https://hao-ai-lab.github.io/dsc291-s24/

DSC 291: ML Systems
Spring 2024

LLMs

Parallelization

Single-device Optimization

Basics

Enrollment Request

- The instructor team have approved all requests
- It is pending the DSC to decide if they want to enroll you or not
- I have written an email to Julia (DSC manager), waiting for response.
- If you are still in queue (Pending approval)
  - Send us (me/Will/Anze) a message to be added as an observer
  - Wait for people to drop and you will be automatically enrolled until EoW2
- If you have been rejected by department
  - You are likely an undergrad
  - Recommendation: send an email to DSC to sincerely express your strong need for this course
- If our queue is still long by end of week 2 (no one is willing to drop)
  - I'll write a second email to DSC Dean

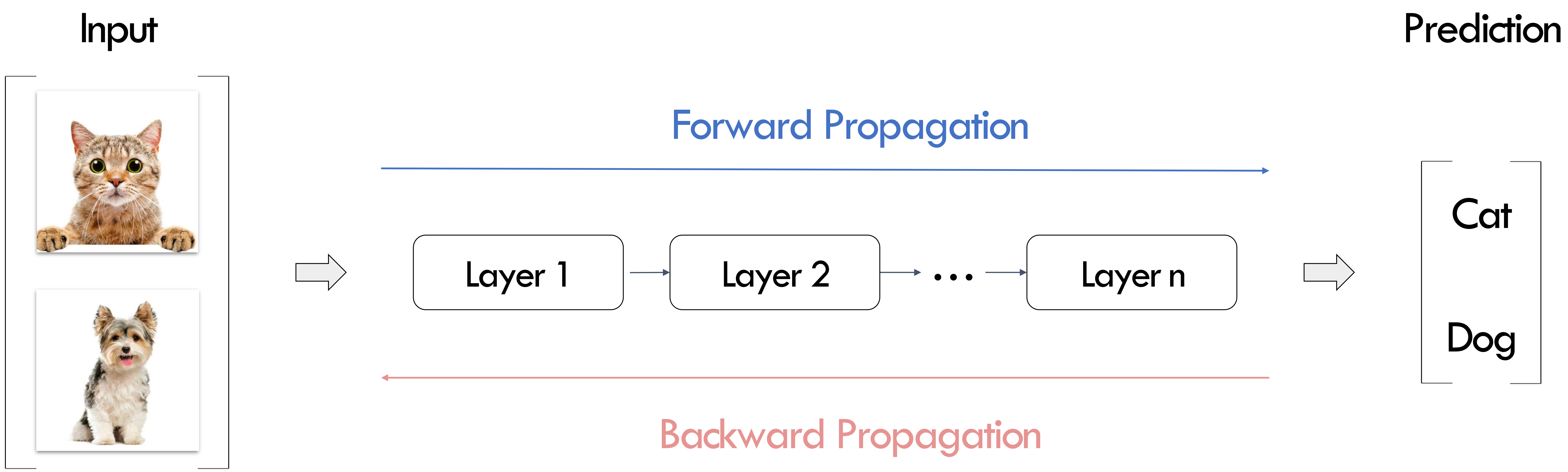
Two forms worth your attention

- Beginning of quarter survey
  - Please fill the survey
    - If >=80% of you filled the survey, all of you get 0.5%
    - If <80%, all of you do not get 0.5%
- Final presentation team-up spreadsheet:
  - https://docs.google.com/spreadsheets/d/1foOkwrumTpuhd6xpNI0QHx9R31Biu-h0UdTp5wMItsQ/edit#gid=0
  - Each team <= 5 people
  - We put 14 projects there (more than needed)
  - Do some Google search before you put your name

Today

- Understand our Workloads: Deep Learning
- Dataflow graph representation
  - Flavors of different ML frameworks

Background: DL Computation



$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update} \\ \text{(sgd, adam, etc.)}}}\Big(\theta^{(t)}, \underbrace{\nabla_L}_{\substack{\text{model} \\ \text{(CNN, GPT, etc.)}}}\big(\theta^{(t)}, \underbrace{D^{(t)}}_{\text{data}}\big)\Big)$$

Three important components

Data

- images
- Text
- Audio
- Table
- etc.

Model

- CNNs
- RNNs
- GNNs
- Transformers
- MoEs

Compute

- cpus
- gpus/tpus/lpus
- M3/FPGA/etc.

Model: three parts

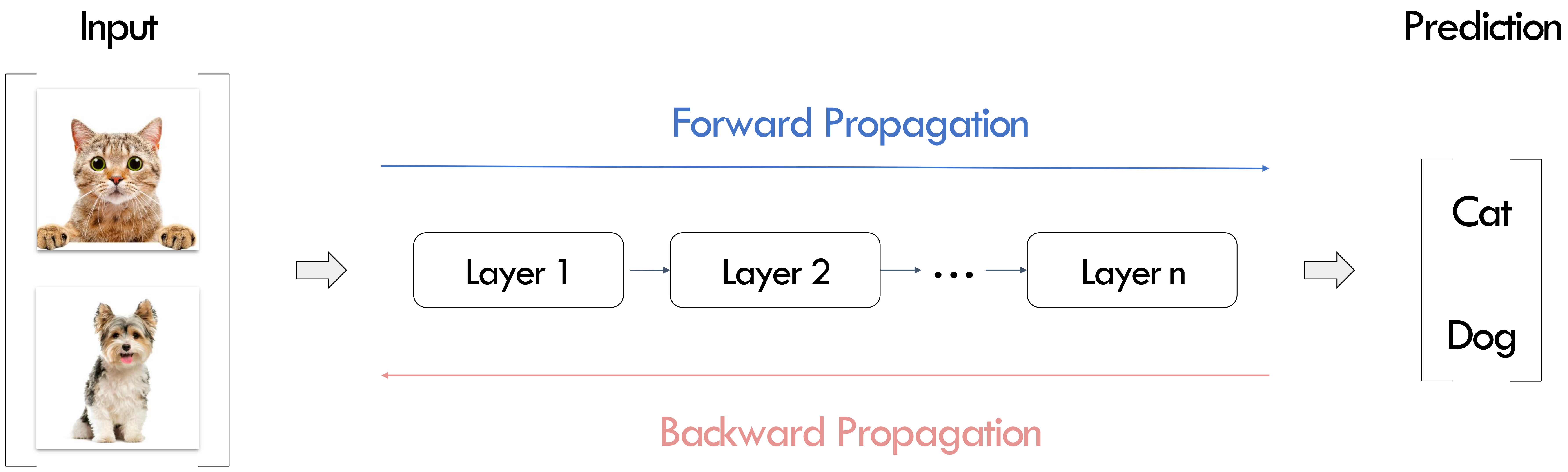
$$\theta^{(t+1)} = f\left(\theta^{(t)},\ \nabla_L\left(\theta^{(t)},\ D^{(t)}\right)\right)$$

parameter | weight update (sgd, adam, etc.) | model (CNN, GPT, etc.) | data

- Model: A parameterized function that describes how do we map inputs to predictions
  - CNNs/RNNs/Tranformers
- Loss function: How “well” are we doing for a given set of parameters
  - L2 loss, hinge loss, softmax loss, ranking loss
- Optimization method: A procedure to find a set of parameters that minimizes the loss
  - SGD, Variational inference, Newton methods

How to express these computation?

- Idea: Composable Layers



Today

- **Understand our Workloads: Deep Learning**
- Dataflow graph representation

Understand Our Workload (a.k.a. DL course in 30 mins)

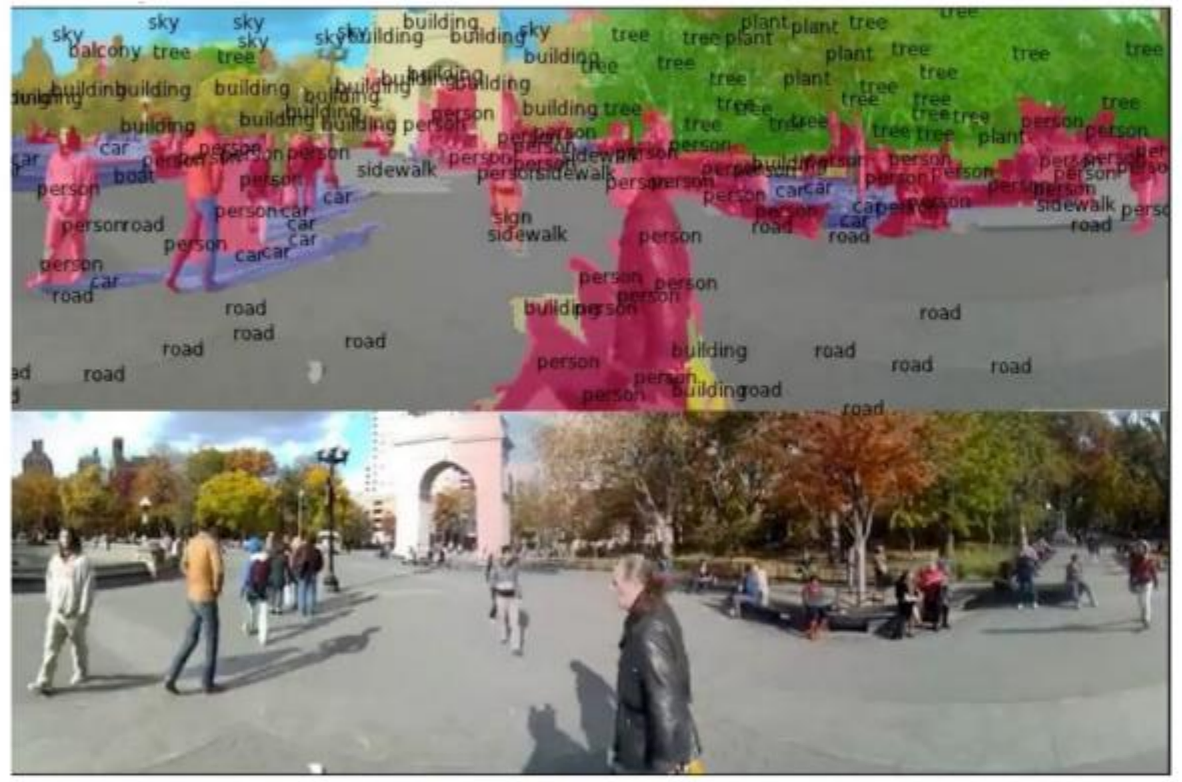
- There are many great models developed in the history
- In this class, we review the most important 5 classes
  - Convolutional Neural Networks
  - Recurrent neural networks
  - Transformers
  - Graph neural networks
  - Mixture-of-Experts
- If you have trouble following this session, read deep learning book or learn https://sites.google.com/view/cse251b

CNNs: Applications

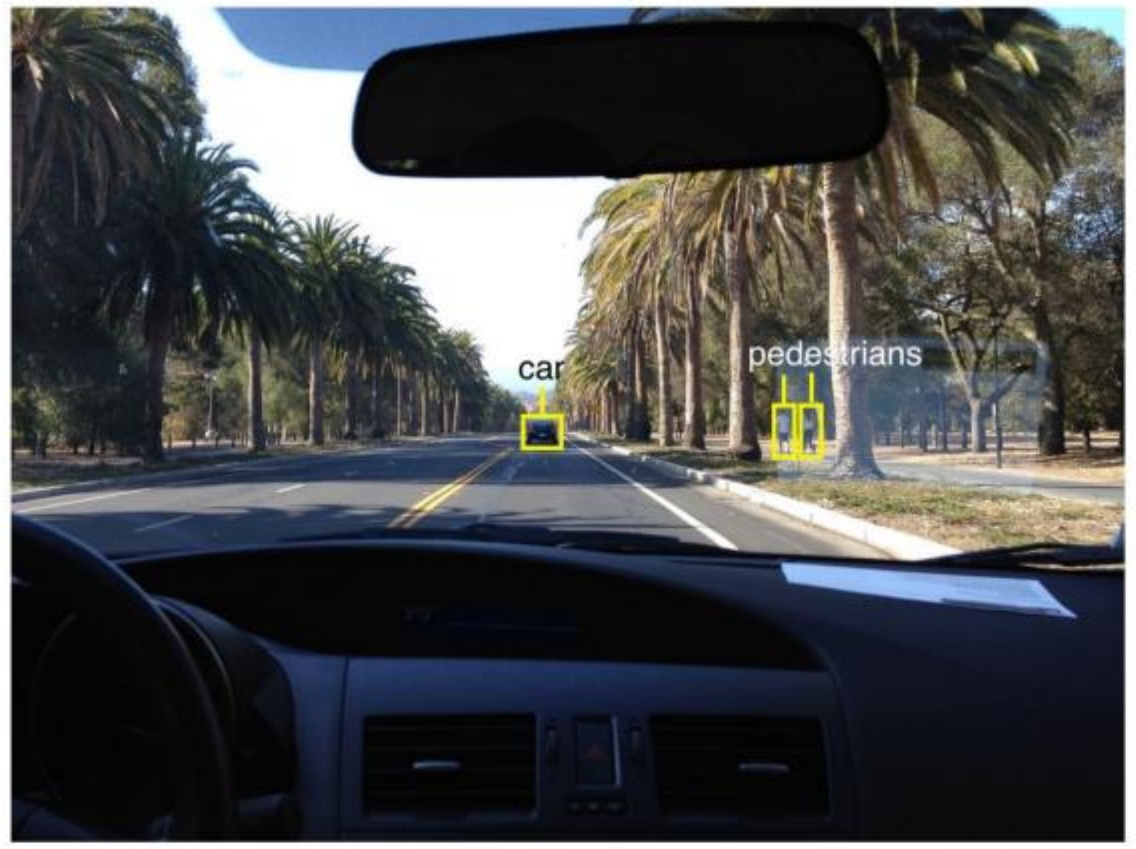
Classification

Retrieval

Detection



Segmentation

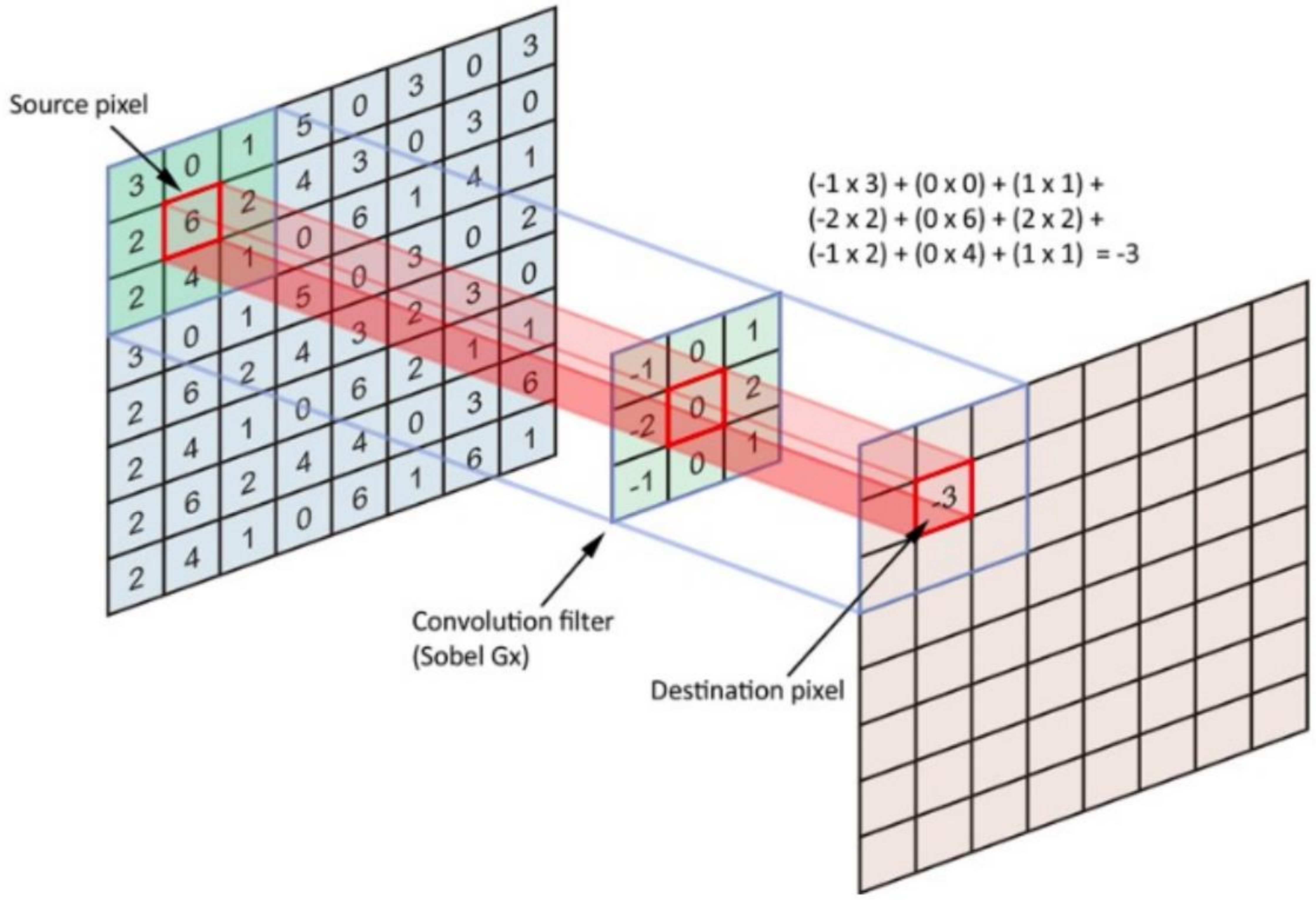


Self-Driving

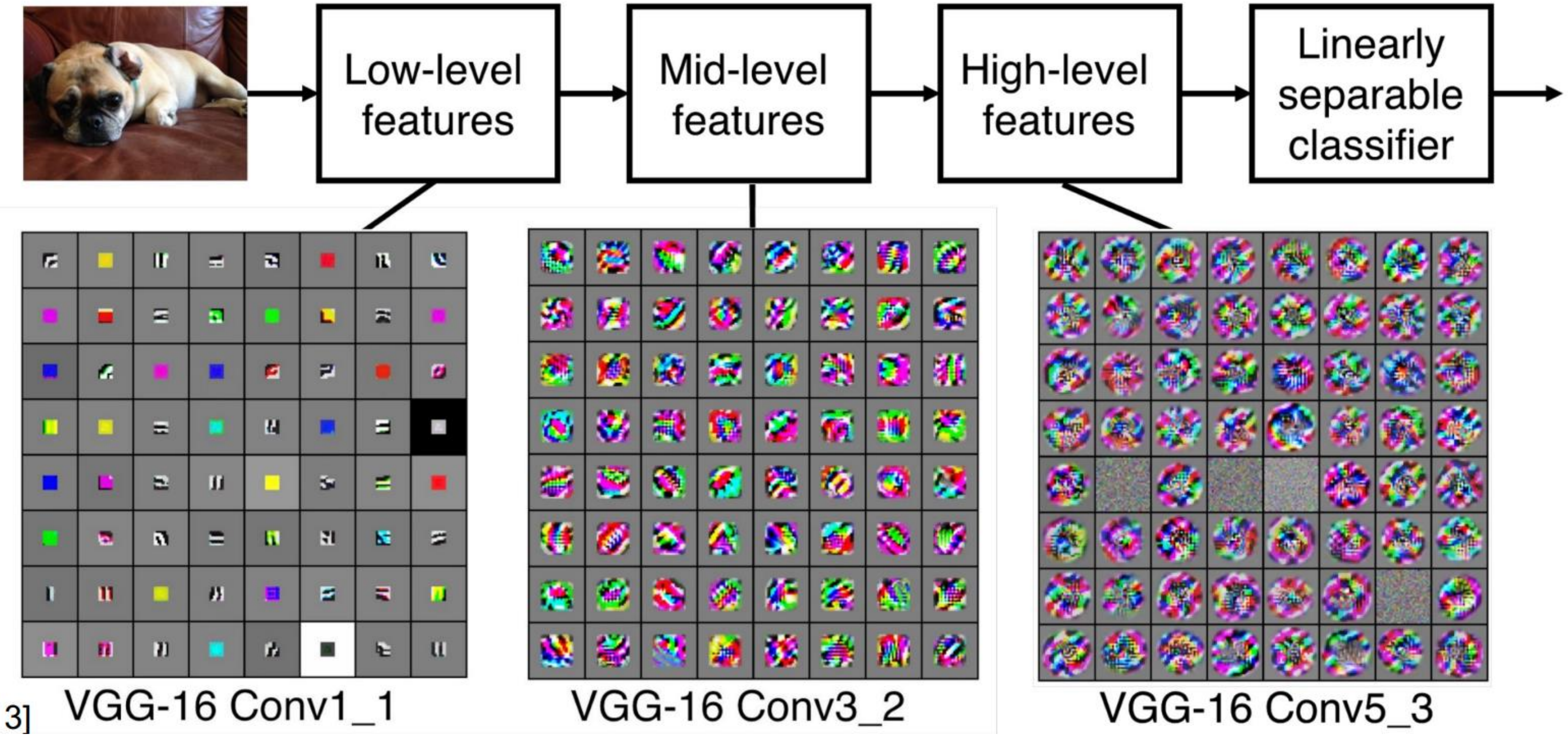
Synthesis

CNN: Key components

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



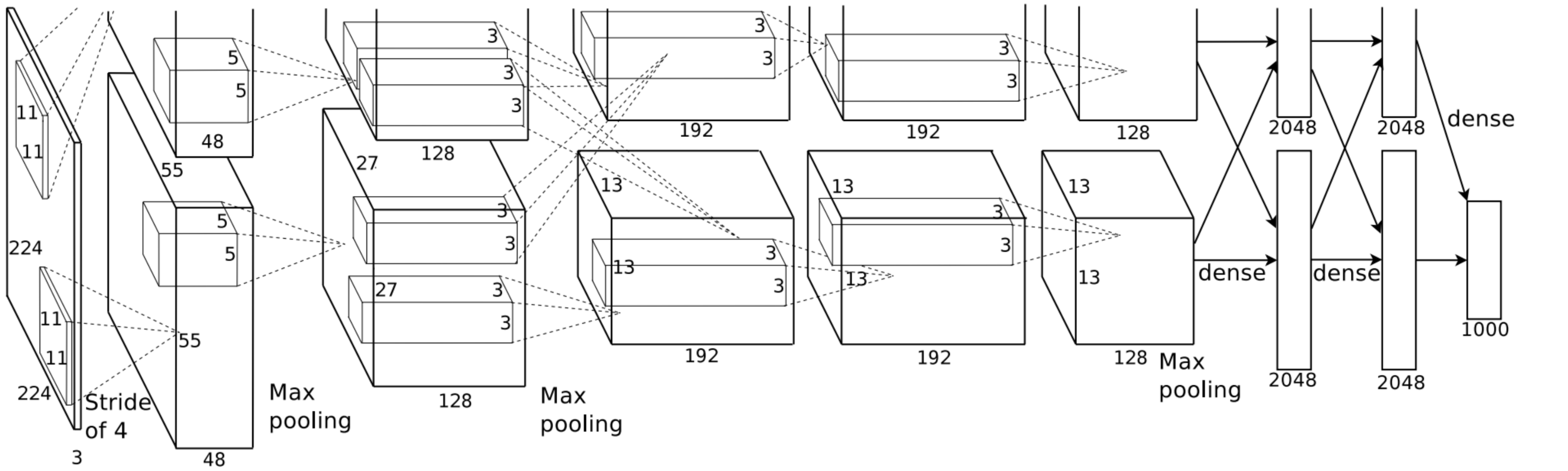
Stacking Conv layers

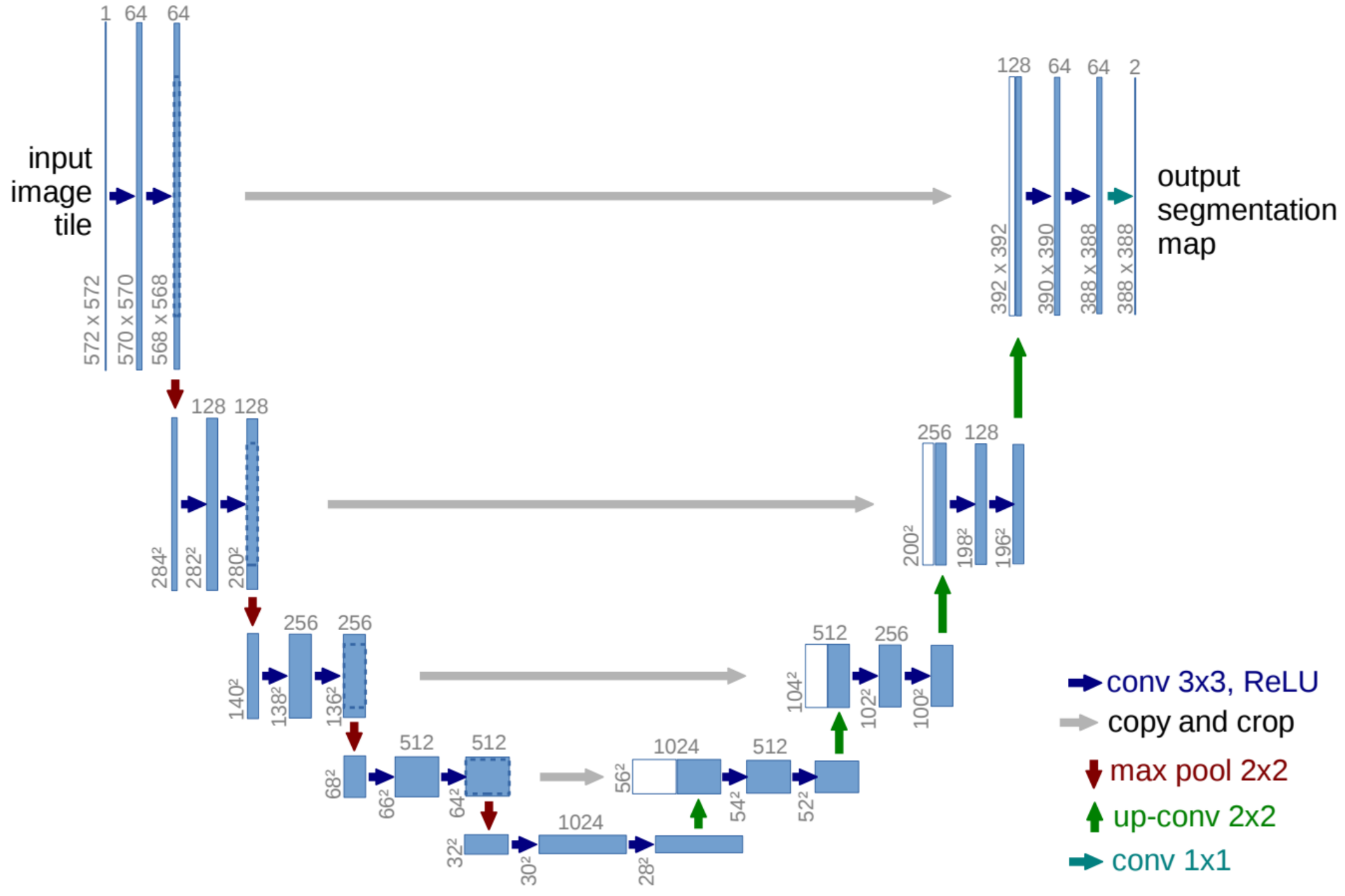


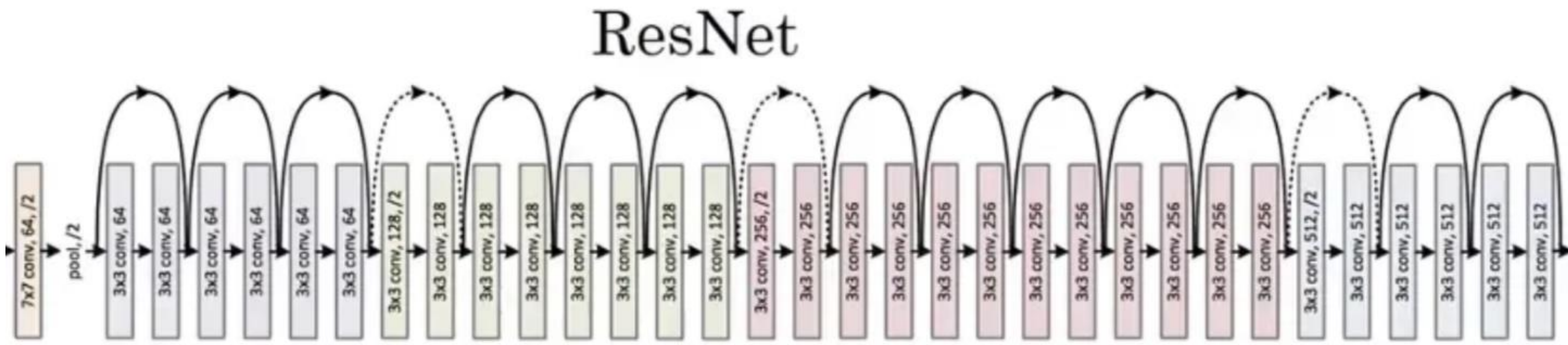
Zeiler and Fergus 2013]

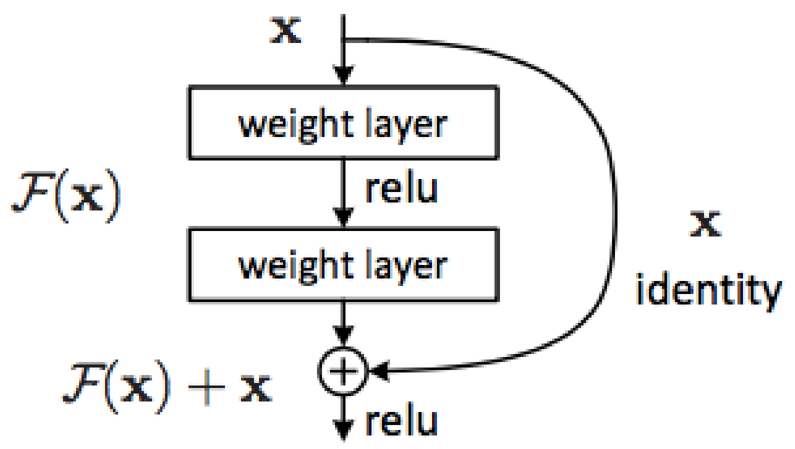
CNN: top3 models

- AlexNet by Alex/Iliya/Hinton
- ResNet by Kaiming etc.
- U-Net by Olaf etc.









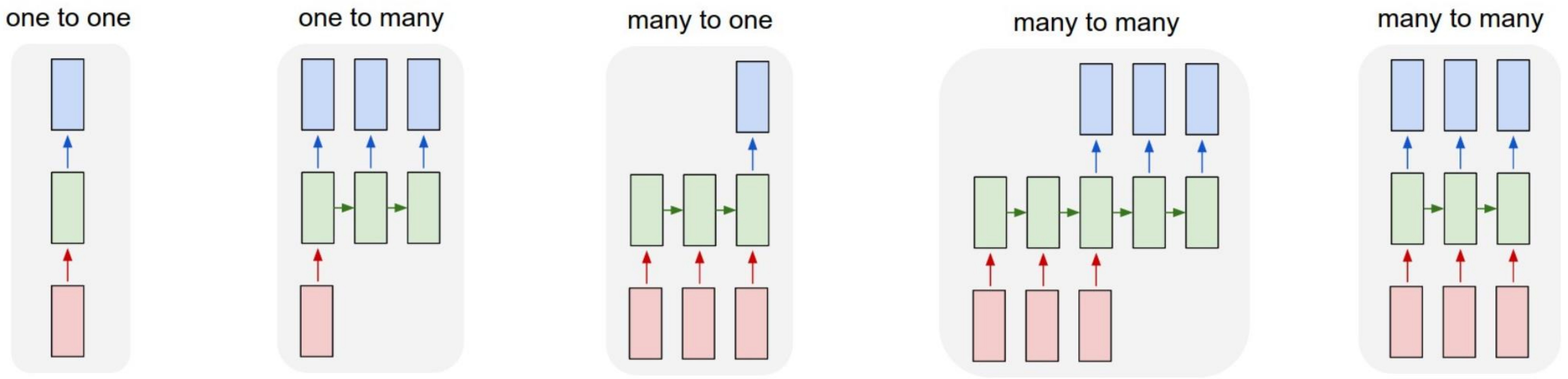
CNN more important components

- Conv
  - Conv1d, Conv2d, conv3d, etc.
- Matmul (linear) :
  - C = A * B
  - Softmax
- Elementwise operations:
  - ReLU, add, sub
- Other ops
  - Pooling, normalization, etc.

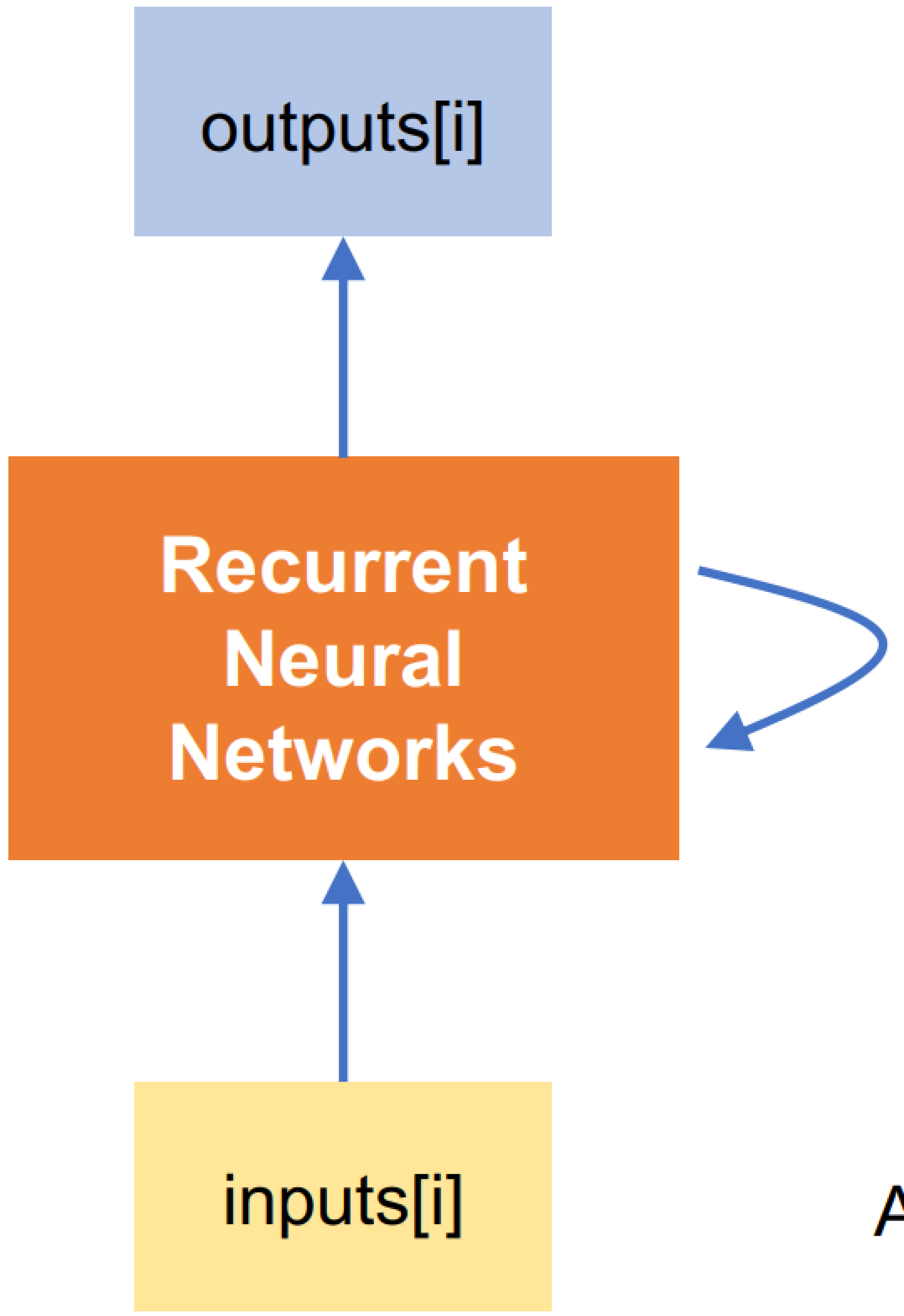
After-class Q

How UpConv works?

Recurrent Neural Networks



Recurrent Neural Networks

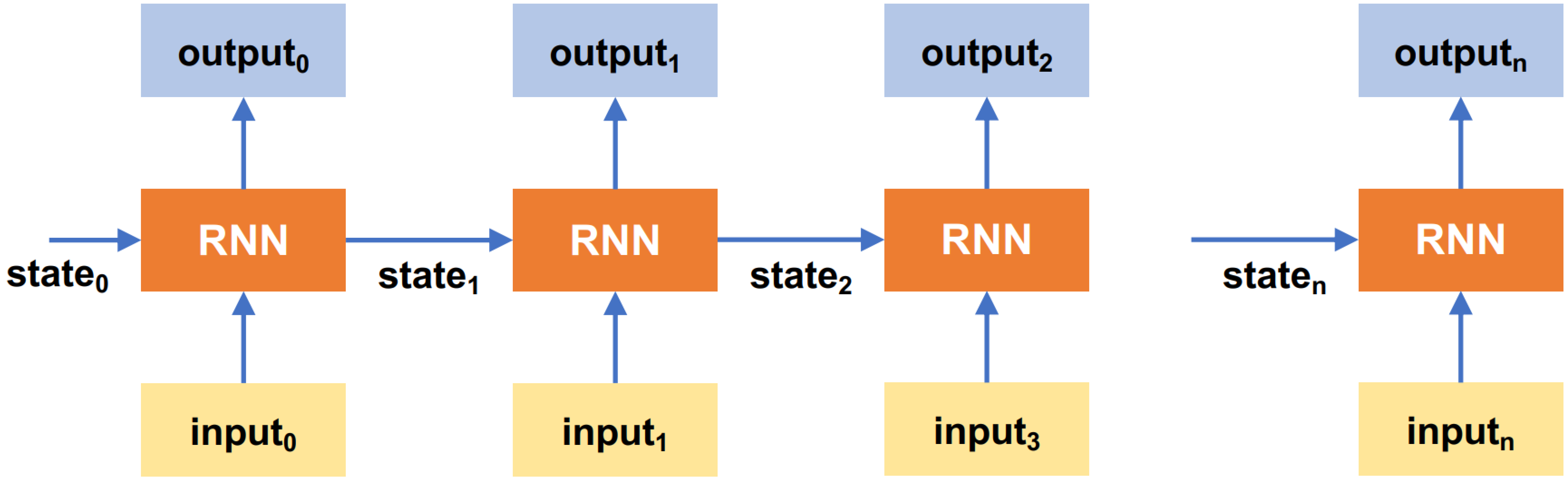


Arbitrary number of outputs

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

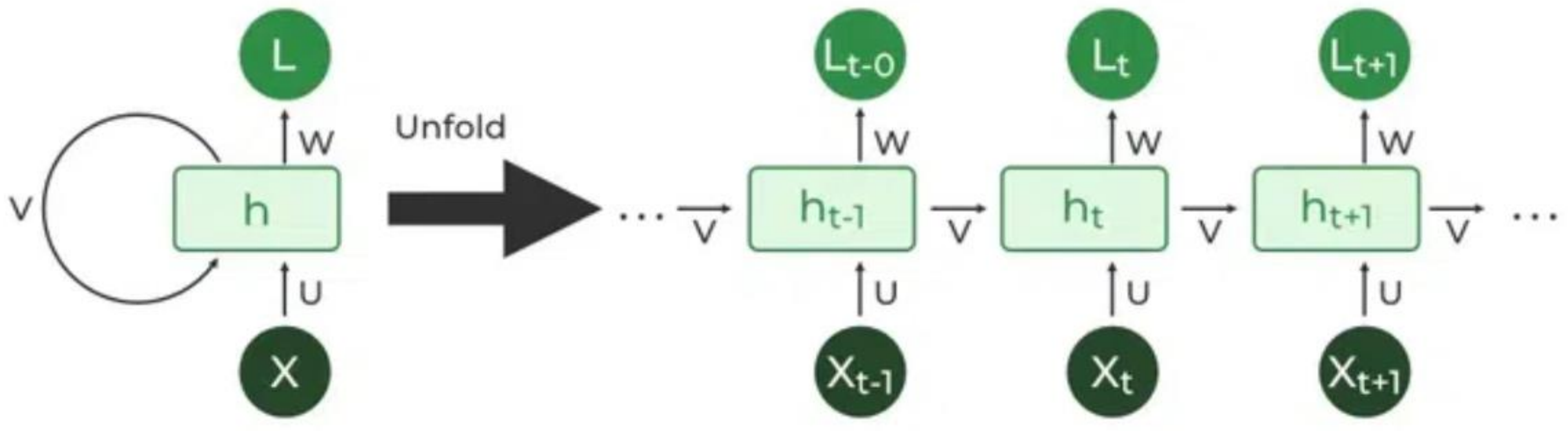
Arbitrary number of inputs

Recurrent Neural Networks: unrolling the computation



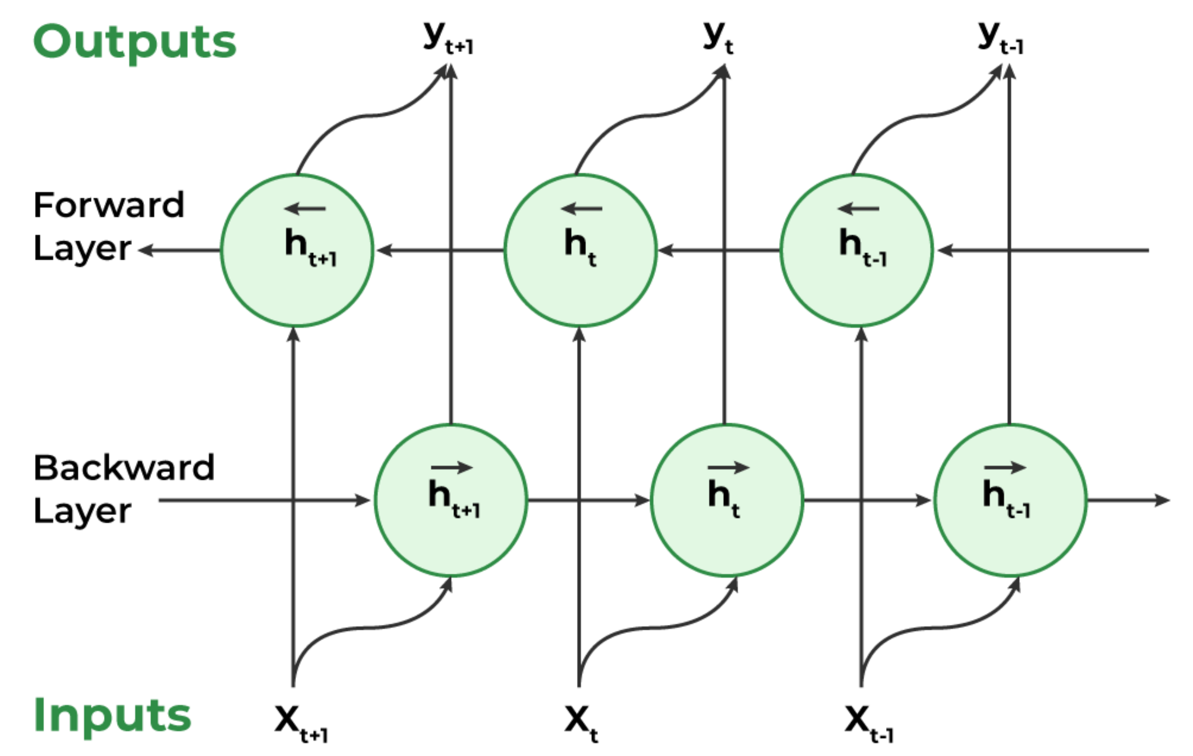
Most Important Components in RNNs

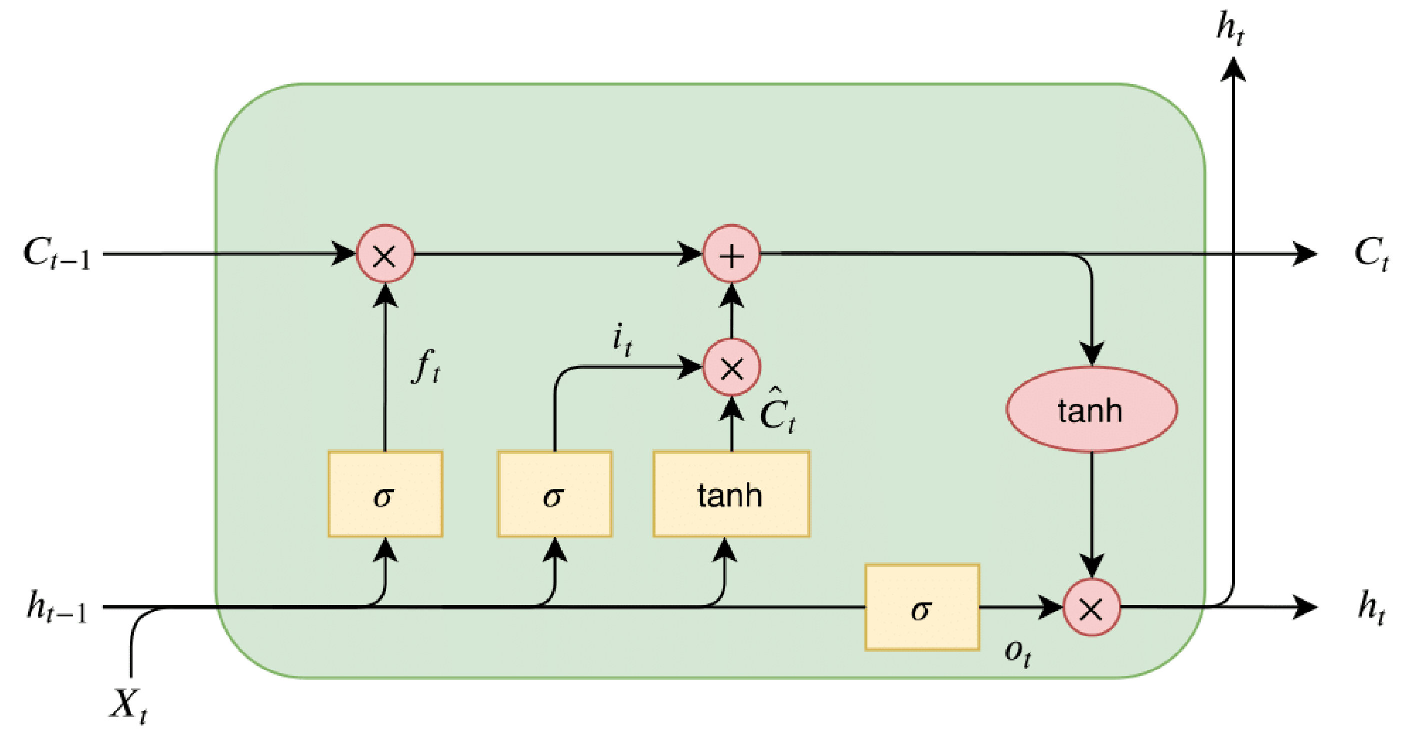
- One can make any basic neural network recurrent
- Matmul
- Elementwise nonlinear
  - ReLU, Tanh, sigmoid, etc.

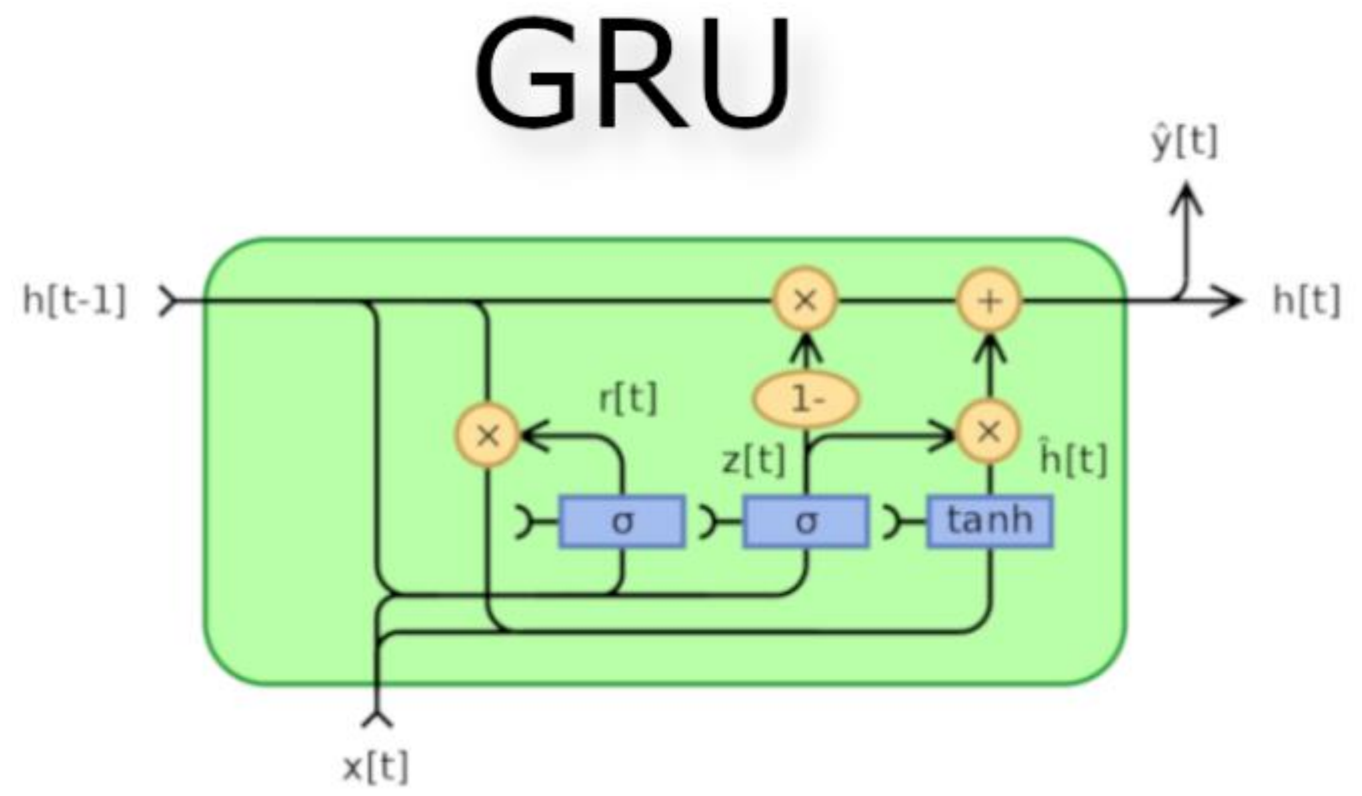


RNN: top3 models

- Bidirectional RNNs
- LSTM
- GRU







Story: Who Invented RNNs?

Jürgen Schmidhuber

Jürgen Schmidhuber (born 17 January 1963) is a German computer scientist noted for his work in the field of artificial intelligence, specifically artificial neural networks.

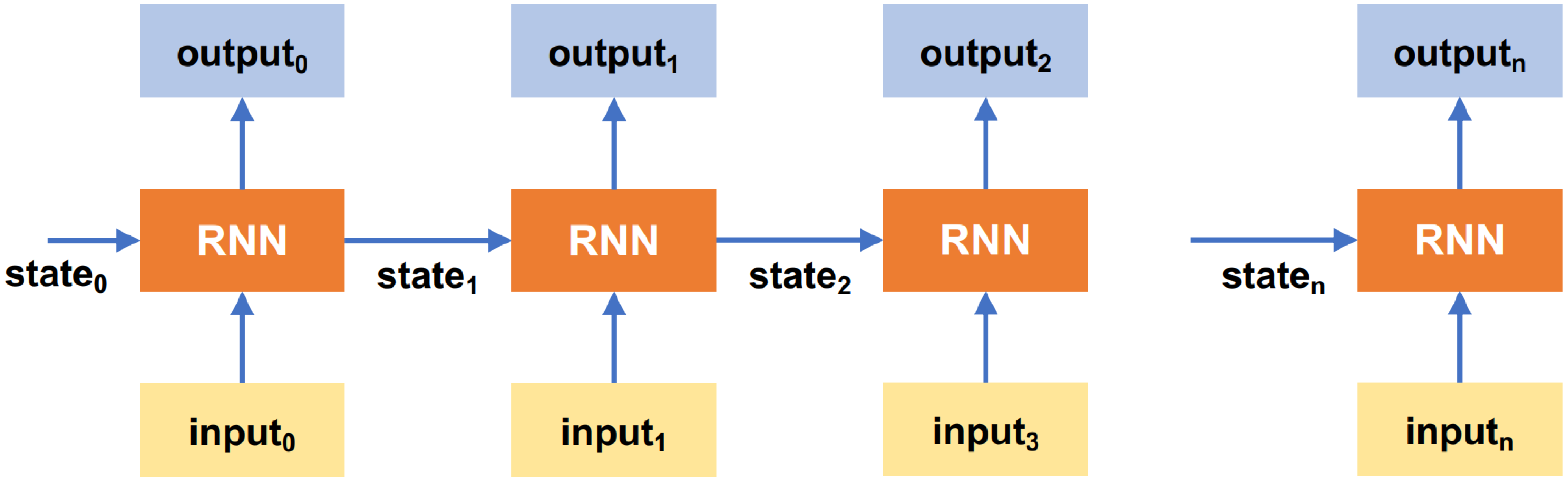
Wikipedia
https://en.wikipedia.org › wiki › Jürgen_Schmidhuber

Jürgen Schmidhuber - Wikipedia

About featured snippets • Feedback

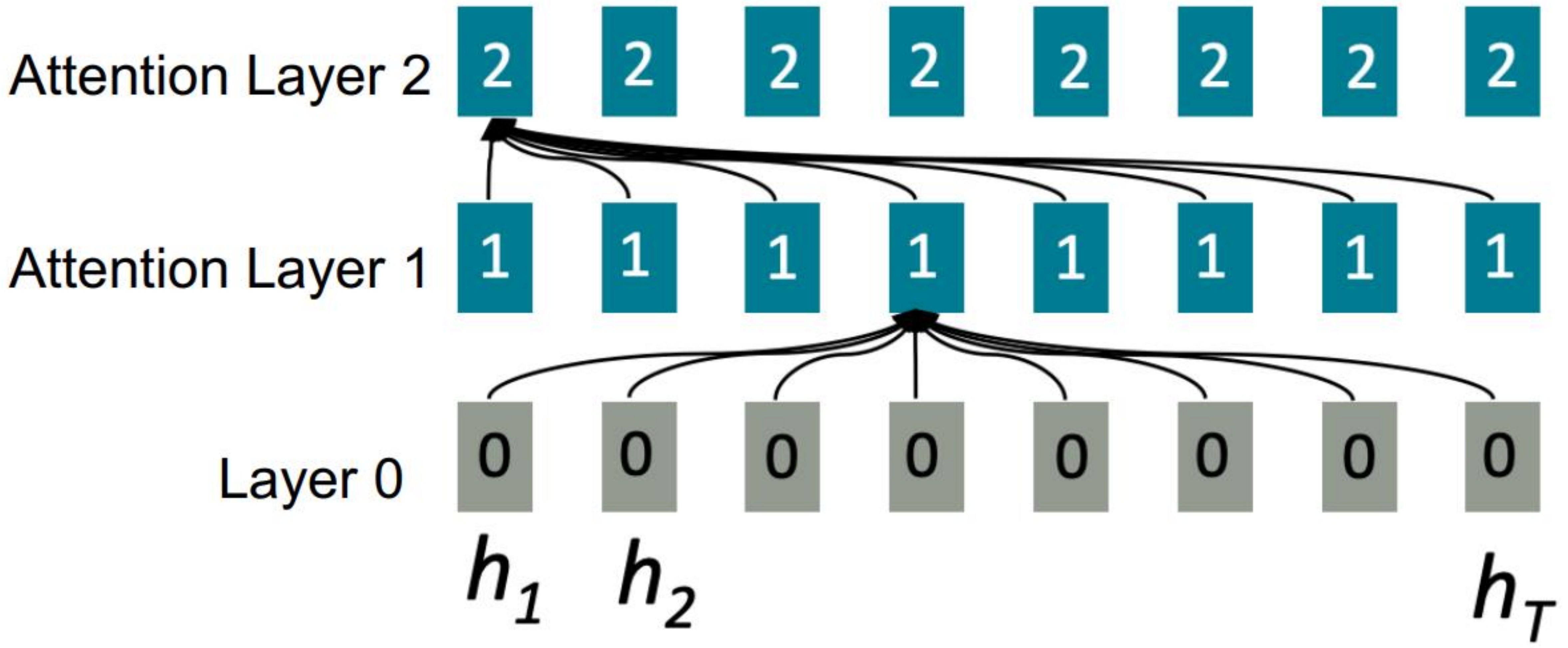
Two Key Problems of RNNs

- Problem 1: **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 sequence
- Problem 2: forgetting.



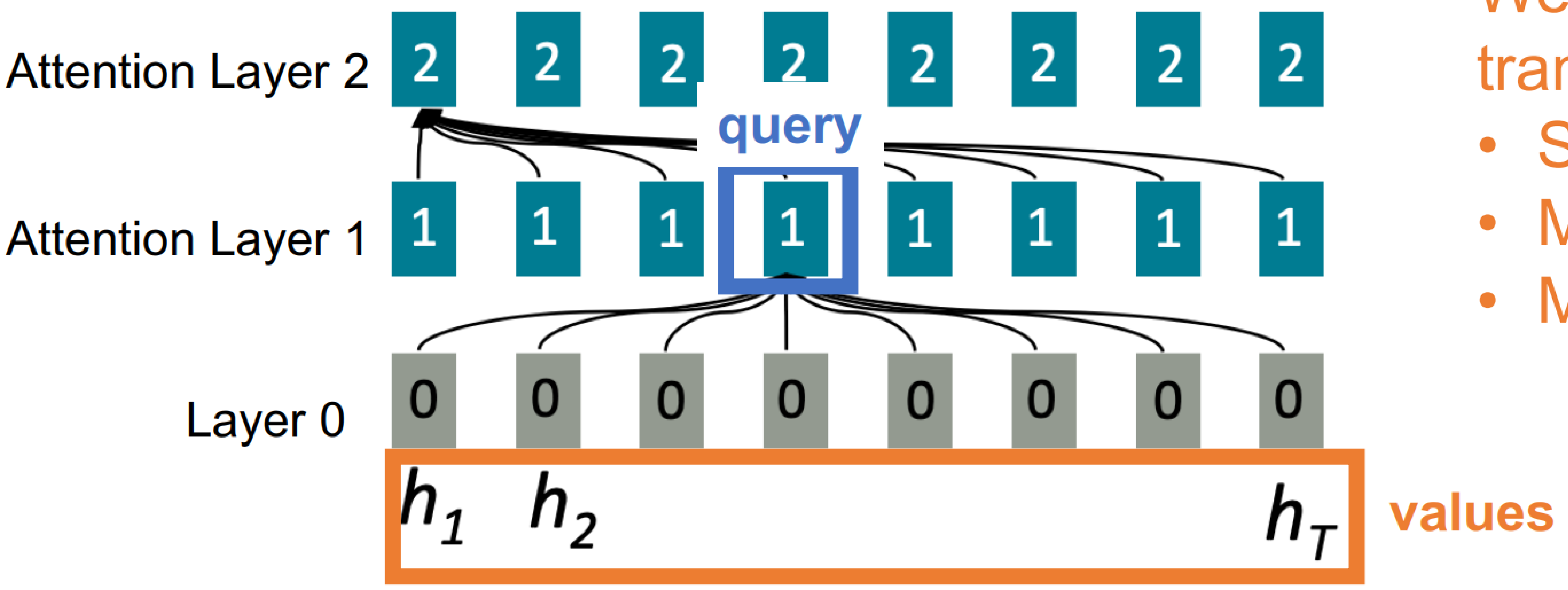
Attention: Enable parallelism

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



Attention

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



We will learn attention and transformers in depth later:

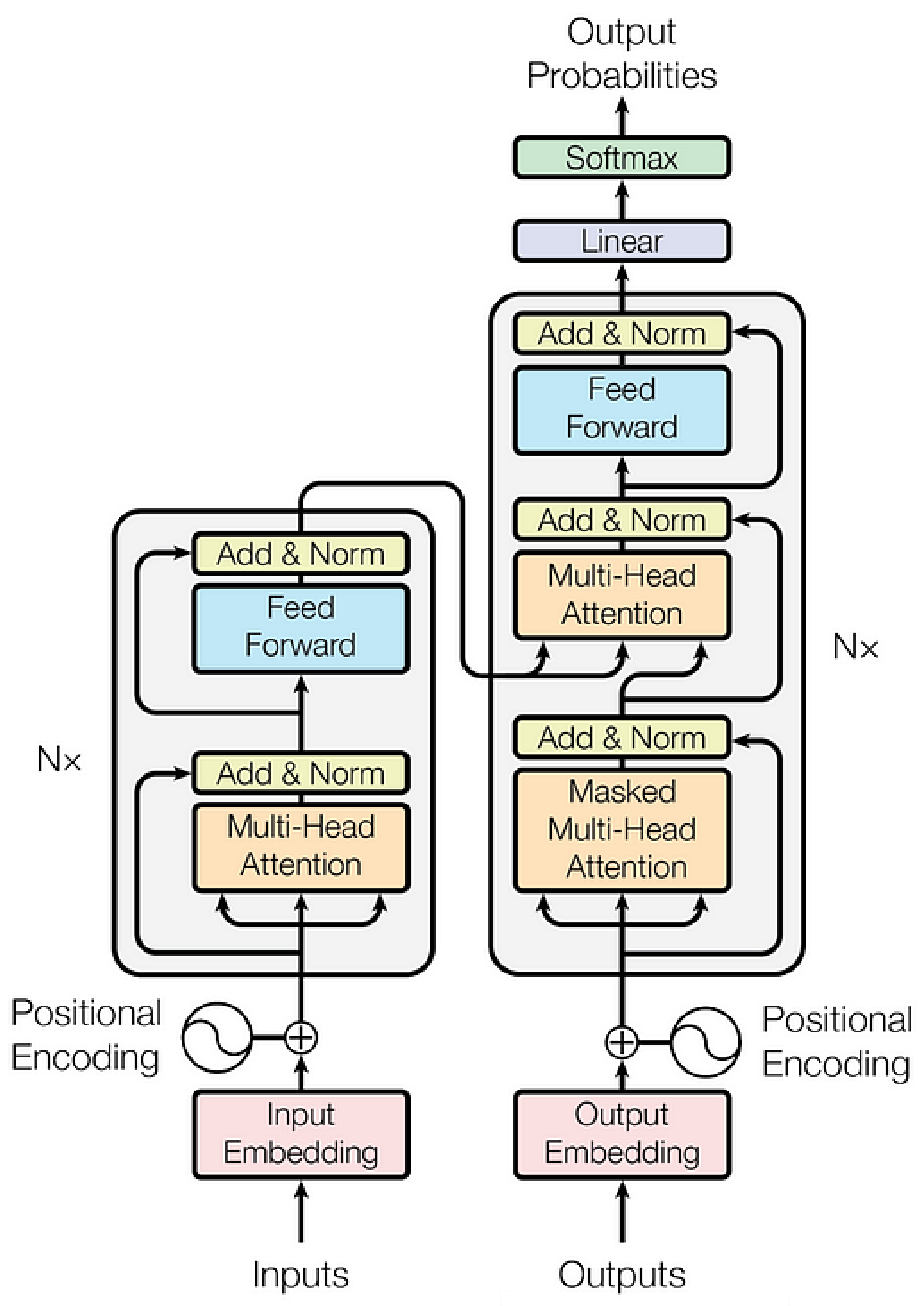
- Self-attention
- Masked attention
- Multi-head attention

Transformers

- Transformer = attention + a few MLPs

BERT

Encoder



GPT

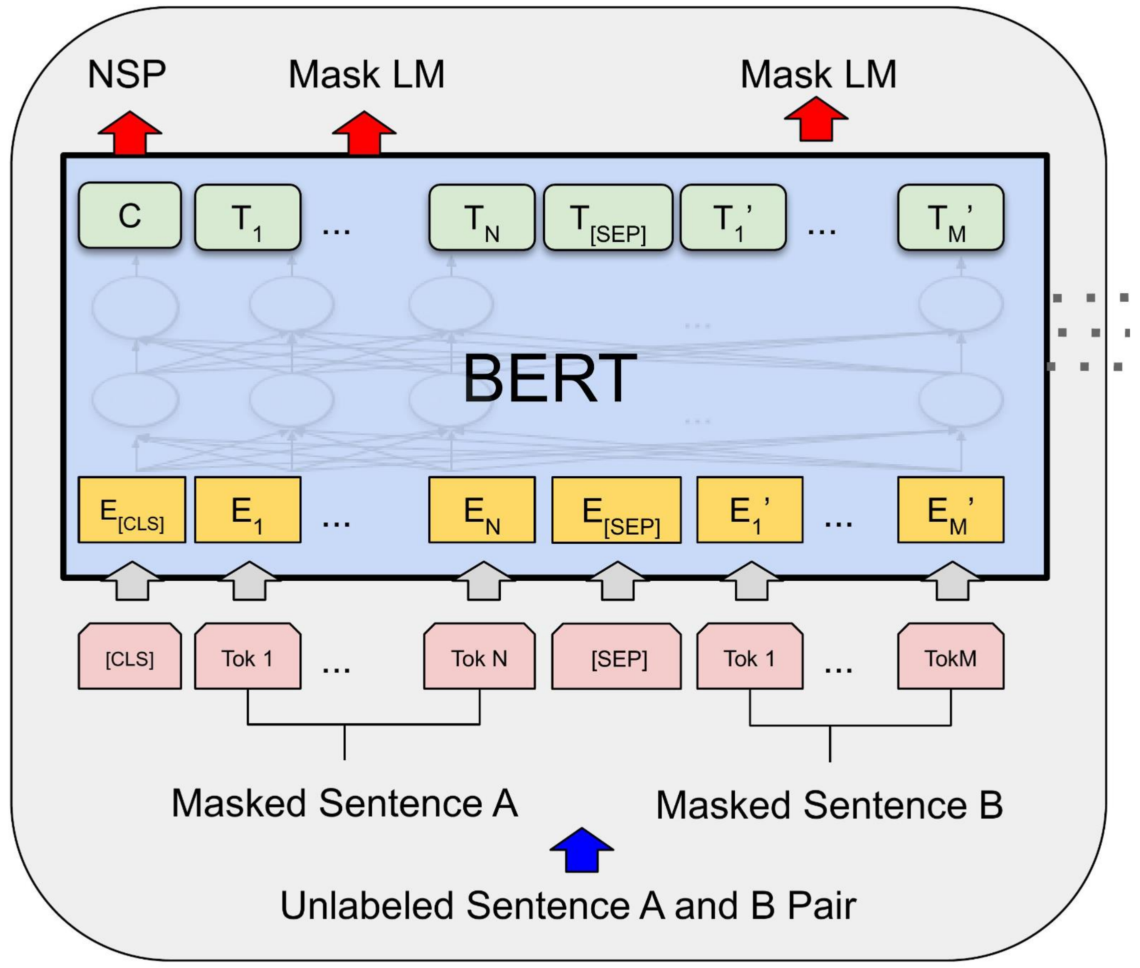
Decoder

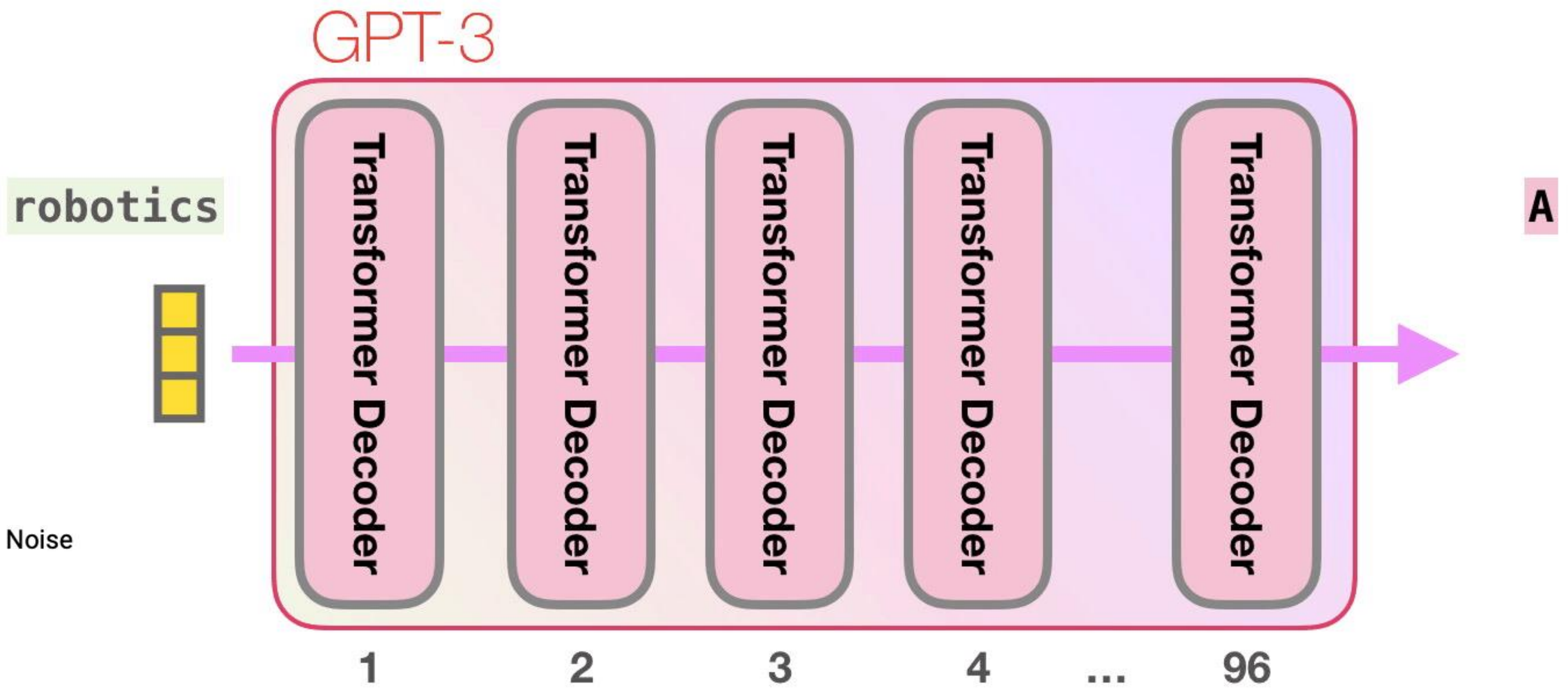
Most Important Components in attentions?

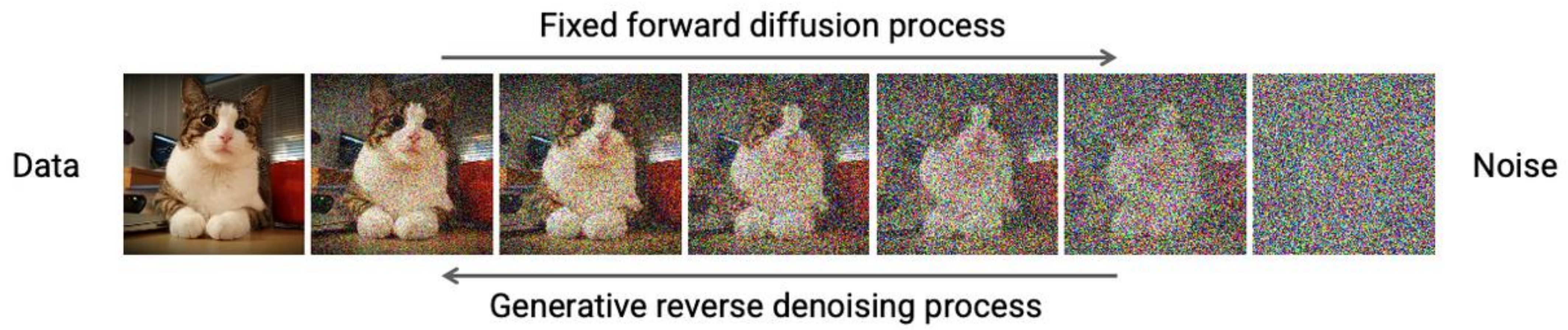
- Attention, which is composed by a set of
  - Matmul
  - Softmax
  - Normalization

Attention: top3 models

- Bert
- GPT/LLMs
- DiT: diffusion

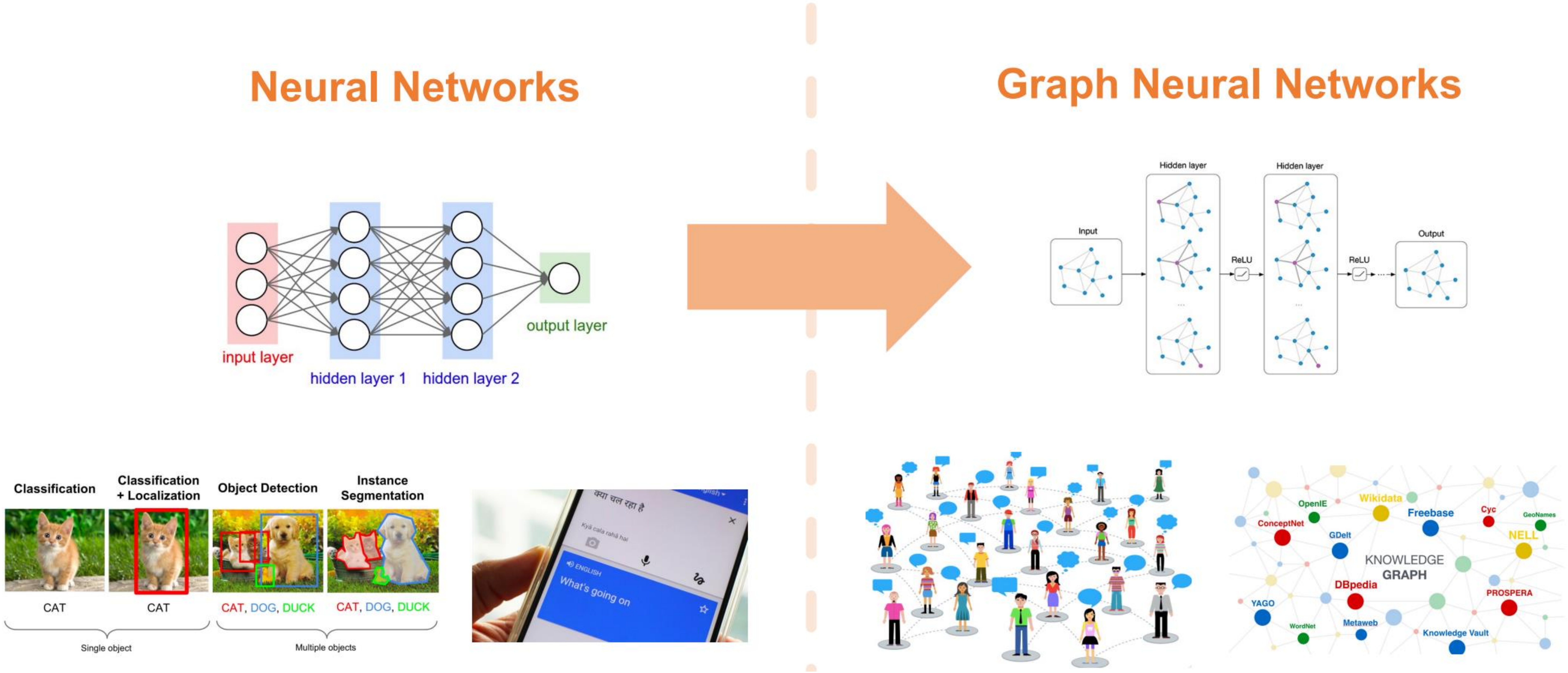




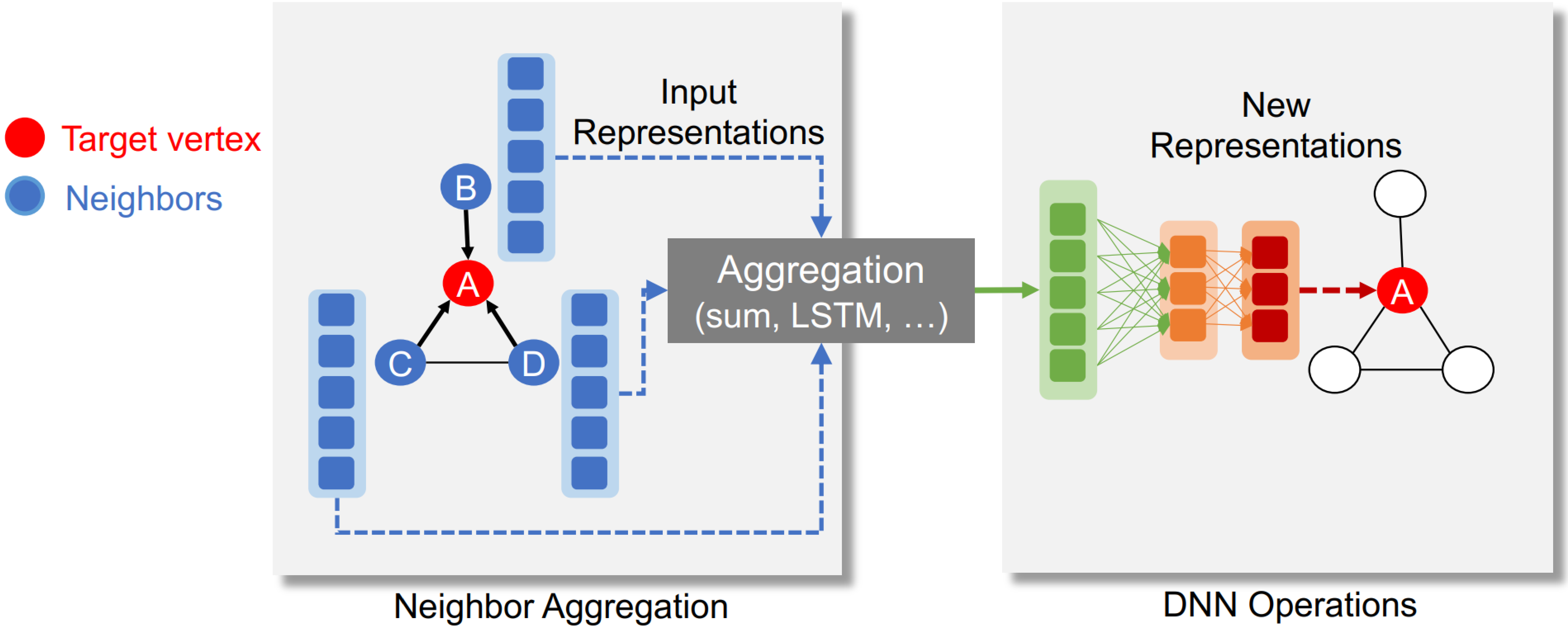


Graph Neural Networks

- Goal: model graph data



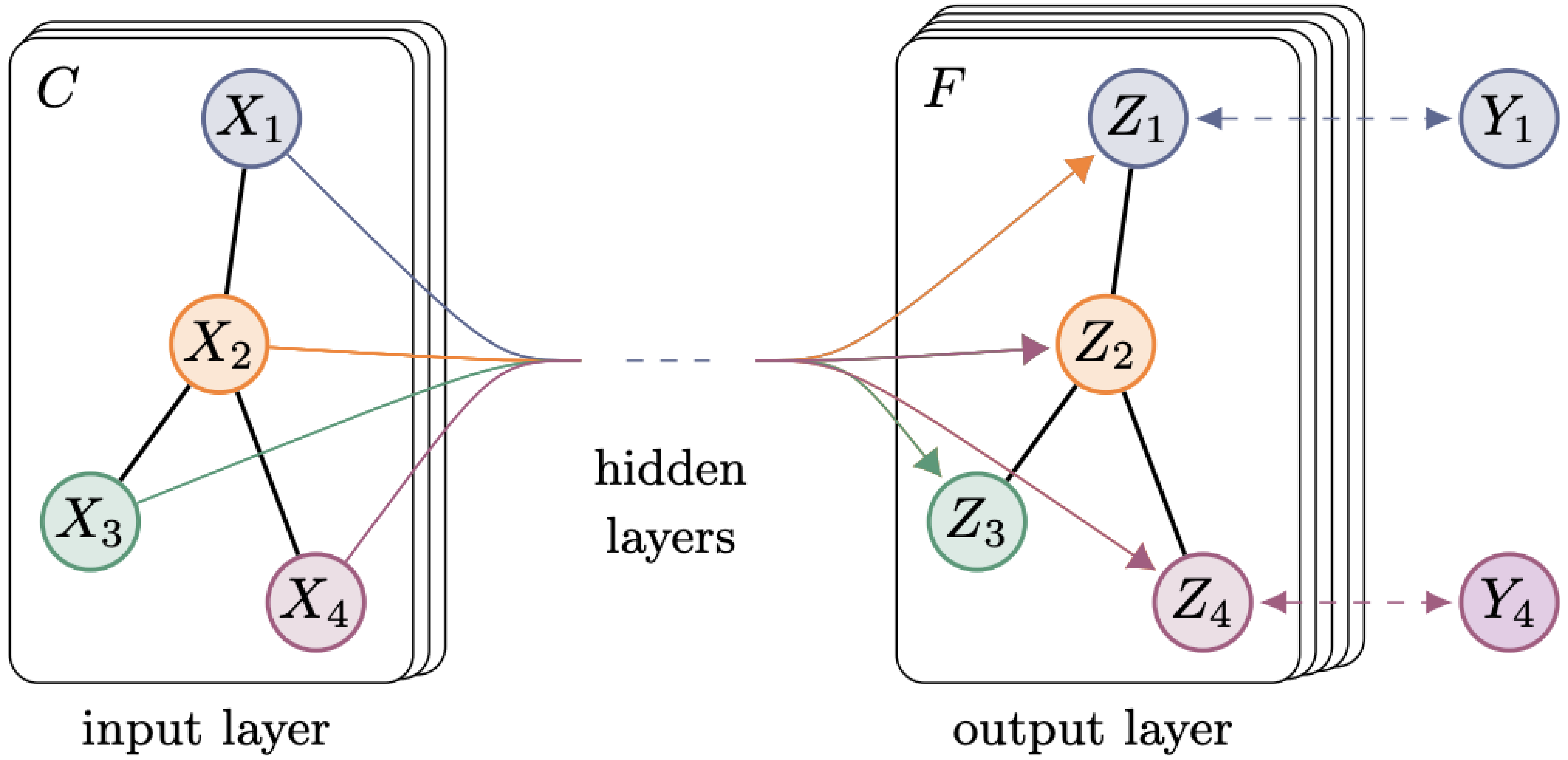
GNN Architecture



Questions

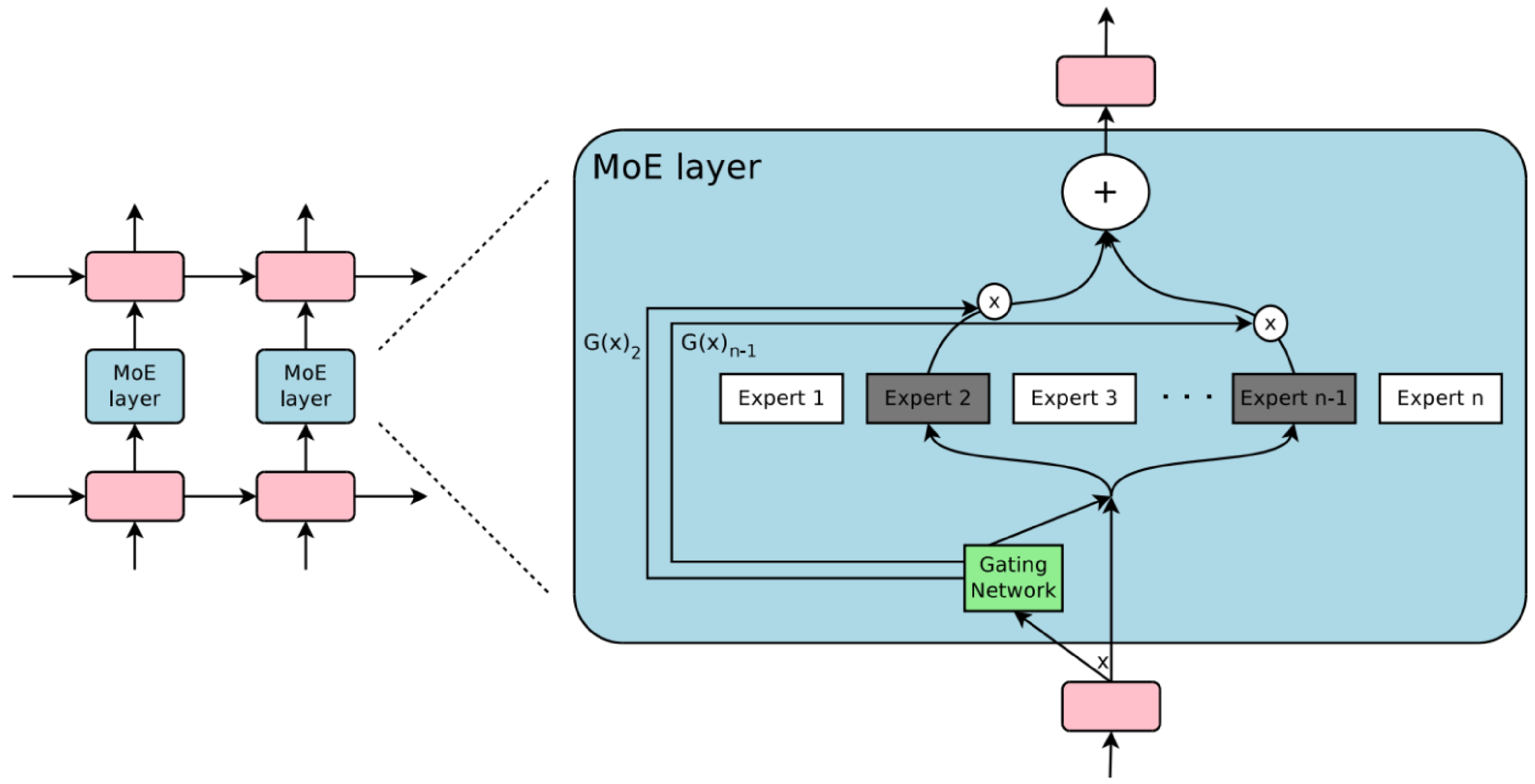
- Any novel component in Graph neural networks?
- Graph neural network vs. recurrent neural networks?

Top-1 GNNs: GCN Graph convolutional Networks



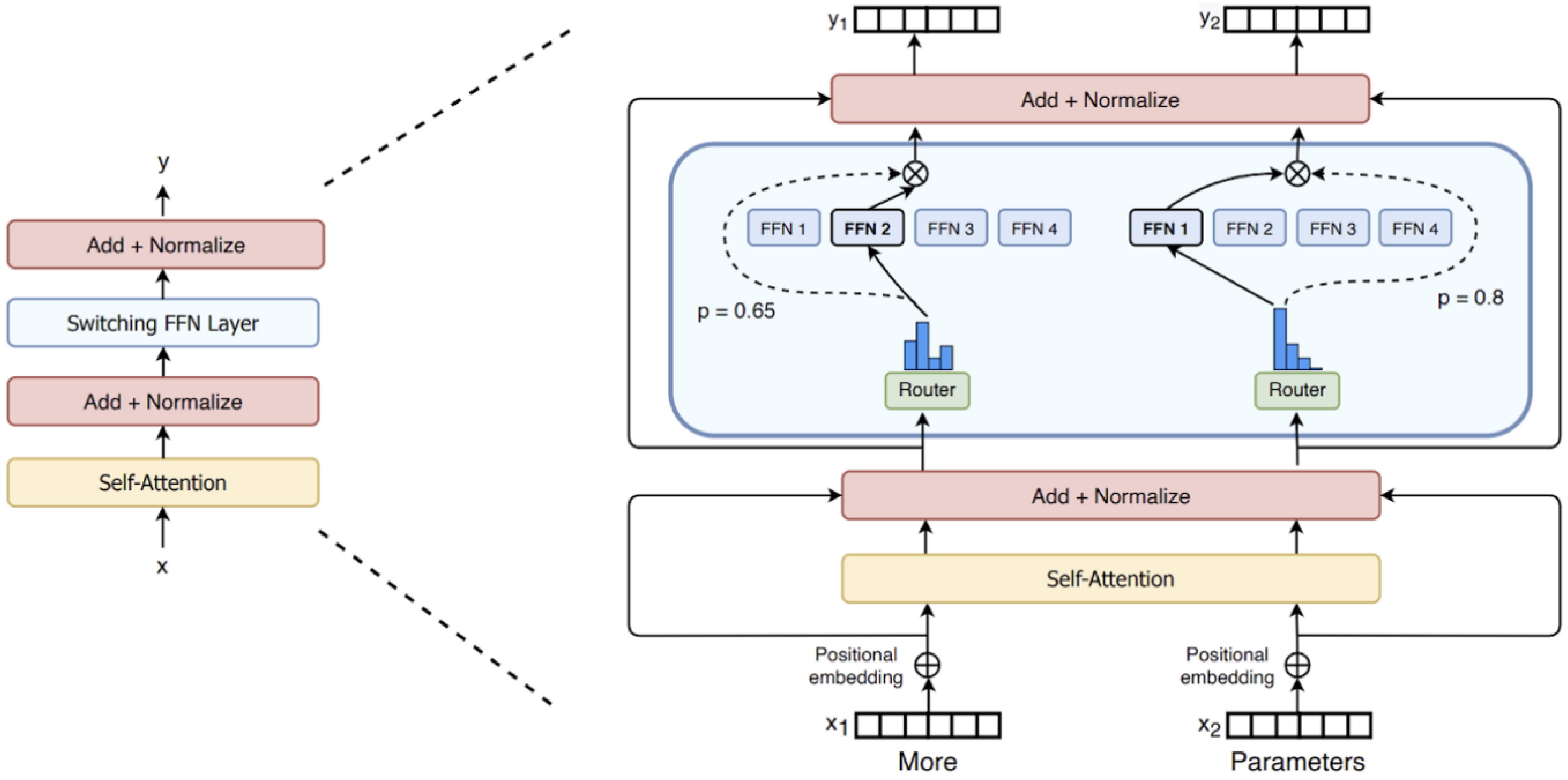
MoE: mixture of experts

- Ideas: More persons voting might be better than one person dictating
- Method: make each expert focus on predicting the right answer for a subset of cases



Novel Component in MoE?

- Latest LLMs are mostly MoEs
- Novel Components in MoE:
  - Router
- After-class Q:
  - Why router makes it hard



Summary of DL class in 30 mins

Matmul is all you need

Today

- Understand our Workloads: Deep Learning
- **Dataflow graph representation**
- Flavors of different ML frameworks

Static Graph vs. Dynamic Graph

- Goal: we want to express as many as model as possible using one set of programming interface
- Let’s abstract out all the components we need:
  - Model and architecture
  - Objective function
  - Optimization computation
    - dropout (part of model and architecture)
    - regularization (part of the objective)
  - Data
  - Hardare:CPUs/GPUs/TPUs/etc.

Applications <-> System Design

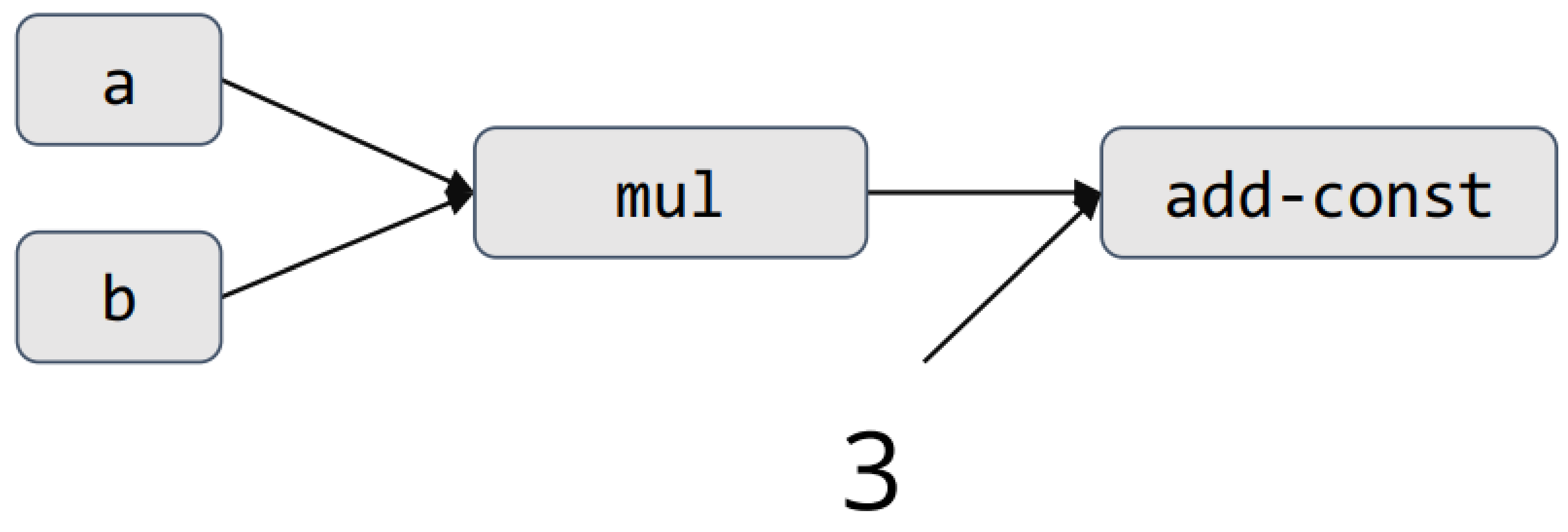
Application	Data management (OLTP)	Big data processing (OLAP)
Systems	SQL Query planner Relational database Storage	Spark/mapreduce Dataflow, lineage Data warehousing Column storage

Discussion: how can these ingredients affect the system design of ML frameworks

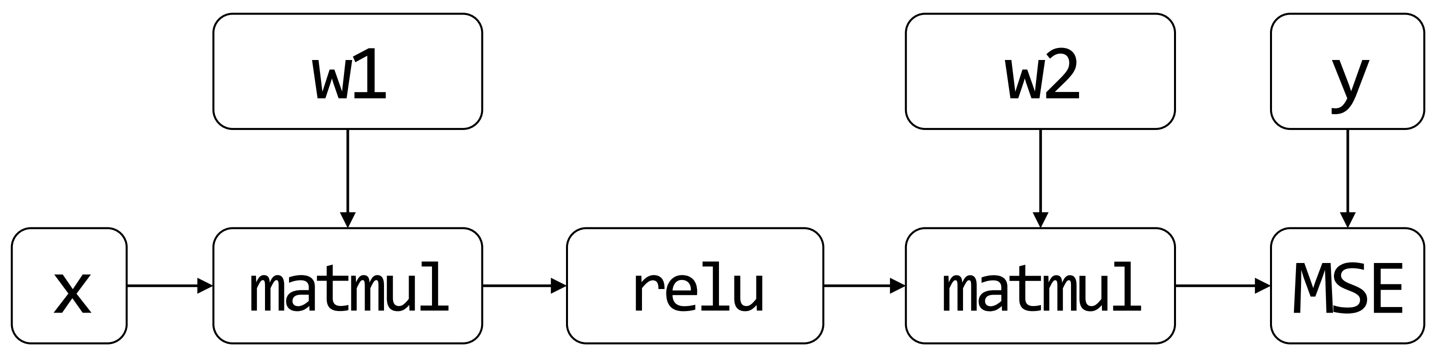
- Model and architecture
- Objective function
- Optimization computation
  - dropout (part of model and architecture)
  - regularization (part of the objective)
- Data
- Hardare:CPUs/GPUs/TPUs/etc.

Computational Dataflow Graph

- Node: represents the computation (operator)
- Edge: represents the data dependency (data flowing direction)
- Node: also represents the *output tensor* of the operator
- Node: also represents an input constant tensor (if it is not an compute operator)



a x b + 3



$$L = \mathrm{MSE}(w_2 \cdot \mathrm{ReLU}(w_1 x),\, y)$$

Case Study: TensorFlow Program

- In the next few slides, we will do a case study of a deep learning program using TensorFlow v1 style API (classic Flavor).
- Note that today most deep learning frameworks now use a different style, but share the same mechanism under the hood
- Think about abstraction and implementation when going through these examples

One linear NN: Logistic Regression

Input

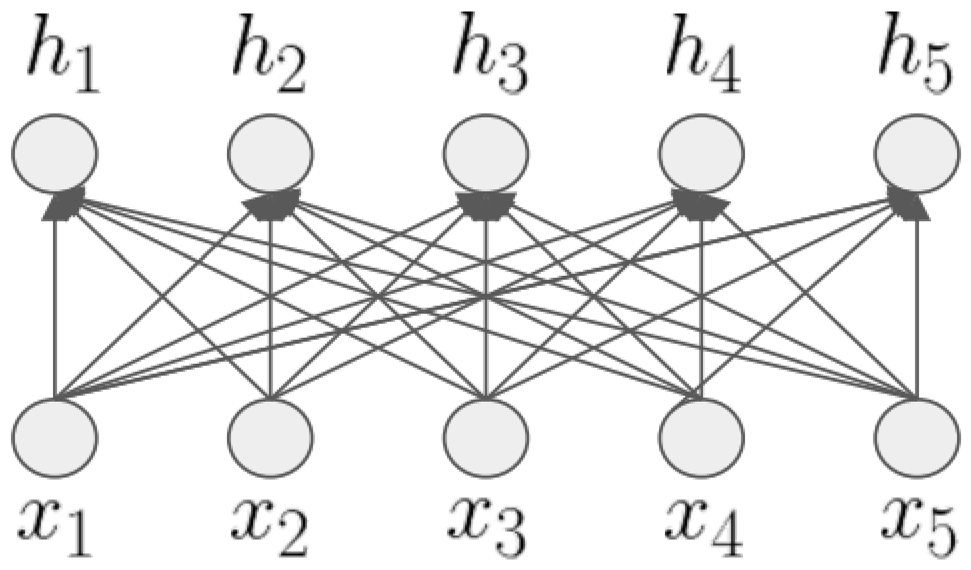
$$x_i = \begin{bmatrix} \text{pixel}_1 \\ \text{pixel}_2 \\ \cdots \\ \text{pixel}_m \end{bmatrix}$$

One Linear Layer

$$h_k = w_k^T x_i$$

Softmax

$$P(y_i = k | x_i) = \frac{\exp(h_k)}{\sum_{j=1}^{10} \exp(h_i)}$$



Whole Program

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
   batch_xs, batch_ys = mnist.train.next_batch(100)
   sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Forward Computation

Declaration

Loss Function

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
   batch_xs, batch_ys = mnist.train.next_batch(100)
   sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Loss function **Declaration**

$$P(\text{label} = k) = y_k$$

$$L(y) = \sum_{k=1}^{10} I(\text{label} = k) \log(y_i)$$

Auto-diff

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
   batch_xs, batch_ys = mnist.train.next_batch(100)
   sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Automatic Differentiation:
Next incoming topic

SGD Update

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

SGD update rule

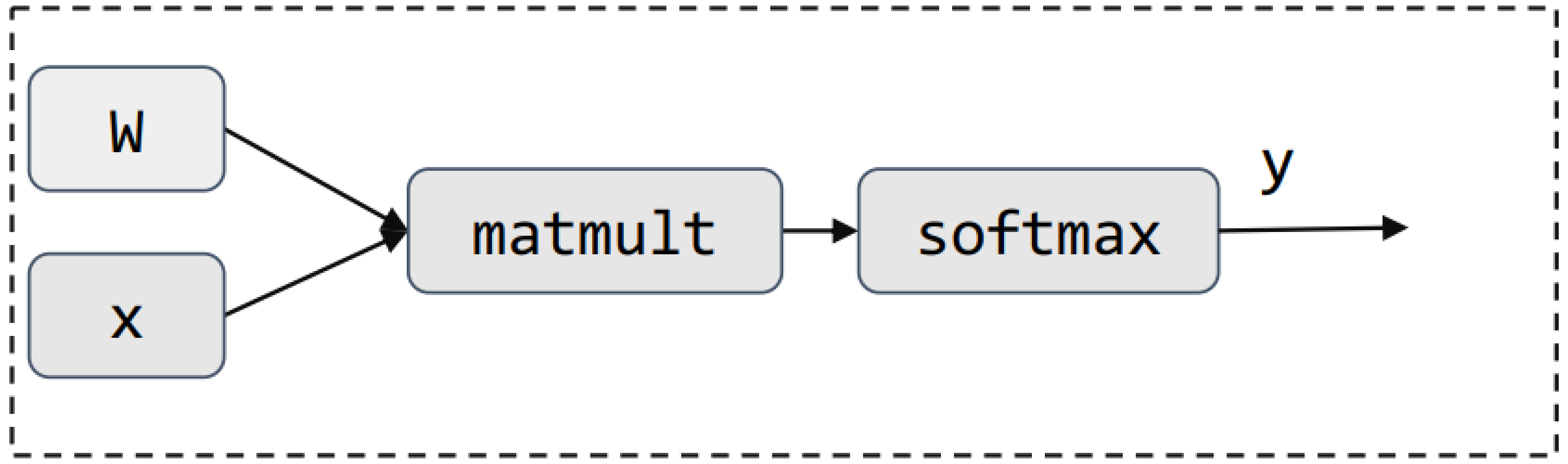
Trigger the Execution

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
   batch_xs, batch_ys = mnist.train.next_batch(100)
   sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Real execution happens here!

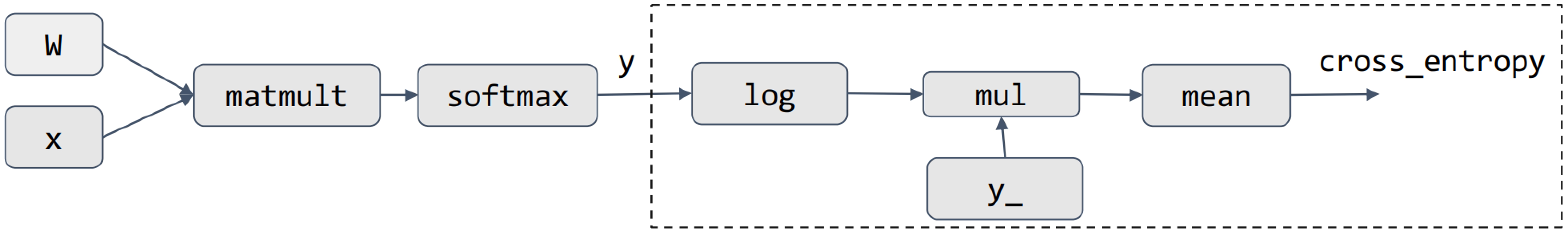
What happens behind the Scene

```
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
```



What happens behind the Scene (Cond.)

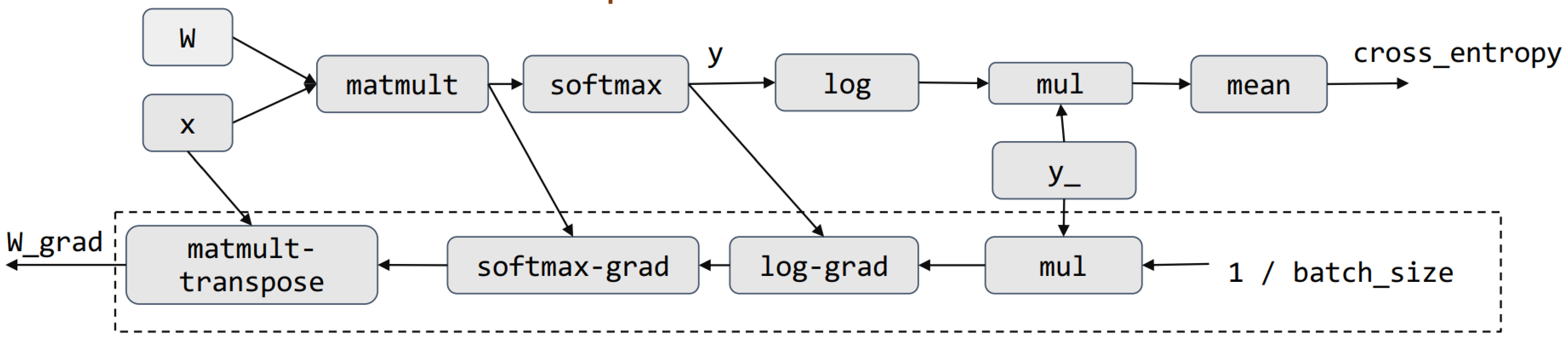
```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```



What happens behind the Scene (Cond.)

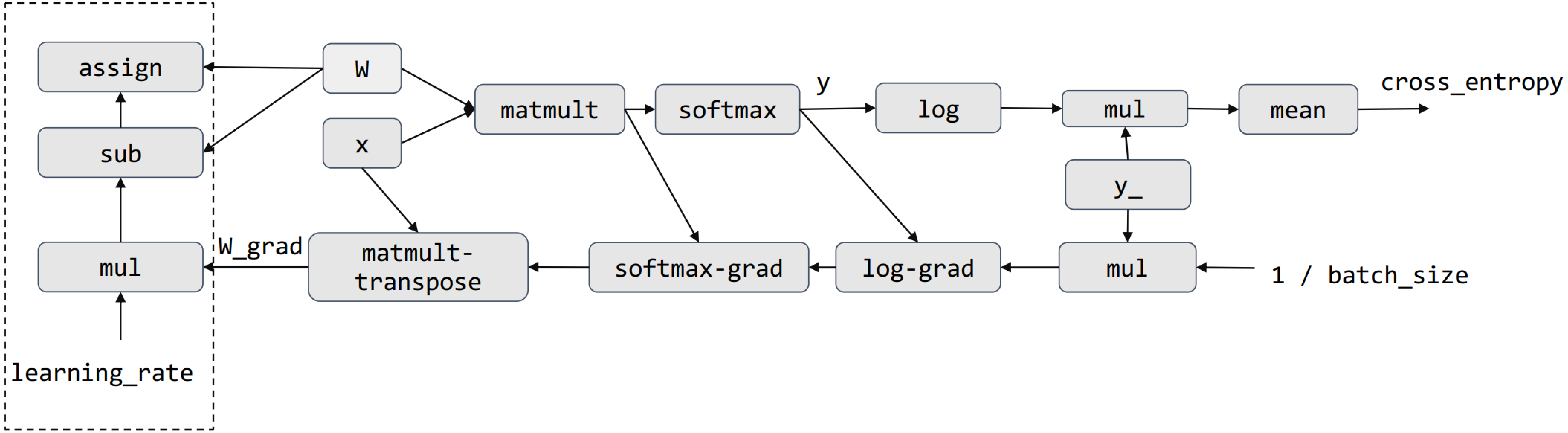
```
W_grad = tf.gradients(cross_entropy, [W])[0]
```

Automatic Differentiation, more details in follow up lectures



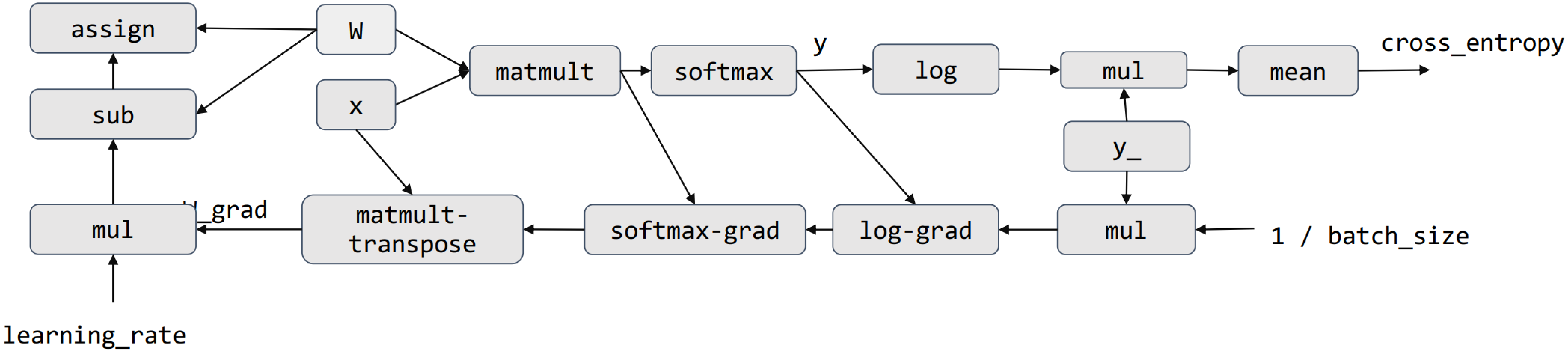
What happens behind the Scene (Cond.)

```
sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```



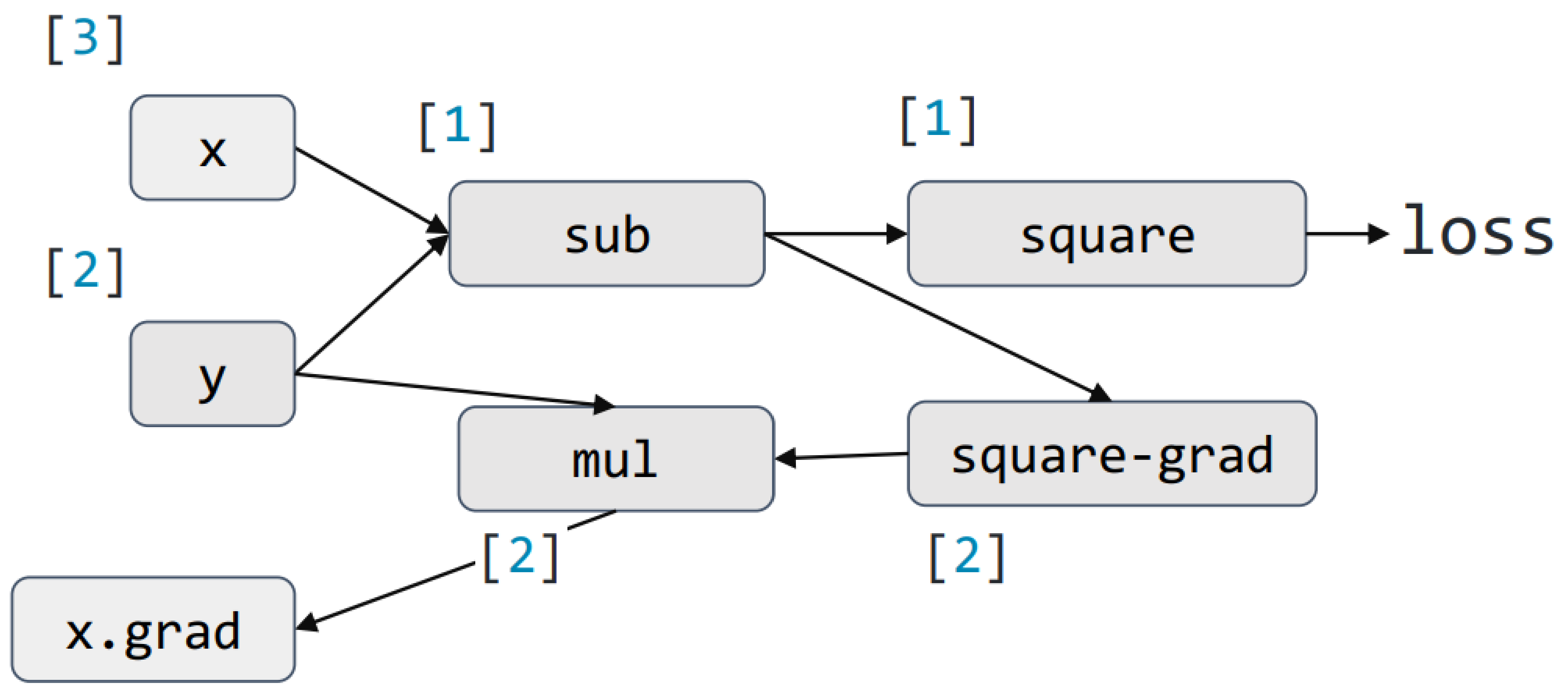
Discussion

- What are the benefits for computational graph abstraction?
- What are possible implementations and optimizations on this graph?
- What are the cons for computational graph abstraction?



A different flavor: PyTorch

```
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```



y.grad's path is omitted

Topic: Symbolic vs. Imperative

- Symbolic vs. imperative programming
- Define-then-run vs. define-and-run
- Define-then-run : write symbols to assemble the networks first, evaluate later
- define-and-run : immediate evaluation

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```

Symbolic

```
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```

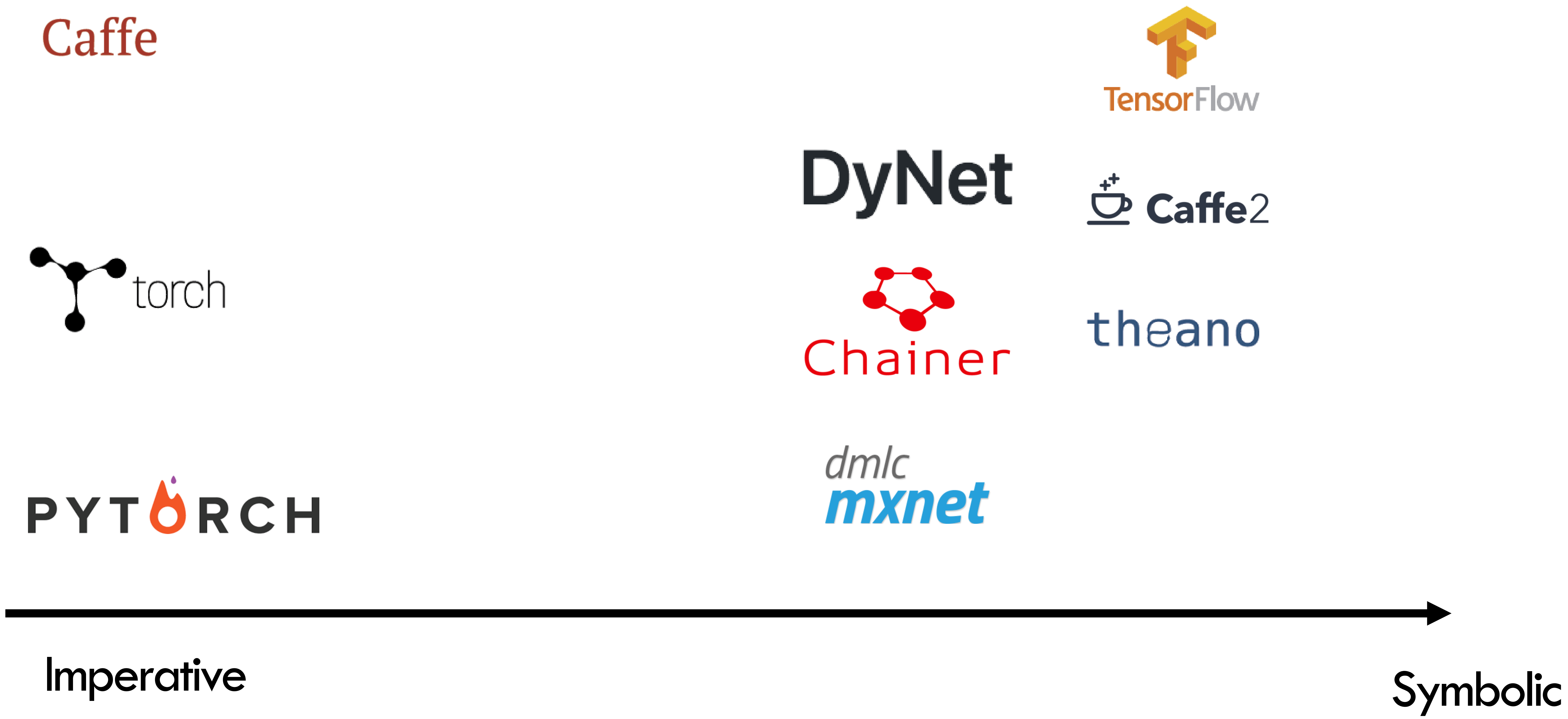
Imperative

Symbolic vs. Imperative

- Symbolic
  - Good
    - easy to optimize (e.g. distributed, batching, parallelization) for developers
    - More efficient
  - Bad
    - The way of programming might be counter-intuitive
    - Hard to debug for user programs
    - Less flexible: you need to write symbols before actually doing anything
- Imperative:
  - Good
    - More flexible: write one line, evaluate one line (that's why we all like Python)
    - Easy to program and easy to debug: because it matches the way we use C++ or python
  - Bad
    - Less efficient
    - More difficult to optimize

Symbolic vs. Imperative

- They are also designed differently
  - Symbolic v.s. imperative programming

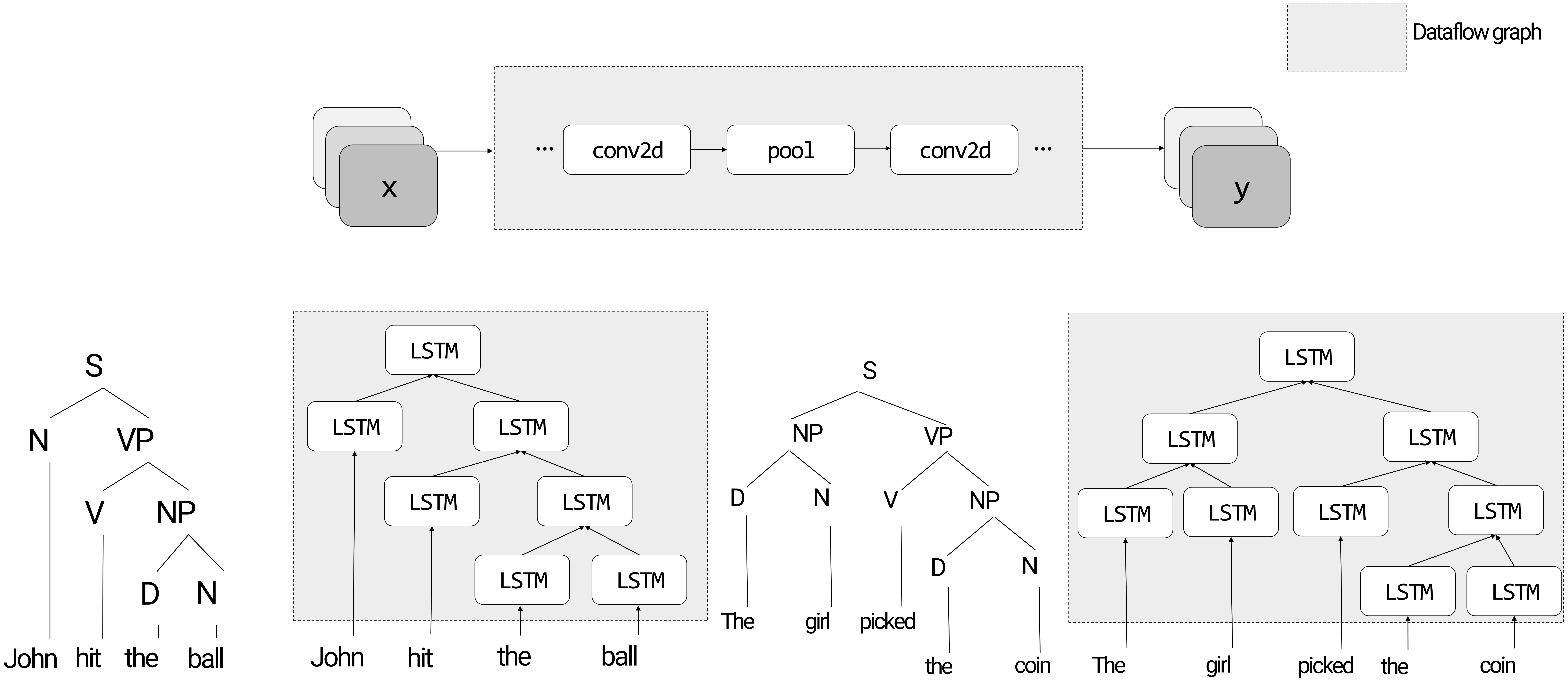


Just-in-time Compilation

- Ideally, we want define-and-run during ______
- We want define-then-run during _____
- Q: how can we have both without rewriting the program?

```
@torch.compile()
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```

Static Models vs. Dynamic Models



Static vs. Dynamic Dataflow Graphs

- Static Dataflow graphs
  - Define once, execute many times
  - Execution: Once defined, all following computation will **follow** the defined computation
  - Advantages
    - No extra effort for batching optimization, because it can be by nature batched
    - It is always easy to handle a static computational dataflow graphs in all aspects, because of its fixed structure
      - Node placement, distributed runtime, memory management, etc.
    - Benefit the developers

Static vs. Dynamic Dataflow Graphs

- Can we handle dynamic dataflow graphs?
  - Difficulty in expressing complex flow-control logic
  - Complexity of the computation graph implementation
  - Difficulty in debugging

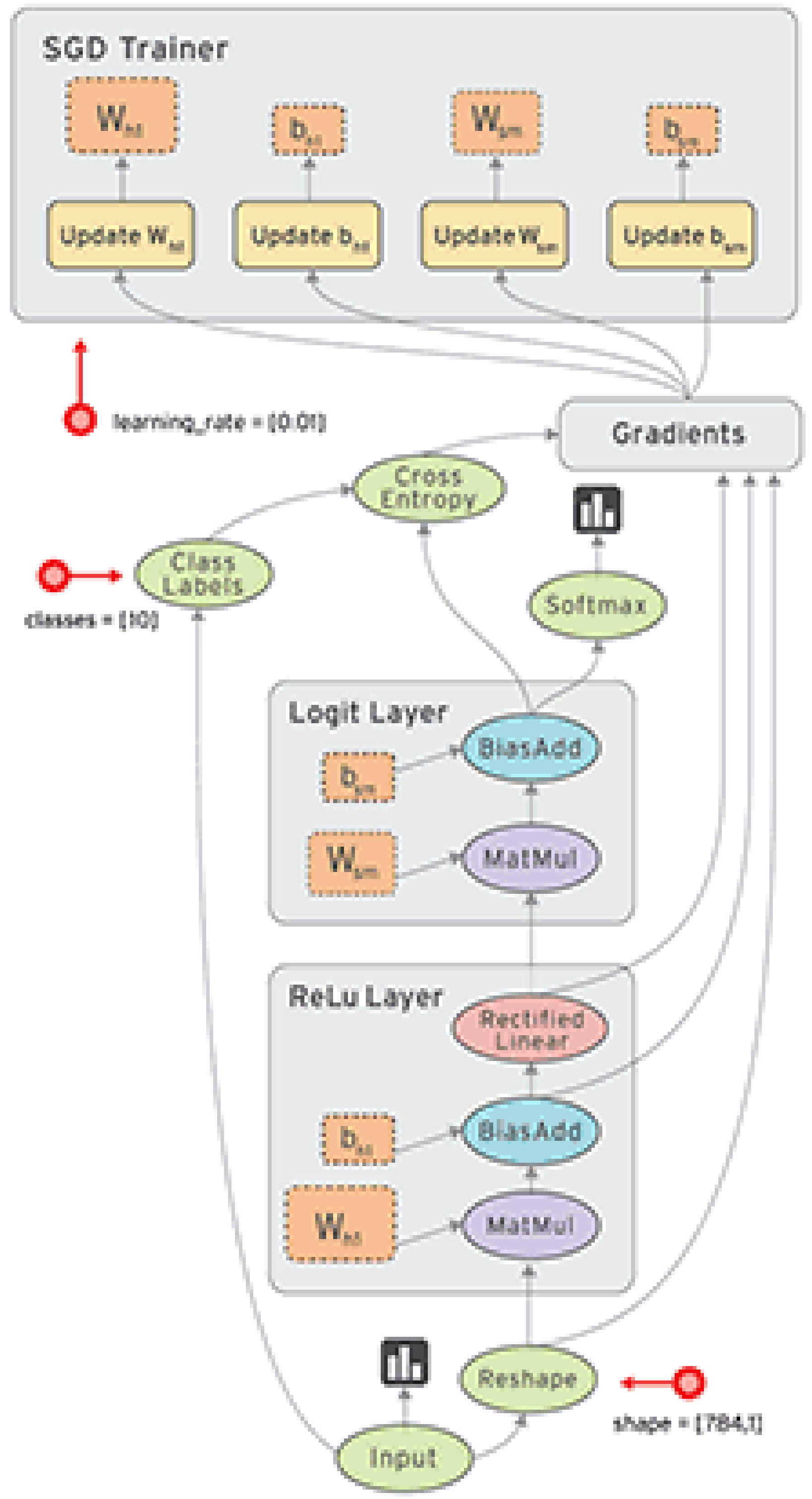
How to Handle Dynamic Dataflow Graph?

- In general two ways:
  - Define-and-run: do not requiring contracting the entire graph before execution
  - Constructing High-level symbols to absorb dynamics

Next week

- Autodiff
- ML System overview

Now we roughly have the problem



ML Systems