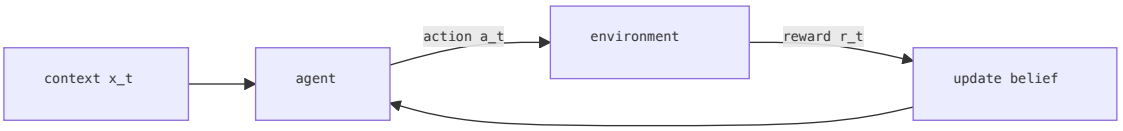
Anton A. Nesterov · an@vski.sh · June 21, 2026

The Essence of Context Bandits

bandits · theory

What a contextual bandit actually optimises, in one diagram and a handful of equations. With live math and a mermaid flow of the explore-exploit loop.

At each step the agent observes a context, picks an action, receives a reward, and updates its belief about which actions pay off in which contexts. The whole game is balancing *exploration* (trying actions whose payoff is still uncertain) against *exploitation* (choosing the action that looks best right now).



The decision rule

LinUCB picks the action whose upper-confidence-bound estimate is highest. For each action a with feature vector $\phi(x, a)$ and ridge estimate $\hat{\theta}_a = A_a^{-1} b_a$, the score is:

$$a_t = \arg\max_{a \in \mathcal{A}} \left(\hat{\theta}_a^\top \phi(x_t, a) + \beta \sqrt{\phi(x_t, a)^\top A_a^{-1} \phi(x_t, a)} \right)$$

The first term is **exploitation** — the predicted reward. The second term is **exploration** — a bonus that grows with the uncertainty (variance) of the estimate for that action. β controls the trade-off.

The update

When reward r_t arrives for action a_t, ridge regression gives a closed-form update — no gradient descent:

$$A_{a_t} \leftarrow A_{a_t} + \phi(x_t, a_t)\phi(x_t, a_t)^\top, \qquad b_{a_t} \leftarrow b_{a_t} + r_t\, \phi(x_t, a_t)$$

So A_a accumulates the outer products of feature vectors (a precision matrix), and b_a accumulates reward-weighted features. The invertibility comes from the λI ridge regularisation: $A_a = \lambda I + \sum \phi\phi^\top$.

Regret

The quantity a bandit actually minimises is **cumulative regret** — the gap between what it collected and what the best-fixed action *in hindsight* would have collected:

$$R_T = \sum_{t=1}^{T} \mathbb{E}[r_t^*] - \mathbb{E}[r_t]$$

A good linear bandit achieves $\mathcal{O}(\sqrt{T})$ regret — sublinear, meaning the average per-step regret $\to 0$ as $T \to \infty$. That is what "the agent learned" means, formally.

Why it runs on a toaster

The whole algorithm is two matrix multiplies and one solve per step. For the Bandit Fisher demo, A_a is 87×87 (the RBF feature dimension) — solving it is microseconds. No GPU, no training run, no backprop. The same family of algorithms decides what you see on TikTok.