An Introduction to Solving Macroeconomic Models with Neural Networks

Raphaël Huleux [1]

[1]University of Copenhagen

Banque de France — 29 January 2026

Why Deep Learning in Macroeconomics?

- Solving most models implies a **dynamic maximization problem**
- **Dynamic programming** is subject to the **curse of dimensionality**
  $\rightarrow$ Number of computation grows exponentially with number of states
- We can't solve many interesting problems!
  $\rightarrow$ e.g. household problem with safe, risky, mortgage and housing?
- In GE, with HA, problem becomes even more untractable

The solution is DL?

Main idea in a nutshell:

- Use a **simulation method** instead of a grid
- Approximate value / policy functions with a **neural network**
- Improve the solution by using (stochastic) **gradient descent**
- Get derivatives using **auto-differentiation**

In practice:

- Use state-of-the-art libraries (Pytorch, Tensorflow, JAX, ...)
- Code is **simpler** than DP
- Bonus: **easy GPU** computing as well

Drawbacks and Limits

- A lot of hyper-parameters (learning rate, size of networks, etc)
- Much slower than DP for small models
- Hard to measure accuracy for interesting models
- Compute can be expensive and long
- Young field: no benchmark / workhorse algorithms yet

This talk

Outline:

1. Basics of Neural Networks
2. Solving the Ramsey model with DL
3. Adding uncertainty (= RBC model)
4. Adding constraints (= consumption-savings model)
5. An overview of the literature for solving HA + GE with DL

Important: Not my own research today!

Most of the results today loosely from Maliar, Maliar, Winant (2022)

Basics of Neural Networks

Definition of Neural Networks

A **neural network** is a parameterised function that maps an input vector x to an output vector $\hat{y}$:

$$\hat{y} = f(x; \theta),$$

where θ denotes the collection of parameters (weights W and biases b).

Each neuron does two things:

1. An affine transformation $z = \sum_i w_i x_i + b$;
2. Followed by a pointwise nonlinear transformation $a = \sigma(z)$,

where the output of one layer serves as the input to the subsequent layer.

Why use neural networks?

Cybenko, 1989; Hornik et al. (1989)

> *A* ***feedforward neural network*** *with a single hidden layer containing a finite number of neurons* ***can approximate any continuous function on compact subsets of*** $\mathbb{R}^n$ ***to arbitrary precision****, provided the activation function satisfies certain mild regularity conditions.*

Neural Network Notation

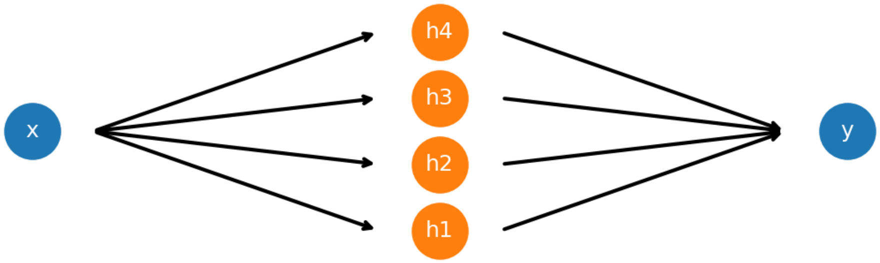
- A network takes as input a batch x of size (N^B, N^x)
  - N^B is the batch size (= number of "observations" in the dataset)
  - N^x the number of input feature (= number of "variables)
- $a_j^{(\ell)}$ is the output of neuron j in layer ℓ, with $a_j^{(\ell)} = \sigma(z_j^{(\ell)})$
  - with $z_{b,j}^{(\ell)} = \sum_i^{N^{\ell-1,h}} a_{b,i}^{(\ell-1)} W_{i,j}^{(\ell)} + b_j^{(\ell)}$
  - and $\sigma(z_j^{(\ell)})$ a (nonlinear) activation function
- Stacking over all neurons in a layer, we can write the affine transformation as $z^{(\ell)} = a^{(\ell-1)} W^{(\ell)} + b^{(\ell)}$

Today’s example

- Train a network with one hidden layer to approximate the function $y = \sin(x)$
- Example of supervised learning: you have pairs (x, y), find parameters such that $f(x;\theta) \approx y$
  (By opposition to unsupervised learning, you have x, minimize a loss)
- Simplest setup: $N^x = N^y = 1$
- We can sample "for free" from the real function
- Implement it without any package! "homemade"

Neural Network with One Layer

Network diagram: 1 input → 4 hidden neurons → 1 output



Approximate a function with a network (supervised-learning)

Algorithm:

- Initialize parameters θ of the network
- Draw pairs of (x, y) of the function we want to approximate (here, $y = \sin(x)$)
- For X iterations, do
  1. Using parameters θ, predict $\hat{y} = f(x;\theta)$
  2. Evaluate the loss function $L(\theta) = \frac{1}{N^b} \sum_i (\hat{y}_i - y_i)^2$
  3. Compute gradients of loss function with respect to parameters
  4. Update parameters using gradient descent: $\theta' = \theta - \eta \nabla L(\theta)$

     $\rightarrow \eta$ = learning rate

Let’s code!

Solving the Ramsey Model with DL

The Ramsey model

A household maximizes

$$V(K_{-1}) = \max_{\{K_t\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t u(C_t), \quad K_t + C_t = K_{t-1}^{\alpha} + (1-\delta)K_{t-1}.$$

The Euler equation writes

$$u'(C_t) = \beta(1 + \alpha K_t^{\alpha-1} - \delta)u'(C_{t+1})$$

Solving with DL (unsupervised learning, Maliar, et al, 2022)

Main idea

Approximate policy function using a neural network, targetting the Euler error.

Unsupervised learning: we sample x, but no y

Trick

Instead of approximating $K_t = g(K_{t-1})$, use $s \equiv K_t/(f(K_{t-1}) + (1-\delta)K_{t-1}) \in (0,1)$

$\rightarrow$ all choices are feasible!

Other algorithms

In the paper, they show value function and lifetime reward approaches.

Euler-residual minimization Algorithm

Algorithm:

- Initialize parameters θ of the network
- For X iterations:
  1. Draw a random sample of states K_{t-1} over $(\underline{K}, \overline{K})$
  2. Using the current policy function, compute C_t and C_{t+1}
  3. Evaluate the EE: $\beta u'(C_{t+1})(1+r)/u'(C_t) - 1$
  4. Compute the Mean Square Error $L(\theta) = \frac{1}{N} \sum_i EE_i^2$
  5. Update parameters using gradients

Let’s code!

Adding risk: the RBC model

The RBC model

Now, assume TFP follows a log AR-1 process:

$$V(K_{-1}) = \max_{\{K_t\}_{t=0}^{\infty}} \beta^t \sum_{t=0}^{\infty} \mathbb{E}[u(C_t)], \quad K_t + C_t = Z_t K_{t-1}^{\alpha} + (1-\delta)K_{t-1},$$

with

$$\log Z_t = \rho \log(Z_{t-1}) + \varepsilon_t$$

The Euler equation writes

$$u'(C_t) = \beta \mathbb{E}[(1 + \alpha Z_{t+1} K_t^{\alpha-1} - \delta) u'(C_{t+1})]$$

Computing expectations

Main difficulty: How to compute the expectation?

Three options:

1. Use standard quadrature method (but subject to curse of dimensionality)
2. Use Monte-Carlo method
3. Use the all-in-one operator of Maliar, Maliar, Winant (2022)

Two Expectations in the Objective Function

When solving the Euler equation for a given pair of states, we can write

$$J(K_{t-1}, Z_t) = u'(C_t) - \beta \mathbb{E}[(1 + \alpha Z_{t+1} K_t^{\alpha-1} - \delta) u'(C_{t+1})]$$

However, we want to solve this for all (relevant) (K_{t-1}, Z_t) (i.e., a global solution). We do so by writing the objective function as

$$L(\theta) = \mathbb{E}_{K,Z}[J(K_{t-1}, Z_t)^2]$$

- Quadrature or Monte-Carlo: for every (K_{t-1}, Z_t), compute expectation $\rightarrow$ works but computationnally heavy.
- Maliar, Maliar, Winant (2022): all-in-one operator

Let’s code!

All-in-one operator

Rewrite the objective function as one single expectation operator

$$L(\theta) = \mathbb{E}_{K,Z,\varepsilon_1,\varepsilon_2}[J(K_{t-1}, Z_t)|_{\varepsilon_1} J(K_{t-1}, Z_t)_{\varepsilon_2}]$$

By drawing two independent draws ε_1 and ε_2, and using

$$E_{\epsilon_1}\left[f(\epsilon_1)\right] E_{\epsilon_2}\left[f(\epsilon_2)\right] = E_{(\epsilon_1,\epsilon_2)}\left[f(\epsilon_1)f(\epsilon_2)\right].$$

Let’s code!

Consumption-saving model

A Consumption-Saving model

Partial equilibrium model of household behavior:

$$V(A_{t-1}, Z) = \sum_{t=0}^{\infty} \beta^t u(C_t), \quad C_t + A_t = A_{t-1}(1+r) + Z_t$$

where Z_t is log AR-1, and we impose a non-borrowing constraint $A_t \geq 0$.

The Euler equation writes

$$u'(C_t) = \beta \mathbb{E}[u'(C_{t+1})](1+r) + \lambda_t$$

where $\lambda_t \geq 0$ and $\lambda_t A_t = 0$.

How do we account for λ_t and the KKT condition?

Approximating the Lagrange Multiplier

- Networks can have mutiplie outputs
- Add a new output $\lambda = g^2(A_{t-1}, Z_t)$
- Error is now: $L(\theta) = 1/N \sum_i (EE_i + \lambda A_{t,i})^2$

But this is harder to learn for the network, because it needs to learn the relationship between λ and the EE.

Avoiding the Lagrange Multiplier

- If $u'(C_t) > \beta\mathbb{E}[u'(C_{t+1})](1+r)$, then $A_t = 0$, and $s \equiv g(A_{t-1}, Z_t) = 0$
- So we have either $u'(C_t) = \beta\mathbb{E}[u'(C_{t+1})](1+r)$ or $s = g(A_{t-1}, Z_t) = 0$
- We use the Fisher-Burmeister $\phi(a,b) = a + b - \sqrt{a^2+b^2}$ to impose this

Literature Review

Partial review of DL papers with GE

1. Maliar, Maliar, Winant (2022) Deep learning for solving dynamic economic models
2. Azinovic, Gaegauf, Schedegger (2022) Deep Equilibrium Nets
3. Kase, Melosi, Rottner (2024) Estimating Nonlinear Heterogeneous Agents Models with Neural Networks
4. Han, Yang (2025) DeepHAM: A Global Solution Method
5. Druedahl, Huleux, Røpke (2025) Deep Learning Solutions of Large Non-Convex Life-Cycle Models
6. Azinovic-Yang, Zemlicka (2025) Deep learning in the sequence space
7. Druedahl, Huleux, Røpke (2026) Global Canonical HANK (early stage)

Deep Equilibrium Nets (Azinovic et al., 2022)

- OLG model with aggregate risk and occasionally binding constraints.
- State x includes the aggregate shock and the endogenous cross-sectional state
- A neural network parameterizes the *functional rational expectations equilibrium*:

  $$x \;\mapsto\; \theta(x),$$

  where $\theta(x)$ bundles all equilibrium objects at state x (policies, prices, and KKT multipliers)
- Output dimension is $4N + 1$: capital, bonds, multipliers for each cohort, plus the bond price
- Network is trained by minimizing the MSR of
  - Euler equations (for each asset and each type),
  - complementarity/KKT conditions (occasionally binding constraints),
  - market-clearing conditions.

DeepHAM (Han et al, 2022)

- Krusell–Smith model with aggregate uncertainty $\rightarrow$ whole distribution becomes a state variable
- Replace the full distribution F_t by *generalized moments*

$$Q_t = \frac{1}{N} \sum_i Q(a_i),$$

  where $Q(\cdot)$ is a learned neural basis.

  (a_i, z_i, Z_t, Q_t) is the effective state; Q_t is permutation-invariant and acts as a learned sufficient statistic.
- Neural networks parameterize policy and value functions

$$c_i = C(a_i, z_i, Z_t, Q_t), \qquad V_i = V(a_i, z_i, Z_t, Q_t),$$

  with prices from firm FOCs and market clearing.
- Simulation-based, global solution:
  - simulate the economy under current policies (ergodic distribution),

Global MA Solution Algorithm

Idea: Solve HANK globally in MA form by learning policy functions as maps from shock histories to allocations.

State

$$\mathcal{E}_t = (\varepsilon_t, \varepsilon_{t-1}, \ldots, \varepsilon_{t-\#E}) \quad \Rightarrow \quad (Y_t, c_t(\cdot))$$

Policies (NNs)

$$Y_t = \Pi(\mathcal{E}_t), \qquad c(z, m; \mathcal{E}_t) = \pi(\mathcal{E}_t)$$

Simulation (given $\mathcal{E}_t$)

- Update government (B_t, T_t)
- EGM $\Rightarrow$ household policies
- Asset-market clearing $\Rightarrow Y_t$
- Forward distribution update