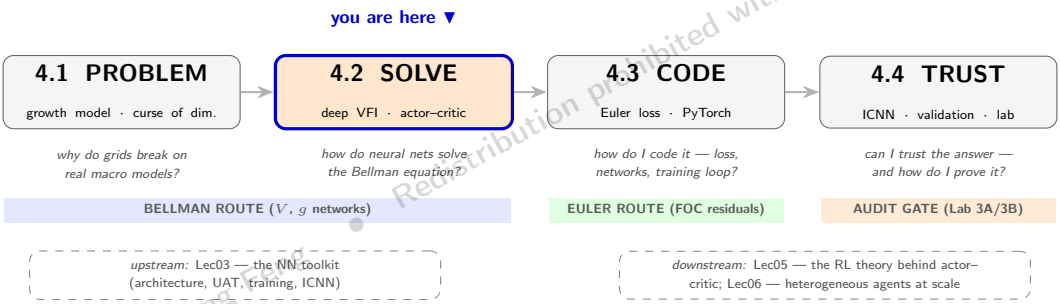
AI for Economic Research: Dynamic Models, Language, and Agents
Lecture 4.2: Deep VFI & Actor-Critic for Growth

Zhigang Feng

2026-07-07 10:05:43

Lecture 4 in One Map

One lecture, four decks, one running example: solve the optimal growth model with deep learning — pose the problem, solve it two ways, code it, and prove the answer right.



Act 4.2 (here) *solves it the Bellman way: replace the grid with networks for V and g, train them on simulated states, and meet actor–critic — the algorithm Lec. 5 T3 formalizes.*

Deep Learning Approach: NNs for Value and Policy Functions

Replace the grid with networks for V and g — mesh-free, dimension-robust

The Core Idea: Replace Grids with Neural Networks

- **Instead of** storing $V(k)$ on a grid, **approximate** it with a neural network:

$$V(k) \approx \Gamma_{\gamma_v}(k), \qquad g(k) \approx \Gamma_{\gamma_g}(k)$$

  where γ_v, γ_g are network parameters (weights and biases)

- **The analogy to VFI**:
  1. Generate states $k \sim D(k)$ from some distribution (replaces grid)
  2. Compute approximate target values or gradients for training
  3. Update γ_v or γ_g using gradient-based optimization

- **Why this helps**: Neural networks handle high-dimensional k naturally
  - No grid needed — states are *sampled*
  - The network generalizes across the state space
  - Parameters grow polynomially, not exponentially, with dimension

Embedding Equilibrium Conditions in Neural Networks

- Neural network approximations can incorporate **equilibrium conditions** directly:
  - **Feasibility constraints**: Ensure $g(k) \in \mathcal{G}(k)$
    - Example: $0 \leq c \leq f(k)$ (can't consume more than output)
  - **Market clearing**: $\sum_i c_i = Y$ or asset market clearing
  - **Budget constraints**: Household or government budget balance

- **Implementation strategies**:
  1. **Output transformations**: Sigmoid $\rightarrow$ bound outputs within feasible range
  2. **Penalty methods**: Add constraint violation terms to the loss
  3. **Lagrangian approaches**: Dual variables updated during training
  4. **Architectural design**: Structure the network so outputs automatically satisfy identities

- Building in economic structure improves convergence and ensures economically meaningful solutions

Deep VFI: The Algorithm at a Glance

Before the details: the whole loop in one place — a value step and a policy step, alternated, on sampled states.

Deep Value Function Iteration — represent $V \approx \Gamma_{\gamma_v}$, $g \approx \Gamma_{\gamma_g}$; iterate:

1. **Sample** a batch of states $\{k_i\} \sim D(k)$ (no grid)
2. **Value step (critic)**: fit Γ_{γ_v} to the Bellman target $\hat{V}(k) = u(k, g(k)) + \beta\, \Gamma_{\gamma_v}\big(f(k, g(k))\big)$ by least squares
3. **Policy step (actor)**: improve Γ_{γ_g} by gradient *ascent* on $u(k, g(k)) + \beta\, \Gamma_{\gamma_v}\big(f(k, g(k))\big)$
4. **Repeat** until $\Gamma_{\gamma_v}, \Gamma_{\gamma_g}$ stabilize

- Same rhythm as classical VFI — *evaluate* the current policy, then *improve* it — but with a **neural network** instead of traditional function approximation (interpolation, splines), and **sampled states** instead of a grid.
- The next slides unpack each step; the following section recasts the same loop in **actor-critic** language.

Expectations over shocks are estimated by Monte-Carlo simulation; that sampling machinery is developed in Lecture 5.

Deep VFI vs. Classical VFI

Same Bellman logic, different machinery — this is what lets it scale.

	Classical VFI	**Deep VFI**
State space	fixed grid $\{k_1, \ldots, k_N\}$	sampled batch $k_i \sim D(k)$
Representation	table / interpolation	neural network $\Gamma_\gamma(k)$
Value update	exact $\max_{k'}$ at each grid point	least-squares fit to $\hat{V}$
Policy	$\arg\max$ by grid search	learned policy net (grad. ascent)
Expectation	quadrature on a grid	Monte-Carlo over simulated shocks
Cost vs. dim d	N^d (curse of dimensionality)	polynomial in d
Convergence	contraction guarantee (Banach)	no general guarantee (empirical)

- The trade: **give up** the exact grid max and the contraction guarantee; **gain** the ability to scale to high dimensions and to fit scattered, simulated data.
- **Monte-Carlo** supplies the expectations; its details (sampling, exploration, variance) are the subject of **Lecture 5**.

Value Function Update (One-Step Look-Ahead)

- Fix a policy $g^{(n-1)}(k) = \Gamma_{\gamma_g}^{(n-1)}(k)$
- The one-step approximation for the value is:

$$\hat{V}(k) = u\big(k, g^{(n-1)}(k)\big) + \beta\, \Gamma_{\gamma_v}^{(n-1)}\big(f(k, g^{(n-1)}(k))\big)$$

- We then solve a **least squares** problem over the sampling distribution:

$$\min_{\gamma_v^{(n)}} \mathbb{E}_{k\sim D(k)}\Big[\big(\Gamma_{\gamma_v}^{(n)}(k) - \hat{V}(k)\big)^2\Big] \;\approx\; \frac{1}{N_v}\sum_{i=1}^{N_v}\big(\Gamma_{\gamma_v}^{(n)}(k_i) - \hat{V}(k_i)\big)^2$$

The empirical average $\frac{1}{N_v}\sum_i$ is the **special case** of $\mathbb{E}_{k\sim D(k)}$ using N_v i.i.d. draws $k_i \sim D(k)$.

- **Key**: States $\{k_i\}$ are *sampled* from distribution $D(k)$, not placed on a grid
- This is **supervised learning**: the "label" for each k_i is the Bellman target $\hat{V}(k_i)$
- **Detach the target** so gradients do not flow through $\Gamma_{\gamma_v}^{(n-1)}$ (target-network style update — crucial for stability)

The Role of Value Function in Policy Evaluation

- **What is $\hat{V}$?** With policy $g^{(n-1)}$ and value net $\Gamma_{\gamma_v}^{(n-1)}$ held *fixed*, the target

$$\hat{V}(k) = u\big(k, g^{(n-1)}(k)\big) + \beta\, \Gamma_{\gamma_v}^{(n-1)}\big(f(k, g^{(n-1)}(k))\big)$$

  is **one Bellman backup** under the current policy: $\hat{V} = T_{g^{(n-1)}}\Gamma_{\gamma_v}^{(n-1)}$.

- **Why $\hat{V} \approx V^*$?** At the optimum the Bellman equation holds exactly, $T_{g^*}V^* = V^*$ — so V^* is a *fixed point* of the backup. Since T_g is a β-contraction,

$$\|\hat{V} - V^*\| \;\le\; \beta \underbrace{\|\Gamma_{\gamma_v}^{(n-1)} - V^*\|}_{\text{value error}} \;+\; \underbrace{\|T_{g^{(n-1)}}V^* - V^*\|}_{\text{policy suboptimality} \to 0}$$

- Each backup **shrinks the value error by a factor** β (plus a term that vanishes as $g^{(n-1)} \to g^*$): fitting $\Gamma_{\gamma_v}^{(n)}$ to $\hat{V}$ and improving the policy drives $\hat{V} \to V^*$.

- **The average is over states**: $\mathbb{E}_{k\sim D(k)}[\cdot]$ averages the fit across the *state space* (not over shocks) — good average fitness on sampled states $\Rightarrow$ a good global approximation.

Policy Function Update

- Given the updated value network $\Gamma_{\gamma_v}^{(n)}$, improve the policy by solving:

$$\max_{\gamma_g^{(n)}} \ \mathbb{E}_{k\sim D(k)}\Big[u\big(k,\Gamma_{\gamma_g}^{(n)}(k)\big)+\beta\,\Gamma_{\gamma_v}^{(n)}\big(f(k,\Gamma_{\gamma_g}^{(n)}(k))\big)\Big]$$

- This is **gradient ascent** on the neural network parameters:

$$\gamma_g^{(n)}=\gamma_g^{(n-1)}+\alpha\cdot\nabla_{\gamma_g}J(\gamma_g)\Big|_{\gamma_g=\gamma_g^{(n-1)}}$$

- **Two roles of the gradient**:
  1. **Direction**: $\nabla_{\gamma_g} J$ tells us *which direction* in parameter space improves the objective
  2. **Magnitude**: Combined with learning rate α, determines *how far* we move

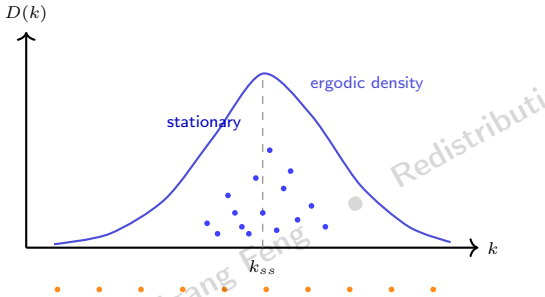
- In practice, optimizers like Adam adaptively adjust step sizes for faster and more stable convergence

Choosing the Sampling Distribution $D(k)$

- The distribution $D(k)$ determines *where* we evaluate the approximation
- **Common choices**:
  - **Uniform**: Simple, but wastes capacity on unlikely states
  - **Stationary distribution**: Focuses on states the economy actually visits
  - **Importance sampling**: Targets high-uncertainty or high-value regions
- **Stationary distribution approach**:
  1. Simulate the economy under current policy for many periods
  2. Wait until the state converges to its ergodic distribution
  3. Draw training points from that distribution
- **Benefits**: Sample efficiency (focus on what matters), faster convergence, no need for a handcrafted grid
- **Limitation**: Rare but important events (tail risks) may be under-represented

Where to Sample? Uniform vs. Stationary $D(k)$

One-sector stochastic growth: the economy spends almost all its time near the ergodic mean k_{ss} — so that is where the approximation must be accurate.



- **Uniform** (orange): spends half its points on low-k states the economy almost never visits — wasted capacity.
- **Stationary** (blue): simulate the model forward, keep the ergodic draws — points land where accuracy matters.
- **Tails**: rare high-/low-k events are under-sampled — add a few importance-sampled draws if they matter (e.g. disaster risk).

Why Not Chebyshev Here?

- Chebyshev and finite-element methods must approximate over the *entire* domain
- As dimensionality grows, covering the entire domain becomes infeasible
- **Stationarity-based sampling** zooms in on the *likely* region of interest
- **Possible concerns and remedies**:
  - **Rare but important events**: Combine stationary sampling with targeted sampling of rare states (importance sampling)
  - **Multiple steady states**: A single stationary distribution might miss transient states
  - **Out-of-distribution robustness**: Conduct stress tests separately
- The combination of **Monte Carlo sampling** (generates the right data in relevant states) and **neural networks** (fits irregular, mesh-free data) is the key synergy

Bias-Variance Tradeoff: One-Step vs. Multi-Step

- **One-step** ($T_{\text{sim}} = 1$):

$$\hat{V}(k_i) = u\big(k_i, g(k_i)\big) + \beta\, \Gamma_{\gamma_v}(k_{i,1})$$

  + Easy to implement, low variance
  − High bias if $V(\cdot)$ approximation is poor

- T_{sim}**-step return**:

$$\hat{V}(k_i) = \sum_{t=0}^{T_{\text{sim}}-1} \beta^t\, u(k_{i,t}, g(k_{i,t})) + \beta^{T_{\text{sim}}}\, \Gamma_{\gamma_v}(k_{i,T_{\text{sim}}})$$

  + Lower bias: uses actual simulated rewards over multiple steps
  − Higher variance: simulated paths can vary significantly

- **In practice**: Choose T_{sim} to balance bias vs. variance (typical values $T_{\text{sim}} = 30$–50)

Actor-Critic Algorithm for the Growth Model

Two networks teach each other: the critic scores, the actor improves (Lec. 5 formalizes)

Why Actor-Critic? The Optimization Bottleneck

We apply the actor-critic idea here as a concrete solver for the optimal-growth model; the full RL theory (MDPs, Monte Carlo, TD, on-/off-policy actor-critic) is developed in Lecture 5.

- **Classical VFI** requires solving $\max_{k'} Q(k, k')$ at every state
- When Q is represented by a neural network (non-convex), this inner maximization is expensive and unreliable

- **The actor-critic solution**: Use *two* networks to avoid the inner max:
  1. **The Actor (Policy Net π)** — "The Doer"
     - Input: state k → Output: action k' directly
     - *Avoids* the expensive maximization step
  2. **The Critic (Value Net V)** — "The Judge"
     - Input: state k → Output: estimated value $V(k)$
     - Minimizes the **Bellman residual** (prediction error)

- The actor proposes actions; the critic evaluates them; both improve iteratively

Mapping Macroeconomics to Reinforcement Learning

To apply RL, we translate economic models into the standard tuple (S, A, P, R, γ):

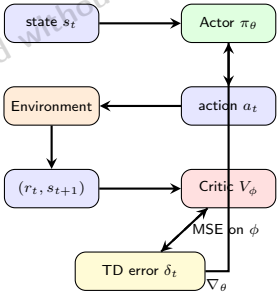
RL Concept	Economic Equivalent	Note
Agent	Household / Firm	The optimizer
Environment	Constraints & Prices	The rules of the game
State (S)	(k, z, Δ)	Wealth, shocks, **distribution**
Action (A)	c, i, l	Consumption, investment
Transition (P)	$k' = (1-\delta)k + i$	**Micro**: Known perfectly
	$\Delta' = H(\Delta)$	**Macro**: Often unknown/intractable
Reward (R)	$u(c)$	Period utility
Discount (γ)	β	Time preference

The synergy: RL handles the complexity of the macro transition $\Delta' = H(\Delta)$ by simply *simulating* it, without needing to derive the law of motion explicitly

Actor-Critic Algorithm (Preview — full treatment in Lec05 T3)

We use actor-critic here as a *solver* for the growth-model Bellman equation. The MDP framing (S, A, P, R, γ), the policy gradient theorem, and the full algorithm family come in Lec05.

1. **Initialize**: $\pi_\theta^{(0)}$ $(\equiv \Gamma_{\gamma_g}^{(0)})$, $V_\phi^{(0)}$ $(\equiv \Gamma_{\gamma_v}^{(0)})$
2. **Critic update** (V_ϕ):
   - Data $\{k_i, \hat{V}(k_i)\}$ via T_{sim}-step MC
   - $\min_{\phi^{(n)}} \mathbb{E}_{k\sim D(k)}\left[(V_\phi^{(n)}(k) - \hat{V}(k))^2\right]$
3. **Actor update** (π_θ):
   - Sample states $\{k_i\}$ (stationary dist.)
   - $\max_{\theta^{(n)}} \mathbb{E}_{k\sim D(k)}\left[u + \beta V_\phi^{(n)} \circ f\right]$
4. **Check convergence**; repeat until π_θ, V_ϕ stabilize.



Actor proposes; Critic judges; TD error updates both.

Lec05 T3 unpacks this as an algorithm family — full pseudocode, policy-gradient theorem, A2C / PPO / DDPG / SAC variants, and convergence theory.

Addressing the Infinite Horizon

The tension: Macro models maximize over an infinite horizon ($\sum_{t=0}^{\infty} \beta^t u_t$), but Monte Carlo simulations are always finite.

The solution: Bootstrapping

- We don't need to simulate to infinity
- We use the **estimated value function** (Critic) as a proxy for the infinite future:

$$\text{Total Value} \approx \underbrace{\sum_{t=0}^{T-1} \beta^t u_t}_{\text{Simulated rewards}} + \beta^T \underbrace{\hat{V}(s_T)}_{\text{Learned proxy for } \sum_{t=T}^{\infty}}$$

- This allows learning infinite-horizon policies using only short simulation rollouts
- The Critic improves over time, making the bootstrap estimate more accurate

Monte Carlo and Uncertainty

- When the model includes **stochastic shocks**, the future state evolves randomly
- Enumerating all possible future states or integrating over a high-dimensional shock space quickly becomes infeasible (another curse of dimensionality)
- **Monte Carlo simulation** circumvents this:
  - Generate a large sample of realized state-and-shock paths
  - Compute empirical averages to approximate expectations
  - No need to explicitly deal with the full distribution
- **In practice**:
  - Choose T_{sim} to balance bias vs. variance
  - Use parallel or GPU-based Monte Carlo for large simulation demands
  - Combine with importance sampling if rare shocks are of interest

Advantage I: Flexible Function Approximation

- **Old way — Structured grids**:
  - Requires tensor product grids or sparse grids (Smolyak)
  - Adding one state variable *multiplies* the grid size
- **New way — Neural networks**:
  - Universal approximators: can represent any continuous function
  - Parameter count grows linearly/polynomially with dimension, *not* exponentially
  - Can handle high-dimensional states (entire shock histories, fine-grained distributions) that break grid-based methods
- **Result**: Models that were previously intractable become solvable

Advantage II: The "Mesh-Free" Synergy

Why is the combination of **Monte Carlo** and **deep learning** so powerful?

- **Monte Carlo simulation** produces a "cloud" of points concentrated in the **ergodic set** (economically relevant states)
  - Traditional interpolators (Chebyshev, splines) require **structured node sets** (tensor or sparse grids) and degrade on scattered, clustered, high-dimensional Monte-Carlo data — the curse of dimensionality
- **Neural networks are "mesh-free"**:
  - They do not require a grid
  - They simply minimize error on *whatever* points you give them
- **Conclusion**: MC generates the *right data* (relevant states), and DL is the *right tool* to fit that irregular data

Bridge: Two Networks Solved It — Can One Loss Do It Too?

Act 4.2 solved the model the Bellman way: a critic network for V, an actor network for g, trained on simulated states — mesh-free exactly where grids exploded.

- Monte Carlo supplies the right data (the ergodic set); neural networks are the right fitter for that scattered, high-dimensional cloud.
- **Next (4.3):** the supervised-learning route — turn the Euler equation's first-order condition into a loss and minimize residuals directly, then implement everything in PyTorch.

*Arc: 4.1 PROBLEM (growth model + curse) → **4.2 SOLVE** (deep VFI + actor–critic) → 4.3 CODE (Euler + PyTorch) → 4.4 TRUST (structure + validation).*