

UNIVERSITY OF PENNSYLVANIA
ESE 546: PRINCIPLES OF DEEP LEARNING
FALL 2022
[09/07] HOMEWORK 1
DUE: 09/26 MON, 10.15AM ET

Changelog

- No changes yet
-

Read the following instructions carefully before beginning to work on the homework.

- You will submit solutions typeset in \LaTeX on Gradescope (strongly encouraged). You can use `hw_template.tex` on Canvas in the “Homeworks” folder to do so. If your handwriting is unambiguously legible, you can submit PDF scans/tablet-created PDFs.
- Clearly indicate the name and Penn email ID of all your collaborators on your submitted solutions.
- Start a new problem on a fresh page and mark all the pages corresponding to each problem. Failure to do so may result in your work not graded completely.
- For each problem in the homework, you should mention the total amount of time you spent on it. This helps us keep track of which problems most students are finding difficult.
- You can be informal while typesetting the solutions, e.g., if you want to draw a picture feel free to draw it on paper clearly, click a picture and include it in your solution. Do not spend undue time on typesetting solutions.
- You will see an entry of the form “HW 1 PDF” where you will upload the PDF of your solutions. You will also see entries like “HW 1 Problem 1 Code” and “HW 1 Problem 3 Code” where you will upload your solution for the respective problems.
- **For each programming problem/sub-problem, you should create a fresh .py file.** This file should contain **all** the code to reproduce the results of the problem/sub-problem, e.g., it should save the plot that is required (correctly with all the axes, title and legend) as a PDF in the same directory. You will upload the .py file as your solution for “HW 1 Problem 3 Code” or “HW 1 Problem 3 Code”. Name your file as `pennkey_hw1_problem3.py`, e.g., I will name my code as `pratikac_hw1_problem3.py`. Note, we will not accept .ipynb files (i.e., Jupyter notebooks), you should only upload .py files. If you are using Google Colab to do your homework (and I suggest that you don’t...), you can export the notebook to a .py file.

- **This is very important.** Note that the instructors will download your code and execute it themselves, so your code should be such that it can be executed independently without any errors to create all output/plots required in the problem.

Credit The points for the problems add up to 140. You only need to solve for 100 points to get full credit, i.e., your final score will be $\min(\text{your total points}, 100)$.

1 **Problem 1 (60 points, Code up on laptop, use Google Colab if it looks like you need more RAM).**

2 In this problem, we will fit the MNIST dataset using a support vector machine (SVM) using the
3 “scikit-learn” library. You can install it using

```
4 [local] pip install scikit-learn scikit-image  
5 [colab] !pip install scikit-learn scikit-image
```

8 An SVM solves an optimization problem for maximizing the margin between two classes. Support
9 that we have a binary classification problem where (x_i, y_i) are the data and ground-truth labels
10 respectively and $y_i \in \{-1, 1\}$. We would like to find a hyper-plane that separates the data such that
11 all examples with labels $y_i = +1$ are on side and all examples with labels $y_i = -1$ are on the other
12 side. This involves solving the problem

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\theta\|^2 \\ & \text{subject to} && y_i(\theta^\top x_i + \theta_0) \geq 1 \quad \forall i = 1, \dots, n; \end{aligned} \tag{1}$$

13 here θ_0 is the offset parameter and θ is the hyper-plane. You can eliminate the offset parameter by
14 appending a 1 to the data, i.e., feeding in $x' = [x, 1]$ as the data with the same labels.

15 (a) (5 point) It may not always be possible to classify a dataset cleanly into positive and negatively
16 labeled samples, i.e., there may not exist a θ that satisfies all constraints in (1). To handle such cases,
17 we relax the problem formulation. We create a “slack” variable that allows the constraint to be written
18 as

$$\text{subject to} \quad y_i(\theta^\top x_i + \theta_0) \geq 1 - \xi_i; \quad \xi_i \geq 0.$$

19 The variable ξ_i measures the degree to which we can violate the original constraint. We would like to
20 minimize the violation of the original constraints and the slack variable-based formulation of (1) will
21 use a different objective that does so. There can be many such objectives, write down one.

22 (b) (2 point) What are support samples in an SVM?

23 (c) (3 points) You can download the dataset using

```
24 from sklearn.datasets import fetch_openml  
25 from sklearn.model_selection import train_test_split  
26  
27 ds = fetch_openml('mnist_784', as_frame=False)  
28  
29 x, x_test, y, y_test = train_test_split(ds.data, ds.target,  
30                                       test_size=0.2, random_state=42)
```

33 Check whether you have downloaded the data correctly; the images in x_{train} and x_{val} are in the
34 form of a vector of length 784, this is really the flattened matrix 28×28 . You can check it by running
35 the following code.

```
36 import matplotlib.pyplot as plt  
37 a = x_train[0].reshape((28,28))  
38 plt.imshow(a)
```

41 **Fitting SVMs requires a decent amount of RAM (think of why, and write your answer in part**
42 **(e)).** We will therefore down-sample the original 28×28 images to 14×14 using the following code.

```

43
44 # code for down-sampling
45 import cv2
46 b = cv2.resize(b, (14,14))
47 plt.imshow(b)
48

```

49 In this problem you will create a dataset of 1000 samples for each digit (10,000 samples in total).
 50 You can sub-sample x to create this dataset (we are doing this step simply to reduce the amount of
 51 time it takes to fit the SVM). From within these 10,000 samples, we will construct our actual training
 52 dataset (80%) and validation dataset (20%) (you can sample randomly).

53 (d) (15 points) Create the SVM classifier in scikit-learn using

```

54
55 classifier = svm.SVC(C=1.0, kernel='rbf', gamma='auto')
56

```

57 What do the parameters C and γ do? What are their default values? Fit the SVM classifier to the
 58 data and predict the labels of the validation dataset using the trained classifier. Report the validation
 59 error. Note down the ratio of the number of support samples to the total number of training samples
 60 for your trained classifier. Run the classifier on x_{test} , y_{test} and report the classification error.
 61 Report the 10-class confusion matrix on the test data. Do you notice any patterns about what kind of
 62 mistakes are being made? Can you explain these mistakes intuitively?

63 (e) (5 points) Read the manual of `svm.SVC` carefully. Identify all the options that you may not
 64 have seen in your previous course on SVMs. Libraries that are used in production such as scikit-learn
 65 will have numerous knobs to improve the performance; these knobs often implement state of the art
 66 research and it is useful to know them. What does the parameter named “shrinking” in `svm.SVC` do?
 67 Explain what optimization algorithm is used to fit the SVM in scikit-learn.

68 (f) (5 points) The mathematical formulation of the SVM that we saw above is for a binary classifier.
 69 The MNIST dataset clearly consists of digits from 0-9 and has 10 classes in total. How does `svm.SVC`
 70 handle multiple classes? Can you think of any alternative ways to use binary classifiers to perform
 71 multi-class classification?

72 (g) (5 points) Use the `sklearn.model_selection.GridSearchCV` function to pick a better value than
 73 the default one for the hyper-parameter C . Try at least 5 different hyper-parameters. Show all the
 74 hyper-parameters tried by the method and their accuracies. How will you pick the best value of the
 75 hyper-parameter using your validation set?

76 (h) **The following part is computationally intensive. In this case, you can use a training**
 77 **dataset of 100 samples/class and a validation set of 100 images/class. Sample x, y randomly to**
 78 **create this dataset. No need for a test set.**

79 The default kernel in `svm.SVC` is a radial basis function. The MNIST dataset consists of images and
 80 since images have local regularities we can build a better classifier by exploiting them. The mammalian
 81 visual cortex consists of cells that can be modeled as Gabor functions (named after Dennis Gabor, a
 82 Hungarian physicist who invented holography). See https://en.wikipedia.org/wiki/Gabor_filter for
 83 examples.

84 Let us represent each image as a function $I(x, y)$, this function gives the intensity at pixel location
 85 (x, y) . A Gabor filter is given by a function

$$g(x, y; \theta, F, \sigma_x, \sigma_y) = \exp(i 2\pi F p) \exp\left(-\pi\left(\frac{p^2}{\sigma_x^2} + \frac{q^2}{\sigma_y^2}\right)\right)$$

86 where $p = x \cos \theta + y \sin \theta$ and $q = -x \sin \theta + y \cos \theta$. First, note that this filter is a complex
 87 function. Convolution of the original image $I(x, y)$ with the filter $g(x, y)$ will result in two sets of
 88 coefficients, one real and the other imaginary. The parameters we will be concerned with are:

- 89 • F this is the spatial frequency of the filter,
- 90 • θ the rotation angle of the Gaussian,
- 91 • σ_x, σ_y : standard deviation of the kernel in the X and Y directions, and
- 92 • the parameter “bandwidth” in the code below is inversely related to the standard deviation
- 93 fixed the frequency.

94 You can read [this webpage](#) for a simple introduction to these filters (this is given in the OpenCV
 95 format). You can also read this more mathematical [tutorial on Gabor filters](#) which is given in the
 96 scikit-image format that we discussed above.

97 We will use the scikit-image library which implements a smaller machine learning-specific set
 98 of image processing functions. Alternatively, you can also use the `cv2.getGaborKernel` function in
 99 OpenCV.

```
100 from skimage.filters import gabor_kernel, gabor
101 import numpy as np
102
103 freq, theta, bandwidth = 0.1, np.pi/4, 1
104 gk = gabor_kernel(frequency=freq, theta=theta, bandwidth=bandwidth)
105 plt.figure(1); plt.clf(); plt.imshow(gk.real)
106 plt.figure(2); plt.clf(); plt.imshow(gk.imag)
107
108 # convolve the input image with the kernel and get co-efficients
109 # we will use only the real part and throw away the imaginary
110 # part of the co-efficients
111 image = x[0].reshape((14,14))
112 coeff_real, _ = gabor(image, frequency=freq, theta=theta,
113                      bandwidth=bandwidth)
114 plt.figure(1); plt.clf(); plt.imshow(coeff_real)
```

117 (j) (20 points) Run the above code a few times with different parameters for F, θ and bandwidth to
 118 see how the filter changes in shape and size and the corresponding output after convolution. We will
 119 create a filter bank that consists of multiple Gabor filters of fixed parameters. Instead of considering
 120 the pixel intensities of the MNIST images as the features for training the SVM, the co-efficients of the
 121 Gabor filter-bank will be used to train the SVM. You can pick

```
122 theta = np.arange(0, np.pi, np.pi/4)
123 frequency = np.arange(0.05, 0.5, 0.15)
124 bandwidth = np.arange(0.3, 1, 0.3)
```

127 This gives a total of 36 filters in the filter-bank. We therefore have converted a $14 \times 14 = 196$ pixel
 128 image into a vector of length $196 \times 36 = 7056$. Plot the filter-bank to see that it gives you a good
 129 spread of different filters. You want a diverse filter bank that can capture different rotations and scales.
 130 Train the SVM on these features and report the training and validation accuracy.

131 Increase the number of filters next. You might have to use PCA to reduce the dimensionality of the
 132 dataset to be able to fit the SVM in RAM; use scikit-learn to do so.

133 **Problem 2 (10 points).** Prove Jensen's inequality: for any random variable X with expectation μ
 134 and a convex, finite function φ

$$\mathbb{E}_X[\varphi(X)] \geq \varphi(\mu).$$

135 You can assume that the random variable X takes values in a finite set. If you want to prove it in a
 136 more general setting, you can assume that the function φ is differentiable.

137 **Problem 3 (70 points, Do this on your laptop).** You will write code to train a neural network
 138 completely from scratch using only Numpy and basic Python (note, you cannot use PyTorch/Tensor-
 139 Flow/other deep learning library except for downloading the data).

140 (a) (5 points) Download the MNIST dataset using the following code.

```
141 import torchvision as thv
142 train = thv.datasets.MNIST('./', download=True, train=True)
143 val = thv.datasets.MNIST('./', download=True, train=False)
144 print(train.data.shape, len(train.targets))
145
```

146 The training dataset has 60,000 images while the validation dataset has 10,000 images spread
 147 roughly equally across 10 classes. Take 50% of the images *from each class* for training and validation,
 148 i.e., about 30,000 training images and 5,000 validation images, almost evenly spread across all classes
 149 with a few minor differences. We will use this smaller dataset in this problem. **Plot the images of a**
 150 **few randomly chosen images from your dataset and if their labels are correct.** This is a good way
 151 to make sure that there is nothing wrong in your data.

152 (b) (10 points) We will next implement different parts of a typical neural network. First write a
 153 linear layer; this includes the forward function

$$h^{(l+1)} = h^{(l)}W^T + b$$

154 and the corresponding backward function that takes the gradient $\overline{h^{(l+1)}}$ and outputs \overline{W} , \overline{b} and $\overline{h^{(l)}}$.
 155 Remember to write your function in such a way that it takes in a mini-batch of vectors $h^{(l)}$ as the
 156 input, i.e., if the feature vector $h^{(l)}$ is a -dimensional, for ℓ images in the mini-batch, your forward
 157 function will take as input

$$h^{(l)} \in \mathbb{R}^{\ell \times a}$$

158 use

$$W \in \mathbb{R}^{c \times a}, \quad b \in \mathbb{R}^c$$

159 and output a mini-batch of feature vectors of size

$$h^{(l+1)} \in \mathbb{R}^{\ell \times c}.$$

160 Note that in this problem we have $a = 784$ because there are 28×28 pixels in MNIST images and
 161 $c = 10$ because there are 10 classes in MNIST. You should use numpy to write the forward function;

do not use a for loop for computing the mini-batch-ed forward because it will be too slow for the next parts of the problem. You are advised to first write this function for $\ell = 1$ to understand the process and then you can extend it to $\ell > 1$. Some pseudo code is given below.

```

class linear_t:
    def __init__(self):
        # initialize to appropriate sizes, fill with Gaussian entires
        # normalize to make the Frobenius norm of w, b equal to 1
        self.w, self.b = ...

    def forward(self, h^l):
        h^{l+1} = ...
        # cache h^l in forward because we will need it to compute
        # dw in backward
        self.hl = h^l
        return h^{l+1}

    def backward(self, dh^{l+1}):
        dh^l, dw, db = ...
        self.dw, self.db = dw, db
        # notice that there is no need to cache dh^l
        return dh^l

    def zero_grad(self):
        # useful to delete the stored backprop gradients of the
        # previous mini-batch before you start a new mini-batch
        self.dw, self.db = 0*self.dw, 0*self.db

```

(c) (5 points) Implement the rectified linear unit (ReLU) layer next. This will take the form of

$$h^{(l+1)} = \max(0, h^{(l)})$$

where the max is performed element-wise on the elements of $h^{(l)}$. Write the forward function and the corresponding backward function.

(d) (10 points) Next we will write a combined softmax and cross-entropy loss layer. This is a layer that first performs the operation

$$h_k^{(l+1)} = \frac{e^{h_k^{(l)}}}{\sum_{k'} e^{h_{k'}^{(l)}}}$$

where $h_k^{(l)}$ is the k^{th} element of the vector $h^{(l)}$. The input to this layer, i.e., $h^{(l)}$ are called the “logits”. The output of this layer is a scalar, it is the negative log-probability of predicting the correct class, i.e.,

$$\ell(y) = -\log(h_y^{(l+1)}).$$

where y is the true label of the image. For a mini-batch with ℓ images, the average loss will be

$$\ell(\{y_i\}_{i=1,\dots,\ell}) = -\frac{1}{\ell} \sum_{i=1}^{\ell} \log(h_{y_i}^{(l+1)}).$$

You will again implement a forward function and a backward function for it yourself; remember to implement both functions to take in a mini-batch of inputs. The pseudo-code for the log-softmax

layer is similar to that of the fully-connected layer. It does not have any parameters to initialize and therefore does not need the zero_grad method.

```

class softmax_cross_entropy_t:
    def __init__(self):
        # no parameters, nothing to initialize

    def forward(self, h^l, y):
        h^{l+1} = ...
        # compute average loss ell(y) over a mini-batch
        ell = ...
        error = ...
        return ell, error

    def backward(self):
        # as we saw in the notes, the backprop input to the
        # loss layer is 1, so this function does not take any
        # arguments
        dh^l = ...
        return dh^l

```

We can also output the error of predictions in the forward function. It is computed as

$$\text{error} = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathbf{1}_{\{y_i \neq \arg\max_k h_k^{(l+1)}\}}$$

and measures the number of mistakes the network makes.

(e) (10 points) Before moving on to training, let us check whether we have implemented the forward and backward correctly for all the three layers. Consider the function for the linear layer. Use a batch-size $\ell = 1$ for this part. The forward function for the linear layer implements

$$h^{(l+1)} = h^{(l)} W^\top + b$$

which is easy enough. However, we would like to check our implementation of the backward function.

```

def backward(self, dh^{l+1}):
    dh^l, self.dw, self.db = ...
    return dh^l

```

Think carefully about your implementation of the backward function. Notice that if you call the backward function with the argument $\overline{h^{l+1}} = [0, 0, \dots, 0, 1, 0, 0, \dots]$, i.e., there is a 1 at the k^{th} element, the function is going to calculate the quantities

$$\text{self.dw} = \frac{\partial h_k^{(l+1)}}{\partial W}, \quad \text{self.db} = \frac{\partial h_k^{(l+1)}}{\partial b}, \quad \text{dh}^{(l)} = \frac{\partial h_k^{(l+1)}}{\partial h^{(l)}}.$$

We now compute the estimate of the derivative using finite-differences, e.g.,

$$\frac{\partial h_k^{(l+1)}}{\partial W_{ij}} \approx \frac{\left(h^{(l)} (W + \epsilon)^\top \right)_k - \left(h^{(l)} (W - \epsilon)^\top \right)_k}{2\epsilon_{ij}}$$

237 where ϵ is a matrix with a Gaussian random variable as the $(ij)^{\text{th}}$ entry and zero everywhere else. In
 238 simple words, you can perturb the $(ij)^{\text{th}}$ element of weight W by ϵ_{ij} , compute the right hand-side of
 239 the finite-difference estimate above and compare it with the $(ij)^{\text{th}}$ element of your variable `self.dw`.

240 This idea checks the gradient with respect to only one element of W , namely W_{ij} . Do this for
 241 about 10 randomly chosen elements of W and a few (5 should be enough) different entries k of
 242 $h_k^{(l+1)}$ and check if the answer matches `self.dw` that you have implemented in the backward function.
 243 Repeat this process for the other two gradients.

244 **Do not move on to the next part until you are convinced your implementation of forward/back-**
 245 **ward is correct for all the three layers. It is essential that the gradient is implemented correctly,**
 246 **your training will not work if the gradient is wrong.**

247 (f) (10 points) You will now train your neural network. The pseudo-code looks as follows:

```

248 # load dataset
249 ...
250
251 # initialize all the layers
252 l1, l2, l3 = linear_t(), relu_t(), softmax_cross_entropy_t()
253 net = [l1, l2, l3]
254
255 # train for at least 1000 iterations
256 for t in range(1000):
257     # 1. sample a mini-batch of size = 32
258     # each image in the mini-batch is chosen uniformly randomly from the
259     # training dataset
260     x, y = ...
261
262     # 2. zero gradient buffer
263     for l in net:
264         l.zero_grad()
265
266     # 3. forward pass
267     h1 = l1.forward(x)
268     h2 = l2.forward(h1)
269     ell, error = l3.forward(h2, y)
270
271     # 4. backward pass
272     dh2 = l3.backward()
273     dh1 = l2.backward(dh2)
274     dx = l1.backward(dh1)
275
276     # 5. gather backprop gradients
277     dw, db = l1.dw, l1.db
278
279     # 6. print some quantities for logging
280     # and debugging
281     print(t, ell, error)
282     print(t, np.linalg.norm(dw/l1.w), np.linalg.norm(db/l1.b))
283
284     # 7. one step of SGD
285 
```

```
286     ll.w = ll.w - lr*dw
287     ll.b = ll.b - lr*db
288
```

289 You can pick the learning rate to be $lr = 0.1$. **Plot the training loss and training error as a**
290 **function of the number of weight updates.** Make sure that the training loss decreases with the
291 number of updates. You should try to get better than/around 15% error on the training dataset after
292 10,000-50,000 updates.

293 (g) (5 points) We have implemented the training loop. Write the corresponding code for computing
294 the validation loss and error.

```
295 def validate(w, b):
296     # 1. iterate over mini-batches from the validation dataset
297     # note that this should not be done randomly, we want to check
298     # every image only once
299
300     loss, tot_error = 0, 0
301     for i in range(0, 5000, 32):
302         x, y = val.data[i:i+32], val.targets[i:i+32]
303
304         # 2. compute forward pass and error
305
```

307 **Plot the validation loss and validation error as a function of the number of weight updates,**
308 **every 1000 weight updates.**

309 If everything works as expected, congratulations! You have implemented your own little library
310 for training neural networks, completely from scratch!

311 (h) (15 points) Repeat the entire process in parts (b)-(g) using the pre-built functions inside PyTorch.
312 You will take help of the code provided in the recitation sessions for this purpose. Train the network
313 for at least 10,000 weight updates this time. Plot the training loss, training error, validation loss and
314 the validation error as a function of the number of weight updates.