CS6421: Deep Neural Networks

Gregory Provan

Spring 2020

Lecture 7: Computational Platforms

Based on notes from Jeff Heaton,
T81-558: Applications of Deep Neural Networks)

Overview

- Computational platforms for deep learning
- Underlying computational framework
  - Computation graph
- Comparison of tools
- In-depth: TensorFlow

Deep Learning Frameworks

- TensorFlow/Keras
  - scientific computing framework in Python
  - symbolic computation and automatic differentiation
  - Supported by Google
- PyTorch
  - scientific computing framework (originally in Lua)
  - supported by Facebook
- Caffe2
  - supported by Facebook
- CNTK
  - Microsoft product
  - Support for Windows/Linux, command line only. GPU support.

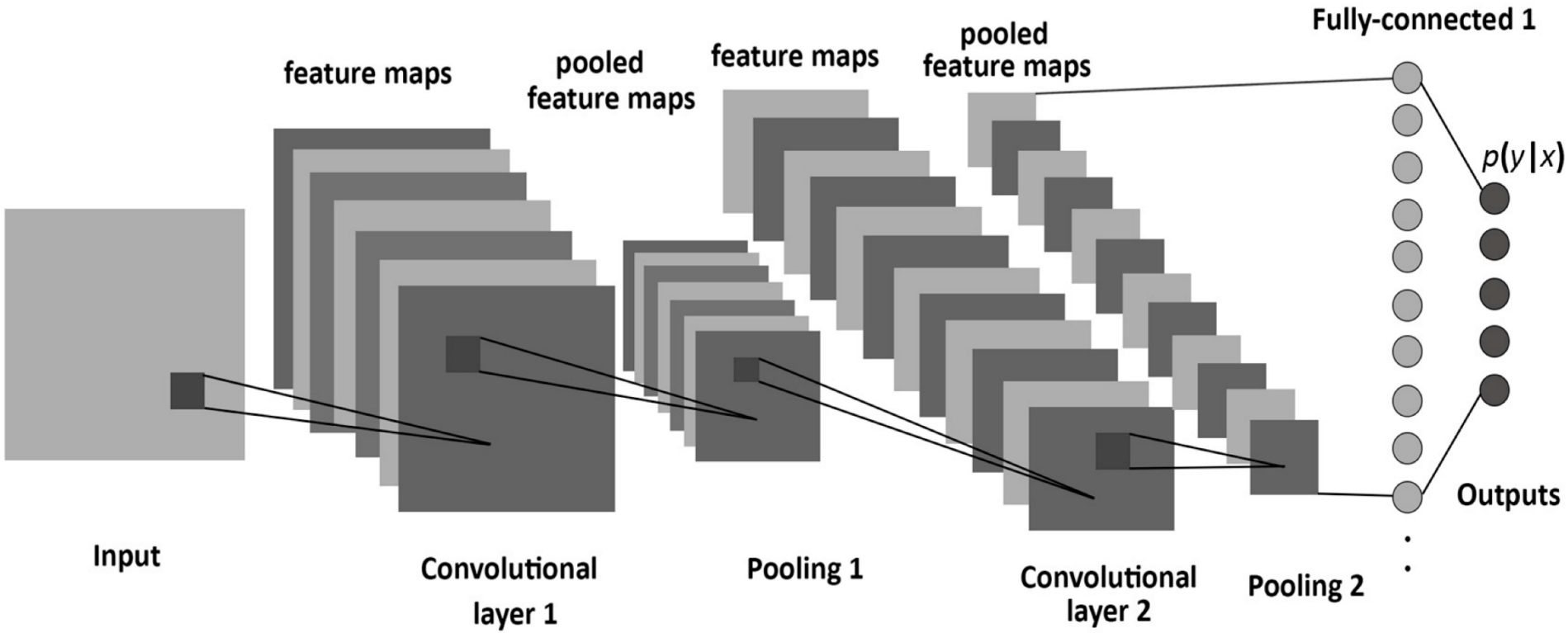
Framework Comparison

- More alike than different
  - All express deep models
  - Many have same computational basis: computation graph
  - All are open-source (contributions differ)
  - Most include scripting for hacking and prototyping

- No strict winners – experiment and choose the framework that best fits your work

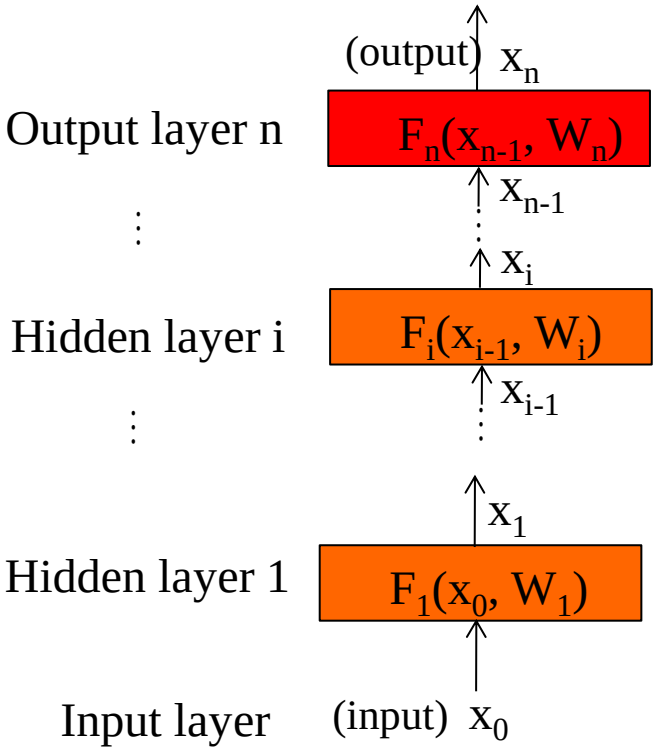
Computation Graph: Basis

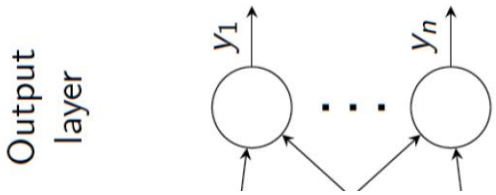
- Structural view of network: CNN

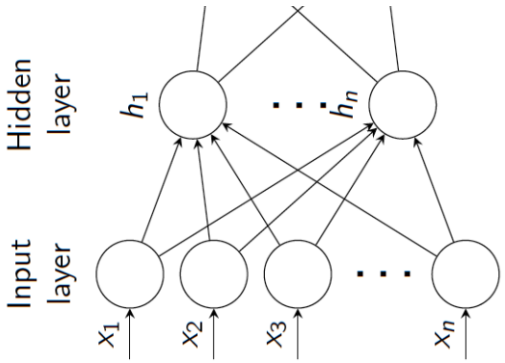


Computation Graph: Basis

- Decompose inference into elements
- Tools differ by granularity of elements

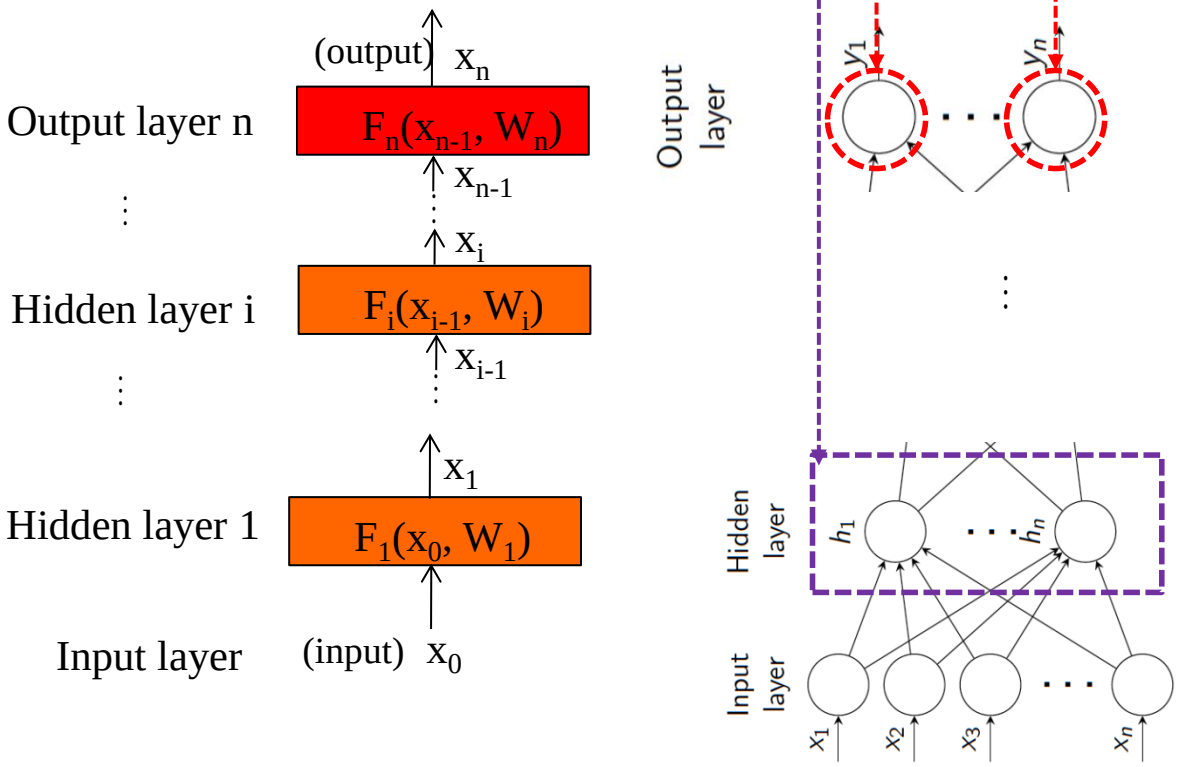






Examine Two Approaches

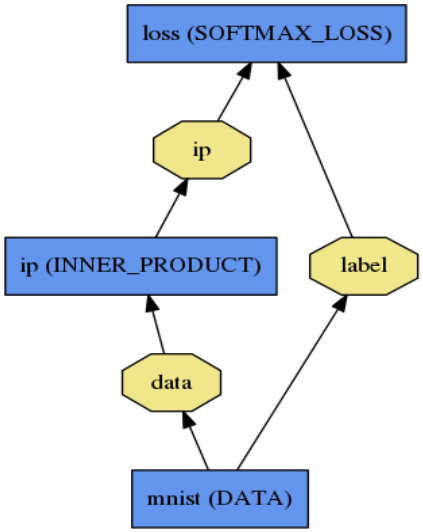
- TensorFlow
  - Fine granularity: computation graph
- Caffe2
  - Granularity: “level” of inference



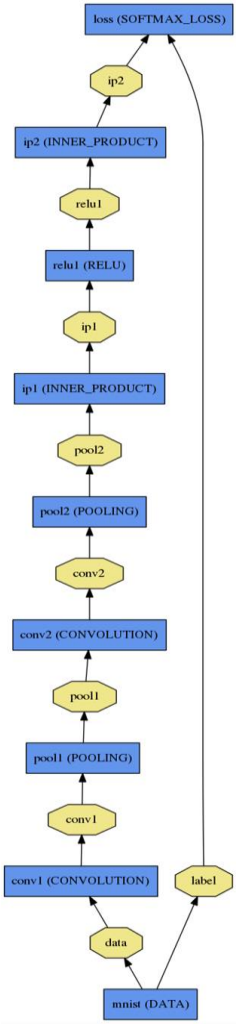
Network: Computation Graph

- A network is a set of layers connected as a DAG:

```
name: "dummy-net"
layers { name: "data" …}
layers { name: "conv" …}
layers { name: "pool" …}
    … more layers …
layers { name: "loss" …}
```



LogReg ↑

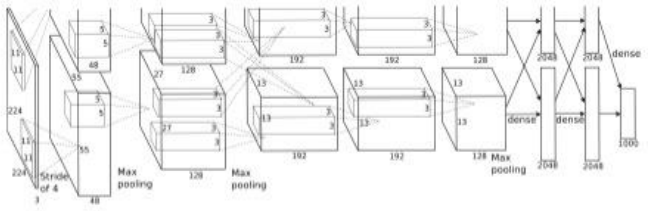


LeNet →

ImageNet, Krizhevsky 2012 →

Forward / Backward the essential Net computations

Forward:
inference $f_W(x)$



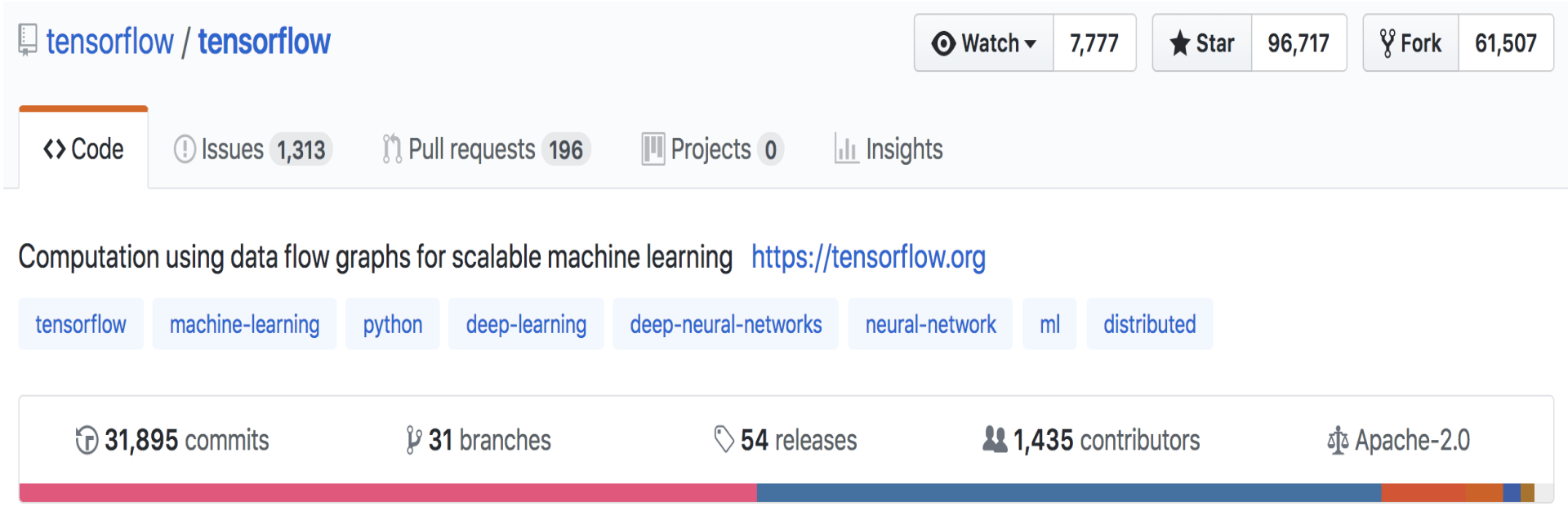
"espresso"
+ loss

$\nabla f_W(x)$ Backward:
learning

TensorFlow

- TensorFlow is an open source software library, originally developed by the Google Brain team, for machine learning in various kinds of tasks.
  - TensorFlow Homepage
  - TensorFlow Install
  - TensorFlow API (Version 1.10 for Python)
- TensorFlow is a low-level mathematics API, similar to Numpy. However, unlike Numpy, TensorFlow is built for deep learning.

What is TensorFlow?



- Open source library for numerical computation using **data flow graphs**
- Developed by Google Brain Team to conduct machine learning research
  - Based on DisBelief used internally at Google since 2011
- “TensorFlow is an interface for expressing machine learning algorithms, and an implementation for executing such algorithms”

What is TensorFlow

- **Key idea**: express a numeric computation as a **graph**
- Graph nodes are **operations** with any number of inputs and outputs
- Graph edges are **tensors** which flow between nodes

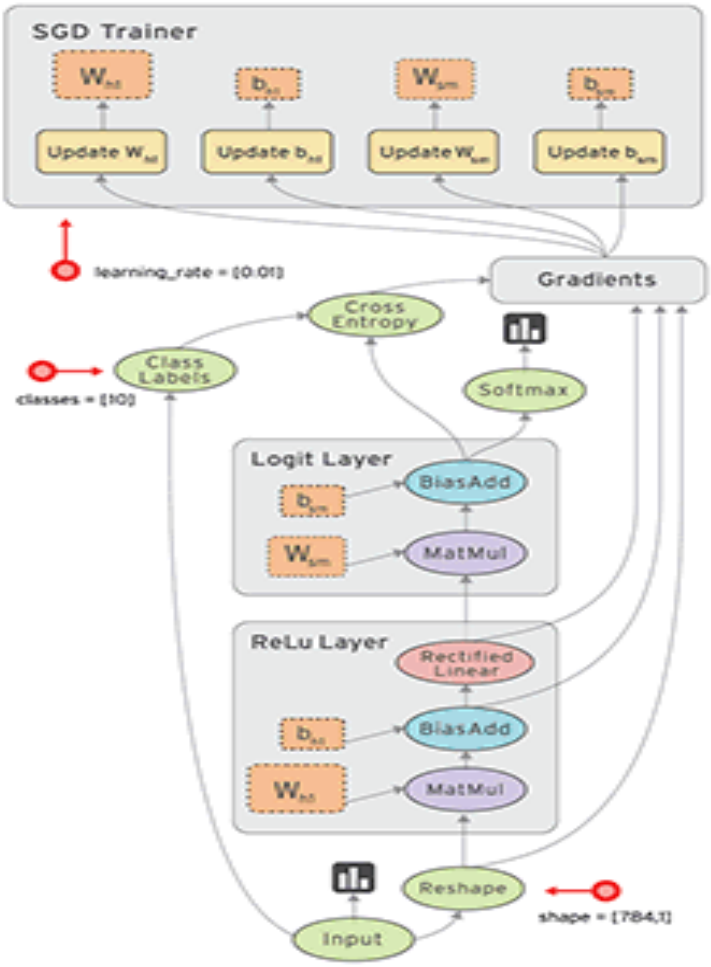
Design Principles

- Dataflow graphs of primitive operators
- Deferred execution (two phases)
  1. Define program i.e., symbolic dataflow graph w/ placeholders
  2. Executes optimized version of program on set of available devices
- Common abstraction for heterogeneous accelerators
  1. Issue a kernel for execution
  2. Allocate memory for inputs and outputs
  3. Transfer buffers to and from host memory

TensorFlow Entities

- Computation graph
- Nodes:
  - computation units
- Edges:
  - flow of Tensors (multidimensional arrays)

TensorBoard: Visualisation



Tensors: Multidimensional arrays

- Forward: data
- Backward: derivatives
  - Jacobian

TensorFlow

- Low-level computation library
  - Can use simple operators in order to implement an algorithm
  - Examples:
    - ‘add’ (element-wise addition of two matrices)
    - ‘matmul’ (matrix multiplication)
- Extensive suite of functions and classes
  - allow users to build models from scratch.

Low-Level Library

- Forward operations
  - Tensor operations
    - Matrix multiplication
    - Vector addition
- Backward operations
  - Derivative of any operator

Forward (Low-Level) Operation

- Each node defines an operation •
  - Perform operation on inputs: e.g., M • b
- Lazy programming
  - computations are scheduled only when necessary

Backward Operation: Automatic Differentiation

- TensorFlow calculates derivatives from the computation graph
  - chain rule
- Every node has attached gradient operation
  - calculates derivatives of input with respect to output
  - calculates gradients with respect to parameters during backpropagation

Jacobian Matrix

- A **non-square matrix** $n \times m$ in general
- Suppose you have a vector-valued function $f(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \end{bmatrix}$
- Let the **gradient operator** be the vector of (first-order) partial derivatives

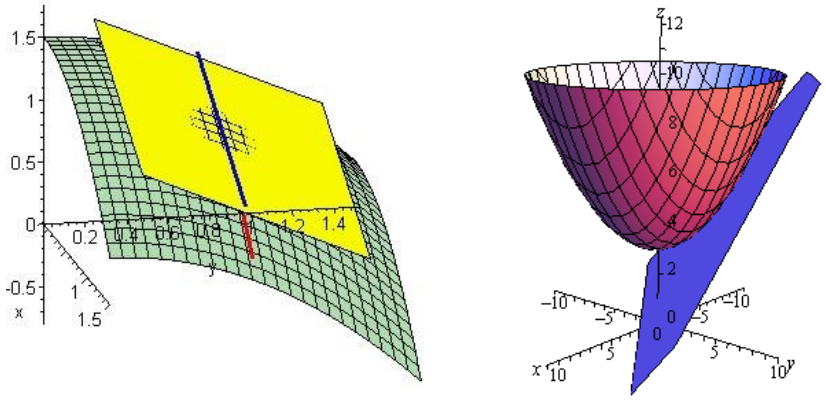
$$\nabla_{\mathbf{x}} = \begin{bmatrix} \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & \cdots & \frac{\partial}{\partial x_n} \end{bmatrix}^T$$

- Then, the **Jacobian matrix** is defined as

$$\mathbf{F}_{\mathbf{x}} = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x_1} & \cdots & \frac{\partial}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \\ \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_2}{\partial x_n} \end{bmatrix}$$

Visualisation of Jacobian

- It's the orientation of the **tangent plane** to the vector-valued function at a given point



- **Generalizes the gradient** of a scalar valued function
- Used for **backprop error propagation...**

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
    a. Makes it easy to extract the parallelism

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
    a. Makes it easy to extract the parallelism
2. Provides DFGs for primitive operators
    a. Softmax, convolution, MM, …
    b. Makes it easy to experiment with novel layers
    c. Automatic gradient calculation

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
    a. Makes it easy to extract the parallelism
2. Provides DFGs for primitive operators
    a. Softmax, convolution, MM, …
    b. Makes it easy to experiment with novel layers
    c. Automatic gradient calculation
3. Deferred execution
    a. Offload the larger chunks where possible...

TensorFlow: Solution Strategy

1. Execution Flexibility via DataFlow abstraction
   a. Makes it easy to extract the parallelism
2. Provides CGs for primitive operators
   a. Softmax, convolution, MM, …
   b. Makes it easy to experiment with novel layers
   c. Automatic gradient calculation
3. Deferred execution
   a. Offload the larger chunks where possible...
4. Common Abstraction for Accelerators
   a. Easy to integrate new accelerators into the fold
   b. The operators are specialized for different devices
5. Common data primitive : **Tensor**

Gradient computation: Backpropagation

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

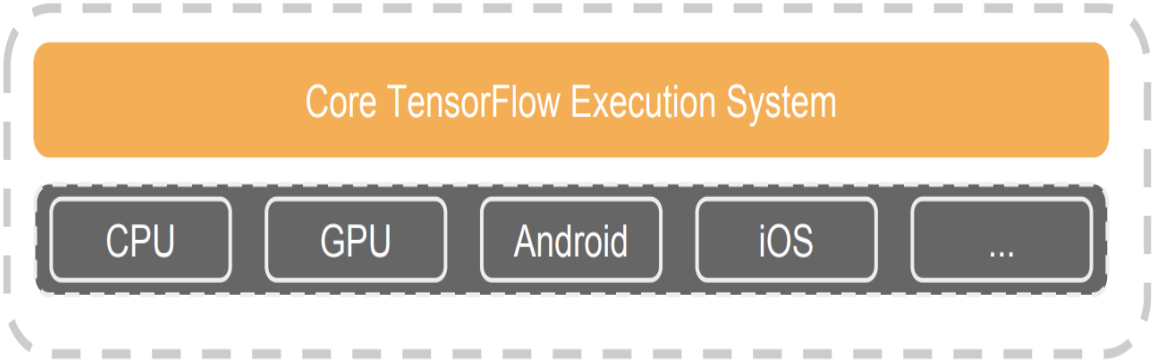
`tf.train.GradientDescentOptimizer` is an **Optimizer** object

`tf.train.GradientDescentOptimizer(lr).minimize(cross_entropy)`
adds optimization **operation** to computation graph

TensorFlow graph **nodes** have **attached gradient operations**
Gradient with respect to **parameters** computed with **backpropagation … automatically**

TensorFlow high-level architecture

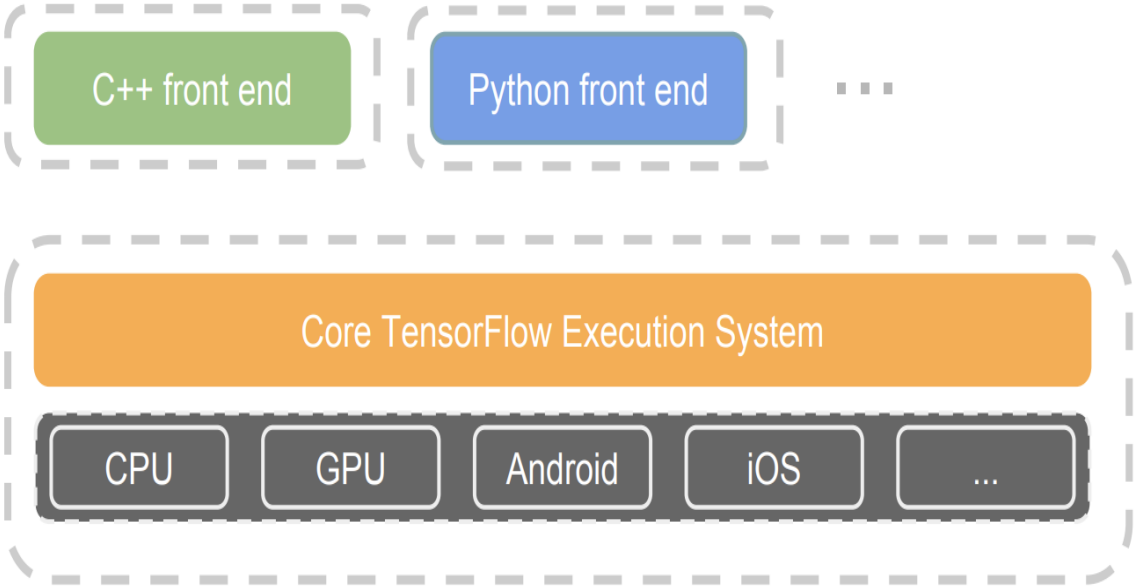
- Core in C++
  - Very low overhead
- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more



From: http://www.wsdm-conference.org/2016/slides/WSDM2016-Jeff-Dean.pdf

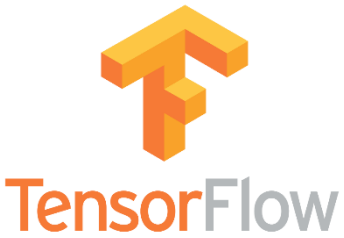
TensorFlow architecture

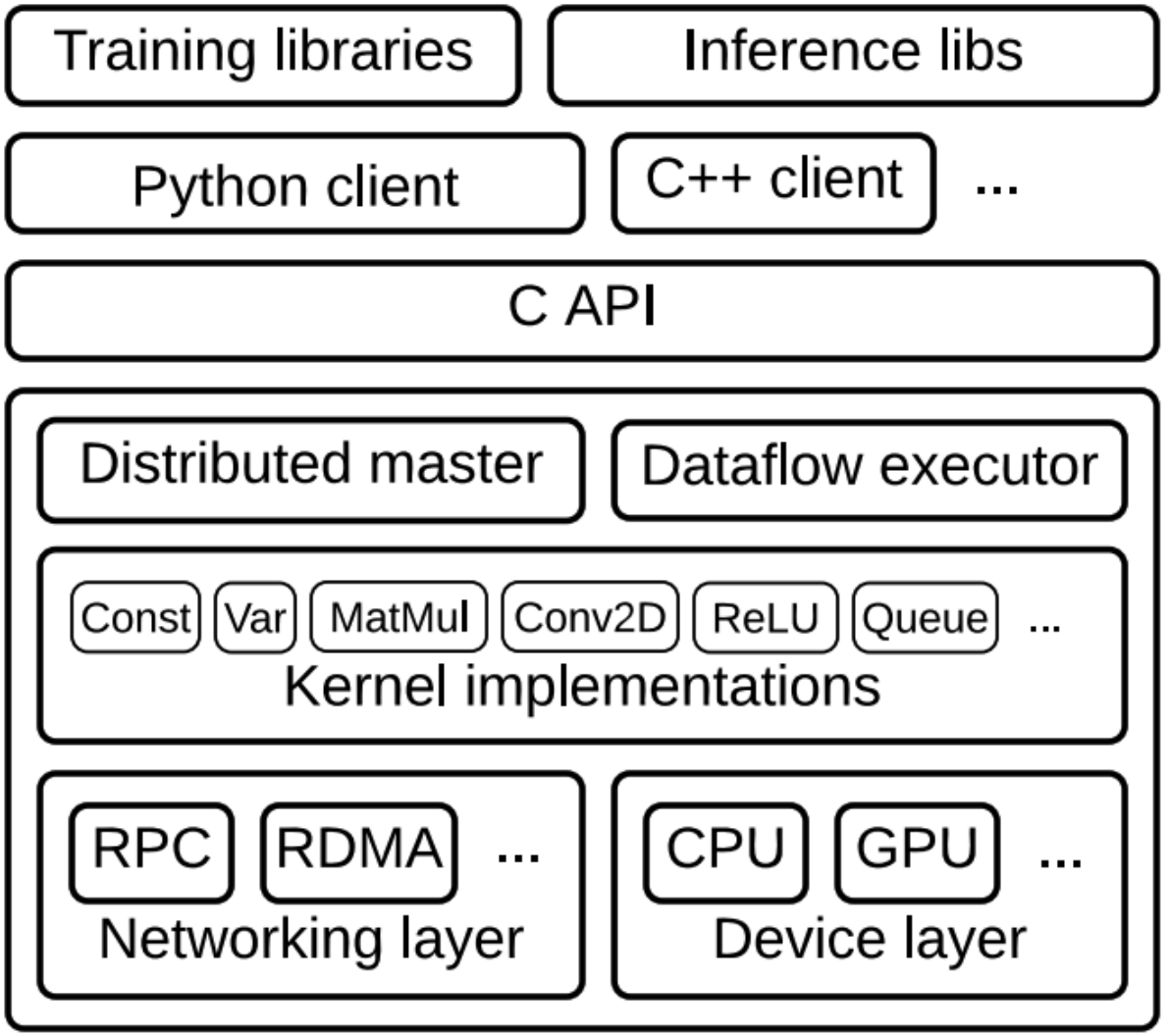
- Core in C++
  - Very low overhead
- Different front ends for specifying/driving the computation
  - Python and C++ today, easy to add more



From: http://www.wsdm-conference.org/2016/slides/WSDM2016-Jeff-Dean.pdf

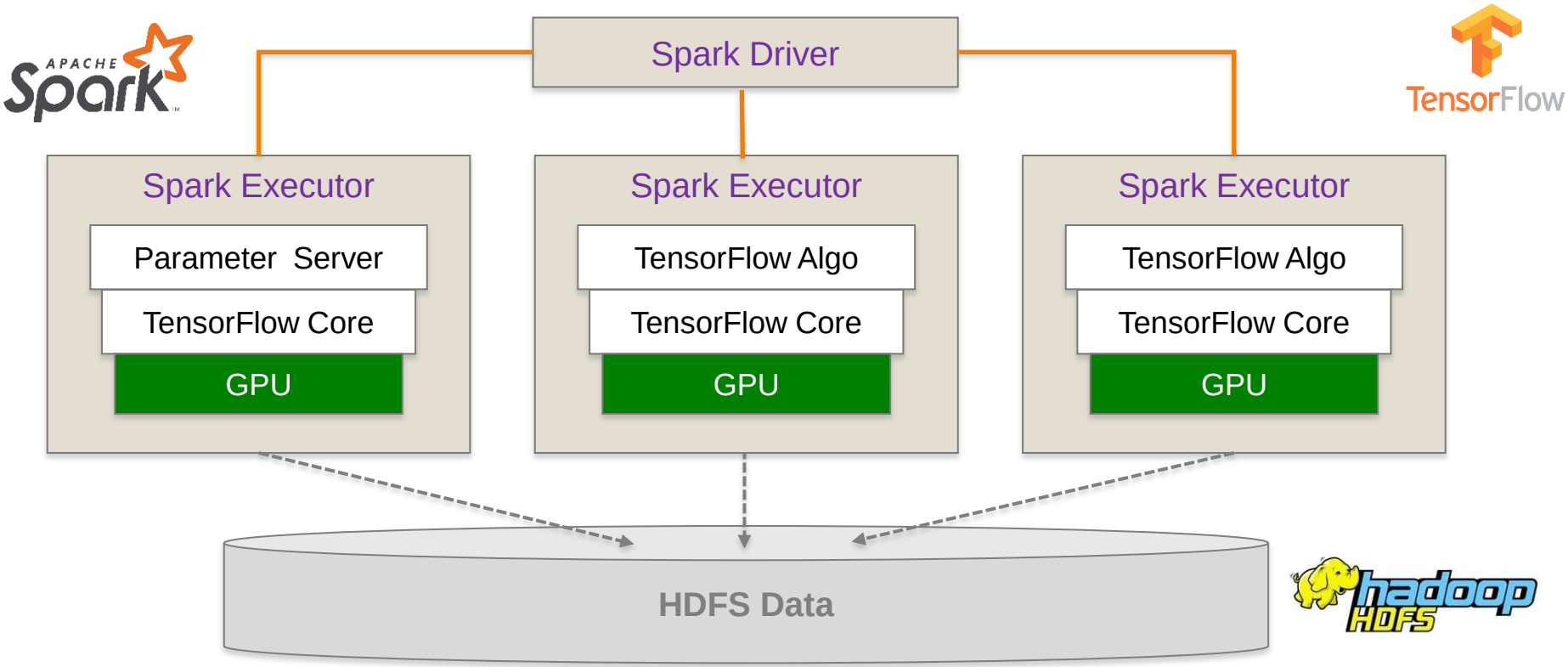
TensorFlow Architecture





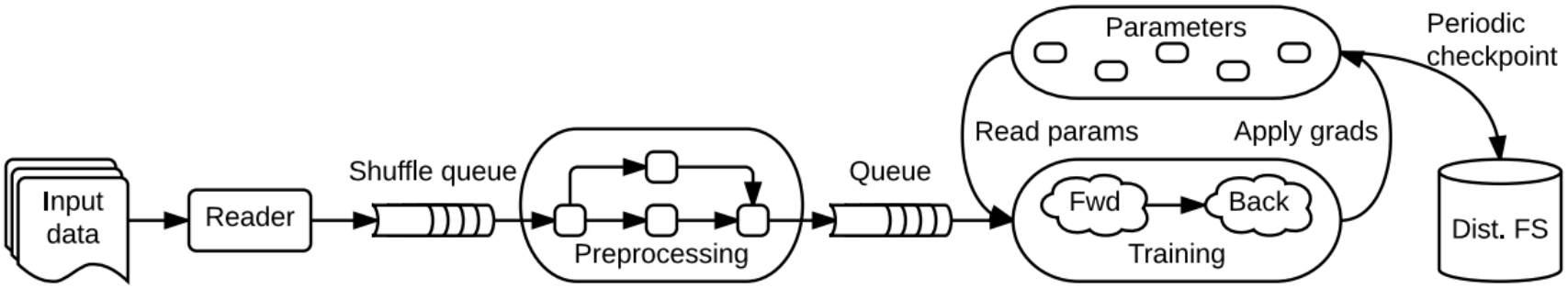
Source: www.tensorflow.org/extend/architecture

TensorFlow with Spark

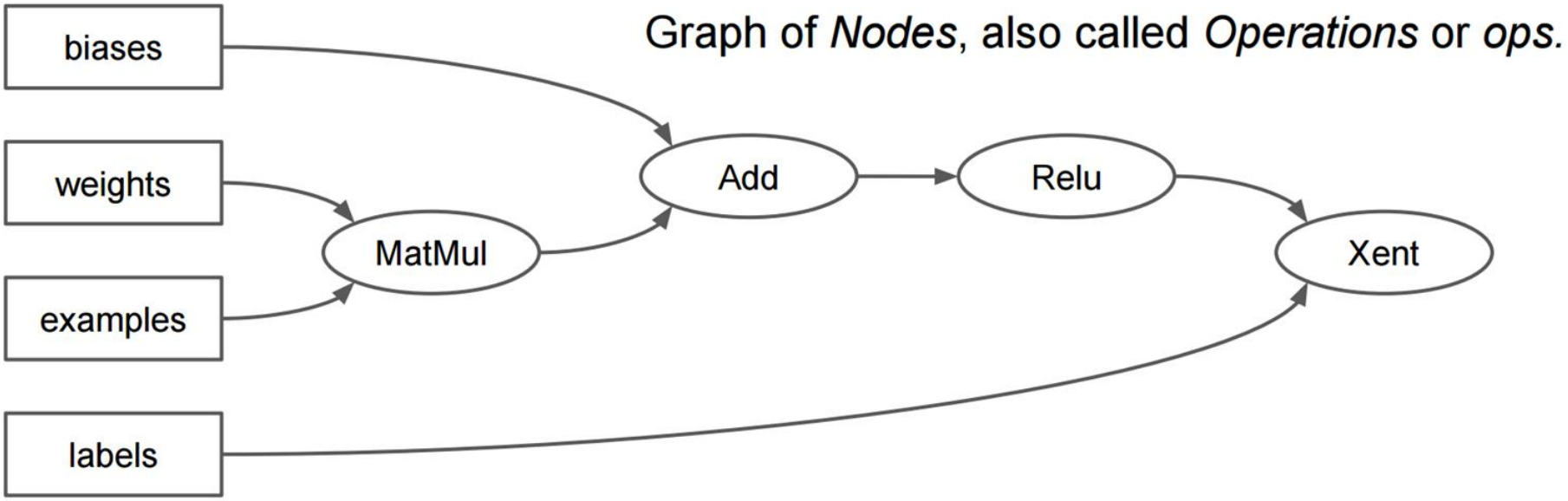


Execution Model

- Single DFG represents all computation and state for ML algorithm
  - Input preprocessing, mathematical operators, parameters, parameter update rules
  - Communication explicit, simplifying scheduling and partitioning



Computation Graph



Execution Model

- Single CG represents all computation and state for ML algorithm
  - Input preprocessing, mathematical operators, parameters, parameter update rules
  - Communication explicit, simplifying scheduling and partitioning
- Differences with existing systems:
  - Concurrent execution on overlapping subgraphs supported
  - Individual vertices contain sharable, mutable state