

CIFAR10 hyperlightspeedbench

I.Alioua - J.Bellet - G.Klajer

École Polytechnique

14 mars 2023






- 1 Présentation du problème
 - État de recherche actuel
 - Notre première approche
- 2 Notre algorithme
 - Nos différences
 - Réalisation de l'ablation study
- 3 Annexes
 - Chronologie du projet
 - References

Performances actuelles

Training Time

[All Submissions](#)

Objective: Time taken to train an image classification model to a test accuracy of 94% or greater on CIFAR10.

Rank	Time to 94% Accuracy	Model	Framework	Hardware
1 	0:00:10	Custom Resnet 9 <i>Santiago Akle Serrano, Hadi Pour Ansari, Vipul Gupta, Dennis DeCoste</i> source	Pytorch 1.1.0	Tesla V100 * 8 GPU / 32 GB / 40 CPU
2 	0:00:11	Custom ResNet 9 <i>Ajay Uppili Arasanipalai</i> source	PyTorch 1.1.0	IBM AC922 + 4 * Nvidia Tesla V100 (NCSA HAL)
3 	0:00:28	Kakao Brain Custom ResNet9 <i>clint@KakaoBrain</i> source	PyTorch 1.1.0	Tesla V100 * 4 GPU / 488 GB / 56 CPU (Kakao Brain BrainCloud)



DawnBench - Performances du code proposé dans le projet

Sur Google Colab on obtient ...

epoch	train_loss	val_loss	train_acc	val_acc	ema_val_acc	total_time_seconds
0	1.4834	1.3932	0.6836	0.7208		6.0767
1	1.2861	1.2713	0.7793	0.7941		12.2093
2	1.1582	1.1496	0.8516	0.8554		18.4445
3	1.1143	1.1157	0.8711	0.8734		24.7386
4	1.0674	1.0770	0.8965	0.8943		30.9409
5	1.0371	1.0510	0.9199	0.9049		37.0578
6	0.9980	1.0284	0.9473	0.9168		43.1090
7	0.9849	1.0030	0.9414	0.9311		49.1220
8	0.9165	0.9956	0.9824	0.9329	0.9368	55.1438
9	0.9175	1.0035	0.9844	0.9312	0.9386	61.1752



Ambition 1 - Adapter le code sur un autre dataset : Imagenette

Pour ce faire la première étape était de faire un DataLoader capable d'importer et préparer les images.
Toutefois, on observe des problèmes de compatibilité entre la librairie fastai et torchvision...



Redéfinition de nos objectifs

- Partir de zéro sur la dataset CIFAR10 afin de mieux saisir les enjeux d'optimisation
- Réaliser une ablation study à partir de notre algorithme "naïf" en se basant sur le guide proposé par Myrtle.ai [1] :
 - Baseline
 - Relation entre batchsize et learning rate
 - Influence des hyperparamètres
- Eventuellement généraliser à CIFAR100



- Réalisation d'un ResNet "classique" en suivant l'architecture proposée par le code de Hire Tysam
- Découverte et implémentation de l'outil **cProfile**
- Tentative de visualisation des résultats obtenus avec **SnakeViz**
- Optimisation basée sur les rapports de performance obtenus
 - Maximisation des opérations réalisées directement sur les tenseurs
 - Pre-processing sur GPU



```
class Add(nn.Module):
    def __init__(self, modules: OrderedDict[str, nn.Module]) -> None:
        super().__init__()

        for name, module in modules.items():
            self.add_module(name, module)

    def forward(self, x: Tensor):
        return sum([module(x) for module in self.children()])

class Id(nn.Module):
    def forward(self, x: Tensor): return x

class Flatten(nn.Module):
    def forward(self, x: Tensor): return x.view(x.size(0), x.size(1))
```


Temps d'exécution des fonctions

3948348 function calls (3903181 primitive calls) in 41.679 seconds

Random listing order was used

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
8993	0.002	0.000	0.002	0.000	{method 'values' of 'collections.OrderedDict' objects}
3220	0.001	0.000	0.001	0.000	{method 'items' of 'collections.OrderedDict' objects}
175040	0.043	0.000	0.043	0.000	{method 'startswith' of 'str' objects}
396	0.001	0.000	0.505	0.001	{method 'format' of 'str' objects}
6347	0.001	0.000	0.001	0.000	{method 'add' of 'set' objects}

Figure – Texte de sortie



Résultats obtenus

Num epoch	Temps total (s)	Accuracy (%)
0	45.35	10.54
1	90.46	39.13
2	135.76	56.85
3	180.80	64.45
4	226.55	69.88
5	271.63	67.06
6	316.73	72.25
7	362.51	75.21
8	407.92	79.02
9	453.14	77.37
10	498.62	79.41



Temps d'exécution des fonctions

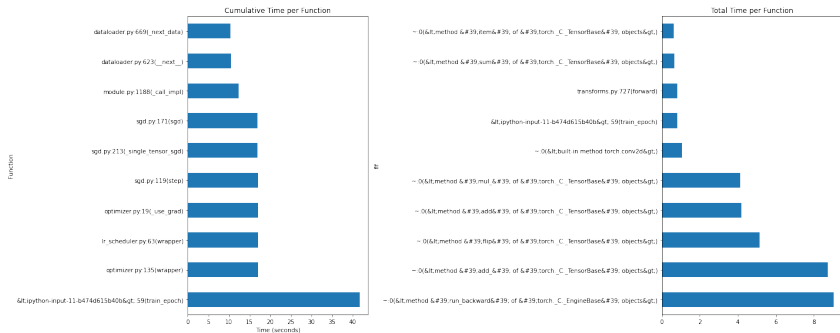


Figure – Cumulative time

- Influence de la batchsize

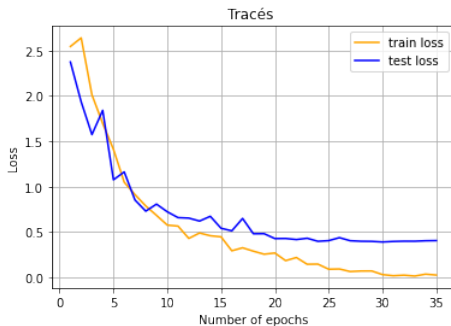


Figure – Batchsize = 512, LR de référence

Mini batches

- Notre algorithme est moins performant pour un faible nombre d'epochs
- Les valeurs asymptotiques sont sensiblement identiques avec Myrtle AI [1].
- On observe que le réseau retient essentiellement les dernières images passées

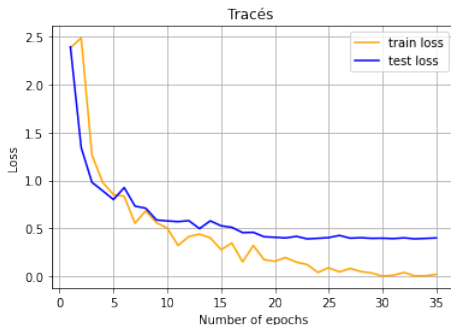


Figure – Batchsize = 128, LR de référence

Mini batches

- En augmentant le batchsize, on augmente la vitesse d'apprentissage (risque d'instabilité toutefois)
- Un learning rate élevé avec petit batch size risque de mal entraîner un dataset plus grand (CIFAR100)

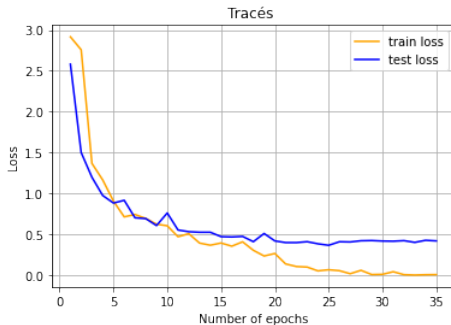


Figure – Batchsize = 128, LR 4 fois plus grands

Mini batches

- Jusqu'à l'epoch 15, notre algorithme sous-performe légèrement
- Moins forte convexité

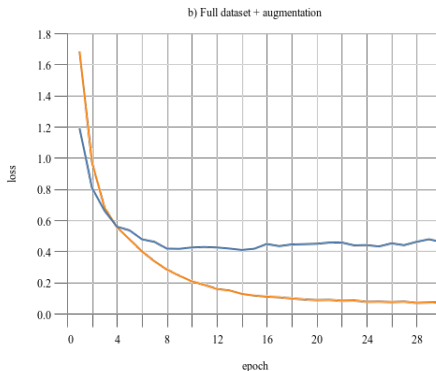


Figure – Résultat de Myrtle Ai

- On trace une réalisation

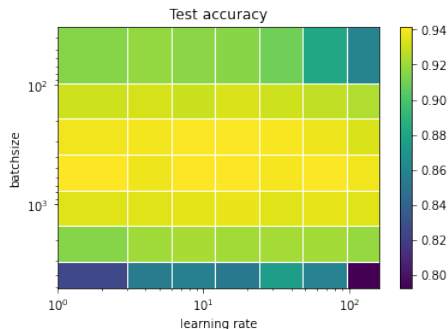


Figure – Accuracy en fonction de la batchsize et LR

Hyperparamètres

- On sous-performe pour des valeurs extrêmes de batchsize et learning rate

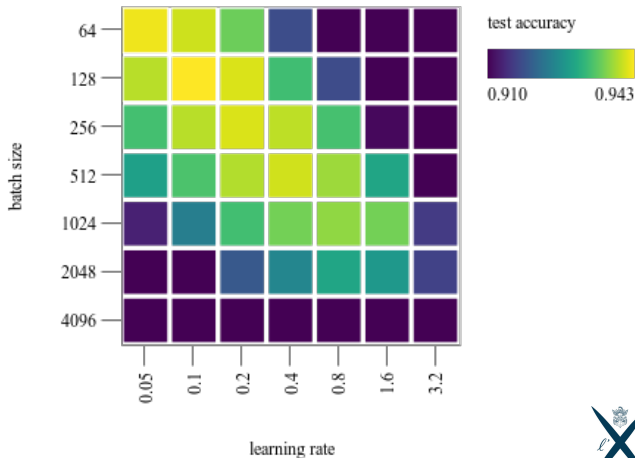


Figure – Accuracy en fonction de la batchsize et LR, version Myrtle AI

- Recherche très chronophage en temps d'exécution
- On aurait aimé étudier l'influence du momentum
- Utiliser une moyenne pour afficher la fonction

- Influence de la batchnorm sur la rapidité, même si plusieurs difficultés potentielles
 - Plus de sensibilité à l'initialisation
 - Difficile de comprendre ses effets d'interaction
 - Dépendance au type de layer
 - La lenteur
- Changer la fonction d'activation ReLU en une autre
- Utiliser un whitening sur les inputs

- Techniques mises en place :
 - Pre-process sur GPU
 - Impact de la BatchSize
 - Étude des HyperParametres
- Résultats obtenus :
 - Environ 91% d'accuracy
 - 25 minutes de durée d'apprentissage

Chronologie du projet

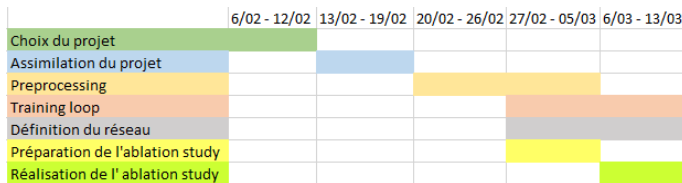


Figure – Vue synthétique de la chronologie du projet

- [1] Myrtle.ai. *How to Train Your ResNet*.
<https://myrtle.ai/learn/how-to-train-your-resnet/>.