Exercises for the course

Machine Learning 2

Summer semester 2021

Abteilung Maschinelles Lernen
Institut für Softwaretechnik und theoretische Informatik
Fakultät IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Exercise Sheet 9

Exercise 1: Convolutional Neural Networks (10 P)

Let $x = (x^{(i)})_i$ be a multivariate time series represented as a collection of one-dimensional signals. For simplicity of the following derivations, we consider these signals to be of infinite length. We feed x as input to a convolutional layer. The forward computation in this layer is given by:

$$\begin{aligned} z_t^{(j)} &= \sum_i \left[w^{(ij)} \star x^{(i)}\right]_t \\ &= \sum_i \sum_{s=-\infty}^{\infty} w_s^{(ij)} \cdot x_{t+s}^{(i)} \qquad \text{for all } \ l \ \text{ and } \ t \in \mathbb{Z} \end{aligned}$$

It results in $z = (z^{(j)})_j$ another multivariate time series again composed of a collection of output signals also assumed to be of infinite length. Convolution filters $w^{(ij)}$ have to be learned from the data. After passing the data through the convolutional layer, the neural network output is given as $y = f(z)$ where f is some top-layer function assumed to be differentiable. To learn the model, parameters gradients need to be computed.

(a) *Express* the gradient $\partial y/\partial w$ as a function of the input x and of the gradient $\partial y/\partial z$.

Exercise 2: Recursive Neural Networks (10 + 10 P)

Consider a recursive neural network that applies some function ϕ recursively from the leaves to the root of a binary parse tree. The function $\phi : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^h \to \mathbb{R}^d$ takes two incoming nodes $a_i, a_j \in \mathbb{R}^d$ and some parameter vector $\theta \in \mathbb{R}^h$ as input, and produces an output $a_k \in \mathbb{R}^d$. Once we have reached the root node, we apply a function $f : \mathbb{R}^d \to \mathbb{R}$ that produces some real-valued prediction. We assume that both ϕ and f are differentiable with their inputs.

Consider the sentence 'the cat sat on the mat', that is parsed as:

$$((\text{the}, \text{cat}), (\text{sat}, (\text{on}, (\text{the}, \text{mat}))))$$

The leaf nodes of the parsing tree are represented by word embeddings $a_{\text{the}}, a_{\text{cat}}, \dots \in \mathbb{R}^d$.

(a) *Draw* the computational graph associated to the application of the recursive neural network to this sentence, and write down the set of equations, e.g.

$$\begin{aligned} a_1 &= \phi(a_{\text{the}}, a_{\text{cat}}, \theta) \\ a_2 &= \phi(a_{\text{the}}, a_{\text{mat}}, \theta) \\ a_3 &= \phi(a_{\text{on}}, a_2, \theta) \\ &\vdots \\ y &= f(a_5) \end{aligned}$$

(b) *Express* the total derivative $dy/d\theta$ (taking into account all direct and indirect dependencies on the parameter θ) in terms of local derivatives (relating adjacent quantities in the computational graph).

(Hint: you can use for this the chain rule for derivatives that states that for some function $h(g_1(t), \dots, g_N(t))$ we have:

$$\frac{dh}{dt} = \frac{\partial h}{\partial g_1} \cdot \frac{dg_1}{dt} + \cdots + \frac{\partial h}{\partial g_N} \cdot \frac{dg_N}{dt}$$

where $d(\cdot)$ and $\partial(\cdot)$ denote the total and partial derivatives respectively.)

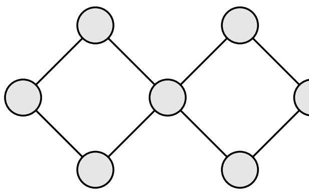
Exercise 3: Graph Neural Networks (10 + 10 P)

Graph neural networks are a fairly flexible class of neural networks that can sometimes be seen as a generalization of convolutional neural networks and recursive neural networks. We would like to study the equivalence of graph neural network with other neural networks. We will consider in this exercise graph neural network that map two consecutive layers using the equation:

$$\boldsymbol{H}_{t+1} = \rho(\boldsymbol{\Lambda}\boldsymbol{H}_t\boldsymbol{W})$$

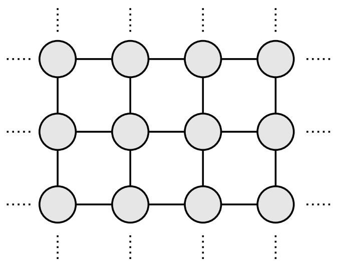
where ρ denotes the ReLU function. The adjacency matrix $\boldsymbol{\Lambda}$ is of size $N \times N$ with value 1 when there is a connection, and 0 otherwise. The matrix of parameters $\boldsymbol{W}$ is of size $d \times d$, where d is the number of dimensions used to represent each node. We also assume that the initial state $\boldsymbol{H}_0$ is a matrix filled with ones.

(a) Consider first the following undirected graph:



Because the graph is unlabeled, the nodes can only be distinguished by their connectivity to other nodes. Consider the case where the graph neural network has depth 2. *Depict* the multiple trees formed by viewing the graph neural network as a collection of recursive neural networks.

(b) Consider now the following infinite lattice graph:



which is like in the previous example undirected and unlabeled. We consider again the case where the graph neural network has depth 2. *Show* that the latter can be implemented as a 2D convolutional neural network with four convolution layers and two ReLU layer, i.e. *give* the sequence of layers and their parameters.

Exercise 4: Programming (50 P)

Download the programming files on ISIS and follow the instructions.

Structured Neural Networks

In this homework, we train a collection of neural networks including a convolutional neural network on the MNIST dataset, and a graph neural network on some graph classification task.

In [1]:
```
import torch
import torch.nn as nn

import torchvision
import torchvision.transforms as transforms
import utils
import numpy

import matplotlib
%matplotlib inline
from matplotlib import pyplot as plt
```

We first consider the convolutional neural network, which we apply in the following to the MNIST data.

In [2]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5,), (0.5,))])

trainset = torchvision.datasets.MNIST(root='./data', train=True,
                                        download=True, transform=transform)

testset = torchvision.datasets.MNIST(root='./data', train=False,
                                       download=True, transform=transform)

Xr,Tr = trainset.data.float().view(-1,1,28,28)/127.5-1,trainset.targets
Xt,Tt = testset.data.float().view(-1,1,28,28)/127.5-1,testset.targets
```

We consider for this dataset a convolution network made of four convolutions and two pooling layers.

In [3]:
```
torch.manual_seed(0)
cnn = utils.NNClassifier(nn.Sequential(
    nn.Conv2d( 1, 8, 5), nn.ReLU(), nn.MaxPool2d(2),
    nn.Conv2d( 8, 24, 5), nn.ReLU(), nn.MaxPool2d(2),
    nn.Conv2d( 24, 72, 4), nn.ReLU(),
    nn.Conv2d( 72, 10, 1)
))
```

The network is wrapped in the class `utils.NNClassifier`, which exposes scikit-learn-like functions such as `fit()` and `predict()`. To evaluate the convolutional neural network, we also consider two simpler baselines: a one-layer linear network, and standard fully-connected network composed of two layers.

In [4]:
```
torch.manual_seed(0)
lin = utils.NNClassifier(nn.Sequential(nn.Linear(784, 10)),flat=True)

torch.manual_seed(0)
fc = utils.NNClassifier(nn.Sequential(
    nn.Linear( 784, 512), nn.ReLU(), nn.Linear( 512, 10)
),flat=True)
```

Evaluating the convolutional neural network (15 P)

We now proceed with the comparision of these three classifiers.

Task:

- **Train each classifier for 5 epochs and print the classification accuracy on the training and test data (i.e. the fraction of the examples that are correctly classified). To avoid running out of memory, predict the training and test accuracy only based on the 2500 first examples of the training and test set respectively.**

In [5]:
```
for name,cl in [('linear',lin),('full',fc),('conv',cnn)]:

    # ------------------------------------
    # TODO: Replace by your code
    # ------------------------------------
    import solution
    errtr,errtt = solution.analyze(cl,Xr,Tr,Xt,Tt)
    # ------------------------------------

    print('%10s train: %.3f  test: %.3f'%(name,errtr,errtt))
```

```
    linear train: 0.910  test: 0.878
      full train: 0.966  test: 0.954
      conv train: 0.990  test: 0.982
```

We observe that the convolutional neural network reaches the higest accuracy with less than 2% of misclassified digits on the test data.

Confidently predicted digits (15 P)

We now ask whether some digits are easier to predict than others for the convolutional neural network. For this, we observe that the neural network produces at its output scores y_c for each class c. These scores can be converted to a class probability using the *softargmax* (also called *softmax*) function:

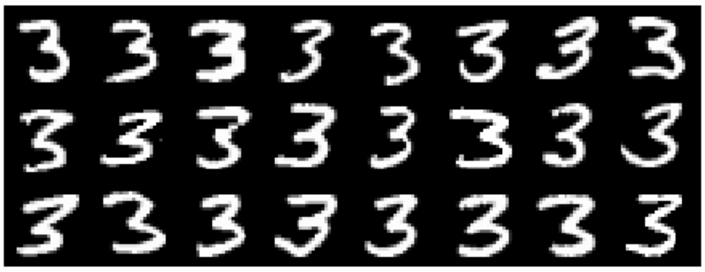
$$p_c = \frac{\exp(y_c)}{\sum_{c'=1}^{10} \exp(y_{c'})}$$

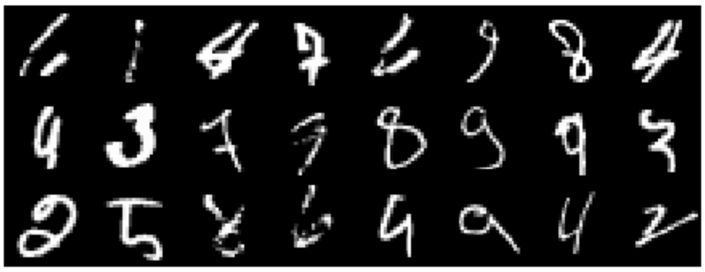
Task:

- **Find for the convolutional network the data points in the test set that are predicted with the highest probability (the lowest being random guessing). To avoid numerical unstability, your implementation should work in the log-probability domain and make use of numerically stable functions of numpy/scipy such as logsumexp.**

In [6]:
```
# ----------------------------------
# TODO: Replace by your code
# ----------------------------------
import solution
highest,lowest = solution.highestlowest(cnn,Xt)
# ----------------------------------

for digits in [highest,lowest]:
    plt.figure(figsize=(8,3))
    plt.axis('off')
    plt.imshow(digits.numpy().reshape(3,8,28,28).transpose(0,2,1,3).reshape(28*3,28*
8),cmap='gray')
    plt.show()
```





We observe that the most confident digits are thick and prototypical. Interestingly, the highest confidence digits are all from the class "3". The low-confidence digits are on the other hand thiner, and are often also more difficult to predict for a human.

Graph Neural Network (20 P)

We consider a graph neural network (GNN) that takes as input graphs of size m given by their adjacency matrix A and which is composed of the following four layers:

$$\begin{aligned} H_0 &= U \\ H_1 &= \rho(\Lambda H_0 W) \\ H_2 &= \rho(\Lambda H_1 W) \\ H_3 &= \rho(\Lambda H_2 W) \\ y &= \mathbf{1}^\top H_3 V \end{aligned}$$

U is a matrix of size $m \times h$, W is a matrix of size $h \times h$, V is a matrix of size $h \times 3$ and Λ is the normalized Laplacian associated to the graph adjacency matrix A (i.e. $\Lambda = D^{-0.5} A D^{-0.5}$ where D is a diagonal matrix containing the degree of each node), and $\rho(t) = \max(0, t)$ is the rectified linear unit that applies element-wise.

Task:

- **Implement the forward function of the GNN. It should take as input a minibatch of adjacency matrices A (given as a 3-dimensional tensor of dimensions (minibatch_size $\times$ number_nodes $\times$ number_nodes)) and return a matrix of size minibatch_size $\times$ 3 representing the scores for each example and predicted class.**

(Note: in your implementation use array operations instead of looping over all individual examples of the minibatch.)

In [7]:
```
class GNN(torch.nn.Module):

    def __init__(self,nbnodes,nbhid,nbclasses):
        torch.nn.Module.__init__(self)
        self.m = nbnodes
        self.h = nbhid
        self.c = nbclasses

        self.U = torch.nn.Parameter(torch.FloatTensor(numpy.random.normal(0,nbnodes**
-.5,[nbnodes,nbhid])))
        self.W = torch.nn.Parameter(torch.FloatTensor(numpy.random.normal(0,nbhid**-.
5,[nbhid,nbhid])))
        self.V = torch.nn.Parameter(torch.zeros([nbhid,nbclasses]))

    def forward(self,A):
        # ------------------------------------
        # TODO: Replace by your code
        # ------------------------------------
        import solution
        Y = solution.forward(self,A)
        # ------------------------------------

        return Y
```

The graph neural network is now tested on a simple graph classification task where the three classes correspond to star-shaped, chain-shaped and random-shaped graphs. Because the GNN is more difficult to optimize and the dataset is smaller, we train the network for 500 epochs. We compare the GNN with a simple fully-connected network built directly on the adjacency matrix.

In [8]:
```
Ar,Tr,At,Tt = utils.graphdata()

torch.manual_seed(0)
dnn = utils.NNClassifier(nn.Sequential(nn.Linear( 225,512), nn.ReLU(),nn.Linear(512,
3)),flat=True)
torch.manual_seed(0)
gnn = utils.NNClassifier(GNN(15,25,3))

for name,net in [('DNN',dnn),('GNN',gnn)]:
    net.fit(Ar,Tr,lr=0.01,epochs=500)
    Yr = net.predict(Ar)
    Yt = net.predict(At)
    acctr = (Yr.max(dim=1)[1] == Tr).data.numpy().mean()
    acctt = (Yt.max(dim=1)[1] == Tt).data.numpy().mean()
    print('name: %10s  train: %.3f  test: %.3f'%(name,acctr,acctt))
```

```
name:        DNN  train: 1.000  test: 0.829
name:        GNN  train: 1.000  test: 0.965
```

We observe that both networks are able to perfectly classify the training data, however, due to its particular structure, the graph neural network generalizes better to new data points.

Exercise Sheet 9

1 Convolutional Neural Networks

$$\frac{\partial y}{\partial w_u} = \sum_j \sum_t \frac{\partial y}{\partial z_t} \cdot \frac{\partial z_t}{\partial w_u} = \sum_j \sum_t \frac{\partial y}{\partial z_t} \frac{\partial}{\partial w_u} \sum_j \sum_s w_s \cdot x_{t+s}$$

$$= \sum_j \sum_t \frac{\partial y}{\partial z_t} \cdot \sum_s I(u=s) \cdot x_{t+s} = \sum_j \sum_t \frac{\partial y}{\partial z_t} x_{t+u}$$

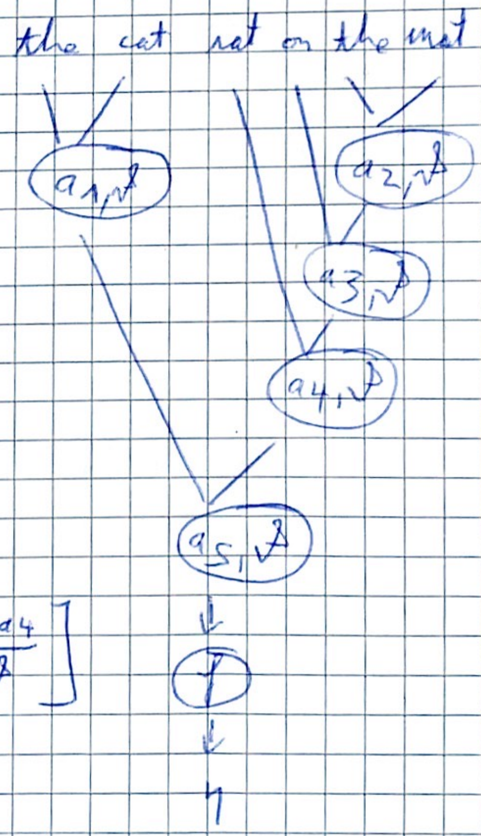
$$= \sum_j \left[\frac{\partial y}{\partial z_t} * x_{t+u} \right]_u$$

$$\frac{\partial y}{\partial w} = \sum_j \frac{\partial y}{\partial z_t} * x$$

2 Recursive Neural Networks

a)

$$a_1 = \phi(a_{the}, a_{cat}, \vartheta)$$
$$a_2 = \phi(a_{the}, a_{mat}, \vartheta)$$
$$a_3 = \phi(a_{on}, a_2, \vartheta)$$
$$a_4 = \phi(a_{sat}, a_3, \vartheta)$$
$$a_5 = \phi(a_1, a_4, \vartheta)$$
$$y = f(a_5)$$



b)

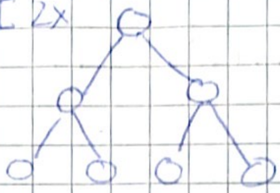
$$\frac{dy}{d\vartheta} = \frac{\partial y}{\partial a_5}\left[\frac{\partial a_5}{\partial \vartheta} + \frac{\partial a_5}{\partial a_1} \cdot \frac{\partial a_1}{\partial \vartheta} + \frac{\partial a_5}{\partial a_4} \frac{d a_4}{d\vartheta}\right]$$

$$\frac{d a_4}{d\vartheta} = \frac{\partial a_4}{\partial \vartheta} + \frac{\partial a_4}{\partial a_3} \frac{d a_3}{d\vartheta}$$

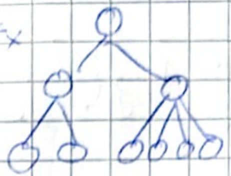
$$\frac{d a_3}{d\vartheta} = \frac{\partial a}{\partial \vartheta} + \frac{\partial a_3}{\partial a_2} \frac{\partial a_2}{\partial \vartheta}$$

3 Graph Neural Networks

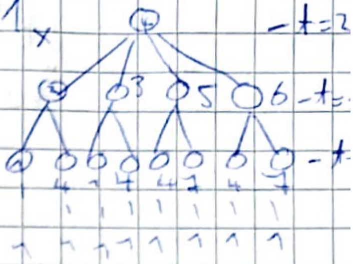
a) I 2x



II 4x



III 1_x



b) 2_x (depth)

$C_{3\times3}$: $w^{ij} = I(i \approx j)$. $\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$

$C_{1\times1}$: $w^{ij} = [w_{ij}]$

ρ : some ReLU function $\rightarrow$ $\max(0, x)$ ✓