RunInference: Machine Learning Inferences in Beam

By Andy Ye
Agenda

1. Background: ML lifecycle, and previous limitations in Beam
2. RunInference: Machine learning inferences in Beam
3. RunInference API: Usage, patterns, and features
4. RunInference in the future
5. RunInference demo
Typical Machine Learning Lifecycle

- Raw Data
- Pre-Processing
- Data Store (BigQuery)
- Model Training (Vertex AI)
- Model Registry (GCS)
- Inference
- Post Processing
- Data Store (BigQuery)
Typical Machine Learning Lifecycle

- Raw Data
- Pre-Processing
- Data Store (BigQuery)

Beam Pipeline
Typical Machine Learning Lifecycle

- **Raw Data**
- **Pre-Processing**
- **Data Store (BigQuery)**
- **Beam Pipeline**
- **Model Training (Vertex AI)**
- **Model Registry (GCS)**
Typical Machine Learning Lifecycle

Raw Data → Pre-Processing → Data Store (BigQuery) → Beam Pipeline → Model Training (Vertex AI) → Model Registry (GCS) → Inference → Post Processing → Data Store (BigQuery) → Beam Pipeline

How can we use these models in our pipelines?
Before Beam 2.40.0: No native support for making inferences

Most Machine Learning frameworks

Users must write a custom DoFn

TensorFlow

Users use RunInference from the `tfx_bsl` utility
RunInference: A new Beam transform to run ML inferences

- Key forwarding
- Processing the output
- Standard Metrics
- GPUs
Supported Frameworks

* TF support via the tensorflow/tfx-bsl repo. A migration to Beam is anticipated soon.
How to use RunInference

from apache_beam.ml.inference.base import RunInference

with pipeline as p:
    predictions = ( p | beam.ReadFromSource('a_source')
                        | RunInference(ModelHandler)))
ModelHandlers

from apache_beam.ml.inference.sklearn_inference import SklearnModelHandlerNumpy
from apache_beam.ml.inference.sklearn_inference import SklearnModelHandlerPandas
from apache_beam.ml.inference.pytorch_inference import PytorchModelHandlerTensor
from apache_beam.ml.inference.pytorch_inference import PytorchModelHandlerKeyedTensor

model_handler = SklearnModelHandlerNumpy(model_uri='model.pkl',
                                          model_file_type=ModelFileType.JOBLIB)

model_handler = PytorchModelHandlerTensor(state_dict_path='linear_regression.pth',
                                            model_class=PytorchLinearRegression,
                                            model_params={'input_dim': 1, 'output_dim': 1})
KeyedModelHandler

```python
from apache_beam.ml.inference.base import KeyedModelHandler

keyed_model_handler = \
    KeyedModelHandler(PytorchModelHandlerTensor(...))

with pipeline as p:
    data = p | beam.Create(["img1", np.array([[1, 2, 3], [4, 5, 6], ...])],
                            "img2", np.array([[1, 2, 3], [4, 5, 6], ...]),
                            "img3", np.array([[1, 2, 3], [4, 5, 6], ...])],
                      ])

    predictions = data | RunInference(keyed_model_handler)
```
Creating Ensembles

- A/B Pattern
- Sequential Pattern
A/B Pattern

with pipeline as p:
    data = p | 'Read' >> beam.ReadFromSource('a_source')
    model_a_predictions = data | RunInference(ModelHandlerA)
    model_b_predictions = data | RunInference(ModelHandlerB)
Sequential Pattern

```python
with pipeline as p:
    data = p | 'Read' >> beam.ReadFromSource('a_source')
model_a_predictions = data | RunInference(ModelHandlerA)
model_b_predictions = (model_a_predictions | RunInference(ModelHandlerB))
```
class PostProcessor(beam.DoFn):
    def process(self, element: Tuple[str, PredictionResult]):
        key, prediction_result = element
        inputs = prediction_result.example
        predictions = prediction_result.inference

        # Post-processing logic
        result = ...
        yield (key, result)

with pipeline as p:
    output = (
        p | 'Read' >> beam.ReadFromSource('a_source')
        | 'PytorchRunInference' >> RunInference(KeyedModelHandler)
        | 'ProcessOutput' >> beam.ParDo(PostProcessor()))
## Metrics

- **Namespaces**
- **num_inferences**
- **Count, min, max, mean of**
  - batch_size
  - msec_per_batch
  - inference_batch_latency_microsecs
  - inference_request_batch_byte_size
  - inference_request_batch_size
  - load_model_latency_millisecs
  - model_byte_size

### Metrics in the UI of a Dataflow job

#### Custom counters

<table>
<thead>
<tr>
<th>Counter name</th>
<th>Value</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>batch_size_MEAN</td>
<td>1</td>
<td>PyTorch RunInference/.../</td>
</tr>
<tr>
<td>msec_per_batch_MEAN</td>
<td>14,146</td>
<td>PyTorch RunInference/.../</td>
</tr>
<tr>
<td>inference_batch_latency_microsecs_MEAN</td>
<td>13,625,472</td>
<td>PyTorch RunInference/.../</td>
</tr>
<tr>
<td>inference_request_batch_byte_size_MEAN</td>
<td>16</td>
<td>PyTorch RunInference/.../</td>
</tr>
<tr>
<td>inference_request_batch_size_MEAN</td>
<td>1</td>
<td>PyTorch RunInference/.../</td>
</tr>
<tr>
<td>load_model_latency_millisecs_MEAN</td>
<td>6,347</td>
<td>PyTorch RunInference/.../</td>
</tr>
<tr>
<td>model_byte_size_MEAN</td>
<td>402,484,199</td>
<td>PyTorch RunInference/.../</td>
</tr>
</tbody>
</table>
RunInference in the future

- Optional batching
- Streaming with side inputs
- Integration with remote services (e.g. Vertex AI)

More frameworks! Please help contribute!

TensorRT  dmIn  JAX

Coming soon!  XGBoost  And more!
Related Links

- Machine Learning in Beam
  https://beam.apache.org/documentation/sdks/python-machine-learning/
- RunInference Transform
  https://beam.apache.org/documentation/transforms/python/elementwise/runinference/
- Pipeline Examples
  https://github.com/apache/beam/tree/master/sdks/python/apache_beam/examples/inference
- RunInference Python Documentation
Thank you!

Demo time!