

# DSCI 565: BUILDER'S GUIDE

*This content is protected and may not  
be shared, uploaded, or distributed.*

Ke-Thia Yao  
Lecture 7: 2025-09-17

# Builder's Guide

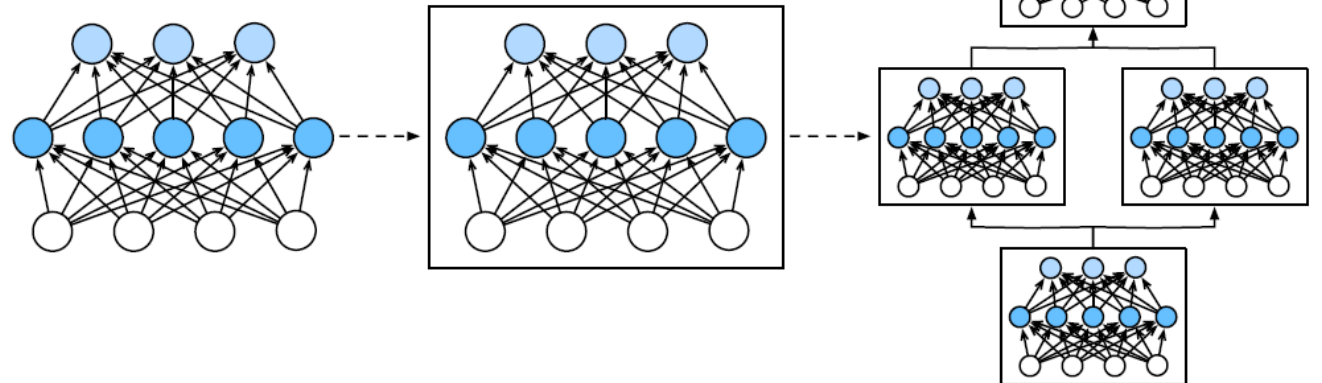
2



# Layers and Modules

3

- So far, we have seen networks with one neuron, one layer and multiple-layers. Each network
  - ▣ Takes inputs
  - ▣ Generates outputs
  - ▣ Described by a set of tunable parameters
- They all fall under the neural network **module** abstraction



# Define a Custom Module

4

- Inherit from PyTorch's `nn.Module`
- Implement `forward` function
  - ▣ Ingest input data
  - ▣ Generate output data
  - ▣ By default, gradient is computed automatically
- Store and provide access to model parameters needed by `forward`
- Initialize model parameters

# Notebook

5

- ❑ `chapter_builders-guide/model-construction.ipynb`
  - ❑ MLP module
  - ❑ MySequential module
    - `nn.Module.add_module(name, a_module)` adds a module
    - `nn.Module.children()` returns iterator over children
  - ❑ FixedHiddenMLP
    - Defining constant parameters that are not updated during optimization
    - E.g., `rand_weight` (not `nn.Parameter`)
  - ❑ NestMLP
    - Can mix and connect sequential module with regular layer

# Parameter Management

6

- Goal of training is to minimize the loss function by tuning model parameters
- Access and examine the parameters by calling
  - ▣ Function `state_dict()`
  - ▣ Function: `named_parameters()`
- Also, we can share parameters across multiple layers: *tied parameters*

Notebook: [chapter\\_builders-guide/parameters.ipynb](#)



# Parameter Initialization

7

- Instead of hardcoding parameter initialization in `Module.__init__()`
  - ▣ Create a separate initialization function
  - ▣ Then call `Module.apply()` with the initialization function
- Easier to try out different initialization approaches
- `Module.apply(initialization_func)`
  - ▣ Recursively applies `initialization_func` to its child modules
  - ▣ Then, apply `initialization_func` to itself

`chapter_builders-guide/init-param.ipynb`

# Custom Layers

8

- Like custom modules, custom layers also inherit from `nn.Module`
  - ▣ Layers without parameters
  - ▣ Layers with parameters
    - `nn.Parameter` defined in within `__init__()` are automatically added to the layer's parameters

See [chapter\\_builders-guide/custom-layer.ipynb](#)



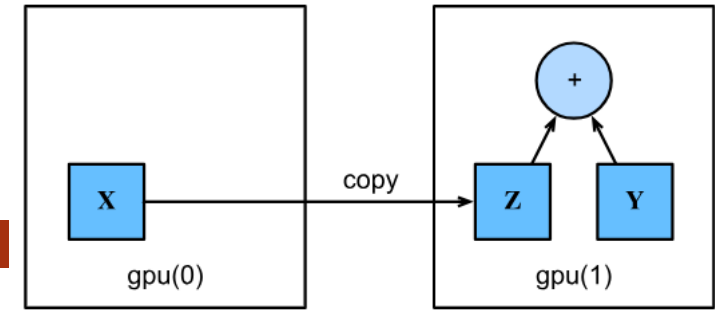
# Saving and Loading Networks

9

- [chapter\\_builders-guide/read-write.ipynb](#)

# Running on GPUs

10



- By default, tensors are allocated on the CPU
- Use the `device` keyword argument to allocate on GPU
  - ▣ E.g., `torch.Tensor([0, 1, 2], device=torch.device('cuda'))`
  - ▣ E.g., `torch.Tensor([0, 1, 2], device=torch.device('cuda:0'))`
- To copy to another device use
  - ▣ `Tensor.gpu(0)`, `Tensor.cpu()`, or `Tensor.to('cuda:1')`
- Tensors must be on the same device in order to perform operations
- Moving tensor across devices is SLOW!!!

# Running on GPUs

11

- To move a network to GPU
  - ▣ Do `net.to('gpu:0')`
- This command moves all the parameters of the network to `gpu:0`
  
- See `chapter_builders-guide/use-gpu.ipynb`