CS6216 Advanced Topics in Machine Learning (Systems)

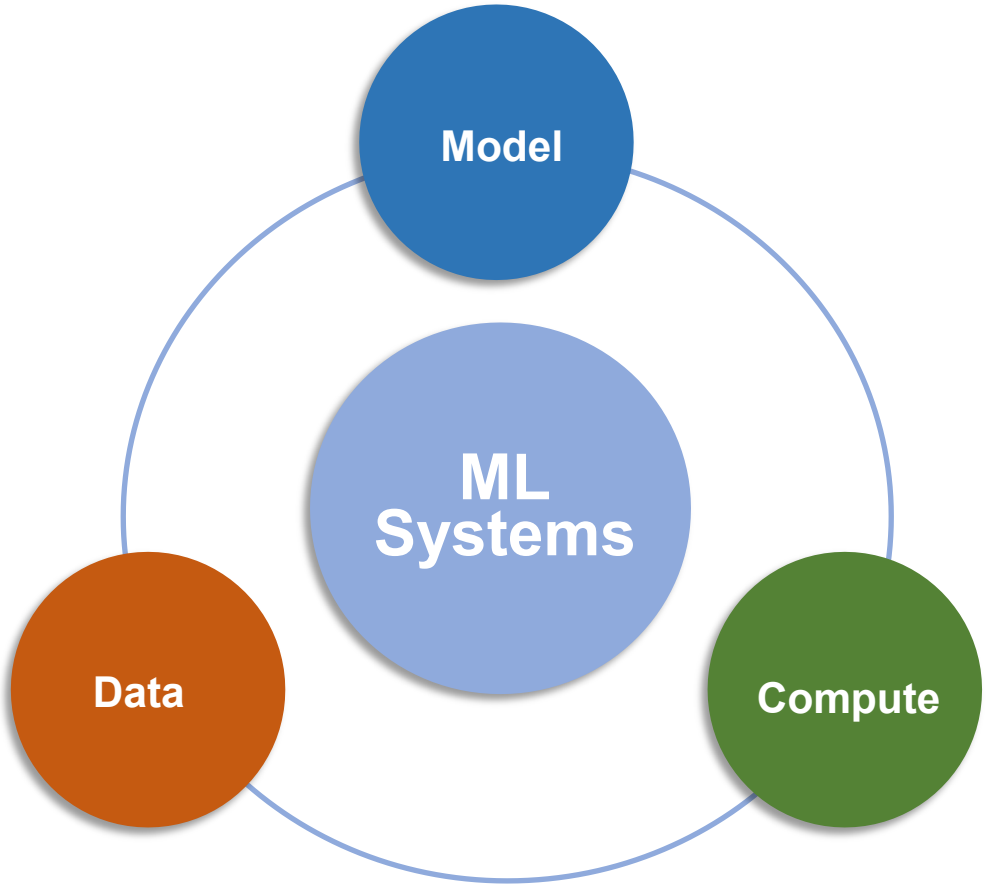
MLsys Foundations

Yao LU

20 Aug 2025

National University of Singapore
School of Computing

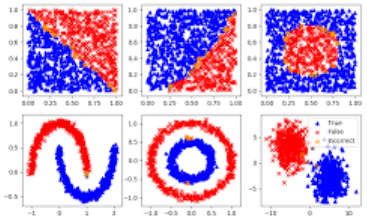
ML Systems Overview



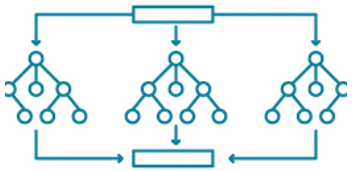
- Three components
- ML tasks
  - Training / tuning
  - Inference

Models

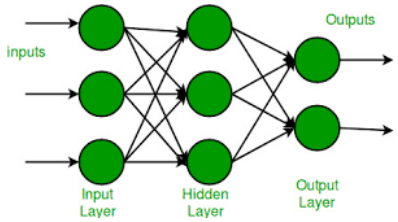
- What are models?



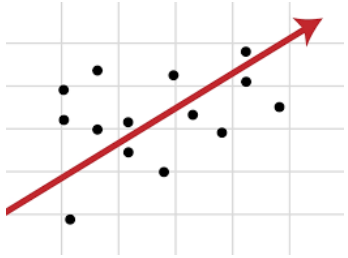
Clustering



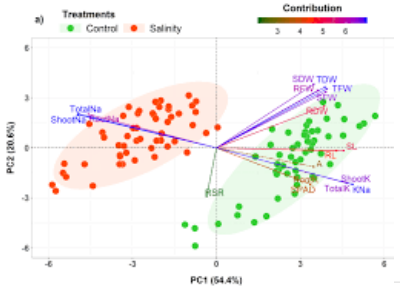
Random Forest



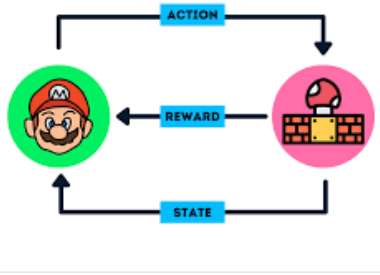
Perceptron



Linear regression



PCA



Reinforcement learning

- Models = algorithms?
- How to define, store & use models?

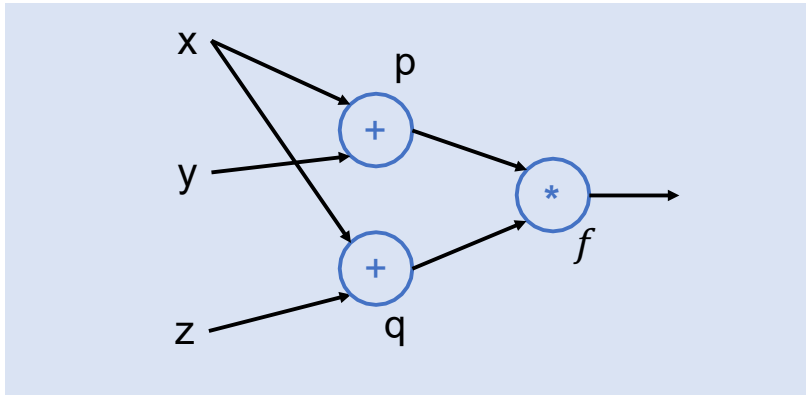
Model definitions

- PyTorch, Tensorflow, JAX etc. use functional declarations
  - Direct mapping to a compute graph, no ambiguity

```
class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

Model definition



(not matching code)

A variety of ML systems

- ML systems exist for Boosting trees, Graph neural networks etc.

- This lecture focuses on Large Generative Models (LGMs)
  - Deep neural networks trained w/ Stochastic Gradient Descent (SGD)

  - Post-training paradigms

A variety of ML systems

- ML systems exist for Boosting trees, Graph neural networks etc.

- This lecture focuses on Large Generative Models (LGMs)
  - Deep neural networks trained w/ Stochastic Gradient Descent (SGD)

  - Post-training paradigms

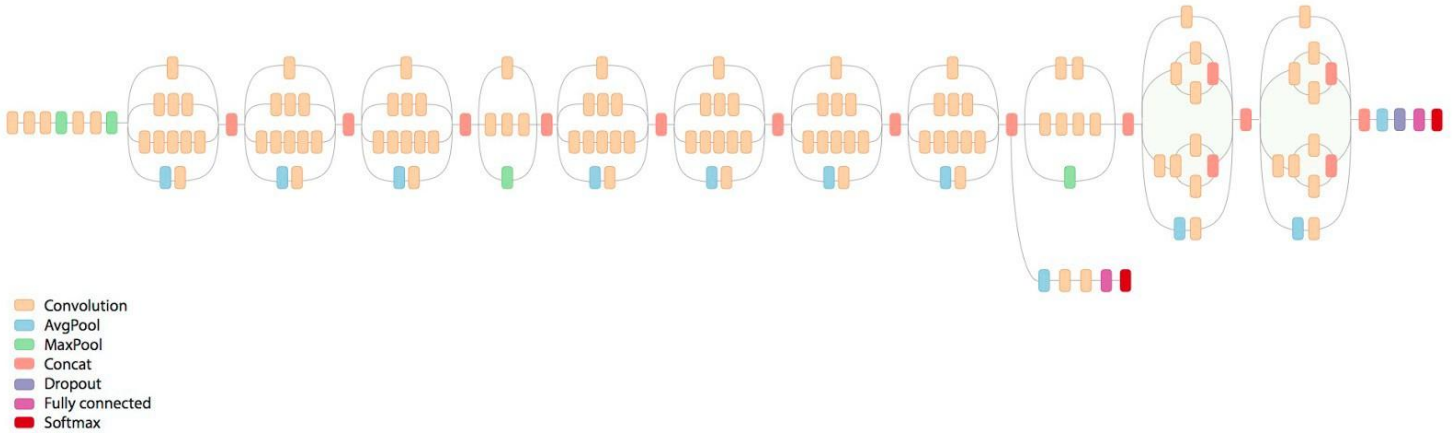
Algorithmic workflows: recap

Stochastic Gradient Descent (SGD)

Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights

Model inputs



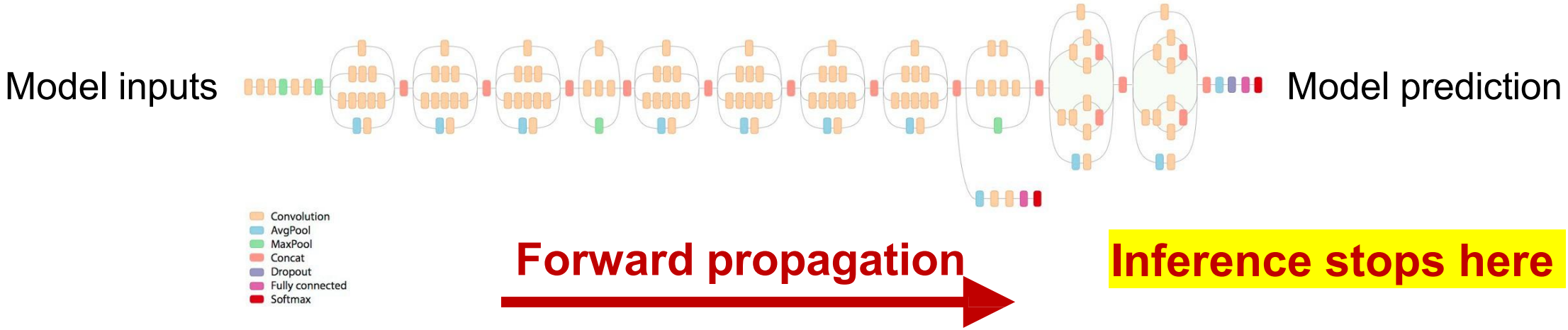
Model prediction

Algorithmic workflows: recap

Stochastic Gradient Descent (SGD)

Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights

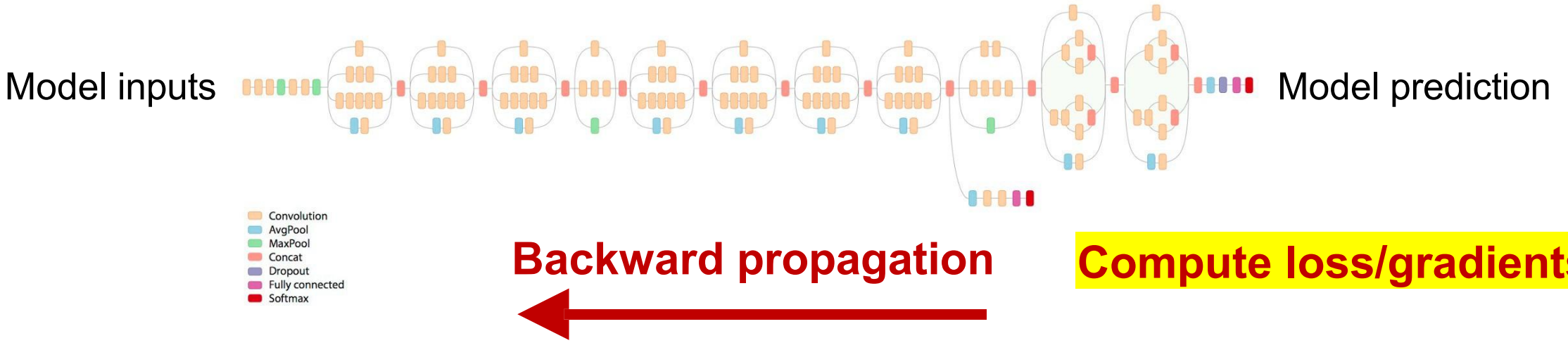


Algorithmic workflows: recap

Stochastic Gradient Descent (SGD)

Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights



Algorithmic workflows: recap

Stochastic Gradient Descent (SGD)

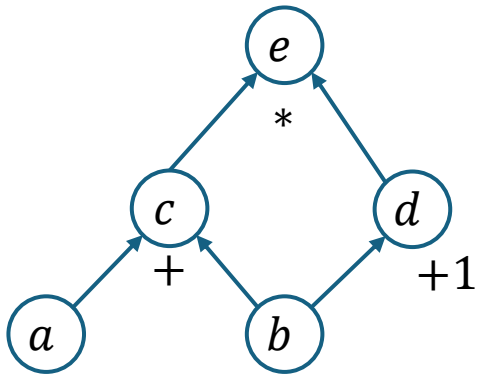
Train ML models through many iterations of 3 stages

1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights

$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$

Back propagation by example

- $e = (a + b) \cdot (b + 1)$, compute the following :



$$\frac{\partial e}{\partial c} =$$

$$\frac{\partial e}{\partial d} =$$

$$\frac{\partial e}{\partial a} =$$

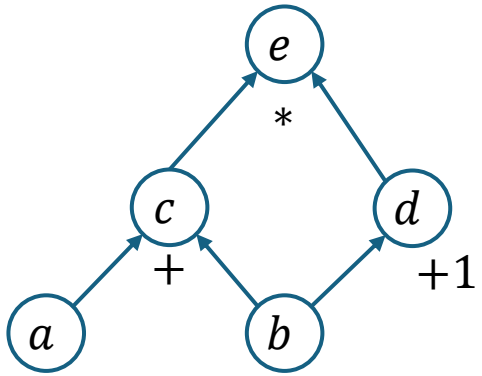
$$\frac{\partial e}{\partial b} =$$

Applying chain rule to compute gradient

- Back-tracking from the root to write down partial derivatives. + for branches, * for adjacent nodes

Back propagation by example

- $e = (a + b) \cdot (b + 1)$, compute the following :



$$\frac{\partial e}{\partial c} = \frac{\partial (c \cdot d)}{\partial c} = d$$

$$\frac{\partial e}{\partial d} = \frac{\partial (c \cdot d)}{\partial d} = c$$

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} = d \cdot 1 = d$$

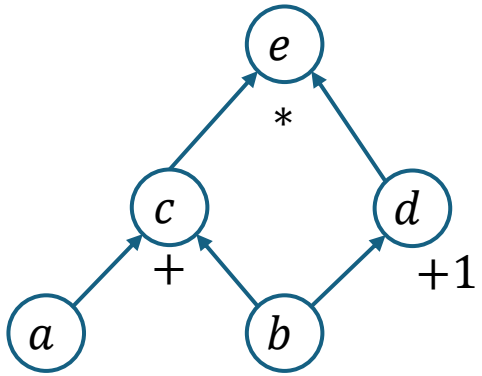
$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b} = c + d$$

Applying chain rule to compute gradient

- Back-tracking from the root to write down partial derivatives. + for branches, * for adjacent nodes

Back propagation by example

- $e = (a + b) \cdot (b + 1)$, compute the following :



$$\frac{\partial e}{\partial c} = \frac{\partial (c \cdot d)}{\partial c} = d$$

$$\frac{\partial e}{\partial d} = \frac{\partial (c \cdot d)}{\partial d} = c$$

$$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a} = d \cdot 1 = d$$

$$\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b} = c + d$$

Applying chain rule to compute gradient

- Back-tracking from the root to write down partial derivatives. + for branches, * for adjacent nodes
- Given the actual Loss, compute gradient digits

A lot of repetitive compute

- Proper caching & reusing in the graph nodes

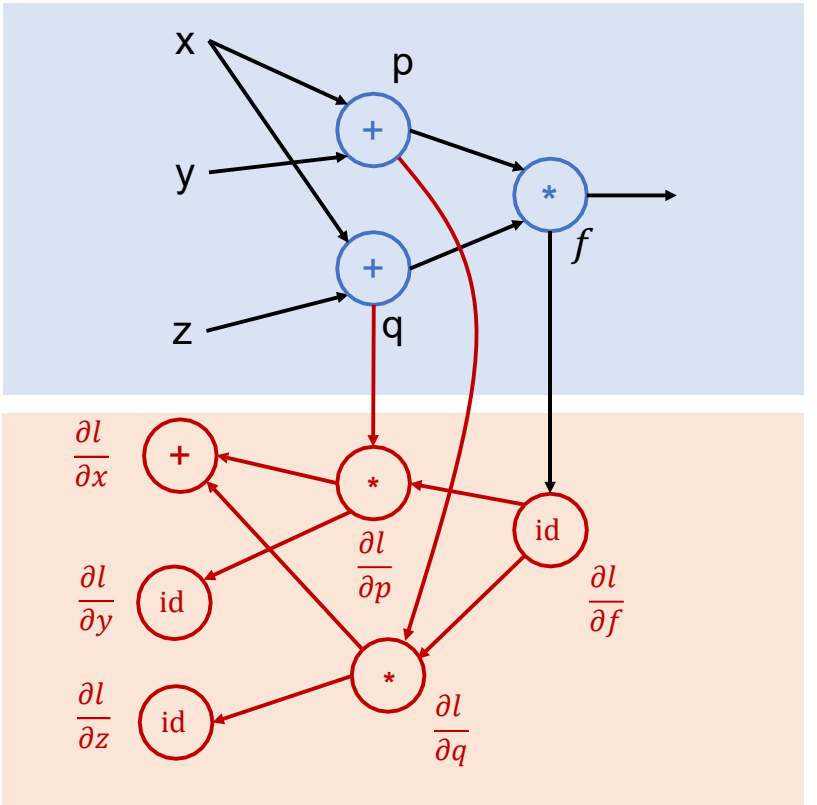
Building forward & backward compute graph

```python
class ToyModel(nn.Module):
    def __init__(self):
        super(ToyModel, self).__init__()
        self.net1 = torch.nn.Linear(10, 10).to('cuda:0')
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to('cuda:1')

    def forward(self, x):
        x = self.relu(self.net1(x.to('cuda:0')))
        return self.net2(x.to('cuda:1'))
```

Model definition

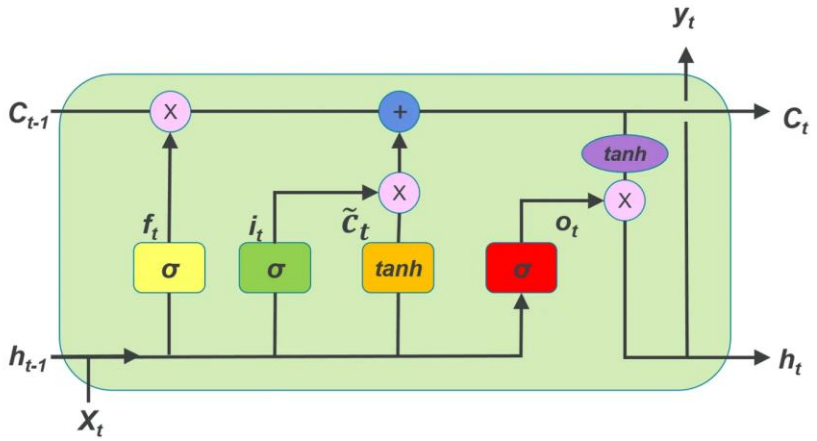
Compute graph builder



(not matching code)

Back propagation for LSTM

- Long-short term memory (LSTM)



$$f_t = \sigma_g\,(W_f \times x_t + U_f \times h_{t-1} + b_f)$$

$$i_t = \sigma_g\,(W_i \times x_t + U_i \times h_{t-1} + b_i)$$

$$o_t = \sigma_g\,(W_o \times x_t + U_o \times h_{t-1} + b_o)$$

$$c'_t = \sigma_c\,(W_c \times x_t + U_c \times h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

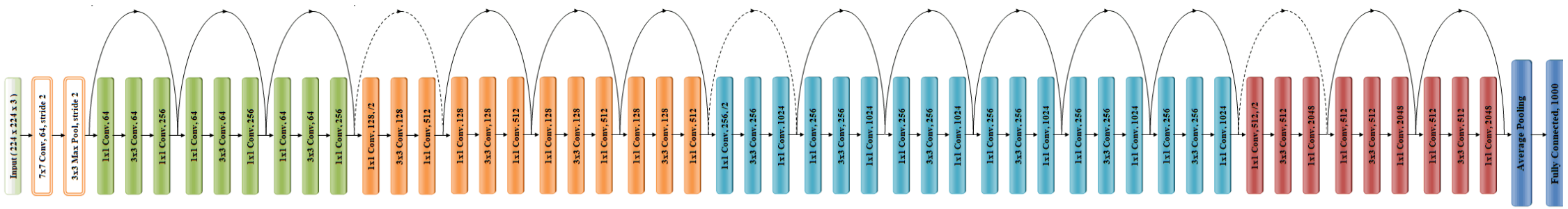
$$h_t = o_t \cdot \sigma_c(c_t)$$

- Derive the back-prop formulations for all parameters

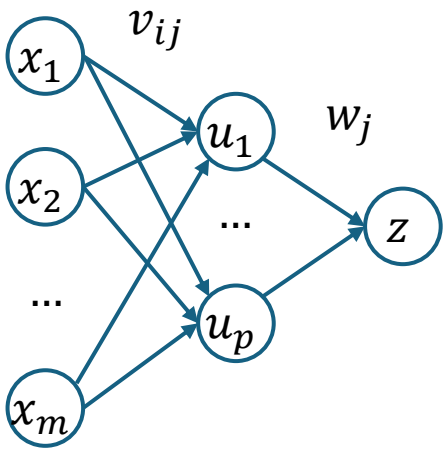
- Instructor's experience 10 years back:
  - 1 full page of equations, 30~40 steps
  - Implementing on GPU, extremely hard to debug

How about very large neural networks?

- We need
  - Automatic computation of gradients
  - Optimization with proper caching and compute node reuse



Quiz : back propagation for MLP



- MLP is a simple DNN, where a single perceptron is defined as:

$$y = \sigma(W \cdot x + b)$$

- A 2-layer perceptron for univariate regression with l_2 loss:

$$z = \sigma(W \cdot u + b)$$
$$u = \sigma(V \cdot x + b)$$
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

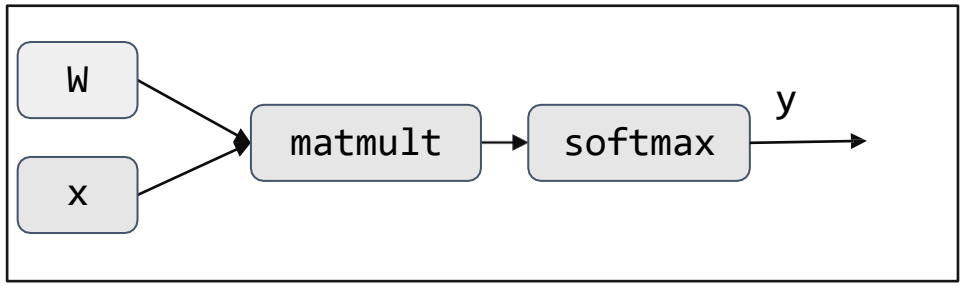
hint: $\sigma'(x) = \sigma(x)(1 - \sigma(x))$

derive gradients for W and V.

Computational graph construction by step

Construct the compute graph for $y = \text{softmax}(W \cdot x)$ with cross entropy loss

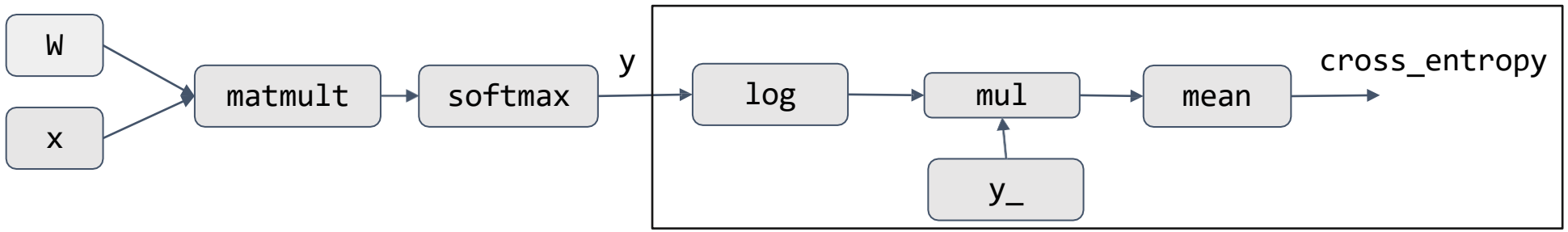
1. Construct forward graph



Computational graph construction by step

Construct the compute graph for $y = \text{softmax}(W \cdot x)$ with cross entropy loss

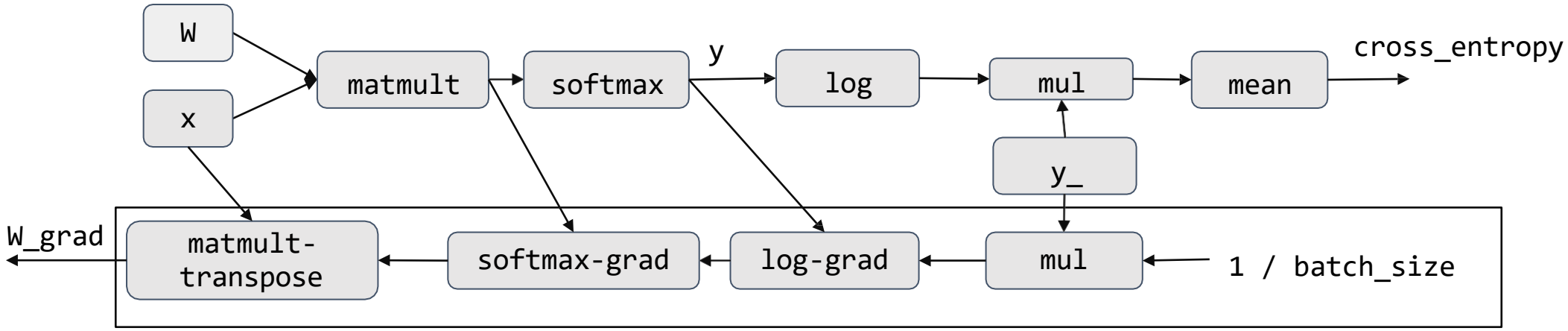
1. Construct forward graph
2. Add loss compute nodes



Computational graph construction by step

Construct the compute graph for $y = \mathrm{softmax}(W \cdot x)$ with cross entropy loss

1. Construct forward graph
2. Add loss compute nodes

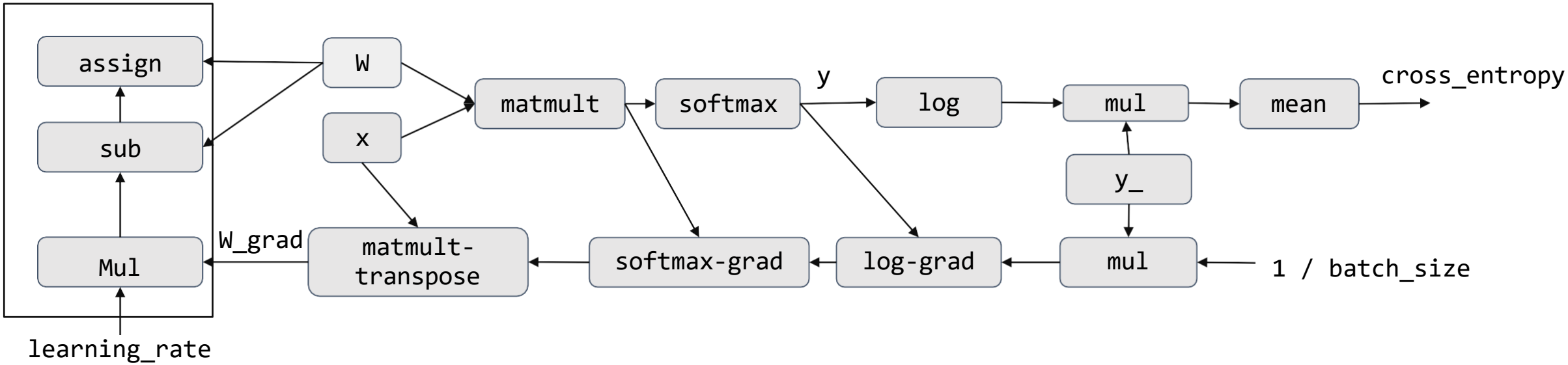


3. Construct backward graph by automatic differentiation More details in the next lecture

Computational graph construction by step

Construct the compute graph for $y = \mathrm{softmax}(W \cdot x)$ with cross entropy loss

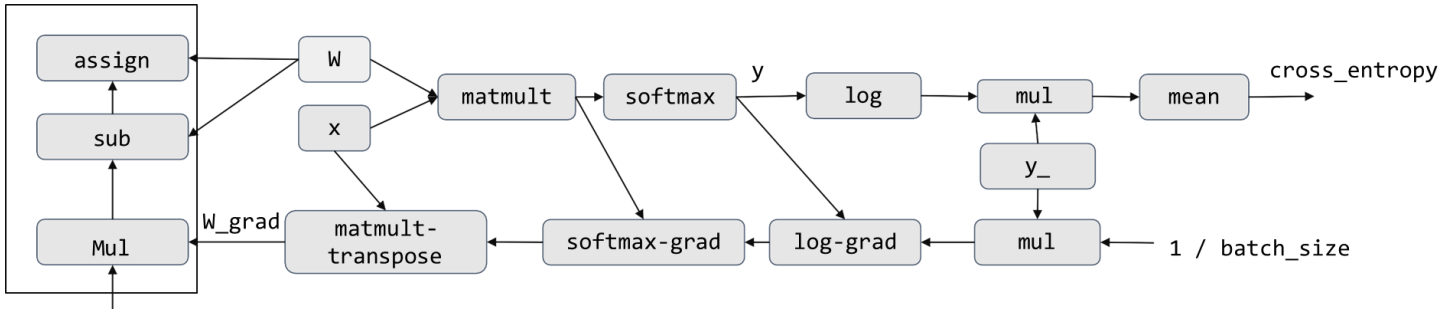
1. Construct forward graph
2. Add loss compute nodes



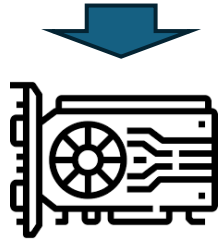
3. Construct backward graph by automatic differentiation
4. Update model weights

Mapping compute graph to actual runtime

- Key factors to consider:
  - Graph dependency
  - Parallelism & batching
  - Driver & API

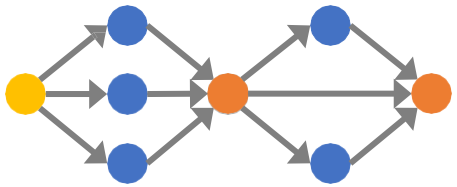


- CPU, GPU, TPU, FPGA, etc.
  - Each architecture has corresponding libraries and APIs



- Optimizations:
  - Operator code-gen and fusion
  - Graph-level optimizations

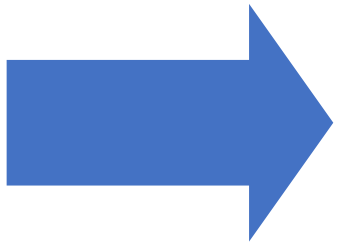
Execution of the compute graph: data parallelism

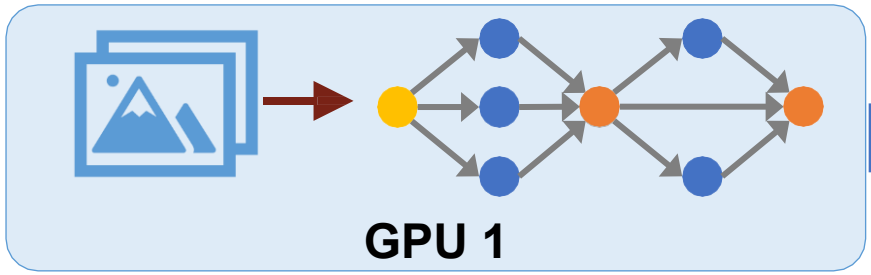


ML Model

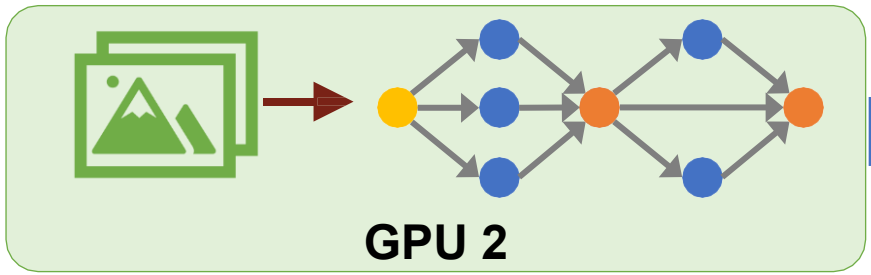
Training Dataset

$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$



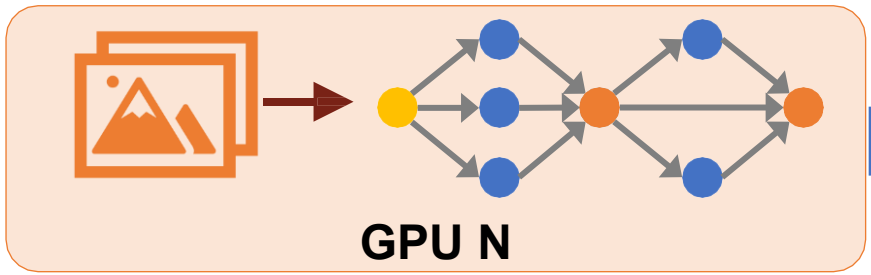


GPU 1

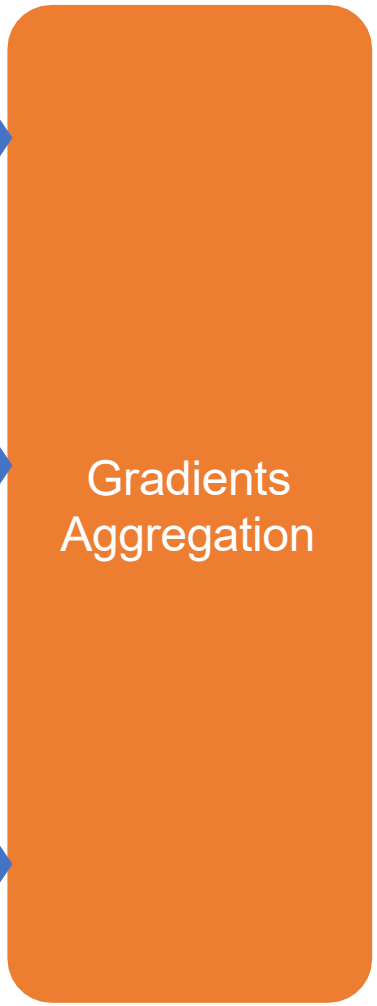


GPU 2

. . .



GPU N



Gradients Aggregation

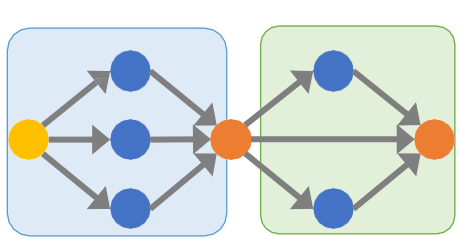
1. Partition training data into batches

2. Compute the gradients of each batch on a GPU

3. Aggregate gradients across GPUs

Execution of the compute graph: model parallelism

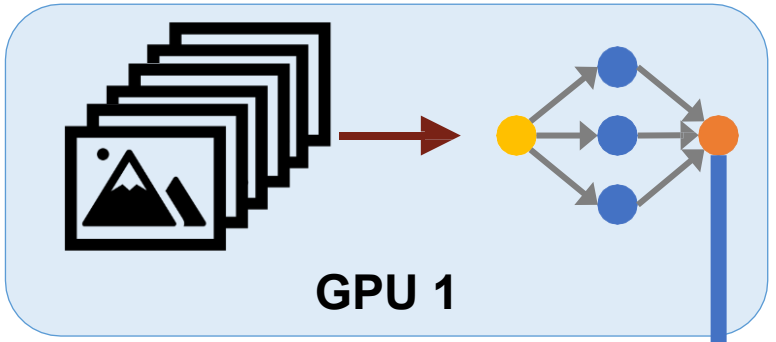
- Split a model into multiple subgraphs and assign them to different devices

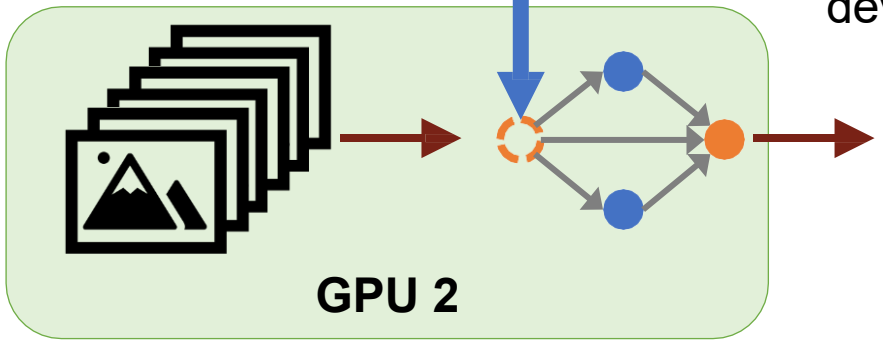


ML Model

Training Dataset

Model Parallelism



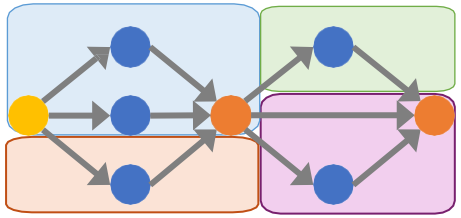


Transfer intermediate results between devices

$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$

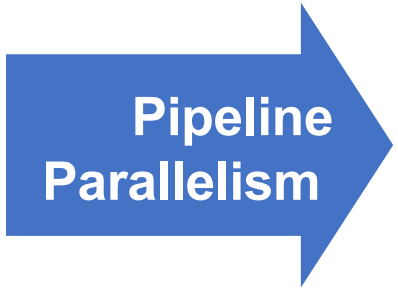
Execution of the compute graph: pipeline parallelism

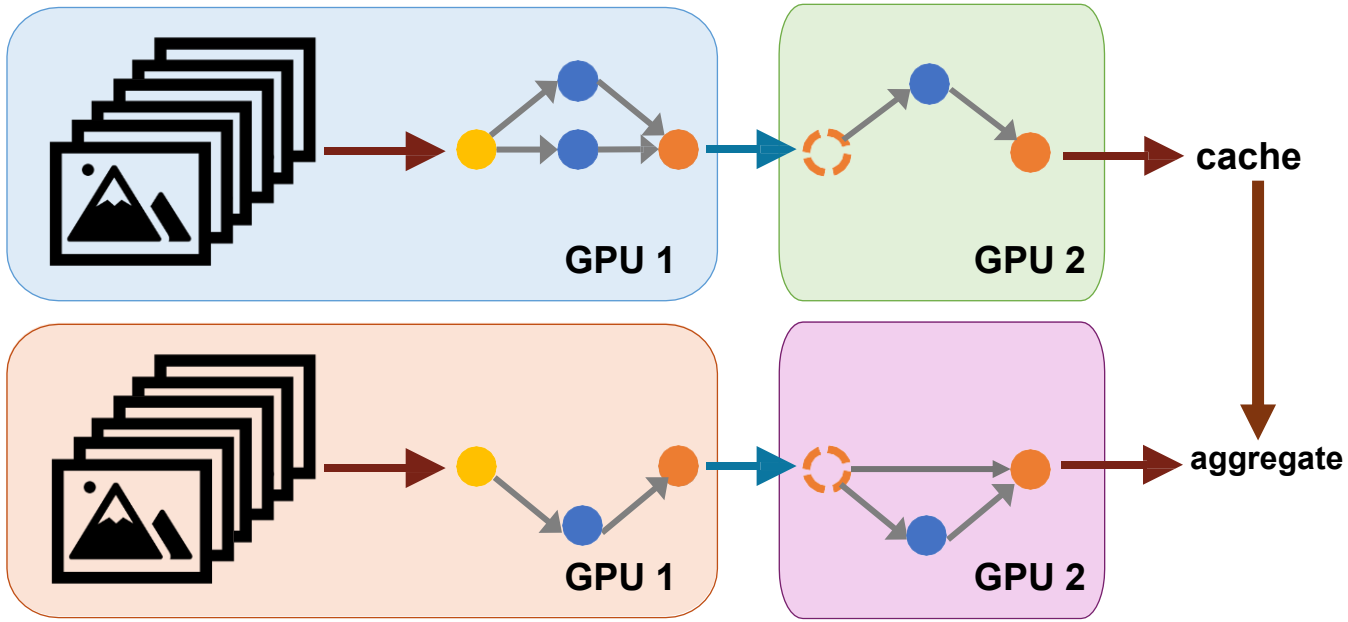
- Split a model into multiple subgraphs and assign them to different devices. Run them by proper scheduling.



ML Model

Training Dataset





$$w_i := w_i - \gamma \nabla L(w_i) = w_i - \frac{\gamma}{n} \sum_{j=1}^{n} \nabla L_j(w_i)$$

A variety of ML systems

- ML systems exist for Boosting trees, Graph neural networks etc.

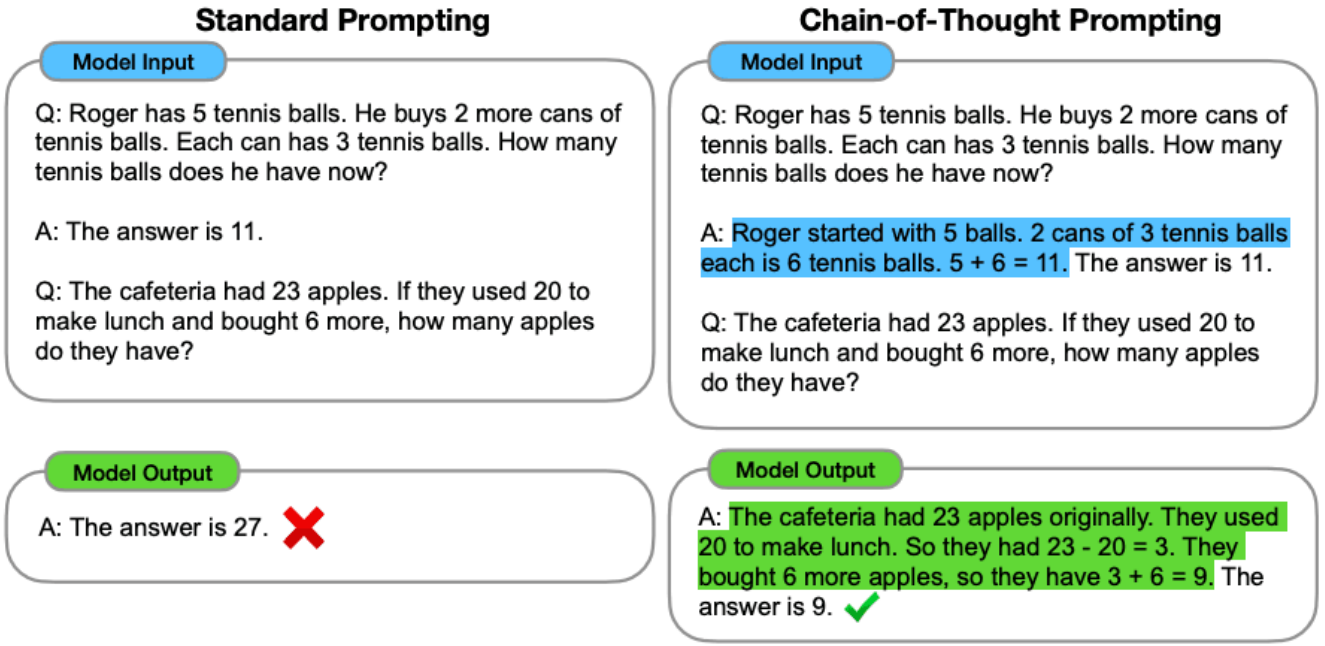
- This lecture focuses on Large Generative Models (LGMs)
  - Deep neural networks trained w/ Stochastic Gradient Descent (SGD)

  - Post-training paradigms

Post-training paradigms

- A large trend to move from pre-training → post-training
  - To *greatly* save costs while preserving model abilities
  - Reasoning ↑↑↑

- Labeling: text → CoT

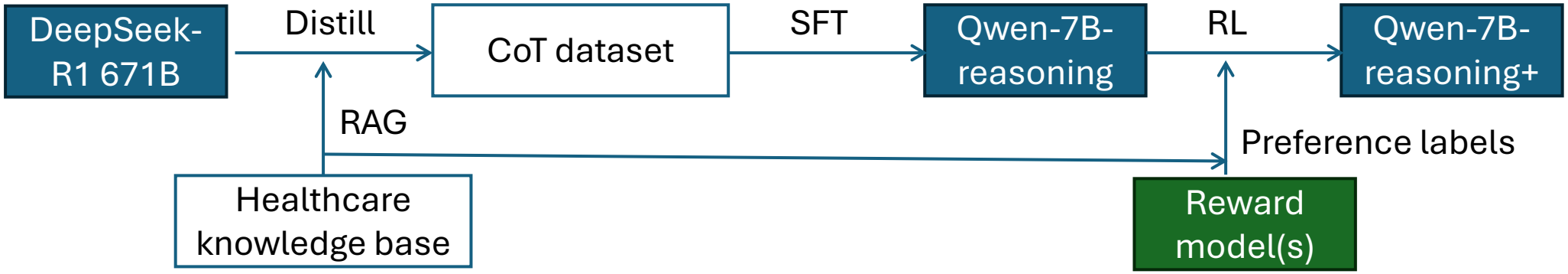


Post-training paradigms

- A large trend to move from pre-training → post-training
  - To *greatly* save costs while preserving model abilities
  - Reasoning ↑↑↑
- Labeling: text → CoT
- Compute: modularized training/inference
  - Distillation, reward models, SFT, RL
  - Agents, RAGs, workflows

Compute graph (again) on a coarse granularity

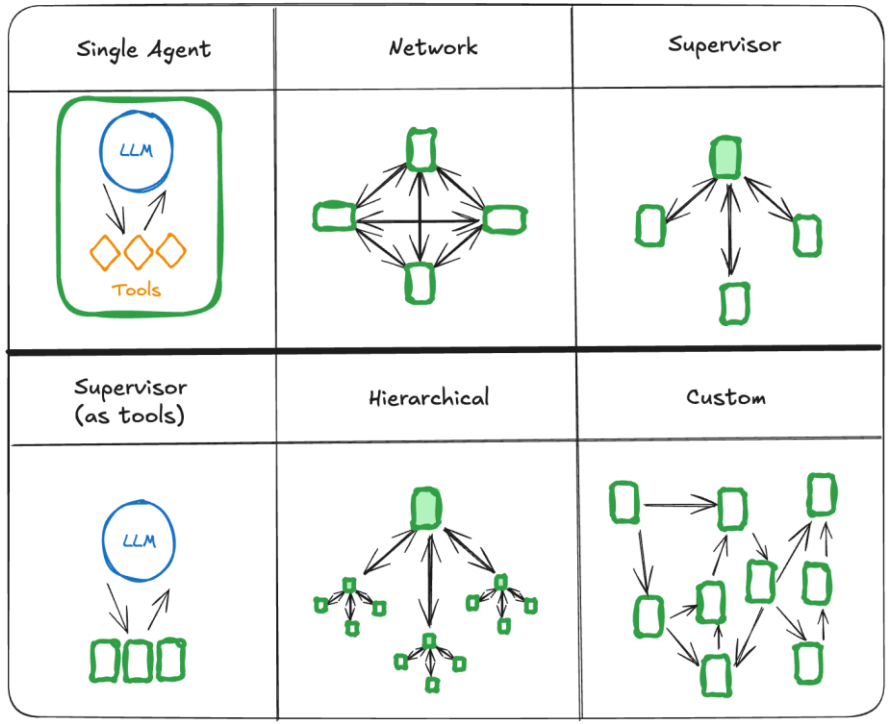
- Example task: give me a 7B reasoning model specialized in healthcare
  - DeepSeek-R1 671B too big & expensive
  - I don't care about astrophysics or Shakespeare



- System designs & optimizations : more details later in this course.

Compute graph (again) on a coarse granularity

- Example task: solving a task using a multi-agent framework
  - Collaborative, reflections, multi-round invocations of LLMs



- System designs & optimizations : more details later in this course.

Course structure

Lecture date	Plan	Lecturer if not Yao	Note
Aug 13	Week 1: **Introduction** [slides]		[HW1 Release]
Aug 20	Week 2: **MLsys foundations**		
Aug 27	Week 3: **Automatic differentiation**		HW1 due
Sep 03	Week 4: **Hardware acceleration**		[HW2 Release]
Sep 10	Week 5: **Parallelism and training techniques**		
Sep 17	Week 6: **Transformers, Attention and Optimizations**		HW2 due, Project proposal due
Sep 24	Recess week		
Oct 01	Week 7: **Serving LLMs**		[HW3 Release]
Oct 08	Week 8: **Post-training techniques**		
Oct 15	Week 9: **Multi-Modal Models**		HW3 due, Mid-term project report due
Oct 22	Week 10: **Application Systems: AI Agents, RAGs, Deep Research and beyond**		[HW4 Release]
Oct 29	Week 11: **LLM Safety**		
Nov 05	Week 12: **Cloud systems for AI**		
Nov 12	Week 13: **Project presentations**		HW4 due, final project report due

Fine-granularity techniques & optimizations

Systems

Coarse-granularity techniques & optimizations

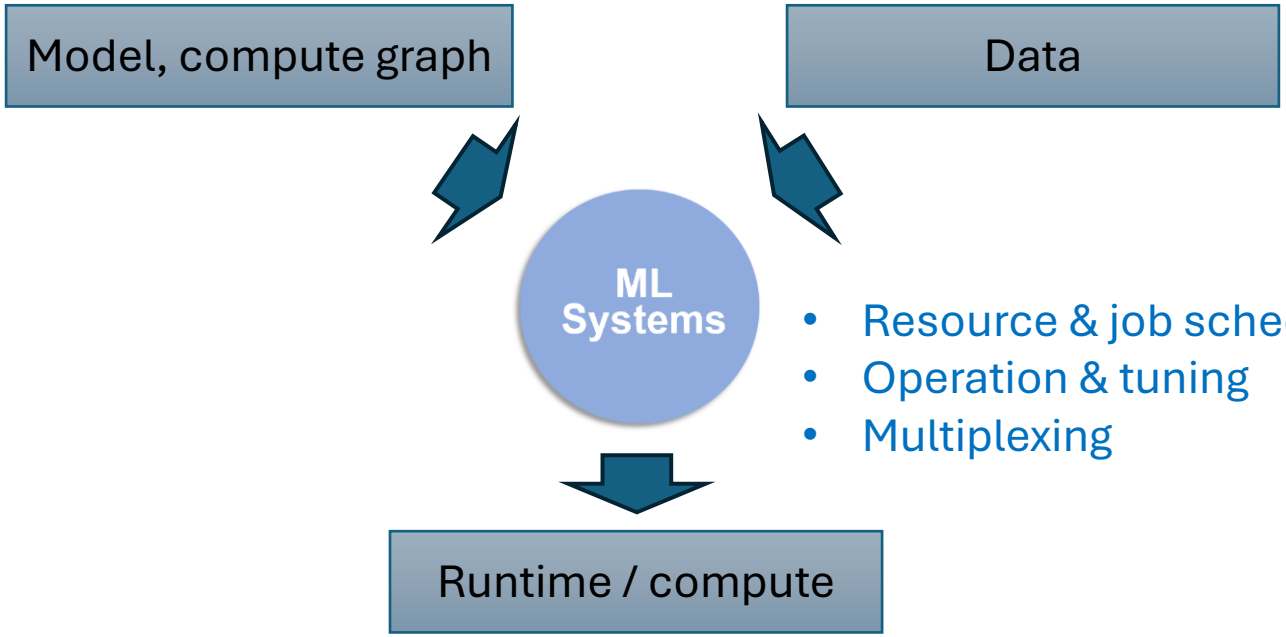
Misc

Systems

Summary: core modules in MLsys

- Graph optimization
- Model specific technologies

- Storage & caching
- Data preparation & quality



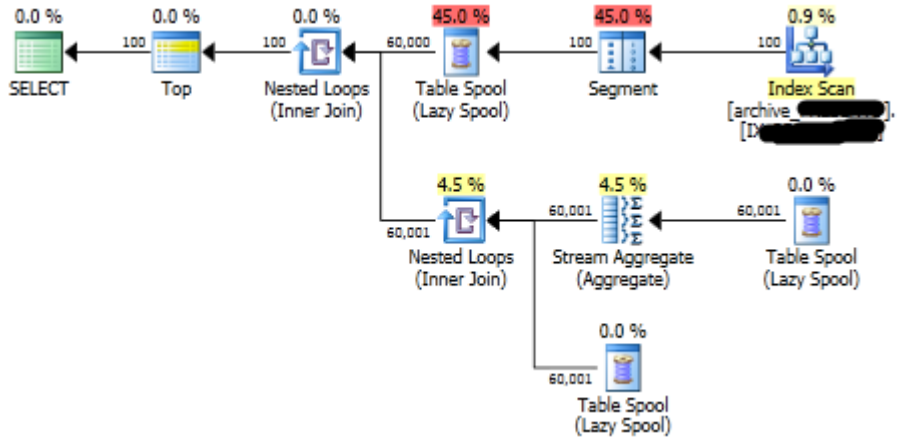
- R&D optimizes for
  - Training / Tuning:

    efficiency & scalability
  - Inference / Serving:

    latency & throughput
  - Cloud efficiency

Archaeology

A SQL query plan



- Data & ML systems share many common ideas
- Compute graph is an old thing
  - SQL query / execution plan
- Difference in programming model
  - Functional: high level language > intermedia representation > optimizer > execution plan
  - Declarative: same, but a much larger search space
  - Graph-based query optimization is an old thing as well
- Lots of idea can often reuse! Come to my other database course.

	Data systems	ML systems
Originate	1970s	2010s
Programming model	Declarative	Functional
Graph-based optimization	Operator fusion, reordering, ….	Operator fusion, reordering…
Parallelism	Data, pipeline	Data, tensor, pipeline

Logistics

- Homework 1 is out

- Overview of Homework 2-4
  - HW2: back propagation and autograd
  - HW3: framework & LLM inference
  - HW4: LLM serving & RAG

Reading for the next lecture

- How to read a paper
- TensorFlow: A System for Large-Scale Machine Learning
  OSDI 2016
- QA / interaction in class