ONNX

OPERATORS SIG

Michal Karzynski (Intel)
G. Ramalingam (Microsoft)

OVERVIEW

- A key part of the ONNX spec is the set of operators (aka "opsets") that make up the spec
  - Organized into domains
    - ONNX domain: focus on DNN operators
    - ONNX-ML domain: focus on classical ML operators
  - Versioned
- The Operators SIG focuses on the definition of the "operator sets"
  - Additions of new operators
  - Clarification of op specs
  - Updates to op specs

CHANGES SINCE LAST COMMUNITY PRESENTATION

OPSETS 16 AND 17

- New Ops/Functions:
  - GridSample
    - used in Spatial Transformer Networks
  - LayerNormalization
    - Widely used, e.g. in language models like BERT
  - Signal processing (DFT, STFT, HannWindow, HammingWindow, BlackmanWindow, MelWeightMatrix)
    - Used in audio models (speech-to-text, audio cleanup, audio classification)
  - SequenceMap
    - enables batched pre-processing, e.g. a batch of images of varying sizes for ResNet-50

  - All are functions except GridSample, DFT, STFT, MelWeightMatrix.
  - DFT and STFT are planned to be promoted to be functions soon.

CHANGES SINCE LAST COMMUNITY PRESENTATION

OPSETS 16 AND 17

- Updates to existing ops
  - Support duplicate index values in scatter ops
    - via reduction (add or multiple all values at an index)
    - ScatterND and ScatterElements
  - Add bfloat16 support (Scan, LessOrEqual, GreaterOrEqual, LeakyRelu, PRelu, Where)
  - Add support for optional types (If, Loop, Identity)
  - RoiAlign: adds attribute coordinate_transformation_mode to adjust half-pixel error

ONNX Roadmap:
What Next?

Key Goals

- Clear and unambiguous specification
  - Improve documentation (Issue #3651)

- Compact specification
  - Make it easier to implement backends, especially on new hardware
  - Reduce operator surface area (of core primitive ops)

- Expressiveness
  - Enable newer models, pre-processing, post-processing
  - … more ops!

- Efficiency
  - Need for more coarse-grained (composite) ops!

ONNX Functions

- ONNX Functions: a key enabler to meeting our goals:
  - Defines the function in terms of other (core operators)
  - Provides an executable specification (reduces ambiguity)
  - Provides a default implementation (reducing core operator surface area)
    - Less concerned about adding new function definitions to increase expressiveness
  - Enable use of specialized kernels, when needed and where available, for efficiency

Next Steps

- Reduce existing *primitive* operator surface area
  - Around 25-30 of existing operators can be promoted into functions (Issue #3877)
- Enable authoring ONNX functions using Python
  - And automatically convert to FunctionProto (ONNX's serialized representation)
  - Easier to author
  - Easier to read and understand (edge case behavior or fine details)
- and execute them in Python debuggers
  - As a tool to understand the ONNX spec, not intended for production-use or perf
  - To test, debug, and understand function definitions
- ONNXScript (a subset of Python)

Example ONNX Functions in Python

```python
M_SQRT1_2 = math.sqrt(0.5)
@script()
def Gelu(X):
    phiX = 0.5 * (op.Erf(M_SQRT1_2 * X) + 1.0)
    return X * phiX
```

```
Gelu (X) => (return_val) {
   tmp = Constant <value = <Scalar Tensor [0.5]>>()
   tmp_0 = Constant <value = <Scalar Tensor [0.7071067690849304]>>()
   tmp_1 = Mul (tmp_0, X)
   tmp_2 = Erf (tmp_1)
   tmp_3 = Constant <value = <Scalar Tensor [1.0]>>()
   tmp_3_4 = CastLike (tmp_3, tmp_2)
   tmp_5 = Add (tmp_2, tmp_3_4)
   tmp_6 = CastLike (tmp, tmp_5)
   phiX = Mul (tmp_6, tmp_5)
   return_val = Mul (X, phiX)
}
```

Another example (with control-flow)

```
def Dropout(data, ratio, training_mode, seed):
  if (training_mode):
    rand = RandomUniformLike(data, seed=seed, dtype=FLOAT)
    mask = (rand >= ratio)
    output = Where(mask, data, 0) / (1.0 - ratio)
  else:
    mask = Expand(True, Shape(data))
    output = data
  return (output, mask)
```

Enable debugging via eager-mode

```
VARIABLES
  Locals
    > A: array([[0.5488135 , 0.7689989 , 0.318294…
    > AW: array([[4.458284 , 4.2592363, 4.490737 …
    > AWBias: array([[4.7571316, 4.89791   , 5.386…
    > Bias: array([0.2988478 , 0.6386737 , 0.8952…
    > (return) Op.__call__: array([[4.458284 , 4.…
    > (return) Opset.__getattr__: <onnxscript.val…
    > W: array([[0.0977016 , 0.84479755, 0.889085…
  > Globals
```

my > eager_mode_evaluation.py > simple

```python
@script()
def simple(A, W, Bias):
    AW = op.MatMul(A, W)
    AWBias = AW + Bias
    Y = op.Relu(AWBias)
    return Y


np.random.seed(0)
m = 2048
k = 16
n = 4096
a = np.random.rand(k, m).astype('float32').T
w = np.random.rand(n, k).astype('float32').T
b = np.random.rand(n,).astype('float32').T

print(simple(a, w, b))
```

Example ONNX Functions in Python

```
def LeakyRelu(X, alpha=0.01):
    return Where(X < 0, alpha * X, X)

def HardSigmoid (X, alpha=0.2, beta=0.5):
    return Max(0, Min(1, alpha * X + beta))

def Shrink(x, bias = 0.0, lambd = 0.5):
    return Where(x < -lambd, x + bias,
           Where(x > lambd, x - bias, 0))

def Softplus(X):
    return Log(Exp(X) + 1)

def Softsign(X):
    return X / (1 + Abs(X))
```

THANKS FOR COMING!!!

Please Get Involved!

- Github: PRs, Issues, and Discussions
- Slack channel: https://slack.lfai.foundation and join onnx-operators
- Monthly SIG meetings (see slack channel for announcements)