Open Neural Network Exchange (ONNX) and OpenAI

Shulei Xu
The Ohio State University
E-mail: xu.2452@osu.edu
Overview

• Open Neural Network Exchange (ONNX)
  – Why need ONNX
  – ONNX intermediate representation spec
  – ONNX runtime
  – Performance evaluation and code examples

• OpenAI
  – Recap of RL
  – Introduction to OpenAI company
  – OpenAI Gym Beta
  – Solving Rubik’s Cube with a Robot Hand
https://onnx.ai/
Why Need ONNX

- Model transfer is important, but less common
- Difficult to optimize the tools for all cases
- Separate but interoperable tools is more efficient
Goal

• Provide a standard way to represent models so that:
  
  – Serialized models are interoperable between frameworks
  
  – Have a common target for optimization for different backends
Key Design Principles

• Support DNN but also allow for traditional ML

• Flexible enough to keep up with rapid advances

• Compact and cross-platform representation for serialization

• Standardized list of well-defined operators informed by real world usage
Deep Learning Framework Zoo

Framework backends

Vendor and numeric libraries

- Apple CoreML
- Nvidia TensorRT
- Intel/Nervana ngraph
- Qualcomm SNPE
- ...
Open Neural Network Exchange

Shared model and operator representation

From $O(n^2)$ to $O(n)$ pairs

Vendor and numeric libraries

Apple CoreML, Nvidia TensorRT, Intel/Nervana ngraph, Qualcomm SNPE, ...
How Standards Proliferate:

See: A/C chargers, character encodings, instant messaging, etc.

Situation: There are 14 competing standards.

14?! Ridiculous! We need to develop one universal standard that covers everyone’s use cases. Yeah!

Soon:

Situation: There are 15 competing standards.
Needs of Open Community

- Framework agnostic
- GitHub from the beginning
- Close partnerships and OSS contributions
ONNX Model Zoo

Collection of Popular Models

- Continuous testing against frameworks and backends
- Target for optimizations
- Useful as Pre-Trained Model Collection

<table>
<thead>
<tr>
<th>Model Class</th>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet</td>
<td>Sandler et al.</td>
<td>Light-weight deep neural network best suited for mc applications. Top-5 error from paper ~ 10%</td>
</tr>
<tr>
<td>ResNet</td>
<td>He et al.</td>
<td>A design for optimizing Deep Neural Network on Mobile.</td>
</tr>
<tr>
<td>VGG</td>
<td>Simonyan et al.</td>
<td>A real-time CNN for object detection that detects 2 more complex YOLOv2 network.</td>
</tr>
<tr>
<td>Tiny YOLOv2</td>
<td>Redmon et al.</td>
<td>A real-time CNN for object detection that detects 2 more complex YOLOv2 network.</td>
</tr>
<tr>
<td>SSD</td>
<td>Liu et al.</td>
<td>Single Stage Detector: real-time CNN for object detection.</td>
</tr>
<tr>
<td>Faster-RCNN</td>
<td>Ren et al.</td>
<td>Increases efficiency from R-CNN by connecting a R network for object detection that detects 80 different objects.</td>
</tr>
<tr>
<td>Mask-RCNN</td>
<td>He et al.</td>
<td>A real-time neural network for object instance segmentation (R-CNN).</td>
</tr>
<tr>
<td>Speech recognition with deep recurrent neural networks</td>
<td>Graves et al.</td>
<td>A RNN model for sequence problems where the input contribute</td>
</tr>
<tr>
<td>Deep voice: Real time neural text to speech</td>
<td>Arik et al.</td>
<td>A DNN model that perform Requires fewer parameters contribute</td>
</tr>
<tr>
<td>Sound Generative models</td>
<td>WaveNet: A Generative Model for Raw Audio</td>
<td>A CNN model that generat distribution for each audio fragments, contribute</td>
</tr>
</tbody>
</table>
Famework Anatomy

Frontend (dev experience)

Backend (HW platform)

- Modelling abstractions
- Data
- Distributed engine

- High level IR / Operators

- ONNX

- Framework glue code

- ATen

- Low level IR
- Kernel compiler
  - TVM, TC, XLA

- Device runtime
  - x86, CUDA, OpenCL, ...

- Graph-level engines
  - TensorRT, CoreML, SNPE

- BLAS
  - MKL, cuBLAS, ...

- NN libraries
  - CUDNN, MPSCNN, ...
Frontend

- ONNX is the core model representation format graph captured from imperative frontend or built directly

```python
def foo(x):
    y = x.mm(x)
    print(y)  # still works!
    return y + 1

x = torch.Tensor([[1,2],[3,4]])
foo(x)

onnx.export(foo, x, "my.onnx")
```
Backend

- ONNX as an **Interface** between framework and hardware libraries
ONNX high-level IR

- Initial focus on exchange for inference
- SSA graph structure, serializable
  - Support for structured control flow
- Standard operator definitions
  - Striking balance on granularity
  - Codified semantics in tests/ref
- Common optimization passes
ONNX is an open specification that consists of the following components:

- A definition of an extensible computation graph model
- Definitions of standard data types
- Definitions of built-in operators

Two variants:
- ONNX
- ONNX-ML
ONNX Model

- High-Level IR
- Operator Specs
- Weights Serialization
Model File format

- **Model**
  - Version info
  - Metadata
  - **Acyclic computation dataflow graph**

- **Graph**
  - Inputs and outputs
  - List of computation nodes
  - Graph name

- **Computation Node**
  - Zero or more inputs of defined types
  - One or more outputs of defined types
  - Operator
  - Operator parameters
Data types

- **Tensor type**
  - Element types supported:
    - int8, int16, int32, int64
    - uint8, uint16, uint32, uint64
    - float16, float, double
    - bool
    - string
    - complex64, complex128

- **Non-tensor types**
  - Dense tensors
  - Sequence
  - Map
Operators

An operator is identified by <name, domain, version>

• Core ops (ONNX and ONNX-ML)
  – Should be supported by ONNX-compatible products
  – Generally cannot be meaningfully further decomposed
  – Currently 124 ops in ai.onnx domain and 18 in ai.onnx.ml
  – Supports many scenarios/problem areas including image classification, recommendation, NLP, etc

• Custom ops
  – Ops specific to framework or runtime
  – Indicated by a custom domain name
  – Primarily meant to be a safety-valve
Functions

- Compound ops built with existing primitive ops
- Runtimes/frameworks/tools can either have an optimized implementation or fallback to using the primitive ops
Current Status

• **ONNX IR spec is V1.6**
• Good coverage for *vision* models
• Beta for NLP
• Iterating on:
  – Optimization friendly RNNs
  – Control Flow
  – More hardware backends
ONNX Runtime

- High performance
- Cross platform
- Lightweight & modular
- Extensible

https://github.com/onnx
ONNX Runtime

High performance runtime for ONNX models

• Flexible
  – Supports full ONNX-ML spec (v1.2 and higher, currently up to 1.6)
  – Python, C#, and C APIs

• Cross Platform
  – Works on Mac, Windows, Linux
  – Support x86, x64, ARM

• Extensible architecture to plug-in optimizers and hardware accelerators

• CPU and GPU support
ONNX Runtime – high level architecture
ONNX Runtime – Architecture

• **Graph Optimization**
  – Node elimination (dropout, identity, etc)
  – Node fusion, constant folding, etc.

• **Graph Partitioning**
  – Graph partitioning based on execution providers’ capability
  – Greedy algorithm on user preference

• **Execution Provider**
  – Plug-in hardware accelerators
  – APIs
More about Partitioning

• ONNX Runtime partitions a model graph into subgraphs
• Simple, greedy partitioning techniques
• Each partition is reduced to a single fused operator
• Currently only support synchronous mode for execution
Key Objectives of Such Design

- Maximally and automatically leverage the custom accelerators and runtimes available on disparate platforms
- Provide the right abstraction and runtime support for custom accelerators and runtimes – execution providers
- Provide support to execute a single model in a heterogeneous environment involving multiple execution providers
- Provide support for high-level optimizations that can be expressed as model-to-model transformations via a graph-transformation API
Expanding ONNX – Spec Contributions

• Control Flow
• Quantization
• Framework APIs
Control Flow

• Loop and if, optimized for neural networks patterns
Quantization and motivation

• Quantization refers to set of techniques for executing deep learning models with a reduced precision (e.g. int8)

• Motivations and benefits: Theoretically delivers
  – 4x smaller model size for Int8 (converting from FP32)
  – 4x lower memory bandwidth
  – Higher TOPS/W
  – Minimal accuracy loss
Int8 Inference on Skylake (SKX) processor

Typical AVX-512 instruction to perform FP32 convolutions: \texttt{vfmadd231ps}

Typical AVX-512 instructions to perform INT8 convolutions: \texttt{vpmaddubsw, vpmaddwd, vpaddd}
VNNI (8 bit inference) on CLX processor

- VNNI enables 8-bit multiplies with 32-bit accumulates using 1 instruction
- VPMADDUBSW, VPMADDWD, VPADD instructions are fused into the VPDBUSD instruction
- U8 * S8 = S32
- Allows for 4x more inputs relative to FP32, 4x more compute with 1/4th the memory requirements
Resnet50 (SKX FP32 vs CLX INT8 VNNI) ONNX

- **Key points & Data:** 1 socket config: (using nGraph EP & MKL-DNN kernels)
- VNNI showing 3.44x perf gains over SKX 317 images/s with BS=32, 1094 images/s (CLX)
- VNNI showing 1.78ms latency with BS=1 (3.11x over SKX 180 images/s), 561 images/s (CLX)
- With close to same accuracy (accuracy loss 0.41 for Top 1 result & 0.21 loss for Top 5)
InceptionV3 (SKX FP32 vs CLX INT8 VNII) ONNX

- **Key points & Data**: 1 socket config: (using nGraph EP & MKL-DNN kernels)
- VNII showing 3.67x perf gains over SKX 209 images/s with BS=32, 769 images/s (CLX)
- VNII showing 3.24ms latency with BS=1 (2.28x over SKX 135 images/s), 308 images/s (CLX)
- With close to same accuracy (accuracy loss 0.30 for Top 1 result & 0.08 loss for Top 5)
ONNX Runtime + TensorRT

- Now released as preview!
- Run any ONNX ML model
- Same cross platform API for CPU, GPU, etc.
- ONNX Runtime partitions the graph and uses TensorRT where support is available
NVIDIA TensorRT – Platform for High-Performance Deep Learning Inference

Optimize and deploy neural networks in production environments

- Maximize throughput for latency-critical apps with optimizer and runtime
- Optimize your network with layer and tensor fusions, dynamic tensor memory and kernel auto tuning
- Deploy responsive and memory efficient apps with INT8 & FP16 optimizations
- **Fully integrated as a backend in ONNX runtime**

[developer.nvidia.com/tensorrt](http://developer.nvidia.com/tensorrt)
ONNX – TensorRT Parser

Available at https://github.com/onnx/onnx-tensorrt
TensorRT Execution Provider in ONNX Runtime
Demo Performance Comparison

ONNXRUNTIME-CPU

Model prediction:  surprise
Inference time:  61.03 ms
Model Input image:

ONNXRUNTIME-GPU
(using CUDA)

Model prediction:  surprise
Inference time:  3.63 ms
Model Input image:

ONNXRUNTIME-TensorRT

Model prediction:  surprise
Inference time:  2.47 ms
Model Input image:

Model: Facial Expression Recognition (FER+) model from ONNX model zoo
Hardware: Azure VM – NC12 (K80 NVIDIA GPU)
CUDA 10.0, TensorRT 5.0.2
ONNX Runtime + TensorRT

- Best of both worlds
- Run any ONNX-ML model
- Easy to use API across platforms and accelerators
- Leverage TensorRT acceleration where beneficial
How to get an ONNX model

• ONNX Model Zoo

• Converting existing Models

• Services like Azure Custom Service

• Train models in systems like Azure Machine Learning service
Convert Models

https://github.com/onnx/tutorials
Examples of Converting Models: Keras & Pytorch

```python
from keras.models import load_model
import keras2onnx
import onnx

keras_model = load_model("model.h5")

onnx_model = keras2onnx.convert_keras(keras_model, keras_model.name)
onnx.save_model(onnx_model, 'model.onnx')
```

```python
import torch
import torch.onnx

model = torch.load("model.pt")
sample_input = torch.randn(1, 3, 224, 224)
torch.onnx.export(model, sample_input, "model.onnx")
```
Examples of Using the ONNX runtime

```python
import onnxruntime

session = onnxruntime.InferenceSession("mymodel.onnx")

results = session.run([], {"input": input_data})
```

```csharp
using Microsoft.ML.OnnxRuntime;

var session = new InferenceSession("model.onnx");

var results = session.Run(input);
```
Questions for ONNX?
Overview

• Open Neural Network Exchange (ONNX)

• OpenAI
  – Recap of RL
  – Introduction to OpenAI company
  – OpenAI Gym
  – Solving Rubik’s Cube with a Robot Hand
Machine Learning Paradigms

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
Supervised Learning

• **Input Data:**
  – (x, y), x is data, y is label

• **Goal**
  – Return a function to map x -> y

• **Examples**
  – Classification, regression, object detection, semantic segmentation, image captioning, etc.
Unsupervised Learning

• **Input Data:**
  – $x$, only data, no labels!

• **Goal**
  – Learn some underlying hidden structure of the data

• **Examples**
  – Clustering, dimensionality reduction, feature learning, density estimation, etc.
Reinforcement Learning

• Reinforcement learning (RL) is the subfield of machine learning concerned with decision making and motor control

• Problem involving an agent, interacting with environment, which provides numeric reward signals

• Goal:
  – Learn how to take actions in order to maximize rewards
Cart-Pole Problem

- **Objective**: Balance a pole on top of a movable cart
- **State**: angle, angular speed, position, horizontal velocity
- **Action**: horizontal force applied on the cart
- **Reward**: 1 at each time step if the pole is upright
Robot Locomotion

- **Objective**: Make the robot move forward
- **State**: Angle and position of the joints
- **Action**: Torques applied on joints
- **Reward**: 1 at each time step upright + forward movement
Atari Games

• **Objective**: Complete the game with the highest score
• **State**: Raw pixel inputs of the game state
• **Action**: Game controls e.g. Left, Right, Up, Down
• **Reward**: Score increase/decrease at each time step
Reinforcement Learning

AlphaGO Zero

DOTA 2 (OPENAI FIVE)
Founders: Elon Musk, Sam Altman
Key person: Ilya Sutskever (University of Toronto -> Stanford University -> DNNResearch (Google Brain) -> OpenAI, co-inventor of AlexNet, AlphaGo and TensorFlow)

https://openai.com
OpenAI

OpenAI is a non-profit artificial intelligence research company, its mission is to ensure that artificial general intelligence (AGI) benefits all of humanity. To that end, they commit to the following principles:

• Broadly Distributed Benefits
• Long-Term Safety
• Technical Leadership
• Cooperative Orientation

OpenAI’s research director is Ilya Sutskever, one of the world experts in machine learning. CTO is Greg Brockman, formerly the CTO of Stripe.
OpenAI Gym Beta

- A toolkit for developing and comparing reinforcement learning (RL) algorithms
- It makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as TensorFlow or Theano
- Collection of test problems – environments
  - Work out your reinforcement learning algorithms
  - Have a shared interface, allowing you to write general algorithms
Background: Why Open AI Gym?

RL studies how an agent can learn how to achieve goals in a complex, uncertain environment

- **It’s exciting for two reasons:**
  - RL is very general, encompassing all problems that involve making a sequence of decisions
  - RL algorithms have started to achieve good results in many difficult environments

- **It’s slowed down by two factors:**
  - The need for better benchmarks
  - Lack of standardization of environments used in publications

- **Gym is an attempt to fix both problems**
Environments

- Example of using built-in CartPole-v0 environment

```python
import gym
env = gym.make('CartPole-v0')
env.reset()
for _ in range(1000):
    env.render()
    env.step(env.action_space.sample()) # take a random action
env.close()
```

- Full list of environments:

  http://gym.openai.com/envs
Observation

The environment’s step function returns exactly what we need. In fact, step returns four values. These are:

- observation (object)
- Reward (float)
- done (boolean)
- info (dict)

```python
import gym
env = gym.make('CartPole-v0')
for i_episode in range(20):
    observation = env.reset()
    for t in range(100):
        env.render()
        print(observation)
        action = env.action_space.sample()
        observation, reward, done, info = env.step(action)
        if done:
            print("Episode finished after {} timesteps".format(t+1))
            break
env.close()
```
Solving Rubik’s Cube with a Robot Hand

- Challenging even for humans
- Solves the Rubik’s Cube 60% of the time (not perfect yet)

https://openai.com/blog/solving-rubiks-cube/
Approaches

Train neural networks to solve the Rubik’s Cube in simulation using:

• Reinforcement learning
• Kociemba’s algorithm
• Domain randomization
Domain Randomization

- Source domain (where training happens)
- Target domain (i.e. physical world)

\[ \theta^* = \arg \max_{\theta} \mathbb{E}_{\xi \sim \mathcal{E}} \left[ \mathbb{E}_{\pi_\theta, \tau \sim e_\xi}[R(\tau)] \right] \]
Automatic Domain Randomization

- Endlessly generates progressively more difficult environments in simulation
  1. Starts with a single, nonrandomized environment
  2. Train network until it reaches a performance threshold
  3. Increase domain randomization automatically
  4. Back to repeat step 2
  5. ......

- Randomized parameters include:
  - Size of Rubik’s cube, mass of the cube, friction of the robot fingers and visual surface materials of the hand.
Performance Comparison – Manual DR vs. ADR

- ADR loses at the beginning
- ADR wins when entropy gets larger
Questions for OpenAI?