Estimating Nonlinear Heterogeneous Agents Models with Neural Networks

Hanno Kase[1], Leonardo Melosi[2], Matthias Rottner[3]

ASSA, January 8, 2023

[1]University of Minnesota
[2]FRB Chicago, CEPR
[3]Deutsche Bundesbank

Macroeconomic models of the future

Include features like:

- Heterogeneous agents facing idiosyncratic risks
- Aggregate uncertainty and nonlinearities

Hard to solve because of their elevated complexity

Difficult to estimate, usually requires **repeated** solving

This paper

- **Develop estimation procedure based on neural networks**
- **Apply to nonlinear HANK model**

Key innovations

There are two key innovations tackling different estimation bottlenecks

1. **Extended Neural Network**
   Allows us to **avoid repeated solving** the model

2. **Neural Network Based Particle Filter**
   Dramatically reduce the **cost of likelihood evaluations**

Solution procedure using deep neural networks

- Building on Maliar, Maliar, and Winant (2021) Euler residual minimization

0. Instead of continuum of agents, there are L agents
1. Parameterize individual and aggregate policy functions with deep neural networks

$$\psi_t^i = \psi_{NN}^I(\mathbb{S}_t^i, \mathbb{S}_t|\Theta) \quad \text{and} \quad \psi_t^A = \psi_{NN}^A(\mathbb{S}_t|\Theta)$$

   Where $\mathbb{S}_t = \{\{\mathbb{S}_t^i\}_{i=1}^L, \mathbb{S}_t^A\}$ is a vector of state variables
   Θ is the set of parameters of the model

2. Construct loss function - weighted mean of squared residuals
3. Train the deep neural networks using stochastic optimization
   - Minimize the loss for points drawn from the state space
   - Simulate model forward to generate a new draw from the state space

Training the neural networks repeatedly would take too long for estimation

Avoid repeated solving - Extended Neural Network

- **Treat model parameters as pseudo state variables**

0. Instead of continuum of agents, there are L agents
1. Parameterize individual and aggregate policy functions with deep neural networks

$$\psi_t^i = \psi_{NN}^I(\mathbb{S}_t^i, \mathbb{S}_t, \boldsymbol{\tilde{\Theta}} | \bar{\Theta}) \quad \text{and} \quad \psi_t^A = \psi_{NN}^A(\mathbb{S}_t, \boldsymbol{\tilde{\Theta}} | \bar{\Theta})$$

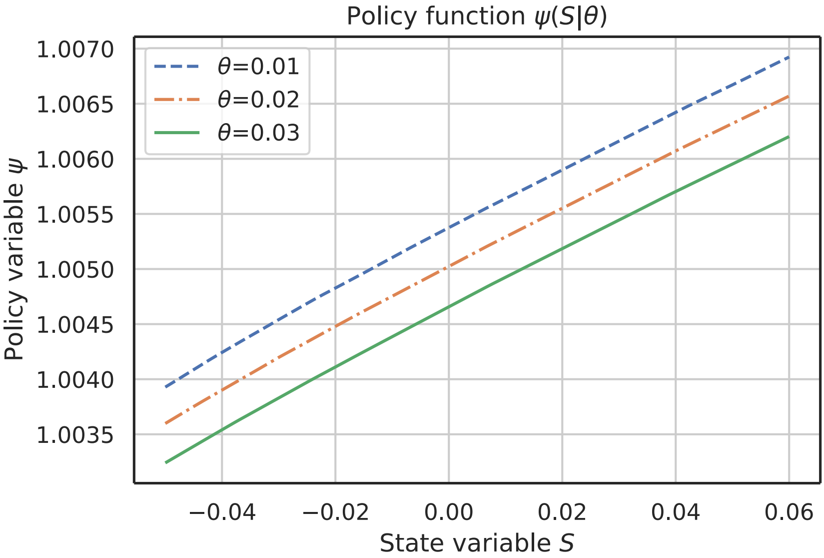
Where $\mathbb{S}_t = \{\{\mathbb{S}_t^i\}_{i=1}^L, \mathbb{S}_t^A\}$ is a vector of state variables

$\bar{\Theta}$ **is the set of calibrated and** $\boldsymbol{\tilde{\Theta}}$ **estimated parameters of the model**

2. Construct loss function - weighted sum of mean of squared residuals
3. Train the deep neural networks using stochastic optimization
   - Minimize the loss for points drawn from the state space
   - **Draw new values for parameters $\boldsymbol{\tilde{\Theta}}$ we are interested in estimating**
   - Simulate model forward to generate a new draw from the state space

More complex problem, but we only need to train the networks ONCE!

Extended Neural Network - output from ONE neural network



Costly likelihood evaluation - Neural Network Based Particle Filter

For nonlinear models we can obtain the likelihood using the **particle filter**

- Model needs to be **evaluated** for thousands of particles and multiple time periods
- Particle filter becomes the bottleneck for estimation

Costly likelihood evaluation - Neural Network Based Particle Filter

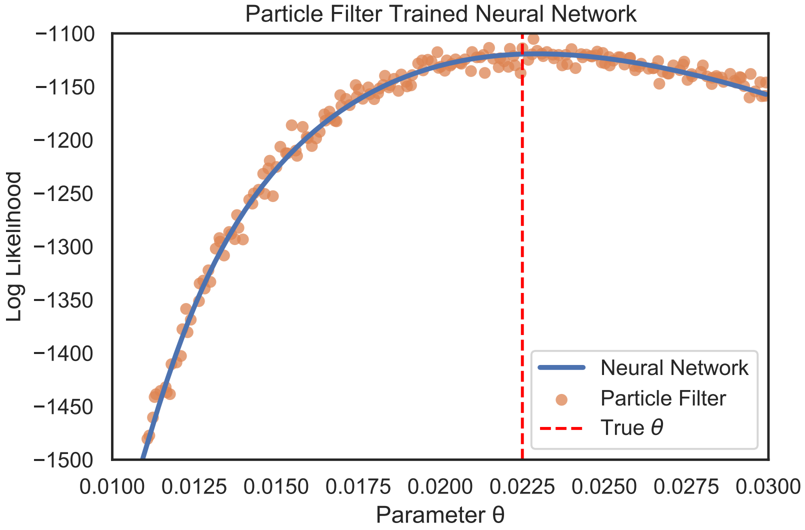
Solution:

1. Create a dataset of parameter values and log likelihoods
2. Split the dataset into training and validation samples
3. Train a neural network on the training sample
    - Use the validation sample to avoid overfitting

Benefits:

- Single likelihood evaluation can be done almost instantly
  $\Rightarrow$ **Allows for a large number of draws in Metropolis-Hastings algorithm**
- Easily parallelized
  $\Rightarrow$ **We can use multiple GPUs to create a bigger dataset**
- Smooths out noise from the particle filter
  $\Rightarrow$ **We can use less particles in the filter**

Neural Network Based Particle Filter - one parameter



Recap

Two key innovations

1. **Extended Neural Network** - avoid repeated solving
2. **Neural Network Based Particle Filter** - fast likelihood evaluations

Proof of the pudding is in the eating

1. Compare the extended NN based solution to a benchmark
   - Linearized three equation NK model with an analytical solution

   Extended Neural Network matches the true solution

2. Compare the estimation results to a conventional method
   - Simple nonlinear RANK model with a ZLB

   Estimation results are very similar

3. Estimating a nonlinear HANK model

Linearized NK model

- Small linearized three equation NK model with a TFP shock

$$
\begin{aligned}
\hat{X}_t &= E_t\hat{X}_{t+1} - \sigma^{-1}\left(\phi_\Pi\hat{\Pi}_t + \phi_Y\hat{X}_t - E_t\hat{\Pi}_{t+1} - \hat{R}_t^F\right)\\
\hat{\Pi}_t &= \kappa\hat{X}_t + \beta E_t\hat{\Pi}_{t+1}\\
\hat{R}_t^F &= \rho_A\hat{R}_{t-1}^F + \sigma(\rho_A - 1)\omega\sigma_A\epsilon_t^A
\end{aligned}
$$

Where $\hat{X}$: output gap, $\hat{\Pi}$: inflation, R^F: risk free rate, ϵ^A: TFP shock

- Analytical solution:

$$
\begin{aligned}
\hat{X}_t &= \frac{1-\beta\rho_A}{(\sigma(1-\rho_A)+\theta_Y)(1-\beta\rho_A)+\kappa(\theta_\Pi-\rho_A)}\hat{R}_t^F,\\
\hat{\Pi}_t &= \frac{\kappa}{(\sigma(1-\rho_A)+\theta_Y)(1-\beta\rho_A)+\kappa(\theta_\Pi-\rho_A)}\hat{R}_t^F.
\end{aligned}
$$

Solving the linearized NK model with an Extended Neural Network

1. Parametrize the policy function with a deep neural network:

$$\begin{pmatrix} \hat{X}_t \\ \hat{\Pi}_t \end{pmatrix} = \psi(\underbrace{\hat{R}_t^F}_{\mathbb{S}_t}, \underbrace{\beta, \sigma, \eta, \phi, \theta_\Pi, \theta_Y, \rho_A, \sigma_A}_{\tilde{\Theta}}) \approx \psi_{NN}\left(\hat{R}_t^F, \beta, \sigma, \eta, \phi, \theta_\Pi, \theta_Y, \rho_A, \sigma_A\right)$$

2. Construct the loss function:

$$err_1 = \hat{X} - \left(E_t \hat{X}_{t+1} - \sigma^{-1}\left(\phi_\Pi \hat{\Pi}_t + \phi_Y \hat{X}_t - E_t \hat{\Pi}_{t+1} - \hat{R}_t^F\right)\right)$$

$$err_2 = \hat{\Pi}_t - \left(\kappa \hat{X}_t + \beta E_t \hat{\Pi}_{t+1}\right)$$

$$L = w_1 \frac{1}{B} \sum_{I=1}^{B} (err_1^i)^2 + w_2 \frac{1}{B} \sum_{i=1}^{B} (err_2^i)^2 \quad \text{, where } B \text{ is the batch size}$$

3. Train the deep neural networks using stochastic optimization ...

Solving the linearized NK model with an Extended Neural Network

3. Train the deep neural networks using stochastic optimization
    - Batch size of 500 (parallel worlds)
    - 100 000 iterations
        1. **Draw parameters** from a bounded parameter space:

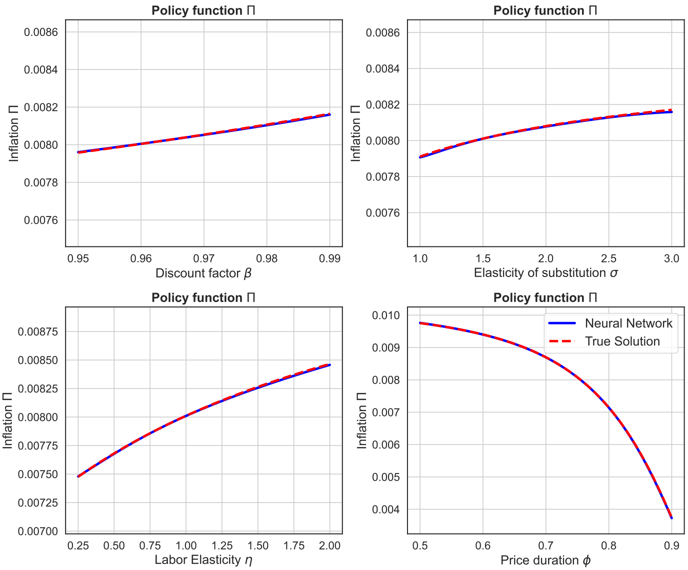
Parameters		LB	UB	Parameters		LB	UB
β	Discount factor	0.95	0.99	θ_Π	MP inflation response	1.25	2.5
σ	Relative risk aver.	1	3	θ_Y	MP output response	0.0	0.5
η	Inverse Frisch elas.	1	4	ρ_A	Persistence TFP shock	0.8	0.95
ϕ	Price duration	0.5	0.9	σ_A	Std. dev. TFP shock	0.02	0.1

        2. **Draw points from the state space** by simulating the model:

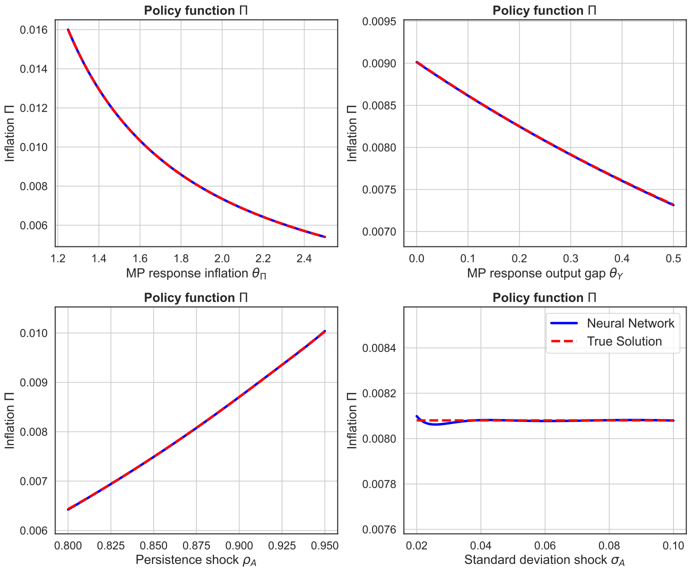
$$\hat{R}_t^F = \rho_A \hat{R}_{t-1}^F + \sigma(\rho_A - 1)\omega\sigma_A\epsilon_t^A$$

        3. **Compute the loss** L
        4. **Optimizer step** (ADAM) to adjust the weights of the NN to minimize L

Extended Neural Network: Inflation over the parameter space



Extended Neural Network: Inflation over the parameter space



Compare the estimation results to a conventional method

- Simple RANK model
  - Only a preference shock
  - Zero lower bound
- Interesting laboratory:
  1. Simple enough to solve and estimate with conventional methods
  2. No solution if the volatility of the demand shock is too large

Estimation comparison

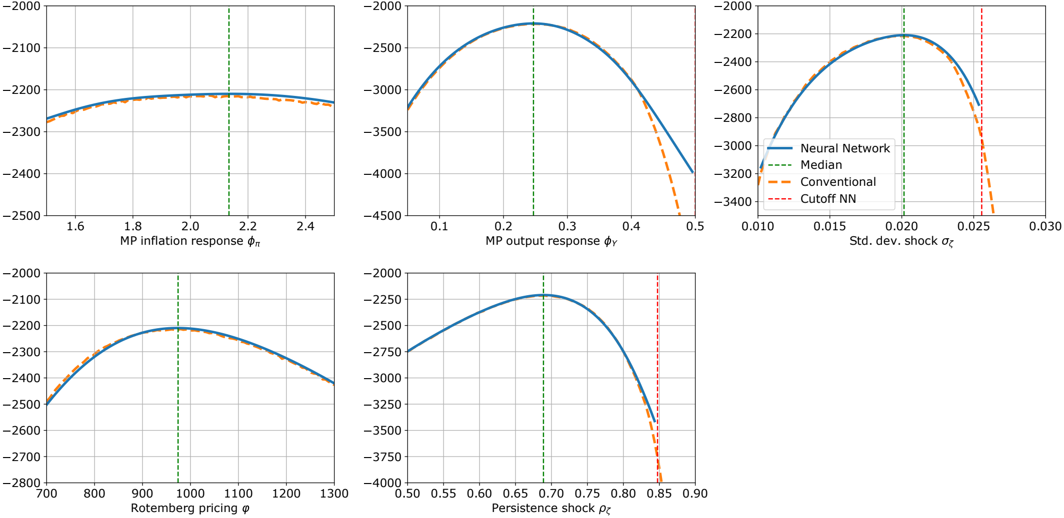
- Use the model to create time series for: output growth, inflation, interest rate
- Recover 5 parameter values using:
  1. Neural networks based approach (**extended NN, NN based PF**, RWMH)
  2. Conventional approach (time iteration, regular particle filter, RWMH)

Estimation comparison

- Use the model to create time series for: output growth, inflation, interest rate
- Recover 5 parameter values using:
  1. Neural networks based approach (**extended NN, NN based PF**, RWMH)
  2. Conventional approach (time iteration, regular particle filter, RWMH)

			Neural Network Posterior			Conventional Approach Posterior		
Parameter		True value	Median	5%	95%	Median	5%	95%
θ_Π	Inflation resp.	2.0	2.02	1.87	2.17	2.06	1.94	2.20
θ_Y	Output resp.	0.25	0.251	0.238	0.263	0.248	0.237	0.258
φ	Rotemberg	1000	988.6	935.1	1036.7	973.7	911.2	1037.2
ρ_ζ	Persistence	0.8	0.686	0.669	0.701	0.691	0.670	0.710
σ^ζ	Std. dev.	0.02	0.020	0.020	0.021	0.020	0.019	0.020

Estimation comparison: posterior



Proof of the pudding is in the eating

1. ~~Compare the extended NN based solution to a benchmark~~
    - Linearized three equation NK model with an analytical solution

    Extended Neural Network matches the true solution

2. ~~Compare the estimation results to a conventional method~~
    - Simple nonlinear RANK model with a ZLB

    Estimation results are very similar

3. **Estimating a nonlinear HANK model**

Estimating a nonlinear HANK model

Solve and estimate a medium scale nonlinear HANK model

- Solve the model
- Create time series
- Forget the true parameters
- Recover the parameters

Estimating a nonlinear HANK model

- Households face idiosyncratic income risk s_t^i and a **borrowing limit** $\underline{B}$

$$E_0 \sum_{t=0}^{\infty} \beta^t \exp(\zeta_t^D) \left[\left(\frac{1}{1-\sigma} \right) (C_t^i - hC_{t-1})^{1-\sigma} - \chi \left(\frac{1}{1+\eta} \right) (H_t^i)^{1+\eta} \right]$$

$$\text{s.t. } C_t^i + B_t^i = W_t s_t^i H_t^i + \frac{R_{t-1}}{\Pi_t} B_{t-1}^i - T_t^i + Div_t^i$$

$$B_t \geq \underline{B}$$

where idiosyncratic risk follows an AR(1) process: $s_t^i = \rho_s s_{t-1}^i + \sigma_s \epsilon_t^i$

- Aggregate shocks: preference ζ^D, growth rate g_t and monetary policy mp_t
- Consumption habit h and persistence in the monetary policy rule ρ_R
- Monetary policy is constrained by the **zero lower bound**

$$R_t = \max \left[1, (R_{t-1}^N)^{\rho_R} \left(R \left(\frac{\Pi_t}{\Pi} \right)^{\theta_\Pi} \left(\frac{Y_t}{Z_t Y} \right)^{\theta_Y} \right)^{1-\rho_R} \exp(mp_t) \right]$$

Estimating a nonlinear HANK model

We are interested in finding the **policy functions over parameter ranges**

0. Instead of continuum of agents there are $L = 100$ agents
1. Policy functions parameterized by deep neural networks
    - Aggregate: inflation and wage
    - Individual: labor choice and multiplier on borrowing constraint
    - **219 state variables**
        - 2 individual, 200 distribution, 5 aggregate and 12 pseudo (parameters) states
2. Loss function is a weighted sum of squared residuals of:
    - Euler equation
    - Fisher-Burmeister eq. for borrowing limit
    - NKPC
    - Bond market clearing
    - Product market clearing
3. Train the deep neural networks ...

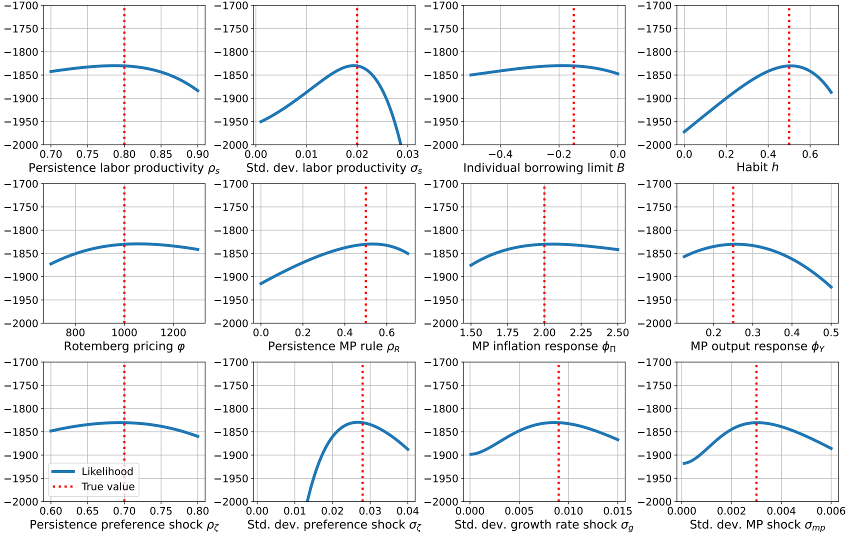
Estimating a nonlinear HANK model

3. Train the deep neural networks
    - Batch size 100 (parallel worlds)
    - 200 000 iterations
    - ADAM optimizer
    - Curriculum learning (RANK $\rightarrow$ HANK $\rightarrow$ HANK with ZLB ...)

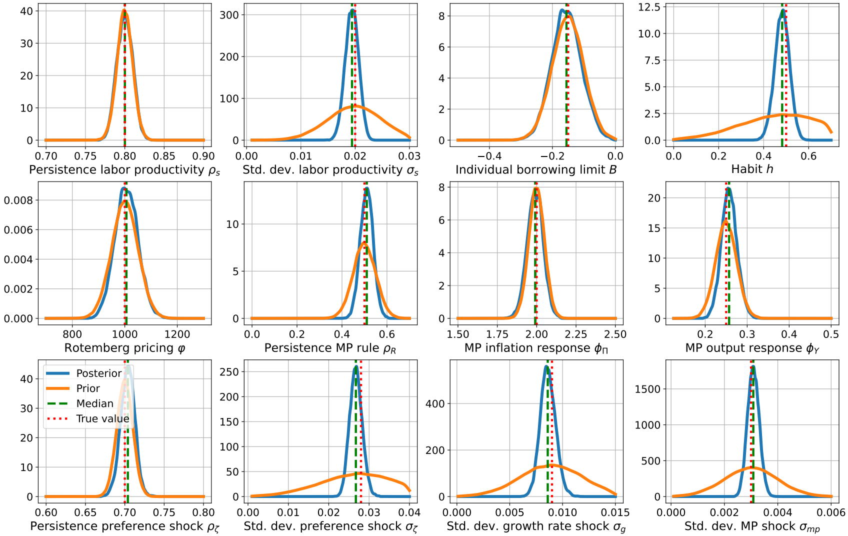
Estimation experiment:

- Using the calibrated model to create time series for:
    - Output growth
    - Inflation
    - Interest rate
- Recover 12 parameters
    1. Generate a dataset of 10 000 parameter values and corresponding log likelihoods
    2. Train the **Neural Network Based Particle Filter**
    3. Metropolis-Hastings algorithm

Output of the Neural Network Based Particle Filter



Estimated posterior distributions of parameters



Conclusion

Novel estimation procedure based on neural networks

1. **Extended Neural Network** - avoid repeated solving
2. **Neural Network Based Particle Filter** - fast likelihood evaluations

Possible to estimate high dimensional models

Appendices

What if there is no solution?

- The Neural Network that approximates the policy functions is trained by minimizing weighted mean of squared residuals
- For parameter values where there is no solution:
  - Conventional solution method: **an error**
  - Extended Neural Network: **a value, but the residual is larger**

Introduce additional neural network that maps parameters to residual error

1. Create a dataset of parameter values and residuals
2. Split the dataset into training and validation samples
3. Train a neural network on the training sample
4. Pick a cutoff value to **discard bad solutions**

Solution to "What if there is no solution?"