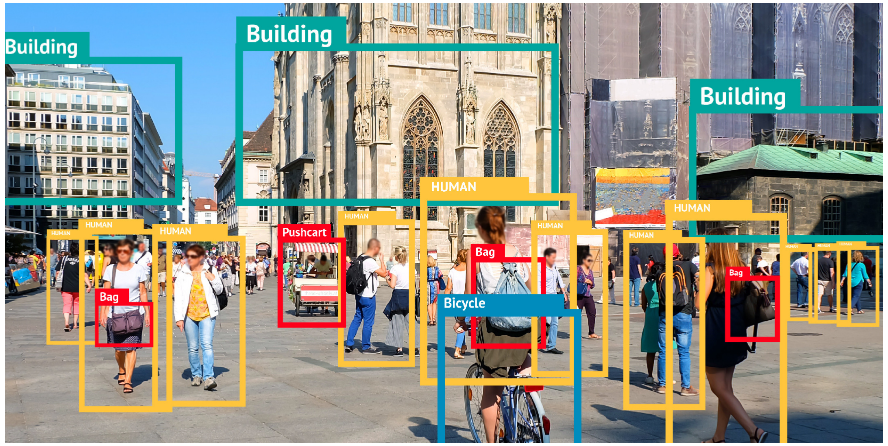
Lecture 23:
Parallel Deep Learning Basics

Parallel Computer Architecture and Programming

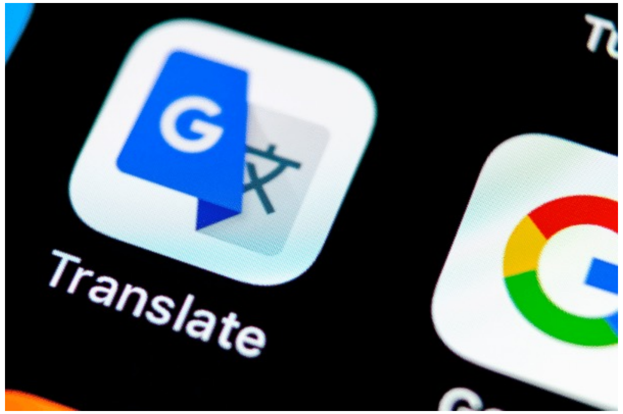
CMU 15-418/15-618, Fall 2023

The Success of Machine Learning Today

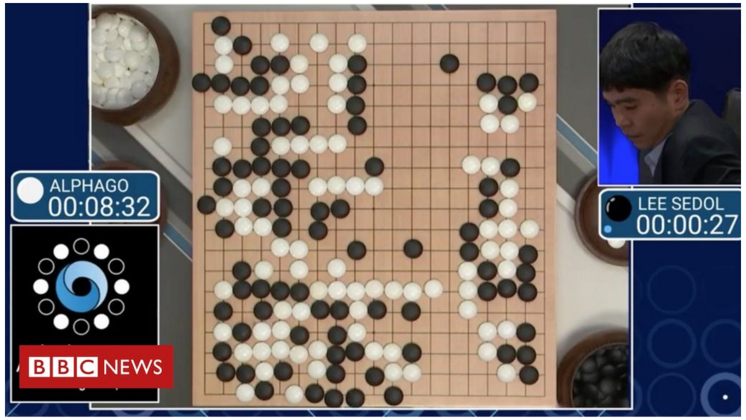


Object detection

Autonomous vehicles

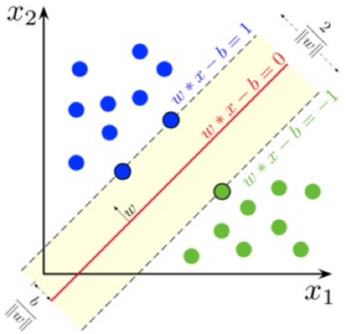


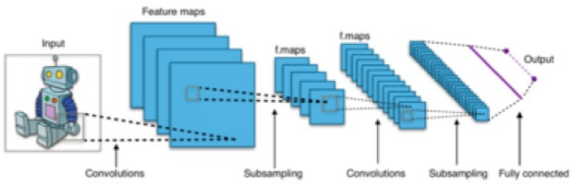
Machine translation

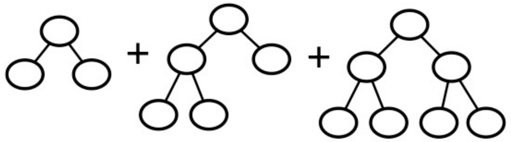


Game playing

Most ML techniques invented in 1980s and 1990s





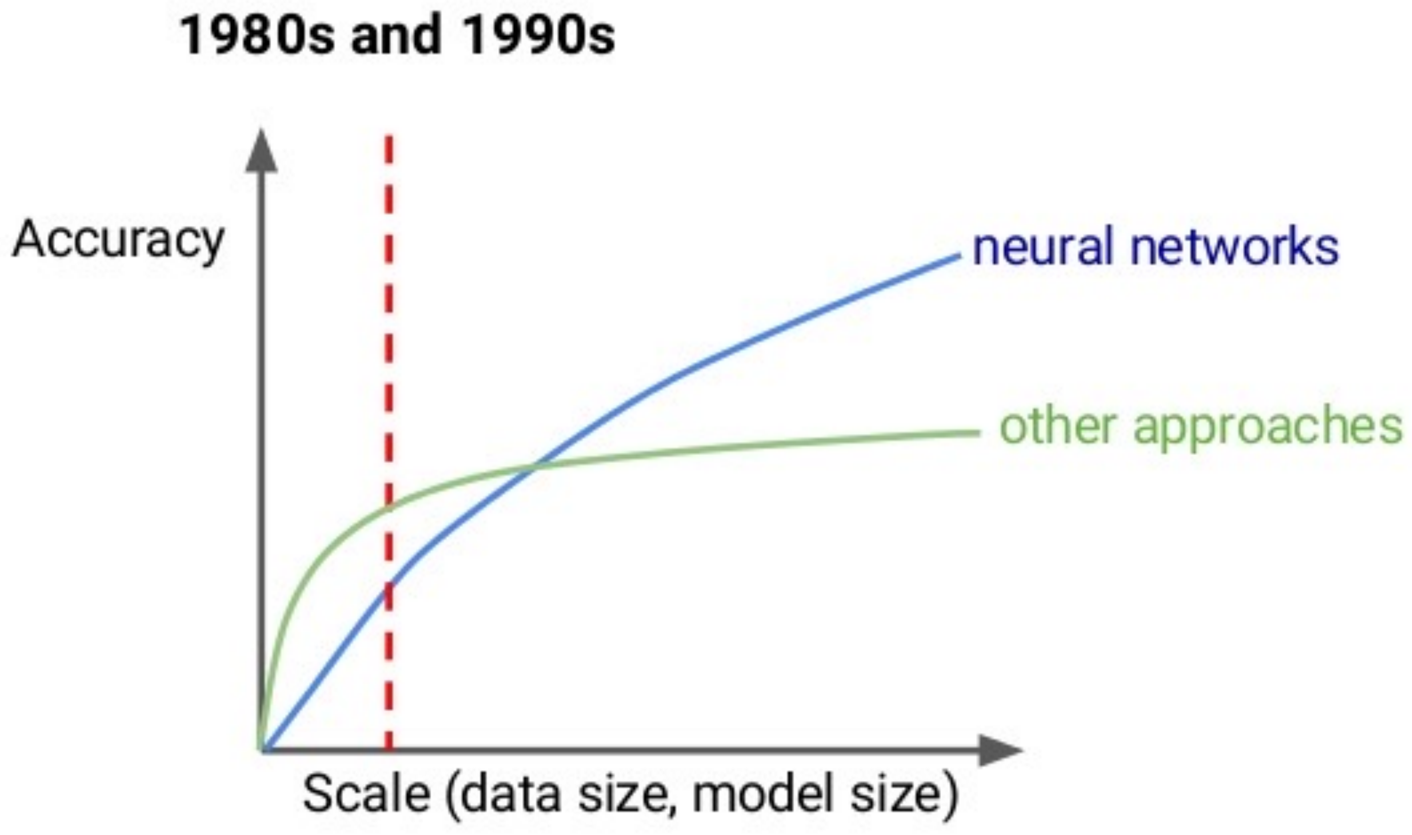


Perceptron Algorithm	Backprop	Support Vector Machine (SVM)	ConvNet	Gradient Boosting Machine (GBM)
1958	1986	1992	1998	1999

Why didn't the success of ML happen in 1990s?

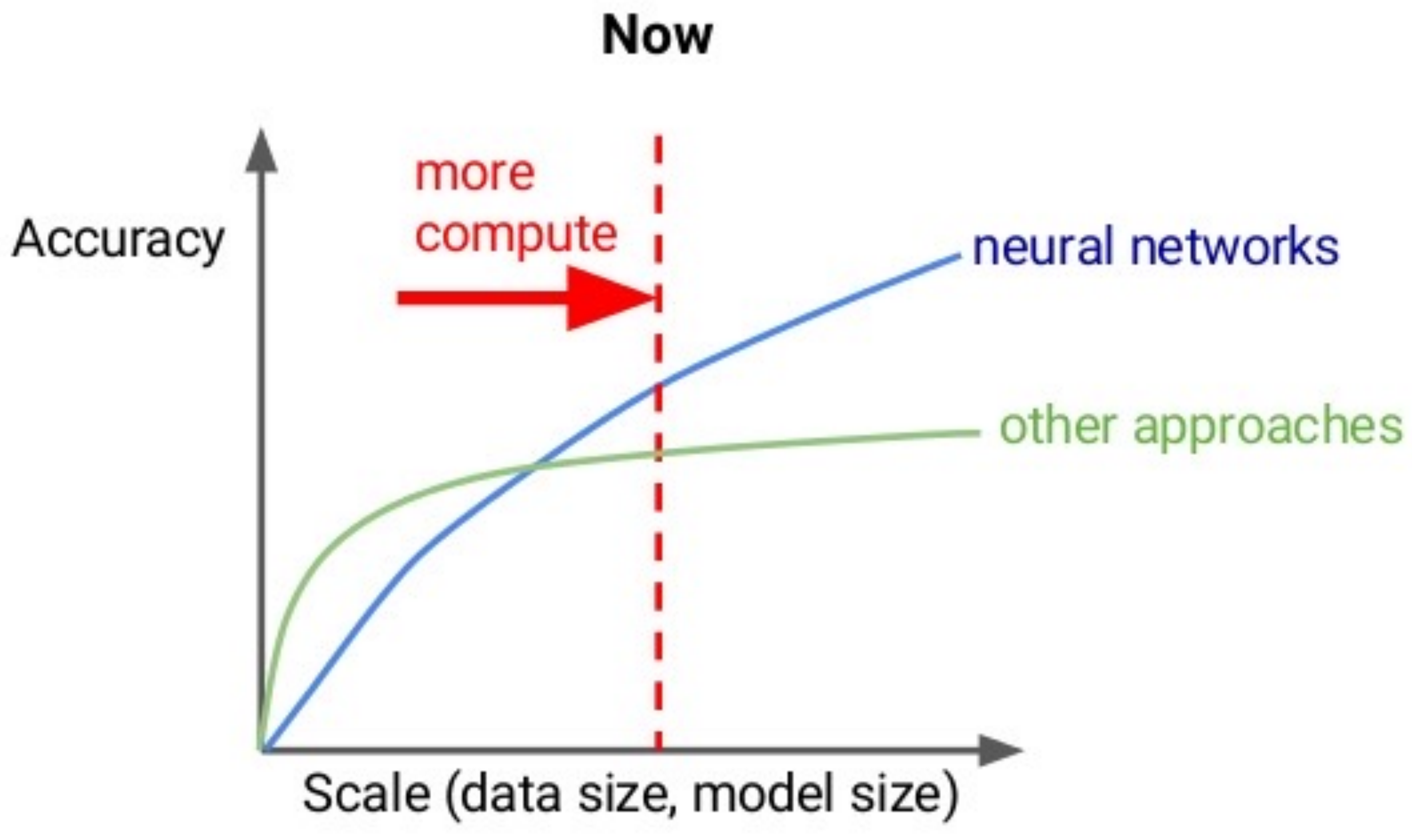
Adapted from TQ's slide

The Rise of ML and Neural Networks



Adapted from Jeff Dean, HotChips 2017

The Rise of ML and Neural Networks

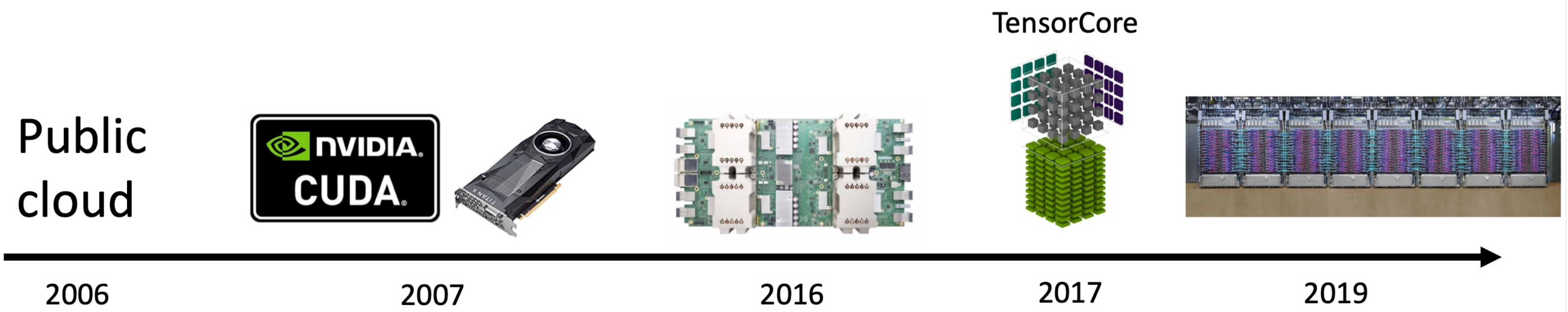


Adapted from Jeff Dean, HotChips 2017

Big data arrives in early 2000

Large-scale training datasets become available

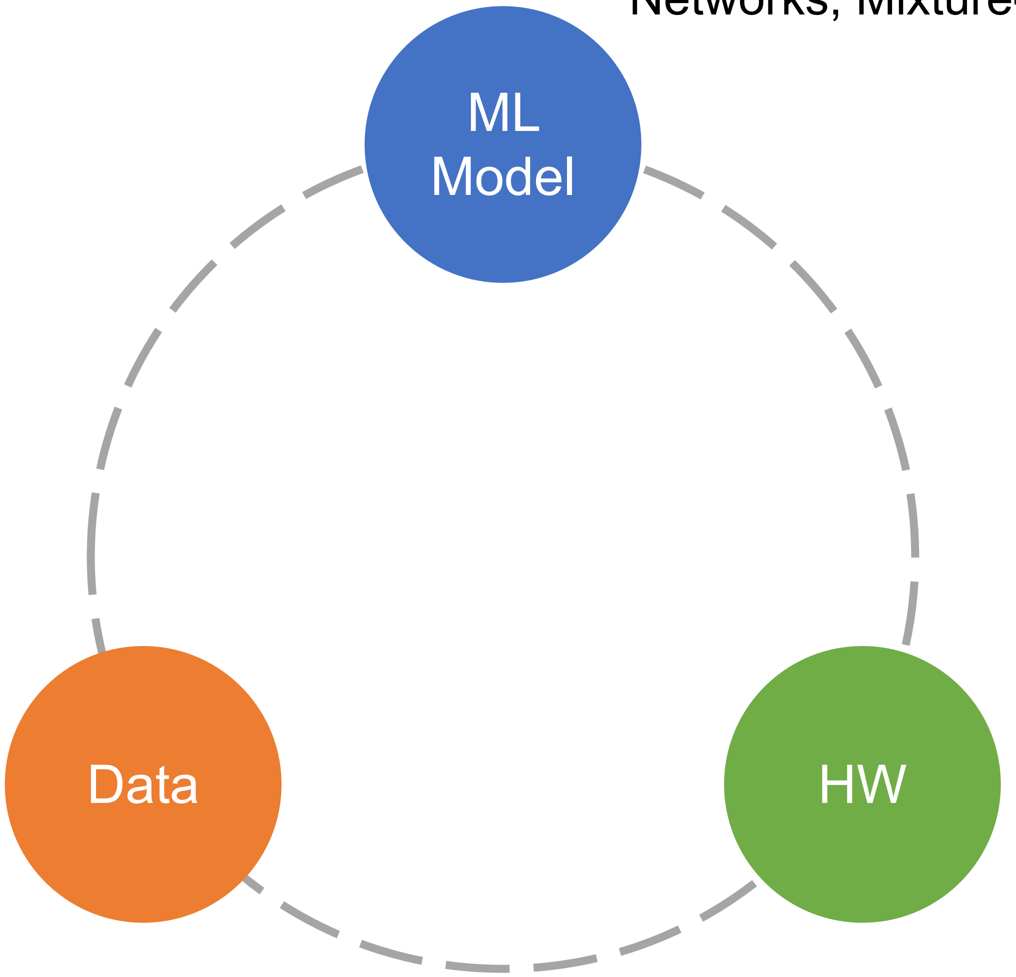
AI hardware becomes widely available in 2010s



Distributed Heterogeneous Hardware Platforms

The Secret Ingredients in ML Success

ResNet, Transformers, Graph Neural Networks, Mixture-of-Experts, …



ImageNet, Kaggle, Flickr, NetFlix, …

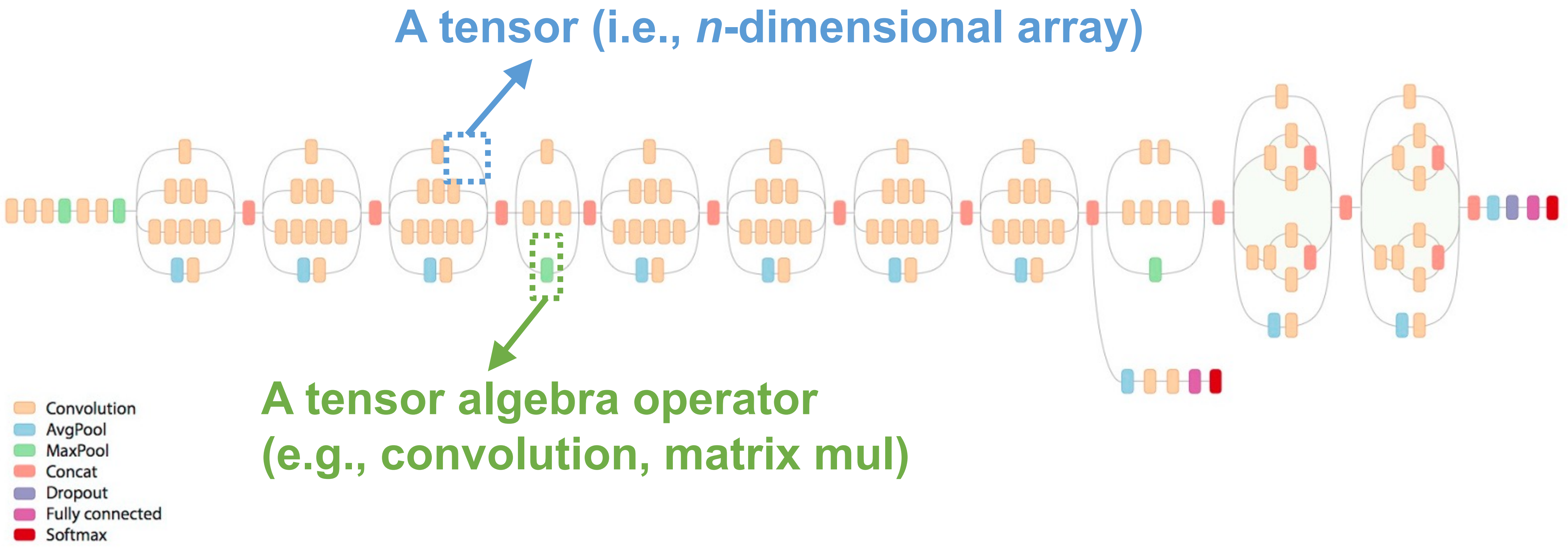
GPUs, TPUs, Tensor Cores, Supercomputers, Graph Cores, …

Today’s Topics

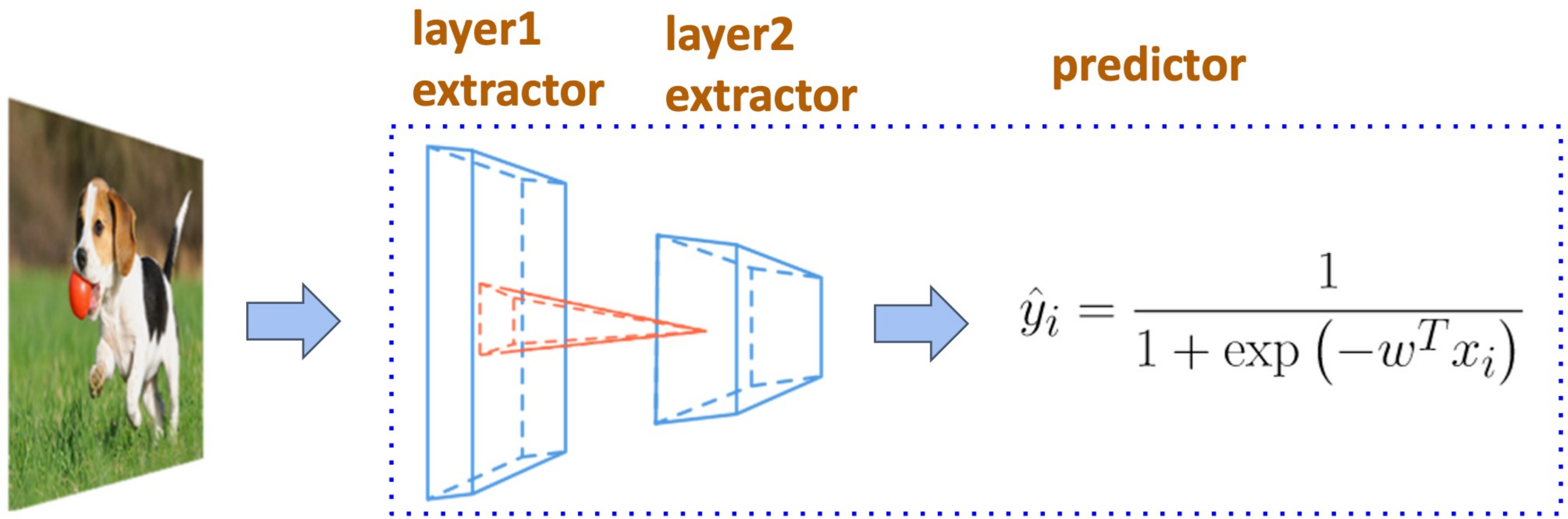
- Stochastic Gradient Descent
- Backpropagation and Automatic Differentiation
- An Overview of Deep Neural Networks

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

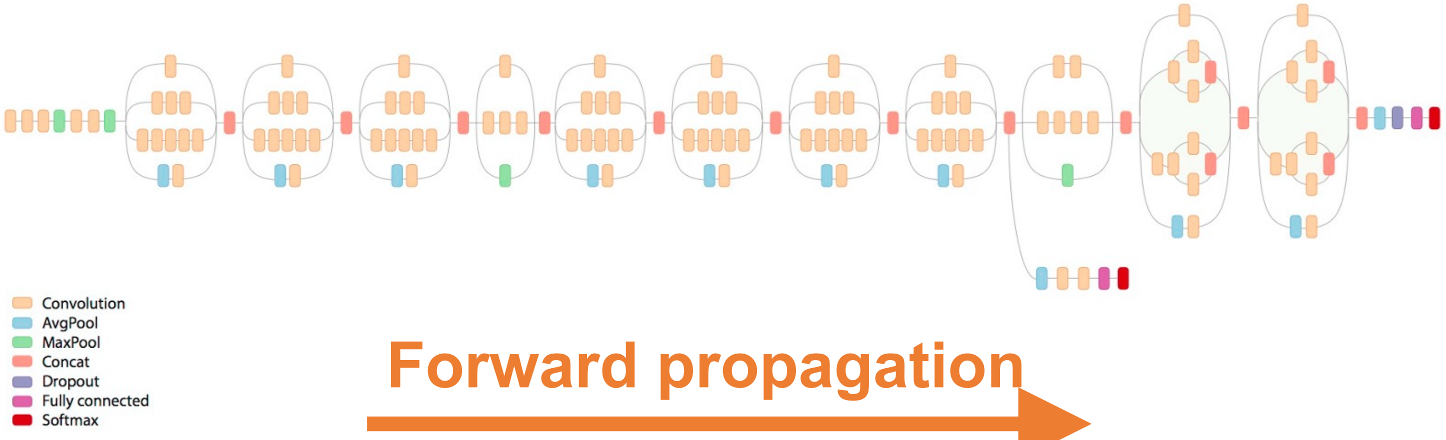
Training $$w \leftarrow w - \eta \nabla_w L(w)$$

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

Model inputs



Model prediction

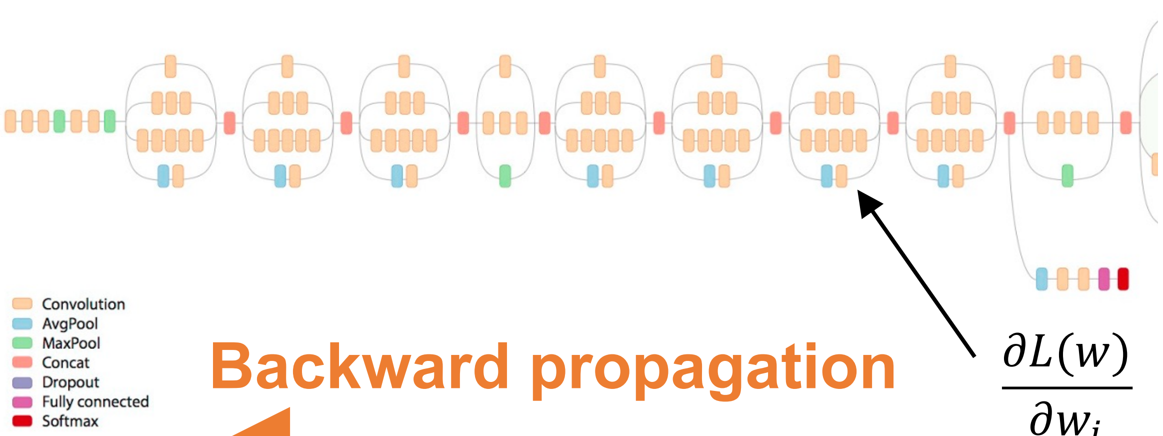
Forward propagation

Gradient Descent (GD)

Train ML models through many iterations of 3 stages

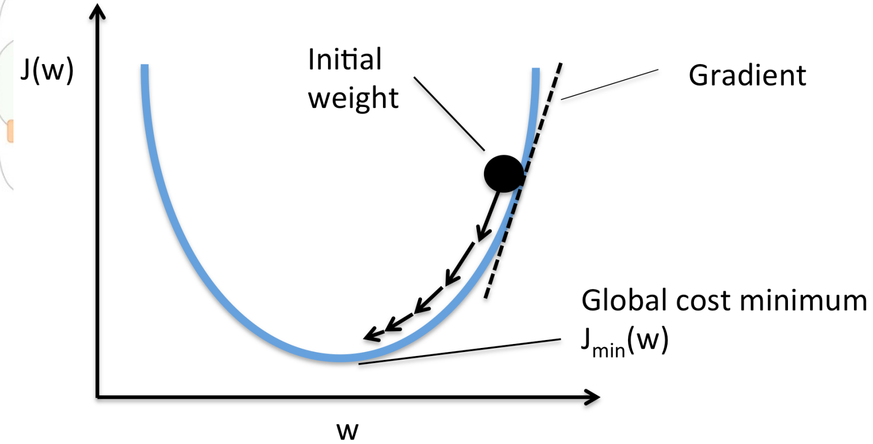
1. **Forward propagation**: apply model to a batch of input samples and run calculation through operators to produce a prediction
2. **Backward propagation**: run the model in reverse to produce a gradient for each trainable weight
3. **Weight update**: use the loss value to update model weights

Model inputs



Backward propagation

$$\frac{\partial L(w)}{\partial w_i}$$



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

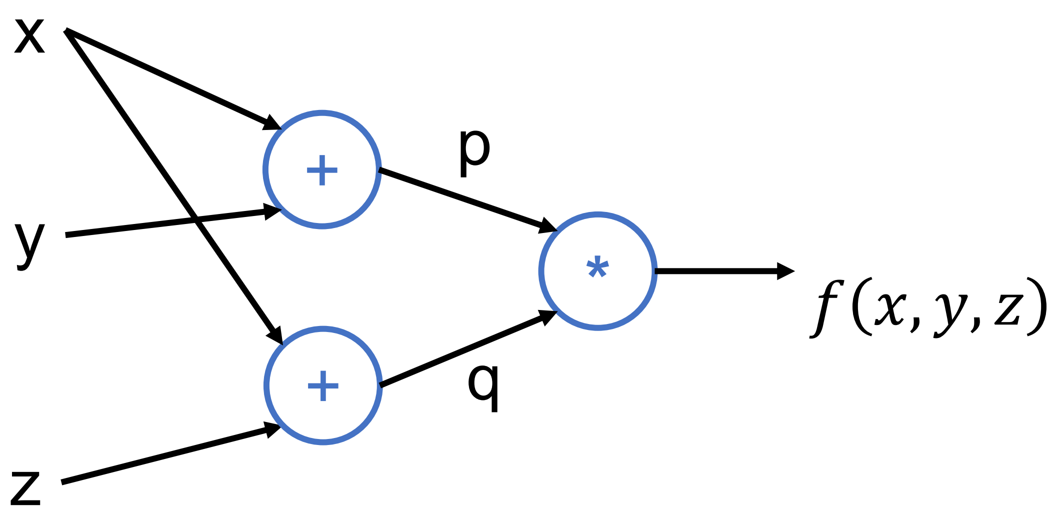
$$\frac{d(f(x)g(x))}{dx} = \frac{df(x)}{dx}g(x) + \frac{dg(x)}{dx}f(x)$$

- Chain rule

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

Backpropagation: a simple example

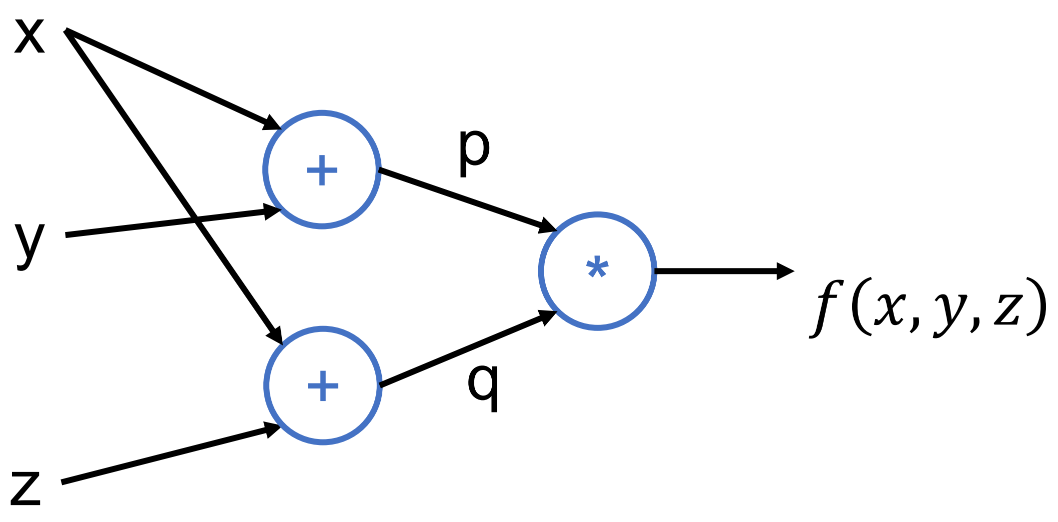
- $f(x, y, z) = (x + y)(x + z)$



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 $\frac{\partial f}{\partial x}$

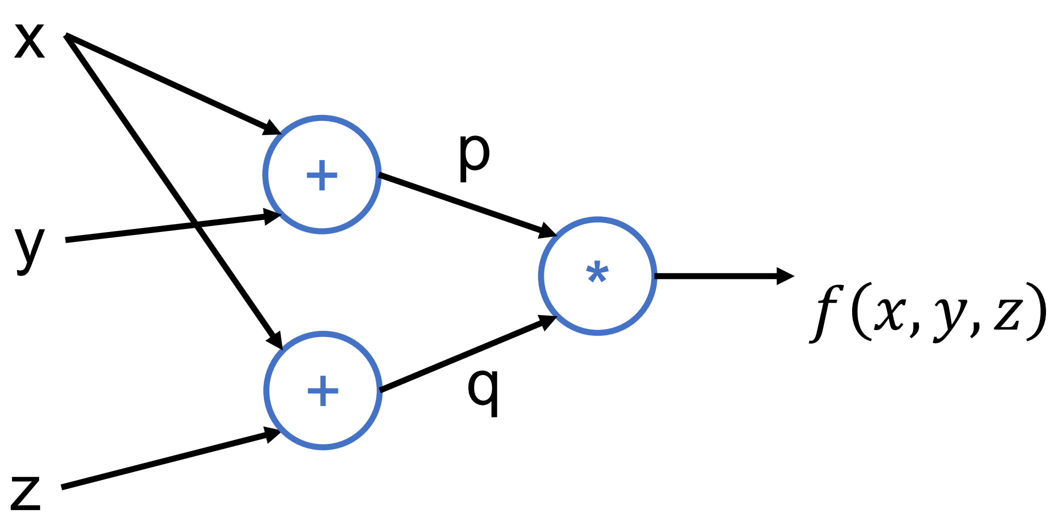
- $f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4

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

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

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

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



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

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

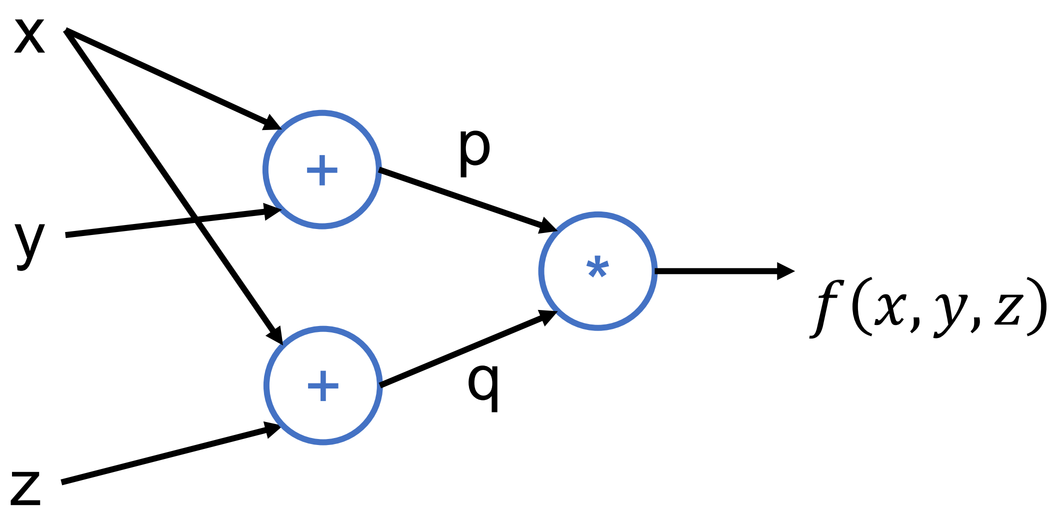
- $f(x,y,z) = (x+y)(x+z)$
- E.g., x = -2, y = 5, z = -4

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

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

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

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



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

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

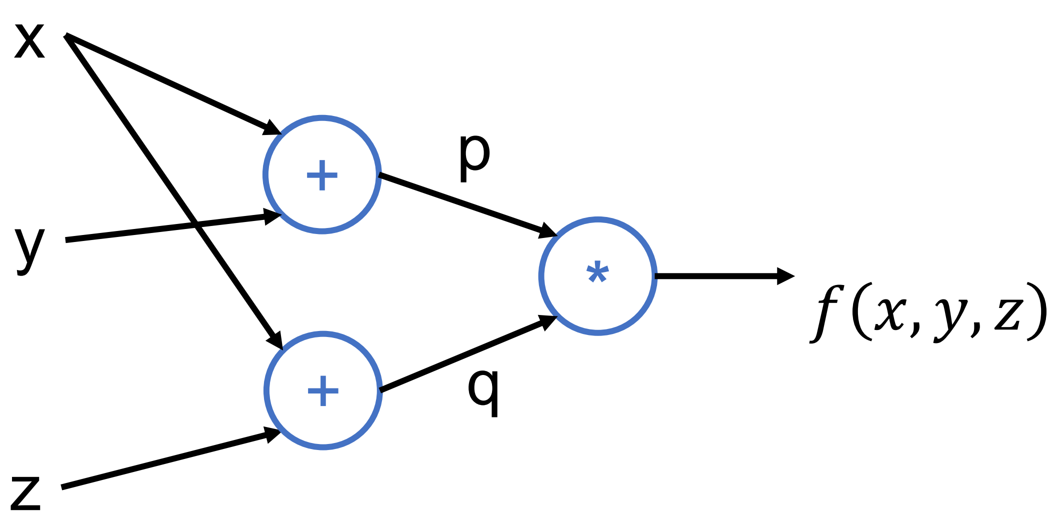
- $f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4

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

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

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

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



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)

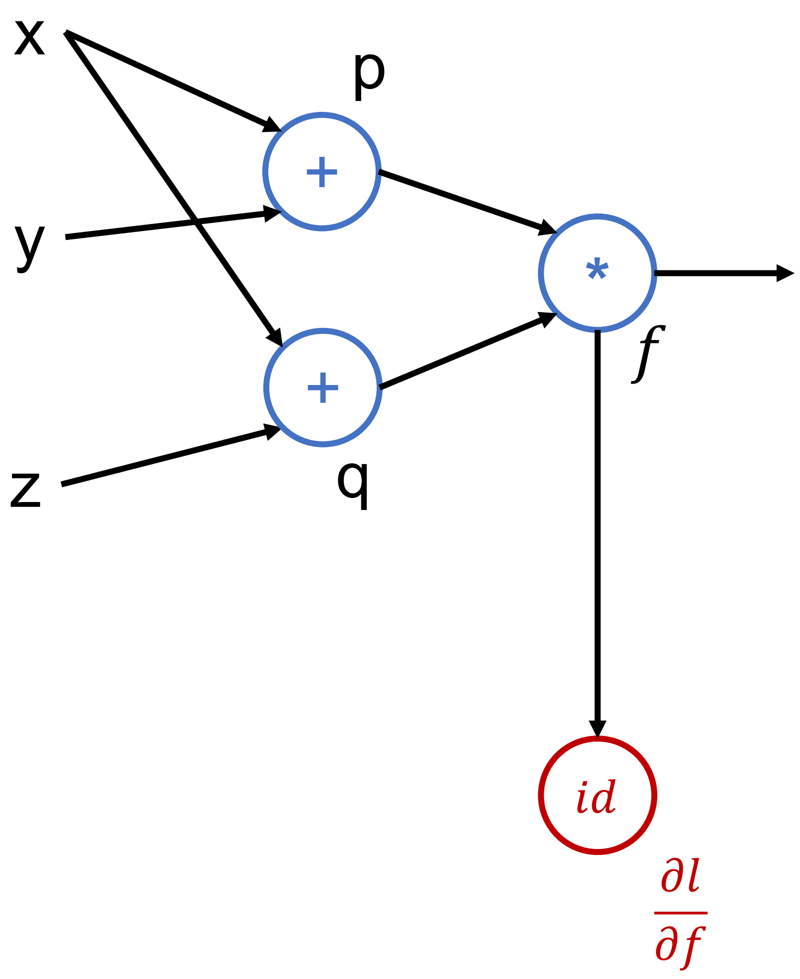
- Solution: reverse mode AutoDiff

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

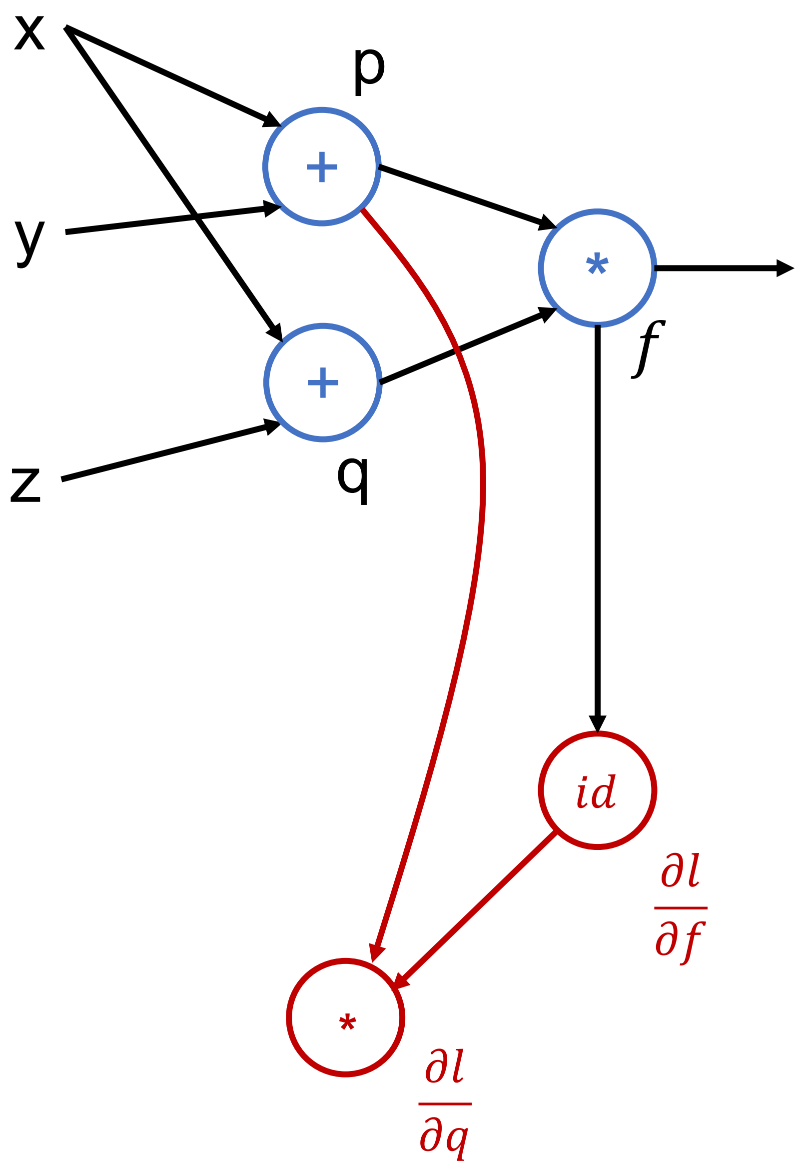
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

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



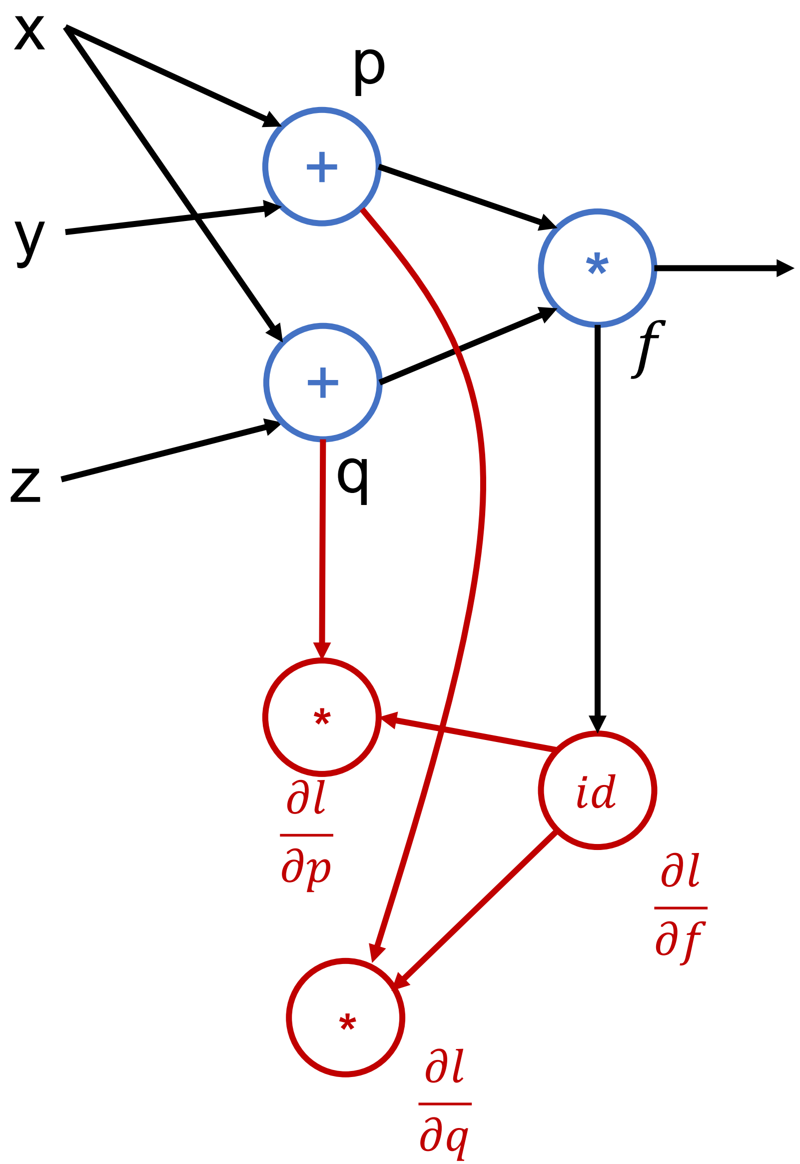
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

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



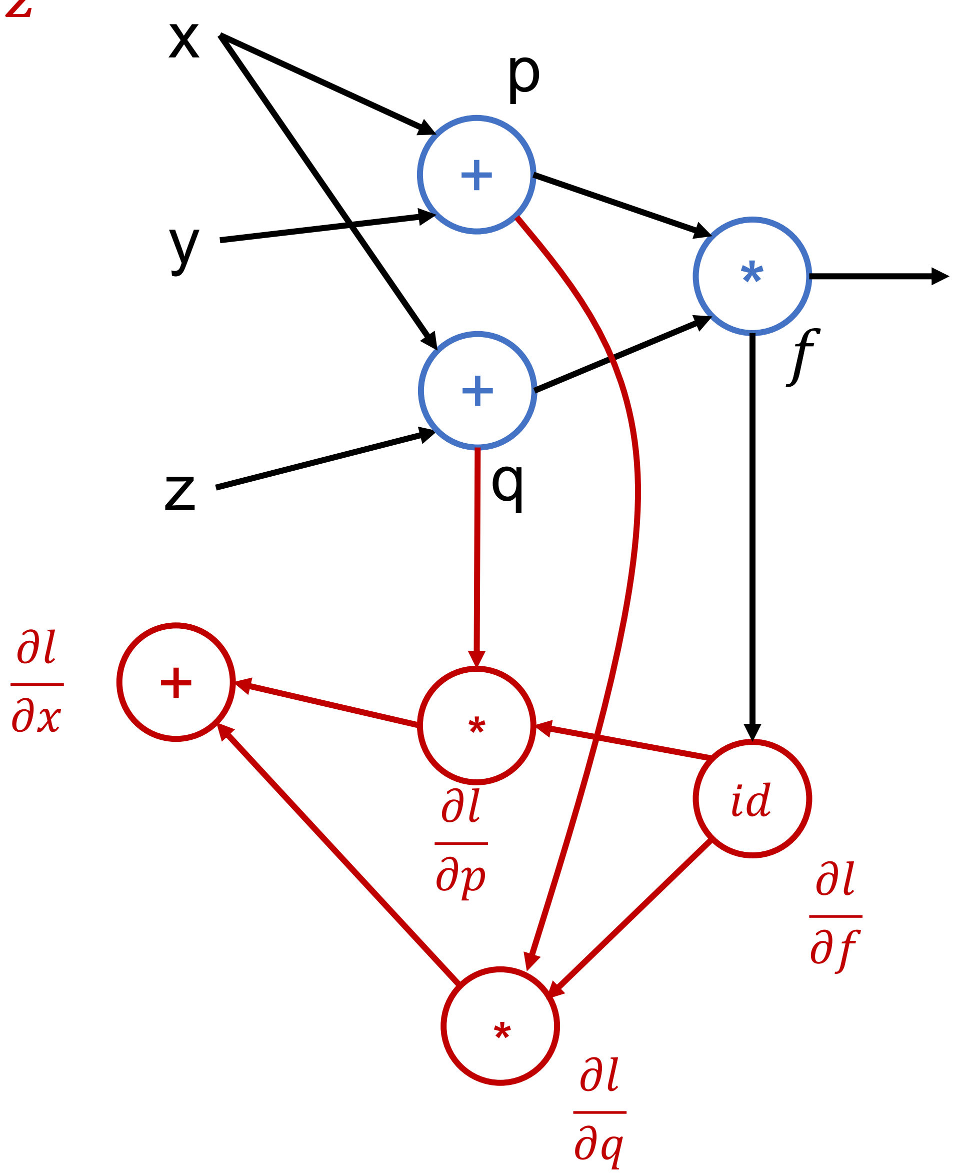
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

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



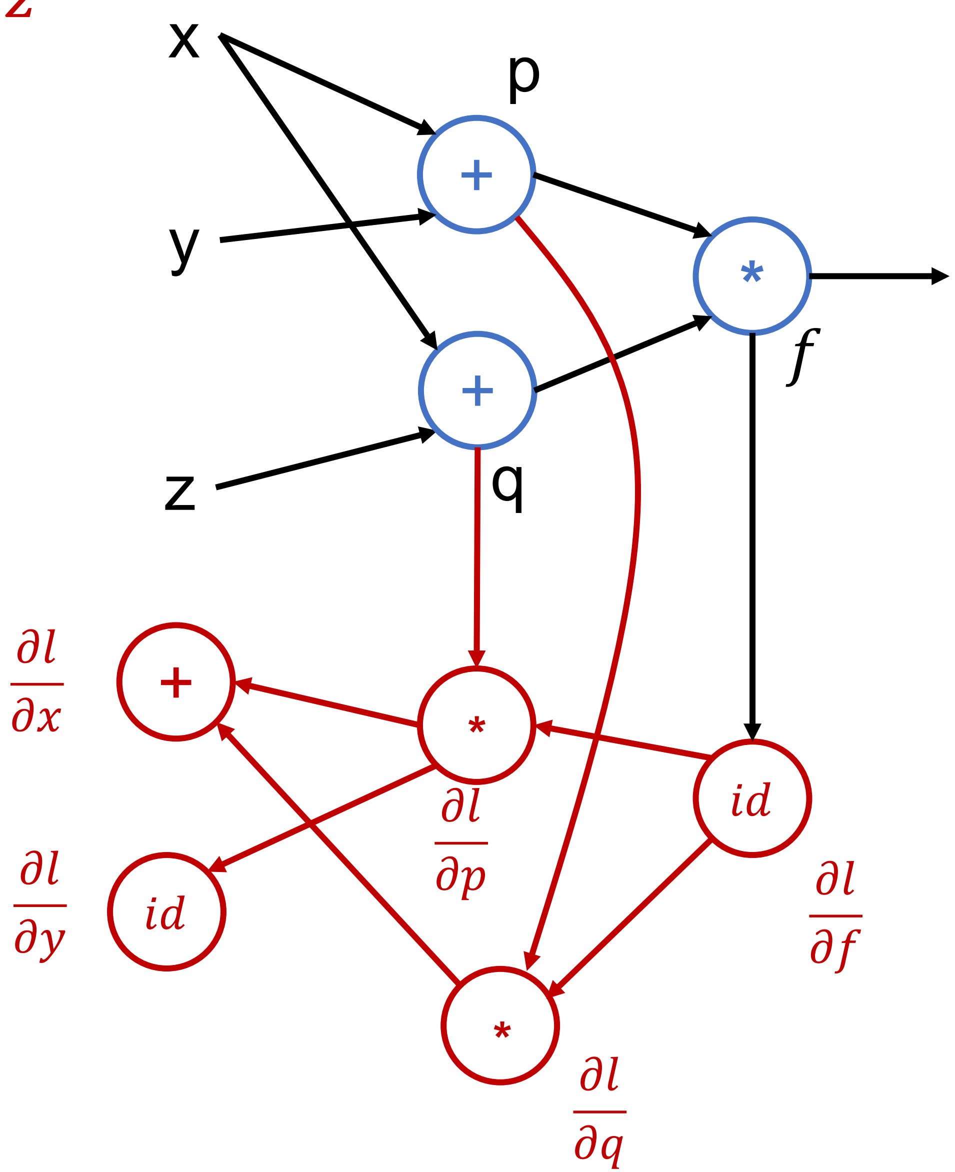
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

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



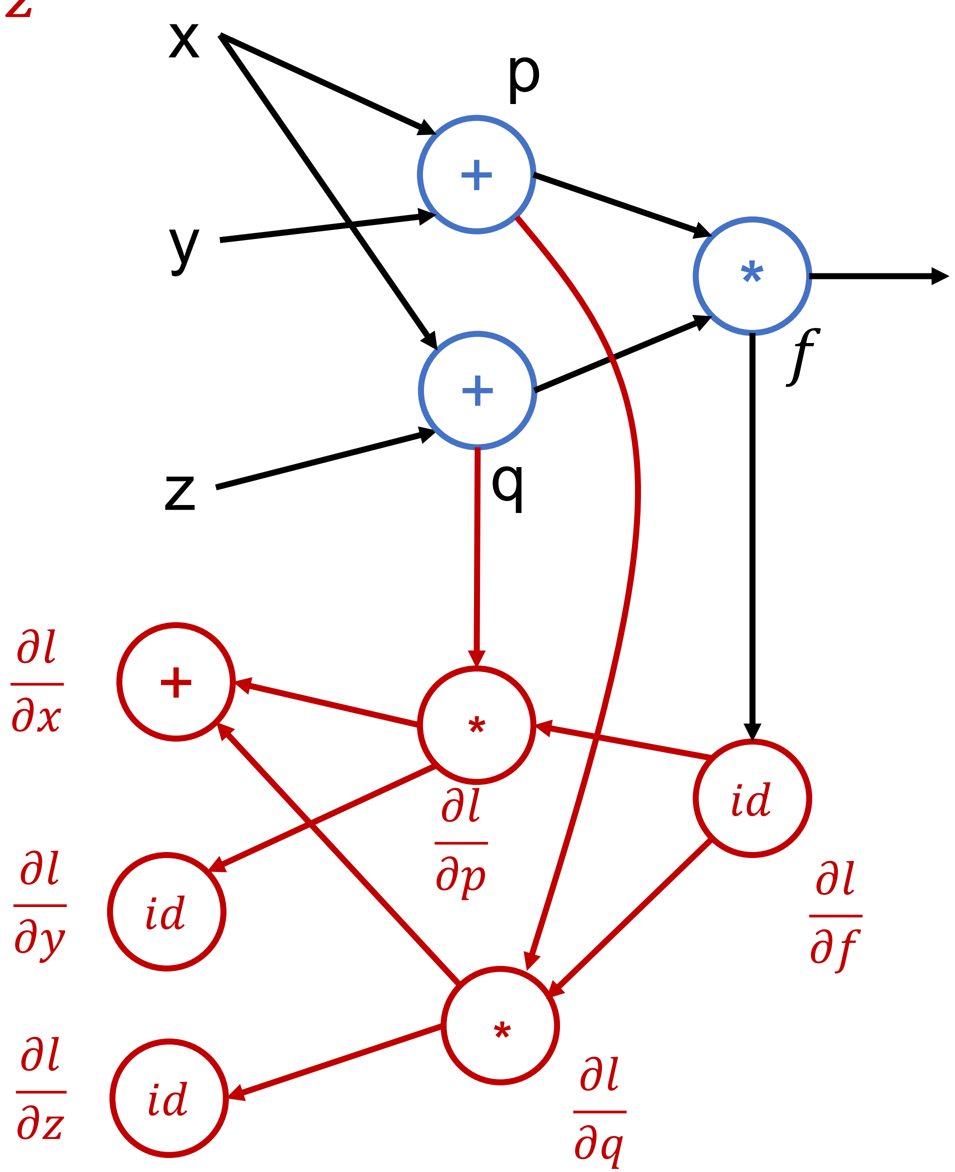
Exercise: Compute $\frac{\partial l}{\partial x}, \frac{\partial l}{\partial y}, \frac{\partial l}{\partial z},$

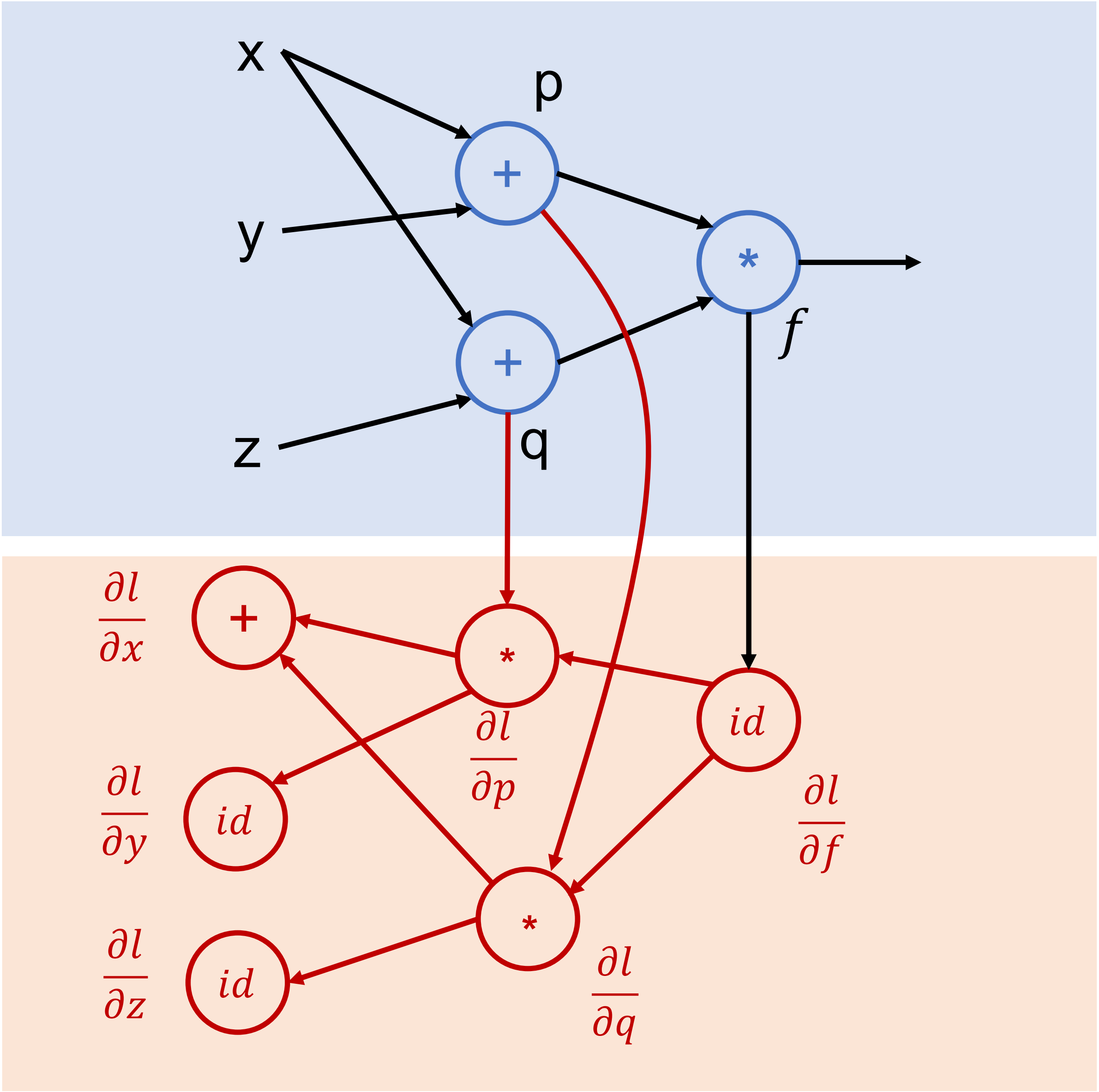
- $l = f(x, y, z) = (x + y)(x + z)$
- E.g., x = -2, y = 5, z = -4
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = 1 \times p = 3$
- $\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f} \times \frac{\partial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



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
- $\frac{\partial l}{\partial f} = 1$
- $\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial q} = 1 \times p = 3$
- $\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f} \times \frac{\partial f}{\partial p} = 1 \times q = -6$
- $\frac{\partial l}{\partial x} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial x} + \frac{\partial l}{\partial q} \times \frac{\partial q}{\partial x} = -3$
- $\frac{\partial l}{\partial y} = \frac{\partial l}{\partial p} \times \frac{\partial p}{\partial y} = -6 \times 1 = -6$
- $\frac{\partial l}{\partial z} = \frac{\partial l}{\partial q} \times \frac{\partial q}{\partial z} = 3 \times 1 = 3$



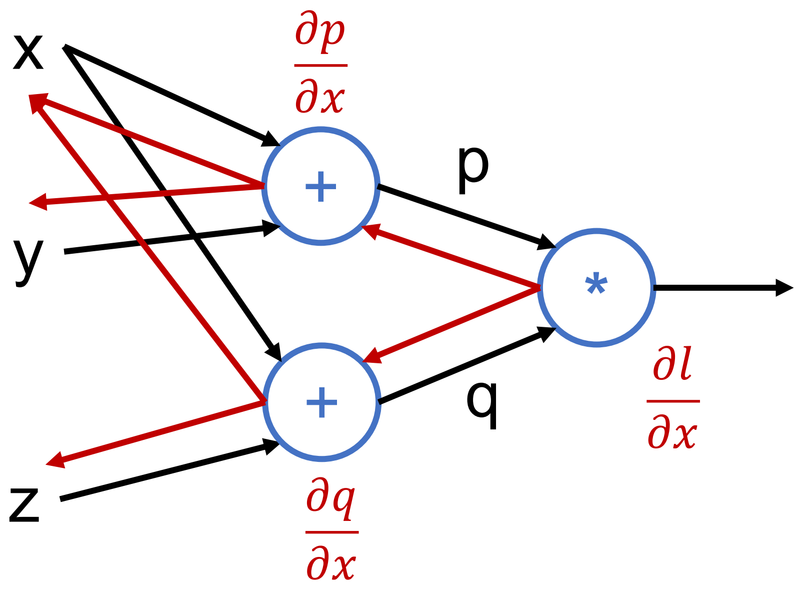


Forward computation graph

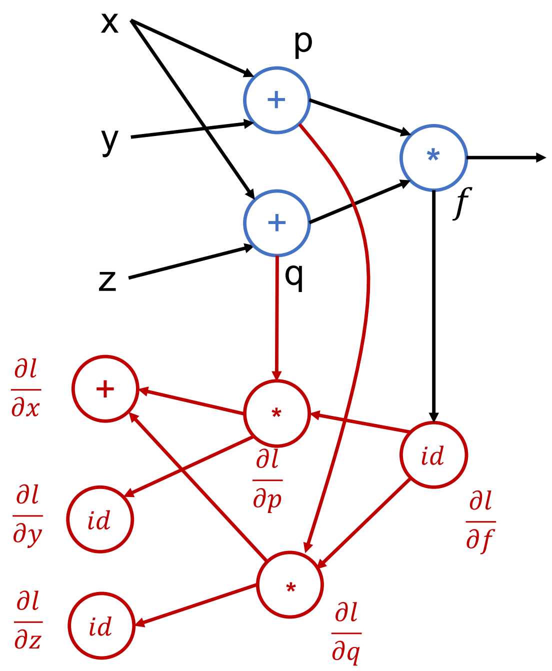
Backward computation graph

Discuss: Backpropagation v.s. Reverse AutoDiff

Backpropagation



Reverse AutoDiff

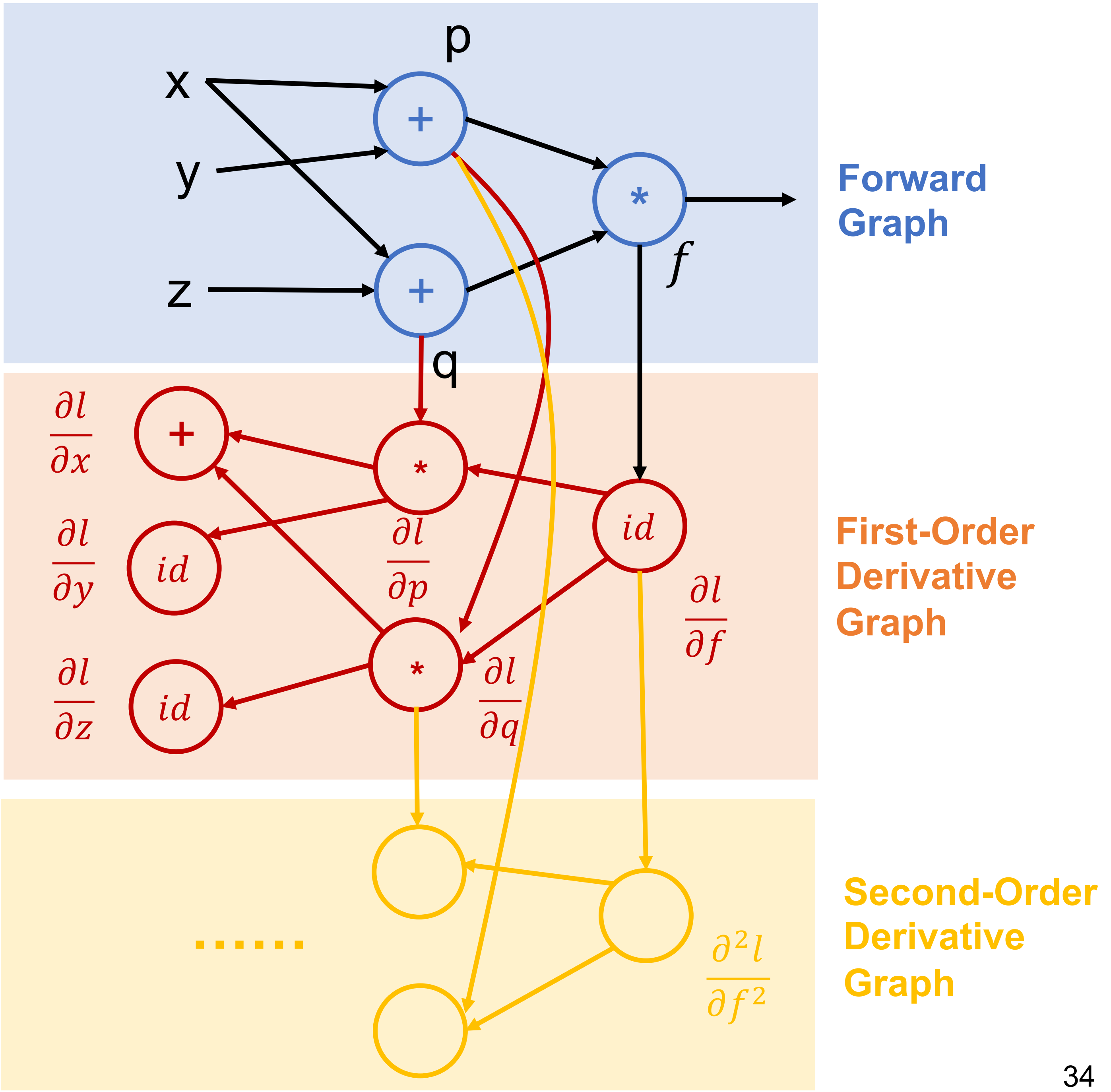


What is the difference between backpropagation and reverse AutoDiff?

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

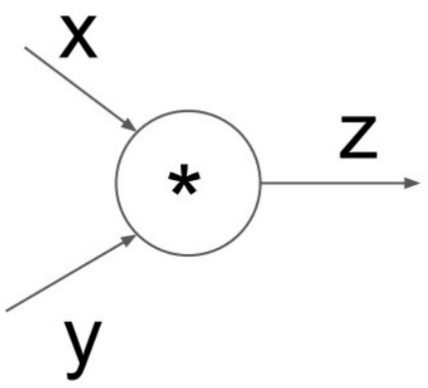
Higher Order Derivatives



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

Summary

We have learnt a core technique of deep learning 🎊

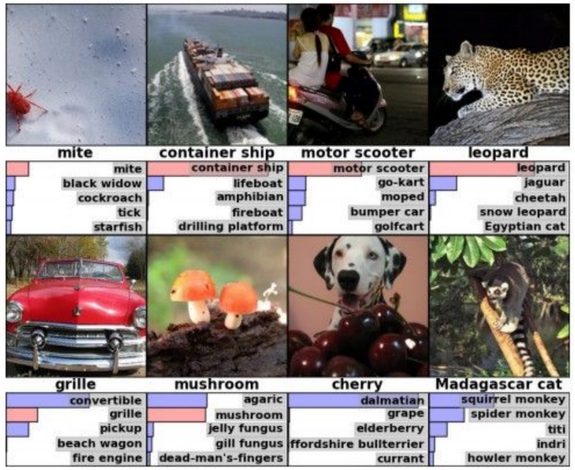
- **Forward pass**: apply model to a batch of input samples and run calculation through operators and save intermediate results
- **Backward pass**: run the model in reverse and apply chain rule to compute gradients
- **Weight update**: use the gradients to update model weights

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

Understand Our Applications:
An Overview of Deep Learning Models

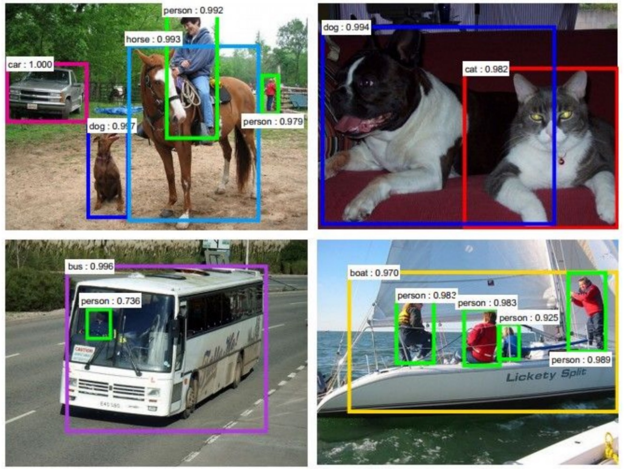
- **Convolutional Neural Networks**
- **Recurrent Neural Networks**
- **Transformers**
- **Graph Neural Networks**
- **Mixture-of-Experts**

CNNs are widely used in vision tasks

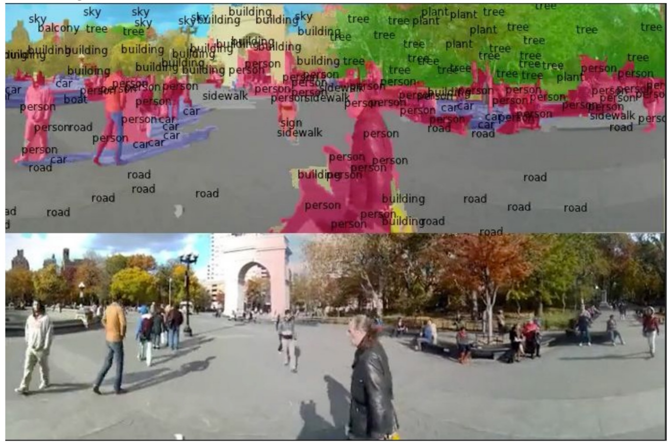


Classification

Retrieval



Detection



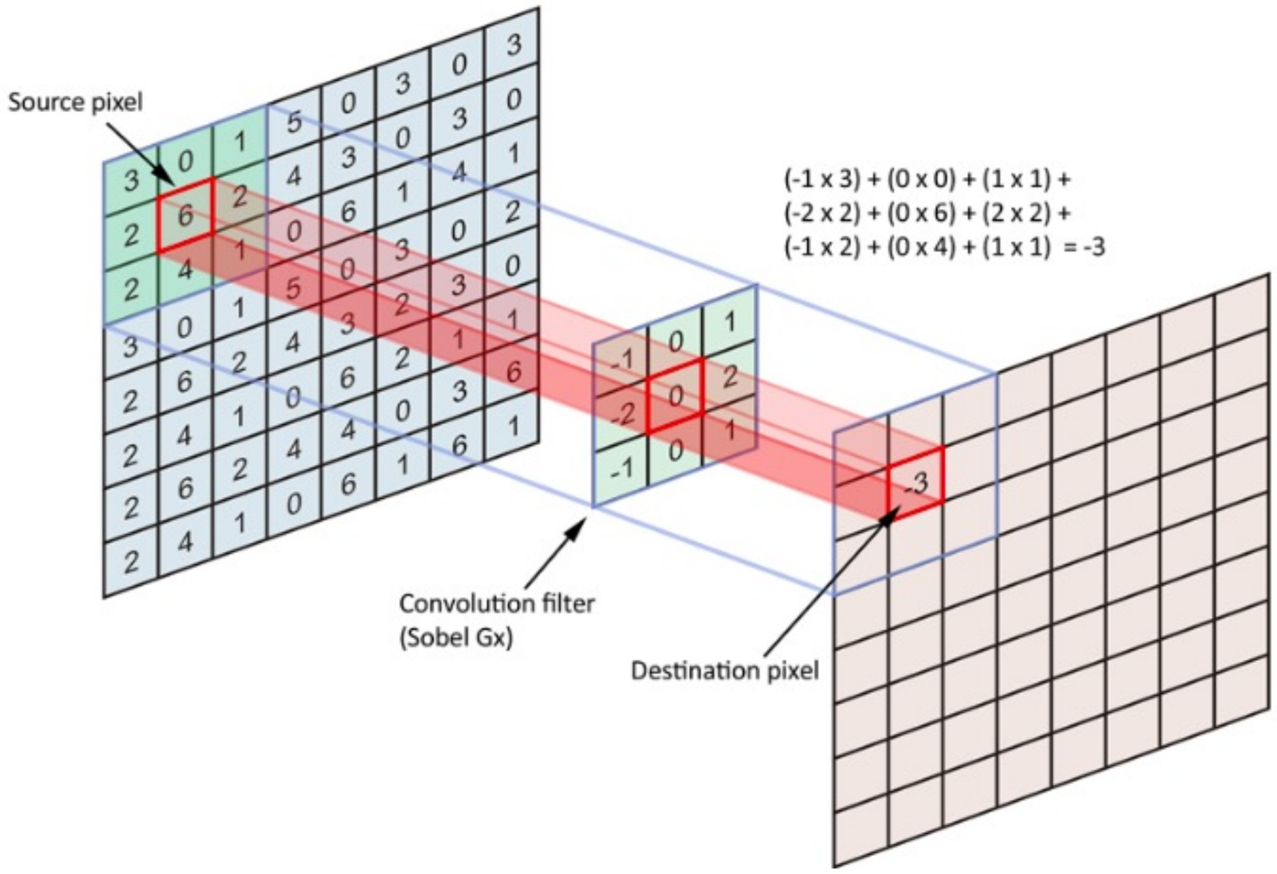
Segmentation

Self-Driving

Synthesis

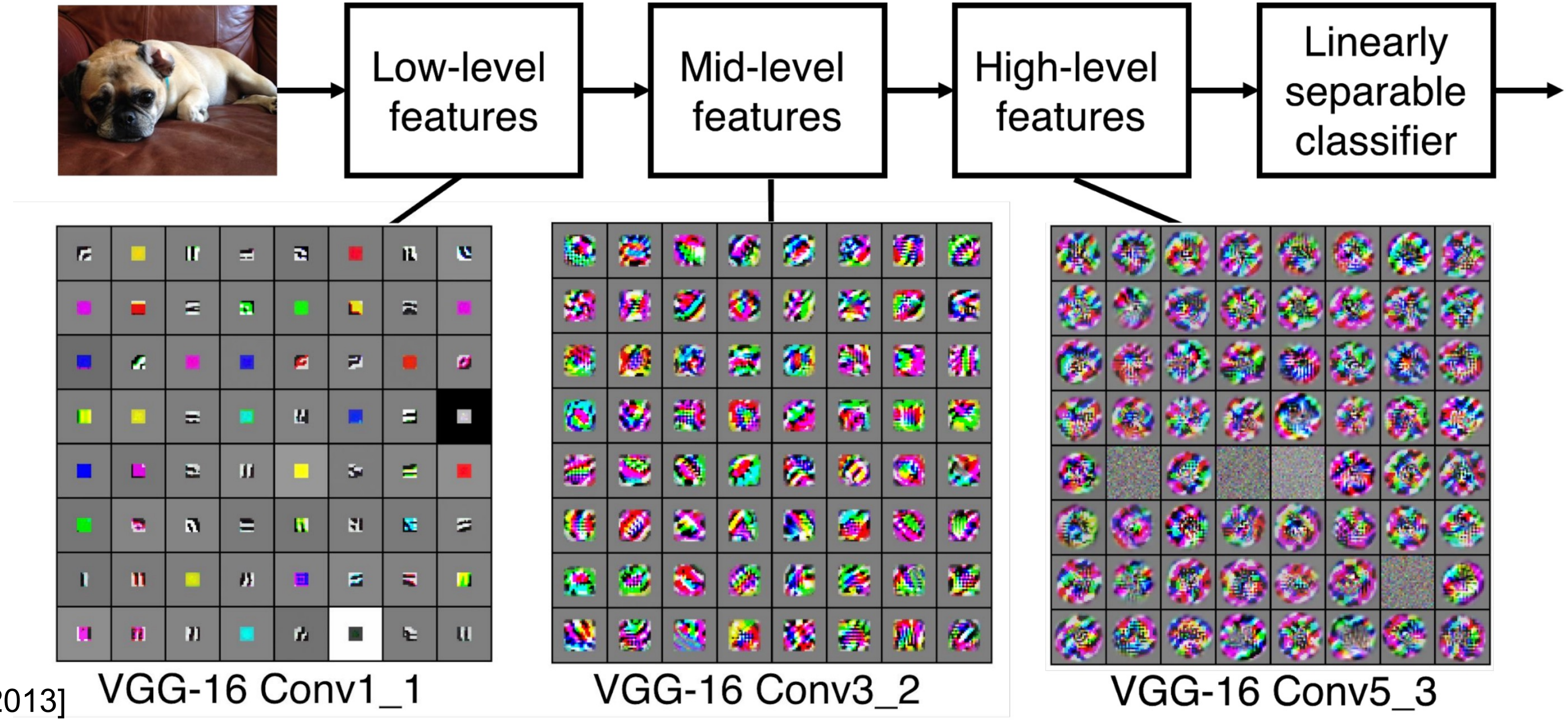
Recap: Convolution

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



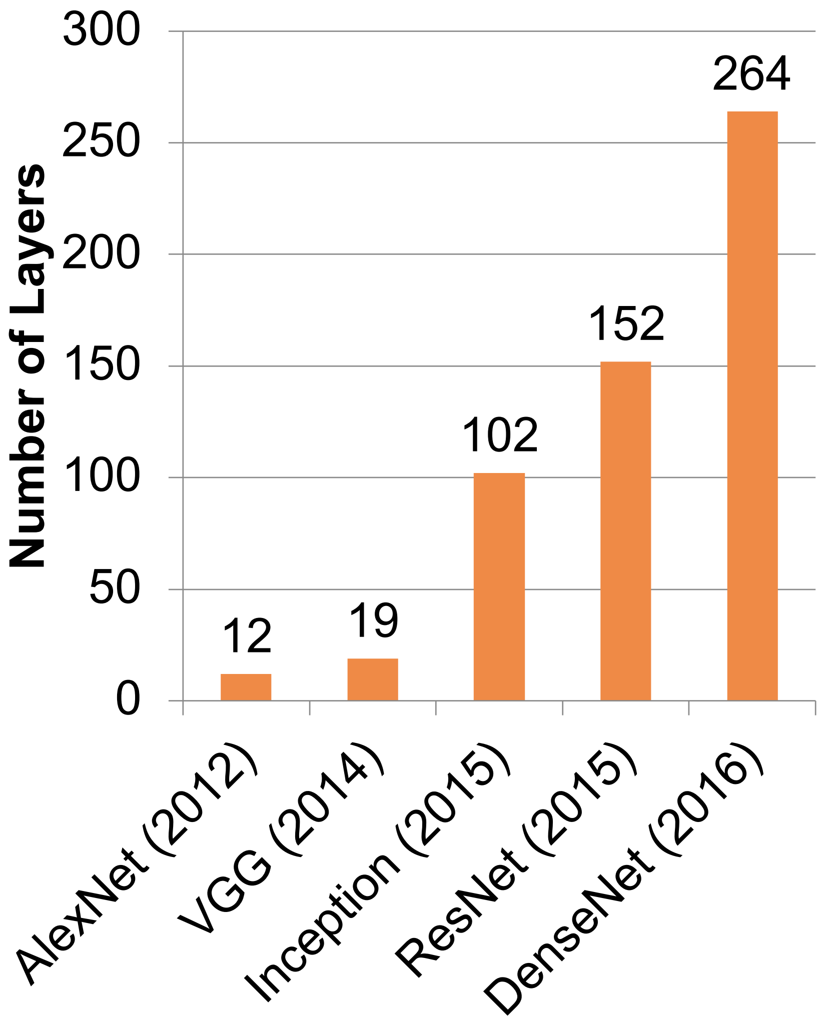
CNNs

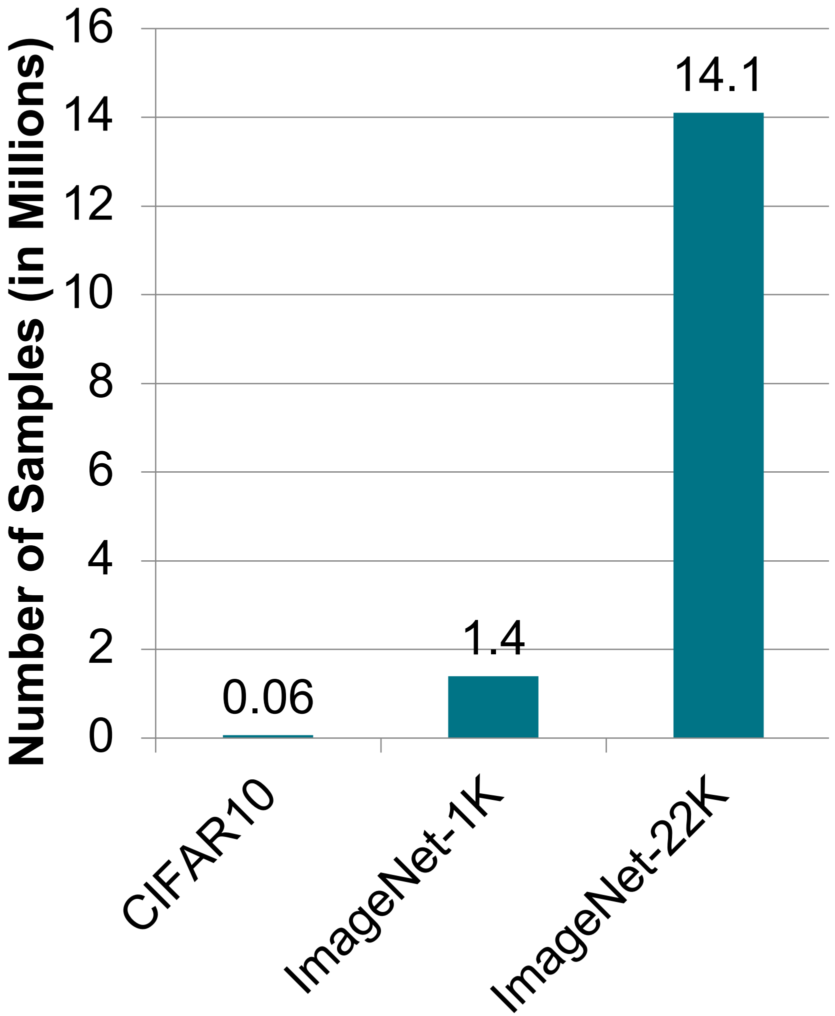
- A sequence of convolutional layers, interspersed by pooling, normalization, and activation functions



[Zeiler and Fergus 2013]

MLSys Challenges in CNNs: Increasing Computational Requirements





MLSys Challenges in CNNs: Increasing Computational Requirements

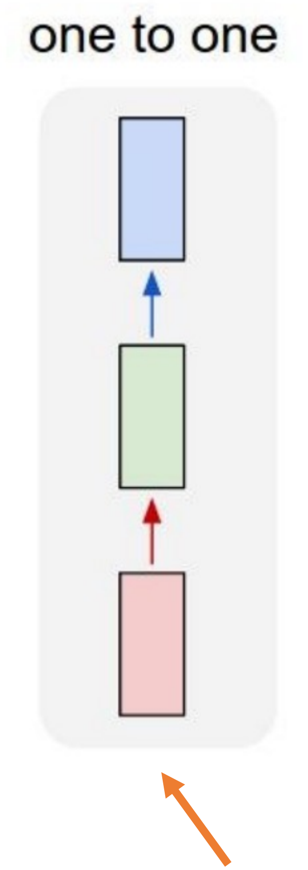
- **Computational cost**: convolutions are extremely compute-intensive
- **Memory requirement**: high-resolution images cannot fit in a single GPU

- **Solution**: parallelize training across GPUs
  - Lecture 24: Data Parallelism for Distributed Training
  - Lecture 25: Pipeline and Model Parallelism for Distributed Training

Understand Our Applications:
An Overview of Deep Learning Models

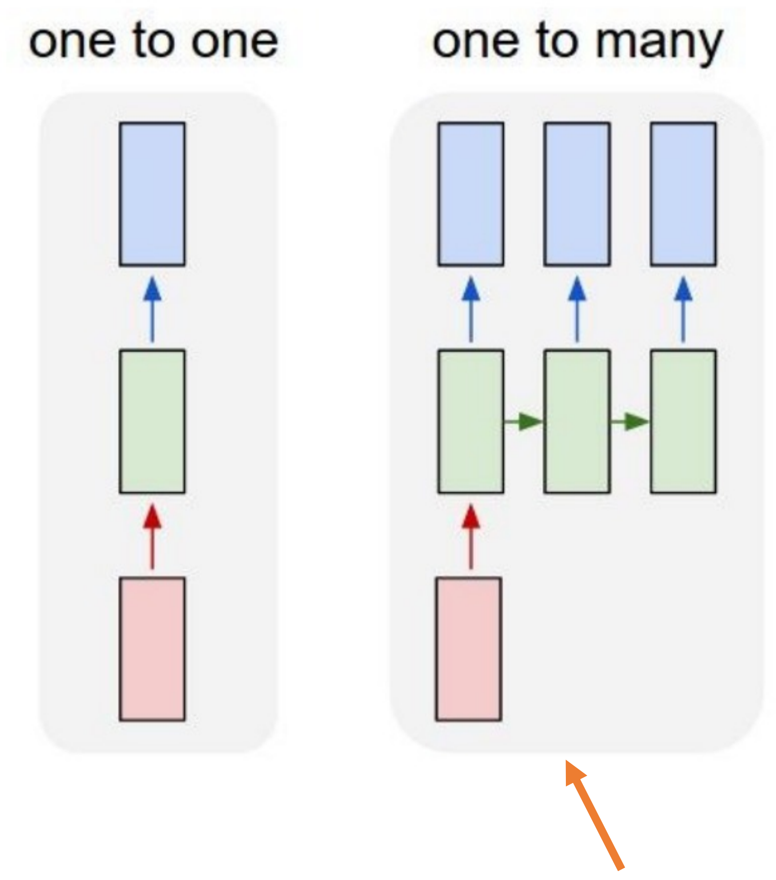
- Convolutional Neural Networks: vision tasks
- **Recurrent Neural Networks**
- Transformer
- Graph Neural Networks
- Mixture-of-Experts

Recurrent Neural Networks: Process Sequences



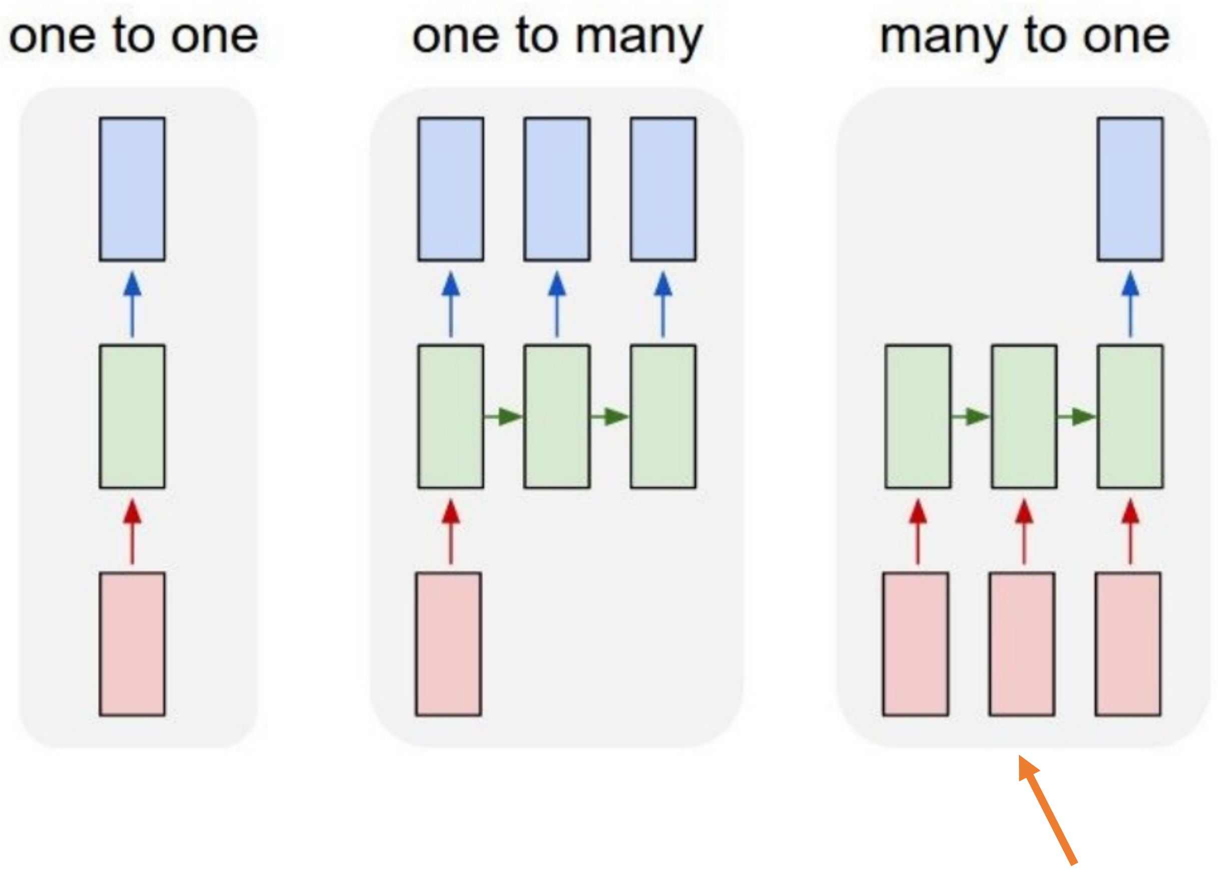
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



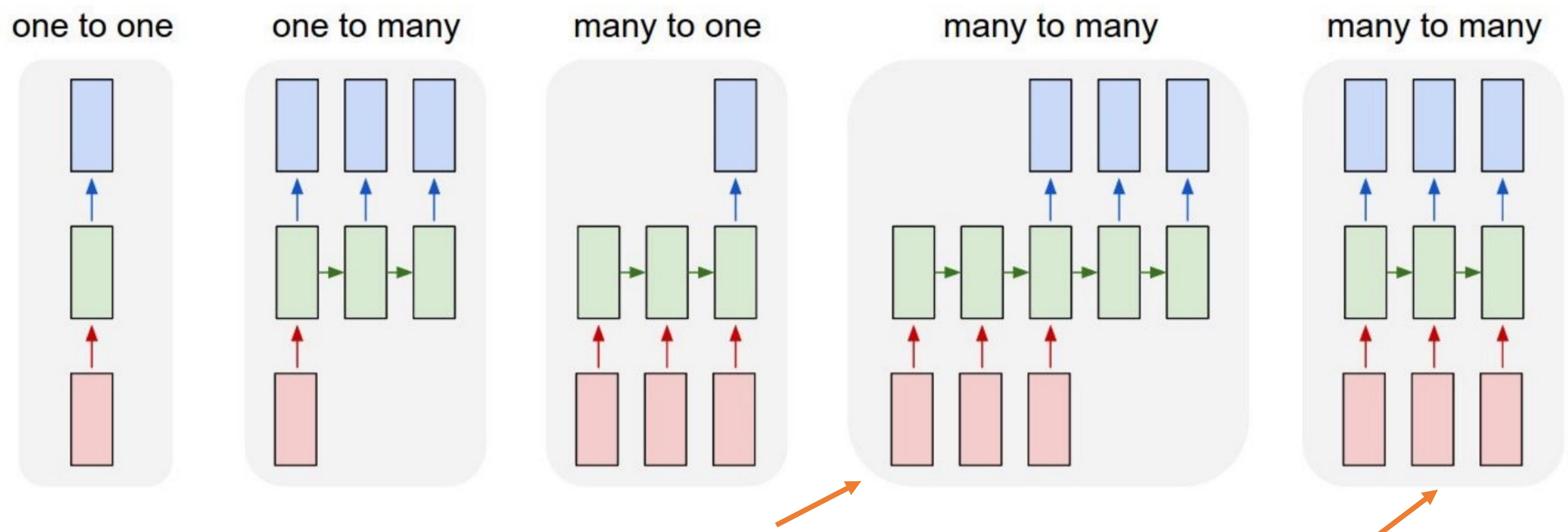
e.g., image captioning
Image -> sequence of words

Recurrent Neural Networks: Process Sequences



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

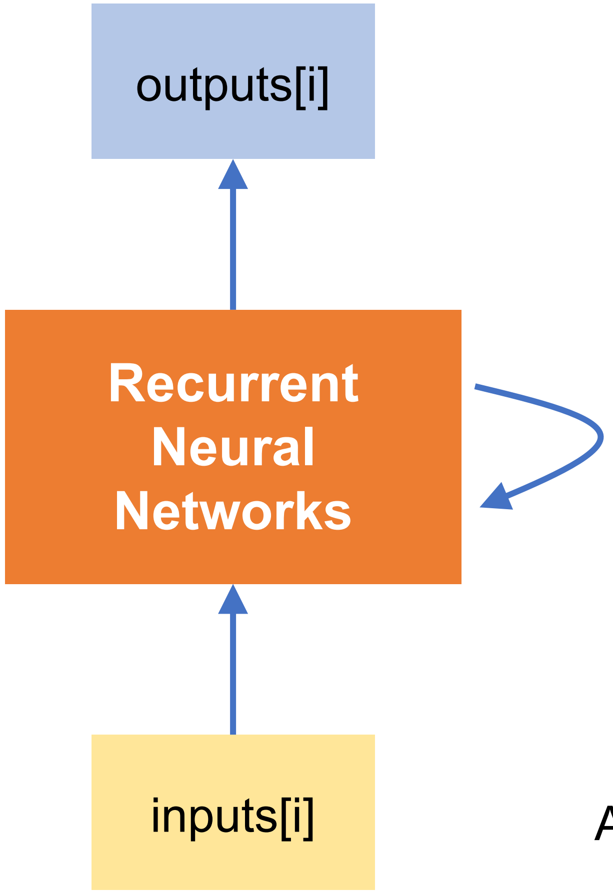
Recurrent Neural Networks: Process Sequences



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

Video classification on frames

Recurrent Neural Networks



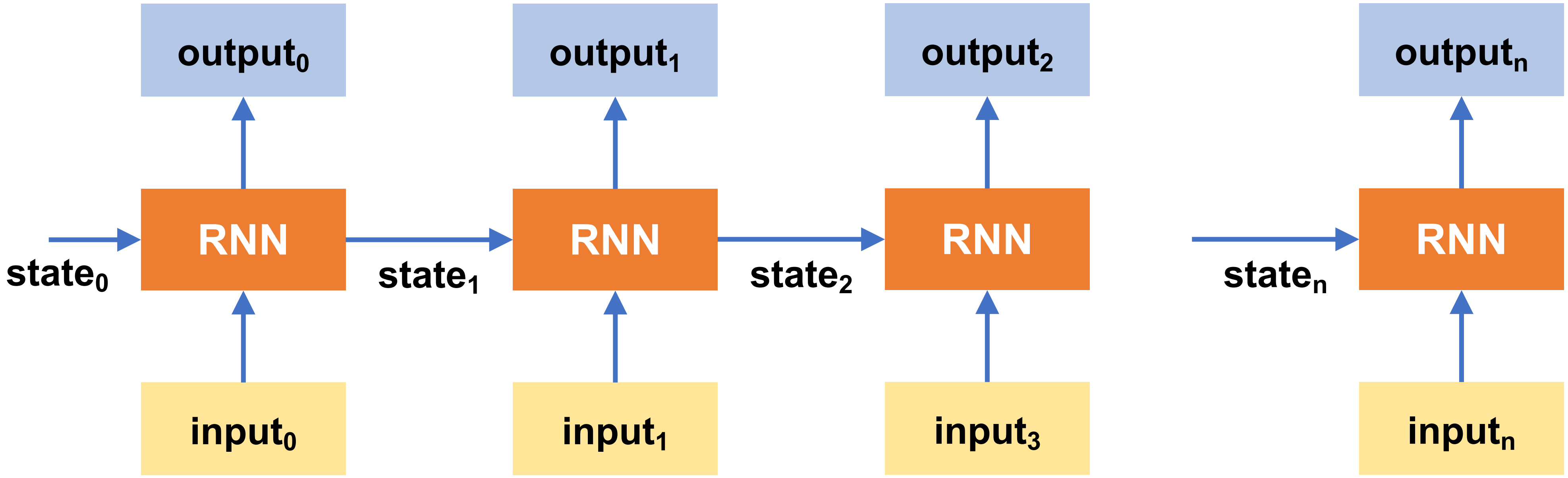
Arbitrary number of outputs

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)

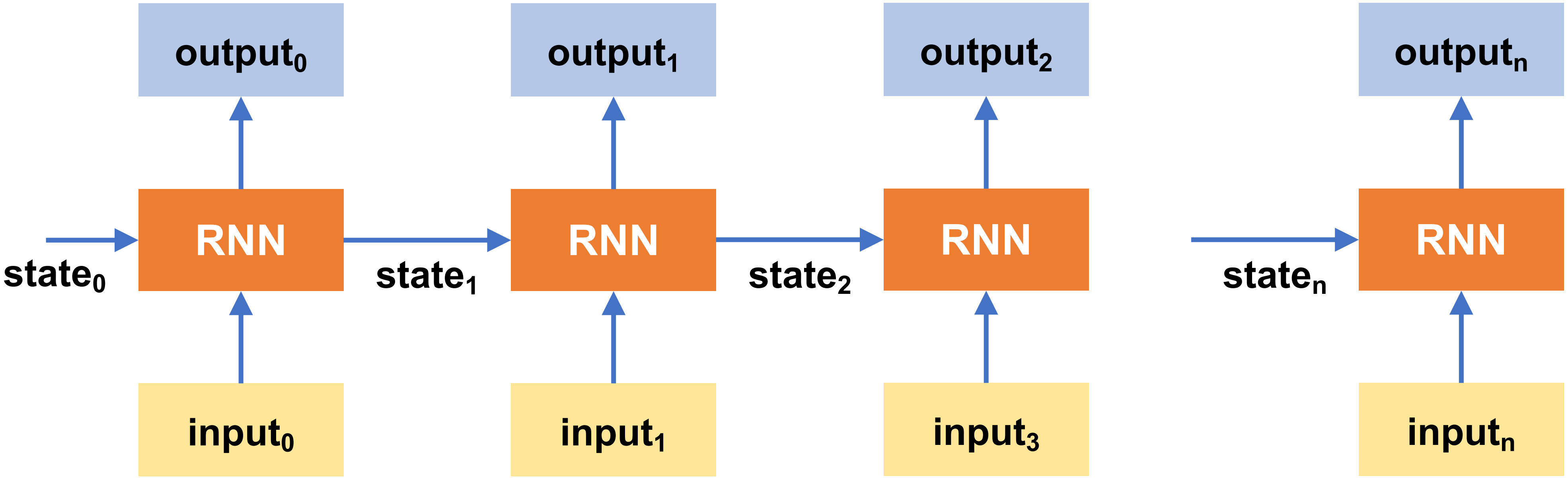


When do we need RNNs?

- 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)

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



Understand Our Applications: An Overview of Deep Learning Models

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

GPT-4 Demo

ZH How should I plan to learn CMU's parallel programming and architecture course?

Learning CMU's parallel programming and architecture course requires a structured approach to ensure that you cover all the necessary concepts and materials effectively. Here are some steps you can take to plan your learning:

1. Familiarize yourself with the course materials: Go through the course syllabus, lecture notes, and assignments to get an idea of what the course covers and the topics you need to learn.
2. Identify your learning objectives: Determine what you want to achieve from taking this course. Are you looking to gain a deeper understanding of parallel programming or

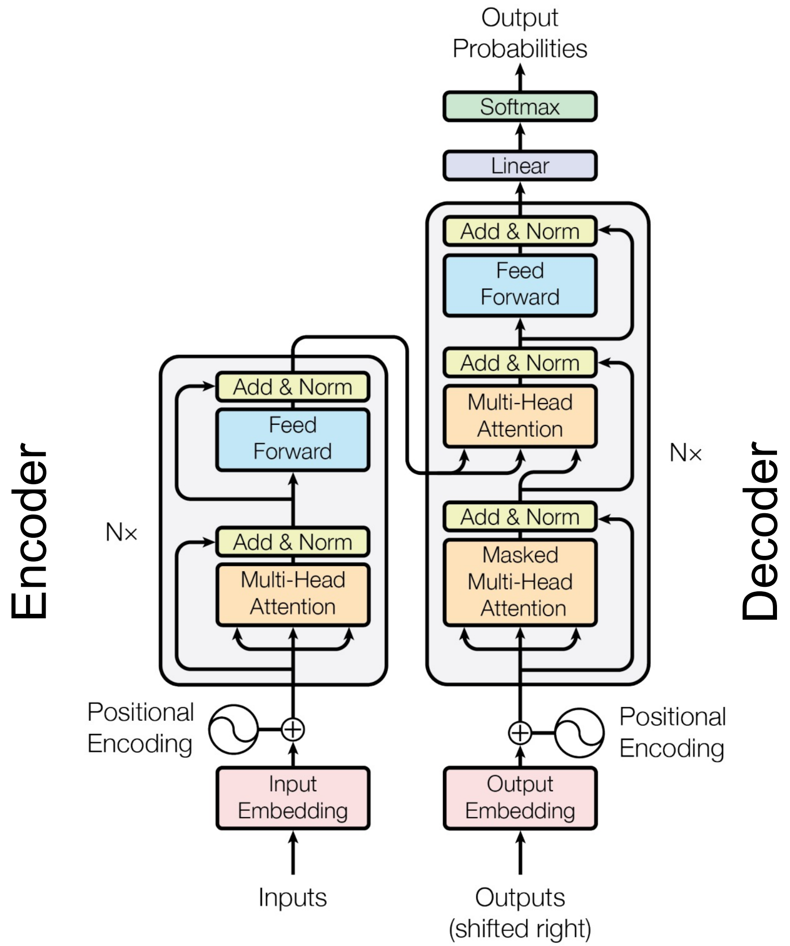
This report focuses on the capabilities, limitations, and safety properties of GPT-4. GPT-4 is a Transformer-style model [33] pre-trained to predict the next token in a document, using both publicly available data (such as internet data) and data licensed from third-party providers. The model was then fine-tuned using Reinforcement Learning from Human Feedback (RLHF) [34]. Given both the competitive landscape and the safety implications of large-scale models like GPT-4, this report contains no further details about the architecture (including model size), hardware, training compute, dataset construction, training method, or similar.

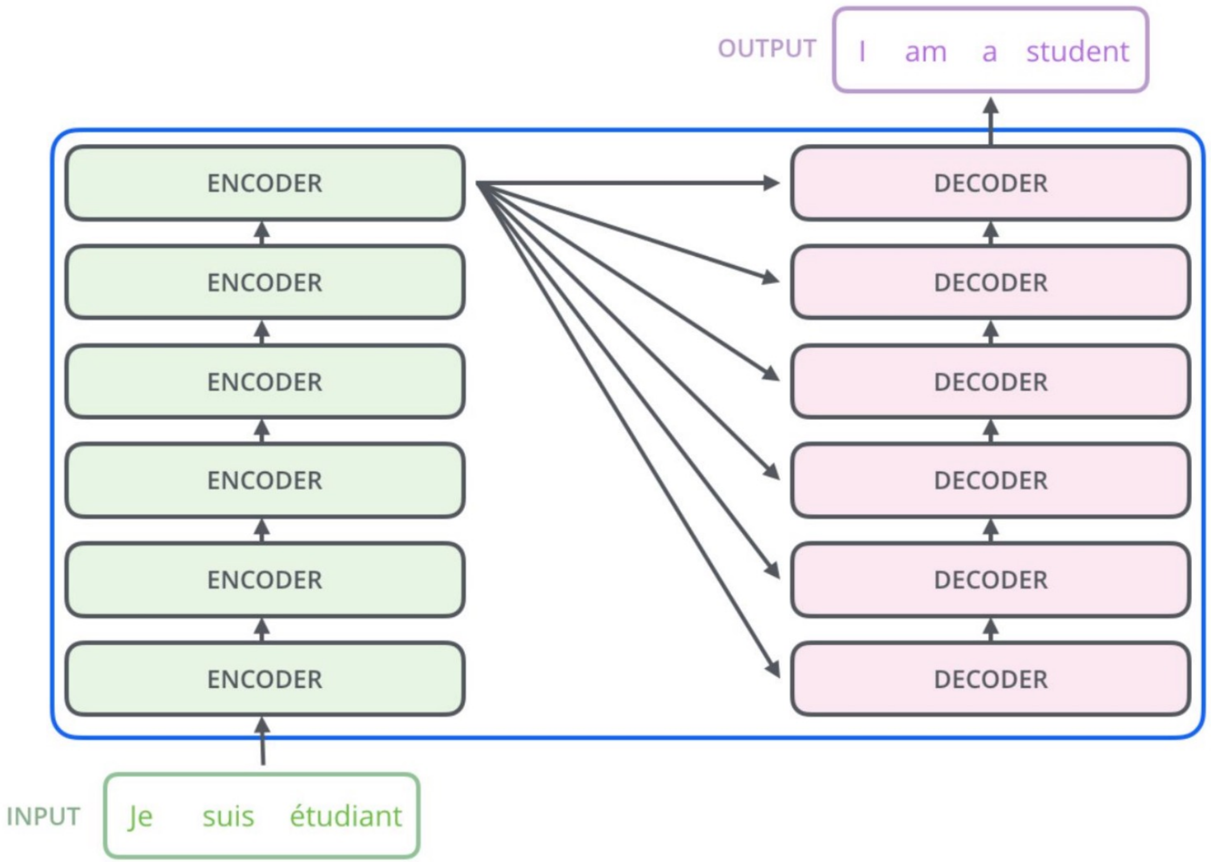
to seek help. You can reach out to your professor, teaching assistants, or classmates for assistance.

7. Practice coding: Parallel programming involves coding, so it's important to practice coding examples to improve your skills.
8. Review and reflect: Periodically review what you have learned, and reflect on how you can apply the concepts in practice. This will help you retain the information better and build a deeper understanding of the course material.

Remember that learning is a process, and it takes time and effort to master new concepts. By following these steps and staying committed to your learning goals, you can effectively learn CMU's parallel programming and architecture course.

Transformer: Self-Attention Mechanism for Language Models

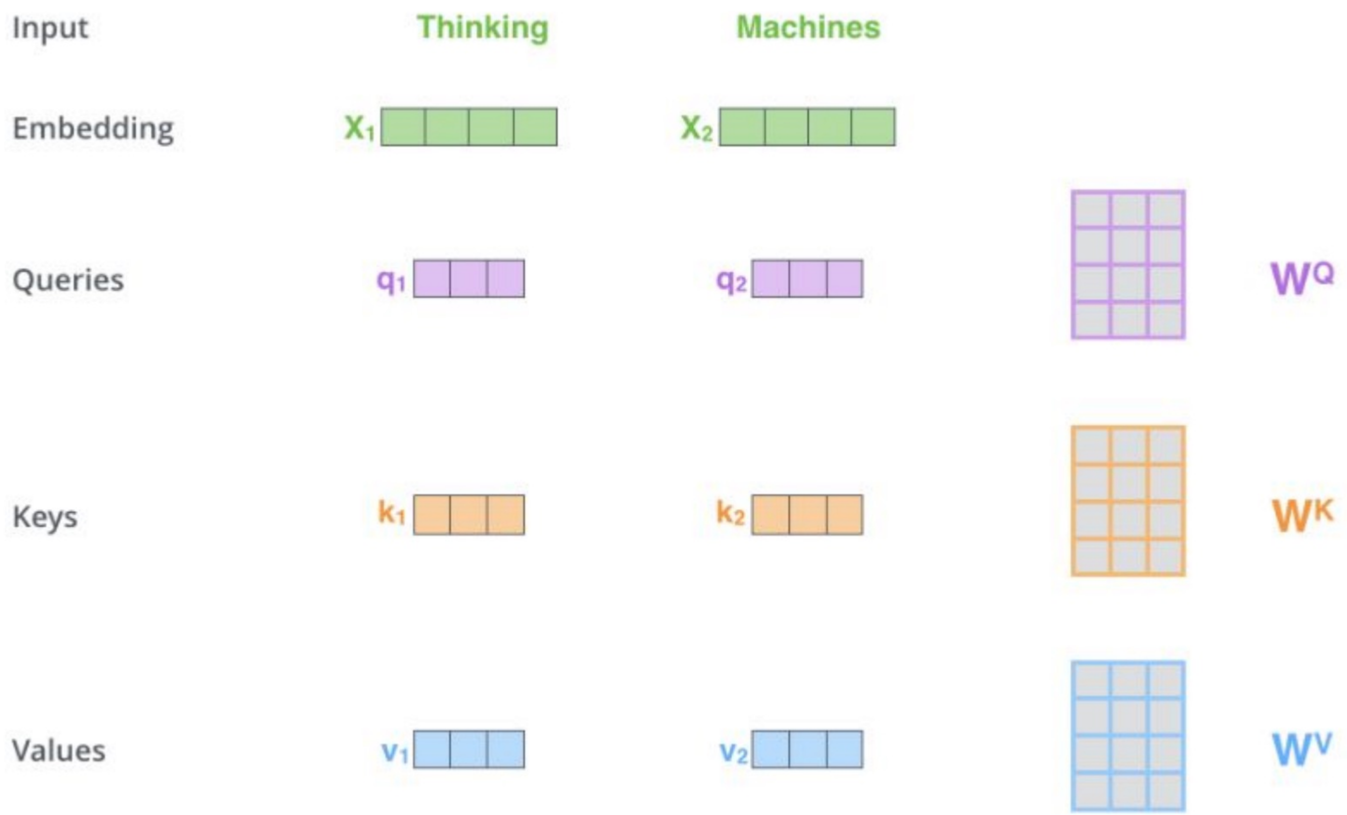




Ashish Vaswani et. al. Attention is all you need.

Self-Attention

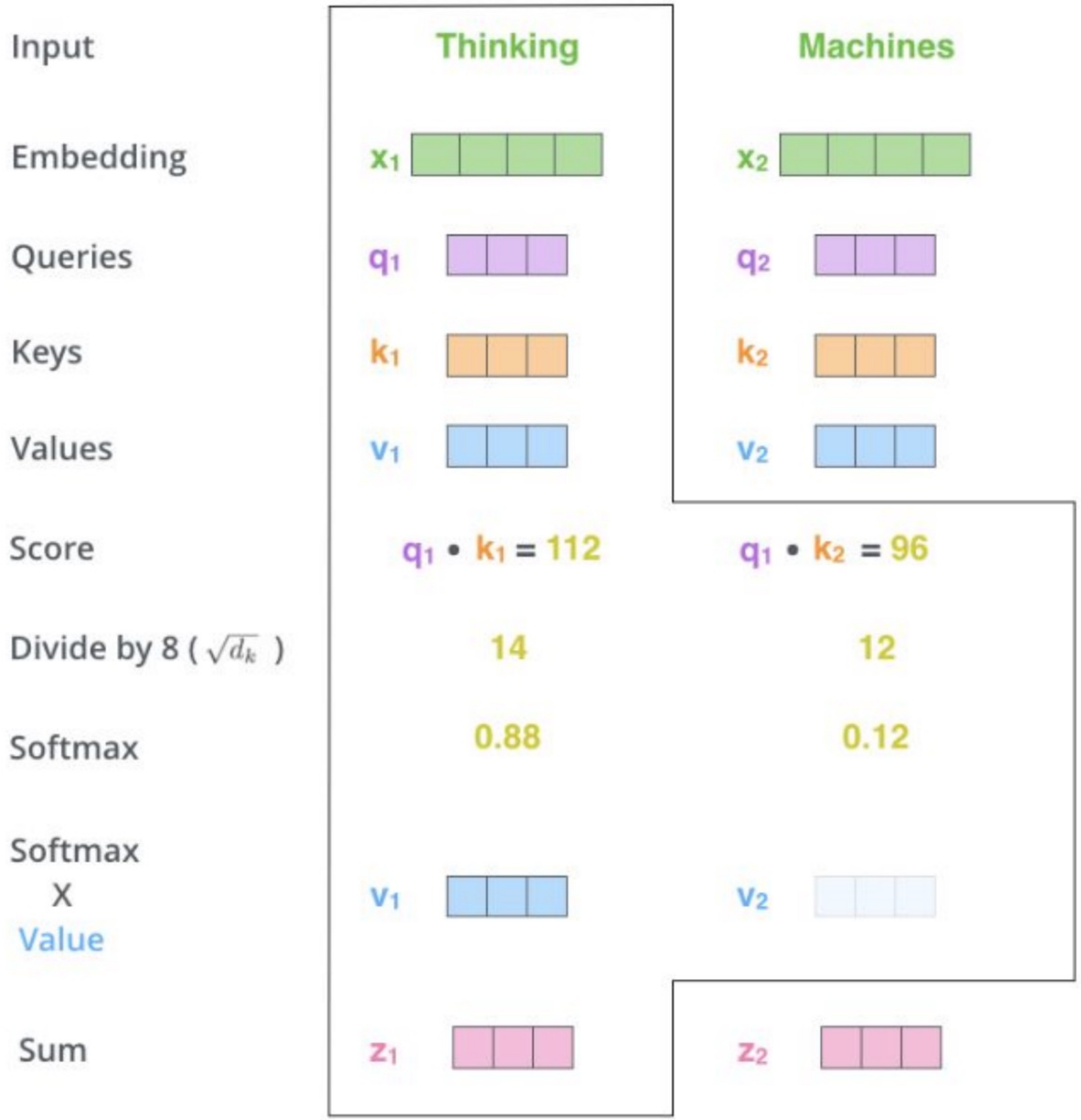
- Mapping a query and a set of key-value pairs to an output



Slide credit: Jay Allamar

Self-Attention

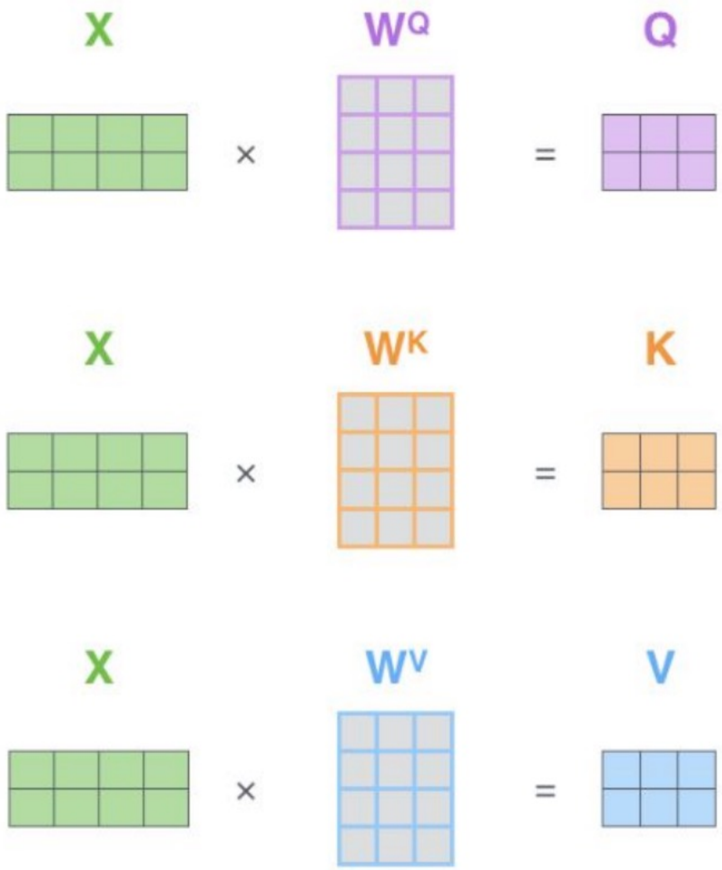
- Mapping a query and a set of key-value pairs to an output



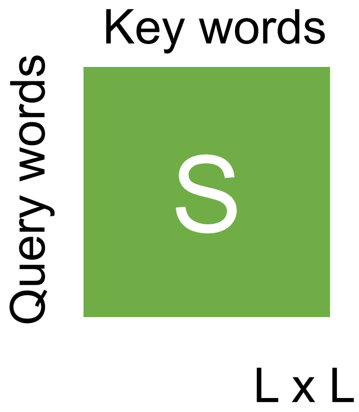
Slide credit: Jay Allamar

Self-Attention

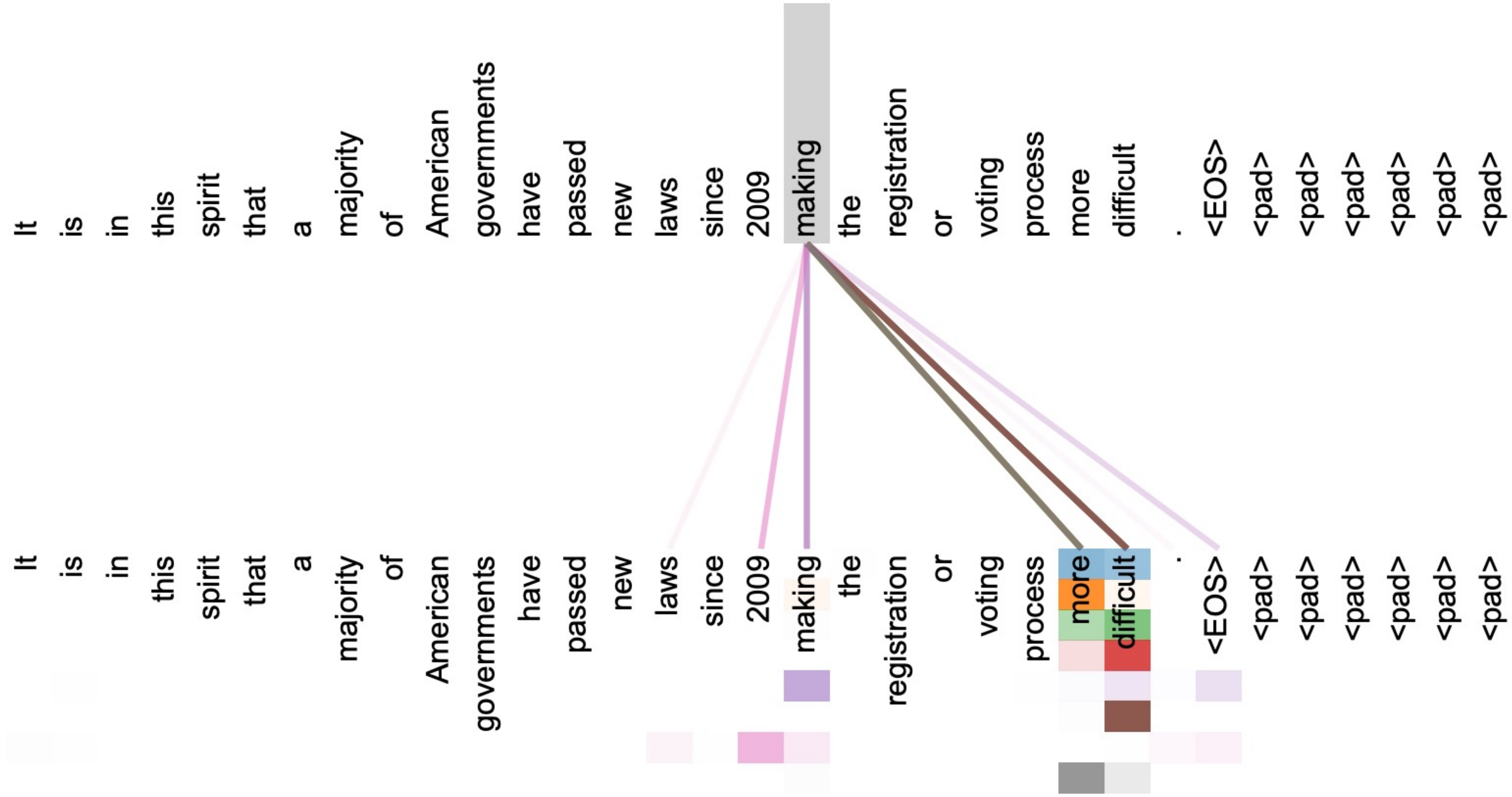
- Multiple matrix multiplications



$$A(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

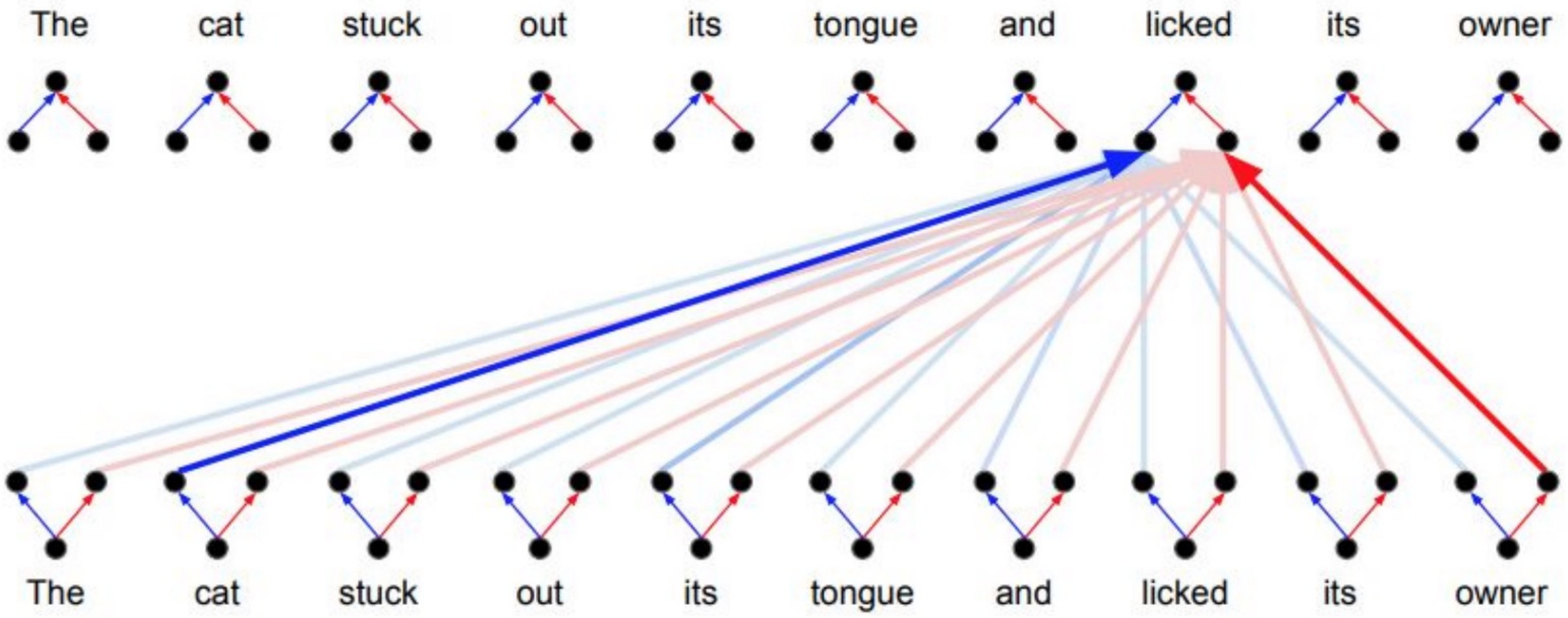


Self-Attention

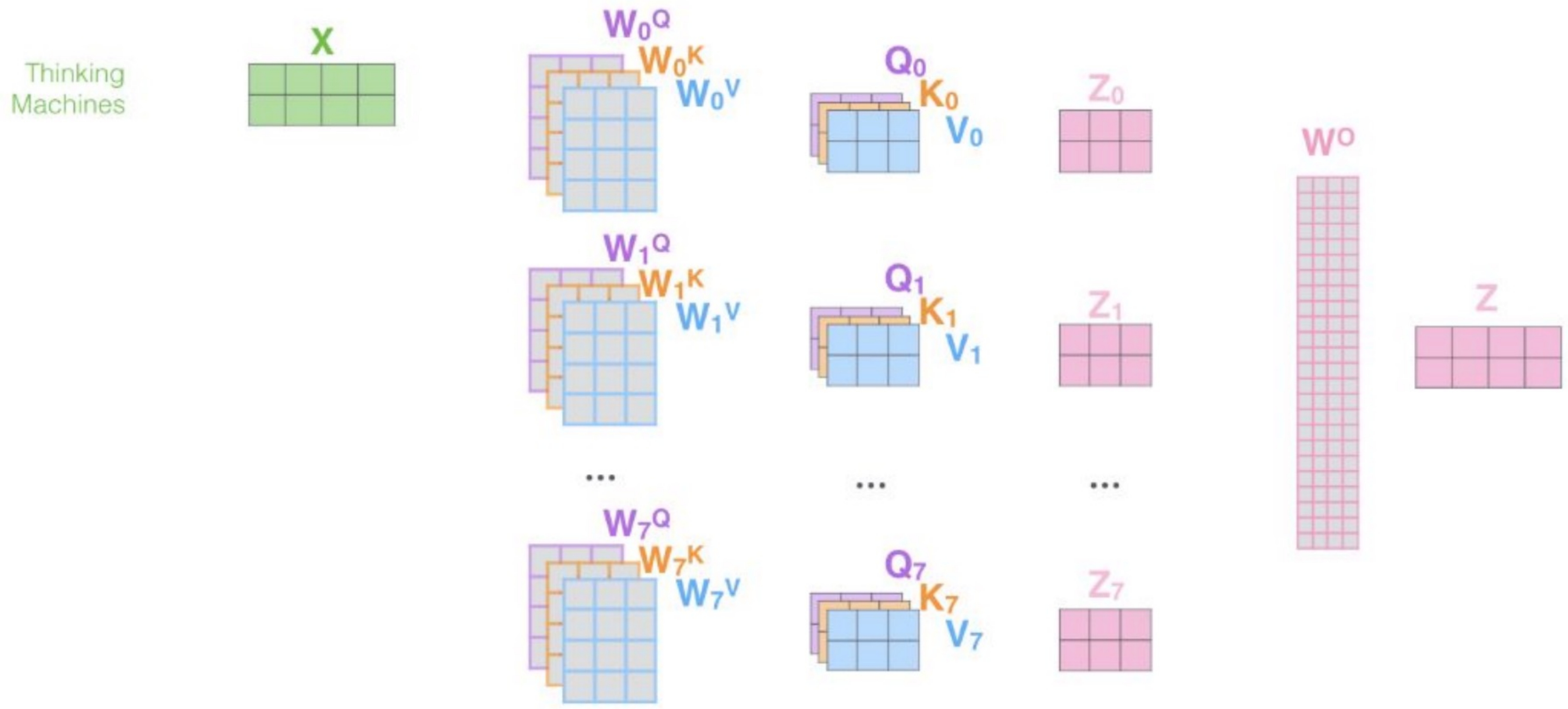


Multi-Head Self-Attention

- Parallelize attention layers with different linear transformations on input and output
- **Benefits: more parallelism, reduced computation cost**



Multi-Head Self-Attention

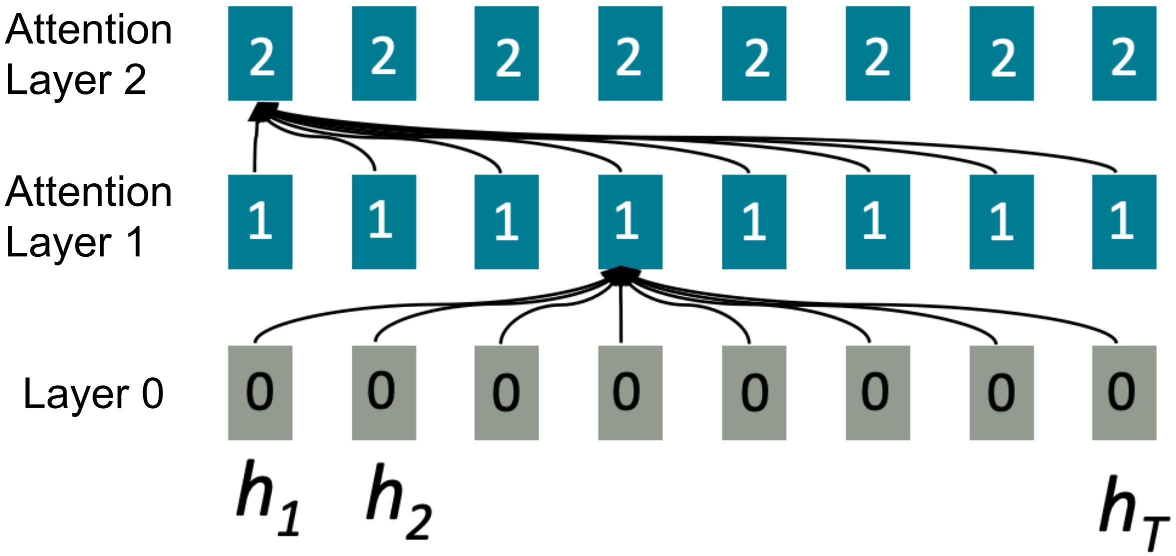


$$Z_i = A(Q_i, K_i, V_i) = softmax\left(\frac{Q_i {K_i}^T}{\sqrt{d}}\right)V_i$$

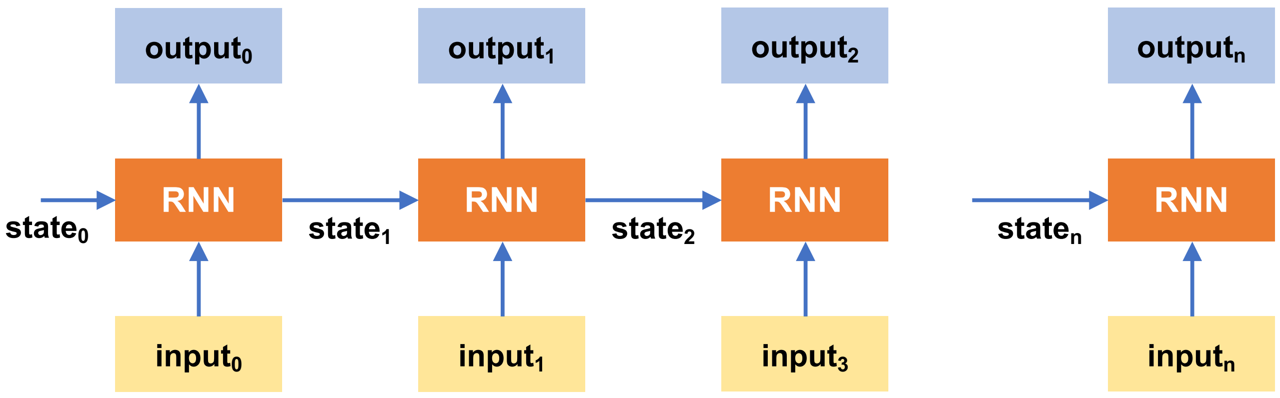
$$Z = MultiHead(Q, K, V) = Concat(Z_0, \ldots, Z_7)W^o$$

Why Transformers is More Computationally Efficient Than RNNs

- Enable parallelism within a sequence



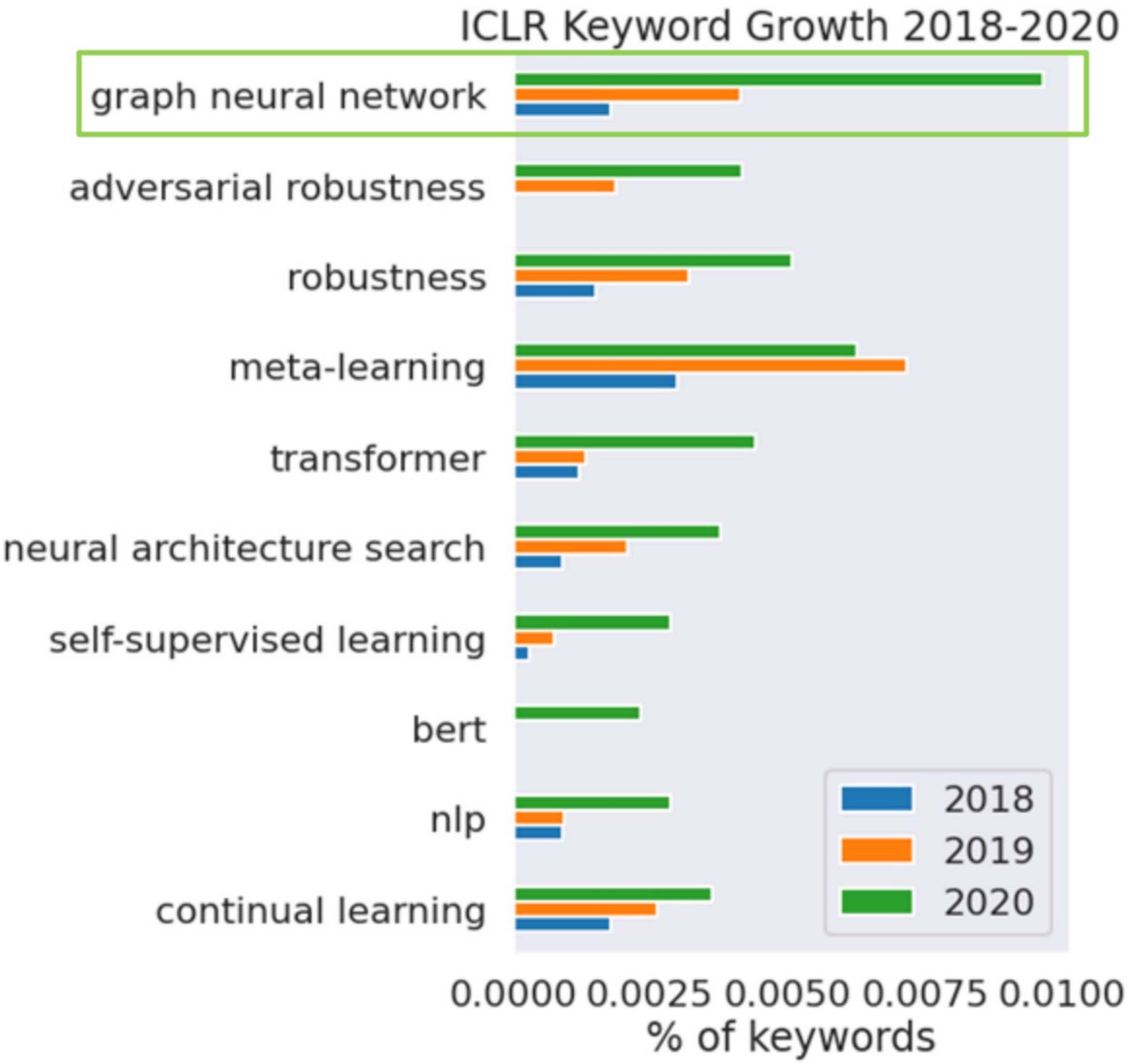
- lack of parallelizability



Understand Our Applications:
An Overview of Deep Learning Models

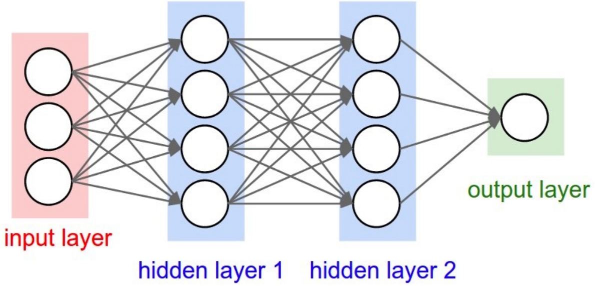
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transformers
- **Graph Neural Networks**
- Mixture-of-Experts

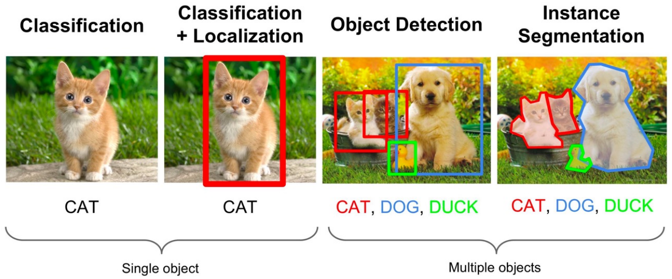
Graph Neural Networks: The Hottest Subfield in ML



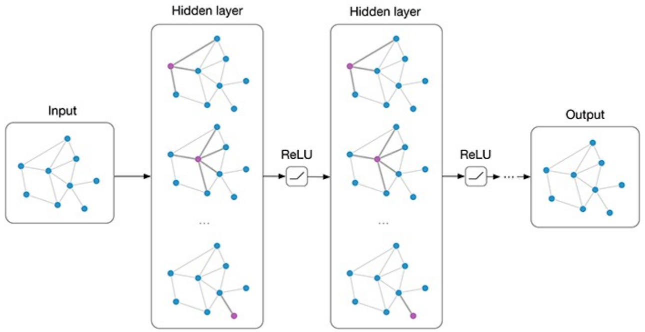
GNNs: Neural Networks on Relational Data

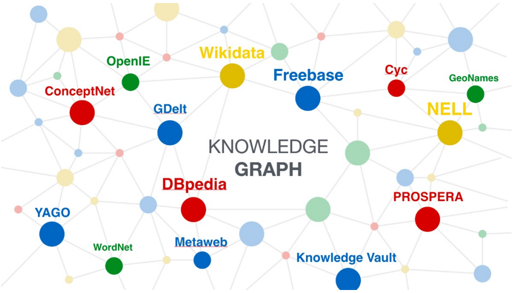
Neural Networks





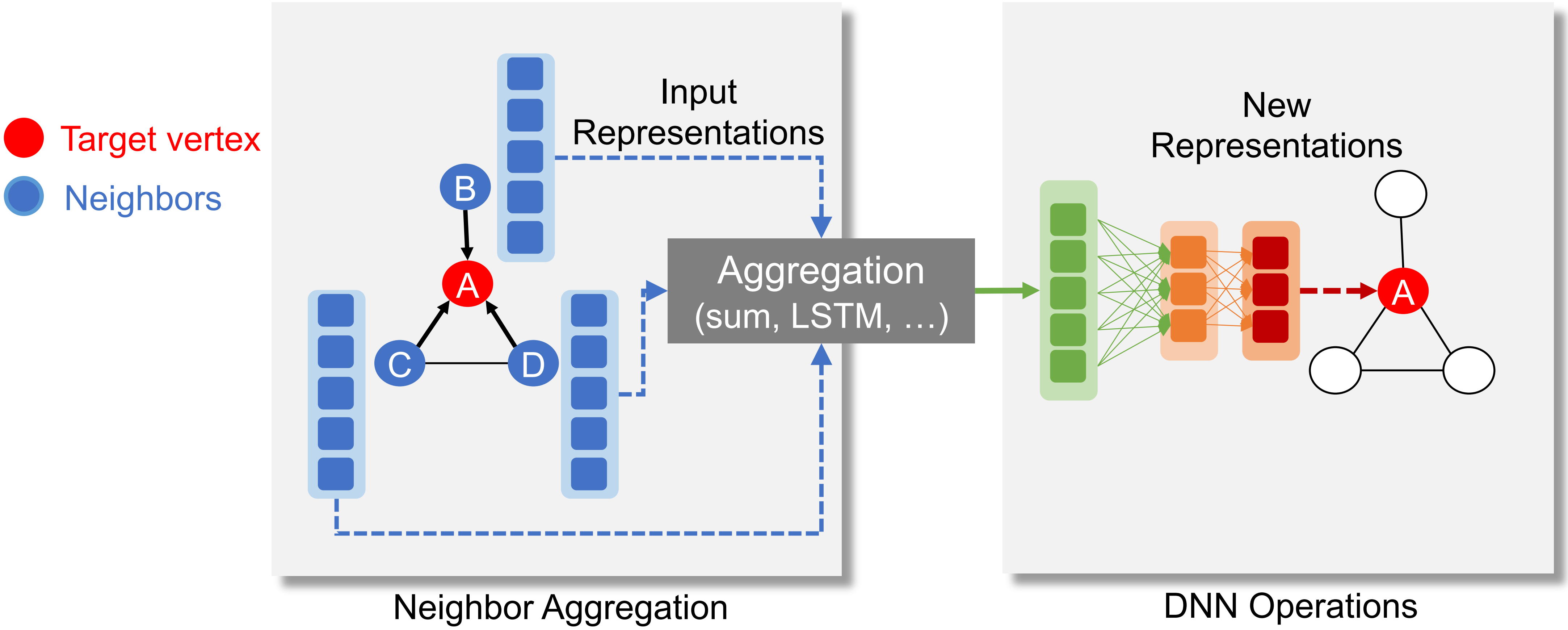
Graph Neural Networks





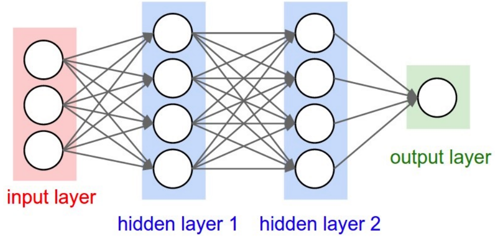
Graph Neural Network Architecture

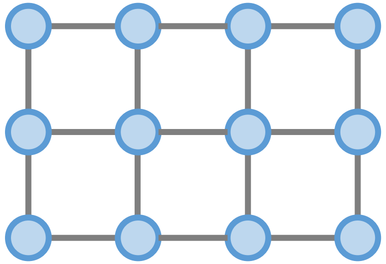
- Combine **graph propagation** w/ **neural network operations**



Challenges of GNN Computations on GPUs

Neural Networks

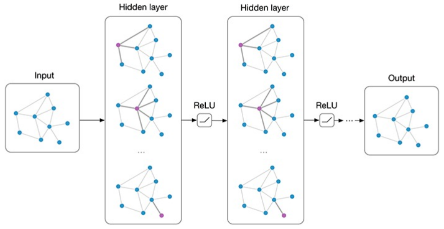


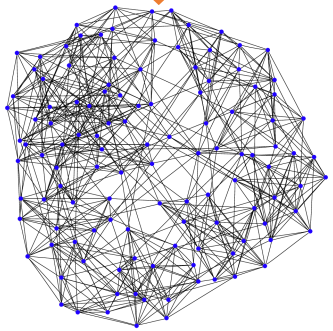


Small and regular intermediate data

☺ **Good efficiency**

Graph Neural Networks





Large and irregular intermediate data

☹ **Efficiency & scalability challenges**

How to Design Systems for Graph Neural Networks

- **New Programming Models**: gather-apply-scatter programming interface for distributed GNN

- **New Systems Infrastructure**: serverless computing for low-cost GNN training

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

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