Deep Learning
Exercise 1

Due Date : 13 November 2017
30.10.2018

Professor: Lior Wolf
TA: Eliya Nachmani

Introduction

1. Use Pytorch
2. In each group should be 3 students
3. Submit the results to - eliyan@mail.tau.ac.il. The subject of the mail should be **DL course EX1**, in the body of the mail write your names & IDs, and attach the zip file.

1 Question 1 : XOR network

In this question we will train a simple network to execute binary XOR function.
Follow these steps:

1. First, create a tensor that will contain a truth table for binary XOR. This will be our training data:

Tabell 1: XOR

input	1	1	0	0
input	1	0	1	0
output	0	1	1	0

2. Define a neural network with the following structure:

   - Input layer: tesnor of size 1×2 (two "bits")
   - Hidden layer: Fully connected with 3 hidden units
   - Activation layer: Tanh
   - Output layer: Fully connected with single output unit

3. Define a criterion for the neural network - use mean squared error.
4. Train the network using Adam optimizer. Train the network and test it for each line in the dataset you created. Write the results in .txt file.

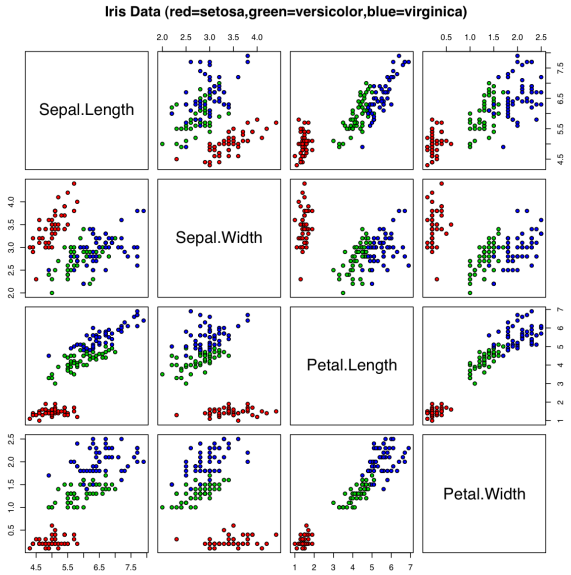
Submit your code (zip file), train and test results.

2 Problem 2: ReQU activation

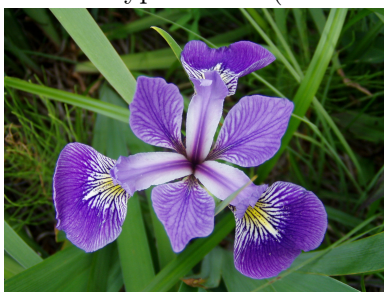
2.1 Network

In this question we will train a model on the iris flower dataset. The dataset is read from iris.data.csv (attach in EX1.zip). Some nice figures showing what the dataset looks like:

Figur 1: scatterplot of the 4 input features, with colour-coded classes (source: Wikipedia)



Figur 2: one of the types of iris (source: Wikipedia)



The model that you should implements is:

input (4 dim) => linear => non-linearity => linear => log softmax => cross-entropy loss

where the non-linearity is "ReQU", new activation function that you should implement.

2.2 ReQU

You should implement a made-up activation function that we'll call the Rectified Quadratic Unit(ReQU). Like the sigmoid and ReLU and several others, it is applied element-wise to all its inputs:

$$z_i = \mathbb{I}[x_i > 0] x_i^2 = \begin{cases} x_i^2, & \text{if } x_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

Or in matrix operations, where $\odot$ is the element-wise (aka component-wise) product, and the parenthesized expression is an element-wise truth test giving a vector of 0s (falses) and 1s (trues):

$$z = (x > 0) \odot x \odot x$$

Since the problem is easy, both models easily overfit to the training data. We don't have test data and we didn't split the training data into parts since it is small. A viable way to evaluate a model on such small data would be k-fold cross validation but we will not do this. Submit your code (zip file), train result.

2.3 Remarks/tips:

- Your layer must be able to handle minibatches. Doing so should not be difficult, though.
- You should not need to write a special case for 1 and 2 dimensions, since you're just doing an element-wise operation.
- Submit your code and the convergence plots for the model using ReQU activation.

3 Problem 3

In the previous problems, all the network trained were fully connected and the output of the networks was a classification of the input. In this problem you will:

1. Train a convolutional neural network.
2. The output of the network will be a regression.

Use the same environment you used in the previous problem.

This problem is based on this deep learning tutorial. We will train a neural network and use it to detect facial feature points: eyes, nose and mouth. It is based on a Kaggle challenge.

3.1 Data

The dataset for the Facial Keypoint Detection challenge consists of 7,049 96×96 gray-scale images. For each image, we are supposed to find the correct position (the x and y coordinates) of 15 keypoints, such as left eye center, right eye outer corner, mouth center bottom lip, and so on.

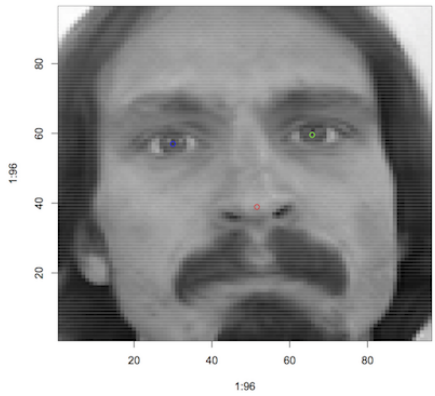
Only 2,140 images in the dataset have all 30 target values present. We'll train with these 2,140 samples only. The full tutorial uses all the samples available. The data is given in CSV format.

Data description:

- Images - are given as a vector of 9216 pixels.
- Labels - are given as a vecor of 15 coordinates (pairs of x,y) in the following order: left eye center, right eye center, left eye inner corner, left eye outer corner, right eye inner corner, right eye outer corner, left eyebrow inner end, left eyebrow outer end, right eyebrow inner end, right eyebrow outer end, nose tip, mouth left corner, mouth right corner, mouth center top lip, mouth center bottom lip.

You can download the data here - Kaggle link or get it from the attach file EX1.zip

Figur 3: Example for a single image



3.2 Question 1: Data normalization

Scale the image pixels from [0..255] to the [0..1] interval and the predication (x,y) values from [0..96] to [-1..1].

3.3 Question 2: Fully connected neural network

We will start with a simple model and improve it along the exercise. For now, consider the following model:

```
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> output]
  (1): nn.Linear(9216 -> 100)
  (2): nn.ReLU
  (3): nn.Linear(100 -> 30)
}
```

The model is composed of fully connected layers with the Rectifier activation (max(0,x)) after the first layer. The final piece of our network will be the cost function. Use mean squared error (MSE) as the cost function - which is suitable for solving regression problems like the one we have:

```
nn.MSELoss
```

1. Implement the suggested model and train it.
2. Plot the cost function value for test and train after each epoch.
3. Report loss function value (train and test) at the final epoch.

Notes:

- Allocate some of the data for test.
- Shuffle the data between epochs.

3.4 Question 3: Convolutional neural network

Let's consider the following model:

```
nn.Sequential {
  [input -> (1) -> (2) -> (3) -> (4) -> (5) -> (6) -> (7) -> (8) -> (9) -> (10) -
  (1): nn.Conv2d
  (2): nn.MaxPool2d
  (3): nn.ReLU
  (4): nn.Conv2d
  (5): nn.MaxPool2d
  (6): nn.ReLU
  (7): nn.Conv2d
  (8): nn.MaxPool2d
  (9): nn.ReLU
  (10): nn.View(15488)
  (11): nn.Linear(15488 -> 500)
  (12): nn.ReLU
  (13): nn.Linear(500 -> 500)
  (14): nn.ReLU
  (15): nn.Linear(500 -> 30)
}
```

The input of the network should be a tensor of size $1 \times 96 \times 96$. Each MaxPool2d is 2×2 with stride of 2. The first conv layer uses 32 filters of size 3×3, the second uses 64 filters of size 2×2 and the third uses 128 filters of size 2×2. The size of the output for each layer is displayed below:

Layer No.	Output Spatial Size	Output Flat Size
1	$32 \times 94 \times 94$	282752
2	$32 \times 47 \times 47$	70688
4	$64 \times 46 \times 46$	135424
5	$64 \times 23 \times 23$	33856
7	$128 \times 22 \times 22$	61952
8	$128 \times 11 \times 11$	15488
11	500	500
13	500	500
15	30	30

1. Implement the suggested model and train it (same hyper parameters). Training this neural net is much more computationally costly. It takes around 15x as long to train. Go grab a coffee.
2. Plot the cost function values (train and test) vs your previous question results.
3. Report loss function value (train and test) at the final epoch.
4. Submit your code (zip file).