Introduction to Quantum Computing
Lecture 27: Quantum Machine Learning

Petros Wallden

School of Informatics, University of Edinburgh

14th November 2024

This Lecture

1. Intro to Machine Learning
2. What can Quantum bring to ML
3. Quantum Neural Networks
4. Classical and Quantum Kernels

Introduction to (Classical) Machine Learning

Quick Intro to Classical Machine Learning

- Disclaimer: basic intro targeted to non-CS students
- Material for understanding Quantum ML part

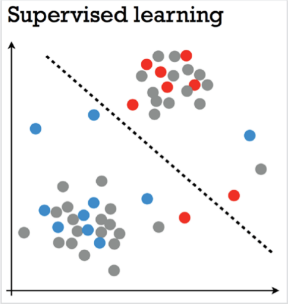
Quick Intro to Classical Machine Learning

- Disclaimer: basic intro targeted to non-CS students
- Material for understanding Quantum ML part
- There are mainly three models of ML (and combinations)
  1. Supervised
  2. Unsupervised
  3. Reinforcement Learning

Quick Intro to Classical Machine Learning

- Disclaimer: basic intro targeted to non-CS students
- Material for understanding Quantum ML part
- There are mainly three models of ML (and combinations)
  1. Supervised
  2. Unsupervised
  3. Reinforcement Learning
- Rest Intro: what (supervised, unsupervised), how (supervised)

Supervised ML: what



,

?

Supervised ML: what

- Encode to (feature) vectors $\vec{x} \in S \subseteq \mathbb{R}^n$
- Labels $y \in \mathrm{Labels}$
- Label function $f : S \to \mathrm{Labels}$
- Data set (training set): $D = \{(\vec{x}_i, y_i) \mid y_i = f(\vec{x}_i)\}$

Supervised ML: what

- Encode to (feature) vectors $\vec{x} \in S \subseteq \mathbb{R}^n$
- Labels $y \in \mathrm{Labels}$
- Label function $f : S \to \mathrm{Labels}$
- Data set (training set): $D = \{(\vec{x}_i, y_i) \mid y_i = f(\vec{x}_i)\}$

Aim: correctly label unlabelled data

Given D output a good guess for f

- Function f can be used for classification (discrete label) or regression (continuous label)

Supervised ML: what

- More generally (probabilistic – c.f. generative)
- Encode to (feature) vectors $\vec{x} \in S \subseteq \mathbb{R}^n$
- Labels $y \in \mathrm{Labels}$
- Label function $P(\vec{x}, y)$
- Data set (training set): $D \sim P^{\times|D|}$

Supervised ML: what

- More generally (probabilistic – c.f. generative)
- Encode to (feature) vectors $\vec{x} \in S \subseteq \mathbb{R}^n$
- Labels $y \in \mathrm{Labels}$
- Label function $P(\vec{x}, y)$
- Data set (training set): $D \sim P^{\times |D|}$

Aim: learning about data-label relationships from samples

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

- Can use sampling from $P(y|\vec{x})$ to label unseen data

Unsupervised ML: what

?

Unsupervised ML: what

- Encode to (feature) vectors $\vec{x} \in S \subseteq \mathbb{R}^n$
- World: $P(\vec{x})$
- Data set (training set): $D \sim P^{\times |D|}$

Unsupervised ML: what

- Encode to (feature) vectors $\vec{x} \in S \subseteq \mathbb{R}^n$
- World: $P(\vec{x})$
- Data set (training set): $D \sim P^{\times |D|}$

Aim: learning about (all) features in a distribution from samples

- Discriminative (clustering) "label without examples"
- Generative (make more cats):

  approximate sampling from P given D
- Promising quantumly (but not covered here)

Supervised ML: how (general)

- Need to guess $f : S \subseteq \mathbb{R}^n \to \mathrm{Labels}$ from $D = \{\vec{x}_i, y_i = f(\vec{x}_i)\}$
- Hypothesis family (model): $\{f^\theta | f^\theta : S \subseteq \mathbb{R}^n \to \mathrm{Labels}\}$

Supervised ML: how (general)

- Need to guess $f : S \subseteq \mathbb{R}^n \to \mathrm{Labels}$ from $D = \{\vec{x}_i, y_i = f(\vec{x}_i)\}$
- Hypothesis family (model): $\{f^\theta | f^\theta : S \subseteq \mathbb{R}^n \to \mathrm{Labels}\}$

Learning = Training ≈ fitting

- "Loss" / "accuracy" e.g. $L(\theta) = \sum_{(\vec{x},y)\in D} |f^\theta(\vec{x}) - y|^2$
- Regularisation $R(\theta)$: to prevent overfitting (favour fewer non-zero/significant parameters)

Supervised ML: how (general)

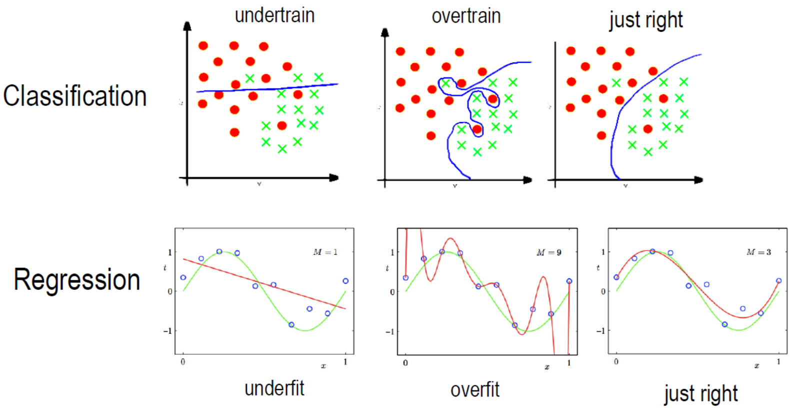
- Need to guess $f : S \subseteq \mathbb{R}^n \to \text{Labels}$ from $D = \{\vec{x}_i, y_i = f(\vec{x}_i)\}$
- Hypothesis family (model): $\{f^\theta | f^\theta : S \subseteq \mathbb{R}^n \to \text{Labels}\}$

Learning = Training ≈ fitting

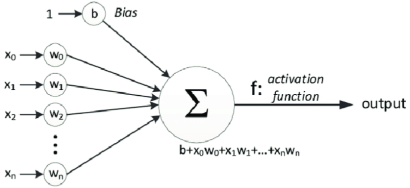
- "Loss" / "accuracy" e.g. $L(\theta) = \sum_{(\vec{x},y)\in D} |f^\theta(\vec{x}) - y|^2$
- Regularisation $R(\theta)$: to prevent overfitting (favour fewer non-zero/significant parameters)
- Find the function from the family that is best for prediction given the data D:

$$f^\theta \mid \arg\min_\theta (L(\theta) + R(\theta))$$

Supervised ML: how (general)

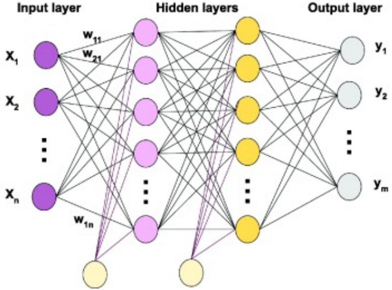


Supervised ML: how (Perceptron)



- Perceptron: Inputs $\vec{x}$, output given by $f(\vec{x}) = h(\vec{w} \cdot \vec{x} + b)$
  - $\vec{w} \cdot \vec{x}$ dot product, where $\vec{w}$ weights /trainable parameters
  - b bias
  - $h(\cdot)$ activation function (e.g. heaviside step-function)
- linearly combines inputs with some weight, adds bias, and then activates neuron or not (depending on threshold)

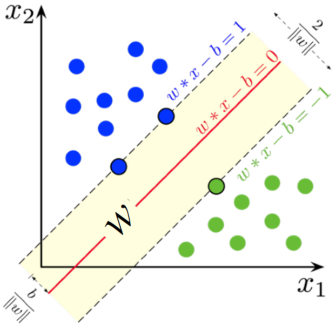
- Combine perceptrons $\rightarrow$ Neural Network



- Input layer: encoding data to input vector
- Training: find $\vec{w}_i, b_i\ \forall i \in \mathrm{NN}$, that min regularised loss

  Optimisation (chain-rule based stochastic gradient decent)
- Classification: input unseen $\vec{x}$ to trained NN to output label

Supervised ML: how (Support Vector Machines)



- Assume data linearly separable
- $D = \{(x_i, y_i)\} \mid x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}$
- Optimal hyperplane given by

$$\arg\max_{\vec{w}, b} \min_{i \in \{1, \cdots, N\}} \frac{y_i(\vec{w}^T \cdot x_i + b)}{\|\vec{w}\|}$$

Supervised ML: how (SVM)

- Points closer and equidistant to hyperplane: determine classification (support vectors)

Lagrangian approach

Primal problem:

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

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

Dual problem:

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

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

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

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

- Dual Problem:
  - Representation in terms of datapoints
  - Sparser evaluation (many α's vanish)
  - Only inner products matter: $\alpha_i \alpha_j y_i y_j (x_i)^T x_j$

Supervised ML: how (SVM)

- Points closer and equidistant to hyperplane: determine classification (support vectors)

Lagrangian approach

Primal problem:

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

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

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

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

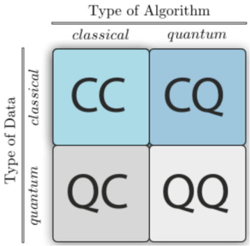
$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i^* y_i \mathbf{x}_i.$$

- Dual Problem:
  - Representation in terms of datapoints
  - Sparser evaluation (many α's vanish)
  - Only inner products matter: $\alpha_i\alpha_j y_i y_j (x_i)^T x_j$
- When non-linearly separable?

  (see later Feature Maps and Kernel trick)

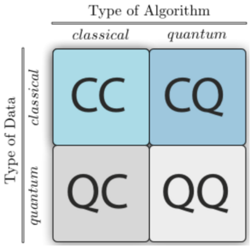
What can Quantum bring to Machine Learning?

What could quantum offer to ML? (general)



- Quantum algorithms that speed-up (classical) ML
  - Grover's/amplitude amplification (perceptron training/computation of attention)
  - VQAs (optimisation subroutines/training)

What could quantum offer to ML? (general)

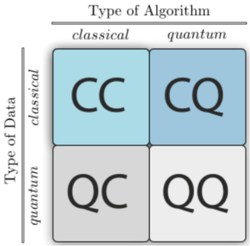


- Quantum algorithms that speed-up (classical) ML
  - Grover's/amplitude amplification (perceptron training/computation of attention)
  - VQAs (optimisation subroutines/training)
  - HHL algorithm: Exponential advantage in linear algebra task

    Given $A|x\rangle = |b\rangle$ can efficiently (log-time) find state $|x\rangle$ that encodes in the amplitudes the solution

    But: need to encode vector $|b\rangle$; A needs to be sparse and well conditioned; readout summary $\langle x| M |x\rangle$ should suffice

What could quantum offer to ML? (general)



- Quantum algorithms that speed-up (classical) ML
  - Grover's/amplitude amplification (perceptron training/computation of attention)
  - VQAs (optimisation subroutines/training)
  - HHL algorithm: Exponential advantage in linear algebra task

    Given $A|x\rangle = |b\rangle$ can efficiently (log-time) find state $|x\rangle$ that encodes in the amplitudes the solution

    But: need to encode vector $|b\rangle$; A needs to be sparse and well conditioned; readout summary $\langle x| M |x\rangle$ should suffice
- New Models/Quantum Neural Networks (QNN)

What could QNN offer?

1. Expressivity

Quantum circuits can efficiently sample from probability distributions that cannot be sampled classically efficiently

- For example "quantum advantage" sampling problems

What could QNN offer?

1. Expressivity

Quantum circuits can efficiently sample from probability distributions that cannot be sampled classically efficiently

- For example "quantum advantage" sampling problems

2. Accuracy

There are problems that quantum models (QNN) can fit easier, with fewer parameters

- Specifically, systems that physically or mathematically resemble quantum systems (quantum-like)

What could QNN offer?

3. Generalisation

Quantum models could predict better unseen data

- For systems quantum-like systems simpler models fit the data giving better generalisation

What could QNN offer?

3. Generalisation

Quantum models could predict better unseen data

- For systems quantum-like systems simpler models fit the data giving better generalisation

4. Speed

Training and/or inferences could be performed faster (generically, but also Quantum Kernel Methods – see later)

What could QNN offer?

3. Generalisation

Quantum models could predict better unseen data

- For systems quantum-like systems simpler models fit the data giving better generalisation

4. Speed

Training and/or inferences could be performed faster (generically, but also Quantum Kernel Methods – see later)

5. Energy efficiency

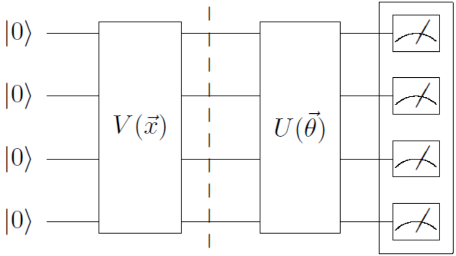
Training may not be as energy demanding for comparable performances

Obstacles

- Not known proofs of the above advantages in practice
- Trainability Vs Expressivity
- Barren Plateaux/Vanishing Gradients
- Classical Simulation (of circuit or model)
- Noise (for NISQ era)

Quantum Neural Networks

Quantum Neural Networks



1. **Encoding** the (classical) data on a quantum state

   $|\Phi(\vec{x})\rangle := V(\vec{x})\,|0\rangle^n$

2. **Variational Circuit** $U(\vec{\theta})$ with trainable parameters $\vec{\theta}$

3. **Output** $f(\langle\vec{z}\rangle)$: repeat multiple times; each time obtain bit-sting $\vec{z}$; average $\langle\vec{z}\rangle$; compute f activation-function

QNN: Encoding

- Input is n-bit string: $\vec{x} = x_{n-1}x_{n-2}\cdots x_0$
- **Basis Encoding**: We have n qubits:
  $|\Phi_B(\vec{x})\rangle := |x_{n-1}x_{n-2}\cdots x_0\rangle$
- **Amplitude Encoding**: We have $m = \log n$ qubits denoting the position in the bit string, where the value of the bit is encoded in the amplitude:

  $|\Phi_A(\vec{x})\rangle := \frac{1}{|\vec{x}|}\sum_{i=0}^{m-1} x_i\,|i\rangle$
- Any other function/unitary $|\Phi(\vec{x})\rangle = V(\vec{x})\,|0\rangle$

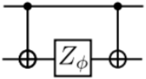
QNN: Encoding

- ZZ **Feature Map**:

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

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

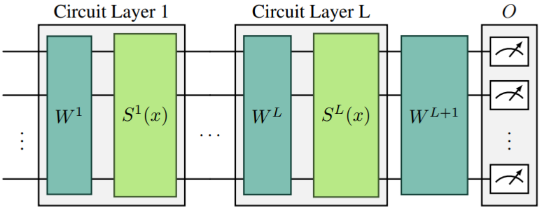
$$e^{i\phi_{\{l,m\}}(\vec{x}) Z_l Z_m} =$$



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

QNN: Training & Output

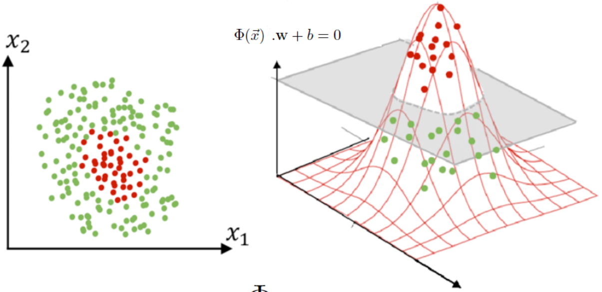
- Optimisation to find parameters $\vec{\theta}$ that minimise the loss:
  $\langle\Phi(\vec{x})| U^{\dagger}(\vec{\theta}) f(z) U(\vec{\theta}) |\Phi(\vec{x})\rangle$
- Once parameters are fixed, can use the quantum circuit for inference
- Other models are possible (e.g. "data re-uploading")



Classical and Quantum Kernels

SVM and Classical/Quantum Kernels

- When non-linearly separable can use a "Feature Map" to a higher dimensional space that they become linearly separable



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

- Hard to work on higher dimensional feature space
- **Kernel Trick**: Can train and evaluate SVM without mapping data points there. Only inner products matter!

- Dual formulation of SVM becomes

$$\arg\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle$$

where $K(x_i, x_j) := \langle \phi(x_i), \phi(x_j) \rangle$ is the Kernel

- Dual formulation of SVM becomes

$$\arg\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle$$

  where $K(x_i, x_j) := \langle \phi(x_i), \phi(x_j) \rangle$ is the Kernel

- Only dependence on data (x's) comes from the Kernel, which is defined as the inner product between "encoded" inputs

- Dual formulation of SVM becomes

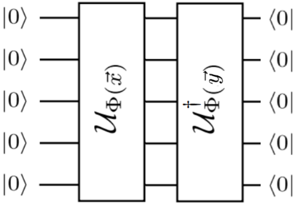
$$\arg\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \langle \phi(x_i), \phi(x_j) \rangle$$

  where $K(x_i, x_j) := \langle \phi(x_i), \phi(x_j) \rangle$ is the Kernel

- Only dependence on data (x's) comes from the Kernel, which is defined as the inner product between "encoded" inputs
- Quantumly is easy to perform inner products!
- Feature Maps = Data Encodings

Classical and Quantum Kernels

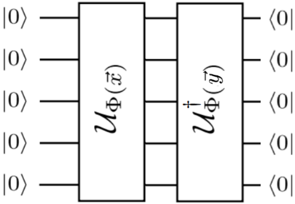
- Inner product is easy quantumly: $|\langle\Phi(\vec{y})|\Phi(\vec{x})\rangle|^2$



- If the inner product is unity, then always get zero's
- Can also measure overlap using the Hadamard-test idea

Classical and Quantum Kernels

- Inner product is easy quantumly: $|\langle\Phi(\vec{y})|\Phi(\vec{x})\rangle|^2$



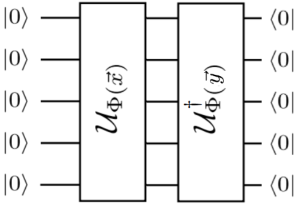
- If the inner product is unity, then always get zero's
- Can also measure overlap using the Hadamard-test idea
- $|\Phi(\vec{x})\rangle = V(\vec{x})|0\rangle)$ data encoding $\rightarrow$ model param. cancel!

$$K(\vec{y},\vec{x}) = |\langle 0| V^\dagger(\vec{y}) U^\dagger(\theta) U(\theta) V(\vec{x}) |0\rangle|^2 =$$

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

Classical and Quantum Kernels

- Inner product is easy quantumly: $|\langle\Phi(\vec{y})|\Phi(\vec{x})\rangle|^2$



- If the inner product is unity, then always get zero's
- Can also measure overlap using the Hadamard-test idea
- $|\Phi(\vec{x})\rangle = V(\vec{x})|0\rangle)$ data encoding $\rightarrow$ model param. cancel!

$$K(\vec{y},\vec{x}) = |\langle 0| V^\dagger(\vec{y})U^\dagger(\theta)U(\theta)V(\vec{x})|0\rangle|^2 =$$

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

- Can do classical SVM using a Kernel computed with quantum feature maps!

References

Quantum Machine Learning Reviews

1. *Machine learning & artificial intelligence in the quantum domain: a review of recent progress*, Dunjko, Briegel, (2018) Rep. Prog. Phys. 81 074001
2. *Quantum machine learning*, Biamonte et al, (2017) Nature **549**, pages 195–202.
3. *Systematic literature review: Quantum machine learning and its applications*, Peral-Garcia, et al (2024) Computer Science Review **51**, 100619.