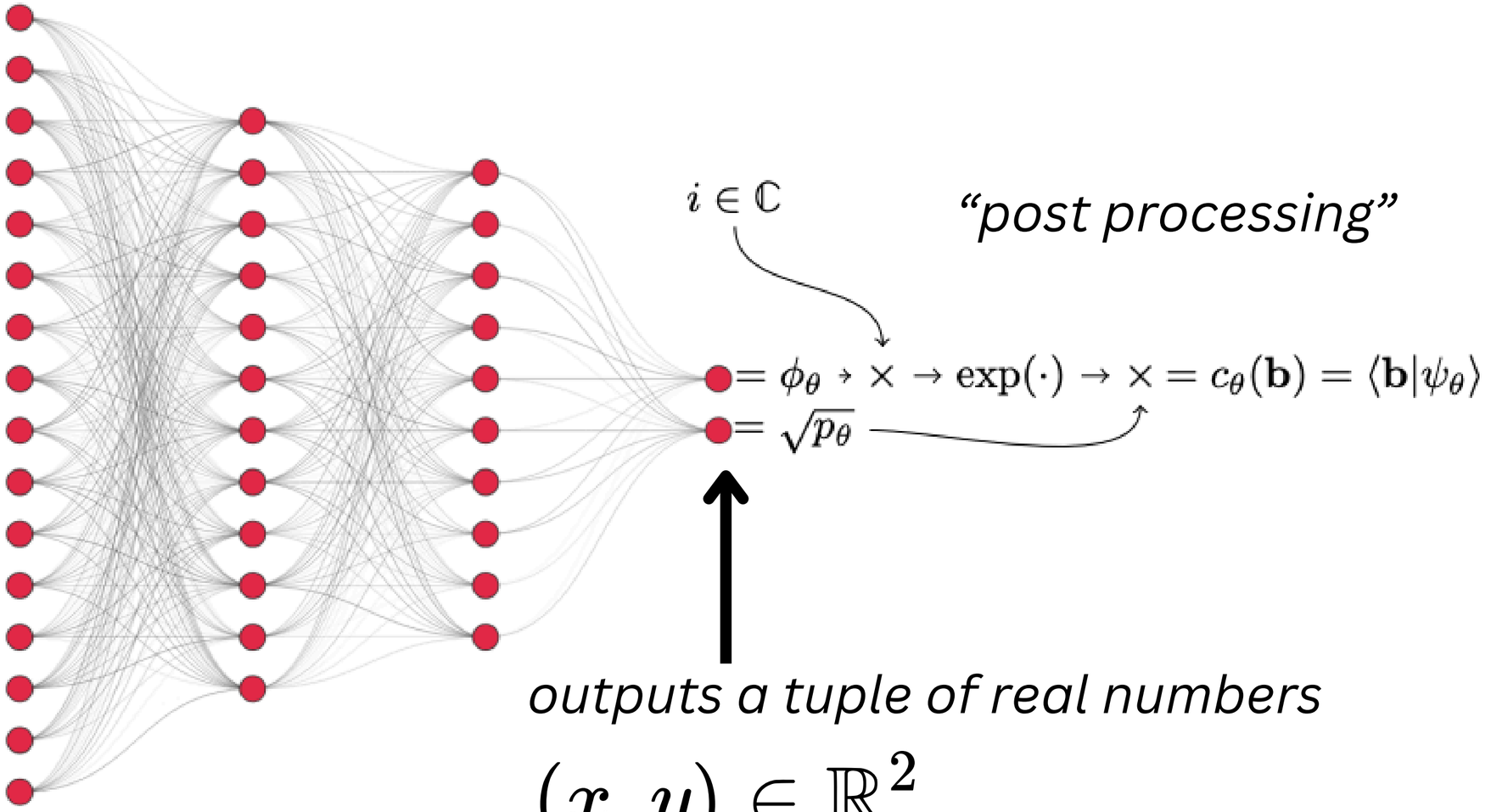
$$\psi_\theta(\mathbf{b}) = \sqrt{p_\theta(\mathbf{b})} e^{i\phi_\theta(\mathbf{b})} \approx c(\mathbf{b})$$

Neural Quantum States

16-bit binary number

$\mathbf{b} \in \{0, 1\}^{16}$





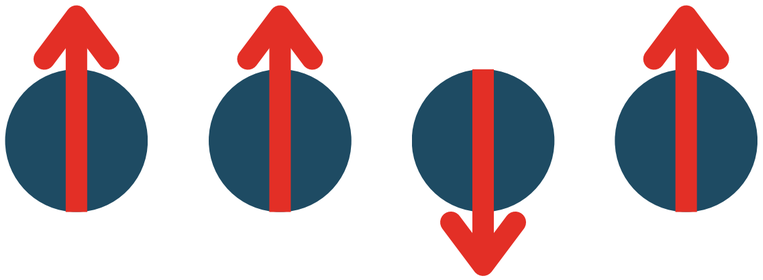
Neural Quantum States

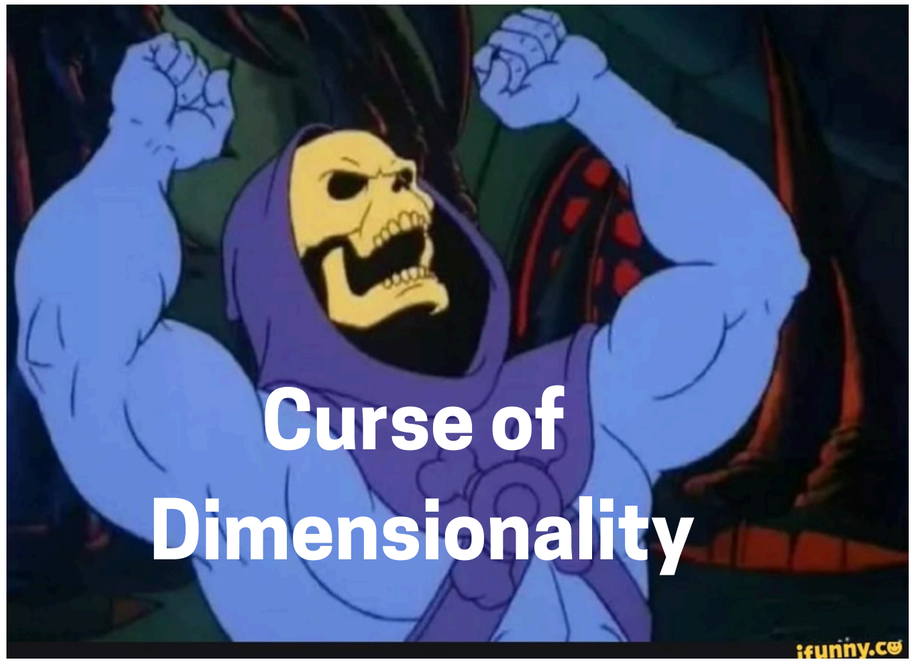
- Is this construction is at all useful?

- In order to know the complete description of

$$|\psi\rangle = \sum_{j=\{0,1\}^4} c_{j_1,\ldots,l_4} |j_1, \ldots, j_4\rangle$$

We need $t \sim \mathcal{O}(2^N)$ for N-bodies





Neural Quantum States

The postulates of quantum mechanics tell us that once we know a wavefunction, every observable quantity can be derived from it.

$$|\psi\rangle \rightarrow \mathbf{tr}(\hat{O}|\psi\rangle)$$

we are *actually* interested in when using a wave function is computing statistics of observables

Neural Quantum States

- One way to do this is with exact wave-functions, inner products, and the Born rule.

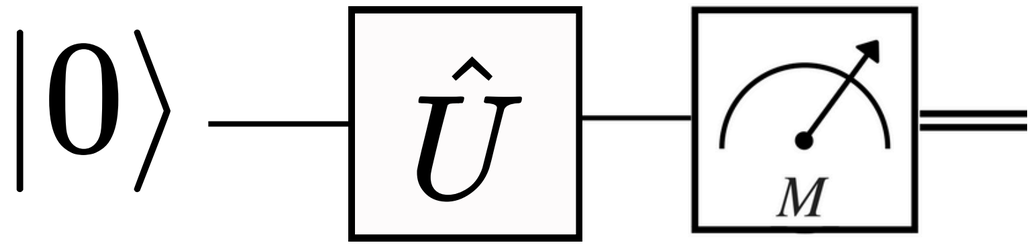
$$|\psi\rangle \in \mathcal{H} \qquad \hat{O} \in \beta(\mathcal{H}) \qquad \langle\ ,\ \rangle$$

Neural Quantum States

- One way to do this is with exact wave-functions, inner products, and the Born rule.

$$|\psi\rangle \in \mathcal{H} \qquad \hat{O} \in \beta(\mathcal{H}) \qquad \langle\ ,\ \rangle$$

$$p(m|\psi) = \mathrm{Tr}(\hat{O}|\psi\rangle\langle\psi|) = \langle\psi|\hat{O}|\psi\rangle$$



Neural Quantum States

- Another way is to employ the *Variational Monte Carlo* method
- VMC says to emulate wave-function statistics we need:

1. Efficiently generate amplitudes for any basis element

$$\langle \mathbf{b} | \psi \rangle$$

2. The ability to sample from the distribution

$$p\big(\mathbf{b}|\psi_\theta\big) = \frac{|\langle \mathbf{b}|\psi_\theta\rangle|^2}{\langle \psi_\theta|\psi_\theta\rangle}.$$

Computational mechanism for executing the Born rule

$$\langle \hat{H} \rangle_\psi \approx \sum_{\mathbf{b}} p(\mathbf{b}|\psi_\theta) \cdot \langle \mathbf{b}|\psi\rangle$$

Neural Quantum States

- VMC says to emulate wave-function statistics we need:

  1. Efficiently generate amplitudes for any basis element $\langle \mathbf{b} | \psi \rangle$

  2. The ability to sample from the distribution $p\left(\mathbf{b} | \psi_\theta\right) = \dfrac{|\langle \mathbf{b} | \psi_\theta \rangle|^2}{\langle \psi_\theta | \psi_\theta \rangle}.$

- With the two properties, we can sample any observable

This is because we can write down any observable in terms of these two quantities...

Neural Quantum States

$$\langle \hat{O} \rangle = \frac{\langle \psi_\theta | \hat{O} | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle} = \sum_{\mathbf{b}} \left(\frac{|\langle \mathbf{b} | \psi_\theta \rangle|^2}{\sum_{\mathbf{b}''} |\langle \mathbf{b}'' | \psi_\theta \rangle|^2} \cdot \sum_{\mathbf{b}'} \frac{\langle \mathbf{b} | \psi_\theta \rangle}{\langle \mathbf{b}' | \psi_\theta \rangle} \langle \mathbf{b} | \hat{O} | \mathbf{b}' \rangle \right).$$

1

This is just the probability distribution

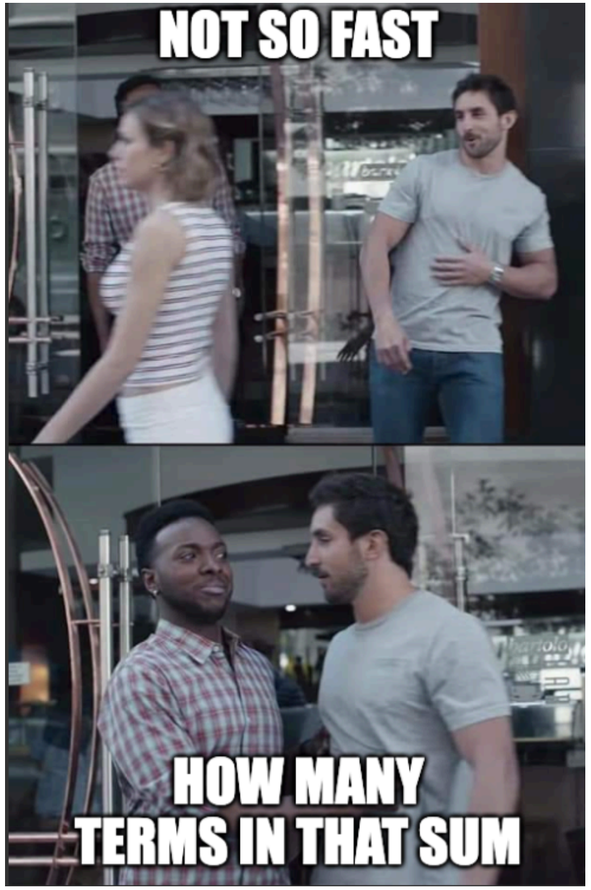
$$p\big(\mathbf{b}|\psi_\theta\big) = \frac{|\langle \mathbf{b} | \psi_\theta \rangle|^2}{\langle \psi_\theta | \psi_\theta \rangle}.$$

2

There are amplitudes times a matrix element in the computational basis

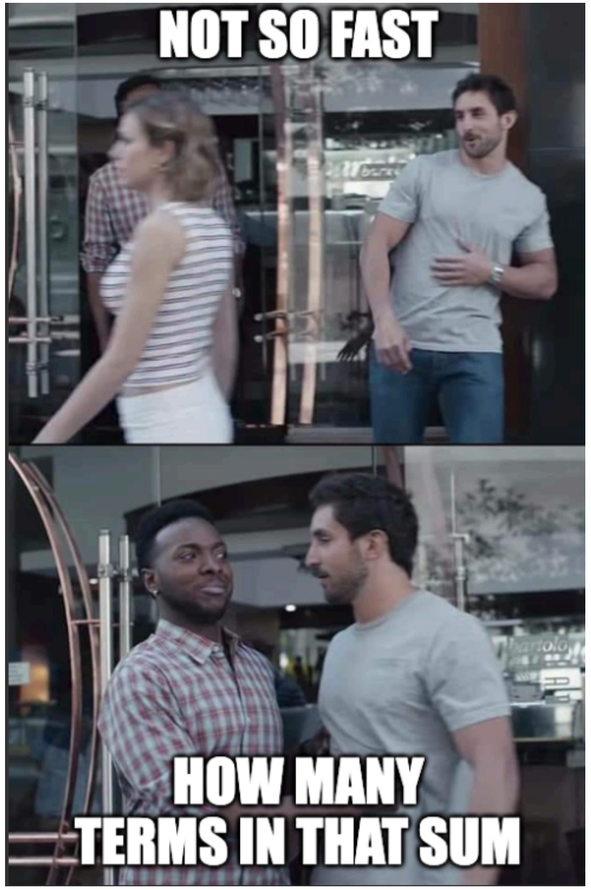
Neural Quantum States

$$\langle \hat{O} \rangle = \sum_{\mathbf{b}} \left(\frac{|\langle \mathbf{b} | \psi_\theta \rangle|^2}{\sum_{\mathbf{b}''} |\langle \mathbf{b}'' | \psi_\theta \rangle|^2} \cdot \sum_{\mathbf{b}'} \frac{\langle \mathbf{b} | \psi_\theta \rangle}{\langle \mathbf{b}' | \psi_\theta \rangle} \langle \mathbf{b} | \hat{O} | \mathbf{b}' \rangle \right).$$



Neural Quantum States

$$\langle \hat{O} \rangle = \sum_{\mathbf{b}} \left(\frac{|\langle \mathbf{b} | \psi_\theta \rangle|^2}{\sum_{\mathbf{b}''} |\langle \mathbf{b}'' | \psi_\theta \rangle|^2} \cdot \sum_{\mathbf{b}'} \frac{\langle \mathbf{b} | \psi_\theta \rangle}{\langle \mathbf{b}' | \psi_\theta \rangle} \langle \mathbf{b} | \hat{O} | \mathbf{b}' \rangle \right).$$



Neural Quantum States

Let’s make a *sparsity* assumption for the operator…

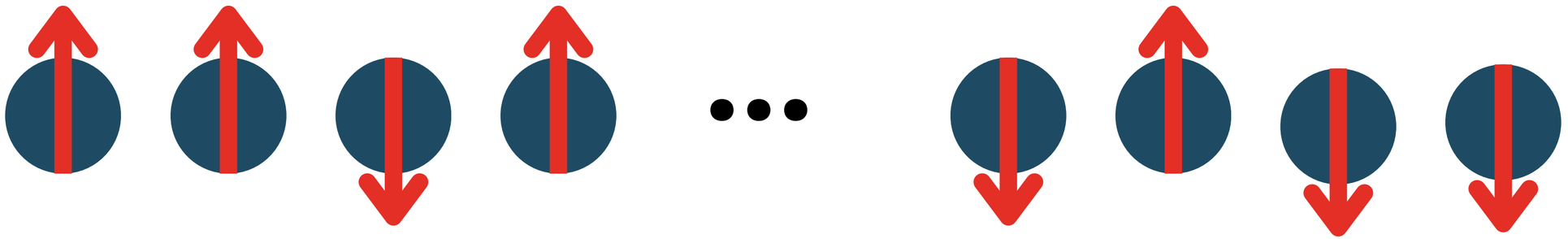
$$\langle \mathbf{b} | \hat{O} | \mathbf{b}' \rangle \neq 0$$

At most a polynomial number of bistrings should have non-zero amplitude w.r.t operator

Good or bad assumtion?

Neural Quantum States

$$|\psi(t)\rangle = \sum_{b_1,\ldots,b_N=0}^{1} c_{b_1,\ldots,b_N}(t)|b_1,\ldots,b_N\rangle$$



Which kinds of operators satisfy

$$\langle \mathbf{b}|\hat{O}|\mathbf{b}'\rangle \neq 0 \sim \mathcal{O}(\mathrm{Poly}(n))?$$

Neural Quantum States

$$\langle \hat{O} \rangle = \sum_{\mathbf{b}} p(\mathbf{b}|\psi_\theta) \sum_{\mathbf{b}'} \frac{\langle \mathbf{b}|\psi_\theta\rangle}{\langle \mathbf{b}'|\psi_\theta\rangle} \langle \mathbf{b}|\hat{O}|\mathbf{b}'\rangle \approx \langle O^{\mathrm{loc}}(\mathbf{b};\theta)\rangle_{\mathbf{b}}$$

Efficient sampling

Sparse

Sampling some bitstrings, $\{\mathbf{s}_1, \ldots, \mathbf{s}_M\} \sim p(\mathbf{b}|\psi_\theta)$

We can estimate observables by the arithmetic mean

$$\langle \hat{O} \rangle \approx \frac{1}{M} \sum_{j=1}^{M} \langle O^{\mathrm{loc}}(\mathbf{s}_j;\theta)\rangle.$$

Neural Quantum States

$$\langle \hat{O} \rangle \approx \frac{1}{M} \sum_{j=1}^{M} \langle O^{\mathrm{loc}}(\mathbf{s}_j; \theta) \rangle.$$

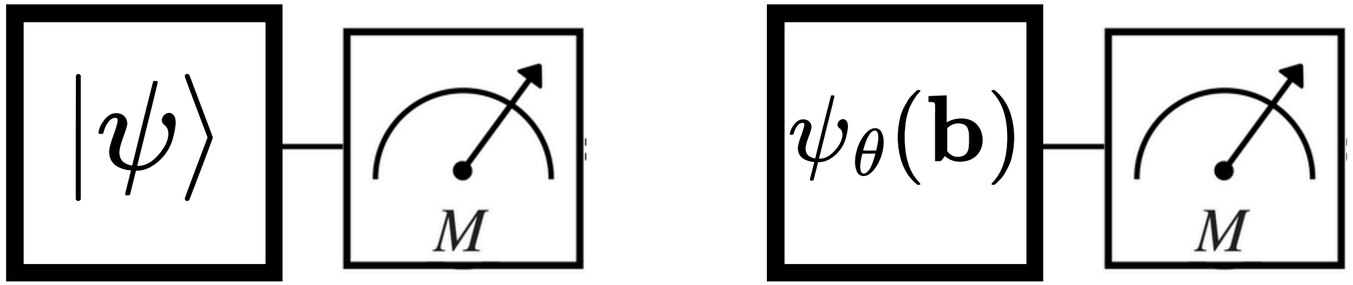
Error in sampling this way...

$$\epsilon = \sqrt{\sigma^2 / M},$$

NB: This is known and controllable - simply change the number of samples!

Neural Quantum States

Representing a given state



Algorithm 7 Estimating Local Observables with a NQS

Input: a NQS $\psi_\theta : \{0,1\}^N \to \mathbb{C}$ and a set of random Binary bitstrings $\mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_M\}$
for each $\mathbf{b}$ in $\mathbf{B}$ **do**
    Forward pass $\psi_\theta(\mathbf{b}) = \langle \mathbf{b}|\psi_\theta\rangle$ ▷ Should be efficient
    Compute $\langle O^{\text{loc}}(\mathbf{b};\theta)\rangle$ ▷ See Eq. (4.103), $\hat{O}$ inputted therefore efficient
end for
Return $\langle \hat{O}\rangle \approx \frac{1}{M}\sum_{\mathbf{b}\in\mathbf{B}} \langle O^{\text{loc}}(\mathbf{b};\theta)\rangle$

NQS Part II

- Variational Princinple
- Training with Energy Gradients
- Zero Variance Principle

Neural Quantum States

The variational principle applies to neural quantum states

Box 4: Variational Principle

For any normalised wavefunction, $|\psi\rangle$, the variational principle states that

$$E_{gs} \leq \langle\psi|\hat{H}|\psi\rangle\,. \tag{4.56}$$

That is, all states of a quantum system with Hamiltonian H overestimate the ground-state energy except for the ground state itself, $|\psi_{gs}\rangle$. Variational methods seek to find a tight upper bound to this.

Neural Quantum States

Using a neural network as a map from a binary bitstring (the index) to an amplitude will still be an upper bound for the ground state energy

$$E(\theta) = \frac{\langle \psi_\theta | \hat{H} | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle} \geq E_{gs}$$

Thanks to Monte Carlo sampling,

$$E(\theta) = \langle \hat{H} \rangle \approx \sum_{\mathbf{b}} p\left(\mathbf{b} | \psi_\theta\right) \langle H^{\mathrm{loc}}(\mathbf{b}; \theta) \rangle_{\mathbf{b}}$$

Neural Quantum States

To create a training loop that solves $\min_{\theta} \frac{\langle \psi_\theta | \hat{H} | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle},$

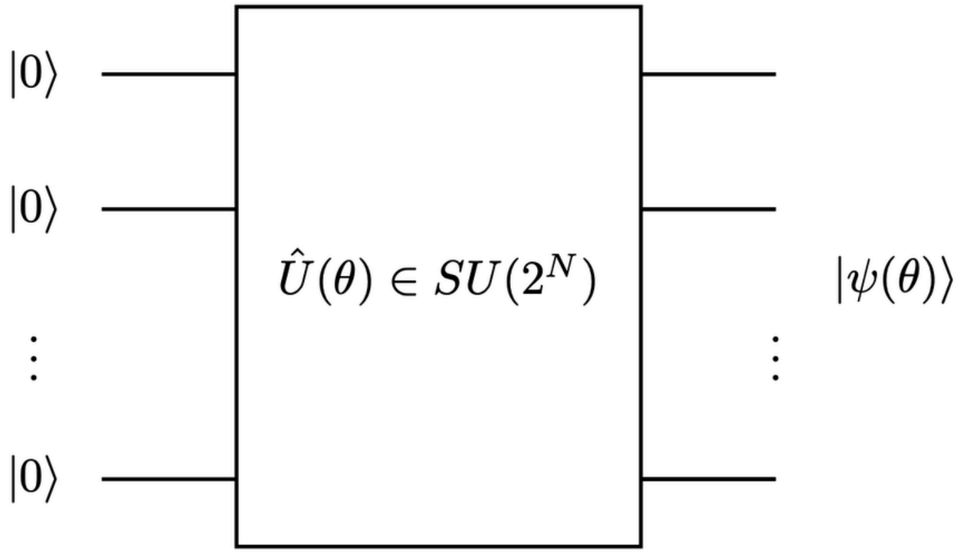
We need to show that our estimate of the system's energy is differentiable if we want to do gradient based updates

$$\theta \leftarrow \theta - \eta \nabla_\theta E(\theta), \qquad E(\theta) = \frac{\langle \psi_\theta | \hat{H} | \psi_\theta \rangle}{\langle \psi_\theta | \psi_\theta \rangle}$$

Neural Quantum States

How is this **different from** a **variational quantum eigensolver**??

All variational **params** in a VQE are ***real***



this is because the **state** is constructed **from** a parametric **circuit**

Neural Quantum States

How is this different from VQE?

In NQS, we have a neural representation of a wave-function rather than access to it as the output of some unitary process....

Complex numbers now form an integral part of the computational graph...

$$E(\psi_\theta) : \mathbb{C} \rightarrow \mathbb{R}$$

Neural Quantum States

- We now want to find states of *minimal energy*...
- Back propagation through a complex-to-real function

Remember complex analysis?

- A function is *holomorphic* when

$$\frac{\partial}{\partial z^*} f(z, z^*) = 0.$$

f is not a function of the conjugate

- We need a key assumption that the wave-function is *holomorphic*

$$\frac{\partial \psi_\theta}{\partial \theta^*} = 0.$$

Remember complex analysis?

- We need a key assumption that the wave-function is *holomorphic*

$$\frac{\partial \psi_\theta}{\partial \theta^*} = 0.$$

- Reasonable assumption - components of a complex neural network can easily be made holomorphic by ensuring the activation functions and sums over layers are taken over θ (not the conjugate)

Neural Quantum States

We can derive the following:

$$\frac{\partial E(\theta)}{\partial \theta^*} = \mathbb{E}_{x\sim p_\theta(x)}\left[\left(\partial_\theta \log \psi_\theta(x)\right)\left(H^{\mathrm{loc}} - \mathbb{E}_{x\sim p_\theta(x)}[H^{\mathrm{loc}}(x)]\right)\right].$$

All that remains is to conjugate

Hence, our model should be the logarithm of the amplitudes

Neural Quantum States

We can derive the following:

$$\frac{\partial E(\theta)}{\partial \theta^*} = \mathbb{E}_{x\sim p_\theta(x)}\left[\left(\partial_\theta \log \psi_\theta(x)\right)\left(H^{\mathrm{loc}} - \mathbb{E}_{x\sim p_\theta(x)}[H^{\mathrm{loc}}(x)]\right)\right].$$

All that remains is to conjugate

(and to write code)

Neural Quantum States

Since this gradient is based on expectation under distributions, we can approximate it with batches of samples $x \sim p_\theta(x)$

$$\partial_{\theta^*} E(\theta) \approx \frac{1}{N} \sum_{i=1}^{M} (\partial_\theta \log \psi_\theta(x_i)) \left(H^{\mathrm{loc}}(x_i) - \frac{1}{N} \sum_{j=1}^{M} H^{\mathrm{loc}}(x_j) \right),$$

Zero-Variance Principle

- Variance of local energy vanishes if (and only if!) we have an eigenstate

$$H^{\mathrm{loc}}(x) = \frac{\langle x|\hat{H}|\psi_\theta\rangle}{\langle x|\psi_\theta\rangle} = \frac{E_n \langle x|\psi_\theta\rangle}{\langle x|\psi_\theta\rangle} = E_n$$

- We can see that

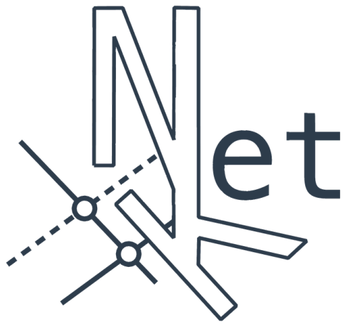
$$\mathrm{Var}_{p_\theta}\left[H^{\mathrm{loc}}\right] = \frac{\langle \psi_\theta|(\hat{H} - E_0)^2|\psi_\theta\rangle}{\langle \psi_\theta|\psi_\theta\rangle} = 0$$

- **Variance is a measure of how close to**

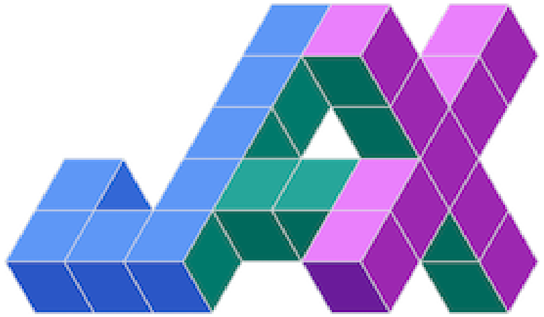
CONTENT

CONTENT

The NetKet package



Built on



Open source package developed by group of Filipo Vicentini

NQS Ground state search - Ising Model

Consider states of a 1D spin-1/2 chain with Ising Hamiltonian

$$\hat{H} = J \sum_j \sigma_j^{(x)} \sigma_{j+1}^{(x)} - h \sum_j \sigma_j^{(z)}$$

NQS Ground state search - Ising Model

Consider states of a 1D spin-1/2 chain with Ising Hamiltonian

$$\hat{H} = J \sum_j \sigma_j^{(x)} \sigma_{j+1}^{(x)} - h \sum_j \sigma_j^{(z)}$$

With reference state found by exact diagonalisation

$$\langle \mathbf{b} | \hat{H} | \mathbf{b}' \rangle \in \mathbb{C}^{2^n \times 2^n} \sim \mathcal{O}(4^n)$$

NQS Ground state search - Ising Model

Consider states of a 1D spin-1/2 chain with Ising Hamiltonian

$$\hat{H} = J \sum_j \sigma_j^{(x)} \sigma_{j+1}^{(x)} - h \sum_j \sigma_j^{(z)}$$

With reference state found by exact diagonalisation

$$\langle \mathbf{b} | \hat{H} | \mathbf{b}' \rangle \in \mathbb{C}^{2^n \times 2^n} \sim \mathcal{O}(4^n)$$

(so we can check an exact
numerical result for small systems)

NQS Ground state search - Ising Model

Consider states of a 1D spin-1/2 chain with Ising Hamiltonian

$$\hat{H} = J \sum_j \sigma_j^{(x)} \sigma_{j+1}^{(x)} - h \sum_j \sigma_j^{(z)}$$

To go beyond the curse of dimensionality we need to look for variational approximations with a *polynomial* scaling for the number of bodies

Let’s look at three example models

NQS Ground state search - Ising Model

Normalised Mean Field Ansatz $\langle \sigma_1^z, \dots, \sigma_N^z | \psi_{mf} \rangle = \prod_{i=1}^{N} \Phi(Z_i)$

Polar representation of the wavefunction

$$\Phi(\sigma_z) = \sqrt{A(\sigma^z)} e^{i\phi(\sigma^z)}$$

$$A : \{0,1\}^n \to \mathbb{R}_+$$

To simplify our lives

NQS Ground state search - Ising Model

Normalised Mean Field Ansatz $\langle \sigma_1^z, \ldots, \sigma_N^z | \psi_{mf} \rangle = \prod_{i=1}^{N} \Phi(Z_i)$

Polar representation of the wavefunction

$$\Phi(\sigma_z) = \sqrt{A(\sigma^z)} e^{i\phi(\sigma^z)}$$

$$A : \{0,1\}^n \to \mathbb{R}_+ \qquad A(\sigma^z;\theta) = \frac{1}{1 - e^{-\theta\sigma_z}}$$

To simplify our lives $\phi = 0$

e.g. in 2D, Onsager's solution is known and real so we can safely assume this here

NQS Ground state search - Ising Model

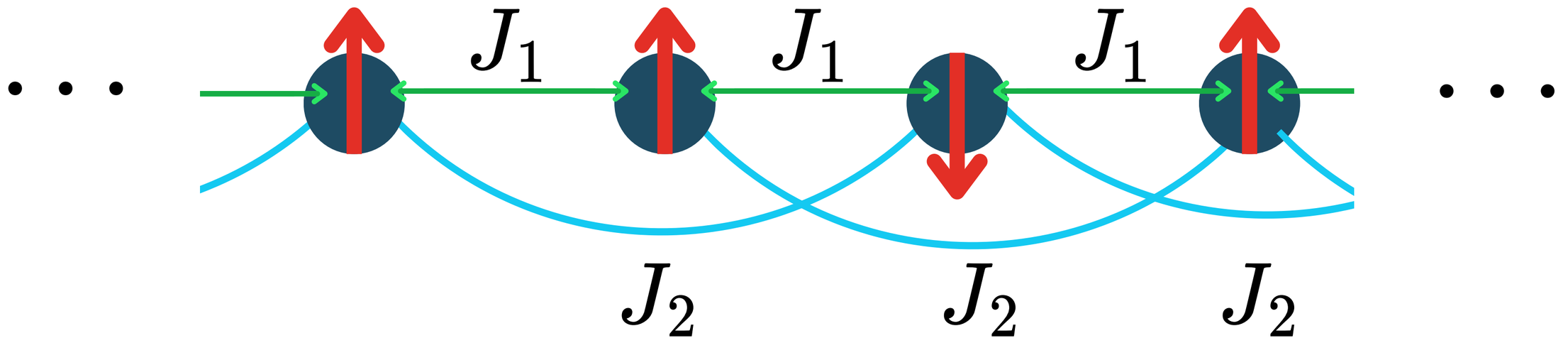
Jastrow Ansatz

$$\langle \sigma_1^z, \dots, \sigma_N^z | \Psi_{\text{jas}} \rangle = e^{\sum_i J_1 \sigma_i^z \sigma_{i+1}^z + J_2 \sigma_i^z \sigma_{i+2}^z}$$

NQS Ground state search - Ising Model

Jastrow Ansatz

$$\langle \sigma_1^z, \dots, \sigma_N^z | \Psi_{\text{jas}} \rangle = e^{\sum_i J_1 \sigma_i^z \sigma_{i+1}^z + J_2 \sigma_i^z \sigma_{i+2}^z}$$

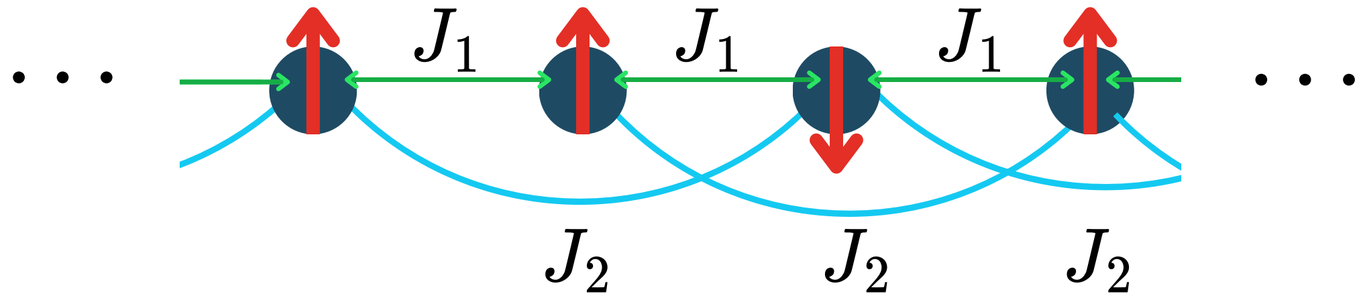


NQS Ground state search - Ising Model

Jastrow Ansatz

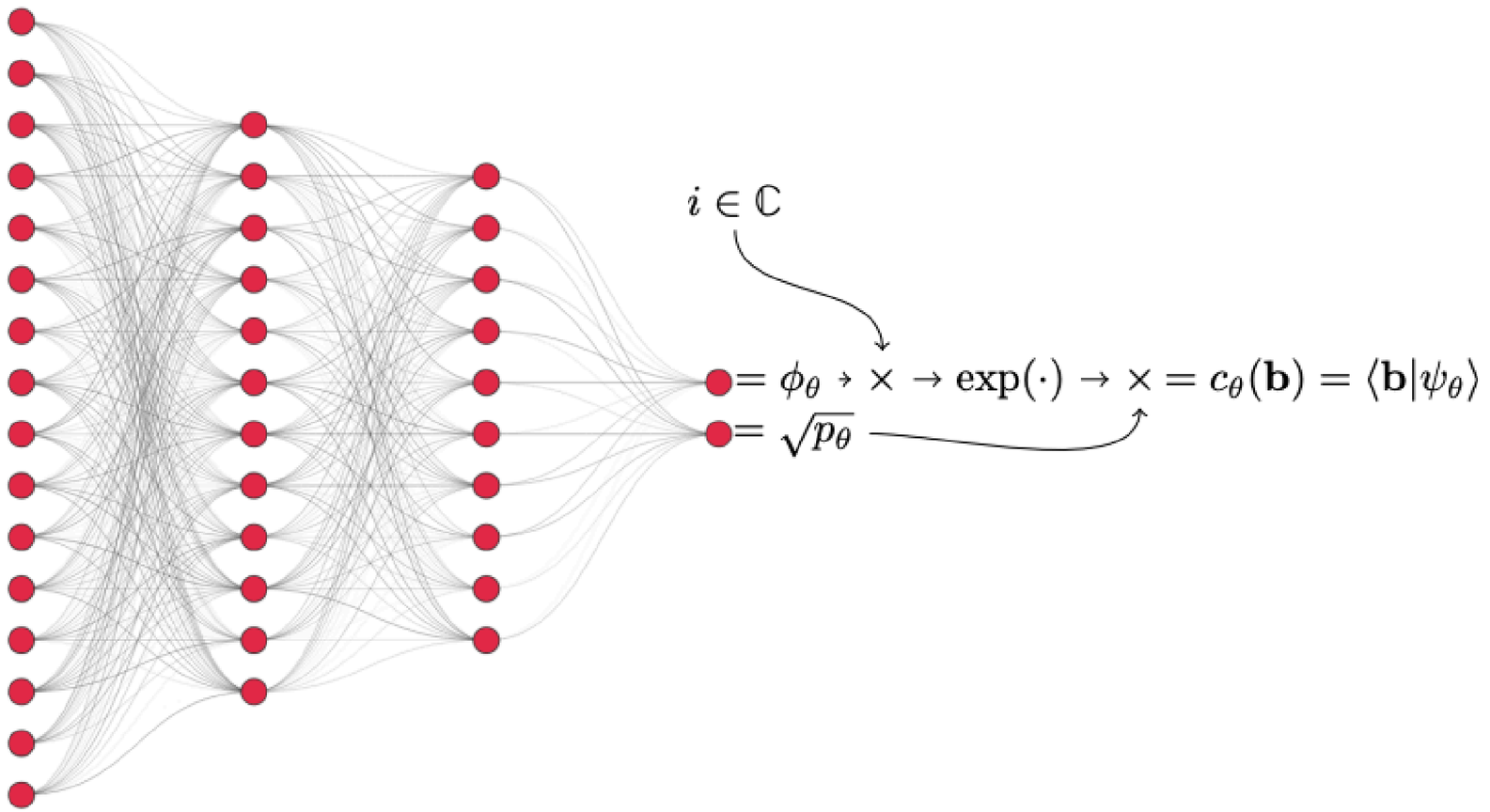
$$\langle \sigma_1^z, \dots, \sigma_N^z | \Psi_{\text{jas}} \rangle = e^{\sum_i J_1 \sigma_i^z \sigma_{i+1}^z + J_2 \sigma_i^z \sigma_{i+2}^z}$$

Variational parameters $\theta = \{J_1, J_2\}$



NQS Ground state search - Ising Model

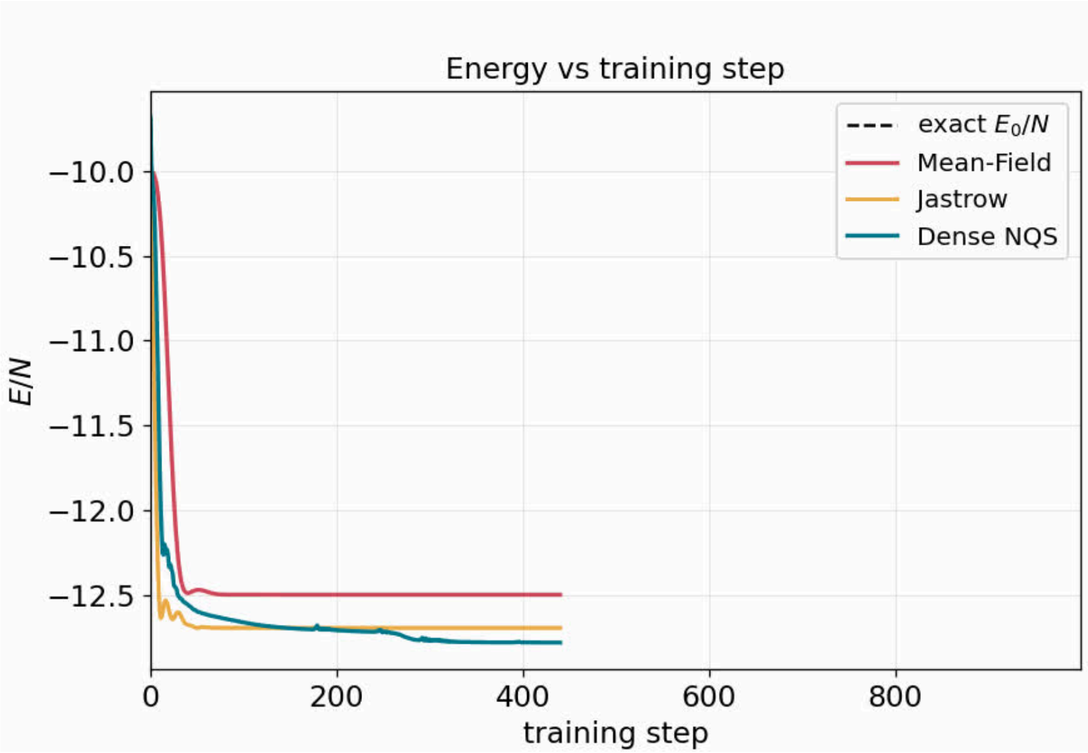
NQS Ansatz

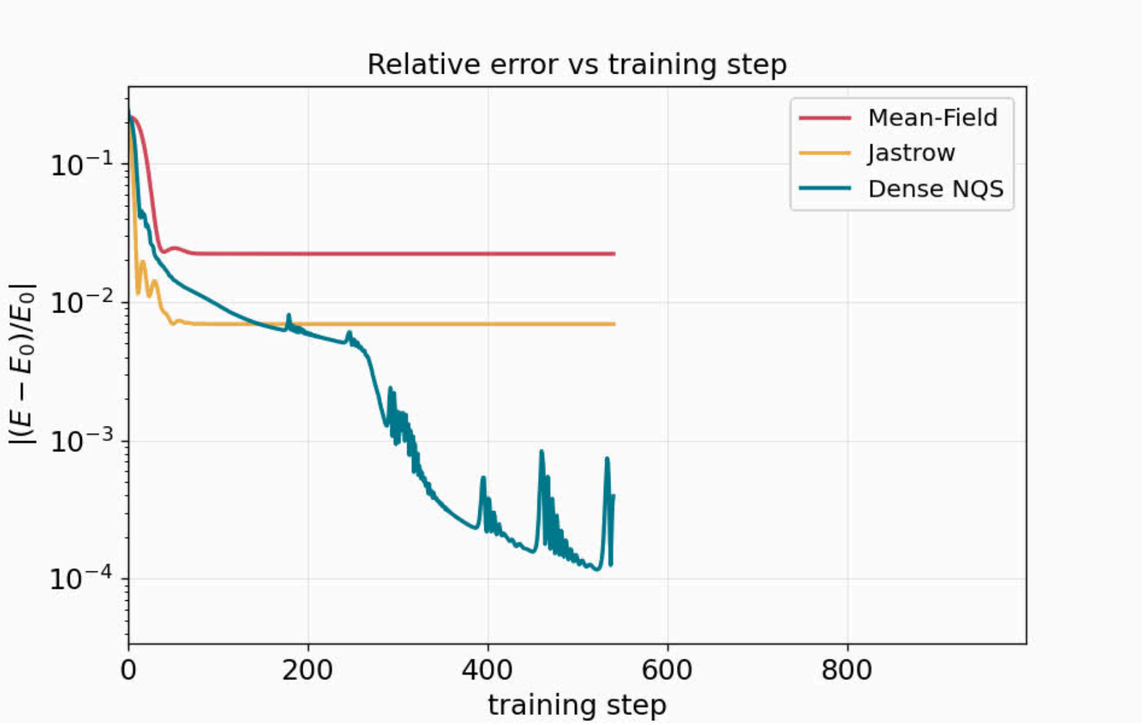


Dense Network

NQS Ground state search - Ising Model

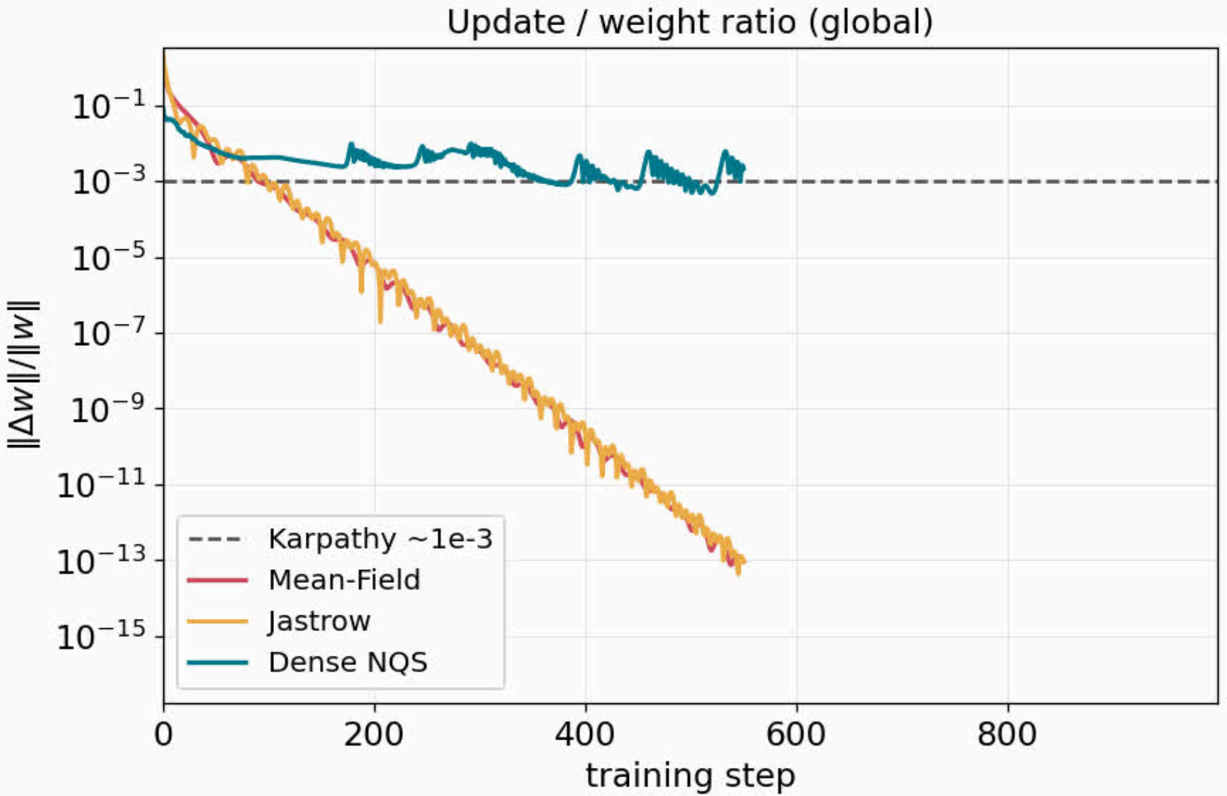
- NQS architecture outperforming Jastrow and mean-field ansatze?

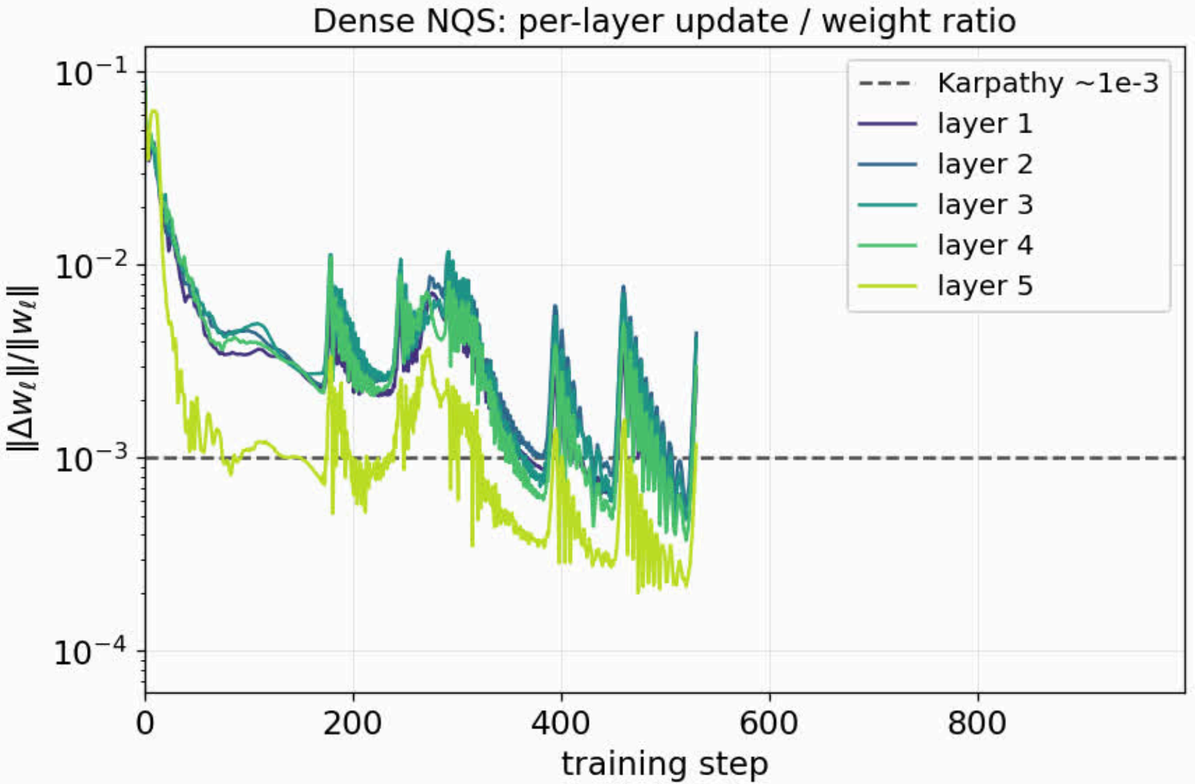




NQS Ground state search - Ising Model

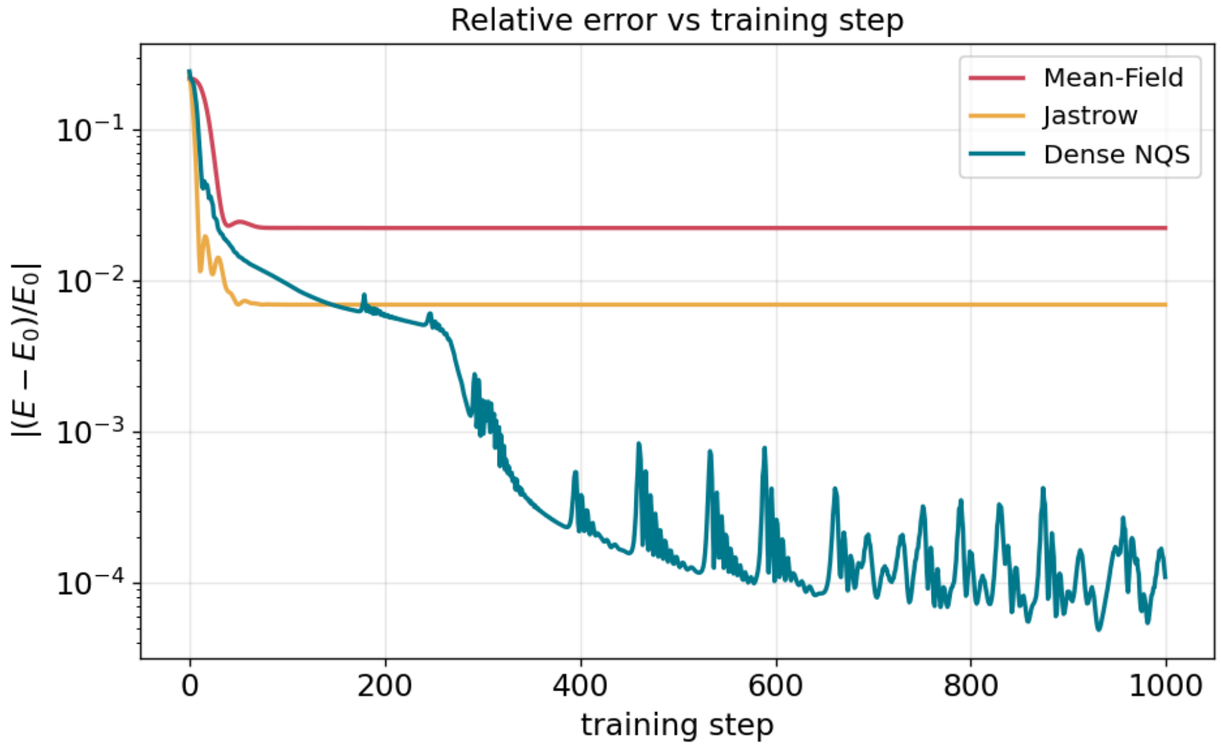
- Yes - our metrics (update ratio) and layer-norms look relatively healthy

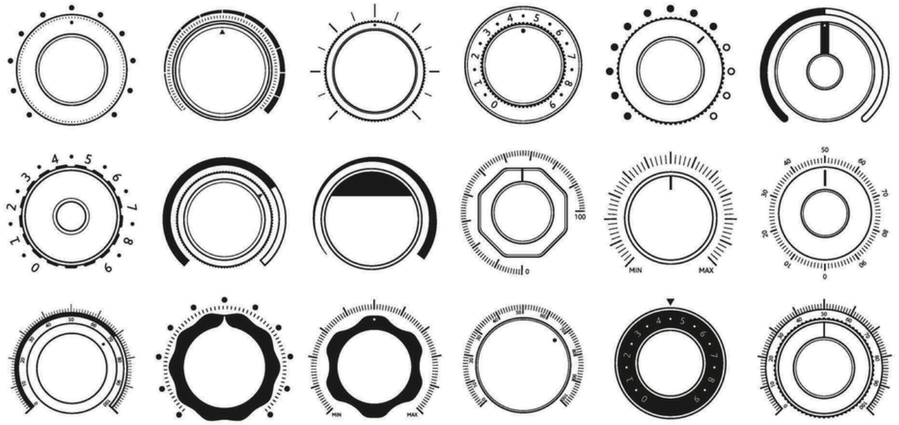




NQS Ground state search - Ising Model

- Is this the best we can do though?

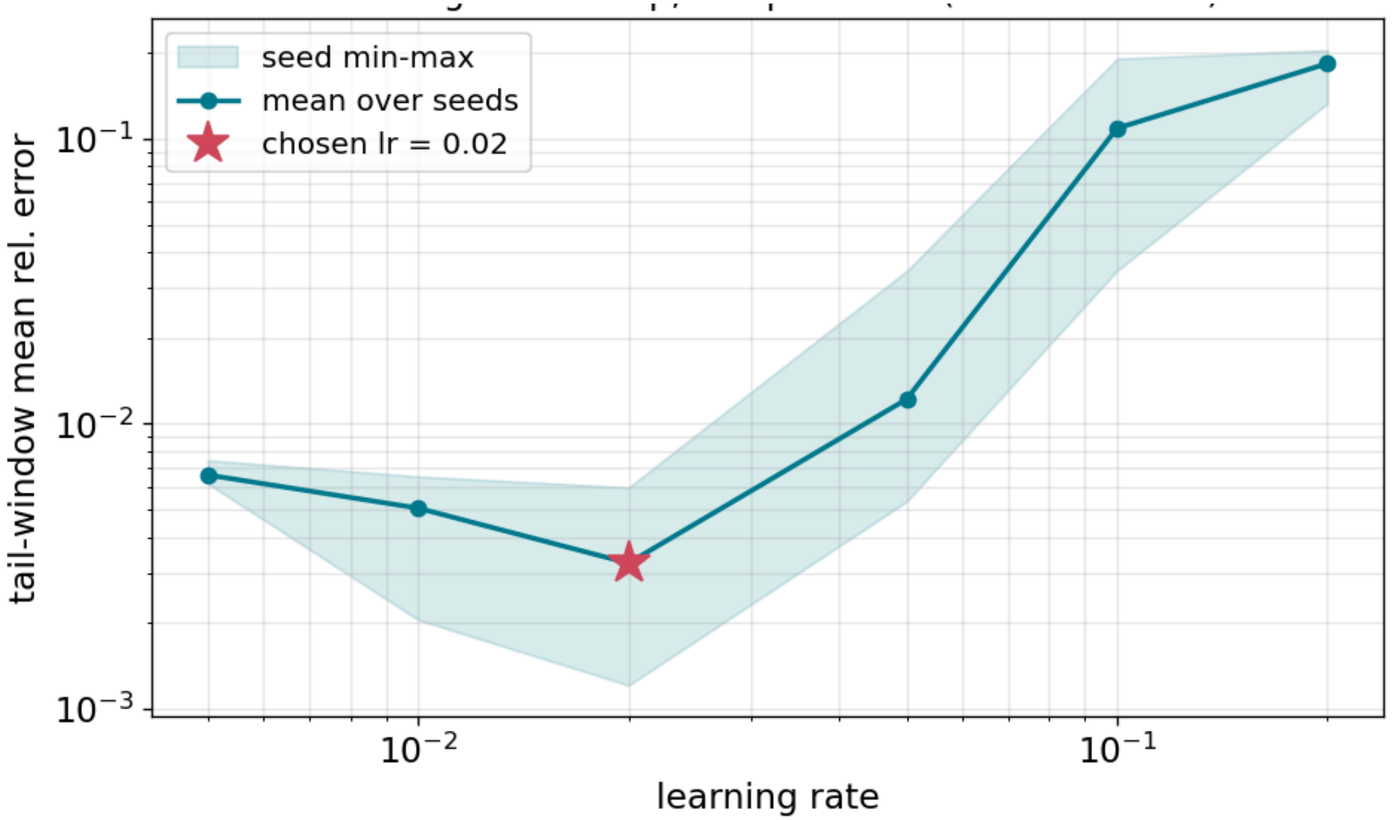




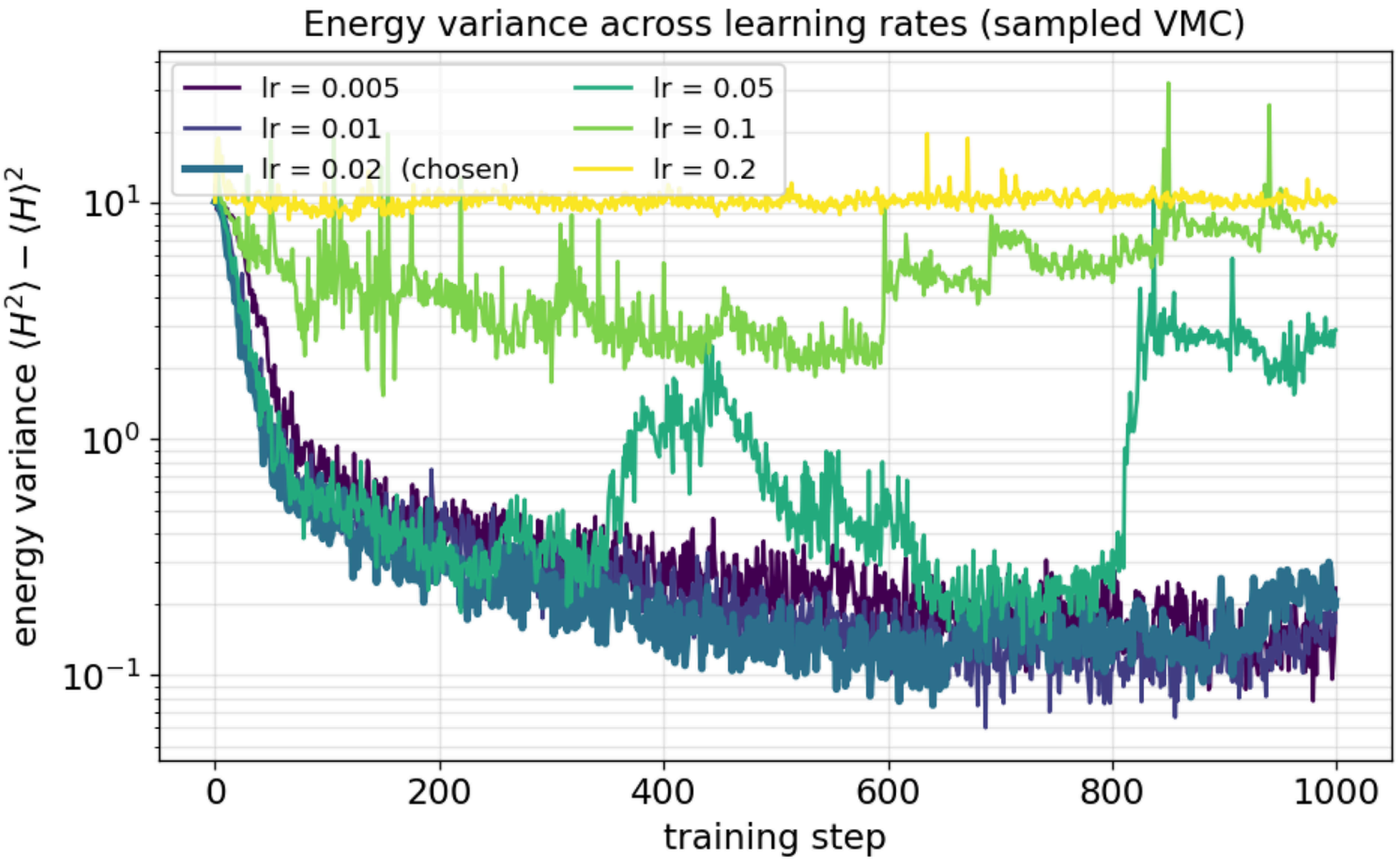
Let’s do some hyperparameter tuning (!)

NQS Ground state search - Ising Model

1. Finding a good learning rate:

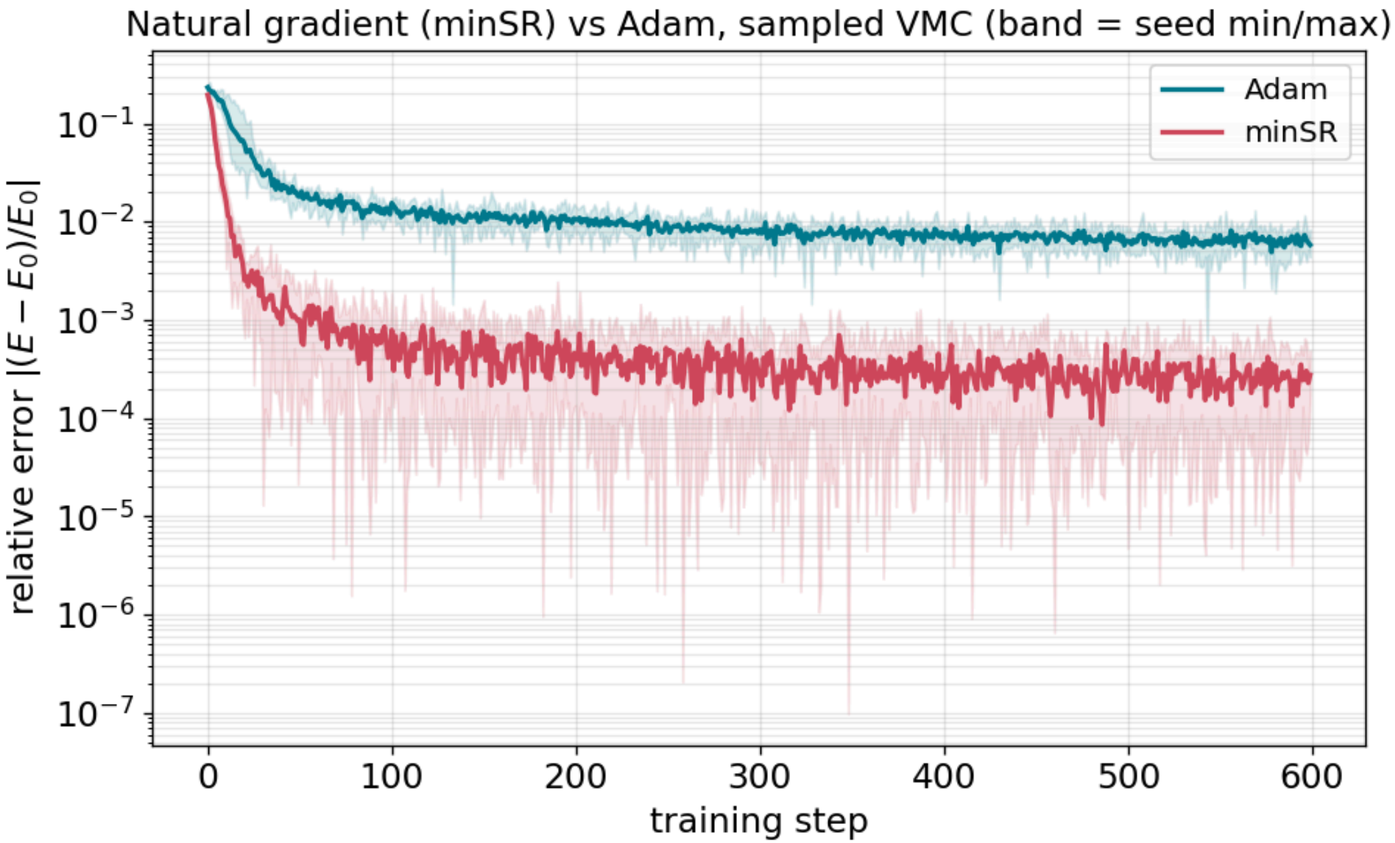


Zero-variance principle in practise:



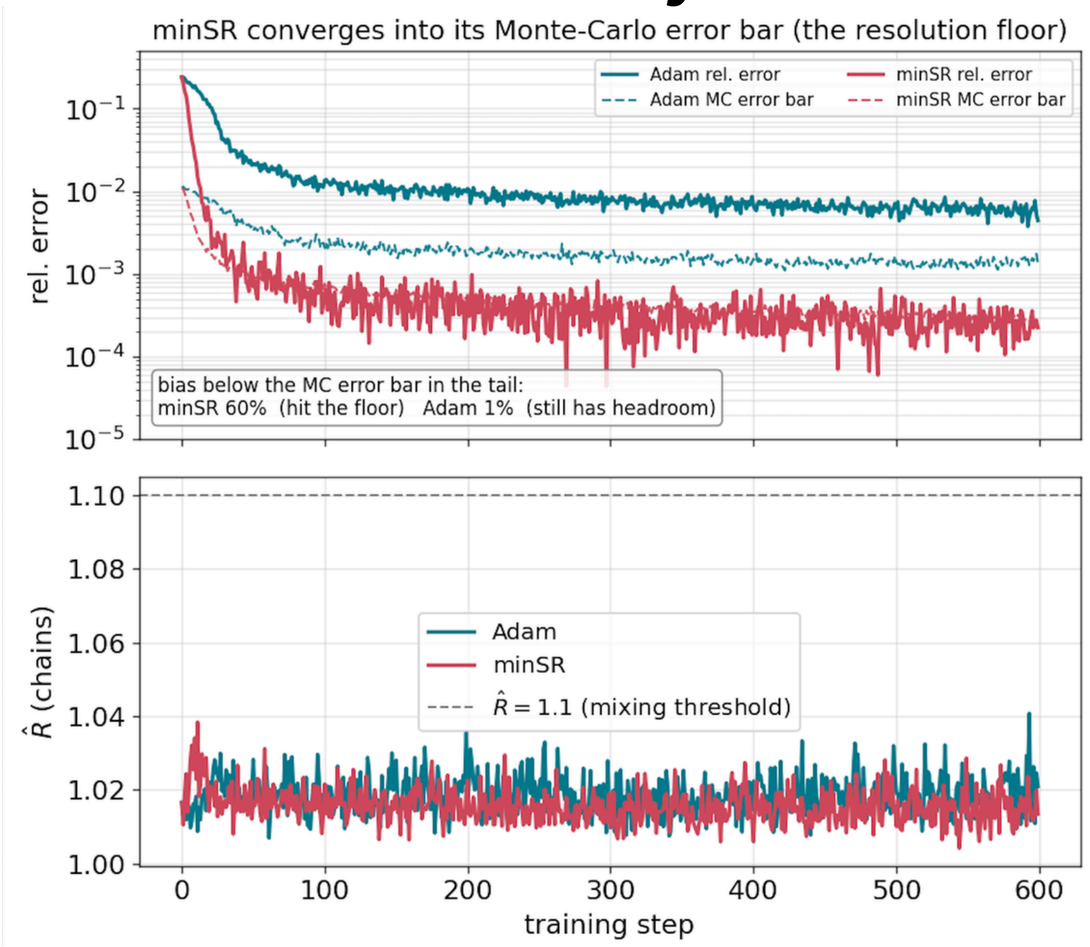
NQS Ground state search - Ising Model

2. Optimiser - ADAM vs minSR (Heyl 2023)



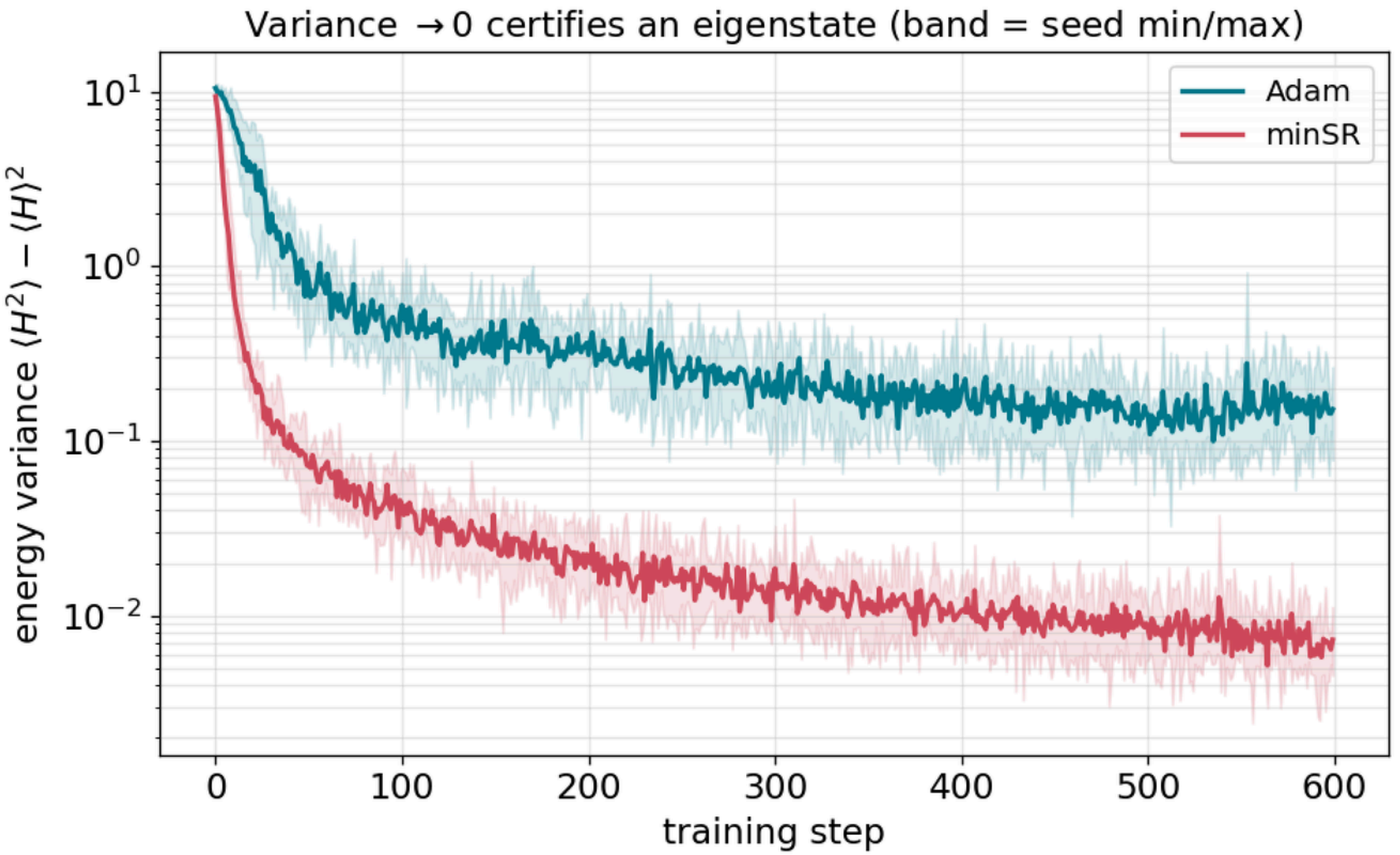
NQS Ground state search - Ising Model

2. Optimiser - ADAM vs minSR (Heyl 2023)



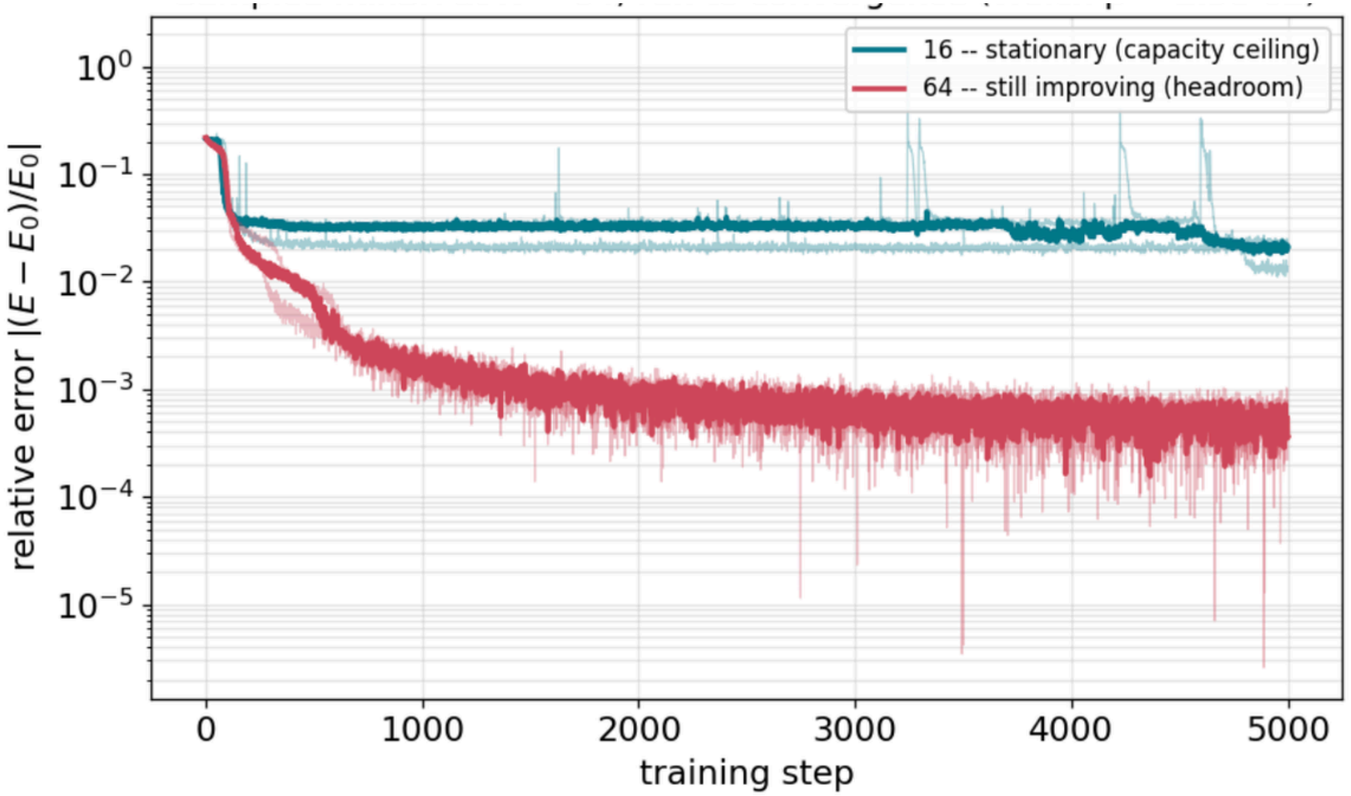
NQS Ground state search - Ising Model

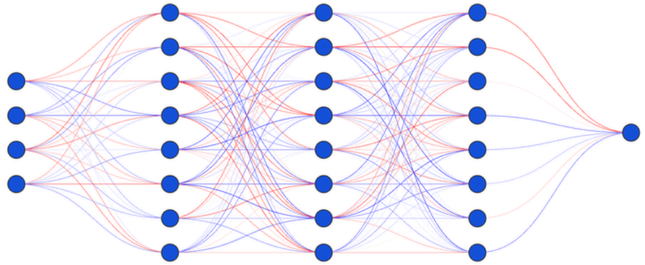
2. Optimiser - ADAM vs minSR (Heyl 2023)



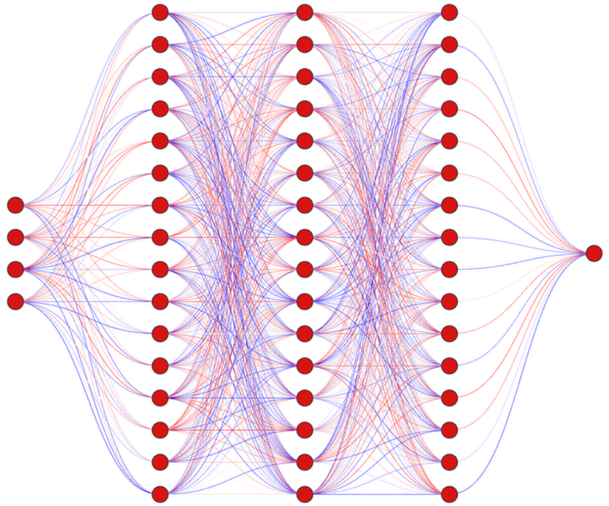
NQS Ground state search - Ising Model

3. **Changing architectures' width**



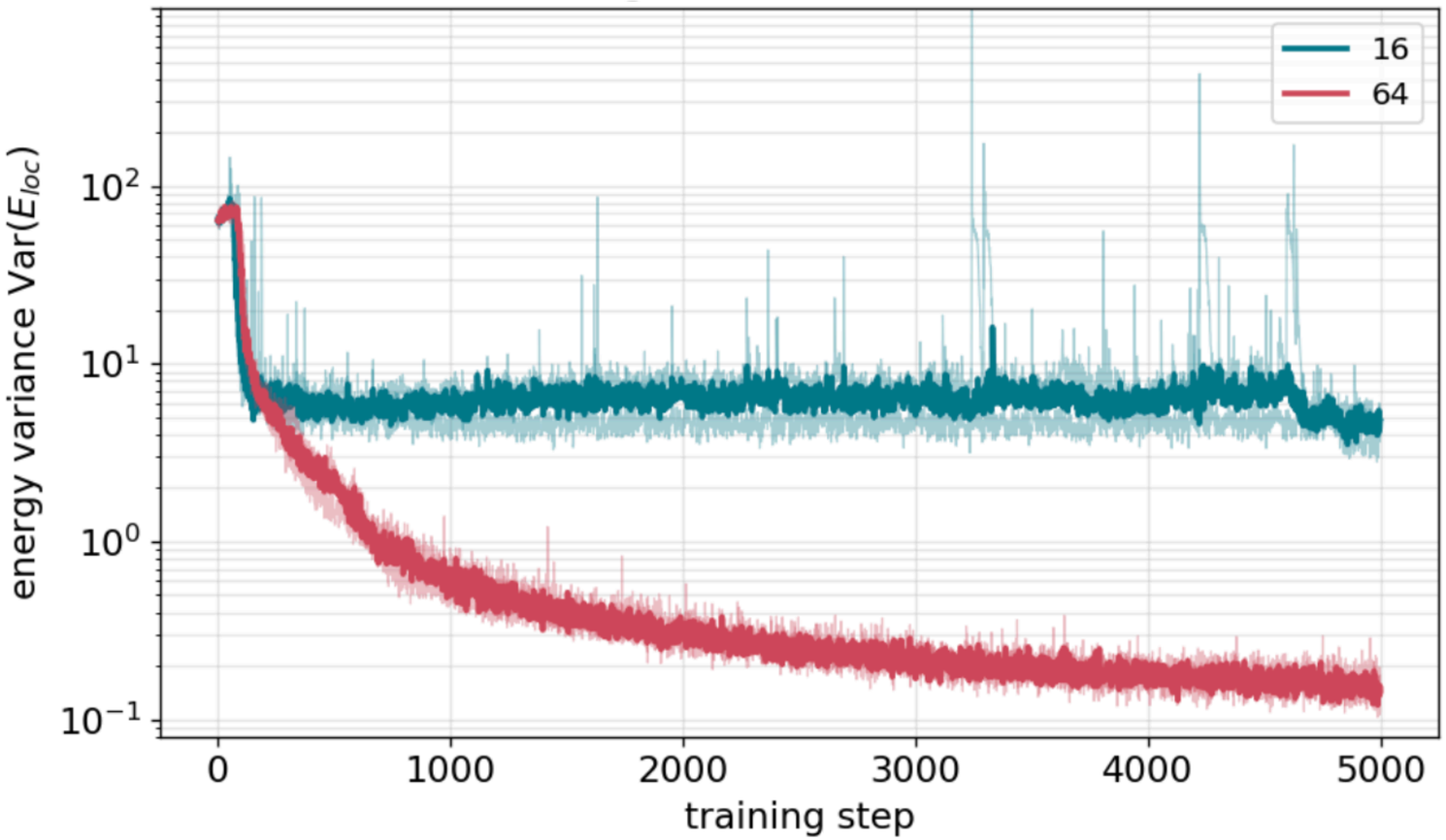


VS



NQS Ground state search - Ising Model

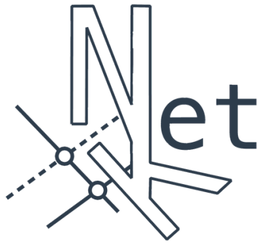
3. **Changing architectures' width - zero variance principle**



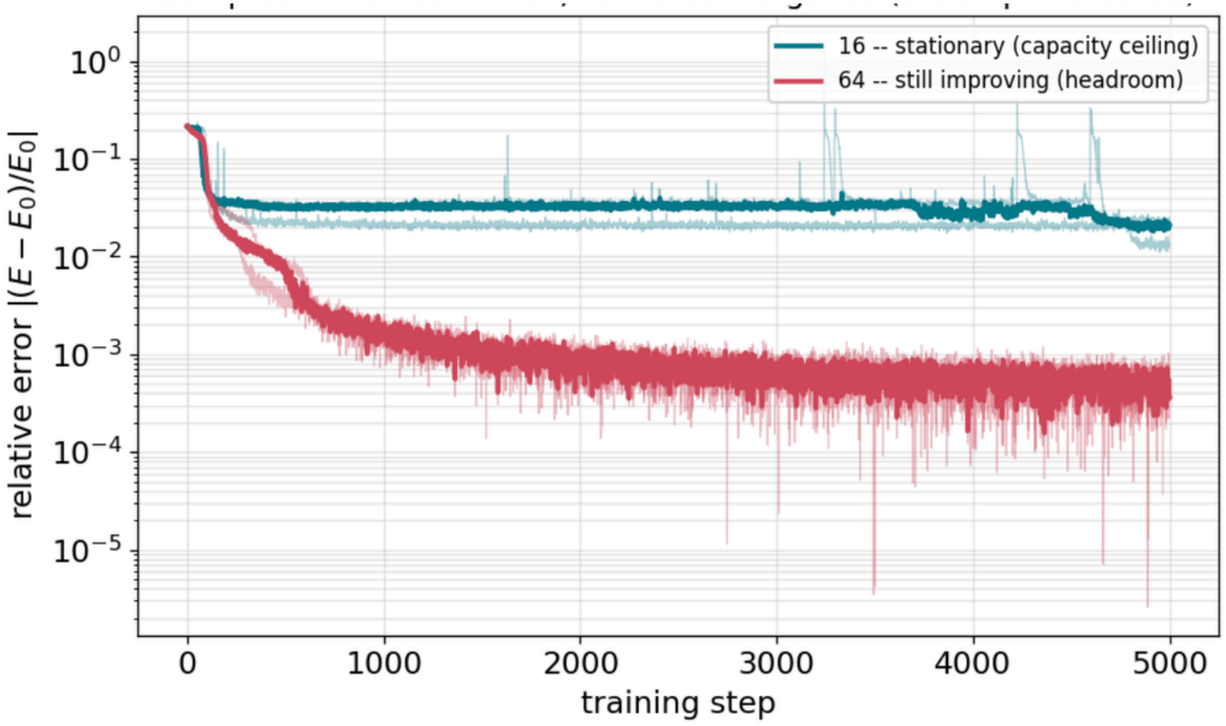
CONTENT

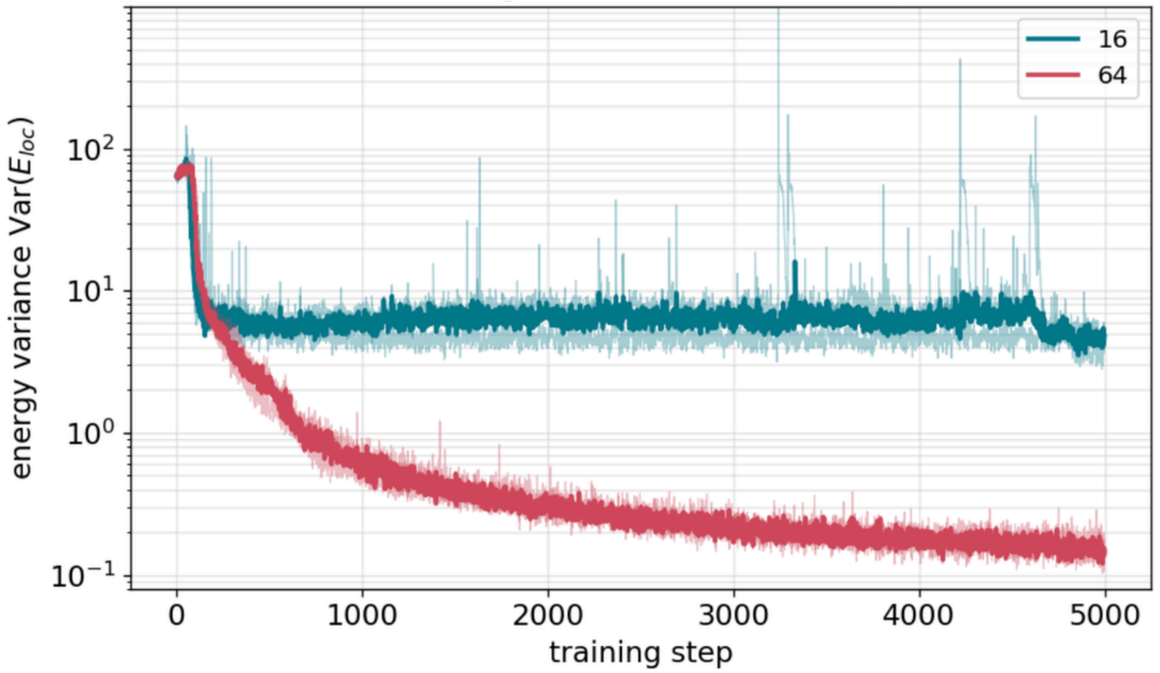
CONTENT

Practical



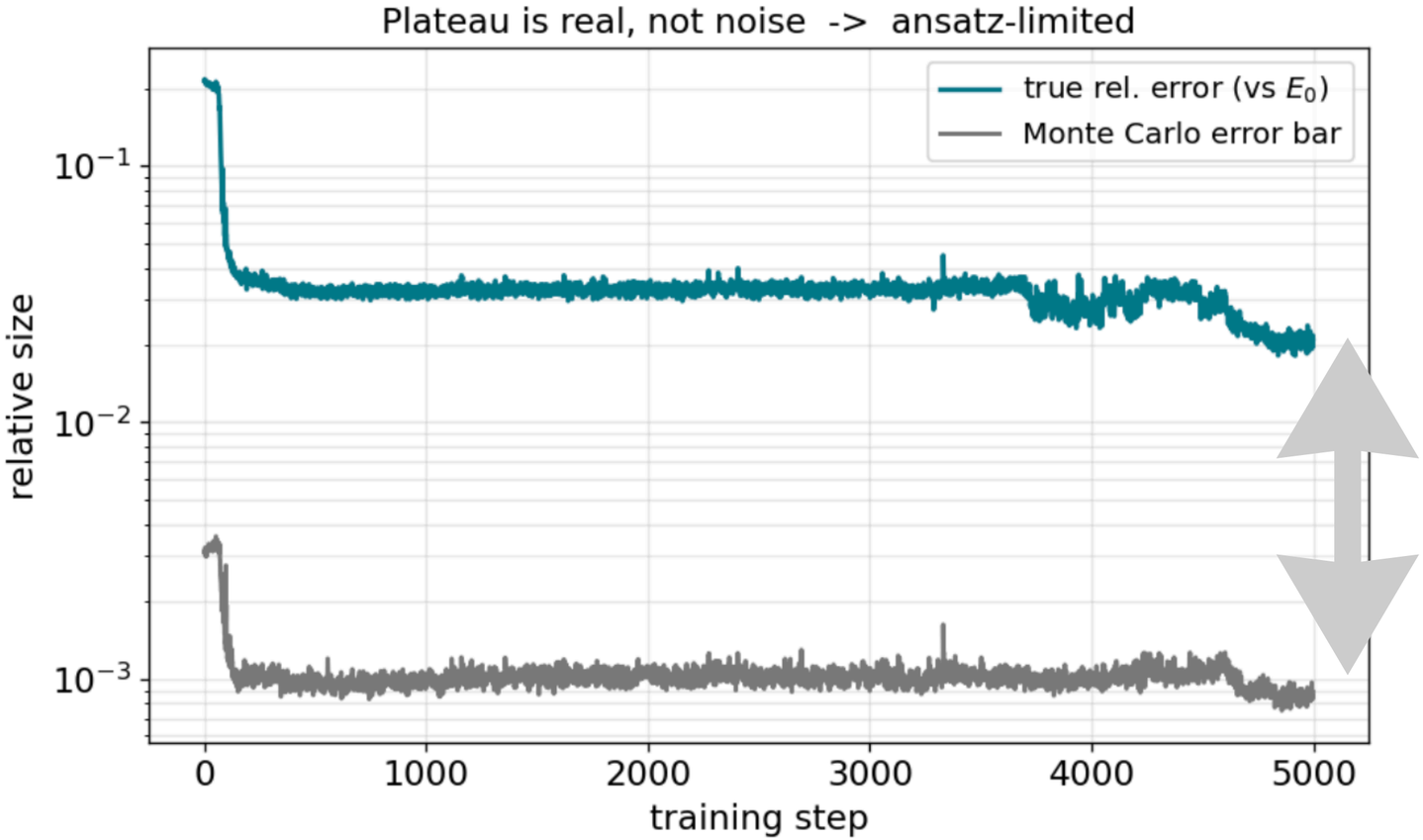
- **Beat the current neural net for the Ising model….**





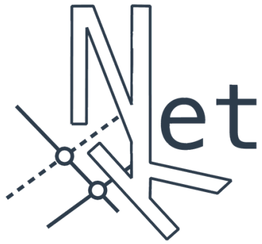
Practical

- **Beat the current neural net for the Ising model….**



- If we plot the shot noise, we can see there is still room for improvement (!)

Practical

- **Two notebooks**
  a. Install packages (jax, flax, optax, netket)
  b. Notebook 1 - Current best
  c. Notebook 2 - Your model (!)