Adam and Adaptive Methods

Variational Quantum Algorithms · Module 6 — Optimization Dynamics

Node 6.3

Position in Knowledge Graph

System: Variational Quantum Algorithms **Structural Domain:** Optimization Dynamics **Node:** 6.3

Vanilla gradient descent (6.1) uses one global learning rate. Natural gradient (6.2) rescales the gradient by the QFIM, which is geometrically principled but expensive. **Adaptive methods** sit in between: they rescale the gradient using *cheap, online estimates* of curvature derived from the gradient history itself, without ever computing the QFIM.

The most famous adaptive method is **Adam** (Kingma and Ba, 2014), which has become the default optimizer in classical deep learning. Variants include RMSProp, AdaGrad, AdaDelta, AMSGrad, and RAdam — all with similar structure but different details.

This node describes the Adam family, explains why they often outperform vanilla GD on VQAs, and identifies their failure modes.

The Adam Update Rule

Adam maintains two moving averages alongside the parameters:

- $\mathbf{m}_k$: exponential moving average of gradients (first moment).
- $\mathbf{v}_k$: exponential moving average of squared gradients (second moment).

At each step:

$$\mathbf{m}_k = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\nabla C(\boldsymbol{\theta}_{k-1}),$$
$$\mathbf{v}_k = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)[\nabla C(\boldsymbol{\theta}_{k-1})]^2,$$
$$\widehat{\mathbf{m}}_k = \frac{\mathbf{m}_k}{1-\beta_1^k}, \quad \widehat{\mathbf{v}}_k = \frac{\mathbf{v}_k}{1-\beta_2^k} \quad \text{(bias correction)},$$
$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \frac{\widehat{\mathbf{m}}_k}{\sqrt{\widehat{\mathbf{v}}_k} + \epsilon}.$$

The default hyperparameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\eta = 10^{-3}$.

The squared-gradient division is the key. **It rescales each component of the update by an estimate of the local gradient variance**, which is roughly an estimate of the local curvature in that direction. Components with consistently large gradients get smaller effective steps; components with consistently small gradients get larger effective steps.

Why The Squared-Gradient Rescaling Works

The intuition is: $\mathbf{v}_k$ is an exponential moving average of $|\nabla C|^2$. For a smooth landscape near a minimum, $|\nabla C|^2$ is approximately related to the local curvature: high-curvature directions have rapidly-changing gradients, so $\mathbf{v}$ is large there; low-curvature directions have slowly-changing gradients, so $\mathbf{v}$ is small.

Dividing by $\sqrt{\mathbf{v}}$ thus produces an update where steps are normalized by $\sqrt{\text{curvature}}$, which is similar to what natural gradient does — except the "curvature" estimate is *empirical*, not derived from the QFIM.

This is much cheaper than computing the QFIM (no extra circuit evaluations), and it's adaptive (the estimate updates as the optimizer moves). The cost is that the curvature estimate is *noisy* and *biased* — $\mathbf{v}$ reflects past gradients, which may not reflect current curvature, and it's contaminated by shot noise.

In classical deep learning, this rescaling is usually a net win. In VQAs, the picture is more nuanced (see "Failure Modes" below).

Why Adam Works Well For VQA

1. Per-coordinate learning rates. Different parameters in a VQA ansatz often have wildly different sensitivities. A single global learning rate either underutilizes the small-sensitivity parameters or overshoots on the large-sensitivity ones. Adam's per-coordinate rescaling handles this automatically.

2. Robustness to gradient noise. The first-moment averaging $\mathbf{m}$ acts like momentum, smoothing out noisy gradient estimates over many iterations. This is helpful when shot noise is comparable to the gradient signal.

3. Adaptive step size near minima. As the optimizer approaches a minimum, the gradient shrinks but $\mathbf{v}$ remains large (it's a moving average of past, larger gradients), so the effective step size shrinks gracefully — no manual learning-rate decay needed.

4. No QFIM. Adam approximates curvature information without ever computing the QFIM, saving the $O(P^2)$ cost.

5. Robust default hyperparameters. The defaults ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\eta = 10^{-3}$) work reasonably well across a wide range of problems. This makes Adam a "fire and forget" choice for an unfamiliar VQA instance.

For these reasons, Adam is the **most common practical optimizer** for VQAs in 2026 software (Pennylane, Qiskit, Yao). It is the default for "I just want to optimize this thing without thinking about it."

Failure Modes Of Adam On VQAs

Adam's strengths come with corresponding weaknesses:

1. Bias from noisy gradients. $\mathbf{v}$ accumulates squared *noisy* gradients, not squared true gradients. This biases the estimate of curvature upward (because variance contributes to expectation of squared values), which biases the step size *downward*. In the high-noise regime, Adam can be more conservative than necessary, taking smaller steps than vanilla GD with the same learning rate.

2. No barren plateau immunity. Like vanilla GD and natural gradient, Adam doesn't escape barren plateaus. If the gradient is below the noise floor, no rescaling can recover signal.

3. Hyperparameter sensitivity in the noisy regime. The default $\beta_2 = 0.999$ (essentially infinite memory) was chosen for classical deep learning where the gradient is computed once per minibatch on a deterministic loss. For VQAs with shot noise, the optimal β_2 is typically smaller (more weight on recent gradients), and the optimal η depends strongly on the shot budget.

4. Coordinate-system dependence. Adam is *less* coordinate-dependent than vanilla GD (the per-coordinate rescaling helps), but it is *not* fully reparameterization-invariant the way natural gradient is. Reparameterizing the ansatz changes the Adam trajectory.

5. Convergence to non-stationary points (Reddi et al. 2018). Adam was shown to fail to converge in some convex examples — the squared-gradient bias can lead the optimizer to oscillate around a non-minimum point. **AMSGrad** is a variant that fixes this by using the maximum of past $\mathbf{v}$ values rather than an exponential average.

These failure modes are usually mild in practice but worth knowing about.

Variants

RMSProp (Hinton, 2012): Adam without the first-moment averaging. Just $\theta_{k+1} = \theta_k - \eta \nabla C / \sqrt{\mathbf{v}_k}$. Older, simpler, sometimes preferable when you don't want momentum dynamics.

AdaGrad (Duchi et al., 2011): Uses the cumulative sum of squared gradients instead of an exponential average. Step sizes monotonically decrease over time. Good for problems with sparse updates; converges very slowly on dense problems.

AdaDelta (Zeiler, 2012): Like RMSProp but with adaptive learning rate. Largely superseded by Adam.

AMSGrad (Reddi et al., 2018): Fixes Adam's non-convergence by using max over past $\mathbf{v}$ values.

RAdam (Liu et al., 2020): Rectified Adam, addresses the warm-up period instability of Adam.

LAMB / LARS: Layer-wise adaptive methods designed for large-batch training in deep learning. Less relevant for VQAs.

For VQA work, **Adam** and **AMSGrad** are the most commonly used. Other variants are situational.

Tuning Adam For VQA

Learning rate η: typically 10^{-2} to 10^{-1} for VQAs (larger than the classical default of 10^{-3}), because VQA gradients are usually smaller than classical NN gradients due to the cosine-bounded nature of expectation values.

β_2: consider lowering from 0.999 to 0.99 or 0.9 for noisy VQA optimization. The lower β_2 gives more weight to recent gradients, which is helpful when the noise level is high.

Shot budget: Adam's smoothing means you can use fewer shots per evaluation than vanilla GD with the same stability — the moving average does some of the noise reduction for you. N can often be reduced by a factor of 4-10 compared to a vanilla GD baseline with the same learning rate.

Restarts: if Adam stalls, restart with the current parameters but reset $\mathbf{m}$ and $\mathbf{v}$ to zero. This sometimes breaks out of bad regions where the moving averages have accumulated stale information.

These tuning adjustments are not from the original Adam paper but are folk wisdom from VQA practice.

Adam vs Natural Gradient

A natural question: does Adam approximate natural gradient?

Sort of, but not really.

Natural gradient rescales by g^{-1}, the inverse QFIM. Adam rescales by $1/\sqrt{\mathbf{v}}$, the inverse-sqrt of moving-average squared gradients. These are *related* — the squared-gradient moving average is a (noisy, biased)

estimate of the diagonal of the cost Hessian, and the cost Hessian is related to the QFIM through the cost function. But they are not the same.

Specifically: - Natural gradient uses **state-space curvature** (independent of cost function). - Adam uses **cost-space curvature estimate** (depends on cost function and gradient history).

In some regimes they agree; in others they differ significantly. Empirically, **Adam is usually faster per step (no QFIM cost) but Natural Gradient is usually fewer steps to convergence** (more accurate descent direction). The total budget winner depends on the problem.

A reasonable rule of thumb: use Adam by default; switch to natural gradient if you're spending too many iterations to converge and the per-iteration QFIM cost is affordable.

Structural Role

This node is the **practical default** of Module 6. Adam is what most VQA implementations use, and understanding its behavior — strengths, failure modes, hyperparameters — is essential for any practitioner.

It also bridges between vanilla GD (6.1) and natural gradient (6.2), occupying a middle ground in the cost-vs-quality tradeoff.

Relations to Other Concepts

- **Kingma and Ba 2014** — the original Adam paper.
- **Reddi et al. 2018** — AMSGrad and the convergence analysis.
- **Momentum methods (Polyak, Nesterov)** — Adam's first-moment averaging is essentially momentum.
- **AdaGrad / RMSProp** — predecessors that motivated Adam.
- **Natural gradient** — the geometric ideal that Adam approximates cheaply.

Worked Example — Adam Vs GD On A Two-Parameter Cost

Consider $C(\theta_1, \theta_2) = \theta_1^2 + 100\theta_2^2$ — a stretched quadratic with very different curvature in the two directions.

Vanilla GD: the optimal step size is $\eta = 1/100 = 0.01$ (limited by the worst direction). At $\boldsymbol{\theta}_0 = (1, 1)$, the gradient is $(2, 200)$. The update is $(1, 1) - 0.01 \cdot (2, 200) = (0.98, -1)$. Notice it overshot $\theta_2 = 0$ and went to -1. The next step will overshoot back. Vanilla GD oscillates in θ_2 while crawling in θ_1.

Adam (with default $\eta = 0.001$): at iteration 1, $\mathbf{m} = (0.2, 20)$ and $\mathbf{v} = (0.004, 40)$. The update is $-0.001 \cdot (0.2/\sqrt{0.004}, 20/\sqrt{40}) = -0.001 \cdot (3.16, 3.16) = (-0.003, -0.003)$. **Both components have the same step size**, despite the very different gradient magnitudes. This is the per-coordinate rescaling at work.

After many iterations, Adam approaches the minimum at roughly equal rates in both directions, while vanilla GD oscillates wildly. Adam reaches a precision of 10^{-3} in ~ 1000 iterations; vanilla GD with the same number of iterations is still oscillating.

For the analogous problem in VQA — a parameter that strongly affects cost (θ_1-like) and one that weakly affects cost (θ_2-like) — Adam handles both gracefully where vanilla GD struggles.

Visualization

Pending. A 2D contour plot of a stretched-quadratic loss with two trajectories: vanilla GD (zig-zag, slow descent in long direction) and Adam (smooth, balanced descent in both directions). Caption: "Adam handles ill-conditioning that GD cannot."

Exercises

1. For the stretched quadratic $C(\theta_1, \theta_2) = a\theta_1^2 + b\theta_2^2$, derive the optimal step size for vanilla GD and compare to Adam's effective step size at each component.
2. The bias correction in Adam (the $1 - \beta^k$ divisions) only matters in early iterations. Show that without bias correction, Adam under-estimates $\mathbf{m}$ and $\mathbf{v}$ for the first $\sim 1/(1-\beta)$ iterations.
3. (VQA) For a 20-parameter HEA with shot budget $N = 1000$, estimate the relative iteration count for vanilla GD vs Adam to converge to a fixed cost precision. Which uses fewer total shots?

Research Extensions

- **Adam variants for shot-noisy gradients** — versions of Adam that explicitly model shot noise in the moment estimates.
- **Adam + natural gradient hybrids** — using Adam as a fast preconditioner and natural gradient as a refinement.
- **Hyperparameter auto-tuning for Adam on VQA** — meta-learning the optimal β_2 for a given problem class.
- **Catalyst-style acceleration of Adam** — combining Adam with classical acceleration techniques.

Cross-links to Other Courses

- **Deep Learning** — Adam is the workhorse optimizer of classical neural network training.
- **Stochastic Optimization** — convergence theory for noisy gradient methods.
- **Module 6 Section 6.5 (Convergence Theory)** — formal convergence analysis of Adam.
- **Module 5 (Regularization)** — Adam is often combined with regularization terms in the objective.

Variational Quantum Algorithms · Module 6 — Optimization Dynamics · Node 6.3
Concepts: gradient-descent, stochastic-optimization, convergence
Prerequisites: 6.1, 6.2
Enables: 6.4, 6.5

hbar.university — own.your.cognition