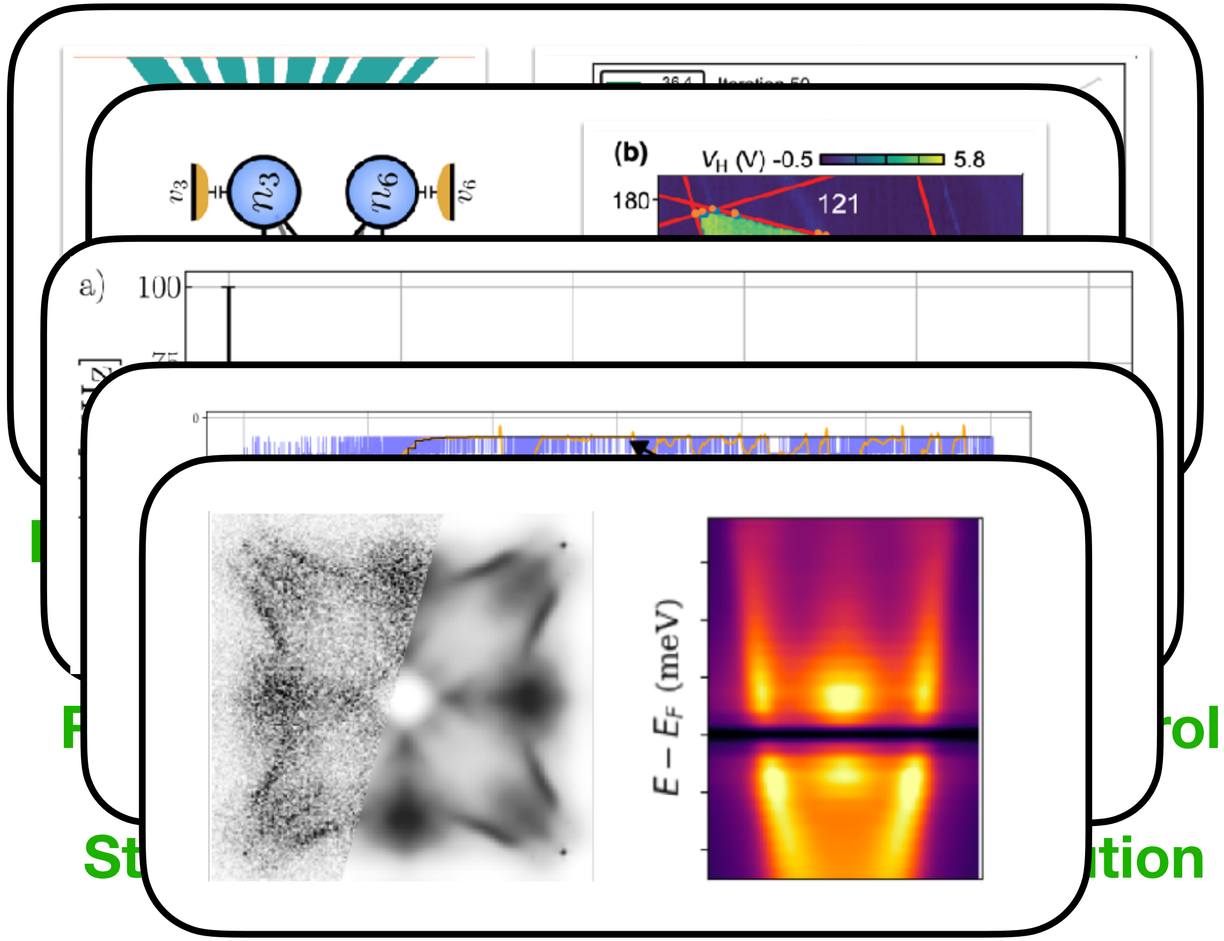
Exploring AI for (quantum) physics

ATTP Autumn School, Module 2

Evert van Nieuwenburg

Chris Chalkias & Tom Daman

AI for Quantum Control



Self-supervised denoising of STM data

Jan Krzywda (PD)

Jan Li

Ilse Kuijf

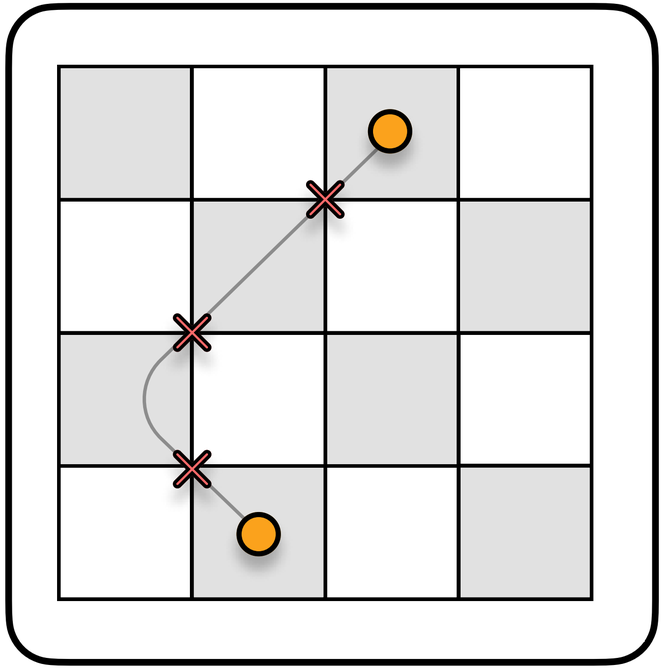
Felix Frohnert

Marie Kempkes

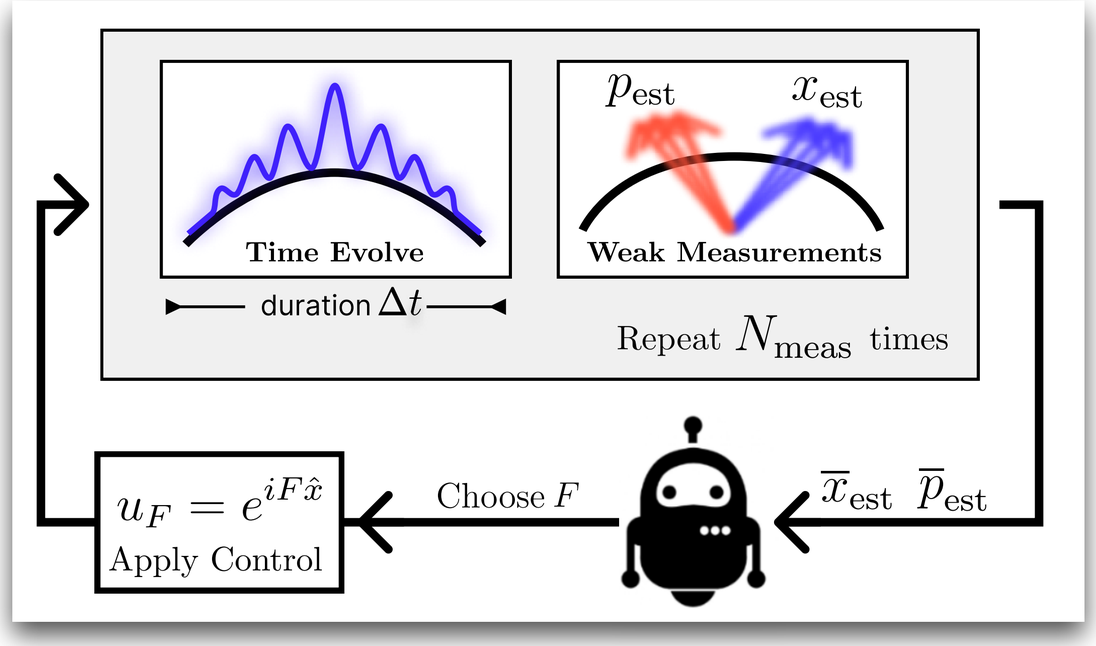
Marien Raat

AI for Quantum Control

How do you control something you cannot look at?



Quantum Error Correction



The Quantum Cartpole

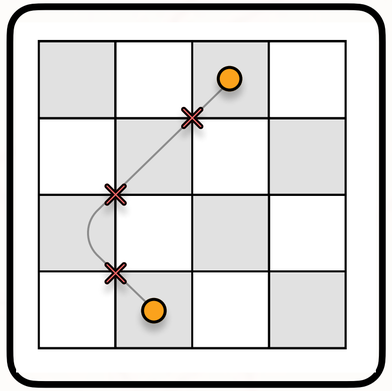
Can a classical AI optimize strategies for quantum systems?
Does a quantum circuit do better than a classical neural network?

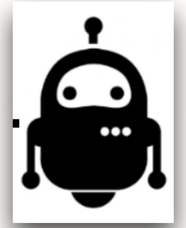
Quantum Games for/as Research

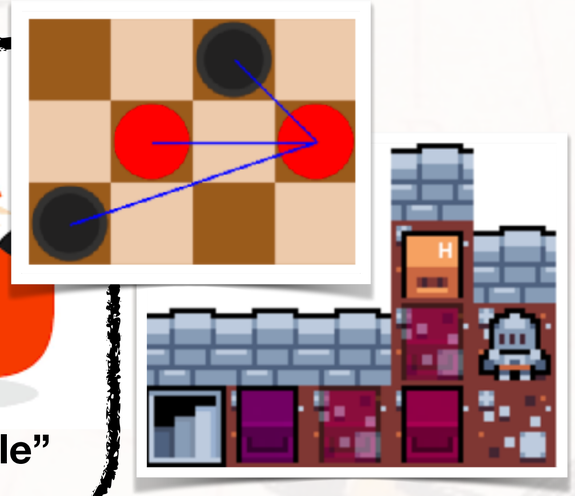
Gamify Quantum Problem
Solve game with AI

Quantum AI for
(Quantum) Games

Quantum Game Theory







$$\rho_{i\ldots N}$$

$$\downarrow$$

$$u_i(\rho) = \mathrm{Tr}(\rho R_i)$$

$$\downarrow$$

$$u_i(\rho) \geq u_i(\rho_i' \otimes \rho_{\neg i})\ \forall i$$

Quantum Nash Equilibrium

Can a classical AI optimize strategies for quantum systems?
Does a quantum circuit do better than a classical neural network?

This module

Lecture 1: Basics, (Un)supervised learning

Lecture 2: Reinforcement learning

Lecture 3: Quantum Machine Learning

Lecture 4: Quantum Game Theory

Most importantly: *we will adjust to your preferences as we go*.

(Un)supervised learning

Supervised learning

Learning from examples

$$\{(\mathbf{X}, \mathbf{Y})\}$$

Inputs/Samples **Targets/Labels**

$\{x_1, y_1\} = \{$, 1 $\}$

$\{x_2, y_2\} = \{$, 0 $\}$

e.g. Learn $p(y \mid x)$

Classification

$\{\mathbf{X}\}$

$x_1 =$

$x_2 =$

$p(x)$

(Un)supervised learning

Supervised learning

Learning from examples

$$\{(\mathbf{X}, \mathbf{Y})\}$$

Inputs/Samples **Targets/Labels**

$\{x_1, y_1\} = \{$, 1 $\}$

$\{x_2, y_2\} = \{$, 0 $\}$

e.g. Learn $p(y \mid x)$

Classification

Unsupervised learning

Learning about examples

$$\{\mathbf{X}\}$$

Inputs/Samples ~~**Targets/Labels**~~

$x_1 =$

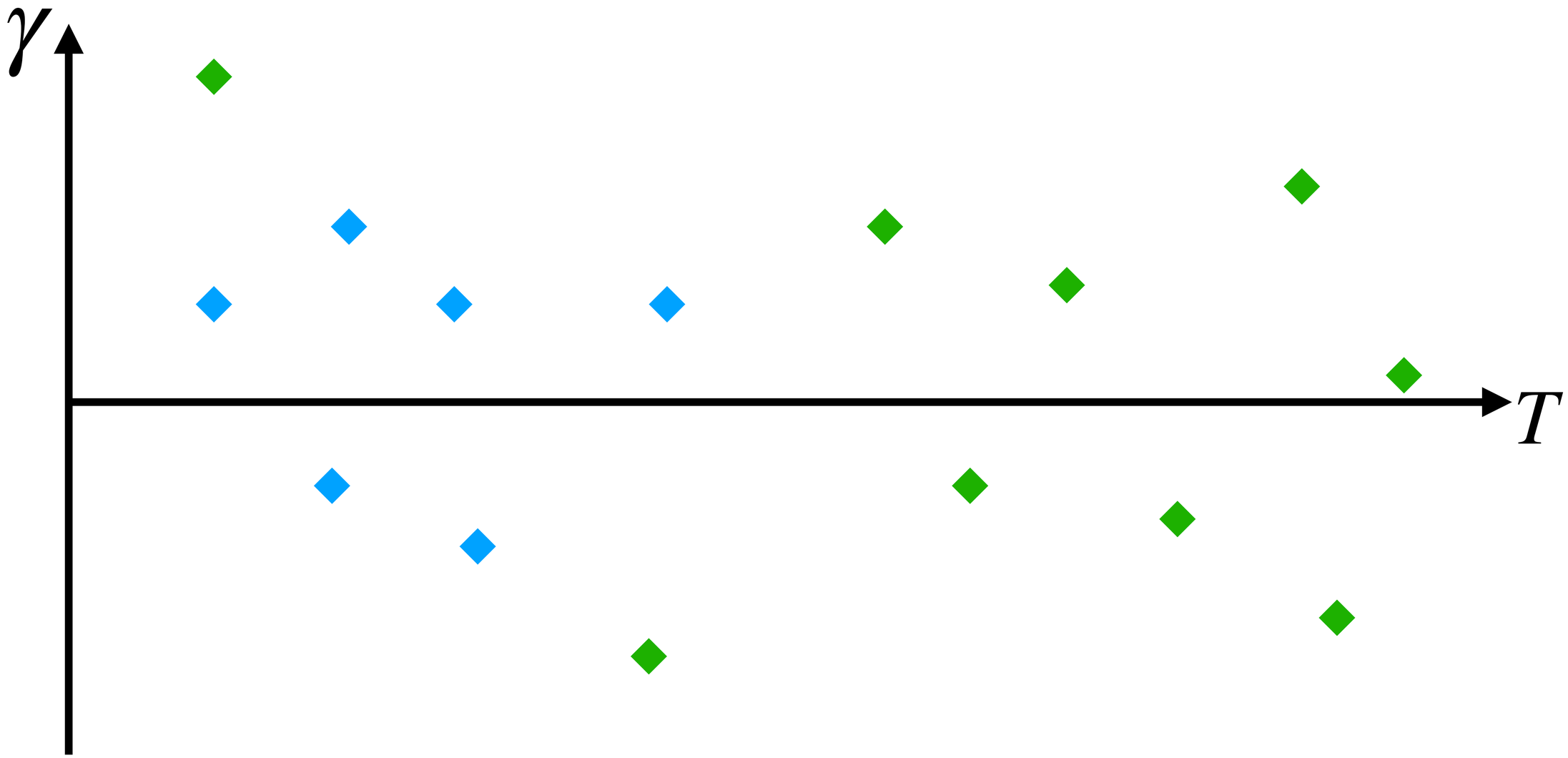
$x_2 =$

e.g. Learn $p(x)$

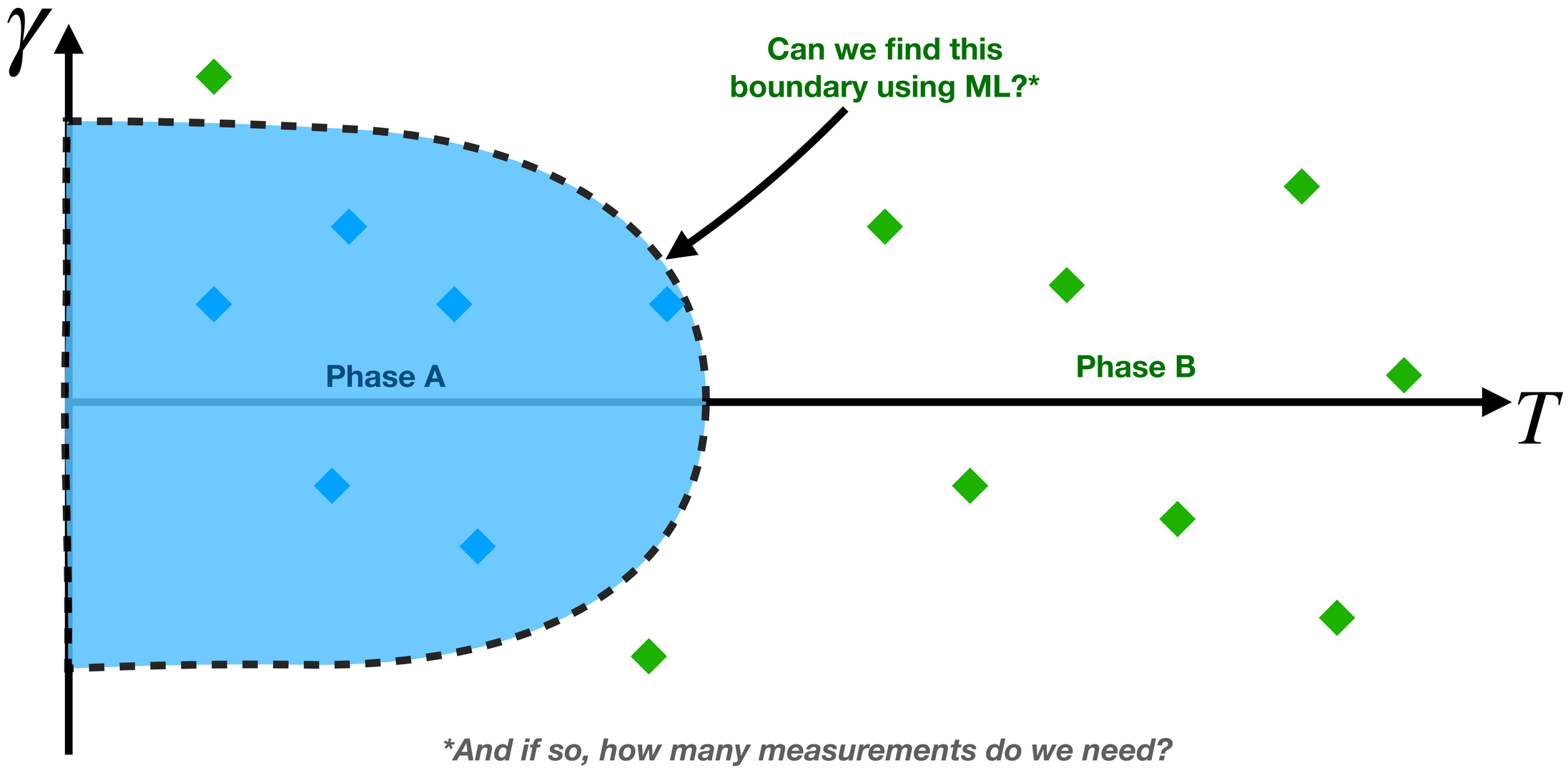
Representation learning

thispersondoesnotexist.com

ML in cond-mat started w/ phase identification



ML in cond-mat started w/ phase identification



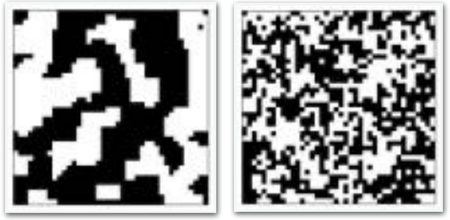
**And if so, how many measurements do we need?*
Or where should we measure next to get the best informed boundary?

Data

The full wavefunction

$$|\psi\rangle = \{c_1, c_2, \ldots, c_D\}$$
$$D = 2^N$$

Projective measurements



$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 & \ldots \\ 0 & 0 & 1 & 1 & 1 & 1 & \ldots \\ \vdots & \ddots & & & & & \vdots \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}$$

Observables

$$\{\langle\sigma_0^z\rangle(t), \langle\sigma_1^z\rangle(t), \ldots\} \qquad \langle\sigma_i^z\rangle = \langle\psi|\sigma_i^z|\psi\rangle$$

$$\{\langle\psi|c_i^\dagger c_j|\psi\rangle\}$$

Entanglement Spectra

$$\rho_A = \mathrm{Tr}_B|\psi\rangle\langle\psi|$$

eigenvalues $\{\epsilon_i\}$

Intro to Neural Networks

Parameterizing $p(y \mid x)$

$$p(y \mid x) = \ldots f_2(W_2 \cdot f_1(W_1 \cdot x + b_1) + b_2) \ldots$$

Matrix Vector

$$F(x, W) = (f^D \circ W^D) \ldots (f^3 \circ W^3)(f^2 \circ W^2)(f^1 \circ W^1)x$$

(Non-Linear) Activation Function

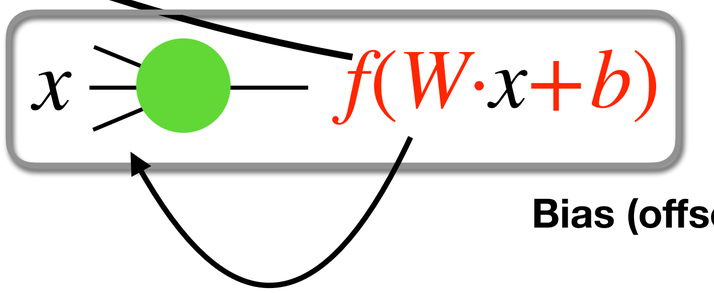
$$f(x) = 1/(1 + e^{-x})$$

```
def sigmoid(x):
    return 1/(1+jnp.exp(-x))
```

$$f(x) = \max(0, x)$$

```
def relu(x):
    return jnp.maximum(0, x)
```

A neuron

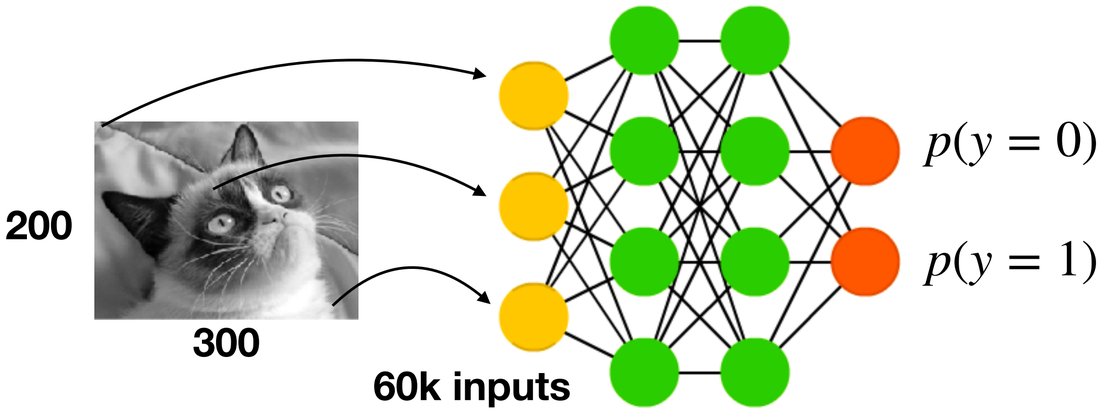


Bias (offset)

Weights W

Intro to Neural Networks

Parameterizing $p(y \mid x)$

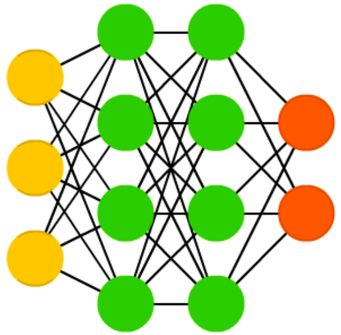



```python
def initialize_feed_forward_network_params(layer_sizes):
    parameters = []
    for i in range(len(layer_sizes)-1):
        parameters.append( [jnp.random(layer_sizes[i], layer_sizes[i+1]), # Weight matrix
                            jnp.random(layer_sizes[i]) ] )                # Bias vector
```

```python
# Feed through the network, layer by layer
for weights, biases in parameters:
    outputs = jnp.dot(weights, activations) + biases
    activations = sigmoid(outputs)
```

Intro to Neural Networks

Learning: optimize the weights and biases so that our model predicts examples correctly



$$p(y=0) = 0.31$$
$$p(y=1) = 0.69$$

$$\tilde{y} = \begin{bmatrix} 0.31 \\ 0.69 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\textbf{Loss}(\{x\}, \{y\}) = \sum_x (\tilde{y} - y)^2$$

W, b

```
def loss(params, inputs, targets):
    preds = predict(params, inputs)
    return jnp.mean( (preds - targets)**2 )
```

```
@jit
def update(params, x, y):
  grads = grad(loss)(params, x, y)
  return [(w - step_size * dw, b - step_size * db)
          for (w, b), (dw, db) in zip(params, grads)]
```

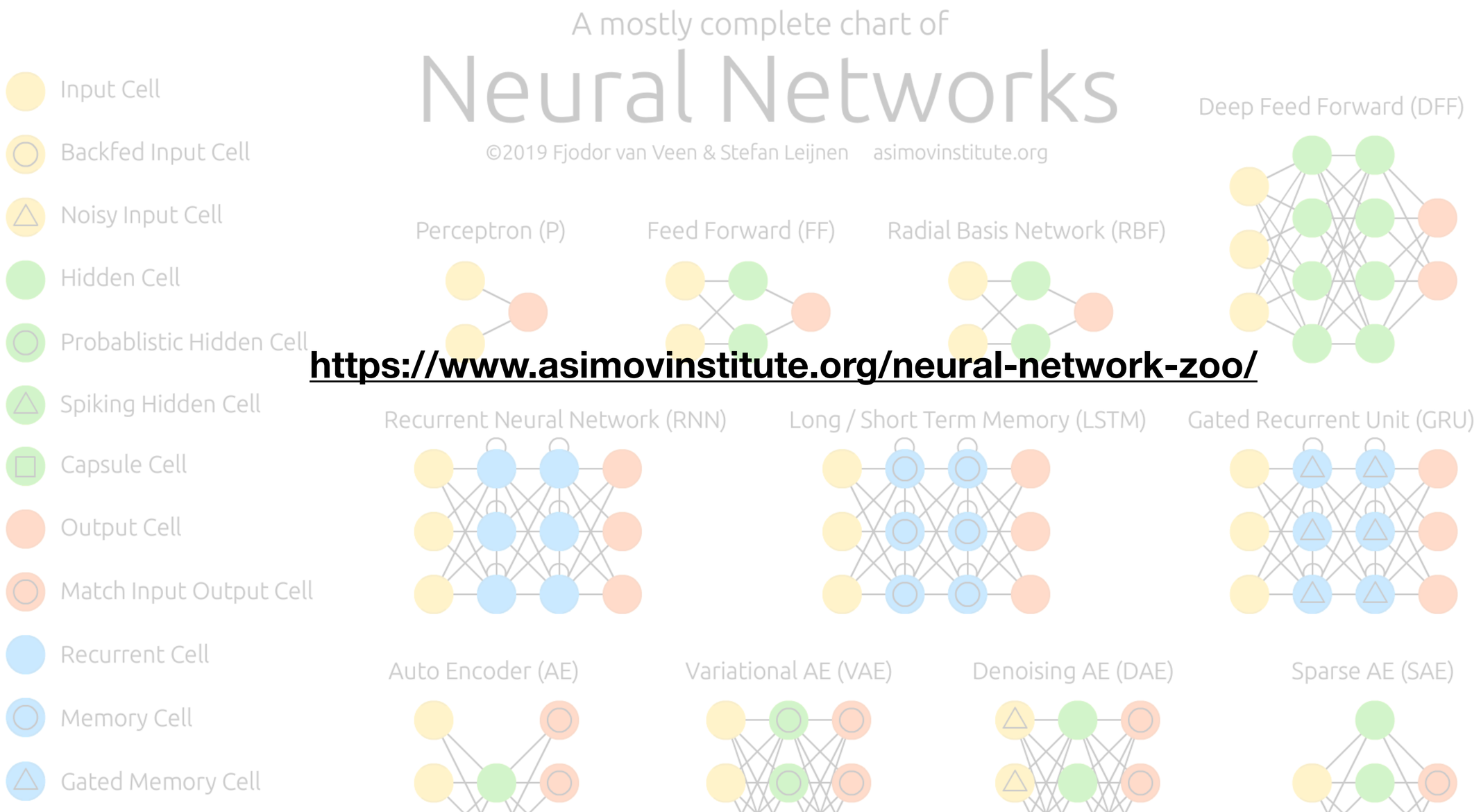
!

Learning Rate / Step Size

$$W \rightarrow W - \alpha \nabla_W L$$

(Stochastic) Gradient Descent. Many improvements -> many optimizers (in particular, `Adam`)

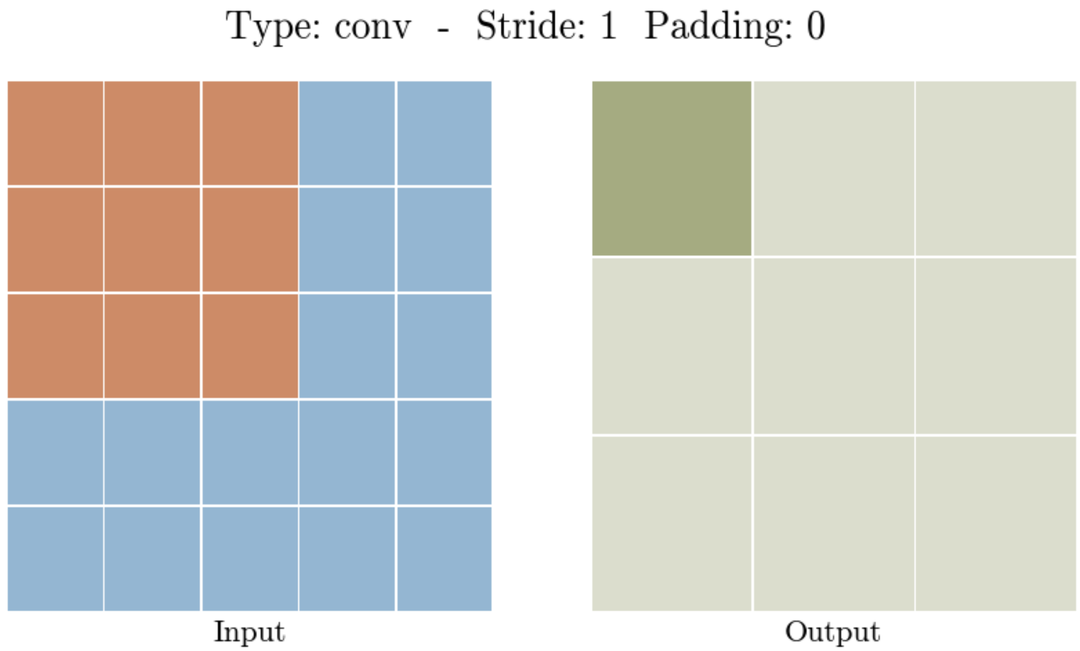
Intro to Neural Networks

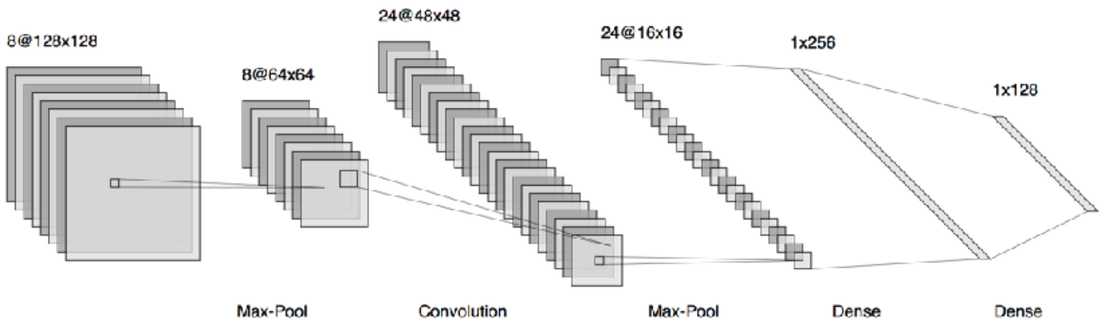


https://www.asimovinstitute.org/neural-network-zoo/

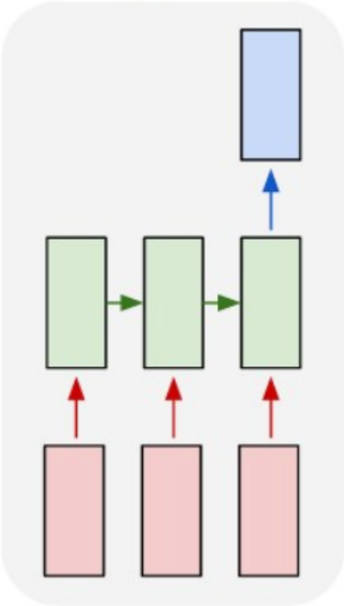
Two more important architectures

Convolutional Networks





Recurrent Neural Networks



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

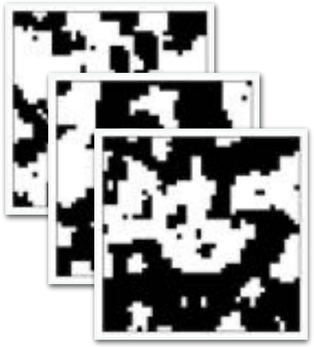
A concrete example

Learn $p(y\,|\,x)$ ← "The model"

Classical $H = -J \sum_{<ij>} \sigma^z \sigma^z$

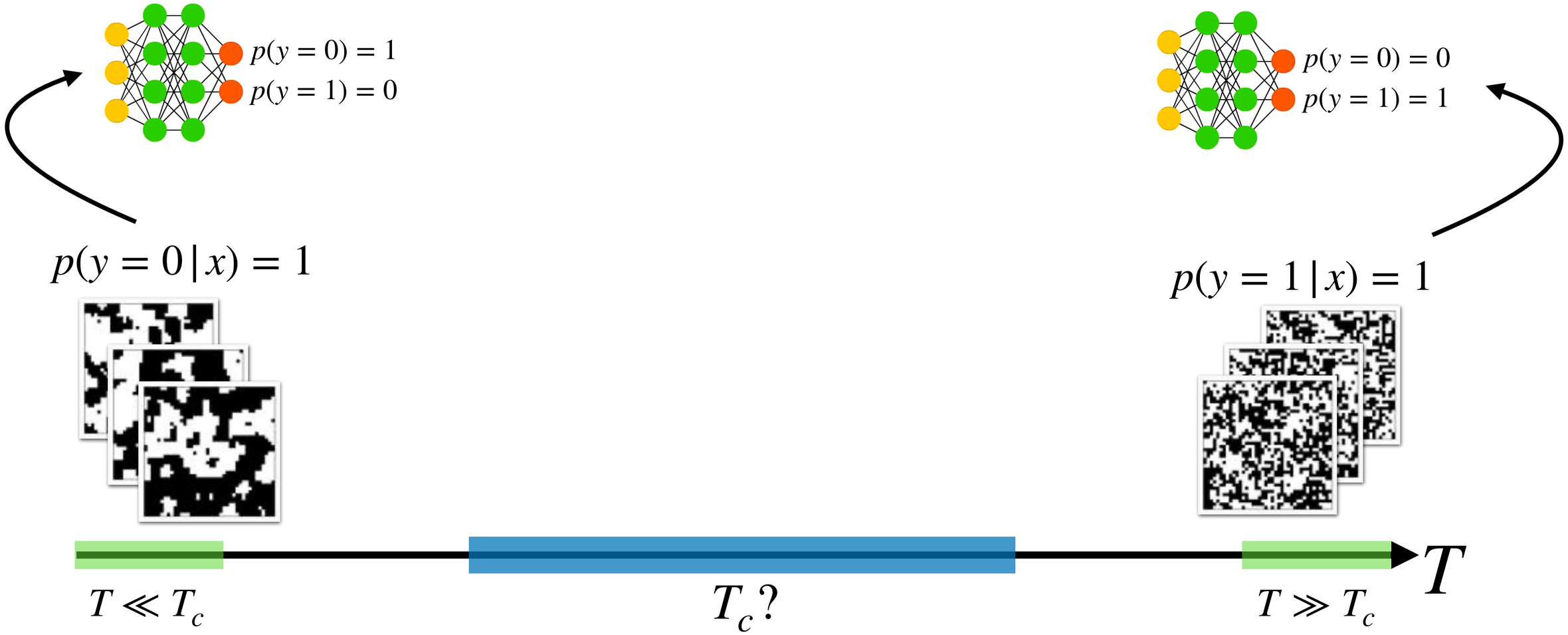
- At very low temperatures, ferromagnet
- At very high temperatures, paramagnet

$p(y = 0\,|\,x) = 1$

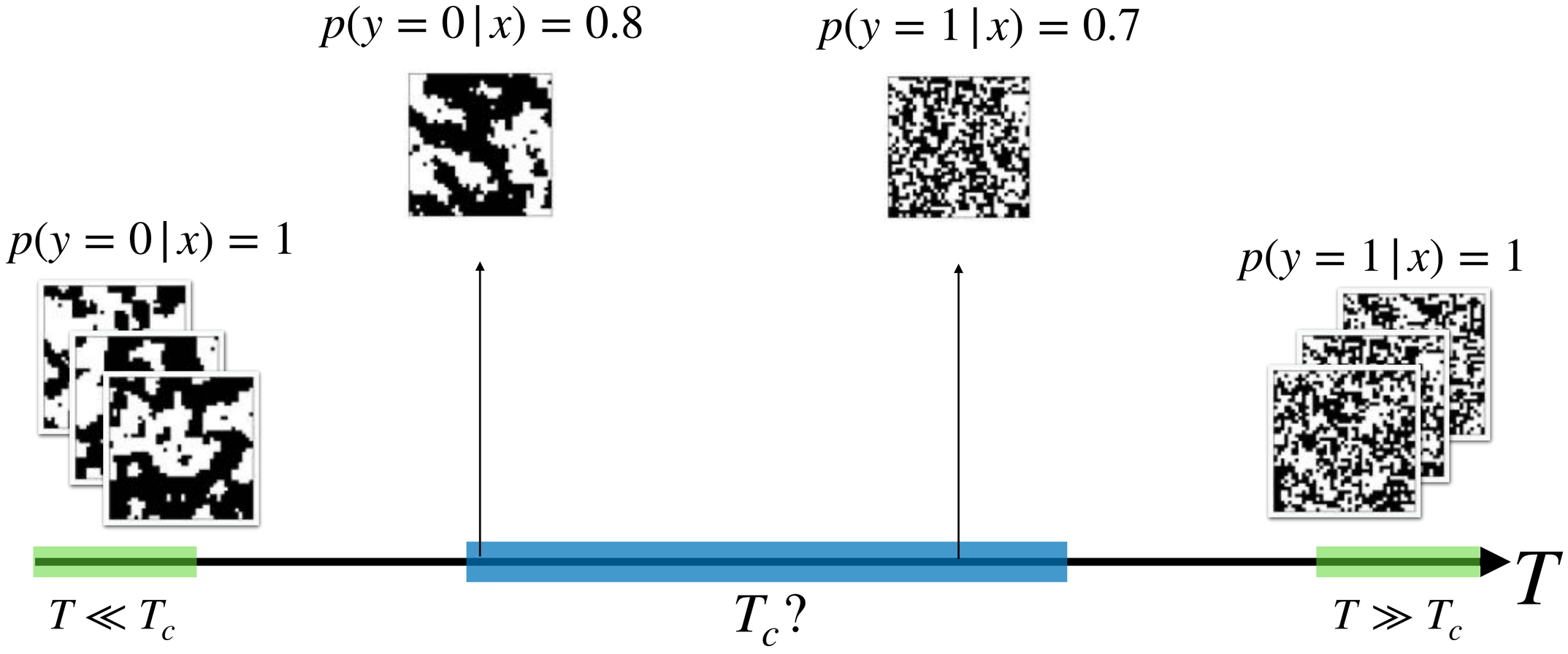


$p(y = 1\,|\,x) = 1$

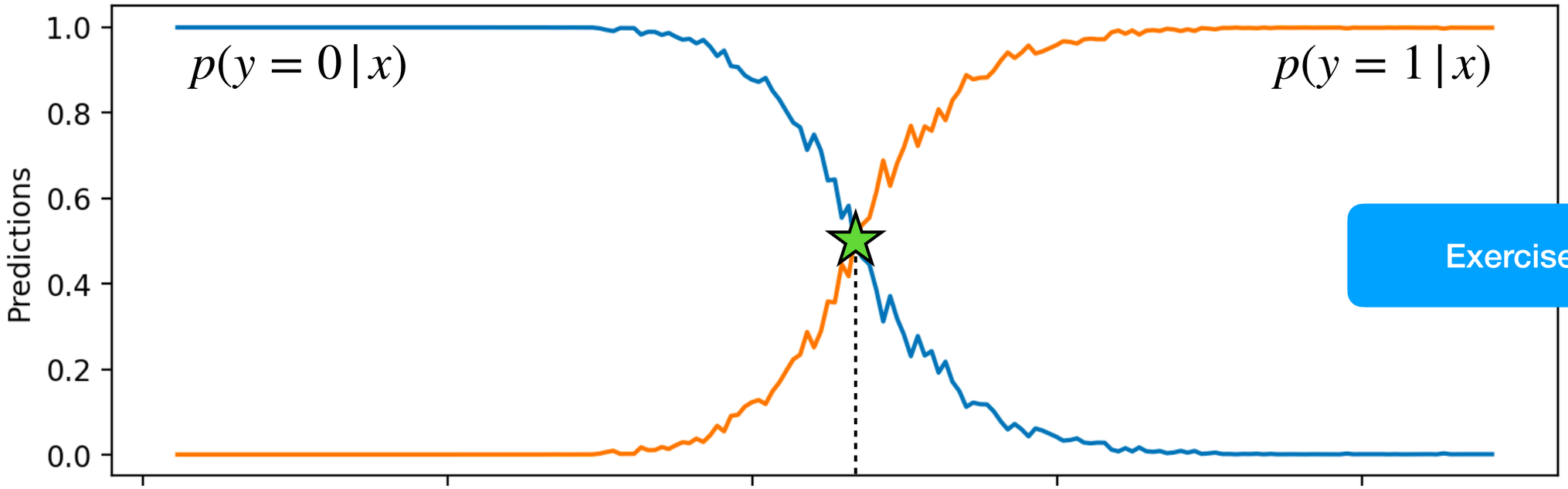
A concrete example



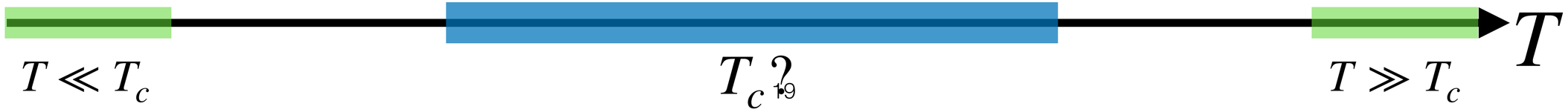
A concrete example



A concrete example



Exercise class!



This is the general paradigm

Quantum

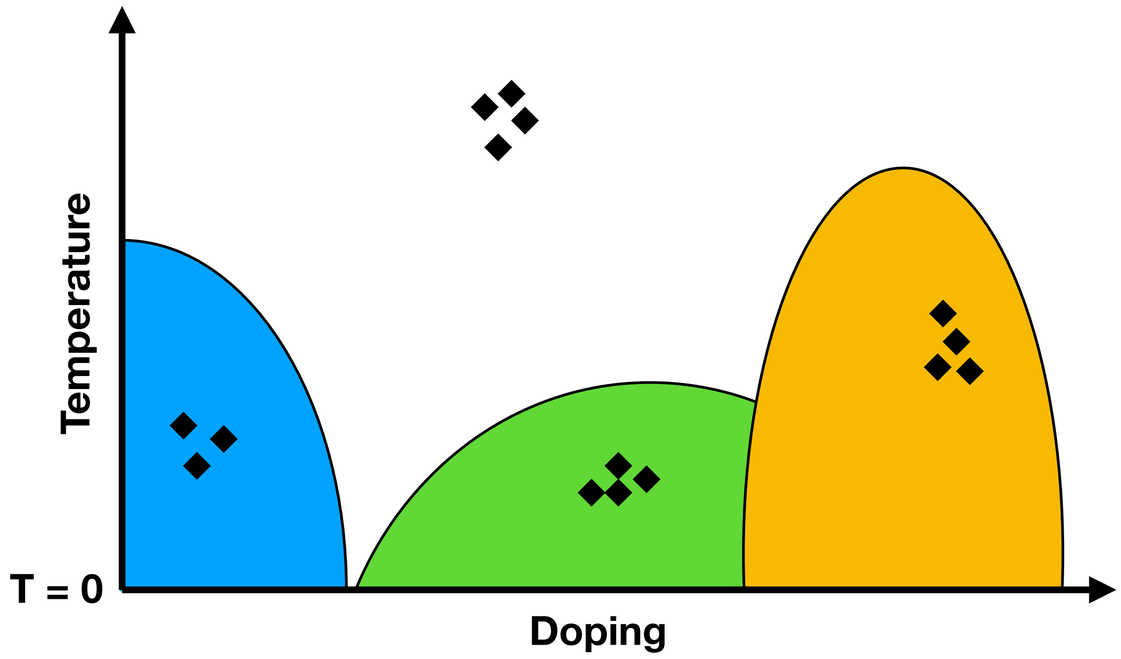
$$H = -J \sum_{<ij>} \sigma^z \sigma^z + h \sum_i \sigma^x$$

$J/h \gg 1,\ y = 0$

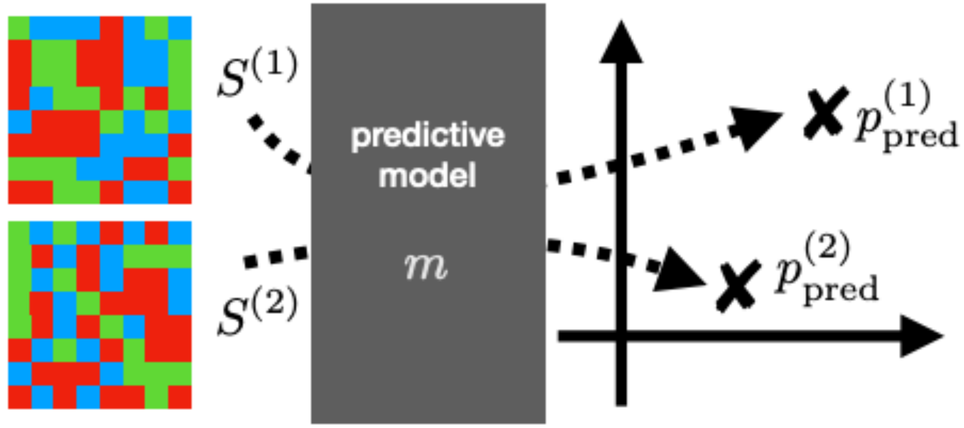
$J/h \ll 1,\ y = 1$

Classical

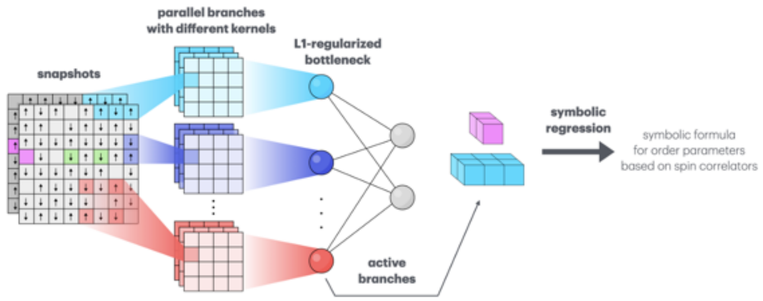
Quantum



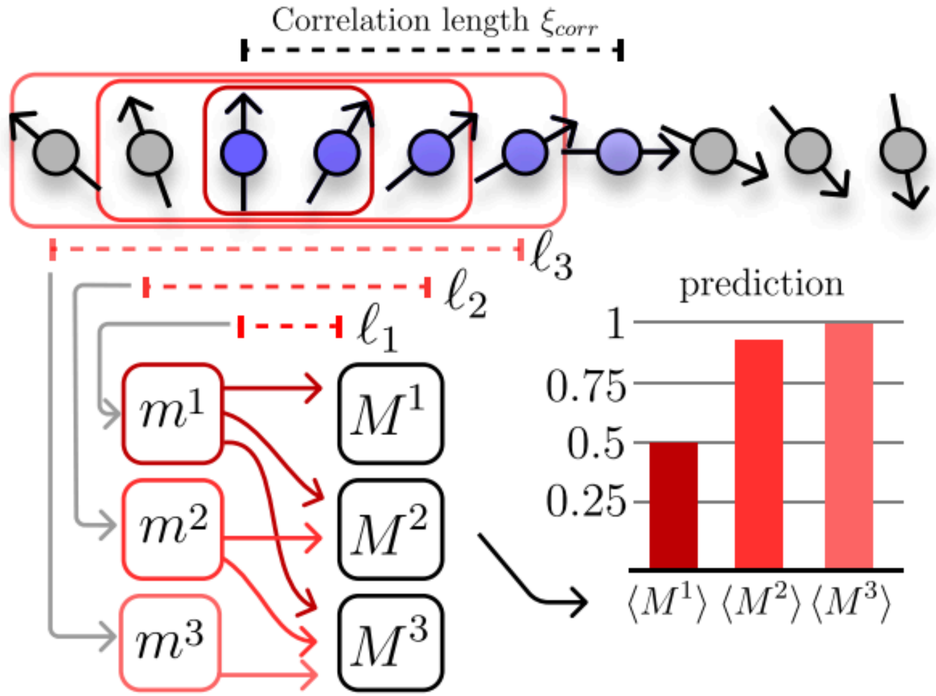
Some highlighted examples



https://arxiv.org/abs/1812.00895



https://arxiv.org/abs/2411.02237



https://arxiv.org/abs/2103.15855

There are many, many more supervised applications

These are a *tiny* fraction of them just to get you started

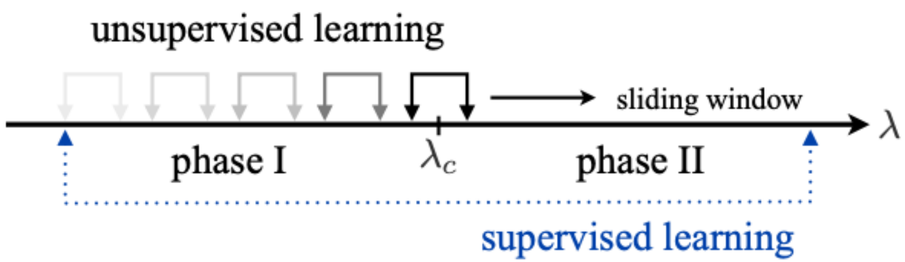
Topic / Application	Typical ML Methods
Learn ground & excited states	Neural-network quantum states (RBMs, CNNs, Transformers)
Phase recognition & classification	CNNs, autoencoders, clustering, t-SNE, diffusion maps
Unsupervised phase discovery	Variational autoencoders (VAEs), self-organizing maps
Quantum state tomography	Generative models, Bayesian neural networks
Entanglement estimation	Supervised regression, GNNs
Learning effective Hamiltonians	Graph neural networks, symbolic regression
Phase diagram prediction (Hubbard, Heisenberg)	CNNs, ensemble classifiers, random forests
Accelerating DMFT solvers	Neural surrogates, Gaussian process regression
Green's function & self-energy interpolation	Deep regression, kernel methods
Learning low-energy theories	Physics-informed neural networks (PINNs)
Predicting material properties	Graph neural networks, message passing networks
Discovering topological materials	Symmetry-aware deep networks
ML-accelerated DFT / XC functional learning	Deep neural networks, kernel ridge regression
Variational circuit optimization	Reinforcement learning (RL), policy gradient
Quantum control & pulse shaping	Deep RL, LSTM controllers
Noise characterization & mitigation	Autoencoders, GANs
Quantum circuit synthesis	Evolutionary algorithms, supervised models

There a cations

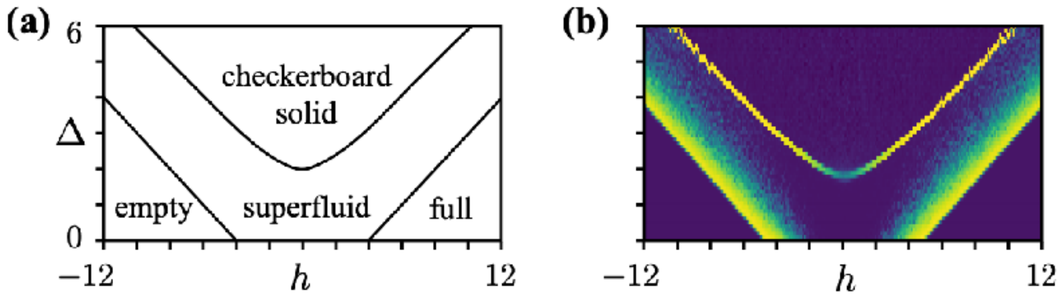
Topic / Application	Typical ML Methods
Cosmological parameter inference	Simulation-based inference (neural density estimators, normalizing flows)
Fast emulation of N-body simulations	CNN-based emulators, graph-based surrogates
Dark matter mapping	CNNs, UNets, diffusion models
Reconstruction of initial conditions	Invertible networks, VAEs
Foreground separation & denoising	CNNs, blind source separation networks
Parameter estimation	Likelihood-free inference (flows, summary networks)
Non-Gaussianity & anomaly detection	Autoencoders, normalizing flows
Strong lens detection	CNNs, transformer-based vision models
Mass distribution reconstruction	CNN inversion, Bayesian neural networks
Weak lensing shear estimation	Regression CNNs, equivariant networks
Galaxy morphology classification	CNNs, ResNets
Generative modeling of galaxies	GANs, VAEs, diffusion models
Emulation of hydrodynamic sims	Surrogate deep nets, physics-informed models
Feedback parameter inference	Bayesian deep learning
Event classification (supernovae, GW)	RNNs, temporal CNNs, transformers
Anomaly detection in light curves	Autoencoders, one-class SVMs
Photometric redshift estimation	Deep regressors, Gaussian processes
Dark energy / inflation emulators	Surrogate modeling, Gaussian processes
Likelihood-free cosmological inference	Neural density estimation, normalizing flows
Model comparison / selection	Bayesian neural networks, classification pipelines

Supervised unsupervised

Or vice versa?



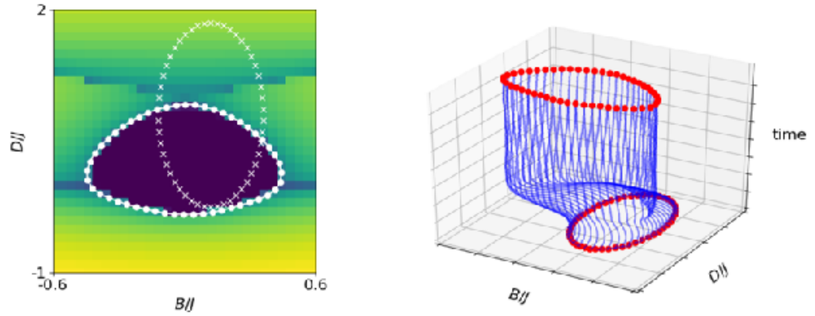
https://arxiv.org/abs/1707.00663



Learning by Confusion

and

Discriminative Cooperative Networks



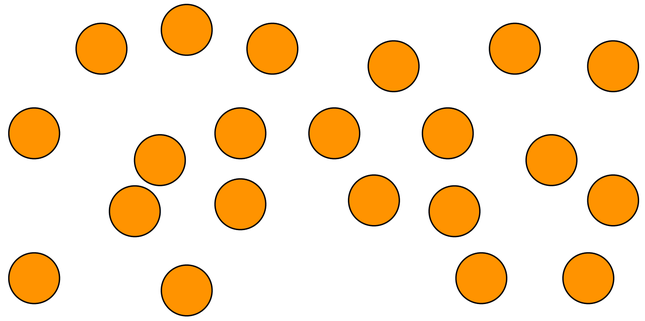
https://arxiv.org/abs/1610.02048

https://arxiv.org/abs/1706.08111

Learning by Confusion

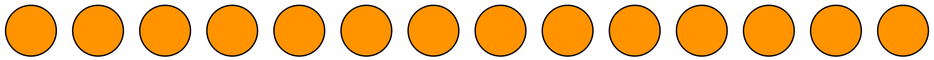
Finding the most consistent labelling

Given N datapoints, cluster into 2 connected subsets



Exponentially many ways!

Given N *ordered* datapoints, there are only linearly many binary splits!

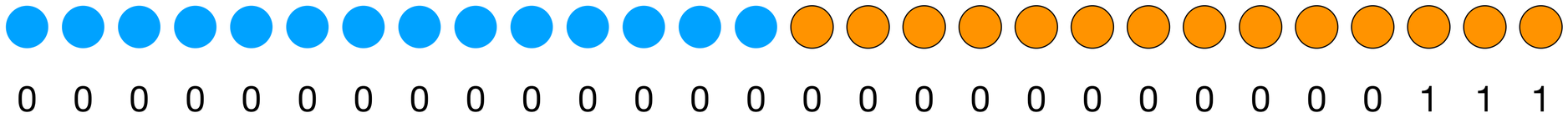


Learning by Confusion

Finding the most consistent labelling

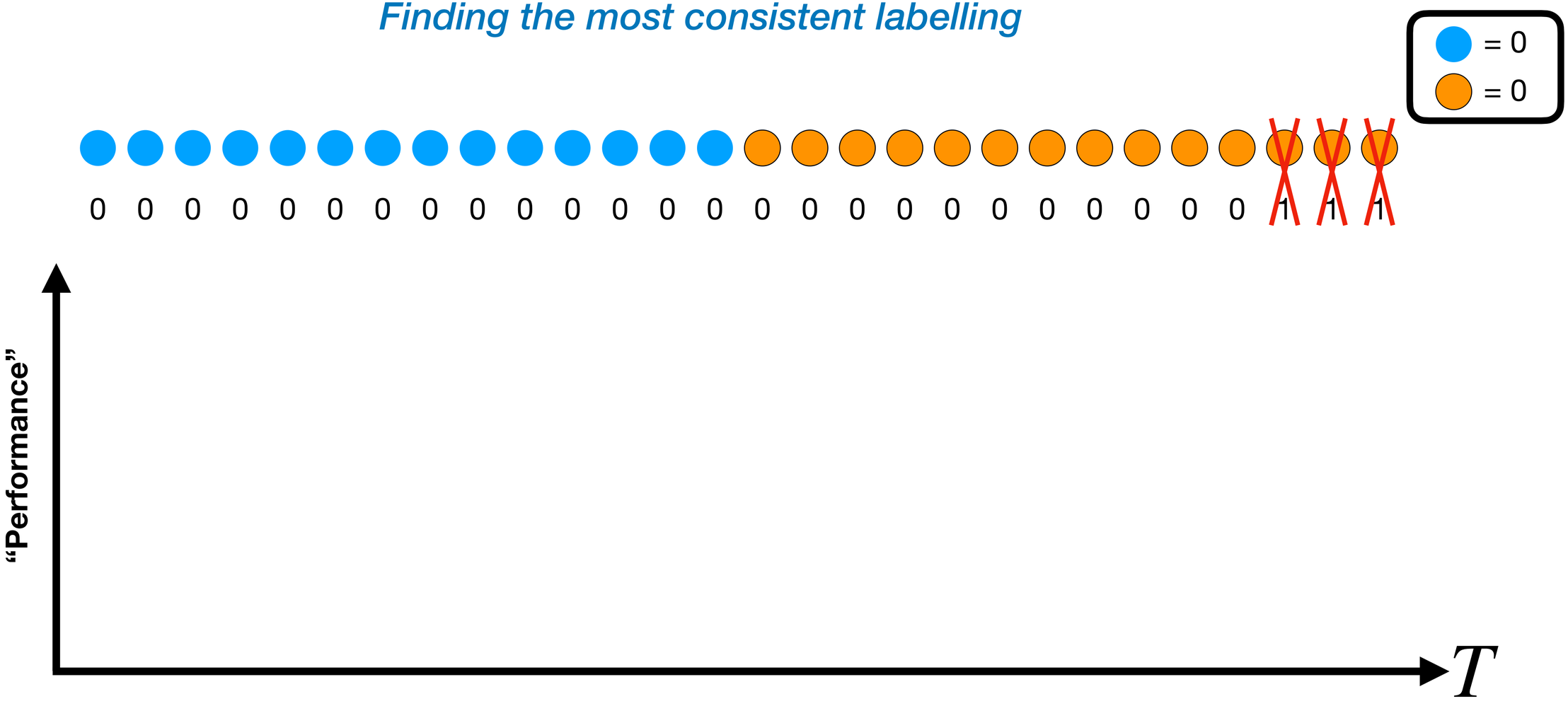
Learning by Confusion

Finding the most consistent labelling



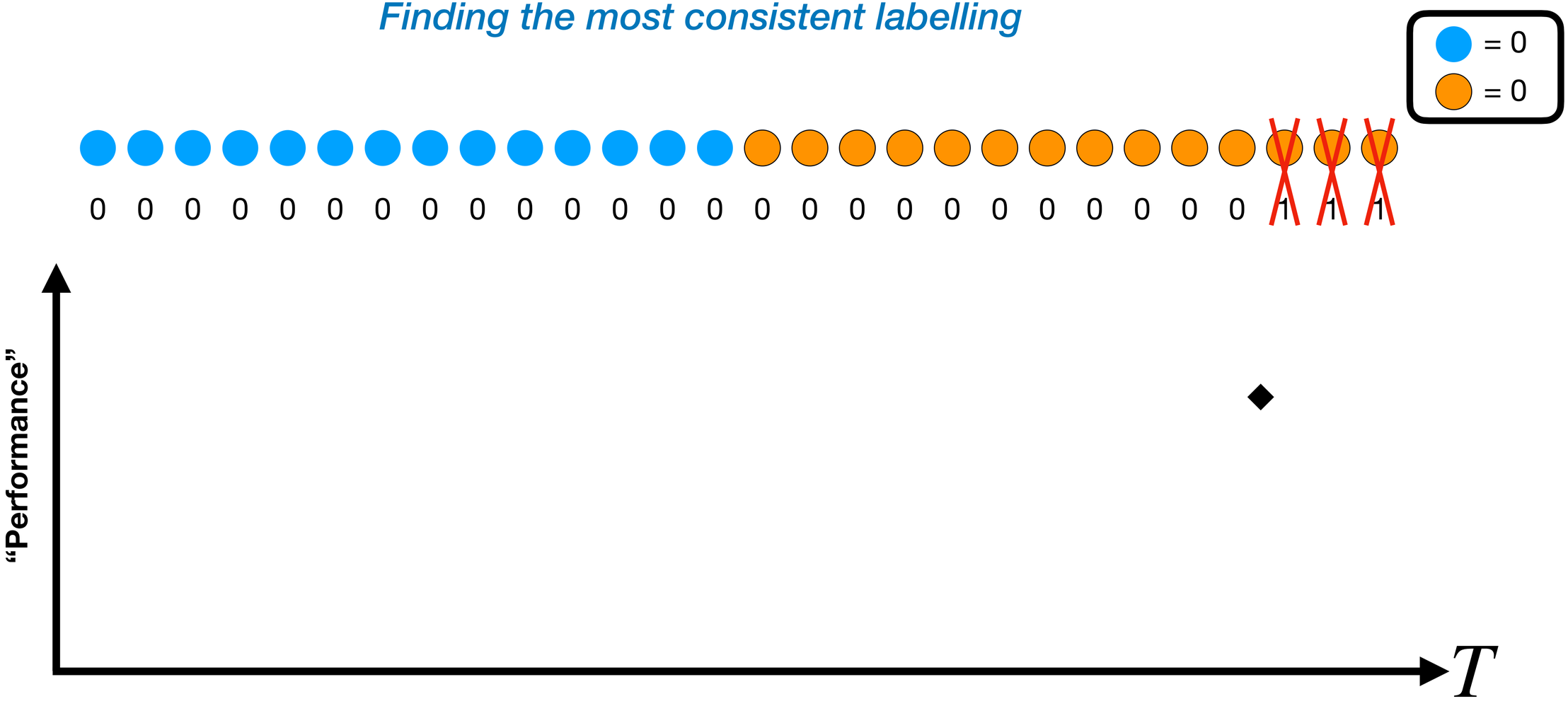
Learning by Confusion

Finding the most consistent labelling



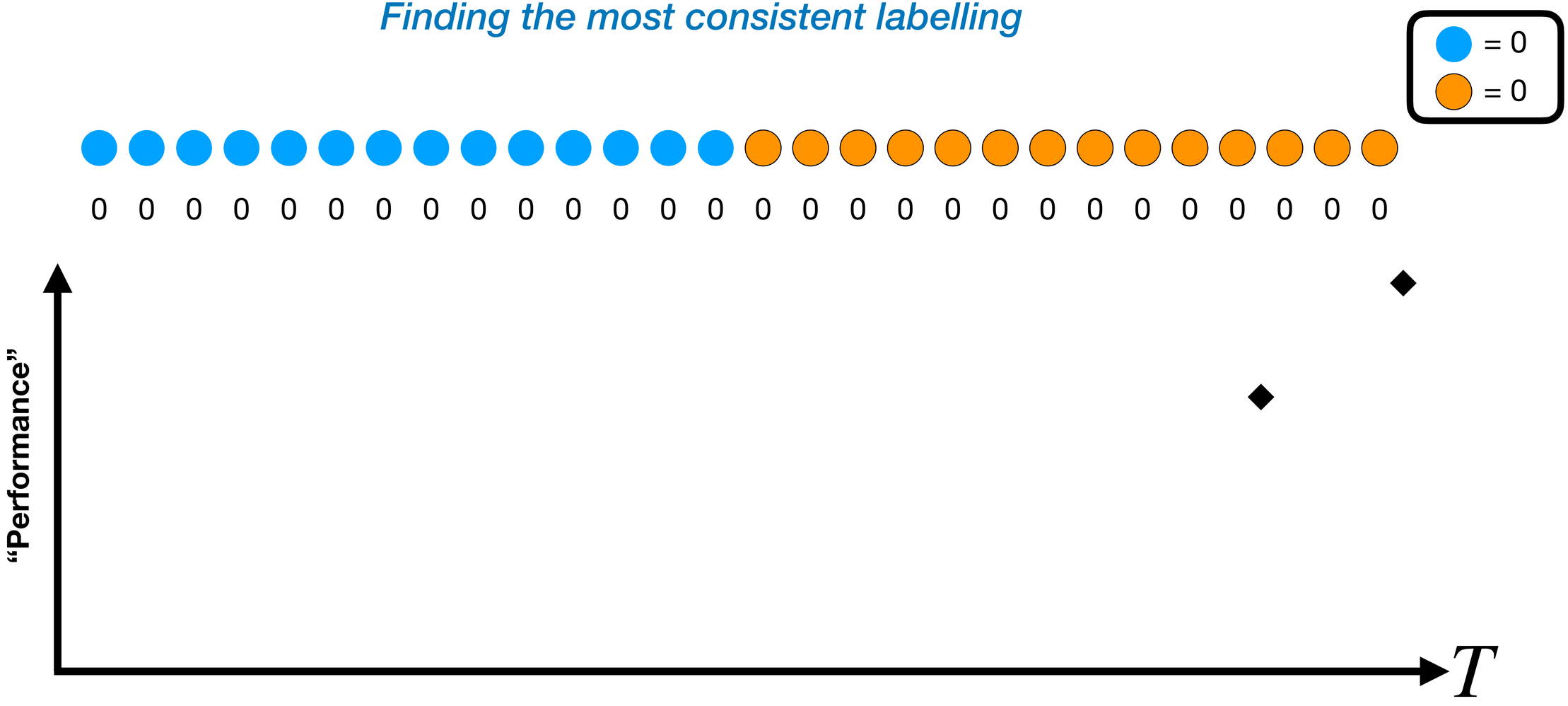
Learning by Confusion

Finding the most consistent labelling



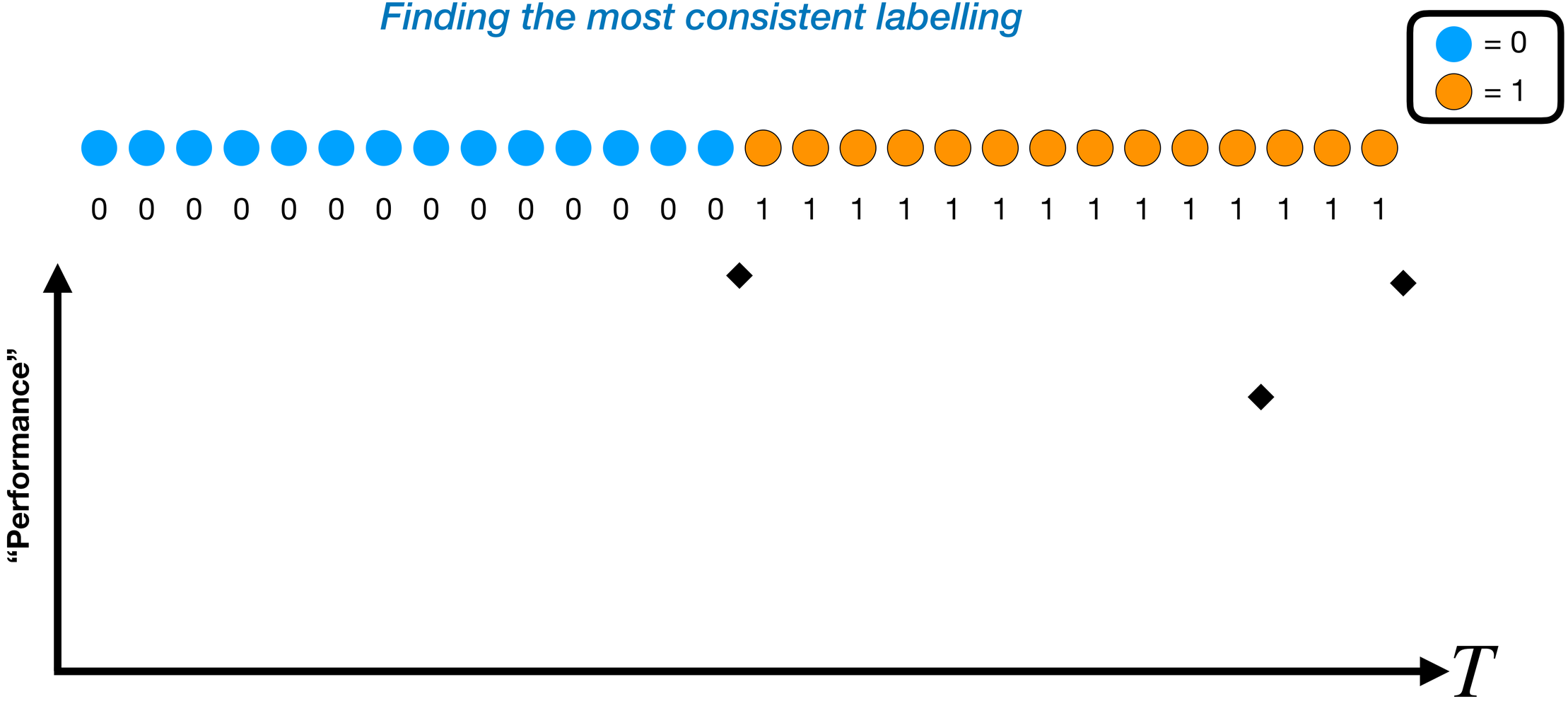
Learning by Confusion

Finding the most consistent labelling



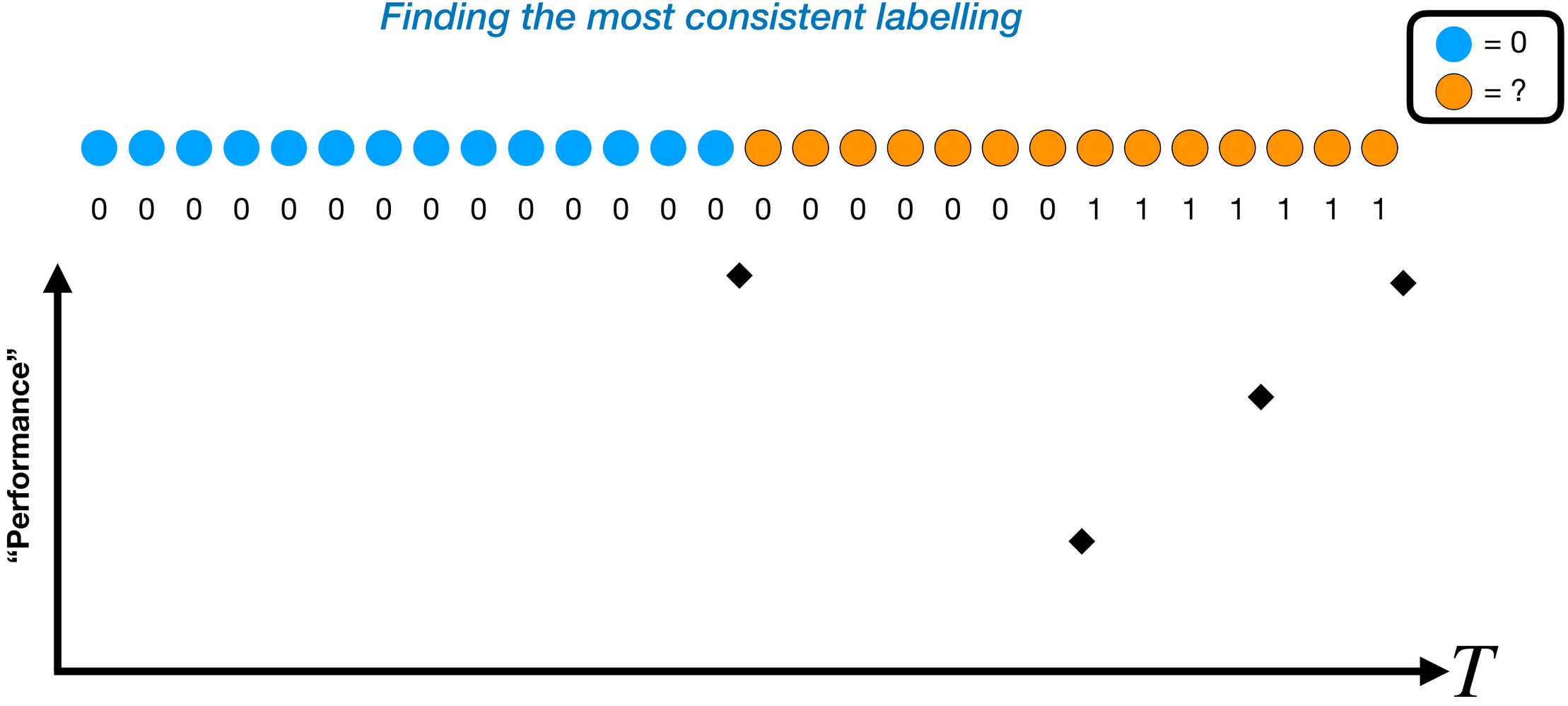
Learning by Confusion

Finding the most consistent labelling



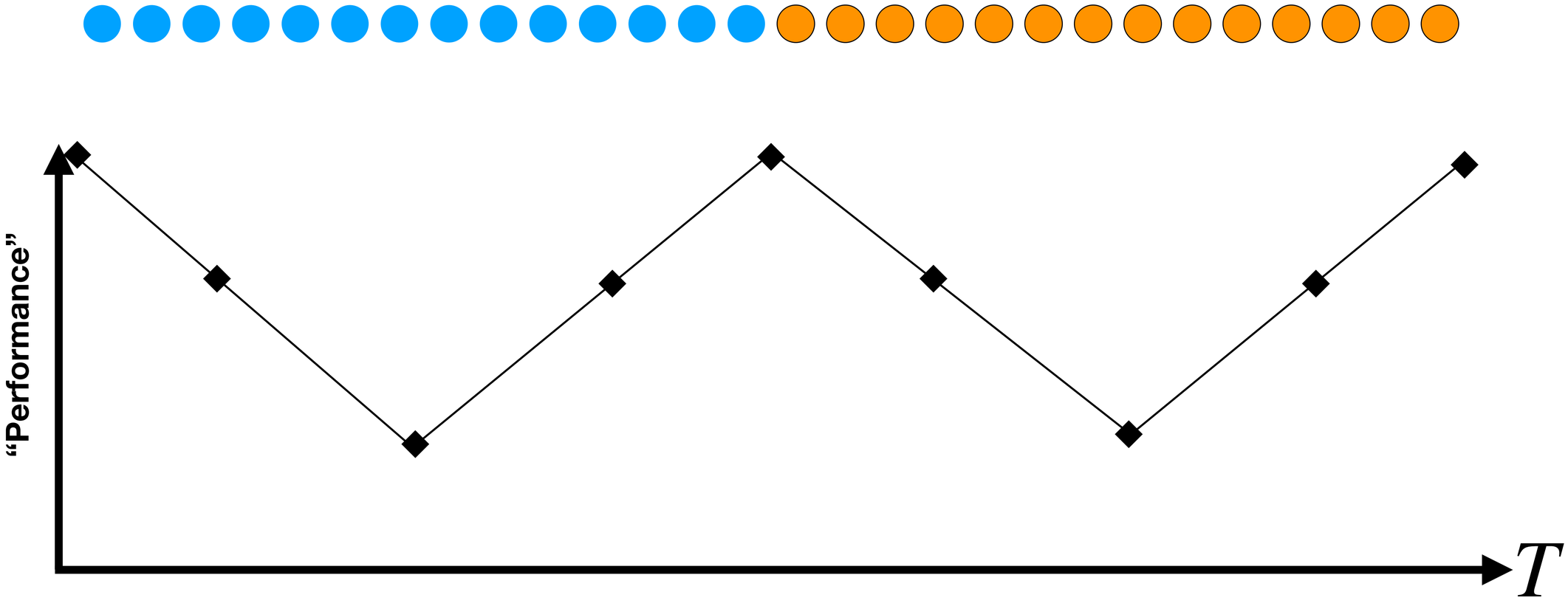
Learning by Confusion

Finding the most consistent labelling



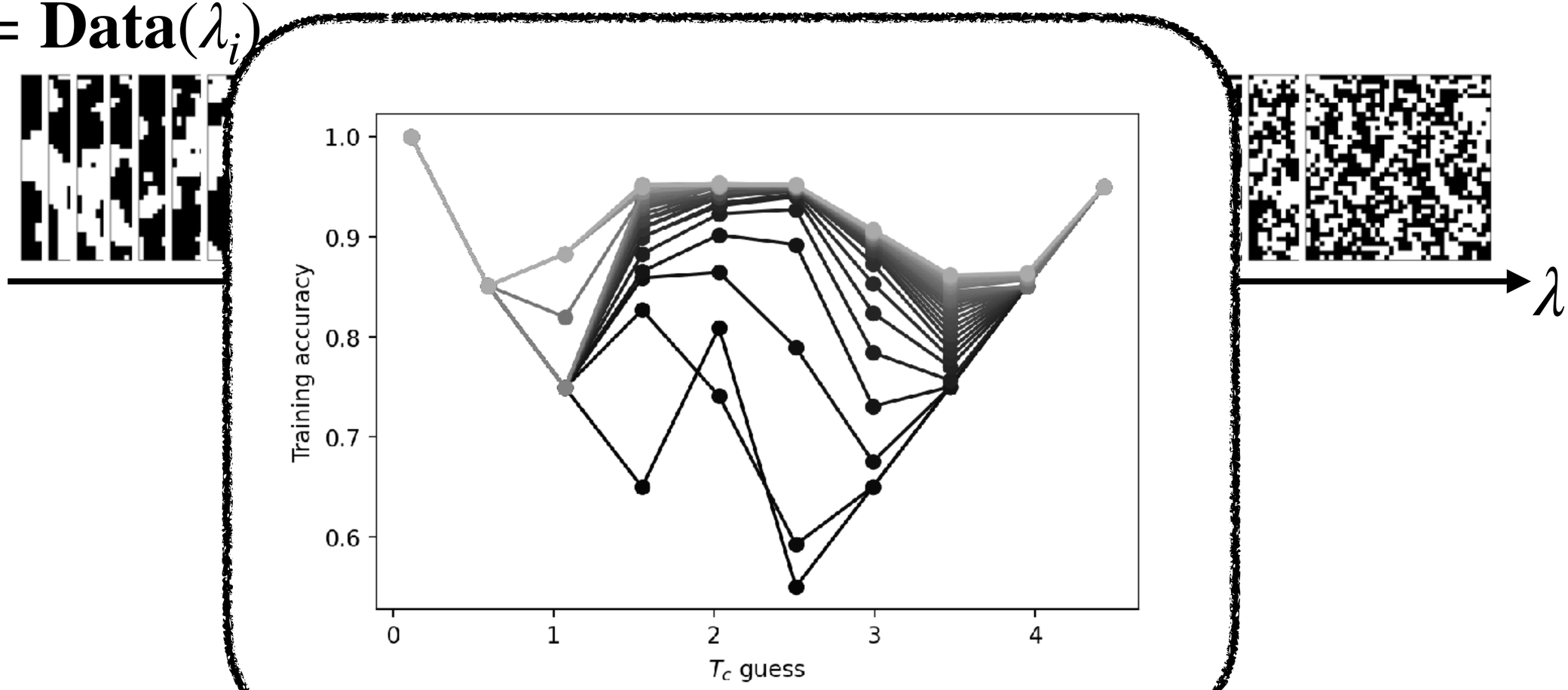
Learning by Confusion

Finding the most consistent labelling



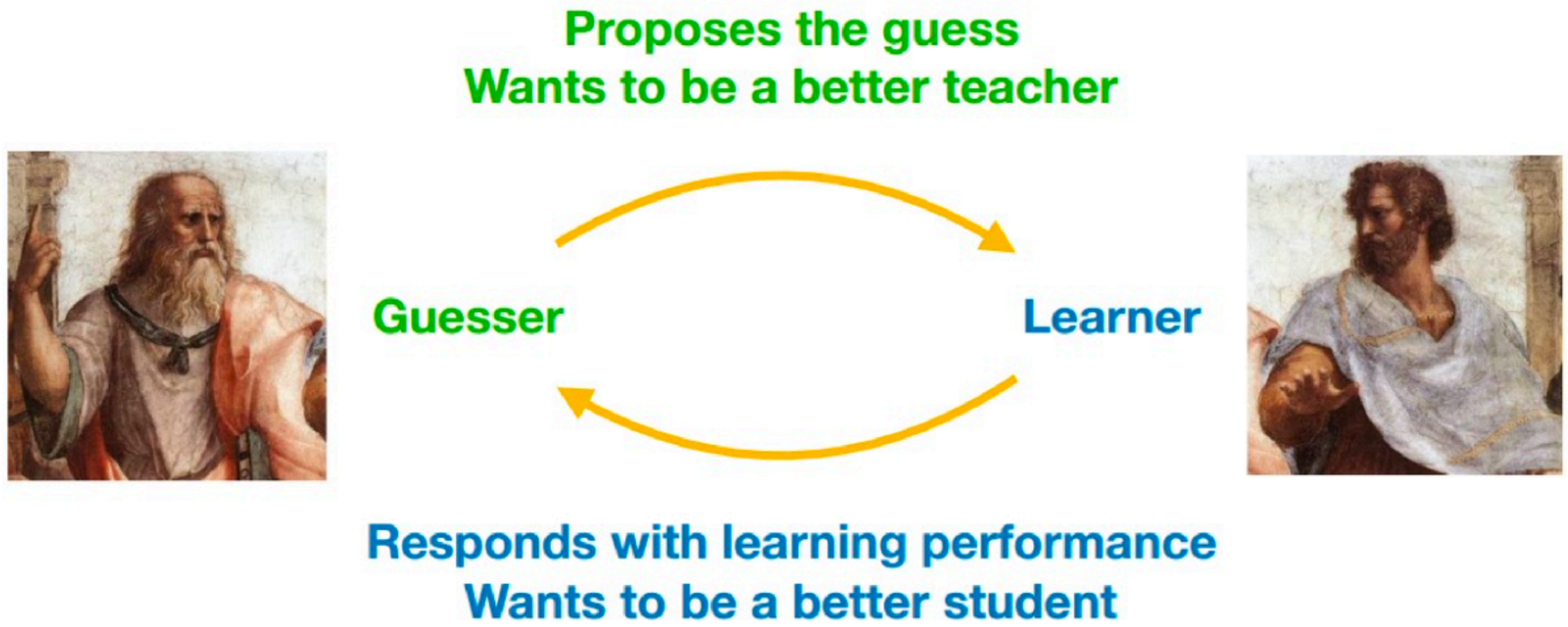
Learning by Confusion

$x_i = \mathbf{Data}(\lambda_i)$




```
for guess
    accuracy_vs_T = train_network_with_guessed_labels(Ts, Tc=guess)
```

Automatic Confusion



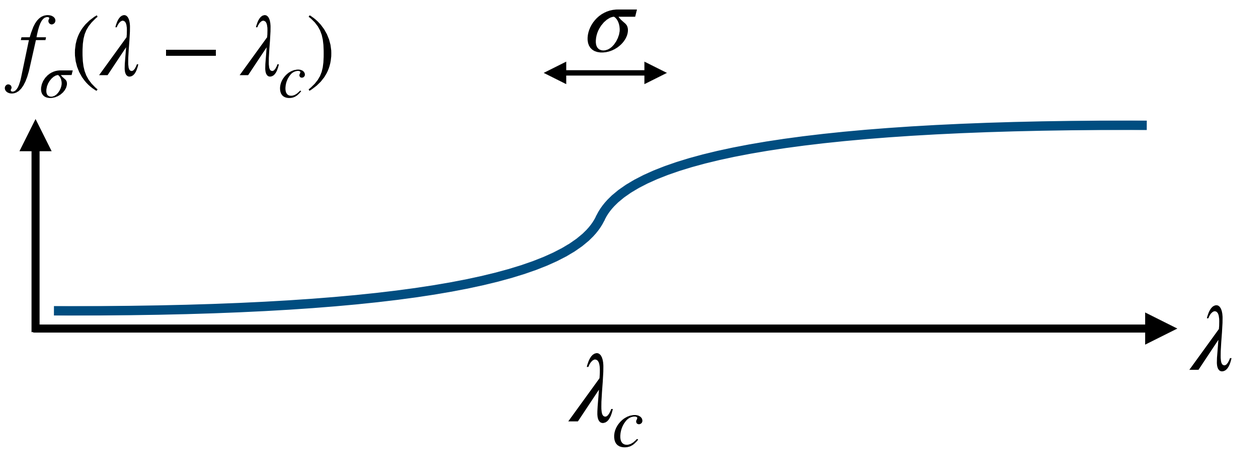
Discriminative Cooperative Networks (DCN)

https://arxiv.org/abs/1706.08111

Improving the search

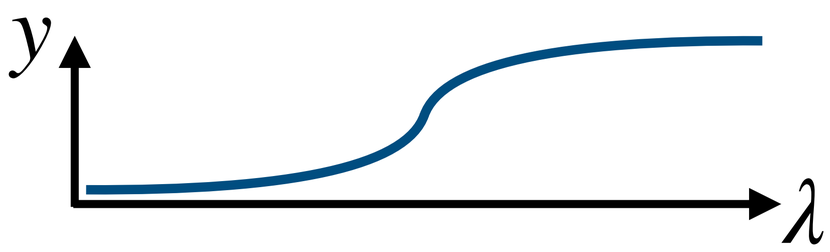
$x_i = \mathbf{Data}(\lambda_i)$

$y_i = f_\sigma(\lambda - \lambda_c)$



Improving the search

Step 1



Guesser : $\lambda_i \rightarrow y_i$

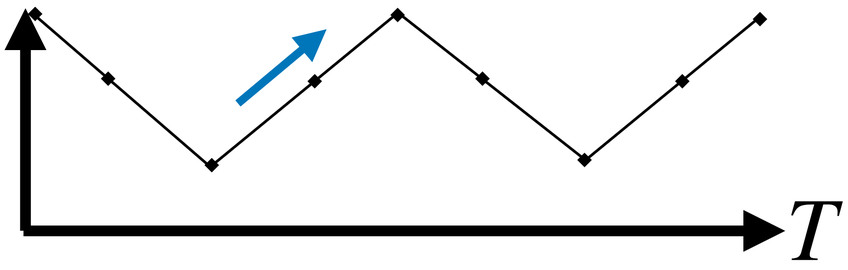
Step 2

Learner : $x_i \rightarrow \tilde{y}_i$

$L_{\text{Learner}} = (y - \tilde{y})^2$

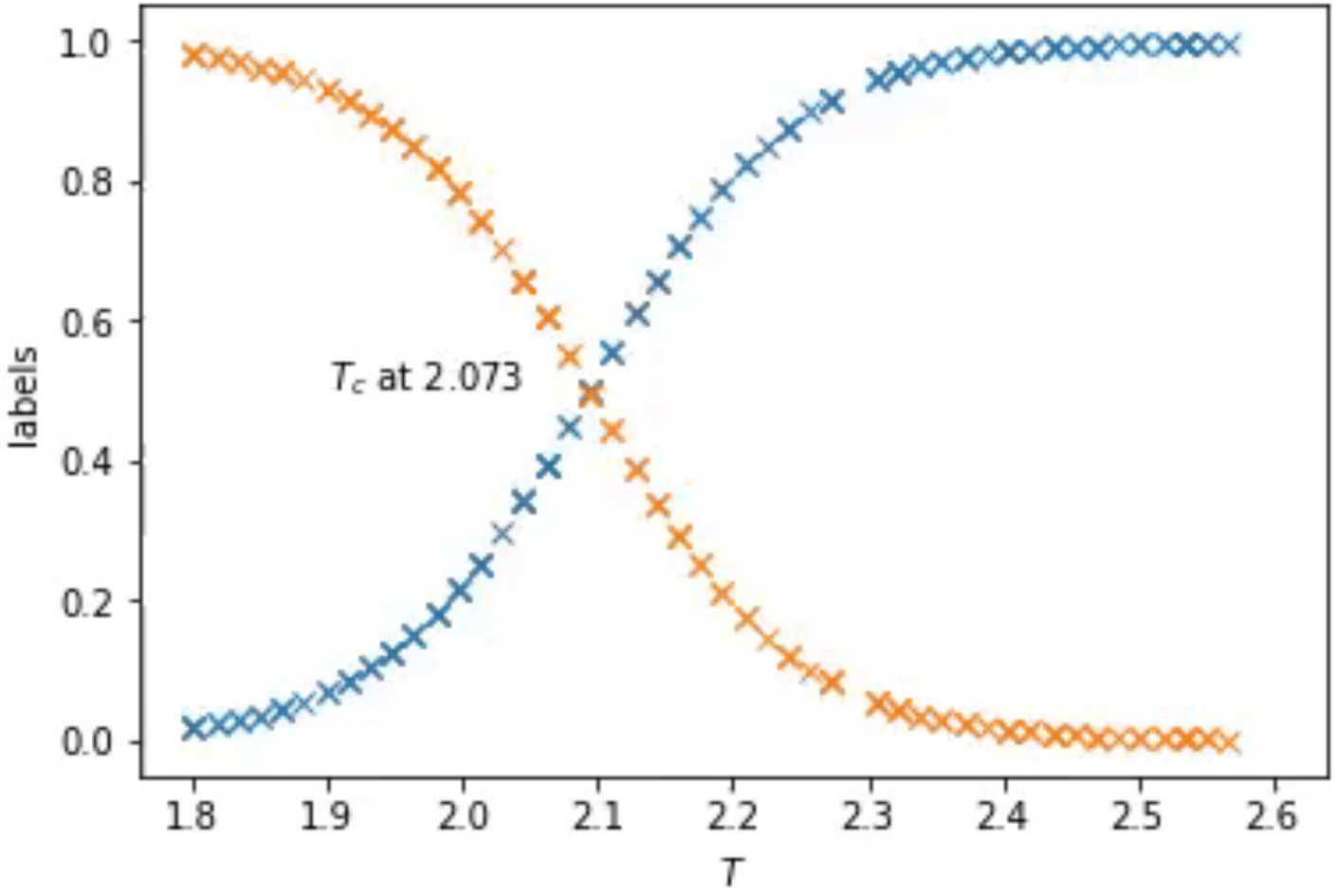
Step 3

$L_{\text{Guesser}} = (\tilde{y} - y)^2$

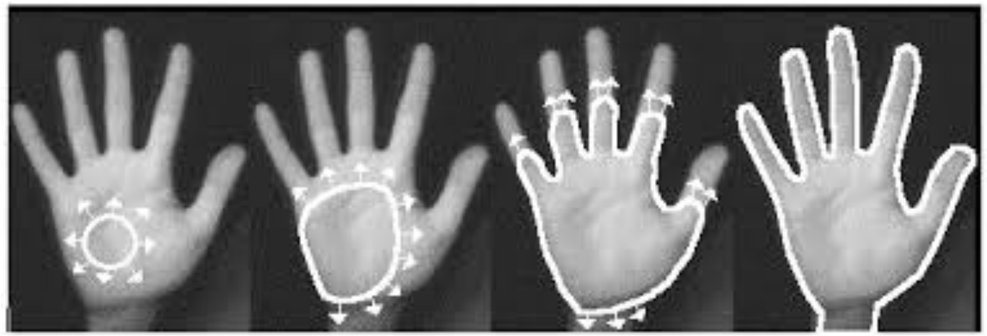


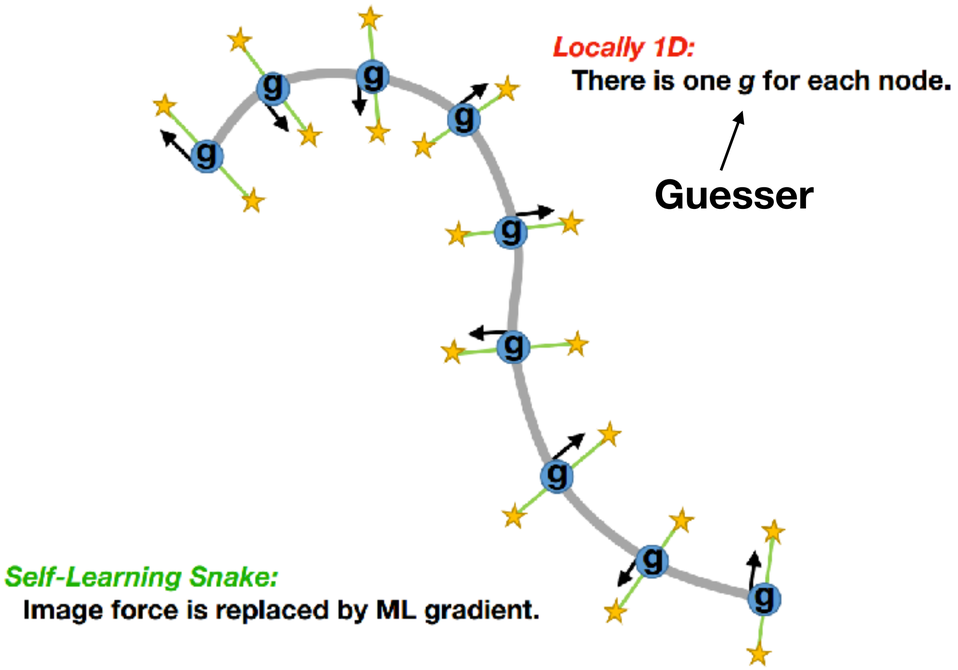
Probe the gradient of the W shape

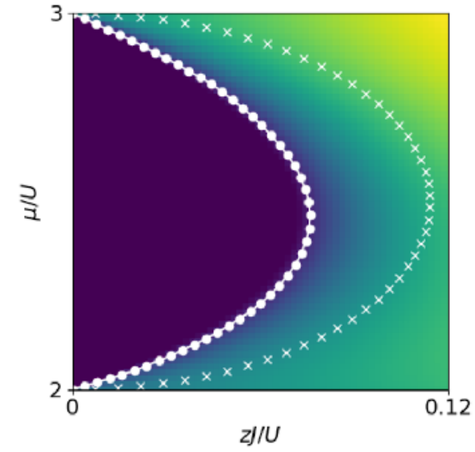
Automatic Confusion

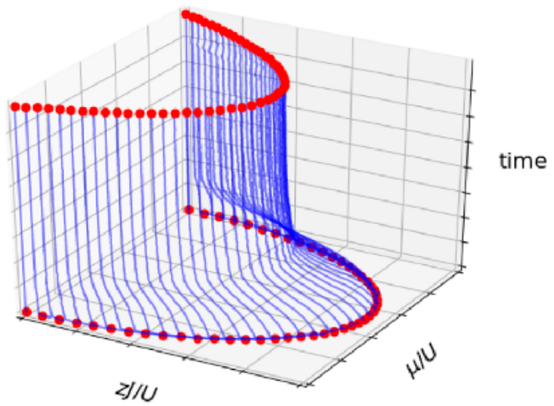


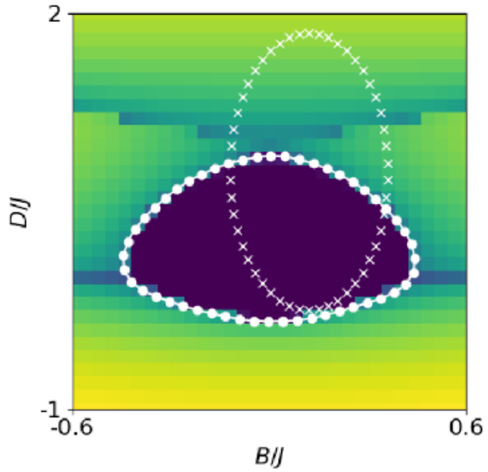
Snakes (from computer vision)

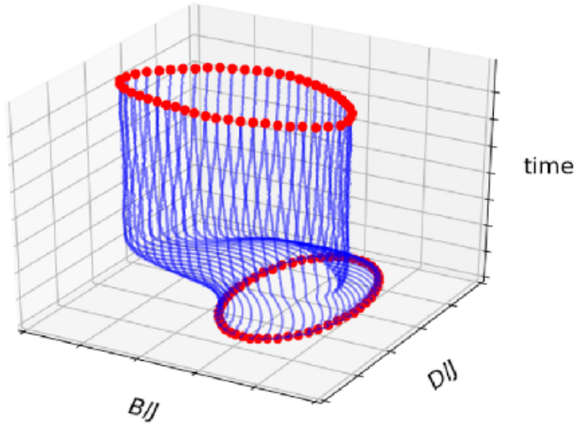












Exploring AI for (quantum) physics

ATTP Autumn School, Module 2

Evert van Nieuwenburg

Chris Chalkias & Tom Daman

Unsupervised learning

$\{(\mathbf{X}, \mathbf{Y})\}$

$\{x_1, y_1\} = \{$, $\}$

$\{x_2, y_2\} = \{$, $\}$

$p(y \mid x)$

Unsupervised learning

Learning about examples

$\{\mathbf{X}\}$

Inputs/Samples ~~**Targets/Labels**~~

$x_1 =$

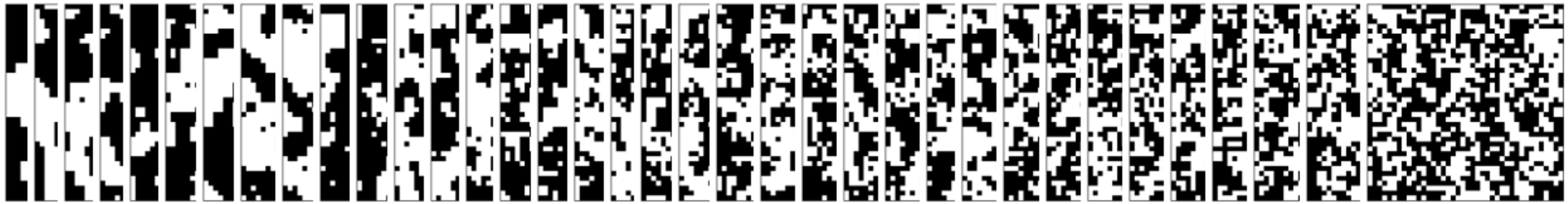
$x_2 =$

e.g. Learn $p(x)$

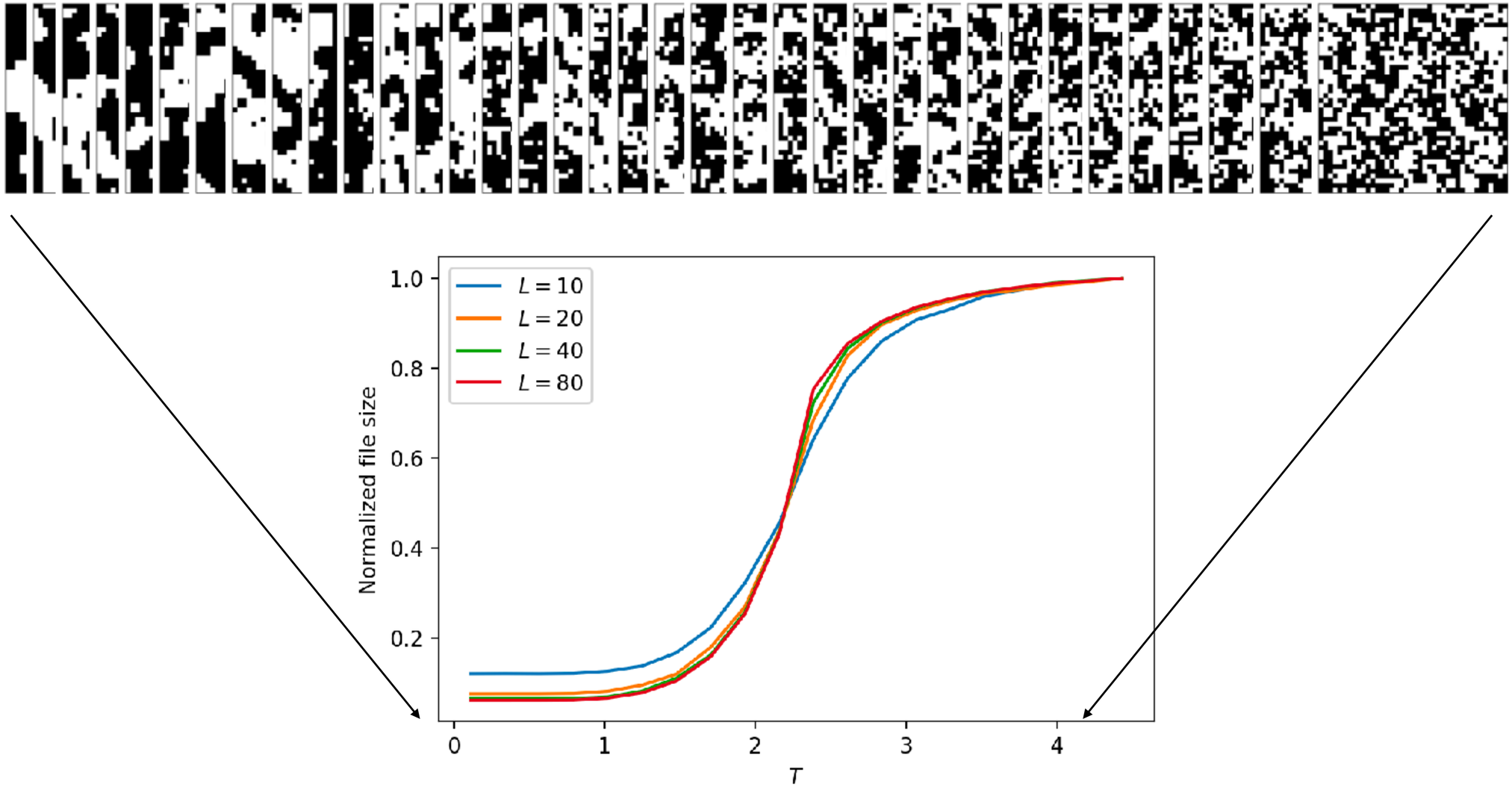
Representation learning

thispersondoesnotexist.com

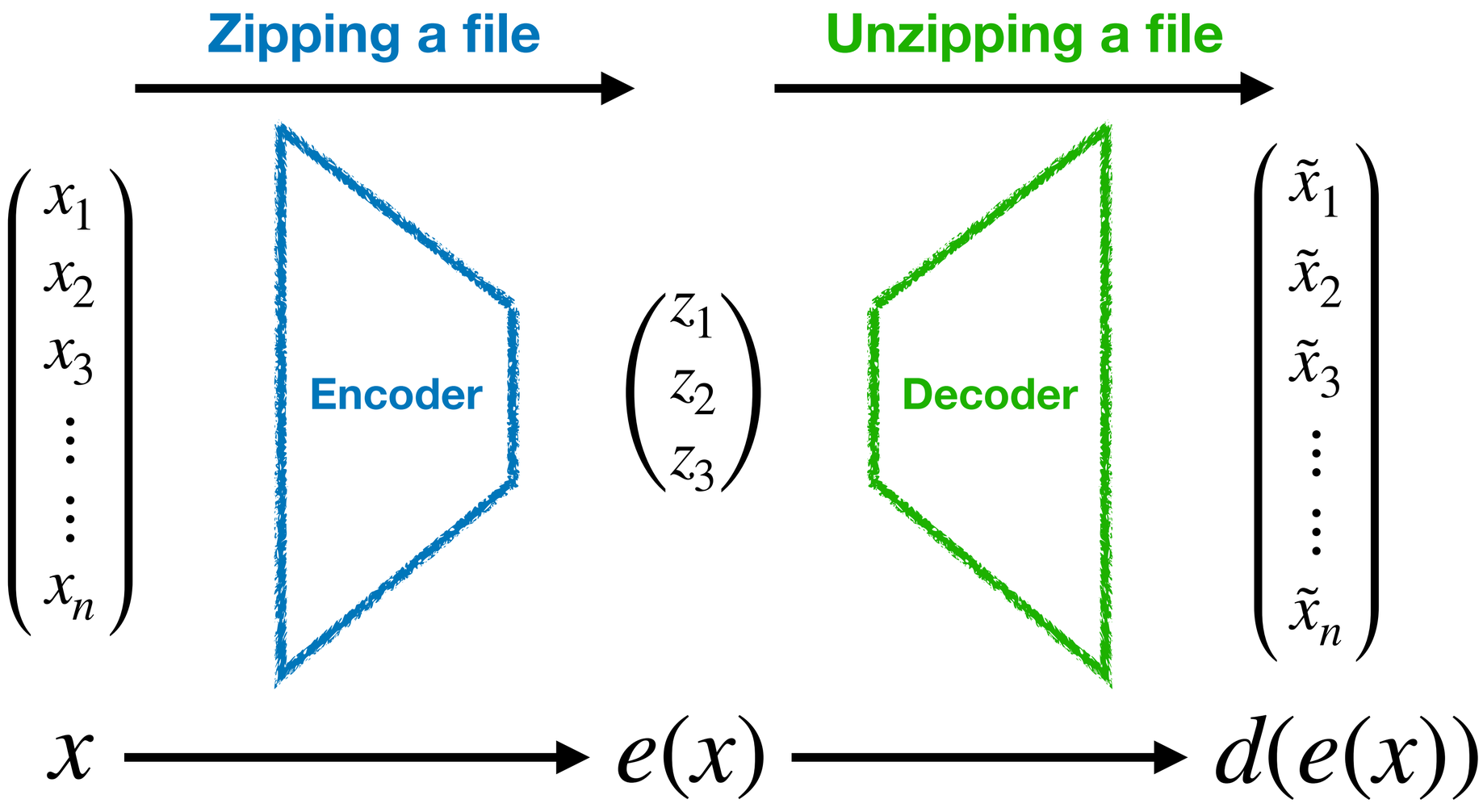
The Ising model dataset



Zipping the Ising model



Intrinsic Dimension and Dimensional Reduction

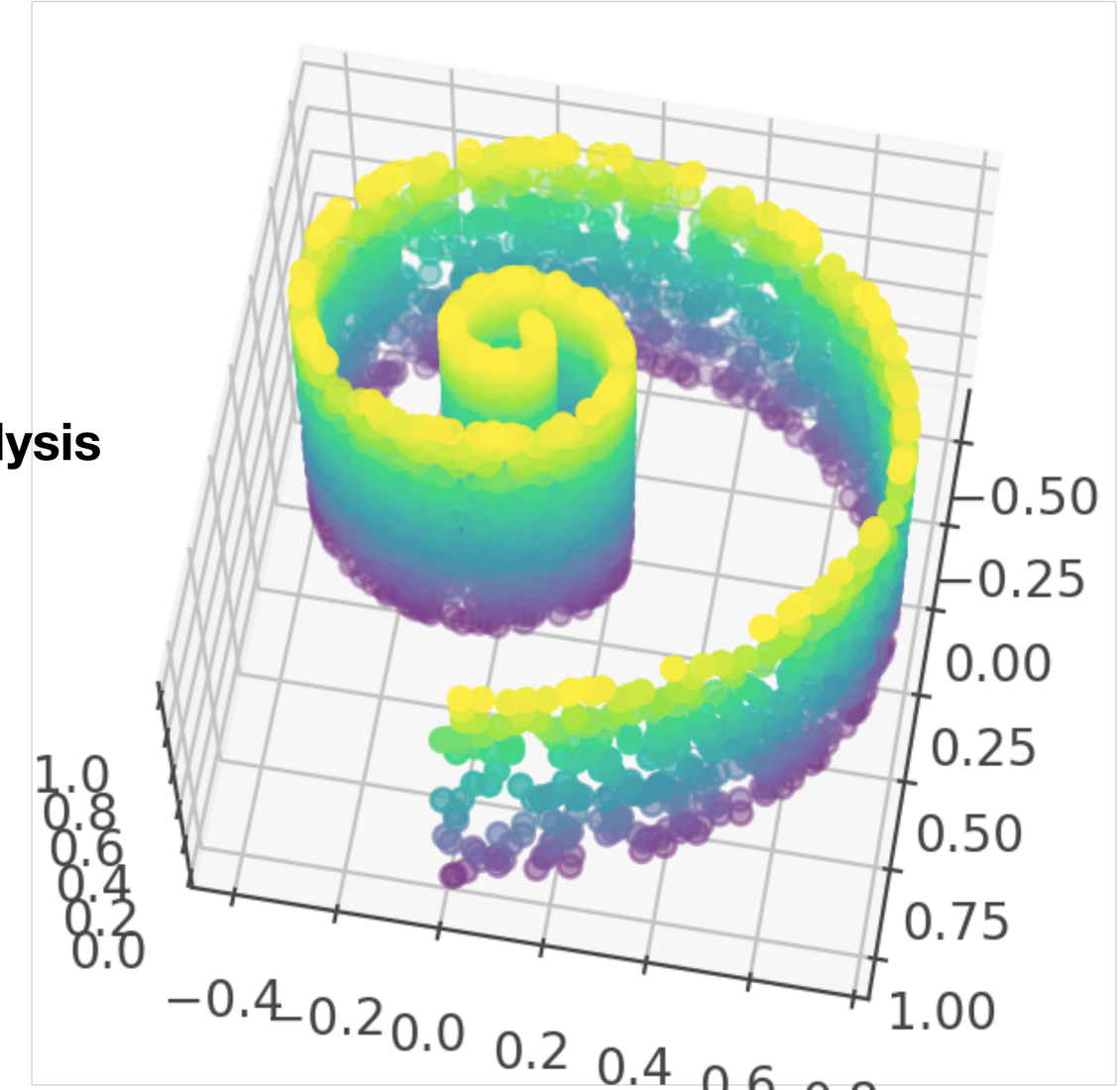


Intrinsic Dimension and Dimensional Reduction

k-Means

Principal Component Analysis

Diffusion maps



Neighbour graphs

2NN

t-SNE

UMAP

AutoEncoder

Variational AutoEncoder

Intrinsic Dimension and Dimensional Reduction

$$x \xrightarrow{\text{Encoding}} e(x) \xrightarrow{\text{Decoding}} d(e(x))$$

Lossless compression

$$x = d(e(x))$$

Lossy compression

$$x \approx d(e(x))$$

Reconstruction loss/error

$$\epsilon(x, d(e(x))) = (x - d(e(x)))^2$$

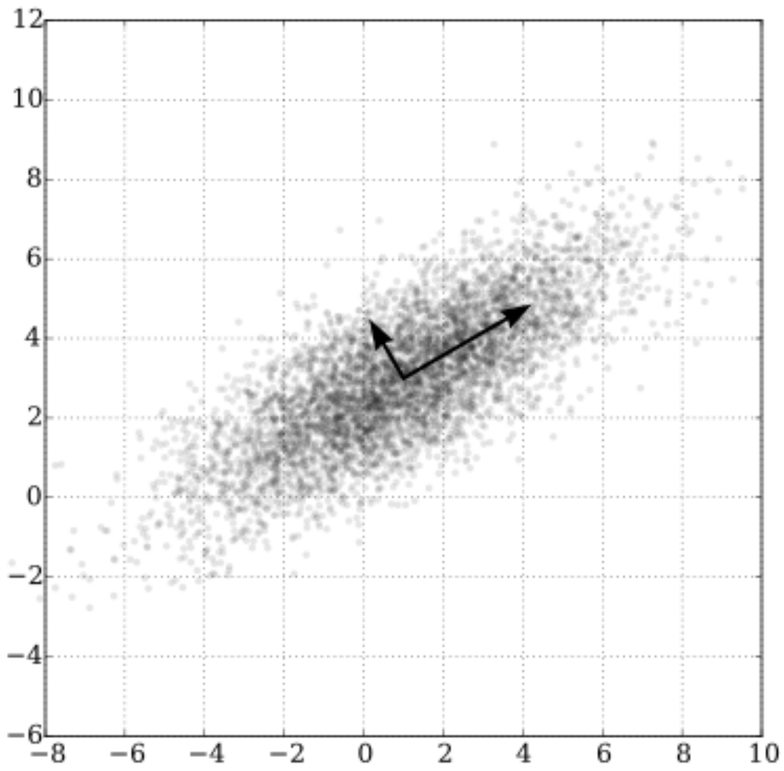
Usually fine

There is a better metric for distributions

PCA'ing the Ising model

Principal Component Analysis

https://arxiv.org/abs/1606.00318



$$P^* = \max_P \sum_i \textbf{var}(P \cdot x_i)$$

$$= \max_P \sum_i (P \cdot x_i)^2$$

$$= \max_P (P \cdot X)^2$$

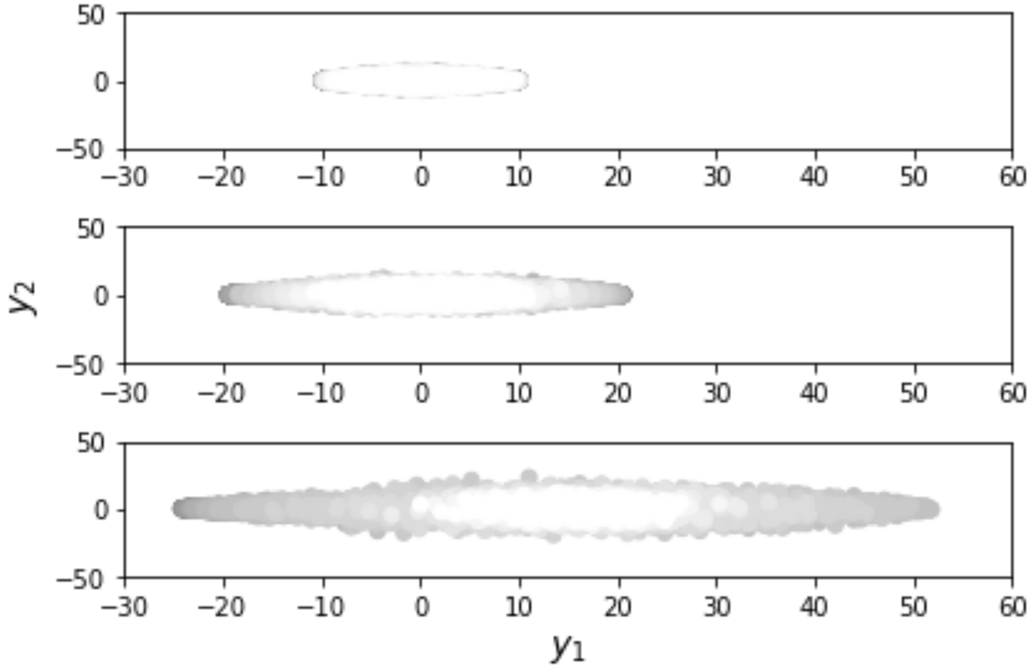
$$= \max_P P^T \cdot X^T \cdot X \cdot P$$

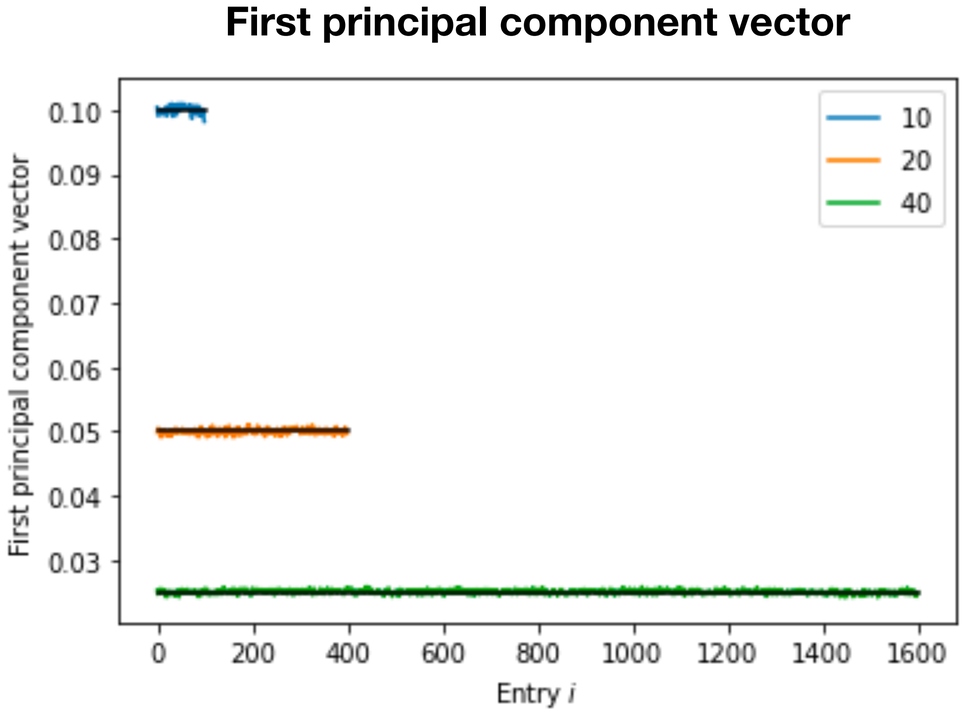
Covariance matrix

$$|P| = 1$$

P^* **given by eigenvector with largest eigenvalue**

PCA'ing the Ising model

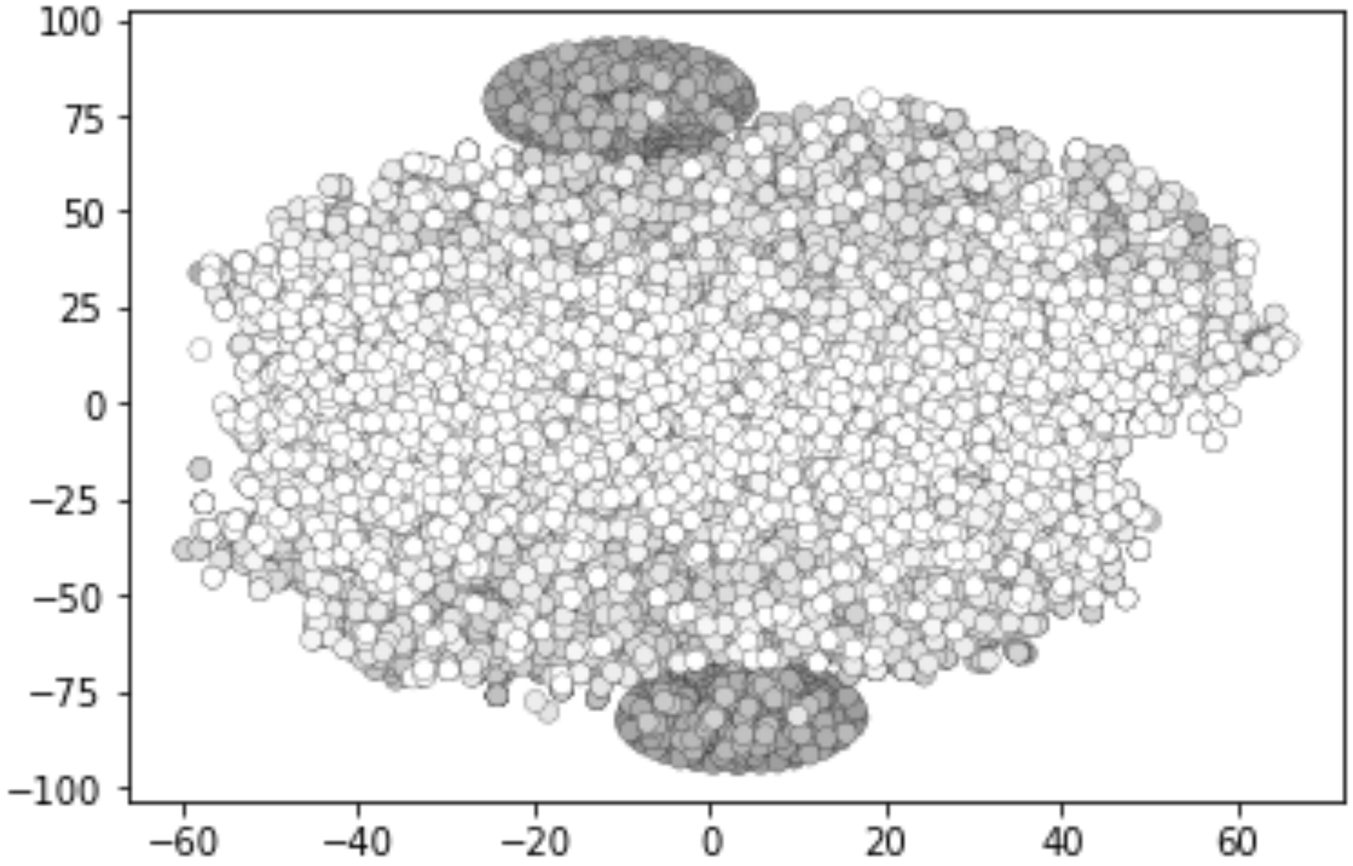




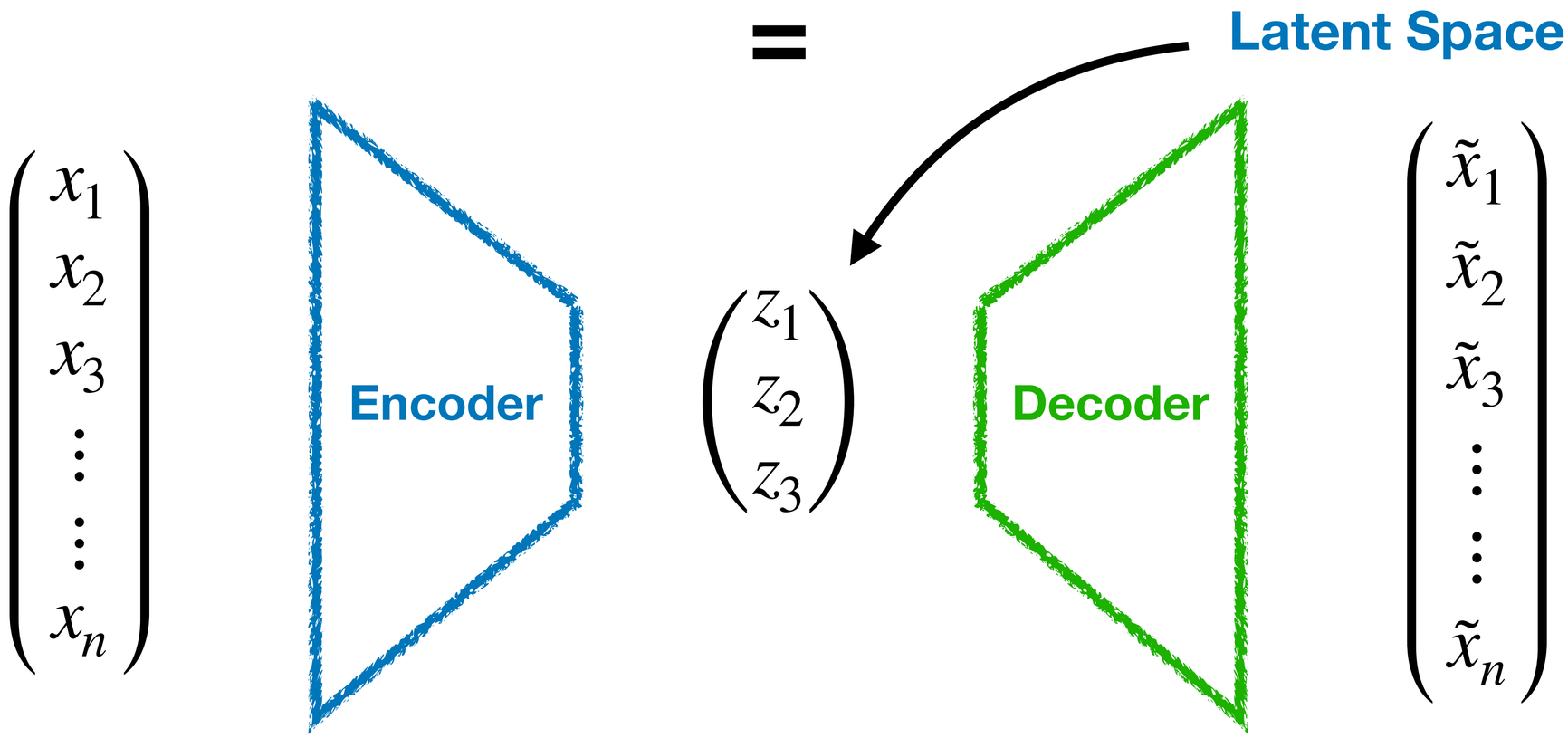
$$x \xrightarrow{\text{Encoding}} e(x) = P \cdot x \xrightarrow{\text{Decoding}} d(e(x)) = P^T \cdot P \cdot x$$

t-SNE'ing the Ising model

https://distill.pub/2016/misread-tsne/



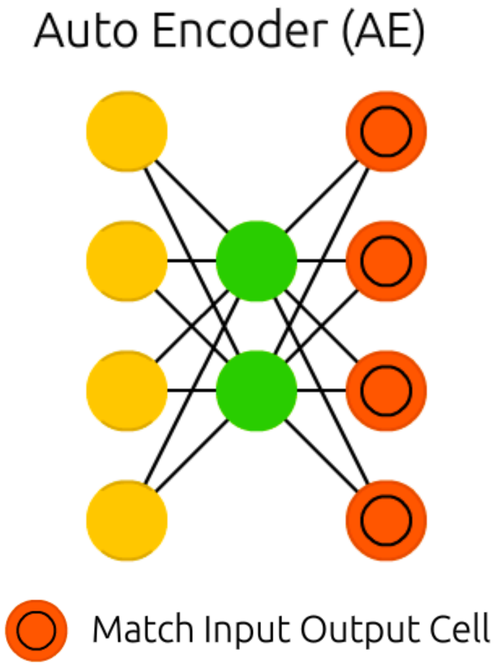
AutoEncoders



Where the encoder and decoder are neural networks

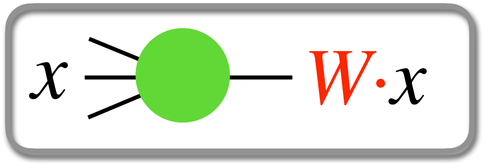
$$\epsilon\Big(x, d(e(x))\Big) = \Big(x - d(e(x))\Big)^2$$

AutoEncoder vs PCA



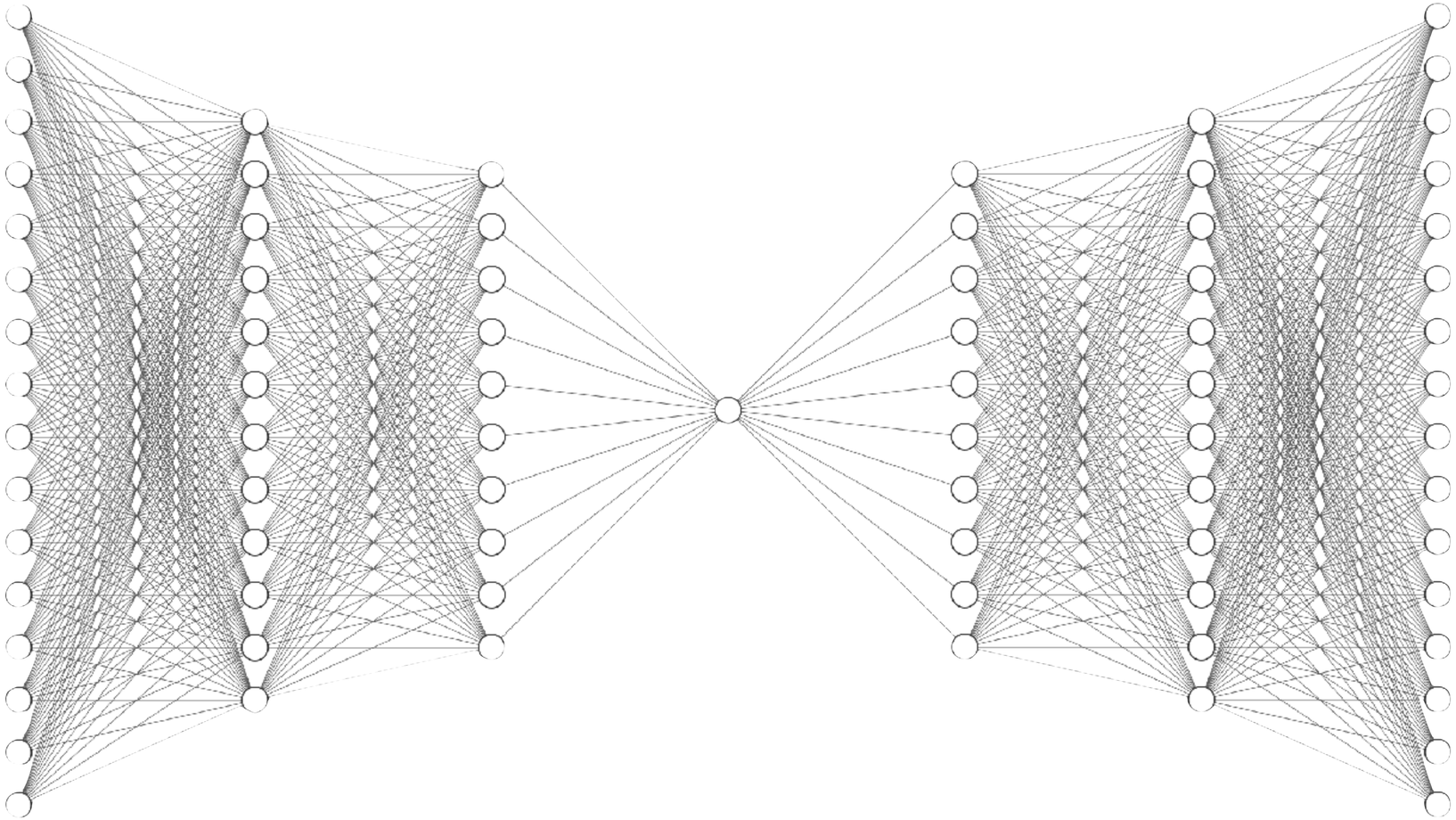
If activation function is linear

A neuron



$$x \xrightarrow{\text{Encoding}} e(x) = \overset{W_1}{P} \cdot x \xrightarrow{\text{Decoding}} d(e(x)) = \overset{W_2 \cdot W_1}{P^T \cdot P} \cdot x$$

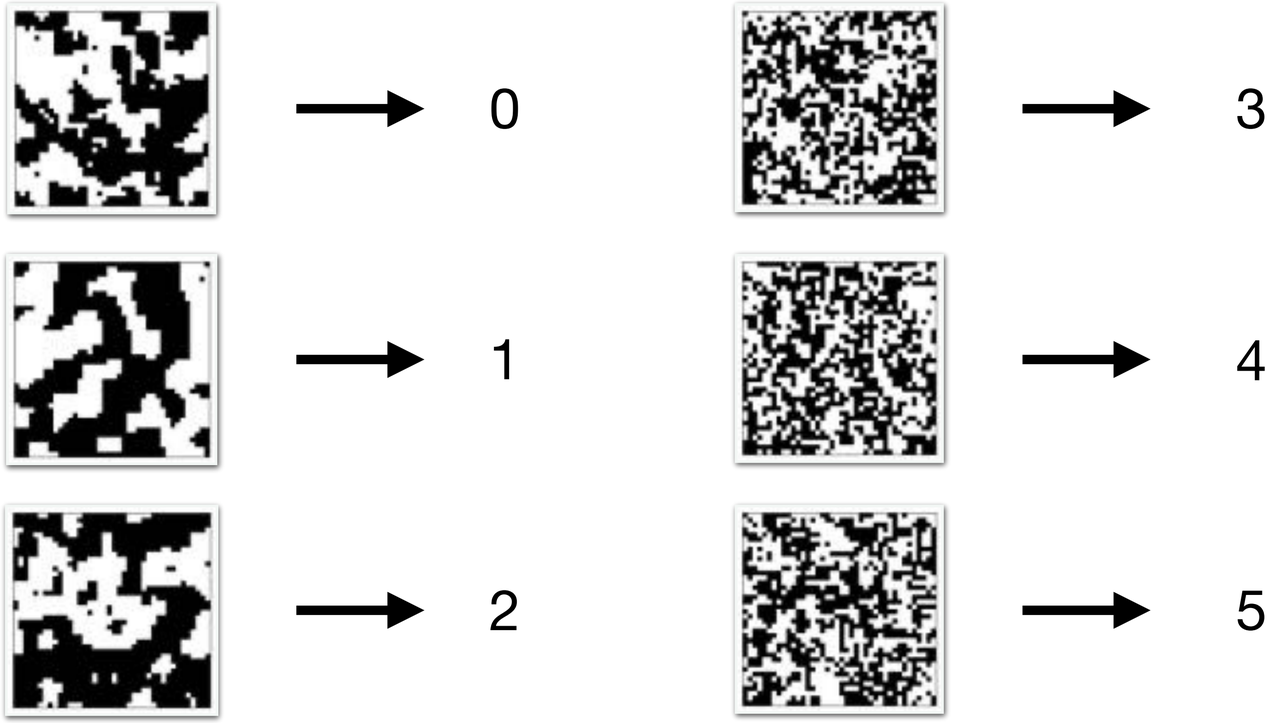
AutoEncoder architecture



If encoder/decoder have enough DoF and non-linearity, anything can be reduced to latent dim = 1

…not very useful (interpretable)

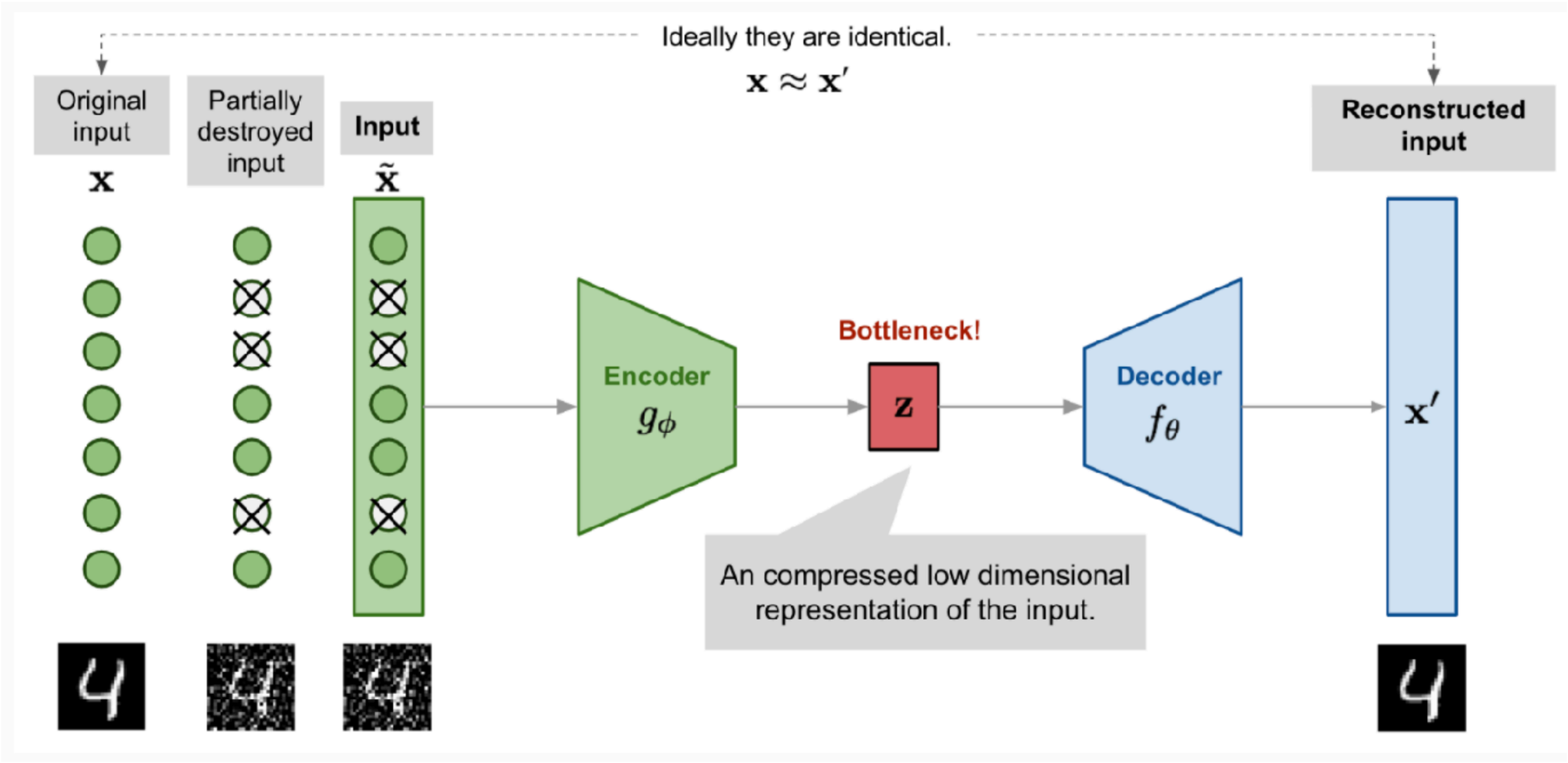
AutoEncoder architecture



A large enough AE could map each unique input to a unique integer, with perfect reconstruction

…but what happens if we input an unseen integer (or non-integer!?) into the decoder?

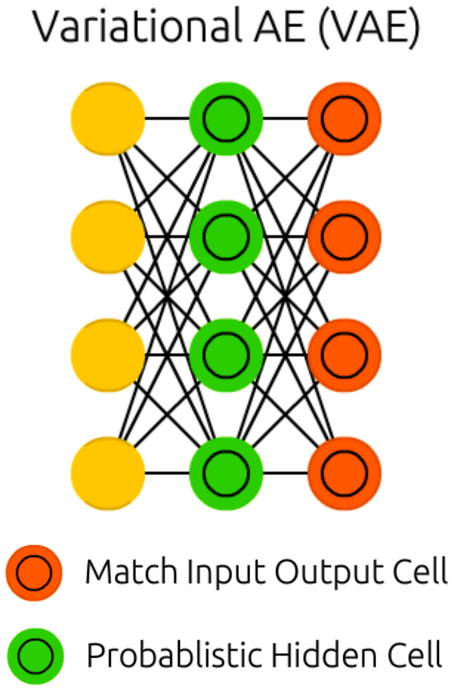
Denoising AE



https://lilianweng.github.io/lil-log/2018/08/12/from-autoencoder-to-beta-vae.html

The Variational AutoEncoder

Instead of optimising the latent space **only** to minimise the reconstruction loss,
we now want the latent variables to 'make sense'



Key point: Encoder maps input to a distribution

$$x \xrightarrow{\textbf{Encoding}} e(x) = p(z \mid x)$$

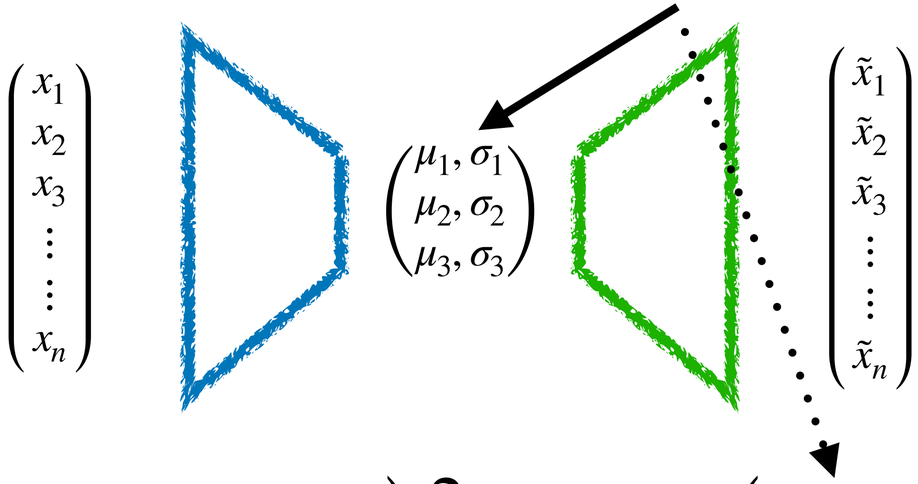
$$z \sim p(z \mid x) \xrightarrow{\textbf{Decoding}} d(e(x))$$

"sample from"

https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

The Variational AutoEncoder

In practise, we assume that p(z|x) is a Gaussian



$$\text{Loss} = \left(x - d(e(x))\right)^2 + \text{KL}\left(N(\mu_i, \sigma_i), N(0,1)\right)$$

"Kullback-Leiber Divergence"
measure of similarity between two distributions

Similar points in latent space are similar after decoding

Why this loss function?

Encoder

$$p(z \mid x)$$

Decoder

$$p(x \mid z) \qquad p(z)$$

Drawn from

$$p(z \mid x) = \frac{p(x \mid z)p(z)}{p(x)}$$

$$= \int p(x \mid z)p(z)\, dz$$

Too costly!

Assume

$$p(z) = N(0,1)$$
$$p(x \mid z) = N(f(z), cI)$$

The Variational AutoEncoder

$$p(z\,|\,x) \leftarrow q_x(z) = N\Big(g(x), h(x)\Big)$$

Neural Networks!

Minimize $\mathbf{KL}\Big(q_x(z), p(z\,|\,x)\Big)$

$$\mathbb{E}_{z\sim q_x}\Big(\log p(x\,|\,z)\Big) - \mathbf{KL}\Big(q_x(z), p(z)\Big)$$

$$\mathbb{E}_{z\sim q_x}\Big(-\frac{(x-f(z))^2}{2c}\Big) - \mathbf{KL}\Big(q_x(z), p(z)\Big)$$

$$p(x\,|\,z) = N(f(z), cI)$$

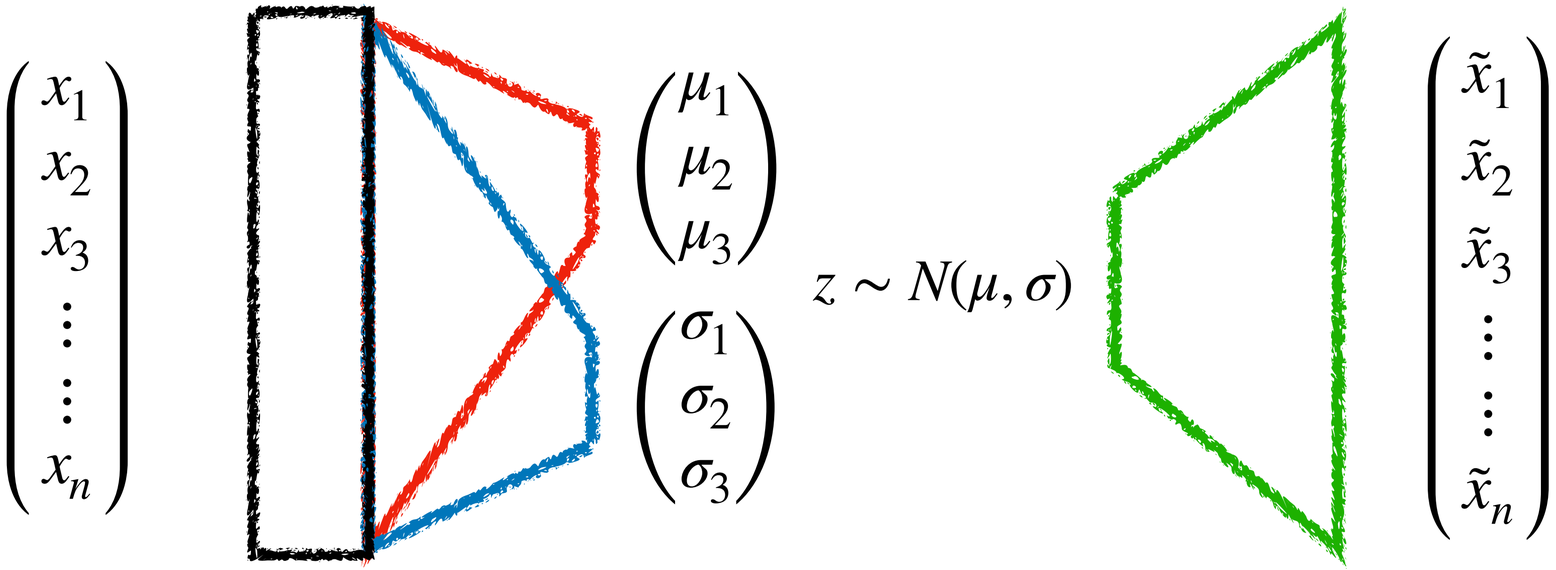
$$\mathbf{KL}(Q,P) = \sum_x Q(x)\ \log\Big(\frac{Q(x)}{P(x)}\Big)$$

$$= \mathbb{E}_{x\sim Q}\Big(\log Q\Big) - \mathbb{E}_{x\sim Q}\Big(\log P\Big)$$

$$\text{Loss} = \Big(x - \mathrm{d}(\mathrm{e}(x))\Big)^2 + \mathrm{KL}\Big(N(\mu_i, \sigma_i), N(0,1)\Big)$$

The Variational AutoEncoder

$$\text{Loss} = \left(x - \text{d}(\text{e}(x))\right)^2 + \text{KL}\left(N(\mu_i, \sigma_i), N(0,1)\right)$$



Reparametrisation trick

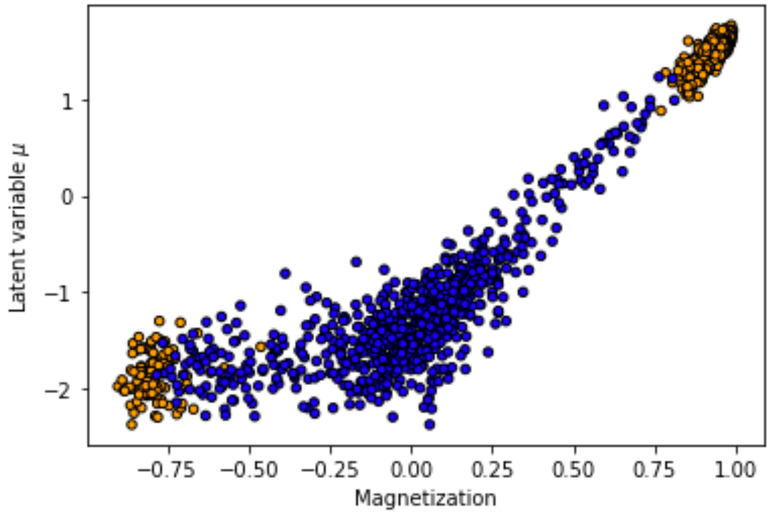
$$z = \mu + \sigma N(0,1)$$

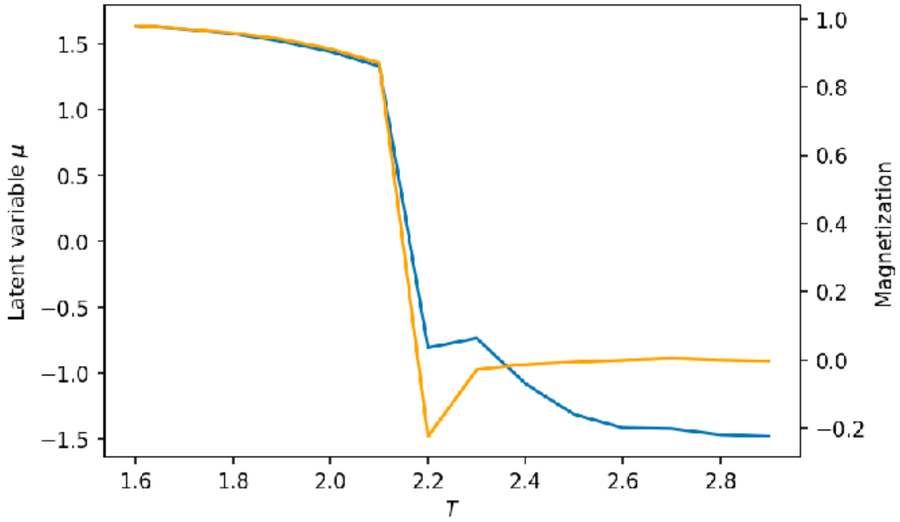
```python
def reparameterize(rng, mean, logvar):
    std = jnp.exp(0.5 * logvar)
    eps = random.normal(rng, logvar.shape)
    return mean + eps * std
```

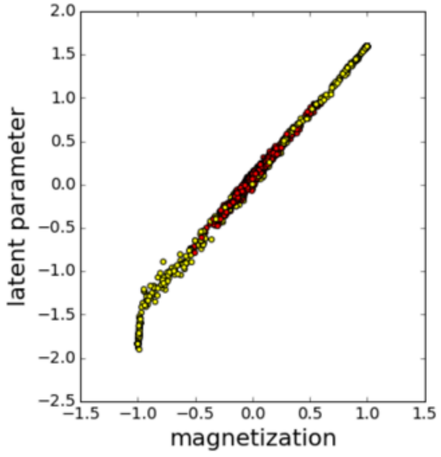
The Variational AutoEncoder

```python
class VAE(nn.Module):
    latent_dim: int = 20

    def setup(self):
        self.encoder = Encoder(self.latent_dim)
        self.decoder = Decoder()

    def __call__(self, x, z_rng):
        mean, logvar = self.encoder(x)
        z = reparameterize(z_rng, mean, logvar)
        recon_x = self.decoder(z)
        return recon_x, mean, logvar
```

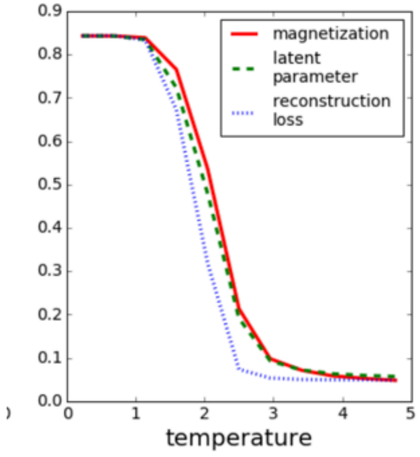
VAE'ing the Ising model







https://arxiv.org/abs/1703.02435



https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

This module

Lecture 1: Basics, (Un)supervised learning

Lecture 2: Reinforcement learning

Lecture 3: Quantum Machine Learning

Lecture 4: Quantum Game Theory

Most importantly: *we will adjust to your preferences as we go*.

Please send us suggestions (Campuswire). If you have an interesting dataset or control problem, we can consider it as an example