TP Siamese Neural Networks for beginners

Gianni Franchi

ENSTA Paris

At the end of the TP, send your work (sources + concise report compress in a zip file) by email " name " - " first name " - work-TP1 (for example Dupont-Pierre-travail-TP1.zip). Subsequent changes to the code at the end of the TP will not be taken into account. Focus on the writing of the report.
Please write one scrip.py by exercise.
Basic tips:

- TP is individual; any fraud will be sanctioned drastically.
- as soon as possible (every few lines ideally), **test** your code.
- even if it does not earn you a better grade right away, try going after
- write comments on your code (IMPORTANT!)

1 Introduction

Humans have a strong ability to acquire and recognize new patterns. In particular, humans can easily learn the similarity between two images. Deep learning has been successfully used to achieve state-of-the-art performance in a variety of applications. The goal of this work will be to code a Siamese network able to learn the similarity between two images.

The goal of this TP is to make deep learning codes with Pytorch. So you will first learn to use Pytorch on MNIST, then you work on the Market-1501 database [1]. You should learn to:

- build a CNN
- train a CNN
- perform a grid search
- build a dataset

2 Exercise 1 : Mnist classification with Pytorch

2.1 Simple Deep neural network

Q1/ Please understand the NumPy script and explain the script.
Q2/ Please write a script using the Module class
help :
The neural network architectures in Pytorch can be defined in a class which inherits the properties from the base class from nn package called Module. This inheritance from the nn.Module class allows us to implement, access, and call a number of methods easily. We can define all the layers inside the constructor of the class, and the forward propagation steps inside the forward function. The big advantage of the nn.Module class is that you do not need to write the backward loop. Let us have look at this small network called Net.

[1] Zheng et al. Scalable Person Re-identification: A Benchmark. ICCV 2015.

```
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.FC1 = nn.Linear(784, 128)
        self.FC2 = nn.Linear(128, 10)
   def forward(self, x):
        x = F.sigmoid(self.FC1(x))
        x = self.FC2(x)
        return x
```

Please start a class with a capital letter. (That is a convention)
Then you write the init function that is the constructor of your class. It defines the different trainable layers and initialised the weights. Don't forget to call the super() to inherit the constructor of Module.

Now you build an object and call it model (for example) and bring it to the gpu.

```
model = Net()
model= model.to('cuda')
```

Now you have to write the optimizer and define the parameters to optimize (here I choose the stochastic gradient descent SGD):

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(),
lr=0.01, weight_decay= 1e-3, momentum = 0.9)
```

After having bring the input to the GPU apply the code of forward/backward loop for Pytorch on the training function. The code is on the box just bellow:

```
# zeroes the gradient buffers of all parameters
optimizer.zero_grad()
outputs = model(input)
loss = criterion(outputs, target)
loss.backward()
# Perform the training parameters update
 optimizer.step()
```

As you can see to compute the backward pass for gradient, we first zero the gradient stored in the network. In Pytorch, every time we backpropagate the gradient from a variable, the gradient is accumulate instead of being reset and replaced. Write the code of the training function and change the testing function as you should.

(I do not give you the code but it is easy)

Now that you have everything, **your task is to write a network called Net that has the same architecture as the one in question 1.**

2.2 Add summaries

Q3/ Please add summaries to understand the training. So you should find a way to keep the training loss and the training accuracy? What two other parameters should we keep?
help :
Here is a code where we save the training loss and the training accuracy. (note that we put random value but you can put the values of interest).

```
from torch.utils.tensorboard import SummaryWriter
import numpy as np

# build a folder to save the summaries
writer = SummaryWriter('runs/MNIST_TEST')
# n_iter = iterations step
for n_iter in range(100):
    writer.add_scalar('Loss/train', np.random.random(), n_iter)
    writer.add_scalar('Accuracy/train', np.random.random(), n_iter)
writer.close()
```

2.3 Convolutional network

Q4/ Please now transform the network to a convolutional neural network (CNN).
More particularly I want you to code Resnet18[2]
help :
For the rest I want you to work with Resnet18.

2.4 Hyper-parameter optimization

Hyper-parameter optimization is a big part of deep learning. The reason is that neural networks might have totally new results with new parameters. So it is important to find good parameters. There are two main techniques: Grid Search , or Bayesian Search. Here we will perform grid search. I want you to fill the accuracy on the following Table 1 and 2

λ	1.0	0.1	0.01	0.001
Weight decay				
0.1				
0.01				
0.001				

Table 1: Weight decay\learning rate (λ) table. For this experiment choose batch size = 64.

λ	1.0	0.1	0.01	0.001
batchsize	0.11		0.9910	0.9905
5				
64				
200				

Table 2: Batchsize\learning rate table. For this experiment choose Weight decay = 0.001.

[2]He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of CVPR 2016.

Q5/ Comment the tables after you have filled it.

2.5 Regularisation

Q6/ What are the regularization present on your network. Explain why?

3 Exercise 2 : Re identification

Now we are going to write a Dataloader on a fully connected network. We focus on the Market-1501 database [3] which is one ReID datasets currently available. As you can see, the dataset is composed of crop arround persons. It contains 32668 images of 1501 persons, split into train/test sets of 12936/19732 images.

Q1/ Please have a look at the dataset and plot the images.

Siamese networks are a special type of neural network architecture. Instead of learning to classify its inputs, the neural networks learns the similarity between 2 inputs.

A Siamese networks consists of two identical neural networks, each taking one of the two input images. The last layers of the two networks are then fed to a contrastive loss function , which calculates the similarity between the two images.

3.1 Dataset

Now, let us go through the details of how to set the Python class Dataset, which will characterize the key features of the dataset you want to generate.

```
import torch
from torch.utils import data
import random
class SimpleDataset(Dataset):

    def __init__(self,train_data,train_labels,transform=None,label_transform=None):
        self.train_data = train_data
        self.train_labels = train_labels
        self.transform = transform
        self.label_transform = label_transform


    def __getitem__(self,index):
        data_path, label_path = self.train_data[index], self.train_labels[index]

        img0 = Image.open(data_path)
        label = Image.open(label_path)

        if self.transform is not None:
            img = self.transform(img)

        if self.label_transform is not None:
            label = self.label_transform(label)

        return img0, label

    def __len__(self):
        return len(self.data_path)
```

[3] Zheng et al. Scalable Person Re-identification: A Benchmark. ICCV 2015.

First, there is the initialization function of the class. We make the latter inherit the properties of torch.utils.data.Dataset so that we can later leverage nice functionalities such as multiprocessing. Then, each call requests a sample index for which the upper bound is specified in the __len__ method. Finally, when you build a batch you call __getitem__ method to generate it. So these 3 functions are mandatory.

On addition you can apply a transformation.

The goal of Reid is to learn a function $f_\theta(x)$ that takes as input an image x and output feature space $f_\theta(x)$ such that $D_{ij} = D(f_\theta(x_i), f_\theta(x_j))$ is small for two similar images x_i and x_j with D a distance for example the Norm 2, and D_{ij} is huge for different images. Hence the goal is to optimize the parameter θ of the triplet loss:

$$\mathcal{L}(\theta) = \sum_{a,p,n} \max\left(0, m + D_{a,p} - D_{a,n}\right)$$

where a is the index of an anchor x_a, p is the index such as x_a and x_p corresponds to the same person. We denote x_p as a positive example. n is the index such as x_a and x_n corresponds to the different person. We denote x_n as a negative example. For more information on ReId please read[4]

Q2/ Please implement the triplet loss.

To implement the triplet loss, you must remember that triplet loss is a differentiable operator, so you must code a nn.Module class. Please complete the triplet loss on ./utils/TripletLoss.py

A major issue of the triplet loss, though, is that the larger the dataset gets the less the triplet get efficient. The reason comes from the fact that larger ReId datasets have more and more ID. Hence the batch have a large fraction of uninformative triplets .

So the DNN will learn that people with differently colored clothes are different persons, that does not teach well the DNN. So researcher worked on sampler that make the training goes faster.

Q3/We have implemented in ./utils/RandomSampler.py a sampler. Please describe it.

Q4/ You have everything you need to write the dataset class. Please complete the class Market1501 on ./data.py. On this project we will need one special method on this class and 2 special attribute.

We need a method called get_id(self,file_path) that returns the person id of the path.

We need self.unique_ids that is a sorted list of the id on the dataset.

We need self.imgs that is a sorted list of the path of all the images on the dataset.

Q5/ Write your DNN.

This question is hard we suggest that you use a Resnet 50 or 101[5] and you delete the last layer and replace by 2 fully connected layers. Careful, you need to have 3 outputs. It is classical on DNN to add auxialiary loss to improve the training. Here we add as auxiliary loss a classification loss. Please check ./loss.py. **Please explain the loss?** Hence, we want that the DNN outputs 3 tensors. The third one is the feature space for the metric learning, the second one is the logit tensor for the classification of the id, and the first one is the concatenation of these tensors. Please complete the DNN on ./network.py

Q6/ Write a training code and train the DNN.

Please complete ./main.py the function train() should train the DNN. Please explain the evaluate function.

Q7/ Save your model, and write a testing script that load the save model (checkpoint) and evaluate the results.

Please called this script ./evaluation_tp.py

[4]Hermans, Alexander, et al. "In defense of the triplet loss for person re-identification." arXiv preprint arXiv:1703.07737 (2017).

[5]He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of CVPR 2016.