

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

- Goutham Gopalakrishna. ALIENs and Continuous Time Economies. 2021. SSRN Working paper.
- Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. 2018a. Journal of Computational Physics.
- Victor Duarte. Machine Learning for Continuous-Time Economics. 2017. SSRN Working paper.
- Jesús Fernández-Villaverde, Samuel Hurtado, and Galo Nuno. Financial Frictions and the Wealth Distribution. 2022. Econometrica (forthcoming).

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)

- 1 Portfolio Choice: [Merton \(1971\)](#), [Cochrane et al \(2008\)](#), [Martin \(2013\)](#)
- 2 Macro-Finance: [He and Krishnamurthy \(2013\)](#), [Brunnermeier and Sannikov \(2014\)](#), [Di Tella \(2017\)](#)
- 3 Monetary Theory: [Silva \(2020\)](#), [Brunnermeier and Sannikov \(2016\)](#)

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)

- 1 Portfolio Choice: [Merton \(1971\)](#), [Cochrane et al \(2008\)](#), [Martin \(2013\)](#)
- 2 Macro-Finance: [He and Krishnamurthy \(2013\)](#), [Brunnermeier and Sannikov \(2014\)](#), [Di Tella \(2017\)](#)
- 3 Monetary Theory: [Silva \(2020\)](#), [Brunnermeier and Sannikov \(2016\)](#)

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)

- 1 Portfolio Choice: [Merton \(1971\)](#), [Cochrane et al \(2008\)](#), [Martin \(2013\)](#)
- 2 Macro-Finance: [He and Krishnamurthy \(2013\)](#), [Brunnermeier and Sannikov \(2014\)](#), [Di Tella \(2017\)](#)
- 3 Monetary Theory: [Silva \(2020\)](#), [Brunnermeier and Sannikov \(2016\)](#)

General setup



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

General setup



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

- Exogenous dividend process of risky asset

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

General setup



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

- Exogenous dividend process of risky asset

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

- There is also a risk free debt market (pays return r). Risky asset has price of risk ζ_t , and volatility σ_t^R
- Problem of the agent is

$$\sup_{\hat{c}, \theta} U_t \quad (3)$$

$$\text{s.t. } \frac{dw_t}{w_t} = (r + \underbrace{\theta_t}_{\text{port. choice}} \underbrace{\zeta_t}_{\text{price of risk}} - \hat{c}_t)dt + \theta_t \underbrace{\sigma_t^R}_{\text{ret. volatility}} dZ_t \quad (4)$$

- If g, σ, r are time varying, then we have a multi-dimensional problem

HJB

- HJB is

$$\sup_{\hat{c}_t, \theta_t} f(c_t, U_t) + E_t(dU_t) = 0$$

- Conjecturing $U = \frac{J w^{1-\gamma}}{1-\gamma}$, where J is the stochastic opportunity process and γ is the risk aversion, the HJB equation reduces to

$$\mu^J(\mathbf{x}, J)J = \sum_{i=1}^d \mu^{x_i}(\mathbf{x}, J) \frac{\partial J}{\partial x_i} + \sum_{i,j=1}^d b^{i,j}(\mathbf{x}, J) \frac{\partial^2 J}{\partial x_i \partial x_j} \quad (5)$$

- 1 State variables are \mathbf{x} . Could be high-dimensional (large d)
- 2 μ^J , μ^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 \rightarrow slowly introduce non-linearity through

$$\mu^J(\mathbf{x}, J^{old})J = \frac{\partial J}{\partial t} + \sum_{i=1}^d \mu^{x_i}(\mathbf{x}, J^{old}) \frac{\partial J}{\partial x_i} + \sum_{i,j=1}^d b^{i,j}(\mathbf{x}, J^{old}) \frac{\partial^2 J}{\partial x_i \partial x_j} \quad (6)$$

- 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)

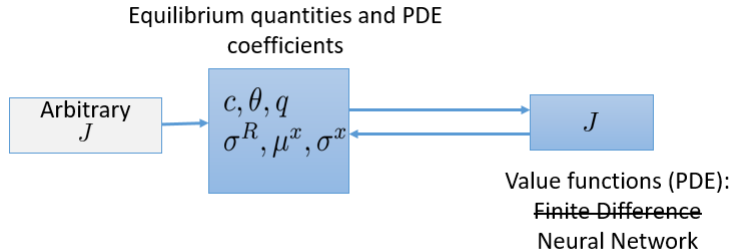


Figure: Overview of methodology.

Neural network solution method

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

where \hat{J}_Θ is a neural network object with parameters Θ , and f is the PDE residual.

Neural network solution method

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

where \hat{J}_Θ is a neural network object with parameters Θ , and f_Θ is the PDE residual. Can be seen as a classical constrained optimization problem

Optimization

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \quad \hat{J}_\Theta - \tilde{J}_0$$
$$\text{s.t.} \quad f = 0$$

Neural network solution method

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

Can be seen as an classical constrained optimization problem

Optimization

$$\Theta^* = \underset{\Theta}{\operatorname{argmin}} \quad \hat{J}_{\Theta} - \tilde{J}_0$$
$$\text{s.t.} \quad \int_t \int_{\mathbf{x}} |f|^2 dt d\mathbf{x} = 0$$

Neural network solution method

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

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

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

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

- Mesh free since we can randomly sample from the state space (t, \mathbf{x}) to train the neural network

Neural network solution method

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

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

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

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

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

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

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

- The subdomain Ω_c is found by inspecting the PDE coefficients which are determined using previous value \tilde{J}

Neural network solution method

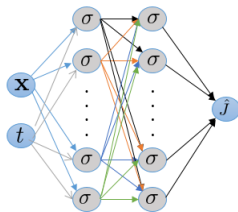
$$f := \frac{\partial \hat{J}(\mathbf{x}|\Theta)}{\partial t} + \sum_i^d \mu^i(\mathbf{x}) \frac{\partial \hat{J}(\mathbf{x}|\Theta)}{\partial x_i} + \sum_{i,j=1}^d b^{i,j}(\mathbf{x}) \frac{\partial^2 \hat{J}(\mathbf{x}|\Theta)}{\partial x_i \partial x_j} - \mu^J \hat{J}(\mathbf{x}|\Theta) = 0; \quad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

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

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

$$\hat{J}(\mathbf{x}|\Theta) = \tilde{J}_0; \quad \forall (t, \mathbf{x}) \in (T - (k-1)\Delta t) \times \Omega_c); \rightarrow \text{Active learning}$$

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



$\mathbf{X} \in \mathbb{R}^d$ Space dimension

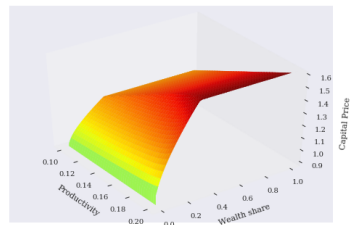
$t \in [0, T]$ Time dimension

σ Tanh activation function. $\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$\hat{J}(x | \Theta)$ Output from neural network

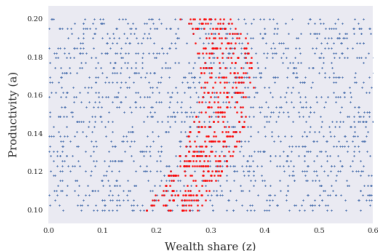
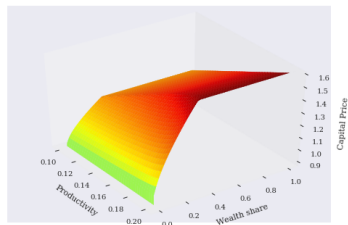
Active learning

Example from [Gopalakrishna \(2021\)](#): Macro-finance model with 2 state variables (productivity, wealth share)



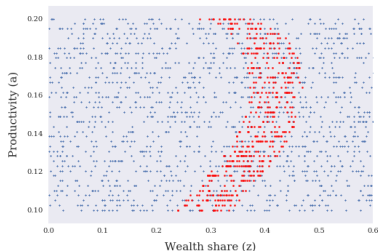
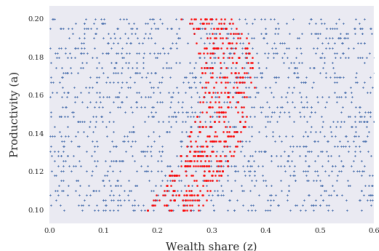
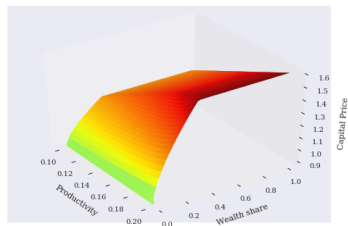
Active learning

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

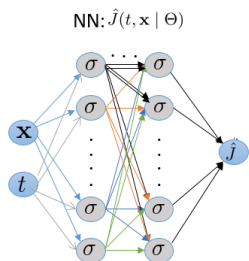


Figure: Methodology.

Solution technique: ALIENs

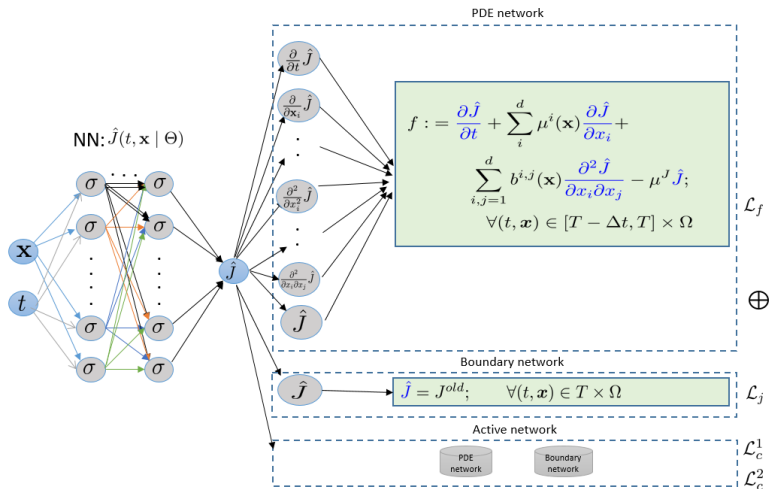


Figure: Methodology.

Solution technique: ALIENs

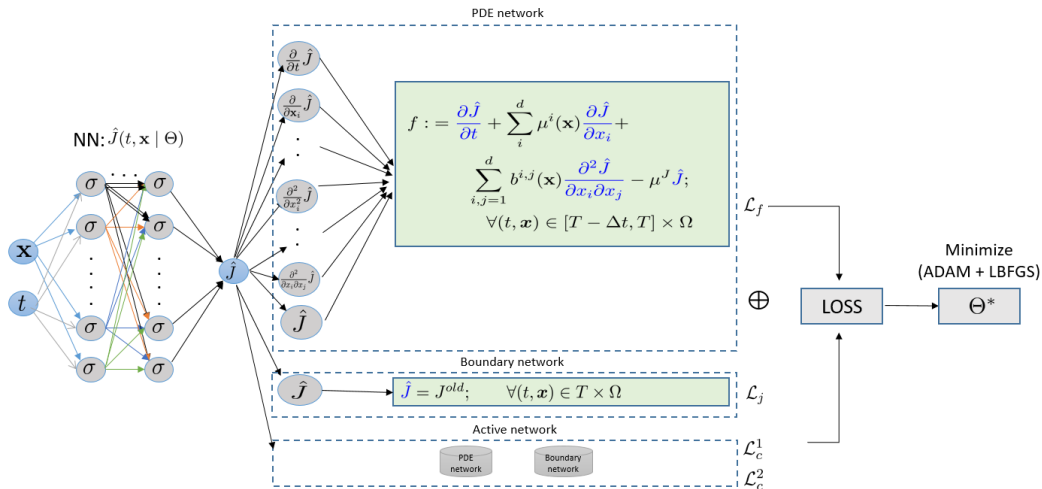


Figure: Methodology.

$$\mathcal{L} = \lambda_f \mathcal{L}_f + \lambda_j \mathcal{L}_j + \lambda_b \mathcal{L}_b + \lambda_c^1 \mathcal{L}_c^1 + \lambda_c^2 \mathcal{L}_c^2 \quad (7)$$

where

$$\text{PDE loss} \quad \mathcal{L}_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(\mathbf{x}_f^i, t_f^i)|^2$$

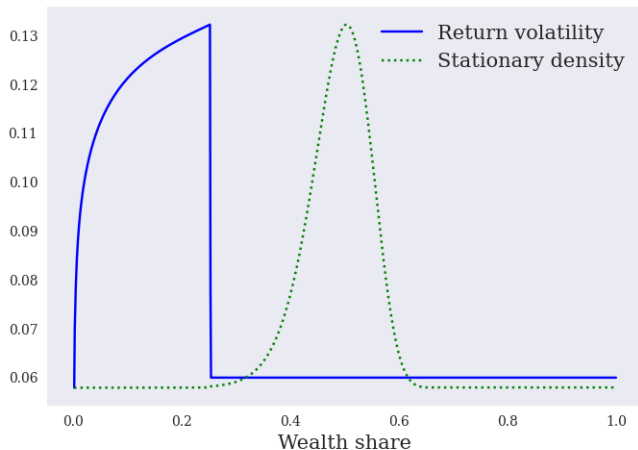
$$\text{Bounding loss-1} \quad \mathcal{L}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} |\hat{J}(\mathbf{x}_j^i, t_j^i) - \tilde{J}_0|^2$$

$$\text{Active loss-1} \quad \mathcal{L}_c^2 = \frac{1}{N_c} \sum_{i=1}^{N_c} |f(\mathbf{x}_c^i, t_c^i)|^2$$

$$\text{Active loss-2} \quad \mathcal{L}_c^1 = \frac{1}{N_c} \sum_{i=1}^{N_c} |\hat{J}(\mathbf{x}_c^i, t_c^i) - \tilde{J}_0|^2$$

Active Learning vs Simulation method

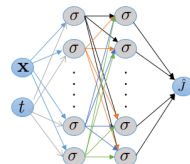
- ALIENs actively learn the region of sharp transition and samples more points → faster convergence
- Sampling procedure is complementary to simulation based methods ([Azinovic et al \(2018\)](#), [Villaverde et al \(2020\)](#)), but also works for models with **rare events** and financial constraints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



Automatic differentiation in practice

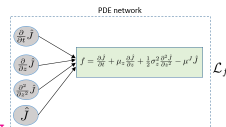
Approximating J using a neural network

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



Constructing regularizer: 1D model

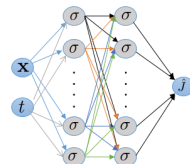
```
def f(z,t):  
    J = J(z,t)  
    J_t = tf.gradients(J,t)[0]  
    J_z = tf.gradients(J,z)[0]  
    J_zz = tf.gradients(J_z,z)[0]  
    f = J_t + advection * J_z + diffusion * J_zz - linearTerm * J  
    return f
```



Automatic differentiation in practice

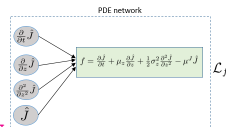
Approximating J using a neural network

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

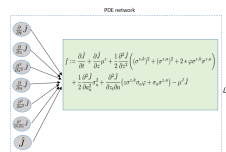


Constructing regularizer: 1D model

```
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
```



```
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)

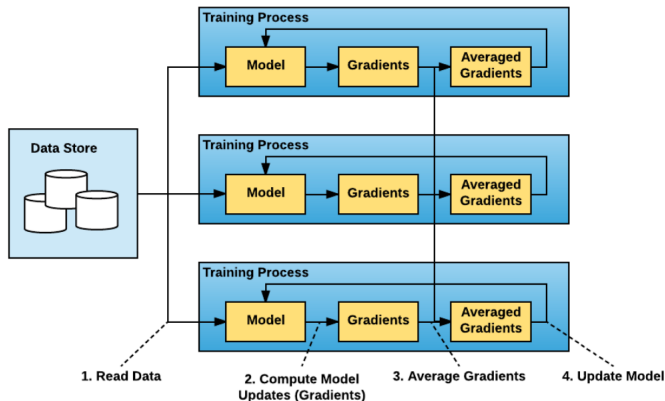


Figure: Source: <https://eng.uber.com/horovod/>

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
```