빅데이터와 머신러닝 소프트웨어

머신러닝/딥러닝 시스템

ML Software Stack

ML Application

Vision | Speech | Language

ML Core

HW (CPU, GPU, AIPU)

MEMO

ML Framework

High-level Structure

- Python frontend
  - define machine learning models (mostly neural nets)
- C++ backend: execute defined models
  - Single-machine: CPU(s), GPU(s), AIPU(s)
  - Distributed multi-machine

MEMO

ML Framework

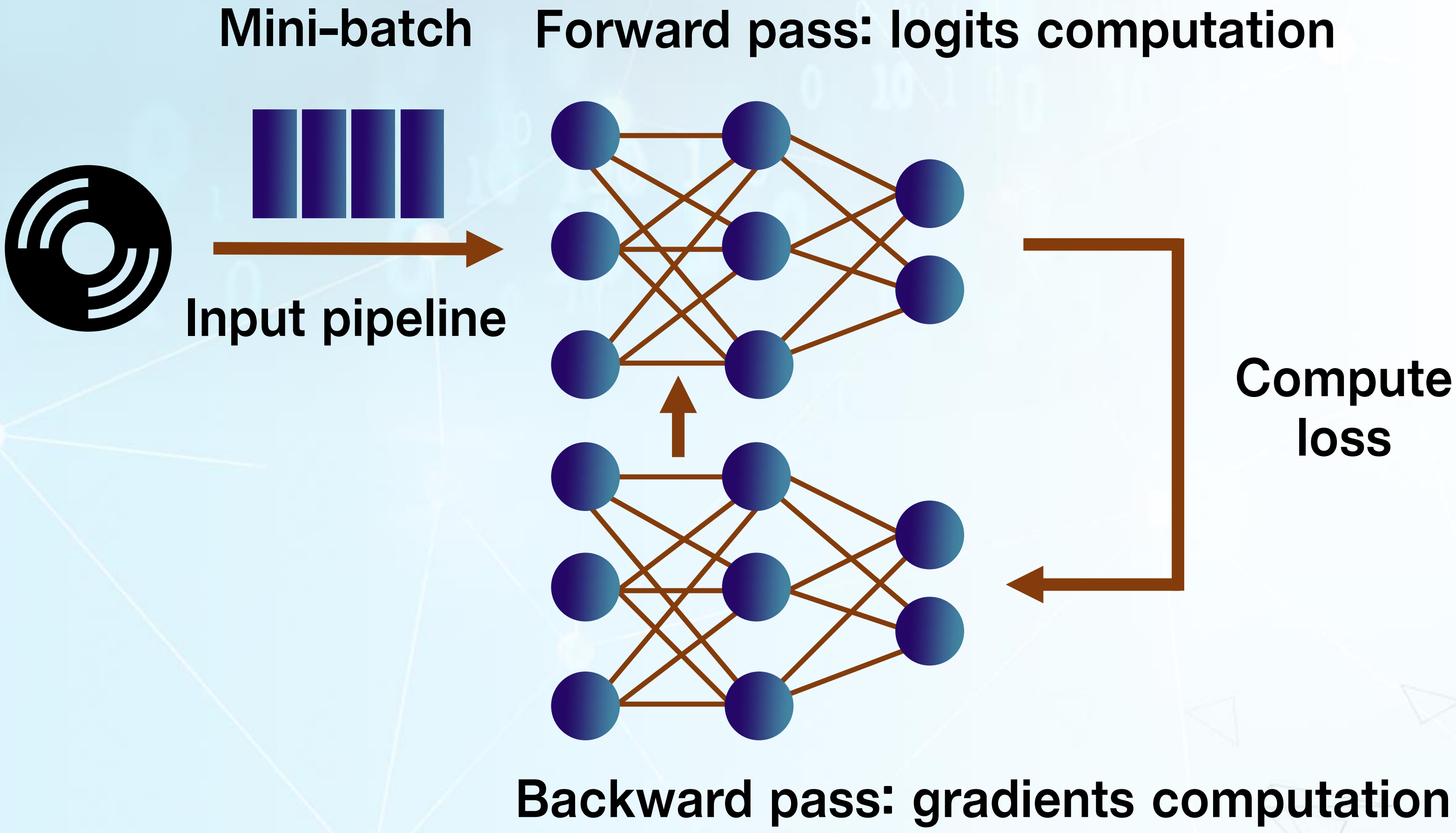
High-level Structure

- Describe mathematical computation
- Execute forward and backward computation with the given data batch
- Training: iterate execution with massive dataset to optimize a goal
  - The framework supports auto-differentiation
- Inference: execute once with a data input to predict

MEMO

ML Framework

- Training



MEMO

ML Framework

Auto-Differentiation

- $f(x, y) = xy^3 + x + 1$
- We need its partial derivative $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ to perform Stochastic Gradient Descent (or its variants)
- What we actually need is to compute the partial derivative of $f(x, y)$ with regards to specific x and y values

  (e.g., $\frac{\partial f}{\partial x}(4, 2)$ at x=4, y=2)

MEMO

ML Framework

Reverse-mode Autodiff

- Goes through the graph in the forward direction to compute the value of each node
- Does a second pass in the reverse direction to compute all the partial derivatives

MEMO

ML Framework

- **Auto-differentiation**
  - The user defines a model to compute the forward pass
  - The ML framework automatically adds gradients operations to compute the backward pass

MEMO

ML Framework Categories

	Symbolic ML Frameworks	Imperative ML Frameworks
Execution	Build a graph Execute the graph	Directly execute statements
Frameworks	TensorFlow, Caffe2, MXNet, CNTK	PyTorch, TensorFlow Eager, MXNet Imperative

* Python: De-facto deep learning programming language

MEMO

ML Framework Categories

	Symbolic ML Frameworks	Imperative ML Frameworks
Execution	Build a graph Execute the graph	Directly execute statements
Frameworks	TensorFlow, Caffe2, MXNet, CNTK	PyTorch, TensorFlow Eager, MXNet Imperative
Pros	Easy to optimize and deploy	Easy to program and debug
Cons	Hard to program and debug	Little room for optimization

MEMO

ML Framework Example

Google TensorFlow

- Symbolic ML framework
- Express numerical computation as a computation graph
  - Tensor: N-dimensional array
  - Variable: mutable state (e.g., parameters)
  - Operation: computation abstraction
- Tensors flow through the graph → TensorFlow

MEMO

ML Framework Example

Facebook PyTorch

- Imperative ML framework
- Express numerical computation like numpy
  - Tensor : imperative N-dimensional array but runs either on CPU or GPU
  - Variable : wrapper of Tensor. It builds a chain of operations between the tensors, so that the gradients can flow back
  - Operation : abstract computation (e.g., matrix multiply)

MEMO

Interoperability between Frameworks

Open Neural Network Exchange (ONNX)

- An open source format for AI models
- An extensible computation graph model and definitions of built-in operators and standard data types
- Caffe2, PyTorch, Cognitive Toolkit, MXNet support ONNX

MEMO

Summary

- ML frameworks
- Symbolic ML frameworks and imperative ML frameworks
- ML framework examples

MEMO

빅데이터와 머신러닝 소프트웨어

텐서플로우

TensorFlow

- **TensorFlow 1.x default mode**
  - symbolic graph style
- **Express numerical computation as a computation graph**
  - Node: operation which has any number of inputs and outputs
  - Edge: tensor which flow between nodes
- **Tensor: N-dimensional array**
  - 1-dimension: Vector
  - 2-dimension: Matrix
  - E.g., image represented as 3-d tensor rows, cols, color
- **Tensors flow through the graph → TensorFlow**

MEMO

Google TensorFlow

- The graph's compiled to CPU / GPU / AIPU code
- Salient features of TensorFlow graphs
  - Fine-grained ops
  - Dynamic control flow: condition, loop
  - Persistent state maintenance/update

MEMO

TensorFlow Programming Model

- **Graph: model computation**
  - ► Tensor: data (N-dimensional array)
  - ► Variable: returns a handle to a persistent mutable tensor that survives across executions of a graph.
  - ► Operation: abstract computation(e.g., matrix multiply)
    - Kernel: a particular implementation of an operation that can be run on a particular type of device(e.g., CPU or GPU)
- **Session: runs a graph**

MEMO

TensorFlow Programming Model

Symbolic Graph Style

- Step 1
  - Define a graph, which contains model architecture, parameter specifications, optimization process, etc.
- Step 2
  - Run the graph through a session (Session.run()), a binding to a particular execution context (e.g. CPU, GPU)
    - Initialize the session
    - Feed data and fetch results

MEMO

TensorFlow Eager Mode

- Statements directly executed without the separation of graph definition and execution

- Write code that you can easily execute in a REPL

MEMO

TensorFlow Programming Example

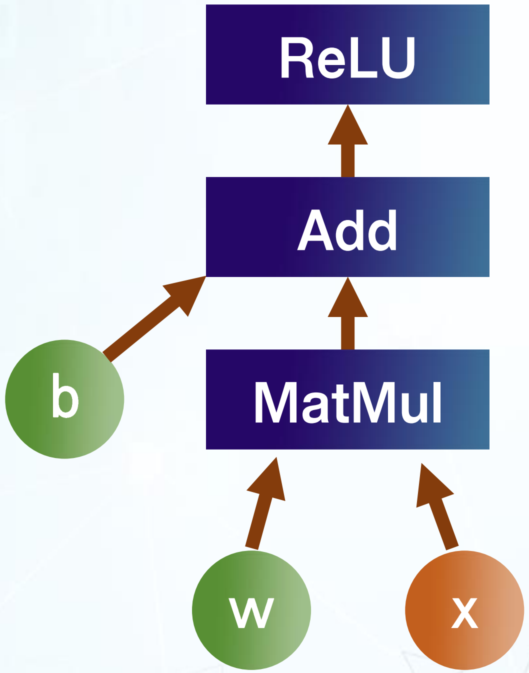
Define a graph: h = ReLU(Wx + b)

```
import tensorflow as tf

b = tf.get_variable('bias', tf.zeros((100,)))
W = tf.get_variable('weights',
        tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (None, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```



MEMO

TensorFlow Programming Example

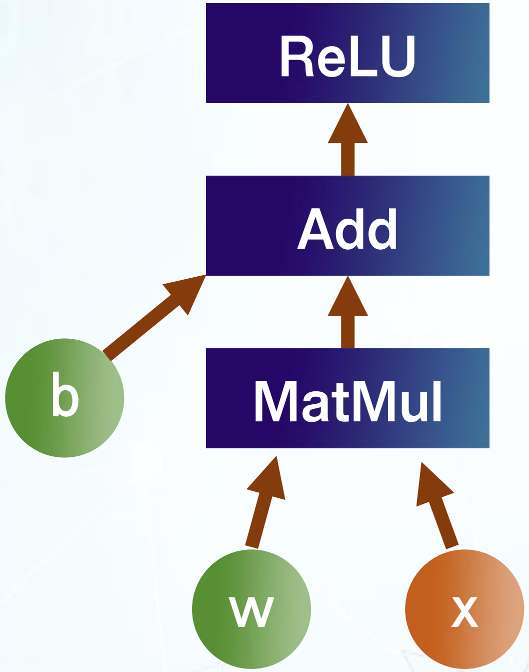
Define a graph

```
import tensorflow as tf

b = tf.get_variable('bias', tf.zeros((100,)))
W = tf.get_variable('weights',
        tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (None, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```



MEMO

TensorFlow Programming Example

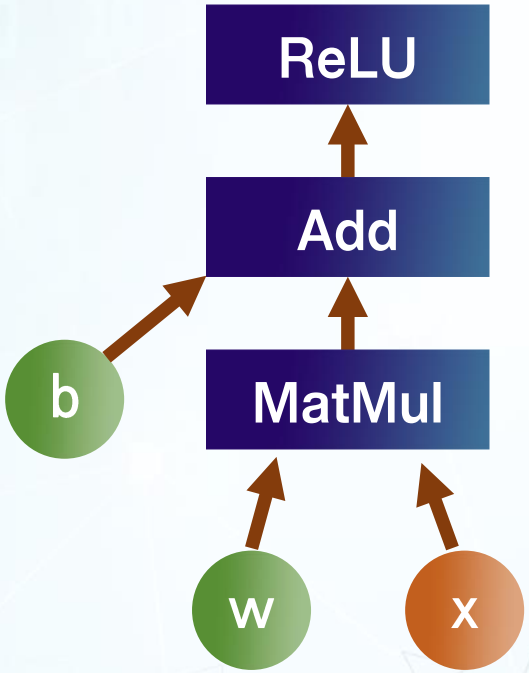
Define a graph

```
import tensorflow as tf

b = tf.get_variable('bias', tf.zeros((100,)))
W = tf.get_variable('weights',
        tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (None, 784))

h = tf.nn.relu(tf.matmul(x, W) + b)
```



MEMO

TensorFlow Programming Example

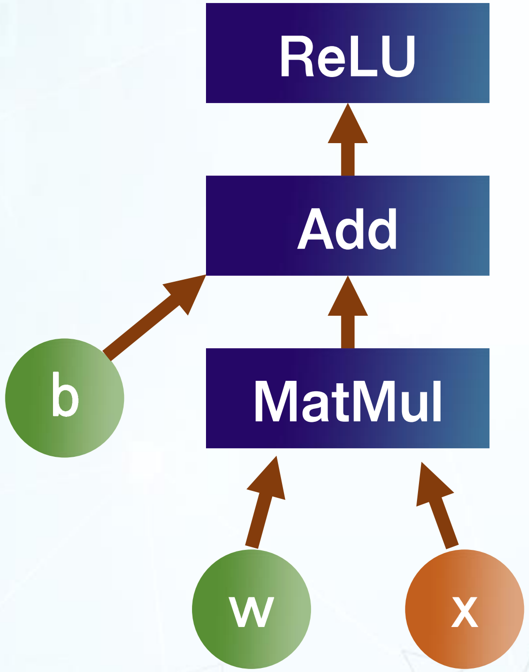
Define a graph

```
import tensorflow as tf

b = tf.get_variable('bias', tf.zeros((100,)))
W = tf.get_variable('weights',
        tf.random_uniform((784, 100), -1, 1))

x = tf.placeholder(tf.float32, (None, 784))
```

$\boxed{\texttt{h = tf.nn.relu(tf.matmul(x, W) + b)}}$



MEMO

TensorFlow Programming Example

- We can deploy the graph with a session

```
import tensorflow as tf

import numpy as np

b = tf.get_variable('bias', tf.zeros((100,)))
W = tf. get_variable('weights',
        tf.random_uniform((784, 100), -1, 1))
x = tf.placeholder(tf.float32, (None, 784))
h = tf.nn.relu(tf.matmul(x, W) + b)


sess = tf.Session()
sess.run(tf. global_variables_initializer())
sess.run(h, {x: np.random.random(64, 784)})
```

MEMO

TensorFlow Eager Mode

- Enabling eager execution requires two lines of code

```
import tensorflow as tf
import tensorflow.contrib.eager as tfe
tfe.enable_eager_execution() # Call this at program start-up
```

- Lets you write code that you can easily execute in a REPL

```
x = [[3.]]  # No need for placeholders!
m = tf.matmul(x, x)
print(m)  # No sessions!
# tf.Tensor([[9.]], shape=(1, 1), dtype=float32)
```

MEMO

TensorFlow Example

Linear Regression

```
import tensorflow as tf
import utils
DATA_FILE = "data/system_cpuutil_applatency.txt"

# Step 1: read in data from the .txt file
# data is a numpy array of shape (100000, 2), each row is a datapoint
data, n_samples = utils.read_system_cpuutil_applatency(DATA_FILE)

# Step 2: create placeholders for X (CPU util) and Y (app latency)
X = tf.placeholder(tf.float32, name='X')
Y = tf.placeholder(tf.float32, name='Y')
```

MEMO

TensorFlow Example

Linear Regression

```
# Step 3: create weight and bias, initialized to 0
w = tf.get_variable('weights', initializer=tf.constant(0.0))
b = tf.get_variable('bias', initializer=tf.constant(0.0))

# Step 4: construct model to predict Y (app latency from CPU util)
Y_predicted = w * X + b

# Step 5: use the square error as the loss function
loss = tf.square(Y - Y_predicted, name='loss')

# Step 6: using gradient descent with learning rate of 0.001 to minimize loss
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
                 .minimize(loss)
```

MEMO

TensorFlow Example

Linear Regression

```
with tf.Session() as sess:
    # Step 7: initialize the necessary variables, in this case, w and b
    sess.run(tf.global_variables_initializer())

    # Step 8: train the model
    for i in range(100): # run 100 epochs
        for x, y in data:
            # Session runs train_op to minimize loss
            sess.run(optimizer, feed_dict={X: x, Y:y})

    # Step 9: output the values of w and b
    w_out, b_out = sess.run([w, b])
```

MEMO

Input Pipeline with TensorFlow Dataset

```
dataset = tf.data.FixedLengthRecordDataset([file1, file2, file3, ...])

iterator = dataset.make_one_shot_iterator()
input, label = iterator.get_next()
...
for i in range(100):
    ...
    try:
        while True:
            sess.run([optimizer])
    except tf.errors.OutOfRangeError:
        pass
```

MEMO

Input Pipeline with TensorFlow Dataset

- **Shuffle, repeat, batch your data**

```
dataset = dataset.shuffle(1000)
dataset = dataset.repeat(100)
dataset = dataset.batch(128)
```

- **Map each element of your dataset to transform it in a specific way to create a new dataset**

```
dataset = dataset.map(lambda x: tf.one_hot(x, 10))
# convert each element of dataset to one_hot vector
```

MEMO

Summary

- TensorFlow programming model
- TensorFlow basic program example
- TensorFlow linear regression example
- TensorFlow Dataset

MEMO