Deep Learning for Poets (Part I)

Amir H. Payberah
payberah@kth.se
19/12/2018

TensorFlow

Linear and Logistic regression

Deep Feedforward Networks

CNN, RNN, Autoencoders

TensorFlow

Linear and Logistic regression

Deep Feedforward Networks

CNN, RNN, Autoencoders

Sheepdog or Mop

Chihuahua or Muffin

Barn Owl or Apple

Raw Chicken or Donald Trump

Artificial Intelligence Challenge

- Artificial intelligence (AI) can solve problems that can be described by a list of formal mathematical rules.

Artificial Intelligence Challenge

- Artificial intelligence (AI) can solve problems that can be described by a list of formal mathematical rules.
- The challenge is to solve the tasks that are hard for people to describe formally.

Artificial Intelligence Challenge

- Artificial intelligence (AI) can solve problems that can be described by a list of formal mathematical rules.

- The challenge is to solve the tasks that are hard for people to describe formally.

- Let computers to learn from experience.

History of AI

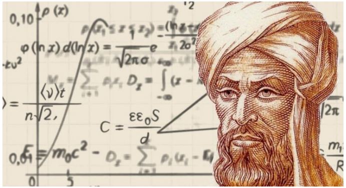
Greek Myths

- Hephaestus, the god of blacksmith, created a metal automaton, called Talos.

[the left figure: http://mythologian.net/hephaestus-the-blacksmith-of-gods]
[the right figure: http://elderscrolls.wikia.com/wiki/Talos]

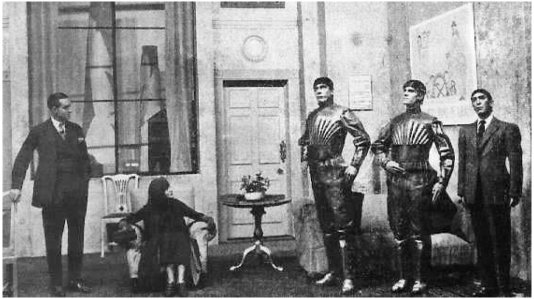
Formal Reasoning

- Mechanizing the process of human thought.



1920: Rossum's Universal Robots (R.U.R.)

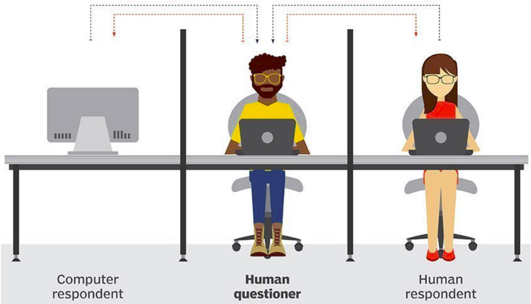
- A science fiction play by Karel Čapek, in 1920.
- A factory that creates artificial people named robots.



[https://dev.to/lschultebraucks/a-short-history-of-artificial-intelligence-7hm]

1950: Turing Test

- In 1950, Turing introduced the Turing test.
- An attempt to define machine intelligence.



[https://searchenterpriseai.techtarget.com/definition/Turing-test]

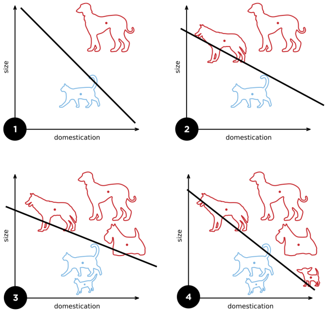
1956: The Dartmouth Workshop

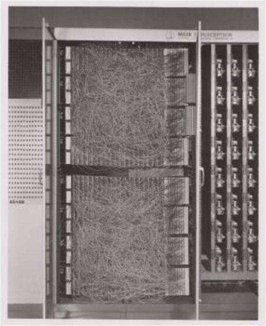
- Probably the first workshop of AI.
- Researchers from CMU, MIT, IBM met together and founded the AI research.

[https://twitter.com/lordsaicom/status/898139880441696257]

1958: Perceptron

- A supervised learning algorithm for binary classifiers.
- Implemented in custom-built hardware as the Mark 1 perceptron.

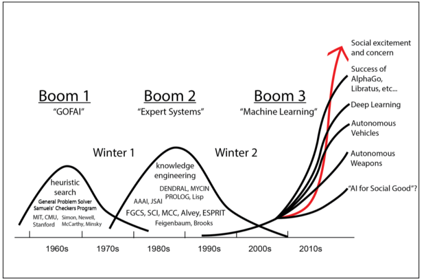




[https://en.wikipedia.org/wiki/Perceptron]

1974–1980: The First AI Winter

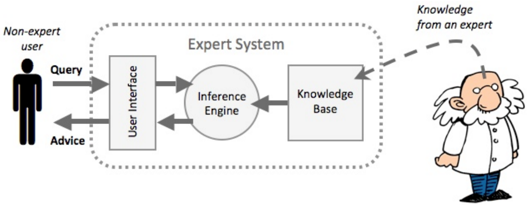
- The over optimistic settings, which were not occurred
- The problems:
  - Limited computer power
  - Lack of data
  - Intractability and the combinatorial explosion



[http://www.technologystories.org/ai-evolution]

1980's: Expert systems

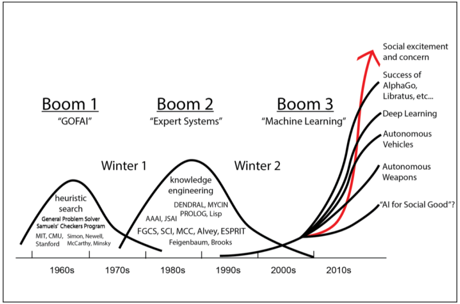
- The programs that solve problems in a specific domain.
- Two engines:
  - Knowledge engine: represents the facts and rules about a specific topic.
  - Inference engine: applies the facts and rules from the knowledge engine to new facts.



[https://www.igcseict.info/theory/7_2/expert]

1987–1993: The Second AI Winter

- After a series of financial setbacks.
- The fall of expert systems and hardware companies.



[http://www.technologystories.org/ai-evolution]

1997: IBM Deep Blue

- The first chess computer to beat a world chess champion Garry Kasparov.

[http://marksist.org/icerik/Tarihte-Bugun/1757/11-Mayis-1997-Deep-Blue-adli-bilgisayar]

2012: AlexNet - Image Recognition

- The ImageNet competition in image classification.
- The AlexNet Convolutional Neural Network (CNN) won the challenge by a large margin.

2016: DeepMind AlphaGo

- DeepMind AlphaGo won Lee Sedol, one of the best players at Go.
- In 2017, AlphaGo Zero that learned Go by playing against itself.

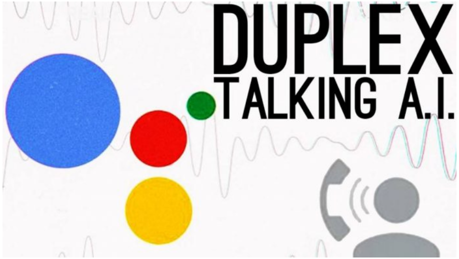
[https://www.zdnet.com/article/google-alphago-caps-victory-by-winning-final-historic-go-match]

2017: DeepStack

- A game of imperfect information.

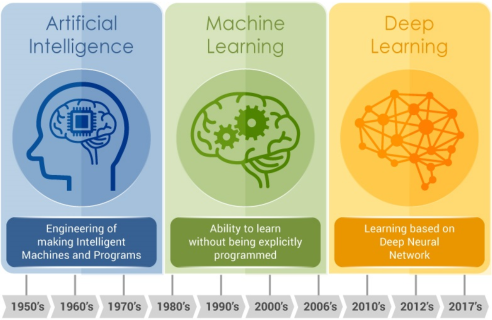
2018: Google Duplex

- An AI system for accomplishing real-world tasks over the phone.
- A Recurrent Neural Network (RNN) built using TensorFlow.



AI Generations

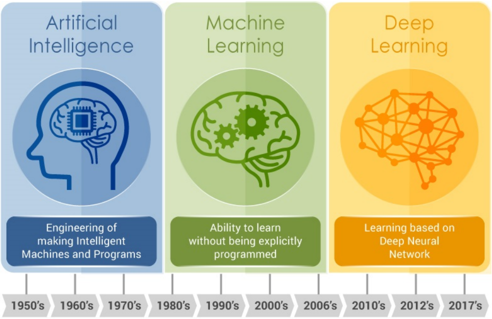
- Rule-based AI
- Machine learning
- Deep learning



[https://bit.ly/2woLEzs]

AI Generations - Rule-based AI

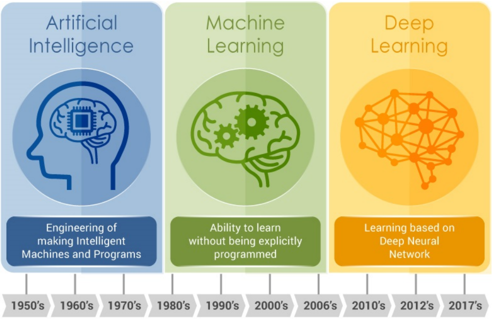
- Hard-code knowledge
- Computers reason using logical inference rules



[https://bit.ly/2woLEzs]

AI Generations - Machine Learning

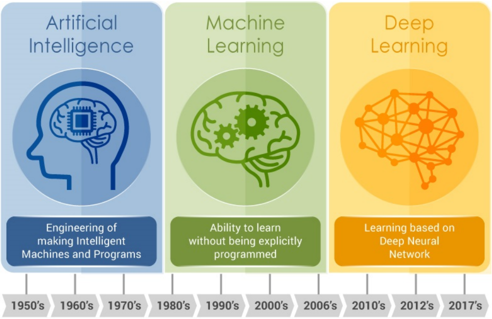
- If AI systems acquire their own knowledge
- Learn from data without being explicitly programmed



[https://bit.ly/2woLEzs]

AI Generations - Deep Learning

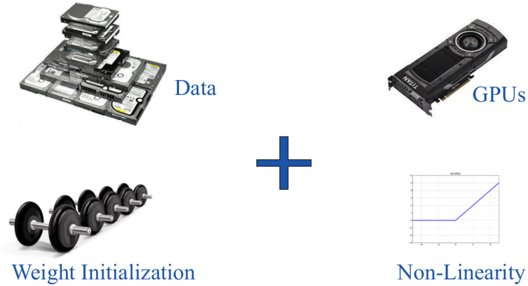
- For many tasks, it is difficult to know what features should be extracted
- Use machine learning to discover the mapping from representation to output



[https://bit.ly/2woLEzs]

Why Does Deep Learning Work Now?

- Huge quantity of data
- Tremendous increase in computing power
- Better training algorithms



Machine Learning and Deep Learning

Learning Algorithms

- A ML algorithm is an algorithm that is able to learn from data.
- What is learning?

Learning Algorithms

- A ML algorithm is an algorithm that is able to learn from data.
- What is learning?
- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. (Tom M. Mitchell)

Learning Algorithms - Example 1

- A spam filter that can learn to flag spam given examples of spam emails and examples of regular emails.

[https://bit.ly/2oiplYM]

Learning Algorithms - Example 1

- A spam filter that can learn to flag spam given examples of spam emails and examples of regular emails.
- Task T: flag spam for new emails
- Experience E: the training data
- Performance measure P: the ratio of correctly classified emails

[https://bit.ly/2oiplYM]

Learning Algorithms - Example 2

- Given dataset of prices of 500 houses, how can we learn to predict the prices of other houses, as a function of the size of their living areas?

[https://bit.ly/2MyiJUy]

Learning Algorithms - Example 2

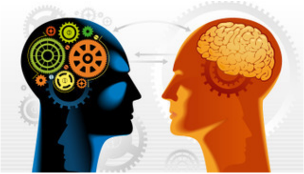
- Given dataset of prices of 500 houses, how can we learn to predict the prices of other houses, as a function of the size of their living areas?
- Task T: predict the price
- Experience E: the dataset of living areas and prices
- Performance measure P: the difference between the predicted price and the real price

[https://bit.ly/2MyiJUy]

Types of Machine Learning Algorithms

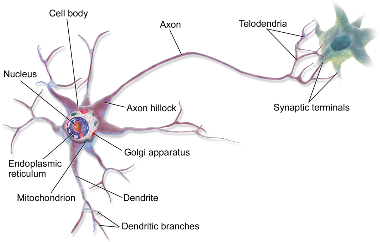
- Supervised learning
  - Input data is labeled, e.g., spam/not-spam or a stock price at a time.
  - Regression vs. classification

- Unsupervised learning
  - Input data is unlabeled.
  - Find hidden structures in data.



From Machine Learning to Deep Learning

- Deep Learning (DL) is part of ML methods based on learning data representations.
- Mimic the neural networks of our brain.



[A. Geron, O'Reilly Media, 2017]

Deep Learning Frameworks

Deep Learning Frameworks

- TensorFlow and Keras
- PyTorch
- Caffe
- ...

TensorFlow

Let's Start with an Example

Hello World

$$\mathtt{c} = \mathtt{a} \times \mathtt{b}$$

Hello World

$$c = a \times b$$

$$d = a + b$$

Hello World

$$c = a \times b$$

$$d = a + b$$

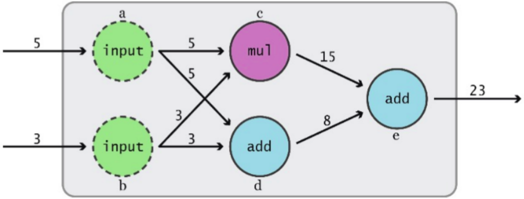
$$e = c + d$$

Hello World

$$c = a \times b$$

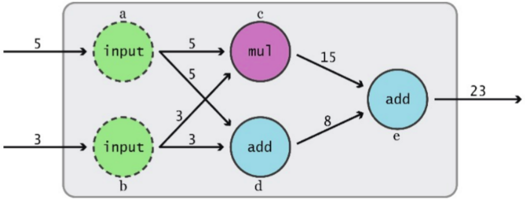
$$d = a + b$$

$$e = c + d$$



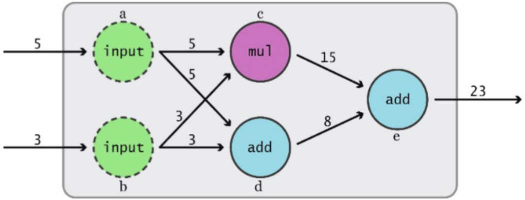
Two Phases of Tensorflow

- Working with TensorFlow involves two main phases.



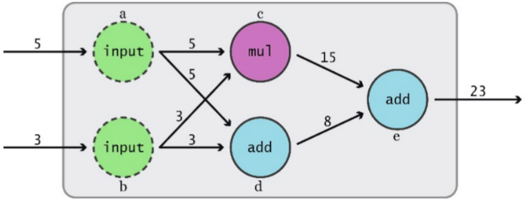
Two Phases of Tensorflow

- Working with TensorFlow involves two main phases.
  1. Build a graph

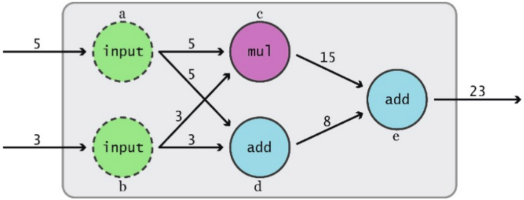


Two Phases of Tensorflow

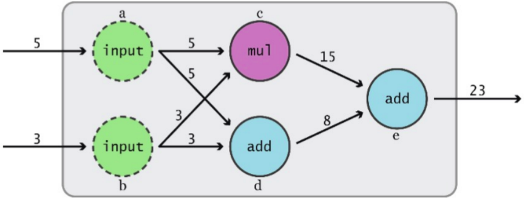
- Working with TensorFlow involves two main phases.
  1. Build a graph
  2. Execute it



Phase 1: Build a Graph



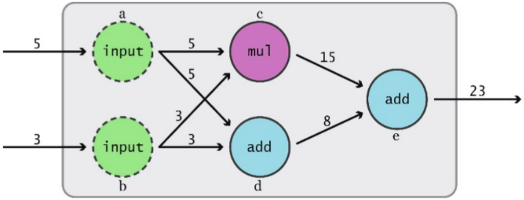
Phase 1: Build a Graph



- `import tensorflow as tf`: it forms an empty default graph.

```
import tensorflow as tf
```

Phase 1: Build a Graph

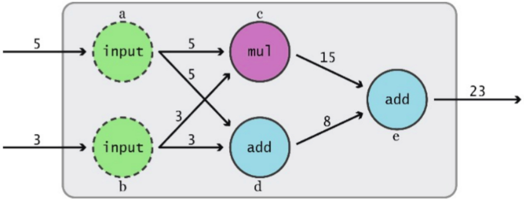


- `import tensorflow as tf`: it forms an empty default graph.

```
import tensorflow as tf
```

```
a = tf.constant(5)
b = tf.constant(3)
```

Phase 1: Build a Graph



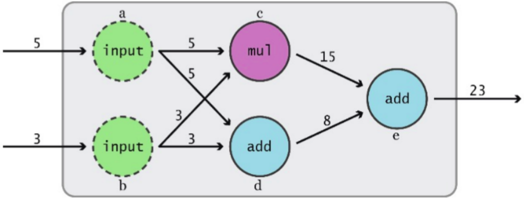
- `import tensorflow as tf`: it forms an empty default graph.

```
import tensorflow as tf
```

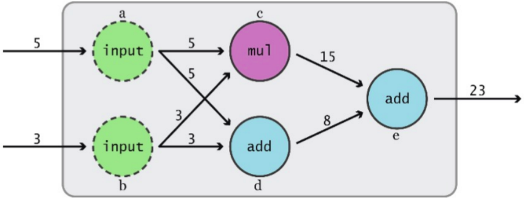
```
a = tf.constant(5)
b = tf.constant(3)
```

```
c = tf.multiply(a, b)
d = tf.add(a, b)
e = tf.add(c, d)
```

Phase 2: Execute a Graph

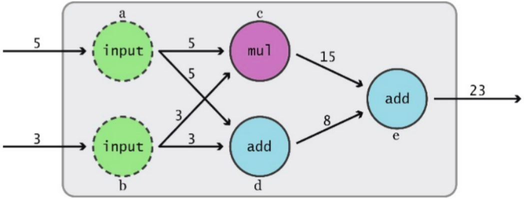


Phase 2: Execute a Graph



- Now run the computations: create and run a session.

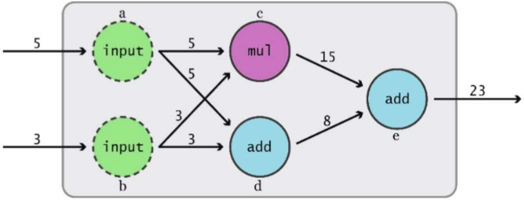
Phase 2: Execute a Graph



- Now run the computations: create and run a session.

```
sess = tf.Session()
print(sess.run(e))
sess.close()
```

Phase 2: Execute a Graph



- Now run the computations: create and run a session.

```
sess = tf.Session()
print(sess.run(e))
sess.close()
```

```
# Alternative way
with tf.Session() as sess:
  print(sess.run(e))
```

The Complete Code

```python
import tensorflow as tf

# Building the Graph
a = tf.constant(5)
b = tf.constant(3)

c = tf.multiply(a, b)
d = tf.add(a, b)
e = tf.add(c, d)

# Executing the Graph
with tf.Session() as sess:
  print(sess.run(e))
```

Visualize the Code

```python
import tensorflow as tf

# Building the Graph
a = tf.constant(5)
b = tf.constant(3)

c = tf.multiply(a, b)
d = tf.add(a, b)
e = tf.add(c, d)

writer = tf.summary.FileWriter("./graphs", tf.get_default_graph())

# Executing the Graph
with tf.Session() as sess:
  print(sess.run(e))
```

Visualize the Code

```python
import tensorflow as tf

# Building the Graph
a = tf.constant(5)
b = tf.constant(3)

c = tf.multiply(a, b)
d = tf.add(a, b)
e = tf.add(c, d)

writer = tf.summary.FileWriter("./graphs", tf.get_default_graph())

# Executing the Graph
with tf.Session() as sess:
  print(sess.run(e))
```

```
tensorboard --logdir="./graphs" --port 6006
```

Let's Give Name to Variables

```python
import tensorflow as tf

# Building the Graph
a = tf.constant(5, name="a")
b = tf.constant(3, name="b")

c = tf.multiply(a, b, name="c_mul")
d = tf.add(a, b, name="d_add")
e = tf.add(c, d, name="e_add")

writer = tf.summary.FileWriter("./graphs", tf.get_default_graph())

# Executing the Graph
with tf.Session() as sess:
  print(sess.run(e))
```

```
tensorboard --logdir="./graphs" --port 6006
```

Tensor Objects

What is Tensor?

- The central unit of data in TensorFlow is the tensor.

What is Tensor?

- The central unit of data in TensorFlow is the tensor.
- An n-dimensional array of primitive values.

Tensor Objects

- `tf.Tensor`

Tensor Objects

- `tf.Tensor`
- Each Tensor object is specified by:
  - Rank
  - Shape
  - Datatype

Tensor Objects - Rank

- The number of dimensions.

Tensor Objects - Rank

- The number of dimensions.
  - rank 0: scalar, e.g., 5

Tensor Objects - Rank

- The number of dimensions.
  - rank 0: scalar, e.g., 5
  - rank 1: vector, e.g., `[2, 5, 7]`

Tensor Objects - Rank

- The number of dimensions.
  - rank 0: scalar, e.g., `5`
  - rank 1: vector, e.g., `[2, 5, 7]`
  - rank 2: matrix, e.g., `[[1, 2], [3, 4], [5, 6]]`

Tensor Objects - Rank

- The number of dimensions.
  - rank 0: scalar, e.g., `5`
  - rank 1: vector, e.g., `[2, 5, 7]`
  - rank 2: matrix, e.g., `[[1, 2], [3, 4], [5, 6]]`
  - rank 3: 3-Tensor

Tensor Objects - Rank

- The number of dimensions.
  - rank 0: scalar, e.g., `5`
  - rank 1: vector, e.g., `[2, 5, 7]`
  - rank 2: matrix, e.g., `[[1, 2], [3, 4], [5, 6]]`
  - rank 3: 3-Tensor
  - rank n: n-Tensor

Tensor Objects - Rank

- The number of dimensions.
  - rank 0: scalar, e.g., 5
  - rank 1: vector, e.g., `[2, 5, 7]`
  - rank 2: matrix, e.g., `[[1, 2], [3, 4], [5, 6]]`
  - rank 3: 3-Tensor
  - rank n: n-Tensor

- `tf.rank` determines the rank of a `tf.Tensor` object.

```
c = tf.constant([[4], [9], [16], [25]])

r = tf.rank(c) # rank 2
```

Tensor Objects - Shape

- The number of elements in each dimension.

Tensor Objects - Shape

- The number of elements in each dimension.
- The `get_shape()` returns the shape of a `tf.Tensor` object.

```
c = tf.constant([[[1, 2, 3], [4, 5, 6]],
                 [[1, 1, 1], [2, 2, 2]]])

s = c.get_shape() # (2, 2, 3)
```

Tensor Objects - Data Types (1/2)

- We can explicitly choose the data type of a `tf.Tensor` object.

Tensor Objects - Data Types (1/2)

- We can explicitly choose the data type of a `tf.Tensor` object.
- `tf.cast()` changes the data type of a `tf.Tensor` object.

```
c = tf.constant(4.0, dtype=tf.float64)

x = tf.constant([1, 2, 3], dtype=tf.float32)
y = tf.cast(x, tf.int64)
```

Tensor Objects - Data Types (2/2)

Data type	Python type	Description
DT_FLOAT	tf.float32	32-bit floating point.
DT_DOUBLE	tf.float64	64-bit floating point.
DT_INT8	tf.int8	8-bit signed integer.
DT_INT16	tf.int16	16-bit signed integer.
DT_INT32	tf.int32	32-bit signed integer.
DT_INT64	tf.int64	64-bit signed integer.
DT_UINT8	tf.uint8	8-bit unsigned integer.
DT_UINT16	tf.uint16	16-bit unsigned integer.
DT_STRING	tf.string	Variable-length byte array. Each element of a Tensor is a byte array.
DT_BOOL	tf.bool	Boolean.
DT_COMPLEX64	tf.complex64	Complex number made of two 32-bit floating points: real and imaginary parts.
DT_COMPLEX128	tf.complex128	Complex number made of two 64-bit floating points: real and imaginary parts.
DT_QINT8	tf.qint8	8-bit signed integer used in quantized ops.
DT_QINT32	tf.qint32	32-bit signed integer used in quantized ops.
DT_QUINT8	tf.quint8	8-bit unsigned integer used in quantized ops.

Tensor Objects - Name

- Each Tensor object has an identifying name.

```
c = tf.constant(4.0, dtype=tf.float64, name="input")
```

Tensor Objects - Name Scopes

- Hierarchically group nodes by their names.

Tensor Objects - Name Scopes

- Hierarchically group nodes by their names.
- `tf.name_scope()` together `with`.

Tensor Objects - Name Scopes

- Hierarchically group nodes by their names.
- `tf.name_scope()` together with.

```
with tf.name_scope("aut"):
  c1 = tf.constant(4, dtype=tf.int32, name="input1") # aut/intput1
  c2 = tf.constant(4.0, dtype=tf.float64, name="input2") # aut/inout2
```

Main Types of Tensors

- Constants, `tf.constant`

Main Types of Tensors

- Constants, `tf.constant`
- Variables, `tf.Variable`

Main Types of Tensors

- Constants, `tf.constant`
- Variables, `tf.Variable`
- Placeholders, `tf.placeholder`

Constants

Constants (1/3)

- `tf.constant`
- The value of a constant Tensor cannot be changed in the future.

Constants (1/3)

- `tf.constant`
- The value of a constant Tensor cannot be changed in the future.

```
tf.constant(<value>, dtype=None, shape=None, name="Const", verify_shape=False)

a = tf.constant([[0, 1], [2, 3]], name="b")
b = tf.constant([[4], [9], [16], [25]], name="c")
```

Constants (2/3)

- The initialization should be with a value, not with operation.

TensorFlow operation	Description
`tf.constant(`*`value`*`)`	Creates a tensor populated with the value or values specified by the argument *`value`*
`tf.fill(`*`shape`*`, `*`value`*`)`	Creates a tensor of shape *`shape`* and fills it with *`value`*
`tf.zeros(`*`shape`*`)`	Returns a tensor of shape *`shape`* with all elements set to 0
`tf.zeros_like(`*`tensor`*`)`	Returns a tensor of the same type and shape as *`tensor`* with all elements set to 0
`tf.ones(`*`shape`*`)`	Returns a tensor of shape *`shape`* with all elements set to 1
`tf.ones_like(`*`tensor`*`)`	Returns a tensor of the same type and shape as *`tensor`* with all elements set to 1
`tf.random_normal(`*`shape`*`, `*`mean`*`, `*`stddev`*`)`	Outputs random values from a normal distribution
`tf.truncated_normal(`*`shape`*`, `*`mean`*`, `*`stddev`*`)`	Outputs random values from a truncated normal distribution (values whose magnitude is more than two standard deviations from the mean are dropped and re-picked)
`tf.random_uniform(`*`shape`*`, `*`minval`*`, `*`maxval`*`)`	Generates values from a uniform distribution in the range [*`minval`*`, `*`maxval`*)
`tf.random_shuffle(`*`tensor`*`)`	Randomly shuffles a tensor along its first dimension

Constants (3/3)

- What's wrong with constants?

Constants (3/3)

- What's wrong with constants?
- Constants are stored in the graph definition.
- This makes loading graphs expensive when constants are big.

Constants (3/3)

- What's wrong with constants?
- Constants are stored in the graph definition.
- This makes loading graphs expensive when constants are big.
- Only use constants for primitive types.
- Use variables for data that requires more memory.

Variables

Variables

- `tf.Variable`
- A variable is a Tensor whose value can be changed.

Variables

- `tf.Variable`
- A variable is a Tensor whose value can be changed.
- `tf.get_variable` creates a variable or returns it if it exists.

Variables

- tf.Variable
- A variable is a Tensor whose value can be changed.
- tf.get_variable creates a variable or returns it if it exists.

```
# not recommended way to make a variable
tf.Variable(<initial-value>, name=<optional-name>)

w = tf.Variable([[0, 1], [2, 3]], name="matrix")

# recommended
tf.get_variable(name, shape=None, dtype=tf.float32, initializer=None,
       regularizer=None, trainable=True, collections=None)

w = tf.get_variable("matrix", initializer=tf.constant([[0, 1], [2, 3]]))
```

Initialize Variables

- Variables should be initialized before being used.

Initialize Variables

- Variables should be initialized before being used.
- Initialize all variables at once.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
```

Initialize Variables

- Variables should be initialized before being used.
- Initialize all variables at once.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
```

- Initialize only a subset of variables.

```
with tf.Session() as sess:
    sess.run(tf.variables_initializer([a, b]))
```

Initialize Variables

- Variables should be initialized before being used.
- Initialize all variables at once.

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
```

- Initialize only a subset of variables.

```
with tf.Session() as sess:
    sess.run(tf.variables_initializer([a, b]))
```

- Initialize a single variable.

```
w = tf.Variable(tf.zeros([784,10]))

with tf.Session() as sess:
    sess.run(w.initializer)
```

Assign Values to Variables (1/3)

- What does it print?

```
w = tf.get_variable("scalar", initializer=tf.constant(2))
w.assign(100)

with tf.Session() as sess:
    sess.run(w.initializer)
    print(sess.run(w))
```

Assign Values to Variables (1/3)

- What does it print?

```
w = tf.get_variable("scalar", initializer=tf.constant(2))
w.assign(100)

with tf.Session() as sess:
    sess.run(w.initializer)
    print(sess.run(w))
```

- Prints 2, because `w.assign(100)` creates an assign op.

```
w = tf.get_variable("scalar", initializer=tf.constant(2))
assign_op = w.assign(100)

with tf.Session() as sess:
    sess.run(w.initializer)
    sess.run(assign_op)
    print(sess.run(w))
```

Assign Values to Variables (2/3)

- What does it print?

```
w = tf.get_variable("scalar", initializer=tf.constant(2))
w_times_two = w.assign(2 * w)

with tf.Session() as sess:
    sess.run(w.initializer)
    print(sess.run(w_times_two))
    print(sess.run(w_times_two))
    print(sess.run(w_times_two))
```

Assign Values to Variables (3/3)

- assign_add() and assign_sub()

```
w = tf.get_variable("scalar", initializer=tf.constant(2))

with tf.Session() as sess:
    sess.run(w.initializer)

    # increment by 10
    print(sess.run(w.assign_add(10)))

    # decrement by 5
    print(sess.run(w.assign_sub(5)))
```

Placeholders

Placeholders

- `tf.placeholder`
- Placeholders are empty variables that will be filled with data later on.

Placeholders

- `tf.placeholder`
- Placeholders are empty variables that will be filled with data later on.

```
tf.placeholder(dtype, shape=None, name=None)

x = tf.placeholder(tf.float32, shape=[None, 10])
```

Feeding Placeholders (1/2)

- What's wrong with this code?

```
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

c = a + b

with tf.Session() as sess:
  print(sess.run(c))
```

Feeding Placeholders (2/2)

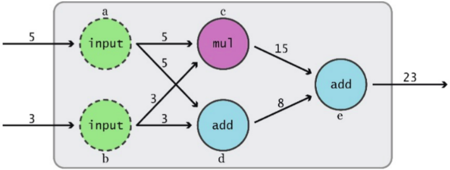
- Supplement the values to placeholders using a dictionary.

```
a = tf.placeholder(tf.float32, shape=[3])

b = tf.constant([5, 5, 5], tf.float32)

c = a + b

with tf.Session() as sess:
  print(sess.run(c, feed_dict={a: [1, 2, 3]}))
```

Dataflow Graphs

Graph

- A graph is composed of two types of objects:
    - Operations
    - Tensors



Common TensorFlow Operations)

TensorFlow operator	Shortcut	Description
`tf.add()`	`a + b`	Adds a and b, element-wise.
`tf.multiply()`	`a * b`	Multiplies a and b, element-wise.
`tf.subtract()`	`a - b`	Subtracts a from b, element-wise.
`tf.divide()`	`a / b`	Computes Python-style division of a by b.
`tf.pow()`	`a ** b`	Returns the result of raising each element in a to its corresponding element b, element-wise.
`tf.mod()`	`a % b`	Returns the element-wise modulo.
`tf.logical_and()`	`a & b`	Returns the truth table of `a & b`, element-wise. `dtype` must be `tf.bool`.
`tf.greater()`	`a > b`	Returns the truth table of `a > b`, element-wise.
`tf.greater_equal()`	`a >= b`	Returns the truth table of `a >= b`, element-wise.
`tf.less_equal()`	`a <= b`	Returns the truth table of `a <= b`, element-wise.
`tf.less()`	`a < b`	Returns the truth table of `a < b`, element-wise.
`tf.negative()`	`-a`	Returns the negative value of each element in a.
`tf.logical_not()`	`~a`	Returns the logical NOT of each element in a. Only compatible with Tensor objects with `dtype` of `tf.bool`.
`tf.abs()`	`abs(a)`	Returns the absolute value of each element in a.
`tf.logical_or()`	`a | b`	Returns the truth table of `a | b`, element-wise. `dtype` must be `tf.bool`.

Managing Multiple Graphs (1/2)

- Calling `import tensorflow` creates the default graph.

Managing Multiple Graphs (1/2)

- Calling `import tensorflow` creates the default graph.
- We can also create additional graphs, by calling `tf.Graph()`.

Managing Multiple Graphs (1/2)

- Calling `import tensorflow` creates the default graph.
- We can also create additional graphs, by calling `tf.Graph()`.
- `tf.get_default_graph()` tells which graph is currently set as the default graph.

Managing Multiple Graphs (1/2)

- Calling `import tensorflow` creates the default graph.
- We can also create additional graphs, by calling `tf.Graph()`.
- `tf.get_default_graph()` tells which graph is currently set as the default graph.

```
import tensorflow as tf

g = tf.Graph()
a = tf.constant(5)

print(a.graph is g)
# Out: False

print(a.graph is tf.get_default_graph())
# Out: True
```

Managing Multiple Graphs (2/2)

- Associate nodes to a right graph using `with` and `as_default()`.

Managing Multiple Graphs (2/2)

- Associate nodes to a right graph using `with` and `as_default()`.

```
import tensorflow as tf

g1 = tf.get_default_graph()
g2 = tf.Graph()

print(g1 is tf.get_default_graph())
# Out: True
```

Managing Multiple Graphs (2/2)

- Associate nodes to a right graph using `with` and `as_default()`.

```
import tensorflow as tf

g1 = tf.get_default_graph()
g2 = tf.Graph()

print(g1 is tf.get_default_graph())
# Out: True
```

```
with g2.as_default():
  print(g1 is tf.get_default_graph())
# Out: False
  print(g2 is tf.get_default_graph())
# Out: True
```

Session

Session

- A `Session` object encapsulates the environment.

Session

- A `Session` object encapsulates the environment.
- Operation objects are executed, and Tensor objects are evaluated.

```
sess = tf.Session()
outs = sess.run(e)
print("outs = {}".format(outs))
sess.close()
```

Session

- A `Session` object encapsulates the environment.
- Operation objects are executed, and Tensor objects are evaluated.
- Session will also allocate memory to store the current values of variables.

```
sess = tf.Session()
outs = sess.run(e)
print("outs = {}".format(outs))
sess.close()
```

Session

- A `Session` object encapsulates the environment.
- Operation objects are executed, and Tensor objects are evaluated.
- Session will also allocate memory to store the current values of variables.

```
sess = tf.Session()
outs = sess.run(e)
print("outs = {}".format(outs))
sess.close()
```

```
# can be written as follows
with tf.Session() as sess:
    outs = sess.run(e)

print("outs = {}".format(outs))
```

Feeding

- A graph can be parameterized to accept external inputs via placeholders.

Feeding

- A graph can be parameterized to accept external inputs via placeholders.
- To feed a placeholder, the input data is passed to the `session.run()`.

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y

with tf.Session() as sess:
    print(sess.run(z, feed_dict={x: 3, y: 4.5}))
    print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

Feeding

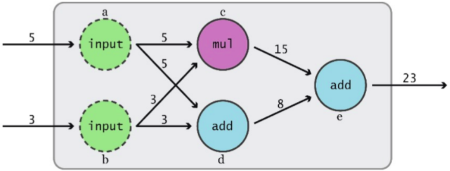
- A graph can be parameterized to accept external inputs via placeholders.
- To feed a placeholder, the input data is passed to the `session.run()`.
- Each key corresponds to a placeholder variable name.

```
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)
z = x + y

with tf.Session() as sess:
    print(sess.run(z, feed_dict={x: 3, y: 4.5}))
    print(sess.run(z, feed_dict={x: [1, 3], y: [2, 4]}))
```

Fetches

- To fetch a list of outputs of nodes.

```
with tf.Session() as sess:
    fetches = [a, b, c, d, e]
    outs = sess.run(fetches)

print("outs = {}".format(outs))
```



Session.run() vs. Tensor.eval()

- Two ways to evaluate part of graph: `Session.run` and `Tensor.eval`.

`Session.run()` vs. `Tensor.eval()`

- Two ways to evaluate part of graph: `Session.run` and `Tensor.eval`.
- You can use `sess.run()` to fetch the values of many tensors in the same step.

Session.run() vs. Tensor.eval()

- Two ways to evaluate part of graph: `Session.run` and `Tensor.eval`.
- You can use `sess.run()` to fetch the values of many tensors in the same step.

```
t = tf.constant(42.0)
u = tf.constant(37.0)
tu = tf.multiply(t, u)
ut = tf.multiply(u, t)
```

Session.run() vs. Tensor.eval()

- Two ways to evaluate part of graph: Session.run and Tensor.eval.
- You can use sess.run() to fetch the values of many tensors in the same step.

```
t = tf.constant(42.0)
u = tf.constant(37.0)
tu = tf.multiply(t, u)
ut = tf.multiply(u, t)
```

```
with sess.as_default():
   tu.eval()  # runs one step
   ut.eval()  # runs one step
```

Session.run() vs. Tensor.eval()

- Two ways to evaluate part of graph: Session.run and Tensor.eval.
- You can use sess.run() to fetch the values of many tensors in the same step.

```
t = tf.constant(42.0)
u = tf.constant(37.0)
tu = tf.multiply(t, u)
ut = tf.multiply(u, t)
```

```
with sess.as_default():
   tu.eval()  # runs one step
   ut.eval()  # runs one step
```

```
with sess.as_default():
   sess.run([tu, ut])  # evaluates both tensors in a single step
```

Saving and Restoring Models

Saving Models

- Save a model's parameters in disk.

Saving Models

- Save a model's parameters in disk.
- Create a `Saver` node at the end of the construction phase.

Saving Models

- Save a model's parameters in disk.
- Create a `Saver` node at the end of the construction phase.
- In the execution phase, call its `save()` method whenever you want to save the model.

Saving Models

- Save a model's parameters in disk.
- Create a `Saver` node at the end of the construction phase.
- In the execution phase, call its `save()` method whenever you want to save the model.

```
w = tf.Variable([[0, 0, 0]], dtype=tf.float32, name="weights")
[...]
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

Saving Models

- Save a model's parameters in disk.
- Create a Saver node at the end of the construction phase.
- In the execution phase, call its save() method whenever you want to save the model.

```
w = tf.Variable([[0, 0, 0]], dtype=tf.float32, name="weights")
[...]
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

```
with tf.Session() as sess:
  sess.run(init)
  sess.run(train, {x: x_data, y_true: y_data})
  saver.save(sess, "/tmp/my_model_final.ckpt")
```

Restoring Models

- Create a `Saver` node at the end of the construction phase.

Restoring Models

- Create a `Saver` node at the end of the construction phase.
- At the begining of the execution phase call its `restore()` method.

Restoring Models

- Create a `Saver` node at the end of the construction phase.
- At the begining of the execution phase call its `restore()` method.
  - Instead of initializing the variables using the `init` node.

Restoring Models

- Create a Saver node at the end of the construction phase.
- At the begining of the execution phase call its restore() method.
  - Instead of initializing the variables using the init node.

```
w = tf.Variable([[0, 0, 0]], dtype=tf.float32, name="weights")
[...]
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

Restoring Models

- Create a Saver node at the end of the construction phase.
- At the begining of the execution phase call its restore() method.
  - Instead of initializing the variables using the init node.

```
w = tf.Variable([[0, 0, 0]], dtype=tf.float32, name="weights")
[...]
init = tf.global_variables_initializer()
saver = tf.train.Saver()
```

```
with tf.Session() as sess:
  saver.restore(sess, "/tmp/my_model_final.ckpt")
  [...]
```

TensorBoard

TensorBoard (1/2)

- TensorFlow provides a utility called TensorBoard.

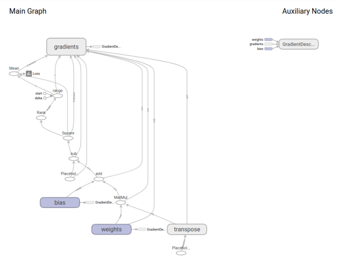
TensorBoard (1/2)

- TensorFlow provides a utility called TensorBoard.
- To visualize your model, you need to write the graph definition and some training stats to a log directory that TensorBoard will read from.

TensorBoard (2/2)

- Add the following code at the very end of the construction phase.

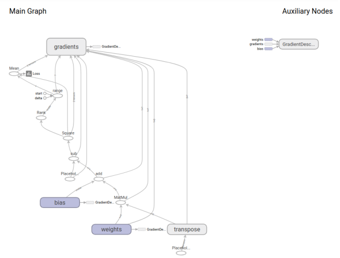
```
logdir = "."
mse_summary = tf.summary.scalar("MSE", cost)
file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())
file_writer.close()
```



TensorBoard (2/2)

- Add the following code at the very end of the construction phase.
- The first line writes the cost.

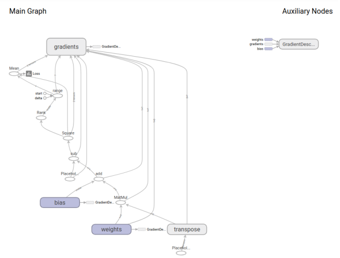
```
logdir = "."
mse_summary = tf.summary.scalar("MSE", cost)
file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())
file_writer.close()
```



TensorBoard (2/2)

- Add the following code at the very end of the construction phase.
- The first line writes the cost.
- The second line creates a FileWriter that writes summaries of the graph.

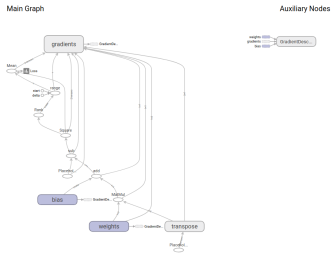
```
logdir = "."
mse_summary = tf.summary.scalar("MSE", cost)
file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())
file_writer.close()
```



TensorBoard (2/2)

- Add the following code at the very end of the construction phase.
- The first line writes the `cost`.
- The second line creates a `FileWriter` that writes summaries of the graph.
- Start the TensorBoard web server (port 6006): `tensorboard --logdir .`

```
logdir = "."
mse_summary = tf.summary.scalar("MSE", cost)
file_writer = tf.summary.FileWriter(logdir, tf.get_default_graph())
file_writer.close()
```



Summary

Summary

- Dataflow graph
- Tensors: constants, variables, placeholders
- Session
- Save and restore models

Questions?