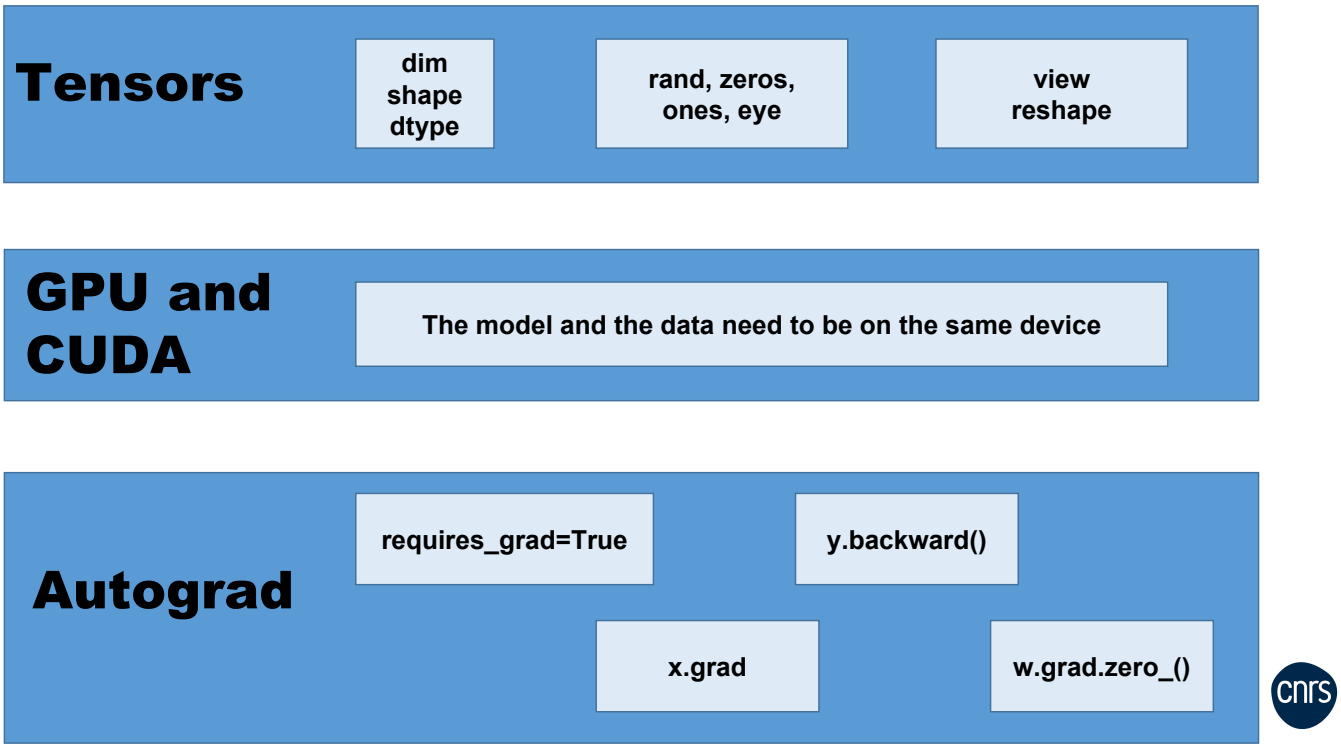
TP2 Pytorch Takeaways



IDRIS



1. Tensors

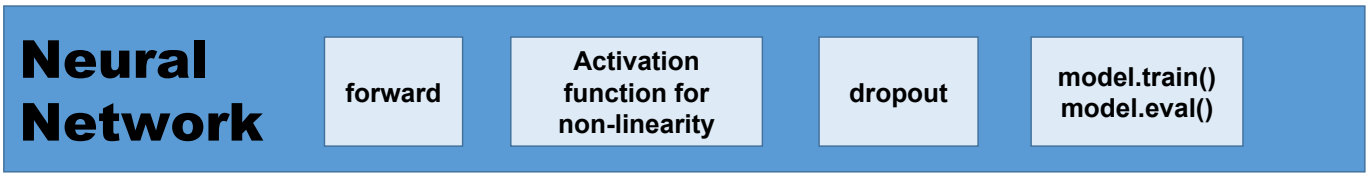
- Un tenseur est un tableau de nombres qui peut aussi tourner sur GPU.
- Trois propriétés à toujours regarder : dim, shape, dtype. (Une grande partie des erreurs PyTorch vient d'une mauvaise forme.)
- 2 façons de créer un tenseur : à partir de valeurs (torch.tensor) ou avec un constructeur (rand, zeros, ones, eye).
- view et reshape changent la forme sans changer les données. reshape est plus souple.
- Le broadcasting permet d'opérer sur des tensors de dimensions différentes

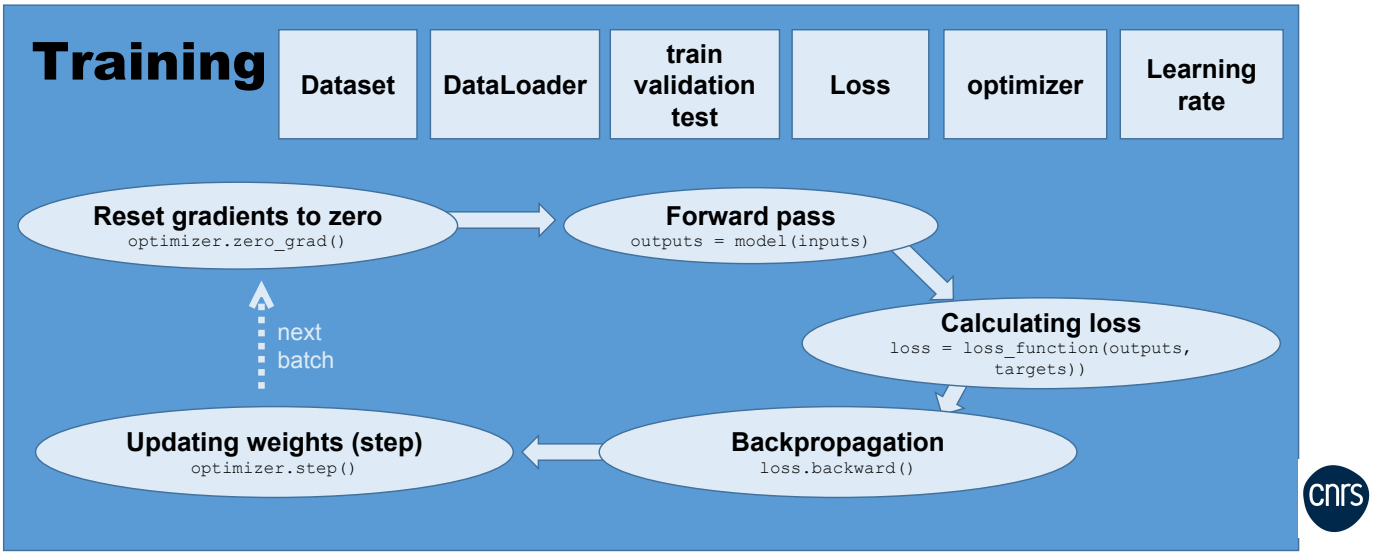
2. GPU et CUDA

- Les données et le modèle doivent être sur le même device, sinon PyTorch lève une erreur.

3. Autograd

- requires_grad=True demande à PyTorch d'enregistrer les opérations. Sans cela, .grad reste None.
- y.backward() remonte le graphe depuis la sortie et calcule les gradients. (Elle ne renvoie rien.)
- x.grad est l'endroit où le résultat est stocké, sur l'entrée.
- Les gradients s'accumulent à chaque backward(). Il faut donc les remettre à zéro (zero_(), ou optimizer.zero_grad() dans la boucle d'entraînement).





4. Construire un réseau (nn.Module)

- __init__ déclare les couches, forward décrit le chemin des données.

Sans activation (ReLU), empiler des couches linéaires revient à une seule couche linéaire. Il faut donc de la non-linéarité.

- Le dropout est une régularisation : il désactive des neurones au hasard à l'entraînement.,

pour éviter que le réseau de neurones apprenne trop bien les données d'entraînement au point de mal généraliser sur de nouvelles données.

- model.train() et model.eval() changent le comportement de certaines couches . En eval, les sorties redeviennent déterministes.
- Le modèle doit aussi être envoyé sur le bon device avec .to(device).

5. Entraînement

- Le **Dataset** organise les données et associe chaque entrée à sa bonne réponse.
- Le **DataLoader** prépare ces données pour les envoyer au réseau par petits groupes (mini-batchs).
- On sépare train et validation pour vérifier que le modèle généralise.
- La **loss** mesure l'erreur (ici MSE).
- L'optimiseur met à jour les poids (ici SGD, learning rate 0,01).
- **La boucle d'entraînement suit toujours le même cycle : zero_grad → forward → loss → backward → maj des poids.**
- Quand on a fait tous les batchs = 1 époque
- La courbe de loss permet de vérifier que le modèle apprend.

6. Conclusion du notebook (pourquoi les résultats ne sont pas parfaits)

- Activation linéaire —> Équivaut à un seul perceptron, frontière linéaire
- Non-linéarité —> Indispensable pour les frontières complexes
- Profondeur / largeur —> Plus de capacité, mais plus de coût et risque de surapprentissage
- Régularisation —> Limite le surapprentissage (L1 : parcimonie, L2 : réduction des poids)
- Learning rate —> Compromis vitesse / stabilité, idéalement décroissant
- Batch size —> Petit : bruité mais rapide ; grand : stable mais lent