Item 312
git_comments:

1. accumulated g accumulated delta
2. update g, delta
3. preprocess grad
4. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
5. pylint: disable=W0223
6. update weight
7. preprocess grad
8. history
9. update history
10. When grad is sparse, update weight with fused kernel
11. When the grad is not sparse, the func step is called to update weight and state
12. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
13. update weight
14. preprocess grad
15. update weight with fused kernel
16. variance
17. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
18. update weight
19. mean
20. update mean and var
21. mean variance
22. preprocess grad
23. pylint: enable=line-too-long
24. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
25. pylint: disable=W0223
26. update weight
27. update mean and var
28. preprocess grad
29. momentum previous weight
30. previous weight
31. update mom, previous_weight
32. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
33. pylint: disable=W0223
34. update weight
35. preprocess grad
36. update weight with fused kernel
37. d_0 v_0 z_0
38. update d, v, z
39. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
40. update weight
41. preprocess grad

42. update weight with fused kernel
43. z n
44. pylint: disable=invalid-name pylint: disable=line-too-long
45. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
46. update weight
47. update z, n
48. update mean, var
49. preprocess grad
50. **comment:** becomes NaN if ratio == NaN or 0, otherwise 0
    label: code-design
51. apply bias correction
52. mean var
53. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
54. update weight
55. calculate lamb_trust_ratio
56. compute lars clip grad + wd * weight is performed after computing lars
57. update mom
58. preprocess grad
59. Same than usual using preloaded sgd functions
60. **comment:** becomes NaN if ratio == NaN or 0, otherwise 0
    label: code-design
61. calculate lars_trust_ratio
62. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
63. update weight
64. mean variance
65. preprocess grad
66. warming momentum schedule
67. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
68. pylint: disable=W0223
69. update weight
70. update mean and var
71. update mom
72. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
73. update weight
74. preprocess grad
75. update var
76. var
77. preprocess grad
78. update weight with fused kernel
79. update mean, var, mom
80. Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
81. mean var mom
82. update weight
83. update `aggregate_num` number of weights in a single kernel. this does not support sparse weight or gradient.

84. update mom
85. weight32 is a float32 copy of weight. in the kernel, we firstly update weight32, and then cast the result to float16 and save it to weight.
86. preprocess grad
87. When either weight or gradient is sparse, aggregate is False.
88. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
89. update weight
90. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
91. pylint: disable=W0223
92. preprocess grad
93. update weight
94. update mom
95. preprocess grad
96. update weight with fused kernel
97. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
98. update weight
99. convert ctypes.char_p.value back to python str if needed
100. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
101. segregate values based on type
102. coding: utf-8 Licensed to the Apache Software Foundation (ASF) under one or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0 Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.
103. **comment:** TODO(junwu): This is a temp solution for allowing converting np.ndarray to mx.nd.NDArray to be fed into the optimizer since users may have custom optimizers implemented using mx.nd.NDArray ops.
    label: code-design
104. pylint: disable=W0223
105. set seed for Gaussian noise replication

git_commits:

1. **summary:** [MXNET-#16167] Refactor Optimizer (#17400)
   message: [MXNET-#16167] Refactor Optimizer (#17400) * refactor optimizer * refactor optimizer * fix svrg test * fix rmsprop param naming * fix signum test * fix pylint and perl test * fix perl test and signsgd test * fix * retrigger ci * reduce ci overheads
   label: code-design

github_issues:

1. **title:** [RFC] Apache MXNet 2.0 Roadmap
   body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-

compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from

existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

2. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E)

3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

3. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add

constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

4. **title:** [RFC] Apache MXNet 2.0 Roadmap
   body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators

1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution

7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

5. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than

one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

6. **title:** [RFC] Apache MXNet 2.0 Roadmap
   body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has

long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data

sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

label: code-design

7. **title:** [RFC] Apache MXNet 2.0 Roadmap

body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by

default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic

Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

8. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT

code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

label: code-design

9. **title:** [RFC] Apache MXNet 2.0 Roadmap

body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards

operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the

development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1.
https://github.com/apache/incubator-mxnet/issues/14883
label: code-design

10. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify

the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

11. **title:** [RFC] Apache MXNet 2.0 Roadmap
    body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several

high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image

modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

12. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape

Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block

sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

13. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of

hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

14. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the

existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and

tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

15. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for

argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

16. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are

permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using

hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

17. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work:

https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for

operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

18. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually,

Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

label: code-design

19. **title:** [RFC] Apache MXNet 2.0 Roadmap

body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-

compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from

existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

label: code-design

20. **title:** [RFC] Apache MXNet 2.0 Roadmap

body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC:

https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial

and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

21. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused

parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

22. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for

index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization

tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

23. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve

interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

label: code-design

24. **title:** [RFC] Apache MXNet 2.0 Roadmap

body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend

since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples

and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

25. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs

with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as

config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

26. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to

symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

27. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of

NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note:

https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

28. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify

the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

29. **title:** [RFC] Apache MXNet 2.0 Roadmap
    body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several

high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image

modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

30. **title:** [RFC] Apache MXNet 2.0 Roadmap
    body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape

Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block

sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

label: code-design

31. **title:** [RFC] Apache MXNet 2.0 Roadmap

body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |-----------|-----------|-----------|-----------|-----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address

usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

32. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with

a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the

design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

33. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there’s no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing)

2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes. 3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

34. **title:** [RFC] Apache MXNet 2.0 Roadmap
body: # Overview Status: https://github.com/apache/incubator-mxnet/projects/18 The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here. The purpose of this RFC is to

organize and present the roadmap towards 2.0. As 2.0 will be a major release, changes that would break backward compatibility are permissible. The proposed changes in this RFC are either collected from past roadmap discussions such as #9686, or are based on various common issues from the past. This RFC organizes these changes into self-contained projects to facilitate clear definition of project, captures the risks and status quo to the best of our knowledge. To help navigate, the projects are further divided into several high-level areas. Some of the listed projects are already in progress, and are included to provide a clear overview. The objectives of Apache MXNet 2.0 include: - Improve expressiveness and usability of user-facing API. - Improve expressiveness and usability of the technical stack for lower development cost and maintainability. In terms of frontend, this roadmap focuses mostly on Python-frontend since MXNet has been taking a Python-first approach. The expectation with respect to other language bindings is that they would evolve along with the backend evolution and make use of the improvements. Given that breaking changes can occur, maintainers of different language bindings are expected to participate in related interface definition discussions. ## 1. MXNet NP Module NumPy has long been established as the standard math library in Python, the most prevalent language for the deep learning community. With this library as the cornerstone, there are now the largest ecosystem and community for scientific computing. The popularity of NumPy comes from its flexibility and generality. In #14253, the MXNet community reached consensus on moving towards a NumPy-compatible programing experience and committed to a major endeavor on providing NumPy compatible operators. The primary goal of the projects below is to provide the equivalent usability and expressiveness of NumPy in MXNet to facilitate Deep Learning model development, which not only helps existing deep learning practitioners but also provides people in the existing NumPy community with a shortcut for getting started in Deep Learning. The efforts towards this goal would also help a secondary goal, which is to enable the existing NumPy ecosystem to utilize GPUs and accelerators to speed up large scale computation. ### 1.1. NumPy Operator Testing Scope: 1. adopt __array_function__ and numpy existing tests. 2. extend testing to GPU 3. investigate numpy testing strategies 4. decide correctness criteria for acceptance ### 1.2. NumPy Operator performance profiling Scope: 1. Automatically profile the performance of NumPy operators ### 1.3. NumPy operator coverage Scope: 1. improve operator until full NumPy coverage, with prioritization towards operators used in the ecosystem and deep learning in general Operator coverage as of 07/03/2019 ``` | module | NumPy | deepNumPy | jax | cupy | |----------|----------|----------|----------|----------| | np | 603 | 89 | 445 | 321 | | ndarray | 71 | 32 | 71 | 56 | | random | 63 | 5 | 15 | 49 | | linalg | 31 | 2 | 8 | 15 | ``` ### 1.4. NumPy Extension Operator Reorganization and Renaming Scope: 1. consistent type usage for index input and return values from sort, topk #11031 #11134, #12197 2. array creation operators with flexible dtype definition #12290. (dtype=None) 3. moving_mean/moving_var in batchnorm 4. consistent usage of axis vs dim 5. promote or deprecate contrib operators ### 1.5. NumPy ndarray type extension Scope: 1. bfloat16 support (not in NumPy yet but useful for deep learning) (low priority — Intel) 2. boolean type support 3. complex (for FFT) ### 1.6. NumPy ndarray boolean indexing Scope: 1. allow boolean masks in NumPy ndarray indexing by adding the operator, potentially through extending op.where ### 1.7. Hybridizable basic (and advanced) indexing Scope: 1. Allow operations such as y = x[1:3, 2, ...] to be hybridizable Note: Preliminary work: https://github.com/apache/incubator-mxnet/pull/15663 ## 2. Graph Enhancement and 3rdparty support The objective of the following projects is to enable easier development of third-party extensions without requiring changes to be checked in the MXNet project. Examples of such extensions include third-party operator library and accelerators. ### 2.1. Graph Partitioning for Dynamic Shape Operators Scope: 1. partition inside control flow operators (and all cached ops) 2. partition on operators with dynamic shapes for partial memory planning and caching. ### 2.2. Improved Third-party Operator Support Scope: 1. allow registering custom operators by exposing C API (and frontend API) to register NNVM op at runtime. 2. verify serialization, deserialization, and graph passes for graphs with these operators are working properly. ### 2.3. Improved Third-party Backend Support (subgraph property) Scope: 1. expose a graph pass for standard graph partitioning with back-end-specific criteria as a C API and frontend API. ### 2.4. Large tensor support by default Scope: 1. enable default support for tensor with int64 dimension sizes 2. make sure there's no significant performance regression in operators Risks: 1. performance regression may happen in a subset of operators, which can disproportionally affect certain models. 2. compatibility and silent behavior change. Notes: in progress (RFC: https://lists.apache.org/thread.html/df53b8c26e9e0433378dd803baba9fec4dd922728a5ce9135dc164b3@%3Cdev.mxnet.apache.org%3E) ## 3. API Changes The objective of the following projects is to address the technical debts accumulated during the development of MXNet 0.x and 1.x with respect to the API definition. ### 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. Scope: 1. use packed function for flexibility (and potentially efficiency through avoiding string parsing) 2. do not expose backend accelerator-specific types such as mkldnn::memory in C-API 3. do not rely on topological ordering for argument passing (#15362). 4. verification of thread-safety and performance for C API Risks: 1. backend integration may require refactoring or even redesign 2. existing use cases such as other frontend may be broken without substitute 3. feedback is scattered and we may miss the opportunity to change some APIs in 2.0 ### 3.2. Unify Executor Scope: 1. SymbolBlock equivalent in C/C++, unify the executor implementation for symbol/module and the one for gluon blocks 2. migrate other versions of inference API 3. Support mirror option in the unified executor ### 3.3. Gradient of Gradient support Scope: 1. higher order gradient support for a subset of operators Risks: 1. large number of backward operators could introduce significant technical debt if not properly verified. 2. ill-informed prioritization may result in usability issue (e.g. common GAN not supported) ### 3.4. Autograd Extension Scope: 1. improve interface to support specifying intermediate output grad nodes 2. improve interface for better usability. (retain_graph → something not involving graph) 3. update graph pass for correctness ### 3.5. NNVM-backend Operator Interface Changes Scope: 1. support more than one temporary spaces 2. split forward shape/type inference and reverse shape/type inference for better error messaging. 3. deferred initialization removal (or improve error/info message) 4. accompanying operator implementation changes Risks: 1. some changes may make operator implementation less error-prone while less flexible, and thus require some reworking. ## 4. Gluon 2.0 Since the introduction of the Gluon API, it has superceded other API for model development such as symbolic API and model API. Conceptually, Gluon is the first attempt in the deep learning community to unify the flexibility of imperative programming with the performance benefits of symbolic programming, through trace-based just-in-time compilation. The objectives of the following projects are: - address usability issue as a result of the divergence in the behavior of NDArray and Symbol. - extend the JIT to improve the coverage of hybridization. - introduce new functionality to facilitate more areas of research such as Baysian methods and AutoML. - improve the usability and performance of the utility in Gluon. ### 4.1. Unifying symbolic and imperative mode for tensor library Scope: 1. unify the operator implementation and behaviors of symbolic and imperative execution modes (#10875) 2. allow naming for ndarray similar to symbol 3. address the necessary changes in shape/type inference. ### 4.2. Unifying Block and HybridBlock Scope: 1. move hybridization logic to a JIT decorator 2. extend parameter management to Block 3. user-friendly warning for native control flow in JIT code. ### 4.3. Gluon Block Enhancement Scope: 1. inspection of graph internals similar to monitor for Module ([PR 15839](https://github.com/apache/incubator-mxnet/pull/15839)) 2. support additional types in argument such as dict, kwargs, None 3. fused parameters and gradients respectively 4. register custom parameter ### 4.4. Enable Symbolic Shape (& Dtype) for Array Creation in NNVM-backend Scope: 1. allow flexible creation of array based on shapes of other arrays that are only known at runtime 2. add constant symbol type as the return value of symbol.shape (?) 3. support constant symbol as operator arguments (?) 4. constant folding for constant symbols ### 4.5. Gluon Distributions Module Scope: 1. sampling and pdf definition for distributions. Distribution https://github.com/amzn/MXFusion. https://github.com/apache/incubator-mxnet/pull/14617. 2. wrap operators into more usable classes.

3. reproducible global seed ### 4.6. Gluon Metrics Module Scope: 1. address usability and performance issues in mxnet.metric using hybridizable NumPy op ### 4.7. Gluon Optimizer Module Scope: 1. API changes such as consistent weight decay (#9881), change default value to not apply wd on bias terms (#11953) 2. hybridizable optimizers 3. new optimizers (#9182) ### 4.8. Gluon Data API Extension and Fixes Scope: 1. address diverging interfaces and remove transform= constructor arg (#11141). 2. reorganize io/image modules and provide data loader instead. 3. lowering dataloader to backend for efficiency (#13593) 4. shared memory propagation? ### 4.9. Gluon Estimator Extension for Experimenting Utilities Scope: 1. logging of configuration (DeepNLU), state, and performance for checkpointing for easier resume 2. pre-defined estimators for common problems ### 4.10. Gluon Estimator Refactoring for Examples and Tutorials Scope: 1. modularize and refactor unstructured scripts and examples into estimator class utilities ### 4.11. Gluon Distributed Training Usability Enhancement Scope: 1. more flexibility for communication with kvstore UDFs 2. add distribution strategies to estimator 3. plugin for communication backends (horovod, byteps, parameter server) for data parallel training 4. data sharding/sampling/streaming enhancement for distributed training ## 5. Documentation Documentation is the most important factor for new adoption of a library. The following projects aim to: - address the usability and discoverability issues in the current MXNet website - improve the quality of documentation to make it correct, clear, and concise. - help adoption of the changes in MXNet 2.0 from existing users. ### 5.1. MXNet 2.0 Migration Guide Scope: 1. document high-level mapping from old functionality to new API for data pipeline, modeling, optimization, training loop, metric, inspection and logging, debugging. Risks: 1. parallel development of the doc may result in outdated doc. 2. auto doc verification is needed. ### 5.2. MXNet 2.0 Developer Guide Scope: 1. carefully document the design and contribution guide for features with low entry bar such as operator, gluon block, doc, optimizer, metric, examples and tutorials. 2. clear and up-to-date system design overview. 3. clear roadmap ### 5.3. Adopt beta.mxnet.io as official website Scope: 1. infrastructure change for new doc build 2. merge into master with NumPy.mxnet.io 3. improve load time and browsing experience 4. CDN in popular region such as China, with automated validation and testing. Note: https://github.com/ThomasDelteil/mxnet.io-v2 ## 6. Profiling and Debugging Profiling and debugging is a common step in the development of deep learning models, and proper tools can help significantly improve developer's productivity. The objective of these projects is to provide such tools to make it easier to discover issues in correctedness and performance of models. ### 6.1. Memory Profiler Scope: 1. memory profiler logging support in backend 2. automatic array naming tool based on scope 3. tree-map visualization tool for inspecting profiler dump ### 6.2. Enhanced Debugging Tool Scope: 1. Enable user-specified error handling 2. Improve error message 3. Stacktrace inspection in debug API 4. Automatic error reporting tool 5. Runtime API for turning off asynchronous execution ## 7. Advanced Operators The objective of these projects are to extend the tensor library and operators for better performance and for advanced use. ### 7.1. Strided ndarray support Scope: 1. support strided array in a subset of operators 2. support auto-transpose of strided array in graph pass and executor ### 7.2. Ragged ndarray and operators Scope: 1. introduce ragged (variable length) tensor as 1st class tensor. Support zero-copy from RaggedNDArray to NDArray when no dimension is ragged. 2. Load balancing strategy for operators that take RaggedNDArray as input 3. cover operators for NLP applications (RNN, transformer) ### 7.3. Improved Sparse Support Scope: 1. sparse format and operator support 2. scipy coverage 3. operators for graph neural-networks (e.g. ops in minigun) Minimum support: * format: csr, * zerocopy to DLPack * integration with minigun kernels Next-level support: * format: coo and block sparse. ## 8. Building and Configuration ### 8.1. CMake improvement and Makefile deprecation Scope: 1. reimplement CMakeLists for DMLC dependencies 2. reimplement CMakeLists for MXNet to support 1) building best performing binary in any platform 2) building portable binary distribution for pip ### 8.2. MXNet Configurator Scope: 1. drop environment variables and centralize them as config. 2. define functionalities that support runtime-switch (candidates: memory pool, engine, worker thread pools) and expose frontend API 3. allow saving and loading of mxnet system config ## 9. Advanced training and deployment ### 9.1. Automatic Quantization and Quantized Training for NumPy Scope: 1. automatic quantization based on heuristic (or learning) 2. BMXNet ### 9.2. Mobile and edge-device deployment Scope: 1. replace amalgamation with more user-friendly function (TF-lite equivalent). 2. tutorial and example 3. metal support ## 10. Performance ### 10.1. MXNet Execution Overhead Scope: 1. https://github.com/apache/incubator-mxnet/issues/14883

github_issues_comments:

1. @szha Really great proposal and we may want to add some items in 2.0 too. Is there a timeline of 2.0?
2. Hey, this is the MXNet Label Bot. Thank you for submitting the issue! I will try and suggest some labels so that the appropriate MXNet community members can help resolve it. Here are my recommended label(s): Feature
3. Is there a plan to create a branch either for the 1.x version and have master reflect 2.0 or to create a branch for the 2.0 version and keep master on 1.x for now?
4. @pengzhao-intel a tentative target date is by end of Q1 2020. @zachgk we will create a branch for 2.0. Initially we will keep master to be 1.x and have 2.0 in a new branch. After 1.6 release we will revisit how to make the 2.0 branch the master.
5. Just a quick cheer up for a new website of MXNet... its way more awesome and beautiful than I expected. Though minor bugs are still there, for ex- most of the link in the tutorials are broken and not working. Anyways great work so far.
6. **body:** Any plan to simplify the build of c and c++ api for mxnet2.0?It is hard(or very hard) to build a working version of mxnet with cpp api on different platforms(windows, linux, mac), every new release of the mxnet may or may not break something and we need to spend many hours to figure out how to make it work. I am happy with python api, but not all of the tasks suitable for python. Almost every deep learning tools are based on c and c++, but almost everyone of them are difficult to or partially work with c and c++.
   label: code-design
7. @stereomatchingkiss good point. What are you using c/c++ api for?
8. **body:** > > > @stereomatchingkiss good point. What are you using c/c++ api for? 1. Develop stand alone app on desktop and mobile(maybe on another devices like rpi4 or jetson nano in the future) 2. Wrapper of another language(ex : php) 3. Run the inference task on aws lambda, we do not want to prune the libs of python manually if we could build a slim library of mxnet/tensorflow/pytorch. Maybe you could open a post to ask the users what are they expect for c or c++ api, I guess most of them only need to use the api to perform inference task but not training(python do a great job about this), this should help you shrink the size of the libs and made the codes less complicated.
   label: code-design
9. **body:** @stereomatchingkiss That's a bit what amalgamation part was for ? a simplified inference interface. The last time I use amalgamation (some years ago) it was often break by update and not really maintain.
   label: code-design
10. The status of MXNet 2.0 project is tracked at: https://github.com/apache/incubator-mxnet/projects/18. The status for each project will be updated by the contributor who's driving it. If you have more projects that you intend to drive please first discuss here.
11. Once 1.6 release is complete, we will create a branch for MXNet 1.x for future releases and start using master branch for 2.0 development.

12. Should we create a new branch for 2.0? I think we are also planing for 1.7.0 https://github.com/apache/incubator-mxnet/issues/16864
13. In the past we always kept development on the master branch, thus how about branching out 1.7.0 release branch and keeping development on master?
14. +1 for using master branch for 2.0 development. I think we need 3 branches at least: 1. master branch: for 2.0 development 2. v1.x: for 1.x development and maintenance 3. v1.7.x: for 1.7.x release
15. That's what I had in mind. The v1.7.x branch doesn't have to be created until code freeze for 1.7.0
16. > 3.1. C-API Clean-up C-API is the foundational API in MXNet that all language bindings depend on. @szha I'm looking at the item 3.1.2. Could you please explain the scope of C-API? Do you mean those APIs sit in the src/c_api/ folder?
17. @TaoLv one promising direction that the community is converging to is the interface based on packed function (motivation as described by @tqchen in https://github.com/apache/incubator-mxnet/issues/17097#issuecomment-567604444). What this means to the project is that the existing c API will be updated to follow the packed function interface.
18. **body:** Is there a plan to remove the cudnn_off argument from the neural network operators such as Dropout, Convolution, Pool etc. It creates a few usability issues: (1) Once a model is exported. It requires users to change this flag in all the layers manually if they want to enable/disable cuDNN. (2) When the cudnn_off is set to true in some layers, the global env variable `MXNET_CUDNN_AUTOTUNE_DEFAULT` becomes don't care. It's very confusing to users to see an error message like "Please turn off MXNET_CUDNN_AUTOTUNE_DEFAULT" by indeed it does not do anything. (3) Why did we expose such implementation detail to users at the first place? In the worst case, we should just provide a global variable to turn on/off cuDNN in all layers instead of at operator level.
    label: code-design
19. **body:** Thanks for this awesome work, it has benefited me a great deal. Here are some disadvantages(may be) listed blow: 1. it seems that c and c++ interface both could work, but can not finish single task only by one; 2. low bit training or inference is not available via c/c++(ver. 1.6.0 fix fp16 training); 3. static linking lib is (very) far away from easy to use, cmake configuration file(like MxNetConfig.cmake, etc.) generated by cmake will enough for end users to integrate libmxnet.a and other large bunch of static third party libs(it's not easy to maintain gentlemanly demeanor all a day when manually linking these day by day). people could easy to hack loading interface of a dynamic library. 4. smaller size of lib will more friendly to edge devices. 5. more c++ training demo, including how to use kvstore(multiple cards and multiple servers), it's really not easy to understand. Good day everyone.
    label: code-design
20. @kalcohol please create a new issue about "static linking lib is (very) far away from easy to use", describing your setup in more detail and potentially suggestions how to improve the user experience.
21. > @kalcohol please create a new issue about "static linking lib is (very) far away from easy to use", describing your setup in more detail and potentially suggestions how to improve the user experience. #17692 add this tiny requist.
22. We use the c++ interface for inference on a sorting machine. But also we would like to provide the users of our machines an easy and integrated user interface for training new sorting recipes. Now we use python or Mathematica scripts which is far of user friendly for non-programmers. So we want to use the c++ (shielded with a C# wrapper) to provide a custom training environment for non-programmers. Unfortunately, building the mxnet library with c++ support on Windows machine with MKL / CUDA is an ongoing nightmare. But we really like MxNet
23. **body:** @szha i checked some docs and projects about distributed training , 'Horovod' is project from uber team , 'Gloo' is project from facebook team. The basic idea is to use trick from HPC computing field which is more efficient then traditional param-server: http://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/?from=timeline There is a tool called openmpi on which the 'Horvod' project is based ,but i found openmpi is too difficult to configure and use . I also check the 'Gloo' which seems to use 'redis' to replace 'openmpi' . I strongly suggest not to use Horovod directly which is based on openmpi that is too complex and old. I also find bytedance has a good project solving the same problem not using MPI , https://github.com/bytedance/byteps maybe we cant better integrate bytedance solution in roadmap 2.0 . or we can have a mxnet internal solution similar to bytedance solution.
    label: code-design
24. @lilongyue the integration of bytePS to mxnet is in this PR https://github.com/apache/incubator-mxnet/pull/17555
25. > @lilongyue the integration of bytePS to mxnet is in this PR #17555 that's great !
26. A quick comment: DGL contains all sampling implementation and no longer relies on the implementation in MXNet. I think we should deprecate the graph sampling implementation in MXNet.
27. @szha is there a recent estimate on the timeline for MXNet 2.0? Would you recommend to develop downstream toolkits (e.g. Sockeye) against the master branch now or rather wait a little bit longer? Is there already documentation on how to transition MXNet 1.x projects to 2.x?
28. @fhieber we are planning to release the first public beta on this somewhere in August. At the moment we are finalizing some API changes and also validating them in GluonNLP. We will publish a transition doc as part of the public beta.
29. @szha We need to add moving AMP package from contrib to core? We will file RFC for this task.
30. **body:** @szha I found an inconvenient thing that there is no `concat` layer for gluon. Is it possible to add a `concat` layer for gluon?
    label: code-design
31. Making `MXNET_SAFE_ACCUMULATION=1` default when running on float16 would be very convenient!
32. +1 for turning it on by default. Get Outlook for iOS<https://aka.ms/o0ukef> _____ From: Davis Liang <notifications@github.com> Sent: Tuesday, August 18, 2020 5:32:29 PM To: apache/incubator-mxnet <incubator-mxnet@noreply.github.com> Cc: Sheng Zha <dmlc.notification@gmail.com>; Mention <mention@noreply.github.com> Subject: Re: [apache/incubator-mxnet] [RFC] Apache MXNet 2.0 Roadmap (#16167) Making `MXNET_SAFE_ACCUMULATION=1` default when running on float16 would be very convenient! -- You are receiving this because you were mentioned. Reply to this email directly or view it on GitHub: https://github.com/apache/incubator-mxnet/issues/16167#issuecomment-675784800

github_pulls:

1. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
    body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly

output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.

2. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.

3. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's

reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.

4. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
   body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.
5. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
   body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.
6. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
   body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I

finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.
label: documentation

7. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.
8. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.
9. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with

mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.

10. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.
11. **title:** [MXNET-507] Set dtype=int32 for ret_indices in ordering ops
body: ## Description ## There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we can not run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` ~~I propose to fix this bug by changing the dtype of indices to int32. This will make the code **backward incompatible**. However, it only breaks some rare usages. Normally we will directly output the indices or use them to slice a tensor. The current change do not break these common usages.~~ (I have used the other solution mentioned below.) Another solution is to support an additional flag `dtype` for these operators (as suggested in https://github.com/apache/incubator-mxnet/issues/11031). The problem of this solution is that we cannot avoid a nested macro, which is extremely slow to compile. So I haven't solved it this way. ```c++ MACRO_SWITCH(dtype, DType, { MACRO_SWITCH(idtype, IDType, { ... }); }); ``` This PR also fixes the issue reported in https://github.com/apache/incubator-mxnet/issues/10085 that ordering ops do not support kNullOp. ## Checklist ## ### Essentials ### Please feel free to remove inapplicable items for your PR. - [x] The PR title starts with [MXNET-$JIRA_ID], where $JIRA_ID refers to the relevant [JIRA issue] (https://issues.apache.org/jira/projects/MXNET/issues) created (except PRs with tiny changes) - [x] Changes are complete (i.e. I finished coding on this PR) - [x] All changes have test coverage: - Unit tests are added for small changes to verify correctness (e.g. adding a new operator) - Nightly tests are added for complicated/long-running ones (e.g. changing distributed kvstore) - Build tests will be added for build configuration changes (e.g. adding a new build option with NCCL) - [x] Code is well-documented: - For user-facing API changes, API doc string has been updated. - For new C++ functions in header files, their functionalities and arguments are documented. - For new examples, README.md is added to explain the what the example does, the source of the dataset, expected performance on test set and reference to the original paper if applicable - Check the API doc at http://mxnet-ci-doc.s3-accelerate.dualstack.amazonaws.com/PR-$PR_ID/$BUILD_ID/index.html - [x] To the my best knowledge, examples are either not

affected by this change, or have been fixed to be compatible with this change ### Changes ### - [x] Arbitrary dtype (exclude float16) for ordering operators, tests - [x] For topk, argsort, add the dtype option to specify the type of the output indices, tests - [x] Fix the bug that ordering ops do not support kNullOp, tests ## Comments ## - ~~This change is backward incompatible. However, I think it's reasonable to set the dtype of indices to int32 because it does not break the common usage where we use these indices to slice a tensor.~~ The change is backward compatible now.

github_pulls_comments:

1. After discussing offline with Eric, I'll add an additional `dtype` flag to make sure that the OP is backward compatible.
2. Feel free to remove "breaking" label when you're ready.
3. I think that it would be helpful if a check for the ranges of the floating point types is added. The recent addtion of `dtype` to `multinomial` implemented the check. https://github.com/apache/incubator-mxnet/blob/3eada3b32aeab5c8cdf7d507bcc3a986c9e5b91f/src/operator/random/sample_multinomial_op.h#L73-L76
4. Thanks a lot! I was just looking for such a check. Get Outlook for iOS<https://aka.ms/o0ukef> _____ From: Deokjae Lee <notifications@github.com> Sent: Monday, June 11, 2018 9:31:02 PM To: apache/incubator-mxnet Cc: Xingjian SHI; Author Subject: Re: [apache/incubator-mxnet] [MXNET-507] Set dtype=int32 for ret_indices in ordering ops (#11134) I think that it would be helpful if a check for the ranges of the floating point types is added. The recent addtion of dtype to multinomial implemented the check. https://github.com/apache/incubator-mxnet/blob/3eada3b32aeab5c8cdf7d507bcc3a986c9e5b91f/src/operator/random/sample_multinomial_op.h#L73-L76 — You are receiving this because you authored the thread. Reply to this email directly, view it on GitHub<https://github.com/apache/incubator-mxnet/pull/11134#issuecomment-396243589>, or mute the thread<https://github.com/notifications/unsubscribe-auth/AE8D7u7vceZYr3aWQORkAyQyQqv645Ntks5t7nEWgaJpZM4UYsvh>.
5. Is it okay to merge this in?
6. **body:** @sxjscience any chance you could add a small update to the documentation saying that the output is sorted. thanks!
   label: documentation

github_pulls_reviews:

1. should we remove this option given that it won't be supported?
2. OK, I'll remove it.
3. @szha Actually, the dtype of indices can be float16. There is no need to remove the option.

jira_issues:

1. **summary:** Two problems in the ordering operators (topk, sort, argsort)
   description: There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we cannot run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` I propose to fix this bug by changing the dtype of the indices to int32. However, this will make the code backward incompatible.
2. **summary:** Two problems in the ordering operators (topk, sort, argsort)
   description: There are two problems in the ordering operators, i.e, topk, sort, argsort: 1) Only real_t is supported. 2) The indices are stored as real_t. This will cause error in the backward pass where the gradient are passed to the wrong locations. For example, we cannot run the following code in the previous version: ```python import mxnet as mx import numpy as np import mxnet.ndarray as nd ctx = mx.cpu() a = mx.nd.arange(54686454, ctx=ctx, dtype=np.int32) a.attach_grad() k = 10 with mx.autograd.record(): b = mx.nd.topk(a, k=k, ret_typ='value') b.backward(mx.nd.ones((k,), ctx=ctx, dtype=np.int32)) a_grad = a.grad.asnumpy() for i in range(-1, - k - 1, -1): assert a_grad[i] == 1 ``` I propose to fix this bug by changing the dtype of the indices to int32. However, this will make the code backward incompatible.

jira_issues_comments: