Quantum-enhanced Machine Learning
(with near-term devices)

Contents & Literature

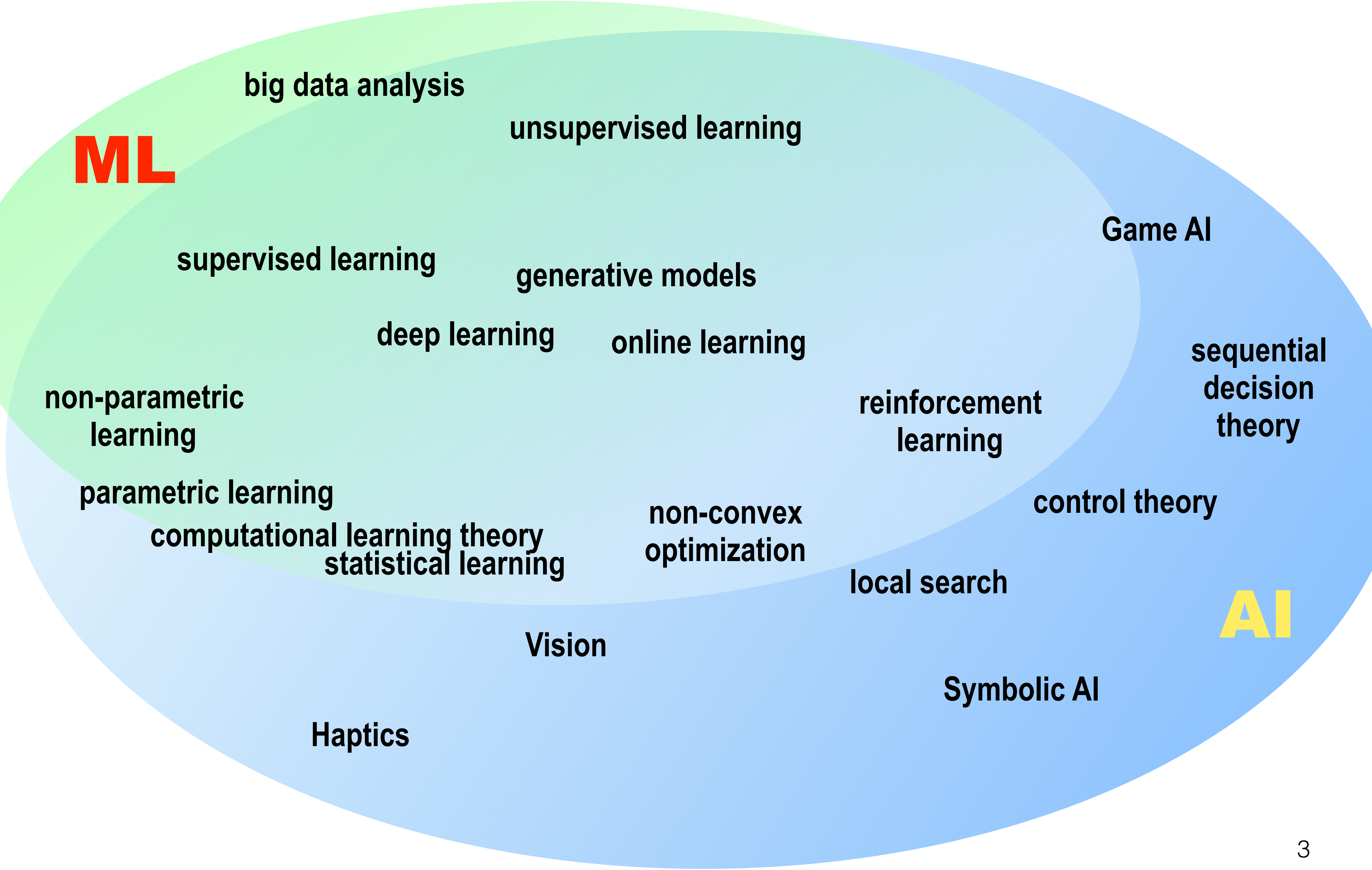
1) *Background 1: machine learning (ML)*
    - *what is ML, and basic ML models*

2) *QC meets ML (big picture) [for more info: arXiv:1709.02779]*

3) *ML and parametrized circuits [for more info: arXiv:1906.07682]*

4) *QeML with quantum feature spaces [based on: arXiv:1804.11326]*
    - *Support vector machines*
    - *Explicit and implicit quantum-embedded SVMs*

Machine learning and AI

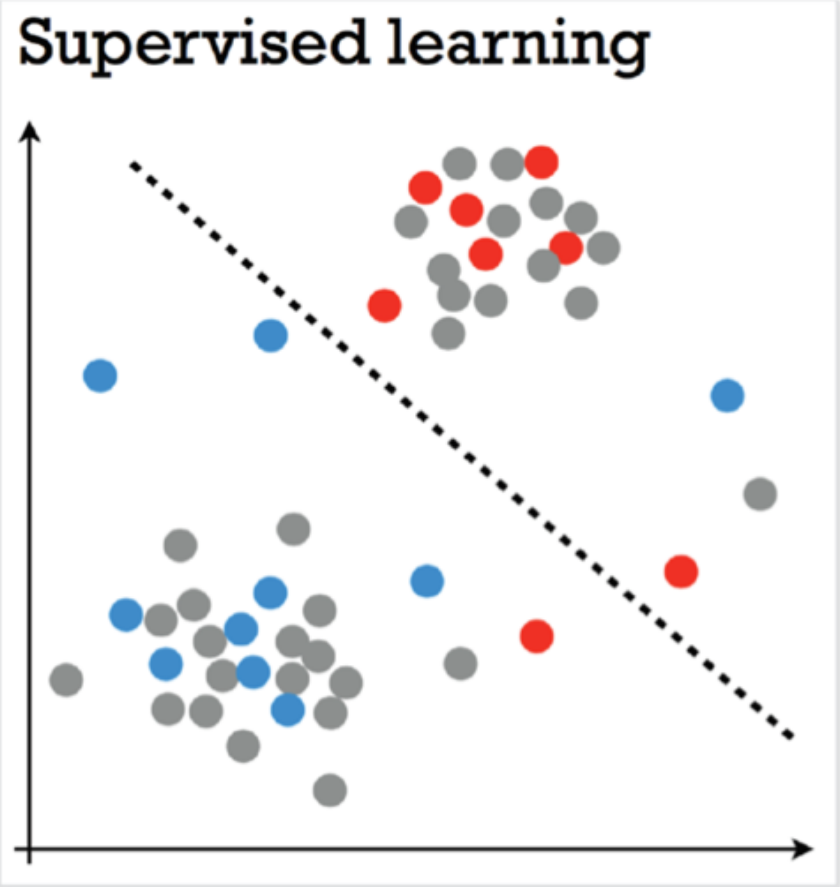


Three main (cannonical) modes of ML:

- Supervised learning
- Unsupervised learning
- Reinforcement learning

- forest of in-between modes; semi-supervised, active, transductive, on-line…

Supervised learning: the ***what*** (is the objective)



,

?

Supervised learning: the ***what*** (is the objective)

Data (feature vectors) & Labels:

$$\mathbf{x} \in S \subseteq \mathbb{R}^n; \;\; y \in Labels$$

Label function:

$$f : S \rightarrow Labels$$

Dataset, "training examples"

$$D = \{(\mathbf{x}_i, y_i) \,|\, \mathbf{x}_i \in S; y_i = f(\mathbf{x})\}$$

-need to correctly label unlabeled data

Given D, output a good guess for f.

-*classification (categorical or discrete label)* v.s. *regression (contiuous label)*

-*classification, prediction, regression….*

Supervised learning: the ***what*** (is the objective)

More generally (probabilistic)

BTW: Distributions generalize functions

Data (feature vectors) & Labels:

$$\mathbf{x} \in S \subseteq \mathbb{R}^n; \;\; y \in Labels$$

Label function:

$$P(\mathbf{x}, y)$$

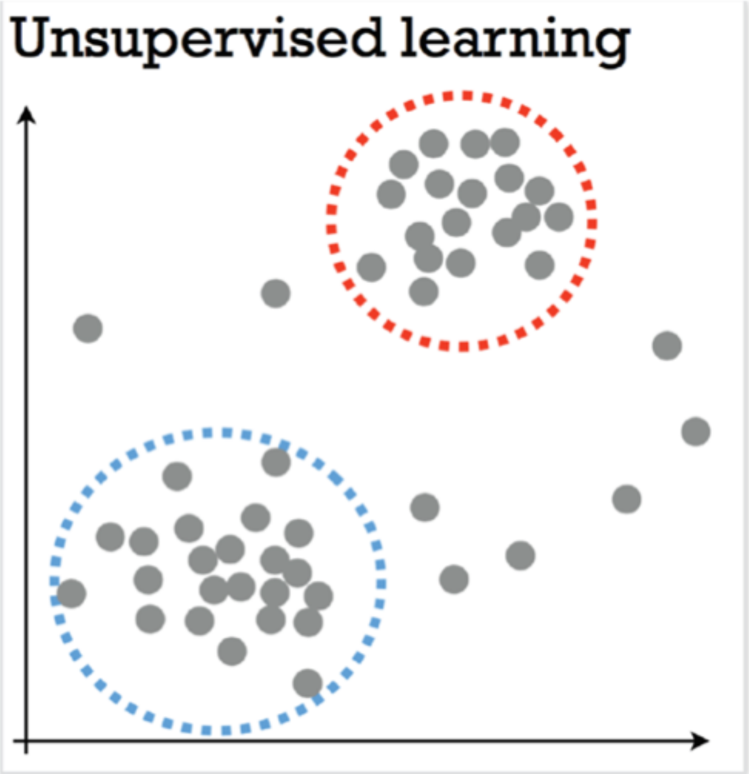
Dataset, “training examples”

$$D \sim P^{\times |D|}$$

Given D, output a good guess for $P(y \,|\, \mathbf{x})$

Learning about <u>data-label relationships</u> in a bivariate distribution <u>from samples</u>

Unsupervised learning: the ***what*** (is the objective)



?

data: $\mathbf{x} \in S \subseteq \mathbb{R}^n$

"world": $P(\mathbf{x})$

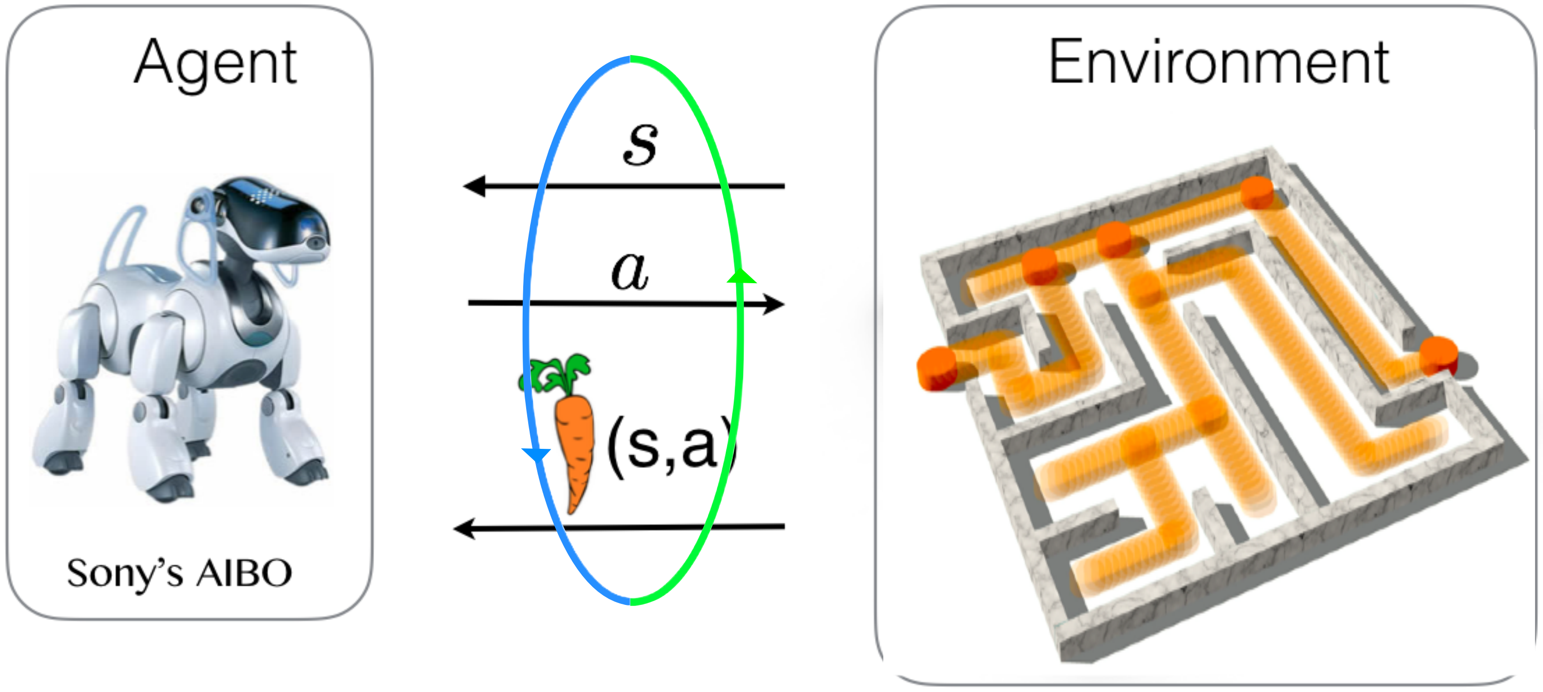
training: $D \sim P^{\times|D|}$

-discriminative (clustering) , "labeling w/o examples"

-generative (make more cats):
approximate sampling from P given D

Learning about (all) features in a distribution from samples

Reinforcement learning: the ***what*** (is the objective)



$$\mathcal{S} = \{s_1, s_2, \ldots\}$$
$$\mathcal{A} = \{a_1, a_2, \ldots\}$$

Closer to AI.
There is a body.
Interaction.
Learning.

$$\pi(a|s)$$

$$T(s|s', a)$$

Learning correct behaviour (policies) by trial-and-error

(incl. data generation online). E.g. AlphaGo.

Supervised learning: the ***how*** (is it achieved)

Recall: need to "guess" $f : S \subseteq \mathbb{R}^n \rightarrow Labels$ from $D = \{(\mathbf{x}_i, y_i = f(\mathbf{x}_i))\}$

- Hypothesis family: $\{f^\theta \mid f^\theta : S \subseteq \mathbb{R}^n \rightarrow Labels, \theta\}$ (c.f. "model/model family")

- Learning = training ≈ fitting:

  $$argmin_\theta \; Error\_on\_D(f^\theta) + R(f^\theta)$$

  R = *regularization term*

- "Loss", "empirical risk", "accuracy", e.g. $\sum_{(\mathbf{x},y)\in D} |f^\theta(\mathbf{x}) - y|^2$

- Generalization performance: (no overfitting, Occam's razor)

Supervised learning: the ***how*** (is it achieved)

Recall: need to “guess” $f : S \subseteq \mathbb{R}^n \rightarrow Labels$ from $D = \{(\mathbf{x}_i, y_i = f(\mathbf{x}_i))\}$

- Hypothesis family: $\{f^\theta \,|\, f^\theta : S \subseteq \mathbb{R}^n \rightarrow Labels, \theta\}$ (c.f. “model/model family”)

- Learning = training ≈ fitting:

$$argmin_\theta \; Error\_on\_D(f^\theta) + R(f^\theta)$$

R = *regularization term*

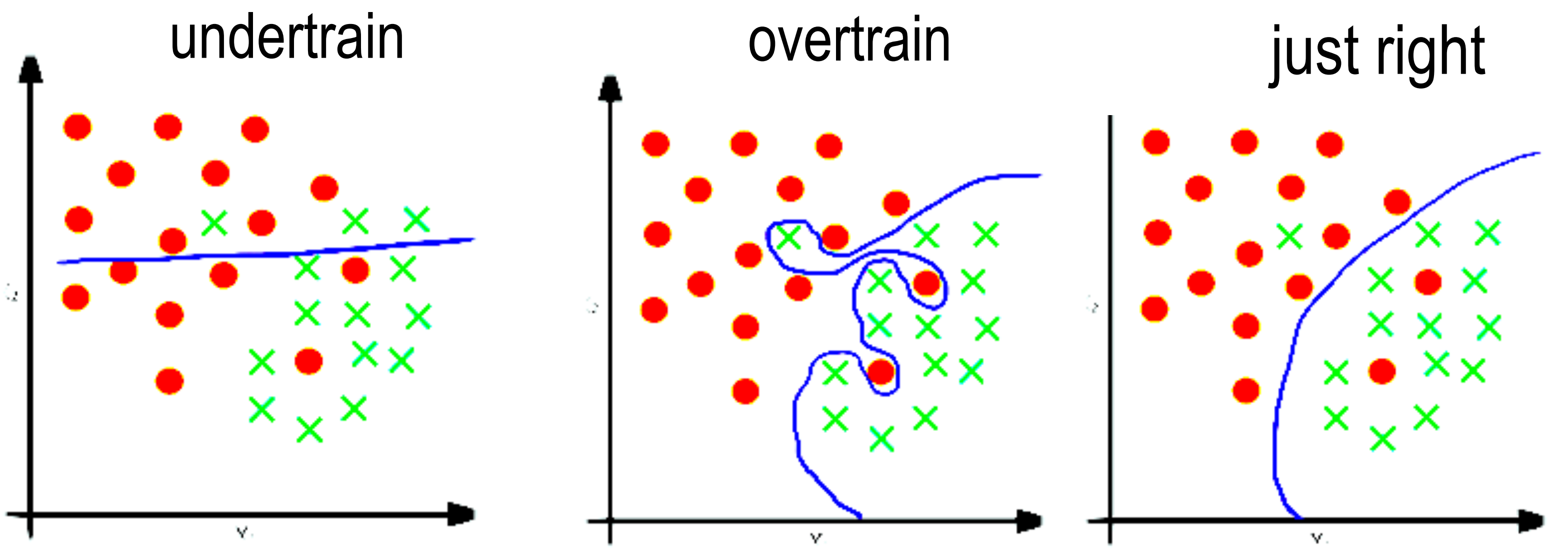
the same elements will be present for unsupervised learning

- “Loss”, “empirical risk”, “accuracy”, e.g. $\sum_{(\mathbf{x},y)\in D} |f^\theta(\mathbf{x}) - y|^2$

- Generalization performance: (no overfitting, Occam’s razor)

Regularization: controling “model complexity” to ensure good generalization

Classification

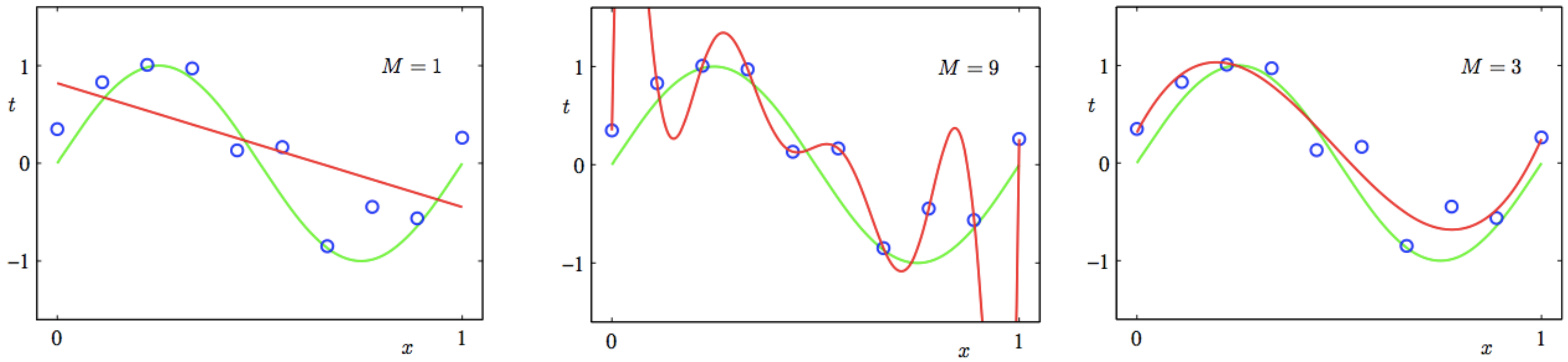


undertrain

overtrain

just right

Regression



underfit

overfit

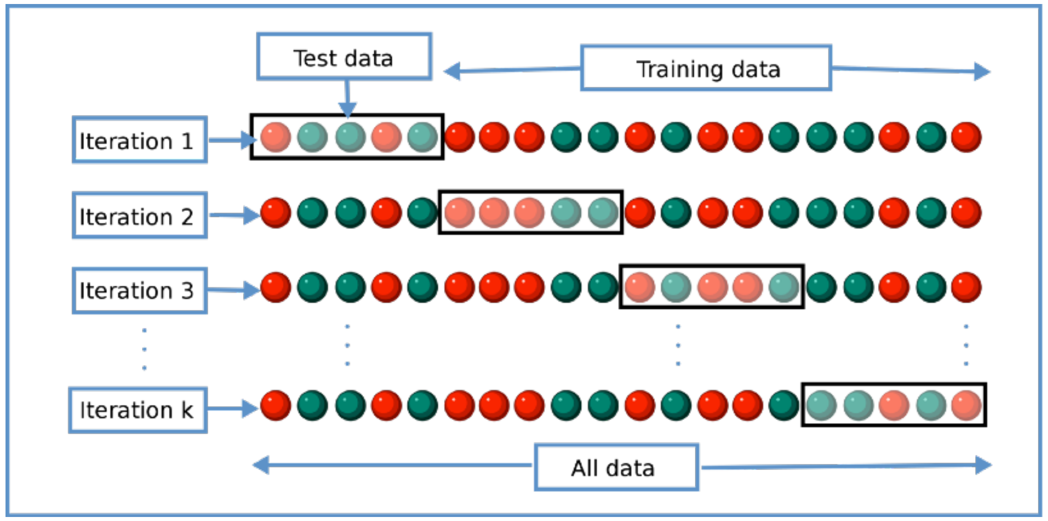
just right

Machine learning is all about generalization performance, that is **performance beyond the training set.**

It is not “just” a best fit problem.

Theory approaches: VC theory, Rademacher complexity…

In practice: cross-validation



https://en.wikipedia.org/wiki/Cross-validation_(statistics)

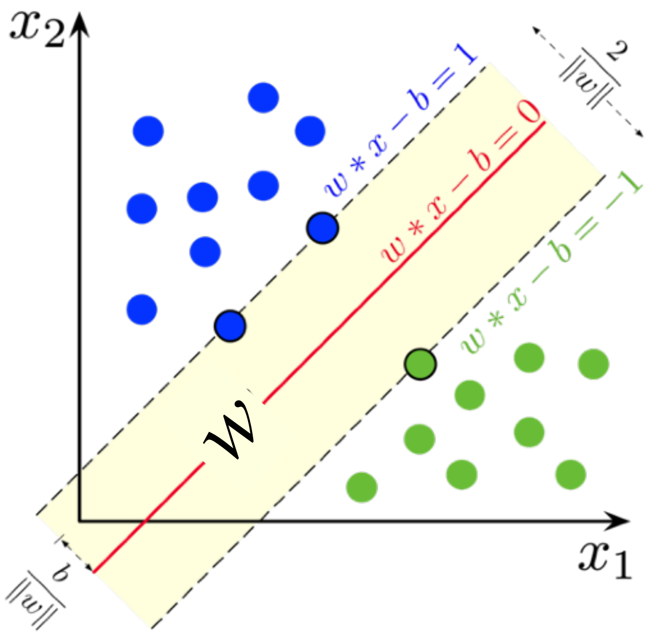
Supervised learning: the ***how*** (is it achieved); examples

- support vector machines (SVM)
- neural networks

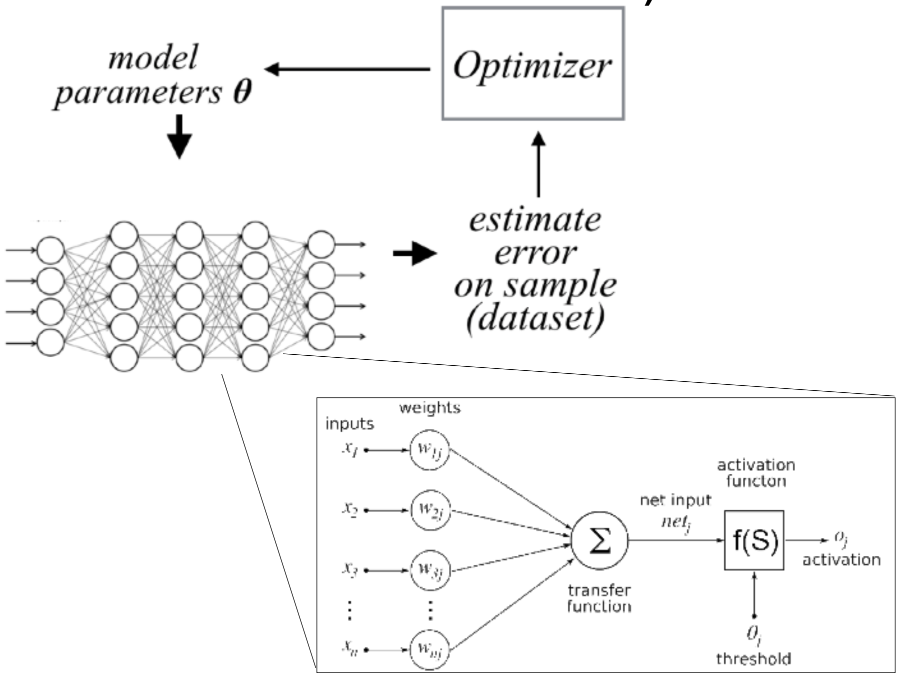
- k-nearest neighbours [classification]
- decision trees [classification]
- naïve Bayes
- (linear) regression [regression]
- Gaussian process regression

Supervised learning: the ***how*** (is it achieved); examples

Support Vector Machines (SVMs)



Neural networks
(not just SL; many types)



"which half-space" ∀ hyperplanes $sign(\mathbf{w}^t\,.\,\mathbf{x}+b)$ + important trick	hypothesis family	specified by n.n. (linear+nonlinear layers)
nomal vector+offset	model parameters	weights + offsets
usually Lagrangian multipler method	training/ optimization	usually backpropagation (chain-rule based (stochastic) gradient descent)
	regularization?	

When discussing QML, keep an eye on

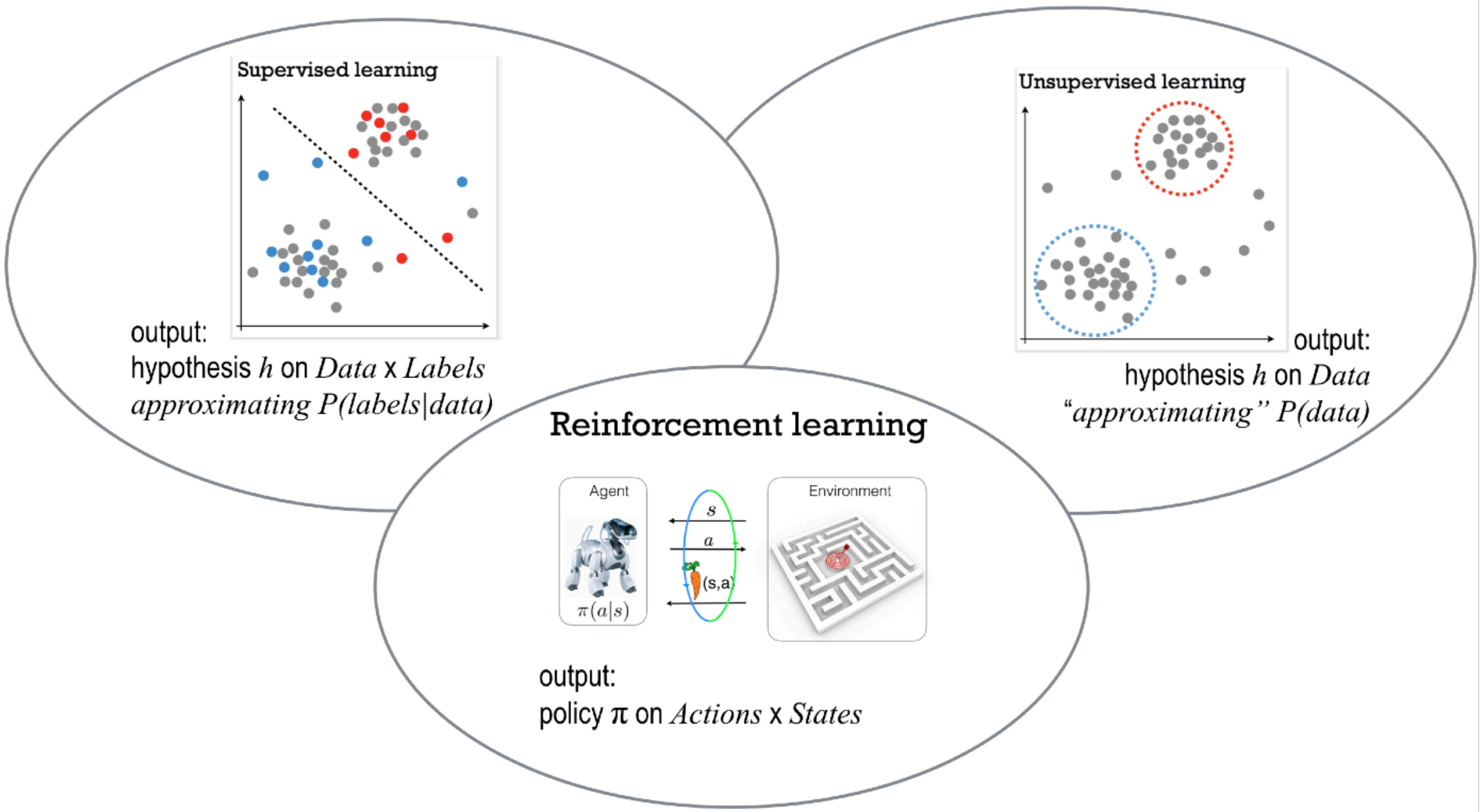
- the **What**
  (what is the objective/goal)
- the **How**
  (how is it done: algorithm; does is achieve the goal)
- the **Why**
  (why do it on a QC; what is the expected advantage/other motivation)

actually, same questions apply to much of *classical* ML approaches

the why is tricky tho; makes a good model model is though

Big picture take home:

the learning/training is optimization: $\mathrm{argmin}_\theta\ \mathrm{Err\_training\_set}(\theta) + \mathrm{Reg}(\theta)$



Supervised learning

output:
hypothesis h on *Data* x *Labels*
approximating P(labels|data)

Unsupervised learning

output:
hypothesis h on *Data*
"approximating" P(data)

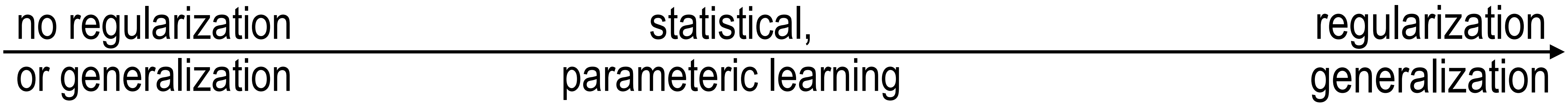
Reinforcement learning

output:
policy π on *Actions* x *States*

but machine learning is more; which model; how it generalizes; good choices…

A connection…
variational methods in physics.. incl VQE are very similar

Var. Q chem	ML
"Ansatz"	model family/hypothesis family
loss: energy	loss: training set error+regul.
explicit, error free ground truth	implicit ground truth, errors
optimization	learning/training



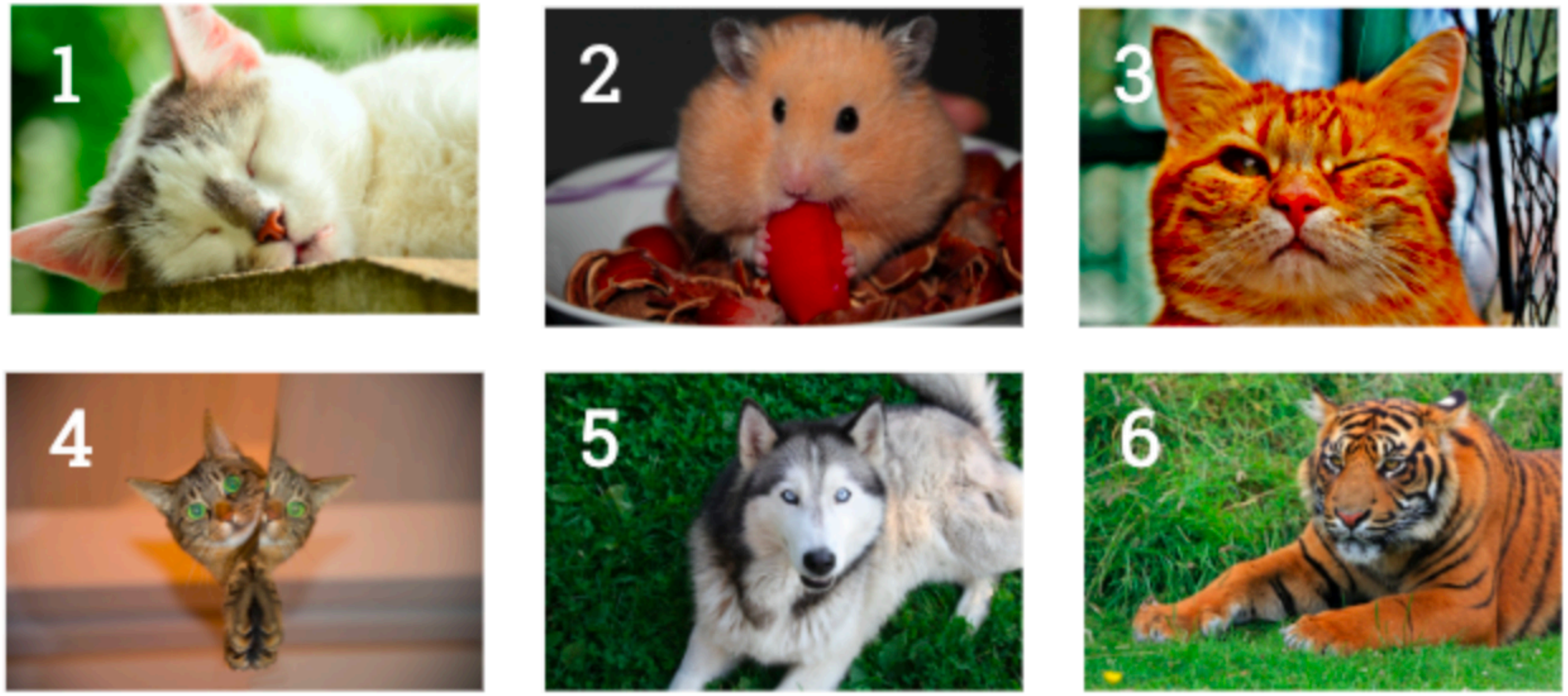
no regularization
or generalization

statistical,
parameteric learning

regularization
generalization

-ground truth…

Cat v.s. no-cat example



ground truth & “objective is subjective”

https://towardsdatascience.com/in-ai-the-objective-is-subjective-4614795d179b

QC meets ML: big picture ideas

-QC and the optimization bottleneck

training *is optimization* and can be hard (NP-hard) → quantum optimization

-QC and the high dimension bottleneck

much of ML is linear algebra; quantum computing is good at that, under conditions

-QC and the hard model bottleneck

topic of the this and next lecture

Supervised Machine learning with Parameterized Quantum Circuits

What: supervised learning for classification

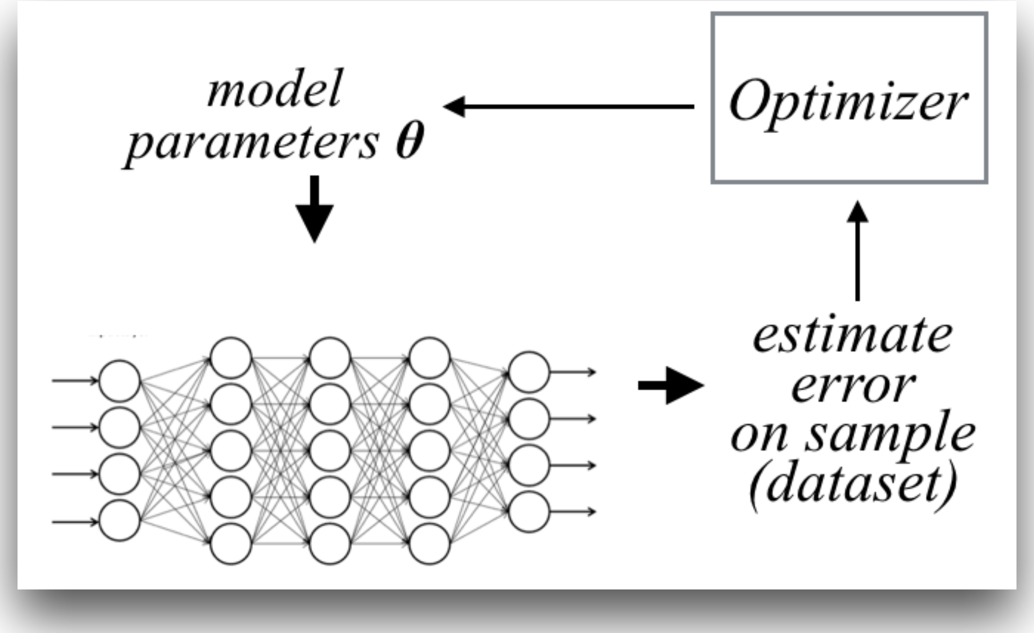
Using quantum computing… but not for optimization needs

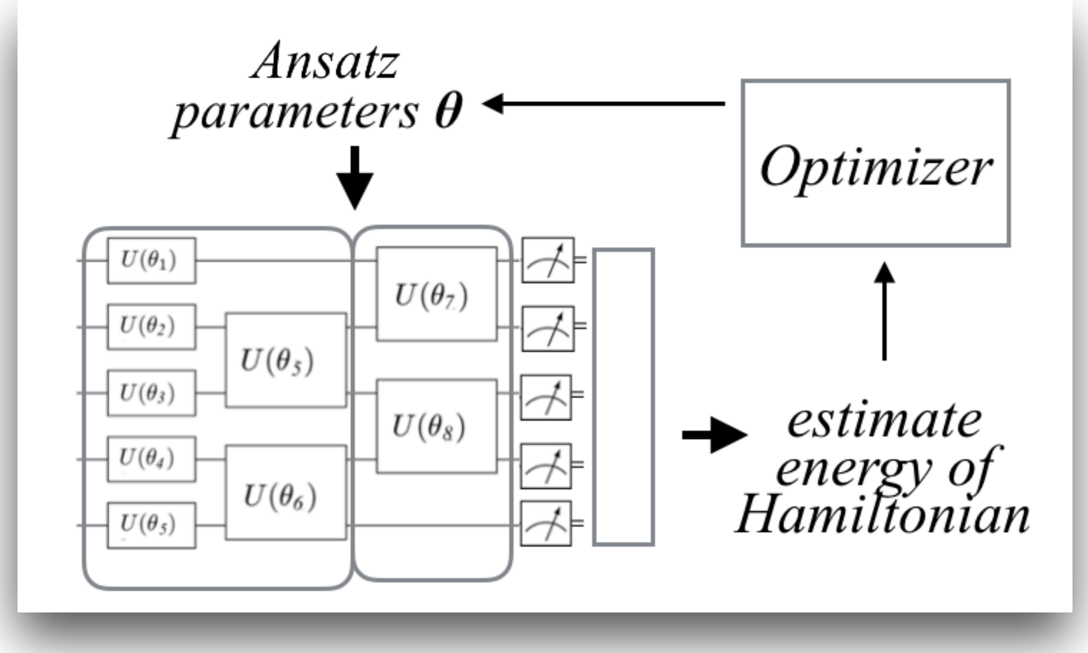
Why? TBD

Supervised Machine learning with Parameterized Quantum Circuits

neural networks

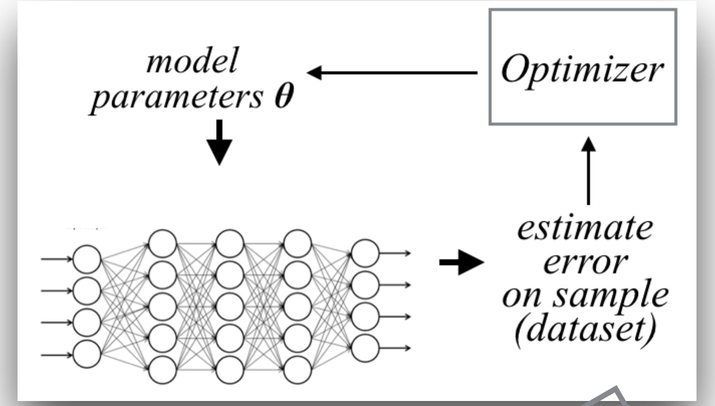
VQE



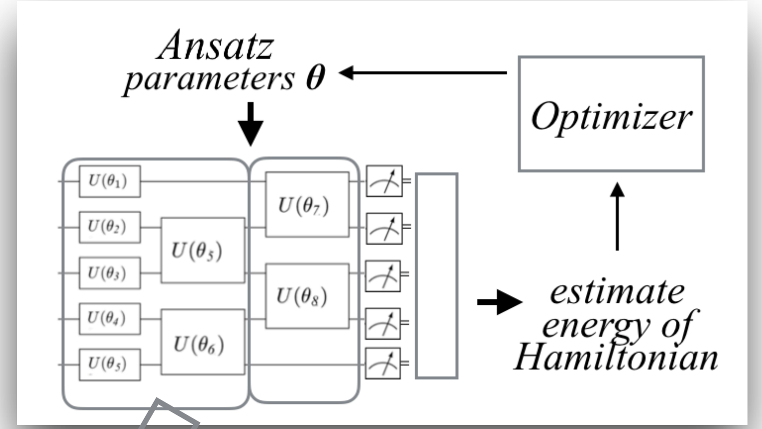


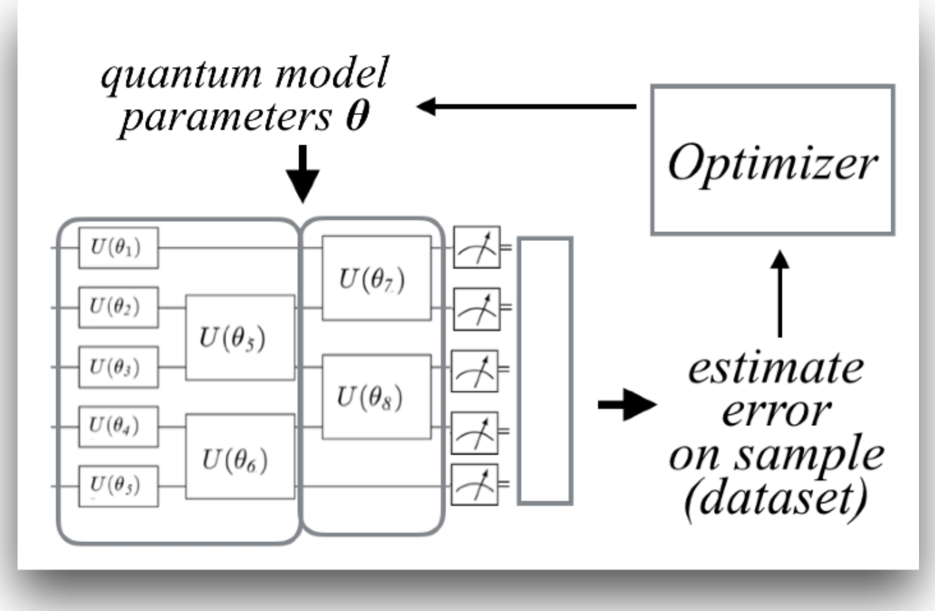
Machine learning with Parameterized Quantum Circuits

neural networks



VQE





PQC-based ML

1) can we train it?

2) does it work?

3) does it do anything interesting?
why do this?

Same Q’s for VQE, but there 3) is clear. Here it is not.

Motivations: cannot do it classically? Curiosity driven?

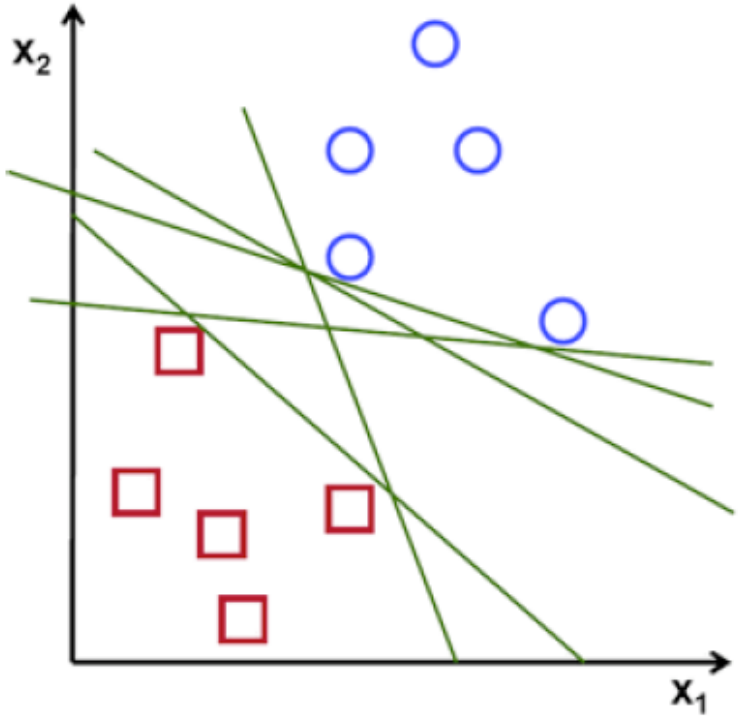
We don’t really understand the model…

Next:

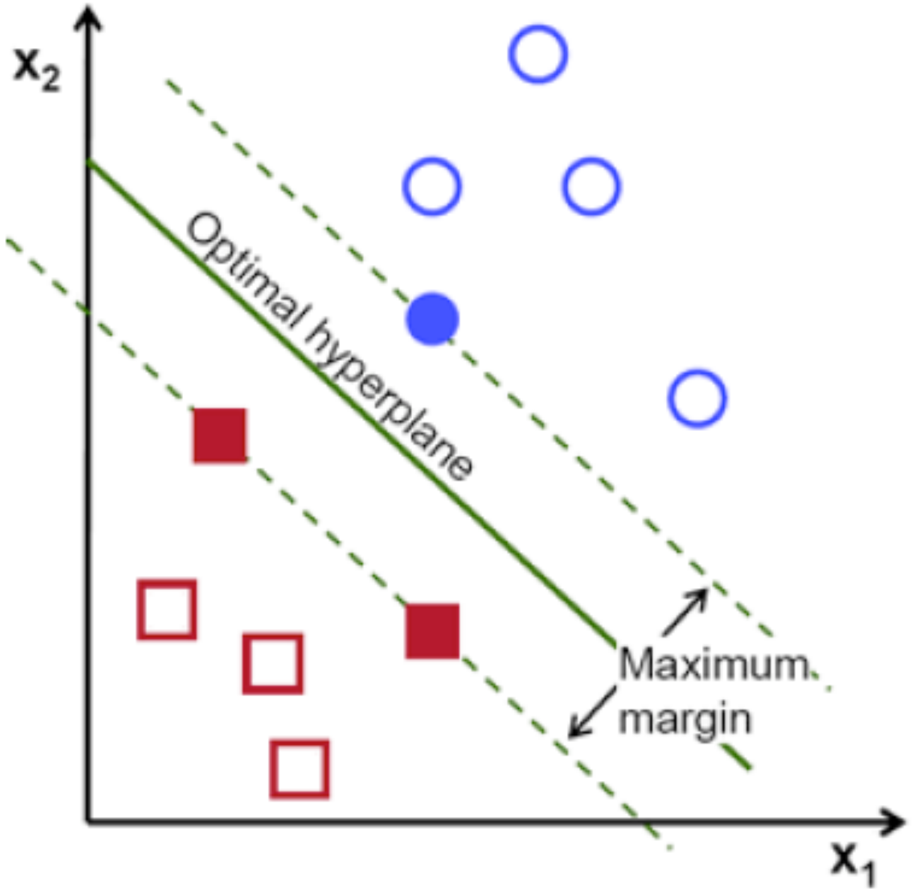
1) a way to understand some of it.
2) reasons to do it

Background 2: SVMs in detail

$$D = \{(x_i, y_i)\}_i \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$



separating hyperplanes (linear classifier, not SVM)



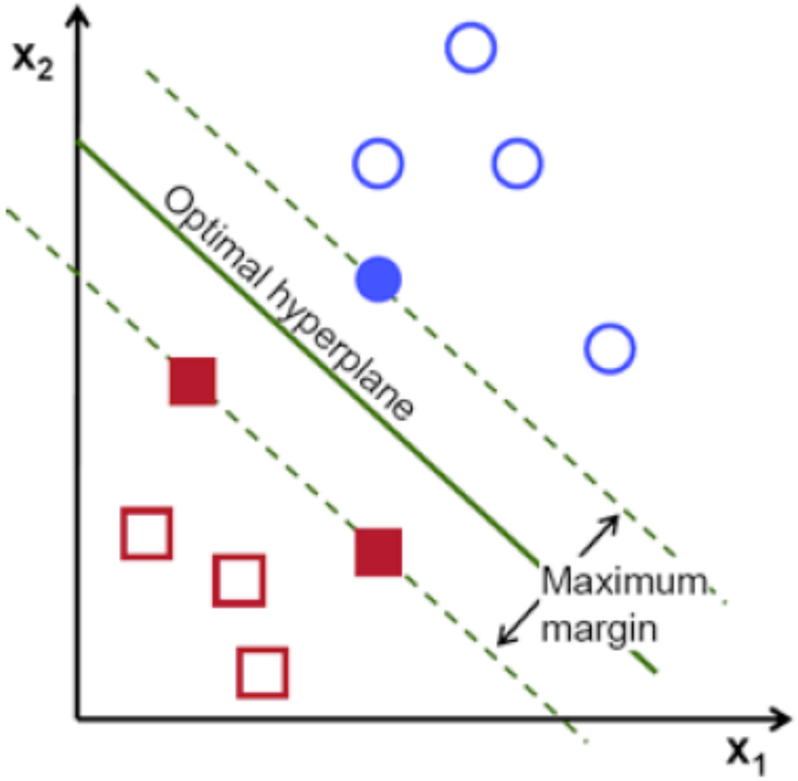
SVM: max-margin hyperplanes

for now, assume linearly separable data

http://opencv-python-tutroals.readthedocs.org

$$D = \{(x_i, y_i)\}_i \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

A number of equivalent formulations…



$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)$ - “functional margin”

$\dfrac{y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)}{\|\mathbf{w}\|}$ - “geometric margin”

$$\underset{\mathbf{w},b}{\arg\max} \min_{i \in \{1,\ldots,N\}} \frac{y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

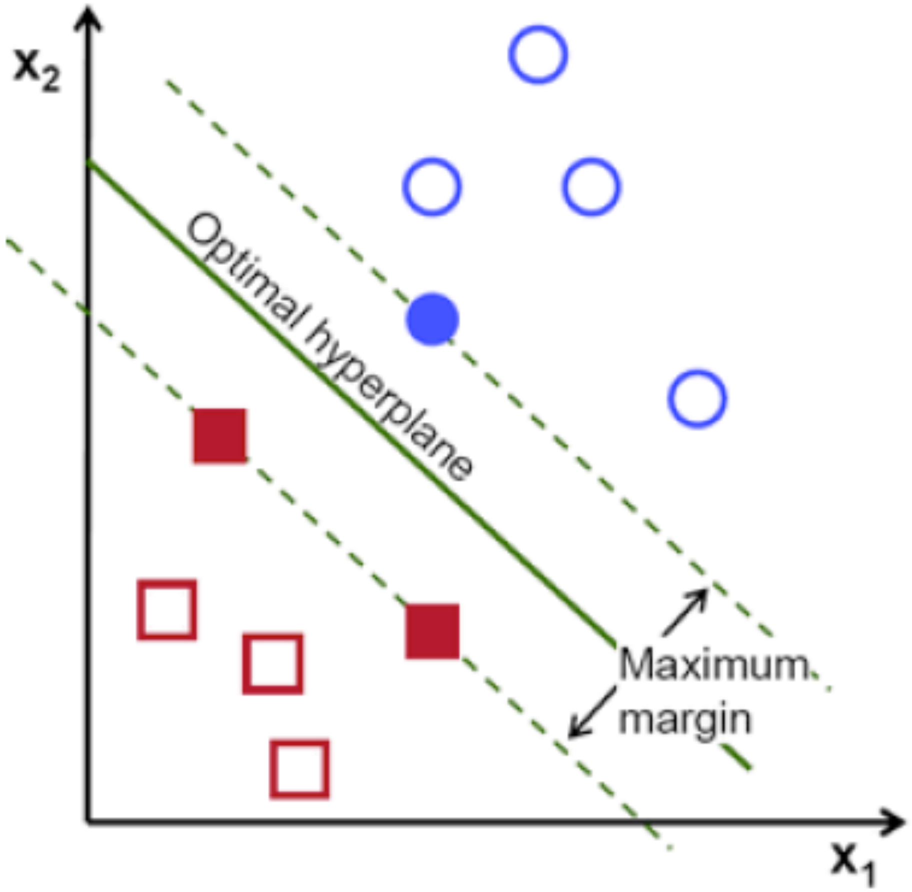
http://opencv-python-tutroals.readthedocs.org

$$D = \{(x_i, y_i)\}_i \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

After some work: quadratic problem

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{such that } y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, N.$$



SVM: max-margin hyperplanes

“*Support vectors*”:
points closest and equidistant
to hyperplane

Hyperplane fully defined
in terms of *support vectors*

http://opencv-python-tutroals.readthedocs.org

Lagrangian approach

Primal problem:

$$\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{such that } y_i(\mathbf{w}^\intercal \mathbf{x}_i + b) \geq 1, \quad i = 1, \ldots, N.$$

Dual problem:

$$\arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i)^\intercal \mathbf{x}_j,$$

$$\text{such that } \alpha_i \geq 0, \qquad \text{for } i = 0, \ldots, N,$$

$$\text{and } \sum_{i=1}^{N} \alpha_i y_i = 0.$$

$$\boxed{\mathbf{w} = \sum_{i=1}^{N} \alpha_i^* y_i \mathbf{x}_i.}$$

Why bother with dual problem? Representation in *terms of datapoints*

- sparser evaluation *(many alpha =0)* $$(\mathbf{w}^*)^\intercal \mathbf{x} + b^* = \left(\sum_{i=1}^{N} \alpha_i y_i (\mathbf{x}_i)^\intercal \mathbf{x}\right) + b^*.$$

- only inner products matter $$\alpha_i \alpha_j y_i y_j (\mathbf{x}_i)^\intercal \mathbf{x}_j,$$

- handy for *quantum tricks*

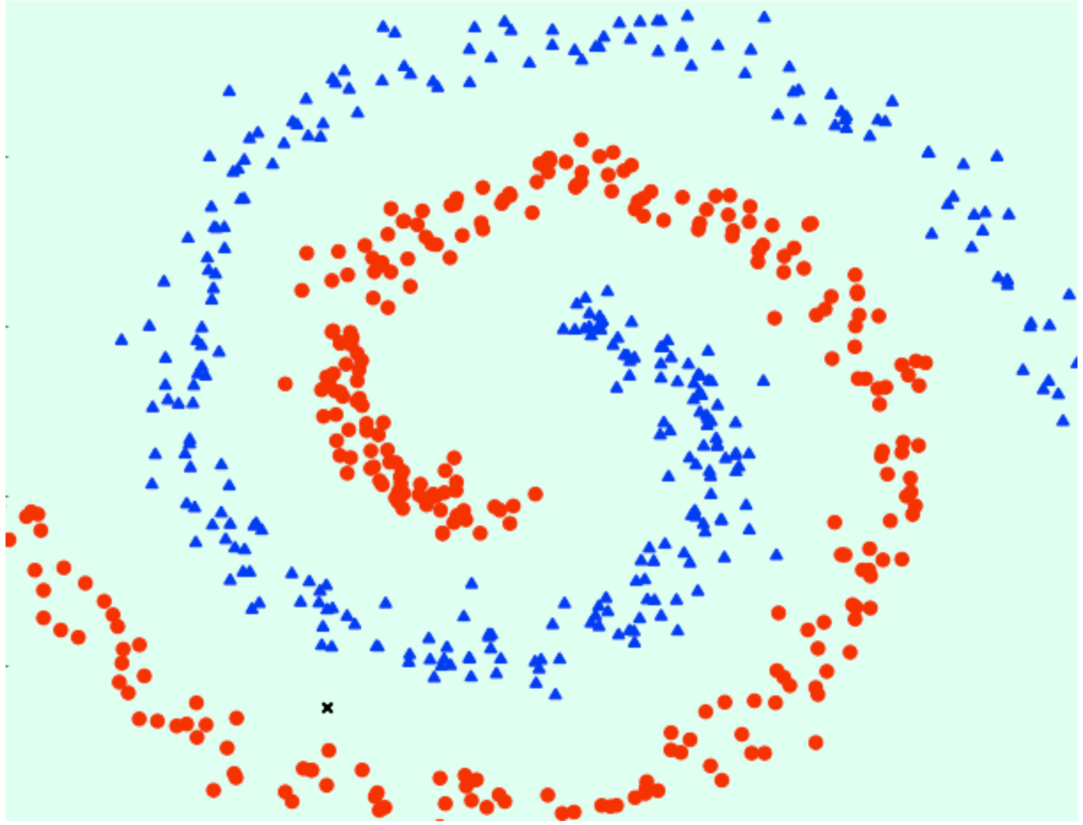
c.f. *Representer theorems*

Comment: the math should not hide the fact we are

simply finding a member of the hypothesis family which is minimizing a loss function

BTW…
almost true: SVMs is “optimized”
to be able to reason about learning performance…

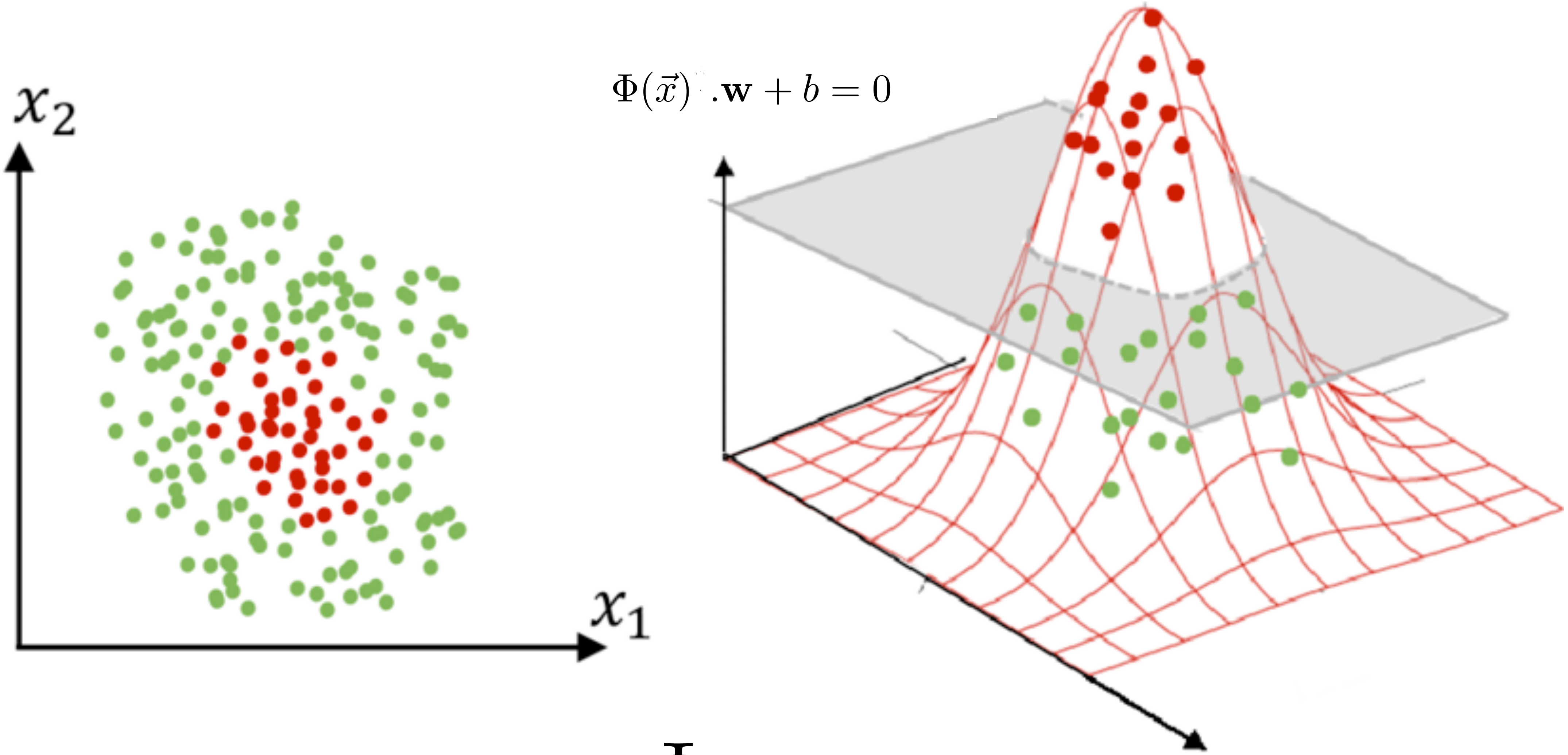
Why should we care about SVMs:
*what about when data is **not** linearly separable?*



Non-separable datasets?

-slack variables (this lead to QSVM - type 1)

-feature mapping and the kernel trick



$$\Phi : \mathbb{R}^d \to \mathbb{R}^D \qquad \vec{x} \xrightarrow{\Phi} \Phi(\vec{x})$$

c.f.: Cover's theorem…

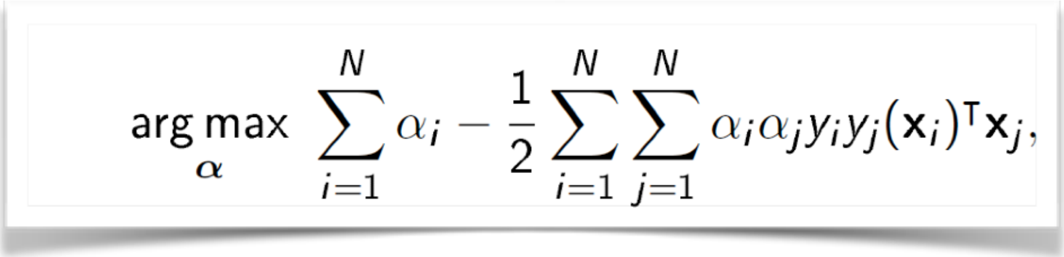
The kernel trick:

one can “train” and evaluate SVM classifiers in rich feature spaces without ever mapping data-points into said spaces. They can even be infinite dimensional

The kernel trick

Note: in dual… only inner products matter

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \quad (\phi = \Phi \ldots)$$



$$\arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i)^{\mathsf{T}} \mathbf{x}_j,$$

$$\arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

$$= \arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \; K(\mathbf{x}_i, \mathbf{x}_j)$$

c.f. Mercer's theorem

The kernel trick

Note: in dual… only inner products matter

$$K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \quad (\phi = \Phi \ldots)$$

$$\arg\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i)^{\mathsf{T}} \mathbf{x}_j,$$

BTW: this thing is called “the kernel”

$\mathbf{x}_i), \phi(\mathbf{x}_j)\rangle$

Note, we really don’t care about the feature map Φ itself…

c.f. Mercer’s theorem
when is a Kernel “valid”?

The kernel trick

Kernels can <u>sometimes</u> be evaluated (much) more efficiently directly:

E.g. (stupidly)

$$(x_1, x_2, x_3) \mapsto \phi(\mathbf{x}) = \begin{pmatrix} x_1x_1 & x_1x_2 & x_1x_3 & x_2x_1 & x_2x_2 & x_2x_3 & x_3x_1 & x_3x_2 & x_3x_3 \end{pmatrix}^\mathsf{T}$$

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \sum_{i=1}^{d} \sum_{j=1}^{d} x_i z_i x_j z_j \qquad \text{Runtime for } \phi(\mathbf{x})\text{: } \mathcal{O}(d^2)$$

The kernel trick

$$\phi(\mathbf{x}) = (x_1x_1 \quad x_1x_2 \quad x_1x_3 \quad x_2x_1 \quad x_2x_2 \quad x_2x_3 \quad x_3x_1 \quad x_3x_2 \quad x_3x_3)^\mathsf{T}$$

reverse-engineered: $K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^\mathsf{T}\mathbf{z})^2 = \left(\sum_{i=1}^{d} x_i z_i\right)\left(\sum_{i=1}^{d} x_i z_i\right) = \sum_{i=1}^{d}\sum_{j=1}^{d} x_i z_i x_j z_j = \langle \phi(\mathbf{x}), \phi(\mathbf{z})\rangle.$

Directly:

Let $\mathbf{x} = (x_1, \ldots, x_d)^\mathsf{T}$, $\mathbf{z} = (z_1, \ldots, z_d)^\mathsf{T}$ and

$$K(\mathbf{x},\mathbf{z}) = (\mathbf{x}^\mathsf{T}\mathbf{z})^2.$$

Runtime: $\mathcal{O}(d)$.

Yay, quadratic speedup

See e.g. Radial basis function kernel

$$K(\mathbf{x},\mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$$

$$K(x,x') = \langle \Phi(x), \Phi(x')\rangle$$

$$\Phi(x) = e^{-x^2/2\sigma^2}\left[1, \sqrt{\frac{1}{1!\sigma^2}}x, \sqrt{\frac{1}{2!\sigma^4}}x^2, \sqrt{\frac{1}{3!\sigma^6}}x^3, \ldots\right]^T$$

inf. dim…..

c.f. Mercer’s theorem

To keep in mind:

-primal v.s. dual:

in primal, optimize over normal vector **explicitly;**
in dual, i**t is implicit**, and the separating hyperplane is expressed in terms of data points

-feature maps:

by raising dimension non-linearly, we can achieve linear-separability

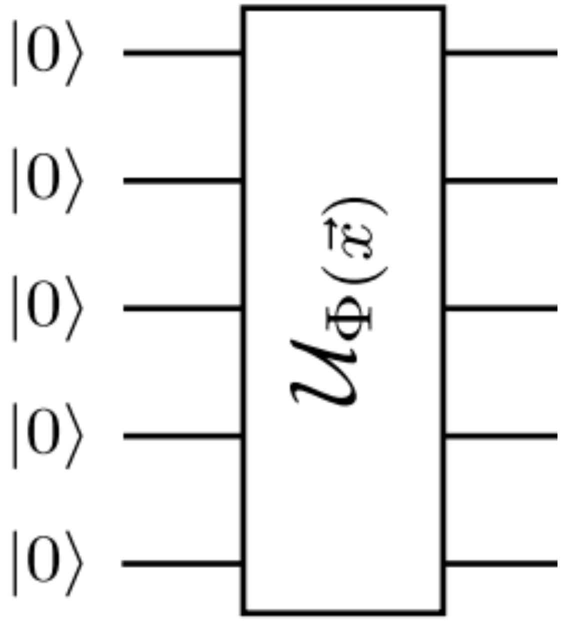
-kernel trick:

in dual formulation, only need kernel evaluation on data points for training.

see axiv:1803.07128

Back to Quantum: an SVM reading of PQC-powered ML

Basic idea: quantum computing offers interesting “natively quantum” feature maps and kernels



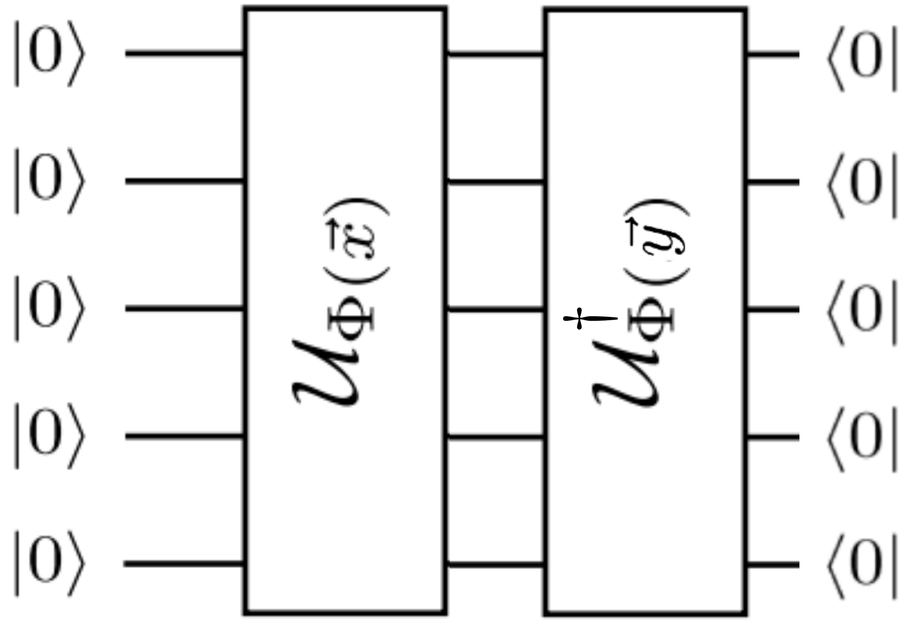
$$\vec{x} \mapsto \mathcal{U}_{\Phi}(\vec{x})|0\rangle = |\Phi(\vec{x})\rangle$$

Data is encoded ***in the circuit parameters*** *(not input state). More general.*

Nature. vol. 567, pp. 209-212 (2019)

Basic idea: quantum computing offers interesting "natively quantum" feature maps and kernels

One thing we can do with this… is evaluate inner products.



Kernel!

$$|\langle\Phi(\vec{y})|\Phi(\vec{x})\rangle|^2$$

Can be hard to compute.

Do this quantumly
(recall QC is good for inner products)
also possible:
swap tests, hadamard tests

But we can do more…

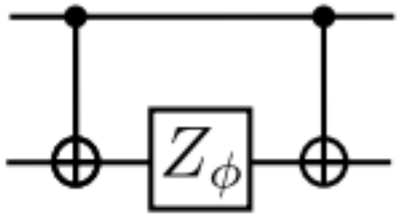
Nature. vol. 567, pp. 209-212 (2019)

Which feature maps should we construct?

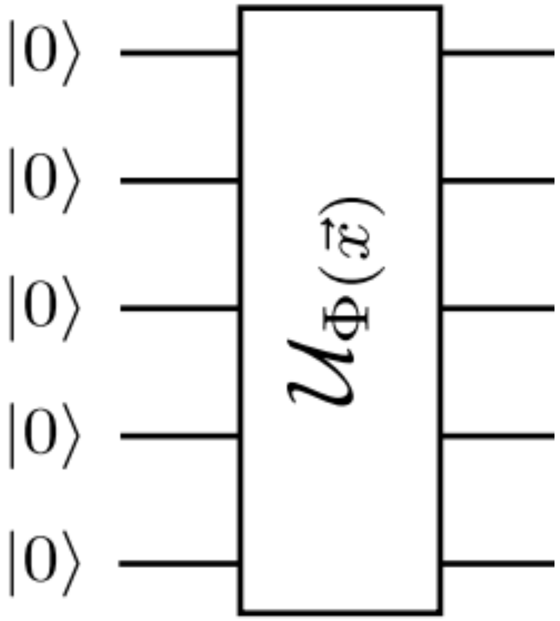
$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S\subseteq[n]} \phi_S(\vec{x}) \prod_{i\in S} Z_i\right)$$

$$\phi_{\{i\}}(\vec{x}) = x_i \text{ and } \phi_{\{1,2\}}(\vec{x}) = (\pi - x_1)(\pi - x_2)$$

$$e^{i\phi_{\{l,m\}}(\vec{x})Z_l Z_m} \quad =$$



$$\mathcal{U}_\Phi = H^{\otimes n}U_\Phi H^{\otimes n}U_\Phi \cdots H^{\otimes n}U_\Phi$$



Nature. vol. 567, pp. 209-212 (2019)

Which feature maps should we construct…elaborated

- DIMENSION OF FEATURE SPACE = $2^{\#\text{QUBITS}}$; # QUBITS = N = INITIAL DIMENSION

- DEFINE "SUBMAPS" ϕ_S S = -individual vector entries ; $S \subseteq \{1 ... N\}$
  -pairs ; ... can be generalized
  := correlators (2 OR k-local)

  $\phi_S : \mathbb{R}$ or $\mathbb{R}^2 \longrightarrow$ "angles"

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \to \phi_S \longrightarrow \theta \longrightarrow \underbrace{\exp\left(i\; Z_1 \otimes Z_3\; \theta\right)}_{U_S(\vec{x})}$$

- $U_{\Phi(\vec{x})} := \prod_S U_S(\vec{x})$ ... All DIAGONAL..

- FEATURE MAP : $\mathcal{U}_{\Phi} = \left(H^{\otimes N}\, U_{\Phi(x)}\right)^{\otimes m}$

m- hyperparameter

First type of PQC SVM: implicit (dual) model

training:

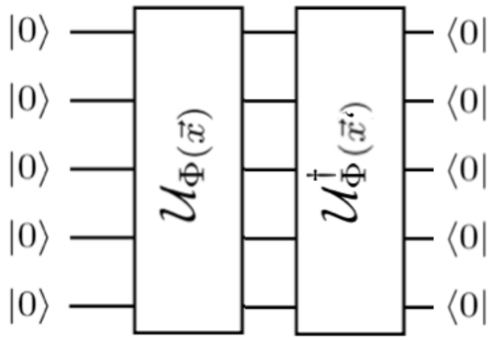
$$\underset{\boldsymbol{\alpha}}{\arg\max} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \; K(\mathbf{x}_i, \mathbf{x}_j)$$

classifying:

$$out(\mathbf{x}) = sign\left(\sum_{i=1}^{N} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}) + b\right)$$

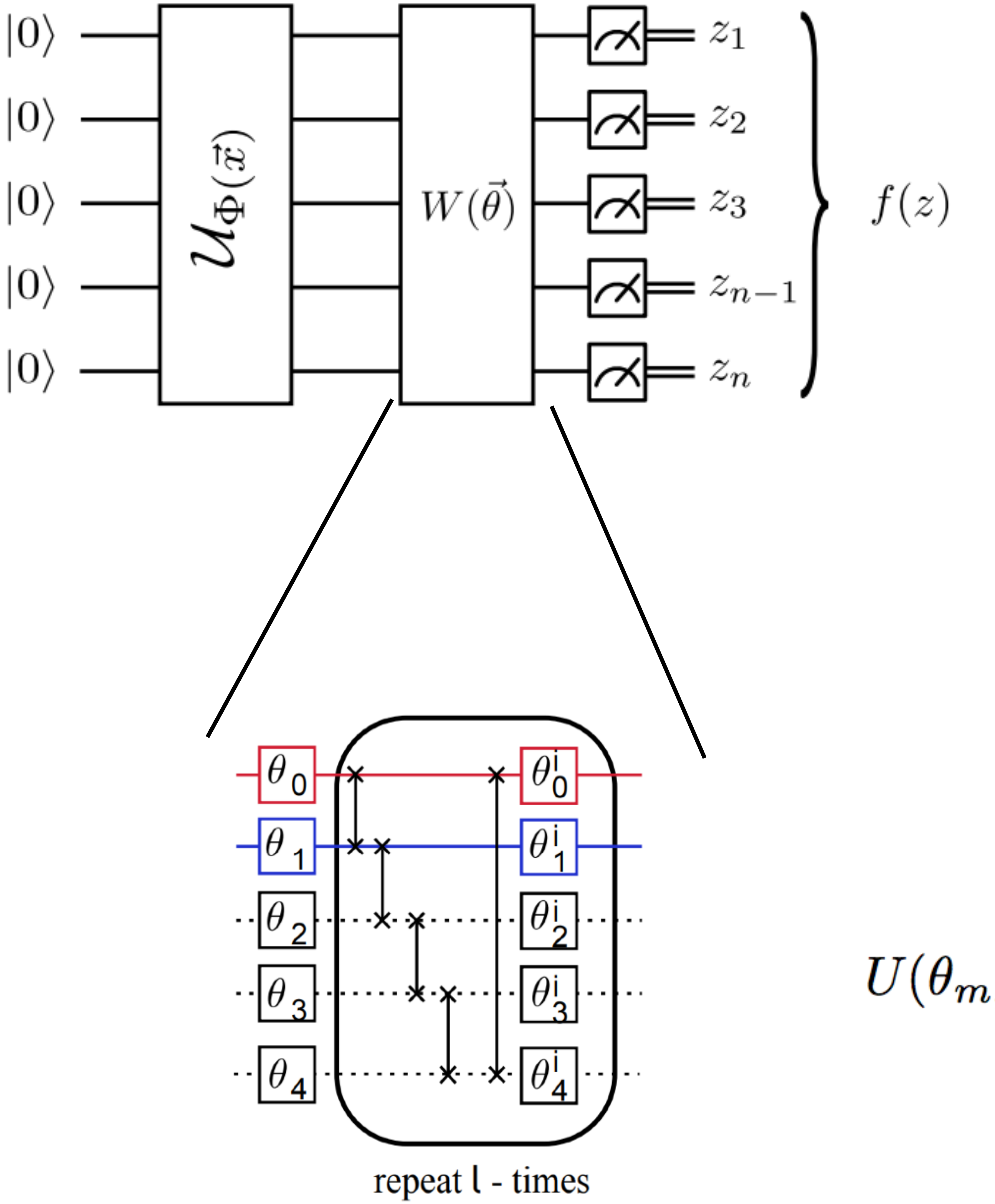
Quantum parts: needed in both to evaluate the kernels

only offline; optimization essentially on classical data.



$$O(N^2/poly(\epsilon))$$

Fully quantum model: explicit (primal) model

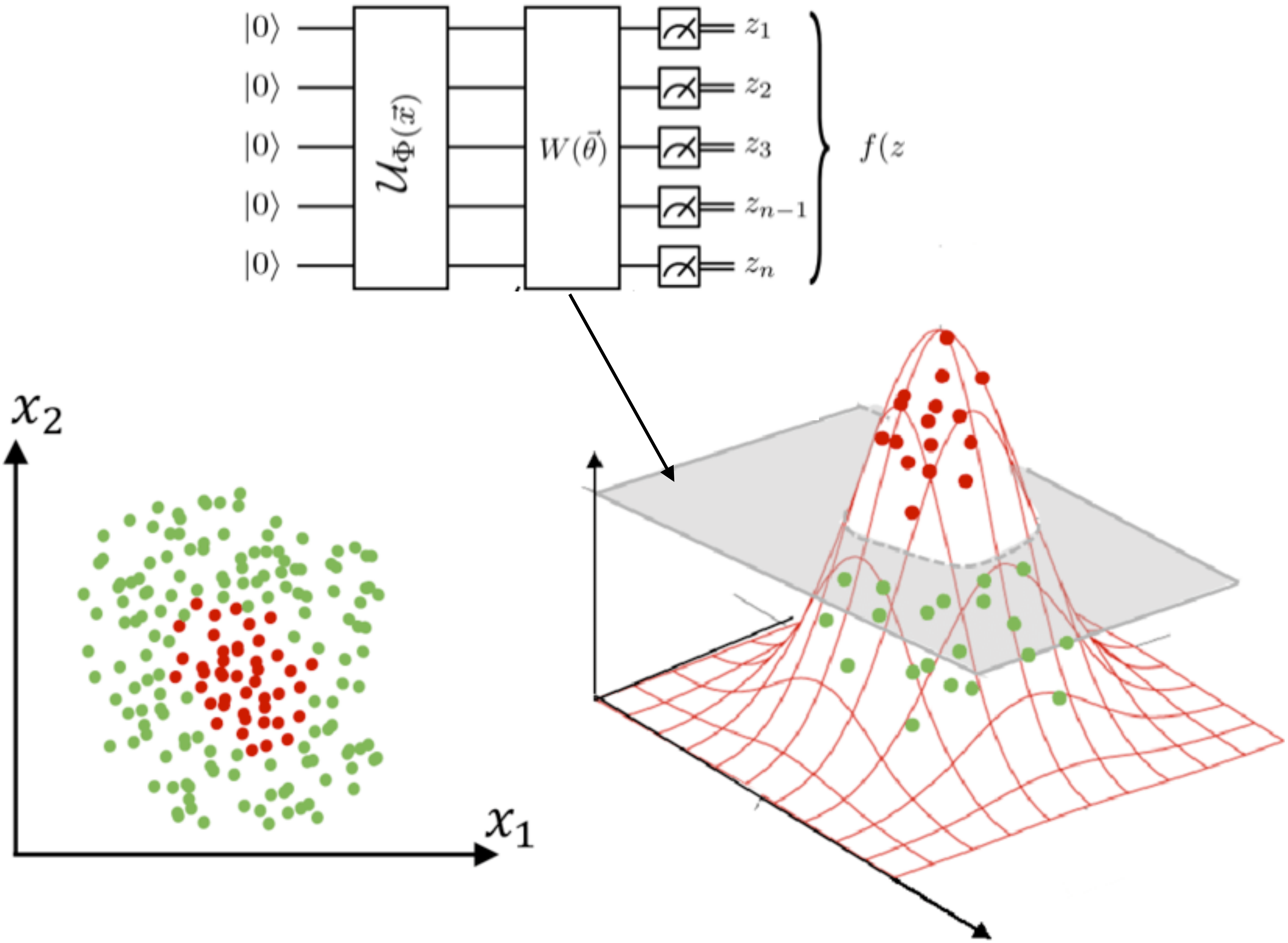


$$f(z) : \{0,1\}^n \rightarrow \{-1,1\}$$

$$U(\theta_{m,t}) = e^{i\frac{1}{2}\theta^z_{m,t} Z_m} e^{i\frac{1}{2}\theta^y_{m,t} Y_m}$$

Nature. vol. 567, pp. 209-212 (2019)

Intuition:



$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$\mathcal{U}_{\Phi(\vec{x})}$
$W(\vec{\theta})$
z_1
z_2
z_3
z_{n-1}
z_n
$f(z$
x_2
x_1

How does it output a label? What is the achieved classfier?

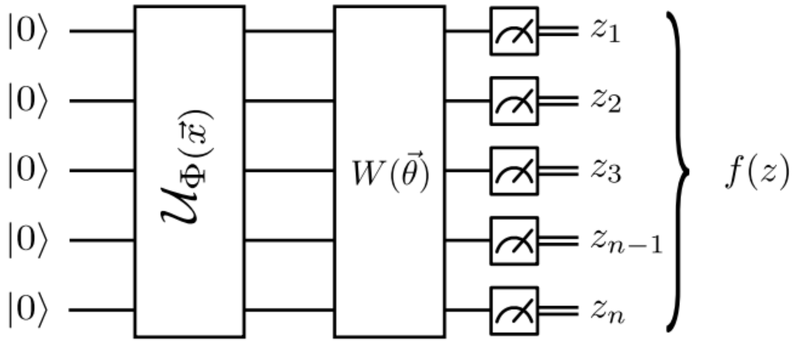
The label (output): (approximately) sign of the expected value *f,* shifted by *b:*

$$out(\mathbf{x}, \theta) \approx sign(\mathbb{E}_{z \sim Q.circ}[f(z)] + b)$$

$$out(\mathbf{x}, \theta) \approx \operatorname{sign}(\langle \Phi(\vec{x}) | W^{\dagger}(\vec{\theta}) \mathbf{f} W(\vec{\theta}) | \Phi(\vec{x}) \rangle + b)$$

The algorithm:

-sample z many times (“shots”)
-average, shift, compute sign.



Nature. vol. 567, pp. 209-212 (2019)

Comment:

$$out(\mathbf{x}, \theta) \approx sign(\mathbb{E}_{z \sim Q.circ}[f(z)] + b)$$

$$out(\mathbf{x}, \theta) \approx \operatorname{sign}(\langle \Phi(\vec{x}) | W^{\dagger}(\vec{\theta}) \mathbf{f} W(\vec{\theta}) | \Phi(\vec{x}) \rangle + b)$$

"measure each qubit in comp basis, compute $f(\vec{z})$" = observable

$$\mathbf{f} := \sum_{\vec{z}} f(\vec{z}) |\vec{z}\rangle\langle\vec{z}|$$

"measure each qubit in comp basis, compute $f(\vec{z})$" := a realization of measurement of $\mathbf{f}$. AVERAGING YIELDS THE EXPECTED VALUE.

Nature. vol. 567, pp. 209-212 (2019)

How does it learn?

Optimize θ to minimize some loss/error/empirical risk on dataset

Involves evaluation of classifier function many times…

Often: stochastic gradient descent

Q. chemistry optimization and optimization here very similar

Nature. vol. 567, pp. 209-212 (2019)

But what does it *do?*

SVM CLASSIFIER : $\mathrm{sign}\left(\vec{n}.\vec{x}+b\right)$

HERE : $\mathrm{SIGN}\left(\underbrace{\langle\phi(x)|W^{\dagger}(\theta)\,f\,W(\theta)\,|\phi(x)\rangle}_{\text{inner product?}}+b\right)$

$$\langle\phi(x)|W^{\dagger}fW|\phi(x)\rangle = \mathrm{Tr}\Big[\underbrace{W^{\dagger}fW}_{A}\,\overbrace{|\phi(x)\rangle\langle\phi(x)|}^{B}\Big] = \left(A\,,\,B\right)_{Fr}$$

Def. $(\vec{w})_{\alpha} = \mathrm{Tr}\left[W^{\dagger}fW\,P_{\alpha}\right]$; P_{α} - Pauli string $\alpha\in[0\,..\,4^{n-1}]$

$$\left(\vec{\phi}(\vec{x})\right)_{\alpha} = \mathrm{Tr}\Big(|\phi(x)\rangle\langle\phi(x)|\,P_{\alpha}\Big]$$

$$\boxed{\mathrm{out}(\vec{x}) \doteq \mathrm{SIGN}\left(\vec{w}\,.\,\vec{\phi}(\vec{x})\;+b\right)}$$

the feature space is that of density operators…

Nature. vol. 567, pp. 209-212 (2019)
arxiv:1804.11326. p.g. 12.

What does it *do?*

$$[\vec{w}(\theta)]_{\alpha} = \mathrm{tr}\left[W^{\dagger}(\vec{\theta})\mathbf{f}W(\vec{\theta})P_{\alpha}\right]$$

$$[\vec{\phi}(\vec{x})]_{\alpha} = \langle\Phi(\vec{x})\,|P_{\alpha}|\,\Phi(\vec{x})\rangle$$

$$out(\vec{\mathbf{x}}) \approx sign(\overrightarrow{w}\ .\ \overrightarrow{\Phi}(\vec{\mathbf{x}}) + b)$$

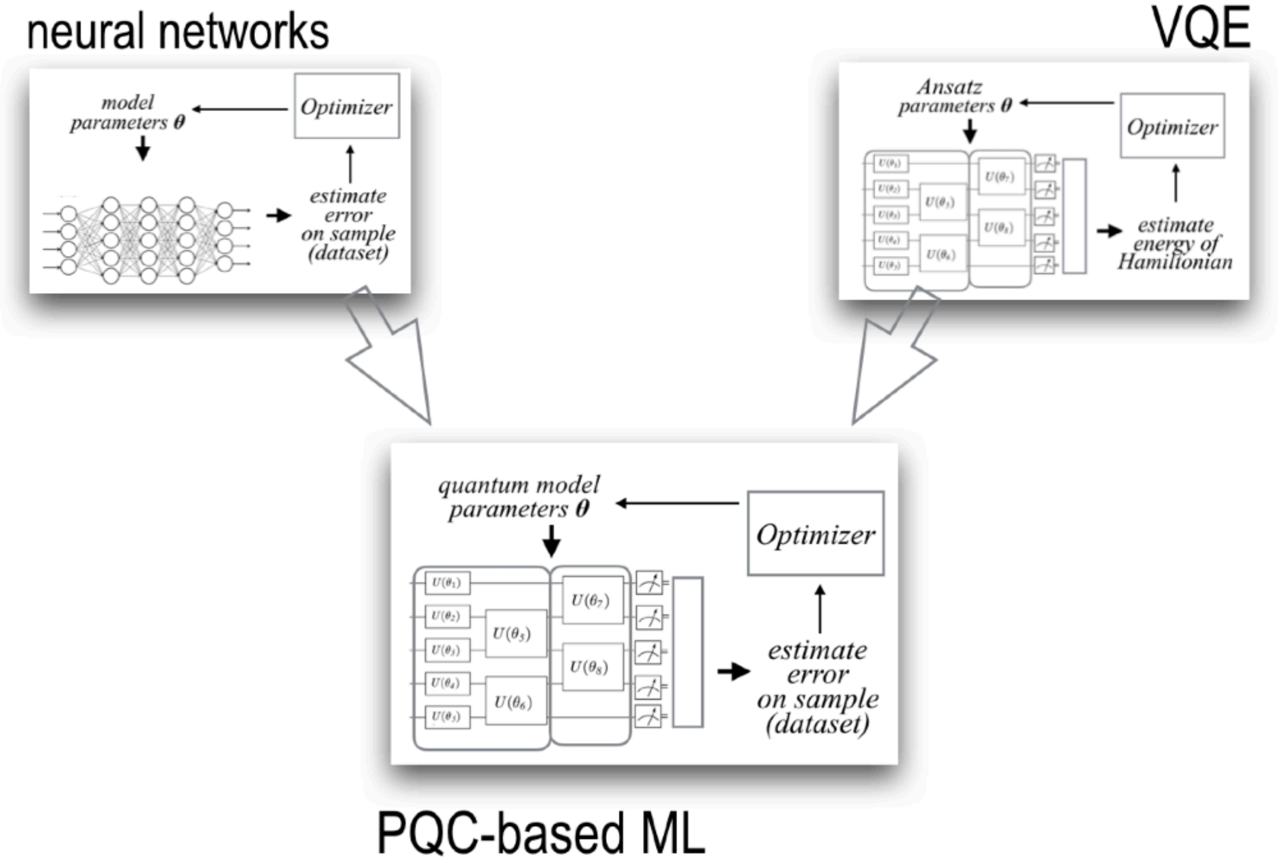
-limitations on the model come into play here…
-not *all hyperplanes* reachable…

-not maximal margin attained!

BECAUSE $W(\theta)$ & $\mathbf{f}$ ARE RESTRICTED.

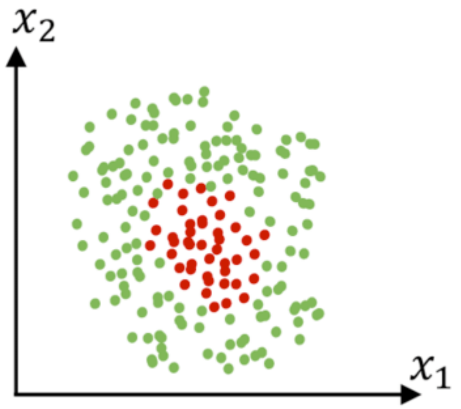
Nature. vol. 567, pp. 209-212 (2019)

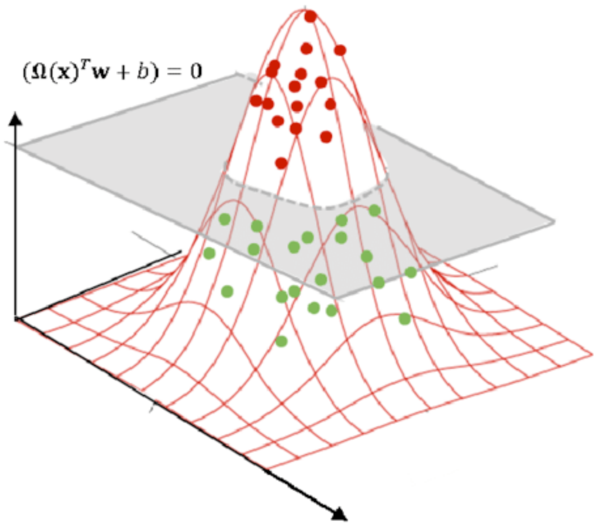
Note the explicit model is much like training a NN/VQE


neural networks
model parameters θ
Optimizer
estimate error on sample (dataset)
VQE
Ansatz parameters θ
Optimizer
estimate energy of Hamiltonian
quantum model parameters θ
Optimizer
$U(\theta_1)$
$U(\theta_2)$
$U(\theta_3)$
$U(\theta_4)$
$U(\theta_5)$
$U(\theta_6)$
$U(\theta_7)$
$U(\theta_8)$
estimate error on sample (dataset)
PQC-based ML

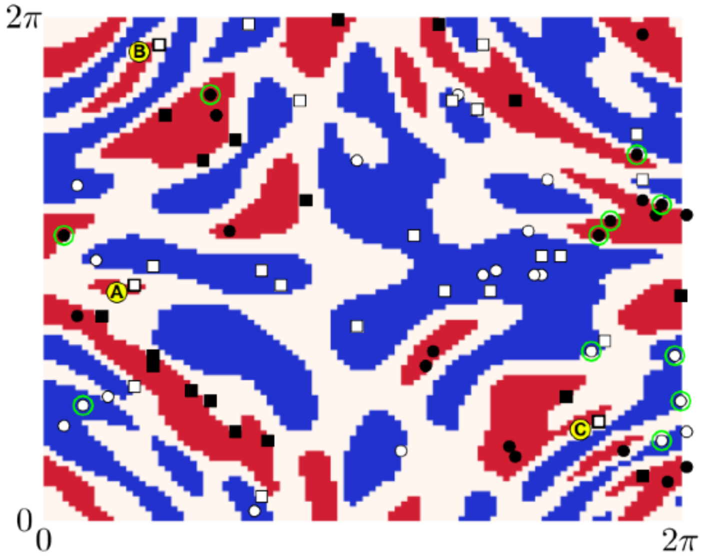
-but with a connection with a well-understood classical model

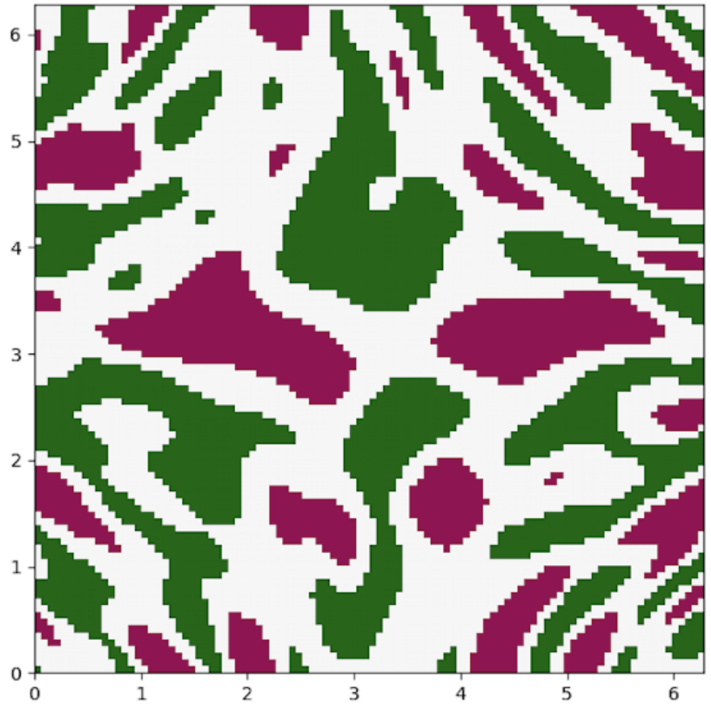
Illustration of quantum decision boundaries

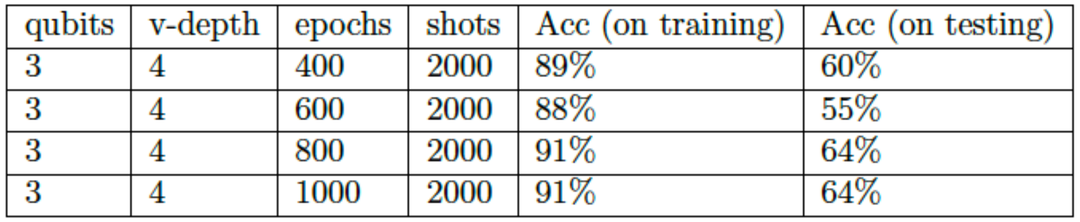




Two slices of quantum kernels (*decision boundaries*):

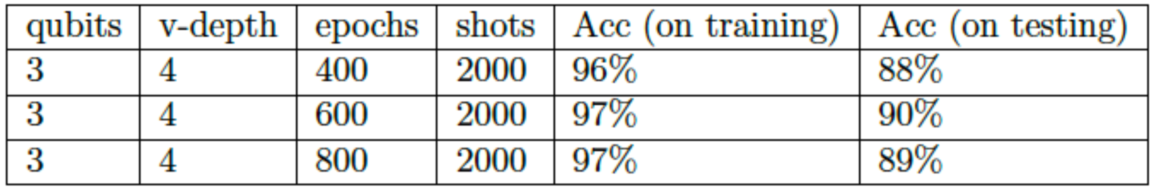






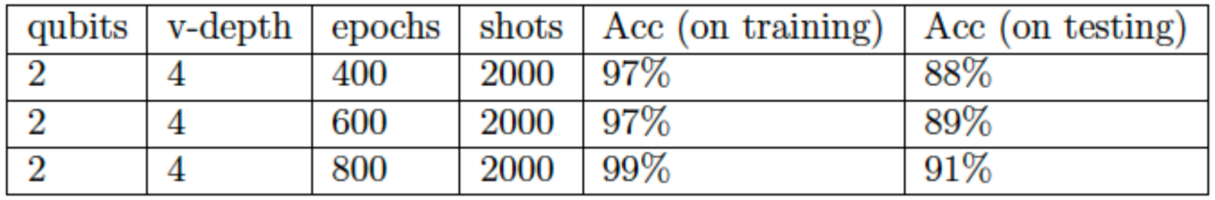
qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	400	2000	89%	60%
3	4	600	2000	88%	55%
3	4	800	2000	91%	64%
3	4	1000	2000	91%	64%

Table 5.4: results of Wine dataset on 3-qubits



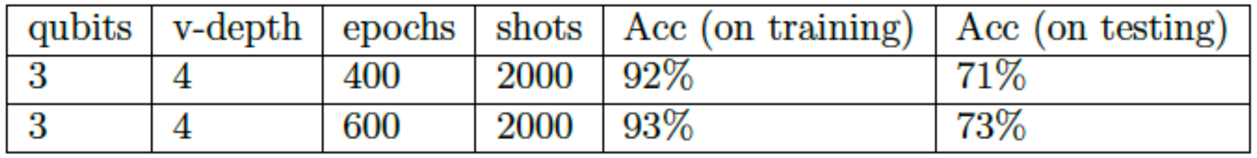
qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	400	2000	96%	88%
3	4	600	2000	97%	90%
3	4	800	2000	97%	89%

Table 5.8: results of MNIST dataset on 3-qubits



qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
2	4	400	2000	97%	88%
2	4	600	2000	97%	89%
2	4	800	2000	99%	91%

Table 5.5: results of breast cancer dataset on 2-qubits



qubits	v-depth	epochs	shots	Acc (on training)	Acc (on testing)
3	4	400	2000	92%	71%
3	4	600	2000	93%	73%

Table 5.6: results of Cancer dataset on 3-qubits

Sevak Mardirosian MSc thesis, LIACS, 2019

Quantum advantage, and advantage for (near term) quantum

-for quantum advantage: *useful* and *classically* hard

-for advantage for near-term quantum: *useful* and *doable*

Quantum advantage, and advantage for (near term) quantum

-for quantum advantage: *useful* and *classically* hard

-useful: remains to be seen;

- almost all models useful in some settings;
  here when data has complex correlations.
- Bleeding edge reasearch:
- theory for ML is difficult;
- QCs just becoming large enough for experiments

Quantum advantage, and advantage for (near term) quantum

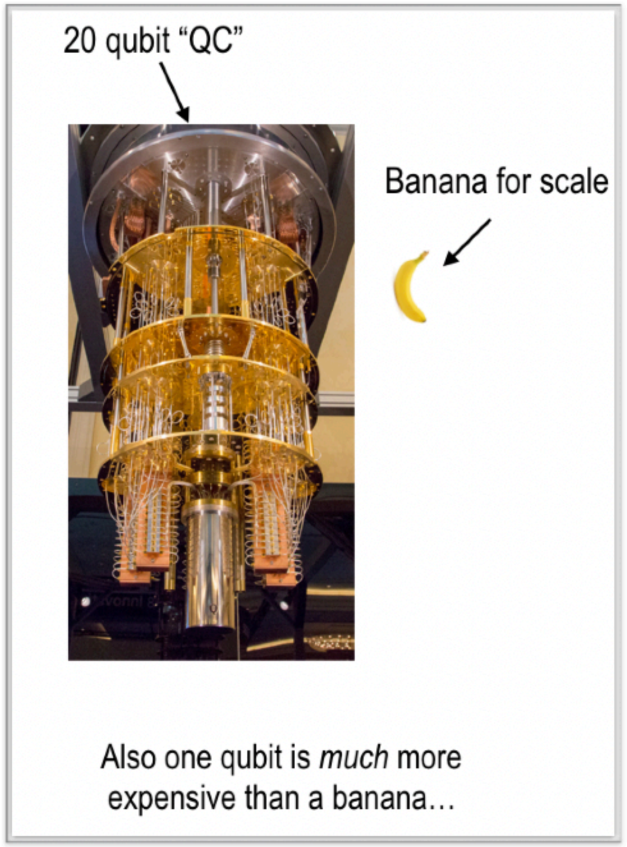
-for quantum advantage: *useful* and *classically* hard

-classically hard:

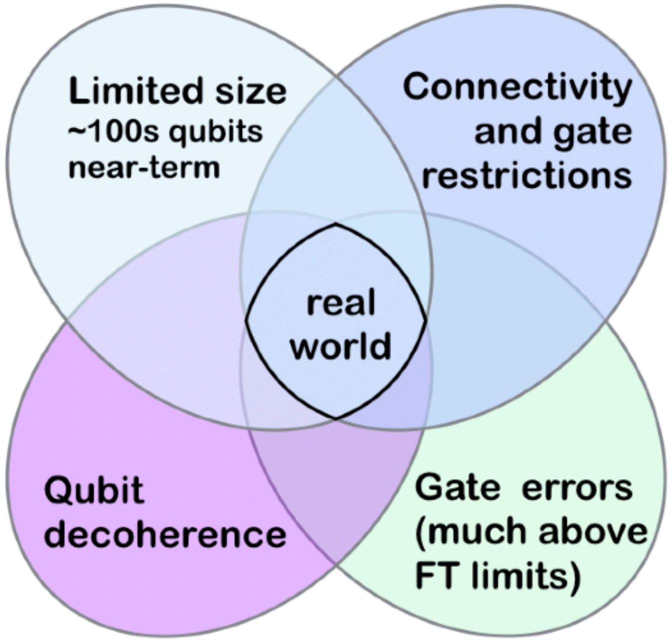
- trivially there exist “BQP-hard“ kernels (for deep circuits)
- for “functional problems” no hard separation results but; very likely hard.
- more interestingly; likely hard in shallow circuit regime

Quantum advantage, and advantage for (near term) quantum

-for advantage for near-term quantum: *useful* and *doable*



Also one qubit is *much* more expensive than a banana…



Limited size ~100s qubits near-term

Connectivity and gate restrictions

real world

Qubit decoherence

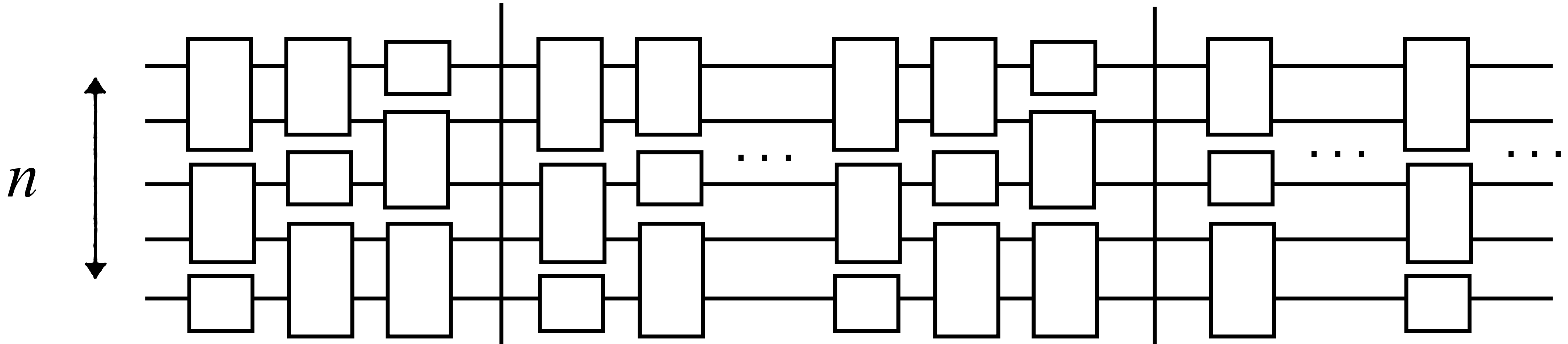
Gate errors (much above FT limits)

doable: makes sense with: ~100 qubits, limited depth, errors

Quantum advantage, and advantage for (near term) quantum

1) *~100 qubits - probably yes* 2^{100} *is interesting*
2) *depth?*
3) *noise?*

Recall ***Quantum depth complexity***



O(1)* *(indep. from n)

-**better** than *classical const depth* for relational problems

-***likely*** better for sampling problems, no matter what depth of classical computer

-**NOT better** than CC for decision problems

O(log^k(n))

Hard part *of Shor's algo.*

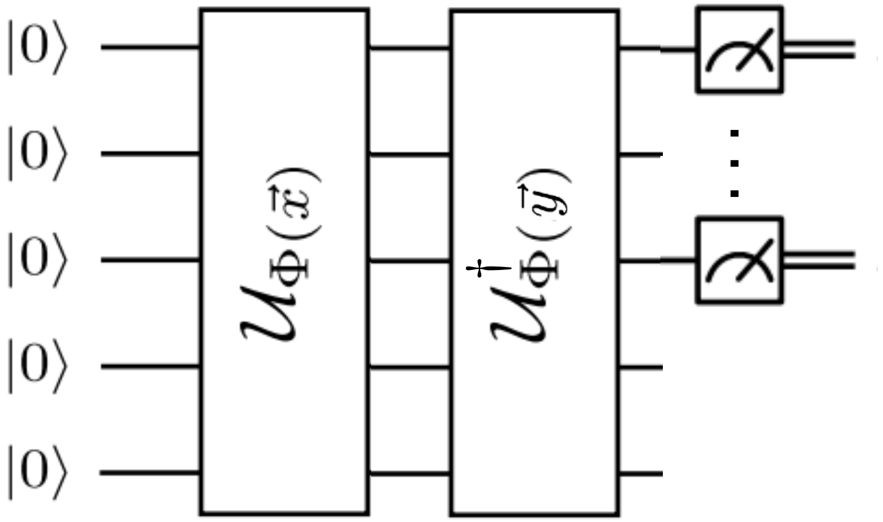
Ground states
of complex systems in polytime
(multi-scale entanglement renormalization ansatz)

O(poly(n))

"BQP" = full QC

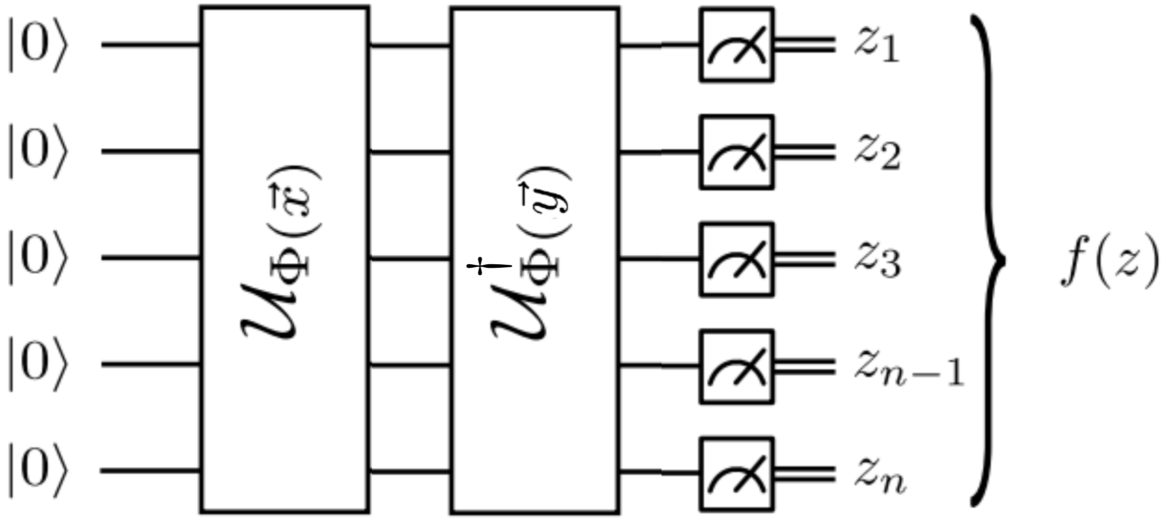
10.1126/science.aar3106

can we have limited depth **and** classically hard?


$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$\mathcal{U}_{\Phi(\vec{x})}$
$\mathcal{U}^{\dagger}_{\Phi(\vec{y})}$

Lm: full exact simulation of output of **log-many qubits** in constant depth, can be done in poly-time

This is the situation in chemistry with log-local Hamiltonians - depth must be at least log.



Not log-many!
No known efficient classical algorithm

1) *~100 qubits* ✓
2) *depth* ✓
3) *noise?*

Reasons for optimism:
a) ML as signal-from-noise + source shifting
b) stochastic hypothesis families and noisy data (distinct from mathematical optimization)
c) brains are noisy :)