And now for something completely different:

Quantum Support Vector Machines

but this time without quantum databases, or
quantum-linear-systems-HHL quantum linear algebra tricks

this one could work on near-term quantum computers

BASED ON :

Supervised learning with quantum enhanced feature spaces

Havlicek, Córcoles, Temme, Harrow, Kandala, Chow, Gambetta

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

& a bunch of other literature...

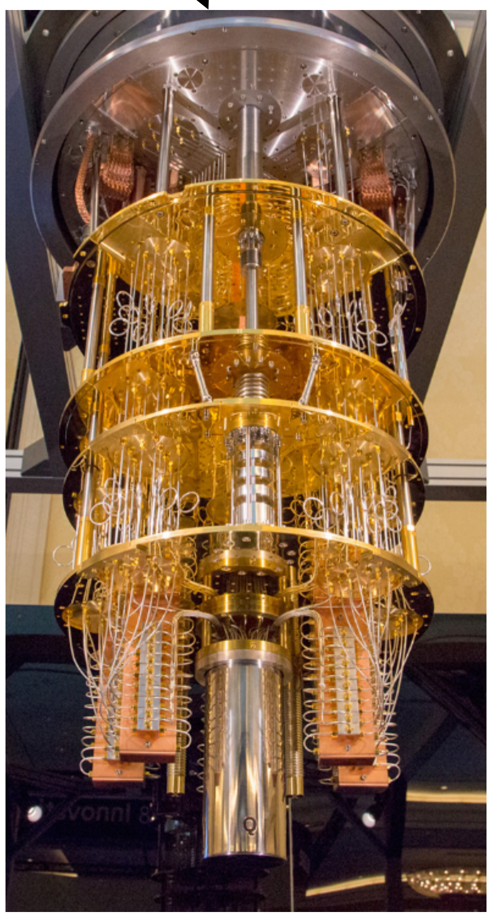
1) *Background*
    - *limitations of QCs*
    - *Support Vector Machines, QSVM (1) and the kernel trick*
    - *Variational (parametrized) quantum circuits*

2) *Support Vector Machine with quantum kernels*
    1) *version 1: “quantum-assisted”*
    2) *version 2: “full quantum”*

3) *Some results*

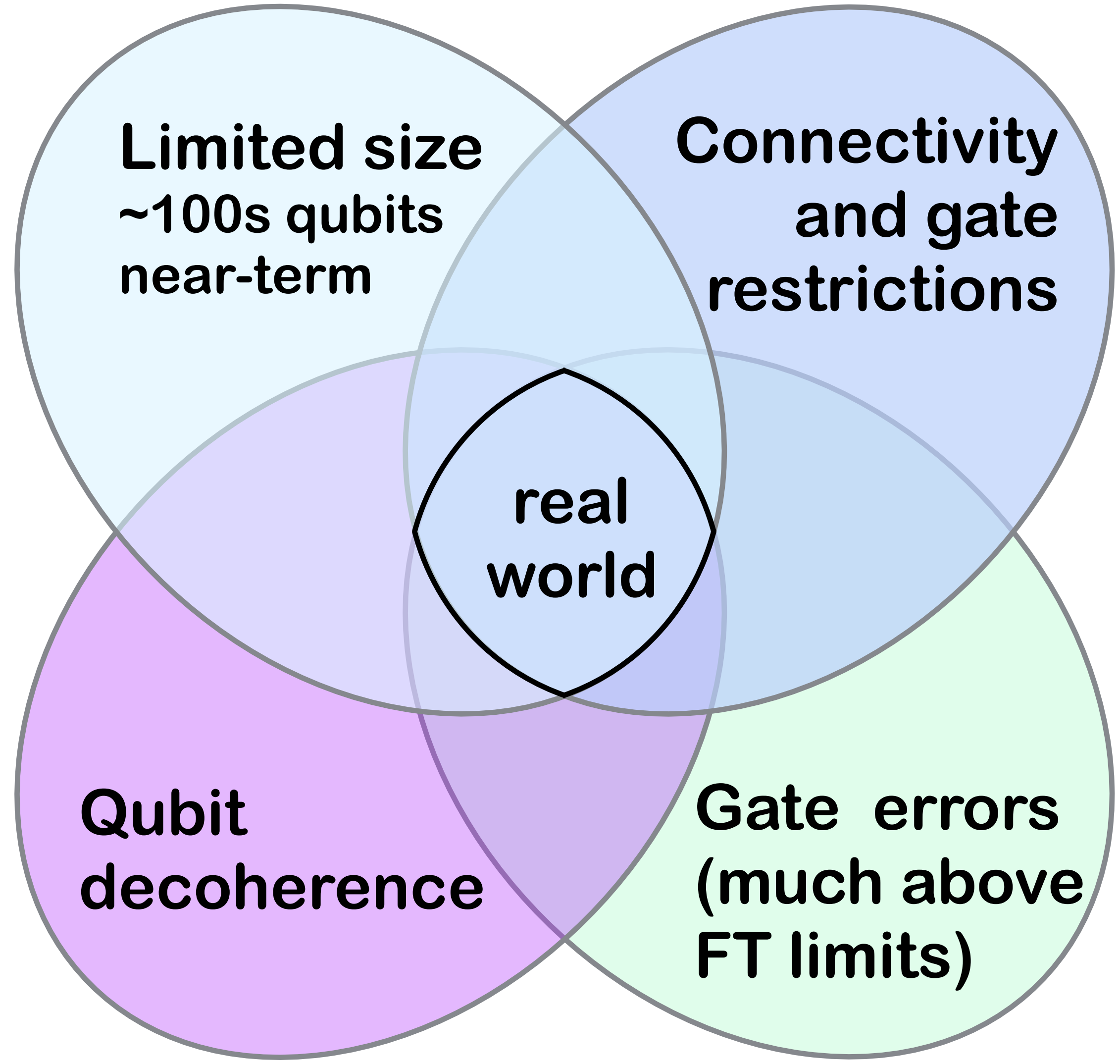
1) *Limitations of real-world QCs*

20 qubit "QC"



Banana for scale

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



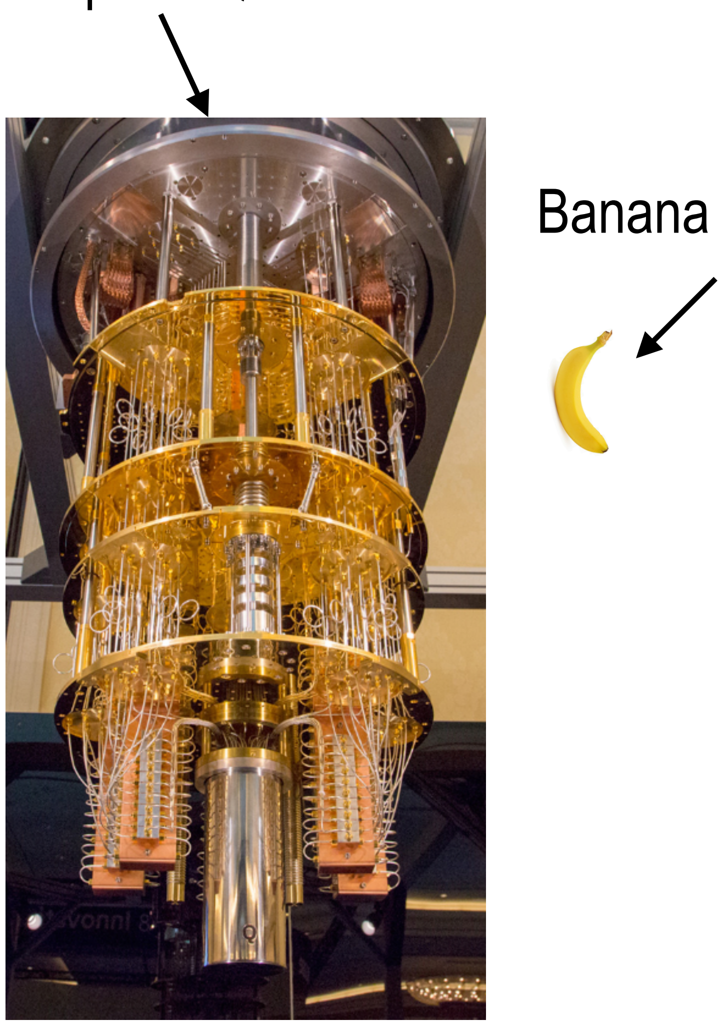
Decoherence: effects leading to degradation of qubit(s) state usually from "coupling to environment" and *relaxation*

- dephasing (environment "measures" qubit)
- de-polarization (gets noisy)
- relaxation (collapses to "ground state")
- dissipation

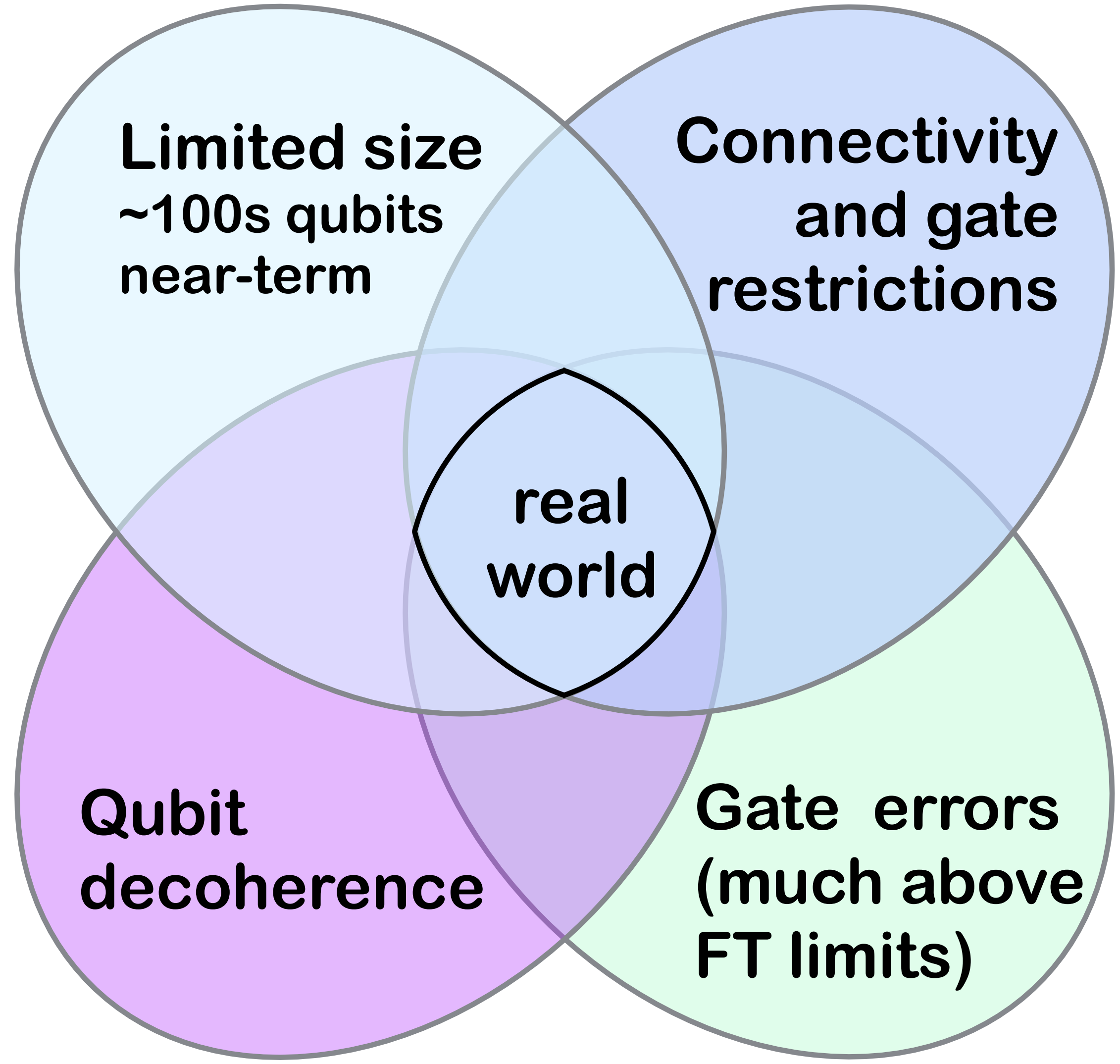
1) *Limitations of real-world QCs*

20 qubit “QC”

Banana for scale



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



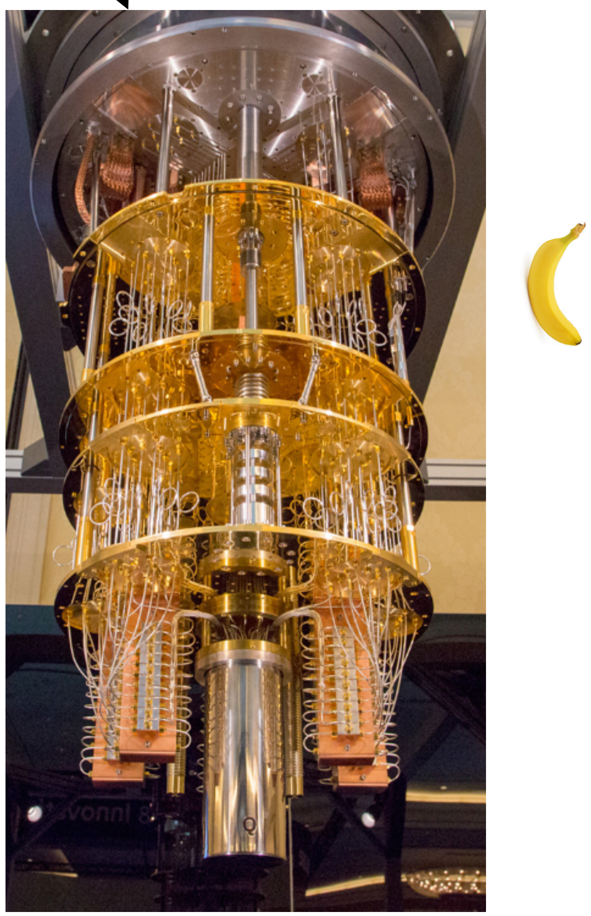
In short: qubits have a life-time (half-life)…. up to miliseconds

gates can take 10s-100s of nanoseconds

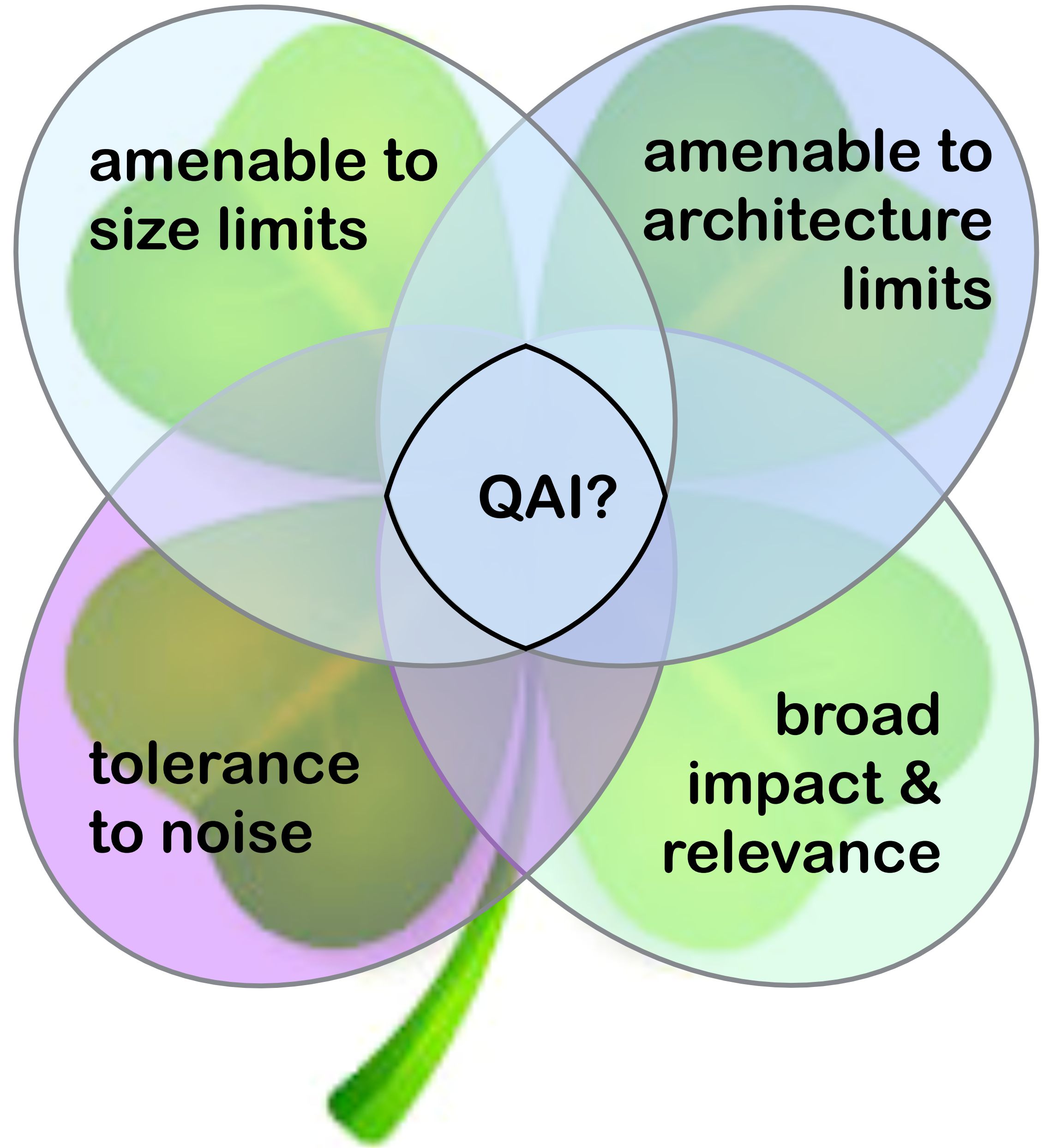
1) Limitations of real-world QCs

20 qubit "QC"

Banana for scale



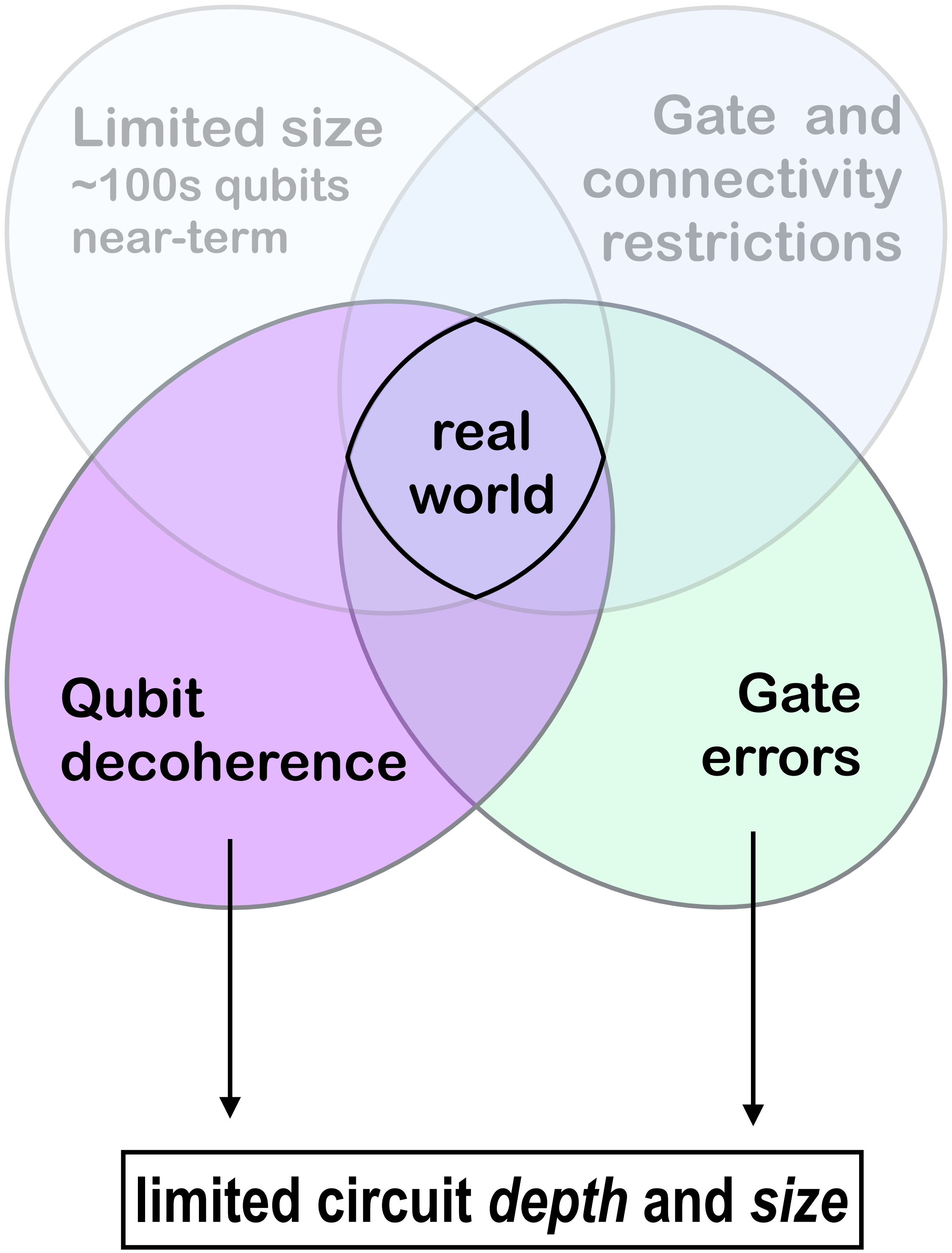
Also one qubit is *much* more expensive than a banana…

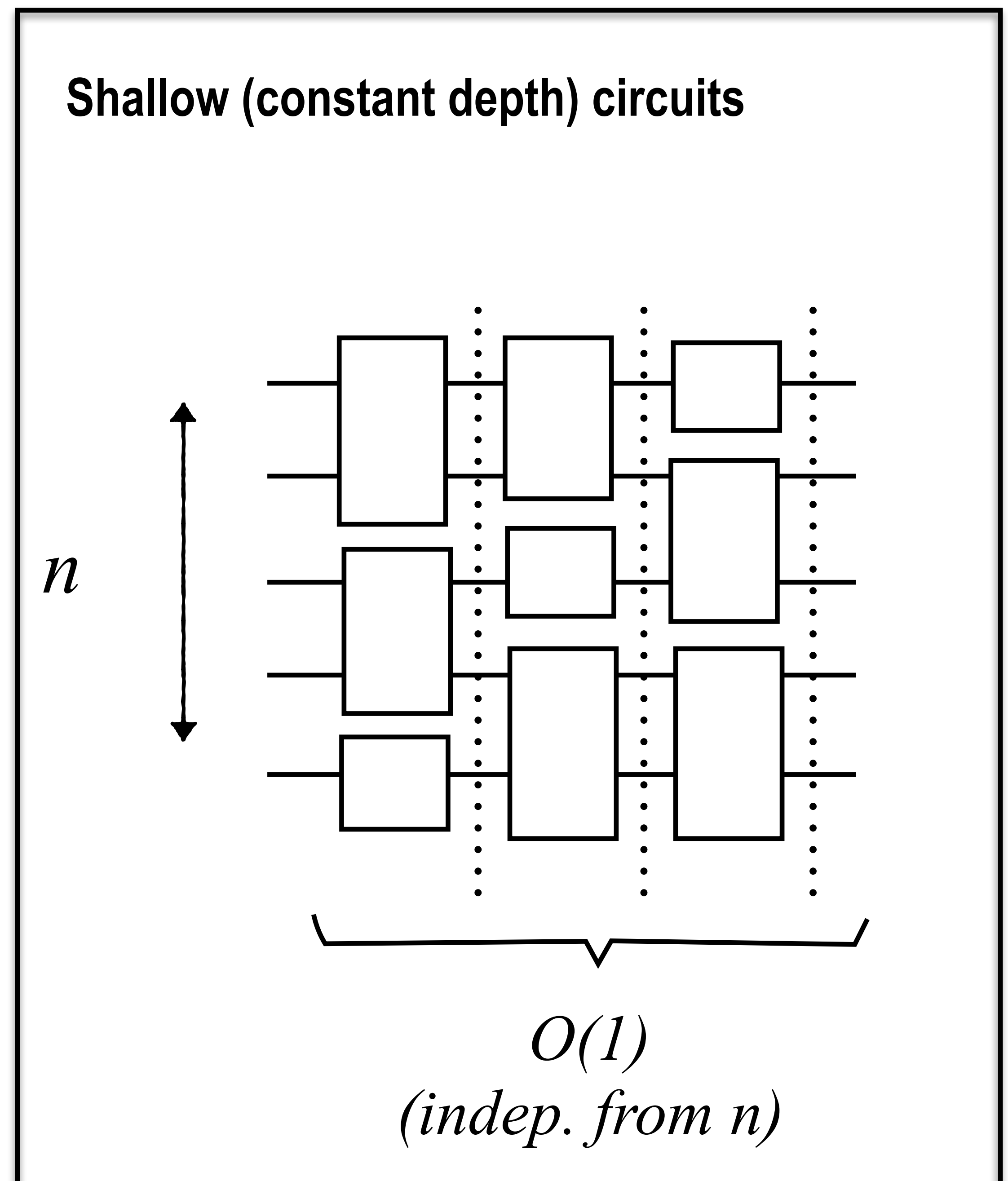


1) Limitations of real-world QCs

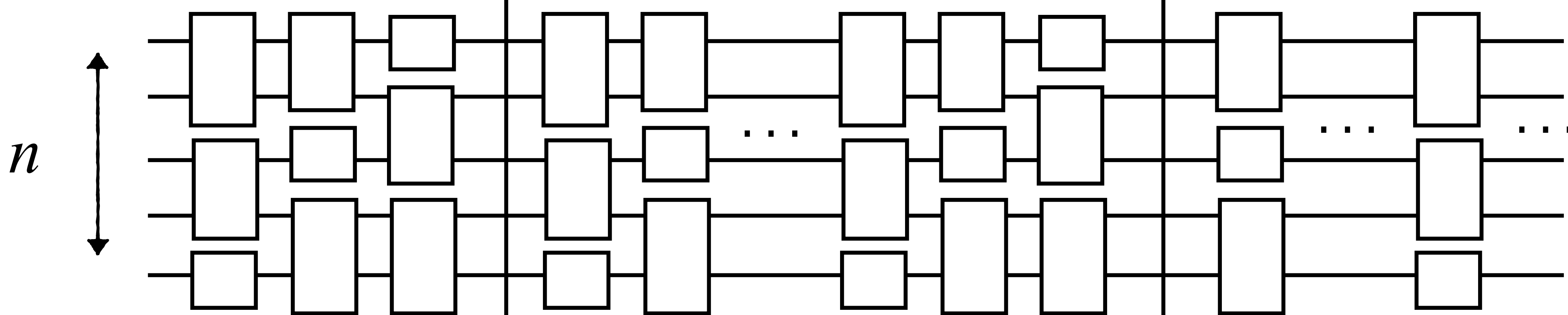
recall: **qubits have a life-time (half-life)…. up to *ms***
gates can be dozens of ns

Not yet in the *same system*… nowadays
whatever you can to in 10 - 100 (parallel) gate times





Tangent: ***Quantum depth complexity***



O(1)
(indep. from n)

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

-***likely*** better for sampling problems, <u>no matter what depth of classical computer</u>

-**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)

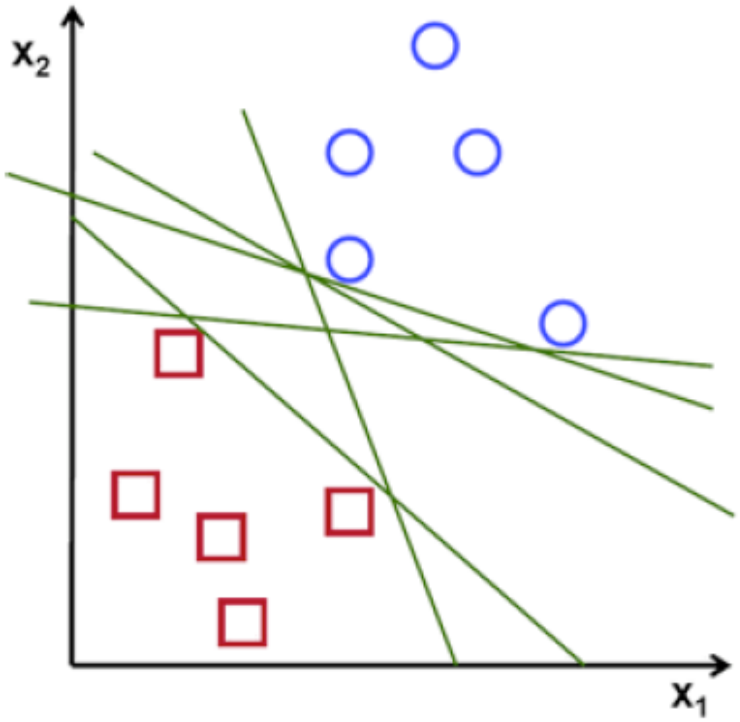
O(poly(n))

"BQP" = full QC

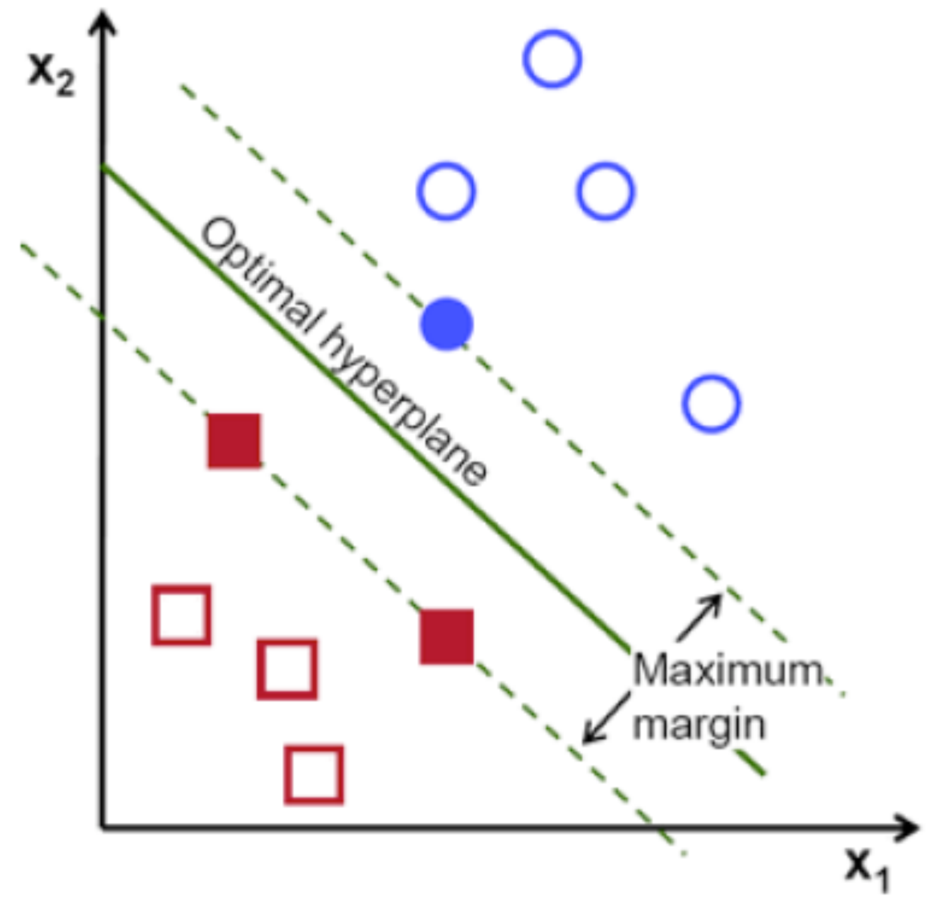
10.1126/science.aar3106

1.2) Support vector machines

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



separating hyperplanes
(linear classifier, not SVM)



SVM: max-margin hyperplanes

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

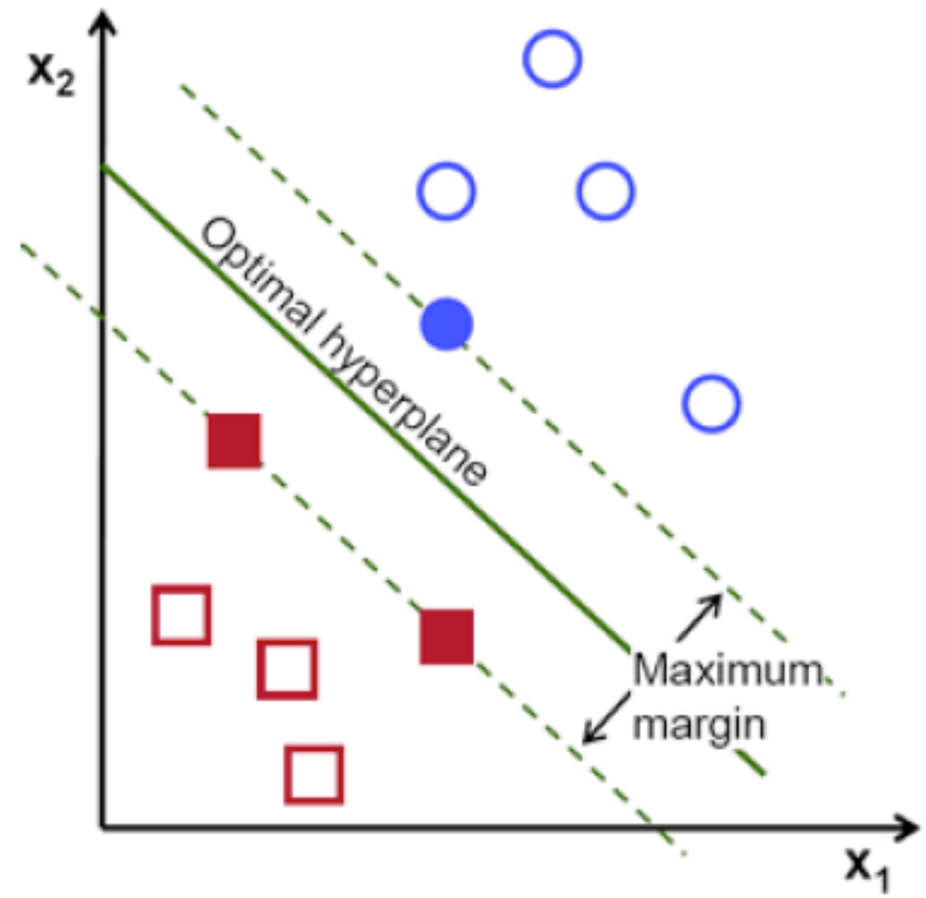
$$D = \{(x_i, y_i)\}_i \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$$

Quadratic problem:

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

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

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



SVM: max-margin hyperplanes

Note: defined on the basis of "*support vectors*"

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

Primal problem:

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

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

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

Dual problem:

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

such that $\alpha_i \geq 0, \quad$ for $i = 0, \ldots, N,$

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

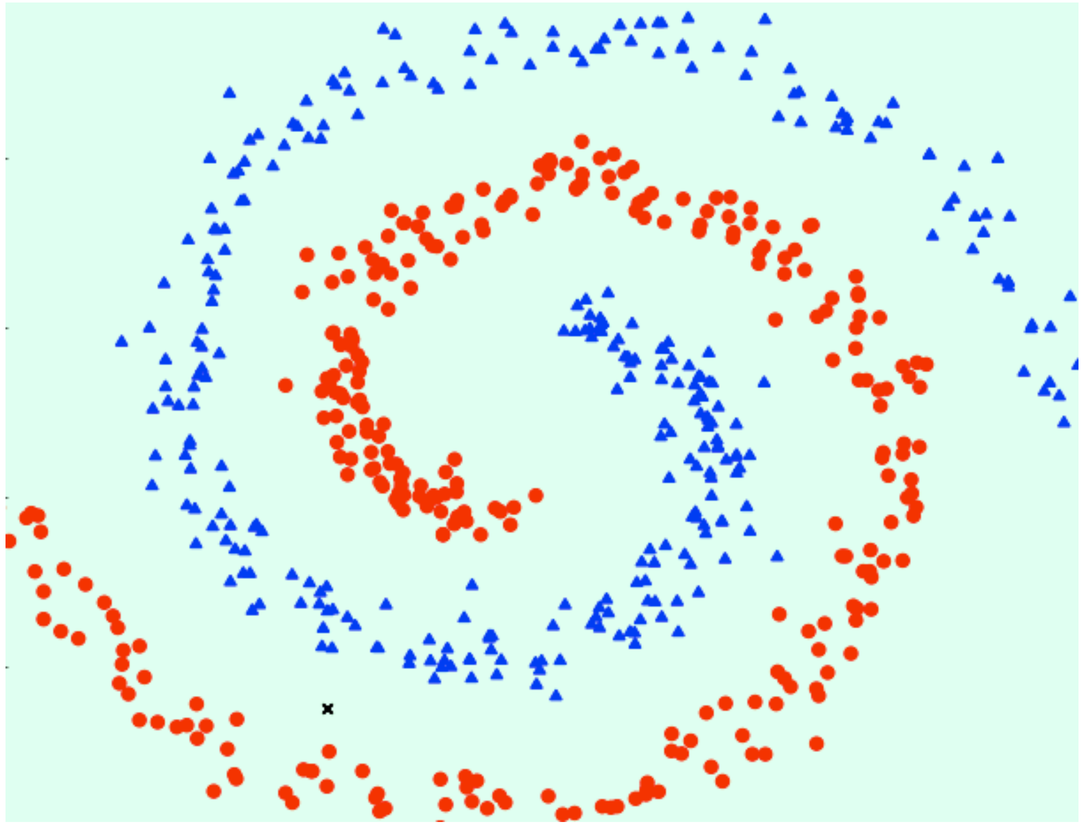
$$\mathbf{w} = \sum_{i=1}^{N}\alpha_i^* y_i \mathbf{x}_i.$$

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

- sparser evaluation $\quad (\mathbf{w}^*)^\mathsf{T}\mathbf{x} + b^* = \left(\sum_{i=1}^{N}\alpha_i y_i (\mathbf{x}_i)^\mathsf{T}\mathbf{x}\right) + b^*.$
- only inner products matter $\quad \alpha_i\alpha_j y_i y_j (\mathbf{x}_i)^\mathsf{T}\mathbf{x}_j,$
- was handy for *quantum tricks*

c.f. *Representer theorems*

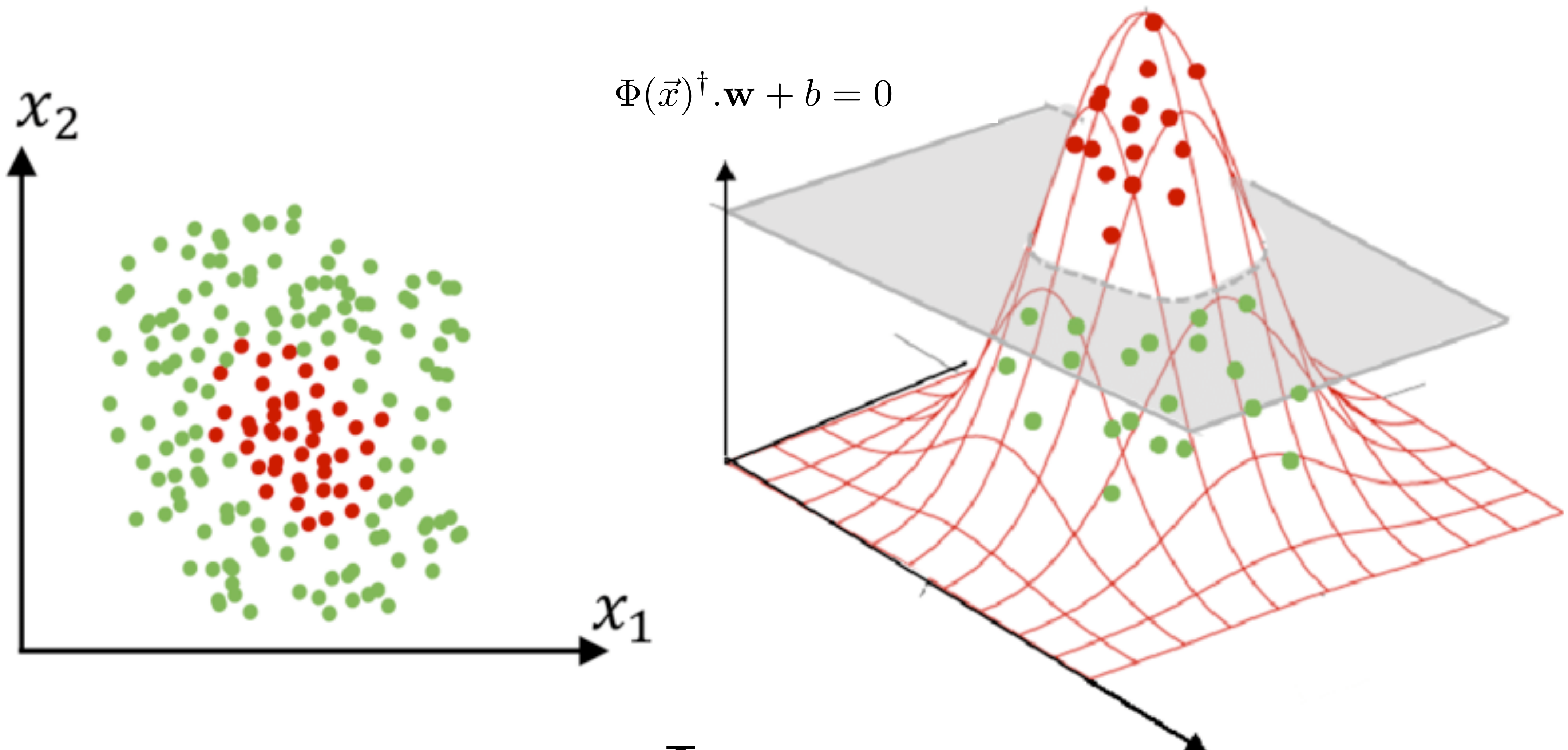
Why one should actually bother with SVMs:
when data is NOT linearly separable



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

Recall… only inner products matter: $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ $(\phi = \Phi \ldots)$

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

kernels can sometimes be evaluated (much) more efficiently directly:

E.g. (stupidly)

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

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \sum_{i=1}^{d} \sum_{j=1}^{d} x_i z_i x_j z_j$$

Runtime for $\phi(\mathbf{x})$: $\mathcal{O}(d^2)$

c.f. Mercer's theorem

The kernel trick

Recall… only inner products matter: $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ $(\phi = \Phi \ldots)$

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

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

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

c.f. Mercer's theorem

The kernel trick:

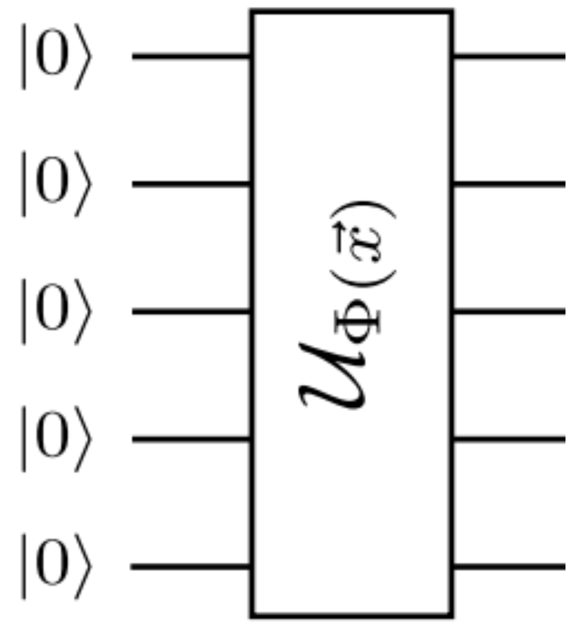
one can "train" and evaluate SVM classifiers in rich feature spaces without ever mapping data-points into said spaces. They can even be infinite dimensional

Feature maps matter,

and sometimes kernels are *not* efficiently computable…

Feature maps matter,

and sometimes kernels are *not* efficiently computable…



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

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

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

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

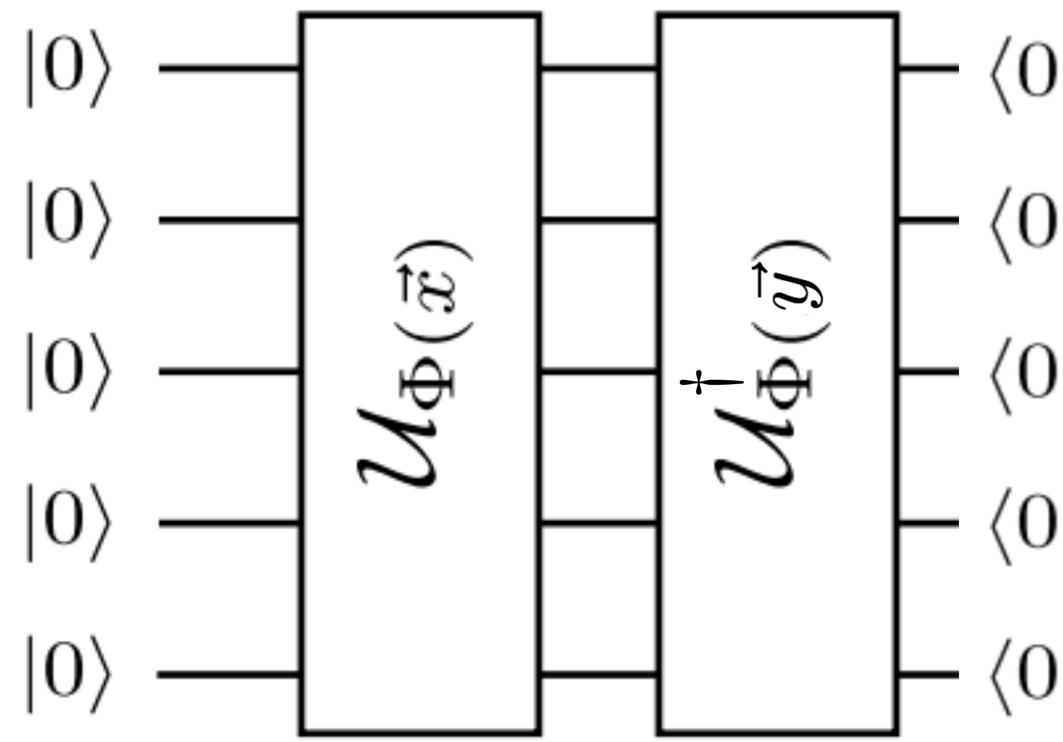
Feature maps matter,

and sometimes kernels are *not* efficiently computable…

Kernel!

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

Can be hard to compute.



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

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

Do this quantumly
(recall QC is good for inner products)

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

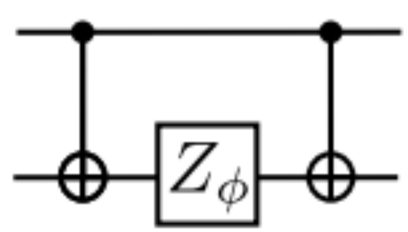
Feature maps matter,

and sometimes kernels are *not* efficiently computable…

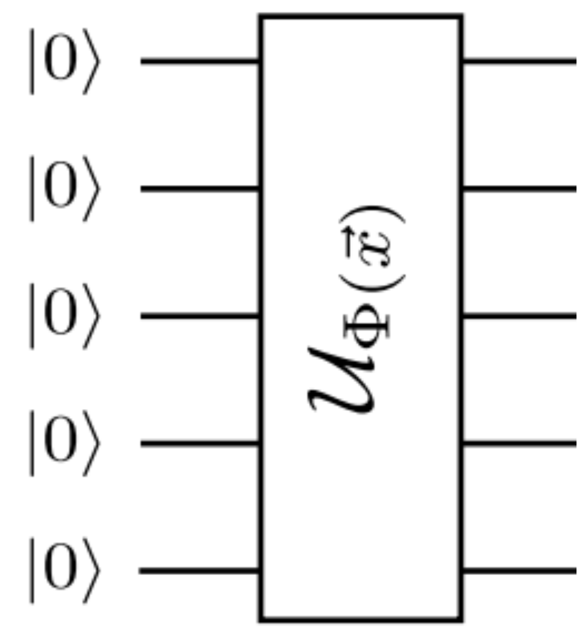
$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S\subseteq[n]} \phi_S(\vec{x}) \prod_{i\in S} Z_i\right)$$

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

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

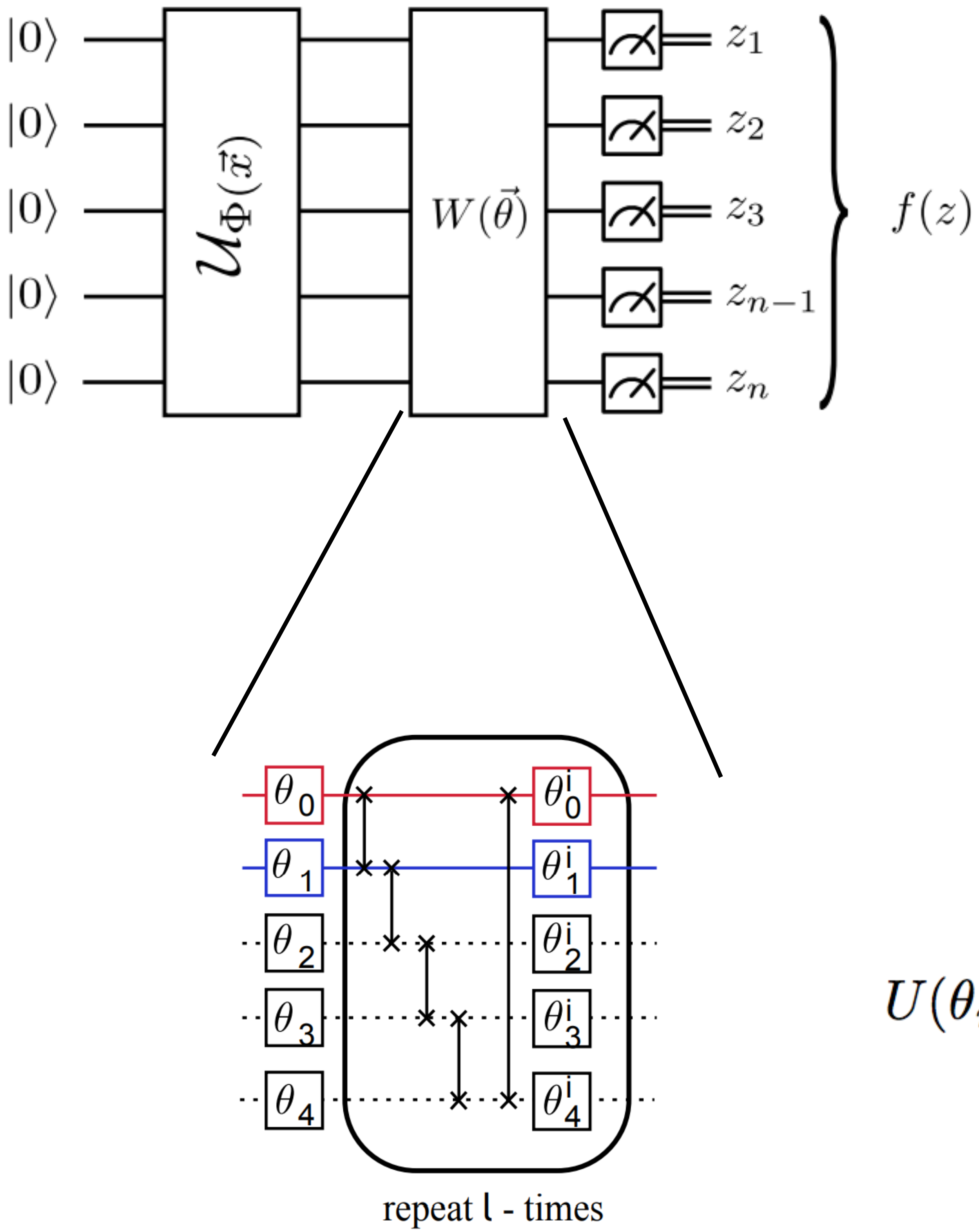


$$\mathcal{U}_\Phi = H^{\otimes n} U_\Phi H^{\otimes n} U_\Phi \cdots H^{\otimes n} U_\Phi$$



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

But there is also the fully quantum version:

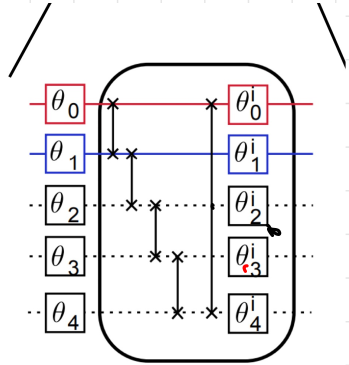


$$f(z) : \{0,1\}^n \to \{-1,1\}$$

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

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

BY THE WAY... CIRCUITS OF THIS TYPE



WHICH DEVIATE FROM THE DISCRETE GATESET $\{H, \pi/8, CNOT\}$

BUT UTILIZE (A NUMBER OF) CONTINUOUS PARAMETER ELEMENTS ARE CALLED

PARAMETRIZED OR VARIATIONAL CIRCUITS

THEY ARE EXPERIMENTALLY WELL-MOTIVATED

$$\exp(i H \Delta t) \rightarrow \quad U(\theta) = \exp(i H \theta) \ldots$$

How does it output a label?

$$label(\vec{y}) = \tilde{m}(\vec{x}) = \mathrm{sign}(\underbrace{\langle\Phi(\vec{x})|W^{\dagger}(\vec{\theta})\mathbf{f}W(\vec{\theta})|\Phi(\vec{x})\rangle}_{\text{involves running circuit many times}} + b)$$

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

How does it output a label?

$$label(\vec{y}) = \tilde{m}(\vec{x}) = \mathrm{sign}(\langle\Phi(\vec{x})|W^\dagger(\vec{\theta})\mathbf{f}W(\vec{\theta})|\Phi(\vec{x})\rangle + b)$$

How does it learn?

$\underbrace{\text{Optimize } \theta \text{ to minimize some loss/error/empirical risk on dataset}}$

Involves evaluation of label function many times…

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

How does it output a label?

$$label(\vec{y}) = \tilde{m}(\vec{x}) = \mathrm{sign}(\langle\Phi(\vec{x})|W^{\dagger}(\vec{\theta})\mathbf{f}W(\vec{\theta})|\Phi(\vec{x})\rangle + b)$$

How does it learn?

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

What does it *do?*

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

$$\Phi_{\alpha}(\vec{x}) = \langle\Phi(\vec{x})|P_{\alpha}|\Phi(\vec{x})\rangle$$

$$\tilde{m}(x) = \mathrm{sign}\left(2^{-n}\sum_{\alpha} w_{\alpha}(\vec{\theta})\Phi_{\alpha}(\vec{x}) + b\right)$$

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

How does it output a label?

$$label(\vec{y}) = \tilde{m}(\vec{x}) = \mathrm{sign}(\langle\Phi(\vec{x})|W^{\dagger}(\vec{\theta})\mathbf{f}W(\vec{\theta})|\Phi(\vec{x})\rangle + b)$$

How does it learn?

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

What does it *do?*

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

$$\Phi_{\alpha}(\vec{x}) = \langle\Phi(\vec{x})|P_{\alpha}|\Phi(\vec{x})\rangle$$

$$\tilde{m}(x) = \mathrm{sign}\left(2^{-n}\sum_{\alpha} w_{\alpha}(\vec{\theta})\Phi_{\alpha}(\vec{x}) + b\right)$$

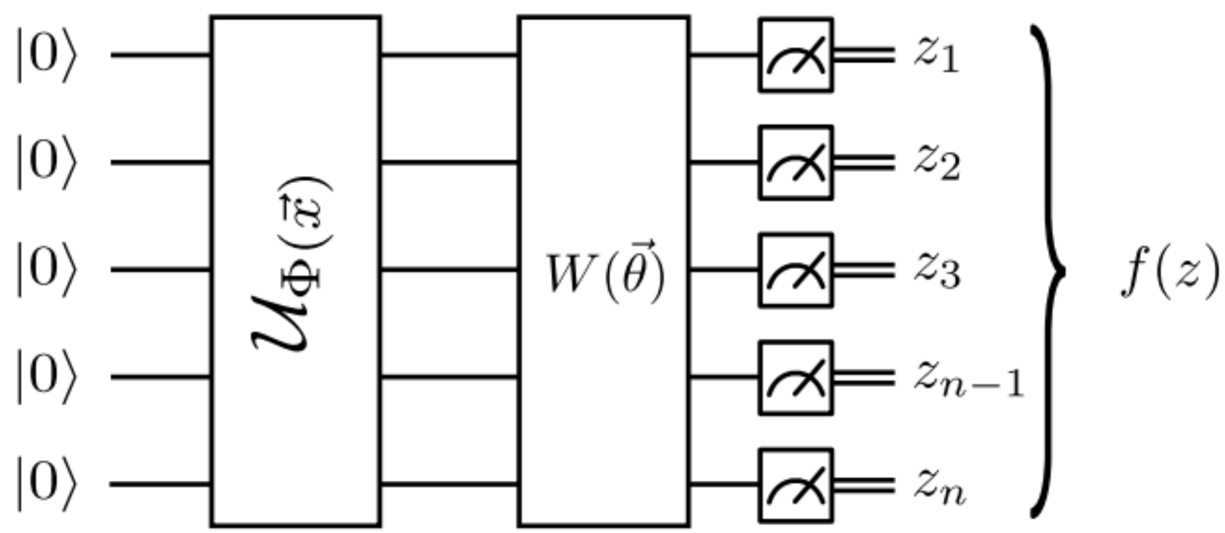
-limitations on the model come into play here…
-not *all hyperplanes* reachable…

-not maximal margin attained!

The group with this project will clarify this in report.

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

Summary:

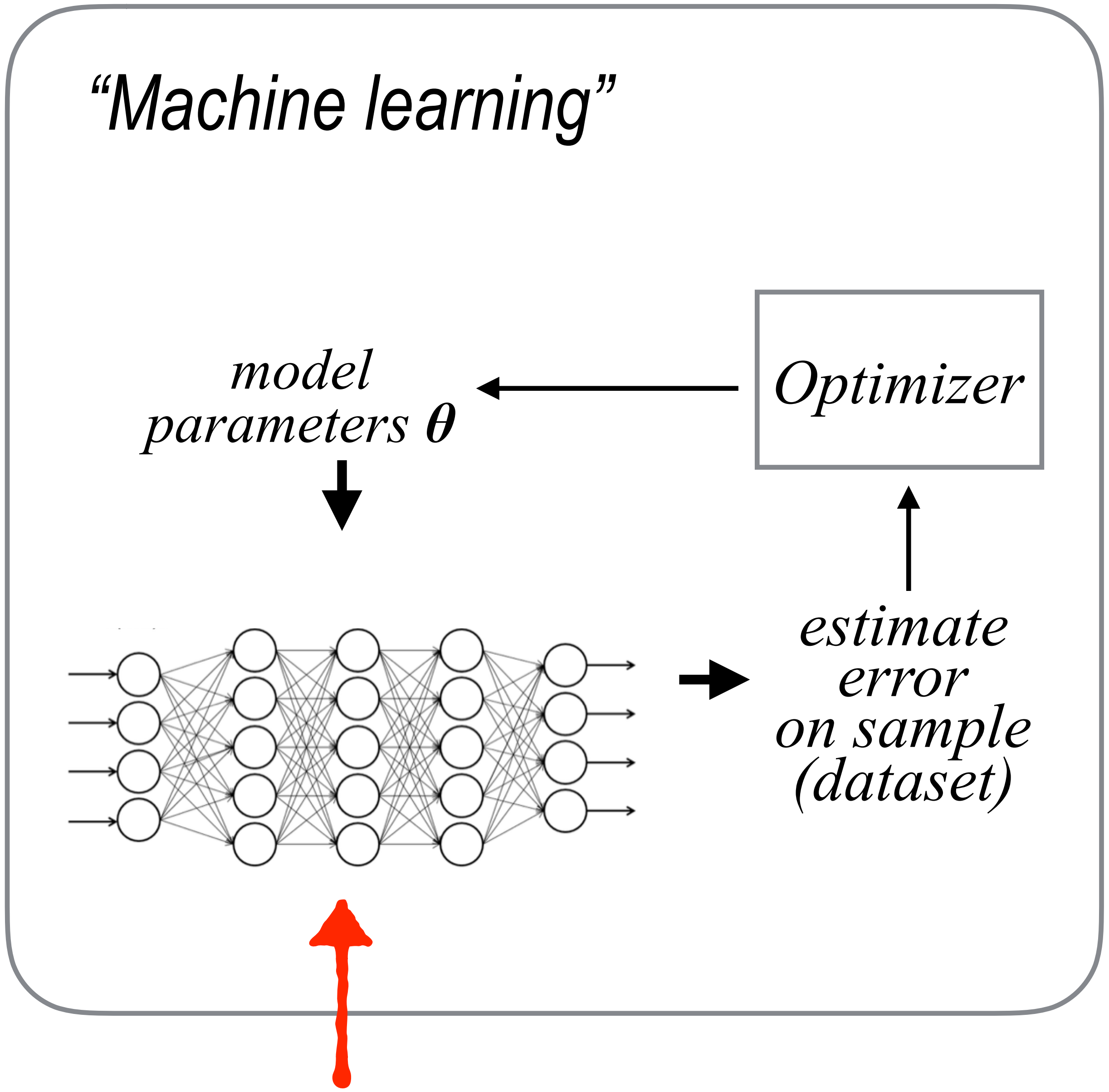


- -estimate probability of -1/1
- -take estimate of expected value
- -use this to label
- -loop optimizing θ on dataset

These are basic ideas, with (some) steps omitted.
The group with this project will report this precisely.

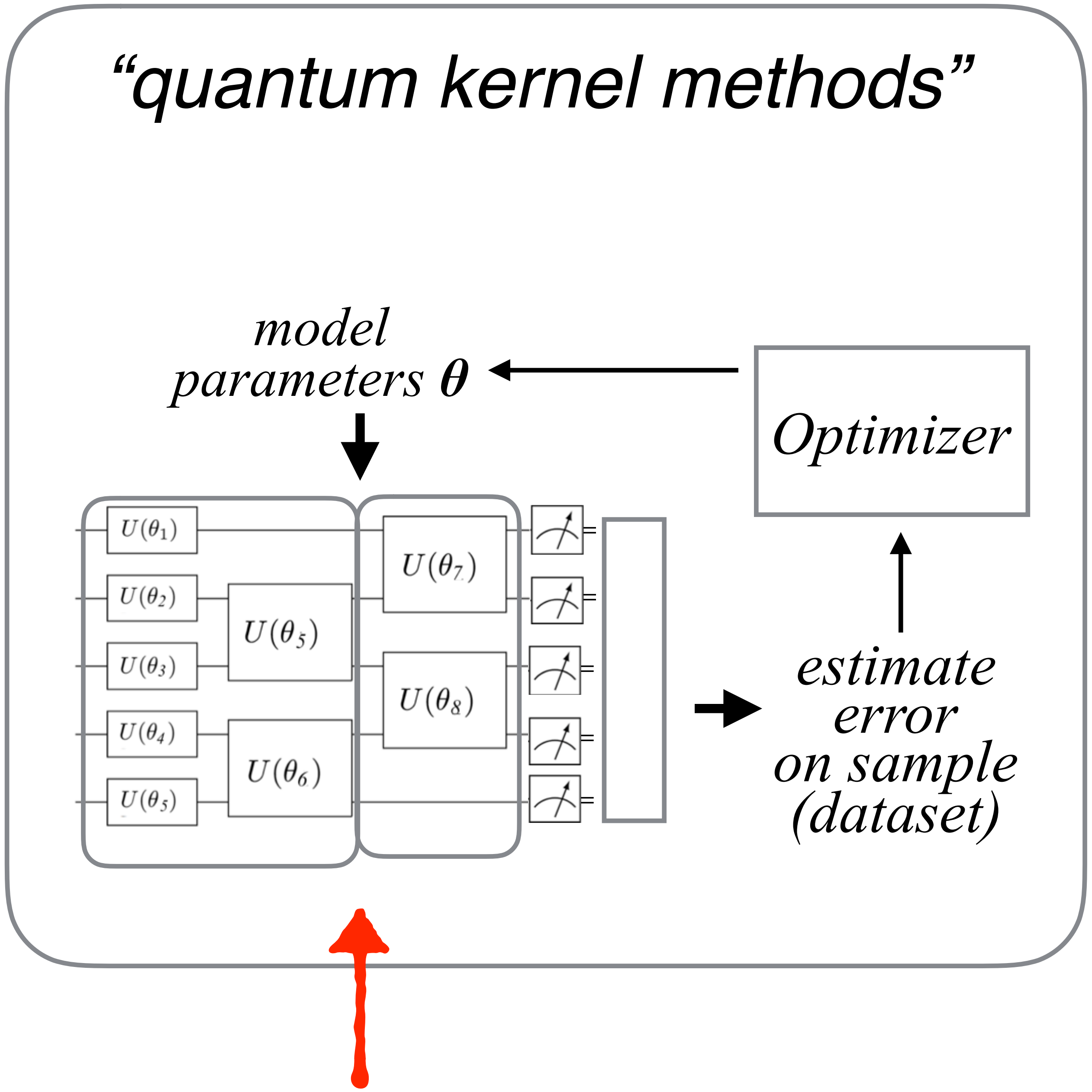
Nature. vol. 567, pp. 209-212 (2019)

Note this is much like training NNs or other general models



family of functions.
if it's "good", we can generalize well

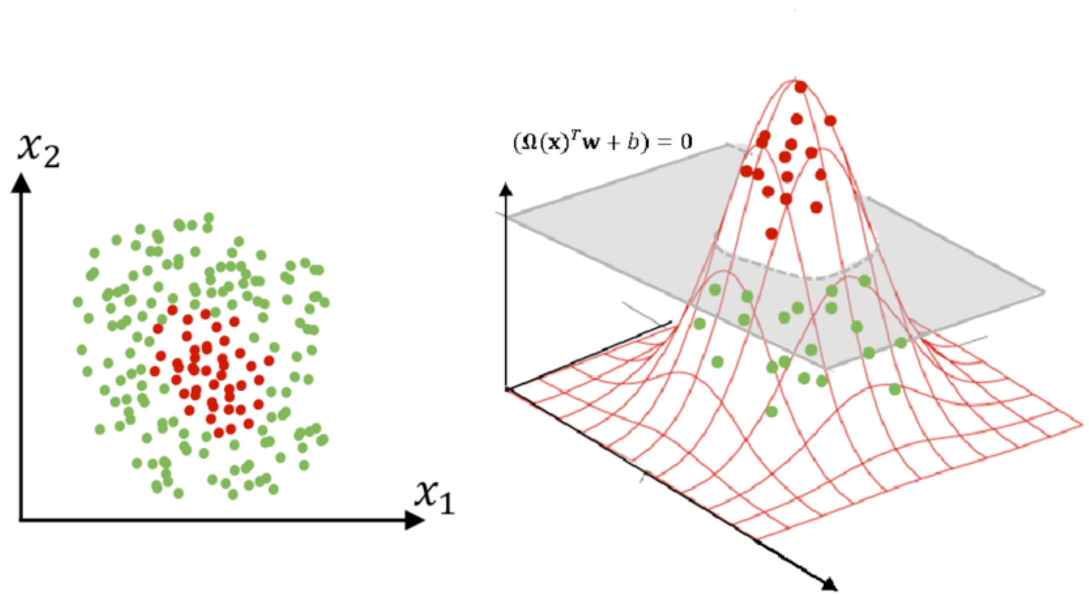
But you train a “quantum” network, without backprop, ofc.



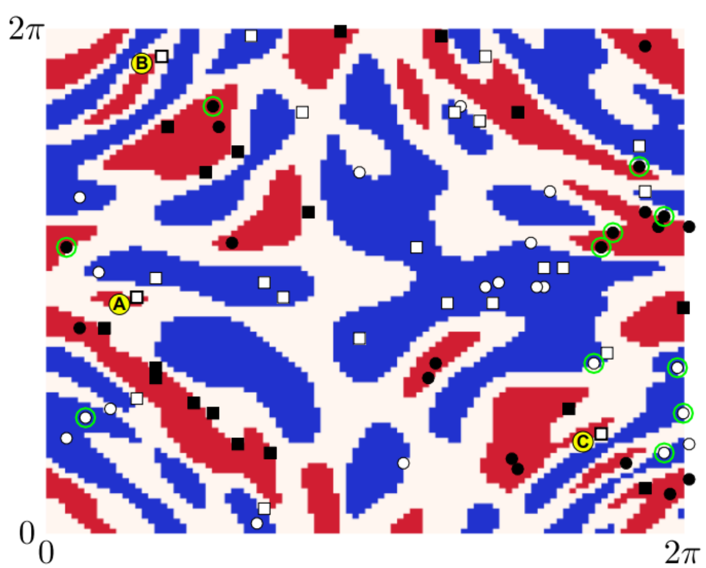
How about “shallow quantum circuits”?
-instead neural network, train a QC!
-related to ideas from q. condensed-matter physics (VQE)

Phys. Rev. Lett. 122, 040504 2019
Nature 567, 209–212 (2019)
(c.f. Elizabeth Behrman in ‘90s)

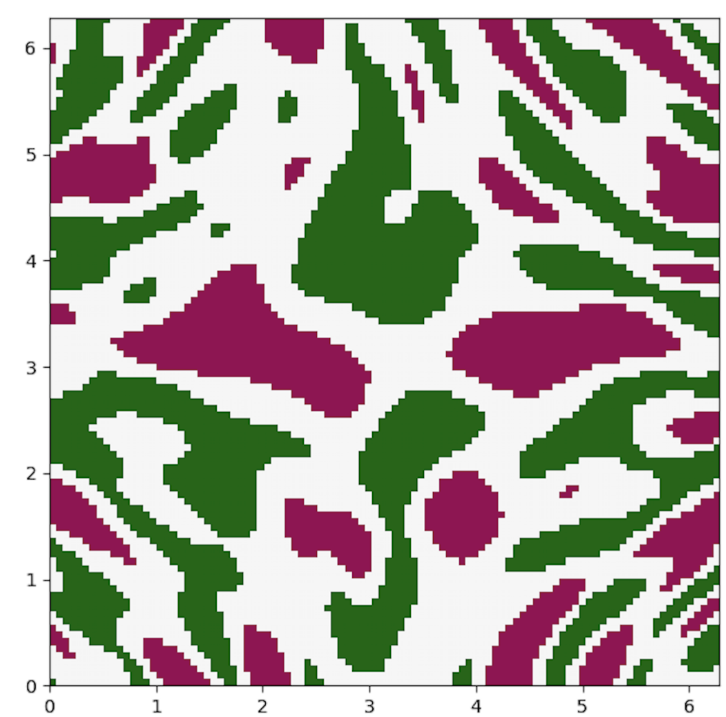
BUT it can be interpreted as SVM
So what does it do?



Two slices of quantum kernels:



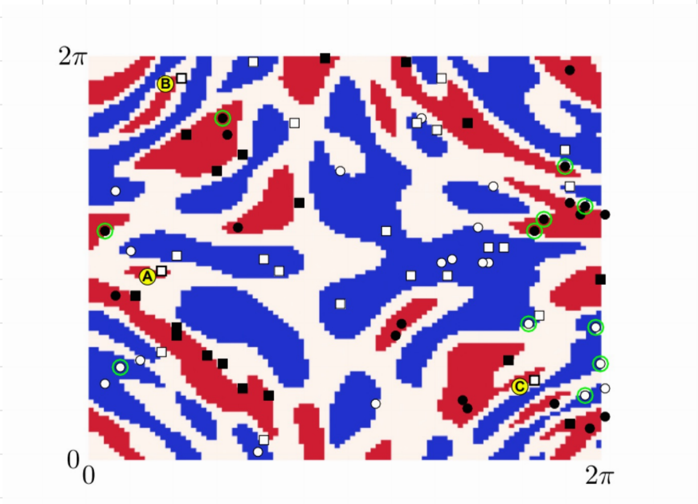
ORIGINAL PAPER

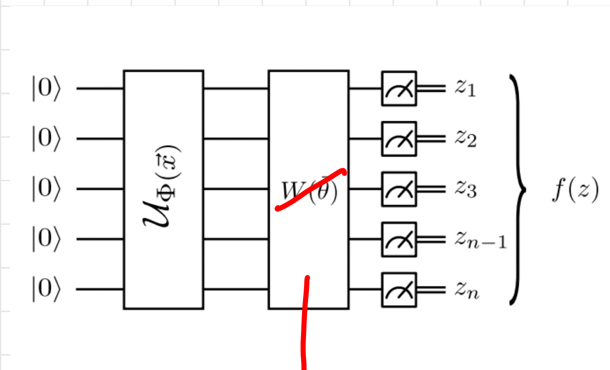


MSc THESIS MARDIROSIAN (LIACS)

- PERFORMANCE OF SUCH Q-KERNELS STUDIED IN A NUMBER OF WORKS

- IN ORIGINAL PAPER: GENERALIZATION PERFORMANCE ON ARTIFICIAL DATASETS





→ Point labels generated in the same way as classification

RANDOMLY CHOSEN UNITARY (BUT IMPLEMENTABLE!)

PROBABLY = CHOOSE θ RANDOMLY ..

100% correct classification...

Cost functions and optimization?

Noise tolerance!

Advantages?

Two models, back-to-back?

That's the question…

Supervised learning with quantum enhanced feature spaces

Havlicek, Córcoles, Temme, Harrow, Kandala, Chow, Gambetta

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