BCF mini course: Deep Learning and Macro-Finance Models

Goutham Gopalakrishna

École Polytechnique Fédérale de Lausanne (EPFL)
Swiss Finance Institute (SFI)
VSRC, Princeton University

February, 2023

Princeton University
Roadmap

■ Part-1: Introduction to numerical methods, challenges faced by traditional methods
  ➢ Why neural networks and deep learning
  ➢ Function approximators
  ➢ Comparison with existing methods

■ Part-2: Deep learning principles, high-dimensional optimization techniques in machine learning
  ➢ Gradient descent and variants
  ➢ Under the hood: Activation functions, Parameter initialization
  ➢ Object oriented programming principles

■ Part-3: Application to solve macro-finance models with aggregate shocks
References

- Course materials (slides and code): Github page.
ALIENs: What is it about?

- **ENs**: Use neural network to solve general equilibrium continuous time finance models to capture global dynamics (portfolio choice, macro-finance, monetary policy)

ALIENs: What is it about?

- **I**: Encode economic information as regularizer
- **ENs**: Use neural network to solve general equilibrium continuous time finance models to capture global dynamics (portfolio choice, macro-finance, monetary policy)

**ALIENs: What is it about?**

- **AL**: Actively learn about state space with stark non-linearity/large prediction error
- **I**: Encode economic information as regularizer
- **ENs**: Use neural network to solve general equilibrium continuous time finance models to capture global dynamics (portfolio choice, macro-finance, monetary policy)

General setup

\[ U_t = E_t \left[ \int_t^\infty f(c_s, U_s) \, ds \right] \] (1)

Exogenous dividend process of risky asset

Brownian shock

There is also a risk free debt market (pays return \( r \)). Risky asset has price of risk \( \zeta_t \), and volatility \( \sigma_R \).

Problem of the agent is

\[ \sup c, \theta \quad U_t \] (3)

s.t

\[ dw_t = p \left[ \theta_t \right] \, dt \]

port. choice

price of risk \( \hat{c}_t \)

\[ \theta_t \sigma_R \] return volatility \( dZ_t \) (4)

If \( g, \sigma, r \) are time varying, then we have a multi-dimensional problem
General setup

- $U_t = E_t\left[\int_t^\infty f(c_s, U_s) ds\right]$ (1)

- Exogenous dividend process of risky asset

$$\frac{dy_t}{y_t} = gdt + \sigma \underbrace{dZ_t}_{\text{Brownian shock}}$$ (2)
General setup

- General setup

\[ U_t = E_t \left[ \int_t^\infty f(c_s, U_s) ds \right] \]  \hfill (1)

- Exogenous dividend process of risky asset

\[ \frac{dy_t}{y_t} = gdt + \sigma \quad dZ_t \]  \hfill (2)

Brownian shock

- There is also a risk free debt market (pays return \( r \)). Risky asset has price of risk \( \zeta_t \), and volatility \( \sigma^R_t \)

- Problem of the agent is

\[ \sup_{\hat{c}, \theta} U_t \]  \hfill (3)

\[ \text{s.t} \quad \frac{dw_t}{w_t} = (r + \theta_t \zeta_t - \hat{c}_t) dt + \theta_t \sigma^R_t dZ_t \]  \hfill (4)

\[ \text{port. choice price of risk ret. volatility} \]

- If \( g, \sigma, r \) are time varying, then we have a multi-dimensional problem
HJB

- HJB is
  \[
  \sup f(c_t, U_t) + E_t(dU_t) = 0
  \]

- Conjecturing \( U = \frac{Jw^{1-\gamma}}{1-\gamma} \), where \( J \) is the stochastic opportunity process and \( \gamma \) is the risk aversion, the HJB equation reduces to
  \[
  \mu^J(x, J) J = \sum_{i=1}^{d} \mu^x_i(x, J) \frac{\partial J}{\partial x_i} + \sum_{i,j=1}^{d} b^{i,j}(x, J) \frac{\partial^2 J}{\partial x_i \partial x_j} \tag{5}
  \]

  1. State variables are \( x \). Could be high-dimensional (large \( d \))
  2. \( \mu^J, \mu^x, \) and \( b^{i,j} \) are linear, advection, and diffusion coefficients

- PDE (5) can be highly non-linear elliptical PDE depending on the problem

- Past literature: Convert it into **quasi-linear parabolic PDE** and use finite difference → slowly introduce non-linearity through
  \[
  \mu^J(x, J^{old}) J = \frac{\partial J}{\partial t} + \sum_{i=1}^{d} \mu^x_i(x, J^{old}) \frac{\partial J}{\partial x_i} + \sum_{i,j=1}^{d} b^{i,j}(x, J^{old}) \frac{\partial^2 J}{\partial x_i \partial x_j} \tag{6}
  \]

- Works well in low dimensions, but breaks down in high dimensions (d'Adrien and Vandeweyer, 2019)
Methodology overview

Focus of this part is to introduce a technique to solve macro models involving PDEs of type (5) in high dimensions

1. Benchmark model (BS2016 with recursive preference)
2. Capital misallocation model with productivity shock (Gopalakrishna 2021)

Figure: Overview of methodology.
Neural network solution method

\[ f := \frac{\partial \hat{J}}{\partial t} + \sum_{i}^{d} \mu^{i}(x) \frac{\partial \hat{J}}{\partial x_i} + \sum_{i,j=1}^{d} b^{ij}(x) \frac{\partial^2 \hat{J}}{\partial x_i \partial x_j} - \mu^{J} \hat{J} = 0; \]

\[ \forall (t, x) \in [ T - k\Delta t, T - (k - 1)\Delta t ] \times \Omega \]

\[ \hat{J} = \tilde{J}_0 \quad \forall (t, x) \in ( T - (k - 1)\Delta t ) \times \Omega; \]

where \( \hat{J} \) is a neural network object with parameters \( \Theta \), and \( f \) is the PDE residual.
Neural network solution method

\[ f := \frac{\partial \hat{J}}{\partial t} + \sum_{i} \mu^i(x) \frac{\partial \hat{J}}{\partial x_i} + \sum_{i,j=1}^{d} b^{i,j}(x) \frac{\partial^2 \hat{J}}{\partial x_i \partial x_j} - \mu^J \hat{J} = 0; \]

\[ \forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega \]

\[ \hat{J} = \tilde{J}_0 \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \Omega; \]

where \( \hat{J} \) is a neural network object with parameters \( \Theta \), and \( f \) is the PDE residual. Can be seen as a classical constrained optimization problem

\[ \Theta^* = \arg\min_{\Theta} \hat{J} - \tilde{J}_0 \]

s.t. \( f = 0 \)
Neural network solution method

\[ f := \frac{\partial \hat{J}}{\partial t} + \sum_{i} \mu^{i}(x) \frac{\partial \hat{J}}{\partial x_{i}} + \sum_{i,j=1}^{d} b^{i,j}(x) \frac{\partial^{2} \hat{J}}{\partial x_{i} \partial x_{j}} - \mu^{J} \hat{J} = 0; \]

\[ \forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega \]

\[ \hat{J} = \tilde{J}_{0} \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \Omega; \]

Can be seen as an classical constrained optimization problem

Optimization

\[ \Theta^{*} = \arg\min_{\Theta} \hat{J} - \tilde{J}_{0} \]

s.t. \[ \int_{t} \int_{x} |f|^{2} dtdx = 0 \]
Neural network solution method

\[ f := \frac{\partial \hat{J}}{\partial t} + \sum_{i}^{d} \mu^i(x) \frac{\partial \hat{J}}{\partial x_i} + \sum_{i,j=1}^{d} b^{i,j}(x) \frac{\partial^2 \hat{J}}{\partial x_i \partial x_j} - \mu^J \hat{J} = 0; \]

\[ \forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega \]

\[ \hat{J} = \tilde{J}_0 \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \Omega; \]

\[ \frac{\partial \hat{J}}{\partial x} = J_0 \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \partial \Omega; \]

- Mesh free since we can randomly sample from the state space \((t, x)\) to train the neural network
Neural network solution method

\[
f := \frac{\partial \hat{J}}{\partial t} + \sum_{i}^{d} \mu^i(x) \frac{\partial \hat{J}}{\partial x_i} + \sum_{i,j=1}^{d} b^{ij}(x) \frac{\partial^2 \hat{J}}{\partial x_i \partial x_j} - \mu^J \hat{J} = 0; \\
\forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega \\
\hat{J} = \tilde{J}_0 \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \Omega; \\
\frac{\partial \hat{J}}{\partial x} = J_0 \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \partial \Omega;
\]

- Mesh free since we can randomly sample from the state space \((t, x)\) to train the neural network.
- Sparse training points in region of importance leads to instability in future iterations.  
  **Solution:** Track subdomain \(\Omega_c\) and sample more points from there

\[
f = 0 \quad \forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega_c; \\
\hat{J} = \tilde{J}_0 \quad \forall (x, t) \in (T - (k - 1)\Delta t) \times \Omega_c;
\]

- The subdomain \(\Omega_c\) is found by inspecting the PDE coefficients which are determined using previous value \(\tilde{J}\)
Neural network solution method

\[ f := \frac{\partial \hat{J}(x|\Theta)}{\partial t} + \sum_{i}^{d} \mu_{i}(x) \frac{\partial \hat{J}(x|\Theta)}{\partial x_{i}} + \sum_{i,j=1}^{d} b_{i,j}(x) \frac{\partial^{2} \hat{J}(x|\Theta)}{\partial x_{i} \partial x_{j}} \]

\[ - \mu^{J} \hat{J}(x|\Theta) = 0; \quad \forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega \]

\[ \hat{J}(x|\Theta) = \tilde{J}_{0}; \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \Omega; \]

\[ (f = 0; \quad \forall (t, x) \in [T - k\Delta t, T - (k - 1)\Delta t] \times \Omega_{c}; \]

\[ \hat{J}(x|\Theta) = \tilde{J}_{0}; \quad \forall (t, x) \in (T - (k - 1)\Delta t) \Omega_{c}; \rightarrow \text{Active learning} \]

\[ \frac{\partial \hat{J}(x|\Theta)}{\partial x} = J_{0}; \quad \forall (t, x) \in (T - (k - 1)\Delta t) \times \partial \Omega; \]

- \( X \in \mathbb{R}^{d} \) Space dimension
- \( t \in [0, T] \) Time dimension
- \( \sigma \) Tanh activation function. \( \sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \)
- \( \hat{J}(x | \Theta) \) Output from neural network
Active learning

Example from Gopalakrishna (2021): Macro-finance model with 2 state variables (productivity, wealth share)
Active learning

Example from Gopalakrishna (2021): Macro-finance model with 2 state variables (productivity, wealth share)
Active learning

Example from Gopalakrishna (2021): Macro-finance model with 2 state variables (productivity, wealth share)
Solution technique: ALIENs

Figure: Methodology.
Solution technique: ALIENs

Figure: Methodology.

\[ f = \frac{\partial \hat{J}}{\partial t} + \sum_{i}^{d} \mu^i(x) \frac{\partial \hat{J}}{\partial x_i} + \sum_{i,j=1}^{d} b^{i,j}(x) \frac{\partial^2 \hat{J}}{\partial x_i \partial x_j} - \mu^f \hat{J}; \]
\[ \forall (t, x) \in [T - \Delta t, T] \times \Omega \]
Solution technique: ALIENs

Figure: Methodology.

\[ f = \frac{\partial j}{\partial t} + \sum_{i} \mu^{i}(x) \frac{\partial j}{\partial x_{i}} + \sum_{i,j=1}^{d} b^{i,j}(x) \frac{\partial^{2} j}{\partial x_{i} \partial x_{j}} - \mu^{j} \dot{j}; \]
\[ \forall (t, x) \in [T - \Delta t, T] \times \Omega \]

\[ \mathcal{L}_{f} \]
\[ \oplus \]
\[ \text{LOSS} \]
\[ \Theta^{*} \]

Minimize (ADAM + LBFGS)
\[ \mathcal{L} = \lambda_f \mathcal{L}_f + \lambda_j \mathcal{L}_j + \lambda_b \mathcal{L}_b + \lambda_c^1 \mathcal{L}_c^1 + \lambda_c^2 \mathcal{L}_c^2 \]  

(7)

where

- **PDE loss**
  \[ \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(x^i_f, t^i_f)|^2 \]

- **Bounding loss-1**
  \[ \mathcal{L}_j^1 = \frac{1}{N_j} \sum_{i=1}^{N_j} |\hat{J}(x^i_j, t^i_j) - \tilde{J}_0|^2 \]

- **Active loss-1**
  \[ \mathcal{L}_c^2 = \frac{1}{N_c} \sum_{i=1}^{N_c} |f(x^i_c, t^i_c)|^2 \]

- **Active loss-2**
  \[ \mathcal{L}_c^1 = \frac{1}{N_c} \sum_{i=1}^{N_c} |\hat{J}(x^i_c, t^i_c) - \tilde{J}_0|^2 \]
Active Learning vs Simulation method

- ALIENs actively learn the region of sharp transition and samples more points → faster convergence
- Sampling procedure is complementary to simulation based methods (Azinovic et al (2018), Villaverde et al (2020)), but also works for models with rare events and financial constraints that bind far away from the steady state
Automatic differentiation in practice

Approximating $J$ using a neural network

```python
def J(z,t):
    J = neural_net(tf.concat([z,t],1),weights,biases)
    return J
```

Constructing regularizer: 1D model

```python
def f(z,t):
    J = J(z,t)
    J_t = tf.gradients(J,t)[0]
    J_z = tf.gradients(J,z)[0]
    J_zz = tf.gradients(J_z,z)[0]
    f = J_t + advection * J_z + diffusion * J_zz - linearTerm * J
    return f
```
Approximating $J$ using a neural network

```python
def J(z,t):
    J = neural_net(tf.concat([z,t],1), weights, biases)
    return J
```

Constructing regularizer: 1D model

```python
def f(z,t):
    J = J(z,t)
    J_t = tf.gradients(J,t)[0]
    J_z = tf.gradients(J,z)[0]
    J_zz = tf.gradients(J_z,z)[0]
    f = J_t + advection * J_z + diffusion * J_zz - linearTerm * J
    return f
```

```python
def f(z,a,t):
    J = J(z,a,t)
    J_t = tf.gradients(J,t)[0]
    J_z = tf.gradients(J,z)[0]
    J_a = tf.gradients(J,a)[0]
    J_zz = tf.gradients(J_z,z)[0]
    J_aa = tf.gradients(J_a,a)[0]
    J_az = tf.gradients(J_a,z)[0]
    f = J_t + advection_z * J_z + advection_a * J_a + diffusion_z * J_zz +
    diffusion_a * J_aa + crossTerm * J_az - linearTerm * J
    return f
```
Horovod

- Data parallelism as opposed to Model parallelism
- Horovod uses ringAllReduce operation to average gradients (improves efficiency)

Figure: Source: https://eng.uber.com/horovod/
def J():
  ...

def f():
  ...

hvd.init()  # initialize Horovod
config = tf.ConfigProto()  # pin GPUs to processes
config.gpu_options.visible_device_list = str(hvd.local_rank())  # assign chief worker
config.gpu_options.allow_growth = True  # enable GPU
sess = tf.Session(config=config)  # Configure tensorflow
if hvd.rank() == 0:
  ... # assign a piece of data to chief worker
else:
  while hvd.rank() < hvd.size():
    ... # assign a piece of data to each worker

def build_model():
  # initialize parameters using Xavier initialization
  # parametrize the function J using J()
  # build loss function using net_f()
  # set up tensorflow optimizer in the variable name opt
  optimizer = hvd.DistributedOptimizer(opt)
  # minimize loss
  # initialize Tensorflow session
  bcast = hvd.broadcast_global_variables(0)  # Broadcast parameters to all workers
  sess.run(bcast)
  # train the deep learning model
Interactive mode

Sinteract -q gpu -p gpu -g gpu -m 12G -t 10:00:00
virtualenv --system-site-packages venv-for-tf
source ./venv-for-tf/bin/activate
pip install --user --no-cache-dir tensorflow-gpu==2.7.0

ipythonCores: 1
Tasks: 1
Time: 10:00:00
Memory: 128G
Partition: gpu
Account: sfi-pcd
Jobname: interact
Resource: gpu
QOS: gpu
salloc: job 124415 allocated