Data Encoding and Expressivity

Variational Quantum Algorithms · Module 8 — Expressivity Vs Trainability

Node 8.6

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Expressivity vs Trainability **Node:** 8.6

So far, Module 8 has treated expressivity in terms of *parameter* dependence: how many states the ansatz can reach as $\boldsymbol{\theta}$ varies. This node generalizes to the **machine-learning setting**, where the ansatz also depends on a *data input* x.

In quantum machine learning (QML), the circuit becomes $U(x, \boldsymbol{\theta})$ — a function of both data and trainable parameters. The expressivity questions are now:

1. **What functions $f(x)$ can the model express** as x varies?
2. **How does the choice of data encoding affect the function class?**
3. **Is there a tradeoff between data-encoding expressivity and trainability** analogous to the parameter-side tradeoff?

The answers reveal that data encoding is *the* key design choice in QML — more important than the parameterized part. The encoding determines the function class entirely; the trainable part only selects within it.

This node walks through the major encoding strategies, the Schuld-Sweke-Meyer Fourier characterization, and the implications for QML model design.

Why Data Encoding Matters

In classical ML, data is fed to a neural network via a single input layer. The network's expressivity grows with depth and width regardless of how the input was fed in (modulo choice of features).

In quantum ML, **the encoding is part of the model**. Different encoding choices give *fundamentally different* function classes, even with identical trainable layers. There is no "universal feature encoding" that lets the trainable layers do all the work.

The reason: a quantum state $|\psi(x)\rangle$ is a *nonlinear* function of x (because of the cosines and sines in the rotation gates). The set of functions you can express as expectation values $\langle\psi(x)|M|\psi(x)\rangle$ is constrained by what kinds of nonlinearities the encoding produces. Trainable parameters can mix and combine these nonlinearities, but they can't introduce nonlinearities that aren't already in the encoding.

So **encoding choice = function class**. Get it wrong and no amount of training will help.

The Major Encoding Strategies

The literature has converged on a few standard strategies.

1. Basis encoding

$$x \in \{0,1\}^n \to |x\rangle.$$

Each bit of x becomes a computational basis state. Simple but only works for binary inputs.

Pros: trivial to implement; preserves all information. **Cons:** doesn't generalize to continuous inputs; doesn't introduce useful nonlinearities.

2. Angle encoding

$$x \in \mathbb{R}^n \to \bigotimes_{i=1}^{n} R_Y(x_i)|0\rangle.$$

Each component of x becomes a rotation angle on a separate qubit. The state is a product of single-qubit rotations.

Pros: simple, continuous-friendly, hardware-friendly. **Cons:** the function class is restricted to first-order Fourier modes (linear in $\cos x_i, \sin x_i$).

3. Amplitude encoding

$$x \in \mathbb{R}^{2^n}, \|x\| = 1 \to \sum_i x_i |i\rangle.$$

The data vector becomes the amplitudes of a quantum state. **Exponential compression** — 2^n dimensional vectors fit in n qubits.

Pros: maximally compact, supports arbitrary inputs. **Cons:** state preparation is expensive — $O(2^n)$ gates in general. Often a bottleneck.

4. Re-uploading encoding

$$x \in \mathbb{R}^d \to U_L(\boldsymbol{\theta}_L) S(x) U_{L-1}(\boldsymbol{\theta}_{L-1}) S(x) \cdots S(x) U_0(\boldsymbol{\theta}_0)|0\rangle,$$

where $S(x)$ is a data-encoding layer (e.g., a tensor product of $R_Y(x_i)$) repeated L times, interleaved with trainable layers $U_l(\boldsymbol{\theta}_l)$.

Pros: the function class grows polynomially with L — Fourier modes up to degree L become accessible. Universal function approximator in the limit $L \to \infty$. **Cons:** depth scales with desired Fourier degree.

5. Hamiltonian encoding

$$x \in \mathbb{R} \to e^{-ixH}|0\rangle,$$

where H is a problem-specific Hamiltonian. The data is encoded as evolution time.

Pros: physically natural, exploits quantum dynamics directly. **Cons:** Hamiltonian needs to be implementable; expressivity depends on H's spectrum.

The dominant choice in 2026 QML is **re-uploading**, because it gives the most controllable expressivity. Angle encoding is the simplest baseline.

The Schuld-Sweke-Meyer Theorem (Detailed)

The most important theoretical result about data encoding is the **Fourier characterization** of re-uploading models.

Setup. Consider a 1D input $x \in \mathbb{R}$ and a re-uploading model

$$f(x; \boldsymbol{\theta}) = \langle 0|U^\dagger(x, \boldsymbol{\theta}) M U(x, \boldsymbol{\theta})|0\rangle,$$

where the data is encoded via e^{-ixH} gates with Hamiltonian H, and there are L such encoding gates interleaved with trainable layers.

Theorem (Schuld-Sweke-Meyer 2021). $f(x; \boldsymbol{\theta})$ is a *truncated Fourier series* in x:

$$f(x; \boldsymbol{\theta}) = \sum_{\omega \in \Omega} c_\omega(\boldsymbol{\theta}) e^{i\omega x},$$

where the **frequency set** Ω is determined by the spectrum of H and the depth L, and the **Fourier coefficients** $c_\omega(\boldsymbol{\theta})$ are determined by the trainable circuits.

Frequency set details: if H has eigenvalues $\{\lambda_i\}$, then Ω is the set of integer linear combinations of differences $\lambda_i - \lambda_j$ with at most L terms.

Implications:

1. **The encoding determines what frequencies are reachable.** Different H gives different Ω.
2. **The depth controls the Fourier degree.** L layers give frequencies up to roughly degree L.
3. **The trainable parameters control coefficients, not frequencies.** No amount of training will give you a frequency that isn't in Ω.
4. **The function class is fully characterized.** Two re-uploading models with the same encoding and depth have the same function class — only the coefficient flexibility may differ.

This is the cleanest theoretical result in QML expressivity, and it reframes data encoding as **frequency-set design**: pick the encoding that gives you the frequencies you need.

Multi-Dimensional Inputs

For $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, the Fourier characterization generalizes:

$$f(\mathbf{x}; \boldsymbol{\theta}) = \sum_{\boldsymbol{\omega} \in \Omega^d} c_{\boldsymbol{\omega}}(\boldsymbol{\theta}) e^{i\boldsymbol{\omega} \cdot \mathbf{x}},$$

where Ω^d is now a *set of frequency vectors* rather than scalars.

The structure of Ω^d depends on:

1. Whether the data components are encoded in **the same qubit** (in which case the frequencies couple) or **different qubits** (in which case the frequencies separate).
2. The **depth** of the re-uploading.
3. The **mixing** of data components in the trainable layers (a non-trivial entanglement layer between encoding gates can mix frequencies that would otherwise be separable).

This multi-dimensional case is more complex and less fully characterized than the 1D case, but the qualitative picture is the same: **encoding determines what's expressible; trainable parameters fine-tune within that space**.

The Encoding-Trainability Tradeoff

There is an analogue of the parameter-side tradeoff (8.5) for the data-encoding side.

Statement (informal). Increasing the data-encoding depth gives a larger Fourier set (more expressivity) but, all else equal, also brings the model closer to the barren plateau regime in the parameters. So there is a similar tradeoff: more data-encoding depth = more function expressivity = harder to train.

Quantitative version. Schuld and Killoran (2022) showed that for re-uploading models on n qubits with L encoding layers, the gradient variance scales as $O(2^{-nL})$ for generic encoding and trainable layers. The plateau is *both* in qubit count and encoding depth.

Mitigation: as in the parameter-side tradeoff, structured encodings (with restricted frequency sets) have better trainability. The right encoding for QML is the one that includes the frequencies needed for the data, but no more.

Encoding As Feature Engineering

A useful classical analogy: **data encoding in quantum ML is like feature engineering in classical ML**.

In classical ML, a kernel or feature map $\phi(x)$ lifts the data into a feature space where a linear (or simple) classifier can separate it. The expressive power of the model is largely determined by the feature map, not the classifier.

In quantum ML, the data encoding plays the same role: it lifts x into a quantum state $|\psi(x)\rangle$, where the trainable observable $\langle M(\boldsymbol{\theta})\rangle$ performs the "linear classification."

The implications:

1. **Expressivity comes mostly from the encoding.** If the encoding is linear in x, the model is at most a linear function of trigonometric features.
2. **Universal QML requires universal encoding.** If you want to learn any function, you need a re-uploading encoding with enough Fourier modes.
3. **Domain knowledge goes into the encoding.** Just as in classical ML, choosing the right features (encoding) for the problem is the most important design decision.

This analogy is not perfect — quantum encodings introduce specifically *trigonometric* nonlinearities that don't have a clean classical counterpart — but it's close enough to import most of the intuition from classical feature engineering.

The Quantum Kernel Perspective

A related view: the encoding defines a **quantum kernel**.

$$K(x, x') = |\langle\psi(x)|\psi(x')\rangle|^2.$$

This is an inner product between encoded data points, computable on a quantum computer. A kernel-based classifier (SVM, ridge regression) can be applied directly.

Properties of quantum kernels:

1. **They capture the encoding's similarity structure.** Two inputs that map to similar quantum states have high kernel value.
2. **They can be exponentially powerful.** For some encodings, the quantum kernel computes inner products in an exponentially-large feature space (related to the IQP encoding of Havlicek et al. 2019).
3. **They sidestep the trainability problem.** Kernel methods don't require gradient descent over a deep parameterized circuit — you compute the kernel matrix and run a classical SVM.

The downside: kernel methods scale as $O(N^2)$ with dataset size N, where N is the number of training examples. For large datasets, this is prohibitive.

The kernel view is the clean theoretical alternative to variational QML, and recent work (Schuld 2021, Liu et al. 2021) suggests it's often *better* than variational training when the right kernel is available.

Practical Encoding Recommendations

For practitioners building QML models in 2026:

For low-dimensional regression / classification ($d < 10$): - Re-uploading with angle encoding, $L = 2 - 5$ layers. - Frequency set design: pick H so the frequencies match the expected smoothness of the target function.

For high-dimensional data ($d > 100$): - Amplitude encoding (if state preparation is feasible). - Or aggressive feature reduction (PCA) before encoding.

For structured data (images, time series): - Hamiltonian encoding with a structured H (e.g., a translation-invariant one for images). - Or specialized encodings (data convolutional layers, quantum CNNs).

For exploratory work: - Start with simple angle encoding to establish a baseline. - Add re-uploading layers until performance plateaus or trainability degrades. - Switch to kernel methods if variational training stalls.

The dominant message: **think about encoding before trainable parameters**. The encoding sets the ceiling on what you can learn; the parameters only navigate within that ceiling.

Structural Role

This node is the **machine-learning extension** of Module 8. It generalizes the expressivity framework from parameters to data inputs, introduces the Schuld-Sweke-Meyer Fourier characterization, and establishes encoding choice as the primary design decision in QML.

It enables node 8.7 (Quantum Advantage Criteria) by clarifying what kinds of functions quantum models can express that classical models can't — the prerequisite for any advantage claim.

Relations to Other Concepts

- **Schuld-Sweke-Meyer (2021)** — the Fourier characterization theorem.
- **Havlicek et al. (2019)** — IQP-based quantum kernels.
- **Schuld (2021)** — quantum kernel methods.
- **Pérez-Salinas et al. (2020)** — original re-uploading paper.
- **Module 4 (Barren plateaus)** — the trainability cost of expressive encodings.

Worked Example — Frequency Set For A Simple Re-Uploading Model

Consider a 1-qubit re-uploading model:

$$U(x, \boldsymbol{\theta}) = R_Y(\theta_4)e^{-ixZ}R_Y(\theta_3)e^{-ixZ}R_Y(\theta_2)e^{-ixZ}R_Y(\theta_1).$$

Three data encoding gates with $H = Z$, so the eigenvalues are $\{+1, -1\}$ and the differences are $\{2, 0, -2\}$.

Compute the frequency set. With $L = 3$ encoding layers and frequency differences $\{0, \pm 2\}$, the reachable frequencies are integer combinations of $\{2, 0, -2\}$ with at most 3 terms. This gives $\{-6, -4, -2, 0, 2, 4, 6\}$ — degrees up to 6.

Function class. $f(x; \boldsymbol{\theta})$ is a Fourier series with frequencies in $\{-6, -4, -2, 0, 2, 4, 6\}$ — i.e., a degree-6 trigonometric polynomial in $2x$.

Function expressibility. Can this model express $\sin(3x)$? The frequencies in $\sin(3x)$ are ± 3, which are *not* in the model's frequency set (which only has even frequencies). So no — the model cannot exactly express $\sin(3x)$.

To get $\sin(3x)$, you'd need $H = 3Z/2$ so the frequency differences include ± 3, or a different encoding that produces odd frequencies.

This is the kind of analysis you do *before* training: check that your encoding can represent the function family you care about.

Visualization

Pending. A diagram showing two re-uploading models: (a) with $L = 1$ encoding layer, frequency set $\{0, \pm 2\}$, expressing only low-frequency trigonometric functions; (b) with $L = 4$ encoding layers, frequency set $\{0, \pm 2, \pm 4, \pm 6, \pm 8\}$, expressing higher-frequency functions. Side by side with target function plots that the models can/can't reach. Caption: "Encoding sets the frequency budget. Training spends it."

Exercises

1. For a 1D re-uploading model with $H = X$ and $L = 2$ encoding layers, compute the frequency set. Which simple functions $\sin(\omega x), \cos(\omega x)$ can it represent for integer ω?
2. Can amplitude encoding represent the function $f(x) = x^2$? If yes, how? If no, what would you change?
3. (VQA) For a 2D classification task with circular decision boundary, design a re-uploading encoding that has the right frequency content. Which value of L is sufficient?

Research Extensions

- **Frequency-set design** — algorithmic methods for picking H and L to match a target function class.
- **Encoding-trainability optimization** — joint optimization of encoding and parameters.
- **Quantum kernels with provable advantage** — encodings that produce kernels not classically computable.
- **Multi-dimensional Fourier characterization** — extending Schuld-Sweke-Meyer to high-dimensional inputs.

Cross-links to Other Courses

- **Fourier Analysis** — trigonometric polynomials, Fourier series, Stone-Weierstrass.
- **Kernel Methods (classical ML)** — kernel SVMs, RKHS, feature maps.
- **Module 4 (Barren plateaus)** — trainability cost of deep encodings.
- **Module 8 Section 8.7 (Quantum Advantage)** — what encodings enable provable quantum advantage.

Variational Quantum Algorithms · Module 8 — Expressivity Vs Trainability · Node 8.6
Concepts: parameterized-circuits, hilbert-space
Prerequisites: 8.1, 8.5
Enables: 8.7

hbar.university — own.your.cognition