

# ML in Economics and Finance: Where Do We Go Now? - Part II

---

Raul Riva

FGV EPGE

December, 2025

INSPER - São Paulo



- 1. What is ML, anyway?
  - 2. Causality in High Dimensions
  - 3. (Seriously) Heterogeneous Partial Effects
  - 4. Solving Large-Scale Dynamic Models
- } Yesterday
- } Today

## Heterogeneous Partial Effects

---

## Motivation

Let  $Y$  be an outcome and  $X, Z$  be features (covariates). We frequently want to approximate

$$h(x, z) \equiv \mathbb{E}[Y|X = x, Z = z]$$

and the **partial effects**

$$\frac{\partial}{\partial x} h(x, z) = \frac{\partial}{\partial x} \mathbb{E}[Y|X = x, Z = z].$$

## Motivation

Let  $Y$  be an outcome and  $X, Z$  be features (covariates). We frequently want to approximate

$$h(x, z) \equiv \mathbb{E}[Y|X = x, Z = z]$$

and the **partial effects**

$$\frac{\partial}{\partial x} h(x, z) = \frac{\partial}{\partial x} \mathbb{E}[Y|X = x, Z = z].$$

- This is a prediction problem after all!

Let  $Y$  be an outcome and  $X, Z$  be features (covariates). We frequently want to approximate

$$h(x, z) \equiv \mathbb{E}[Y|X = x, Z = z]$$

and the **partial effects**

$$\frac{\partial}{\partial x} h(x, z) = \frac{\partial}{\partial x} \mathbb{E}[Y|X = x, Z = z].$$

- This is a prediction problem after all!
- Approach 1: impose a parametric model for  $h$ , e.g. linear regression. Pros and cons?

Let  $Y$  be an outcome and  $X, Z$  be features (covariates). We frequently want to approximate

$$h(x, z) \equiv \mathbb{E}[Y|X = x, Z = z]$$

and the **partial effects**

$$\frac{\partial}{\partial x} h(x, z) = \frac{\partial}{\partial x} \mathbb{E}[Y|X = x, Z = z].$$

- This is a prediction problem after all!
- Approach 1: impose a parametric model for  $h$ , e.g. linear regression. Pros and cons?
- Approach 2: use fully nonparametric methods. Pros and cons?

Let  $Y$  be an outcome and  $X, Z$  be features (covariates). We frequently want to approximate

$$h(x, z) \equiv \mathbb{E}[Y|X = x, Z = z]$$

and the **partial effects**

$$\frac{\partial}{\partial x} h(x, z) = \frac{\partial}{\partial x} \mathbb{E}[Y|X = x, Z = z].$$

- This is a prediction problem after all!
- Approach 1: impose a parametric model for  $h$ , e.g. linear regression. Pros and cons?
- Approach 2: use fully nonparametric methods. Pros and cons?
- The third way is the charm: a bit of structure, a bit of ML!

## Example I - Heterogenous Treatment Effects

---

- Outcomes  $Y_i$  depend on a treatment  $X_i \in \mathbb{R}$  and covariates  $Z_i \in \mathbb{R}^p$ ;
- The dose  $X_i$  depends on observables  $Z_i$ ;
- Potential outcomes  $Y_i(x)$  for each dose  $x \in \mathbb{R}$ ;

## Example I - Heterogenous Treatment Effects

---

- Outcomes  $Y_i$  depend on a treatment  $X_i \in \mathbb{R}$  and covariates  $Z_i \in \mathbb{R}^p$ ;
- The dose  $X_i$  depends on observables  $Z_i$ ;
- Potential outcomes  $Y_i(x)$  for each dose  $x \in \mathbb{R}$ ;
- The conditional average effect of increasing the dose is  $\tau(x, z) \equiv \frac{\partial}{\partial x} \mathbb{E}[Y(x)|Z = z]$ ;
- If  $\mathbb{E}[Y(x)|X = x, Z = z] = \mathbb{E}[Y(x)|Z = z]$  (common assumption in the literature), then

$$\tau(x, z) = \frac{\partial}{\partial x} \mathbb{E}[Y|X = x, Z = z] = \frac{\partial h(x, z)}{\partial x}$$

## Example II - Grouped Heterogeneity

Consider the following model:

$$Y_i = \alpha(Z_i) + X'_i \beta + \varepsilon_i, \quad \mathbb{E}[\varepsilon_i | X_i, Z_i] = 0,$$

- $X_i$  affects  $Y_i$  homogeneously;
- Intercept  $\alpha(Z_i)$  varies with  $Z_i$ , maybe in a highly nonlinear way;
- Since  $Z_i$  and  $X_i$  can be correlated, this can affect inference about  $\beta$ ;
- [Bonhomme and Manresa \(2015\)](#) studied how democracy affects national income using this model;
- In their case:  $\alpha(Z_i)$  is constant across groups but  $Z_i$  defines membership;
- In our notation:  $h(z, x) = \alpha(z) + x' \beta$

## How can we balance flexibility and interpretability?

---

🇧🇷 Masini and Medeiros (2025) 🇧🇷 proposed some middle ground:

$$h(x, z) = x^\top \beta(z), \quad \frac{\partial h(x, z)}{\partial x} = \beta(z)$$

- The partial effect of  $X$  on  $Y$  varies with  $Z$  through  $\beta(z)$ ;
- $\beta(\cdot)$  is a Lipschitz function that can be estimated with ML methods;
- No need to numerically approximate  $\frac{\partial}{\partial x} h(x, z)$ ;

## How can we balance flexibility and interpretability?

---

🇧🇷 Masini and Medeiros (2025) 🇧🇷 proposed some middle ground:

$$h(x, z) = x^\top \beta(z), \quad \frac{\partial h(x, z)}{\partial x} = \beta(z)$$

- The partial effect of  $X$  on  $Y$  varies with  $Z$  through  $\beta(z)$ ;
- $\beta(\cdot)$  is a Lipschitz function that can be estimated with ML methods;
- No need to numerically approximate  $\frac{\partial}{\partial x} h(x, z)$ ;
- Explicit conditions for consistency *and* asymptotic normality of  $\hat{\beta}(z)$ ;

## How can we balance flexibility and interpretability?

Masini and Medeiros (2025) proposed some middle ground:

$$h(x, z) = x^\top \beta(z), \quad \frac{\partial h(x, z)}{\partial x} = \beta(z)$$

- The partial effect of  $X$  on  $Y$  varies with  $Z$  through  $\beta(z)$ ;
- $\beta(\cdot)$  is a Lipschitz function that can be estimated with ML methods;
- No need to numerically approximate  $\frac{\partial}{\partial x} h(x, z)$ ;
- Explicit conditions for consistency and asymptotic normality of  $\hat{\beta}(z)$ ;
- Secret sauce: a variant of the **Random Forest** algorithm!

## How can we balance flexibility and interpretability?

🇧🇷 Masini and Medeiros (2025) 🇧🇷 proposed some middle ground:

$$h(x, z) = x^\top \beta(z), \quad \frac{\partial h(x, z)}{\partial x} = \beta(z)$$

- The partial effect of  $X$  on  $Y$  varies with  $Z$  through  $\beta(z)$ ;
- $\beta(\cdot)$  is a Lipschitz function that can be estimated with ML methods;
- No need to numerically approximate  $\frac{\partial}{\partial x} h(x, z)$ ;
- Explicit conditions for consistency *and* asymptotic normality of  $\hat{\beta}(z)$ ;
- Secret sauce: a variant of the **Random Forest** algorithm!

But what is a Random Forest, anyway? 🤔

**Questions?**

## Quick Intro to Random Forests

---

## Random Trees

---

Recall the general ML framework:

$$Y = f(X) + \varepsilon$$

A **Random Tree** is a particular way of parametrizing  $f$ !

## Random Trees

---

Recall the general ML framework:

$$Y = f(X) + \varepsilon$$

A **Random Tree** is a particular way of parametrizing  $f$ !

- If  $X \in \mathbb{R}^p$ , consider a finite partition  $\{S_1, S_2, \dots, S_m\}$  of  $\mathbb{R}^p$ ;
- Each  $S_i$  is a hyperrectangle defined by recursive binary splits on the covariates;
- On each  $S_i$ ,  $f$  is constant:  $f(x) = \mu_i$  for all  $x \in S_i$ ;
- After a tree has been estimated (“grown”),  $\hat{f}(x_i) = \mu_i$  if  $x_i \in S_i$ ;

## Random Trees

---

Recall the general ML framework:

$$Y = f(X) + \varepsilon$$

A **Random Tree** is a particular way of parametrizing  $f$ !

- If  $X \in \mathbb{R}^p$ , consider a finite partition  $\{S_1, S_2, \dots, S_m\}$  of  $\mathbb{R}^p$ ;
- Each  $S_i$  is a hyperrectangle defined by recursive binary splits on the covariates;
- On each  $S_i$ ,  $f$  is constant:  $f(x) = \mu_i$  for all  $x \in S_i$ ;
- After a tree has been estimated (“grown”),  $\hat{f}(x_i) = \mu_i$  if  $x_i \in S_i$ ;

The really complicated part: there are *so many* partitions... how to pick one?

## Random Trees

Recall the general ML framework:

$$Y = f(X) + \varepsilon$$

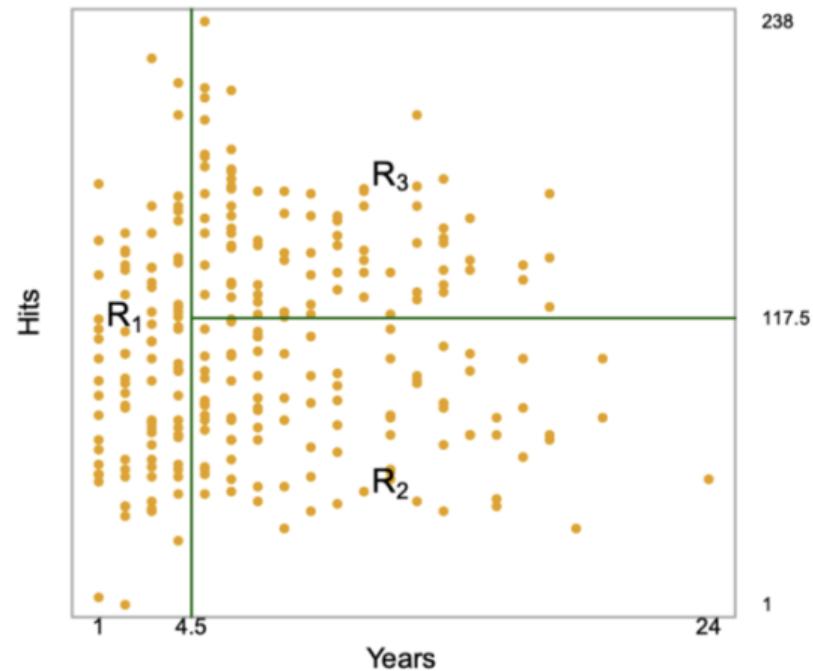
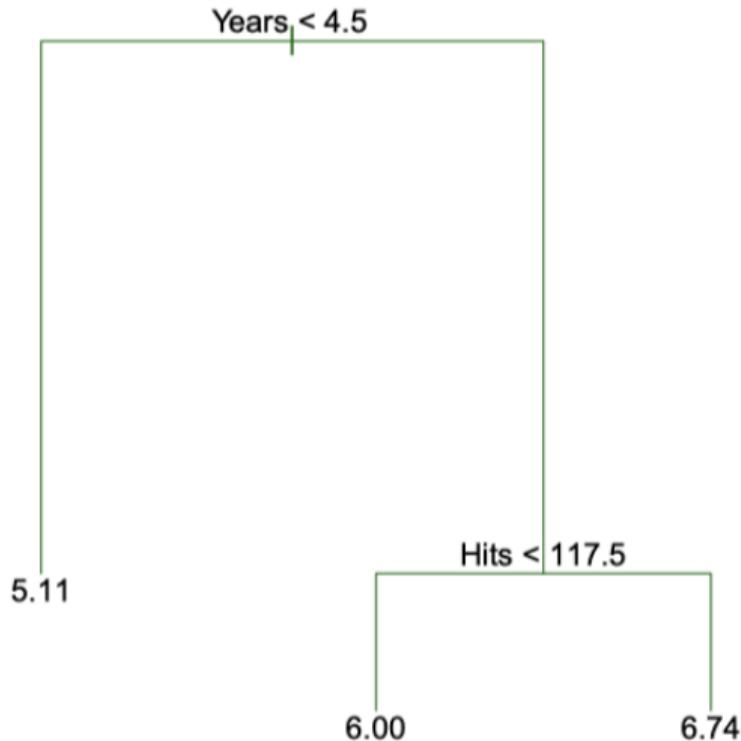
A **Random Tree** is a particular way of parametrizing  $f$ !

- If  $X \in \mathbb{R}^p$ , consider a finite partition  $\{S_1, S_2, \dots, S_m\}$  of  $\mathbb{R}^p$ ;
- Each  $S_i$  is a hyperrectangle defined by recursive binary splits on the covariates;
- On each  $S_i$ ,  $f$  is constant:  $f(x) = \mu_i$  for all  $x \in S_i$ ;
- After a tree has been estimated (“grown”),  $\hat{f}(x_i) = \mu_i$  if  $x_i \in S_i$ ;

The really complicated part: there are *so many* partitions... how to pick one?

**Hyafil and Rivest (1976):** this is harder than you think! The problem is NP-complete!

## Example: Predicting Baseball Player Salaries



## How to pick a split point? Use some greed!

Let's say you want to split on feature  $X_j$  at point  $\delta$ :

$$S_1 \equiv \{x \in \mathbb{R}^p : x_j \leq \delta\}, \quad S_2 \equiv \{x \in \mathbb{R}^p : x_j > \delta\}$$

$$\mu_1(\delta) \equiv \sum_{i:x_i \in S_1} \frac{Y_i}{n_1}, \quad \mu_2(\delta) \equiv \sum_{i:x_i \in S_2} \frac{Y_i}{n_2}$$

## How to pick a split point? Use some greed!

Let's say you want to split on feature  $X_j$  at point  $\delta$ :

$$S_1 \equiv \{x \in \mathbb{R}^p : x_j \leq \delta\}, \quad S_2 \equiv \{x \in \mathbb{R}^p : x_j > \delta\}$$

$$\mu_1(\delta) \equiv \sum_{i:x_i \in S_1} \frac{Y_i}{n_1}, \quad \mu_2(\delta) \equiv \sum_{i:x_i \in S_2} \frac{Y_i}{n_2}$$

- Define  $SSR(\delta) \equiv \sum_{i:x_i \in S_1} (Y_i - \mu_1(\delta))^2 + \sum_{i:x_i \in S_2} (Y_i - \mu_2(\delta))^2$
- Choose  $\delta$  to minimize  $SSR(\delta)$   $\implies$  this is usually very fast to compute!

## How to pick a split point? Use some greed!

Let's say you want to split on feature  $X_j$  at point  $\delta$ :

$$S_1 \equiv \{x \in \mathbb{R}^p : x_j \leq \delta\}, \quad S_2 \equiv \{x \in \mathbb{R}^p : x_j > \delta\}$$

$$\mu_1(\delta) \equiv \sum_{i:x_i \in S_1} \frac{Y_i}{n_1}, \quad \mu_2(\delta) \equiv \sum_{i:x_i \in S_2} \frac{Y_i}{n_2}$$

- Define  $SSR(\delta) \equiv \sum_{i:x_i \in S_1} (Y_i - \mu_1(\delta))^2 + \sum_{i:x_i \in S_2} (Y_i - \mu_2(\delta))^2$
- Choose  $\delta$  to minimize  $SSR(\delta)$   $\implies$  this is usually very fast to compute!
- Repeat this for all features  $X_j$  and pick the best one;
- Important: you need some stopping rule! There is a huge literature on this...
- Example: minimum number of observations per leaf;

## How to pick a split point? Use some greed!

Let's say you want to split on feature  $X_j$  at point  $\delta$ :

$$S_1 \equiv \{x \in \mathbb{R}^p : x_j \leq \delta\}, \quad S_2 \equiv \{x \in \mathbb{R}^p : x_j > \delta\}$$

$$\mu_1(\delta) \equiv \sum_{i:x_i \in S_1} \frac{Y_i}{n_1}, \quad \mu_2(\delta) \equiv \sum_{i:x_i \in S_2} \frac{Y_i}{n_2}$$

- Define  $SSR(\delta) \equiv \sum_{i:x_i \in S_1} (Y_i - \mu_1(\delta))^2 + \sum_{i:x_i \in S_2} (Y_i - \mu_2(\delta))^2$
- Choose  $\delta$  to minimize  $SSR(\delta)$   $\implies$  this is usually very fast to compute!
- Repeat this for all features  $X_j$  and pick the best one;
- Important: you need some stopping rule! There is a huge literature on this...
- Example: minimum number of observations per leaf;

This is the so-called the **CART** algorithm due to Breiman et al. (1984).

## How to get a Random Forest?

A Random Forest is an **ensemble** of Random Trees:

$$\hat{f}^{(1)}(x), \hat{f}^{(2)}(x), \dots, \hat{f}^{(B)}(x)$$

## How to get a Random Forest?

A Random Forest is an **ensemble** of Random Trees:

$$\hat{f}^{(1)}(x), \hat{f}^{(2)}(x), \dots, \hat{f}^{(B)}(x)$$

Each tree is grown on a **perturbed version** of the data:

- Bootstrap sample of the observations;
- Random subset of features considered at each split;

## How to get a Random Forest?

A Random Forest is an **ensemble** of Random Trees:

$$\hat{f}^{(1)}(x), \hat{f}^{(2)}(x), \dots, \hat{f}^{(B)}(x)$$

Each tree is grown on a **perturbed version** of the data:

- Bootstrap sample of the observations;
- Random subset of features considered at each split;

The forest prediction is the **average**:

$$\hat{f}_{RF}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{(b)}(x)$$

# How to get a Random Forest?

A Random Forest is an **ensemble** of Random Trees:

$$\hat{f}^{(1)}(x), \hat{f}^{(2)}(x), \dots, \hat{f}^{(B)}(x)$$

Each tree is grown on a **perturbed version** of the data:

- Bootstrap sample of the observations;
- Random subset of features considered at each split;

The forest prediction is the **average**:

$$\hat{f}_{RF}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{(b)}(x)$$

Key insight:

- Each tree is noisy and biased, but averaging them reduces variance dramatically;
- Randomness *decorrelates* the trees, making averaging powerful;

**Questions?**

**Back to Partial Effects**

---

## The Main Insight

---

We have a random sample  $\{(Y_i, X_i, Z_i)\}_{i=1}^n$  from

$$Y_i = X_i^\top \beta(Z_i) + \varepsilon_i, \quad \mathbb{E}[\varepsilon_i | X_i, Z_i] = 0$$

## The Main Insight

We have a random sample  $\{(Y_i, X_i, Z_i)\}_{i=1}^n$  from

$$Y_i = X_i^\top \beta(Z_i) + \varepsilon_i, \quad \mathbb{E}[\varepsilon_i | X_i, Z_i] = 0$$

- [Masini and Medeiros \(2025\)](#) proposed to estimate  $\beta(z)$  using a modified Random Forest;
- Key modification: at each split, try to minimize the **local least squares** criterion;

## The Main Insight

We have a random sample  $\{(Y_i, X_i, Z_i)\}_{i=1}^n$  from

$$Y_i = X_i^\top \beta(Z_i) + \varepsilon_i, \quad \mathbb{E}[\varepsilon_i | X_i, Z_i] = 0$$

- [Masini and Medeiros \(2025\)](#) proposed to estimate  $\beta(z)$  using a modified Random Forest;
- Key modification: at each split, try to minimize the **local least squares** criterion;

Suppose you want to split at  $Z_j \leq \delta$  as before. Then:

$$S_1 \equiv \{z \in \mathbb{R}^p : z_j \leq \delta\}, \quad S_2 \equiv \{z \in \mathbb{R}^p : z_j > \delta\}$$

$$(\hat{\beta}_1) \equiv \arg \min_{\beta} \sum_{i: Z_i \in S_1} (Y_i - X_i^\top \beta)^2, \quad (\hat{\beta}_2) \equiv \arg \min_{\beta} \sum_{i: Z_i \in S_2} (Y_i - X_i^\top \beta)^2$$

$$SSR(\delta) \equiv \sum_{i: Z_i \in S_1} (Y_i - X_i^\top \hat{\beta}_1)^2 + \sum_{i: Z_i \in S_2} (Y_i - X_i^\top \hat{\beta}_2)^2$$

## The Main Insight

We have a random sample  $\{(Y_i, X_i, Z_i)\}_{i=1}^n$  from

$$Y_i = X_i^\top \beta(Z_i) + \varepsilon_i, \quad \mathbb{E}[\varepsilon_i | X_i, Z_i] = 0$$

- [Masini and Medeiros \(2025\)](#) proposed to estimate  $\beta(z)$  using a modified Random Forest;
- Key modification: at each split, try to minimize the **local least squares** criterion;

Suppose you want to split at  $Z_j \leq \delta$  as before. Then:

$$S_1 \equiv \{z \in \mathbb{R}^p : z_j \leq \delta\}, \quad S_2 \equiv \{z \in \mathbb{R}^p : z_j > \delta\}$$

$$(\hat{\beta}_1) \equiv \arg \min_{\beta} \sum_{i: Z_i \in S_1} (Y_i - X_i^\top \beta)^2, \quad (\hat{\beta}_2) \equiv \arg \min_{\beta} \sum_{i: Z_i \in S_2} (Y_i - X_i^\top \beta)^2$$

$$SSR(\delta) \equiv \sum_{i: Z_i \in S_1} (Y_i - X_i^\top \hat{\beta}_1)^2 + \sum_{i: Z_i \in S_2} (Y_i - X_i^\top \hat{\beta}_2)^2$$

Pick  $\delta$  to minimize  $SSR(\delta)$ !

## The Algorithm

Pick a number of trees  $B$  and a minimum leaf size  $k$ . For  $b = 1, \dots, B$ :

1. Draw a bootstrap sample of size  $s \leq n$  from the data;
2. Divide the data into two halves  $\mathcal{A}$  and  $\mathcal{B}$ ;
3. Using  $\mathcal{B}$ , keep splitting at random dimensions  $j$  using the previous criterion;
4. Stop when all leaves have less than  $2k - 1$  and more than  $k$  observations;
5. Using  $\mathcal{A}$ , estimate  $\beta(z)$  using only observations in the leaf where  $z$  falls;

The final estimate is

$$\hat{\beta}(z) = \frac{1}{B} \sum_{b=1}^B \hat{\beta}^{(b)}(z)$$

## The Algorithm

Pick a number of trees  $B$  and a minimum leaf size  $k$ . For  $b = 1, \dots, B$ :

1. Draw a bootstrap sample of size  $s \leq n$  from the data;
2. Divide the data into two halves  $\mathcal{A}$  and  $\mathcal{B}$ ;
3. Using  $\mathcal{B}$ , keep splitting at random dimensions  $j$  using the previous criterion;
4. Stop when all leaves have less than  $2k - 1$  and more than  $k$  observations;
5. Using  $\mathcal{A}$ , estimate  $\beta(z)$  using only observations in the leaf where  $z$  falls;

The final estimate is

$$\hat{\beta}(z) = \frac{1}{B} \sum_{b=1}^B \hat{\beta}^{(b)}(z)$$

This algorithm uses **honest trees!** Similar intuition to cross-fitting.

## Cool Properties and Limitations

---

### Cool properties:

- Highly interpretable and relatively mild assumptions on  $\beta(z)$ ;
- Easy confidence intervals for  $\beta(z)$  at any point  $z$ :

$$\Omega^{-1/2}(z) \left( \hat{\beta}(z) - \beta(z) \right) \xrightarrow{d} \mathcal{N}(0, I_q)$$

for some complicated  $\Omega(x)$  that can be estimated consistently;

- There is also a Lagrange multiplier test for homogeneity of  $\beta(z)$ ;

# Cool Properties and Limitations

## Cool properties:

- Highly interpretable and relatively mild assumptions on  $\beta(z)$ ;
- Easy confidence intervals for  $\beta(z)$  at any point  $z$ :

$$\Omega^{-1/2}(z) \left( \hat{\beta}(z) - \beta(z) \right) \xrightarrow{d} \mathcal{N}(0, I_q)$$

for some complicated  $\Omega(x)$  that can be estimated consistently;

- There is also a Lagrange multiplier test for homogeneity of  $\beta(z)$ ;

## Limitations:

- The dimension of  $X_i$  should be small relative to  $n$ ;
- The dimension of  $Z_i$  cannot be *that* large relative to  $n$ ;
- Pointwise inference only;
- It can be computationally demanding in large datasets;

**Questions?**

**Take a deep breath... we are changing topics!**



## Solving Large Models Through Deep Learning

---

## There is tension in the air...

---

- Dynamic programming/stochastic control everywhere: Macro, Finance, IO, Labor, etc;
- Pen and paper won't cut it: we need sophisticated numerical methods;

## There is tension in the air...

---

- Dynamic programming/stochastic control everywhere: Macro, Finance, IO, Labor, etc;
- Pen and paper won't cut it: we need sophisticated numerical methods;
- Tension: **the models we can solve** vs **the ones we would like to solve**;
- Main challenge: computational cost increases exponentially with state space dimension;
- Traditional practice: simpler economics, simpler models;

## There is tension in the air...

---

- Dynamic programming/stochastic control everywhere: Macro, Finance, IO, Labor, etc;
- Pen and paper won't cut it: we need sophisticated numerical methods;
- Tension: **the models we can solve** vs **the ones we would like to solve**;
- Main challenge: computational cost increases exponentially with state space dimension;
- Traditional practice: simpler economics, simpler models;
- In practice: forget solving models with more than  $\approx 10\text{-}15$  state variables;

Deep Learning to the rescue!

## The Three Musketeers

Let  $\mathbf{s} \in \mathbb{R}^n$  be a state vector,  $\mathbf{c} \in \mathbb{R}^m$  be a control vector, and  $\mathbf{u} \in \mathbb{R}^k$  be shocks:

$$V(\mathbf{s}) = \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + e^{-\rho} \mathbb{E} [V(\mathbf{s}') \mid \mathbf{s}, \mathbf{c}] \right\}$$
$$\mathbf{s}' = f(\mathbf{s}, \mathbf{c}, \mathbf{u}), \quad \mathbf{u} \sim F(\cdot)$$

## The Three Musketeers

Let  $\mathbf{s} \in \mathbb{R}^n$  be a state vector,  $\mathbf{c} \in \mathbb{R}^m$  be a control vector, and  $\mathbf{u} \in \mathbb{R}^k$  be shocks:

$$V(\mathbf{s}) = \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + e^{-\rho} \mathbb{E} [V(\mathbf{s}') \mid \mathbf{s}, \mathbf{c}] \right\}$$
$$\mathbf{s}' = f(\mathbf{s}, \mathbf{c}, \mathbf{u}), \quad \mathbf{u} \sim F(\cdot)$$

Following [Powell \(2011\)](#)

- **The curse of representation:** a Cartesian grid for  $\mathbf{s}$  grows exponentially with  $n$ ;
  - Ex: 100 grid points per dimension  $\implies 10^{2n}$  total points;
  - Ex:  $n = 26$ , 100 points per dimension =  $O(10^{40})$  TB of RAM (AWS  $\approx 10^9$  TB);

## The Three Musketeers

Let  $\mathbf{s} \in \mathbb{R}^n$  be a state vector,  $\mathbf{c} \in \mathbb{R}^m$  be a control vector, and  $\mathbf{u} \in \mathbb{R}^k$  be shocks:

$$V(\mathbf{s}) = \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + e^{-\rho} \mathbb{E} [V(\mathbf{s}') \mid \mathbf{s}, \mathbf{c}] \right\}$$
$$\mathbf{s}' = f(\mathbf{s}, \mathbf{c}, \mathbf{u}), \quad \mathbf{u} \sim F(\cdot)$$

Following [Powell \(2011\)](#)

- **The curse of representation:** a Cartesian grid for  $\mathbf{s}$  grows exponentially with  $n$ ;
  - Ex: 100 grid points per dimension  $\implies 10^{2n}$  total points;
  - Ex:  $n = 26$ , 100 points per dimension =  $O(10^{40})$  TB of RAM (AWS  $\approx 10^9$  TB);
- **The curse of expectation:**

$$\mathbb{E} [V(\mathbf{s}') \mid \mathbf{s}, \mathbf{c}] = \int_{u_1} \int_{u_2} \cdots \int_{u_k} V(f(\mathbf{s}, \mathbf{c}(\mathbf{s}), \mathbf{u})) \phi(\mathbf{u}) du_1 du_2 \cdots du_k$$

## The Three Musketeers

Let  $\mathbf{s} \in \mathbb{R}^n$  be a state vector,  $\mathbf{c} \in \mathbb{R}^m$  be a control vector, and  $\mathbf{u} \in \mathbb{R}^k$  be shocks:

$$V(\mathbf{s}) = \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + e^{-\rho} \mathbb{E} [V(\mathbf{s}') \mid \mathbf{s}, \mathbf{c}] \right\}$$
$$\mathbf{s}' = f(\mathbf{s}, \mathbf{c}, \mathbf{u}), \quad \mathbf{u} \sim F(\cdot)$$

Following [Powell \(2011\)](#)

- **The curse of representation:** a Cartesian grid for  $\mathbf{s}$  grows exponentially with  $n$ ;
  - Ex: 100 grid points per dimension  $\implies 10^{2n}$  total points;
  - Ex:  $n = 26$ , 100 points per dimension =  $O(10^{40})$  TB of RAM (AWS  $\approx 10^9$  TB);
- **The curse of expectation:**

$$\mathbb{E} [V(\mathbf{s}') \mid \mathbf{s}, \mathbf{c}] = \int_{u_1} \int_{u_2} \cdots \int_{u_k} V(f(\mathbf{s}, \mathbf{c}(\mathbf{s}), \mathbf{u})) \phi(\mathbf{u}) d\mathbf{u}_1 d\mathbf{u}_2 \cdots d\mathbf{u}_k$$

- **The curse of optimization:** Given  $V(\cdot)$ ,  $10^{2n}$  optimizations to find  $\mathbf{c}(\mathbf{s})$ ;

## Can Ito Calculus Save Us?

$$V(\mathbf{s}_t) = \sup_{\mathbf{c}} \mathbb{E}_t \left\{ \int_t^{\infty} e^{-\rho(\nu-t)} u(\mathbf{c}_{\nu}) d\nu \right\}$$

subject to  $d\mathbf{s}_t = f(\mathbf{s}_t, \mathbf{c}_t)dt + g(\mathbf{s}_t, \mathbf{c}_t)d\mathbf{B}_t$

## Can Ito Calculus Save Us?

$$V(\mathbf{s}_t) = \sup_{\mathbf{c}} \mathbb{E}_t \left\{ \int_t^{\infty} e^{-\rho(\nu-t)} u(\mathbf{c}_{\nu}) d\nu \right\}$$

$$\text{subject to } d\mathbf{s}_t = f(\mathbf{s}_t, \mathbf{c}_t) dt + g(\mathbf{s}_t, \mathbf{c}_t) d\mathbf{B}_t$$

The Stochastic Control gang is smart... give up  $\mathbb{E}$ , use Ito's lemma, and solve a PDE!

## Can Ito Calculus Save Us?

$$V(\mathbf{s}_t) = \sup_{\mathbf{c}} \mathbb{E}_t \left\{ \int_t^{\infty} e^{-\rho(\nu-t)} u(\mathbf{c}_{\nu}) d\nu \right\}$$

$$\text{subject to } d\mathbf{s}_t = f(\mathbf{s}_t, \mathbf{c}_t) dt + g(\mathbf{s}_t, \mathbf{c}_t) d\mathbf{B}_t$$

The Stochastic Control gang is smart... give up  $\mathbb{E}$ , use Ito's lemma, and solve a PDE!

Under some conditions,  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  are such that:

$$HJB(\mathbf{s}, \mathbf{c}, V) \equiv -\rho V(\mathbf{s}) + u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{ss}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

$$HJB(\mathbf{s}, \mathbf{c}(\mathbf{s}), V(\mathbf{s})) = 0, \quad \forall \mathbf{s} \in \mathbb{R}^n \text{ at the optimum policy}$$

$$\mathbf{c}(\mathbf{s}) = \arg \max_{\mathbf{c}} u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{ss}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

## Can Ito Calculus Save Us?

$$V(\mathbf{s}_t) = \sup_{\mathbf{c}} \mathbb{E}_t \left\{ \int_t^{\infty} e^{-\rho(\nu-t)} u(\mathbf{c}_{\nu}) d\nu \right\}$$

$$\text{subject to } d\mathbf{s}_t = f(\mathbf{s}_t, \mathbf{c}_t) dt + g(\mathbf{s}_t, \mathbf{c}_t) d\mathbf{B}_t$$

The Stochastic Control gang is smart... give up  $\mathbb{E}$ , use Ito's lemma, and solve a PDE!

Under some conditions,  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  are such that:

$$HJB(\mathbf{s}, \mathbf{c}, V) \equiv -\rho V(\mathbf{s}) + u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{ss}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

$$HJB(\mathbf{s}, \mathbf{c}(\mathbf{s}), V(\mathbf{s})) = 0, \quad \forall \mathbf{s} \in \mathbb{R}^n \text{ at the optimum policy}$$

$$\mathbf{c}(\mathbf{s}) = \arg \max_{\mathbf{c}} u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{ss}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

## Good News and Bad News

---

- **Good news:** no expectation operator! Solve the PDE and we are done;

## Good News and Bad News

---

- **Good news:** no expectation operator! Solve the PDE and we are done;
- **Bad news:** we still need to represent  $V(s)$  and  $c(s)$  somehow;

## Good News and Bad News

---

- **Good news:** no expectation operator! Solve the PDE and we are done;
- **Bad news:** we still need to represent  $V(s)$  and  $c(s)$  somehow;
- **Worse news:** we still need to optimize  $c(s)$  at each  $s$ ;

## Good News and Bad News

---

- **Good news:** no expectation operator! Solve the PDE and we are done;
- **Bad news:** we still need to represent  $V(s)$  and  $c(s)$  somehow;
- **Worse news:** we still need to optimize  $c(s)$  at each  $s$ ;
- **Even worse news:** solving PDEs in high dimensions is notoriously difficult;

## Good News and Bad News

---

- **Good news:** no expectation operator! Solve the PDE and we are done;
- **Bad news:** we still need to represent  $V(s)$  and  $c(s)$  somehow;
- **Worse news:** we still need to optimize  $c(s)$  at each  $s$ ;
- **Even worse news:** solving PDEs in high dimensions is notoriously difficult;
- Typical approach from engineering: **Finite Differences** or **Finite Elements**;
- Let  $n = 1$  (scalar state) and an equally spaced grid  $\{s_1, s_2, \dots, s_M\}$ . Then:

$$\frac{dV(s)}{ds} \Big|_{s=s_i} \approx \frac{V(s_{i+1}) - V(s_{i-1})}{2\Delta s}, \quad \frac{d^2V(s)}{ds^2} \Big|_{s=s_i} \approx \frac{V(s_{i+1}) - 2V(s_i) + V(s_{i-1})}{\Delta s^2}$$

## Good News and Bad News

---

- **Good news:** no expectation operator! Solve the PDE and we are done;
- **Bad news:** we still need to represent  $V(s)$  and  $c(s)$  somehow;
- **Worse news:** we still need to optimize  $c(s)$  at each  $s$ ;
- **Even worse news:** solving PDEs in high dimensions is notoriously difficult;
- Typical approach from engineering: **Finite Differences** or **Finite Elements**;
- Let  $n = 1$  (scalar state) and an equally spaced grid  $\{s_1, s_2, \dots, s_M\}$ . Then:

$$\frac{dV(s)}{ds} \Big|_{s=s_i} \approx \frac{V(s_{i+1}) - V(s_{i-1})}{2\Delta s}, \quad \frac{d^2V(s)}{ds^2} \Big|_{s=s_i} \approx \frac{V(s_{i+1}) - 2V(s_i) + V(s_{i-1})}{\Delta s^2}$$

- Drawback: we need very fine grids to approximate derivatives well;

# The Beauty of Spectral Methods

---

**Key idea:** Parametrize  $V(\mathbf{s})$  using a convenient *basis of functions!*

# The Beauty of Spectral Methods

---

**Key idea:** Parametrize  $V(s)$  using a convenient *basis of functions*!

- Example for  $n = 1$ :

$$V(s) = \sum_{k=0}^K a_k T_k(s), \quad \{T_k(\cdot)\}_{k=0}^K \text{ are Chebyshev polynomials}$$

# The Beauty of Spectral Methods

---

**Key idea:** Parametrize  $V(s)$  using a convenient *basis of functions!*

- Example for  $n = 1$ :

$$V(s) = \sum_{k=0}^K a_k T_k(s), \quad \{T_k(\cdot)\}_{k=0}^K \text{ are Chebyshev polynomials}$$

- Derivatives of  $V(s)$  are analytically available (*fast... like really fast!*);

# The Beauty of Spectral Methods

**Key idea:** Parametrize  $V(s)$  using a convenient *basis of functions!*

- Example for  $n = 1$ :

$$V(s) = \sum_{k=0}^K a_k T_k(s), \quad \{T_k(\cdot)\}_{k=0}^K \text{ are Chebyshev polynomials}$$

- Derivatives of  $V(s)$  are analytically available (*fast... like really fast!*);
- Excellent global approximation properties (minimax interpolation error);

# The Beauty of Spectral Methods

**Key idea:** Parametrize  $V(s)$  using a convenient *basis of functions!*

- Example for  $n = 1$ :

$$V(s) = \sum_{k=0}^K a_k T_k(s), \quad \{T_k(\cdot)\}_{k=0}^K \text{ are Chebyshev polynomials}$$

- Derivatives of  $V(s)$  are analytically available (*fast... like really fast!*);
- Excellent global approximation properties (minimax interpolation error);
- **However:** if  $n > 1$ , the basis relies on **tensor products**:

$$V(s_1, \dots, s_n) = \sum_{k_1=0}^K \cdots \sum_{k_n=0}^K a_{k_1, \dots, k_n} \prod_{j=1}^n T_{k_j}(s_j)$$

# The Beauty of Spectral Methods

**Key idea:** Parametrize  $V(s)$  using a convenient *basis of functions!*

- Example for  $n = 1$ :

$$V(s) = \sum_{k=0}^K a_k T_k(s), \quad \{T_k(\cdot)\}_{k=0}^K \text{ are Chebyshev polynomials}$$

- Derivatives of  $V(s)$  are analytically available (*fast... like really fast!*);
- Excellent global approximation properties (minimax interpolation error);
- **However:** if  $n > 1$ , the basis relies on **tensor products**:

$$V(s_1, \dots, s_n) = \sum_{k_1=0}^K \cdots \sum_{k_n=0}^K a_{k_1, \dots, k_n} \prod_{j=1}^n T_{k_j}(s_j)$$

- Number of coefficients grows exponentially with  $n$ :

$$(K + 1)^n$$

## A Deep Contribution

---

- Chebyshev polynomials are amazing but only in lower dimensions (say  $n < 8$ );

## A Deep Contribution

---

- Chebyshev polynomials are amazing but only in lower dimensions (say  $n < 8$ );
- A good basis of functions for us should feature:
  - **Richness:** be able to approximate a wide variety of functions;
  - **Performance:** Fast computation derivatives/gradients;
  - **Expressiveness:** Few parameters should suffice to approximate complex functions;

( ×3) Duarte et al. (2024): let's embrace Neural Networks!

# A Deep Contribution

---

- Chebyshev polynomials are amazing but only in lower dimensions (say  $n < 8$ );
- A good basis of functions for us should feature:
  - **Richness:** be able to approximate a wide variety of functions;
  - **Performance:** Fast computation derivatives/gradients;
  - **Expressiveness:** Few parameters should suffice to approximate complex functions;

( ×3) Duarte et al. (2024): let's embrace Neural Networks!

## But why?

- Neural networks can approximate almost any function;
- Computing gradients is super fast due to breakthroughs in *Automatic Differentiation*;
- If we use *deep* Neural Nets, we need few parameters!

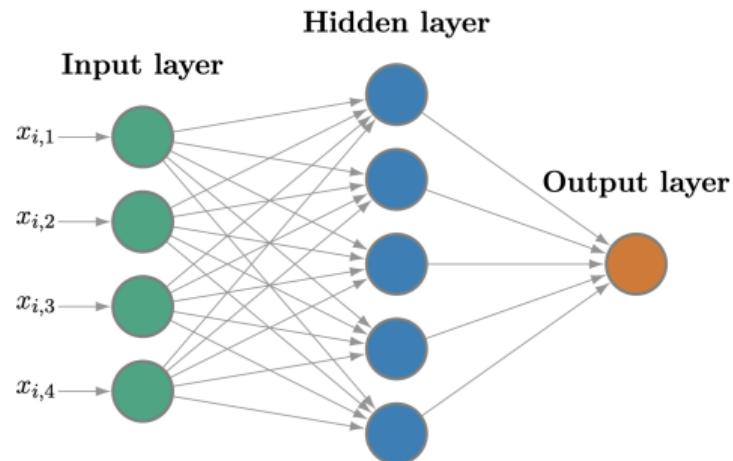
**Questions?**

**But what is a Neural Network?**

---

# A Smart Way of Composing Functions

- Each node is equipped with weights  $w$ , a bias  $b$ , and an activation function  $\sigma(\cdot)$ ;
- Hidden nodes compute  $\sigma(w^\top x + b)$ ;
- Examples of  $\sigma(\cdot)$ :  $\underbrace{\max\{0, x\}}_{\text{ReLU}}$ ,  $\tanh(x)$ ;
- Output of hidden layer:  
$$\mathbf{h}(\mathbf{x}, \theta) = \sigma(W\mathbf{x} + \mathbf{b});$$
- Final output:  $\hat{y} = w_{out}^\top \mathbf{h}(\mathbf{x}, \theta) + b_{out}$ ;
- Abuse notation:  $\theta \equiv (W, b, w_{out}, b_{out})$
- *Shallow neural network* = 1 hidden layer;



## Does it have good properties?

Cybenko (1989) proved that this basis of functions is rich:

### Theorem (Universal Approximation)

*Any continuous function on a compact subset of  $\mathbb{R}^n$  can be approximated to arbitrary precision by a shallow neural network with any non-polynomial activation function.*

## Does it have good properties?

Cybenko (1989) proved that this basis of functions is rich:

### Theorem (Universal Approximation)

*Any continuous function on a compact subset of  $\mathbb{R}^n$  can be approximated to arbitrary precision by a shallow neural network with any non-polynomial activation function.*

Some bad news: how many hidden nodes do we need?

## Does it have good properties?

Cybenko (1989) proved that this basis of functions is rich:

### Theorem (Universal Approximation)

*Any continuous function on a compact subset of  $\mathbb{R}^n$  can be approximated to arbitrary precision by a shallow neural network with any non-polynomial activation function.*

Some bad news: how many hidden nodes do we need?

## Does it have good properties?

Cybenko (1989) proved that this basis of functions is rich:

### Theorem (Universal Approximation)

*Any continuous function on a compact subset of  $\mathbb{R}^n$  can be approximated to arbitrary precision by a shallow neural network with any non-polynomial activation function.*

Some bad news: how many hidden nodes do we need?

---

Some **great** news at last:

- If we allow for *more layers*, we can use fewer nodes in each layer;
- More layers and fewer nodes = fewer parameters than a single super wide layer;

## Does it have good properties?

Cybenko (1989) proved that this basis of functions is rich:

### Theorem (Universal Approximation)

*Any continuous function on a compact subset of  $\mathbb{R}^n$  can be approximated to arbitrary precision by a shallow neural network with any non-polynomial activation function.*

Some bad news: how many hidden nodes do we need?

---

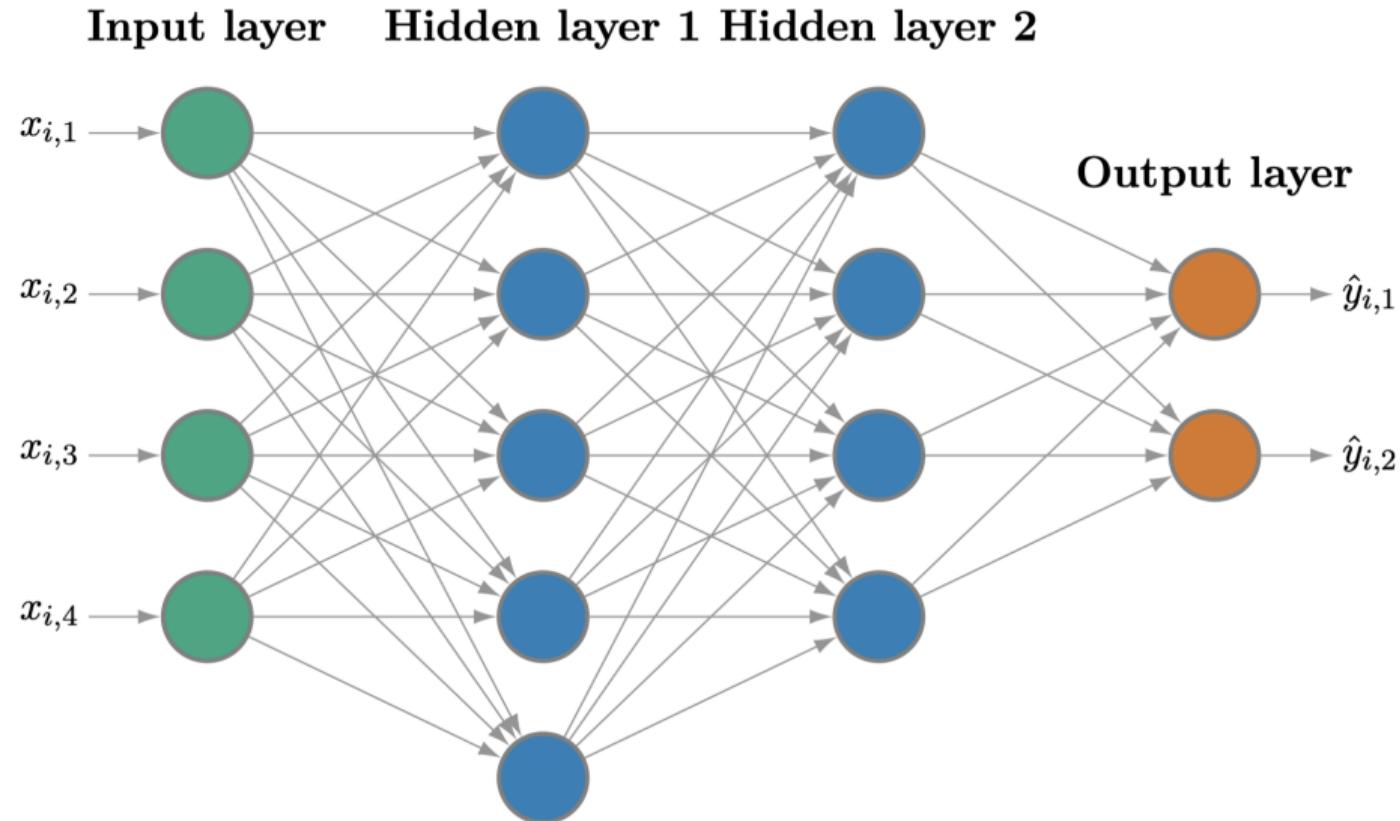
Some **great** news at last:

- If we allow for *more layers*, we can use fewer nodes in each layer;
- More layers and fewer nodes = fewer parameters than a single super wide layer;

### Theorem (Lu et al. (2017))

*For any Lebesgue integrable function  $f \in L^1(\mathbb{R}^n)$ , there exists a ReLU deep neural network with width at most  $n + 4$  in every hidden layer that approximates  $f$  to arbitrary precision.*

# What is a Deep Neural Network?



## How do we find these parameters?

---

*Training the network* = picking  $\theta$  to minimize some loss function:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \theta))^2$$

## How do we find these parameters?

---

*Training the network* = picking  $\theta$  to minimize some loss function:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \theta))^2$$

One scheme to minimize  $\mathcal{L}(\theta)$  is Gradient Descent:

$$\nabla_{\theta} \mathcal{L}(\theta) = -\frac{2}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \theta)) \nabla_{\theta} f(\mathbf{x}_i, \theta)$$

## How do we find these parameters?

*Training the network* = picking  $\theta$  to minimize some loss function:

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \theta))^2$$

One scheme to minimize  $\mathcal{L}(\theta)$  is Gradient Descent:

$$\nabla_{\theta} \mathcal{L}(\theta) = -\frac{2}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \theta)) \nabla_{\theta} f(\mathbf{x}_i, \theta)$$

$$\begin{aligned}\theta_{k+1} &= \theta_k - \gamma \nabla_{\theta} \mathcal{L}(\theta_k) \\&= \theta_k + \frac{2\gamma}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \theta_k)) \nabla_{\theta} f(\mathbf{x}_i, \theta_k) \\&= \theta_k + \tilde{\gamma} \sum_{i=1}^n \hat{\varepsilon}_i(\theta_k) \nabla_{\theta} f(\mathbf{x}_i, \theta_k)\end{aligned}$$

**Questions?**

**Back to Solving Models!**

---

## Big Picture

---

Recall that we need to find  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  such that:

$$HJB(\mathbf{s}, \mathbf{c}, V) \equiv -\rho V(\mathbf{s}) + u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{\mathbf{ss}}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

$$HJB(\mathbf{s}, \mathbf{c}(\mathbf{s}), V(\mathbf{s})) = 0, \quad \forall \mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n \text{ at the optimum policy}$$

$$\mathbf{c}(\mathbf{s}) = \arg \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{\mathbf{ss}}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right) \right\}$$

## Big Picture

---

Recall that we need to find  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  such that:

$$HJB(\mathbf{s}, \mathbf{c}, V) \equiv -\rho V(\mathbf{s}) + u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{\mathbf{ss}}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

$$HJB(\mathbf{s}, \mathbf{c}(\mathbf{s}), V(\mathbf{s})) = 0, \quad \forall \mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n \text{ at the optimum policy}$$

$$\mathbf{c}(\mathbf{s}) = \arg \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{\mathbf{ss}}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right) \right\}$$

- Key insight in [Duarte et al. \(2024\)](#): approximate  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  using DNNs;
- $V(\mathbf{s}) = V(\mathbf{s}, \theta_V)$ ,  $\mathbf{c}(\mathbf{s}) = \mathbf{c}(\mathbf{s}, \theta_c)$ ;
- Derivatives of  $V(\cdot)$  are super fast to compute using Automatic Differentiation;

## Big Picture

---

Recall that we need to find  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  such that:

$$HJB(\mathbf{s}, \mathbf{c}, V) \equiv -\rho V(\mathbf{s}) + u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{\mathbf{ss}}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right)$$

$$HJB(\mathbf{s}, \mathbf{c}(\mathbf{s}), V(\mathbf{s})) = 0, \quad \forall \mathbf{s} \in \mathcal{S} \subset \mathbb{R}^n \text{ at the optimum policy}$$

$$\mathbf{c}(\mathbf{s}) = \arg \max_{\mathbf{c}} \left\{ u(\mathbf{c}) + \nabla_{\mathbf{s}} V(\mathbf{s})^\top f(\mathbf{s}, \mathbf{c}) + \frac{1}{2} \text{tr} \left( g(\mathbf{s}, \mathbf{c})^\top \nabla_{\mathbf{ss}}^2 V(\mathbf{s}) g(\mathbf{s}, \mathbf{c}) \right) \right\}$$

- Key insight in [Duarte et al. \(2024\)](#): approximate  $V(\mathbf{s})$  and  $\mathbf{c}(\mathbf{s})$  using DNNs;
- $V(\mathbf{s}) = V(\mathbf{s}, \theta_V)$ ,  $\mathbf{c}(\mathbf{s}) = \mathbf{c}(\mathbf{s}, \theta_c)$ ;
- Derivatives of  $V(\cdot)$  are super fast to compute using Automatic Differentiation;
- We look for a pair  $(\theta_V, \theta_c)$  that solves minimizes the HJB residuals!
- Notation:  $HJB(\mathbf{s}, \mathbf{c}(\mathbf{s}, \theta_c), V(\mathbf{s}, \theta_V)) \equiv HJB(\mathbf{s}, \theta_c, \theta_V)$ ;

## The Algorithm

---

- Step 1:** sample  $l$  points  $\{\mathbf{s}_i\}_{i=1}^l$  from the state space  $\mathcal{S}$  (for example, uniformly);  
**Step 2:** Initialize some  $(\boldsymbol{\theta}_V^{(0)}, \boldsymbol{\theta}_c^{(0)})$ ;

## The Algorithm

**Step 1:** sample  $I$  points  $\{\mathbf{s}_i\}_{i=1}^I$  from the state space  $\mathcal{S}$  (for example, uniformly);

**Step 2:** Initialize some  $(\boldsymbol{\theta}_V^{(0)}, \boldsymbol{\theta}_c^{(0)})$ ;

**Step 3:** If we change  $\boldsymbol{\theta}_C$  by a little, the HJB error at  $\mathbf{s}_i$  moves by  $\nabla_{\boldsymbol{\theta}_c} HJB(\mathbf{s}_i, \boldsymbol{\theta}_c, \boldsymbol{\theta}_V)$ . At iteration  $j$ , compute:

$$\boldsymbol{\theta}_c^{(j)} = \boldsymbol{\theta}_c^{(j-1)} + \frac{\gamma}{I} \left( \sum_{i=1}^I \nabla_{\boldsymbol{\theta}_c} HJB \left( \mathbf{s}_i, \boldsymbol{\theta}_c^{(j-1)}, \boldsymbol{\theta}_V^{(j-1)} \right) \right)$$

## The Algorithm

**Step 1:** sample  $I$  points  $\{\mathbf{s}_i\}_{i=1}^I$  from the state space  $\mathcal{S}$  (for example, uniformly);

**Step 2:** Initialize some  $(\boldsymbol{\theta}_V^{(0)}, \boldsymbol{\theta}_c^{(0)})$ ;

**Step 3:** If we change  $\boldsymbol{\theta}_C$  by a little, the HJB error at  $\mathbf{s}_i$  moves by  $\nabla_{\boldsymbol{\theta}_c} HJB(\mathbf{s}_i, \boldsymbol{\theta}_c, \boldsymbol{\theta}_V)$ . At iteration  $j$ , compute:

$$\boldsymbol{\theta}_c^{(j)} = \boldsymbol{\theta}_c^{(j-1)} + \frac{\gamma}{I} \left( \sum_{i=1}^I \nabla_{\boldsymbol{\theta}_c} HJB \left( \mathbf{s}_i, \boldsymbol{\theta}_c^{(j-1)}, \boldsymbol{\theta}_V^{(j-1)} \right) \right)$$

**Step 4:** Define the loss function  $\mathcal{L}(\boldsymbol{\theta}_V)$  and compute its gradient:

$$\mathcal{L}(\boldsymbol{\theta}_V) \equiv \frac{1}{2I} \sum_{i=1}^I \left( HJB \left( \mathbf{s}_i, \boldsymbol{\theta}_c^{(j)}, \boldsymbol{\theta}_V \right) \right)^2$$

$$\nabla_{\boldsymbol{\theta}_V} \mathcal{L}(\boldsymbol{\theta}_V) = \frac{1}{I} \sum_{i=1}^I HJB \left( \mathbf{s}_i, \boldsymbol{\theta}_c^{(j)}, \boldsymbol{\theta}_V \right) \nabla_{\boldsymbol{\theta}_V} HJB \left( \mathbf{s}_i, \boldsymbol{\theta}_c^{(j)}, \boldsymbol{\theta}_V \right)$$

## The Algorithm (continued)

**Step 5:** Update  $\theta_V$  using Gradient Descent:

$$\theta_V^{(j)} = \theta_V^{(j-1)} - \eta \nabla_{\theta_V} \mathcal{L}(\theta_V^{(j-1)}) = \theta_V^{(j-1)} - \frac{\eta}{I} \sum_{i=1}^I HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)}) \nabla_{\theta_V} HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)})$$

## The Algorithm (continued)

**Step 5:** Update  $\theta_V$  using Gradient Descent:

$$\theta_V^{(j)} = \theta_V^{(j-1)} - \eta \nabla_{\theta_V} \mathcal{L}(\theta_V^{(j-1)}) = \theta_V^{(j-1)} - \frac{\eta}{I} \sum_{i=1}^I HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)}) \nabla_{\theta_V} HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)})$$

**Step 6:** Repeat Steps 3-5 until convergence.

## The Algorithm (continued)

**Step 5:** Update  $\theta_V$  using Gradient Descent:

$$\theta_V^{(j)} = \theta_V^{(j-1)} - \eta \nabla_{\theta_V} \mathcal{L}(\theta_V^{(j-1)}) = \theta_V^{(j-1)} - \frac{\eta}{I} \sum_{i=1}^I HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)}) \nabla_{\theta_V} HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)})$$

**Step 6:** Repeat Steps 3-5 until convergence.

At the end, we have approximations for the value function and policy function:

$$V(s) \approx V(s, \theta_V^{(*)}), \quad c(s) \approx c(s, \theta_c^{(*)})$$

## The Algorithm (continued)

**Step 5:** Update  $\theta_V$  using Gradient Descent:

$$\theta_V^{(j)} = \theta_V^{(j-1)} - \eta \nabla_{\theta_V} \mathcal{L}(\theta_V^{(j-1)}) = \theta_V^{(j-1)} - \frac{\eta}{I} \sum_{i=1}^I HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)}) \nabla_{\theta_V} HJB(s_i, \theta_c^{(j)}, \theta_V^{(j-1)})$$

**Step 6:** Repeat Steps 3-5 until convergence.

At the end, we have approximations for the value function and policy function:

$$V(s) \approx V(s, \theta_V^{(*)}), \quad c(s) \approx c(s, \theta_c^{(*)})$$

---

Summarizing:

- DNNs are a rich basis of functions with cheap derivatives and not so many parameters;
- They are an extremely flexible class;

## Wrap-Up

In two days, we covered a lot of ground!

- Always think about ML methods as an array of *functional bases*;
- A bit of LASSO and how to use it when we have many controls
- A bit of Random Forests and how to use them to estimate heterogeneous partial effects;
- A tiny drop of Deep Learning and how to tweak it to solve HJB equations;
- Dejanir Silva has a full class on this: <https://github.com/dejanirsilva/mlce>

## Wrap-Up

In two days, we covered a lot of ground!

- Always think about ML methods as an array of *functional bases*;
- A bit of LASSO and how to use it when we have many controls
- A bit of Random Forests and how to use them to estimate heterogeneous partial effects;
- A tiny drop of Deep Learning and how to tweak it to solve HJB equations;
- Dejanir Silva has a full class on this: <https://github.com/dejanirsilva/mlce>

**Important: I hope you had fun!**

# Thank you! 🙏

Scan below to get in touch!



# Appendix and References

# References

-  Bonhomme, Stéphane and Elena Manresa (May 2015). "Grouped Patterns of Heterogeneity in Panel Data: Grouped Patterns of Heterogeneity". In: *Econometrica* 83.3, pp. 1147–1184. ISSN: 0012-9682. DOI: [10.3982/ECTA11319](https://doi.org/10.3982/ECTA11319). URL: <http://dx.doi.org/10.3982/ECTA11319>.
-  Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone (1984). *Classification and Regression Trees*. en.
-  Cybenko, G. (Dec. 1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4, pp. 303–314. ISSN: 1435-568X. DOI: [10.1007/bf02551274](https://doi.org/10.1007/bf02551274). URL: <http://dx.doi.org/10.1007/BF02551274>.
-  Duarte, Victor, Diogo Duarte, and Dejanir H Silva (Sept. 2024). "Machine Learning for Continuous-Time Finance". In: *The Review of Financial Studies* 37.11. Ed. by Itay Goldstein, pp. 3217–3271. ISSN: 1465-7368. DOI: [10.1093/rfs/hhae043](https://doi.org/10.1093/rfs/hhae043). URL: <http://dx.doi.org/10.1093/rfs/hhae043>.
-  Hyafil, Laurent and Ronald L. Rivest (May 1976). "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1, pp. 15–17. ISSN: 0020-0190. DOI: [10.1016/0020-0190\(76\)90095-8](https://doi.org/10.1016/0020-0190(76)90095-8). URL: [http://dx.doi.org/10.1016/0020-0190\(76\)90095-8](http://dx.doi.org/10.1016/0020-0190(76)90095-8).
-  Lu, Zhou, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang (2017). "The Expressive Power of Neural Networks: A View from the Width". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/32cbf687880eb1674a07bf717761dd3a-Paper.pdf).
-  Masini, Ricardo and Marcelo Medeiros (2025). *Balancing Flexibility and Interpretability: A Conditional Linear Model Estimation via Random Forest*. DOI: [10.48550/ARXIV.2502.13438](https://arxiv.org/abs/2502.13438). URL: <https://arxiv.org/abs/2502.13438>.
-  Powell, Warren B (Sept. 2011). *Approximate dynamic programming*. en. 2nd ed. Wiley Series in Probability and Statistics. Wiley-Blackwell.