Introduction to MXNet and Keras

Dheeraj Mummareddy
Schedule

- Foundational concepts for understanding MXNet
- Introduction to MXNet
- Building blocks of MXNet
- Advantages of MXNet
- Introduction to keras
Imperative vs Declarative programming

- Imperative programming is a programming paradigm that uses statements that change a program state
  
  **Example:**
  
  ```python
  arr = []
  for i in range(20):
      if i < 5:
          small_nums.append(i)
  ```

  **Advantages:** More flexibility for the programmer
  
  **Disadvantages:** Need to implement everything from scratch and chance of errors
Declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow.

**Example:** SQL statement

```
SELECT score FROM games WHERE id < 100;
```

**Advantages:** Hide the complexity and lets the programmer concentrate on logic.

**Disadvantages:** Less flexibility for the programmer.
Execution of computation

- Execution can be of two types Concrete or Asynchronous/Delayed
- In Concrete, results are executed right away
- In Asynchronous, statements are first gathered and transformed into a dataflow graph as an intermediate representation before executing them
- In Asynchronous approach, there is more room for optimization by identifying all possible parallelism within the scope of a graph
MXNet

- Open Source project on Github
- **Founded by**: U.Washington, Carnegie Mellon U.
- **State of the Art Model Support**: Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM)
- **Scalable**: Near-linear scaling equals fastest time to model
- **Multi-language**: Support for Scala, Python, R etc.
- **Portable**: Runs on CPUs, GPUs, servers, desktop, or mobile phones
- **Distributed on Cloud**: Supports training on multiple CPU/GPU machines, including AWS, GCE and Azure
# MXNet features

<table>
<thead>
<tr>
<th>System</th>
<th>Core Lang</th>
<th>Binding Langs</th>
<th>Devices (beyond CPU)</th>
<th>Distributed</th>
<th>Imperative Program</th>
<th>Declarative Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe [7]</td>
<td>C++</td>
<td>Python/Matlab</td>
<td>GPU</td>
<td>×</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>Torch7 [3]</td>
<td>Lua</td>
<td>-</td>
<td>GPU/FPGA</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Theano [1]</td>
<td>Python</td>
<td>-</td>
<td>GPU</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>MXNet</td>
<td>C++</td>
<td>Python/R/Julia/Go</td>
<td>GPU/Mobile</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Source: https://arxiv.org/abs/1512.01274
Core Components of MXNet

1. Symbol
2. NDArray
3. KVStore
Symbols are compositied by operators, such as simple matrix operations (e.g. “+”), or a complex neural network layer (e.g. convolution layer).

An operator can take several input variables, produce more than one output variables, and have internal state variables.

```
using MXNet
mlp = @mx.chain mx.Variable(:data) =>
    mx.FullyConnected(num_hidden=64) =>
    mx.Activation(act_type=:relu) =>
    mx.FullyConnected(num_hidden=10) =>
    mx.Softmax()
```
NDArray

- The NDArray is the core data structure for all mathematical computations
- It is a multidimensional fix-sized homogeneous array
- It enables imperative computations
- Fill the gap between the declarative symbolic expression and the host language

```python
>>> import mxnet as mx
>>> a = mx.nd.ones((2, 3), ... mx.gpu())
>>> print (a * 2).asnumpy()
[[ 2.  2.  2.]
 [ 2.  2.  2.]]
```
KVStore

- KVStore is a distributed key value store for data synchronization over multiple devices.
- It supports two primitives: push a key-value pair from a device to the store and pull the value on a key from the store.
- The following example implements the distributed gradient descent by data parallelization
  ```java
  while(1){ kv.pull(net.w); net.foward_backward(); kv.push(net.g); }
  ```
- Each worker repeatedly pulls the newest weight from the store and then pushes out the locally computed gradient.
Parallel execution example

- Once, the graph is generated A, B and C, D can be executed parallelly.
- Here A, B, C, D and E are NDArray
- * and + are the Symbols
- Use KVStore to get the values from different machines
### Advantages of MXNet

<table>
<thead>
<tr>
<th>Languages</th>
<th>Tutorials and training materials</th>
<th>CNN modeling capability</th>
<th>RNN modeling capability</th>
<th>Architecture: easy-to-use and modular front end</th>
<th>Speed</th>
<th>Multiple GPU support</th>
<th>Keras compatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theano</td>
<td>Python, C++</td>
<td>+++</td>
<td>++</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Tensor-Flow</td>
<td>Python</td>
<td>+++</td>
<td>+++</td>
<td>+++</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Torch</td>
<td>Lua, Python (new)</td>
<td>+</td>
<td>+++</td>
<td>++</td>
<td>+++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Caffe</td>
<td>C++</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>MXNet</td>
<td>R, Python, Julia, Scala</td>
<td>++</td>
<td>++</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Neon</td>
<td>Python</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>CNTK</td>
<td>C++</td>
<td>+</td>
<td>+</td>
<td>+++</td>
<td>+</td>
<td>++</td>
<td>+</td>
</tr>
</tbody>
</table>
Introduction to Keras

- Keras is the official high-level API of TensorFlow
- Used Keras follows best practices for reducing cognitive load
- It minimizes the number of user actions required for common use cases
Features of Keras

- A focus on user experience
- Large adoption in the industry and research community
- Multi-backend, multi-platform
- Develop in R, Python on Unix, Windows, OSX. Run the same code with TensorFlow, MXNet, CNTK etc.
- Easy productization of models
API styles

1. Sequential Model
   1. Good for 70+% of use cases
   2. Only for single-input, single-output, sequential layer stacks

2. The functional API
   1. Good for 95% of use cases
   2. Multi-input, multi-output, arbitrary static graph topologies

3. Model subclassing
   1. Maximum flexibility
The three ways of using Keras API

1. Sequential API
   - CONV → BATCH NORM → RELU

2. Functional API
   - INPUT
   - CONV (size=3x3, stride=2x2)
   - POOL (size=3x3, stride=2x2)
   - CONCATENATE

3. Model Subclassing
   - `tensorflow.keras.Model`
     ```python
class MySimpleNN(Model):
    ...
```

Source: https://pyimagesearch.com/wp-content/uploads/2019/10/keras_3_model_types_header.png
Questions?