TP2. Régression en PyTorch.

Frédéric Richard

Cours apprentissage statistique et réseaux de neurones
Master mathématiques appliquées, statistique,
Parcours Data Science.

2025, AMU

L'objectif de ce TP est d'apprendre à utiliser PyTorch pour construire et apprendre des réseaux de neurones. Dans une première partie, on s'intéresse à la régression linéaire que l'on traite avec un perceptron. Puis on étend le modèle de régression à un perceptron multicouche. Pour finir, on traite de la régression logistisque pour un problème de classification.

0.0.1 1. Régression linéaire.

1.1. Données sur le cancer de la prostate. On s'intéresse au jeu de données téléchargé sur le site kaggle prostate-cancer-forensic. Ces données sont contenues dans le fichier prostate.csv.

Les variables de ce jeu de données sont les suivantes:

- **lcavol** : Logarithme du volume cancéreux (log(volume cancéreux)). Il s'agit d'une mesure de la taille de la tumeur cancéreuse.
- **lweight** : Logarithme du poids de la prostate (log(poids de la prostate)). Cela indique le poids de la glande prostatique.
- **age** : Âge du patient. C'est l'âge de l'individu au moment du diagnostic.
- **lbph** : Logarithme de la quantité d'hyperplasie bénigne de la prostate (log(hyperplasie bénigne de la prostate)). Il s'agit d'une mesure de l'élargissement non cancéreux de la prostate.
- **svi** : Invasion des vésicules séminales (binaire : 0 ou 1). Cela indique si le cancer s'est propagé aux vésicules séminales.
- **lcp** : Logarithme de la pénétration capsulaire (log(pénétration capsulaire)). Cela mesure jusqu'où le cancer s'est propagé dans la capsule prostatique.
- **gleason** : Score de Gleason. C'est un système de classification utilisé pour évaluer l'agressivité du cancer de la prostate en fonction de l'apparence microscopique de la tumeur.
- **pgg45** : Pourcentage des scores de Gleason 4 ou 5. Cela indique la proportion de cellules cancéreuses ayant un score de Gleason de 4 ou 5, qui sont des formes plus agressives de cancer.
- **lpsa** : Logarithme de l'antigène prostatique spécifique (log(PSA)). Le PSA est une protéine produite par la prostate, et des niveaux élevés peuvent indiquer un cancer de la prostate.
- **train** : Indicateur d'entraînement (booléen : True ou False). Cela indique si le point de données fait partie de l'ensemble d'entraînement (True) ou non (False).

Il s'agit d'établir un lien entre la variable lpsa et les autres variables (exceptée train).

Le fichier peut se charger avec pandas de la manière suivante.

[3]:
```
import pandas as pd

# Chargement du jeux de données.
prostate = pd.read_csv('./prostate.csv')

# Informations sur les variables.
prostate.info()

# Visualisation partielle
prostate.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 97 entries, 0 to 96
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   lcavol   97 non-null     float64
 1   lweight  97 non-null     float64
 2   age      97 non-null     int64
 3   lbph     97 non-null     float64
 4   svi      97 non-null     int64
 5   lcp      97 non-null     float64
 6   gleason  97 non-null     int64
 7   pgg45    97 non-null     int64
 8   lpsa     97 non-null     float64
 9   train    97 non-null     bool
dtypes: bool(1), float64(5), int64(4)
memory usage: 7.0 KB
```

[3]:

	lcavol	lweight	age	lbph	svi	lcp	gleason	pgg45	lpsa	train
0	-0.579818	2.769459	50	-1.386294	0	-1.386294	6	0	-0.430783	True
1	-0.994252	3.319626	58	-1.386294	0	-1.386294	6	0	-0.162519	True
2	-0.510826	2.691243	74	-1.386294	0	-1.386294	7	20	-0.162519	True
3	-1.203973	3.282789	58	-1.386294	0	-1.386294	6	0	-0.162519	True
4	0.751416	3.432373	62	-1.386294	0	-1.386294	6	0	0.371564	True

On sépare le jeu de données en un ensemble d'apprentissage et un ensemble de test.

[ ]:
```
# Création des ensembles d'apprentissage et de test.
train_set = prostate[prostate["train"] == True]
test_set = prostate[prostate["train"] == False]
```

Par gérer ces données en Pytorch, on crée des Dataset de PyTorch; voir Datasets & DataLoaders.

[26]:
```
from torch.utils.data import TensorDataset
from torch import Tensor

# Definition des variables de régressions et cible.
x_var = ['lcavol', 'lweight', 'age', 'lbph', 'svi', 'lcp', 'gleason', 'pgg45']
y_var = ['lpsa']

# Creation d'un dataset Pytorch.
train_dataset = TensorDataset(Tensor(train_set[x_var].values),␣
↪Tensor(train_set[y_var].values))
test_dataset = TensorDataset(Tensor(test_set[x_var].values),␣
↪Tensor(test_set[y_var].values))

print("Taille de l'ensemble d'apprentissage: ", len(train_dataset))
print("Taille de l'ensemble de test: ", len(test_dataset))
```

```
Taille de l'ensemble d'apprentissage:  67
Taille de l'ensemble de test:  30
```

1.2. Création d'un réseau de neurones en PyTorch. La construction d'un réseau de neurones particulier en PyTorch se fait en héritant la classe Module du package torch.nn et en adaptant le constructeur et la méthode *forward* de cette classe. La définition du constructeur permet de définir les couches et les variables spécifiques du réseau. La méthode forward permet de spécifier les opérations qui conduisent de l'entrée du réseau à sa sortie. Une mise en oeuvre du modèle de régression linéaire est présentée ci-dessous.

[27]:
```
from torch.nn import Module, Linear

class RegressionLineaire(Module):
    def __init__(self, input_dim, output_dim):
        super(RegressionLineaire, self).__init__()
        # Définition des paramètres du modèle.
        self.p = input_dim
        self.k = output_dim
        # Definition de la couche du réseau.
        self.couche_dense = Linear(in_features=self.p, out_features=self.k,␣
↪bias=True)

    def forward(self, x):
        # Application de la couche dense à l'entrée.
        x = self.couche_dense(x)
```

```
        return x
```

On observe que la régression linéaire est mise en oeuvre au travers d'un perceptron à une seule couche et sans activation.

Après avoir défini le réseau, il faut en créer une instance.

Exercice

Pour créer cette instance, compléter le code ci-dessous en spécifiant les paramètres p et k.

[35]:
```
# p =
# k =

# Instanciation du modèle.
modele = RegressionLineaire(p, k)
```

On peut visualiser la structure du réseau avec la commande print:

[36]:
```
print(modele)
```

```
RegressionLineaire(
  (couche_dense): Linear(in_features=8, out_features=1, bias=True)
)
```

On peut également faire un bilan de ses paramètres.

[37]:
```
print("Bilan des paramètres du modèle:")
total_param = 0
for nom, param in modele.named_parameters():
    if param.requires_grad:
        print(f"{nom}: {param.numel()}")
        total_param += param.numel()
print(f"Total: {total_param}")
```

```
Bilan des paramètres du modèle:
couche_dense.weight: 8
couche_dense.bias: 1
Total: 9
```

1.2. Apprentissage du modèle. Après la définition du modèle, nous passons à l'étape d'apprentissage qui va permettre d'estimer les paramètres du modèle.

On commence par choisir la fonction de perte (loss function) et l'algorithme d'optimisation ou optimiseur.

[38]:
```
from torch.nn import MSELoss as MSE
from torch.optim import Adam

critere = MSE()
```

```
optimiseur = Adam(modele.parameters(), lr=0.001)
```

Des algorithmes d'optimisation comme Adam fonctionnent à partir de lots (batch). On va définir comment sont chargés ces lots sur les données de la manière suivante.

[50]:
```
from torch.utils.data import DataLoader

trainbatch = DataLoader(train_dataset, batch_size=6, shuffle=True)
testbatch = DataLoader(test_dataset, batch_size=1, shuffle=True)
```

On passe ensuite à l'apprentissage en tant que tel en définissant les époques (epoch).

[71]:
```
num_epochs = 100  # Nombre d'époques pour l'entraînement.

for epoch in range(num_epochs):
    running_loss = 0.0
    for i, (X, Y) in enumerate(trainbatch):
        # Réinitialisation des gradients
        optimiseur.zero_grad()

        # Passage avant
        outputs = modele(X)
        loss = critere(outputs, Y)

        # Rétropropagation et optimisation
        loss.backward()
        optimiseur.step()

        # Affichage des statistiques de perte
        running_loss += loss.item()

    print(f"[{epoch + 1}] loss = {running_loss / len(trainbatch):.3f}")
```

```
[1] loss = 0.481
[2] loss = 0.462
[3] loss = 0.477
[4] loss = 0.463
[5] loss = 0.450
[6] loss = 0.535
[7] loss = 0.516
[8] loss = 0.457
[9] loss = 0.453
[10] loss = 0.438
[11] loss = 0.511
[12] loss = 0.531
[13] loss = 0.452
[14] loss = 0.441
[15] loss = 0.625
```

```
[16] loss = 0.604
[17] loss = 0.509
[18] loss = 0.634
[19] loss = 0.467
[20] loss = 0.447
[21] loss = 0.451
[22] loss = 0.455
[23] loss = 0.438
[24] loss = 0.450
[25] loss = 0.450
[26] loss = 0.432
[27] loss = 0.484
[28] loss = 0.459
[29] loss = 0.450
[30] loss = 0.438
[31] loss = 0.466
[32] loss = 0.761
[33] loss = 0.567
[34] loss = 0.538
[35] loss = 0.452
[36] loss = 0.655
[37] loss = 0.459
[38] loss = 0.614
[39] loss = 0.464
[40] loss = 0.436
[41] loss = 0.458
[42] loss = 0.466
[43] loss = 0.435
[44] loss = 0.494
[45] loss = 0.527
[46] loss = 0.439
[47] loss = 0.447
[48] loss = 0.441
[49] loss = 0.451
[50] loss = 0.465
[51] loss = 0.491
[52] loss = 0.452
[53] loss = 0.446
[54] loss = 0.481
[55] loss = 0.464
[56] loss = 0.434
[57] loss = 0.436
[58] loss = 0.433
[59] loss = 0.555
[60] loss = 0.491
[61] loss = 0.469
[62] loss = 0.480
[63] loss = 0.442
```

```
[64] loss = 0.438
[65] loss = 0.449
[66] loss = 0.472
[67] loss = 0.461
[68] loss = 0.657
[69] loss = 0.662
[70] loss = 0.440
[71] loss = 0.462
[72] loss = 0.467
[73] loss = 0.432
[74] loss = 0.474
[75] loss = 0.494
[76] loss = 0.433
[77] loss = 0.463
[78] loss = 0.433
[79] loss = 0.433
[80] loss = 0.430
[81] loss = 0.440
[82] loss = 0.478
[83] loss = 0.431
[84] loss = 0.439
[85] loss = 0.431
[86] loss = 0.473
[87] loss = 0.452
[88] loss = 0.433
[89] loss = 0.431
[90] loss = 0.437
[91] loss = 0.439
[92] loss = 0.460
[93] loss = 0.453
[94] loss = 0.455
[95] loss = 0.539
[96] loss = 0.496
[97] loss = 0.463
[98] loss = 0.439
[99] loss = 0.711
[100] loss = 0.515
```

Exercice

1. Tracer le graphe du coût en fonction des itérations.
2. Fixer une itération à partir de laquelle on peut considérer qu'il y a eu convergence de l'algorithme de minimisation.

1.3. Test du modèle. On peut appliquer le réseau de neurones sur les données de test:

[75]:
```
from torch import no_grad
```

```
with no_grad():
    modele.eval()

    # Erreur d'apprentissage.
    loss_train = 0
    for i, (X, Y) in enumerate(trainbatch):
        outputs = modele(X)
        loss = critere(outputs, Y)
        loss_train += loss.item()
    loss_train = loss_train / len(trainbatch)

    # Erreur de généralisation.
    loss_test = 0
    for i, (X, Y) in enumerate(testbatch):
        outputs = modele(X)
        loss = critere(outputs, Y)
        loss_test += loss.item()
    loss_test = loss_test / len(testbatch)

print('Train loss :', loss_train)
print('Test loss :', loss_test)
```

```
Train loss : 0.5424202630917231
Test loss : 0.5950335302448366
```

Exercice

1. Modifier le modèle de départ
   - en changeant les activations (prendre par exemple une activation ReLU),
   - en ajoutant une couche dense.
2. Comparer le premier modèle à ces derniers modèles.

0.0.2 2. Régression logistique.

2.1. Le jeu de données leaf. Ce jeu de données a été recueilli en vue d'établir un lien entre un type d'architecture de feuilles d'arbre (orthotropic ou plagiotropic) à partir de caractéristiques mesurables de la feuille.

Ces données sont contenues dans le fichier leafshape17.csv. En voici la description complète:

Le jeu de données leafshape17 data a 61 lignes and 8 colonnes dont

- bladelen: leaf length (in mm)
- petiole: a numeric vector
- bladewid: leaf width (in mm)
- latitude: latitude
- logwid: natural logarithm of width
- logpet: logarithm of petiole measurement

- loglen: logarithm of length
- arch: leaf architecture (0 = orthotropic, 1 = plagiotropic)

Référence:

King, D.A. and Maindonald, J.H. 1999. Tree architecture in relation to leaf dimensions and tree stature in temperate and tropical rain forests. Journal of Ecology 87: 1012-1024.

Exercice

Importer ce jeu de données et en faire un dataset en Pytorch.

2.2. Classification. On fait la classification de ces données au moyen d'une régression logistique.

Exercice

1. Définir un réseau de neurones qui fait la classification de ces données
2. En faire l'apprentissage.
3. L'évaluer en calculant le nombre d'erreur de classification sur des données de test.
4. Etendre cette approche à un perceptron à deux couches.
5. Comparer les performances des deux modèles.