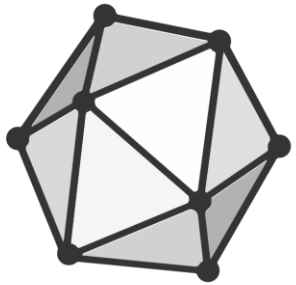


ONNX

Overview

Xavier Dupré

12/12/2022

Plan

1. **ONNX and its ecosystem**: anything you need to know
   onnx, onnxruntime, skl2onnx, tf2onnx, torch.onnx, onnxmltools
2. **onnxruntime: the main ONNX runtime**, providers, training
   optimization, onnxruntime-training, ORTModule
3. Interesting topic along the way
   1. Quantization, int8, uint8, float8, …
   2. Parallelization (deepspeed, …)
   3. Customization (custom operator, custom kernel)
   4. Float, double and trees
   5. Fusion, AITemplate, Triton, TVM, OpenVino, TensorRT
   6. What about Einsum?

A little about me

- PhD in artificial intelligence before the deep learning (2004)
- Teacher at the ENSAE since 2001 (programming, machine learning)
- A2iA, Yahoo (query rewriting), Microsoft (Bing Local, ONNX)
- Hackathon

ONNX and its ecosystem

- ONNX objective
- ONNX is a language
- ONNX + runtime
- Model conversion → ONNX → runtime
- Results

ONNX objectives

Before

- Multiple framework to train a model
- Deployment must handle many dependencies
- Stack deprecates fast
- Difficulty to version

After

- A common way to describe a model
- One technology to run a model
- Deployment is lighter
- Old models still work with new versions
- Metadata can be added to the model
- Training and inference do not depend from each others

ONNX is a language

[ONNX operators]

- A common set to describe every machine learned model
- Every operator is versioned
- It supports float, double, float16, bfloat16, int64, …
- It covers all necessary operators to describe a numerical models (still lacking with strings)

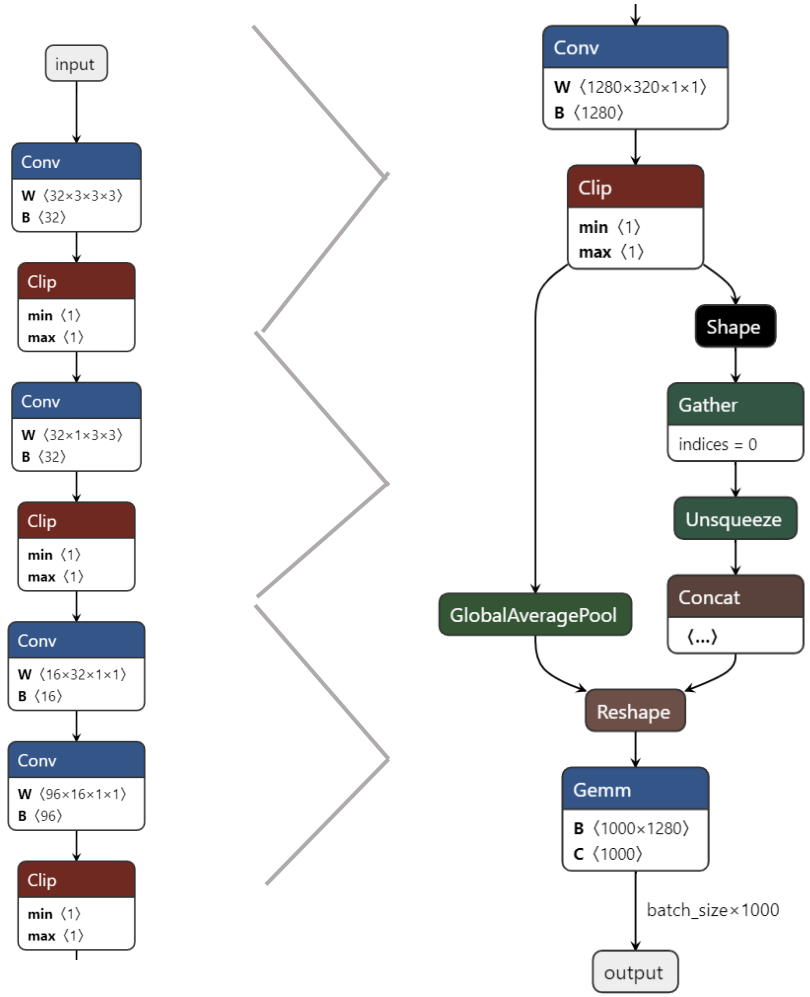
ai.onnx | ai.onnx.ml | ai.onnx.preview.training

operator	versions	differences
Abs	13, 6, 1	13/6, 13/1, 6/1
Acos	7	
Acosh	9	
Add	14, 13, 7, 6, 1	14/13, 14/7, 13/7, 14/6, 13/6, 7/6, 14/1, 13/1, 7/1, 6/1
And	7, 1	7/1
ArgMax	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1, 12/1, 11/1
ArgMin	13, 12, 11, 1	13/12, 13/11, 12/11, 13/1, 12/1, 11/1
Asin	7	
Asinh	9	

Example

Mobilenetv2 from ONNX Model Zoo,
Opened with Netron (screeshots)

- Easy to follow what the model does.



ONNX model can easily be evaluated

ReferenceEvaluator (released soon)

However this runtime is based on python + numpy and slow.

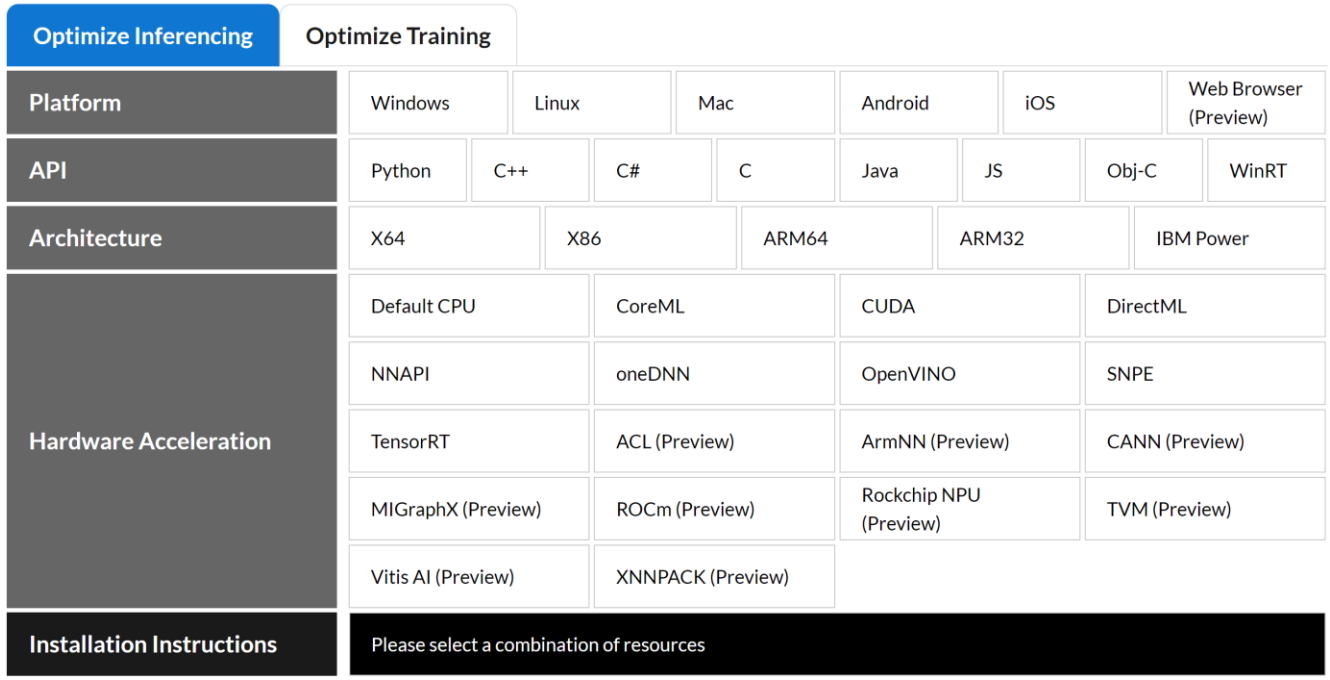
It is meant for debugging purposes.

```
import numpy as np
from onnx.reference import ReferenceEvaluator

X = np.array(...)
sess = ReferenceEvaluator("model.onnx")
results = sess.run(None, {"X": X})
print(results[0])
```

Faster runtime: onnxruntime

onnxruntime is a production runtime.

- Faster than pytorch / tensorflow
- Faster than scikit-learn / lightgbm / xgboost
- Available on many platforms
- Enable thirs party accelerators (TVM, TensorRT, …)



Example in python

onnxruntime

CPU

```
from onnxruntime import InferenceSession

sess = InferenceSession("linreg_model.onnx")
result = sess.run(None, {'X': X_test[:5]})
print(result)
```

CUDA

```
sess = Inference("linreg_model.onnx",
                 providers=["CUDAExecutionProviders"])
```

CUDA or default to CPU if no implementation is available for CUDA

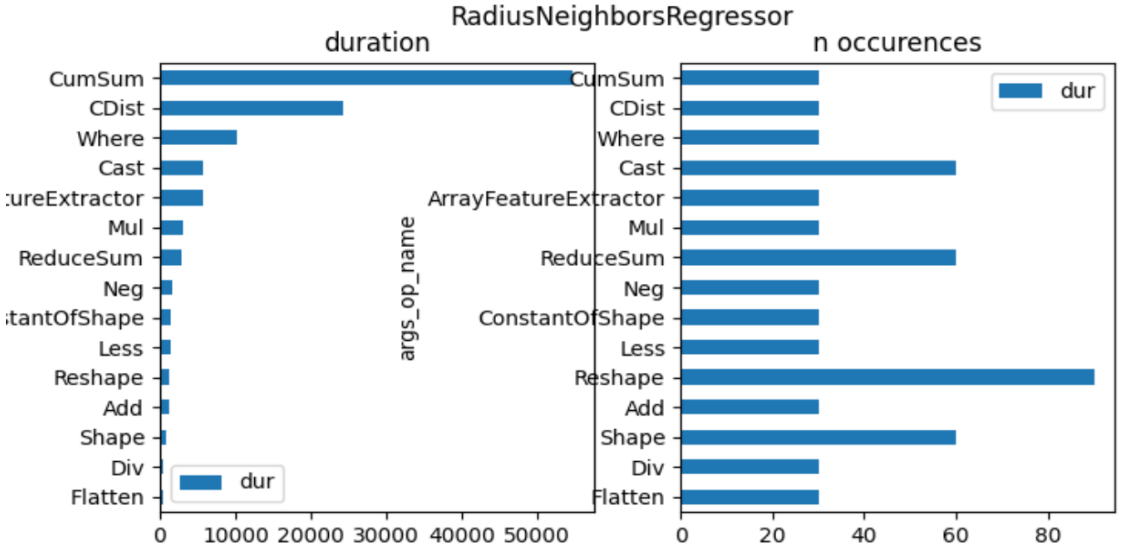
```
sess = Inference("linreg_model.onnx",
                 providers=["CUDAExecutionProviders",
                            "CPUExecutionProviders"])
```

Profiling

Finds which operator slows the model down.

- Change the implementation
- Change the model
- Quantize
- …

```
so = SessionOptions()
so.enable_profiling = True
sess = InferenceSession(onx.SerializeToString(), so,
                        providers=['CPUExecutionProvider'])
feeds = {'X': X[:100]}

for i in tqdm(range(0, 10)):
    sess.run(None, feeds)

prof = sess.end_profiling()
print(prof)
```



Create an ONNX model from scratch

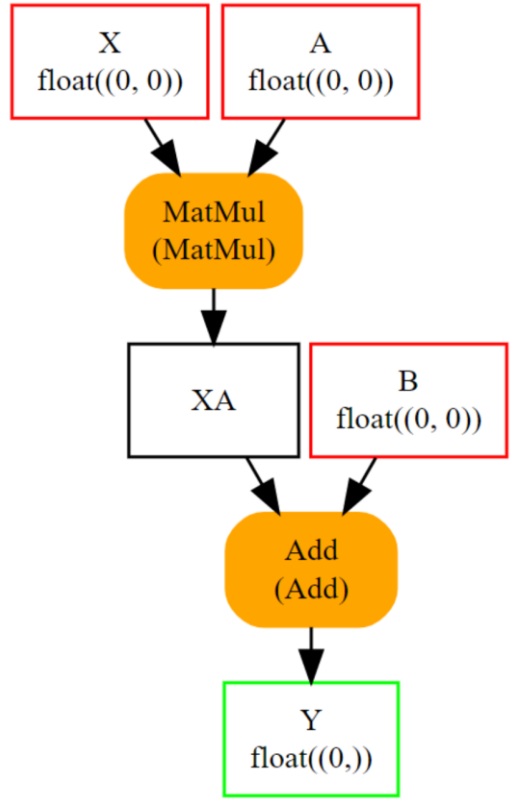
[Introduction to ONNX]

Ok…

But I already created the model with torch or scikit-learn.

Oh sorry…

```
import numpy
from onnx import numpy_helper, TensorProto
from onnx.helper import (
    make_model, make_node, set_model_props, make_tensor,
    make_graph, make_tensor_value_info)
from onnx.checker import check_model

# inputs

# 'X' is the name, TensorProto.FLOAT the type, [None, None] the shape
X = make_tensor_value_info('X', TensorProto.FLOAT, [None, None])
A = make_tensor_value_info('A', TensorProto.FLOAT, [None, None])
B = make_tensor_value_info('B', TensorProto.FLOAT, [None, None])

# outputs, the shape is left undefined

Y = make_tensor_value_info('Y', TensorProto.FLOAT, [None])

# nodes

# It creates a node defined by the operator type MatMul,
# 'X', 'A' are the inputs of the node, 'XA' the output.
node1 = make_node('MatMul', ['X', 'A'], ['XA'])
node2 = make_node('Add', ['XA', 'B'], ['Y'])

# from nodes to graph
# the graph is built from the list of nodes, the list of inputs,
# the list of outputs and a name.

graph = make_graph([node1, node2],  # nodes
                   'lr',  # a name
                   [X, A, B],  # inputs
                   [Y])  # outputs

# onnx graph
# there is no metata in this case.

onnx_model = make_model(graph)
```



Convert a model into ONNX

A converter takes an existing model and translates it with ONNX operators.

- sklearn-onnx
- torch.onnx
- tf2onnx
- onnxmltools (lightgbm, xgboost, sparkml)

```
X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y)

# Train classifiers
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})
```

A runtime usually supports all onnx operators up to a certain version: target_opset forces the converter to use operators only available for that version

Torch

[torch.onnx]

This approach is very similar to what [pytorch 2.0] offers.

```
import torch
import torchvision

dummy_input = torch.randn(10, 3, 224, 224, device="cuda")
model = torchvision.models.alexnet(pretrained=True).cuda()
```

```
input_names = [ "actual_input_1" ] + [ "learned_%d" % i for i in range(16) ]
output_names = [ "output1" ]

torch.onnx.export(model, dummy_input, "alexnet.onnx", verbose=True, input_names=input_names
output_names=output_names)
```

Torch 1.x: eager evaluation

onnxruntime / torch 2.0: graph evaluation

```
import torch
import torchvision.models as models

model = models.resnet18().cuda()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
compiled_model = torch.compile(model)

x = torch.randn(16, 3, 224, 224).cuda()
optimizer.zero_grad()
out = compiled_model(x)
out.sum().backward()
optimizer.step()
```

Scikit-learn

sklearn-onnx

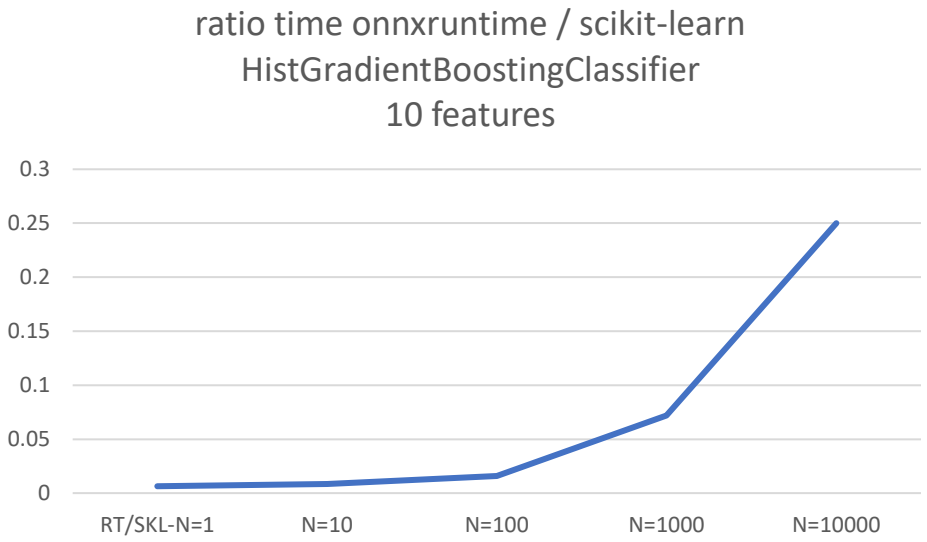
- Supports most of the transformers / predictors enabling inference (no OPTICS)
- Supports pipelines
- Supports custom transformer or prediction (if the user is willing to write the corresponding converter, see Implement a new converter

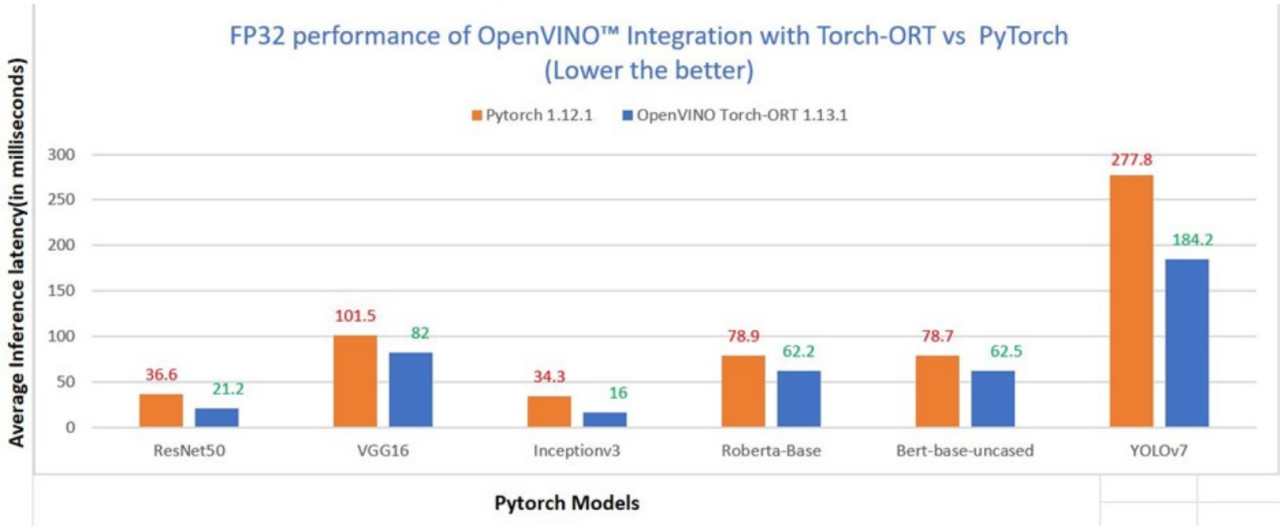
```python
# Train a model.
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y)
clr = RandomForestClassifier()
clr.fit(X_train, y_train)

# Convert into ONNX format
from skl2onnx import convert_sklearn
from skl2onnx.common.data_types import FloatTensorType
initial_type = [('float_input', FloatTensorType([None, 4]))]
onx = convert_sklearn(clr, initial_types=initial_type)
with open("rf_iris.onnx", "wb") as f:
    f.write(onx.SerializeToString())

# Compute the prediction with ONNX Runtime
import onnxruntime as rt
import numpy
sess = rt.InferenceSession("rf_iris.onnx")
input_name = sess.get_inputs()[0].name
label_name = sess.get_outputs()[0].name
pred_onx = sess.run([label_name], {input_name: X_test.astype(numpy.float32)})[0]
```

Results

Pytorch

scikit-learn is not optimized for one-off prediction,
onnxruntime is way faster.





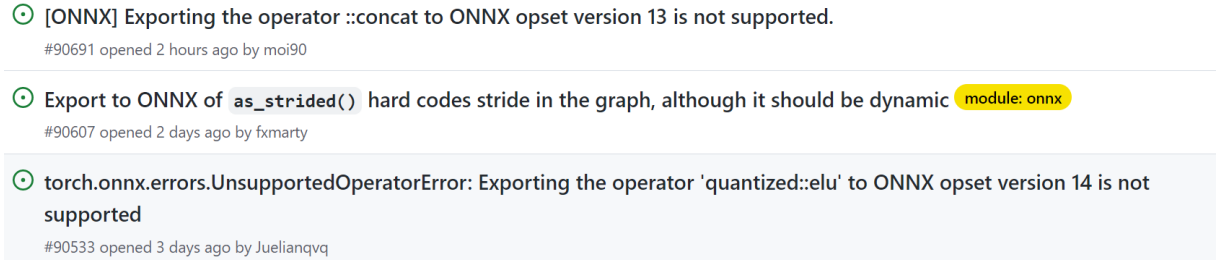
[Faster inference for PyTorch models with OpenVINO Integration with Torch-ORT]

[source]

Issues

What if the conversion does not work?

- No standard procedure to handle strings: hard to define what high level operators would be useful
- Custom model requires to write onnx from scratch…
- Did you say discrepancies?



Any questions?

onnxruntime

- onnxruntime
  - Providers, Optimizations, …
- onnxruntime-extensions
  - strings…
- onnxruntime-training
  - Torch, C, others?

Providers

Operator = mathematical function

Kernel = implementation for this function on a specific architecture

Provider = set of kernels for a specific architecture

main ▾ onnxruntime / onnxruntime / core / providers / armnn / **math** /

wejoncy code clean (#12392) ...	
..	
gemm.cc	Add ArmNN Execution Provider (#3714)
gemm.h	code clean (#12392)

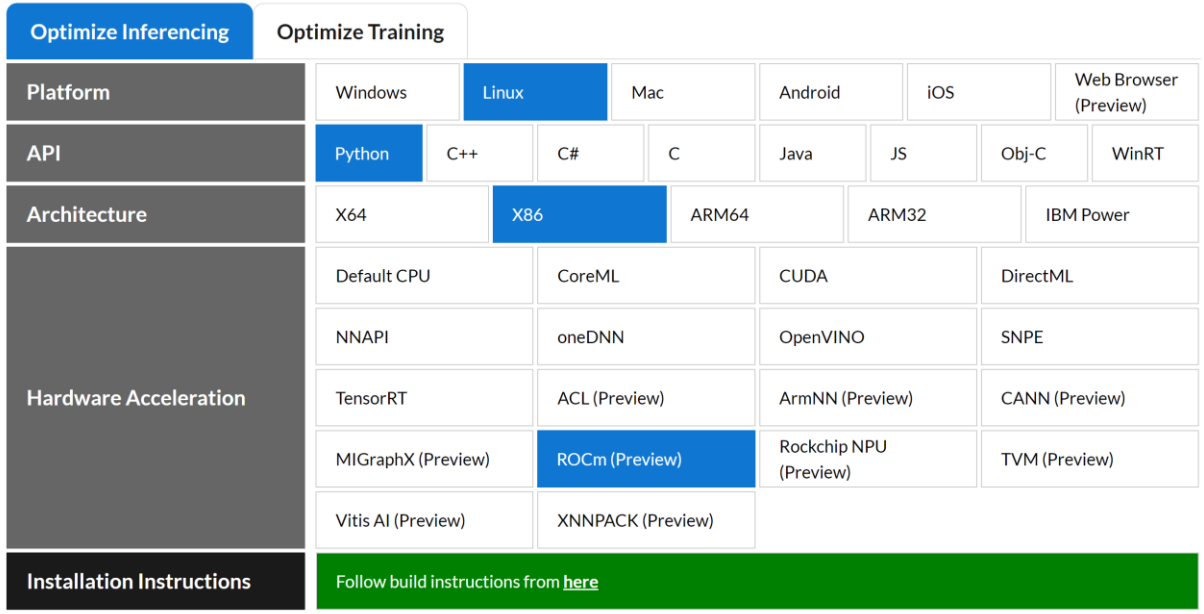
- acl
- armnn
- cann
- coreml
- cpu
- cuda
- dml
- dnnl
- migraphx
- nnapi
- openvino
- rknpu
- rocm
- shared
- shared_library
- snpe
- tensorrt
- tvm
- vitisai
- winml
- xnnpack

```python
import pprint
import onnxruntime
print("all providers")
pprint.pprint(onnxruntime.get_all_providers())
print("available providers")
pprint.pprint(onnxruntime.get_available_providers())
```

```
all providers
['TensorrtExecutionProvider',
 'CUDAExecutionProvider',
 'MIGraphXExecutionProvider',
 'ROCMExecutionProvider',
 'OpenVINOExecutionProvider',
 'DnnlExecutionProvider',
 'TvmExecutionProvider',
 'VitisAIExecutionProvider',
 'NnapiExecutionProvider',
 'CoreMLExecutionProvider',
 'ArmNNExecutionProvider',
 'ACLExecutionProvider',
 'DmlExecutionProvider',
 'RknpuExecutionProvider',
 'XnnpackExecutionProvider',
 'CANNExecutionProvider',
 'CPUExecutionProvider']
available providers
['CPUExecutionProvider']
```

Providers

May require to compile



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			
	NNAPI	oneDNN	OpenVINO	SNPE			
	TensorRT	ACL (Preview)	ArmNN (Preview)	CANN (Preview)			
	MIGraphX (Preview)	ROCm (Preview)	Rockchip NPU (Preview)	TVM (Preview)			
	Vitis AI (Preview)	XNNPACK (Preview)					
Installation Instructions	Follow build instructions from here						

export CUDA_VERSION=11.3
export CUDACXX=/usr/local/cuda-11.3/bin/nvcc

python ./tools/ci_build/build.py --config Release --skip_tests --build_wheel --build_dir ./build/linux_cuda --build_shared_lib --use_cuda --cuda_home /usr/local/cuda-${CUDA_VERSION}/ --cudnn_home /usr/local/cuda-${CUDA_VERSION}/ --cuda_version=${CUDA_VERSION} --enable_training --enable_training_ops --enable_training_torch_interop --parallel 4

Graph optimization: fusion

MatMul: A B is usually faster when A is transposed first. So A'B is faster with FusedMatMul: transpose + matmul).

FusedConv: avoids allocating too much intermediate memory.

Profiling helps deciding which sequence of operators should be fused.

FusedConv - 1 (com.microsoft)

The fused convolution operator schema is the same as Conv besides it includes an attribute activation.

FusedMatMul - 1 (com.microsoft)

- **alpha**: Scalar multiplier for the product of the input tensors. Default value is ?.
- **transA**: Whether A should be transposed on the last two dimensions before doing multiplication Default value is ?.
- **transB**: Whether B should be transposed on the last two dimensions before doing multiplication Default value is ?.
- **transBatchA**: Whether A should be transposed on the 1st dimension and batch dimensions (dim-1 to dim-rank-2) before doing multiplication Default value is ?.
- **transBatchB**: Whether B should be transposed on the 1st dimension and batch dimensions (dim-1 to dim-rank-2) before doing multiplication Default value is ?.

```python
import onnxruntime as rt

sess_options = rt.SessionOptions()

# Set graph optimization level
sess_options.graph_optimization_level = rt.GraphOptimizationLevel.ORT_ENABLE_EXTENDED

# To enable model serialization after graph optimization set this
sess_options.optimized_model_filepath = "<model_output_path\optimized_model.onnx>"

session = rt.InferenceSession("<model_path>", sess_options)
```

Graph optimization

Always a work in progress. It also depends on the hardware.

- Constant Folding: Statically computes parts of the graph that rely only on constant initializers. This eliminates the need to compute them during runtime.
- Redundant node eliminations: Remove all redundant nodes without changing the graph structure. The following such optimizations are currently supported:
  - Identity Elimination
  - Slice Elimination
  - Unsqueeze Elimination
  - Dropout Elimination
- Semantics-preserving node fusions : Fuse/fold multiple nodes into a single node. For example, Conv Add fusion folds the Add operator as the bias of the Conv operator. The following such optimizations are currently supported:
  - Conv Add Fusion
  - Conv Mul Fusion
  - Conv BatchNorm Fusion
  - Relu Clip Fusion
  - Reshape Fusion

Implementation details

CUDA

onnxruntime relies mostly on CUDA libraries but not always (see topk_impl.cuh)

CPU

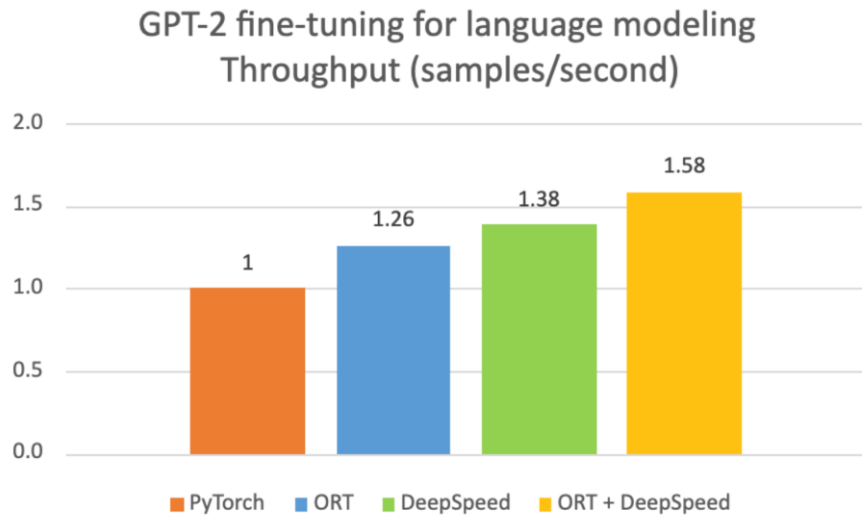
onnxruntime uses eigen, implements its own BLAS libraries, parallelizes intra/inter kernels, optimizes memory management.

Training with onnxruntime

ORTModule wraps a model. Everything else is the same.

First iteration: ORTModule converts into ONNX and computes the gradient. Uses pytorch if the conversion fails.

source




```
cls_net = build_class_model()
nn = cls_net(n_features, hidden_layer_sizes, 1)
```

```
if use_ortmodule:
    nn = ORTModule(nn)
```

```
criterion = torch.nn.MSELoss(reduction='sum')
optimizer = torch.optim.SGD(model.parameters(), lr=1e-6)
for t in range(2000):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = nn(x)

    # Compute and print loss
    loss = criterion(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Training: Python, C API, no pytorch

```
ORT_RETURN_IF_ERROR(runner->Run(training_data_loader.get(), test_data_loader.get(), mapped_dimensions));
```

source

Demo on android

```
fun performTraining(view: View) {
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val trainDataSetCacheDir = copyFileOrDir("train_data")


    // Example of a call to a native method
    binding.durationText.text = trainingResource?.let { train(it, trainDataSetCacheDir) }

    trainingStep++
    binding.trainingProgressBar.setProgress((trainingStep * 100) / totalTrainingSteps)
    if (trainingStep < totalTrainingSteps) {
        val trainingStatus = "Training in Progress " + ((trainingStep * 100) / totalTrainingSteps).toString() + "%."
        binding.sampleText.text = trainingStatus
    }
    else {
        trainingStep = totalTrainingSteps
        val trainingStatus = "Training Complete."
        binding.sampleText.text = trainingStatus
    }
}
```

source

```
# Training Loop :
def train(epoch):
    model.train()
    losses = []
    for batch_idx, (data, target) in enumerate(train_loader):
        forward_inputs = [data.reshape(len(data),784).numpy(),target.numpy().astype(np.int32)]
        train_loss, _ = model(forward_inputs)
        optimizer.step()
        model.reset_grad()
        losses.append(train_loss)

    print(f'Epoch: {epoch+1},Train Loss: {sum(losses)/len(losses):.4f}')
```

Training: scikit-learn API

scikit-learn is very popular because it is easy to use.

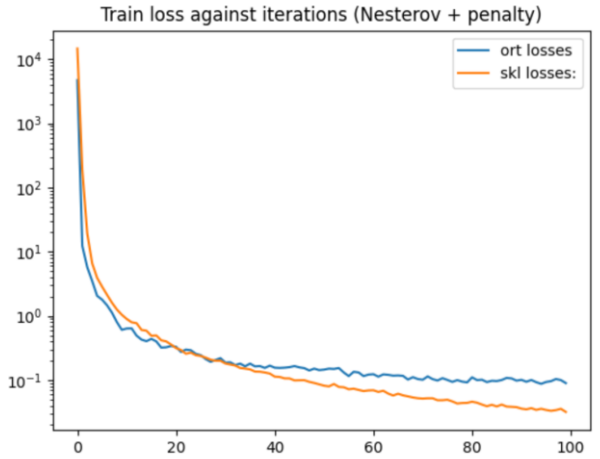
How about training a scikit-learn neural network on CUDA?

Work in progress.

It is worth doing it for big models.

```
train_session = OrtGradientForwardBackwardOptimizer(
    onx, ['coef', 'intercept'], device='CUDA',
    learning_rate=LearningRateSGDNesterov()
    learning_loss=ElasticLearningLoss(l1_weight=0.1, l2_weight=0.9),
    learning_penalty=ElasticLearningPenalty(l1=0.1, l2=0.9))
```

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

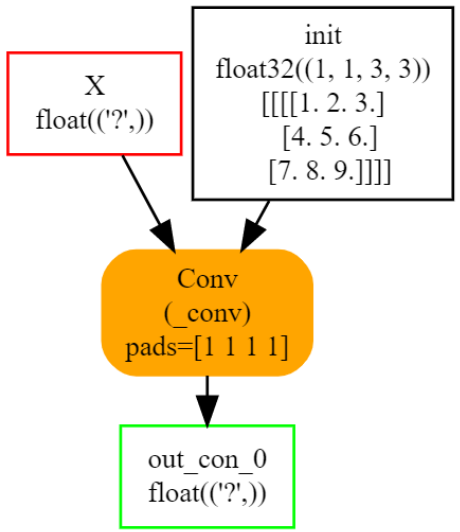


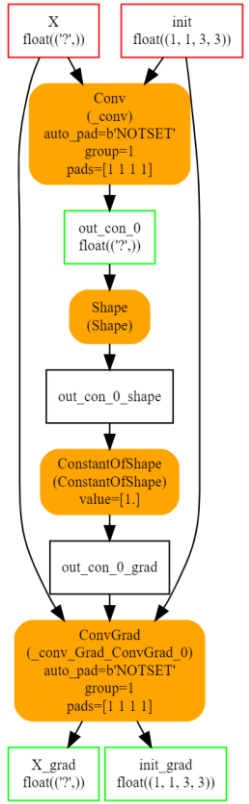
Training: Gradient

onnxruntime can also return the gradient as an ONNX graph and let you build you own training algorithm.

```
export_gradient_graph(model, loss_fn, example_input, example_labels, gradient_graph_path)
```

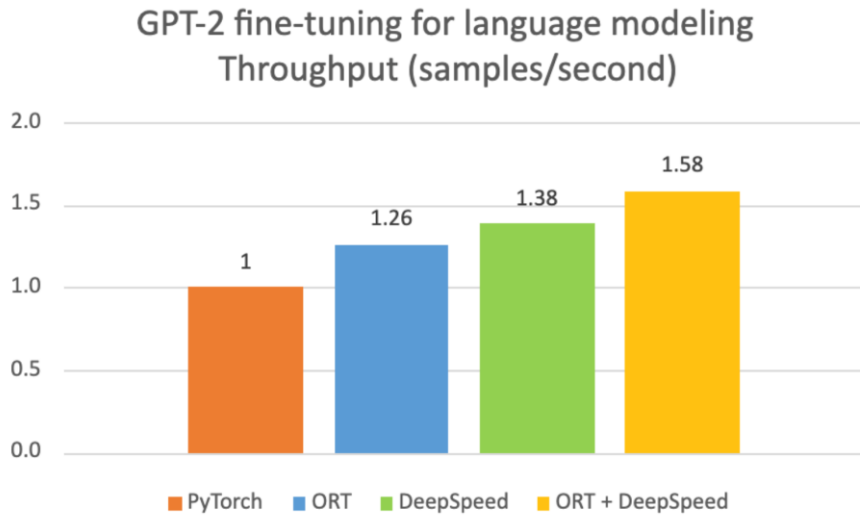
```
grad = onnx_derivative(onx,
```





onnxruntime + deepspeed

- onnxruntime = runtime optimized on one machine
- deepspeed: parallelization of computation with pytorch
- Stage 1, 2 compatible with onnxruntime
- Stage 3: work in progress



Issues

- Performance is not always easy to achieve.
- Python hides implicit operations like memory copy.
- Parallelization does not work well if multiple instances are run in parallel.

Questions?

iobinding support part of inputs #13807

Open jackzhou121 opened this issue 9 days ago · 0 comments

[Performance] getting slower after run 10 times in intel cpu when I loop execute onnxruntime inference #13651

Open allen20200111 opened this issue 25 days ago · 7 comments

[CPUExecutionProvider] PyTorch/Numpy operations following InferenceSession.run() are 50x slower compared to using dummy inputs #13808

Open fxmarty opened this issue 9 days ago · 11 comments

SessionOptions

class onnxruntime.**SessionOptions**(*self: onnxruntime.capi.onnxruntime_pybind11_state.SessionOptions*)

Configuration information for a session.

add_external_initializers(*self: onnxruntime.capi.onnxruntime_pybind11_state.SessionOptions, arg0: list, arg1: list*) → None

add_free_dimension_override_by_denotation(*self: onnxruntime.capi.onnxruntime_pybind11_state.SessionOptions, arg0: str, arg1: int*) → None

Specify the dimension size for each denotation associated with an input's free dimension.

add_free_dimension_override_by_name(*self: onnxruntime.capi.onnxruntime_pybind11_state.SessionOptions, arg0: str, arg1: int*) → None

Specify values of named dimensions within model inputs.

add_initializer(*self: onnxruntime.capi.onnxruntime_pybind11_state.SessionOptions, arg0: str, arg1: object*) → None

add_session_config_entry(*self: onnxruntime.capi.onnxruntime_pybind11_state.SessionOptions, arg0: str, arg1: str*) → None

Set a single session configuration entry as a pair of strings.

property **enable_cpu_mem_arena**

Enables the memory arena on CPU. Arena may pre-allocate memory for future usage. Set this option to false if you don't want it. Default is True.

property **enable_mem_pattern**

Enable the memory pattern optimization. Default is true.

property **enable_mem_reuse**

Enable the memory reuse optimization. Default is true.

property **enable_profiling**

Enable profiling for this session. Default is false.

property **execution_mode**

Sets the execution mode. Default is sequential.

Interesting topic along the way

1. From python to onnx, onnx to python, easier way
2. Quantization, int8, uint8, float8, …
3. Parallelization (CPU, branching, cache story, AVX)
4. Customization (custom operator, custom kernel)
5. Float, double and trees
6. Kernel, Fusion, AITemplate, Triton, TVM, OpenVino, TensorRT

Python to onnx

onnxscript: easier API to create ONNX graphs.

numpy API for onnx?

onnx to python is a lot easier.

```
def export2python(
    model_onnx,
    opset=None,  # pylint: disable=unused-argument
    verbose=True,  # pylint: disable=unused-argument
    name=None,
    rename=False,
    autopep_options=None,
    function_name="main",
    use_operators=False,
    clean_code=True,
    inline_const: bool = False,
):
    """Exports an ONNX model to the *python* syntax.
```

ONNX Script

ONNX Script enables developers to naturally author ONNX functions and models using a subset of Python. ONNX Script is:

- **Expressive:** enables the authoring of all ONNX functions.
- **Simple and concise:** function code is natural and simple.
- **Debuggable:** allows for eager-mode evaluation that provides for a more delightful ONNX model debugging experience.

Note however that ONNX Script does **not** intend to support the entirety of the Python language.

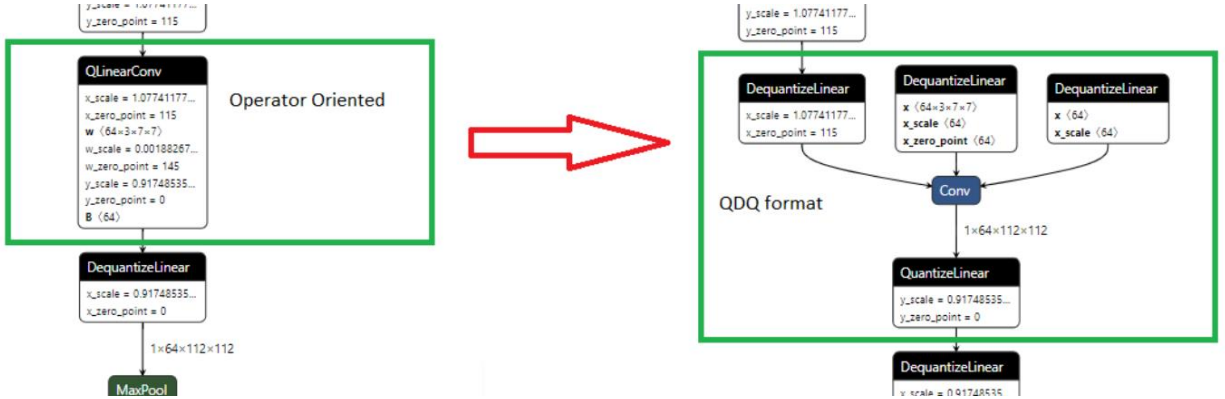
```
@script()
def Selu(
    X,
    alpha: float = 1.6732631921768188476562 5,
    gamma: float = 1.0507010221481323242187 5,
):
    neg = gamma * (alpha * op.Exp(X) - alpha)
    pos = gamma * X
    return op.Where(X <= 0.0, neg, pos)
```

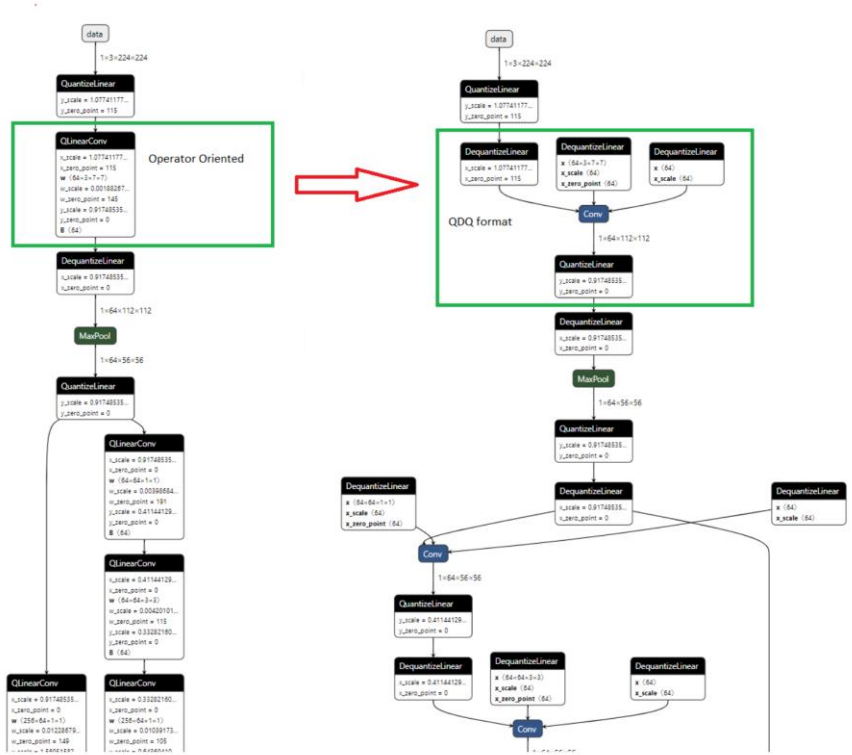
```
from mlprodict.npy import onnxnumpy_np, NDArrayType
import mlprodict.npy.numpy_onnx_impl as npnx

@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)
```

Quantization

Reduce model size.

How to avoid performance loss?



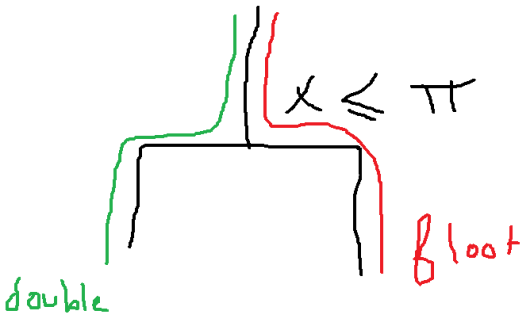


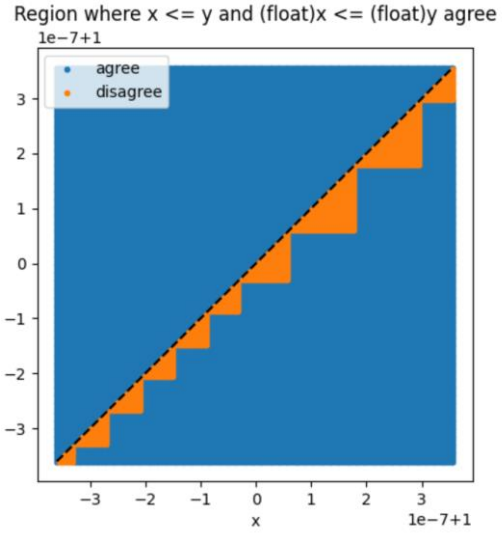
Float and double?

Don’t use doubles if the tree is implemented with floats ok with ligthgbm is implemented with double.

Output = sum_i(tree_i(X)) … but float(X) = X + epsilon

Then float(output(X)) = sum_i (float(tree_i(X)) + sum_i(epsilon_i)





TreeEnsembleRegressor - 1 vs 3

Next section compares an older to a newer version of the same operator after both definition are converted into markdown text. Green means an addition to the newer version, red means a deletion. Anything else is unchanged.

Files changed (1)

TreeEnsembleRegressor1 → TreeEnsembleRegressor3 +0 -12

TreeEnsembleRegressor1 → TreeEnsembleRegressor3 RENAMED

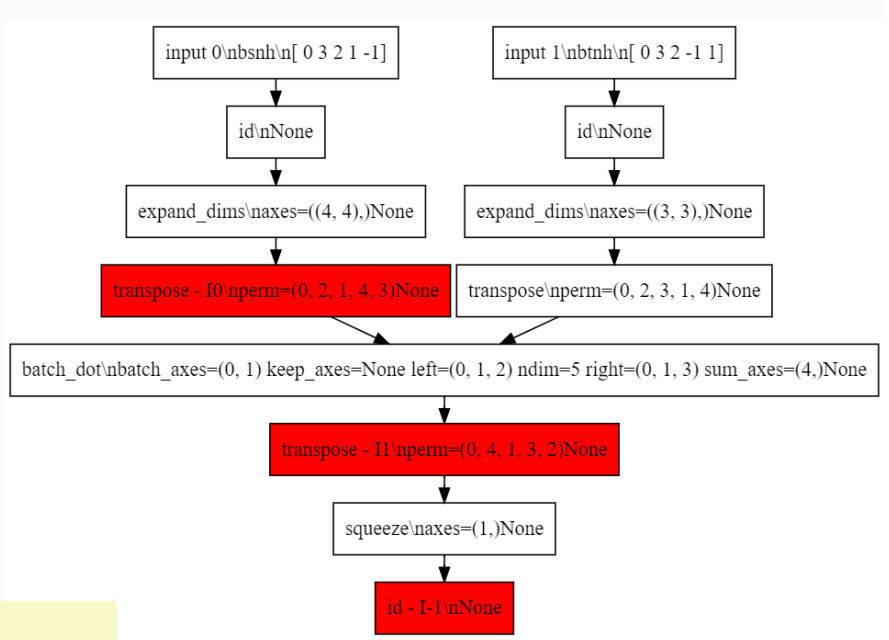
```
@@ -1 +1 @@
Tree Ensemble regressor.  Returns the regressed values for each input in N.
All args with nodes_ are fields of a tuple of tree nodes, and
it is assumed they are the same length, and an index i will decode the
tuple across these inputs.  Each node id can appear only once
for each tree id.
All fields prefixed with target_ are tuples of votes at the leaves.
A leaf may have multiple votes, where each vote is weighted by
the associated target_weights index.
- All fields ending with <i>_as_tensor</i> can be used instead of the
- same parameter without the suffix if the element type is double and not float.
All trees must have their node ids start at 0 and increment by 1.
Mode enum is BRANCH_LEQ, BRANCH_LT, BRANCH_GTE, BRANCH_GT, BRANCH_EQ, BRANCH_NEQ, LEAF
**Attributes**
* **aggregate_function**:
  Defines how to aggregate leaf values within a target. <br>One of
  'AVERAGE,' 'SUM,' 'MIN,' 'MAX.'
* **base_values**:
  Base values for classification, added to final class score; the size
  must be the same as the classes or can be left unassigned (assumed
  0)
- * **base_values_as_tensor**:
-   Base values for classification, added to final class score; the size
-   must be the same as the classes or can be left unassigned (assumed
-   0)
* **n_targets**:
  The total number of targets.
* **nodes falsenodeids**:
```

Einsum and graph optimization

Operator einsum needs to be decomposed into a series of transposition and matrix multiplications.

- Many ways to decompose.
- Decomposition helps graph optimization.

bsnh,btnh->bnts



bac,cd,def=ebc

```
res = numpy.zeros((2, 2, 2))
for a in range(0, 2):
    for b in range(0, 2):
        for c in range(0, 2):
            for d in range(0, 2):
                for e in range(0, 2):
                    for f in range(0, 2):
                        res[e, b, c] += m1[b, a, c] * m2[c, d] * m3[d, e, f]
res
```

```
4.5 mlbest='bsnh,btnh->bnts':   0%|          | 0/121 [00:00<?, ?it/s]
4.5 mlbest='bsnh,btnh->bnts':   1%|          | 1/121 [00:00<01:07,  1.77it/s]
4.5 mlbest='bnth,bsth->btsn':   1%|          | 1/121 [00:00<01:07,  1.77it/s]
4.5 mlbest='bnth,bsth->btsn':   4%|▍         | 5/121 [00:00<00:12,  9.38it/s]
4.5 mlbest='bnth,bsth->btsn':   7%|▋         | 9/121 [00:00<00:06, 16.03it/s]
4.5 mlbest='bnht,bsht->bhsn':   7%|▋         | 9/121 [00:00<00:06, 16.03it/s]
4.5 mlbest='bnht,bsht->bhsn':  11%|▉         | 13/121 [00:00<00:04, 21.60it/s]
4.5 mlbest='bhtn,bstn->btsh':  11%|▉         | 13/121 [00:00<00:04, 21.60it/s]
4.5 mlbest='bhtn,bstn->btsh':  14%|█▍        | 17/121 [00:00<00:04, 25.68it/s]
4.5 mlbest='bhts,bnts->btnh':  14%|█▍        | 17/121 [00:01<00:04, 25.68it/s]
4.5 mlbest='bhts,bnts->btnh':  17%|█▋        | 21/121 [00:01<00:03, 29.15it/s]
4.5 mlbest='bhts,bnts->btnh':  21%|██        | 25/121 [00:01<00:03, 31.85it/s]
```

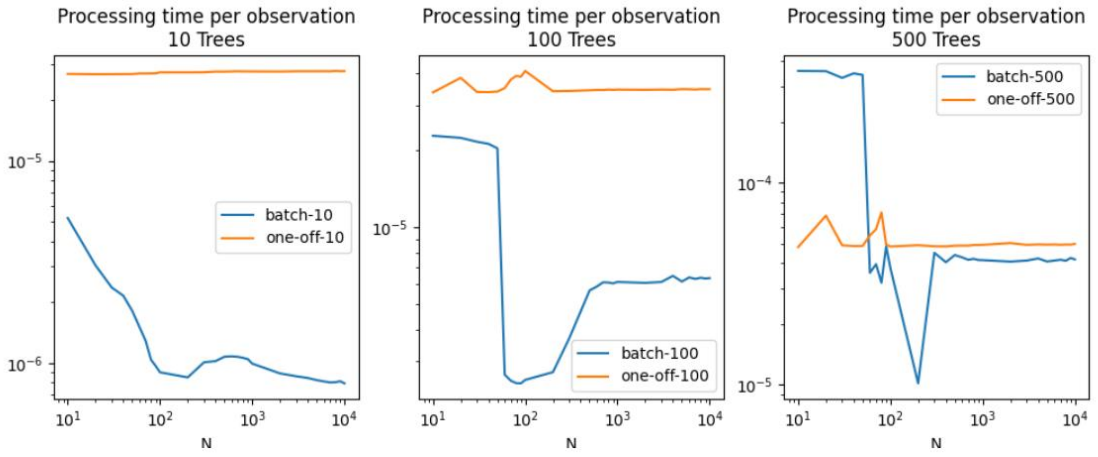
Parallelization, branching, caches L1, L2, L3?

Parallelization a random forest by trees is faster than parallelizing by rows except when the batch size is big?

Maybe but why?

But splitting the batch into chunks of 128 rows and parallelizing by trees is faster even for big batches.

How to use AVX instructions in a tree?

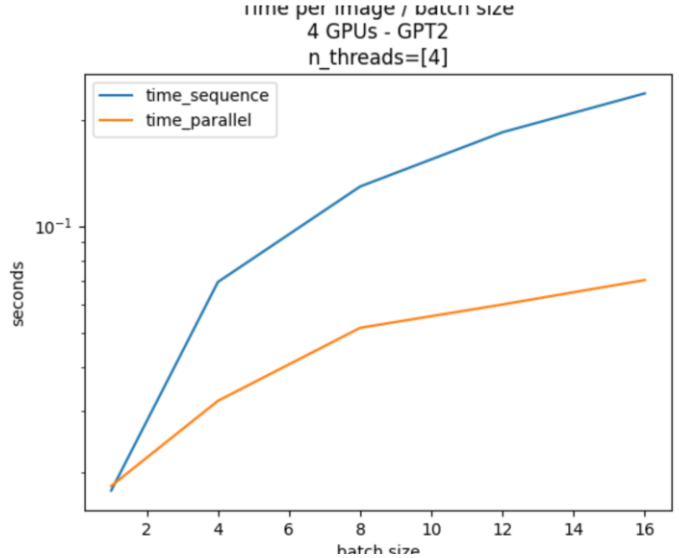


It is a lot of work to write fast code.

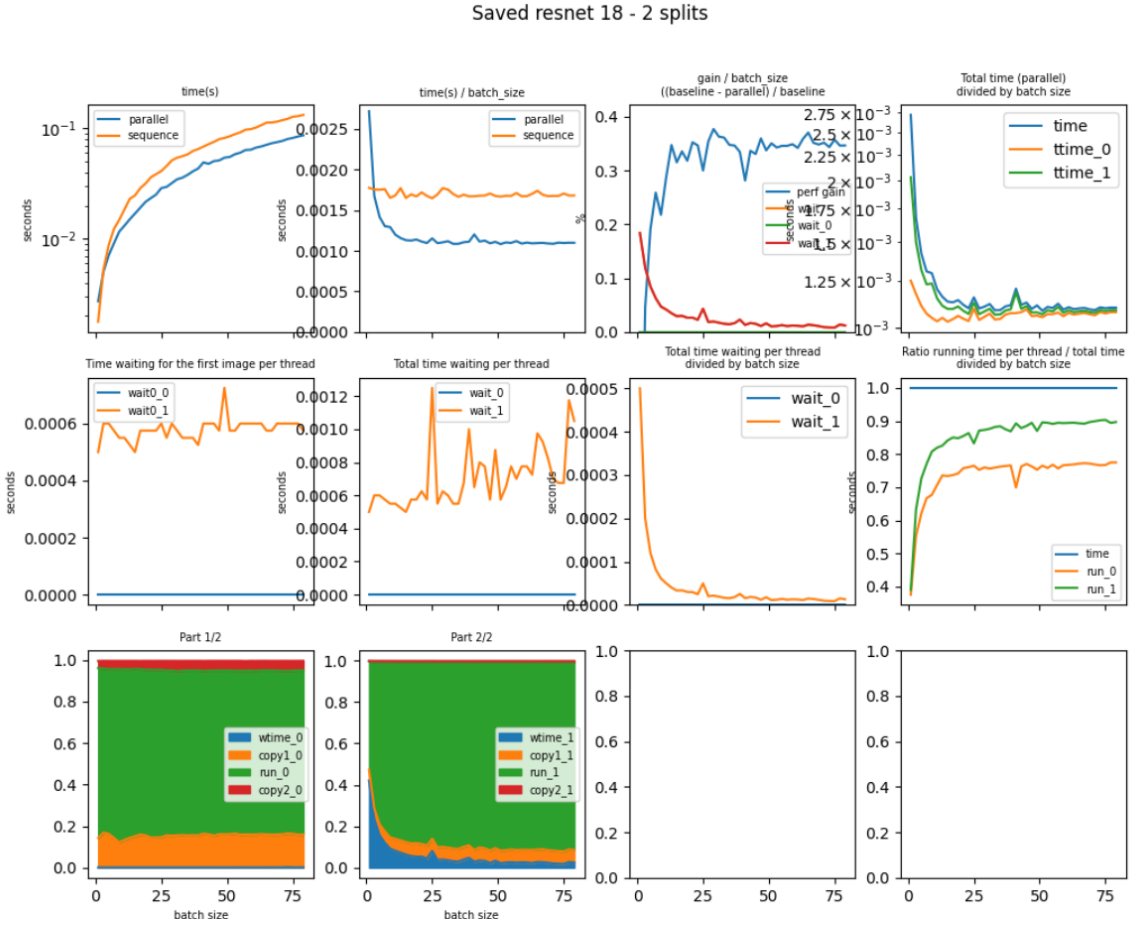
Parallelization over multiple GPUs

One CPU, 2 GPUs. How to parallelize?

Each GPU run inferences or…



The model is big, each GPU runs a piece of it.



No onnx operator, no runtime for it?

Custom operator, custom runtime. Strings are still an issue.

onnxruntime implements a C API to register dynamic libraries implementing new kernels.

onnxruntime-extensions leverages this mechanism to enable popular tokenizers.

Python

```python
import onnxruntime as _ort
from onnxruntime_extensions import get_library_path as _lib_path

so = _ort.SessionOptions()
so.register_custom_ops_library(_lib_path())

# Run the ONNXRuntime Session, as ONNXRuntime docs suggested.
# sess = _ort.InferenceSession(model, so)
# sess.run (...)
```

C++

```cpp
// The line loads the customop library into ONNXRuntime engine to load the ONNX model with the custom op
Ort::ThrowOnError(Ort::GetApi().RegisterCustomOpsLibrary((OrtSessionOptions*)session_options, custom_op_lib

// The regular ONNXRuntime invoking to run the model.
Ort::Session session(env, model_uri, session_options);
RunSession(session, inputs, outputs);
```

Natural language operators

BertTokenizer

▼ BertTokenizer details
BertTokenizer replicates `encode_plus` function of BertTokenizer (huggingface version).

Inputs

text: tensor(string) The string tensor for tokenization

Attributes

vocab_file: string

The content of vocab which has same with huggingface.

do_lower_case: int64_t (default is 1, 1 represents True, 0 represents False)

Whether or not to lowercase the input when tokenizing.

ONNXRuntime-Extensions

Azure Pipelines succeeded

What's ONNXRuntime-Extensions

Introduction: ONNXRuntime-Extensions is a library that extends the capability of the ONNX models and inference with ONNX Runtime, via ONNX Runtime Custom Operator ABIs. It includes a set of ONNX Runtime Custom Operator to support the common pre- and post-processing operators for vision, text, and nlp models. And it supports multiple languages and platforms, like Python on Windows/Linux/macOS, some mobile platforms like Android and iOS, and Web-Assembly etc. The basic workflow is to enhance a ONNX model firstly and then do the model inference with ONNX Runtime and ONNXRuntime-Extensions package.

⚠ **NOTE:** most ONNXRuntime-Extensions packages are in ***active development*** and most packages require building from source. The package information will be updated here if it is published.

OpenVino

Model optimization

[OpenVino]

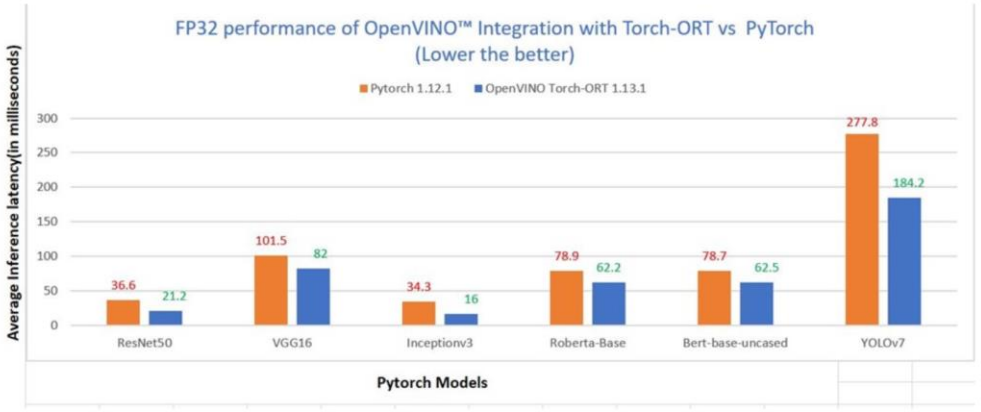
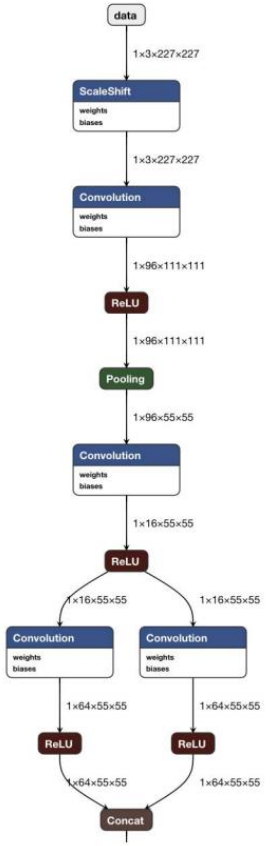
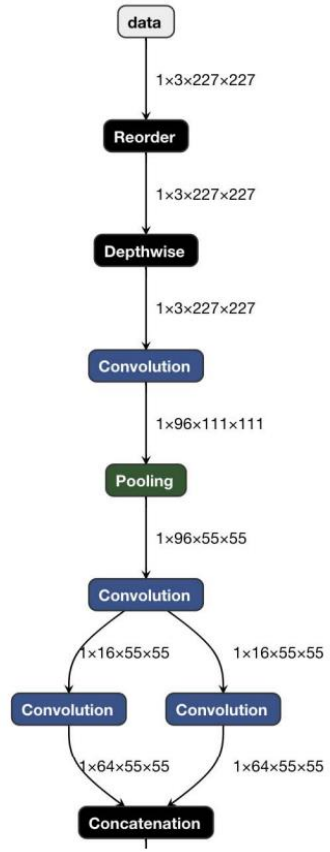


Figure 2: FP32 Model Performance of OpenVINO™ Integration with Torch-ORT as compared to PyTorch. This chart shows average inference latency (in milliseconds) for 100 runs after 15 warm-up iterations on an 11th Gen Intel(R) Core (TM) i7-1185G7 @ 3.00GHz.

[source]





TensorRT

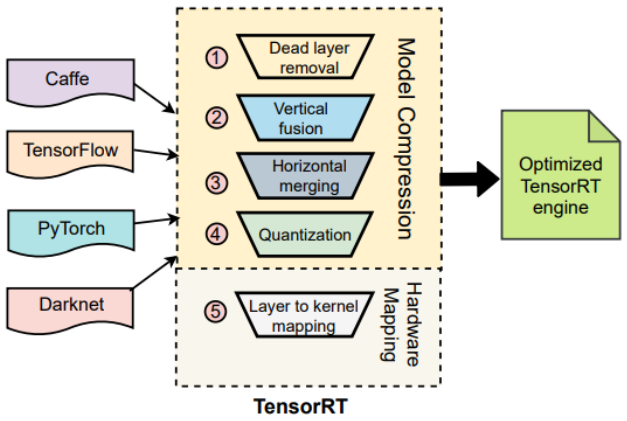
- Model rewriting
- Quantization

[TensorRT]

"Relative Entropy" of two encodings

- INT8 model encodes the same information as the original FP32 model.
- We want to minimize loss of information.
- Loss of information is measured by Kullback-Leibler divergence (AKA *relative entropy* or *information divergence*).
  - P, Q - two discrete probability distributions.
  - `KL_divergence(P,Q) := SUM(P[i] * log(P[i] / Q[i] ), i)`
- Intuition: KL divergence measures the amount of information lost when approximating a given encoding.

source



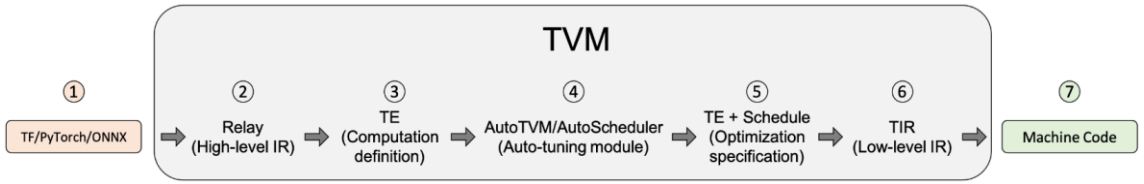
source

A glimpse of the future…

More chips, more processors (see H100), bigger models (GPT-3, 175B), how to quicky implement the best algorithm for every operator?

```
# do in parallel
for m in range(0, M, BLOCK_SIZE_M):
  # do in parallel
  for n in range(0, N, BLOCK_SIZE_N):
    acc = zeros((BLOCK_SIZE_M, BLOCK_SIZE_N), dtype=float32)
    for k in range(0, K, BLOCK_SIZE_K):
      a = A[m : m+BLOCK_SIZE_M, k : k+BLOCK_SIZE_K]
      b = B[k : k+BLOCK_SIZE_K, n : n+BLOCK_SIZE_N]
      acc += dot(a, b)
    C[m : m+BLOCK_SIZE_M, n : n+BLOCK_SIZE_N] = acc;
```

- Triton
- AITemplate
- Apache TVM




```
KERNEL_TEMPLATE = jinja2.Template(
    """
__global__ void add_one(half* output, const half* input, const int64_t num_elements) {
  const int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < num_elements) {
    output[idx] = input[idx] + half(1.0);
  }
}
void invoke_add_one(half* output, const half* input, int64_t num_elements, {{prefix}}Stream_t stream)
  if (num_elements < 1024) {
    dim3 grid(1);
    dim3 block(num_elements);
    add_one<<<grid, block, 0, stream>>>(output, input, num_elements);
  } else {
    dim3 grid((num_elements + 1024 - 1) / 1024);
    dim3 block(1024);
    add_one<<<grid, block, 0, stream>>>(output, input, num_elements);
  }
}
    """
)
```

Conclusion

Don’t sleep.

The world is moving fast these days.

Questions?