ONNX: Preventing Framework Lock in

An introduction to the use of the ONNX standard for the interoperability between Deep Learning frameworks.

Fernando López Oct 27, 2020 · 7 min read ★

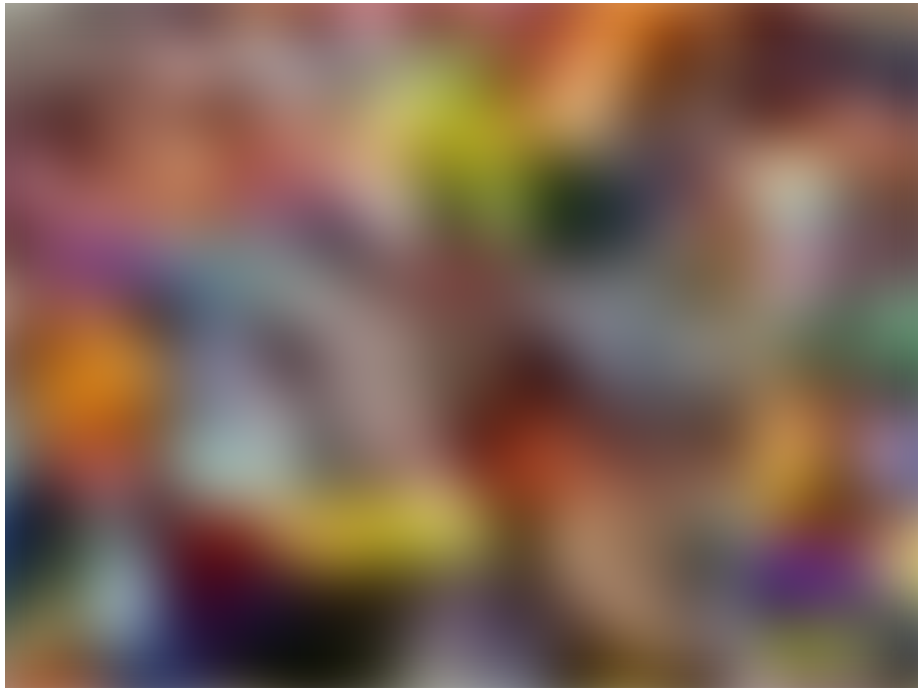

Photo by Rick Mason on Unsplash

In this blog, we are going to see what the **ONNX** standard is, its components and how to carry out interoperability between different *Deep Learning frameworks*. This blog will address the following sections:

- **Introduction**
- **What is ONNX?**
- **What is ONNX Runtime?**
- **Interoperability: from PyTorch to other Frameworks**

So let's get started!

Introduction

PyTorch, *Tensorflow*, *Caffe2*, *MXNet*, etc, are just some of the most popular frameworks today for the development of *Deep Learning models*. Although the common denominator between these frameworks is the training and tuning of *deep learning models*, the "*how they do it*" and the sector they are aimed at are the main differentiator. Some frameworks are more research-oriented (such as *PyTorch*) and some others are mostly used for device deployment (such as *Tensorflow*), likewise, some frameworks base their design architecture on static graphs (such is the case *Tensorflow* and *Caffe2*) and others in dynamic graphs (such is the case of *PyTorch*).

Obviously, each of these frameworks offers different advantages over the others, however, how could we link these advantages? How could we interoperate different frameworks? How could we optimize a model with framework "*x*" and deploy it in an architecture optimized for framework "*y*"? Well, this type of interoperability is achieved thanks to the **ONNX** standard and the **ONNX Runtime** (which we will see later). In Figure 1, it is described the problematic addressed by **ONNX** and **ONNX Runtime.**

Figure 1. Problematic addressed by ONNX and ONNX Runtime | Image by author | Icons taken from flaticon

ONNX has come to break the dependency between frameworks and hardware architectures. **ONNX** seeks to become the default standard for *portability and interoperability* between the different *Deep Learning frameworks*. So, let's see in a little more detail what is **ONNX** and what is **ONNX Runtime**.

What is ONNX?

ONNX is the acronym that stands for ***Open Neural Network Exchange***. Which refers to a standard model that facilitates interoperability between *Deep Learning frameworks.* The **ONNX** standard began in 2017 at the initiative of the giants Microsoft, Facebook and Amazon. The basic idea was to propose a standard that would allow *portability and interoperability* between the already well-known *Deep Learning frameworks*.

> *Open Neural Network Exchange (ONNX) is an open ecosystem that empowers AI developers to choose the right tools as their project evolves [1].*

ONNX has come to streamline the lifecycle of *Machine Learning models*, from research to production, since it is common for some frameworks to be more suitable for prototyping and optimizing models while others provide tools that speed up deployment to different devices. Therefore, it is important to mention that currently **ONNX** has the capacity to support inference, that is, we can train a model in framework "*a*" and perform inference in framework "*b*". Figure 2 provides a visual description of *framework interoperability*.

Figure 2. ONNX interoperability | Image by author | Logos taken from original source

> *The ONNX specification addresses the following three components to enable interoperability:*
> *1. A definition of an extensible computation graph model.*
> *2. Definitions of standard data types.*
> *3. Definitions of built-in operators.*

To date, several frameworks already integrate an extension to be able to export models under the **ONNX** specification, likewise the frameworks that have not yet integrated the export module make use of wrappers that work perfectly.

Well, so far we already know why a standard is required for *interoperability* between frameworks, in the same way we already know what the participation of **ONNX** is in this ecosystem, now we have to know what "***ONNX Runtime***" is and the great impact that it generates in the industry, so let's go for it!

What is ONNX Runtime?

ONNX Runtime is a multiplatform accelerator focused on training and model inferences compatible with the most common *Machine Learning & Deep Learning frameworks* [2]. In other words, **ONNX Runtime** is the implementation of the **ONNX** standard.

ONNX Runtime arises due to the need for an interface that accelerates inference in different hardware architectures. Before **ONNX Runtime**, it was very expensive to deploy models that were optimized mainly for *CUDA-based* architectures towards *NUPHAR*, *nGraph*, *OpenVINO*-based architectures, etc. In other words, there was a dependency between the framework and the hardware architecture for which the model was optimized. It is with the **ONNX** standard and the **ONNX Runtime** accelerator that the doors are opened to a wide spectrum of *interoperability* between frameworks and hardware architectures.

Some key benefits of **ONNX Runtime** are:

- Improvement in inference performance, inference time is considerably reduced.
- Reduced training time

- Develop and train models in Python and deploy in C, C ++ or Java based applications.

Great, now we know the impact of **ONNX** and **ONNX Runtime** in terms of *interoperability and portability*, let's see an example!

Interoperability: from PyTorch to other Frameworks

In the following example we are going to demonstrate how to use the **ONNX** standard to be able to interoperate between different *Deep Learning frameworks*.

The architecture of the example is given as follows, we are going to train a classifier in **PyTorch**, then we are going to use this trained model to perform inference in **Tensorflow**, **Caffe2** and **ONNX Runtime**. The architecture of the example is given as follows:

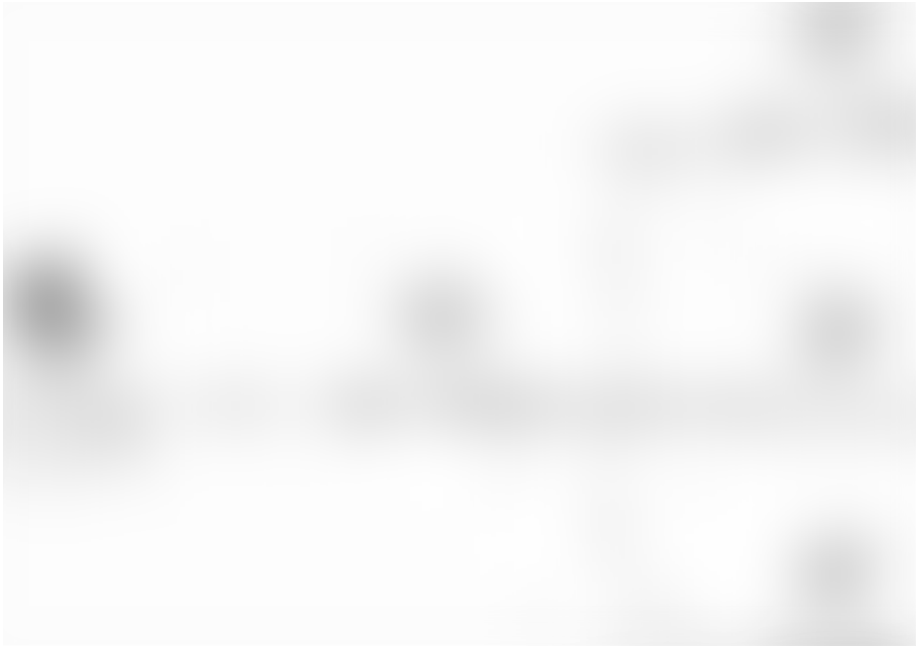

Figure 3. Example architecture | Image by author | Icons taken from flaticon

Let's start!

> *If you want to take a look at the complete code, this is the implementation: https://github.com/FernandoLpz/ONNX-PyTorch-TF-Caffe2 . Feel free to clone or fork!*

First we are going to create *generic data*, the idea is to create a dummy model, so let's define the following generator that will return the training and test data:

Code snippet 1. Data Generator

Now let's define the structure of our dummy model as well as the forward function. We basically define two linear layers, that's enough.

Code snippet 2. PyTorch model definition

Perfect! So far we already have the generic data as well as our dummy model, it is time to train the model!

Code snippet 3. PyTorch training model

As we can see, the training phase is very simple and does not require much explanation. So let's move on to the following, it's time to export our model to the **ONNX** standard, for this **PyTorch** already provide us with an extension to export models in **ONNX format**, let's see how we do it!

Code snippet 4. Exporting to ONNX

Let's first look at the if-else statement of line 4. We are defining a "*dummy input*" because **ONNX** needs to traverse the entire graph defined by **PyTorch**, in this way **ONNX** will be in charge of tracking each

of the layers and parameters defined in each graph instance. In this case, we are defining a generic input with the "*variable*" or in its case, we are defining the dummy input as the real input that is used in the training.

Later, in lines 9 and 10, we define the names that we will assign to each layer in the graph that **ONNX** will generate, if not, **ONNX** will use generic names.

Finally, in line 13 we make use of the **ONNX** extension of **PyTorch**, we pass as arguments the trained model, the dummy input and the names that we assign to each element of the graph. It is important to look at the argument passed in line 22, since inference tensors can have some variations in dimension size (commonly it is the dimension that refers to the batch size), we define such dimension as "*dynamic*", therefore at the time of inferring, we can pass any batch size, not specifically the one used when training the original model.

Well, so far we have already trained a model in **PyTorch** and saved under the **ONNX** standard, now we see how to load this model with different frameworks to make the inference.

ONNX Runtime Inference

To perform inference with **ONNX Runtime**, we need to import the *onnxruntime* library, then we just need to load the *onnx model* with the *onnx module* and generate the predictions.

Code snippet 5. ONNX Runtime inference

Caffe2 Inference

To make predictions with the **caffe2** framework, we need to import the *caffe2 extension* for *onnx* which works as a backend (similar to the session in tensorflow), then we would be able to make predictions.

Code snippet 6. Caffe2 inference

Tensorflow Inference

To make predictions with **Tensorflow** it is required to make use of the *onnx_tf* module which provides a wrapper (which simulates the session), in order to make predictions.

Code snippet 7. Tensorflow inference

Congratulations! we reached the end of the blog.

> *If you want to take a look at the complete code, this is the implementation: https://github.com/FernandoLpz/ONNX-PyTorch-TF-Caffe2 . Feel free to clone or fork!*

Conclusion

In this blog, we explained the need for which the **ONNX** standard arises. We also addressed the idea of how **ONNX** helps prevent *framework lock in*. On the other hand, we explained how the **ONNX** standard in conjunction with **ONNX** Runtime allows us to reduce the time in the *Deep Learning lifecycle*, since it accelerates the connection between the training phase and deployment phase.
Finally, we saw an example where, using **ONNX**, we can train a model in a given framework and perform inference in other frameworks.

References

[1] https://github.com/onnx/onnx

[2] https://microsoft.github.io/onnxruntime/docs/

miss. Take a look.

Get this newsletter

You'll need to sign in or create an account to receive this newsletter.

19 1

Onnx Onnx Runtime Pytorch TensorFlow Interoperability

More from Towards Data Science

Follow

Your home for data science. A Medium publication sharing concepts, ideas and codes.

Read more from Towards Data Science

More From Medium

Run Your Python Code as Fast as C

Marcel Moosbrugger in Towards Data Science

Get Interactive plots directly with pandas.

Parul Pandey in Towards Data Science

17 Clustering Algorithms Used In Data Science & Mining.

Mahmoud Harmouch in Towards Data Science

5 Deep Learning Trends Leading Artificial Intelligence to the Next Stage

Alberto Romero in Towards Data Science

Automate Microsoft Excel and Word using Python

M Khorasani in Towards Data Science

How to generate automated PDF documents with Python

M Khorasani in Towards Data Science

Top 10 Data Science Projects for Beginners

Natassha Selvaraj in Towards Data Science

Clean Code for Data Scientists

Ella Bor in Towards Data Science