Deep Learning Optimisé - Jean Zay

Résultats Imagenet Race et bonnes pratiques

INSTITUT DU DÉVELOPPEMENT ET DES RESSOURCES EN INFORMATIQUE SCIENTIFIQUE

DLO-JZ

5ème partie de la formation de l'IDRIS.

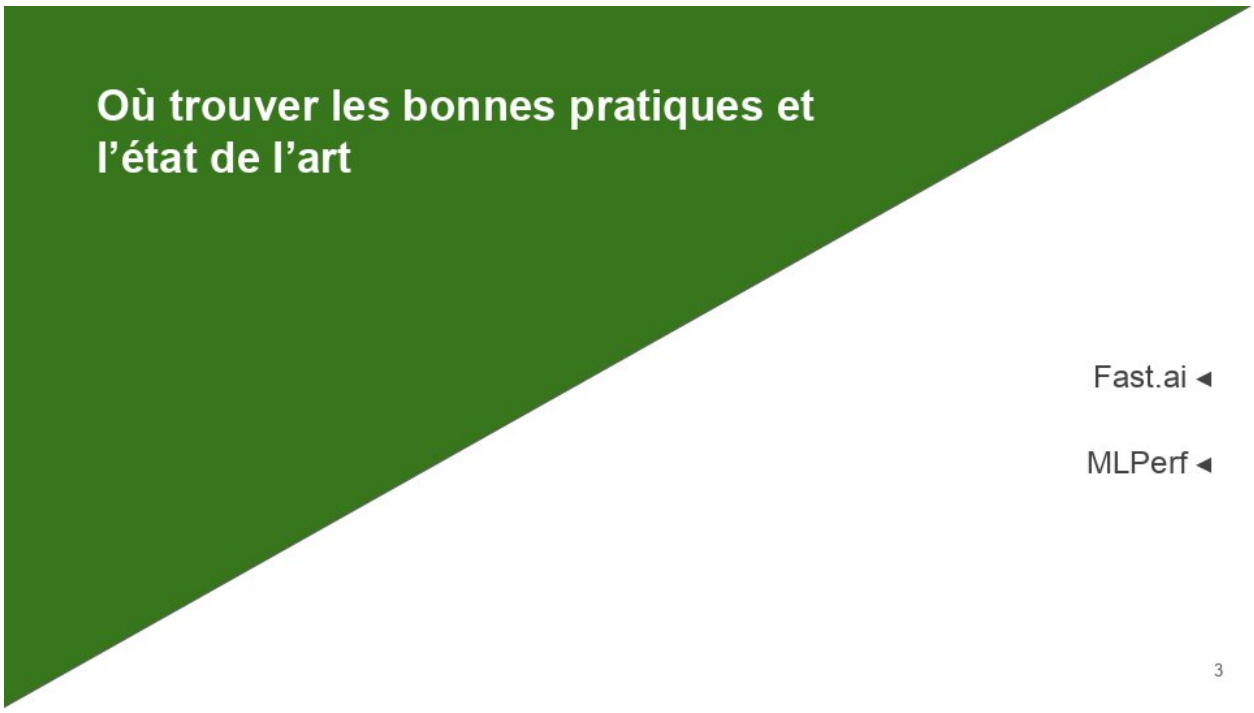
Diapositives commentées.

Auteur : Bertrand Cabot

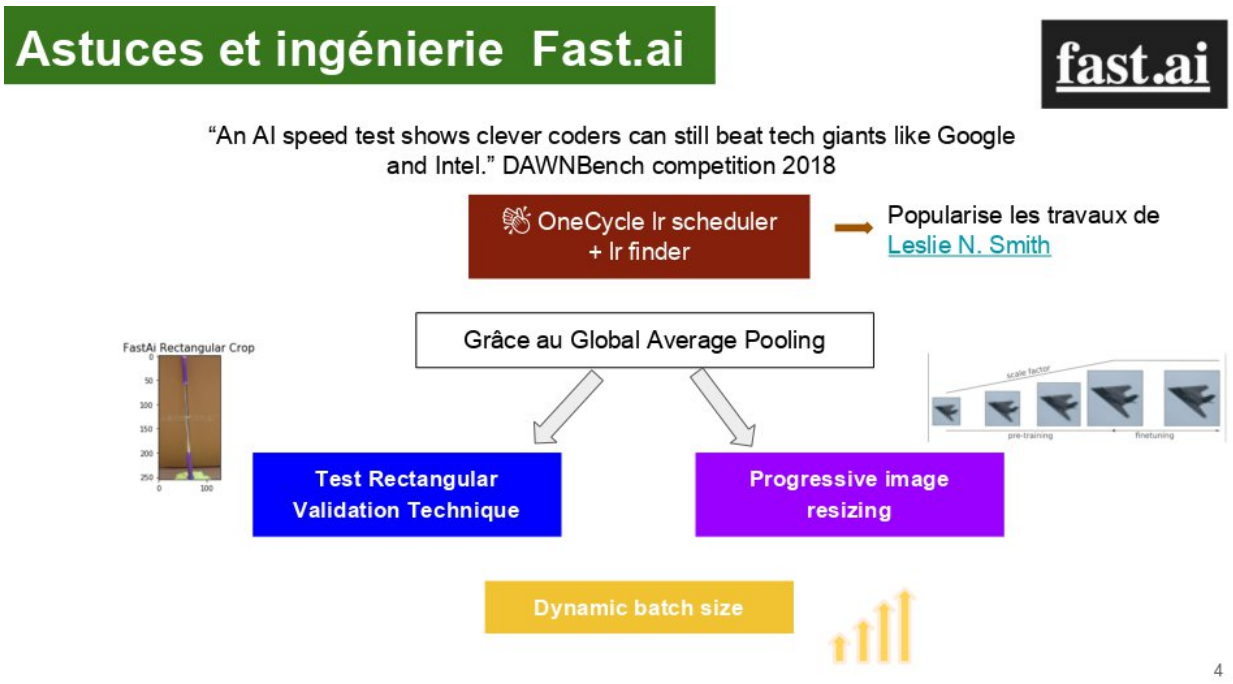
Février 2023

Chapitres :

- Où trouver les bonnes pratiques et l'état de l'art ?
- Hyper Parameters Optimization



Cette partie présente quelques sources d'informations importantes sur les bonnes pratiques et l'état de l'art pour l'accélération de l'apprentissage en Deep Learning.



Fast.ai est une initiative de personnalités du *Queensland University of Technology* ayant pour objectif de démocratiser le Deep Learning grâce à une communauté de développeurs, des cours en ligne, une sur-couche méthodologique de PyTorch avec toutes les bonnes pratiques de l'apprentissage en Deep Learning.
Notamment en 2018, la communauté a participé et remporté la compétition *DAWNBench* devant les géants de la *tech*, montrant ainsi qu'avec des bonnes pratiques, de la méthodologie, de l'intuition et des astuces, il était possible de surclasser les moyens énormes *hardware* des géants de la *tech*.
Ils ont publiés ensuite la recette pour entraîner *« Imagenet from scratch* pour seulement 40 dollars » ou en 18 minutes sur 256 GPU (équivalence du prix de la location d'une instance sur AWS).

Pour ce qui nous intéresse par rapport à notre présentation, ils ont popularisé les travaux de *Leslie N. Smitch* (vus ce matin dans DLO-JZ4) notamment les *OneCycle lr scheduler* et le *lr finder*.
Ils ont poussé le fait d'utiliser le *Global Average Pooling* à la fin des architectures CNN pour pouvoir jouer sur le format des images pendant l'apprentissage. Ainsi, 2 astuces ont été mises en avant :

- l'utilisation d'image rectangulaire pour *booster* la validation
- La possibilité d'agrandir petit à petit la taille des images lors de l'apprentissage pour accélérer au début et affiner à la fin.

De plus, ils ont popularisé aussi le *Dynamic batch size* qui consiste à augmenter la taille du batch progressivement pendant l'apprentissage. Cela peut remplacer ou compléter le *lr scheduler* que nous avons vu.

Tout cela, vers 2018, a été un apport conséquent pour l'accélération et la précision de l'apprentissage en Deep Learning.

ML Perf - Référence pour le Supercomputing en IA

Référence pour l'industrie

- Hardware
- Framework de Deep Learning
- Techniques SOTA

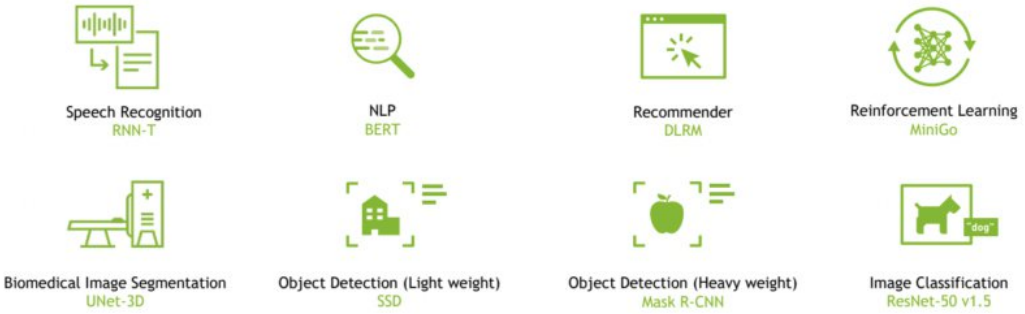
5

Cependant pour avoir une vue plus globale et actuelle sur l'optimisation de l'accélération des différentes applications en Deep Learning, le suite *ML Perf* devient une référence incontournable.

ML Perf est un ensemble de *benchmark* de Deep Learning permettant d'abord un affichage publicitaire des constructeurs de matériel pour comparer les performances des *Hardware*, des *Software* pour l'ensemble des applications du *Deep Learning*. Ce qui va nous permettre, on va le voir ensuite, de rassembler toutes les bonnes pratiques et les méthodologies pour accélérer un apprentissage.

ML Perf - Benchmarks

Training

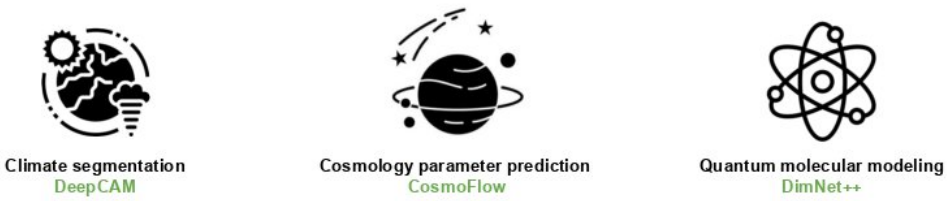


6

La suite de *benchmark ML Perf* propose 8 applications classiques d'apprentissage de Deep Learning.

ML Perf - Benchmarks

Training HPC



Inference :

- Datacenter
- Edge
- Mobile
- Tiny

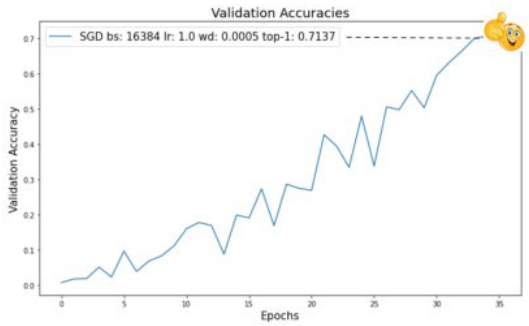
7

Elle comprend aussi :

- 3 applications d'apprentissage de Deep Learning orientées "calcul scientifique",
- Un nombre d'applications d'inférence selon la taille des équipements.

ML Perf - Benchmarks

Training rule



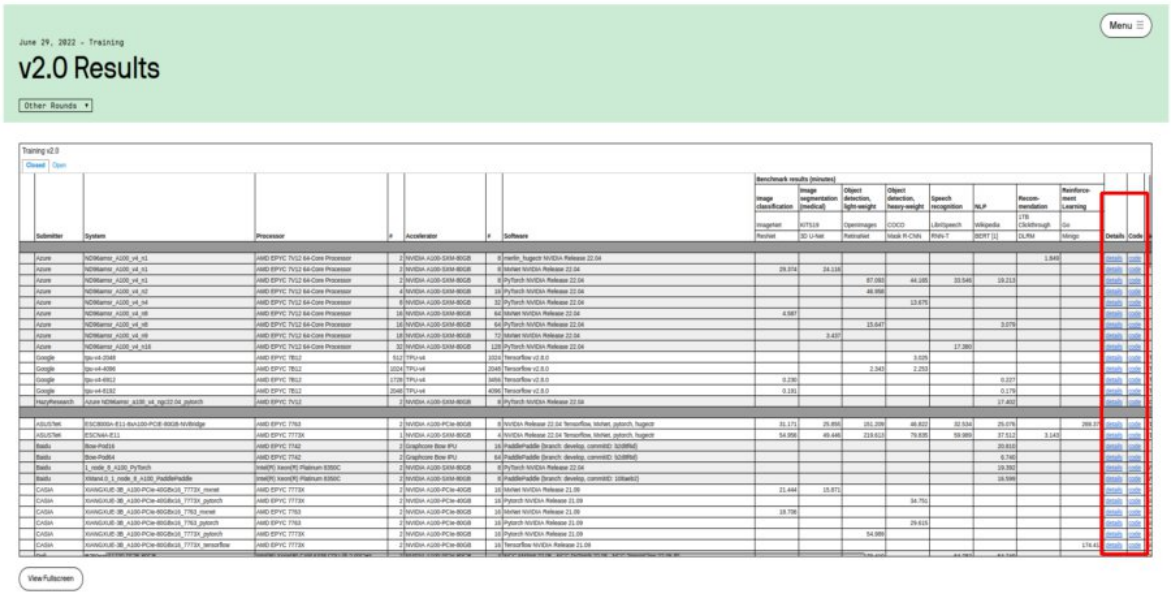
A metric threshold to reach

Time Results

8

Pour valider l'apprentissage et pouvoir publier un temps d'apprentissage dans le tableau de résultat, il faut atteindre un seuil prédéfini pour la métrique d'évaluation de l'apprentissage.

ML Perf – sources pour connaître l'état de l'art



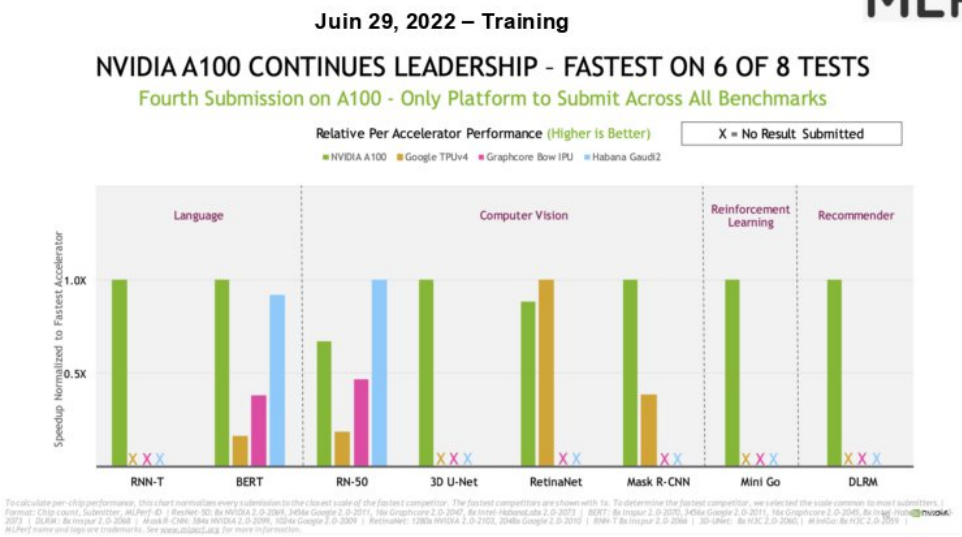
Cela met effectivement en avant les performances des équipements et des frameworks, et pousse surtout la recherche des géants de la *tech*.

La plupart des résultats sont donnés dans la partie *closed*, où un certain nombre de paramètres sont fixés (le framework, l'*optimizer*, etc), ce qui permet surtout de comparer les systèmes entre eux.

Mais il existe aussi une section *open* qui permet à des entités de valoriser leur solution plutôt software ou algorithmique.

Mais pour nous, simple utilisateur, cela nous permet d'avoir une source de codes et de documentations importante sur les bonnes pratiques et méthodologies de l'accélération distribuée, très actualisée.

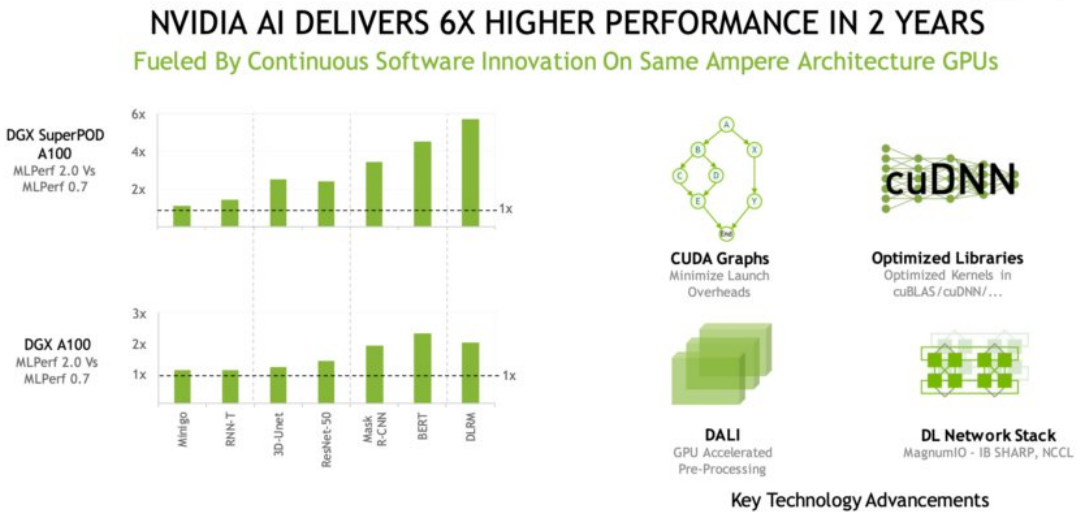
ML Perf - Result v2.0



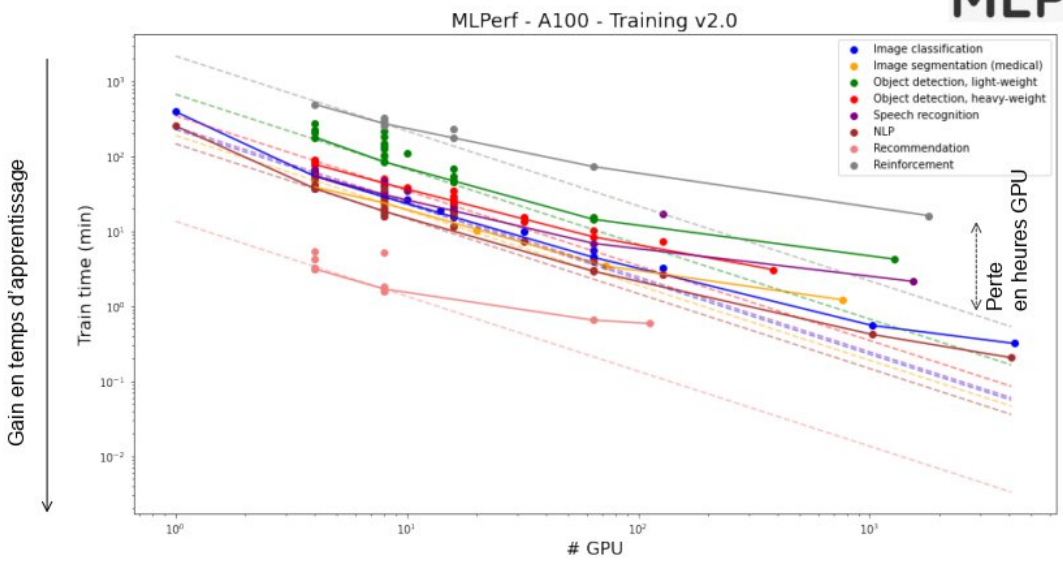
Les résultats permettent de comparer les différents types d'accélérateur IA.

Les résultats permettent aussi de suivre l'évolution du matériel d'accélération (GPU Nvidia) en *Deep Learning*.

ML Perf - Évolution



ML Perf - Scaling



En récupérant les résultats MLPerf, on peut dessiner une représentation du *scaling* de la distribution et des bonnes pratiques associées, sur les supers calculateurs, en fonction du type d'application.

On peut ainsi observer que certaines applications sont sujettes à une distribution plus large sur un nombre important d'accélérateurs GPU par rapport aux autres applications moins sujettes (Classification d'image, NLP).

Dans tous les cas, utiliser plus de GPU en *Data Parallelism* réduit le temps global d'apprentissage. Cependant l'écart entre la courbe réelle et la courbe idéale montre une consommation d'heure GPU et donc une consommation électrique plus importante à résultat équivalent.

De ce point de vue économie énergétique, on voit que pour les Benchmarks MLPerf 1 ou 2 nœuds de calcul correspond au meilleur compromis pour la consommation énergétique.

Frameworks & Optimisation du compileur

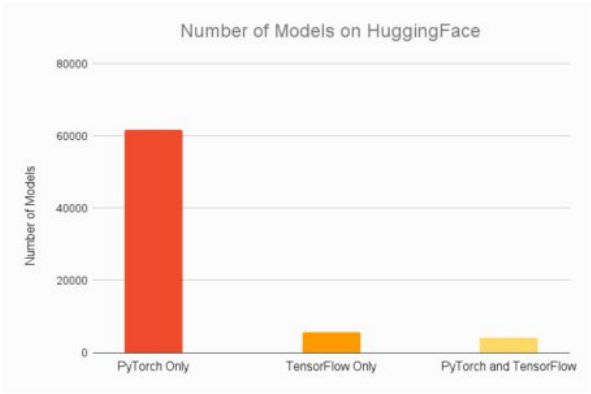
Pytorch vs Tensorflow ◂

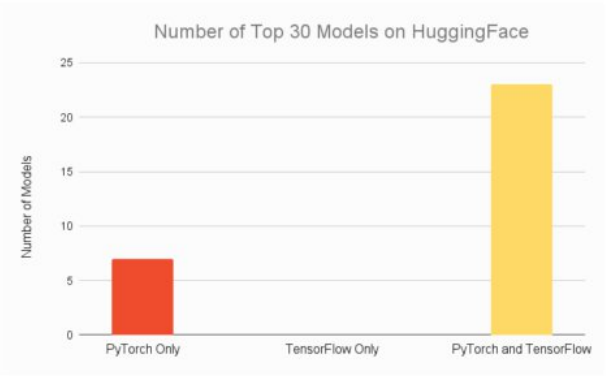
Compilation ◂

JAX ◂

Pytorch 2.0 ◂

Pytorch vs Tensorflow





https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/ 15

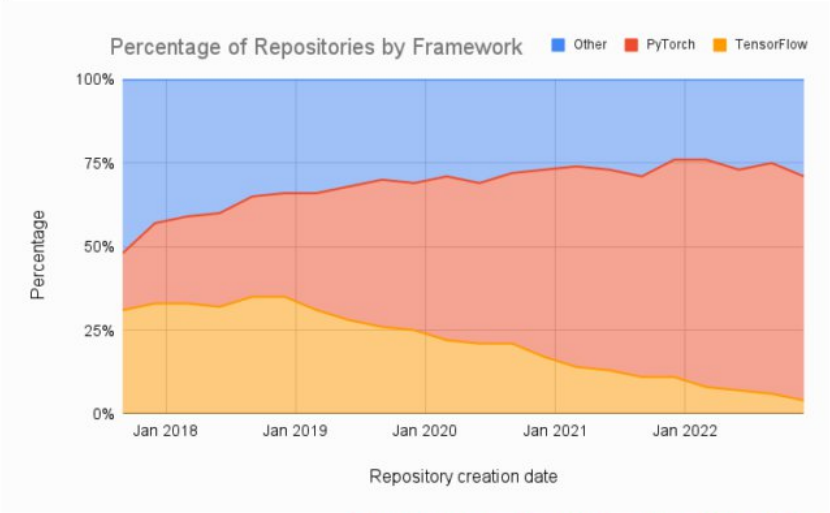
La base de la recherche d'hyper-paramètres en Deep Learning lorsque l'on cherche à massivement paralléliser ses essais sur plusieurs machines afin de gagner du temps, est d'utiliser un **Grid Search**.

Cependant, on ne gagne rien en terme d'économie d'heure GPU. Il est donc préférable d'utiliser du **Random Search** en *Deep Learning*, dans le but de trouver une solution avec moins d'essais et donc d'économiser de la consommation électrique.

En effet, définir aléatoirement et indépendamment chaque paramètre permet de tester davantage de valeurs possibles en moins d'essais.

Pytorch vs Tensorflow

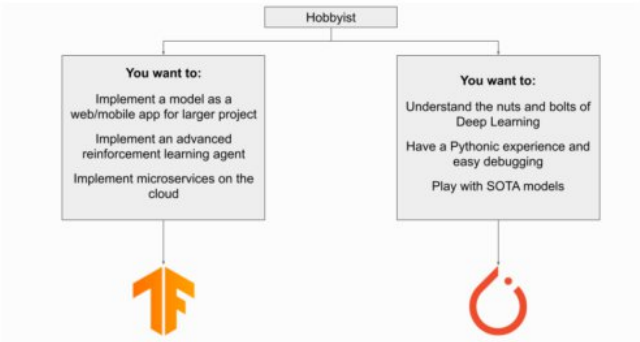
Paper With Code :



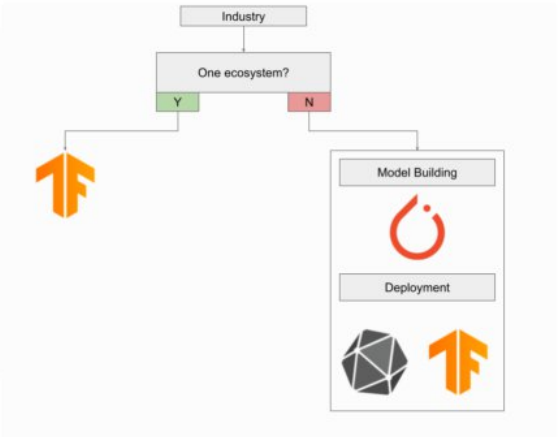
https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/ 16

Pytorch vs Tensorflow

What if I'm in Industry?



What if I'm in Industry?



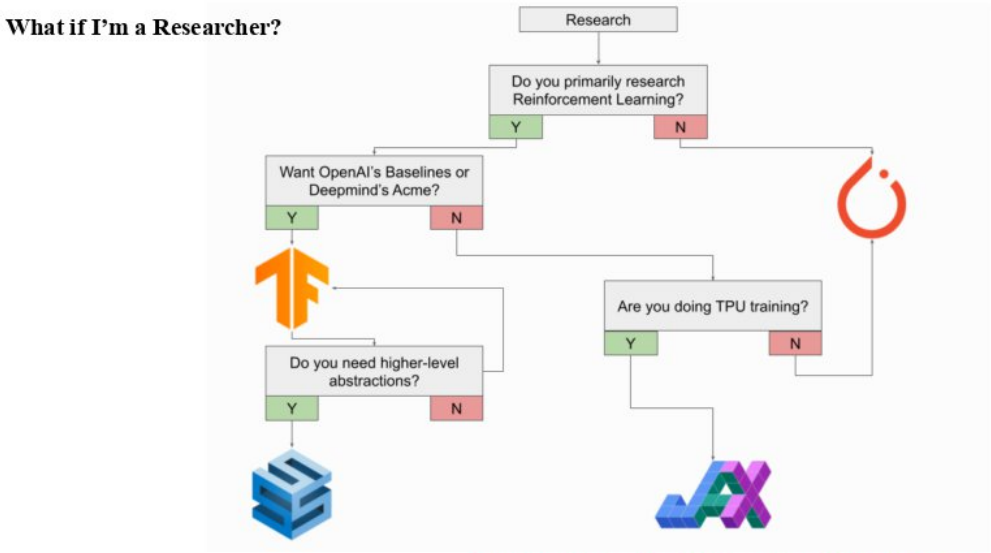
https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/ 17

Ensuite, une excellente technique de recherche d'hyper-paramètres est la *Bayesian Optimization.* Elle permet en peu d'essais de trouver les optimums.

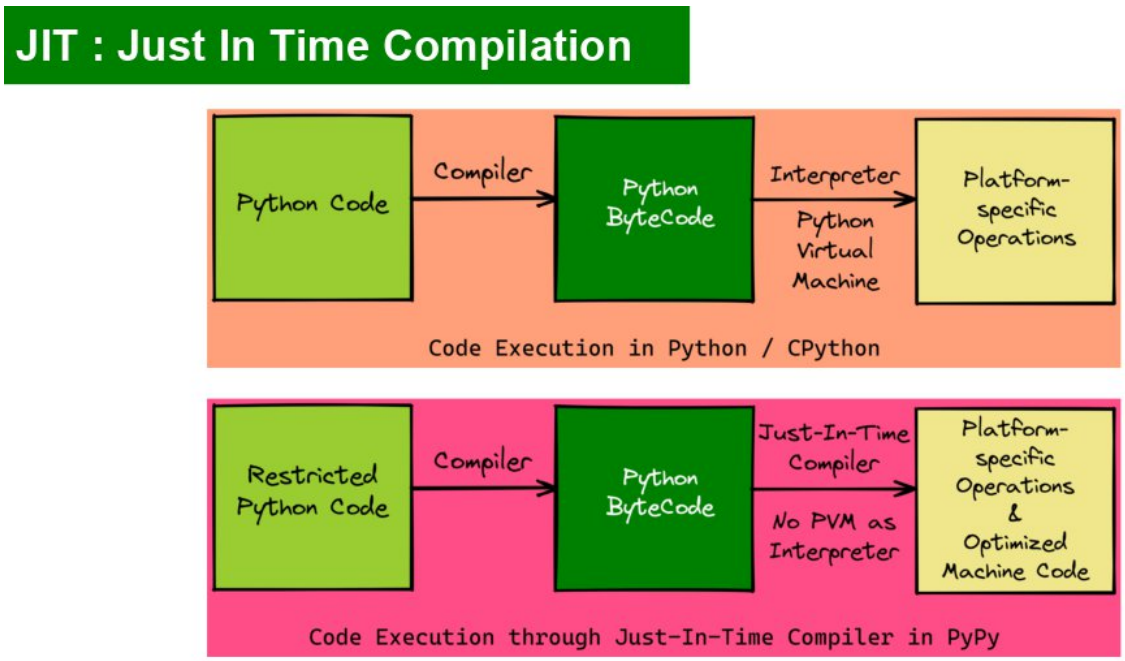
Une approche statistique en fonction des résultats des précédents essais permet de suggérer les hyper-paramètres du prochain essai afin de trouver la meilleure solution rapidement.

Cependant la *Bayesian Optimization* empêche une parallélisation massive des essais. En effet, cette approche est séquentielle puisque chaque essai dépend des résultats des essais précédents.

Pytorch vs Tensorflow vs JAX



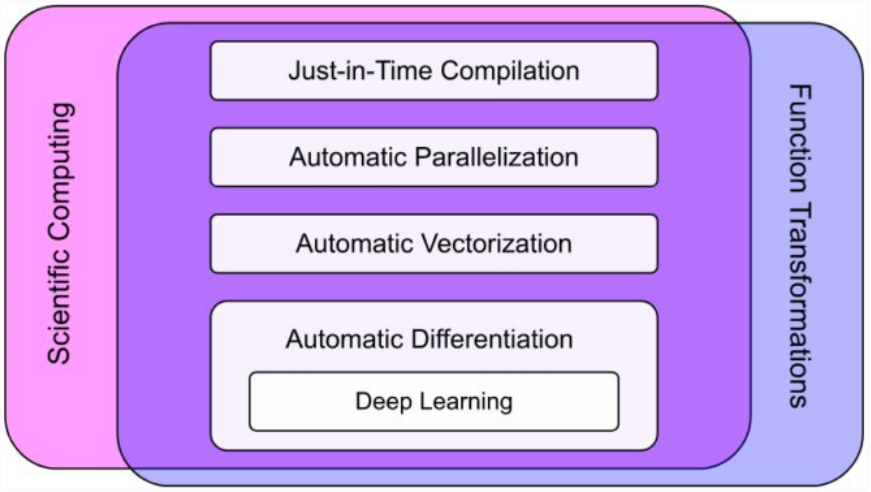
https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/



Le *Population Based Training* est un algorithme d'HPO complexe inspiré des algorithmes génétiques ou généalogiques, adaptés aux gros modèles avec des apprentissages longs et avec des essais parallélisables sur seulement quelques machines.

La recherche d'hyper-paramètres se fait pendant l'apprentissage, par étapes. À chacune de ces étapes de génération de populations, chaque essai parallèle copie les poids de modèle du meilleur essai et applique une *Bayesian Optimization* pour définir un nouvel ensemble d'hyper-paramètres optimisés. Ainsi, à la fin de la séquence totale économe et faiblement parallélisée, on obtient une solution optimale.

Jax



JAX lies at the intersection of Scientific Computing and Function Transformations, yielding a wide range of capability beyond the ability to train Deep Learning models

20

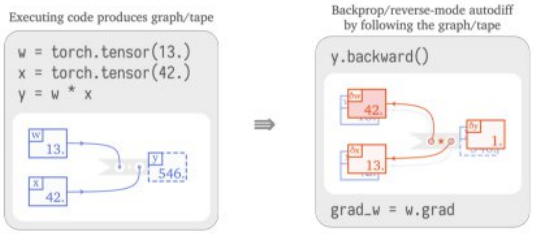
Un autre algorithme d'HPO « état de l'art » et économe est le *Successive Halving Algorithm* (SHA), ou **ASHA** pour sa version avec des essais massivement parallélisables qui profite intelligemment de toute la ressource de calcul disponible.

Il s'agit de mener des essais parallèles et d'arrêter par étapes la moitié des essais les moins prometteurs.

Cet algorithme est ce qui se fait de mieux pour des petits ou moyens modèles avec des essais d'apprentissage pas trop longs.

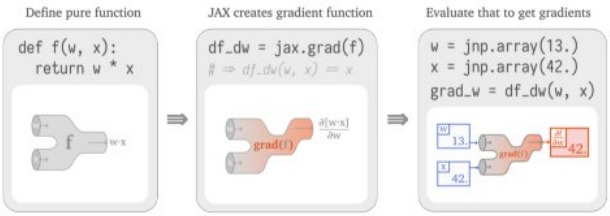
Pytorch vs Jax

PyTorch



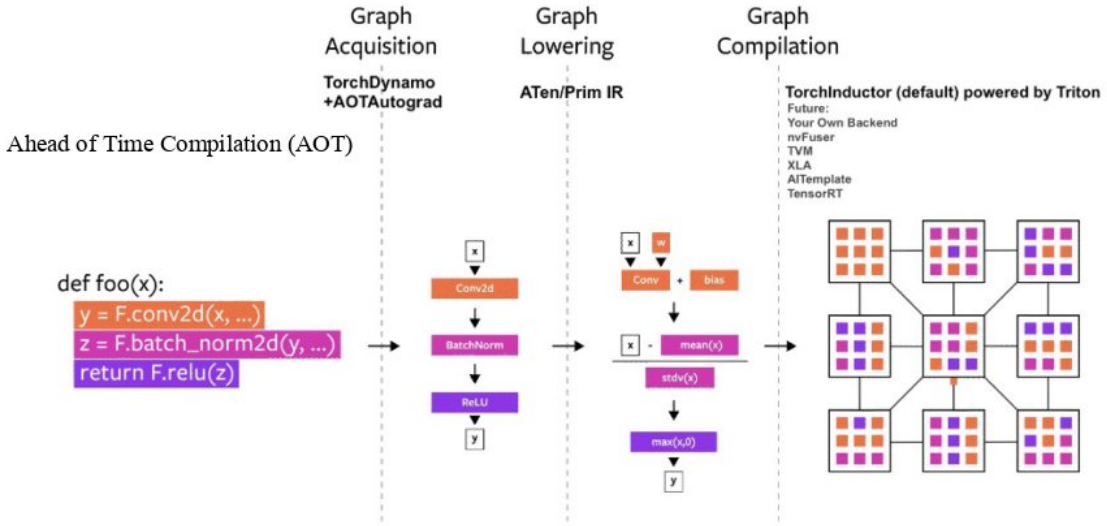
Pros :
Pythonic, dynamic, popular framework,
NVIDIA GPU oriented, Jean Zay adapted
Cons :
Slower compare to some other frameworks (Jax, Mxnet, ...)

JAX



Pros :
Fast Computation
numpy-like, functional programming,
Hessian computation (2nd order) efficiency
Cons :
Google/TPU oriented, Jean Zay unadapted

Pytorch 2.0



Pytorch 2.0

```
# API NOT FINAL
# default: optimizes for large models, low compile-time
#          and no extra memory usage
torch.compile(model)

# reduce-overhead: optimizes to reduce the framework overhead
#                and uses some extra memory. Helps speed up small models
torch.compile(model, mode="reduce-overhead")

# max-autotune: optimizes to produce the fastest model,
#               but takes a very long time to compile
torch.compile(model, mode="max-autotune")
```

Pytorch 2.0