Dask

- Opis
- Dostupne verzije
- Dokumentacija
- Korištenje
- Primjeri
  - Dask-ML
  - scikit
  - PyTorch
  - TensorFlow

Opis

Dask je "fleksibilna knjižnica namijenjena paralelizaciji proračuna u Pythonu". Osim što je usmjerena razvoju paralelnog koda, glavna uloga je omogućiti lagano skaliranje tipičnih data science problema i aplikacija na klaster, koje se tipično razvijaju na osobnim računalima. Ovo postiže kroz kroz imitaciju poznatijih API-ja usmjerenih obradi podataka (poput Numpya i Pandasa), oslanjanje na integriran raspoređivač poslova i činjenicu da je u potpunosti napisana u Pythonu.

Glavna su joj sučelja:

- Array - obrada tenzora s Numpy API-jem
- DataFrame - obrada strukturiranih podataka s Pandas API-jem
- Bag - obrada lista i iteratora namijenjeno tekstualnim podacima, JSON datotekama ili Python objektima
- Delayed - direktna paralelizacija koda funkcijama ili dekoratorima
- Futures - paralelizacija namijenjena izvršavanju poslova koji se mogu vršiti istodobno

Jedna od srodnih knjižnica je i Dask-ML, koja je namijenjena distribuiranom strojnom učenju putem poznatog scikit API-ja i koja omogućava skaliranje na više čvorova putem knjižnice joblib, s kojom scikit paralelizira svoje algoritme. Više o tipičnim problemima koji se rješavaju i primjerima korištenja svakog od sučelja možete naći na online stranicama Daska.

Dostupne verzije

verzija	modul	Supek	Padobran
2023.7.0	scientific/dask/2023.7.0-ghcr	✔	

Korištenje aplikacije na Supeku

Python aplikacije i knjižnice na Supeku su dostavljene u obliku kontejnera i zahtijevaju korištenje wrappera kao što je opisano ispod.

Više informacija o python aplikacijama i kontejnerima na Supeku možete dobiti na sljedećim poveznicama:

- Python, pip i conda
- Apptainer

Dokumentacija

- Službena stranica https://www.dask.org/
- Dokumentacija https://docs.dask.org/en/stable/
- Dask-ML https://ml.dask.org/
    - scikit https://ml.dask.org/joblib.html
    - Skorch (PyTorch) https://skorch.readthedocs.io/en/stable/
    - Scikeras (TensorFlow) https://adriangb.com/scikeras/stable/

Korištenje

Za korištenje na Supeku, skriptu python potrebno je pokrenuti wrapperom **dask-launcher.sh** kojim se stvara Dask klaster i putem kojeg se distribuiraju poslovi korištenjem Client API-ja.

Primjer skripte PBS

```
#!/bin/bash

# aktiviraj dask
module load scientific/dask/2023.7.0-ghcr

# pokreni python program
dask-launcher.sh moj_program.py
```

Primjer skripte python

```
# pozovi python modul
import os
from dask.distributed import Client

# spoji klijenta
client = Client(os.environ['SCHEDULER_ADDRESS'])

# ostatak programa
...
```

Primjeri

Ispod se nalaze primjeri korištenja Daska za najpoznatije ML knjižnice (SciKit, PyTorch i TensorFlow) kao i njegovo izvorno ML sučelje Dask-ML .

Dask-ML

dask-kmeans.sh

```
#!/bin/bash

#PBS -l select=4:ncpus=4
#PBS -l place=scatter

# environment
module load scientific/dask/2023.7.0-ghcr

# cd
cd ${PBS_O_WORKDIR:-""}

# run
dask-launcher.sh dask-kmeans.py
```

dask-kmeans.py

```
import os
import time
import pprint

from dask.distributed import Client
import dask_ml.datasets
import dask_ml.cluster

def main():

    # spoji klijenta putem datoteke scheduler.json
    client = Client(os.environ['SCHEDULER_ADDRESS'])

    # kreiraj podatke
    n_clusters = 10
    n_samples = 3*10**7
    n_chunks = int(os.environ['PMI_SIZE'])
    X, _ = dask_ml.datasets.make_blobs(centers = n_clusters,
                                       chunks = n_samples//n_chunks,
                                       n_samples = n_samples)

    # izraunaj
    km = dask_ml.cluster.KMeans(n_clusters=n_clusters)
    now = time.time()
    km.fit(X)
    print('GB: %f' % (int(X.nbytes)/1073741824))
    print('elapsed fit: %f' % (time.time()-now))

    # shutdown
    client.shutdown()

if __name__ == '__main__':
    main()
```

scikit

scikit-svc.sh

```
#!/bin/bash

#PBS -l select=4:ncpus=4
#PBS -l place=scatter

# environment
module load scientific/dask/2023.7.0-ghcr

# cd
cd ${PBS_O_WORKDIR:-""}

# run
dask-launcher.sh scikit-svc.py
```

scikit-svc.py

```
import os
import time
import joblib
import pprint
import numpy as np

from dask.distributed import Client

from sklearn.datasets import load_digits
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVC

def main():

    # client
    client = Client(os.environ['SCHEDULER_ADDRESS'])

    # data
    digits = load_digits()

    # model
    param_space = {
        'C': np.logspace(-6, 6, 30),
        'tol': np.logspace(-4, -1, 30),
        'gamma': np.logspace(-8, 8, 30),
        'class_weight': [None, 'balanced'],
    }
    model = SVC(kernel='rbf')
    search = RandomizedSearchCV(model,
                                param_space,
                                cv=3,
                                n_iter=1000,
                                verbose=10)

    # fit
    with joblib.parallel_backend('dask'):
        search.fit(digits.data,
                   digits.target)

    # shutdown
    client.shutdown()

if __name__ == '__main__':
    main()
```

PyTorch

pytorch-skorch.sh

```
#!/bin/bash

#PBS -l select=2:ngpus=2:ncpus=2
#PBS -l place=scatter

# environment
module load scientific/dask/2023.7.0-ghcr

# cd
cd ${PBS_O_WORKDIR:-""}

# run
dask-launcher.sh pytorch-skorch.py
```

pytorch-skorch.py

```
import os
import time
import pprint
import numpy as np

import torch
import torch.nn as nn
import torch.optim as optim

from dask.distributed import Client
from joblib import parallel_backend

from skorch import NeuralNetClassifier

from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV

class MyModule(nn.Module):
    def __init__(self, num_units=10, nonlin=nn.ReLU()):
        super().__init__()
        self.dense0 = nn.Linear(20, num_units)
        self.nonlin = nonlin
        self.dropout = nn.Dropout(0.5)
        self.dense1 = nn.Linear(num_units, num_units)
        self.output = nn.Linear(num_units, 2)

    def forward(self, X, **kwargs):
        X = self.nonlin(self.dense0(X))
        X = self.dropout(X)
        X = self.nonlin(self.dense1(X))
        X = self.output(X)
        return X

def main():

    # client
    client = Client(os.environ['SCHEDULER_ADDRESS'])

    # vars
    n_samples = 256*100
    batch_size = 256
    max_epochs = 10

    # data
    X, y = make_classification(n_samples, 20, n_informative=10, random_state=0)
    X = X.astype(np.float32)
    y = y.astype(np.int64)

    # net
    net = NeuralNetClassifier(module = MyModule,
                              max_epochs = max_epochs,
                              criterion = nn.CrossEntropyLoss,
                              batch_size = batch_size,
                              train_split = None,
                              device = 'cuda')
    # search
    search = GridSearchCV(net,
                          param_grid = {'max_epochs': [1, 3, 10, 30],
                                        'module__num_units': [1, 10, 100, 1000],
                                        'module__nonlin': [nn.ReLU(), nn.Tanh()]},
                          scoring = 'accuracy',
                          error_score = 'raise',
                          refit = False,
                          verbose = 3,
                          cv = 3)

    # fit
    now = time.time()
```

```
    with parallel_backend('dask'):
        search.fit(X, y)
    print('fit elapsed in %0.2f' % (time.time()-now))

    # shutdown
    client.shutdown()

if __name__ == "__main__":
    main()
```

TensorFlow

tensorflow-scikeras.sh

```
#!/bin/bash

#PBS -l select=1:ngpus=2:ncpus=2
#PBS -l place=scatter

# environment
module load scientific/dask/2023.7.0-ghcr

# cd
cd ${PBS_O_WORKDIR:-""}

# run
export TF_FORCE_GPU_ALLOW_GROWTH="true"
dask-launcher.sh tensorflow-scikeras.py
```

tensorflow-scikeras.py

```
import os
import time
import pprint
import numpy as np

import tensorrt
from tensorflow import keras
from scikeras.wrappers import KerasClassifier

from dask.distributed import Client
from joblib import parallel_backend

from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV

def get_model(hidden_layer_dim, meta):
    # note that meta is a special argument that will be
    # handed a dict containing input metadata
    n_features_in_ = meta["n_features_in_"]
    X_shape_ = meta["X_shape_"]
    n_classes_ = meta["n_classes_"]

    model = keras.models.Sequential()
    model.add(keras.layers.Dense(n_features_in_, input_shape=X_shape_[1:]))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
```

```
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(hidden_layer_dim))
    model.add(keras.layers.Activation("relu"))
    model.add(keras.layers.Dense(n_classes_))
    model.add(keras.layers.Activation("softmax"))
    return model

def main():

    # client
    client = Client(os.environ['SCHEDULER_ADDRESS'])

    # vars
    n_samples = 256*1000
    batch_size = 256
    max_epochs = 10

    # data
    X, y = make_classification(1000, 20, n_informative=10, random_state=0)
    X = X.astype(np.float32)
    y = y.astype(np.int64)

    # clf
    clf = KerasClassifier(get_model,
                          loss="sparse_categorical_crossentropy",
                          hidden_layer_dim=100)

    # gs
    params = {
        "hidden_layer_dim": [50, 100, 200],
        "loss": ["sparse_categorical_crossentropy"],
        "optimizer": ["adam", "sgd"],
        "optimizer__learning_rate": [0.0001, 0.001, 0.1],
    }
    gs = GridSearchCV(clf,
                      param_grid = {
                          "hidden_layer_dim": [50, 100, 200],
                          "loss": ["sparse_categorical_crossentropy"],
                          "optimizer": ["adam", "sgd"],
                          "optimizer__learning_rate": [0.0001, 0.001, 0.1],
                      },
                      refit=False,
                      cv=3,
                      scoring='accuracy')

    # fit
    now = time.time()
    with parallel_backend('dask'):
        gs.fit(X, y)
    print('fit elapsed in %0.2f' % (time.time()-now))

if __name__ == "__main__":
    main()
```