

Parallel Machine Learning and Artificial Intelligence

Dr. Handan Liu

h.liu@northeastern.edu

Northeastern University

Parallel Machine Learning

Why needs Parallel Machine Learning?

- A few inconvenience with Machine Learning:

- Algorithms
- Hyperparameters
- Cross-validation
- Big model + big data → Big computation costs

Example: Training ResNet-50 on ImageNet (run 90-epochs) using a single NVIDIA K80 GPU takes 14 days.

Most of them require significant amount of CPU, RAM and sometimes GPU in order to be applied efficiently.

→ These requirements can be implemented on a HPC cluster/supercomputer

→ We need to parallelize Machine Learning techniques

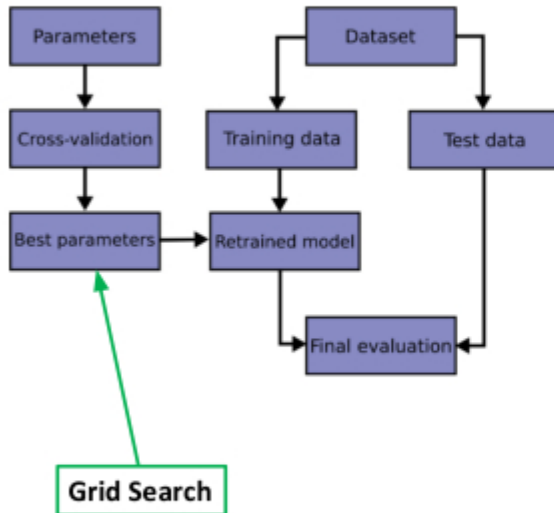
Parallelize Machine Learning Techniques

- What is the best way to parallelize Machine Learning techniques?
- There is no "best way" or “one method” for Machine Learning Parallel.

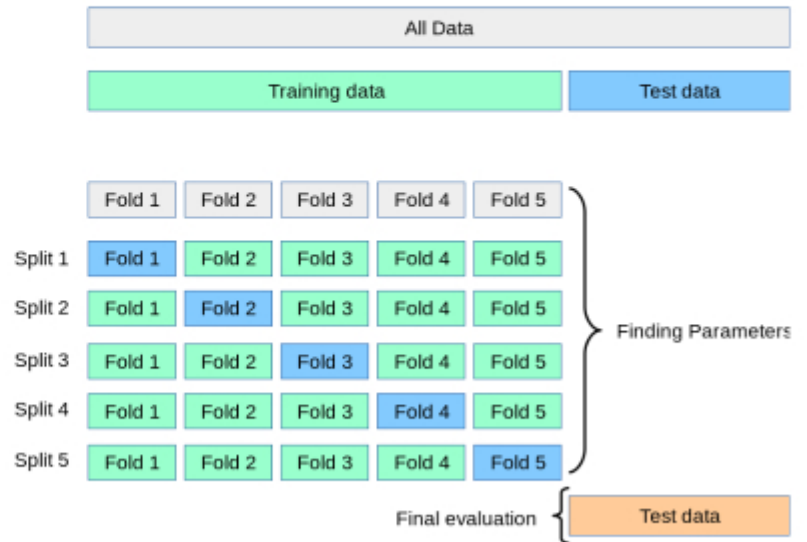
Parallel Approaches in Machine Learning

- In machine learning, the following areas can be parallelized:
 - Cross validating
 - Grid searching

Model Evaluation with Cross Validation



k-fold CV



Implement k -fold Cross-Validation

- K -fold cross-validation has many different formulations and is mainly used for 2 things:
 - Comparing learning algorithms
 - Tuning hyperparameters for a single learning algorithm
- Typically, K is chosen to be 5 or 10 and there is no general theoretical best K to choose.

Implement k -fold Cross-Validation

Algorithm: K-fold Cross-validation

Given learning algorithms A_1, \dots, A_M , and K as the number of folds.

Partition the data into K of approximately equal parts $\mathcal{P}_1, \dots, \mathcal{P}_K$.

(In parallel or not)

for $k \in 1, \dots, K$

(In parallel or not)

for $i \in 1, \dots, M$

$f_{A_i}|_{\mathcal{P}_{-k}} \leftarrow$ Train method A_i to all of the partitions except the k th.

$L(y, f_{A_i}|_{\mathcal{P}_{-k}}(X))|_{\mathcal{P}_k} \leftarrow$ Compute the loss over the k th partition.

for $i \in 1, \dots, M$

$L_{CV}(y, f_{A_i}(X)) \leftarrow \frac{1}{K} \sum_{k=1}^K L(y, f_{A_i}|_{\mathcal{P}_{-k}}(X))|_{\mathcal{P}_k}$

- This algorithm is parallelized by construction, and the computational complexity is dependent on the training algorithms' complexity.
- K-fold CV and repeated K-fold CV can be parallelized in Embarrassingly Parallel.
- The k -fold cross-validation supported in scikit-learn supports multiprocessing and others.
- Usually multiprocessing.Pool and Process can be used for k -fold CV.





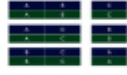


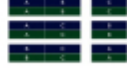

Tuning the Hyperparameters with Grid Search

For example, SVM Classifier

The param_1: 1, 10, 100

The param_2: 1e3, 1e4, 1e5

- Typical examples include C , $kernel$ and $gamma$ for Support Vector Classifier, $alpha$ for Lasso, etc.
- It is possible and recommended to search the hyperparameter space for the best cross validation score.

(1, 1e3) 	(10, 1e3) 	(100, 1e3) 
(1, 1e4) 	(10, 1e4) 	(100, 1e4) 
(1, 1e5) 	(10, 1e5) 	(100, 1e5) 

Implement Parallel in Scikit-learn

- Some scikit-learn estimators and utilities can parallelize costly operations using multiple CPU cores, thanks to the following components:
 - via the joblib library. In this case the number of threads or processes can be controlled with the `n_jobs` parameter.
 - via OpenMP, used in C or Cython code.

Scikit-Learn Joblib-based Parallelism

- Joblib is able to support both multi-processing and multi-threading. Whether joblib chooses to spawn a thread or a process depends on the backend that it's using.
- When the underlying implementation uses joblib, the number of workers (threads or processes) that are spawned in parallel can be controlled via the `n_jobs` parameter.

Joblib: running Python functions as pipeline jobs

- Joblib is a set of tools to provide **lightweight pipelining in Python**:
 1. transparent disk-caching of functions and lazy re-evaluation (memoize pattern)
 2. **easy simple parallel computing**
- Joblib is optimized to be **fast** and **robust** on large data in particular and has specific optimizations for *numpy* arrays.
- Joblib – Embarrassingly parallel for loops
 - Ability to use shared memory efficiently for large *numpy-based data structures*.

Loky

- Scikit-learn generally relies on the loky backend, which is joblib's default backend.
- Loky is a multi-processing backend.
- When doing multi-processing, in order to avoid duplicating the memory in each process (which isn't reasonable with big datasets), joblib will create a memmap that all processes can share, when the data is bigger than 1MB.

NumPy memmap in joblib.Parallel

- joblib.Parallel provides a special handling for large arrays to automatically dump them on the filesystem and pass a reference to the worker to open them as memory map on that file using the numpy.memmap subclass of numpy.ndarray.
- If your code can release the GIL, passing prefer='threads' is even more efficient because of less overhead for communication.
- Scientific Python libraries such as numpy, scipy, pandas and scikit-learn often release the GIL in performance critical code paths.
- In addition, some of the NumPy routines that are used internally by scikit-learn may also be parallelized if NumPy is installed with specific numerical libraries such as MKL, OpenBLAS, or BLIS.

Numpy Memmap: <https://numpy.org/doc/stable/reference/generated/numpy.memmap.html>

Main Parameters in `joblib.Parallel`

- Supported backends are:
 - 'loky': single-host, process-based parallelism (used by default in new version)
 - 'multiprocessing': previous process-based backend based on *multiprocessing.Pool*.
 - 'threading': single-host, thread-based parallelism
- The `n_jobs` parameter controls the number of workers (threads or processes) in parallel.
 - If `n_jobs = -1`, all CPUs are used.
 - If `n_jobs = 1`, no parallel computing code is used at all, which is useful for debugging.
 - For `n_jobs = <integer number>`, use this number of CPUs for parallel

```
from joblib import parallel_backend  
  
with parallel_backend('threading', n_jobs=2):  
    # Your scikit-learn code here
```

'multiprocessing' vs 'loky'

```
import ...  
  
def function1(...):  
    ...  
  
def function2(...):  
    ...  
  
...  
if __name__ == '__main__':  
    # do stuff with imports and functions defined about  
    ...
```

Under Windows, when using multiprocessing.Pool:

No code should *run* outside of the "if __name__ == '__main__'" blocks, only imports and definitions.

The 'loky' backend used by default in *joblib* 0.12 and later does not impose this anymore.

Using the 'multiprocessing' backend can cause a crash when using third-party libraries that manage their own native processes pool if the library is first used in the main process and subsequently called again in a worker process (inside the `joblib.Parallel` call).

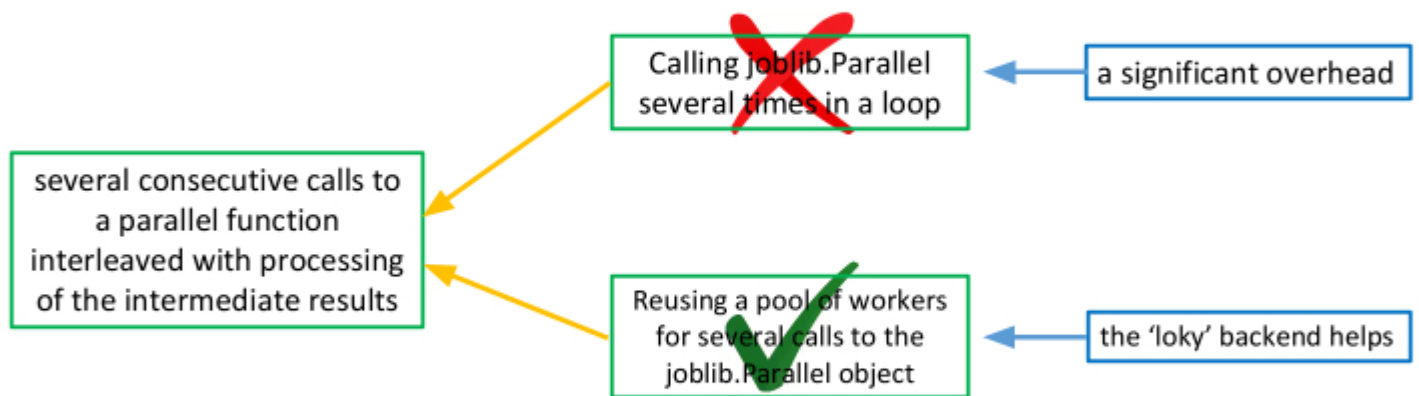


Using the backend 'loky'

'loky' – Reusable Process Pool Executor

- Consistent and robust spawn behavior:
 - All processes are started using **fork + exec** on POSIX systems.
- Reusable executor:
 - strategy to avoid re-spawning a complete executor every time.
 - A singleton executor instance can be reused (and dynamically resized if necessary) across consecutive calls to limit spawning and shutdown overhead.
 - The worker processes can be shutdown automatically after a configurable idling timeout to free system resources.
- No need for `if __name__ == "__main__":` in scripts:
 - it is not required to protect the code calling parallel functions under Windows.
-

Reusing a pool of workers: joblib.Parallel



Working with numerical data in shared memory

- When `n_jobs != 1`, the Python processes are forked using the multiprocessing module of Python standard library.
 - The arguments are serialized and reallocated in the memory of each process.
- This can be problematic for large arguments as they will be reallocated `n_jobs` times by the workers.
- Especially often occurs in Numpy-based datastructure.

Scikit-Learn OpenMP-based Parallelism

- OpenMP is used to parallelize code written in Cython or C, relying on multi-threading exclusively.
- By default, the implementation will use as many threads as possible.
- You can control the exact number of threads that are used via the OMP_NUM_THREADS environment variable:
 - \$ OMP_NUM_THREADS=4 python my_script.py

Parallelism in Scikit-Learn Estimators

- Scikit-learn package is integrated with the **joblib** library.
- A common parameter `n_jobs` is used to specify how many concurrent processes (by default) should be used for routines that are parallelized with joblib.

Parallelism in Ensemble machine-learning methods

- `sklearn.ensemble.` :
 - `BaggingClassifier` and `BaggingRegressor`
 - `ExtraTreesClassifier` and `ExtraTreesRegressor`
 - `IsolationForest`
 - `RandomForestClassifier` and `RandomForestRegressor`
 - `RandomTreesEmbedding`

Parallelism in Multilabel/Multiclass Prediction

- `sklearn.multiclass`.
 - `OneVsRestClassifier`
 - `OneVsOneClassifier`
 - `OutputCodeClassifier`

Parallelism in Supervised ML

- Linear Models:
 - Classifiers
 - Regressors
- Nearest Neighbors
 - KNeighborsClassifier
 - KNeighborsRegressor

Parallelism in Unsupervised ML

- Manifold Learning

- Isomap
- MDS
- TSNE
-

- Clustering

- KMeans
- DBSCAN
- SpectralClustering
-

- Stay safe!
- See you next class!

Next Lecture

Continue: Parallel Machine Learning



