Simio

Forward Thinking

ONNX Format Utilities

Table of Contents

1 Neural network ONNX conversion packages

1.1 From: TensorFlow/Keras/TFLite To: ONNX

This operation can be completed using a package called tensorflow-onnx, which you can find at the link below. Further documentation can be found in the project's GitHub repository.

https://github.com/onnx/tensorflow-onnx

Command Line Interface
The tf2onnx package contains a command line interface (CLI) that allows you to convert tensorflow models to ONNX files. The example below shows how you might convert a neural network model trained in TensorFlow to the ONNX format. A complete CLI reference can be found at https://github.com/onnx/tensorflow-onnx#cli-reference

```
python -m tf2onnx.convert --saved-model TENSORFLOW_NEURAL_NETWORK --output ONNX_NEURAL_NETWORK.onnx
```

Python API
The tf2onnx package also has a python API which can be used to save TensorFlow models to an ONNX file. A complete reference to the Python API can be found at https://github.com/onnx/tensorflow-onnx#python-api-reference.

```
pip install tf2onnx
```

See below an example from the tensorflow-onnx GitHub repository where a model created in Keras is saved to an ONNX file. Please see the source link for the complete example.

```
import tf2onnx

spec = (tf.TensorSpec((None, 224, 224, 3), tf.float32, name="input"),)
output_path = model.name + ".onnx"

model_proto, _ = tf2onnx.convert.from_keras(model, input_signature=spec, opset=13, output_path=output_path)
```

From https://github.com/onnx/tensorflow-onnx/blob/master/tutorials/keras-resnet50.ipynb

Version requirements:

Package	Version
Python	3.7-3.9 (Required by TensorFlow)
TensorFlow	1.12-1.15, 2.1-2.7

Notes:
Python 3.9 support requires TensorFlow 2.5 or later, while Python 3.8 support requires TensorFlow 2.2 or later. The tensorflow-onnx package recommends using Python 3.7 for performance reasons. For more information, see the complete compatibility notes at the sources below.

https://www.tensorflow.org/install/pip
https://github.com/onnx/tensorflow-onnx#supported-versions

1.2 From: ONNX To: Tensorflow/Keras/Tflite

This operation can be completed using a package called onnx-tensorflow, which you can find at the link below. Further documentation and examples can also be found in the project's GitHub repository.

https://github.com/onnx/onnx-tensorflow

Command Line Interface

The onnx-tensorflow package contains a command line interface (CLI) that allows you to convert convert ONNX files to TensorFlow models. A complete CLI reference can https://github.com/onnx/onnx-tensorflow/blob/master/doc/CLI.md

```
onnx-tf convert -i ONNX_NEURAL_NETWORK.onnx -o TENSORFLOW_NEURAL_NETWORK
```

Python API

The onnx-tensorflow package also has a python API which can be used to convert ONNX files to TensorFlow models. You can install onnx-tensorflow from PyPi using the command below.

```
pip install onnx-tf
```

See below an example from the onnx-tensorflow GitHub repository where an ONNX file is loaded using ONNX, converted to a TensorFlow graph, and then exported.

```
import onnx
from onnx_tf.backend import prepare

onnx_model = onnx.load("input_path")  # load onnx model
tf_rep = prepare(onnx_model)  # prepare tf representation
tf_rep.export_graph("output_path")  # export the model
```

From https://github.com/onnx/onnx-tensorflow/blob/master/example/onnx_to_tf.py

Version requirements:

Package	Version
Python	3.7-3.9 (Required by TensorFlow)
TensorFlow	2.6.0
ONNX	See specific version of ONNX here.

Notes

The onnx-tensorflow package requires a specific version of ONNX, which can be found at the link shown in the Version requirements table. If installing via pip, this requirement should be fulfilled automatically. Python 3.9 support requires TensorFlow 2.5 or later, while Python 3.8 support requires TensorFlow 2.2 or later. Additional dependency information can be found at the sources below.

https://github.com/onnx/onnx-tensorflow#converting-models-from-onnx-to-tensorflow
https://www.tensorflow.org/install/pip

1.3 From: PyTorch To: ONNX

PyTorch contains a module that can be used to export a model to an ONNX file. The example below, taken from the PyTorch website, shows how a model can be exported by calling torch.onnx.export().

```
import torch.onnx
# Input to the model
x = torch.randn(batch_size, 1, 224, 224, requires_grad=True)
torch_out = torch_model(x)

# Export the model
torch.onnx.export(torch_model,               # model being run
                  x,                         # model input (or a tuple for multiple inputs)
                  "super_resolution.onnx",   # where to save the model
                  export_params=True,        # store the parameter weights inside the model file
                  opset_version=10,          # the ONNX version to export the model to
                  do_constant_folding=True,  # whether to execute constant folding for optimization
                  input_names = ['input'],   # the model's input names
                  output_names = ['output'], # the model's output names
                  dynamic_axes={'input' : {0 : 'batch_size'},    # variable length axes
                                'output' : {0 : 'batch_size'}})
```

From https://pytorch.org/tutorials/advanced/super_resolution_with_onnxruntime.html

For more information on exporting Pytorch models in the ONNX format, please see the PyTorch tutorial linked below. In addition to walking through the process of exporting a PyTorch model to ONNX, the article also includes advice regarding operations or types to avoid.

https://pytorch.org/docs/master/onnx.html

Version requirements:

Package	**Version**
Python	3.X (Required by PyTorch)

1.4 From: ONNX To: PyTorch

PyTorch does not support conversions from ONNX to PyTorch; however, some users recommend the onnx2pytorch package. This package is still in development and may not convert all types of models. The onnx2pytorch Python package can be found at https://github.com/ToriML/onnx2pytorch.

You can install onnx2pytorch from PyPi using the command below.

```
pip install onnx2pytorch
```

The example code below shows how you can load your ONNX neural network using ONNX, then converting the model to PyTorch using onnx2pytorch.

```
import onnx
from onnx2pytorch import ConvertModel

onnx_model = onnx.load("ONNX_NEURAL_NETWORK.onnx")
pytorch_model = ConvertModel(onnx_model)
```

From https://github.com/ToriML/onnx2pytorch

1.5 From: scikit-learn To: ONNX

This operation can be completed using a Python package called skl2onnx, which you can find at the link below. Further documentation can be found at http://onnx.ai/sklearn-onnx/.

https://github.com/onnx/sklearn-onnx

Python API
The skl2onnx package can convert many different types of models to ONNX files. A complete reference for the types of models supported by sklearn-onnx can be found at http://onnx.ai/sklearn-onnx/supported.html. You can install sklearn-onnx from PyPi using the command below.

```
pip install skl2onnx
```

See below an example from http://onnx.ai/sklearn-onnx/introduction.html where a logistic regression model is trained and then exported.

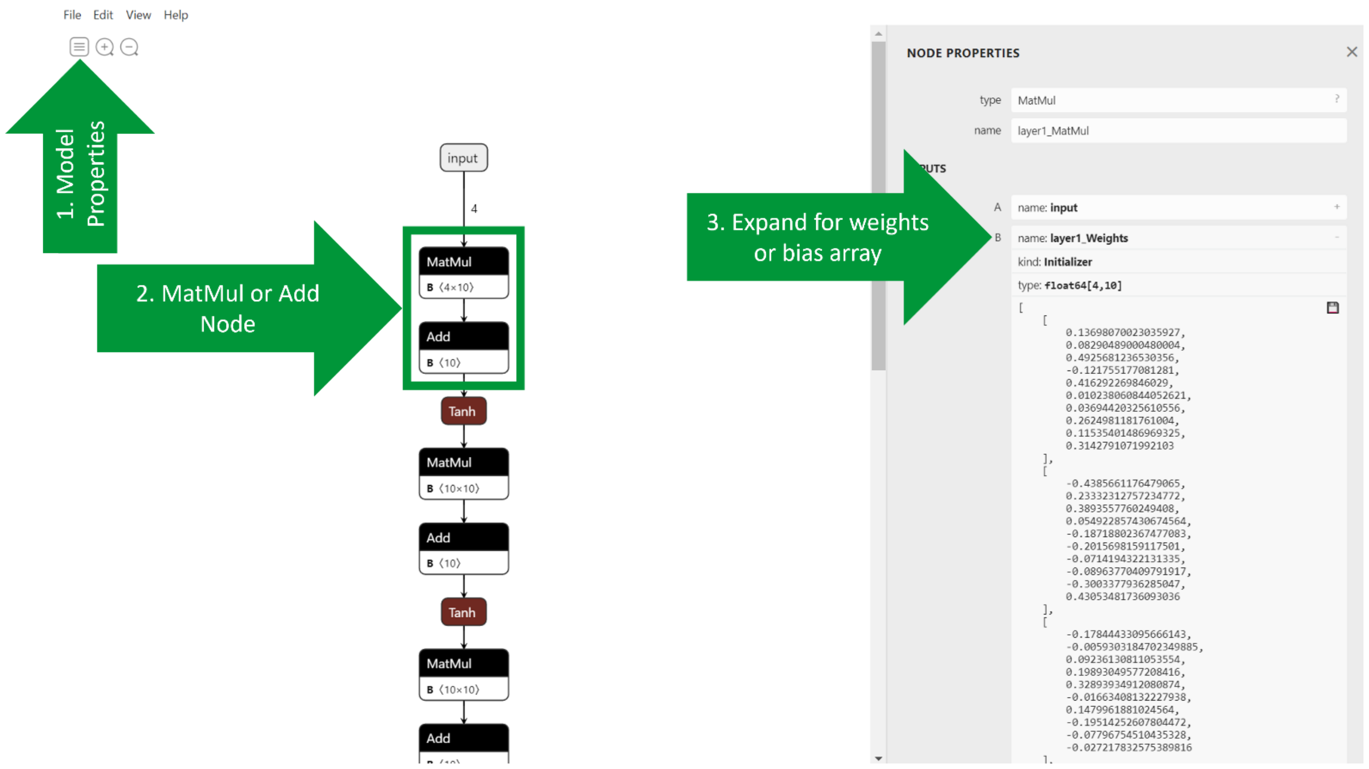
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y)

from sklearn.linear_model import LogisticRegression
clr = LogisticRegression()
clr.fit(X_train, y_train)

from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType

initial_type = [('float_input', FloatTensorType([None, 4]))]
onnx = convert_sklearn(clr, initial_types=initial_type)
with open("logreg_iris.onnx", "wb") as f:
    f.write(onx.SerializeToString())
```

2 ONNX viewing utilities

2.1 Netron

Netron is an application that allows users to view an ONNX file in a readable format. Netron is available either as an executable or as a webapp. You can download the executable at https://github.com/lutzroeder/netron, or access the webapp at netron.app.

When you open an ONNX file using Netron, you can select the "Model Properties" button in the upper left corner of the screen to open the model's properties. If you select any of the MatMul or Add nodes, you can expand the appropriate input to view the array of weights and biases. The array can then be copied, or you can save the weights as a NumPy Array.



```
type     MatMul
name     layer1_MatMul

A        name: input
B        name: layer1_Weights
         kind: Initializer
         type: float64[4,10]
[
    [
        0.13698070023035927,
        0.08290489000480004,
        0.4925681236530356,
        -0.121755177081281,
        0.416292269846029,
        0.010238060844052621,
        0.03694420325610556,
        0.2624981181761004,
        0.11535401486969325,
        0.3142791071992103
    ],
    [
        -0.4385661176479065,
        0.23332312757234772,
        0.3893557760249408,
        0.054922857430674564,
        -0.18718802367477083,
        -0.2015698159117501,
        -0.0714194322131335,
        -0.08963770409791917,
        -0.3003377936285047,
        0.43053481736093036
    ],
    [
        -0.17844433095666143,
        -0.005930318470234988 5,
        0.09236130811053554,
        0.19893049577208416,
        0.32893934912080874,
        -0.01663408132227938,
        0.1479961881024564,
        -0.19514252607804472,
        -0.07796754510435328,
        -0.027217832575389816
    ],
```

2.2 Protoc

Protoc is a protobuf utility that can be used to decode ONNX files into a readable format. To use Protoc, download it from the link below using either the win32 or win64 downloads and extract it. Then download the onnx.proto3 file which defines the ONNX protobuf schema.

Protoc download: https://github.com/protocolbuffers/protobuf/releases
onnx.proto3 download: https://github.com/onnx/onnx/blob/master/onnx/onnx.proto3

Once these files are downloaded, you can use the command line interface as shown below to convert the ONNX files to txt files.

```
protoc --decode=onnx.ModelProto -I ONNX_PROTO3_FILEPATH < ONNX_NEURAL_NETWORK.onnx > TEXT_OUTPUT.txt
```

For additional information, see the sources below.

Nvidia tutorial for working with ONNX:
https://developer.nvidia.com/blog/using-windows-ml-onnx-and-nvidia-tensor-cores/

Google’s protobuf documentation:
https://developers.google.com/protocol-buffers/