

IIT KHARAGPUR

IIT GANDHINAGAR


NPTEL ONLINE
CERTIFICATION COURSES

Scalable Data Science

Lecture 20: Distributed Machine Learning

Sourangshu Bhattacharya
Computer Science and Engineering
IIT KHARAGPUR

In the previous lectures:

- Outline:
  - Big Data platforms.

In this Lecture:

- Outline:
  - Motivation
    - Large scale machine learning -
    - Edge computing – autonomous vehicles
  - Architectures
  - Platforms
    - Tensorflow

Large scale Machine Learning

Large Scale Machine Learning

- Big businesses collect lots of data
- These data can be analyzed to provide:
  - Insights – analytics
  - Patterns – Data mining
  - Predictions / Decisions – Supervised learning
  - Anomaly / Surprise – Unsupervised learning
- Processing should be:
  - Fast
  - Relaible
  - Flexible

Large scale Machine Learning

Industry	Use Case
Financial Services	• Show correlation between services purchased and investments/trades made • Identify customer segments • Recommendations for research articles to drive trading
eCommerce	• Show types of events person will like • Decision tree based on likelihood to click through • Recommendations for a large “cold start” population
Gaming	• Clustering for user profiles • Correlation between attributes of a game and behavior • Churn analysis
Healthcare	• Recommend tests or other offerings • Identify factors/trends that lead to disease

Big Data

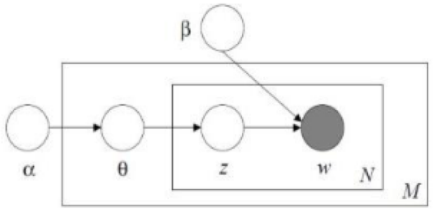
Search engine index: 10^{10} pages (10^{12} tokens)

Search engine logs: 10^{12} impressions and 10^{9} clicks every year

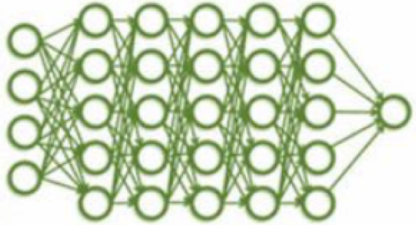
Social networks: 10^{9} nodes and 10^{12} edges

Tasks	Typical training data
Image classification	Millions of labeled images
Speech recognition	Thousands of hours of annotated voice data
Machine translation	Tens of millions of bilingual sentence pairs
Go playing	Tens of millions of expert moves

Big Models



LightLDA: LDA with 10^6 topics (10^{11} parameters); More topics → better performance in ad selection and click predictions



DistBelief: DNN with 10^{10} weights; Deeper and larger networks → better performance in image classification.

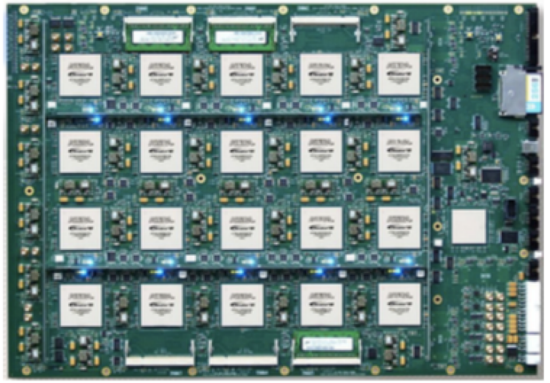
Human brain: 10^{11} neurons and 10^{15} connections, much larger than any existing ML model.

Big Compute

- Large computer clusters and highly parallel computational architectures

Cloud Computing

GPU Cluster

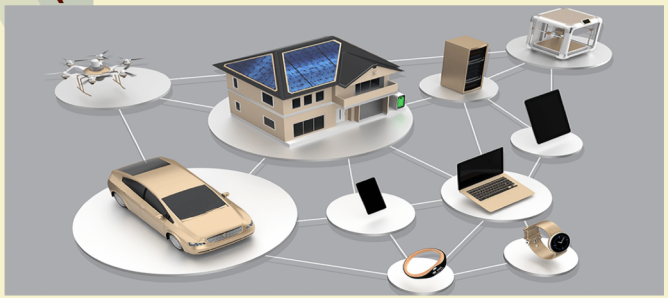


FPGA Farm

Edge Computing

Intelligent Transportation

IOT (connected vehicles / devices)



Autonomous vehicles

Monitoring / Surveillance system with traffic forecasts route recommendation

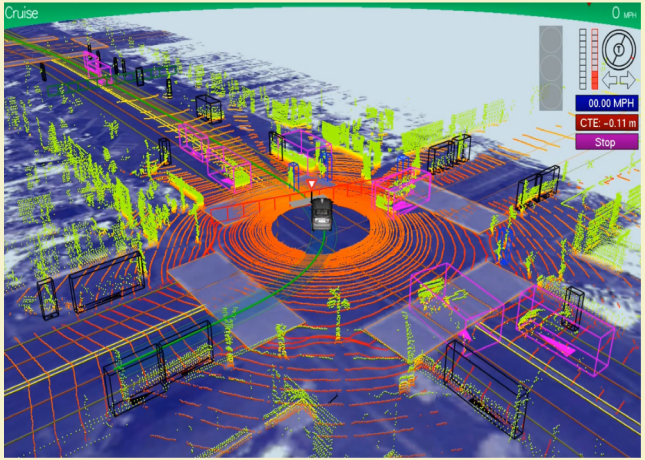
Autonomous vehicles

Lots of sensor data are generated every second

Autonomous cars contain:

- Embedded Computers
- GPS receivers
- Short-range wireless network interfaces
- In-car sensors

Google's self driving car gathers 750Mb/s [Source]



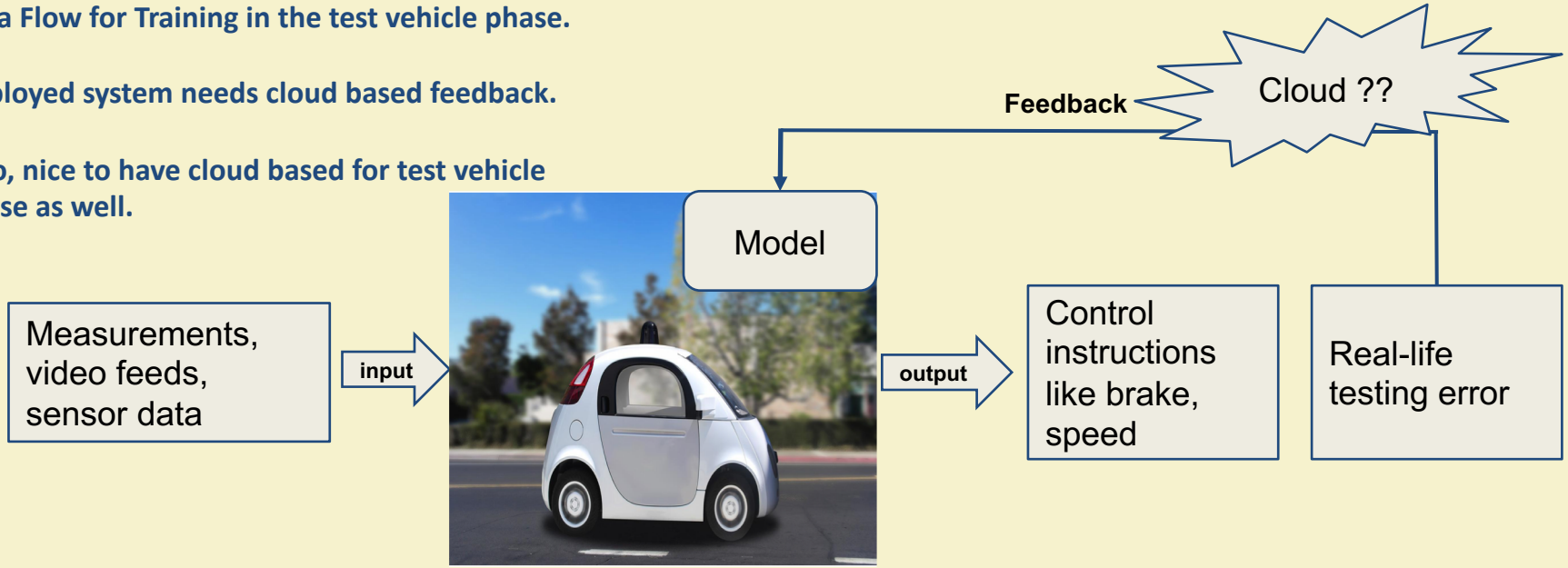
Google Car "sees" while making a turn

Autonomous Vehicles

Data Flow for Training in the test vehicle phase.

Deployed system needs cloud based feedback.

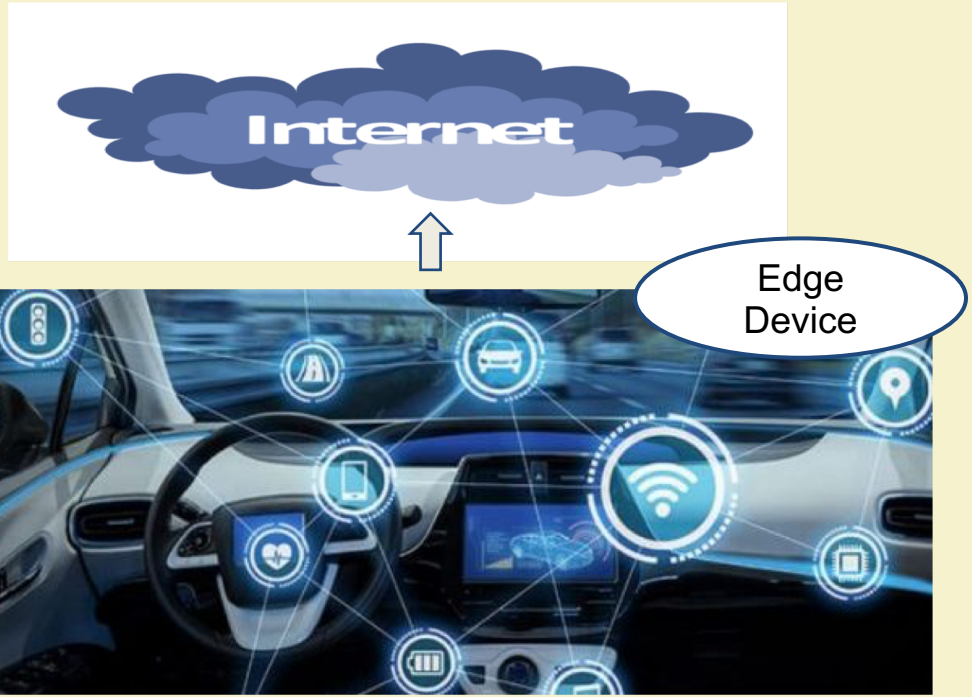
Also, nice to have cloud based for test vehicle Phase as well.



Device to Cloud data transfer

All vehicles are connected to cloud.

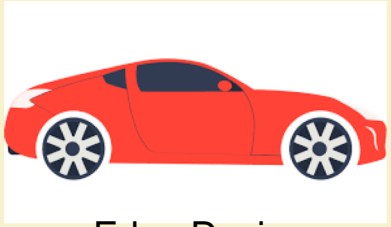
Transportation related Data **needs to be** transferred From Device to cloud



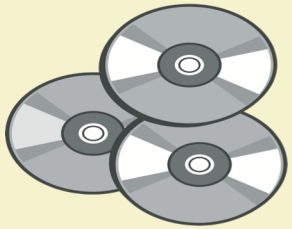
Data Transfer scheme

Collection of training data from test vehicles in the initial phase.

Current Scenario:



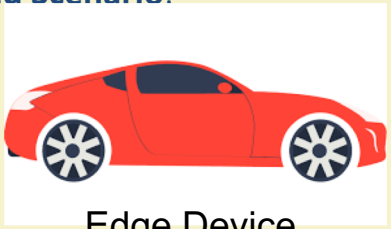
Edge Device



PBytes of Data

Data synced up offline with the cloud

Desired Scenario:



Edge Device

Data Gradation and store

Online sync-up of graded data

Need for data reduction

Futuristic scenario: Realtime edge to core data transfer from autonomous vehicles.

- Raw data collected by an autonomous vehicle per hour: 1 TB
  - 1440 Mbits per second (2 megapixel * 24 bits per pixel * 30 frames per second)
- Data compression: 1 : 0.014
  - Compression ratio: 70 : 1 using MPEG (source: Wikipedia)
- Bandwidth needed per car for realtime data upload: 3.9 MBps
  - Assuming 100 cars connect to a base station: 390 MBps
- Target edge to core data transfer rate: ~ 10 MBps (Fast ethernet)
-
- Compression required 39 : 1 over and above the current MPEG compression

Reference: https://devblogs.nvidia.com/training-self-driving-vehicles-challenge-scale/

Architectures

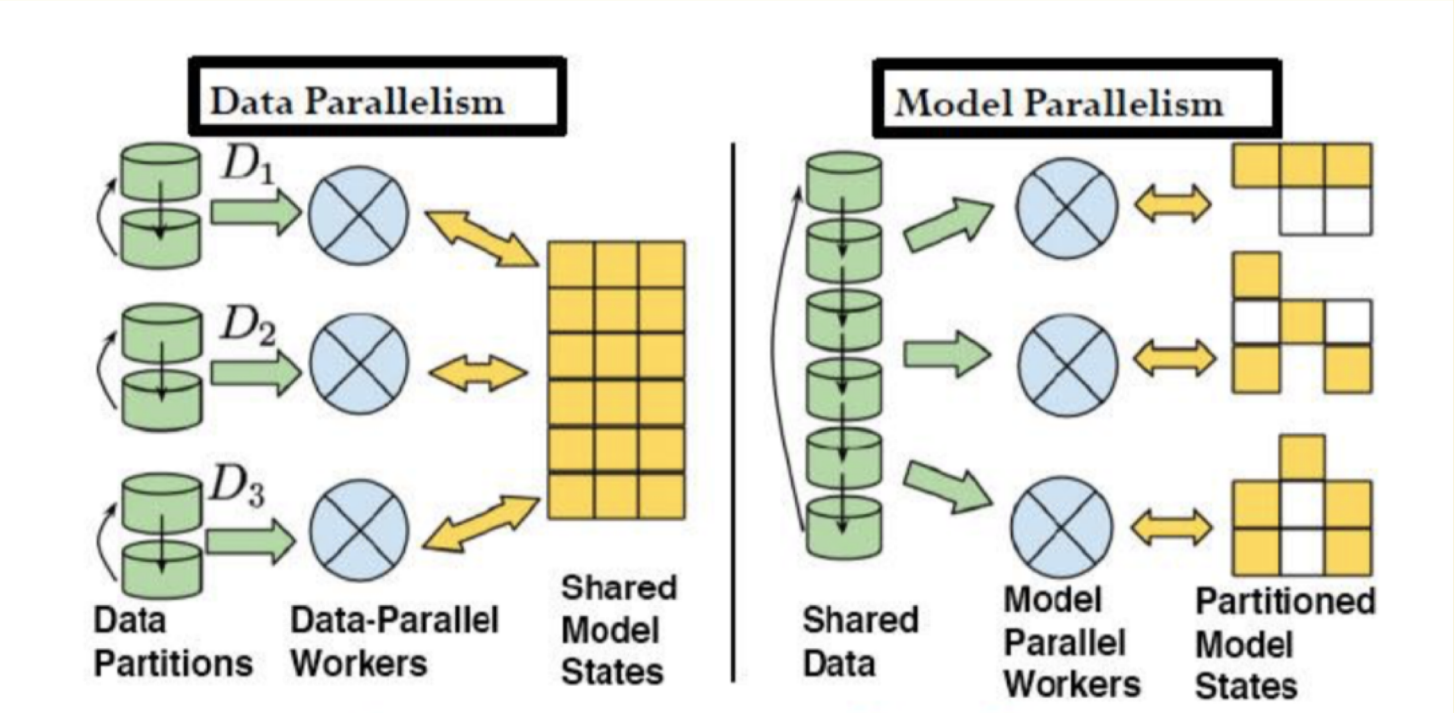
Architectures for Distributed ML

- Data Centric: Train over large data
  - Data split over multiple machines
  - Model replicas train over different parts of data and communicate model information periodically
- Model Centric: Train over large models
  - Models split over multiple machines
  - A single training iteration spans multiple machines
- Graph Centric: Train over large graphs
  - Partitions data as graph associated with every vertex/edge;
  - Parallel apply update functions are operations on a vertex and transforming data in the scope of the vertex;

Architectures for Distributed ML

- Data Centric: Train over large data
  - Map Reduce, Spark
- Model Centric: Train over large models
  - Disbelief, Parameter Server, Petuum
- Graph Centric: Train over large graphs
  - Pregel, GraphLab

Data vs Model parallel



TensorFlow

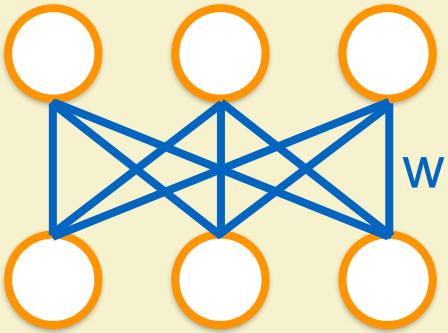
Tensors: n-dimensional arrays

- Vector: 1-D tensor
- Matrix: 2-D tensor

Deep learning process are flows of tensors

- A sequence of tensor operations
- Can represent also many machine learning algorithms

As ReLu network

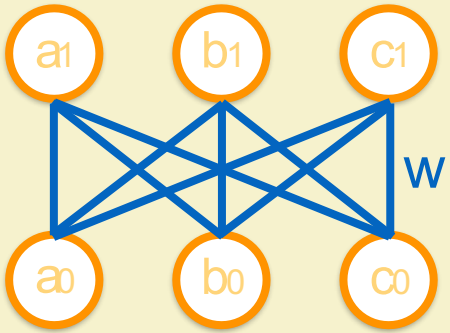


$$\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} \cdot \begin{bmatrix} w_{a,a} & w_{a,b} & w_{a,c} \\ w_{b,a} & w_{b,b} & w_{b,c} \\ w_{c,a} & w_{c,b} & w_{c,c} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix}$$

$$a_1 = \text{relu}(a_1)$$
$$b_1 = \text{relu}(b_1)$$
$$c_1 = \text{relu}(c_1)$$

With TensorFlow

```
import tensorflow as tf
```



x w

$$\begin{bmatrix} a_0 & b_0 & c_0 \end{bmatrix} \cdot \begin{bmatrix} w_{a,a} & w_{a,b} & w_{a,c} \\ w_{b,a} & w_{b,b} & w_{b,c} \\ w_{c,a} & w_{c,b} & w_{c,c} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix}$$

```
y = tf.matmul(x, w)
```

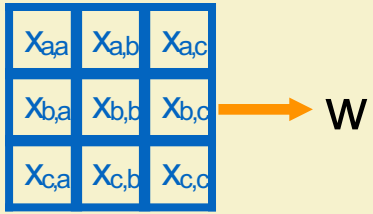
$a_1 = relu(a_1)$

$b_1 = relu(b_1)$

$c_1 = relu(c_1)$

```
out = tf.nn.relu(y)
```

Define Tensors



Variable(<initial-value>,
name=<optional-name>)

```
import tensorflow as tf
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
```

Variable stores the state of current execution

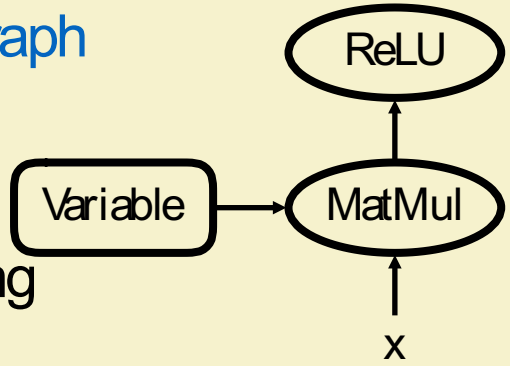
Others are operations

TensorFlow

Code so far defines a data flow graph

Each variable corresponds to a node in the graph, not the result

Can be confusing at the beginning




```
import tensorflow as tf
w =tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
```

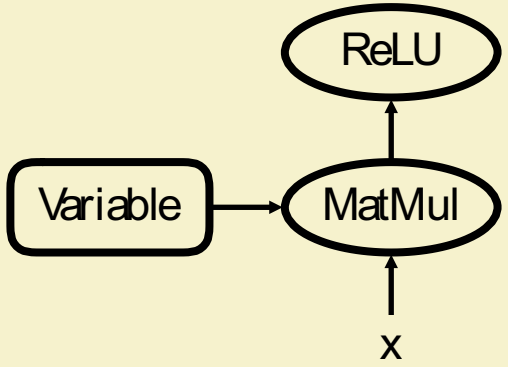
TensorFlow

Code so far defines a data flow graph

Needs to specify how we

- want to execute the graph

Session :Manage resource for graph execution



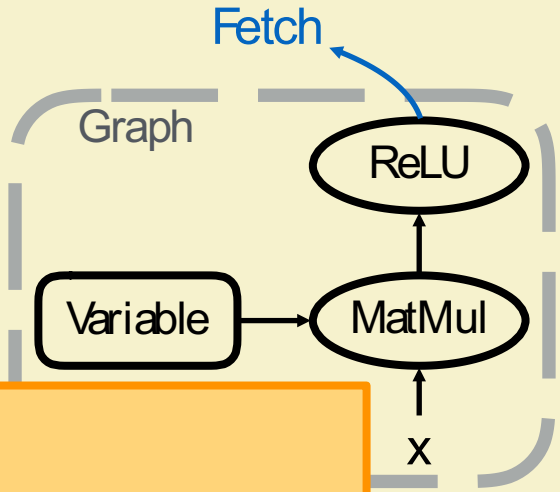

```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
result = sess.run(relu_out)
```

TensorFlow

Retrieve content from a node

We have assembled the pipes

Fetch the liquid

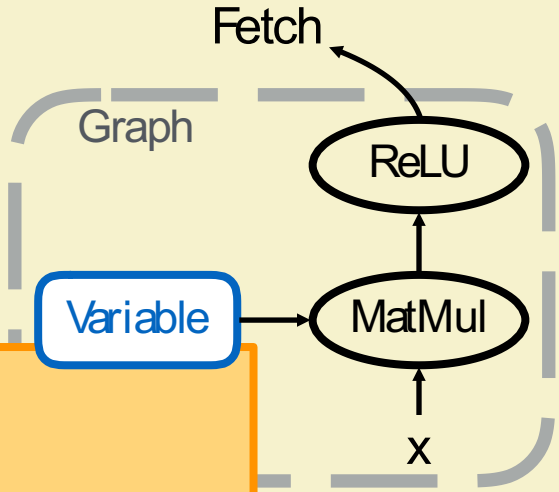



```
import tensorflow as tf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
print sess.run(relu_out)
```

Variable

Variable is an empty node

Fill in the content of a Variable node




```
import tensorflow astf
sess = tf.Session()
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out)
```

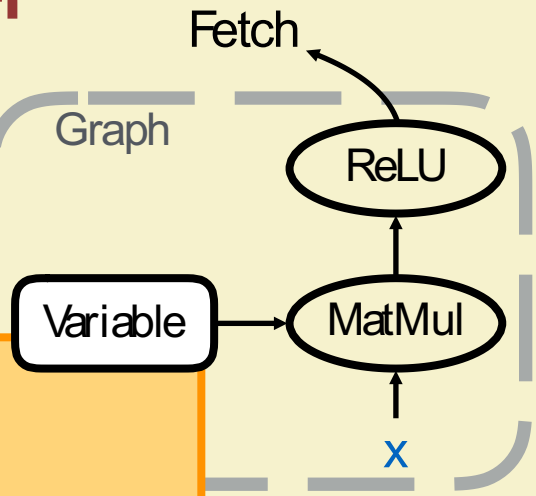
Placeholder

How about x?

placeholder(<data type>,
shape=<optional-shape>,
name=<optional-name>)

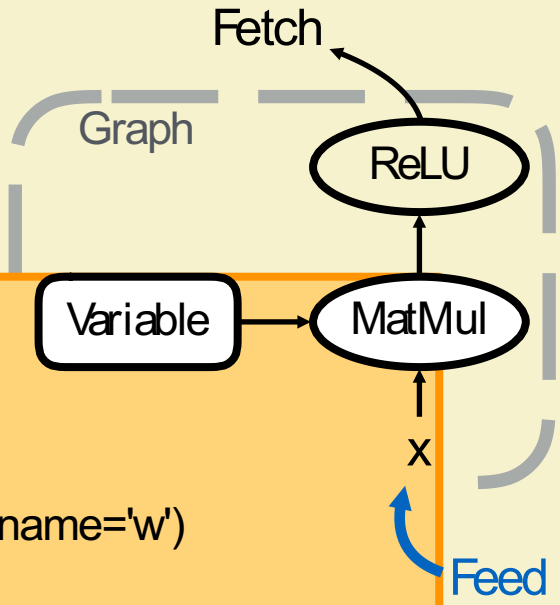
Its content will be fed

```
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float", [1,3])
w = tf.Variable(tf.random_normal([3, 3]),name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out)
```



Feed

Pump liquid into the pipe

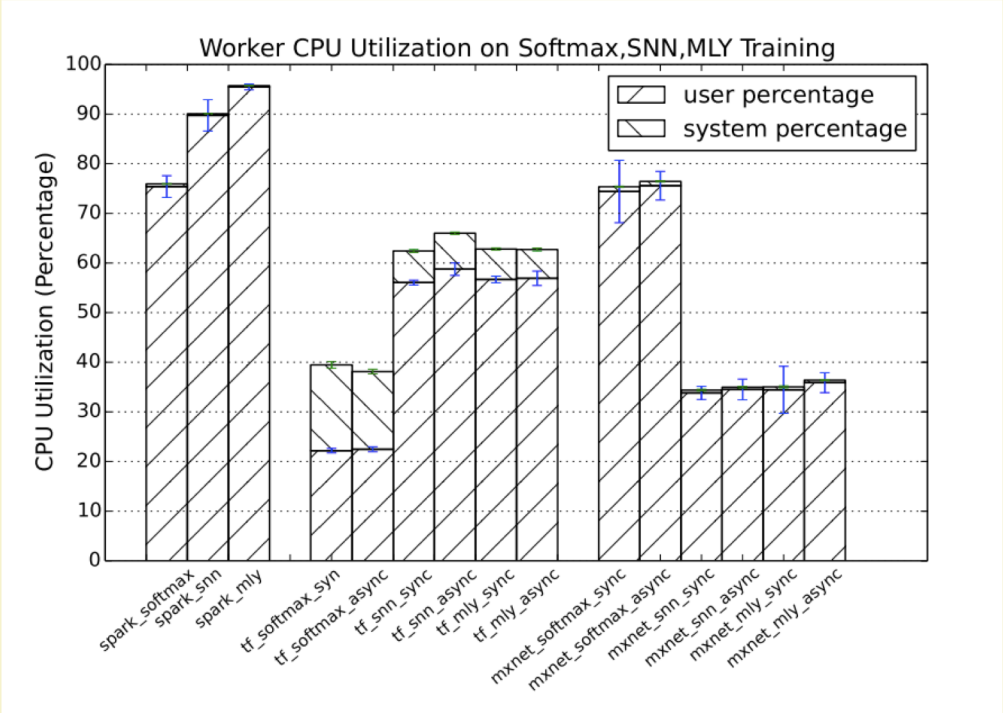



```
import numpy as np
import tensorflow as tf
sess = tf.Session()
x = tf.placeholder("float", [1, 3])
w = tf.Variable(tf.random_normal([3, 3]), name='w')
y = tf.matmul(x, w)
relu_out = tf.nn.relu(y)
sess.run(tf.initialize_all_variables())
print sess.run(relu_out, feed_dict={x:np.array([[1.0, 2.0,3.0]])})
```

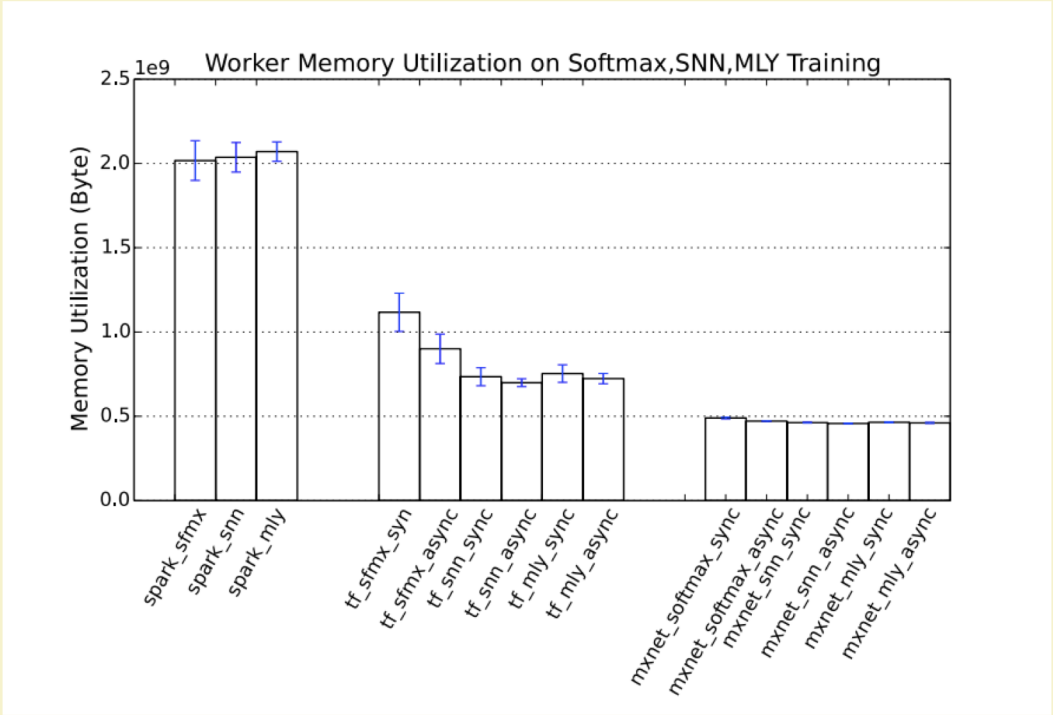
Comparison

- Spark
- Tensorflow
- MXNet

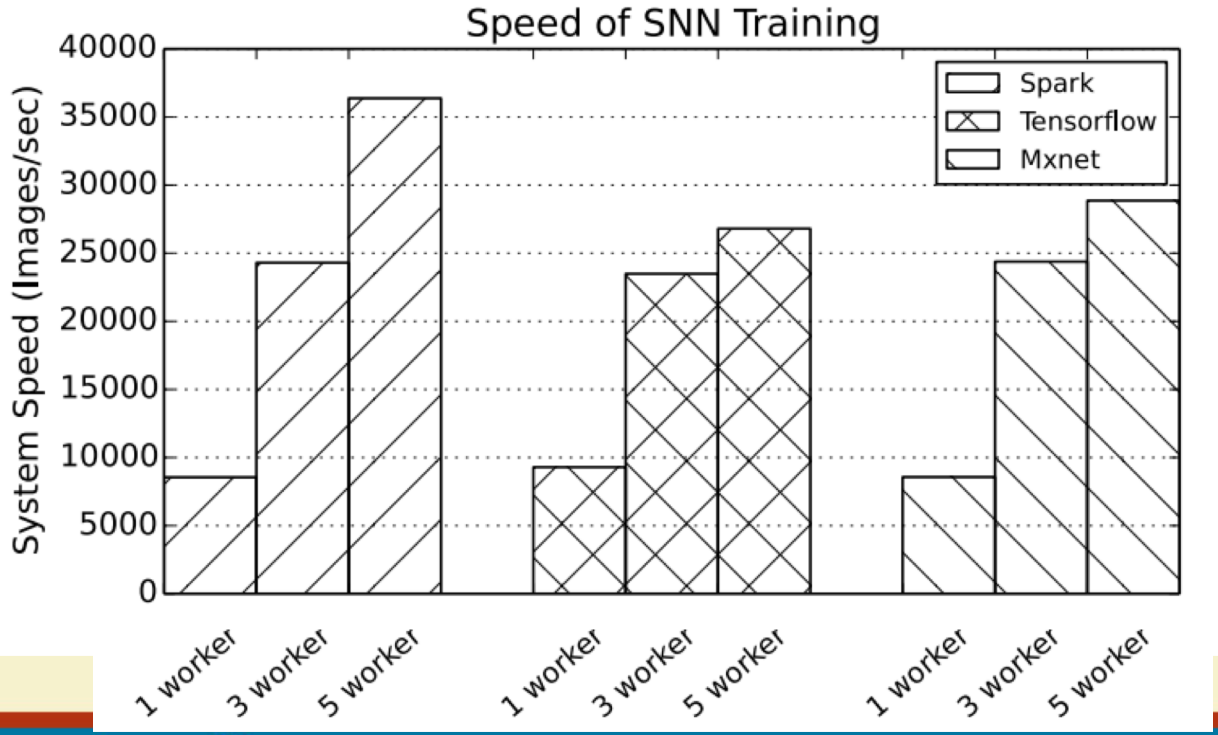
Comparison


Worker CPU Utilization on Softmax,SNN,MLY Training
CPU Utilization (Percentage)
user percentage
system percentage
0
10
20
30
40
50
60
70
80
90
100
spark_softmax
spark_snn
spark_mly
tf_softmax_syn
tf_softmax_async
tf_snn_sync
tf_snn_async
tf_mly_sync
tf_mly_async
mxnet_softmax_sync
mxnet_softmax_async
mxnet_snn_sync
mxnet_snn_async
mxnet_mly_sync
mxnet_mly_async

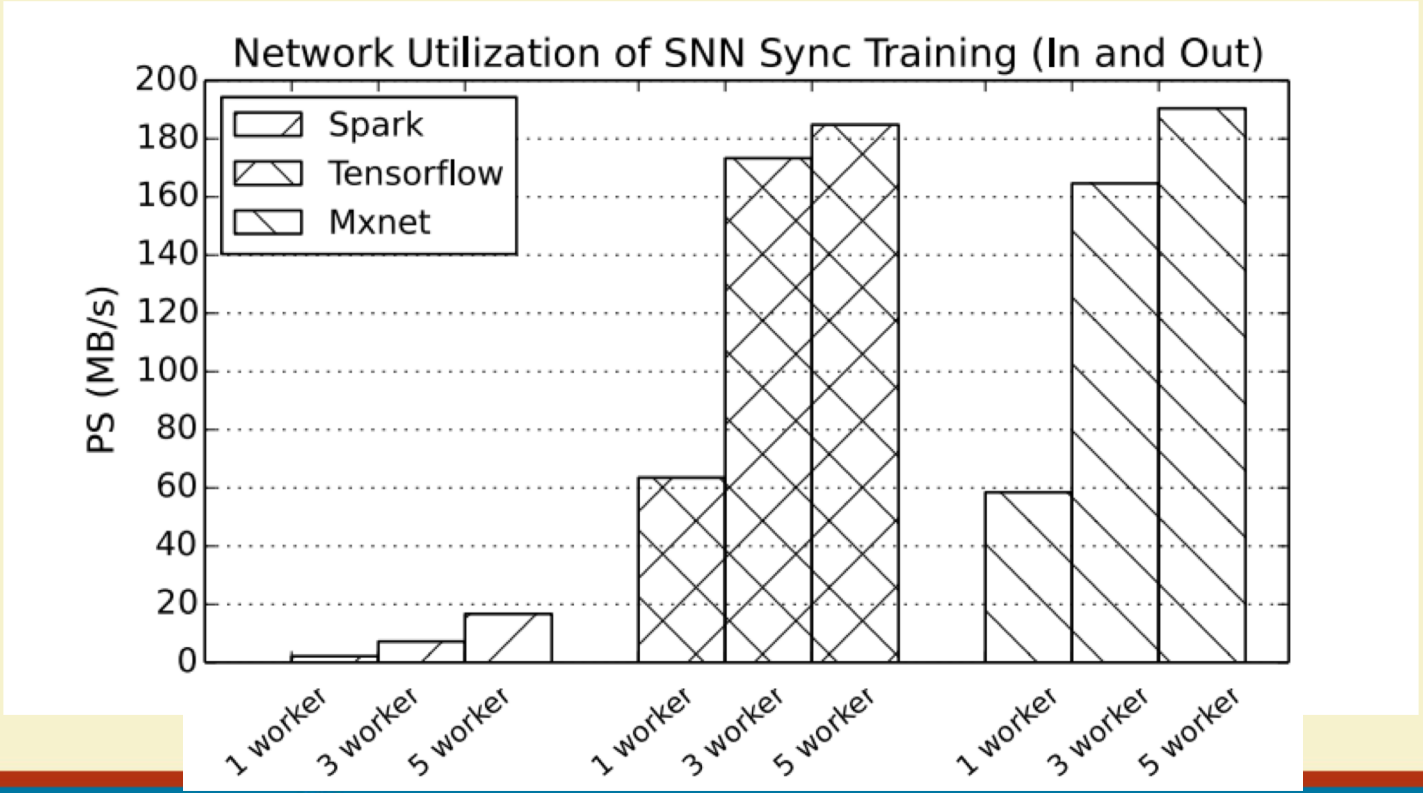
Comparison


Worker Memory Utilization on Softmax,SNN,MLY Training
1e9
Memory Utilization (Byte)
2.5
2.0
1.5
1.0
0.5
0.0
spark_sfmx
spark_snn
spark_mly
tf_sfmx_syn
tf_sfmx_async
tf_snn_sync
tf_snn_async
tf_mly_sync
tf_mly_async
mxnet_softmax_sync
mxnet_softmax_async
mxnet_snn_sync
mxnet_snn_async
mxnet_mly_sync
mxnet_mly_async

Comparison


Speed of SNN Training
System Speed (Images/sec)
40000
35000
30000
25000
20000
15000
10000
5000
0
Spark
Tensorflow
Mxnet
1 worker
3 worker
5 worker
1 worker
3 worker
5 worker
1 worker
3 worker
5 worker

Comparison


Network Utilization of SNN Sync Training (In and Out)
PS (MB/s)
200
180
160
140
120
100
80
60
40
20
0
Spark
Tensorflow
Mxnet
1 worker
3 worker
5 worker
1 worker
3 worker
5 worker
1 worker
3 worker
5 worker

Conclusion:

- We have seen:
  - Motivation
    - Large scale machine learning -
    - Edge computing – autonomous vehicles
  - Architectures
  - Platforms
    - Tensorflow
    - Comparison

References:

- A Comparison of Distributed Machine Learning Platforms. Kuo Zhang Salem Alqahtani Murat Demirbas. 2017 26th International Conference on Computer Communication and Networks (ICCCN)

- Introduction to Tensor flow.

Thank You!!