Big Data Analytics

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Computer Science
University of Hildesheim, Germany

C. Distributed Computing Environments / 3. Computational Graphs (TensorFlow)

Syllabus

Tue.	9.4.	(1)	0. Introduction
			A. Parallel Computing
Tue.	16.4.	(2)	A.1 Threads
Tue.	23.4.	(3)	A.2 Message Passing Interface (MPI)
Tue.	30.4.	(4)	A.3 Graphical Processing Units (GPUs)
			B. Distributed Storage
Tue.	7.5.	(5)	B.1 Distributed File Systems
Tue.	14.5.	(6)	B.2 Partioning of Relational Databases
Tue.	21.5.	(7)	B.3 NoSQL Databases
			C. Distributed Computing Environments
Tue.	28.5.	(8)	C.1 Map-Reduce
Tue.	4.6.	(9)	C.2 Resilient Distributed Datasets (Spark)
Tue.	11.6.	—	— *Pentecoste Break* —
Tue.	18.6.	(10)	C.3 Computational Graphs (TensorFlow)
			D. Distributed Machine Learning Algorithms
Tue.	25.6.	(11)	D.1 Distributed Stochastic Gradient Descent
Tue.	2.7.	(12)	D.2 Distributed Matrix Factorization
Tue.	9.7.	(13)	Questions and Answers

Outline

1. The Computational Graph
2. Variables
3. Example: Linear Regression
4. Automatic Gradients
5. Large Data I: Feeding
6. Large Data II: Reader Nodes
7. Debugging

Outline

1. The Computational Graph

2. Variables

3. Example: Linear Regression

4. Automatic Gradients

5. Large Data I: Feeding

6. Large Data II: Reader Nodes

7. Debugging

TensorFlow

- Computational framework
- multi-device, distributed
- Core in C/C++, standard interface in Python
  - several further language bindings, e.g., R, Java
- open source
  - developed by Google
  - initially released Nov. 2015
  - 2nd generation framework
    - 1st generation framework was called DistBelief
- alternative: pytorch
  - developed by facebook, since 2016, open source

Tensors

- tensor = multidimensional array
  - rank = number of dimensions
  - shape = vector of sizes,
    one size for each dimension.

rank	common name	shape
0	scalar	()
1	vector	(size)
2	matrix	(numrows, numcols)
≥ 3	tensor of higher order	$(\text{numdim}_1, \text{numdim}_2, \ldots, \text{numdim}_r)$

- examples:

$$A = \begin{pmatrix} 1.0 & -3.0 & 2.3 & 1.7 \\ 5.6 & 0.0 & -1.3 & 3.4 \\ -7.7 & -3.3 & -2.1 & 5.2 \end{pmatrix}, \quad \text{shape}(A) = (3,4), \quad \text{rank}(A) = 2$$

Computational Graphs

- TensorFlow organizes a computation as a directed graph.
- Nodes represent a tensor.
  - or a list of tensors.
- Tensors can be:
  - stored tensors
    - immutable, value provided at creation time: **tf.constant**
    - immutable, value provided when running the graph: **tf.placeholder**
    - mutable: **tf.Variable**
  - computed tensors (**operations**):
    - having one or more input tensors
    - having one or more output tensors
      – output index: **port**
- Edges represent dependencies.
  - Edge $x \to y$ if y is computed and x one of its inputs.

Sessions

- A session represents the state of an ongoing computation on a computational graph.
- create with default constructor **tf.Session**.
- compute the value of a tensor node with **run**.

Two Phases

1. Construct the Computational Graph
   - create tensor nodes
   - possibly referencing other tensor nodes as inputs
2. Compute values of a node of the Computation Graph (**running**)
   - usually specify target tensor(s)
   - computes all intermediate tensors required for this tensor
   - yield the value of the target tensor(s)

Hello TensorFlow: Add two Constants

```
import tensorflow as tf

a = tf.constant(3.0)
b = tf.constant(4.0)
x = tf.add(a, b)

print(a)
print(b)
print(x)

sess = tf.Session()
x_val = sess.run(x)
print(x_val)
```

Output:

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=float32)
Tensor("Add:0", shape=(), dtype=float32)
7
```

Hello TensorFlow: Add two Constants

```python
import tensorflow as tf

a = tf.constant([3.0, -2.7, 1.2])
b = tf.constant([4.0, 5.1, -1.7])
x = tf.add(a, b)

print(a)
print(b)
print(x)

sess = tf.Session()
x_val = sess.run(x)
print(x_val)
```

Output:

```
Tensor("Const_2:0", shape=(3,), dtype=float32)
Tensor("Const_3:0", shape=(3,), dtype=float32)
Tensor("Add_1:0", shape=(3,), dtype=float32)
[ 7.0   2.4   -0.5 ]
```

Tensor Types

- Different element types are represented by tf.DType:

dtype	description
tf.float16	16-bit half-precision floating-point
tf.float32	32-bit single-precision floating-point
tf.float64	64-bit double-precision floating-point
tf.bfloat16	16-bit truncated floating-point
tf.complex64	64-bit single-precision complex
tf.complex128	128-bit double-precision complex
tf.int8	8-bit signed integer
tf.uint8	8-bit unsigned integer
tf.uint16	16-bit unsigned integer
tf.int16	16-bit signed integer
tf.int32	32-bit signed integer
tf.int64	64-bit signed integer
tf.bool	Boolean
tf.string	String
tf.qint8	Quantized 8-bit signed integer
tf.quint8	Quantized 8-bit unsigned integer
tf.qint16	Quantized 16-bit signed integer
tf.quint16	Quantized 16-bit unsigned integer
tf.qint32	Quantized 32-bit signed integer
tf.resource	Handle to a mutable resource

- if omitted, inferred from values:

```
a = tf.constant(4.0)
b = tf.constant(4)
print(a)
print(b)
```

Output:

```
Tensor("Const:0", shape=(), dtype=float32)
Tensor("Const_1:0", shape=(), dtype=int32)
```

Operations: Overloaded Operators

```
import tensorflow as tf

a = tf.constant(3.0)
b = tf.constant(4.0)
x = a + b

print(x)

sess = tf.Session()
x_val = sess.run(x)
print(x_val)
```

Output:

```
Tensor("add_1:0", shape=(), dtype=float32)
7
```

operator	operation node
+	tf.add
-	tf.subtract
*	tf.multiply
/	tf.divide

Outline

1. The Computational Graph

2. Variables

3. Example: Linear Regression

4. Automatic Gradients

5. Large Data I: Feeding

6. Large Data II: Reader Nodes

7. Debugging

$$v = \text{tf.Variable}(\text{initial\_value} = \text{None}, \ldots, \text{name} = \text{None}, \ldots, \text{dtype} = \text{None})$$

- has immutable element type
- has mutable shape (**set_shape**).
- has mutable element values.
  - set by **tf.assign**, **tf.assign_add** (operations)
  - separate values in each session
- has to be initialized before first use:
  - run **v.initializer** operation or
  - run initializers of all variables:

```
init = tf.global_variables_initializer()
sess.run(init)
```

Variables

```python
import tensorflow as tf

x = tf.Variable(3.0)
sess = tf.Session()
sess.run( x.initializer )
x_val = sess.run(x)
print(x_val)

x_plus_one = tf.assign_add(x, 1.0)

for t in range(5):
    x_val = sess.run(x_plus_one)
    print(t, x_val)
```

Output:

```
3.0

0 4.0
1 5.0
2 6.0
3 7.0
4 8.0
```

Initializing from Other Variables

```
import tensorflow as tf

x = tf.Variable(3.0)
y = tf.Variable( tf.multiply(tf.constant(2.0),
                             x.initialized_value()) )
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
y_val = sess.run(y)
print(y_val)
```

Output:

```
6.0
```

- v.initialized_value assures that a variable has been initialized before
  - do not use

    ```
    y = tf.Variable( tf.multiply(tf.constant(2.0), x) )
    ```

    as x may be selected to be initialized after y.

Outline

1. The Computational Graph

2. Variables

3. Example: Linear Regression

4. Automatic Gradients

5. Large Data I: Feeding

6. Large Data II: Reader Nodes

7. Debugging

Example: Linear Regression

$$\hat{y} := \beta_0 + X\beta \qquad \text{prediction}$$

$$r := y - \hat{y} \qquad \text{residuum}$$

$$\ell := \frac{1}{2}\sum_{n=1}^{N} r_n^2 \qquad \text{loss/error}$$

$$-\frac{\partial \ell}{\partial \beta} := X^T r \qquad \text{negative gradient w.r.t. } \beta$$

$$-\frac{\partial \ell}{\partial \beta_0} := \sum_{n=1}^{N} r_n \qquad \text{negative gradient w.r.t. } \beta_0$$

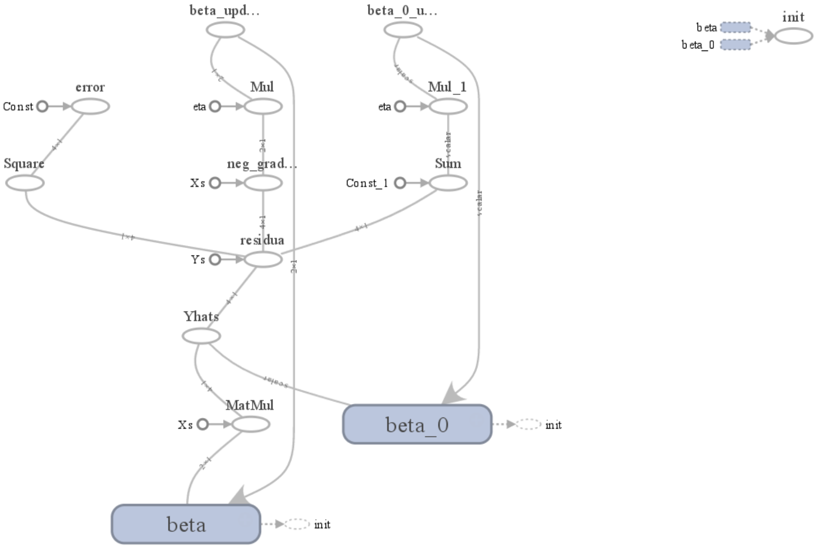
$$\beta^{\text{next}} := \beta - \eta \frac{\partial \ell}{\partial \beta} \qquad \text{update of } \beta$$

$$\beta_0^{\text{next}} := \beta_0 - \eta \frac{\partial \ell}{\partial \beta_0} \qquad \text{update of } \beta_0$$

Example: Linear Regression

```
import tensorflow as tf

Xs_data = [[2,1], [1,2], [4,3], [3,4]]
Ys_data = [[+1], [+1], [-1], [-1]]
eta_data = 0.01

Xs = tf.constant(Xs_data, dtype=tf.float32)
Ys = tf.constant(Ys_data, dtype=tf.float32)
eta = tf.constant(eta_data)

beta = tf.Variable([[0], [0]], dtype=tf.float32)
beta_0 = tf.Variable(0, dtype=tf.float32)

Yhats = tf.add(beta_0, tf.matmul(Xs, beta))
residua = tf.subtract(Ys, Yhats)
error = tf.reduce_sum(tf.square(residua))

neg_grad_beta = tf.matmul(Xs, residua, adjoint_a=True)
beta_update = tf.assign_add(beta, tf.multiply(eta, neg_grad_beta))
beta_0_update = tf.assign_add(beta_0, tf.multiply(eta, tf.reduce_sum(residua)))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run( init )

for t in range(100):
    error_val, beta_val, beta_0_val = sess.run([error,beta_update,beta_0_update])
    print(t, error_val, beta_0_val, beta_val[0,0], beta_val[1,0])
```

Example: Linear Regression / Computational Graph



Outline

1. The Computational Graph

2. Variables

3. Example: Linear Regression

4. Automatic Gradients

5. Large Data I: Feeding

6. Large Data II: Reader Nodes

7. Debugging

$$\text{tf.gradients}(\text{ys}, \text{xs}, \ldots)$$

- create operations whose final node computes all the gradients

$$\left(\frac{\partial y_n}{\partial x_m}\right)_{n=1,\ldots,N, m=1,\ldots,M} \qquad \text{ys} = (y_1, \ldots, y_N), \text{xs} = (x_1, \ldots, x_M)$$

Example: Linear Regression w. Automatic Gradients

```
import tensorflow as tf

Xs_data = [[2,1], [1,2], [4,3], [3,4]]
Ys_data = [[+1], [+1], [-1], [-1]]
eta_data = 0.01

Xs = tf.constant(Xs_data, dtype=tf.float32)
Ys = tf.constant(Ys_data, dtype=tf.float32)
eta = tf.constant(eta_data)

beta = tf.Variable([[0], [0]], dtype=tf.float32)
beta_0 = tf.Variable(0, dtype=tf.float32)

Yhats = tf.add(beta_0, tf.matmul(Xs, beta))
error = tf.reduce_sum(tf.square(tf.subtract(Ys, Yhats)))

grads = tf.gradients(error, [beta, beta_0])
beta_update = tf.assign_sub(beta, tf.multiply(eta, grads[0]))
beta_0_update = tf.assign_sub(beta_0, tf.multiply(eta, grads[1]))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run( init )

for t in range(100):
    error_val, beta_val, beta_0_val = sess.run([error,beta_update,beta_0_update])
    print(t, error_val, beta_0_val, beta_val[0,0], beta_val[1,0])
```

How do Automatic Gradients Work?

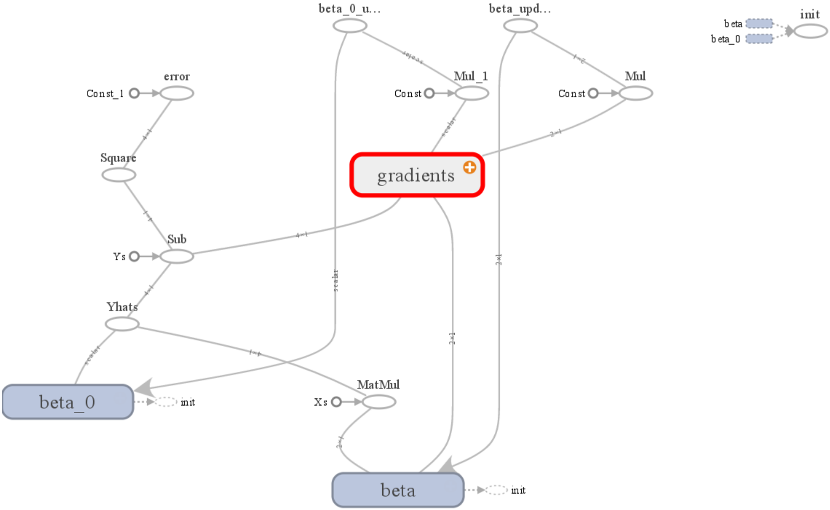
to compute $\frac{\partial y}{\partial x}$:

- find all paths $p^1, \ldots, p^K \in G^*$ in the graph G from x to y
- use chain rule:

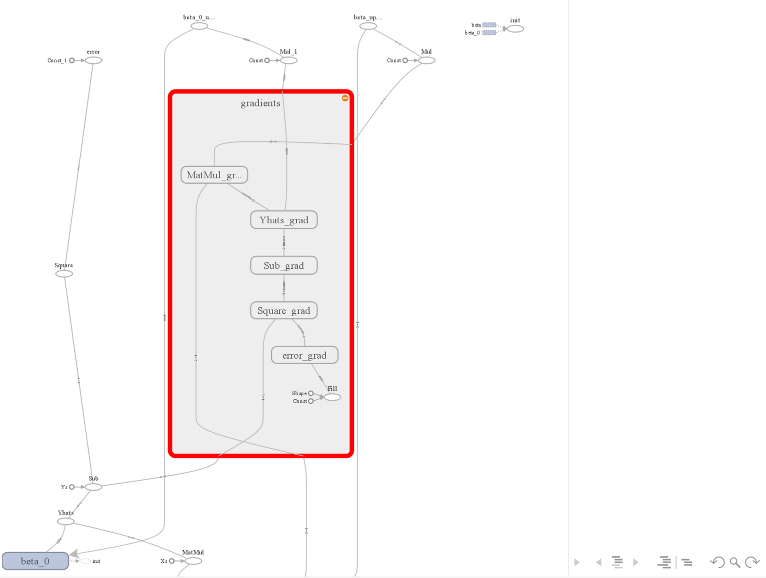
$$\frac{\partial y}{\partial x} = \sum_{k=1}^{K} \prod_{l=|p^k|}^{2} \frac{\partial p_l^k}{\partial p_{l-1}^k}$$

- each operation $p_l^k =: o$ has to provide its gradient $\frac{\partial o}{\partial i}$ for each of its inputs i.
  - then $\frac{\partial \boldsymbol{p_l^k}}{\partial \boldsymbol{p_{l-1}^k}} = \frac{\partial o}{\partial i}$ for $i = p_{l-1}^k$.

Example: LinReg w. Auto Grads / Computational Graph



Example: LinReg w. Auto Grads / Computational Graph



Outline

1. The Computational Graph

2. Variables

3. Example: Linear Regression

4. Automatic Gradients

5. Large Data I: Feeding

6. Large Data II: Reader Nodes

7. Debugging

- representing large data as a whole as a constant is not so useful
  - e.g., if its size exceeds GPU memory, it cannot be deployed to GPU at all.
- better break data into smaller pieces
  - e.g., single instances or minibatches
  - batch GD $\rightarrow$ SGD
- build a graph for a single instance / minibatch
- create placeholder nodes for the instance / minibatch
- placeholders are filled with the **feed_dict** parameter of **run**.

Placeholder Nodes and Feeding

```
import tensorflow as tf

a = tf.placeholder(shape=(), dtype=tf.float32)
b = tf.placeholder(shape=(), dtype=tf.float32)
x = a + b

sess = tf.Session()
print( sess.run(x, {a: 3, b: 7}) )
print( sess.run(x, {a: -2, b: 4}) )
```

Output:

```
10.0
2.0
```

Example: Feeding SGD

```
import tensorflow as tf

Xs_data = [[2,1], [1,2], [4,3], [3,4]]
Ys_data = [+1, +1, -1, -1]
eta_data = 0.01

X = tf.placeholder(shape=(2), dtype=tf.float32)
Y = tf.placeholder(shape=(), dtype=tf.float32)
eta = tf.constant(eta_data)

beta = tf.Variable([0, 0], dtype=tf.float32)
beta_0 = tf.Variable(0, dtype=tf.float32)

Yhat = tf.add(beta_0, tf.reduce_sum(tf.multiply(X, beta)))
error = tf.reduce_sum(tf.square(tf.subtract(Y, Yhat)))

grads = tf.gradients(error, [beta, beta_0])
beta_update = tf.assign_sub(beta, tf.multiply(eta, grads[0]))
beta_0_update = tf.assign_sub(beta_0, tf.multiply(eta, grads[1]))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run( init )

for t in range(100):
    error_epoch = 0
    for X_data, Y_data in zip(Xs_data, Ys_data):
        error_val, beta_val, beta_0_val = sess.run([error,beta_update,beta_0_update],
                                          { X: X_data, Y: Y_data })
        error_epoch += error_val
    print(t, error_epoch, beta_0_val, beta_val[0], beta_val[1])
```

Outline

1. The Computational Graph
2. Variables
3. Example: Linear Regression
4. Automatic Gradients
5. Large Data I: Feeding
6. Large Data II: Reader Nodes
7. Debugging

Reader Node

```
import tensorflow as tf

data_files = ['lr-data.csv']; eta_data = 0.01

filename_queue = tf.train.string_input_producer(data_files)
reader = tf.TextLineReader(skip_header_lines=1)
_, line = reader.read(filename_queue)

sess = tf.Session()
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(coord=coord, sess=sess)

for t in range(6):
    line_val = sess.run(line)
    print(t, line_val)

coord.request_stop()
coord.join(threads)
```

file lr-data.csv:

```
X1, X2, Y
2, 1, +1
1, 2, +1
4, 3, -1
3, 4, -1
```

Output:

```
0 b'2,␣1,␣+1'
1 b'1,␣2,␣+1'
2 b'4,␣3,␣-1'
3 b'3,␣4,␣-1'
4 b'2,␣1,␣+1'
5 b'1,␣2,␣+1'
```

Example: SGD Reading On The Fly

```
import tensorflow as tf
data_files = ['lr-data.csv']; eta_data = 0.01

filename_queue = tf.train.string_input_producer(data_files)
reader = tf.TextLineReader(skip_header_lines=1)
key, value = reader.read(filename_queue)
X1, X2, Y = tf.decode_csv(value, record_defaults=[[0.0],[0.0],[0.0]])
X = tf.stack([X1, X2])
eta = tf.constant(eta_data)

beta = tf.Variable([0, 0], dtype=tf.float32)
beta_0 = tf.Variable(0, dtype=tf.float32)
Yhat = tf.add(beta_0, tf.reduce_sum(tf.multiply(X, beta)))
error = tf.reduce_sum(tf.square(tf.subtract(Y, Yhat)))
grads = tf.gradients(error, [beta, beta_0])
beta_update = tf.assign_sub(beta, tf.multiply(eta, grads[0]))
beta_0_update = tf.assign_sub(beta_0, tf.multiply(eta, grads[1]))
init = tf.global_variables_initializer()
sess = tf.Session() ; sess.run( init )
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(coord=coord, sess=sess)

error_epoch = 0
for t in range(400):
    error_val, beta_val, beta_0_val = sess.run([error,beta_update,beta_0_update])
    error_epoch += error_val
    if t % 10 == 0:
        print(t, error_epoch, beta_0_val, beta_val[0], beta_val[1])
        error_epoch = 0
coord.request_stop()
coord.join(threads)
```

Outline

1. The Computational Graph

2. Variables

3. Example: Linear Regression

4. Automatic Gradients

5. Large Data I: Feeding

6. Large Data II: Reader Nodes

7. Debugging

Debugging: Visualize Computational Graph

1. Create a summary.FileWriter for the session and graph before running the session:

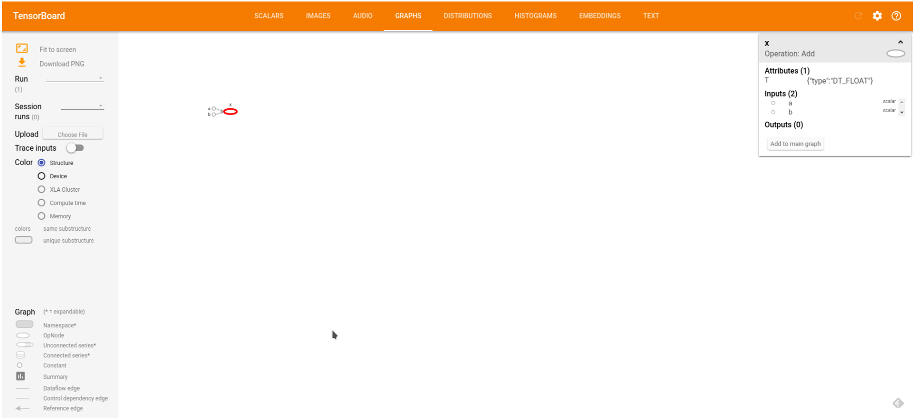
```
import tensorflow as tf

a = tf.constant(3.0, name='a')
b = tf.constant(4.0, name='b')
x = tf.add(a, b, name='x')

print(a)

sess = tf.Session()
log = tf.summary.FileWriter('logs/add-two-constants.log', sess.graph)
x_val = sess.run(x)
log.close()
print(x_val)
```

2. run tensorboard on the logdir:

```
> tensorboard --logdir logs/add-two-constants.log
```

3. open localhost:6006 in your browser

Debugging: Visualize Computational Graph



Summary (1/3)

- TensorFlow represents computations as **graphs**.
  - nodes representing (a list of) **tensors**.
    - stored:
      — immutable: **constant**, **placeholder**
      — mutable: **variable**
    - computed: **operation**
  - edges representing dependencies
    - $x \rightarrow y$: y is computed and x is one of its inputs
- Two phases:
  - graph construction
  - executing (parts of) the graph (**running**)

Summary (2/3)

- Nodes can be **distributed over different devices**.
  - cores of a CPU, GPUs, different compute nodes
  - **automatic placement** based on cost heuristics
    - eligible: sufficient memory available
    - expected runtime
      — based on cost heuristics
      — possibly also based on past runs
    - expected time for data movement between devices
- Operations can be assembled from dozens of **elementary operations**.
  - elementary math: add, subtract, multiply, divide
  - elementwise functions: log, exp, etc.
  - matrix operations: matrix product, inversion, etc.
  - structual tensor operations: slicing, stacking etc.

Summary (3/3)

- **Gradients** can be computed automatically.
  - simply using the chain rule
  - and explicit gradients for all elementary operations.
  - gradients add nodes to the graph.
- Medium-sized data should be broken into parts and **fed into a placeholder** for parts
  - e.g., SGD: single instances or minibatches
  - medium-sized data:
    - too large for the GPU
    - still can be read on a single data node
- Large data must be read by **reader nodes** as part of the graph execution.
  - large data: must be read on different data nodes in a distributed fashion

Further Readings

- TensorFlow white paper:
  - Abadi et al. [2016]
  - not yet fully complete: evaluation section is missing

References I

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.