TD 8: premiers pas avec pytorch

1 Premiers pas avec PyTorch

PyTorch est un ens. de librairies python pour l'apprentissage profond, développé principalement par Facebook, mais en open source. C'est une réimplémentation en python de torch, initialement développé à l'IDIAP (Suisse) par Collobert, Bengio et Mariéthoz.
PyTorch utilise des librairies C sous-jacentes pour d'efficacité.

Une particularité de PyTorch par rapport à TensorFlow (également très utilisé, développé par Google) est que le graphe de calcul peut être défini dynamiquement.

Pour commencer, étudiez les tutoriels:

https://pytorch.org/tutorials/beginner/deep_learning_nlp_tutorial.html

En particulier, suivez pas à pas :
https://pytorch.org/tutorials/beginner/nlp/pytorch_tutorial.html#sphx-glr-beginner-nlp-pytorch-tutorial-py

Puis ce tutoriel créant un simple réseau log-linéaire, prenant en entrée un BOW.
https://pytorch.org/tutorials/beginner/nlp/deep_learning_tutorial.html#sphx-glr-beginner-nlp-deep-learning-tutorial-py

Objectifs de TP:

- introduire une variable pour la taille du mini-batch (qui ici est de 1)
- introduire une couche cachée avec activation tanh

Quelques informations pour commencer:

- Le module python à importer est `torch`.
- Le module gérant les réseaux de neurones est `torch.nn`.

- le type de données central est tensor, pour représenter ... des tenseurs, mais qui est conçu pour conserver un historique de la manière dont il est créé
- si z est un tenseur obtenu en sommant deux tenseurs x et y
- alors utiliser z.backward() va mettre à jour x.grad, en lui ajoutant le gradient de z par rapport à x (et idem pour y)

- Pour définir un réseau de neurones: déclarer une classe héritant de nn.Module
  - par exemple MyNetwork
  - **constructeur:**
    - doit commencer par appeler le constructeur de la superclasse:
      - ***super(MyNetwork, self).__init__()***
    - on peut ensuite déclarer les différentes couches qui composent le réseau
    - pour chaque couche, les paramètres à apprendre sont répertoriés automatiquement, voir MyNetwork.parameters()
  - **propagation avant**:
    - il faut implémenter impérativement la méthode **forward** qui doit contenir le code pour la propagation avant du réseau
    - mais NB: bizarrerie, si mynet est une instance de MyNetwork, alors on appelle pas directement **mynet.forward(…)** , on doit utiliser **mynet(….)** appeler
  - **propagation arrière:**
    - ➔ la rétropropagation n'a pas besoin d'être explicitée, pytorch la calcule pour nous!
  - NB: toutes les méthodes forward de classes héritant de nn.Module prennent en entrée un mini-batch d'exemples et pas un seul exemple
  - REM: pour mieux lire le tutoriel, voici comment passer d'un exemple contenu dans un tenseur x à un minibatch de taille 1 contenant x
    - x.unsqueeze(0)
    - x.view(1, -1)
  -
- déclarer un optimiseur optim (par ex. SGD), avec comme arguments les paramètres du réseau (= ce que l'on veut apprendre)

- une itération d'entraînement pour un mini-batch d'exemples consistera alors en:
  - remettre à zéro tous les gradients des paramètres
  - calculer la perte loss pour le mini-batch courant
  - appeler loss.backward() pour calculer le gradient par rapport à tous les paramètres
  - mettre à jour les paramètres (méthode step de l'optimiseur : optim.step())