ONNX Runtime

Introduction:

ONNX Runtime, a high-performance inference engine for machine learning models in the Open Neural Network Exchange (ONNX) format. ONNX Runtime is compatible with ONNX version 1.2 and comes in Python packages that support both CPU and GPU to enable inferencing using Azure Machine Learning service and on any Linux machine running Ubuntu 16.

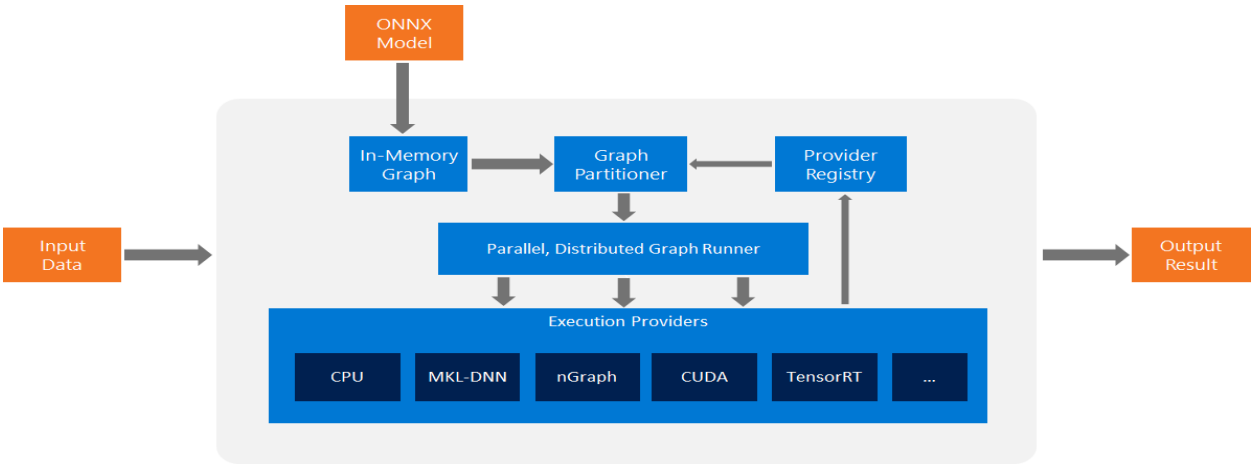
ONNX is an open source model format for deep learning and traditional machine learning. ONNX gives data scientists and developers the freedom to choose the right framework for their task, as well as the confidence to run their models efficiently on a variety of platforms with the hardware of their choice.

ONNX Runtime, a high-performance inference engine for machine learning models in the Open Neural Network Exchange (ONNX) format.

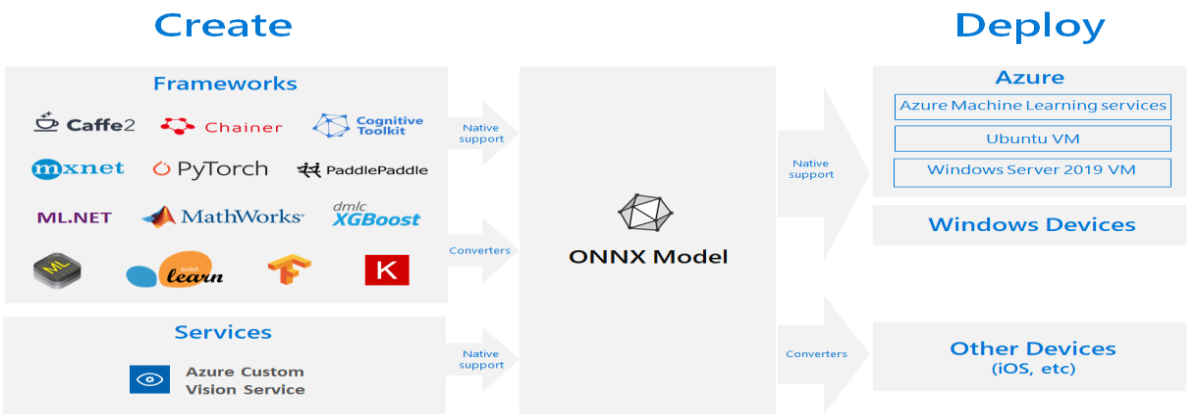
Details:

The ONNX Runtime inference engine provides comprehensive coverage and support of all operators defined in ONNX. Developed with extensibility and performance in mind, it leverages a variety of custom accelerators based on platform and hardware selection to provide minimal compute latency and resource usage. Given the platform, hardware configuration, and operators defined within a model, ONNX Runtime can utilize the most efficient execution provider to deliver the best overall performance for inferencing.

The pluggable model for execution providers allows ONNX Runtime to rapidly adapt to new software and hardware advancements. The execution provider interface is a standard way for hardware accelerators to expose their capabilities to the ONNX Runtime. We have active collaborations with companies including Intel and NVIDIA to ensure that ONNX Runtime is optimized for compute acceleration on their specialized hardware. Examples of these execution providers include Intel's MKL-DNN and nGraph, as well as NVIDIA's optimized TensorRT.



The release of ONNX Runtime expands upon Microsoft's existing support of ONNX, allowing you to run inferencing of ONNX models across a variety of platforms and devices.
Azure: Using the ONNX Runtime Python package, you can deploy an ONNX model to the cloud with Azure Machine Learning as an Azure Container Instance or production-scale Azure Kubernetes Service. Here are some examples to get started.
.NET: You can integrate ONNX models into your .NET apps with ML.NET.
Windows Devices: You can run ONNX models on a wide variety of Windows devices using the built-in Windows Machine Learning APIs available in the latest Windows 10 October 2018 update.



Training a digit recognizer using PyTorch, and inferencing on CPU with

ONNX Runtime:

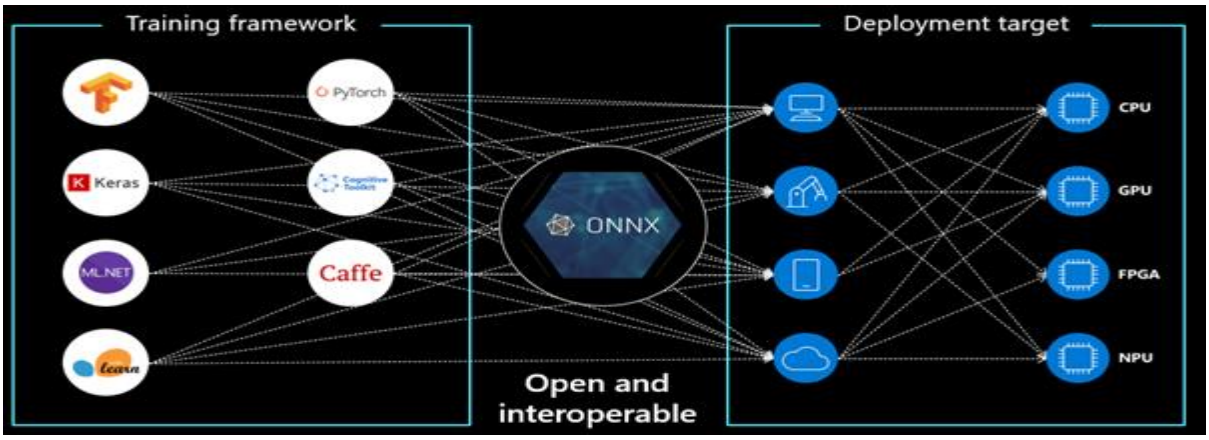
In any machine learning problem, the goal of our neural network is to do well on the new unseen data, training a deep learning model helps to achieve this goal. We have to focus on running an inference session and ensuring the model works perfectly when deployed in a specific environment.
You can train your model anywhere in your preferred framework but the environment where you need to run the inference session need not be favorable to that particular framework. For example, some time ago, applications preferred the Caffe model for deployment. Does this mean we have to use the Caffe framework for training as well? No, this is where the Open Neural Network Exchange (ONNX) model format comes into the picture.
With ONNX, one can switch between deep learning frameworks such as PyTorch and Caffe2.
Similarly, we may want to deploy our model on GCP, a mobile app, or somewhere else. During the inference session, we do not give much importance

to the framework that we used to train our model. In fact, we don't need to use any framework at all for running an inference session, all we need is the trained core neural network in whichever format that can produce good and fast results. And this is where inference engines like ONNX Runtime, TensorRT, etc. can help us.

Let's dive deep into ONNX and ONNX Runtime before using them in our task. Our task is to train a neural network with PyTorch using the MNIST dataset to make predictions of handwritten digits in real-time and deploying the model on ONNX Runtime. I'm not going to discuss how to train your model for digit classification. We'll focus more on ONNX and ONNX Runtime.

What is ONNX?

ONNX is an intermediate representation of our trained model that can be used for switching between frameworks and also for running inference sessions either on-prem or at the edge.



From the above image, it is clear that once you train your deep neural networks in whichever framework you prefer, ONNX takes that model, usually saved as .pth file, and converts it into a format that is suitable for deployment.

To understand how it is suitable, let's start from the basics, neural network that we build in whichever frameworks are nothing but computations achieved through a dataflow graph, in other words, a computational graph. Some of these graphs are static and some are dynamic. These graphs are just an

intermediate representation of the neural network that we build and they are translated to run on a specific device like GPUs, CPUs, etc.

Different frameworks have their intermediate representations of the graph. Also, the available frameworks are optimized for particular tasks in the ML pipeline, for example, although PyTorch is used for deploying your models nowadays, in the recent past, the majority of its usage came from research. Conversion of models between frameworks used to delay the deployment process. ONNX format provides a common intermediate representation with which one can choose their framework of choice and expedite the process of model deployment. Currently, ONNX is focused more on inferencing.

Here are the benefits of using ONNX Runtime:

- Improve inference performance for a wide variety of ML models
- Reduce the 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

To run the model in ONNX Runtime:

1. Create an inference session for the model with the chosen configuration parameters.
2. Evaluate the model using the run API. The output of this call is a list containing the outputs of the model computed by the ONNX Runtime.

One way to verify the model's output with ONNX Runtime is to compare it with the PyTorch model's output for the same input. This is achieved by the last two lines in the above implementation.

Conclusion:

The time taken by our model running on ONNX Runtime to give predictions is very much less than the time taken by our model when evaluating directly on CPU. ONNX Runtime for this task is 35% faster. This depends on CPU performance and can vary for different tasks. The below image shows the time taken by our model to evaluate a single image on CPU and ONNX Runtime.