Machine Learning for Computational Economics

Module 01: Introduction

EDHEC Business School

Dejanir Silva

Purdue University

January 2026

About me

- Brazilian economist
  - Masters from University of Sao Paulo
  - PhD in Economics from MIT
- Research on *macro-finance* topics
  - Monetary policy / financial intermediation
  - Applications of ML to macro-finance
- Professor at Purdue University 🌐
  - Previously, Princeton and UIUC

The need for high-dimensional dynamic problems

Dynamic programming is one of the cornerstones of modern economics:

- It is a central part of how we tackle many of the biggest topics in our field
- e.g.: consumption and firm behavior, asset prices, climate economics, public finance, etc.

The need for high-dimensional dynamic problems

Dynamic programming is one of the cornerstones of modern economics:

- It is a central part of how we tackle many of the biggest topics in our field
- e.g.: consumption and firm behavior, asset prices, climate economics, public finance, etc.

Need to solve **high-dimensional** dynamic problems with many:

- **investors / assets**
- countries / regions
- consumers / goods

The need for high-dimensional dynamic problems

Dynamic programming is one of the cornerstones of modern economics:

- It is a central part of how we tackle many of the biggest topics in our field
- e.g.: consumption and firm behavior, asset prices, climate economics, public finance, etc.

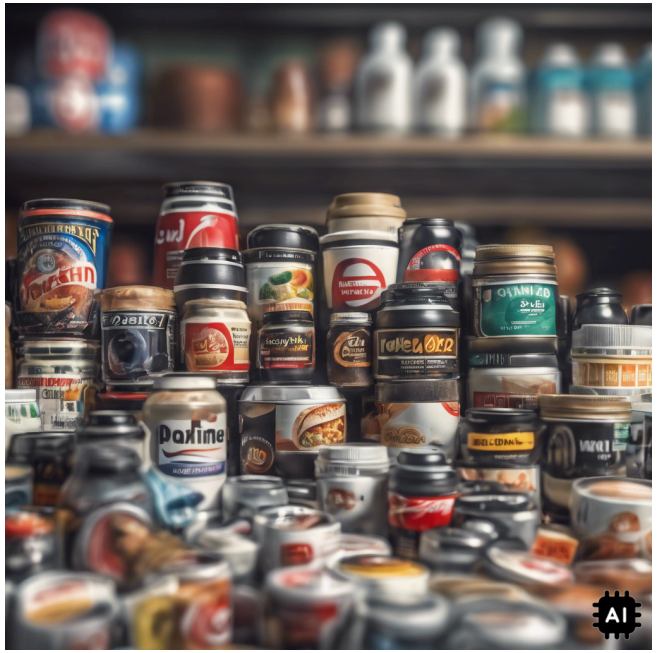
Need to solve **high-dimensional** dynamic problems with many:

- investors / assets
- **countries / regions**
- consumers / goods

The need for high-dimensional dynamic problems

Dynamic programming is one of the cornerstones of modern economics:

- It is a central part of how we tackle many of the biggest topics in our field
- e.g.: consumption and firm behavior, asset prices, climate economics, public finance, etc.



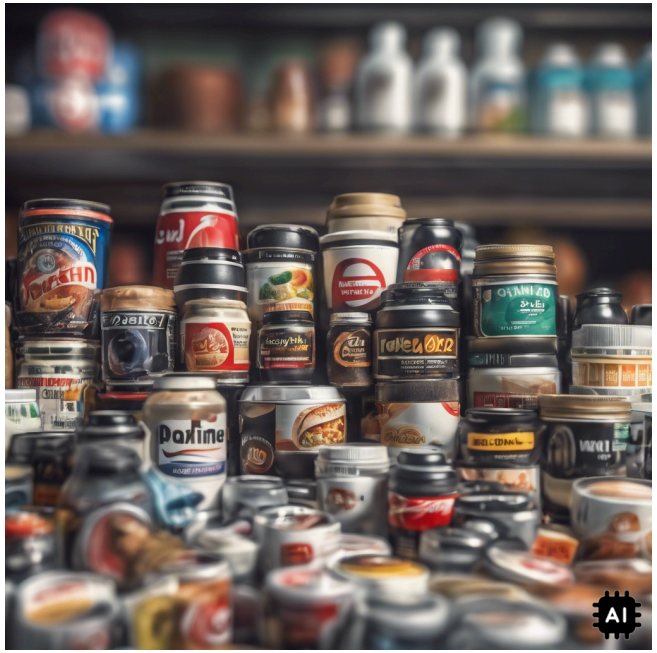
Need to solve **high-dimensional** dynamic problems with many:

- investors / assets
- countries / regions
- **consumers / goods**

The need for high-dimensional dynamic problems

Dynamic programming is one of the cornerstones of modern economics:

- It is a central part of how we tackle many of the biggest topics in our field
- e.g.: consumption and firm behavior, asset prices, climate economics, public finance, etc.



Need to solve **high-dimensional** dynamic problems with many:

- investors / assets
- countries / regions
- **consumers / goods**

Takeaway: realistic models *must* be high-dimensional.

The three curses of dimensionality

Challenge: problem is plagued by the **curse of dimensionality** Powell (2011)

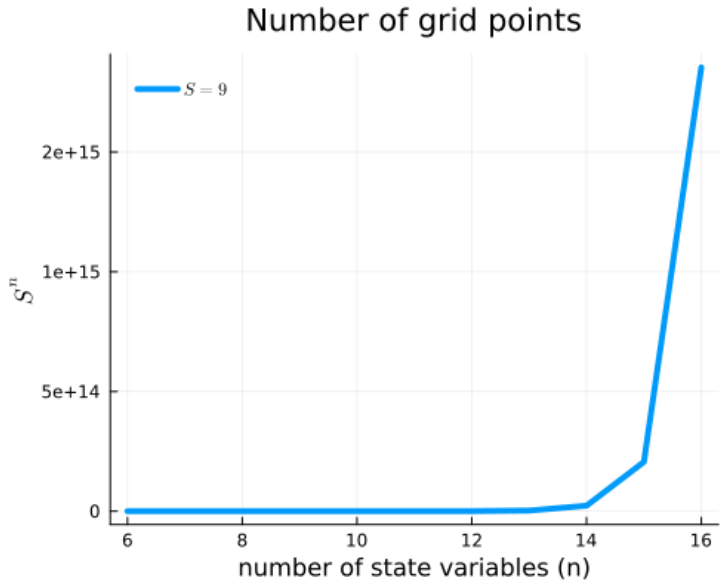
- Computational cost grows *exponentially* with dimensionality
- This curse appears in three forms — each requiring a different solution.

The three curses of dimensionality

Challenge: problem is plagued by the **curse of dimensionality** Powell (2011)

- Computational cost grows *exponentially* with dimensionality
- This curse appears in three forms — each requiring a different solution.

1) The curse of representation

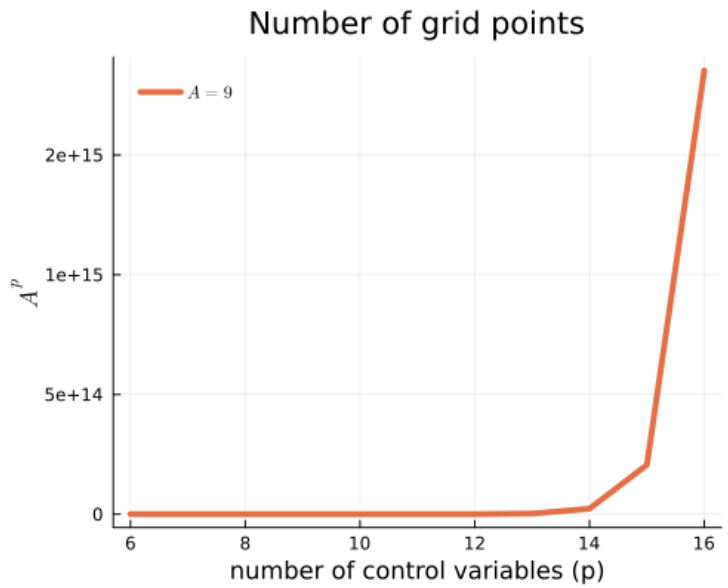


The three curses of dimensionality

Challenge: problem is plagued by the **curse of dimensionality** Powell (2011)

- Computational cost grows *exponentially* with dimensionality
- This curse appears in three forms — each requiring a different solution.

2) The curse of optimization

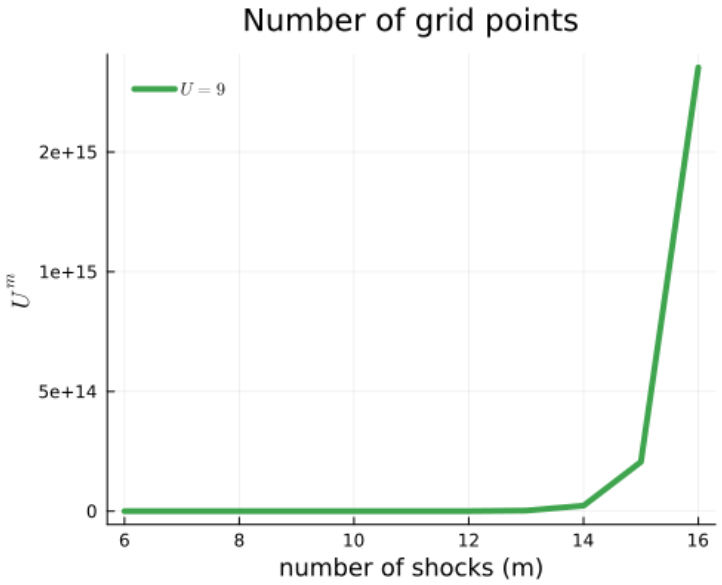


The three curses of dimensionality

Challenge: problem is plagued by the **curse of dimensionality** Powell (2011)

- Computational cost grows *exponentially* with dimensionality
- This curse appears in three forms — each requiring a different solution.

3) The curse of expectation



The continuous-time problem

Optimal control problem for an infinitely-lived agent:

- States: $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n$

The continuous-time problem

Optimal control problem for an infinitely-lived agent:

- States: $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n$
- Controls: $\mathbf{c} \in \Gamma(\mathbf{s}) \subset \mathbb{R}^p$

The continuous-time problem

Optimal control problem for an infinitely-lived agent:

- States: $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n$
- Controls: $\mathbf{c} \in \Gamma(\mathbf{s}) \subset \mathbb{R}^p$
- Shocks: $\mathbf{B}_t \in \mathbb{R}^m$

The continuous-time problem

Optimal control problem for an infinitely-lived agent:

- States: $\mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n$
- Controls: $\mathbf{c} \in \Gamma(\mathbf{s}) \subset \mathbb{R}^p$
- Shocks: $\mathbf{B}_t \in \mathbb{R}^m$

Optimal control problem:

$$
\begin{aligned}
V(\mathbf{s}) &= \max_{\{\mathbf{c}_t\}_{t=0}^{\infty}} \mathbb{E}\left[\int_0^{\infty} e^{-\rho t} u(\mathbf{c}_t) dt \,\middle|\, \mathbf{s}\right] \\
\text{subject to} \quad d\mathbf{s}_t &= \underbrace{\mathbf{f}(\mathbf{s}_t, \mathbf{c}_t)}_{\text{drift } (n\times 1)} dt + \underbrace{\mathbf{g}(\mathbf{s}_t, \mathbf{c}_t)}_{\text{diffusion } (n\times m)} d\mathbf{B}_t \\
&\mathbf{c}_t \in \Gamma(\mathbf{s}_t), \forall t \in [0, \infty),
\end{aligned}
$$

given $\mathbf{s}_0 = \mathbf{s}$.

Example: NGM with O-U productivity shocks

Example: neoclassical growth model (NGM)

- Ornstein-Uhlenbeck productivity process:

$$dA_t = -\phi(A_t - \overline{A})dt + \sigma_A dB_t.$$

Mapping to general problem:

- $\mathbf{s}_t = (k_t, A_t)'$
- $\mathbf{c}_t = c_t$
- $\mathbf{f}(\mathbf{s}_t, \mathbf{c}_t) = [A_t k_t^\alpha - \delta k_t - c_t, -\phi(A_t - \overline{A})]'$
- $\mathbf{g}(\mathbf{s}_t, \mathbf{c}_t) = [0, \sigma_A]'$

Formulation is quite flexible

- It accommodates multiple controls (e.g. multiple assets/goods) or shocks
- Duarte, Duarte, Silva (2024) shows a range of macro-finance problems fit this framework

The discrete-time problem

Discrete-time problem:

$$V(\mathbf{s}) = \max_{\{\mathbf{c}_t\}_{t=0}^{\infty}} \mathbb{E}\left[\sum_{k=0}^{\infty} e^{-\rho k \Delta_t} u(\mathbf{c}_k)\Delta_t \middle| \mathbf{s}\right]$$

subject to

$$\begin{aligned} \mathbf{s}_{t+1} - \mathbf{s}_t &= \mathbf{f}(\mathbf{s}_t, \mathbf{c}_t)\Delta_t + \mathbf{g}(\mathbf{s}_t, \mathbf{c}_t)\sqrt{\Delta_t}\, u_{t+1} \\ \mathbf{c}_t &\in \Gamma(\mathbf{s}_t), \forall t \in \{0, 1, \dots, \infty\}. \end{aligned}$$

Bellman equation:

$$V(\mathbf{s}) = \max_{\mathbf{c}\in\Gamma(\mathbf{s})} u(\mathbf{c})\Delta_t + e^{-\rho\Delta_t}\mathbb{E}\left[V(\mathbf{s}')|\mathbf{s}\right],$$

where $\mathbf{s}' = \mathbf{s} + \mathbf{f}(\mathbf{s}, \mathbf{c})\Delta_t + \mathbf{g}(\mathbf{s}, \mathbf{c})\sqrt{\Delta_t}\, u'$.

VFI and PFI

Value Function Iteration (VFI):

- Given initial guess $V^0(\mathbf{s})$, iterate until convergence:

$$V^j(\mathbf{s}) = \max_{\mathbf{c}\in\Gamma(\mathbf{s})} u(\mathbf{c})\Delta_t + e^{-\rho\Delta_t}\,\mathbb{E}\left[V^{j-1}(\mathbf{s}')|\mathbf{s}\right] \equiv TV^{j-1}(\mathbf{s})$$

Policy Function Iteration (PFI):

- Given initial guess for policy function $\pi(\mathbf{s})$, perform policy evaluation step:

$$V_{\boldsymbol{\pi}}(\mathbf{s}) = u(\boldsymbol{\pi}(\mathbf{s}))\Delta_t + e^{-\rho\Delta_t}\,\mathbb{E}\left[V_{\boldsymbol{\pi}}(\mathbf{s}')\right] \equiv TV_{\boldsymbol{\pi}}(\mathbf{s}).$$

- Then, perform policy improvement step

$$\pi'(\mathbf{s}) = \arg\max_{\mathbf{c}} \left\{u(\mathbf{c})\Delta_t + e^{-\rho\Delta_t}\,\mathbb{E}\left[V_{\boldsymbol{\pi}}(\mathbf{s}')\right]\right\},$$

  and iterate until convergence.

The first curse

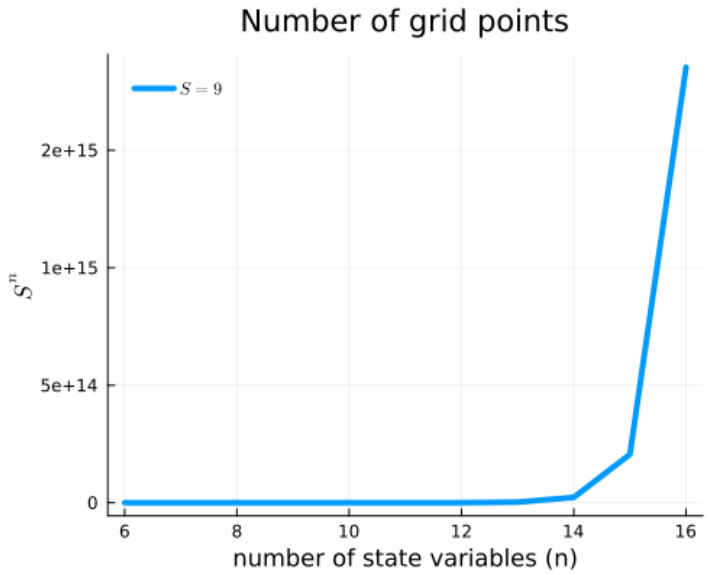
Suppose that each state variable takes a finite number of values S

- # elements in state space: $|\mathcal{S}| = S^n$

The first curse

Suppose that each state variable takes a finite number of values S

- # elements in state space: $|\mathcal{S}| = S^n$



Example: multi-region RBC

- Model with a region for each US state: $j = 1, \ldots, 50$
- Regional state variables: $(K_{jt}, A_{jt}) \Rightarrow$ total $n = 100$ state variables
- If $S = 10$, then $|\mathcal{S}| = 10^{100}$

A challenge even for sparse grid methods such as Smolyak

- Needed: a parsimonious way of representing $V(\mathbf{s})$

1) The curse of representation

> **"The state space explodes so quickly that even representing $V(\mathbf{s})$ becomes impossible."**

The second curse

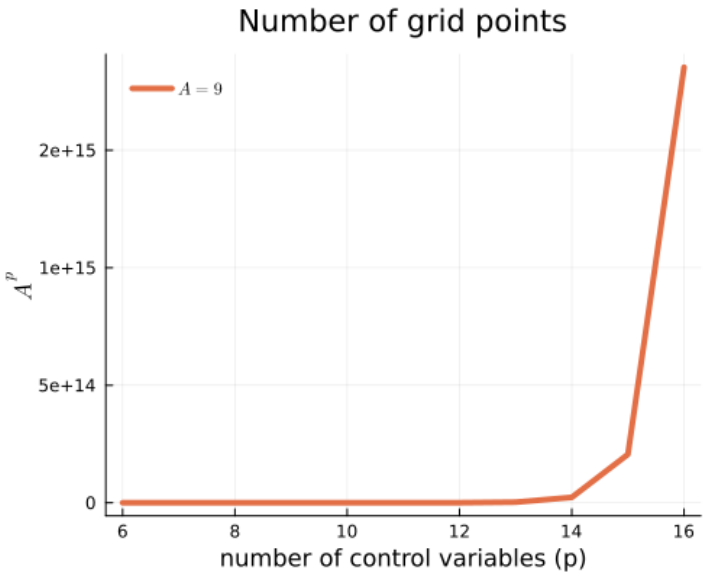
First-order conditions:

$$\nabla_{\mathbf{c}} u(\mathbf{c}_{\pi}(\mathbf{s})) + e^{-\rho \Delta t} \mathbb{E}\left[\nabla_{\mathbf{c}} V_{\pi}(\mathbf{s} + \mathbf{f}(\mathbf{s}, \mathbf{c}_{\pi}(\mathbf{s}))\Delta t + \mathbf{g}(\mathbf{s}, \mathbf{c}_{\pi}(\mathbf{s}))\sqrt{\Delta_t} u')\right] = 0,$$

The second curse

First-order conditions:

$$\nabla_{\mathbf{c}} u(\mathbf{c}_{\pi}(\mathbf{s})) + e^{-\rho \Delta t} \mathbb{E}\left[\nabla_{\mathbf{c}} V_{\pi}(\mathbf{s} + \mathbf{f}(\mathbf{s}, \mathbf{c}_{\pi}(\mathbf{s}))\Delta t + \mathbf{g}(\mathbf{s}, \mathbf{c}_{\pi}(\mathbf{s}))\sqrt{\Delta_t} u')\right] = 0,$$



Brute-force maximization infeasible with many controls

- Total number of control vectors: $|\square| = A^p$

The alternative is a costly non-linear **root-finding** step

- Endogenous gridpoint method (EGM) avoids such step
  - But only in the case of a single control
- Needed: algorithm that avoids root-finding at every step

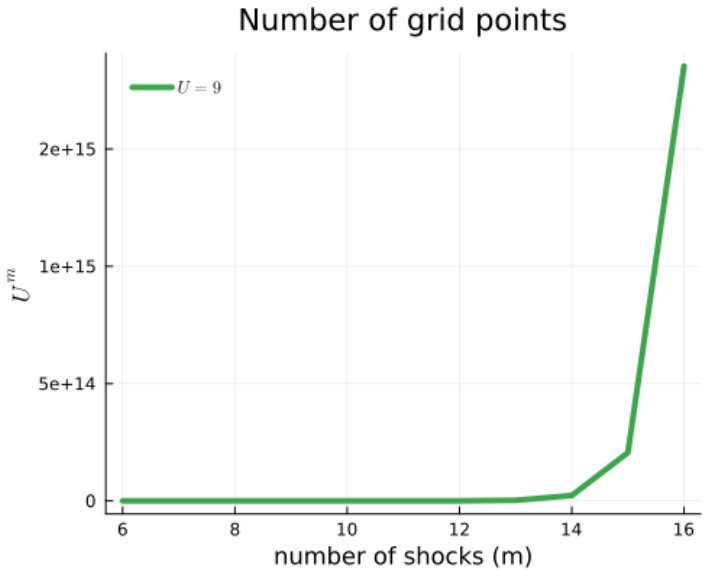
2) The curse of optimization

> **"Solving the FOCs becomes a high-dimensional nonlinear root-finding problem."**

The third curse

When u_{t+1} is a m–dimensional vector of standard normal r.v., $\mathbb{E}[V_\pi(\mathbf{s}')]$ corresponds to the integral:

$$\mathbb{E}[V_\pi(\mathbf{s}')] = \int_{u_1} \int_{u_2} \dots \int_{u_m} V_\pi(\mathbf{s}')\phi(\mathbf{u}) du_1 du_2 \dots du_m.$$



Evaluating integral is very costly with large number of shocks

- Cost of quadrature solution increases **exponentially** with m
- With U possible values, the total number of points is $|\square| = U^m$

Consider multi-region RBC example:

- With a shock for every region, we have $m = 50$
- Needed: an efficient way to compute expectations

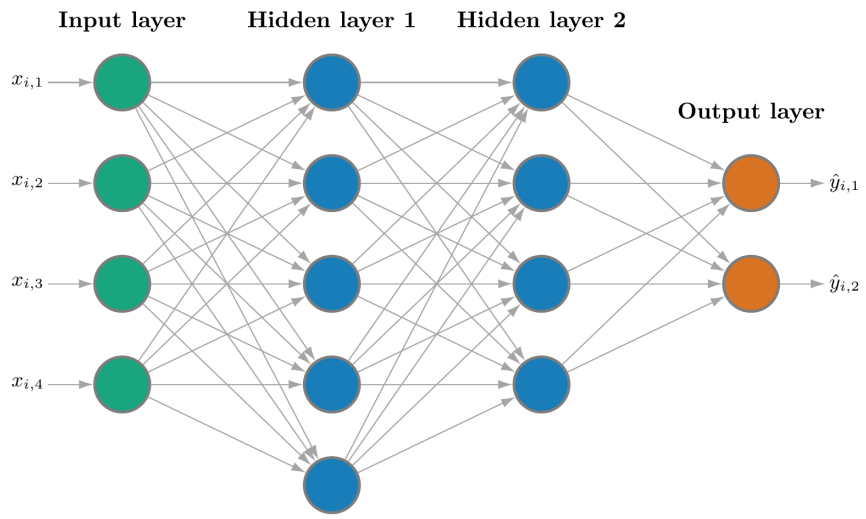
3) The curse of expectation

> **"Computing $\mathbb{E}[V(\mathbf{s}')]$ requires integrating over a huge shock space."**

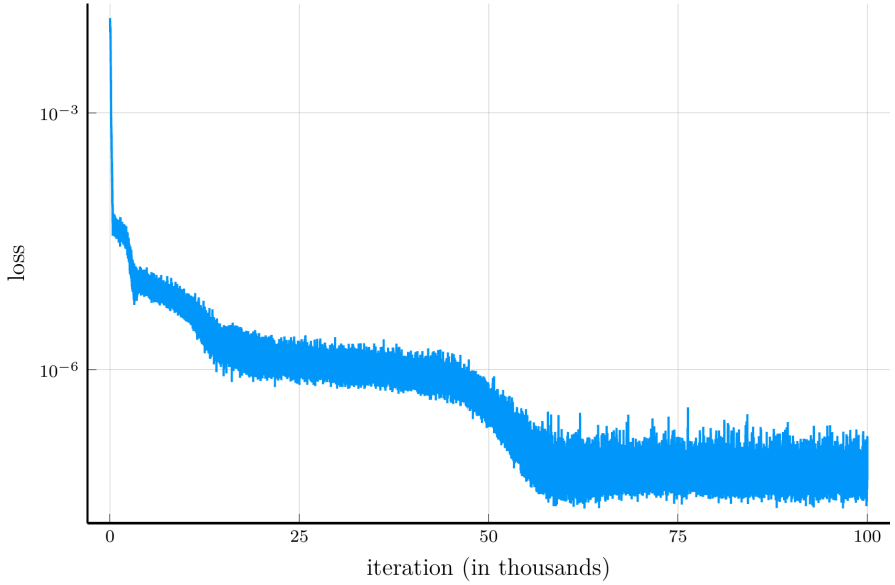
Overcoming the curses

The goal of this course is to show how to overcome the three curses of dimensionality

- The key will be to use **machine learning** techniques to handle each of the curses
- We will learn how to represent functions, train models, and how to use automatic differentiation



Deep neural network architecture



Training history

Overview of the course

Besides this introductory module, the course is organized into four modules.

- The first two modules cover *classical numerical methods*
- The last two modules cover *machine learning techniques*

Overview of the course

Besides this introductory module, the course is organized into four modules.

- The first two modules cover *classical numerical methods*
- The last two modules cover *machine learning techniques*

Module 02: Discrete-Time Methods

- Tauchen's discretization method
- Value function iteration
- Endogenous gridpoint method

Module 03: Continuous-Time Methods

- Finite-difference methods
- Stability/consistency/monotonicity
- Spectral methods

Overview of the course

Besides this introductory module, the course is organized into four modules.

- The first two modules cover *classical numerical methods*
- The last two modules cover *machine learning techniques*

Module 02: Discrete-Time Methods

- Tauchen's discretization method
- Value function iteration
- Endogenous gridpoint method

Module 03: Continuous-Time Methods

- Finite-difference methods
- Stability/consistency/monotonicity
- Spectral methods

Module 04: Fundamentals of Machine Learning

- Supervised learning and neural networks
- Optimization algorithms
- Automatic differentiation

Module 05: The Deep Policy Iteration (DPI) Method

- Hyper-dual approach to Itô's lemma
- Deep policy iteration algorithm
- Applications

The Julia programming language

The course is meant to be practical and hands-on.

- The course will teach you the theory
- But also focus on the practical implementation

The Julia programming language

The course is meant to be practical and hands-on.

- The course will teach you the theory
- But also focus on the practical implementation

We will use the **Julia programming language** to implement the methods discussed in the course.

- Julia is a modern, high-level, high-performance programming language
- Great resource to learn more about Julia: Quantitative Economics with Julia

The Julia programming language

The course is meant to be practical and hands-on.

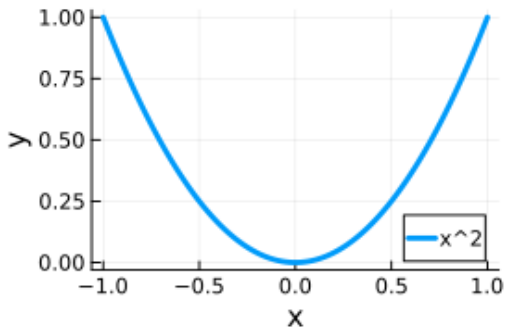
- The course will teach you the theory
- But also focus on the practical implementation

We will use the **Julia programming language** to implement the methods discussed in the course.

- Julia is a modern, high-level, high-performance programming language
- Great resource to learn more about Julia: Quantitative Economics with Julia

Here is an example of a Julia code block:

```julia
using Plots
x = range(-1, 1, length = 100)
y = x .^ 2
plot(x, y, label = "x^2", line = 3, xlabel = "x", ylabel = "y", size = (300, 200))
```



Conclusion

There is a large gap between the models we want to solve and the models classical methods can handle.

- Classical numerical methods are limited by the three curses of dimensionality
- Related to how to represent functions, how to find optimal decisions, and how to compute expectations

This course introduces techniques to allow us to bridge this gap.

- We will learn how to use machine learning methods to overcome the three curses
- We will learn how to use these methods to solve models that were previously intractable

The course will be a mix of theory and practice.

- We will learn the theory behind the methods
- But also focus on the practical implementation using Julia.

References

Duarte, Duarte, Silva.(2024). Machine learning for continuous-time finance. *The Review of Financial Studies*. 37. (11). :3217–3271 retrieved, from https://academic.oup.com/rfs/article/37/11/3217/7749384

Powell.(2011). Approximate dynamic programming: Solving the curses of dimensionality.