4. vježba: Metričko ugrađivanje

Predavanje o metričkom ugrađivanju moguće je naći na neslužbenim stranicama predmeta.

Vježba

1. zadatak: Učitavanje podataka (10%)

Izvedite učitavanje podataka tako da se omogući učenje modela za metričko ugrađivanje trojnim gubitkom. Da bismo to napravili, potrebno je prilagoditi MNIST dataset tako da se prilikom dohvata primjera za treniranje (sidra), dohvaćaju i odgovarajući pozitivan i negativan primjer.

```
from torch.utils.data import Dataset
from collections import defaultdict
from random import choice
import torchvision


class MNISTMetricDataset(Dataset):
    def __init__(self, root="/tmp/mnist/", split='train'):
        super().__init__()
        assert split in ['train', 'test', 'traineval']
        self.root = root
        self.split = split
        mnist_ds = torchvision.datasets.MNIST(self.root, train='train' i
        self.images, self.targets = mnist_ds.data.float() / 255., mnist_
        self.classes = list(range(10))
```

```
        self.target2indices = defaultdict(list)
        for i in range(len(self.images)):
            self.target2indices[self.targets[i].item()] += [i]

    def _sample_negative(self, index):
        # YOUR CODE HERE


    def _sample_positive(self, index):
        # YOUR CODE HERE


    def __getitem__(self, index):
        anchor = self.images[index].unsqueeze(0)
        target_id = self.targets[index].item()
        if self.split in ['traineval', 'val', 'test']:
            return anchor, target_id
        else:
            positive = self._sample_positive(index)
            negative = self._sample_negative(index)
            positive = self.images[positive]
            negative = self.images[negative]
            return anchor, positive.unsqueeze(0), negative.unsqueeze(0),

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

Implementirajte metode `_sample_positive` i `_sample_negative` tako da njihove povratne vrijednosti odgovaraju indeksima uzorkovanih slika u listi `self.images`. Za potrebe ove vježbe dovoljno je implementirati jednostavno uzorkovanje koje će za pozitivni primjer uzorkovati slučajnu sliku koja pripada istom razredu kao sidro, a za negativni primjer slučajnu sliku koja pripada bilo kojem razredu različitom od razreda sidra.

2. zadatak: Definicija modela za metričko ugrađivanje (40%)

Zadan je kod koji definira grubu strukturu modela za metričko ugrađivanje.

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

```
class _BNReluConv(nn.Sequential):
    def __init__(self, num_maps_in, num_maps_out, k=3, bias=True):
        super(_BNReluConv, self).__init__()
        # YOUR CODE HERE

class SimpleMetricEmbedding(nn.Module):
    def __init__(self, input_channels, emb_size=32):
        super().__init__()
        self.emb_size = emb_size
        # YOUR CODE HERE

    def get_features(self, img):
        # Returns tensor with dimensions BATCH_SIZE, EMB_SIZE
        # YOUR CODE HERE
        x = ...
        return x

    def loss(self, anchor, positive, negative):
        a_x = self.get_features(anchor)
        p_x = self.get_features(positive)
        n_x = self.get_features(negative)
        # YOUR CODE HERE
        loss = ...
        return loss
```

Nadopunite naš predložak prema sljedećim uputama.

a) gubitak

Implementirajte trojni gubitak po uzoru na pytorchev `TripletMarginLoss`.

b) konvolucijska jedinica BNReLUConv

U praksi je praktično izdvojiti dio modela koji se često ponavlja u dijeljeni diferencijabilni modul. Oblikujte konvolucijsku jedinicu `BNReLUConv` koji se sastoji od normalizacije po grupi, aktivacije ReLU i konvolucije. Primijetite da naš predložak nasljeđuje razred Sequential. To znači da za dodavanje slojeva u konstruktoru možete koristiti metodu `append`.

c) metričko ugrađivanje

Dovršite izvedbu modela za metričko ugrađivanje. Neka se vaš model sastoji od 3 uzastopne konvoluijske jedinice `BNReLUConv` (neka veličina jezgre bude 3, a broj mapa značajki - `emb_size`) razdvojena sažimanjem maksimumom s veličinom jezgre 3 i korakom 2. Konačno ugrađivanje slike izlučite globalnim sažimanjem prosjekom. Pripazite da izlazni tenzor u metodi `get_features` zadrži prvu dimenziju koja označava veličinu minigrupe, čak i kada je ona jednaka 1.

3. zadatak: Učenje i vrednovanje (40%)

Zadan je kod za učenje modela za metričko ugrađivanje na MNIST podatacima. Ovaj kod koristi pomoćnu skriptu `utils.py` dostupnu ovdje.

```
import time
import torch.optim
from dataset import MNISTMetricDataset
from torch.utils.data import DataLoader
from model import SimpleMetricEmbedding
from utils import train, evaluate, compute_representations

EVAL_ON_TEST = True
EVAL_ON_TRAIN = False


if __name__ == '__main__':
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    print(f"= Using device {device}")

    # CHANGE ACCORDING TO YOUR PREFERENCE
    mnist_download_root = "./mnist/"
    ds_train = MNISTMetricDataset(mnist_download_root, split='train')
    ds_test = MNISTMetricDataset(mnist_download_root, split='test')
    ds_traineval = MNISTMetricDataset(mnist_download_root, split='traine

    num_classes = 10

    print(f"> Loaded {len(ds_train)} training images!")
    print(f"> Loaded {len(ds_test)} validation images!")

    train_loader = DataLoader(
```

```
        ds_train,
        batch_size=64,
        shuffle=True,
        pin_memory=True,
        num_workers=4,
        drop_last=True
    )

    test_loader = DataLoader(
        ds_test,
        batch_size=1,
        shuffle=False,
        pin_memory=True,
        num_workers=1
    )

    traineval_loader = DataLoader(
        ds_traineval,
        batch_size=1,
        shuffle=False,
        pin_memory=True,
        num_workers=1
    )

    emb_size = 32
    model = SimpleMetricEmbedding(1, emb_size).to(device)
    optimizer = torch.optim.Adam(
        model.parameters(),
        lr=1e-3
    )

    epochs = 3
    for epoch in range(epochs):
        print(f"Epoch: {epoch}")
        t0 = time.time_ns()
        train_loss = train(model, optimizer, train_loader, device)
        print(f"Mean Loss in Epoch {epoch}: {train_loss:.3f}")
        if EVAL_ON_TEST or EVAL_ON_TRAIN:
            print("Computing mean representations for evaluation...")
            representations = compute_representations(model, train_loade
        if EVAL_ON_TRAIN:
            print("Evaluating on training set...")
            acc1 = evaluate(model, representations, traineval_loader, de
```

```
            print(f"Epoch {epoch}: Train Top1 Acc: {round(acc1 * 100, 2)}")
        if EVAL_ON_TEST:
            print("Evaluating on test set...")
            acc1 = evaluate(model, representations, test_loader, device)
            print(f"Epoch {epoch}: Test Accuracy: {acc1 * 100:.2f}%")
        t1 = time.time_ns()
        print(f"Epoch time (sec): {(t1-t0)/10**9:.1f}")
```

a) Analiza modula `utils.py`

Proučite funkcije za učenje i vrednovanje u modelu `utils.py`. Kako se računaju reprezentacije razreda? Kako se provodi klasifikacija primjera? Probajte smisliti alternativne pristupe za klasifikaciju primjera.

b) Klasifikacija na temelju metričkog ugrađivanja

Naučite model za metričko ugrađivanje iz zadatka 2.c na podskupu za treniranje skupa MNIST. Provedite klasifikaciju slika iz podskupa za validaciju i izmjerite točnost.

c) Klasifikacija na temelju udaljenosti u prostoru slike

Ponovo provedite klasifikaciju na podskupu za validaciju, ali ovaj put u prostoru slike. Ostvarite taj zadatak oblikovanjem razreda koji u metodi `get_features` provodi jednostavnu vektorizaciju slike.

```
class IdentityModel(nn.Module):
    def __init__(self):
        super(IdentityModel, self).__init__()

    def get_features(self, img):
        # YOUR CODE HERE
        feats = ...
        return feats
```

Implementirajte razred `IdentityModel` prema prikazanom predlošku. Modificirajte funkciju za učenje tako da se klasifikacija provodi u prostoru slike. Primijetite da se `IdentityModel` ne

može trenirati. Izmjerite točnost klasifikacije na podskupu za validaciju.

d) Pohranjivanje parametara modela

U praksi je praktično pohraniti parametre naučenog modela za kasnije korištenje u fazi zaključivanja. Modificirajte funkciju za učenje tako da pohranite naučene parametre korištenjem funkcije 'torch.save'. Iznova naučite model za metričko ugrađivanje i pohranite dobivene parametre.

e) Klasifikacija neviđenih razreda

Jedna od prednosti učenja metričkog ugrađivanja nad standardnim klasifikacijskim modelima jest mogućnost dodavanja novih klasa u fazi zaključivanja. Modificirajte konstruktor `MNISTMetricDataset` tako da se omogući uklanjanje primjera odabrane klase iz skupa za učenje:

```
    def __init__(self, root="/tmp/mnist/", split='train', remove_class=N
        super().__init__()
        assert split in ['train', 'test', 'traineval']
        self.root = root
        self.split = split
        mnist_ds = torchvision.datasets.MNIST(self.root, train='train' i
        self.images, self.targets = mnist_ds.data.float() / 255., mnist_
        self.classes = list(range(10))

        if remove_class is not None:
            # Filter out images with target class equal to remove_class
            # YOUR CODE HERE

        self.target2indices = defaultdict(list)
        for i in range(len(self.images)):
            self.target2indices[self.targets[i].item()] += [i]
```

Iz podskupa za treniranje uklonite razred 0 te istrenirajte novi model za metričko ugrađivanje iz zadatka 2. Klasificirajte sve slike (uključujući i razred 0) iz podskupa za validaciju na temelju sličnosti u prostoru značajki. Primijetite da ćete trebati imati dva loadera podskupa za učenje. Prvi razred će ignorirati slike znamenke 0, a koristit ćete ga za učenje modela. Drugi razred će čitati slike sa svim znamenkama, a iskoristit ćete ga za dobivanje prosječne reprezentacije znamenki svih razreda. Pohranite parametre naučenog modela i prikažite postignutu klasifikacijsku točnost.

4. zadatak: Vizualizacija podataka (10%)

Kvalitetu metričkog ugrađivanja možemo i kvalitativno procijeniti uspordbom razmještaja podataka u prostoru značajki i prostoru slike. S obzirom na to da je visokodimenzionalne podatke nemoguće vizualizirati u originalnom prostoru, primjere je potrebno prebaciti u 2D prostor. Ovo možemo provesti analizom svojstvenih komponenti. Obratite pažnju na to da torchu nudi gotovu implementaciju kroz funkciju pca_lowrank.

```
import numpy as np
import torch

from dataset import MNISTMetricDataset
from model import SimpleMetricEmbedding
from matplotlib import pyplot as plt


def get_colormap():
    # Cityscapes colormap for first 10 classes
    colormap = np.zeros((10, 3), dtype=np.uint8)
    colormap[0] = [128, 64, 128]
    colormap[1] = [244, 35, 232]
    colormap[2] = [70, 70, 70]
    colormap[3] = [102, 102, 156]
    colormap[4] = [190, 153, 153]
    colormap[5] = [153, 153, 153]
    colormap[6] = [250, 170, 30]
    colormap[7] = [220, 220, 0]
    colormap[8] = [107, 142, 35]
    colormap[9] = [152, 251, 152]
    return colormap


if __name__ == '__main__':
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    print(f"= Using device {device}")
    emb_size = 32
    model = SimpleMetricEmbedding(1, emb_size).to(device)
    # YOUR CODE HERE
    # LOAD TRAINED PARAMS

    colormap = get_colormap()
    mnist_download_root = "./mnist/"
```

```
    ds_test = MNISTMetricDataset(mnist_download_root, split='test')
    X = ds_test.images
    Y = ds_test.targets
    print("Fitting PCA directly from images...")
    test_img_rep2d = torch.pca_lowrank(ds_test.images.view(-1, 28 * 28),
    plt.scatter(test_img_rep2d[:, 0], test_img_rep2d[:, 1], color=colorma
    plt.show()
    plt.figure()

    print("Fitting PCA from feature representation")
    with torch.no_grad():
        model.eval()
        test_rep = model.get_features(X.unsqueeze(1))
        test_rep2d = torch.pca_lowrank(test_rep, 2)[0]
        plt.scatter(test_rep2d[:, 0], test_rep2d[:, 1], color=colormap[Y
        plt.show()
```

Modificirajte kod tako da učitava parametre naučene u prethodnom zadatku. Više o pohrani i učitavanju parametara možete naći u dokumentaciji pytorcha. Vizualizirajte primjere u prostoru slike te u prostoru značajki za model koji je učen sa svim znamenkama i za model koji prilikom učenja nije vidio slike sa znamenkom 0.

---

dlunizg
ivan.kreso@fer.hr