ONNX

(Open Neural Network Exchange)

December 15th, 2020

Prepared by:
Sanjana Balasubramanian, Vinayak Jethe, Julian Jaramillo, Jaabaal Shah

Version History

Name	Date	Reason For Changes	Version
Pre - release	Sep 07 2017	This is the pilot pre - release version so there are no reasons for changes.	v0.1
Pre - release	Oct 09 2017	The protobuf structures were not backwards compatible.	v0.2
Stable release	Dec 06 2017	The previous pre-releases were only for DNN models and there was no proper documentation and samples.	v1.0
Patch release	Jan 26 2018	There were a number of bugs that needed to be addressed.	v1.0.1
Stable release	Mar 20 2018	The number of models for DNN and ML increased significantly so there was a need to include additional support for these models and operators.	v1.1.0
Patch release	Apr 25 2018	There were a number of bugs that needed to be addressed.	v1.1.2
Stable release	May 24 2018	There was a need to add new operators and make changes to the IR.	v1.2.1
Patch release	Jun 20 2018	There were a number of bugs that needed to be addressed.	v1.2.2
Stable release	Aug 30 2018	There was a need for a new operator set and enhanced optimization passes and ONNXIFI 1.0 was released for loading and executing ONNX graphs.	v1.3.0
Patch release	Sep 10 2018	The build configurations were updated and hence the previous version was bumped up.	v1.2.3
Stable release	Jan 22 2019	The ONNX project got a major boost and was now used as a de facto standard among 25+ companies and the size of models supported was increased significantly.	v1.4.0
Patch release	Jan 23 2019	There was a line ending issue on Linux environments that needed to be fixed.	v1.4.1
Stable release	Apr 23 2019	There was a need to include the new models to the Model Zoo and Quantization support was added and ONNX Function was promoted as an official feature	v1.5.0
Stable release	Sep 27 2019	The traditional ML data types needed to be supported to be converted to the ONNX graph structure and also the IR and operators needed updating.	v1.6.0

Stable release	May 8 2020	There was a need for proper training support for easy usage among the engineers and updating was needed for operators. In addition to that, several bugs needed to be addressed.	v1.7.0
Stable release	Nov 6 2020	There were a number of bugs that needed to be addressed.	v1.8.0

Table of Contents

1. Deliverables

This document is accessible only to the team members and by Professor Gheni Abla through their respective GitHub and Gmail accounts as of Tuesday Dec 15th, 2020 11:59 PM. The deliverables for this project are described below:

Code Repo (if any) (URL): https://github.com/jaabaal/ONNX_Report

Presentation Slides (URL): https://github.com/jaabaal/ONNX_Report

Presentation/Demo Video (URL): https://www.youtube.com/watch?v=_nsnO5im7j8&feature=youtu.be&ab_channel=SanjanaBalasubramanian

2. Executive Summary

ONNX is a project which was started in 2017 by three companies mainly AWS, Facebook and Microsoft with the sole motivation of creating an interoperable ecosystem between the myriad number of frameworks available in the DNN and ML domain. Unlike ML and DNN, ONNX is not a black box system and is very well documented. Throughout the project, we have dived deep into its workings and have explored and answered the most relevant questions about the same which are described below:

- ➔ What exactly is ONNX?
- ➔ Why is it that ONNX gained popularity in such a short period of time?
- ➔ How do you work with ONNX?
- ➔ What is an example of an ONNX workflow?

Since its birth in 2017, ONNX has been continuously evolving and is currently becoming the de facto standard for major AI and ML applications focused on inference Being an open-source project under the LFAI which is the Linux Foundation's charter for Artificial Intelligence, its community has also been growing rapidly and has already gained popularity among the AI, DL and ML communities. Some interesting insights that we gained during the process are as follows:

- We gained better understanding about the entire process of developing a model and deploying it on the specific target environment.
- We were made aware of the numerous hardware and software domain specific tools that are available for visualization, development and deployment.

3. Project Overview

3.1 Problem Statement

The production of the model and of course, its implementation are two important aspects about an Artificial Intelligence Model. The growth in deep learning frameworks renders it a cumbersome task. The technical skill provided by ONNX enables a developer to generate great ideas more quickly.

ONNX is an open-source project which is mainly an intermediate representation in the form of a file format which maps the high-level graphs of a model in specific frameworks to a similar graph in the ONNX file format.

The Problems ONNX tries to solve:

Framework Interoperability: After the invention of ONNX, this can be considered as the key requirement. It provides data scientists/developers with a free hand to train a model in a framework and produce inference in another framework.

Hardware Optimizations: It is easy to offer developers the ability to optimize by using Open Neural Network Exchange. It gives any tool that uses the exported ONNX models the advantages of ONNX compatible runtimes and libraries and results in the maximization of performance on some of the best hardware in the technical field.

3.2 Project Team

Name of the project: ONNX (Open Neural Network eXchange)

Name of the Team member	Responsibility	Contribution %	Notes
Sanjana Balasubramanian	About ONNX	25%	Overview of ONNX and Software Aspects
Vinayak Jethe	Working of ONNX	25%	Machine Learning Aspects and Problem Statement
Julian Jaramillo	Example Workflow	25%	Example Workflow and Use Cases
Jaabaal Shah	Importance of ONNX	25 %	Version History, Executive summary and Importance of ONNX

4. About ONNX

Open Neural Network Exchange (ONNX) is an open ecosystem that enables AI developers to pick the right tools as their work proceeds. For AI models, both deep learning and conventional ML, ONNX offers an open-source format. As well as definitions of built-in operators and regular data types, it defines an extensible computation graph model.

ONNX is commonly supported in many frameworks, tools, and hardware and can be found. Enabling interoperability and streamlining the path from research to production between different frameworks helps to increase the pace of innovation in the AI community.

ONNX can be developed without thinking about downstream inferencing effects in your chosen system. With your chosen inference engine, ONNX allows you to use your preferred system. ONNX makes it easier to access hardware optimizations. Use compatible ONNX runtimes and libraries designed to optimize hardware-wide performance.

As well as definitions of built-in operators and standard data types, ONNX offers a description of an extensible computation graph model. As a list of nodes that form an acyclic graph, each computation dataflow graph is organized. Nodes have one or more outputs and one or more inputs. Each node represents a call to an operator. To help record its function, author, etc. the graph also has metadata.

Operators are applied externally to the graph, but through frameworks, the set of built-in operators are portable. Implementations of these operators on the relevant data types will be supported by any system supporting ONNX.

5. Importance of ONNX

The major challenge to the current deep learning domain is that the trained models cannot be deployed directly as they need to be optimized depending upon the target environment they are going to be deployed on.

"There is no interoperability which exists among the various frameworks."

In the last decade, deep learning has evolved at an exponential rate due to the simultaneous growth in the fields of big data, hardware components and algorithms. The extremely distributed and fragmented nature of this domain requires the engineers to develop models according to specific use cases. For instance, a certain use case may use the NVIDIA GPUs as accelerators for training and would therefore require only the CUDA or the cuDNN platforms to be used in order to interact with them. This would pose a problem if the inference is using a different framework as there was no way to map the operators among these two frameworks. This is where the ONNX file format comes into play.

"The ONNX file format is to DNN what CLR is to programming languages."

5.1 Design Philosophy

- Built initially to support DNN but also capable to provide support for traditional ML.
- Flexible enough to keep up with rapid advances and enhancements.
- Compact and cross-platform representation.
- Allow interoperability/portability by enabling the users to export a fully trained model in a certain framework to be imported to a different framework.

5.2 Fragmentation in the Training Phase

During the training phase there is a requirement for setting up the environment using the necessary hardware and software. Some instances are described below:

Hardware	Software
Nvidia GPU	Nvidia CUDA and Nvidia cuDNN
Intel FPGA	oneAPI
cloud TPU	TensorFlow Toolkit

5.3 Fragmentation in the Deployment Phase

Similar to the training phase, the target environment too is very diverse and is domain specific, hence it is not directly optimizable. Some instances are described below:

Hardware	Domain	Software
Nvidia T4	GPU	Nvidia CUDA and cuDNN
Intel CPU	CPU	Intel OpenVINO
Jetson Nano	Edge Computing	JetsonPack SDK
Intel Movidius Myriad X	VPU	Intel OpenVINO

6. Machine Learning Aspects

ONNX is an open-source file format built to represent machine learning models to allow for portability and interoperability. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.

6.1 Dataset

Developers can obtain their own set of data using online resources/research or also use the ONNX Model Zoo too and directly have a pre-trained set of models in the ONNX format.

6.2 Model Creation and Training

We can obtain ONNX models by the following ways:

- ONNX Model Zoo: This contains several Pre-trained ONNX models for different types of tasks. The developer can download a version that is supported by Windows ML.

- Native export from ML training frameworks: Several training frameworks support native export functionality to ONNX, like Chainer, Caffee2, and PyTorch, allowing you to save your trained model to specific versions of the ONNX format. In addition, services such as Azure Machine Learning and Azure Custom Vision also provide native ONNX export.

- Convert existing models using WinMLTools: This Python package allows models to be converted from several training framework formats to ONNX. As a developer, you can specify which version of ONNX you would like to convert your model to, depending on which builds of Windows your application targets. If the developer is

not familiar with Python, they can use Windows ML's UI-based Dashboard to easily convert the models with just a few clicks.

6.3 Inference

ONNX Runtime is a cross-platform inferencing and training accelerator compatible with many popular Machine Learning or Deep neural networks, including PyTorch, TensorFlow/Keras, scikit-learn, and many more.

Benefits from ONNX Runtime include the following

- Improve inference performance for a wide variety of ML models
- Reduce time and cost of training large models
- Train in Python but deploy into a C#/C++/Java app
- Run on different hardware and operating systems
- Support models created in several different frameworks

6.3.1 Training ONNX Runtime

The ONNX Runtime training feature enables easy integration with existing Pytorch trainer code to accelerate the execution.

With a few lines of code, you can add ONNX Runtime into your existing training scripts and start seeing acceleration. The current preview version supports training acceleration for transformer models on NVIDIA GPUs.

6.3.2 Deploying ONNX Runtime

Cloud

- ONNX Runtime can be deployed to any cloud for model inference.
- ONNX Runtime Server (beta) is a hosting application for serving ONNX models using ONNX Runtime, providing a REST API for prediction.

IoT and edge devices

- The growing emphasis and selection of sensors and reliable signal streams from IoT devices offers new opportunities to transfer AI workloads to the edge. This is especially important because there are large quantities of incoming data/signals that, due to storage or latency considerations, might not be effective or useful to push to the cloud.

- For Example, surveillance tapes where 99% of the footage has no event occurring, and scenarios of real-time individual detection can be effective. In these scenarios, directly executing model inferencing on the target device is crucial for optimal assistance.

Client applications

- Install or build the package you need to use in your application.
- On newer Windows 10 devices (1809+), ONNX Runtime is available by default as part of the OS and is accessible via the Windows Machine Learning APIs.

7. Software

7.1 System as a Software

ONNX is supported by various Architectures/Languages/Providers such as:

- Operating System: Windows, Linux and Mac
- Language: Python (3.5 to 3.7), C++, C#, C
- Architecture: x64, x86, ARM64, ARM32
- Hardware: CPU, CUDA, TensorRT, DirectML, MKL-DNN, MKL-ML, nGraph, NUPHAR, OpenViINO

ONNX System Design Principles

- Supports both DNN and Traditional Machine Learning.
- Interoperable and backward Compatible which are important for production usage.
- Compact and cross-platform representation for serialization.

ONNX specifications consist of the following

- A definition of an extensible computation graph model.
- Definitions of standard data types.
- Definition of built-in operators (which belong to a versioned operator set).

7.2 System Architecture

In ONNX architecture, the nodes architecture is very much similar to a structure of Convolutional Neural Network. The basic components are as follows:

Model

- Version Information - Producer to keep track of which version of the model we are producing at different times.
- Metadata associated with the model
- Acyclic computation dataflow graph which is the important part consisting of inputs, outputs, names of the graphs and the list of computational nodes.

Graph

- Inputs and Outputs of the model
- List of computational nodes
- Graph name for the model

Computation Node

- Zero or more Input of defined types.
- Zero or more Output of defined types.
- Operator.
- Operator parameter.

The models are collections of graphs and graphs are collections of several computational nodes.

Every node here is self-sustained. It actually provides operators. Each operator is mapped to deep learning conventions.

7.3 User Interface Overview

For the UI in ONNX developers can use ONNXIFI which is the Interface for Framework Integration.

ONNXIFI is a cross-platform API for loading and executing ONNX graphs on optimized backends. High-level frameworks and applications can use this API to execute neural network and machine learning models. Hardware vendors can implement this API to expose specialized hardware accelerators and highly optimized software infrastructure to the users.

The main core features

- Standardized interface for neural network inference on special-purpose accelerators (NPUs), CPUs, GPUs, DSPs, and FPGAs
- Based on widely supported technologies
    - C API for function calls
    - ONNX format for passing model graphs
    - NCHW tensor layout for passing inputs and outputs
- Dynamic discovery of available backends for model execution
    - Multiple backends from different vendors can co-exist on the same system
- Dynamic discovery of supported ONNX Operators on each backend.

8. Use Cases and tests

- Interoperability of frameworks for machine learning
- Our examples focused on sklearn to ONNX and keras to ONNX
- ONNX Runtime allows for testing of ONNX models to compare to previous models

8.1 Example 1: Sklearn to ONNX

When working with a Sklearn model it is as simple as importing a few open-source libraries to convert your model to an ONNX model. Using skl2onnx, onnx, and onnxruntime we were able to take the linear regression model created with keras and convert it to an ONNX model without loss in accuracy or precision when predicting scores on tests. Below is the code of the original model creation followed by the conversion to onnx and testing in the onnxruntime.

```
import pandas as pd
#importing the data into url
url =
"https://raw.githubusercontent.com/gheniabla/datasets/master/score.csv"
#using pandas to read the csv file into dataset
datasets = pd.read_csv(url)
X = datasets.iloc[:, :-1].values
Y = datasets.iloc[:, 1].values

# Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 1/3,
random_state = 0)

from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_Train, Y_Train)

# Print the intercept part 1
print("Intercept=", regressor.intercept_)
# Print the slope part 2
print("Slope=", regressor.coef_)
# Predict the score of the data in X_Test will compare with Onnx model
for i in range(9):
  score = regressor.predict([X_Test[i]])
  print("score=", score)
pip install skl2onnx
pip install onnx
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
initial_type = [('float_input', FloatTensorType([None, 1]))]
onnx_model = convert_sklearn(regressor, initial_types=initial_type)
```

```
with open("regression.onnx", "wb") as f:
  f.write(onnx_model.SerializeToString())
import skl2onnx
#convert our sklearn model to onnx model
skl2onnx.to_onnx(regressor, X_Train[:1])
#Going to use the onnxruntime to test our model out
import onnxruntime as rt
import numpy
sess = rt.InferenceSession("regression.onnx")
input_name = sess.get_inputs()[0].name
label_name = sess.get_outputs()[0].name
#show each prediction for the X_Test data
pred_onx = sess.run([label_name], {input_name:
X_Test.astype(numpy.float32)})[0]
```

We can see the results after running test data in X_Test.

```
score= [17.04289179]
score= [33.51695377]
score= [74.21757747]
score= [26.73351648]
score= [59.68164043]
score= [39.33132858]
score= [20.91914167]
score= [78.09382734]
score= [69.37226512]
```

```
array([[17.04289 ],
       [33.516953],
       [74.217575],
       [26.733515],
       [59.681637],
       [39.331326],
       [20.91914 ],
       [78.09383 ],
       [69.37227 ]], dtype=float32)
```

8.2 Example 2: Keras to ONNX

In this example we will be taking a Keras model and converting it to an ONNX model using another open source library keras2onnx and in conjunction with onnx and onnxruntime we can evaluate the new model. In this example the python code for the keras model code is left out for brevity but can be found at the github repo linked in this report. What follows is the code to convert the keras model into an ONNX model.

```
import numpy as np
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input
import keras2onnx
import onnxruntime
#convert our keras model to onnx
onnx_model = keras2onnx.convert_keras(keras_model, keras_model.name)
```

```
# runtime prediction via inference

temp_model_file = 'model.onnx'
keras2onnx.save_model(onnx_model, temp_model_file)
sess = onnxruntime.InferenceSession(temp_model_file)
```

8.3 Working with ONNX conclusion

The above examples only scratch the surface of what ONNX is capable of but show how accessible it can be and how quickly it can be used. There are ONNX projects for most if not all of the major frameworks out there with examples and decent documentation to accompany them. There can be issues that arise when working with different frameworks but there seems to be a solid community willing to help due to its open-source nature.

9. References

- https://www.youtube.com/watch?v=Ij5MoUnLQ0E, *Microsoft Research, Dec 06, 2019*
- https://www.youtube.com/watch?v=cK5AyawZSUI, *Janakiram MSV, Jul 12, 2020*
- https://www.youtube.com/watch?v=Mvnn_Iy29es, *FaceBook Developers, Feb 20, 2019*
- https://github.com/onnx/keras-onnx
- https://github.com/onnx/sklearn-onnx
- http://onnx.ai/sklearn-onnx/api_summary.html#skl2onnx.to_onnx
- https://onnx.ai/get-started.html
- https://github.com/onnx/onnx/blob/master/docs/ONNXIFI.md
- https://github.com/microsoft/onnxruntime/tree/master/samples#CC

10. Glossary

- DNN: Deep Neural Network

  A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.

- IR: Intermediate Representation

  An intermediate representation (IR) is the data structure or code used internally by a compiler or virtual machine to represent source code.

- CNTK: Microsoft Cognitive Toolkit

  The Microsoft Cognitive Toolkit is an open-source toolkit for commercial-grade distributed deep learning.

- CUDA: Compute Unified Device Architecture

CUDA is a parallel computing platform and application programming interface model created by Nvidia.

- OpenVINO: Open Visual Inference and Neural Network Optimization

  It is a toolkit provided by Intel to facilitate faster inference of deep learning models.

- cuDNN: Nvidia CUDA Deep Neural Network library

  The Nvidia cuDNN is a GPU-accelerated library of primitives for deep neural networks.

- Chainer

  Chainer is an open-source deep learning framework written purely in Python on top of NumPy and CuPy Python libraries.

- Caffee2

  CAFFE2 is a deep learning framework, originally developed at University of California, Berkeley.

- PyTorch

  PyTorch is an open-source machine learning library based on the Torch library.

- NCHW

  NCHW means a data whose layout is (batch size, channel, height, width) i.e., N denotes Batch, C denotes channel, H denotes Height and W denotes Weight).

- NUPHAR: Neural-network Unified Preprocessing Heterogeneous Architecture

  It is a TVM and LLVM based Execution Provider offering model acceleration by compiling nodes in subgraphs into optimized functions via JIT.