Recent developments with ONNX

May 29-30th

Xavier Dupré
Microsoft France

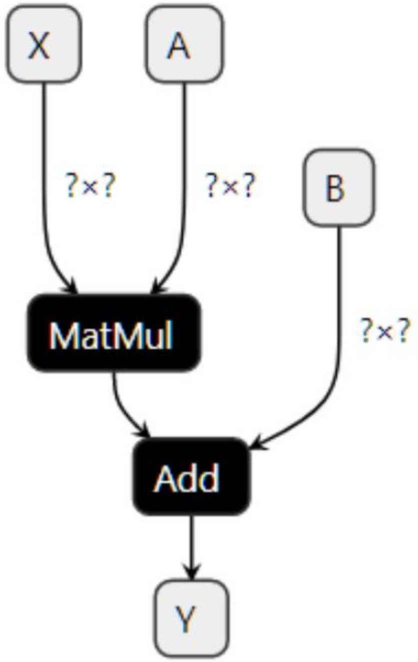
Plan

- About ONNX
- Converters
- onnxruntime
- onnxruntime-training
- onnxruntime-training and scikit-learn
- Write ONNX graphs…

About ONNX

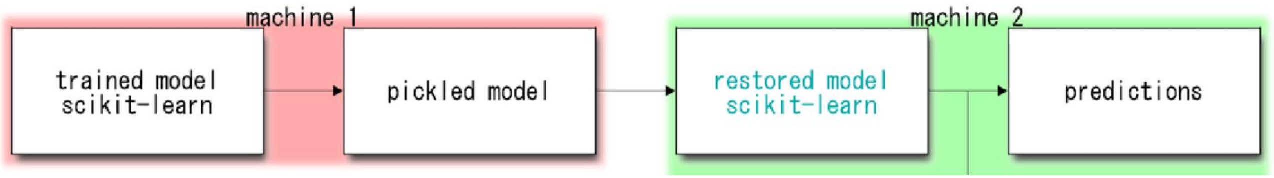
ONNX is a language

- Very close to a programming language
- Primitive are mathematical functions
- Supports Tests, loops, functions

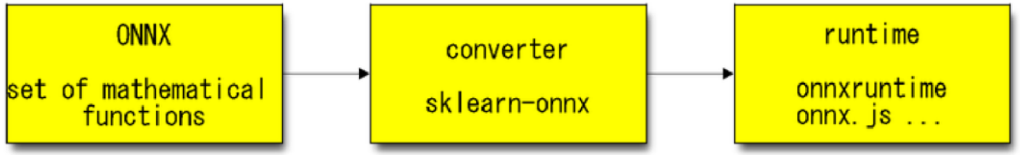


It is used in production

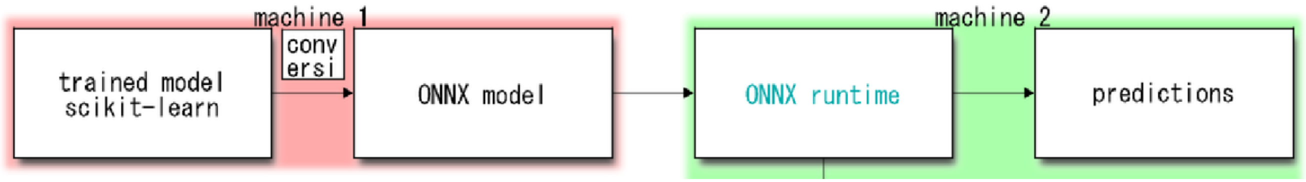
1. Train a model



2. Use a converter to get the implement of the prediction function with ONNX primitves.



3. Execute it with a runtime optimized for the production environment.



Why?

- ONNX primitives are very common and available in many environments.
- ONNX leverages protobuf to store the model coefficients.
- Once converted, a model does not depend on the training framework.
- onnxruntime (one runtime for onnx) is available in many environments and usually faster than the traning framework.
- Backward compatibility: old models are suppored.

History

- 2017/09: first release of onnx
- 2017/09: torch.onnx
- 2017/12: ONNX 1.0
- 2018/09: first release of onnxruntime
- 2018/12: first release of tf2onnx
- 2018/12: first release of onnxmltools
- 2019/01: first release of sklearn-onnx
- 2019/10: onnxruntime 1.0

- 2021/07: ONNX 1.10
- 2021/12: onnxruntime 1.12
- 2022/02: ONNX 1.11
- 2022/03: onnxruntime 1.11
- 2022/05: sklearn-onnx 1.11.2
- 2022/05: tf2onx 1.10.1

News in ONNX 1.12 or opset 17

Audio function (FFT, STFT)

- https://github.com/onnx/onnx/blob/main/docs/Operators.md#DFT
- https://github.com/onnx/onnx/blob/main/docs/Operators.md#STFT

DFT

Computes the discrete Fourier transform of input.

Version

This version of the operator has been available since versi

Attributes

`axis` *: int (default is 1)*
The axis on which to perform the DFT. By default this va

`inverse` *: int (default is 0)*
Whether to perform the inverse discrete fourier transfor

`onesided` *: int (default is 0)*
If onesided is 1, only values for w in [0, 1, 2, ..., floor(n_ff
conjugate symmetry, i.e., X[m, w] = X[m,w]=X[m,n_fft-w
possible. Enabling onesided with real inputs performs a
valued input, the default value is 0. Values can be 0 or 1

Inputs (1 - 2)

`input` *(non-differentiable) : T1*
For real input, the following shape is expected: [batch_ic
shape is expected: [batch_idx][signal_dim1][signal_dim2
dimentions correspond to the signal's dimensions. The t

`dft_length` *(optional, non-differentiable) : T2*
The length of the signal.If greater than the axis dimensic
only the first dft_length values will be used as the signal

Custom ONNX functions

A model can be split into multiple functions.
https://github.com/onnx/onnx/blob/main/docs/IR.md#functions

Name	Type	Description
name	string	The name of the function
domain	string	The domain to which this function belongs
doc_string	string	Human-readable documentation for this function. Markdown is allowed.
attribute	string[]	The attribute parameters of the function
input	string[]	The input parameters of the function
output	string[]	The output parameters of the function.
node	Node[]	A list of nodes, forming a partially ordered computation graph. It must be in topological order.
opset_import	OperatorSetId	A collection of operator set identifiers used by the function implementation.

Converters

Main converting libraries

- Tensorflow2onnx
- Onnxmltools (lightgbm, xgboost, sparkml, libsvm)
- Torch.onnx
- sklearn-onnx

- Other libraries
  - Chainer, matlab, …

Example with scikit-learn

```
reg1 = GradientBoostingRegressor(random_state=1, n_estimators=5)
reg2 = RandomForestRegressor(random_state=1, n_estimators=5)
reg3 = LinearRegression()

ereg = Pipeline(steps=[
    ('voting', VotingRegressor([('gb', reg1), ('rf', reg2), ('lr', reg3)])),
])
ereg.fit(X_train, y_train)
```

```
onx = to_onnx(ereg, X_train[:1].astype(numpy.float32),
              target_opset={'': 14, 'ai.onnx.ml': 2})
```

```
sess = InferenceSession(onx.SerializeToString(),
                        providers=['CPUExecutionProvider'])
pred_ort = sess.run(None, {'X': X_test.astype(numpy.float32)})[0]
```

This last part could be written in C, C++, C#, Java, javascript, obj-C.

Sklearn-onnx

1.11.2 Latest

xiaowuhu released this 3 days ago 1.11.2 f03b521

- LocalOutlierFactor n_neighbors bugfix #821
- MAINT compat link function and loss for sklearn 1.1 #863
- add sgd_oneclass svm converter #860

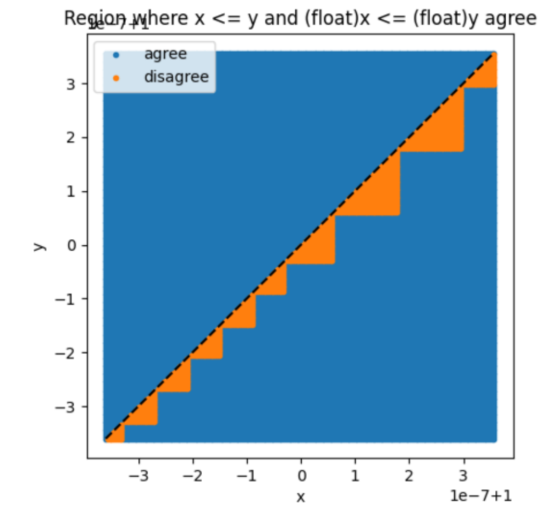
Assets 4

- skl2onnx-1.11.2-py2.py3-none-any.whl
- skl2onnx-1.11.2.tar.gz
- Source code (zip)
- Source code (tar.gz)

About Trees

- ONNX 1.10 only supports float threshold in trees
  - That was a cause of huge discrepancies for models trained with double thresholds.
- ONNX 1.11 supports both float and double
- Implement TreeEnsemble for opset(ai.onnx.ml)==3

https://github.com/microsoft/onnxruntime/pull/10821



p = Orange / Blue : probability that a comparison follows a different path.

1 – (1-p)^depth…

About sparse

- ONNX supports sparse tensors: https://github.com/microsoft/onnxruntime/blob/master/docs/OperatorKernels.md
- Support is still limited in onnxruntime but growing.

SparseToDenseMatMul	*in* A:**T** *in* B:**T1** *out* Y:**T1**	1+	**T** = sparse_tensor(double), sparse_tensor(float), sparse_tensor(int32), sparse_tensor(int64), sparse_tensor(uint32), sparse_tensor(uint64) **T1** = tensor(double), tensor(float), tensor(int32), tensor(int64), tensor(uint32), tensor(uint64)

About text

- Converting text into ONNX is not easy.
- One option is use onnxruntime-extensions a a preprocessing step

StringRegexSplitWithOffsets	Supported
StringECMARegexSplitWithOffsets	Supported
VectorToString	Supported
StringToVector	Supported
StringSlice	Under development
MaskedFill	Supported

Tokenizer

Operator	Support State
GPT2Tokenizer	Supported
WordpieceTokenizer	Supported
SentencepieceTokenizer	Supported
BasicTokenizer	Supported
BertTokenizer	Supported
BertTokenizerDecoder	Supported

ONNX to scikit-learn

- Impossible right now.
- Could be possible with onnx functions and a significant code change.
- Hyperparameters would be serialized in some way.

Revert onnx model to original format (scikit learn, tf...) #866

Open **lherbeur** opened this issue 5 days ago · 1 comment

lherbeur commented 5 days ago

Hi there,
I was looking to through the APIs for sklearn-onnx for the conversion of an onnx model but it seems that's not currently supported. Is there a way to do this and if not, can this be considered for an enhancement?
Similar request for tf reverse conversion - onnx/onnx-tensorflow#489.
Thanks.

Last quest: custom transformer

- Users write python code
- There is no automated way to convert it into ONNX.
- Needs an expert or…
- See in next sections.

onnxruntime

onnxruntime execute onnx graphs

- It executes onnx graphs.
- It is not depend on the OS or the processor.
- It can be called from many languages (python, C/C++, java…)

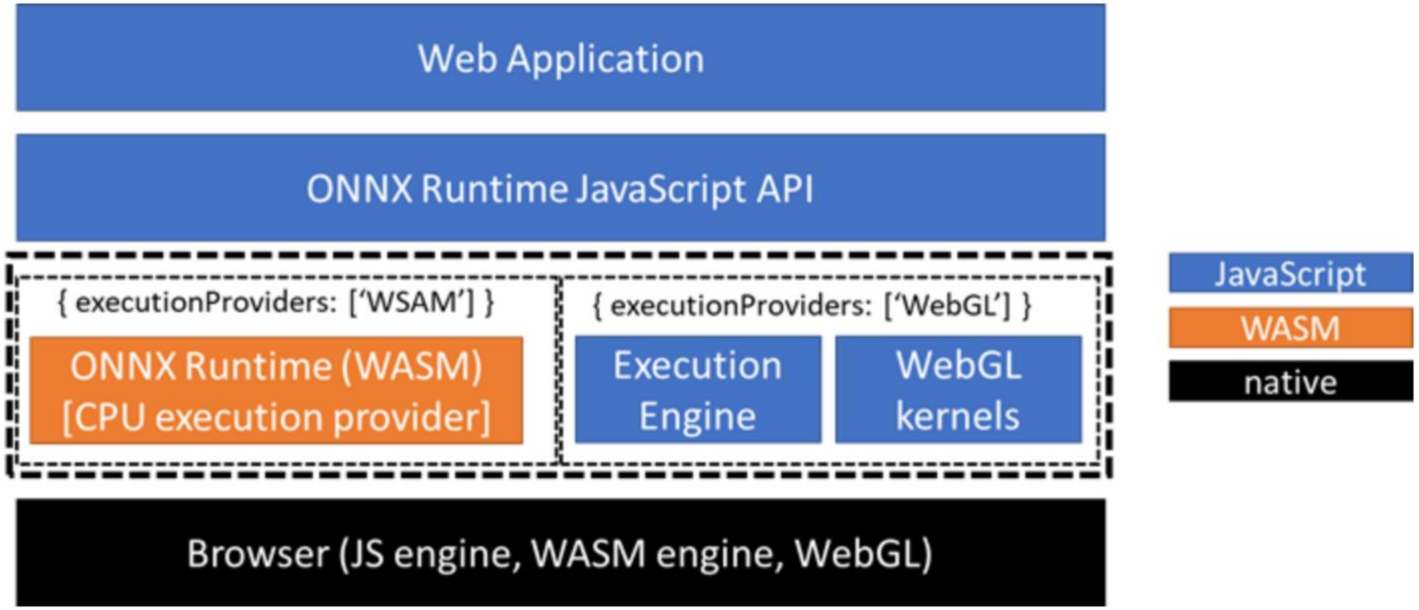
Environments

Optimize Inferencing | **Optimize Training**

Platform	Windows	Linux	Mac	Android	iOS	Web Browser (Preview)		
API	Python	C++	C#	C	Java	JS	Obj-C	WinRT
Architecture	X64	X86	ARM64	ARM32	IBM Power			
Hardware Acceleration	Default CPU	CoreML	CUDA	DirectML	oneDNN			
	OpenVINO	TensorRT	NNAPI	ACL (Preview)	ArmNN (Preview)			
	MIGraphX (Preview)	TVM (Preview)	Rockchip NPU (Preview)	Vitis AI (Preview)				
Installation Instructions	Please select a combination of resources							

Webassembly

- [ONNX Runtime Web—running your machine learning model in browser]()



Custom EP provider

- Provider = one implementation of an operator on a specific device
- Onnxruntime supports custom providers (TVM, ...)
- Optimizing and deploying transformer INT8 inference with ONNX Runtime-TensorRT on NVIDIA GPUs
- TVM Execution Provider

['TensorrtExecutionProvider',
'CUDAExecutionProvider',
'MIGraphXExecutionProvider',
'ROCMExecutionProvider',
'OpenVINOExecutionProvider',
'DnnlExecutionProvider',
'NupharExecutionProvider',
'TvmExecutionProvider',
'VitisAIExecutionProvider',
'NnapiExecutionProvider',
'CoreMLExecutionProvider',
'ArmNNExecutionProvider',
'ACLExecutionProvider',
'DmlExecutionProvider',
'RknpuExecutionProvider',
'CPUExecutionProvider']

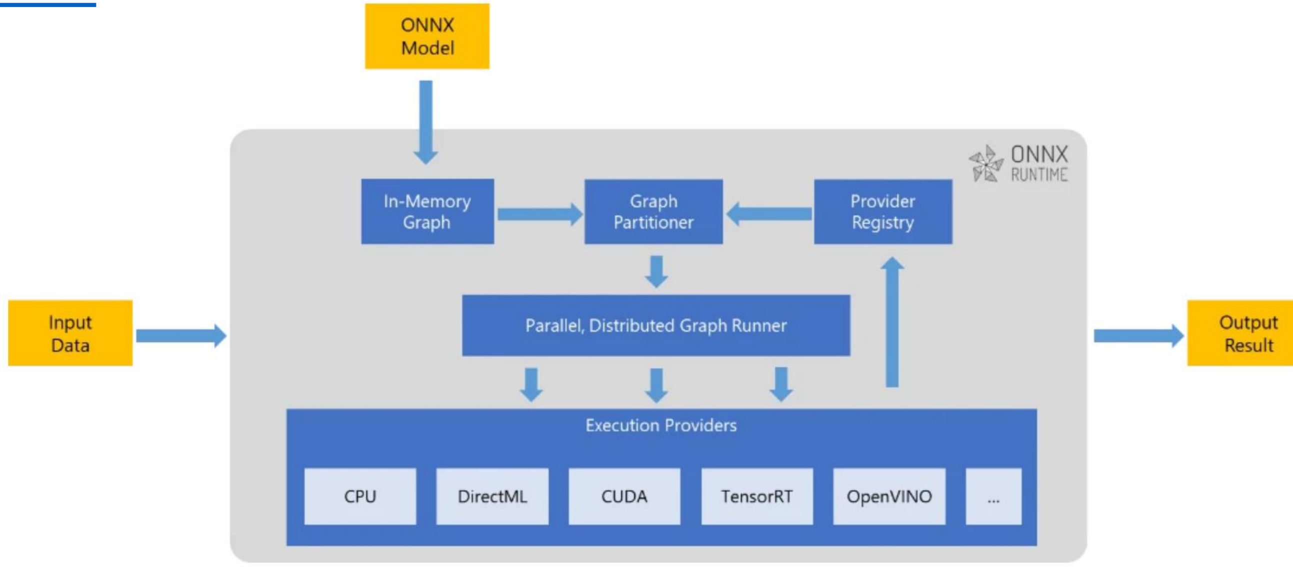
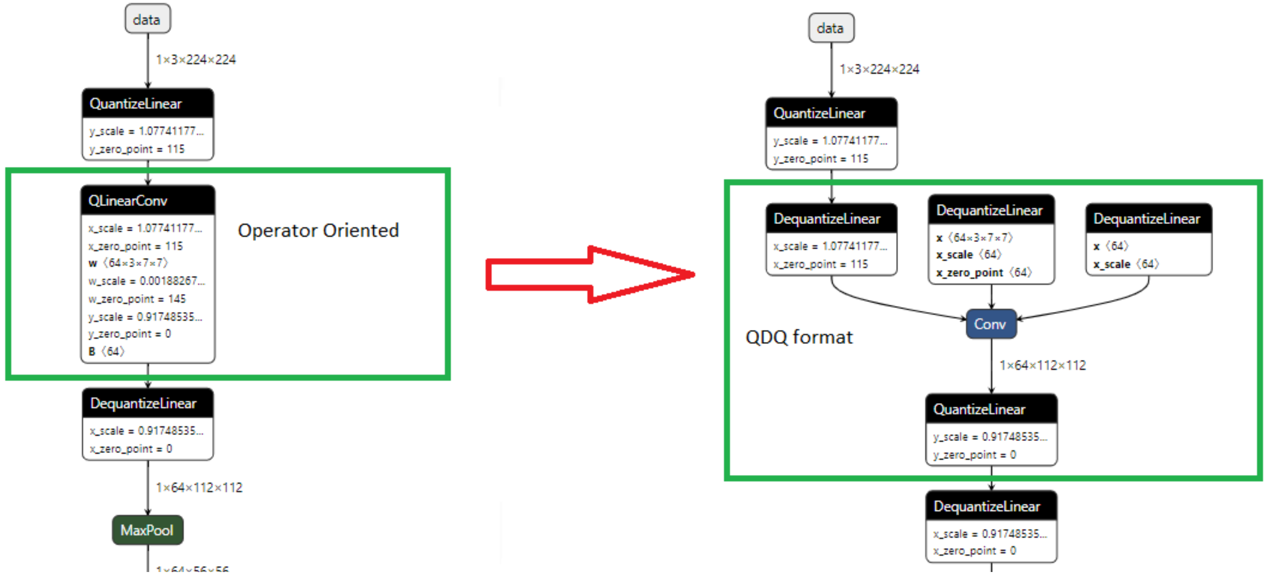


Figure 1: Different execution providers supported by ONNX Runtime.

Quantization, float16

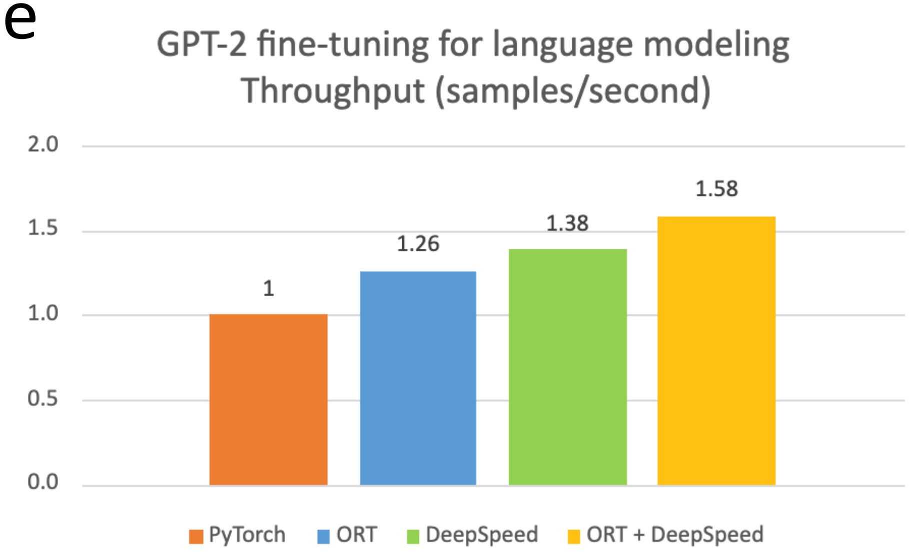
- Quantize ONNX Models
- Supported Operators and Data Types (see also Operators implemented by CUDAExecutionProvider)



Pytorch + onnxruntime

- [Scaling-up PyTorch inference: Serving billions of daily NLP inferences with ONNX Runtime]
- [Accelerate PyTorch training with torch-ort] (7/2021)
- [torch_ort]
- Possibility to use pytorch inside onnxruntime

```
class NeuralNet(torch.nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        ...

    def forward(self, x):
        ...

model = NeuralNet(input_size=784, hidden_size=500, num_classes=10)
model = torch_ort.ORTModule(model)
```



onnxruntime-training

onnxruntime-training

- onnxruntime-training is an extension of onnxruntime
- Compute a gradient over an ONNX graph
- Can update the weights of the graph
- Started to speedup training with pytorch
- GPT-2 fine-tuning with ONNX Runtime – a 34% speedup in training time (2020)

Final goal: train a model on any device.

onnxruntime-training

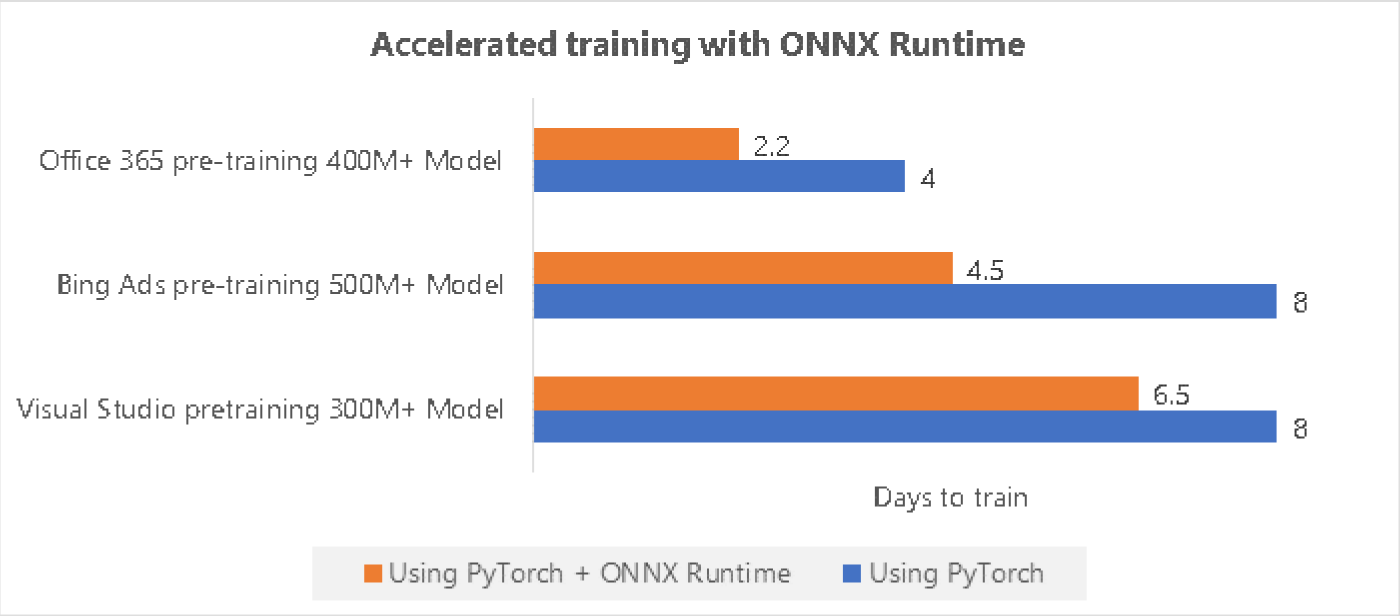
- Only on linux

Optimize Inferencing | Optimize Training

Platform	Linux		Windows		Mac	
API	PyTorch 1.8.1		PyTorch 1.9		C++	
Architecture	X64					
Hardware Acceleration	Default CPU	CUDA 10.2	CUDA 11.1	ROCm 4.2 (Preview)	ROCm 4.3.1 (Preview)	oneDNN
Installation Instructions	pip install torch-ort python -m torch_ort.configure					

Pytorch + onnxruntime to train

- [Announcing accelerated training with ONNX Runtime—train models up to 45% faster]()

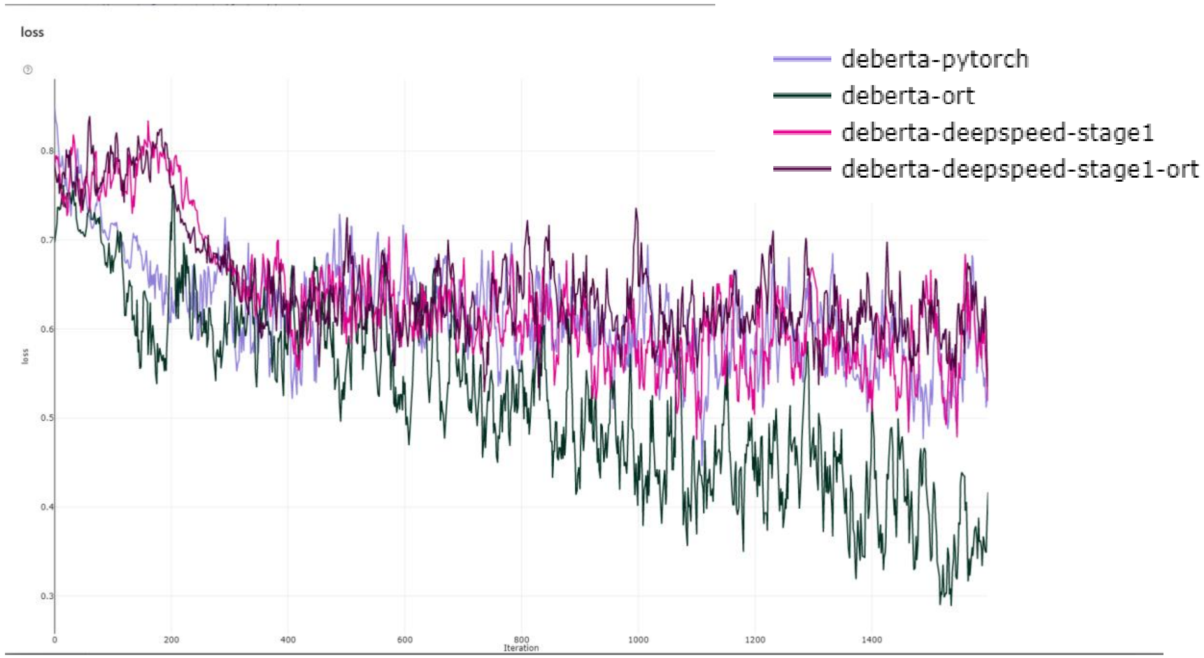


ORTModule faster than pytorch

- https://github.com/pytorch/ort

```
from torch_ort import ORTModule
model = ORTModule(model)
```

- # PyTorch training script follows



In details

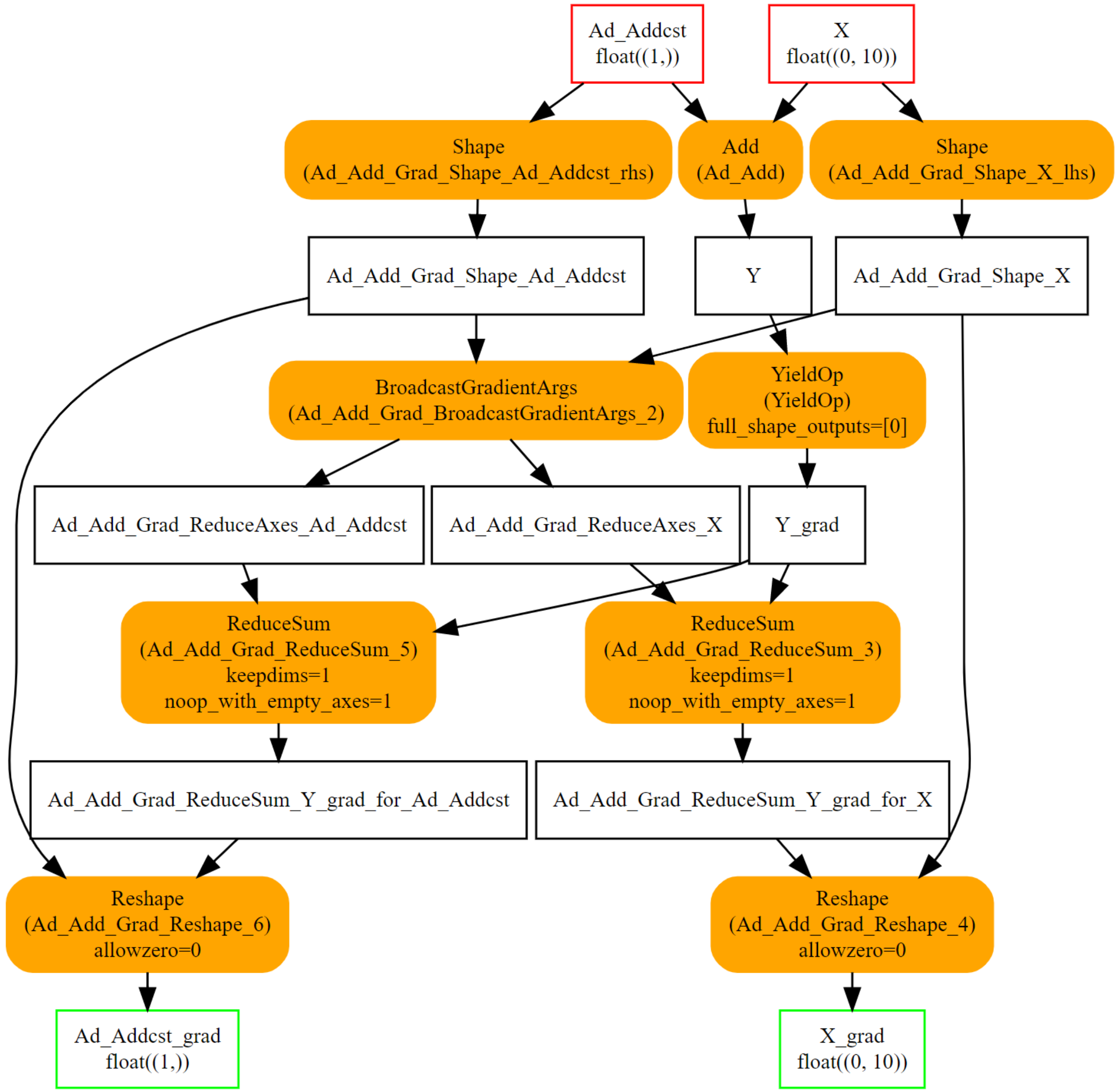
- Onnxruntime computes the ONNX graph of the gradient of another ONNX graph
- Onnxruntime runs forward and backward steps.
- Falls back to torch when onnxruntime does not implement a specific gradient.

```python
class _ORTModuleFunction(torch.autograd.Function):
    """Use a custom torch.autograd.Function to associate self.backward_graph as the
    gradient implementation for self.forward_graph."""

    @staticmethod
    def forward(ctx, *inputs):
        """Performs forward pass based on user input and PyTorch initializer

        Autograd Function's apply() doesn't support keyword arguments,
        so `*inputs` has all the arguments - keyword arguments converted
        to positional/keywords during `TrainingManager.forward`.

        Module outputs are returned to the user
        """
```

```python
@staticmethod
def backward(ctx, *grad_outputs):
    """Performs backward pass based on grad wrt module output"""

    assert ctx.run_info is not None, "forward() or __call__() methods must be called before backward()"
    if self._skip_check.is_set(_SkipCheck.SKIP_CHECK_DEVICE) is False:
        _utils._check_same_device(self._device, "Input argument to backward", *grad_outputs)

    # Unpack saved_tensor to trigger version detection that catches inplace corruption
    _ = ctx.saved_tensors

    # Use IO binding
    # Push user output grads to ONNX backend.
    backward_inputs = C.OrtValueVector()
```

Gradient

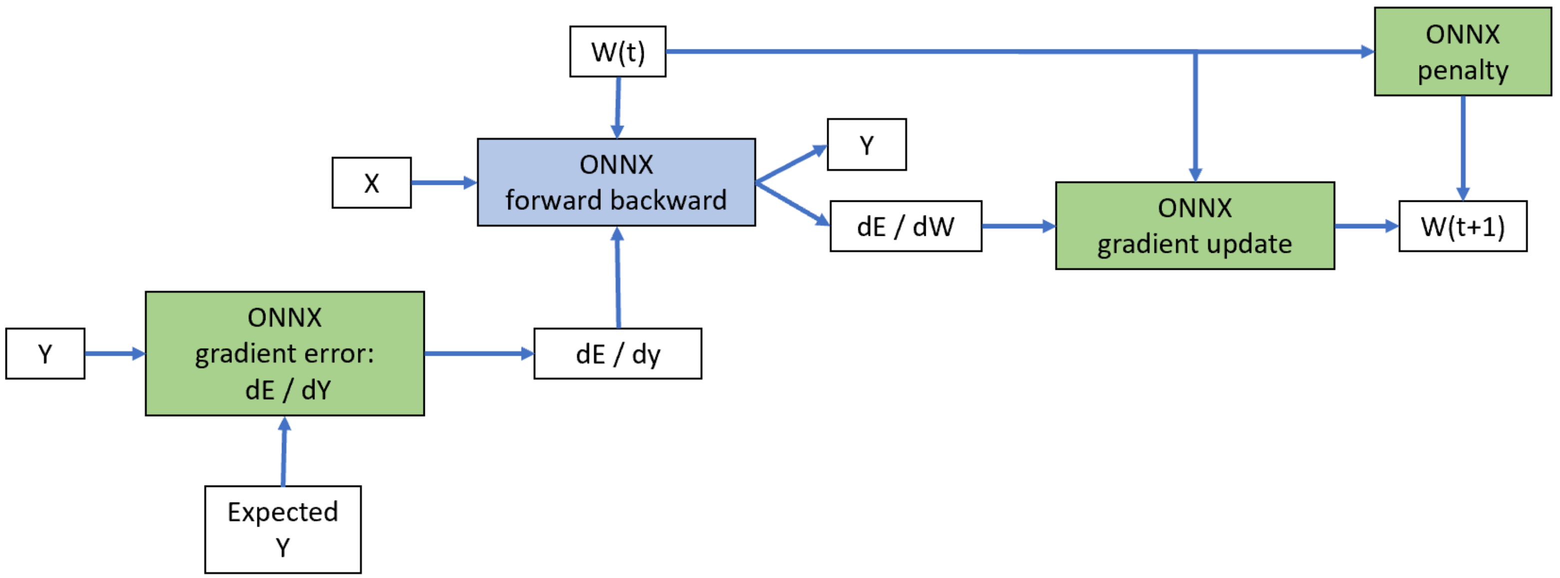
- f(x) = x + a
- ONNX graph for the gradient of f(x) and df/da
- YieldOp: Run until there in forward pass, continue in backward with error information.



onnxruntime-training and scikit-learn?

Design

- onnxruntime-training does not implement training algorithm (yet)
- It only implements functions to compute the gradient and update the weights.
- Neural network could be trained on GPU.



2. API2: scikit-learn template

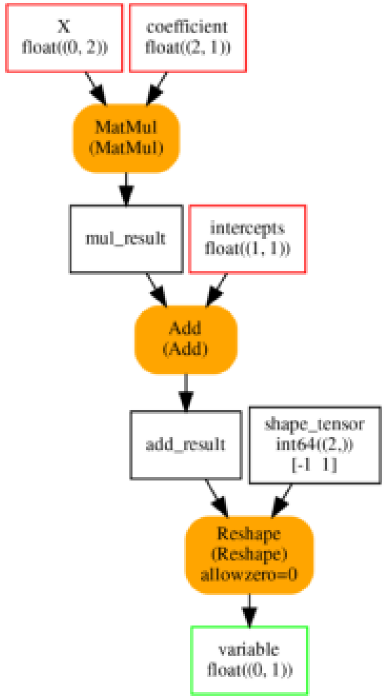
- fit/predict

```
train_session = OrtGradientForwardBackwardOptimizer(
    onx, device='cpu', warm_start=False,
    max_iter=max_iter, batch_size=batch_size,
    learning_loss=NegLogLearningLoss(),
    learning_rate=LearningRateSGDNesterov(
        1e-5, nesterov=True, momentum=0.9),
    learning_penalty=ElasticLearningPenalty(l1=0, l2
```

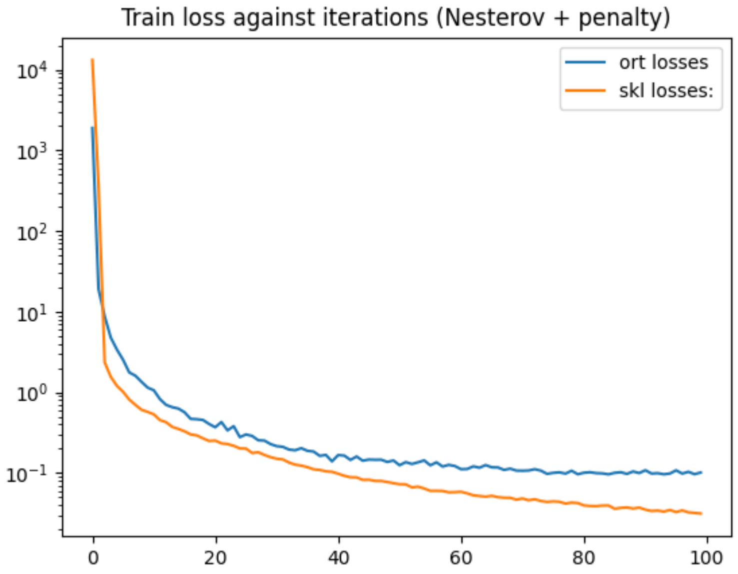
```
train_session .fit(X_train, y_train)
```

POC

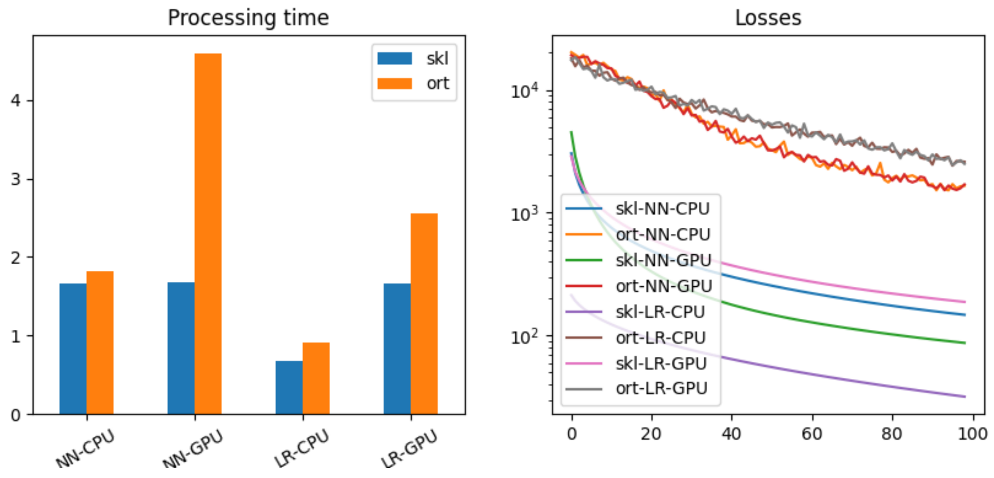
loss



gradient + loss



Exemple with MLPRegressor




```
batch_size = 15
max_iter = 100

nn = MLPRegressor(hidden_layer_sizes=(50, 10), max_iter=max_iter,
                  solver='sgd', learning_rate_init=5e-5,
                  n_iter_no_change=max_iter * 3, batch_size=batch_size,
                  learning_rate="invscaling",
                  # default values
                  momentum=0.9, nesterovs_momentum=True, power_t=0.5)

with warnings.catch_warnings():
    warnings.simplefilter('ignore')
    nn.fit(X_train, y_train)
```

Conversion to ONNX

```
from onnxcustom.utils.onnx_helper import onnx_rename_weights
onx = to_onnx(nn, X_train[:1].astype(numpy.float32), target_opset=15)
onx = onnx_rename_weights(onx)
```

```
train_session = OrtGradientForwardBackwardOptimizer(
    onx, device='cpu', learning_rate=5e-5,
    warm_start=False, max_iter=max_iter, batch_size=batch_size)
```

```
train_session.fit(X_train, y_train)
```

POC

- Almost on par with scikit-learn
- Still needs improvements
- C++ training API for onnxruntime is being developped

Write custom ONNX functions

Why?

- FunctionTransformer can be automatically converted into ONNX
- Training requires custom loss functions
- ONNX Python API is very verbose and slow down the development of simple functions

Many choices

- A more simple API to ONNX
- An API close to numpy
- Write the function with pytorch
- Implement a compiler for a new syntax to define ONNX graphs

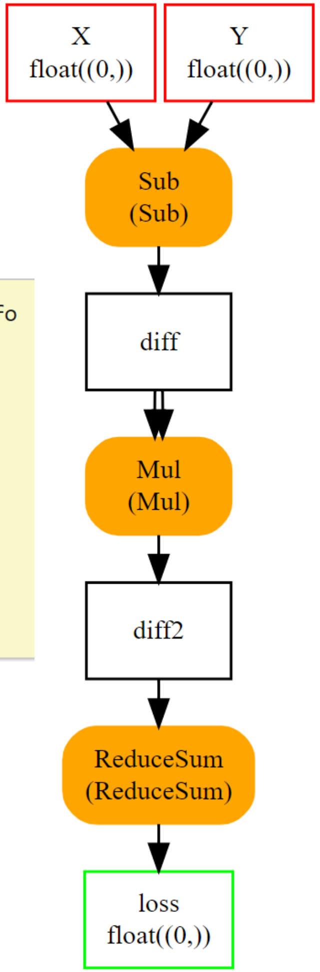
Work still in progress.

Square loss example with ONNX

ONNX API is more verbose than numpy and skl2onnx.

```
from onnx.helper import make_node, make_graph, make_model, make_tensor_value_info
from onnx import TensorProto

nodes = [make_node('Sub', ['X', 'Y'], ['diff']),
         make_node('Mul', ['diff', 'diff'], ['diff2']),
         make_node('ReduceSum', ['diff2'], ['loss'])]

graph = make_graph(nodes, 'square_loss',
                   [make_tensor_value_info('X', TensorProto.FLOAT, [None]),
                    make_tensor_value_info('Y', TensorProto.FLOAT, [None])],
                   [make_tensor_value_info('loss', TensorProto.FLOAT, [None])])
model = make_model(graph)
```

```
sess = InferenceSession(model.SerializeToString())
sess.run(None, {'X': x, 'Y': y})
```



Implementation with numpy

```
def square_loss(X, Y):
    return numpy.sum((X - Y) ** 2, keepdims=1)


x = numpy.array([0, 1, 2], dtype=numpy.float32)
y = numpy.array([0.5, 1, 2.5], dtype=numpy.float32)
square_loss(x, y)
```

Implementation with skl2onnx

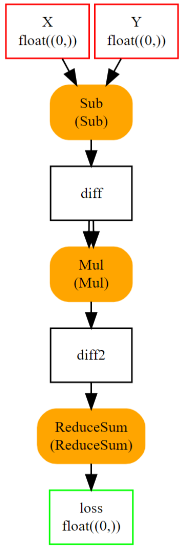
```
from skl2onnx.algebra.onnx_ops import OnnxSub, OnnxMul, OnnxReduceSum

diff = OnnxSub('X', 'Y')
nodes = OnnxReduceSum(OnnxMul(diff, diff))
model = nodes.to_onnx({'X': x, 'Y': y})
```

```
sess = InferenceSession(model.SerializeToString())
sess.run(None, {'X': x, 'Y': y})
```

import onnx_numpy_api as npnx

- A decorator:
  - runs the code to build the ONNX,
  - creates a InferenceSession
  - replaces the function by a call to onnxruntime

- But test and loops are difficult to translate nicely.



Implementation with numpy

```
def square_loss(X, Y):
    return numpy.sum((X - Y) ** 2, keepdims=1)


x = numpy.array([0, 1, 2], dtype=numpy.float32)
y = numpy.array([0.5, 1, 2.5], dtype=numpy.float32)
square_loss(x, y)
```

Implementation with numpy API

```
@onnxnumpy_np(runtime='onnxruntime',
              signature=NDArrayType(("T:all", "T"), dtypes_out=('T',)))
def onnx_square_loss(X, Y):
    return npnx.sum((X - Y) ** 2, keepdims=1)

onnx_square_loss(x, y)
```

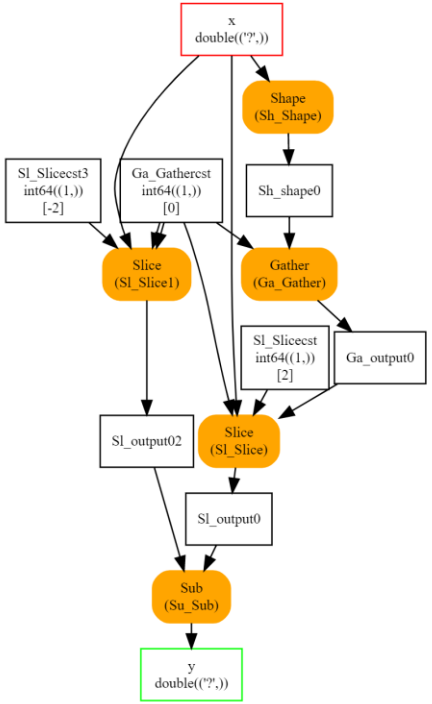
```
array([0.5], dtype=float32)
```

```
onx = onnx_square_loss.to_onnx(key=numpy.float64)
```

Indices and ONNX… not easy!

- Simple function: compute lagged series
- Indices are easy with numpy
- And really not obvious with ONNX

```
opset: domain='' version=15
input: name='x' type=dtype('float64') shape=()
init: name='Sl_Slicecst' type=dtype('int64') shape=(1,) -- array([2], dtype=int64)
init: name='Ga_Gathercst' type=dtype('int64') shape=(1,) -- array([0], dtype=int64)
init: name='Sl_Slicecst3' type=dtype('int64') shape=(1,) -- array([-2], dtype=int64)
Shape(x) -> Sh_shape0
  Gather(Sh_shape0, Ga_Gathercst) -> Ga_output0
    Slice(x, Sl_Slicecst, Ga_output0, Ga_Gathercst) -> Sl_output0
Slice(x, Ga_Gathercst, Sl_Slicecst3, Ga_Gathercst) -> Sl_output02
  Sub(Sl_output0, Sl_output02) -> y
output: name='y' type=dtype('float64') shape=()
```




```
@onnxnumpy_np(runtime='onnxruntime',
              signature=NDArrayType(("T:all", ), dtypes_out=('T',)))
def lagged(x, lag=2):
    return x[lag:] - x[:-lag]

x = numpy.array([[0, 1], [2, 3], [4, 5], [10, 21]])
lagged(x)
```

```
array([[ 4,  4],
       [ 8, 18]], dtype=int32)
```

Many choices

- A more simple API to ONNX
- An API close to numpy
- Write the function with pytorch
- **Implement a compiler for a new syntax to define ONNX graphs**

Many next time.

With that tool, onnxruntime could be used instead numpy.

Conclusion

ONNX ecosystem is growing.