STAT3007 Deep Learning, Assignment 2

2021 Semester 1, due 5pm on 1 Apr

Instructions. Please read the instructions carefully --- not following them may result in a penalty.

(a) Write down your name and student number below.

Name: [Your name]
Student Number: [Your student number]

(b) There are 4 questions. Provide your answers (text/code/output) in the cells marked with **Answer** in `Q1.ipynb` to `Q4.ipynb`. Avoid excessive output (e.g. debugging messages). Constantly save your work. Make sure that we can reproduce your answers by running your notebooks. You may find the `Tips` section in `readme.ipynb` helpful for completing the assignment.

(c) When you finish your assignment, follow the submission instructions in `readme.ipynb` to create a zip file. Log on to Blackboard, go to Assessment, Assignment 2 to submit the zip file. You can submit as many times as you want before the deadline. The last submission will be graded.

(d) Follow integrity rules, and provide citations as needed. You can discuss with your classmates, but you are required to write your solutions independently, and specify who you have discussed with in your solution. If you do not know how to solve a problem, you can get 15\% of the mark by writing down "I don't know".

Tips

For programming questions, if you want to use the free computing power offered by Colab, you can upload the notebook and work on it using Colab. Once you've all the results in the notebook, you can download it and replace your local version with it.

For theory questions, you are encouraged to write your solution using LaTeX in the notebook. However, if you want, you can also write your solutions in another file, and then use the `show_file` function provided in `util.py`. You can run the following code to import the `show_file` function. If you use such files, please add them to the `supplements` folder.

In [ ]:
```
from util import *
```

In case the `show_file` function doesn't show your images correctly, you can also use HTML command to display images in your notebooks. For example, if you've an image `supplements/Q1-sol.png`, you can also include it by running this HTML command `<img src='supplements/Q1-sol.png'>` in a text cell.

Submitting your solution

Once you are ready to submit your solution, run the code below to create `A2-sol.zip` for submission.

In [ ]:
```
from util import *
zip_sol()
```

Voila! You're ready to submit `A2-sol.zip` on Blackboard.

Q1. Perceptrons (15 marks)

In this question, we examine the representational power of the perceptron.

Consider a function $f(x_1, x_2, x_3)$ for three binary variables $x_1, x_2, x_3 \in \{0, 1\}$, with the function taking value -1 or 1. The function can be specified using the set of examples $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_8, y_8)\}$, where $\mathbf{x}_1, \ldots, \mathbf{x}_8$ are the 8 input configurations for the three binary variables, and each $y_i = f(\mathbf{x}_i)$. Now augment each input vector in D with a constant variable 1.

We say that the perceptron algorithm learns the function f if the perceptron algorithm converges on the training set D, and the model at convergence gives exactly the same input-output relationship as f.

For each of the two functions below, does the perceptron algorithm learn it? Justify your answer. If your answer is yes, derive an upper bound on the number of mistakes that the perceptron algorithm makes before convergence, when the initial parameters are 0.

(a) (5 marks) The XOR function for three binary variables $x_1, x_2, x_3 \in \{0, 1\}$ takes value 1 if an odd number of the variables are 1, and value -1 if an even number of the variables are 1.

Answer. [Write your solution here. Add cells as needed.]

(b) (10 marks) The function $f(x_1, x_2, x_3) = \begin{cases} 1 & \text{if } (x_1 \wedge x_2) \vee (x_2 \wedge x_3) \vee (x_3 \wedge x_1), \\ -1 & \text{otherwise.} \end{cases}$, where $x_1, x_2, x_3 \in \{0, 1\}$, and $\wedge$ is the logical and, and $\vee$ is the logical or.

Answer. [Write your solution here. Add cells as needed.]

Q2 Hopfield networks (35 marks)

In this question, we train a Hopfield network and use it to reconstruct noisy patterns.

(a) (5 marks) The supplments folder contain three animal image files image1.png, image2.png, and image3.png. Load the images in Python. What are the sizes of the images and their average pixel values?

You may find the skimage.io.imread function from the scikit-image library useful.

Answer. [Write your solution here. Add cells as needed.]

(b) (5 marks) For each of the 3 images, convert it into a binary pattern by setting each pixel value to -1 if it is smaller than the average pixel value, and +1 otherwise.

Answer. [Write your solution here. Add cells as needed.]

(c) (5 marks) If we want to use a Hopfield network to store the above 3 binary patterns, how many neurons do we need?

Answer. [Write your solution here. Add cells as needed.]

(d) (5 marks) Train a Hopfield network to store the 3 binary patterns. How many weights are positive and how many are negative?

Answer. [Write your solution here. Add cells as needed.]

(e) (5 marks) Write a function that accepts an input pattern, and synchronously updates the activation states of the neurons at each iteration until convergence, or up to 100 iterations.

The supplements folder also contain three images, noisy1.png, noisy2.png, and noisy3.png, obtained by adding noise to one of the 3 animal images. Use the trained Hopfield network and the above synchronous update to reconstruct the original images. Display the reconstructed images.

Note that you need to perform conversion between images and binary patterns.

Answer. [Write your solution here. Add cells as needed.]

(f) (5 marks) Write a function that accepts an input pattern, and updates all the acti- vation states of the neurons in a random order at each iteration until convergence, or up to 100 iterations.

Use the trained Hopfield network and the above semi-random update to reconstruct the original images. Display the reconstructed images.

To make your results reproducible, set the seed for the random number generator to 1 before using it. For example, if you are using numpy to generate the random ordering, you can then set the random seed using the function numpy.random.seed.

Answer. [Write your solution here. Add cells as needed.]

(g) (5 marks) Construct an input image such that it is more similar to image2 than image1, but the reconstructed image obtained using the synchronous update function is more similar to image1 than image2. Use the number of common pixels as the similarity measure. Report the similarity numbers, and display image1, image2, the input image, and the reconstructed image. Explain the reasoning behind your construction.

Answer. [Write your solution here. Add cells as needed.]

Q3. Multilayer Perceptrons (30 marks)

In this question, we will train and evaluate a single hidden layer MLP on the MNIST dataset. You should use the pytorch library useful for answering the questions below.

(a) (5 marks) Load the MNIST training and test set images and labels. You can use the torchvision.datasets.MNIST class, or load them from the data files at http://yann.lecun.com/exdb/mnist/ using your own code. Convert each digit label to a one-hot representation, which is all zero except at the position corresponding to the digit label (e.g., the one-hot representation for 1 is the vector $(0,1,0,\ldots,0) \in \mathbf{R}^{10}$).

How many examples are in the training and test sets? What is the size of each digit image?

Answer. [Write your solution here. Add cells as needed.]

(b) (10 marks) Consider a single hidden layer MLP with 100 hidden units and 10 output units, defined by $f(\mathbf{x}; W_1, W_2, b_1, b_2) = \sigma(W_2\sigma(W_1\mathbf{x} + b_1) + b_2)$, where each $\mathbf{x} \in \mathbf{R}^d$ is the vector representation of a digit image, $W_1 \in \mathbf{R}^{100\times d}$ and $W_2 \in \mathbf{R}^{10\times 100}$ are the weight matrices, $b_1 \in \mathbf{R}^{100}$ and $b_2 \in \mathbf{R}^{10}$ are the biases, and $\sigma(u)$ applies the sigmoid function to each component of a vector u.

We want to train this MLP by minimizing the quadratic loss

$$R_n(W_1, W_2, b_1, b_2) = \frac{1}{n}\sum_{i=1}^{n} ||f(\mathbf{x}_i; W_1, W_2, b_1, b_2) - y_i||_2^2,$$

where $(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n) \in \mathbf{R}^d \times \mathbf{R}^{10}$ is the training set. Note that you need to implement exactly the same loss as defined above

Starting with W_1, W_2, b_1, b_2 being 0, run gradient descent with step size 0.01 for 100 iterations.

Compute the training and test set classification errors for the trained model. Here we use the trained model to make predictions by assigning an input $\mathbf{x}$ to the index of the largest output.

Answer. [Write your solution here. Add cells as needed.]

(c) (5 marks) Repeat (b) by training the MLP starting from parameters randomly chosen from $[-0.5, 0.5]$. What are the training set and test set classification errors?

Answer. [Write your solution here. Add cells as needed.]

(d) (5 marks) Repeat (b) and (c) with each full gradient replaced by a stochastic gradient computed over mini-batches of 100 examples, and running SGD for 100 epochs (one epoch means one pass through the training data; thus 100 epochs is equivalent to 100n/100=n SGD iterations).

Answer. [Write your solution here. Add cells as needed.]

(e) (5 marks) Train the MLP using the following learning rates 0.001, 0.1, and 1 for both gradient descent and SGD, starting from 0 and random parameters. Report the training set and test set classification errors for these 12 models.

Comment the performance of all the 16 models that you obtained above. Which one has the best performance? Can you explain why?

Answer. [Write your solution here. Add cells as needed.]

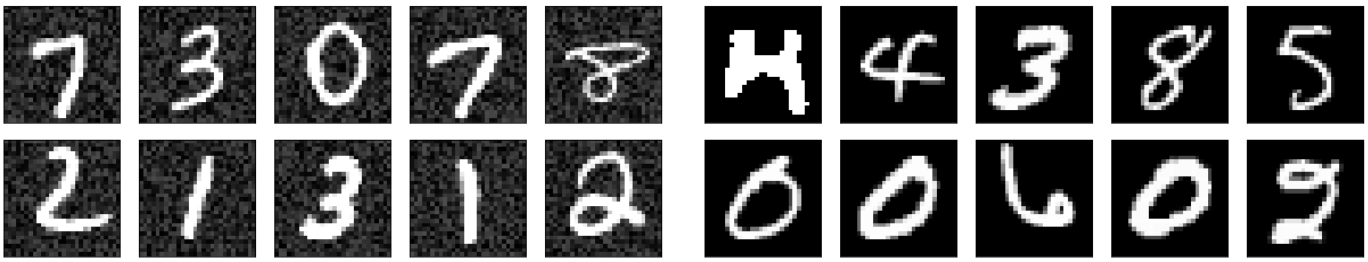
Q4. Transfer learning (20 marks)

When we learn to solve a new problem, we often leverage on knowledge that we have learned on related tasks. For example, when we learn how to play chess after learning how to play Chinese chess, we can often quickly learn new tactics for chess by relating to tactics that we have learned for Chinese chess.

Exploiting previously acquired knowledge on related tasks to learn how to solve new problems is known as transfer learning, and this idea has been applied to build machine learning systems too.

In this question, we will exploit a model learned on a noisy version of MNIST to learn a model on MNIST.

(a) (0 marks) The code below loads a model trained on a noisy version of MNIST, with all pixel values normalized to the range [0,1]. Some of the noisy MNIST images are shown on the left figure below, together with some of the original MNIST images shown on the right figure below for comparsion.



The model is actually a feature network that converts an input image into a feature vector. Read and run the code to understand how it works.

In [ ]:
```
import torch
torch.manual_seed(1)
# load the feature network
feature_net = torch.load('supplements/fnet.pt')
# create 3 random images of size 1x28x28
images = torch.rand(3, 1, 28, 28)
# compute the features for the 3 random images
feature_net(images)
```

(b) (5 marks) Load the MNIST dataset and convert the images into feature vectors using the provided feature network

Answer. [Write your solution here. Add cells as needed.]

(c) (10 marks) Pick a simple classifier of your choice, train it on MNIST using the pixel values as features, and also train it using the features extracted in (b). Compare and comment on the performances of the two models.

Answer. [Write your solution here. Add cells as needed.]

(d) (5 marks) We investigate how the features learned on noisy MNIST transfer to another variant of MNIST where each image has a small random 3x3 patches cut out. Read and run the code below to understand how it works. Repeat (c) on this damaged MNIST dataset.

In [ ]:
```
import copy
import matplotlib.pyplot as plt

def randcut(x, patch_size=(3,3)):
    x = copy.deepcopy(x)
    h, w = patch_size
    H, W = x.shape[-2:]
    for i in range(len(x)):
        a = np.random.choice(H-h)
        b = np.random.choice(W-w)
        x[i, ..., a:a+h, b:b+w] = 255
    return x

# randomly cut 3x3 patches out from MNIST images
np.random.seed(3)
x_tr_cut = randcut(x_train)
x_ts_cut = randcut(x_test)
```

In [ ]:
```
# display the first modified training image
plt.imshow(x_tr_cut[0, 0], cmap='gray')
plt.show()
```

Answer. [Write your solution here. Add cells as needed.]