Deep Learning for Corporate Finance: Brief Report

Zhaoxuan Wang

2026-01-31

1 Introduction

This report documents the implementation of deep learning methods for solving dynamic corporate finance models. The core problem is to find optimal investment and financing policies for firms facing adjustment costs, productivity shocks, and (in the extended model) risky debt with endogenous default.

The approach uses neural networks to approximate policy and value functions, trained via three complementary methods:

1. **Lifetime Reward (LR)**: Maximize discounted cumulative cash flows over simulated trajectories
2. **Euler Residual (ER)**: Minimize violations of the first-order optimality conditions
3. **Bellman Residual (BR)**: Actor-critic training to satisfy the Bellman equation

The document covers two models of increasing complexity:

- **Basic Model**: Firm chooses capital investment $(k, z) \to k'$ subject to convex and fixed adjustment costs
- **Risky Debt Model**: Firm chooses capital and borrowing $(k, b, z) \to (k', b')$ with endogenous default and risky interest rates

Key implementation challenges addressed include: reproducible data generation via deterministic RNG scheduling, numerical stability through input normalization and bounded outputs, and handling discontinuities (adjustment cost thresholds, default boundaries) via temperature-annealed soft gates.

1.1 Notation

Trainable Parameters

- Policy network parameters: θ_{policy}
- Value network parameters: θ_{value}
- Price network parameters (risky debt only): θ_{price}

Conventions

- Current period variable: x
- Next period variable: x'
- Parameterized function: $\Gamma(\cdot; \theta)$

State and Action Variables

- Exogenous productivity shock: $z > 0$
- Capital stock: $k \geq 0$
- Debt (borrowing): $b \geq 0$

AR(1) Shock Process

The log-productivity follows an AR(1) process:

$$\ln z' = (1 - \rho)\mu + \rho \ln z + \sigma\varepsilon', \quad \varepsilon' \sim \mathcal{N}(0, 1) \text{ i.i.d.}$$

The ergodic (unconditional) standard deviation is $\sigma_{\ln z} = \sigma/\sqrt{1-\rho^2}$.

Policies and Prices

- Basic model policy: $(k, z) \mapsto k'$
- Risky debt policy: $(k, b, z) \mapsto (k', b')$
- Risky debt bond price: $(k', b', z) \mapsto q = 1/(1+\tilde{r})$

Training Methods

- **LR**: Lifetime Reward loss $\mathcal{L}_{\text{LR}}$
- **ER**: Euler Residual loss $\mathcal{L}_{\text{ER}}$
- **BR**: Bellman Residual with critic loss $\mathcal{L}^{\text{BR}}_{\text{critic}}$ and actor loss $\mathcal{L}^{\text{BR}}_{\text{actor}}$

---

2 Economic Models

2.1 Basic Model

State and Action

- State: (k, z) where k is capital stock and z is productivity
- Action: k' (next-period capital)

Investment

$$I = k' - (1-\delta)k$$

Investment can be positive (expansion) or negative (disinvestment).

Operating Profit

$$\pi(k, z) = z \cdot k^{\theta}, \quad \theta \in (0, 1)$$

where θ is the production technology parameter (decreasing returns to scale).

Capital Adjustment Cost

$$\psi(I, k) = \phi_0 \cdot \frac{I^2}{2k} + \phi_1 \cdot k \cdot \mathbf{1}\{I \neq 0\}$$

where ϕ_0 is the convex adjustment cost coefficient and ϕ_1 is the fixed adjustment cost coefficient. The indicator $\mathbf{1}\{I \neq 0\}$ triggers whenever the firm invests or disinvests.

Cash Flow (Payout)

$$e(k, k', z) = \pi(k, z) - \psi(I, k) - I$$

Objective

The firm maximizes expected discounted lifetime cash flows:

$$\max_{\{k_{t+1}\}_{t=0}^{\infty}} \mathbb{E}\left[\sum_{t=0}^{\infty} \beta^t \cdot e(k_t, k_{t+1}, z_t)\right]$$

where $\beta = 1/(1+r)$ is the discount factor and r is the risk-free rate.

Bellman Equation

$$V(k, z) = \max_{k'} \{e(k, k', z) + \beta\, \mathbb{E}[V(k', z') \mid z]\}$$

2.2 Risky Debt Model

The risky debt model extends the basic model by allowing firms to borrow at an endogenous risky interest rate, with the option to default.

State and Action

- State: (k, b, z) where $b \geq 0$ is outstanding debt
- Action: (k', b') where $b' \geq 0$ is new borrowing

Cash Flow

$$e(\cdot) = (1-\tau)\pi(k,z) - \psi(I,k) - I + q \cdot b' + \frac{\tau \tilde{r}\, b'}{(1+\tilde{r})(1+r)} - b$$

where:

- τ is the corporate tax rate
- b is repayment of last-period debt
- $q \cdot b' = b'/(1+\tilde{r})$ is proceeds from issuing new risky debt
- The third term is the tax shield from debt interest

External Financing Cost

When cash flow is negative, the firm must raise costly external equity:

$$\eta(e) = (\eta_0 + \eta_1 |e|) \cdot \mathbf{1}\{e < 0\}$$

Endogenous Risky Interest Rate

The bond price $q = 1/(1+\tilde{r})$ is determined by the lender's zero-profit condition:

$$b'(1+r) = (1+\tilde{r})b'\, \mathbb{E}[1 - D \mid z] + \mathbb{E}[D \cdot R(k', b', z') \mid z]$$

where:

- LHS: Opportunity cost of lending at risk-free rate
- RHS: Expected return accounting for default probability and recovery

Endogenous Default

The firm defaults when its continuation (latent) value is negative:

$$D(k', b', z') = \mathbf{1}\{\tilde{V}(k', b', z') < 0\}$$

Shareholders walk away with zero under limited liability:

$$V(k', b', z') = \max\{0, \tilde{V}(k', b', z')\}$$

Recovery Under Default

$$R(k', z') = (1-\alpha)\left[(1-\tau)\pi(k', z') + (1-\delta)k'\right]$$

where $\alpha \in [0, 1]$ is the deadweight loss from liquidation.

Bellman Equation

$$\tilde{V}(k, b, z) = \max_{k', b'} \left\{ e(\cdot) - \eta(e) + \beta\, \mathbb{E}[V(k', b', z') \mid z] \right\}$$

The Nested Fixed-Point Problem

A key computational challenge is that the latent value $\tilde{V}$ depends on the risky rate $\tilde{r}$, but solving for $\tilde{r}$ requires knowing the default probability $\mathbb{E}[D]$, which depends on $\tilde{V}$. Traditional methods solve this via nested iteration. The neural network approach trains policy, value, and pricing networks jointly, avoiding explicit nested loops.

---

3 Data Generation

This section describes how synthetic training data is generated with emphasis on reproducibility and numerical stability.

3.1 State Space Bounds

The state space bounds can be specified in two ways:

1. **Model-based (recommended)**: Specify bounds as multipliers on the steady-state capital k^*, which the framework converts to levels
2. **Direct specification**: Provide bounds directly in arbitrary units

Productivity Shock Bounds

Given AR(1) parameters (μ, σ, ρ), the ergodic distribution of $\ln z$ is truncated at m standard deviations:

$$\ln z \in [\mu - m \cdot \sigma_{\ln z},\ \mu + m \cdot \sigma_{\ln z}], \quad \sigma_{\ln z} = \frac{\sigma}{\sqrt{1-\rho^2}}$$

The default $m = 3$ covers the mass of the ergodic distribution.

Capital Stock Bounds

The frictionless steady-state capital (where marginal product equals user cost) is:

$$k^*(z) = \left(\frac{z \cdot \theta}{r + \delta}\right)^{\frac{1}{1-\theta}}$$

evaluated at the stationary mean productivity $z = e^{\mu}$.

Users specify bounds as multipliers on k^*:

$$k_{\min} = k_{\min}^{\text{mult}} \times k^*, \quad k_{\max} = k_{\max}^{\text{mult}} \times k^*$$

For example, multipliers $(0.2, 3.0)$ yield a state space from 20% to 300% of steady-state capital.

Debt Bounds

The maximum debt is the natural borrowing limit—the maximum repayable in the best state:

$$b_{\max} = z_{\max} \cdot k_{\max}^{\theta} + k_{\max}$$

which equals maximum production plus full liquidation value. The lower bound is $b_{\min} = 0$.

Validation Constraints

When using model-based bounds, the framework validates:

- $m \in (2, 5)$: Sufficient coverage without extreme outliers
- $k_{\min}^{\text{mult}} \in (0, 0.5)$: Allows starting below steady state
- $k_{\max}^{\text{mult}} \in (1.5, 5)$: Allows starting above steady state

These constraints prevent numerical overflow while permitting economically meaningful variation.

3.2 Input Normalization

Neural networks perform best with inputs in a standardized range. The framework normalizes all state variables to $[0, 1]$ internally using min-max normalization:

Capital: $\hat{k} = (k - k_{\min})/(k_{\max} - k_{\min})$

Debt: $\hat{b} = b/b_{\max}$

Log-productivity: $\widehat{\ln z} = (\ln z - \ln z_{\min})/(\ln z_{\max} - \ln z_{\min})$

This normalization:

- Prevents exploding gradients from large input magnitudes
- Ensures consistent scale across different parameterizations
- Is invertible, so outputs can be mapped back to economic levels

Networks accept inputs in levels and perform normalization internally. Outputs are also returned in levels via bounded sigmoid activations scaled to the appropriate range.

3.3 Dataset Structure

3.3.1 Training Set

The training set is a stream of batches $\{\mathcal{B}^j\}_{j=1}^{J}$ where each batch contains n i.i.d. samples:

$$\mathcal{B}^j = \left\{ \left(k_{0,i}, b_{0,i}, z_{0,i}, \{\varepsilon_{t,i}^{(1)}, \varepsilon_{t,i}^{(2)}\}_{t=1}^{T} \right) \right\}_{i=1}^{n}$$

Each sample i represents an independent firm with:

- Initial states (k_0, b_0, z_0) drawn uniformly from the state space
- Two independent shock sequences $\varepsilon^{(1)}, \varepsilon^{(2)}$ for the cross-product estimator

The initial debt b_0 is used only in the risky debt model.

3.3.2 Validation and Test Sets

- **Validation set**: Fixed dataset of size $N_{\text{val}} = 10n$, used for model selection and early stopping
- **Test set**: Fixed dataset of size $N_{\text{test}} = 50n$, used only for final evaluation

Both are generated from the same distribution but with different RNG seeds.

3.4 Main Path vs. Fork Path

For each sample, two shock realizations are generated to enable unbiased estimation of squared expectations.

Main Path (AR(1) Chain)

The main shock sequence $\{z_t^{(1)}\}$ forms a continuous AR(1) chain:

$$\ln z_{t+1}^{(1)} = (1-\rho)\mu + \rho \ln z_t^{(1)} + \sigma \varepsilon_{t+1}^{(1)}$$

This path is used for lifetime reward calculations in the LR method.

Fork Path (One-Step Branches)

The fork sequence $\{z_t^{(2)}\}$ branches from the main path at each period:

$$\ln z_{t+1}^{(2)} = (1-\rho)\mu + \rho \ln z_t^{(1)} + \sigma \varepsilon_{t+1}^{(2)}$$

Each fork $z_{t+1}^{(2)}$ is a one-step transition from the main path $z_t^{(1)}$, not a parallel chain.

```
Main (AR1 Chain):   z0 -> z1 -> z2 -> z3 -> ... -> zT
                      \     \     \     \
Fork (1-step AR1):    z1F   z2F   z3F   zTF
```

Usage by Method

- LR: Uses main path only (continuous chain for lifetime rewards)
- ER/BR: Uses cross-product of main and fork at each transition for variance reduction

3.5 Trajectory vs. Flattened Data

The data generator produces two formats optimized for different training methods.

3.5.1 Trajectory Format (for LR)

The LR method requires full trajectories to compute cumulative discounted rewards:

$$\text{Shape: } (N, T+1) \text{ for shock paths}$$

Each sample preserves the temporal sequence $(z_0, z_1, \ldots, z_T)$.

3.5.2 Flattened Format (for ER and BR)

The ER and BR methods operate on single-step transitions and require i.i.d. samples for valid stochastic gradient descent. A key design choice is that states (k, b) are sampled **independently** for each transition rather than extracted from simulated trajectories.

Rationale: At data generation time, no policy exists to generate the capital/debt sequence $(k_1, k_2, \ldots, k_T)$. Using an arbitrary behavioral policy would introduce bias. Instead, the framework samples (k, b) directly from the state space, treating each draw as from the ergodic distribution. This approach:

1. **Ensures i.i.d. samples**: Each transition is statistically independent, satisfying the assumptions of SGD
2. **Eliminates serial correlation**: Consecutive time steps in a trajectory are correlated; independent sampling removes this dependency
3. **Approximates ergodic coverage**: Uniform sampling over the bounded state space approximates draws from the ergodic distribution that would arise under an optimal policy

The flattened dataset has shape $(N \times T,)$ with each observation:

$$\text{Obs} = (k, b, z, z'_1, z'_2)$$

where (z'_1, z'_2) are the main and fork next-period shocks. After flattening, the dataset is randomly shuffled to further eliminate any residual structure.

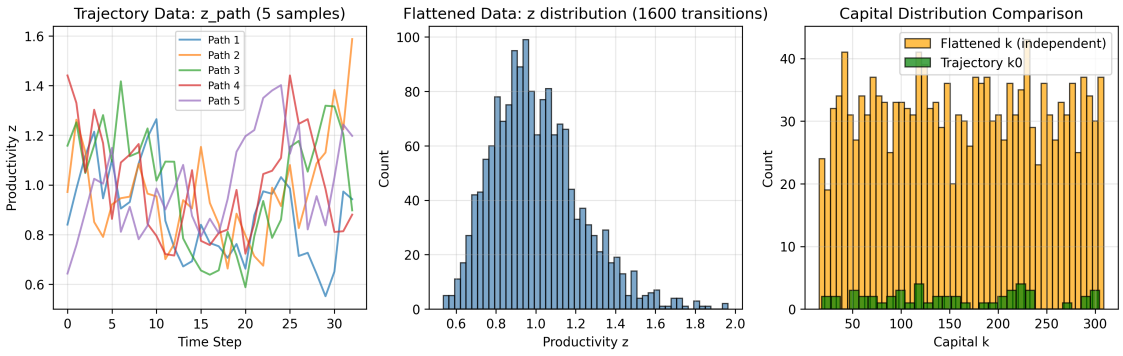

Figure 1: Comparison of trajectory and flattened data formats. Left: trajectory format preserves temporal structure for LR method. Right: flattened format with independent state sampling for ER/BR methods. *Example output from debug-mode training with small sample size.*

3.6 RNG Seed Schedule

Reproducibility requires deterministic random number generation. The framework uses TensorFlow's stateless random functions (`tf.random.stateless_*`) which produce identical outputs given identical seed pairs, regardless of call order.

Master Seed

A master seed pair $\mathbf{s}^{\text{master}} = (m_0, m_1)$ anchors all randomness.

Split Seeds

Disjoint seeds for each dataset:

$$\mathbf{s}^{\text{train}} = (m_0 + 100, m_1), \quad \mathbf{s}^{\text{val}} = (m_0 + 200, m_1), \quad \mathbf{s}^{\text{test}} = (m_0 + 300, m_1)$$

Variable IDs

Each random variable has a fixed ID:

Variable	ID
k_0	1
z_0	2
b_0	3
$\varepsilon^{(1)}$	4
$\varepsilon^{(2)}$	5

Training Seeds by Step

For training step j and variable x:

$$\mathbf{s}^{\text{train}}_{j,x} = (m_0 + 100 + \text{VarID}(x),\ m_1 + j)$$

This ensures each batch j receives unique, reproducible random draws.

Validation/Test Seeds

These are single fixed datasets (no step index):

$$\mathbf{s}^{\text{val}}_{x} = (m_0 + 200 + \text{VarID}(x),\ m_1)$$

$$\mathbf{s}^{\text{test}}_{x} = (m_0 + 300 + \text{VarID}(x),\ m_1)$$

This schedule guarantees:

1. **Reproducibility**: Identical seeds produce identical data across runs
2. **Common random numbers**: Different methods train on the same data, enabling fair comparison
3. **No leakage**: Train/validation/test sets use disjoint RNG streams

---

4 Network Architecture

4.1 Basic Model Networks

Policy Network

$$k' = \Gamma_{\text{policy}}(k, z; \theta_{\text{policy}})$$

- **Input**: (k, z) in levels, normalized internally to $[0, 1]$

- **Hidden layers**: 2 layers with 32 units each, SiLU activation
- **Output**: Sigmoid scaled to $[k_{\min}, k_{\max}]$:

$$k' = k_{\min} + (k_{\max} - k_{\min}) \cdot \sigma(\text{raw})$$

Value Network (BR method only)

$$V(k, z) = \Gamma_{\text{value}}(k, z; \theta_{\text{value}})$$

- **Input/Hidden**: Same as policy network
- **Output**: Linear (unbounded), since value can be any real number

4.2 Risky Debt Model Networks

Policy Network

$$(k', b') = \Gamma_{\text{policy}}(k, b, z; \theta_{\text{policy}})$$

- **Input**: (k, b, z) in levels, normalized internally to $[0, 1]$
- **Hidden layers**: Shared trunk with 2 layers, 32 units, SiLU activation
- **Output heads**:
  - $k' = k_{\min} + (k_{\max} - k_{\min}) \cdot \sigma(\text{raw}_k)$
  - $b' = b_{\max} \cdot \sigma(\text{raw}_b)$ (ensures $b' \geq 0$)

Value Network

$$\tilde{V}(k, b, z) = \Gamma_{\text{value}}(k, b, z; \theta_{\text{value}})$$

- **Output**: Linear (latent value can be positive or negative)

Price Network

$$q(k', b', z) = \Gamma_{\text{price}}(k', b', z; \theta_{\text{price}})$$

- **Output**: Sigmoid scaled to $(0, 1/(1+r)]$:

$$q = \frac{1}{1+r} \cdot \sigma(\text{raw})$$

  This ensures $\tilde{r} \geq r$ (risky rate at least equals risk-free rate).

4.3 Common Design Choices

Architecture

- Fully connected networks with configurable depth and width
- Default: 2 hidden layers, 32 units each
- SiLU (swish) activation: $\text{SiLU}(x) = x \cdot \sigma(x)$, which provides smoother gradients than ReLU

Input Processing

- Networks receive inputs in economic levels
- Internal min-max normalization to $[0, 1]$
- Economic primitives (π, ψ, e, etc.) computed using levels

Output Constraints

- Bounded outputs use sigmoid scaled to valid ranges
- Unbounded outputs (values) use linear activation

---

5 Training Methods

5.1 Cross-Product Estimator

The ER and BR methods require minimizing squared expectations of the form $\mathbb{E}[f]^2$. A naive estimator $(\frac{1}{n}\sum_i f_i)^2$ is biased. Instead, we use two independent shock realizations (z_1', z_2') per state to form an unbiased cross-product estimator:

$$\widehat{\mathbb{E}[f]^2} = \frac{1}{n}\sum_{i=1}^{n} f_{i,1} \times f_{i,2}$$

where $f_{i,1}$ and $f_{i,2}$ are residuals computed using the two independent shocks. This estimator is unbiased because $\mathbb{E}[f_1 \cdot f_2] = \mathbb{E}[f_1] \cdot \mathbb{E}[f_2] = \mathbb{E}[f]^2$ when f_1, f_2 are i.i.d.

5.2 Lifetime Reward (LR) Method

The LR method directly maximizes expected discounted lifetime rewards.

Objective

$$\max_{\theta} \; \mathbb{E}_{k_0, z_0, \{\varepsilon_t\}} \left[\sum_{t=0}^{T-1} \beta^t e(k_t, k_{t+1}, z_t) + \beta^T V^{\text{term}}(k_T, z_T) \right]$$

Terminal Value

To correct for finite-horizon truncation, a terminal value approximates continuation:

$$V^{\text{term}}(k_T, z_T) = \frac{e(k_T, k_T, z_T)}{1-\beta}$$

This assumes the policy has converged to steady state by period T, maintaining $k_{T+1} = k_T$ with replacement investment $I = \delta k_T$.

Empirical Loss

$$\mathcal{L}^{\text{LR}}(\theta) = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left(\sum_{t=0}^{T-1} \beta^t e(k_{t,i}, k_{t+1,i}, z_{t,i}) + \beta^T V^{\text{term}}(k_{T,i}, z_{T,i}) \right)$$

Algorithm

1. Load batch with initial states (k_0, z_0) and shock sequences $\{\varepsilon_t^{(1)}\}$
2. Simulate trajectory: $k_{t+1} = \Gamma_{\text{policy}}(k_t, z_t; \theta)$ for $t = 0, \ldots, T-1$
3. Compute discounted rewards and terminal value
4. Update θ via gradient descent on $\mathcal{L}^{\text{LR}}$

Key Points

- Uses trajectory data (main path only)
- Entire simulation is differentiable (gradients flow through time via backpropagation)
- Fork path not needed (no cross-product estimator)

5.3 Euler Residual (ER) Method

The ER method minimizes violations of the first-order optimality conditions.

Euler Equation

From the Lagrangian, the optimal policy satisfies:

$$1 + \psi_I(I_t, k_t) = \beta\, \mathbb{E}\left[\pi_k(k_{t+1}, z_{t+1}) - \psi_k(I_{t+1}, k_{t+1}) + (1-\delta)(1 + \psi_I(I_{t+1}, k_{t+1}))\right]$$

where subscripts denote partial derivatives.

Marginal Benefit Function

$$m(k', k'', z') = \pi_k(k', z') - \psi_k(I', k') + (1 - \delta)(1 + \psi_I(I', k'))$$

where $I' = k'' - (1 - \delta)k'$.

Unit-Free Residual

Dividing by the LHS gives a unit-free residual:

$$f = 1 - \beta \cdot \frac{m(k', k'', z')}{1 + \psi_I(I, k)}$$

At the optimum, $\mathbb{E}[f] = 0$.

Empirical Loss (Cross-Product)

$$\mathcal{L}^{\text{ER}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} f_{i,1} \times f_{i,2}$$

Target Network for Stability

Computing $k'' = \Gamma_{\text{policy}}(k', z'; \theta)$ creates a recursive dependency. To stabilize training, a target network θ^- computes future actions:

$$k'' = \Gamma_{\text{policy}}(k', z'; \theta^-)$$

The target is updated via Polyak averaging: $\theta^- \leftarrow \nu\theta^- + (1 - \nu)\theta$.

Algorithm

1. Load flattened batch (k, z, z'_1, z'_2)
2. Compute $k' = \Gamma_{\text{policy}}(k, z; \theta)$ and $I = k' - (1 - \delta)k$
3. Compute $k''_\ell = \Gamma_{\text{policy}}(k', z'_\ell; \theta^-)$ for $\ell = 1, 2$
4. Compute residuals f_1, f_2 and cross-product loss
5. Update θ and Polyak-update θ^-

5.4 Bellman Residual (BR) Method

The BR method uses actor-critic training to satisfy the Bellman equation.

5.4.1 Critic Update

The critic (value network) is trained to satisfy:

$$V(k, z) = e(k, k', z) + \beta\, \mathbb{E}[V(k', z')]$$

Bellman Target

$$y_\ell = e(k, k', z) + \beta\, \Gamma_{\text{value}}(k', z'_\ell; \theta^-_{\text{value}}), \quad \ell = 1, 2$$

where actions come from the target policy $k' = \Gamma_{\text{policy}}(k, z; \theta^-_{\text{policy}})$.

The target is detached from the gradient (treated as a constant label).

Critic Loss

$$\mathcal{L}^{\text{BR}}_{\text{critic}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} (V_i - y_{i,1})(V_i - y_{i,2})$$

where $V_i = \Gamma_{\text{value}}(k_i, z_i; \theta_{\text{value}})$.

5.4.2 Actor Update

The actor (policy network) maximizes the Bellman RHS:

$$\max_{\theta_{\text{policy}}} \mathbb{E}\left[e(k, k', z) + \beta V(k', z')\right]$$

Actor Loss

$$\mathcal{L}^{\text{BR}}_{\text{actor}} = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left[e(k_i, k'_i, z_i) + \beta \Gamma_{\text{value}}(k'_i, z'_{i,1}; \theta_{\text{value}})\right]$$

where $k'_i = \Gamma_{\text{policy}}(k_i, z_i; \theta_{\text{policy}})$ uses the current (not target) policy.

Only the main shock z'_1 is used (sample mean estimator, not cross-product).

5.4.3 Algorithm

1. Load flattened batch (k, z, z'_1, z'_2)
2. **Critic step** (repeat N_{critic} times per actor step):
   - Compute target y_ℓ using target networks
   - Update θ_{value} to minimize $\mathcal{L}^{\text{BR}}_{\text{critic}}$
   - Polyak-update θ^-_{value}
3. **Actor step**:
   - Compute $k' = \Gamma_{\text{policy}}(k, z; \theta_{\text{policy}})$
   - Update θ_{policy} to minimize $\mathcal{L}^{\text{BR}}_{\text{actor}}$
   - Polyak-update θ^-_{policy}

5.5 Gradient Flow and Target Networks

A common source of bugs in implementing ER and BR methods is incorrect handling of gradient flow. This section clarifies when to detach gradients and why target networks with Polyak averaging are essential.

5.5.1 The Problem: Recursive Dependencies

In both ER and BR methods, the loss depends on future quantities computed by the same network being trained:

- **ER**: The residual f depends on $k'' = \Gamma_{\text{policy}}(k', z'; \theta)$, where k' itself depends on θ
- **BR Critic**: The target y depends on $V(k', z') = \Gamma_{\text{value}}(k', z'; \theta_{\text{value}})$
- **BR Actor**: The objective depends on $V(k', z')$ evaluated at the actor's chosen k'

If gradients flow through these future quantities naively, training becomes unstable: the network chases a moving target that changes with every parameter update.

5.5.2 Solution: Target Networks with Gradient Detachment

The solution has two components:

1. **Target networks** (θ^-): Maintain a slowly-updating copy of the trainable parameters
2. **Gradient detachment**: Stop gradients from flowing through the target computation

Why both are needed: Detachment alone (using the current θ but stopping gradients) still causes the target to shift every step. Target networks provide a stable reference that changes slowly via Polyak averaging.

5.5.3 Polyak Averaging

After each training step, the target network is updated as a weighted average:

$$\theta^- \leftarrow \nu \theta^- + (1 - \nu)\, \theta$$

where $\nu \in (0.99, 0.999)$ is the averaging coefficient. With $\nu = 0.995$ (default), the target network tracks the current network but with significant lag, providing stability.

5.5.4 Gradient Flow Rules by Method

ER Method

Computation	Network	Gradient?	Rationale
$k' = \Gamma_{\text{policy}}(k, z; \theta)$	Current	Yes	This is what we're optimizing
$k'' = \Gamma_{\text{policy}}(k', z'; \theta^-)$	Target	No	Stable reference for future action

The residual $f(k, k', k'', z')$ has gradients only w.r.t. k'. If we accidentally used the current policy for k'', the gradient would try to adjust the policy to make k'' satisfy the Euler equation—but k'' is a future decision that should be determined by the *current* state (k', z'), not by backpropagation.

BR Critic Update

Computation	Network	Gradient?	Rationale
$V = \Gamma_{\text{value}}(k, z; \theta_{\text{value}})$	Current	Yes	This is what we're fitting
$k' = \Gamma_{\text{policy}}(k, z; \theta^-_{\text{policy}})$	Target	No	Fixed action for target
$y = e + \beta\Gamma_{\text{value}}(k', z'; \theta^-_{\text{value}})$	Target	No	Fixed label (regression target)

The critic update is a regression: fit $V(k, z)$ to the target y. The target must be treated as a constant label (like supervised learning), otherwise the critic could "cheat" by adjusting the target rather than improving its prediction.

BR Actor Update

Computation	Network	Gradient?	Rationale
$k' = \Gamma_{\text{policy}}(k, z; \theta_{\text{policy}})$	Current	Yes	This is what we're optimizing
$V(k', z') = \Gamma_{\text{value}}(k', z'; \theta_{\text{value}})$	Current	No	Frozen value landscape

The actor optimizes policy to maximize $e(k, k', z) + \beta V(k', z')$. Gradients flow through k' (to improve the action) but not through θ_{value} (the value function is held fixed during the actor step). If gradients flowed into θ_{value}, the actor could artificially inflate V rather than finding genuinely better actions.

5.5.5 Common Mistakes (that I've made and corrected)

1. **Forgetting to detach targets**: Results in unstable training where loss oscillates or diverges. The network optimizes by moving the target rather than improving predictions.
2. **Using current network for future actions**: In ER, computing k'' with the current policy instead of the target creates a circular dependency that destabilizes learning.
3. **Allowing gradients through value in actor update**: The actor manipulates the value function instead of finding better policies. Symptom: actor loss decreases but policy quality doesn't improve.
4. **Not using Polyak averaging**: Hard target updates (periodically copying $\theta \to \theta^-$) cause discontinuous jumps in the target, leading to training instability.

5.5.6 Implementation Pattern (TensorFlow)

```
# Critic update: detach target
with tf.GradientTape() as tape:
    V_pred = value_net(k, z)                          # Trainable
    k_next = target_policy_net(k, z)                  # Detached (no tape)
    V_next = target_value_net(k_next, z_next)         # Detached (no tape)
    y = tf.stop_gradient(e + beta * V_next)           # Explicit stop
    loss = cross_product_loss(V_pred, y)
grads = tape.gradient(loss, value_net.trainable_variables)

# Actor update: gradient through action, not through value
with tf.GradientTape() as tape:
    k_next = policy_net(k, z)                         # Trainable
    V_next = value_net(k_next, z_next)                # Used but frozen
    loss = -tf.reduce_mean(e + beta * V_next)
grads = tape.gradient(loss, policy_net.trainable_variables)  # Only policy
```

The key insight: anything computed *outside* the `GradientTape` context is automatically detached. Target networks are never inside the tape.

5.6 Handling Discontinuities

Several model components involve discontinuous indicator functions that have zero gradient almost everywhere, blocking learning signals.

5.6.1 Sources of Discontinuity

1. **Fixed adjustment cost**: $\mathbf{1}\{I \neq 0\}$ triggers for any nonzero investment
2. **External financing cost**: $\mathbf{1}\{e < 0\}$ triggers for negative cash flow
3. **Default indicator**: $\mathbf{1}\{\tilde{V} < 0\}$ triggers when continuation value is negative

5.6.2 Sigmoid Smoothing with Annealing

Replace hard indicators with temperature-controlled sigmoids that:

- Provide gradient signal during early training (high temperature)
- Converge to true indicators as temperature decreases

Adjustment Cost Gate

$$\sigma\left(\frac{|I/k| - \epsilon}{\tau}\right) \to \mathbf{1}\{|I/k| > \epsilon\} \quad \text{as } \tau \to 0$$

Normalizing by k prevents saturation for large investments.

External Financing Gate

$$\sigma\left(-\frac{e/k + \epsilon}{\tau}\right) \to \mathbf{1}\{e/k < -\epsilon\} \quad \text{as } \tau \to 0$$

Default Gate (Gumbel-Sigmoid)

The default boundary requires exploration of both solvent and default regions. A deterministic sigmoid risks getting stuck (e.g., learning "never default"). The Gumbel-Sigmoid adds stochastic exploration:

$$p = \sigma\left(\frac{-\tilde{V}/k + \log u - \log(1-u)}{\tau}\right), \quad u \sim \text{Uniform}(0,1)$$

The noise $\log u - \log(1-u)$ (logistic distribution) forces the network to explore around the default boundary when τ is large.

5.6.3 Annealing Schedule

Temperature decays exponentially:

$$\tau_j = \tau_0 \cdot d^j$$

where τ_0 is the initial temperature (default: 1.0), d is the decay rate (default: 0.995), and j is the iteration.

Annealing stops at $\tau_{\min}$ (default: 10^{-4}), after which the temperature is held constant. The number of decay steps is:

$$N_{\text{decay}} = \frac{\log(\tau_{\min}) - \log(\tau_0)}{\log(d)}$$

A stabilization buffer (default: 25%) allows fine-tuning at low temperature:

$$N_{\text{anneal}} = \lceil N_{\text{decay}} \cdot (1 + \text{buffer}) \rceil$$

5.7 Convergence and Early Stopping

The stopping logic is hierarchical: first ensure annealing is complete, then check method-specific convergence on the validation set.

5.7.1 Annealing Gatekeeper

If current step $< N_{\text{anneal}}$, ignore all early stopping triggers. This ensures the soft gates have sharpened before evaluating convergence.

5.7.2 Method-Specific Criteria

Computed on the validation set to avoid overfitting.

LR Method (Relative Improvement Plateau)

Stop when relative improvement over a window s falls below threshold:

$$\frac{|\bar{\mathcal{L}}^{\text{LR}}(j) - \bar{\mathcal{L}}^{\text{LR}}(j-s)|}{|\bar{\mathcal{L}}^{\text{LR}}(j-s)|} < \epsilon_{\text{LR}}$$

where $\bar{\mathcal{L}}$ is a moving average.

ER Method (Absolute Threshold)

Stop when loss is near zero:

$$\mathcal{L}^{\text{ER}} < \epsilon_{\text{ER}}$$

The ER loss has a known optimum of zero (Euler equation holds exactly).

BR Method (Dual Convergence)

Both conditions must be satisfied:

1. Critic accuracy: $\mathcal{L}^{\text{BR}}_{\text{critic}} < \epsilon_{\text{critic}}$
2. Actor stability: Relative improvement in actor objective $< \epsilon_{\text{actor}}$

5.7.3 Patience Counter

A patience counter tracks consecutive checks where criteria are met:

- Increment if criteria satisfied
- Reset to zero if criteria violated
- Stop when counter exceeds patience limit (default: 5)

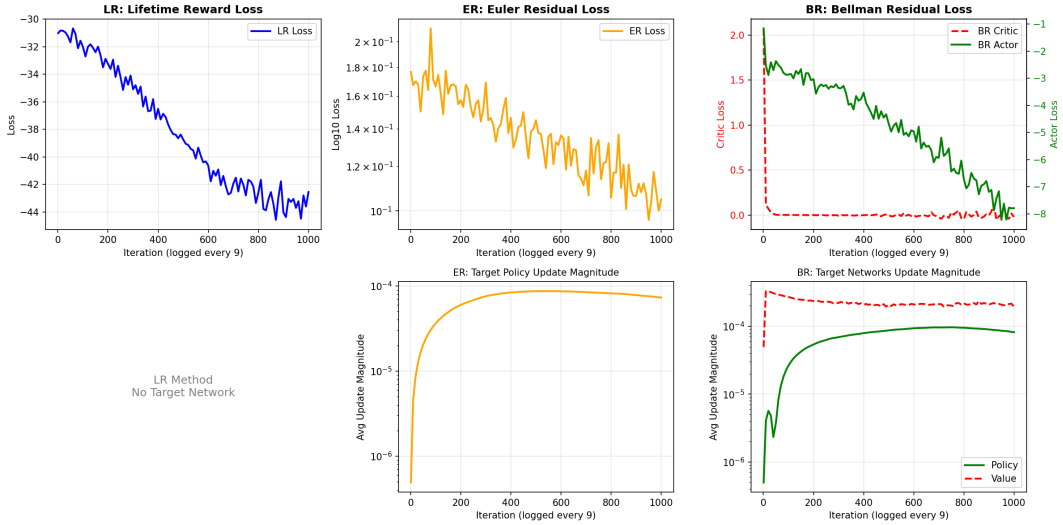

Figure 2: Training loss curves for LR, ER, and BR methods on the basic model. LR minimizes negative lifetime reward; ER and BR minimize squared residuals (shown in log scale). *Example output from debug-mode training; production results will differ.*

---

6 Risky Debt Training

6.1 Method Selection

The risky debt model can in principle use LR, ER, or BR methods, but practical considerations favor BR.

LR Method: Applicable but challenging. The nested fixed-point problem (risky rate depends on value, which depends on risky rate) is implicit in the lifetime simulation. The network must simultaneously learn the policy and the equilibrium pricing, which can be unstable without the explicit critic structure.

ER Method: Problematic. Deriving Euler equations for the risky debt model requires differentiating through the default indicator and bond pricing equilibrium. The resulting conditions are complex, and the indicator discontinuities are harder to handle without the value function providing a stable learning target.

BR Method (Recommended): The actor-critic structure naturally decomposes the equilibrium conditions:

- Critic learns value and pricing to satisfy Bellman and zero-profit conditions
- Actor optimizes policy given the learned value/pricing functions

This decomposition handles the nested fixed-point problem without explicit iteration and provides stable learning targets for each component.

6.2 BR Method for Risky Debt

6.2.1 Pricing Loss

The lender's zero-profit condition determines bond prices. Define the pricing residual:

$$f_\ell = q \cdot b' \cdot (1 + r) - [p_\ell \cdot R(k', z'_\ell) + (1 - p_\ell) \cdot b']$$

where:

- $q = \Gamma_{\text{price}}(k', b', z; \theta_{\text{price}})$ is the predicted bond price
- p_ℓ is the Gumbel-Sigmoid default probability at shock z'_ℓ
- $R(k', z')$ is the recovery value

Pricing Loss

$$\mathcal{L}^{\text{price}} = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} f_{i,1} \times f_{i,2}$$

6.2.2 Critic Update

The critic target incorporates limited liability via the smooth effective value:

$$V_{\text{eff}} = (1 - p) \cdot \tilde{V}$$

where p is the default probability. As $\tau \to 0$, this converges to $\max\{0, \tilde{V}\}$.

Bellman Target

$$y_\ell = e(\cdot) - \eta(e) + \beta \cdot V_{\text{eff}}(\Gamma_{\text{value}}(k', b', z'_\ell; \theta^-_{\text{value}}))$$

Combined Critic Loss

$$\mathcal{L}_{\text{critic}} = \omega_1 \mathcal{L}^{\text{BR}} + \omega_2 \mathcal{L}^{\text{price}}$$

where (ω_1, ω_2) are weights to balance the two loss components (tuned to match scales).

6.2.3 Actor Update

The actor maximizes expected continuation value:

$$\mathcal{L}^{\text{BR}}_{\text{actor}} = -\frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left[e(\cdot) - \eta(e) + \beta \cdot V_{\text{eff}}(\Gamma_{\text{value}}(k'_i, b'_i, z'_{i,1}; \theta_{\text{value}})) \right]$$

The smooth V_{eff} (rather than hard max) is essential here. A hard max has zero gradient in the default region, providing no signal for the actor to adjust (k', b') to exit default.

6.2.4 Algorithm Summary

1. **Critic step** (repeat N_{critic} times):
   - Compute actions $(k', b') = \Gamma_{\text{policy}}(k, b, z; \theta^-_{\text{policy}})$
   - Compute Bellman targets y_ℓ and pricing residuals f_ℓ
   - Update $(\theta_{\text{value}}, \theta_{\text{price}})$ on combined loss
   - Polyak-update target networks
2. **Actor step**:
   - Compute actions using current policy
   - Update θ_{policy} to minimize actor loss
   - Polyak-update target policy

3. **Annealing**: Decay temperature τ for default and other gates
4. **Stopping**: Check convergence after annealing completes

---

7 Implementation Details

7.1 Centralized Configuration

The codebase separates configuration into two layers to prevent circular imports and parameter drift.

Technical Defaults (`src/_defaults.py`)

A leaf module with zero imports containing all training hyperparameters:

- Learning rates, batch sizes, iteration counts
- Temperature annealing parameters
- Early stopping thresholds
- Polyak averaging coefficients

Both trainer configurations and annealing schedules import from this single source.

Economic Parameters (`src/economy/parameters.py`)

Frozen dataclasses for domain knowledge:

- `ShockParams`: AR(1) parameters (ρ, σ, μ)
- `EconomicParams`: Production and cost parameters $(\theta, r, \delta, \phi_0, \phi_1, ...)$

The `frozen=True` setting prevents accidental mutation. Changes require explicit `with_overrides()` calls that log all modifications.

7.2 Numerical Stability

Several design choices ensure stable training:

1. **Input normalization**: All states mapped to $[0, 1]$ before network processing
2. **Bounded outputs**: Sigmoids scaled to valid ranges prevent invalid states
3. **Logit clipping**: Soft gates clip extreme logits (default: ± 20) to prevent saturation
4. **Normalized indicators**: Ratios like I/k and e/k prevent scale-dependent gate behavior

7.3 Reproducibility

1. **Stateless RNG**: All randomness uses `tf.random.stateless_*` functions
2. **Seed scheduling**: Deterministic mapping from master seed to per-variable, per-step seeds
3. **Configuration hashing**: Cache files include MD5 hash of configuration for invalidation
4. **Metadata logging**: All generated datasets store their configuration for verification

---

8 Solution Evaluation

8.1 Verification Strategies

Since analytical solutions generally do not exist, verification uses three approaches:

1. **Cross-method consistency**: Policies learned by LR, ER, and BR should be similar when trained on the same data
2. **Economic sanity checks**: Policies should exhibit expected qualitative behavior

3. **Analytical benchmark**: Special cases with closed-form solutions (e.g., no adjustment costs)

8.2 Basic Model Results

For the basic model, we visualize policies along two dimensions:

- k' vs. k at fixed $\ln z = \mu$ (capital dynamics)
- k' vs. $\ln z$ at fixed $k = k^*$ (response to productivity)

All three methods should produce overlapping policies that cross the 45-degree line at approximately the same steady-state capital level. See Figure 3, Figure 4, and Figure 5 for policy comparisons across scenarios, and Figure 6 for the 3D policy surface.

8.3 Risky Debt Model Results

The risky debt model produces joint policies for capital and borrowing (k', b'). See Figure 7 for 2D policy slices and Figure 8 for 3D policy surfaces.

8.4 Analytical Benchmark

With zero adjustment costs ($\psi = 0$), the Euler equation simplifies to:

$$\mathbb{E}[\theta z'(k')^{\theta-1} \mid z] = r + \delta$$

Using properties of the lognormal distribution:

$$\mathbb{E}[z' \mid z] = \exp\left((1-\rho)\mu + \rho \ln z + \frac{1}{2}\sigma^2\right)$$

This yields the analytical policy:

$$k'(z) = \left[\frac{\theta \exp\left((1-\rho)\mu + \rho \ln z + \frac{1}{2}\sigma^2\right)}{r + \delta}\right]^{\frac{1}{1-\theta}}$$

Key properties of this benchmark:

- k' is independent of current capital k
- k' increases in z when $\rho > 0$

Learned policies should match this solution in the zero-adjustment-cost case (Figure 9).

8.5 Figures

> **Note**
>
> **Note on figures**: The results below are from debug-mode training with small batch sizes and limited iterations. They demonstrate the methodology but not converged solutions. Production-quality results will be generated after full training runs.

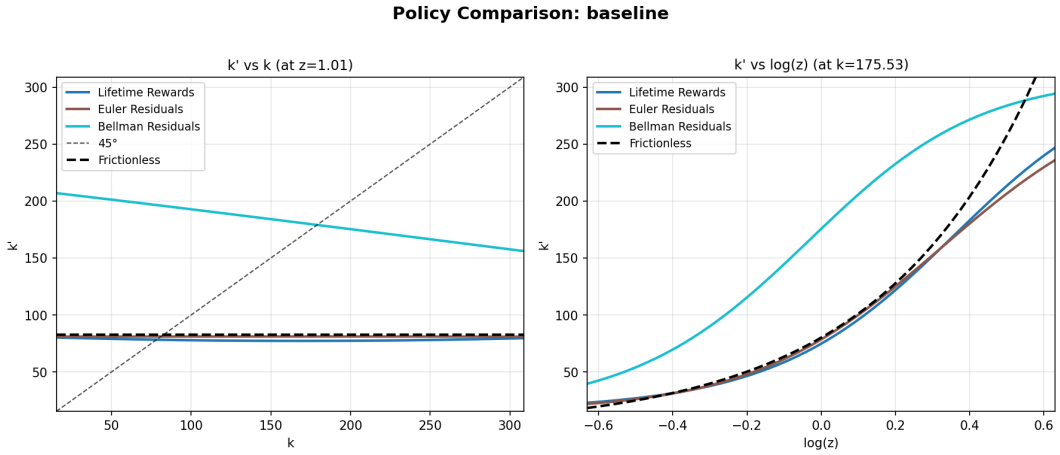

Figure 3: Policy comparison at baseline parameters (no adjustment costs). All three methods converge to similar policies.

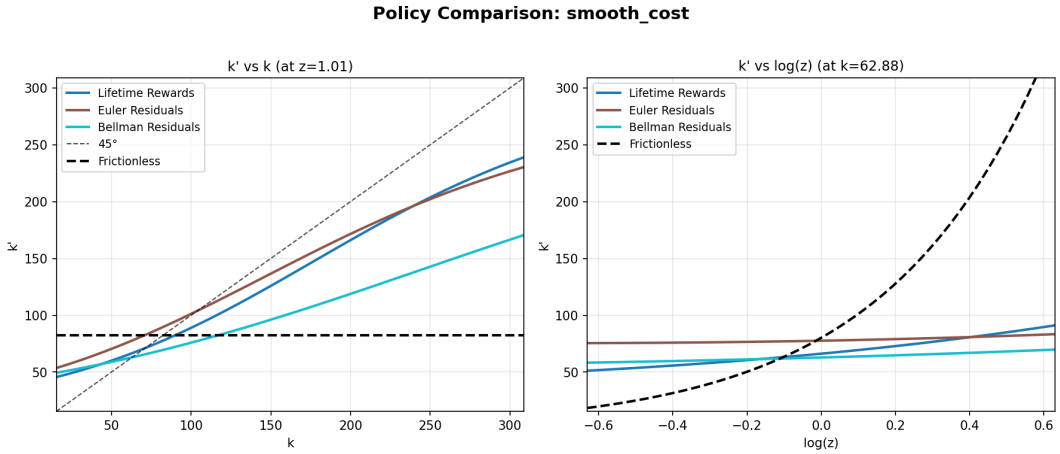

Figure 4: Policy comparison with convex adjustment costs ($\phi_0 > 0$). The smooth cost creates gradual adjustment toward steady state.

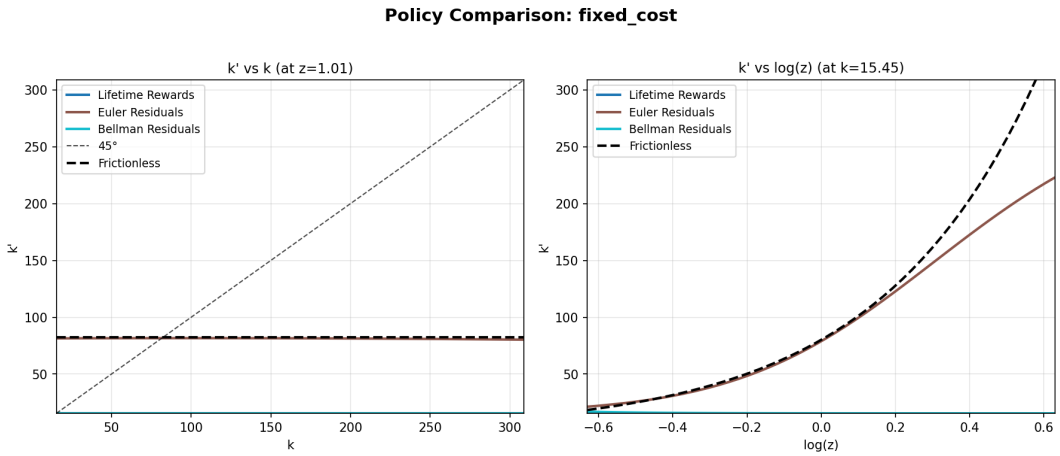

Figure 5: Policy comparison with fixed adjustment costs ($\phi_1 > 0$). The fixed cost creates an inaction region near the 45-degree line.

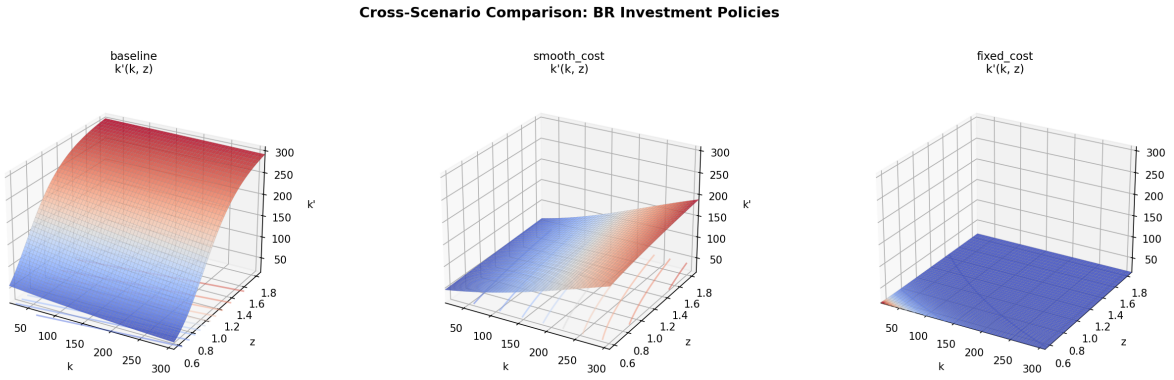

Figure 6: 3D visualization of optimal capital policy $k'(k, z)$ across all three scenarios. The surface shows how policy responds jointly to current capital and productivity.

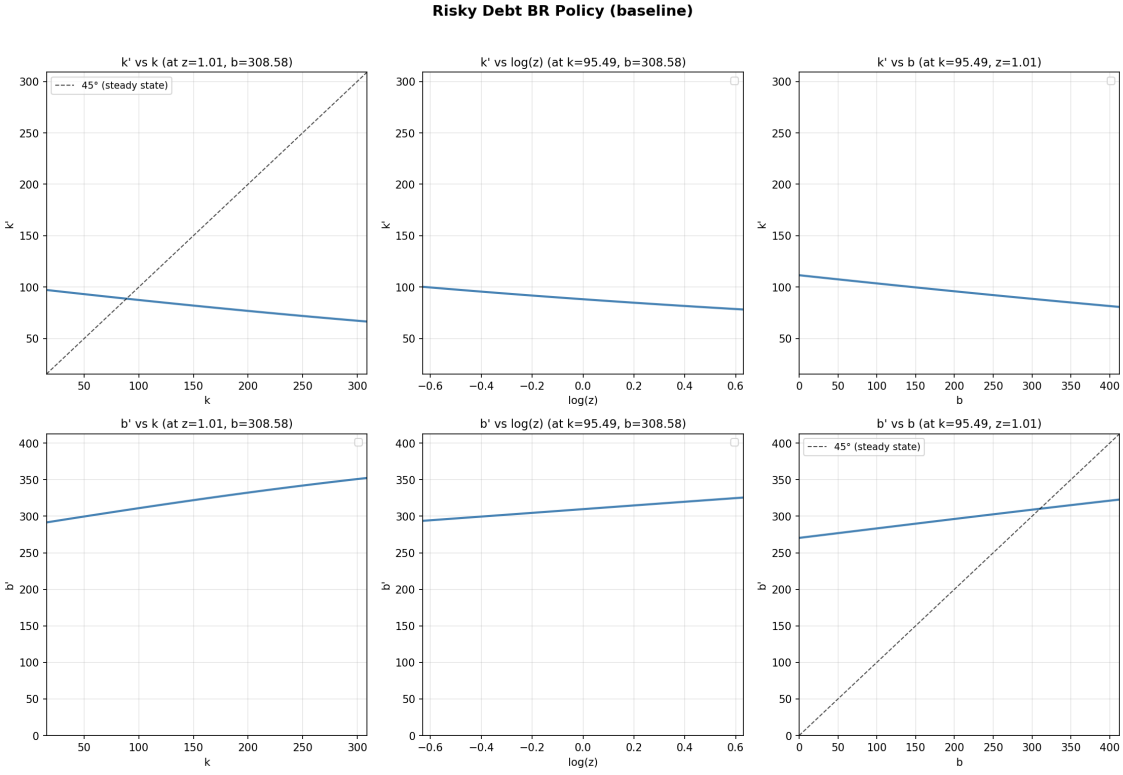

Figure 7: Risky debt BR policy: optimal capital k' and borrowing b' as functions of current state. Left panels show response to capital; right panels show response to productivity.

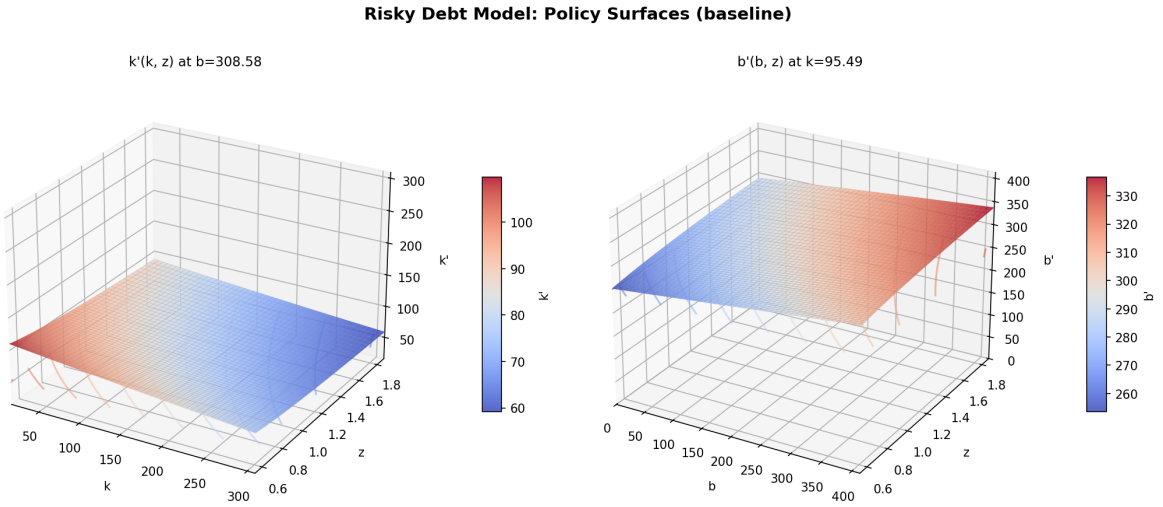

Figure 8: 3D policy surfaces for the risky debt model showing $k'(k, b, z)$ and $b'(k, b, z)$ at fixed productivity levels.

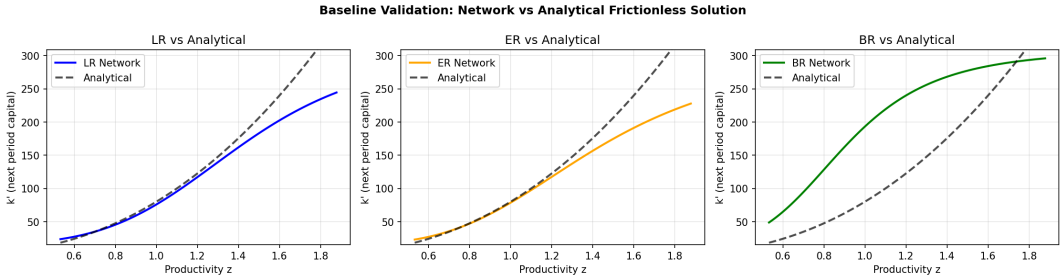

Figure 9: Validation against analytical solution at baseline (no adjustment costs). Learned policies from all three methods are compared against the closed-form Euler solution. Close agreement validates the implementation. *Debug-mode results; full training will improve accuracy.*