INSA - ModIA, 4^e année, Pierre Weiss.

Overfitting et surparamétrisation

1 Objectifs pédagogiques

Ce TP représente une ouverture sur une piste de recherche actuelle pour comprendre le fonctionnement des réseaux de neurones. Nous allons ici reproduire une expérience numérique du papier suivant et essayer de comprendre ce qu'on peut en tirer. Ce TP a plusieurs ambitions :

- Améliorer la maîtrise du langage PyTorch.
- Explorer de manière critique le concept d'"overfitting".
- Comprendre le phénomène de double descente et les effets surprenants d'une surparamétrisation.
- Introduire le modèle "random ReLU Feature" (un réseau de neurones simple à une couche).
- Définir une procédure d'entraînement et continuer à explorer la dynamique des algorithmes d'optimisation.
- Comprendre certaines parties du papier de Mikhail Belkin, *Fit without fear: remarkable mathematical phenomena of deep learning through the prism of interpolation*, Acta Numerica 2021.

2 Random ReLU features

Nous allons ici appliquer un réseau de neurone à une couche très simple appelé random ReLU feature. Il possède la forme suivante :

$$h(x,w) = \sum_{d=1}^{D} w_d \phi_d(x) \quad \text{où} \quad \phi_d(x) = \max(x - b_d, 0). \tag{1}$$

Nous nous intéresserons à une application en 1D, i.e. où $x \in \mathbb{R}$. Ceci permettra de mieux visualiser certains phénomènes, même si les mêmes observations peuvent être reproduites en dimension arbitraire. N'hésitez pas à le faire par exemple sur un problème de classification MNIST si ces phénomènes vous surprennent. Dans notre cadre, les vecteurs b_d sont tirés indépendamment et uniformément au hasard sur $[-1, 1]$. C'est pour cette raison qu'on emploie le terme "random" feature.

1. Tracez sur papier les fonctions $\phi(\cdot)$ pour différentes valeurs de $b_d \in \mathbb{R}$. Est-ce qu'elle vous rappellent une fonction connue en apprentissage profond ?
2. Quelle est la structure de la fonction h ? Comment interpréter les poids w_k ?
3. Pouvez-vous déterminer la structure de la fonction h si $x \in \mathbb{R}^P$, $P \in \mathbb{N}$ et si $\phi_d(x) = \max(\langle x, v_d \rangle - b_d, 0)$, où v_d vit sur la sphère unité ?

3 Mise en place sous PyTorch

Dans ce TP, nous allons complètement nous affranchir de numpy. C'est une librairie de calcul pratique et plutôt efficace. Cependant, PyTorch offre des fonctionnalités similaires et plusieurs de ses fonctions sont mieux optimisées. De plus, cette librairie permet de travailler sur GPU ou CPU sans effort.

```
import matplotlib.pyplot as plt
import torch
from torch import nn
import torch.optim as optim

#%% GPU or CPU ?
use_cuda=torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
if use_cuda :
    dtype = torch.cuda.FloatTensor
else:
    dtype = torch.FloatTensor
```

```
print("GPU: ", use_cuda)
```

Nous définissons aussi la fonction f à apprendre sur l'intervalle $[-1, 1]$ dans le code ci-dessous (vous pouvez la modifier si vous le souhaitez).

```
#%% The function to learn
def f(X) :
    return (torch.abs(X)*torch.sin(2*2*torch.pi*X)).type(dtype)
```

Préliminaires

1. Recopiez ce bout de code.
2. Tracez la fonction f.
3. Générez un jeu d'apprentissage (x_n, y_n) en échantillonnant la fonction f suivant les positions $x_n = -1 + 2n/N$ où N est le nombre total de mesures d'entraînement et $y_n = f(x_n)$. L'objectif ici va être d'approcher la fonction f à partir des points d'échantillonnage (x_n, y_n).

Considérons le problème d'optimisation du risque empirique suivant :

$$\inf_{w \in \mathbb{R}^D} R_N(w) \quad \text{avec} \quad R_N(w) = \frac{1}{2N} \sum_{n=1}^{N} |h(x_n, w) - y_n|_2^2. \tag{2}$$

Un peu de géométrie

1. Déterminez une condition sur les coefficients b_d pour qu'il existe un poids w tel que $R_N(w) = 0$.
2. Est-ce que le problème (2) est convexe ? Différentiable ? Est-ce que la solution est unique ?

Opérations sur les tenseurs PyTorch permet de faire des opérations sur des tenseurs de taille différentes. Pour essayer de comprendre ce mécanisme, je vous propose l'expérience suivante :

1. Construisez un tenseur A de taille 3×1 et un tenseur B de taille 1×4.
2. Pouvez-vous calculer $A + B$ en PyTorch ? Quelle opération est effectuée ?
3. Construisez maintenant un tenseur A de taille 3 et un tenseur B de taille 4. Calculez $A[:, None] + B[None, :]$. Comprenez comment l'opération est effectuée.

Le code principal Nous sommes prêts à attaquer l'entraînement.

1. Construisez une classe PyTorch définissant le modèle h. Il prend la forme suivante, où il faut remplacer les ... par un bout de code [1] :

```
class one_layer_NN(nn.Module):
def __init__(self, D = 3):
    super(one_layer_NN, self).__init__()
    self.b = ...
    self.w = ...
def forward(self, x):
    ...
```

2. En prenant exemple sur le TP2, codez un algorithme d'optimisation stochastique pour minimiser le risque empirique sous PyTorch.
3. Prenez soin d'afficher la fonction $h(\cdot, w)$ ainsi que son adéquation aux données tous les 1, 10, 100 ou 1000 itérations. Ceci permet de vérifier le fonctionnement du code et de trouver un pas (un learning rate) efficace.

1. Vous aurez besoin d'utiliser la fonction *torch.nn.Parameter* pour définir les paramètres à optimiser, ainsi qu'une construction de type *[None, :]* pour coder efficacement certaines opérations PyTorch. Allez voir la documentation.

4 Analyse des résultats

Nous sommes maintenant prêts à analyser les propriétés d'approximation de notre réseau de neurones simpliste.

1. Posez $N = 10$ par exemple (ou une autre valeur assez petite). Affichez les données ainsi que la fonction à retrouver avec MatPlotLib.
2. Calculez la solution de (2) pour $D = 3, 5, 10$ (régime sous-paramétré), $D = 15, 20, 30$ (régime limite), $D = 50, 100, 1000$ (régime sur-paramétré). [2]
3. Qu'observez vous dans chaque cas ? Pour ce problème, quel est le régime qui fournit les meilleurs résultats qualitatifs ?
4. On va essayer de quantifier cette observation. Pour ce faire, tracez le risque moyen pour différentes valeurs de D, e.g. $D \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 25, 30, 40, 50, 70, 90, 100\}$. Décrivez l'allure de la courbe essayez de faire un lien avec une courbe du papier de référence.

5 Gradient stochastique et surparamétrisation

Pour finir ce TP, on se propose d'analyser empiriquement le comportement du gradient stochastique dans le régime surparamétré. Da façon informelle, on peut établir le résultat suivant :

À retenir

Dans un régime surparamétré, le gradient stochastique à pas constant converge, tandis qu'il ne converge pas sinon.

La raison intuitive derrière cette propriété est que les gradients des fonctions individuelles f_i s'annulent de façon synchrone sur les minimiseurs globaux. C'est un peu comme s'il y avait une réduction de variance automatique.

1. Lancer une descente de gradient stochastique avec un mini-batch pour $D = 5$ et $D = 1000$. Enregister le risque empirique à chaque itération et l'afficher.
2. Est-ce que vous voyez une différence de comportement dans le régime sur-paramétré et dans le régime sous-paramétré ?

2. Pour cette partie, vous pouvez utiliser un gradient standard, i.e. calculer le gradient par rapport à l'intégralité des données.