CS 446 / ECE 449 — Homework 3

your NetID here

Version 2.0

Instructions.

- Homework is due **Sunday, October 16**, at 11:59 **AM** CST; you have **3** late days in total for **all Homeworks**.
- The template for coding problems are available at this link.
- Everyone must submit individually at gradescope under `Homework 3` and `Homework 3 Code`.
- The "written" submission at `Homework 3` **must be typed**, and submitted in any format gradescope accepts (to be safe, submit a PDF). You may use LaTeX, markdown, google docs, MS word, whatever you like; but it must be typed!
- When submitting at `Homework 3`, gradescope will ask you to **mark out boxes around each of your answers**; please do this precisely!
- Please make sure your NetID is clear and large on the first page of the homework.
- Your solution **must** be written in your own words. Please see the course webpage for full **academic integrity** information. You should cite any external reference you use.
- We reserve the right to reduce the auto-graded score for `Homework 3 Code` if we detect funny business (e.g., your solution lacks any algorithm and hard-codes answers you obtained from someone else, or simply via trial-and-error with the autograder).
- When submitting to `Homework 3 Code`, only upload `hw3.py`. Additional files will be ignored.

Version History.

1. Initial version.
2. Updated for Fall 2022.

1. Support Vector Machines.

Recall that the dual problem of a hard-margin SVM is

$$\max_{\boldsymbol{\alpha}\in\mathcal{C}} \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i\alpha_j y_i y_j \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}),$$

where the domain $\mathcal{C} = [0, \infty]^N = \{\boldsymbol{\alpha} : \alpha_i \geq 0\}$

(a) **Soft margin SVMs.** Derive the dual problem and prove that the domain is $\mathcal{C} = [0, C]^N = \{\boldsymbol{\alpha} : 0 \leq \alpha_i \leq C\}$ for a soft-margin SVM. We assume that the optimization objective for this soft-margin SVM is

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{N} \xi_i \quad s.t.\ 1 - \xi_i \leq y^{(i)}(\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)}) + b),\ \xi_i \geq 0, \forall i \in \{1, 2, ..., N\}$$

Hint: A sketch proof is briefly mentioned in Lecture 11 slides. The purpose of this problem is to ensure that you comprehend the Lagrangian and are familiar with the basic notations.

(b) L_2 **norm soft margin SVMs.** Derive the dual problem. We assume that the optimization objective for this L_2 soft-margin SVM is

$$\min_{\boldsymbol{w},\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C}{2}\sum_{i=1}^{N} \xi_i^2 \quad s.t.\ 1 - \xi_i \leq y^{(i)}(\boldsymbol{w}^\top \phi(\boldsymbol{x}^{(i)}) + b),\ \xi_i \geq 0, \forall i \in \{1, 2, ..., N\}$$

(c) i. Denote the margin for hard-margin SVM as ρ, and denote the optimal primal solution as $\boldsymbol{w}^*$, prove that

$$\frac{1}{\rho^2} = \|\boldsymbol{w}^*\|_2^2$$

Hint: Think about the definition of margin and its expression. Also think about the complementary slackness.

ii. Assume we use radial basis kernel function $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp(-\frac{1}{2}\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2^2)$. If the test point $\boldsymbol{x}_0$ is far away from any training point $\boldsymbol{x}^{(i)}$, prove that $\boldsymbol{w}^{*\top}\phi(\boldsymbol{x}_0) + b \approx b$. The definition of $\boldsymbol{w}^*$ is the same as above.

iii. Assume we use kernel function $\kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp(-\frac{1}{\tau^2}\|\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}\|_2^2)$, where $\tau \in \mathbb{R}$. The training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^N$ consists of N points separated by at least a distance of ϵ, which means $\|\boldsymbol{x}^{(j)} - \boldsymbol{x}^{(i)}\| \geq \epsilon$ for any $i \neq j$. Recall that SVM classifier can be written as $f(\boldsymbol{x}) = \sum_i \alpha_i y^{(i)} \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}) + b$. Here we simplify that $\alpha_1 = \alpha_2 = \ldots = \alpha_N = 1$ and $b = 0$.
Prove that τ should satisfy $\tau \leq \frac{m}{\ln(N-1)}$, if the training data $\{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^N$ can be correctly classified ($\|f(x) - y^{(i)}\| < 1$ for all $\boldsymbol{x} \in \{\boldsymbol{x}^{(i)}\}_{i=1}^N$).
Hint. Consider triangle inequality $|a + b| \leq |a| + |b|$

Solution.

Your solution here.

2. Neural Networks

Deep narrow networks & shallow wide networks

ReLU. For $\boldsymbol{x} = [x_1, ..., x_d]^T \in \mathbb{R}^d$, ReLU function is defined as $\vec{\sigma_\text{r}}(\boldsymbol{x}) = [\max\{0, x_1\}, ..., \max\{0, x_d\}]^T$.

Deep network. Consider the mapping $f : [0, 1]^d \to \mathbb{R}$ can be written as a network with L ReLU layer of width $w_1, w_2, \ldots, w_L$, specifically,

$$f(\boldsymbol{x}) = \boldsymbol{w}_L \vec{\sigma_\text{r}}(\boldsymbol{w}_{L-1} \vec{\sigma_\text{r}}(\ldots(\boldsymbol{w}_2 \vec{\sigma_\text{r}}(\boldsymbol{w}_1 \boldsymbol{x} + b_1) + b_2)\ldots) + b_{L-1}) + b_L,$$

where $\boldsymbol{w}_1 \in \mathbb{R}^{w_1 \times d}$, $\boldsymbol{w}_2 \in \mathbb{R}^{w_2 \times w_3}, \ldots \boldsymbol{w}_L \in \mathbb{R}^{w_L \times 1}$ and $b_1, \ldots, b_L \in \mathbb{R}$.

A shallow network $g : [0, 1]^d \to \mathbb{R}$ has only a single ReLU layer of width m,

$$g(\boldsymbol{x}) = \boldsymbol{w}_2 \vec{\sigma_\text{r}}(\boldsymbol{w}_1 \boldsymbol{x} + b_1),$$

where $\boldsymbol{w}_1 \in \mathbb{R}^{m \times d}$ and $\boldsymbol{w}_2 \in \mathbb{R}^{1 \times m}$.

(a) Prove $\forall \boldsymbol{x} \in \mathbb{R}^d, \vec{\sigma_\text{r}}^{\,n} \boldsymbol{x} = \vec{\sigma_\text{r}}(\vec{\sigma_\text{r}}(\ldots \vec{\sigma_\text{r}}(\boldsymbol{x}))) = \vec{\sigma_\text{r}} \boldsymbol{x}$

(b) Prove $\forall \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{x} = \vec{\sigma_\text{r}}(\boldsymbol{x}) - \vec{\sigma_\text{r}}(-\boldsymbol{x})$

(c) Construct a deep network with m ReLU layers and width $d + 3$ which exactly computes g. (width of each ReLU layer satisfies $w_1 = w_2 = \ldots = w_{d+3} = m$)
Remark: this reveals some convenient properties of ReLUs.

Note: For all questions below which require numerical answers, round up your final answers to **four decimal places**. For integers, you may drop trailing zeros.

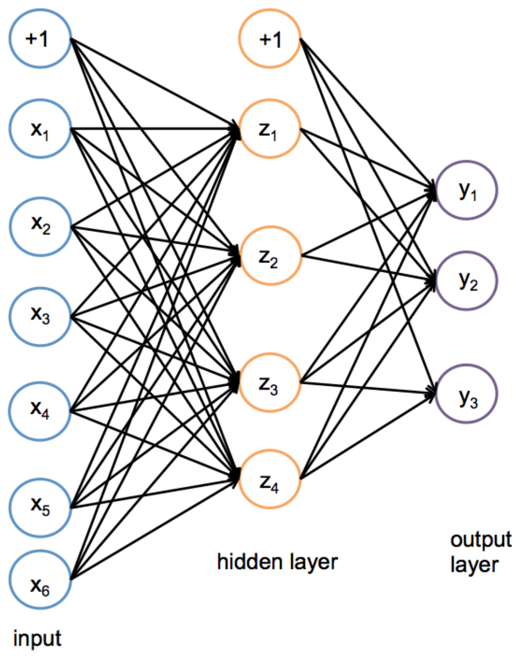

Figure 1: A One Hidden Layer Neural Network

Network Overview

Consider the neural network with one hidden layer shown in Figure 1. The input layer consists of 6 features $\boldsymbol{x} = [x_1, ..., x_6]^T$, the hidden layer has 4 nodes $\boldsymbol{z} = [z_1, ..., z_4]^T$, and the output layer is a probability distribution $\boldsymbol{y} = [y_1, y_2, y_3]^T$ over 3 classes. We also add a bias to the input, $x_0 = 1$ and the hidden layer $z_0 = 1$, both of which are fixed to 1.

We adopt the following notation:

(a) Let $\boldsymbol{\alpha}$ be the matrix of weights from the inputs to the hidden layer.
(b) Let $\boldsymbol{\beta}$ be the matrix of weights from the hidden layer to the output layer.
(c) Let $\alpha_{j,i}$ represents the weight going *to* the node z_j in the hidden layer *from* the node x_i in the input layer (e.g. $\alpha_{1,2}$ is the weight from x_2 to z_1)
(d) Let $\beta_{k,j}$ represents the weight going *to* the node y_k in the output layer *from* the node z_j in the hidden layer.
(e) We will use a *sigmoid activation function (σ)* for the hidden layer and a *softmax* for the output layer.

Network Details

Equivalently, we define each of the following.

The input:

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]^T$$

Linear combination at the first (hidden) layer:

$$a_j = \alpha_{j,0} + \sum_{i=1}^{6} \alpha_{j,i} x_i, \quad j \in \{1, \dots, 4\}$$

Activation at the first (hidden) layer:

$$z_j = \sigma(a_j) = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \dots, 4\}$$

Linear combination at the second (output) layer:

$$b_k = \beta_{k,0} + \sum_{j=1}^{4} \beta_{k,j} z_j, \quad k \in \{1, \dots, 3\}$$

Activation at the second (output) layer:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum\limits_{l=1}^{3} \exp(b_l)}, \quad k \in \{1, \dots, 3\}$$

Note that the linear combination equations can be written equivalently as the product of the weight matrix with the input vector. We can even fold in the bias term α_0 by thinking of $x_0 = 1$, and fold in β_0 by thinking of $z_0 = 1$.

Loss

We will use cross entropy loss, $\ell(\hat{\mathbf{y}}, \mathbf{y})$. If $\mathbf{y}$ represents our target (true) output, which will be a **one-hot vector** representing the correct class, and $\hat{\mathbf{y}}$ represents the output of the network, the loss is calculated as:

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{3} y_k \log(\hat{y}_k)$$

Prediction

When doing prediction, we will predict the arg max of the output layer. For example, if $\hat{\mathbf{y}}$ is such that $\hat{y}_1 = 0.3,\ \hat{y}_2 = 0.2,\ \hat{y}_3 = 0.5$ we would predict class 3 for the input $\mathbf{x}$. If the true class from the training data $\mathbf{x}$ was 2 we would have a **one-hot vector y** with values $y_1 = 0,\ \ y_2 = 1,\ \ y_3 = 0$.

(a) We initialize the weights as:

$$\boldsymbol{\alpha} = \begin{bmatrix} 1 & 2 & 3 & 0 & 1 & 3 \\ 3 & 1 & 2 & 1 & 0 & 2 \\ 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 0 & 2 & 1 & 2 & 2 \end{bmatrix}$$

$$\boldsymbol{\beta} = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & 1 & 1 & 2 \\ 3 & 1 & 1 & 1 \end{bmatrix}$$

And weights on the bias terms ($\alpha_{j,0}$ and $\beta_{k,0}$) are initialized to 1.
You are given a training example $\mathbf{x}^{(1)} = [1, 1, 0, 0, 1, 1]^T$ with label class 2, so $\mathbf{y}^{(1)} = [0, 1, 0]^T$. Using the initial weights, run the feed forward of the network over this training example (without rounding during the calculation) and then answer the following questions.

i. What is the value of a_1?
ii. What is the value of z_1?
iii. What is the value of a_3?
iv. What is the value of z_3?
v. What is the value of b_2?
vi. What is the value of $\hat{y}_2$?
vii. Which class value we would predict on this training example?
viii. What is the value of the total loss on this training example?

(b) Now use the results of the previous question to run backpropagation over the network and update the weights. Use the learning rate $\eta = 1$.

During your backpropagation calculations, **DON'T** do any rounding. Then answer the following questions: (in your final responses round to four decimal places)

i. What is the updated value of $\beta_{2,1}$?
ii. What is the updated weight of the hidden layer bias term applied to y_1 (i.e. $\beta_{1,0}$)?
iii. What is the updated value of $\alpha_{3,4}$?
iv. If we ran backpropagation on this example for a large number of iterations and then ran feed forward over the same example again, which class would we predict?

Solution.

Your solution here.

3. Implementing Support Vector Machine

(a) Recall the dual problem of SVM in the previous problem and the domain $\mathcal{C} = [0, \infty]^N = \{\boldsymbol{\alpha} : \alpha_i \geq 0\}$ for a hard-margin SVM and $\mathcal{C} = [0, C]^N = \{\boldsymbol{\alpha} : 0 \leq \alpha_i \leq C\}$ for a soft-margin SVM. We can solve this dual problem by projected gradient descent, which starts from some $\boldsymbol{\alpha}_0 \in \mathcal{C}$ (e.g., $\mathbf{0}$) and updates as follows:

$$\boldsymbol{\alpha}_{t+1} = \Pi_{\mathcal{C}} \left[\boldsymbol{\alpha}_t - \eta \nabla f(\boldsymbol{\alpha}_t) \right].$$

Here $\Pi_{\mathcal{C}}[\boldsymbol{\alpha}]$ is the *projection* of $\boldsymbol{\alpha}$ onto $\mathcal{C}$, defined as the closest point to $\boldsymbol{\alpha}$ in $\mathcal{C}$:

$$\Pi_{\mathcal{C}}[\boldsymbol{\alpha}] := \underset{\boldsymbol{\alpha}' \in \mathcal{C}}{\arg\min} \|\boldsymbol{\alpha}' - \boldsymbol{\alpha}\|_2.$$

If $\mathcal{C}$ is convex, the projection is uniquely defined. With such information, in your **written submission**, **prove that**

$$\left(\Pi_{[0,\infty)^n}[\boldsymbol{\alpha}]\right)_i = \max\{\alpha_i, 0\},$$
$$\left(\Pi_{[0,C]^n}[\boldsymbol{\alpha}]\right)_i = \min\{\max\{0, \alpha_i\}, C\}.$$

Hint: In this setting, since we have exactly the same domain for α_i in $\mathcal{C}$ for all is, each α_i can be considered independently. In this case, the minimization of $\|\boldsymbol{\alpha}' - \boldsymbol{\alpha}\|$ can also be considered independently for each i.

(b) Implement an `svm_solver()`, using projected gradient descent formulated as above. Initialize your $\boldsymbol{\alpha}$ to zeros. See the docstrings in `hw3.py` for details.

Remark: In this problem, you are allowed to use the `.backward()` function in PyTorch. However, then you may have to use in-place operations like `clamp_()`, otherwise the gradient information is destroyed.

Library routines: `torch.outer, torch.clamp, torch.Tensor.backward, torch.tensor.detach, with torch.no_grad():, torch.Tensor.requires_grad_, torch.tensor.grad.zero_, .`

(c) Implement an `svm_predictor()`, using an optimal dual solution, the training set, and an input. See the docstrings in `hw3.py` for details.

Note: You don't need to convert the output of `svm_predictor()` to ± 1. Please just return the original output of SVM i.e., $\boldsymbol{w}^\top \boldsymbol{u} + b$ at the bottom of Lecture 10 Page 21.

Hint: Just in this subproblem, feel free to use iterations.

(d) On the area $[-8, 8] \times [-8, 8]$, plot the contour lines of the following kernel SVMs, trained on the XOR data. Different kernels and the XOR data are **provided in `hw3_utils.py`**. Learning rate 0.1 and 10000 steps should be enough. To draw the contour lines, you can use `hw3_utils.svm_contour()`.

- The polynomial kernel with degree 3.
- The RBF kernel with $\sigma = 1$.
- The RBF kernel with $\sigma = 2$.
- The RBF kernel with $\sigma = 5$.

Include these four plots in your **written submission**.

Solution.

Your solution here.

4. Implementing Convolutional Neural Networks.

In this problem, you will use convolutional neural networks to learn to classify handwritten digits. The digits will be encoded as 8x8 matrices with 1 input channel.

The layers of your neural network should be:

- A 2D convolutional layer (`torch.nn.Conv2d`) with 7 output channels, with kernel size 3
- A 2D maximimum pooling layer (`torch.nn.MaxPool2d`), with kernel size 2
- A 2D convolutional layer (`torch.nn.Conv2d`) with 3 output channels, with kernel size 2
- A fully connected (`torch.nn.Linear`) layer with 10 output features

Implementation details:

- A convolutional layer takes an input shape of `(batch_size, num_channels, H, W)`, but the input of the network should have a shape `(batch_size,H, W)`. Please convert the shape to the compatible shape with `torch.unsqueeze`.
- Apply a 2D batch normalization (`torch.nn.BatchNorm2d`) **and then** a ReLU activation function, to **the output of each** of your convolutional layers **before** feeding them to your next layer.
- Before feeding to the last fully connected layer, if the input has a shape `(batch_size, C, H, W)`, you should convert it to shape `(batch_size, C×H×W)` with `torch.tensor.reshape`.
- For all of the sub-components of the network, use the unmentioned parameters' default values (e.g., `stride=1, padding=0, dilation=1, groups=1, bias=True` for all convolutional layers).

Hint: If you are not familar with some of the library routines above, you could check them by searching their names in PyTorch Docs, e.g., `torch.nn.Conv2d`.

(a) In the description of the network, we only mentioned the number of output channels and features. But when you define these layers in PyTorch, you are required to give the number of input channels and features too.

In your **written submission**, derive all the required input channels and features. This would be helpful for your implementation.

Specifically, you should derive:

- Number of input channels of the two convolutional layers.
- Number of input features of the last fully connected layer.

(b) Implement the class `DigitsConvNet`.

IMPORTANT: Please read the following instructions carefully. To keep the parameter initialization consistent for autograding, you should:

- DO NOT remove or modify the code `torch.manual_seed(0)` in the `__init__` function.
- Define ALL of the sub-modules in the `__init__` function, under `torch.manual_seed(0)`.
  DO NOT define sub-modules anywhere else, e.g., in `forward`.
- Define ALL of the sub-modules **in the order that the input data goes through**.
  DO NOT change the order of sub-modules. For example, DO NOT define all convolutional layers together, since it may lead to another parameter initialization.
- Define ALL the sub-modules with an instance of `torch.nn.Module`, e.g., `torch.nn.Conv2d`.
  DO NOT use the functions in `torch.nn.functional` if you are not familar with them, since these operations may contain some parameter.
- DO NOT use CUDA even if your computer supports.
  GradeScope does not have GPU and your code won't run if you do so.

Otherwise, you may fail ALL the autograder tests.

Please refer to the dosctrings in hw3.py for details.

Library routines: `torch.nn.Conv2d, torch.nn.MaxPool2d, torch.nn.BatchNorm2d, torch.nn.Linear.`

(c) Implement `fit_and_evaluate` for use in the next several parts. The utility functions `train_batch` and `epoch_loss` will be useful. See the docstrings in `hw3.py` and `hw3_util.py` for details.

Library routines: `torch.no_grad`.

(d) Fit a `DigitsConvNet` on the train dataset from `torch_digits` in `hw3_util.py`.

Use `torch.nn.CrossEntropyLoss` as the loss function and `torch.optim.SGD` as the optimizer with learning rate 0.005 and no momentum. Train your model for 30 epochs with a batch size of 8.

Plot the epochs vs training and test loss. Include the plot in your **written submission**.

Library routines: `torch.optim.SGD`, `torch.nn.CrossEntropyLoss`, `plt.plot`, `plt.legend`, `torch.load`.

Solution.

Your solution here.