MXNet: Flexible and Efficient Library for Deep Learning

from Distributed GPU Clusters to Embedded Systems

Tianqi Chen

Mu Li
Deep Learning

Learns multiple levels of representations of data
Significantly improve many applications on multiple domains

- image understanding
- speech recognition
- natural language processing

“deep learning” trend in the past 10 years
Image classification

multilevel feature extractions from raw pixels to semantic meanings

explore spatial information with convolution layers
Image Classification

State-of-the-art networks have tens to hundreds layers

- Hard to define the network
  - the definition of the inception network has >1k lines of codes in Caffe
- A single image requires billions floating-point operations
  - Intel i7 ~500 GFLOPS
  - Nvidia Titan X: ~5 TFLOPS
- Memory consumption is linear with number of layers
Language Modeling

- Variable length of input and output sequences
- State-of-the-art networks have many layers
  - Billions of floating-point operations per sentence
  - Memory consumption is linear with both sequence length and number of layers

Diagram:

- Recurrent neural networks:
  - Input
  - State
  - Output

Example:

<go> hello world !
MXNet Highlights

- Flexibility
- Efficiency
- Portability
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**
  - Memory Optimization

- **Portability**
  - Runs Everywhere
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**
  - Memory Optimization

- **Portability**
  - Runs Everywhere
Deep Learning Workflow

Computational Graph of the Deep Architecture

forward

input → fullc → sigmoid → fullc → softmax

backward

∂ softmax → ∂ fullc → ∂ sigmoid → ∂ fullc → ∂ input

loss → label
Deep Learning Workflow

Computational Graph of the Deep Architecture

- **forward**
  - `input` → `fullc` → `sigmoid` → `fullc` → `softmax` → `loss` → `label`
- **backward**
  - `softmax` → `fullc` → `sigmoid` → `fullc` → `input`

Updates and Interactions with the graph

- Parameter update
- Beam search
- Feature extraction ...

\[ w = w - \eta \frac{\partial f(w)}{\partial w} \]

- Involves high dimensional array (tensor) operations in both direction
- How to program a typical DL application?
Imperative Programs

- Execute operations step by step.
- \( c = b \times a \) invokes a kernel operation
- Numpy programs are imperative

```python
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```
Declarative Programs

- Declares the computation
- Compiles into a function
- \( C = B \times A \) only specifies the requirement
- SQL is declarative

\[
\begin{align*}
A &= \text{Variable('A')} \\
B &= \text{Variable('B')} \\
C &= B \times A \\
D &= C + 1 \\
f &= \text{compile}(D) \\
d &= f(A=np.ones(10), B=np.ones(10)*2)
\end{align*}
\]
Imperative vs. Declarative Programs

- Imperative programs are straightforward and flexible.
- Take advantage of language native features (loop, condition)

```python
import numpy as np

a = np.ones(10)
b = np.ones(10) * 2
c = b * a
print(c)
d = c + 1
```

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + 1
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```

Easy to tweak with python codes
The diagram illustrates the concept of imperative vs. declarative programs.

Declarative programs see the entire graph, offering more chances for optimization and being easy to save and load the computation structure.

Which program uses less memory to obtain $d$?

**Imperative Program**

```python
import numpy as np
a = np.ones(10)
b = np.ones(10) * 2
c = b * a
d = c + 1
```

**Declarative Program**

```
A = Variable('A')
B = Variable('B')
C = B * A
D = C + 1
f = compile(D)
d = f(A=np.ones(10), B=np.ones(10)*2)
```

- **c cannot** share memory with **d**, because it could be used in future.
- **C can** share memory with **D**, because **C** cannot be seen by user.
Imperative vs. Declarative for Deep Learning

Computational Graph of the Deep Architecture

- forward
- backward

- Needs heavy optimization, fits **declarative** programs

Updates and Interactions with the graph

- Parameter update
- Beam search
- Feature extraction ...

\[ \mathbf{w} = \mathbf{w} - \eta \frac{\partial f}{\partial \mathbf{w}} \]

- Needs mutation and more language native features, good for **imperative** programs
MXNet: Mix the Flavors Together

**Imperative NDArray API**

```python
>>> import mxnet as mx
>>> a = mx.nd.zeros((100, 50))
>>> a.shape
(100L, 50L)
>>> b = mx.nd.ones((100, 50))
>>> c = a + b
>>> b += c
```

**Declarative Symbolic Executor**

```python
>>> import mxnet as mx
>>> net = mx.symbol.Variable('data')
>>> net = mx.symbol.FullyConnected(data=net, num_hidden=128)
>>> net = mx.symbol.SoftmaxOutput(data=net)
>>> type(net)
<class 'mxnet.symbol.Symbol'>
>>> texec = net.simple_bind(data=data_shape)
```
Mixed Style Training Loop in MXNet

executor = declarative_symbol.bind()
for i in range(3):
    train_iter.reset()
    for dbatch in train_iter:
        args["data"][:] = dbatch.data[0]
        args["softmax_label"][:] = dbatch.label[0]
        executor.forward(is_train=True)
        executor.backward()
    for key in update_keys:
        args[key] -= learning_rate * grads[key]
Mixed API for Quick Extensions

- Runtime switching between different graphs depending on input
- Useful for sequence modeling and image size reshaping

Various length examples

Bucketing

Make use of imperative code in python, **10 lines** of additional python code
3D Image Construction

100 lines of Python codes
3D Image Construction

100 lines of Python codes
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**

- **Portability**
  - Memory Optimization
  - Runs Everywhere
Need for Parallelization

- Parallelize workload on multiple GPUs
- Fine grained parallelization of small kernels
- Overlap of memory copy with computation

👍 Fully concurrent

👎 Serial
Writing Parallel Programs is Painful

Hard to overlap computation with communication due to dependencies
Auto Parallelization for Mixed Programs

Write **serial** programs

```python
>>> import mxnet as mx
>>> A = mx.nd.ones((2,2)) * 2
>>> C = A + 2
>>> B = A + 1
>>> D = B * C
```

Run in **parallel**

```
A = 2
C = A + 2
B = A + 1
D = B \times C
```
Auto Parallelization for Mixed Programs

- Schedules any resources includes array, random number generator

```python
>>> import mxnet as mx
>>> A = mx.nd.ones((2,2)) *2
>>> C = A + 2
>>> B = A + 1
>>> del A

>>> import mxnet as mx
>>> A = mx.nd.uniform(shape, 10, -10)
>>> B = mx.nd.uniform(shape, 10, -10)
```
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**
  - Memory Optimization

- **Portability**
  - Runs Everywhere
Model Parallelism

Layer 1

Layer 2

Layer 3
Model Parallelism

Layer 1

Layer 2

Layer 3
Model Parallelism

Layer 1
Layer 2
Layer 3

Time for one epoch on PTB:

- Layer 1: 100 sec
- Layer 2: 200 sec
- Layer 3: 300 sec

Bar chart showing time reduction with increasing number of GPUs:

- 1 GPU: 400 sec
- 2 GPUs: 300 sec (2.1x faster)
- 4 GPUs: 200 sec

num of GPUs

Data Parallelism
Data Parallelism

1. Read a data partition
Data Parallelism

1. Read a data partition
2. Pull the parameters
Data Parallelism

1. Read a data partition
2. Pull the parameters
3. Compute the gradient

key-value store
Data Parallelism

1. Read a data partition
2. Pull the parameters
3. Compute the gradient
4. Push the gradient
Data Parallelism

1. Read a data partition
2. Pull the parameters
3. Compute the gradient
4. Push the gradient
5. Update the weight
% create executor for each GPU
execs = [symbol.bind(mx.gpu(i)) for i in range(ngpu)]
% w -= learning_rate * grad
kvstore.set_updater(…)
% iterating on data
for dbatch in train_iter:
    % iterating on GPUs
    for i in range(ngpu):
        % read a data partition
        copy_data_slice(dbatch, execs[i])
        % pull the parameters
        for key in update_keys:
            kvstore.pull(key, execs[i].weight_array[key])
        % compute the gradient
        execs[i].forward(is_train=True)
        execs[i].backward()
        % push the gradient
        for key in update_keys:
            kvstore.push(key, execs[i].grad_array[key])
% create executor for each GPU
execs = [symbol.bind(mx.gpu(i)) for i in range(ngpu)]
% w -= learning_rate * grad
kvstore.set_updater(...)
% iterating on data
for dbatch in train_iter:
    % iterating on GPUs
    for i in range(ngpu):
        % read a data partition
        copy_data_slice(dbatch, execs[i])
        % pull the parameters
        for key in update_keys:
            kvstore.pull(key, execs[i].weight_array[key])
        % compute the gradient
        execs[i].forward(is_train=True)
        execs[i].backward()
    % push the gradient
    for key in update_keys:
        kvstore.push(key, execs[i].grad_array[key])
Results

- **IMAGENET** with 1.2m images and 1,000 classes
- 4 x Nvidia GTX 980
- Google Inception Network
Results

- **IMAGENET** with 1.2m images and 1,000 classes
- 4 x Nvidia GTX 980
- Google Inception Network

Time for one epoch:

![Graph showing time for one epoch compared to the number of GPUs.](image-url)
Distributed Computing

key-value store

examples
Distributed Computing

key-value store

Store data in a distributed filesystem
Distributed Computing

key-value store

multiple worker machines

Store data in a distributed filesystem
Distributed Computing

- multiple server machines
- multiple worker machines
- Store data in a distributed filesystem
Distributed Computing

Multiple server machines

Multiple worker machines

Read over network

Store data in a distributed filesystem

Examples
Distributed Computing

- multiple server machines
- push and pull over network
- multiple worker machines
- read over network
- Store data in a distributed filesystem

examples
Distributed Computing

No code change comparing to single machine

multiple server machines
push and pull over network
multiple worker machines
read over network

Store data in a distributed filesystem
examples
Distributed Experiments

- ImageNet with 1.2m images and 1,000 classes
- AWS EC2 GPU instance, 4 GPUs per machine
- Google Inception Network
Distributed Experiments

- ImageNet with 1.2m images and 1,000 classes
- AWS EC2 GPU instance, 4 GPUs per machine
- Google Inception Network

validation accuracy versus epoch

- Single machine
- 10 machines
Distributed Experiments

- ImageNet with 1.2m images and 1,000 classes
- AWS EC2 GPU instance, 4 GPUs per machine
- Google Inception Network

![Graph showing validation accuracy versus epoch]

- Single machine converges faster
- Multiple machines converge faster
Distributed Experiments

- ImageNet with 1.2m images and 1,000 classes
- AWS EC2 GPU instance, 4 GPUs per machine
- Google Inception Network

validation accuracy versus epoch

- single machine
- 10 machines

Time for one epoch

- 9.8x faster

Accuracy vs. epoch

- multiple machines converge faster
- single machine converges faster
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**
  - Memory Optimization

- **Portability**
  - Runs Everywhere
Multiple Languages
Multiple Languages

frontend

backend
Multiple Languages

- C++
- Python
- Scala
- R
- Julia
- JavaScript
- Go

single implementation of backend system and common operators

performance guarantee regardless which frontend language is used
Minpy: MXNet Numpy Package

NumPy is the de facto scientific computing package in Python. Great flexibility (500+ operators) but CPU-only.
Minpy: MXNet Numpy Package

NumPy is the de facto scientific computing package in Python
Great flexibility (500+ operators) but CPU-only

✧ Native Numpy Integration

```python
>>> import numpy as np  ➭  >>> import minpy as np
```
Minpy: MXNet Numpy Package

NumPy is the de facto scientific computing package in Python.
Great flexibility (500+ operators) but CPU-only

✧ Native Numpy Integration

```python
>>> import numpy as np  # call GPU function
>>> y = np.sort(x)      # call CPU function; copy GPU->CPU
>>> z = np.log(y)       # call GPU function; copy CPU->GPU
```

✧ Transparent CPU and GPU co-execution
Minpy: MXNet Numpy Package

- Small operators (Numpy) + Big operators (MXNet)

```python
>>> symbol = mx.symbol.FullyConnected(…)
>>> bigop = minpy.core.function(sigmoid, …)
>>> def training_loss(w, x, y):
...    pred = bigop(input=x, fc_weight=w)
...    prob = pred * y + (1 - pred) * (1 - y)
...    return -np.sum(np.log(prob))
```
Minpy: MXNet Numpy Package

- Small operators (Numpy) + Big operators (MXNet)

```python
>>> symbol = mx.symbol.FullyConnected(…)
>>> bigop = minpy.core.function(sigmoid, …)
>>> def training_loss(w, x, y):
...     pred = bigop(input=x, fc_weight=w)
...     prob = pred * y + (1 - pred) * (1 - y)
...     return -np.sum(np.log(prob))
```

- Imperative style auto-differentiation

```python
>>> grad_func = minpy.core.grad_and_loss(train_loss)
>>> dw = grad_fn(w, x, y)
```
Bring Torch to MXNet

Torch is a popular Lua framework for both scientific computing and deep learning.
**Bring Torch to MXNet**

Torch is a popular Lua framework for both scientific computing and deep learning.

**Tensor Computation**

```python
>>> import mxnet as mx
>>> x = mx.th.randn(2, 2, ctx=mx.gpu(0))
>>> y = mx.th.abs(x)
>>> print y.asnumpy()
```

**Modules (Layers)**

```python
>>> import mxnet as mx
>>> data = mx.symbol.Variable('data')
>>> fc = mx.symbol.TorchModule(data_0=data,
... lua_string='nn.Linear(784, 128)',
...)
>>> mlp = mx.symbol.TorchModule(data_0=fc,
... lua_string='nn.LogSoftMax()',
...)
```
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**
  - Memory Optimization

- **Portability**
  - Runs Everywhere
Memory Optimization

Traverse the computation graph to reduce the memory footprint with linear time complexity

aliveness analysis

shared space between variables

now a is deletable

share a and b
Results for Deep CNNs

IMAGENET winner neural networks

Training

<table>
<thead>
<tr>
<th>Network</th>
<th>baseline</th>
<th>mxnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>alexnet</td>
<td>1.8x</td>
<td></td>
</tr>
<tr>
<td>inception</td>
<td>2.6x</td>
<td>1.8x</td>
</tr>
<tr>
<td>vgg</td>
<td></td>
<td>6.75x</td>
</tr>
</tbody>
</table>

Prediction

<table>
<thead>
<tr>
<th>Network</th>
<th>baseline</th>
<th>mxnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>alexnet</td>
<td>3.2x</td>
<td></td>
</tr>
<tr>
<td>inception</td>
<td>4.4x</td>
<td>4x</td>
</tr>
<tr>
<td>vgg</td>
<td></td>
<td>4x</td>
</tr>
</tbody>
</table>
Neural Art
Neural Art

1M pixels
GTX 980 Ti 6G
in 20x speed
MXNet Highlights

- **Flexibility**
  - Mixed Programming API
  - Auto Parallel Scheduling
  - Distributed Computing
  - Language Supports

- **Efficiency**
  - Memory Optimization

- **Portability**
  - Runs Everywhere
Train on the Cloud

Consume data from distributed filesystems

- HDFS
- S3
- Blob

multithreaded read/write to hide network latency
Train on the Cloud

Consume data from distributed filesystems
- HDFS
- S3
- Blob
- ...

Launch distributed jobs
- SSH
- MPI
- qsub
- Yarn
- ...

multithreaded read/write to hide network latency
easily extend to other cluster resource management software
Deploy Everywhere

Beyond 🐧 🍎 🌐
Deploy Everywhere

Amalgamation

✧ Fit the core library with all dependencies into a single C++ source file
✧ Easy to compile on 📱 📱 ...
Deploy Everywhere

Beyond

Amalgamation

✦ Fit the core library with all dependencies into a single C++ source file
✦ Easy to compile on Android, Apple, ...
Deploy Everywhere

Beyond

Amalgamation

- Fit the core library with all dependencies into a single C++ source file
- Easy to compile on Android, Apple, ...

Runs in browser with Javascript

BlindTool by Joseph Paul Cohen, demo on Nexus 4
Deploy Everywhere

**Beyond**

**Amalgamation**

- Fit the core library with all dependencies into a single C++ source file
- Easy to compile on Android, Apple, ...  

Runs in browser with Javascript

The first image for search “dog” at images.google.com

BlindTool by Joseph Paul Cohen, demo on Nexus 4
Deploy Everywhere

Amalgamation

✦ Fit the core library with all dependencies into a single C++ source file
✦ Easy to compile on Android, iOS, Windows ...

Beyond

Runs in browser with Javascript

The first image for search “dog” at images.google.com

Outputs “beagle” with prob = 73% within 1 sec

BlindTool by Joseph Paul Cohen, demo on Nexus 4
TX1 on Flying Drone

TX1 with customized board

Drone
TX1 on Flying Drone

TX1 with customized board

Realtime detection and tracking on TX1
~10 frame/sec with 640x480 resolution
Conclusion

Flexibility
- Mixed Programming API
- Auto Parallel Scheduling
- Distributed Computing
- Language Supports

Efficiency

Portability
- Memory Optimization
- Runs Everywhere
Acknowledgement

MXNet is developed by over 100 collaborators

Major Developers

- Bing Xu
  - Dato
- Eric Xie
  - U Washington
- Chiyuan Zhang
  - MIT
- Minjie Wang
  - NYU
- Naiyan Wang
  - TuSimple
- Yizhi Liu
  - MediaV
- Tianjun Xiao
  - Microsoft
- Yuan Tang
  - Uptake
- Qian Kou
  - Indiana University
- Min Lin
  - Qihoo360
- Yutian Li
  - Stanford
- Hu Shiwen
  - Shanghai
- Tong He
  - Simon Fraser University
- Chutao Hong
  - Microsoft

Advisors

- Zheng Zhang
  - NYU Shanghai
- Alex Smola
  - CMU
- Carlos Guestrin
  - U Washington

Hardware and software supports

- NVIDIA
Go mxnet.dmlc.ml to Get Started