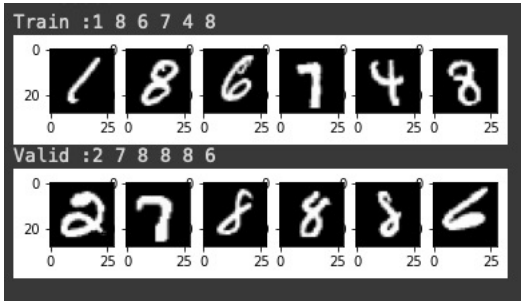
Séance de TP 2 –
Perceptron Multicouches et descente de gradient

Sujet

Au cours de cette séance nous allons nous intéresser à l'optimisation d'un modèle de classification de type perceptron multi couches pour la classification des chiffres manuscrits de la base MNIST



- Nous mettrons en évidence l'influence des paramètres de l'algorithme SGD
- Nous comparerons SGD et ADAM en terme de convergence (temps et Loss)
- Nous comparerons les fonctions d'activation sigmoïde et Relu
- Nous mettrons en évidence l'influence de l'architecture choisie pour un réseau prenant en entrée les pixels des images uniquement, et le nombre de paramètres

Les principales méthodes sont expliquées en annexe

Séance de TP 2 –
Perceptron Multicouches et descente de gradient

1- Mettre en évidence l'influence des paramètres de l'algorithme SGD

Utiliser le programme **MLP_pixels.ipynb** et réaliser les expériences suivantes

Architecture : MLP comportant 3 couches 512 – 128 – 10

Pour toutes les expériences

- relever l'évolution de la loss en train et valid
- relever la performance sur le jeu de test

Paramètres SGD: batch=64, lr=1e-2, epoch=20

Paramètres SGD: batch=64, lr=1e-2, epoch=60

Paramètres SGD: batch=64, lr=1e-3, epoch=60

Paramètres SGD: batch=16, lr=1e-2, epoch=20

Conclure sur le rôle et l'influence des trois paramètres du SGD

Séance de TP 2 –
Perceptron Multicouches et descente de gradient

2- Mettre en évidence l'apport de la fonction d'activation RELU() par rapport à la fonction Sigmoïde()

Architecture : MLP comportant 3 couches 512 – 128 – 10 activation nn.Sigmoid()
Paramètres SGD: batch=64, lr=1e-2, epoch=20

Architecture : MLP comportant 3 couches 512 – 128 – 10 activation nn.ReLU()
Paramètres SGD: batch=64, lr=1e-2, epoch=20

Conclusion ?

Séance de TP 2 –
Perceptron Multicouches et descente de gradient

3- Comparer SGD et ADAM en terme de convergence (temps et Loss)

Utiliser le programme **MLP_pixels.ipynb** et réaliser les expériences suivantes

Architecture : MLP comportant 3 couches 512 – 128 – 10 Sigmoid()

Paramètres ADAM: batch=64, lr=1e-2, epoch=20 puis 60

Paramètres ADAM: batch=64, lr=1e-3, epoch= 20 puis 60

Conclusion ?

Séance de TP 2 –
Perceptron Multicouches et descente de gradient

4- Mettre en évidence l'influence de l'architecture et du nombre de paramètres

Architecture : MLP comportant 3 couches 512 – 128 – 10 activation nn.ReLU()
Paramètres ADAM: batch=64, lr=1e-3, epoch=20

Architecture : MLP comportant 4 couches 512 – 256 – 128 – 10 activation nn.ReLU()
Paramètres ADAM: batch=64, lr=1e-3, epoch=20

Architecture : MLP comportant 2 couches 256 – 10 activation nn.ReLU()
Paramètres ADAM: batch=64, lr=1e-3, epoch=20

Vérifier le nombre de paramètres retourné par la méthode
count_parameters(my_best_MLP)

Conclusion ?

Séance de TP 2 –
Annexe : MLP_pixels.ipynb

Architecture du perceptron

Dans un premier temps nous choisissons un Perceptron à 3 couches prenant en entrée les 28 x 28 = pixels des images d'entrée et fournissant en sortie les valeurs des probabilités des 10 classes de chiffres. L'architecture choisie, permet de structurer progressivement l'information dans le réseau. La première couche possède 512 neurones, la seconde 128, et la troisième 10.

Remarque : l'activation softmax est mise automatiquement dans la loss CrossEntropy il faut l'ajouter au moment du test si on le souhaite, voir la méthode perf_loop(dataloader, model)

```
Class MLP(nn.Module):
''' Multilayer Perceptron. '''
def __init__(self,input_features=128):
super().__init__()
self.layers = nn.Sequential(
nn.Flatten(), # on écrit l'image en 1D
nn.Linear(input_features, 512),
nn.Sigmoid(),
nn.Linear(512, 128),
nn.Sigmoid(),
nn.Linear(128, 10)
)
def forward(self, x):
'''Forward pass'''
return self.layers(x)
```

Séance de TP 2 –
Annexe : MLP_pixels.ipynb

Chargement des données

Nous chargeons le dataset MNIST d'apprentissage et le découpons en Train et Valid

L'évaluation de performance est réalisée sur le dataset de test MNIST

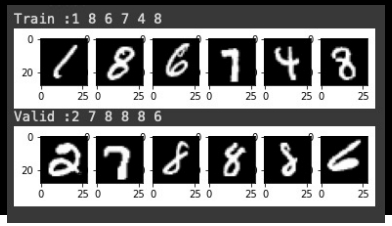
```
# Prepare MNIST dataset
dataset = MNIST(os.getcwd(), train = True, download=True, transform=transforms.ToTensor())
N = len(dataset)

N_train = int(N * 0.9)
N_valid = N - N_train

train_dataset, valid_dataset = random_split(dataset, [N_train, N_valid])

trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True,
num_workers=1)
validloader = torch.utils.data.DataLoader(valid_dataset, batch_size=batch_size, shuffle=True,
num_workers=1)

affichage(trainloader,"Train :")
affichage(validloader,"Valid :")
```



Séance de TP 2 – Annexe : MLP_pixels.ipynb

Instanciation du MLP et définition de la loss

```
my_MLP = MLP(input_features = dim)
my_loss = nn.CrossEntropyLoss()

if OPTIM == "SGD":
  my_optimizer = torch.optim.SGD(my_MLP.parameters(), lr=learning_rate)
elif OPTIM == "ADAM":
  my_optimizer = torch.optim.Adam(my_MLP.parameters(), lr=learning_rate)
```

Boucle d'apprentissage

```
for t in range(epochs):
  print("\nEpoque :",t+1)
  train_loss.append(train_loop(trainloader, my_MLP, my_loss, my_optimizer))
  # on test à chaque itération
  valid_loss.append(valid_loop(validloader, my_MLP, my_loss))
  # Early Stopping sans patience
  if t == 0:
    best_valid_loss = valid_loss[-1]
  else:
    if valid_loss[-1] < best_valid_loss:
      # on mémorise ce modèle
      best_iter = t
      torch.save(my_MLP.state_dict(), Model_name)
```

Séance de TP 2 – Annexe : MLP_pixels.ipynb

Test de performance

```
def perf_loop(dataloader, model):
  Total = len(list(dataloader.dataset))
  nb_batches = len(dataloader)
  Positifs = 0
  with torch.no_grad():
    for X, y in dataloader:
      pred = model(X.float())
      prob_pred = nn.Softmax(dim=1)(pred) # pas nécessaire pour prédire le meilleur

  y_pred = torch.argmax(prob_pred,dim=1)
  similaires = np.array(y_pred==y)
  Positifs += np.sum(similaires)
  return Positifs, Total, y_pred
```

Séance de TP 2 –
Annexe : MLP_pixels.ipynb

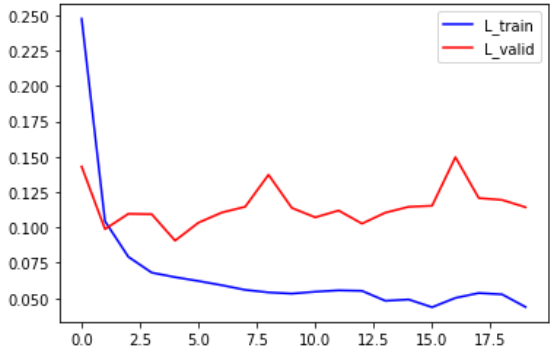
Appel le test de performance

```
############## on teste maintenaant les performances ################
test_dataset = MNIST(os.getcwd(), train = False, download=True, transform=transforms.ToTensor())
N_test = len(test_dataset)

##############
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=N_test, shuffle=False,
num_workers=1)
affichage(testloader,"Test :")

my_best_MLP = MLP(input_features = dim)
my_best_MLP.load_state_dict(torch.load(Model_name))
Positifs, Total, y_pred = perf_loop(testloader, my_best_MLP)
print("Taux de reco en Test:",Positifs,"/", Total)
print("Meilleure époque :",best_iter+1)

nb_train_param = count_parameters(my_best_MLP)
print("Nombre de paramètres libres:",nb_train_param)
```



Taux de reco en Test: 9755 / 10000 Meilleure époque : 5