**Research Into ONNX**

Cal State San Marcos

Professor G. Abla

Chet Gurevitch, Alexandre Renaud, Rodolfo Rodriguez

“ONNX is an open format built to represent machine learning model” (“ONNX”,2020). As of writing there are over 21 major deep learning frameworks, each with its own characteristics, limitations and strengths. Likely hundreds of smaller projects just getting started, considering the industry has billions of dollars in backing from major tech and financial corporations. As with any type of software, these machine learning platforms don’t work in a vacuum, they are parts of software projects with many moving parts. So if you have this many different standards and a need to get them to work nicely in as many use cases as possible, you’re going to need to start creating a standard. ONNX is such standard, created based on foresight that we as a community would need to start working towards a similar goal, but allow everyone to still do so in their own framework. The ONNX framework gives a standard Machine learning model that many of the key machine learning frameworks can export to, along with a runtime that allows them to be used and created with our current software infrastructure.

Portability is the primary goal of ONNX, and its success in enabling it has earned it rapid adoption across the industry. Often the best framework to train a model is not the best to use for inference on your target platform, either because hardware acceleration is missing, or any number of other reasons. Previously this would create large obstacles for any number of projects and goals. The ability to export models to ONNX format, either directly from the framework or using an external/third party tool is likely possible with whatever given framework you are using just a few years after the project’s creation.

As we it should be clearly apparent the main purpose of ONNX is a shared format for all ai frameworks. Having a standard format is a key first step, but we can move further beyond that using ONNX. Say we have a Machine learning data scientist who is working in a Pytorch workflow. He wants to try and review the machine learning model a fellow scientist has, but the other scientist has a workflow geared around TensorFlow. Now if they setup a ONNX runtime, they could just end it here and run the model using the .ONNX model. However, via ONNX we can do one better, this theoretical data scientist can convert the TensorFlow model into a ONNX model, and then from there convert that model into a Pytorch model. This level of flexibility is a first in this field and offers data scientist and those working with these machine learning ai an opportunity to share and work cooperatively with extremely high level of ease.

Initially started by Facebook and Microsoft, ONNX development has been driven by large tech companies, with many more having signed on since such as Nvidia, intel, Huawei and more. All these parties have a shared interest in allowing themselves to easily switch between different machine learning frameworks, some of which may support more models, and some may perform better, either generally or for a specific task. Facebook’s initial use case was for transferring models from Pytorch over to Caffe2, the latter being a backend that would perform much faster but didn’t support the models they needed. Microsoft and Intel also collaborated to integrate support for the graph compiler into the runtime, which provides a large boost for CPU inference, while Nvidia helped integrate TensorRT, which also provides major gains.

With Microsoft being a founding developer of ONNX, it’s no surprise that integration into windows apps, as well as their Azure cloud platform, is a focus. Microsoft has a suite of AI integration options, such as Windows Machine Learning, for integrating a model and running it locally on user’s devices, Windows Vision Skills for image processing specific uses, and DirectML for more precise control over resources and execution, particularly for game developers. Relatedly, they maintain their own framework called ML.net, which while a general machine learning framework is particularly easy to integrate within .NET apps, and can consume other frameworks such as ONNX. Microsoft also have a number of visual tools for the above, such as AutoML, which allows users to build a model entirely within a GUI, and the WinML dashboard, which allows manipulating, visualizing, and testing Windows ML models, using the ONNX format. The ability to convert between ONNX models is also integrated within the gui.

In our practical example we go through the process of creating a deep learning model in both TensorFlow and in Pytorch, after which we then convert them into . ONNX models. We then use the ONNX runtime to validate that they are both readable and ready for use with ONNX. Due to the length of time it would take to learn how to implement two different frameworks, we have elected to using default settings for our machine learning model, and based them off MNIST dataset. TensorFlow does not have a robust set of tools for exporting your model into a . ONNX model. In "Tensor.py" We train a Sequential Keras model, with 128 layers. This returns a set of logits that are converted into predictions via the softmax() function.

At this point the model is untrained, so most predictions are random, so we train and compile the model. TensorFlow/keras does it's magic, and begins to train the model. By the time we complete, our model is 98% accurate with the dataset, so we save this for converting as "/saved\_model". As we mentioned above tensorflow/keras does not have built in support for conversion into ONNX models, but there exists small libraries for just this. A single string command, (referenced in comments in code) we convert the tensorflow model into a ONNX model as "model.onnx". Now when proceeding with the Pytorch model, in "pytorch.py" we go with a much simpler approach. We load in a pre-trained model from the Pytorch defaults, and using built in exporting tools we create a . ONNX file named "pytorch.onnx" When looking at the line count, we can see that the process is drastically easier with Pytorch, but this should not be a discouragement to using other models. To complete our practical example, in "OnnxTest.py" I load in both of the . ONNX models we have created, one via Pytorch the other via tensorflow. We proceed with running a checker() function built into the ONNX runtime and both pass fine. This showed that yes the ONNX model can and does move past the limitations of working with a single machine learning framework.

The rapid growth of ai and machine learning has been shocking to view, as it has seemingly exploded in such a short window of time. This is of course due to the many doors it opens for us as software engineers, allowing for problems that were very recently insurmountable with modern software being conquered by machine learning. It is clear that there is a interest in maintaining this level of growth, and it is clear that issues such as compatibility and interoperability are soon to be trivial with ONNX. The ONNX framework is in our view an essential steppingstone for future growth in the technology sector that is machine learning and ai. It will help ensure open and collaborative growth in machine learning.

Works Cited

<https://onnx.ai/about.html>  
<https://research.fb.com/facebook-and-microsoft-introduce-new-open-ecosystem-for-interchangeable-ai-frameworks/>

<https://docs.microsoft.com/en-us/windows/ai/windows-ml/get-onnx-model>

<https://medium.com/@santhoshkolloju/convert-your-pytorch-models-to-tensorflow-with-onnx-84c3bdd8d722>

<https://github.com/onnx/onnxmltools>

<https://medium.com/moonvision/onnx-the-long-and-collaborative-road-to-machine-learning-portability-a6416a96e870>