Scalable training and inference with Ray AIR

Key problems of existing ML infrastructure

- Scaling is hard, especially for data scientists.
- Platforms solutions can limit flexibility.
- But custom distributed apps are too hard.

Ray AI Runtime (AIR) is a scalable toolkit for end-to-end ML applications

- Scalable data prep + loading
  - Dataset library built for ML tasks
  - Seamlessly load distributed data from MB to TB scale
  - Preprocessors for unified training<>inference

- Scalable model training
  - Single API to run the most popular ML training frameworks
  - Seamless integration with other AIR libraries

- Scalable hyperparameter tuning
  - Run multiple concurrent Training jobs
  - Cutting edge optimization algorithms
  - Fault tolerance at scale

- Scalable batch prediction
  - Execute inference on distributed data using CPUs and GPUs
  - Bring your own model or load existing checkpoints from Train

- Scalable online inference
  - Deploy single models as HA inference services in Ray
  - Build multi-model pipelines with custom business logic

Why Ray AIR for PyTorch

- Seamless move from laptop to cluster
- Orchestrates distributed training
- Fault tolerance
- Multi GPU training
- Works with libraries (e.g. PyTorchLightning)
- Submit your Ray AIR training script via TorchX
- Use your existing training loop - scale with Ray

Example Code:

```python
import ray
from ray.data import Dataset, read_parquet
from ray.data.preprocessors import MinMaxScaler
from ray.train.torch import Trainer, ScalingConfig
from ray.train.torch import TorchTrainer
from ray.air import tune
from ray.air.torch.train import train_loop
from ray.air.train.torch.trainer import TorchTrainer
from ray.air.preprocessor import MinMaxScaler
from ray.air.preprocessors import MinMaxScaler

# Dataset
dataset = ray.data.read_parquet("...")
preprocessor = \
    ray.data.preprocessors.MinMaxScaler(["value")

# Trainer
trainer = ray.train.TorchTrainer(\n    train_loop,\n    scaling_config=ScalingConfig(num_workers=100)\n    preprocessor=preprocessor,\n    dataset=dataset)
result = trainer.fit()

# Tuner
param_space = {"batch_size": tune.grid_search([1, 2, 3])}
tuner = Tuner(\n    trainer,\n    param_space=param_space)
results = tuner.fit()

# Predictions
checkpoint = results.get_best_result().checkpoint
predictor = BatchPredictor.from_checkpoint(checkpoint, TorchPredictor)(...)
predictions = predictor.predict(dataset)
predictions.write_parquet("s3://...")
```

AIR's BATCH PREDICTOR

- Prediction using distributed data using CPUs and GPUs
- Load existing checkpoints from Train

RAY SERVE

- Deploy single models as HA inference services in Ray
- Build multi-model pipelines with custom business logic

Example:

```python
depl = PredictorDeployment.options(\n    name="TorchService")
depl.deploy(TorchPredictor, checkpoint, ...)
print(deployment.url)
```