Introduction to Deep Neural Networks

Zhihao Jia

Computer Science Department

Carnegie Mellon University

Administrative

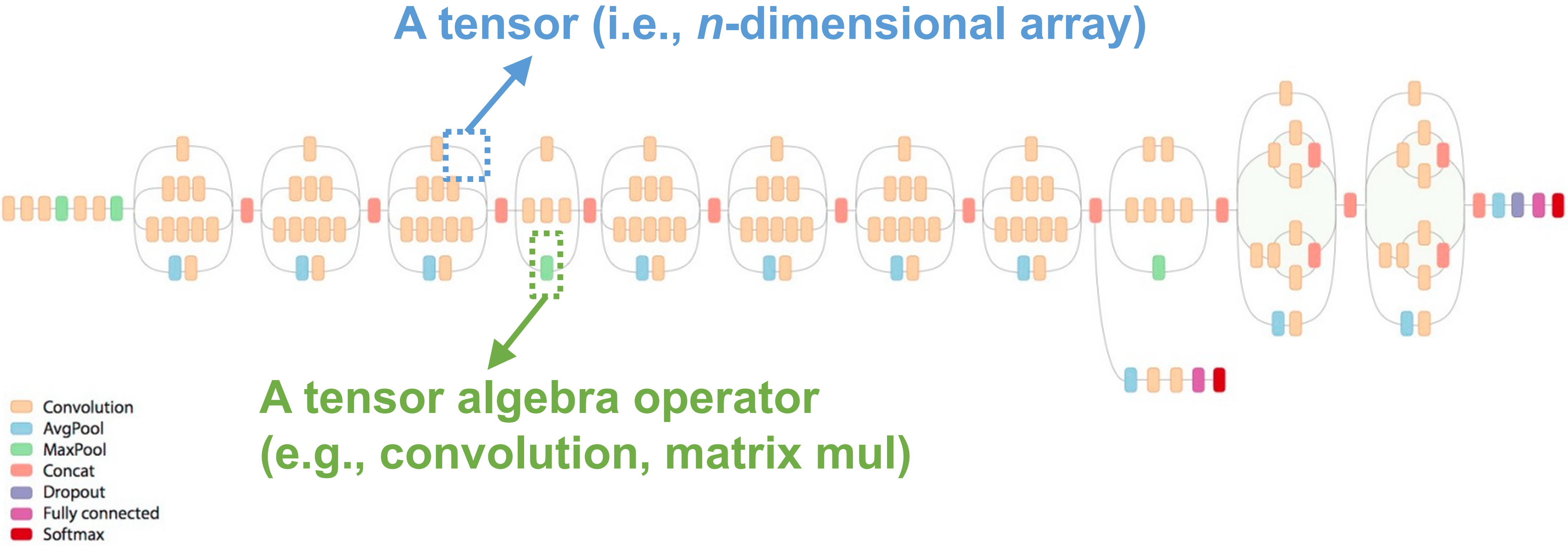
- Sign up for paper presentations (presentations start on week 3)
- Initial assignments will be **released tonight**
- First reading assignments **due next Monday before lecture**
- All lectures will be available on Canvas

Content

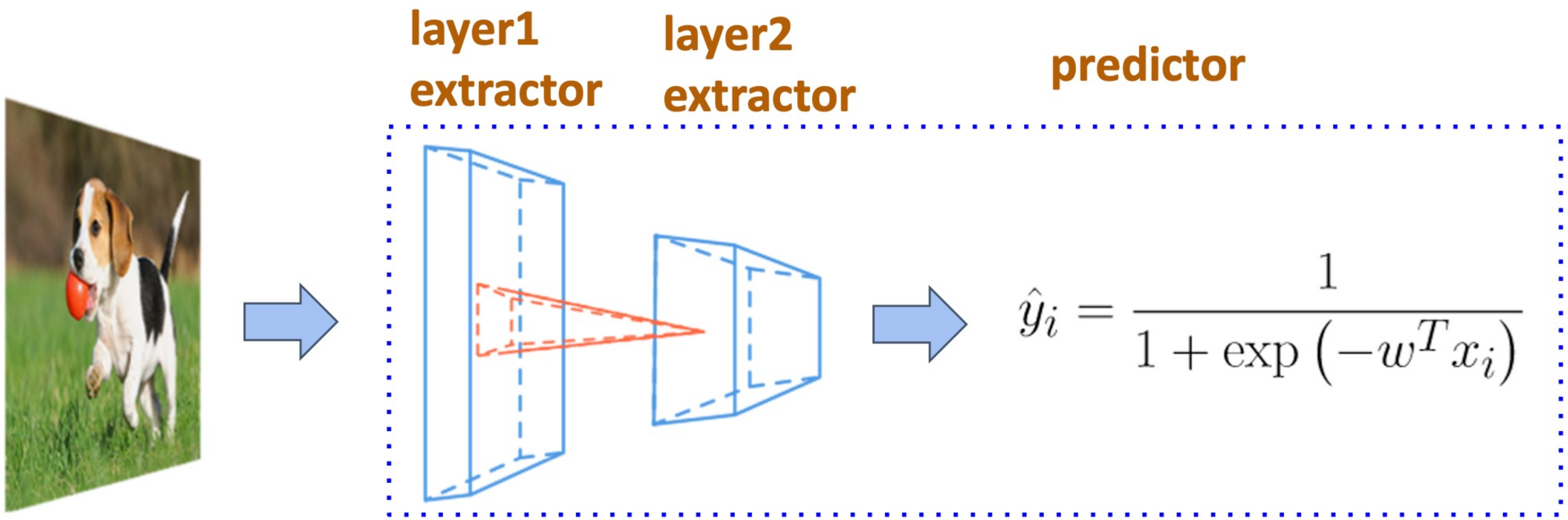
- Stochastic Gradient Descent
- Backpropagation and Automatic Differentiation
- Understand Our Applications: An Overview of Neural Networks

Recap: Deep Neural Network

- Collection of simple trainable mathematical units that work together to solve complicated tasks



DNN Training Overview



$$\hat{y}_i = \frac{1}{1 + \exp\left(-w^T x_i\right)}$$

Objective

$$L(w) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \|w\|^2$$

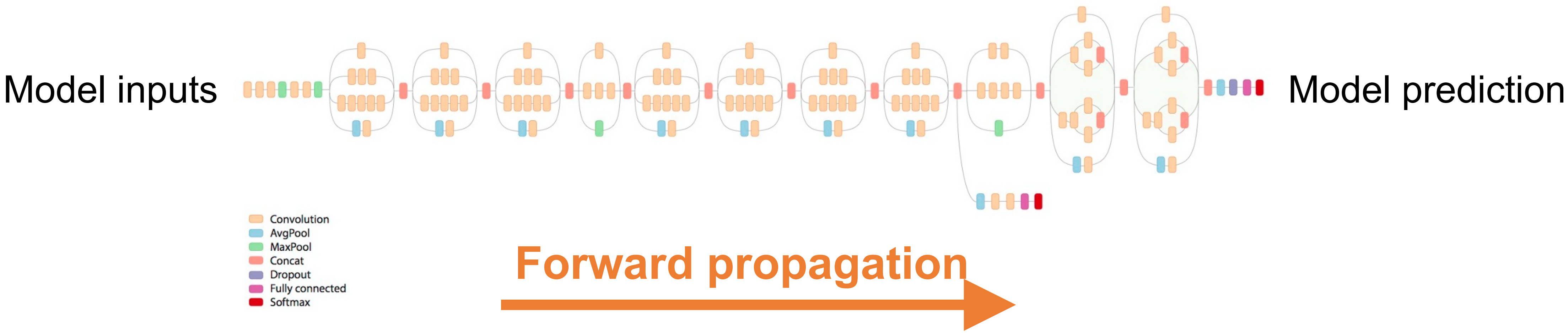
Training

$$w \leftarrow w - \eta \nabla_w L(w)$$

Gradient Descent (GD)

Train ML models through many iterations of 3 stages

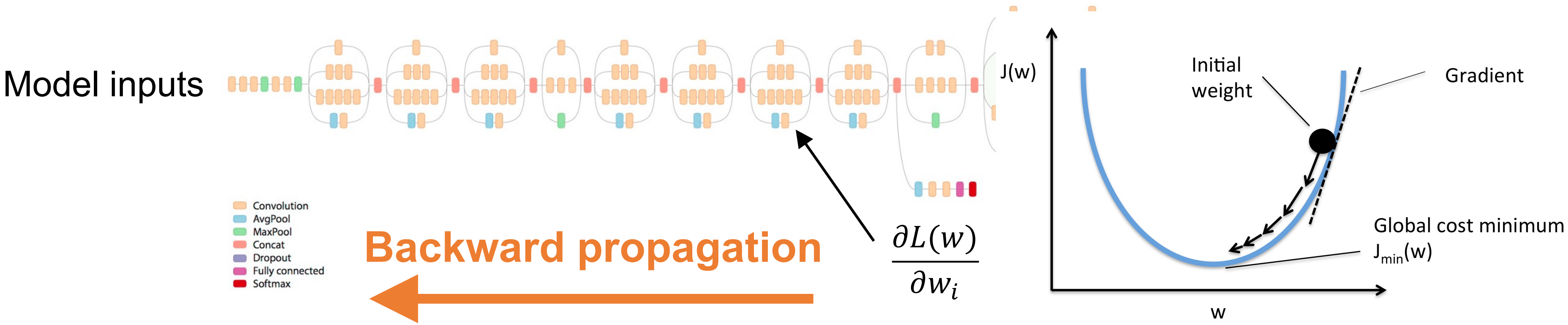
1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce error for each trainable weight
3. **Weight update**: use the loss value to update model weights



Gradient Descent (GD)

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 a gradient for each trainable weight
3. **Weight update**: use the loss value to update model weights



Gradient Descent (GD)

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 a gradient for each trainable weight
3. **Weight update**: use the gradients to update model weights

$$w_i := w_i - \gamma \frac{\partial L(w)}{\partial w_i} = w_i - \frac{\gamma}{N} \sum_{j=1}^{N} \boxed{\frac{\partial l_i(w)}{\partial w_i}}$$

Gradients of individual samples

Stochastic Gradient Descent (SGD)

- Inefficiency in gradient descent
  - Too expensive to compute gradients for all training samples
  - Especially for todays large-scale training datasets (e.g., ImageNet-22K with 14 million images)

- **Stochastic gradient descent**

$$w_i := w_i - \gamma \frac{\partial L(w)}{\partial w_i} = w_i - \frac{\gamma}{N} \sum_{j=1}^{N} \frac{\partial l_i(w)}{\partial w_i} \approx w_i - \frac{\gamma}{b} \sum_{j=1}^{b} \frac{\partial l_i(w)}{\partial w_i}$$

N is the size of the entire training dataset

b is called batch size

Content

- Stochastic Gradient Descent
- **How to compute gradients: Backpropagation and Automatic Differentiation**
- Understand Our Applications: An Overview of Neural Networks

How to compute gradients? Backpropagation

- Sum rule

$$\frac{d(f(x)+g(x))}{dx} = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$$

- Product rule

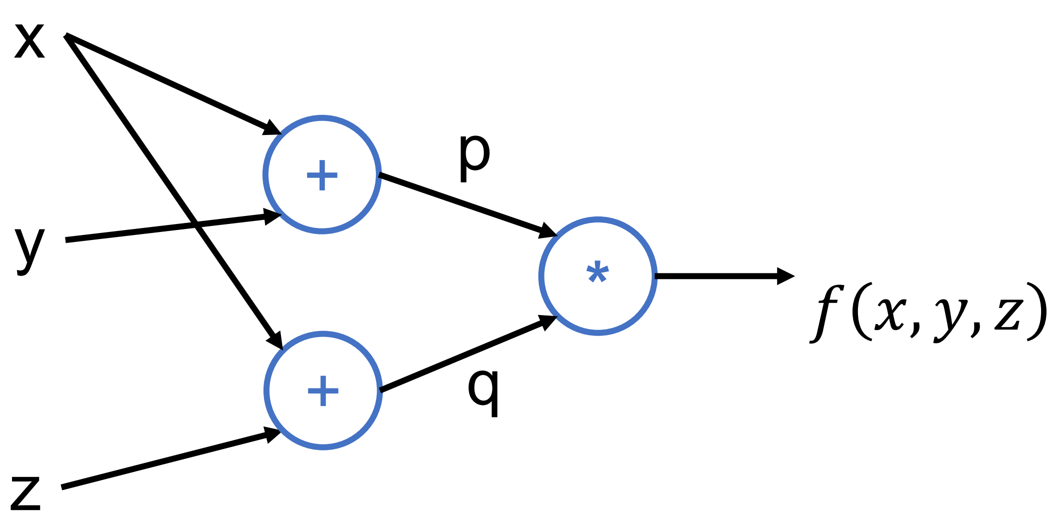
$$\frac{d(f(x)g(x))}{dx} = \frac{df(x)}{dx}g(x) + \frac{dg(x)}{dx}f(x)$$

- Chain rule

$$\frac{df(g(x))}{dx} = \frac{df(y)}{dy}\frac{dg(x)}{dx}$$

Backpropagation: a simple example

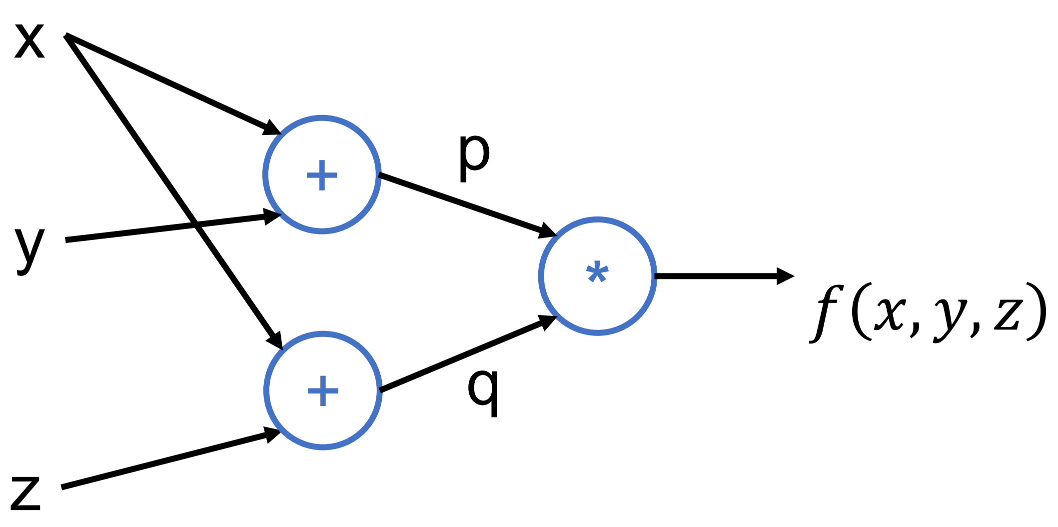
- $f(x, y, z) = (x + y)(x + z)$



Each node is an intermediate variable.
Computation graph (a DAG) with variable ordering from topological sort

Backpropagation: a simple example

- $f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4

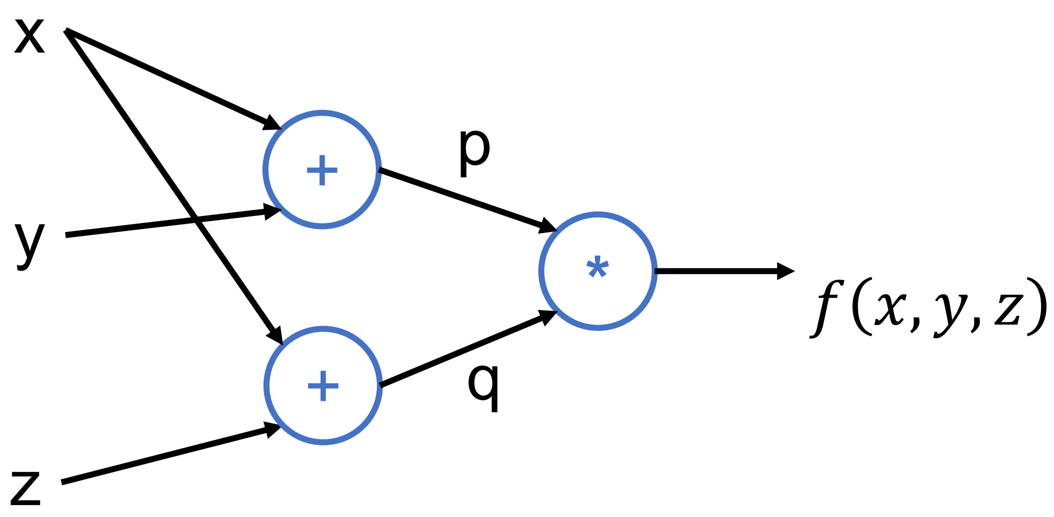


Each node is an intermediate variable.
Computation graph (a DAG) with variable ordering from topological sort

Exercise: Compute $\frac{\partial f}{\partial x}$

- $f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4

$$p = x + y \;\Rightarrow \frac{\partial p}{\partial x} = 1$$
$$q = x + z \;\Rightarrow \frac{\partial q}{\partial x} = 1$$
$$f = p * q \;\Rightarrow \frac{\partial f}{\partial x} = \frac{\partial p}{\partial x} * q + \frac{\partial q}{\partial x} * p$$
$$= 1 * -6 + 1 * 3 = -3$$



Each node is an intermediate variable.
Computation graph (a DAG) with variable ordering from topological sort

Exercise: Compute $\frac{\partial f}{\partial y}$

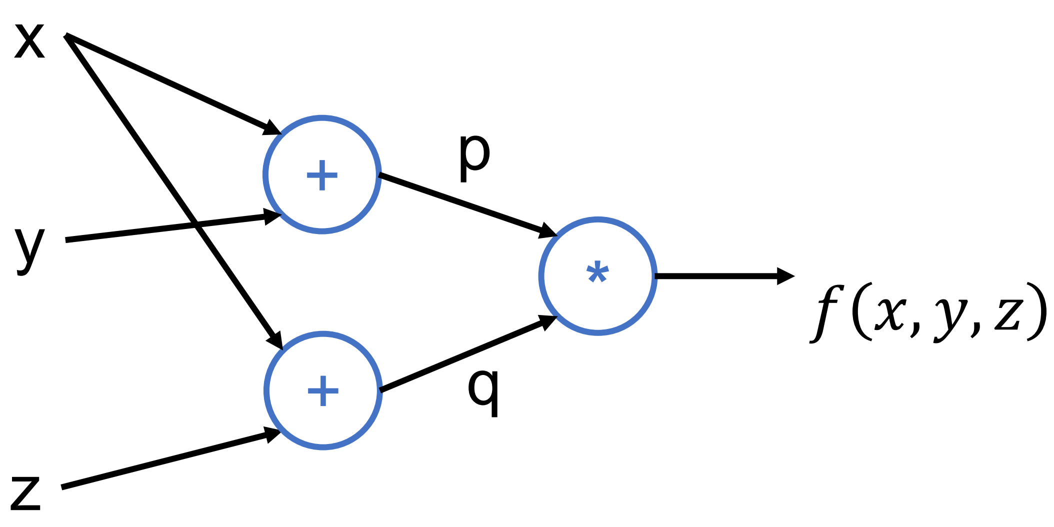
- $f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4

$$p = x + y \Rightarrow \frac{\partial p}{\partial y} = 1$$

$$q = x + z \Rightarrow \frac{\partial q}{\partial y} = 0$$

$$f = p * q \Rightarrow \frac{\partial f}{\partial y} = \frac{\partial p}{\partial y} * q + \frac{\partial q}{\partial y} * p$$

$$= 1 * -6 + 0 * 3 = -6$$



Each node is an intermediate variable. Computation graph (a DAG) with variable ordering from topological sort

Exercise: Compute $\frac{\partial f}{\partial z}$

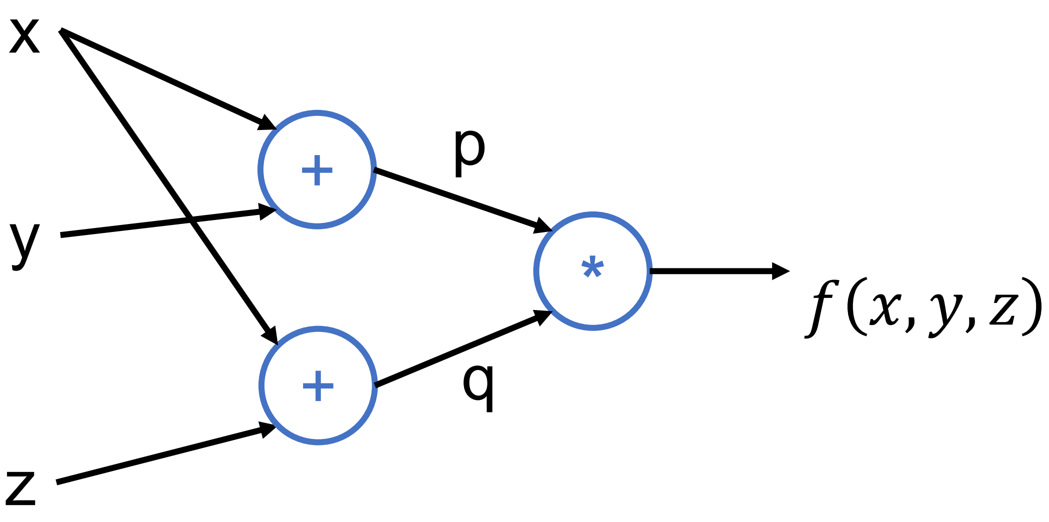
- $f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4

$$p = x + y \;\Rightarrow \frac{\partial p}{\partial z} = 0$$

$$q = x + z \;\Rightarrow \frac{\partial q}{\partial z} = 1$$

$$f = p * q \;\Rightarrow \frac{\partial f}{\partial z} = \frac{\partial p}{\partial z} * q + \frac{\partial q}{\partial z} * p$$

$$= 0 * -6 + 1 * 3 = 3$$



Each node is an intermediate variable.
Computation graph (a DAG) with variable ordering from topological sort

Issues?

- If a model has n input variables, we need n forward passes to compute the gradient with respect to each input
- Deep learning models have large number of inputs (e.g., billions or up to trillions of trainable weights)

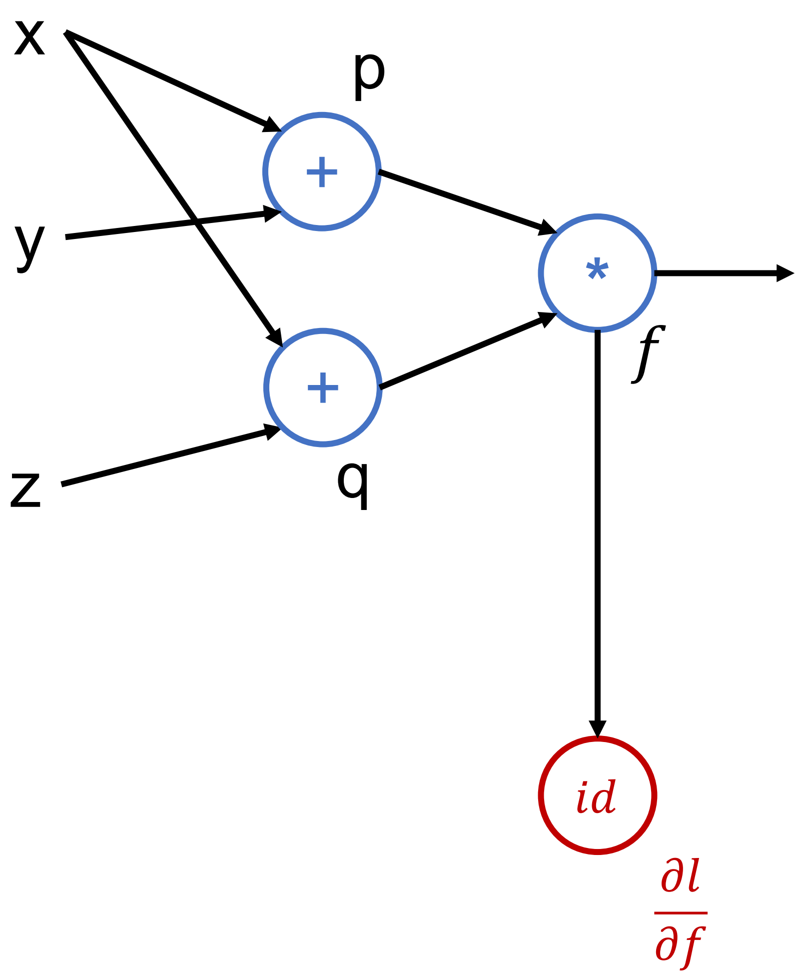
- Solution: reverse mode AutoDiff

Reverse Mode Automatic Differentiation

- For each node v, we introduce an adjoint node $\bar{v}$ corresponding to the gradient of output wrt to this node $\frac{\partial f}{\partial v}$
- Compute nodes' gradients in a reverse topological order

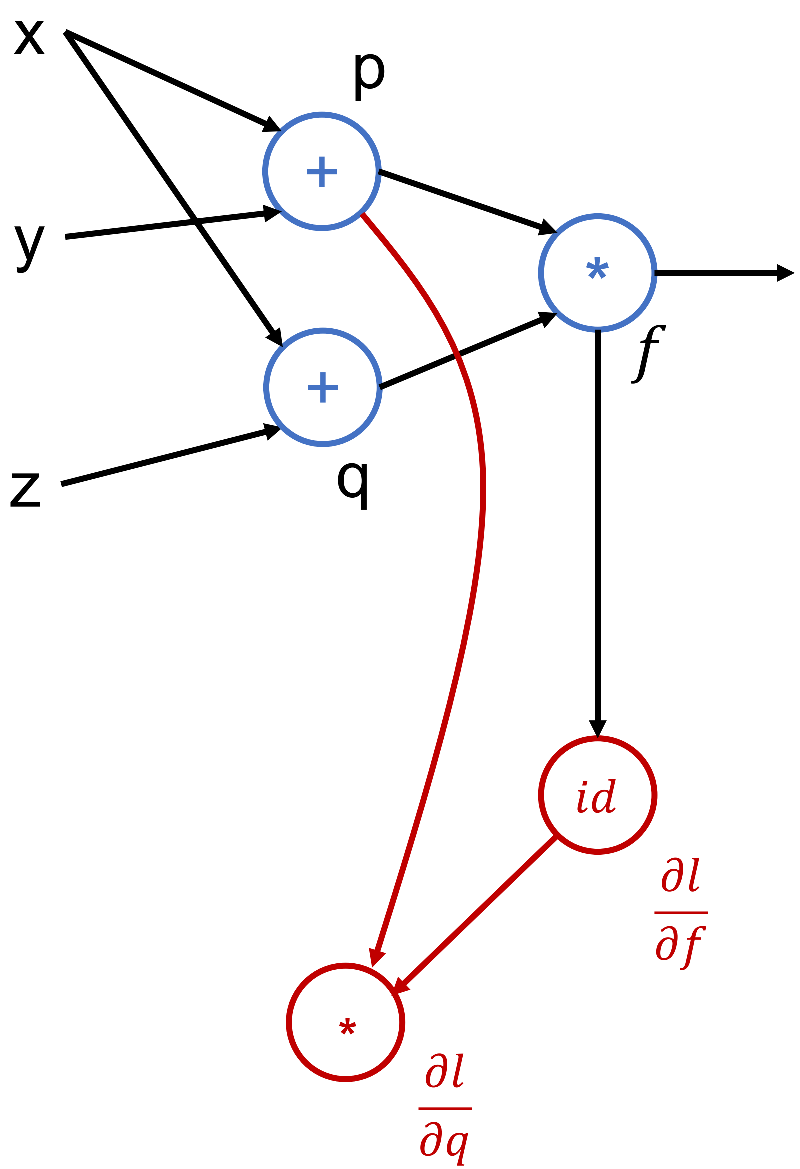
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$



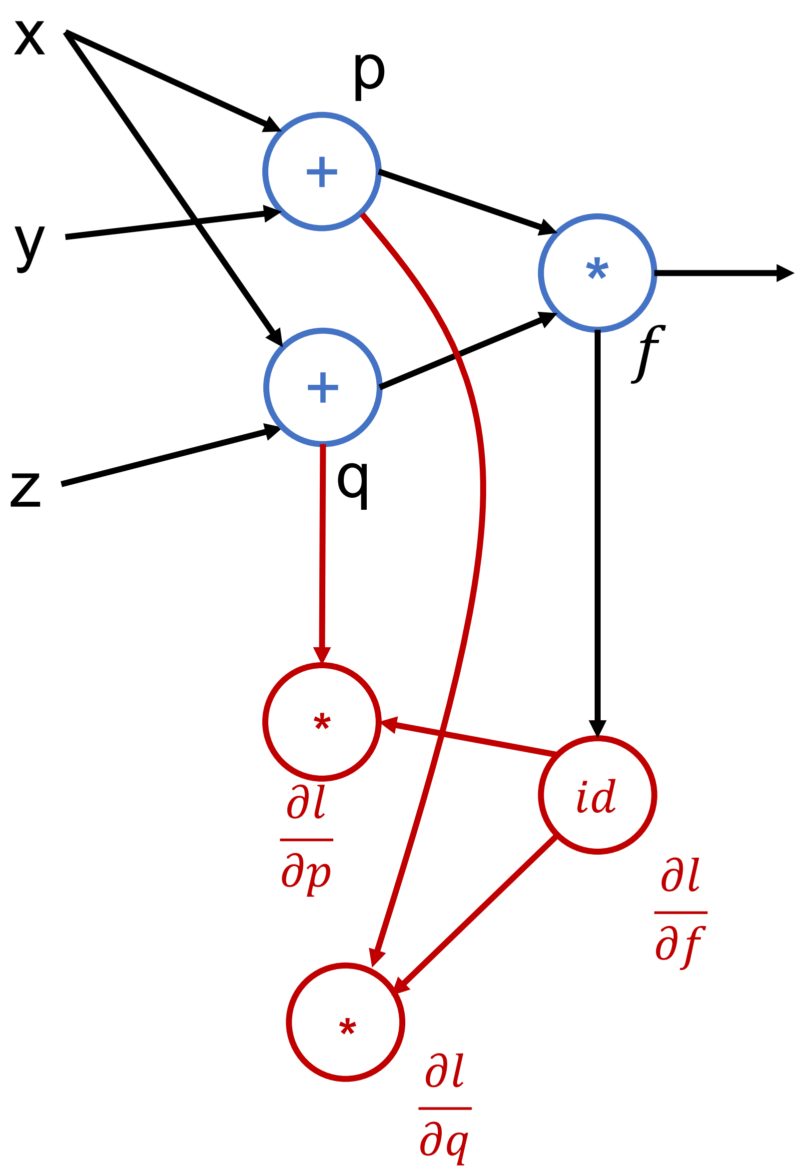
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = \frac{\partial l}{\partial f} \times p = 3$



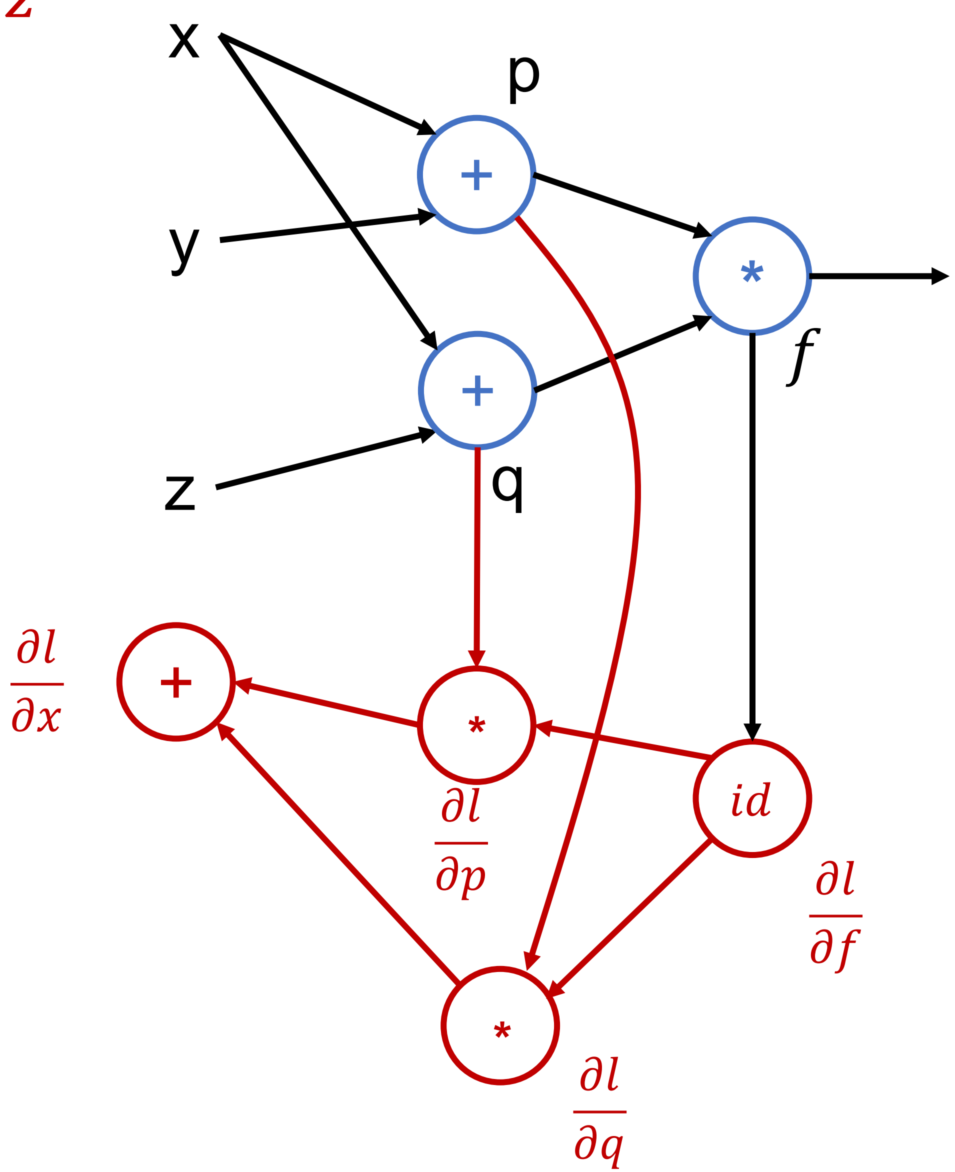
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = 1 \times p = 3$
- $\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial p} = \frac{\partial l}{\partial f} \times q = -6$



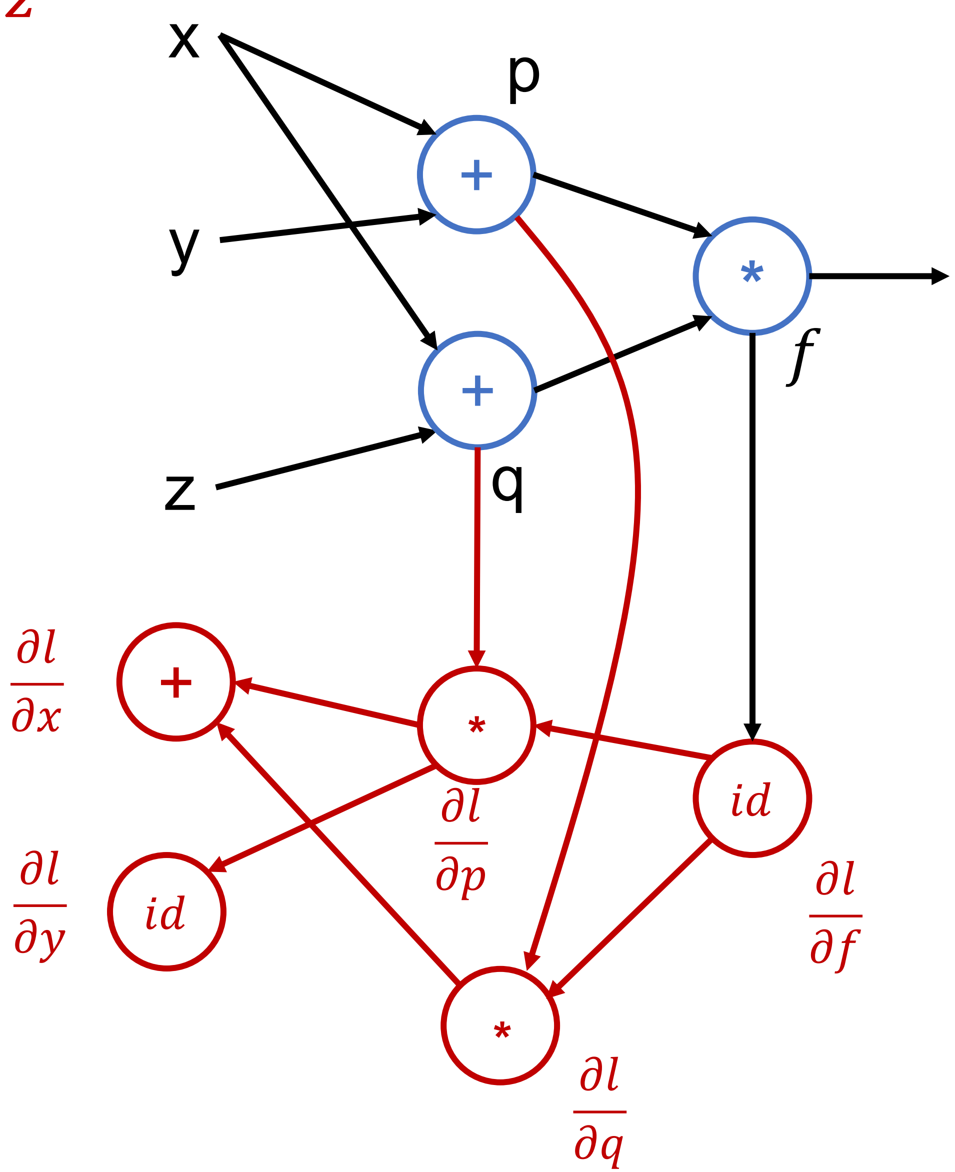
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = 1 \times p = 3$
- $\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial p} = 1 \times q = -6$
- $\frac{\partial l}{\partial x} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial x} + \frac{\partial l}{\partial q} \times \frac{\partial q}{\partial x} = \frac{\partial l}{\partial p} + \frac{\partial l}{\partial q}$



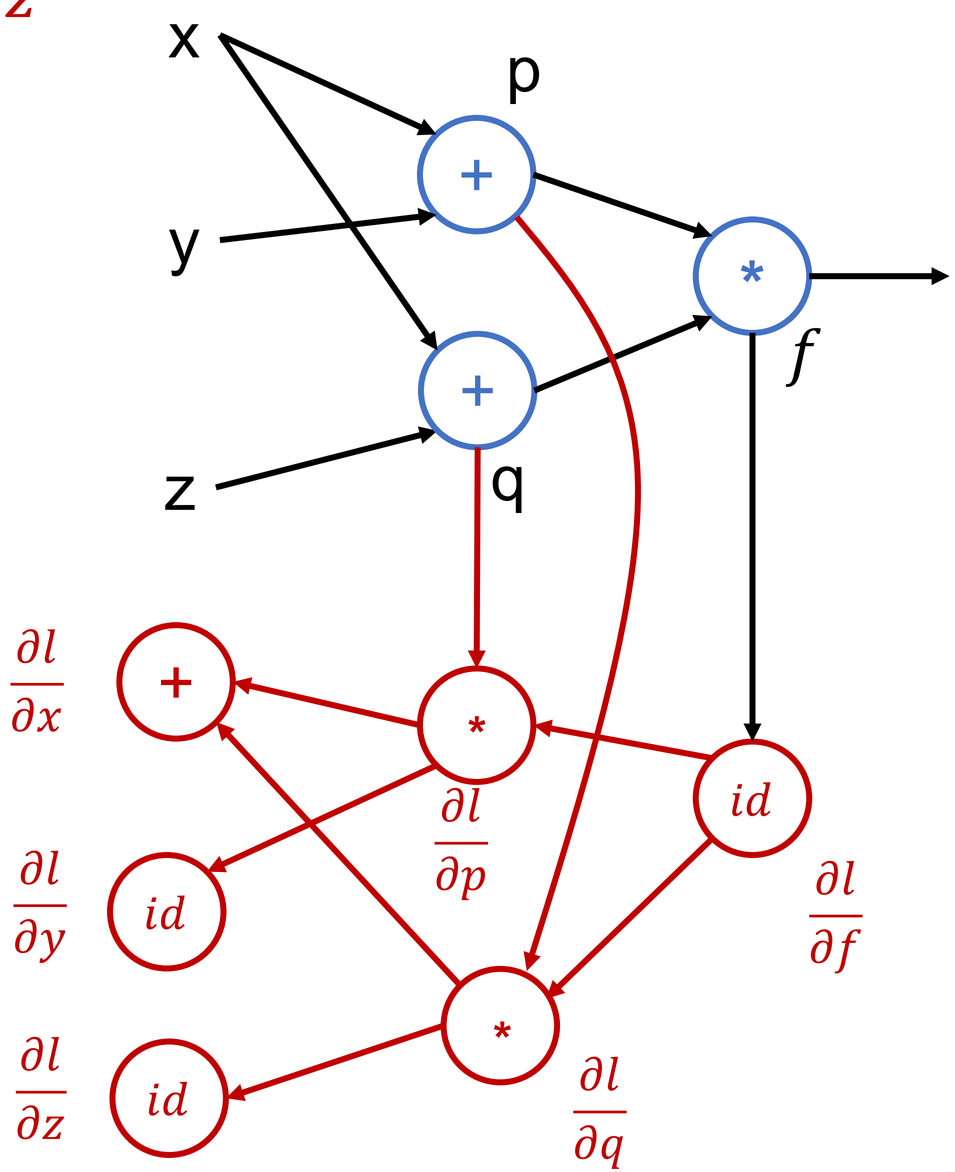
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

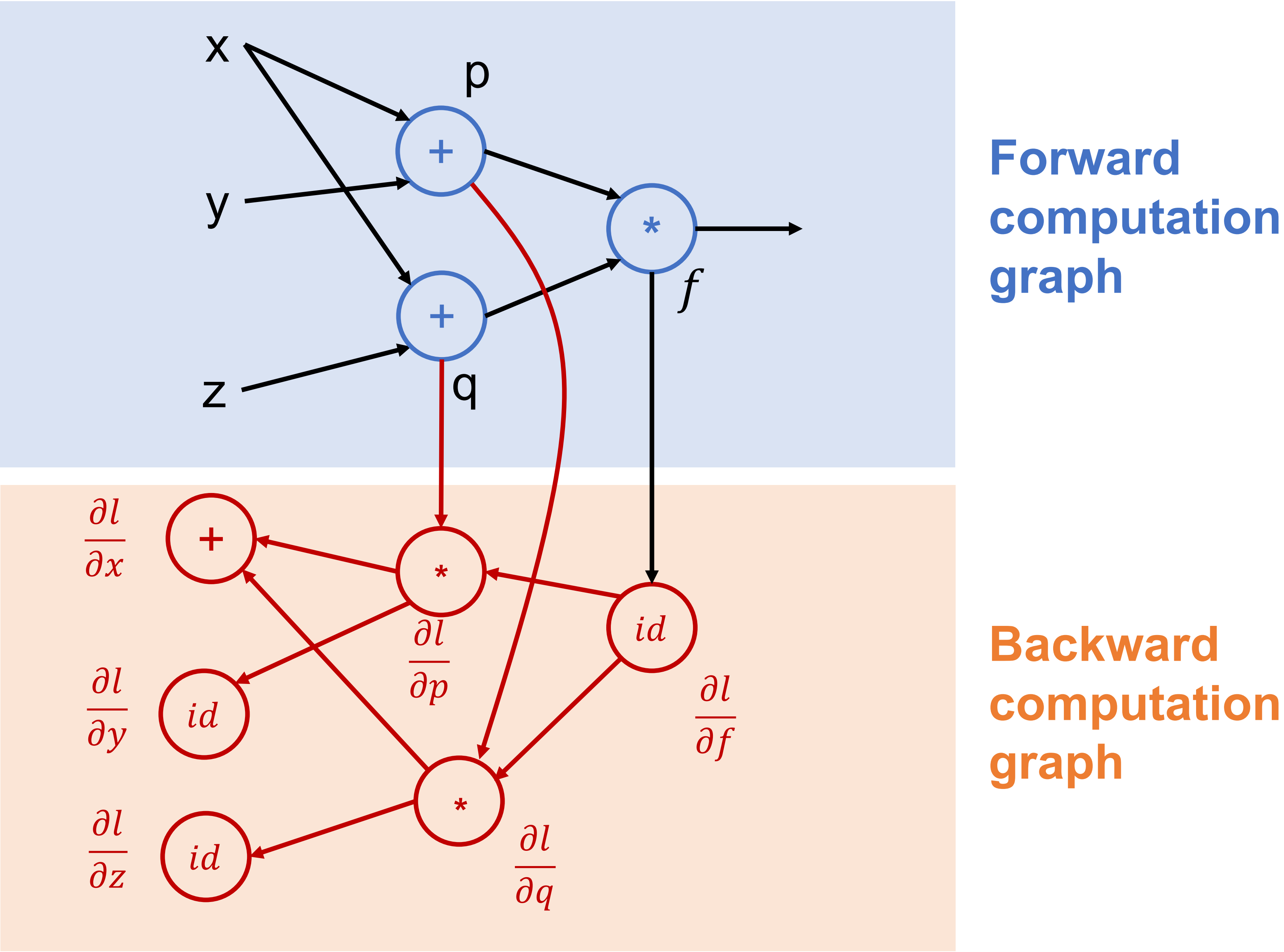
- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = 1 \times p = 3$
- $\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial p} = 1 \times q = -6$
- $\frac{\partial l}{\partial x} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial x} + \frac{\partial l}{\partial q} \times \frac{\partial q}{\partial x} = -3$
- $\frac{\partial l}{\partial y} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial y} = -6 \times 1 = -6$



Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

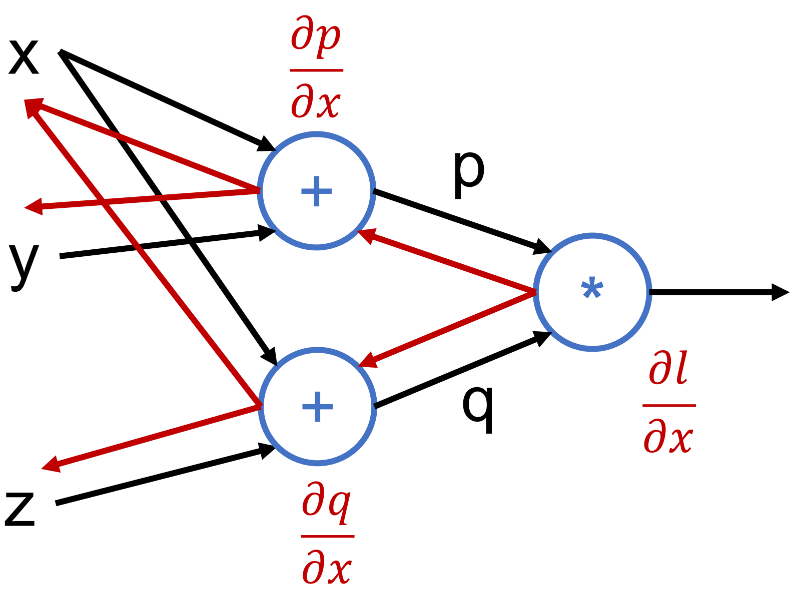
- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = 1 \times p = 3$
- $\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial p} = 1 \times q = -6$
- $\frac{\partial l}{\partial x} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial x} + \frac{\partial l}{\partial q} \times \frac{\partial q}{\partial x} = -3$
- $\frac{\partial l}{\partial y} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial y} = -6 \times 1 = -6$
- $\frac{\partial l}{\partial z} = \frac{\partial l}{\partial q} \times \frac{\partial q}{\partial z} = 3 \times 1 = 3$



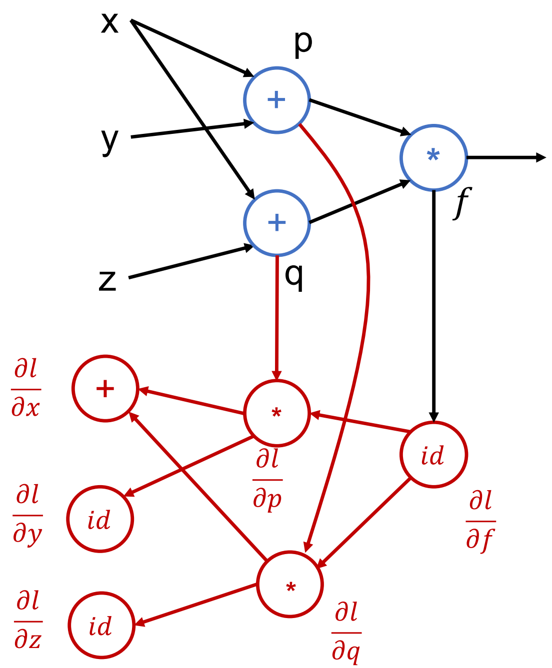

x
y
z
p
q
+
+
*
f
$\frac{\partial l}{\partial x}$
+
*
$\frac{\partial l}{\partial p}$
id
$\frac{\partial l}{\partial f}$
$\frac{\partial l}{\partial y}$
id
*
$\frac{\partial l}{\partial q}$
$\frac{\partial l}{\partial z}$
id
Forward computation graph
Backward computation graph

Discuss: Backpropagation v.s. Reverse AutoDiff

Backpropagation



Reverse AutoDiff

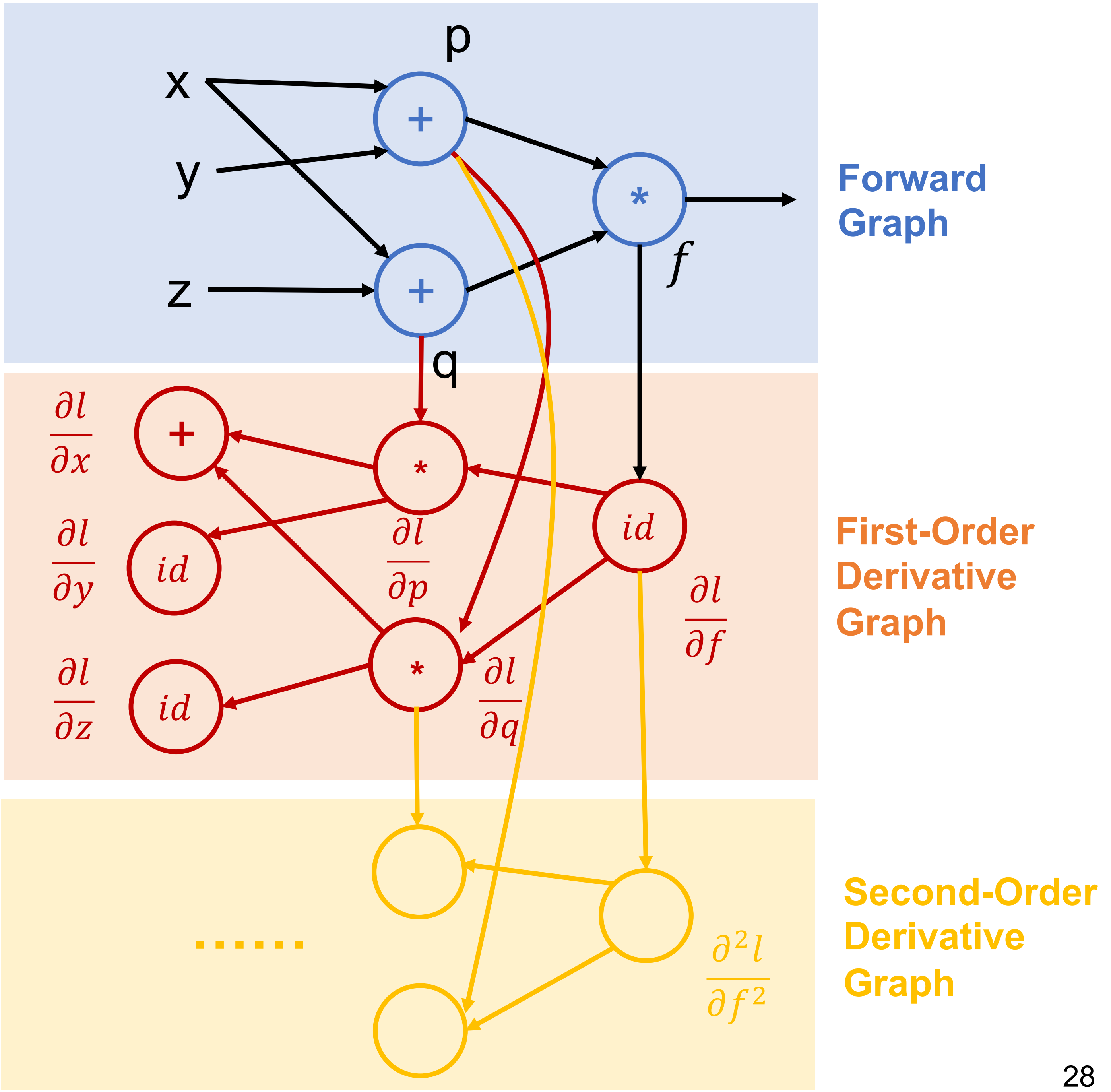


What is the difference between backpropagation and reverse AutoDiff?

Discuss: Backpropagation v.s. Reverse AutoDiff

- **Complexity**: backpropagation requires a forward-backward pass for each variable, while reverse AutoDiff only requires one forward-backward pass
- **Optimization**: reverse AutoDiff represents the forward-backward in a single computation graph, make it easier to apply graph-level optimizations
- **Higher order derivatives**: we can take derivative of derivative nodes in reserve AutoDiff, while it's much harder to do so in backpropagation

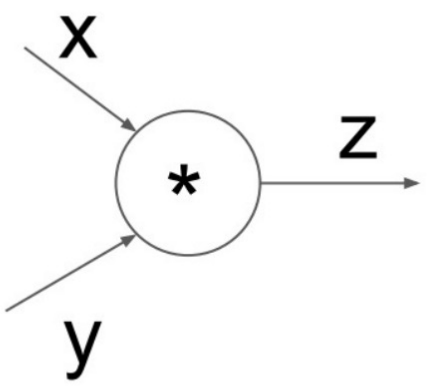
Higher Order Derivatives



Reverse AutoDiff Implementation

```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Forward/Backward Implementation



(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Manual Gradient Checking: Numeric Gradient

- How do we check the correctness of our implementation?

- For small δ, $\frac{\partial l}{\partial x_i} \approx \frac{l(x+\delta \overrightarrow{x_i}) - l(x-\delta \overrightarrow{x_i})}{2\delta}$
  - **Pros**: easy to implement
  - **Cons**: approximate and very expensive to compute; need to recompute $l(x + \delta \overrightarrow{x_i})$ for **every parameter**
  - Useful for checking the correctness of our implementation; serve as unit test in today's DNN systems

given x_0.

$\forall x$,

Summary

We have learnt a core technique of MLSys 🎊

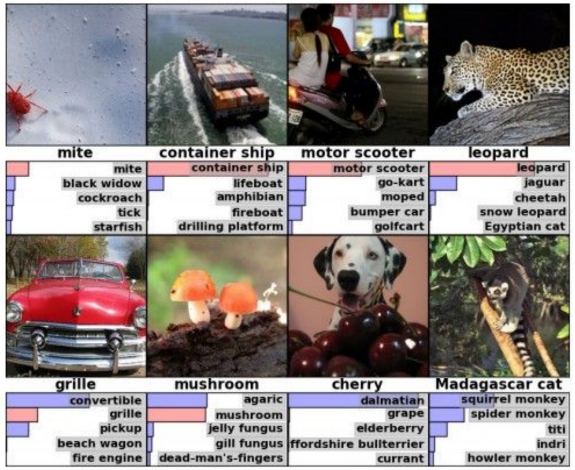
- **Forward pass**: apply model to a batch of input samples and run calculation through operators and save intermediate results
- **Backward pass**: run the model in reverse and apply chain rule to compute gradients
- **Weight update**: use the gradients to update model weights

$$w_i := w_i - \gamma \frac{\partial L(w)}{\partial w_i} = w_i - \frac{\gamma}{N} \sum_{j=1}^{N} \frac{\partial l_i(w)}{\partial w_i} \approx w_i - \frac{\gamma}{b} \sum_{j=1}^{b} \frac{\partial l_i(w)}{\partial w_i}$$

Understand Our Applications:
An Overview of Deep Learning Models

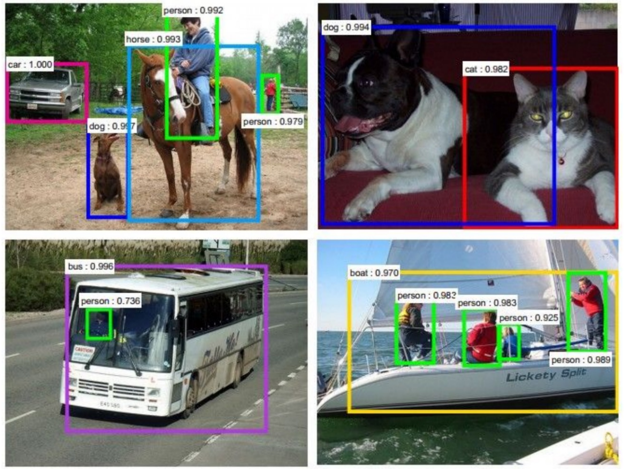
- **Convolutional Neural Networks**
- **Recurrent Neural Networks**
- **Transformers**
- **Graph Neural Networks**
- **Mixture-of-Experts**

CNNs are widely used in vision tasks

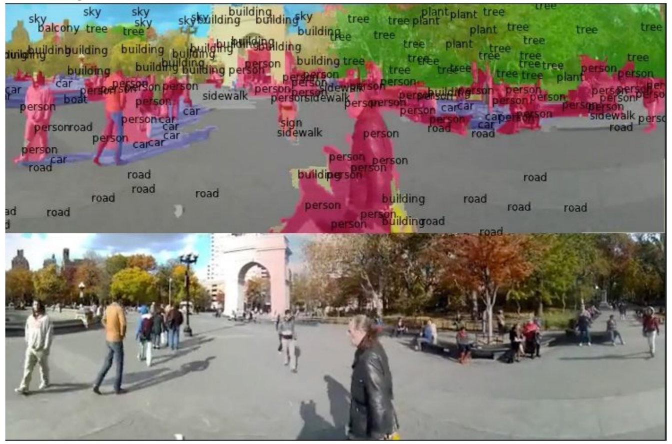


Classification

Retrieval



Detection



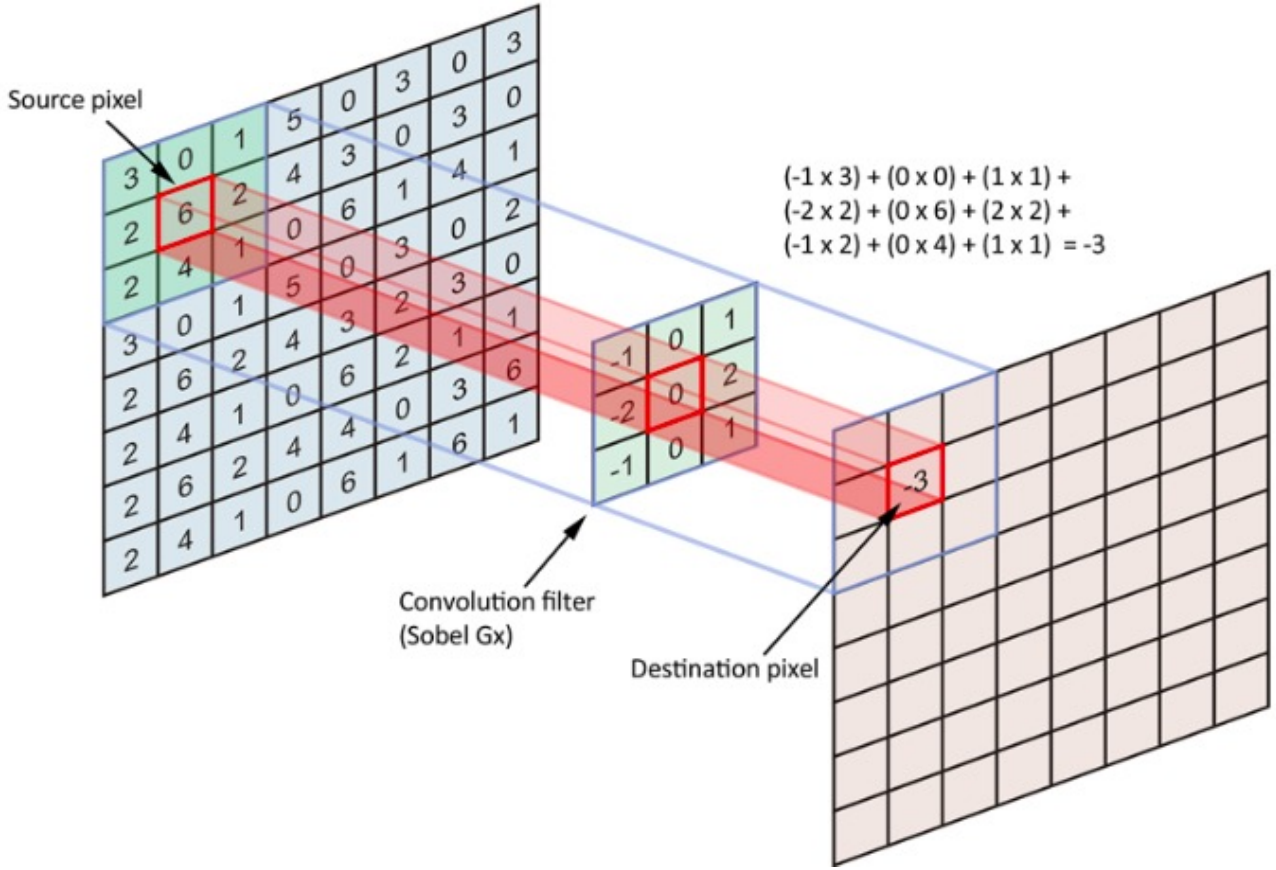
Segmentation

Self-Driving

Synthesis

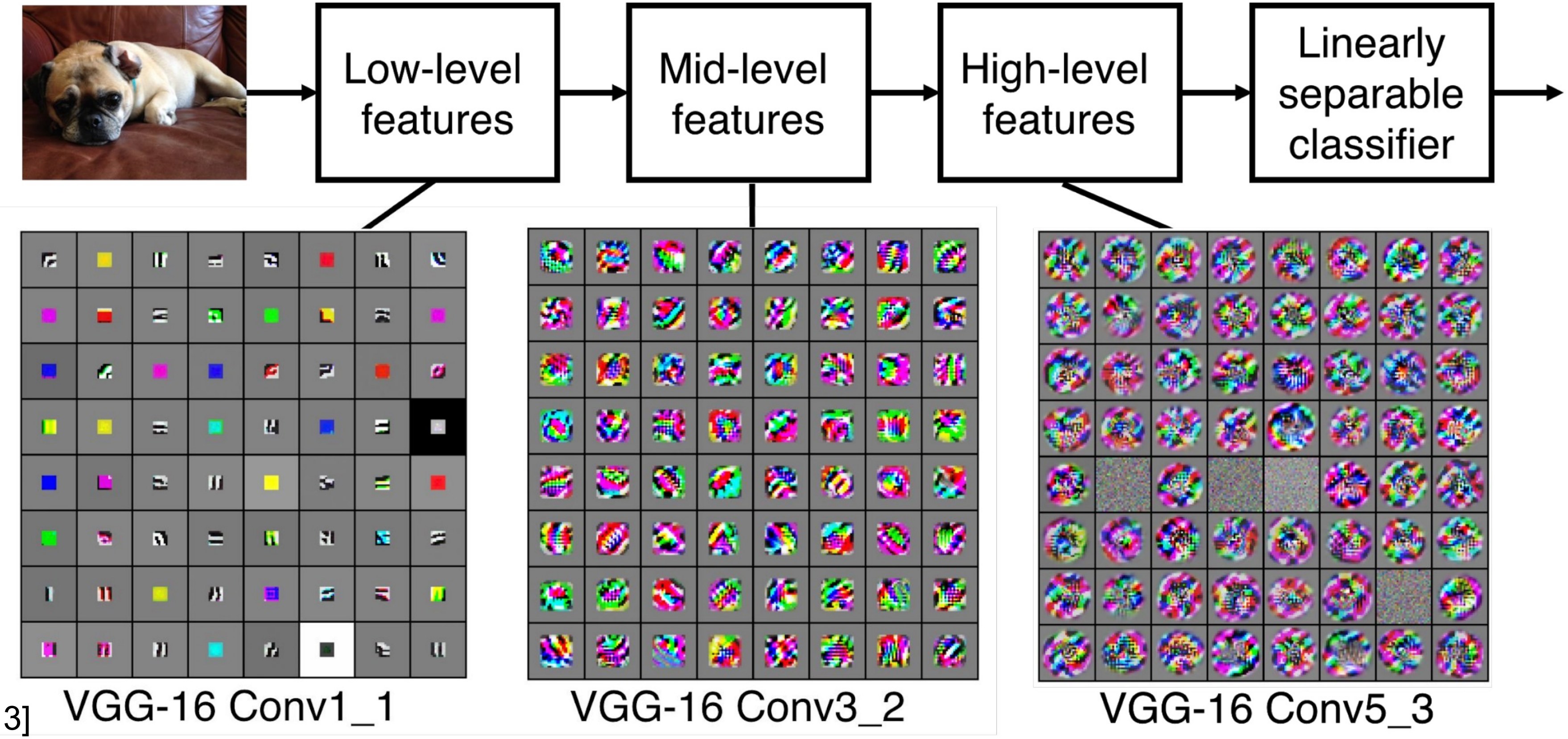
Recap: Convolution

- Convolve the filter with the image: slide over the image spatially and compute dot products



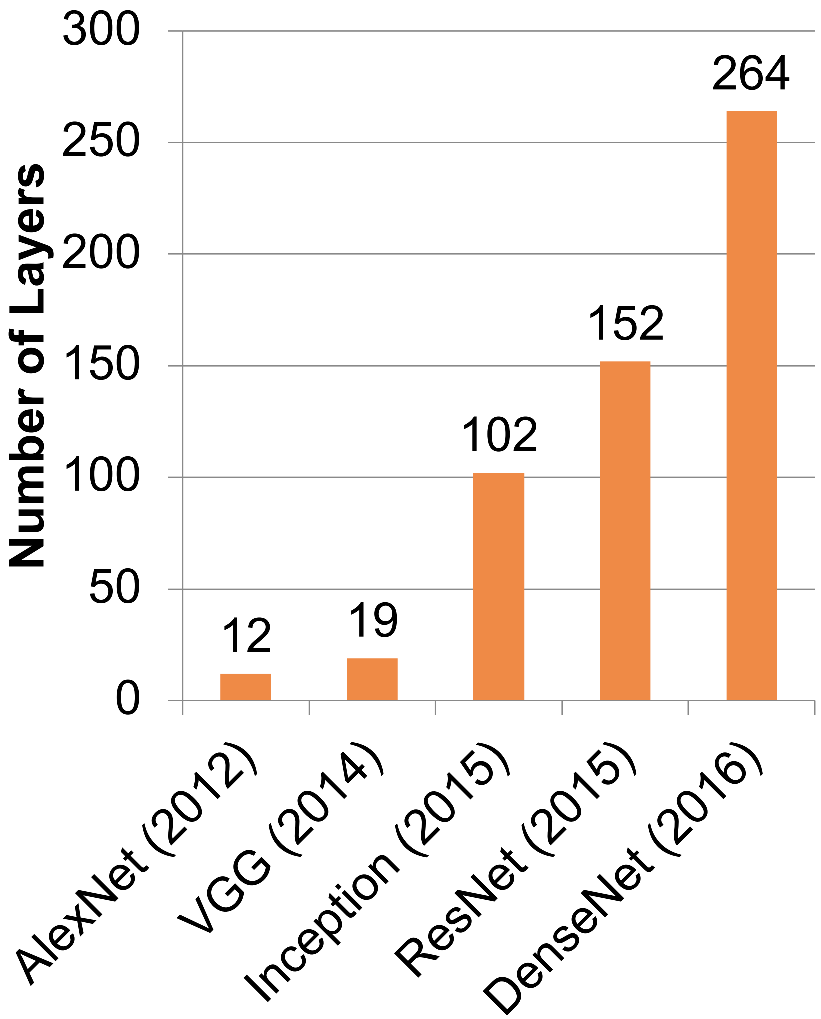
CNNs

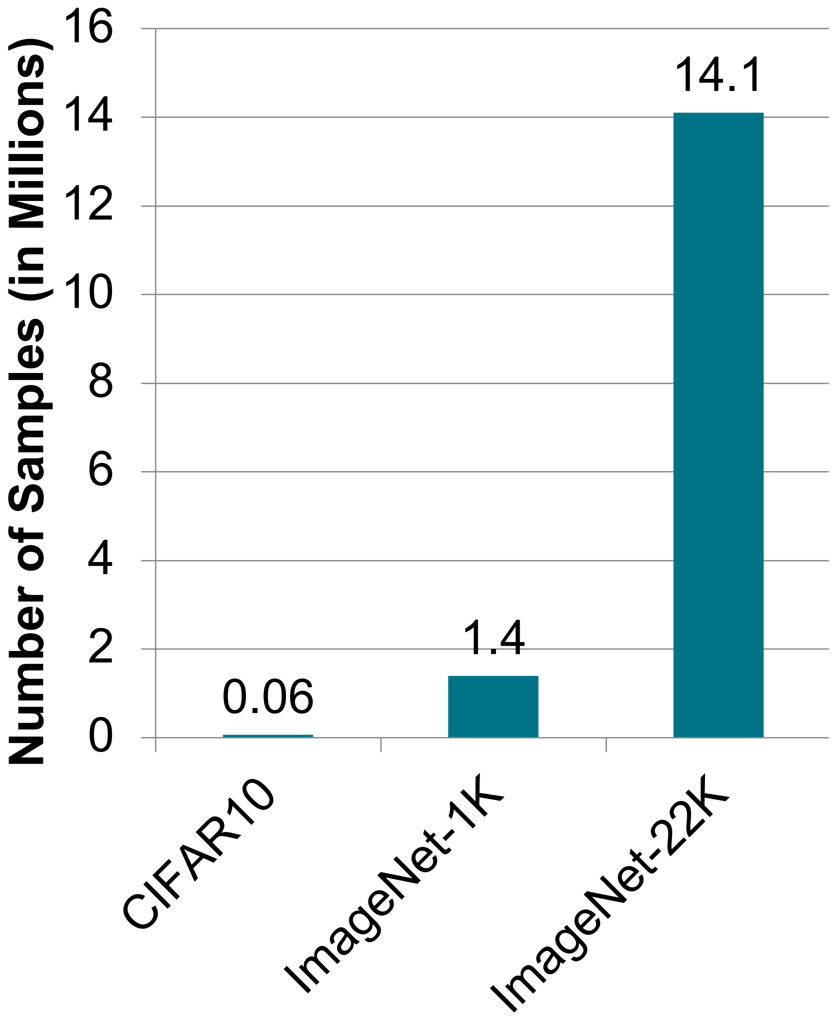
- A sequence of convolutional layers, interspersed by pooling, normalization, and activation functions



[Zeiler and Fergus 2013]

MLSys Challenges in CNNs: Increasing Computational Requirements





MLSys Challenges in CNNs: Increasing Computational Requirements

- **Computational cost**: convolutions are extremely compute-intensive
- **Memory requirement**: high-resolution images cannot fit in a single GPU

- **Solution**: parallelize training across GPUs
  - Week 6: Data and Model Parallelism for Distributed Training
  - Week 6: Pipeline Parallelism for Distributed Training
  - Week 7: Memory-Efficient Training

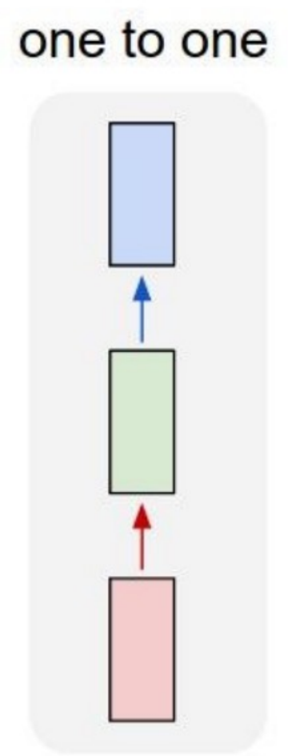
MLSys Challenges in CNNs: Efficiency

- **Efficiency**: difficult to deploy most accurate CNNs on edge devices with limited compute resources, memory capacity, and energy
- **Solution**: model compression, efficient convolution algorithms, neural architecture search
  - Week 11: Model Compression
  - Week 4: ML Compilation
  - Week 12: ML Hardware

Understand Our Applications:
An Overview of Deep Learning Models

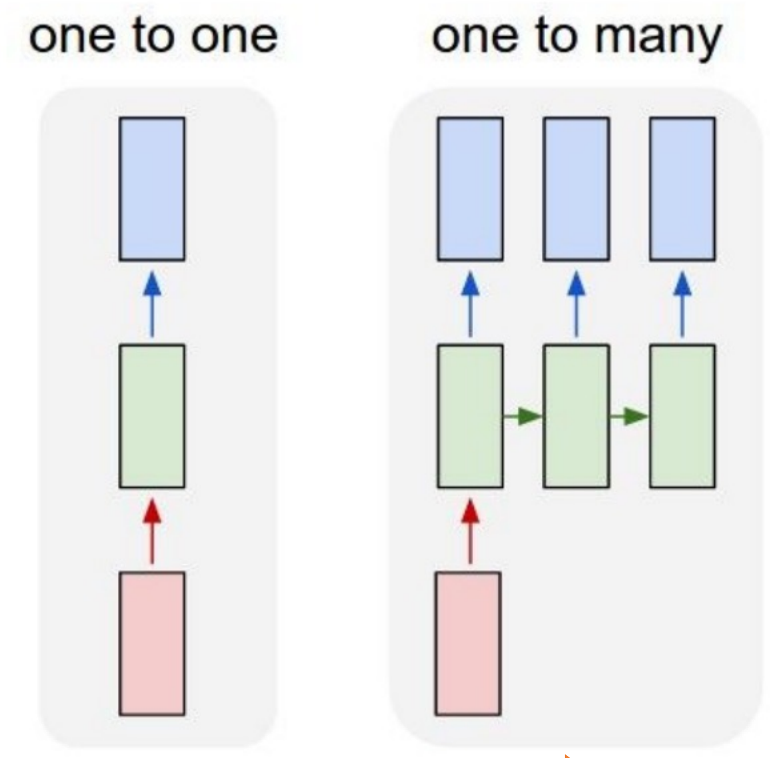
- Convolutional Neural Networks: vision tasks
- **Recurrent Neural Networks**
- Transformer
- Graph Neural Networks
- Mixture-of-Experts

Recurrent Neural Networks: Process Sequences



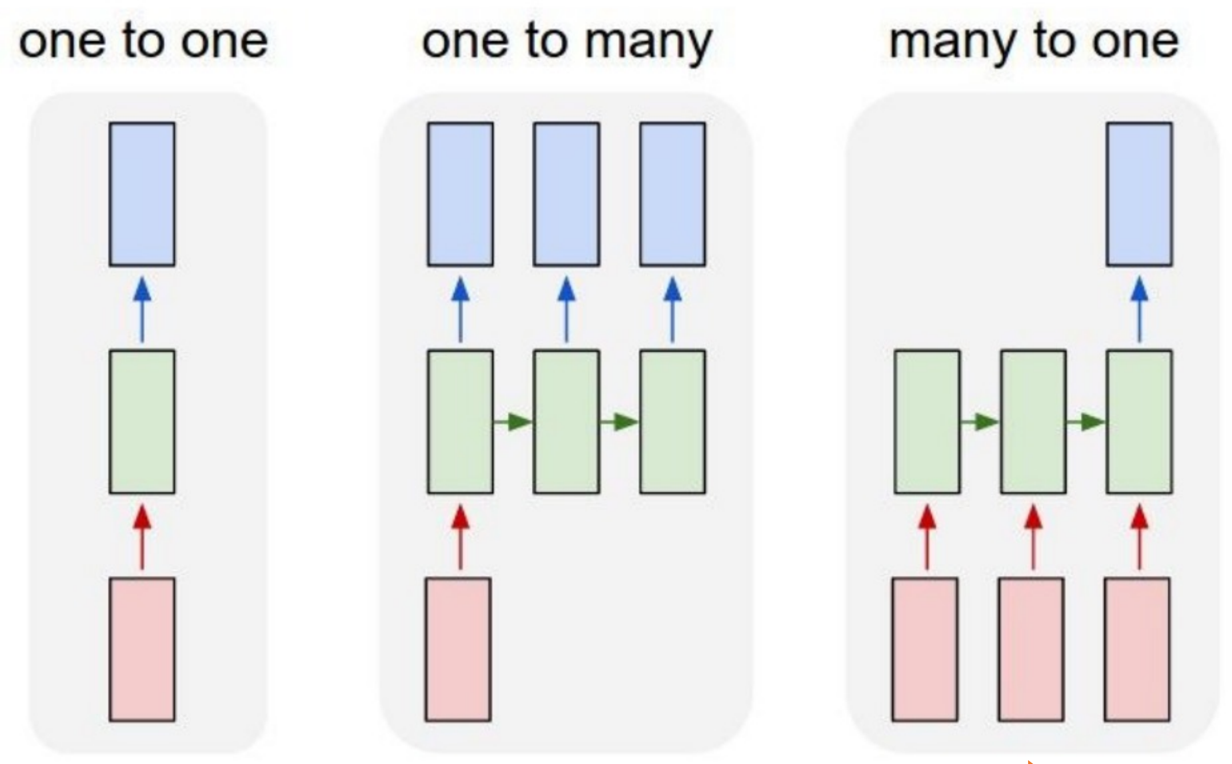
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



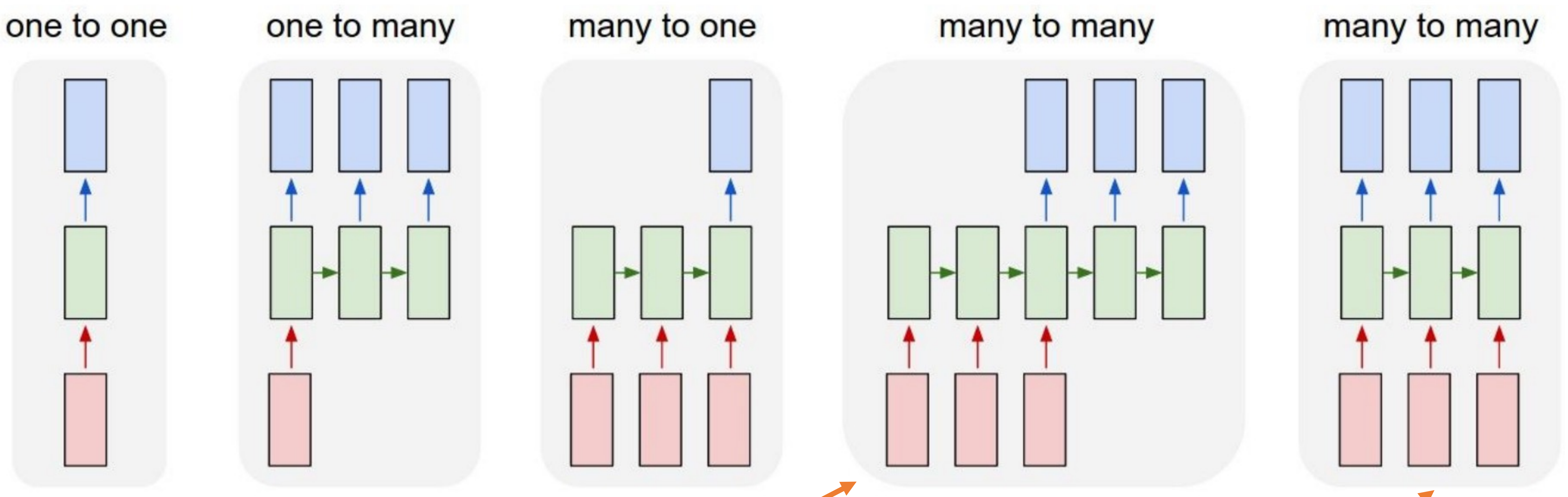
e.g., image captioning
Image -> sequence of words

Recurrent Neural Networks: Process Sequences



e.g., action prediction
sequence of video frames -> action

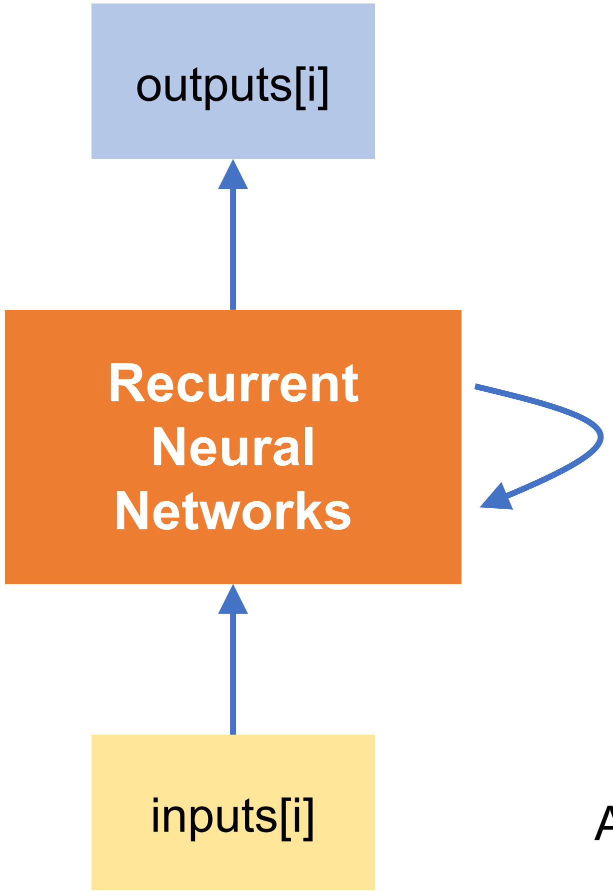
Recurrent Neural Networks: Process Sequences



Video captioning: sequence of video frames -> sequence of words
Machine translation

Video classification on frames

Recurrent Neural Networks



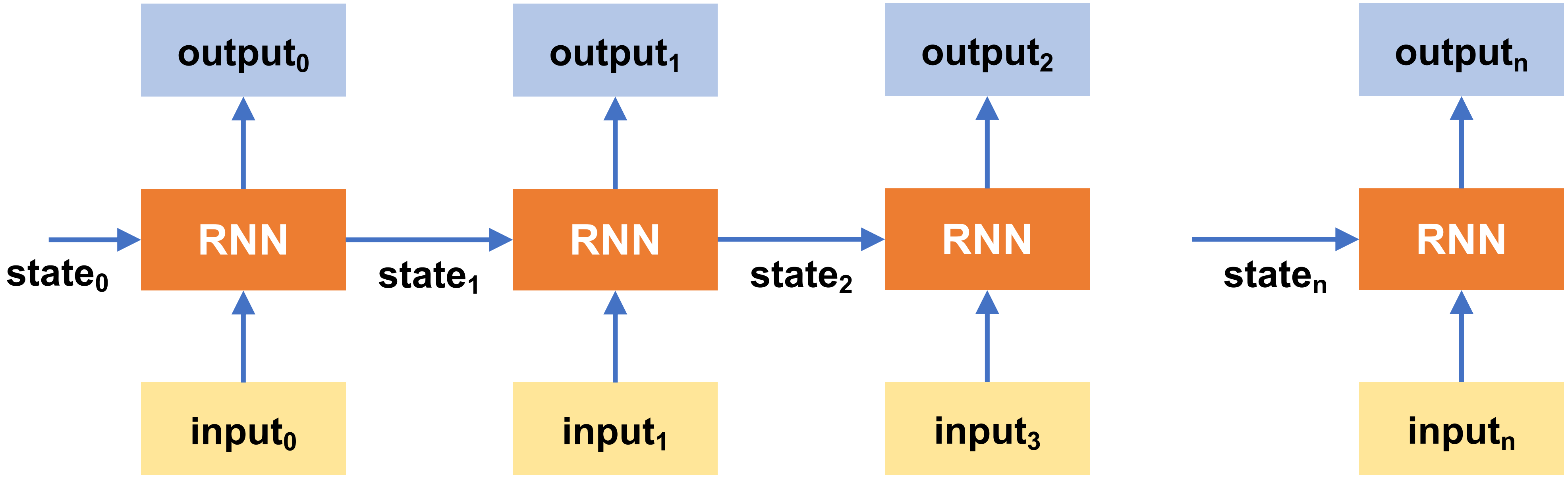
Arbitrary number of outputs

Key idea: RNNs have an internal state that is updated as a sequence is processed

Arbitrary number of inputs

How to Represent RNNs in Computation Graphs

- Computation graphs must be direct acyclic graphs (DAGs) but RNNs have self loops
- **Solution**: unrolling RNNs (define maximum depth)



When do we need RNNs?

- RNNs are designed to process sequences (texts, videos)
- RNNs are extremely useful when you want your model to have internal states when a sequence is processed
  - Commonly used in reinforcement learning (RL)

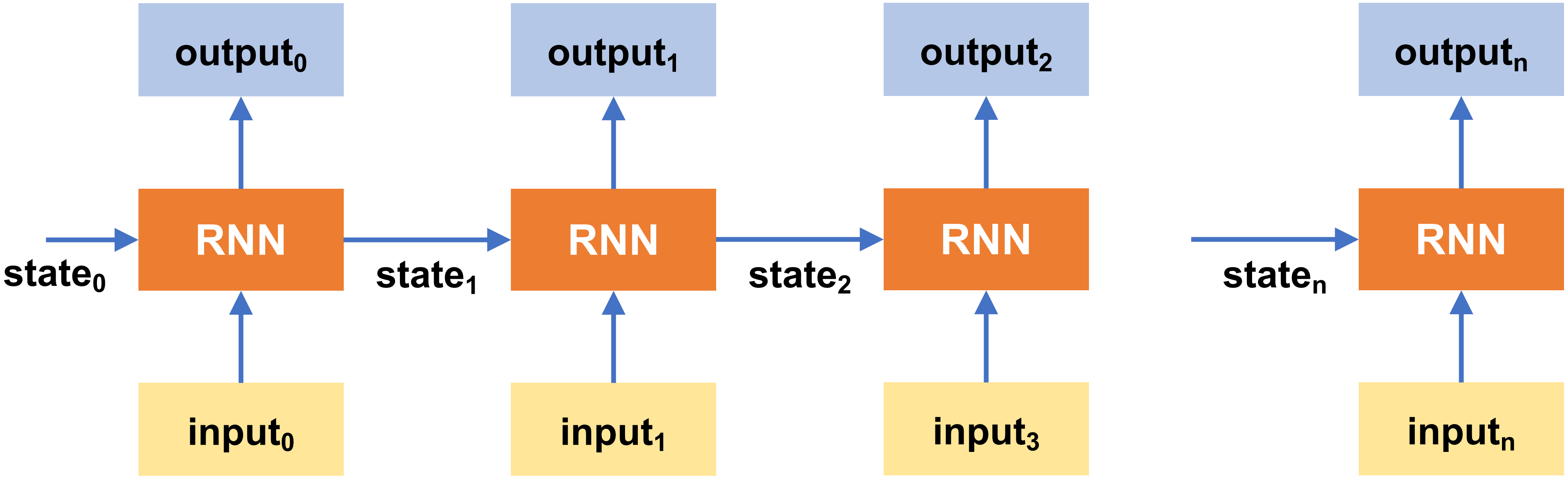
- Week 5: RL for device placement and graph optimizations

Understand Our Applications:
An Overview of Deep Learning Models

- Convolutional Neural Networks
- Recurrent Neural Networks
- **Transformers**
- Graph Neural Networks
- Mixture-of-Experts

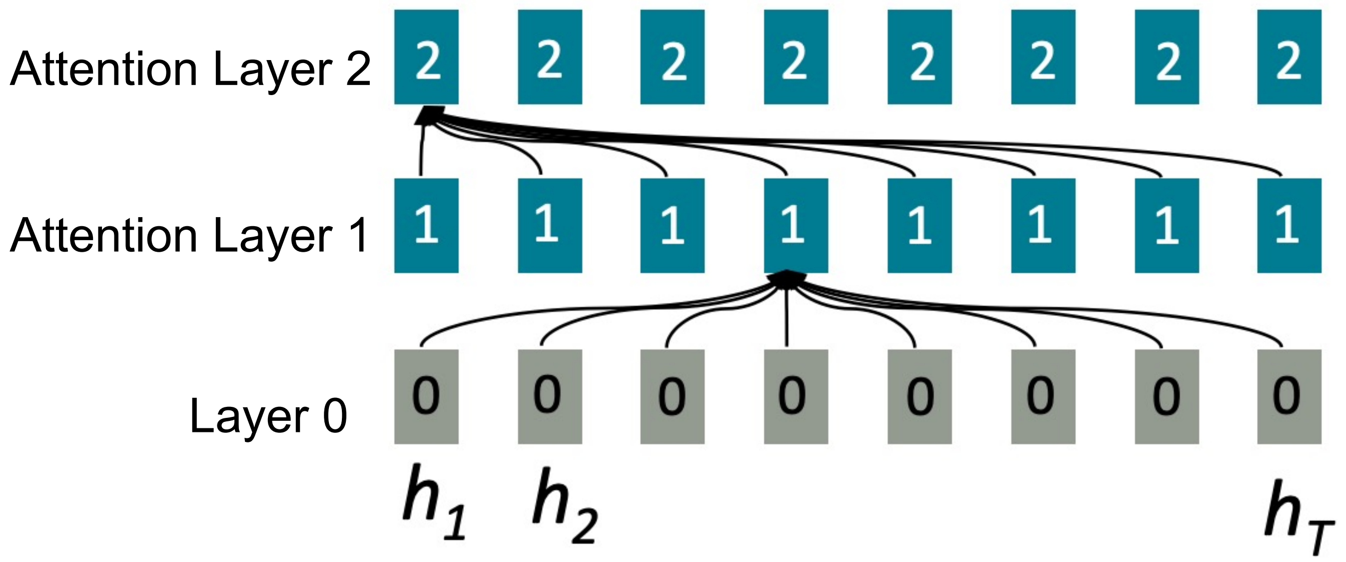
Inefficiency in RNNs?

- **Problem**: lack of parallelizability. Both forward and backward passes have O(sequence length) unparallelizable operators
  - A state cannot be computed before all previous states have been computed
  - Inhibits training on very long sequences



Attention: Enable Parallelism within a Sequence

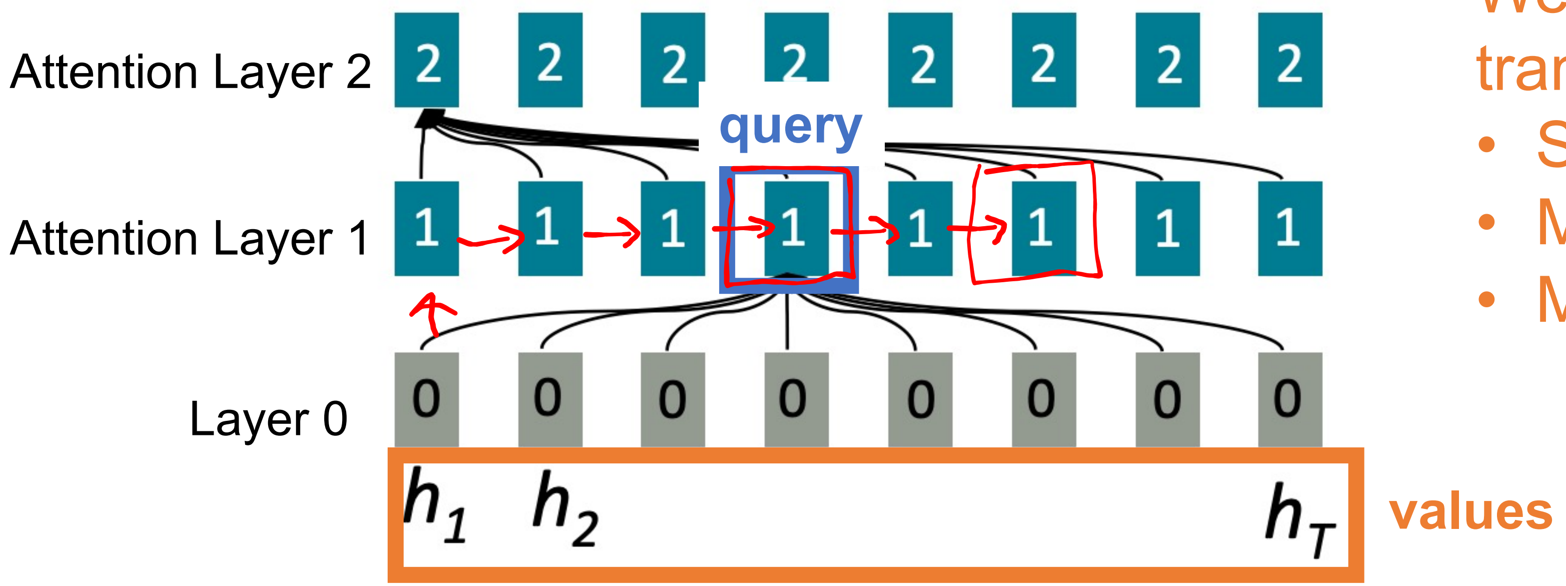
- Idea: treat each position's representation as a **query** to access and incorporate information from a set of **values**



Attention: Enable Parallelism within a Sequence

- Idea: treat each position's representation as a **query** to access and incorporate information from a set of **values**

- Massively parallelizable: number of unparallelizable operations does not increase sequence length



We will learn attention and transformers in depth later:
- Self-attention
- Masked attention
- Multi-head attention

MLSys Challenges: Exponentially Increasing Computational Costs for Transformer Models

We have seen similar trend in CNNs before.

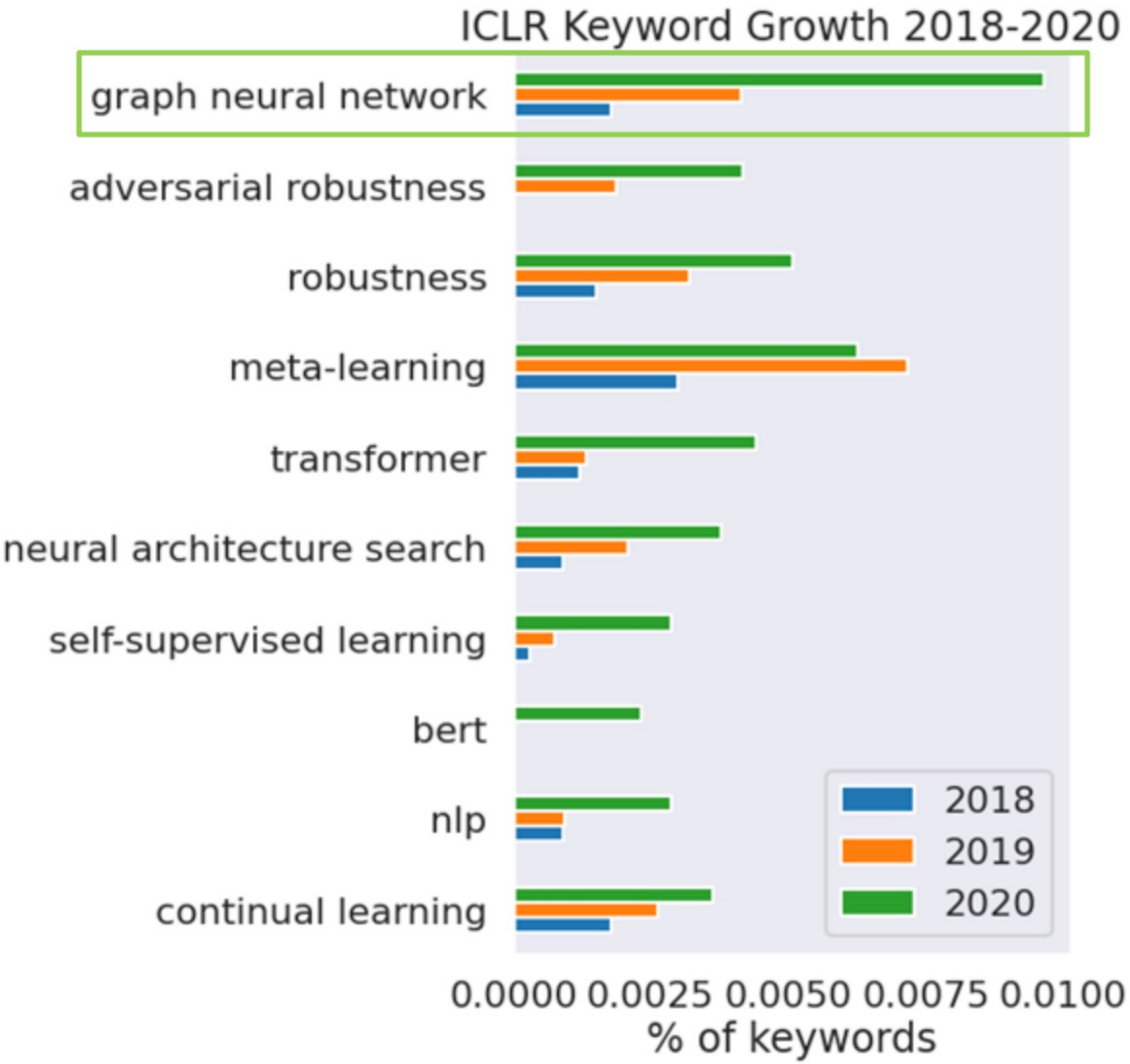
How to Design Systems for Transformers Models?

- **Memory efficiency** (week 7): how to train large Transformers on GPUs with limited memory
- **Distributed training** (week 9): how to design customized parallelization strategies for multi-head attention computation
- **Advanced model design** (week 11): Switch Transformers = Transformers + Mixture-of-Experts

Understand Our Applications:
An Overview of Deep Learning Models

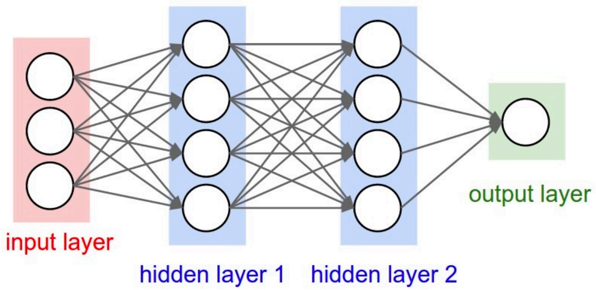
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transformers
- **Graph Neural Networks**
- Mixture-of-Experts

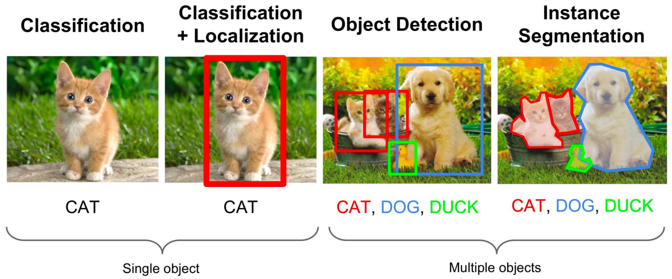
Graph Neural Networks: The Hottest Subfield in ML



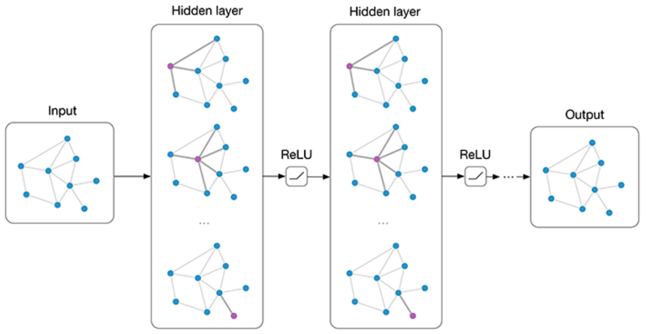
GNNs: Neural Networks on Relational Data

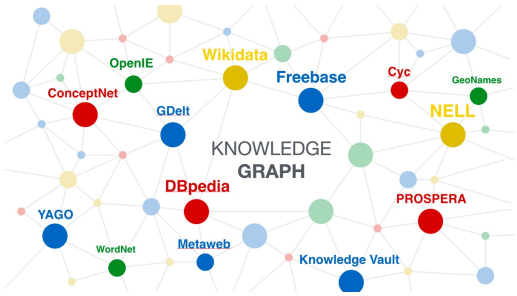
Neural Networks





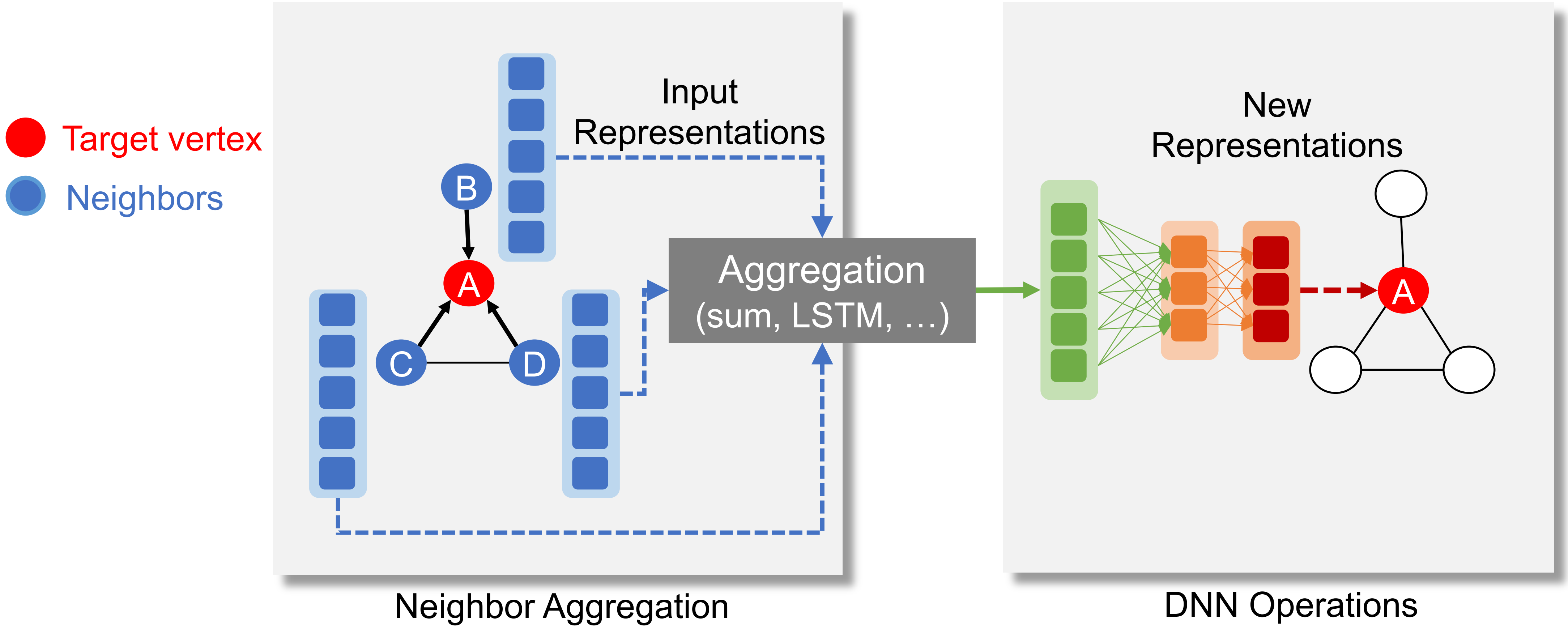
Graph Neural Networks





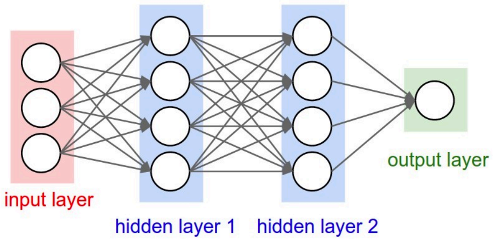
Graph Neural Network Architecture

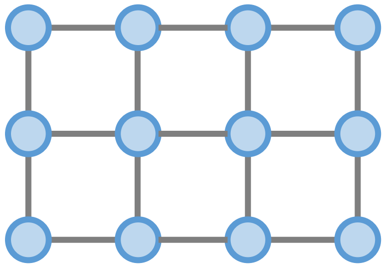
- Combine **graph propagation** w/ **neural network operations**



Challenges of GNN Computations on GPUs

Neural Networks

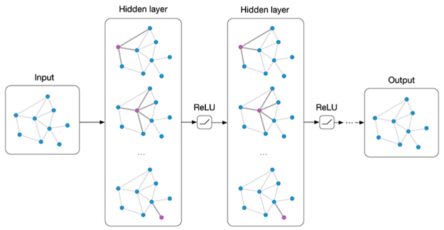


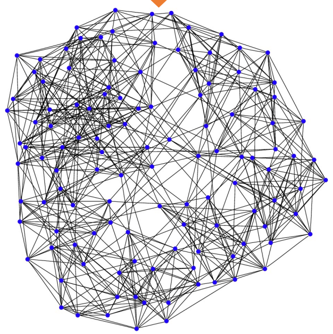


Small and regular intermediate data

Good efficiency

Graph Neural Networks





Large and irregular intermediate data

Efficiency & scalability challenges

How to Design Systems for Graph Neural Networks

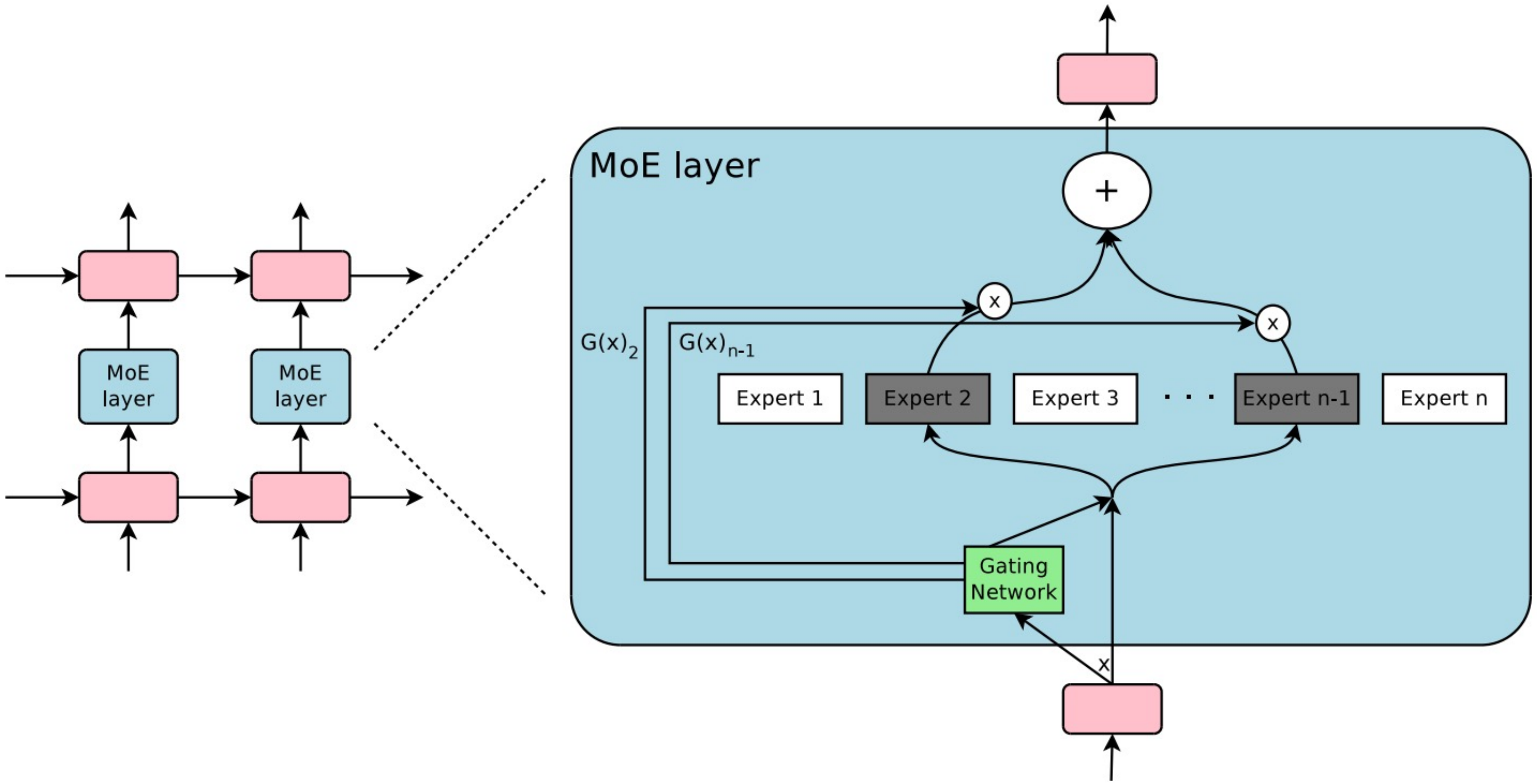
- **New Programming Models** (week 10): gather-apply-scatter programming interface for distributed GNN
- **New Systems Infrastructure** (week 10): serverless computing for low-cost GNN training

Understand Our Applications: An Overview of Deep Learning Models

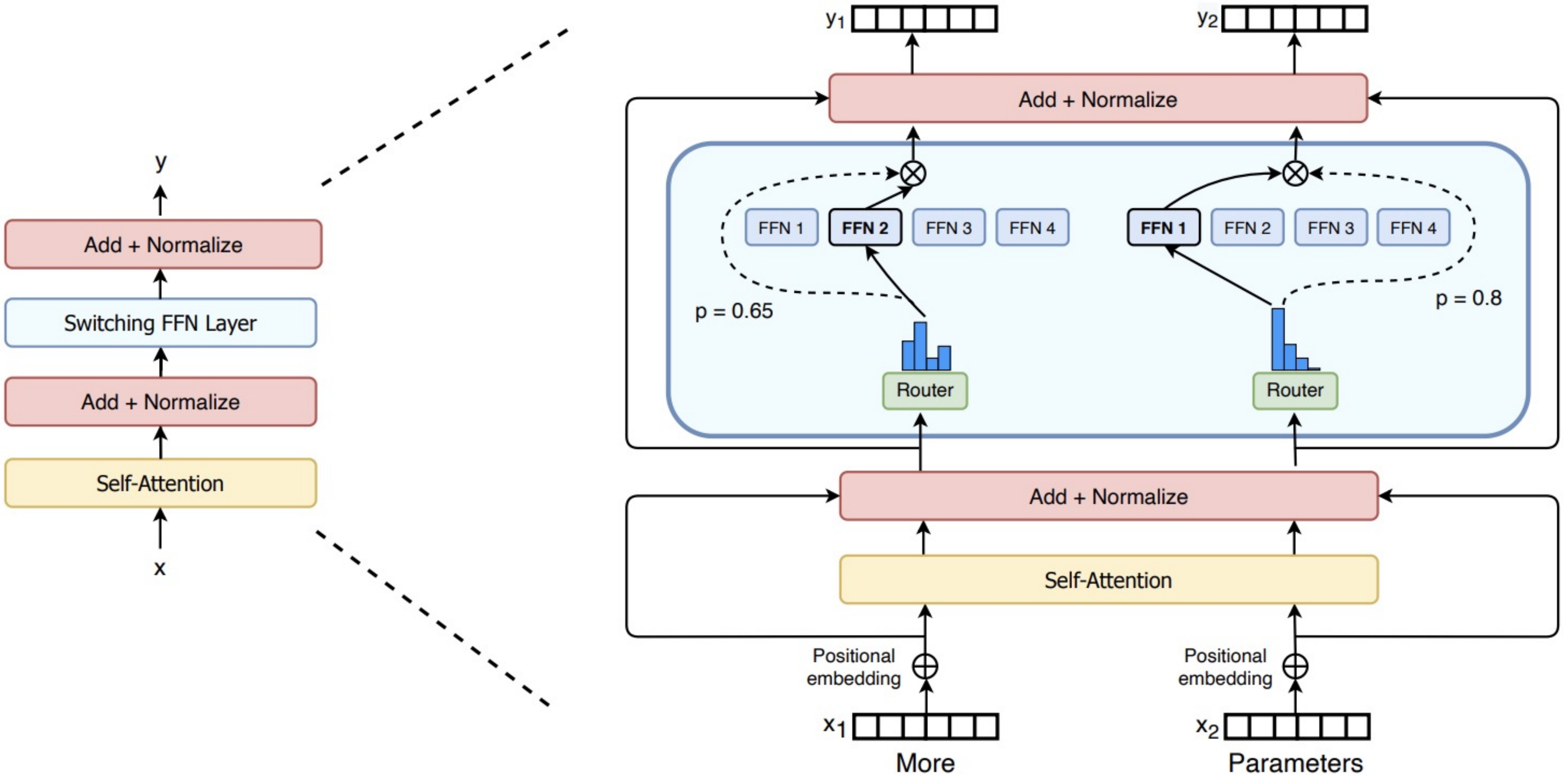
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transformers
- Graph Neural Networks
- **Mixture-of-Experts**

Mixture-of-Experts

- **Key idea**: make each expert focus on predicting the right answer for a subset of cases



Switch Transformers = Transformers + Mixture of Experts

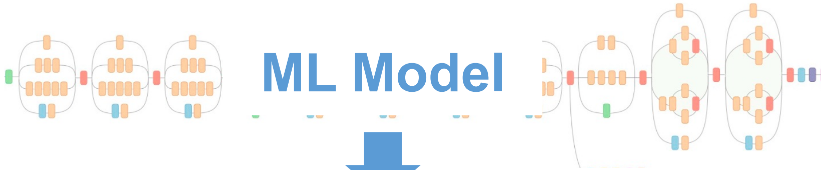


Recap: An Overview of Deep Learning Models

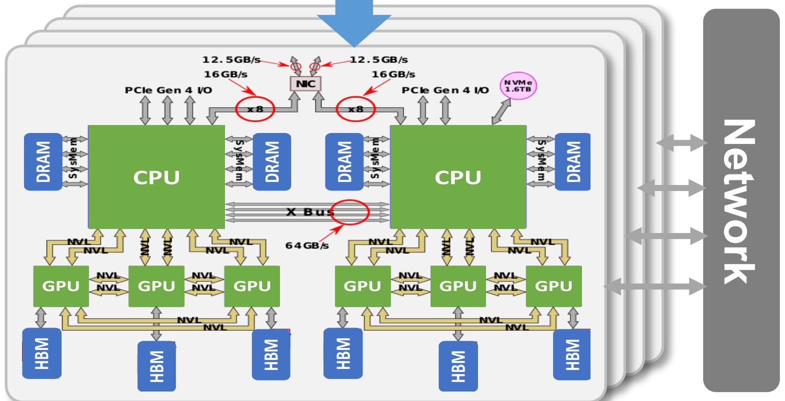
- Convolutional neural networks: various computer vision tasks
- Recurrent neural networks: processing sequences
- Transformers: efficient natural language processing
- Graph neural networks: deep learning on relational data
- Mixture of experts: ensemble deep learning

- A key takeaway: DNN techniques are not applied in isolation. Solving real-world problems require ``clever'' integration of DNN techniques

Next Lecture: Deep Learning Systems



- Automatic Differentiation
- Graph-Level Optimization
- Parallelization / Distributed Training
- Data Layout and Placement
- Kernel Optimizations
- Memory Optimizations



We will learn the current design and key techniques of each stack in ML systems