CS 446 / ECE 449 — Homework 2

your NetID here

Version 1.0

Instructions.

- Homework is due **Wednesday, September 29, at noon CST**; you have **3** late days in total for **all Homeworks**.
- Everyone must submit individually on Gradescope under `hw2` and `hw2code`.
- The "written" submission at `hw2` **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 `hw2`, 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. Briefly, you may have high-level discussions with at most 3 classmates, whose NetIDs you should place on the first page of your solutions, and you should cite any external reference you use; despite all this, your solution must be written in your own words.
- We reserve the right to reduce the auto-graded score for `hw2code` 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).
- The list of library routines with the coding problems are only suggestive.
- When submitting to `hw2code`, upload `hw2.py` and `hw2_utils.py`. The `CAFE Gamma` directory will be provided on the autograder.

Version History.

1. Initial version.

1. Support Vector Machines.

Recall that the dual problem of an 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 $\mathcal{C} = [0,\infty]^N = \{\boldsymbol{\alpha} : \alpha_i \geq 0\}$ for a hard-margin SVM and $\mathcal{C} = [0,C]^N = \{\boldsymbol{\alpha} : 0 \leq \alpha_i \leq C\}$ for a soft-margin SVM.

Equivalently, we can frame the dual problem of SVM as the minimization problem

$$\min_{\boldsymbol{\alpha}\in\mathcal{C}} f(\boldsymbol{\alpha}) := \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i\alpha_j y^{(i)} y^{(j)} \kappa(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) - \sum_{i=1}^{N} \alpha_i.$$

(a) 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

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

Hint: A sketch proof is briefly mentioned in Lecture 6 slides. The purpose of this problem is to ensure that you comprehend the Lagrangian and are familiar with the basic notations.

(b) Prove some theorems related to the margins of SVM, which connects different variables in SVM. In the questions below, we assume that the data are linearly separable and all the parameters subject to the same conditions as in the dual problem:

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

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. Prove the following equation under the same conditions of the previous question.

$$\frac{1}{\rho^2} = \sum_{i=1}^{n}\alpha_i$$

Hint: Use the conclusion from the previous question and $\|\boldsymbol{w}^*\|_2^2 = {\boldsymbol{w}^*}^\top\boldsymbol{w}^*$. Also recall what is support vectors and complementary slackness.

(c) Prove some conclusions about kernel methods.

i. For arbitrary 2D vectors $\boldsymbol{x} = [x_0, x_1]^T$ and $\boldsymbol{z} = [z_0, z_1]^T$, we define a kernel $\kappa(\boldsymbol{x},\boldsymbol{z}) = (1+\boldsymbol{x}^T\boldsymbol{z})^2$. Derive the equation of $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{z})$. To keep your answer consistent with the standard solution, please note that $\phi(\boldsymbol{x})$ and $\phi(\boldsymbol{z})$ are both vectors of monomials.

ii. Show that the dual objective for the RBF kernel SVM is given by:

$$h(\boldsymbol{\alpha}) = -\frac{1}{2}\boldsymbol{\alpha}^\top\boldsymbol{A}\boldsymbol{\alpha} + \mathbf{1}^\top\boldsymbol{\alpha},$$

where $\mathbf{1}$ is a vector of ones and $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ whose (i, j)-th entry is given by

$$A_{ij} = y^{(i)} y^{(j)} \exp\left(-\frac{\left\| \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)} \right\|_2^2}{2\sigma^2} \right).$$

Solution.

2. 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, **prove that**

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

Note: Include this question in the written submission.

Hint: Show that the i'th component of any $\boldsymbol{\alpha}' \in \mathcal{C}$ is further from the i'th component of $\boldsymbol{\alpha}$ than the i'th component of the projection is. Specifically, show that $\left|\alpha_i' - \alpha_i\right| \geq \left|\max\{0, \alpha_i\} - \alpha_i\right|$ for $\boldsymbol{\alpha}' \in [0,\infty)^n$ and that $\left|\alpha_i' - \alpha_i\right| \geq \left|\min\left\{\max\{0,\alpha_i\}, C\right\} - \alpha_i\right|$ for $\boldsymbol{\alpha}' \in [0,C]^n$.

(b) Implement an `svm_solver()`, using projected gradient descent formulated as above. Initialize your $\boldsymbol{\alpha}$ to zeros. See the docstrings in `hw2.py` for details.

Remark: Consider using 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.autograd.backward, torch.tensor(..., requires_grad=True), with torch.no_grad():, torch.tensor.grad.zero_, torch.tensor.detach.`

(c) Implement an `svm_predictor()`, using an optimal dual solution, the training set, and an input. See the docstrings in `hw2.py` for details.

(d) On the area $[-5,5] \times [-5,5]$, plot the contour lines of the following kernel SVMs, trained on the XOR data. Different kernels and the XOR data are provided in `hw2_utils.py`. Learning rate 0.1 and 10000 steps should be enough. To draw the contour lines, you can use `hw2_utils.svm_contour()`.

- The polynomial kernel with degree 2.
- The RBF kernel with $\sigma = 1$.
- The RBF kernel with $\sigma = 2$.
- The RBF kernel with $\sigma = 4$.

Include these four plots in your written submission.

Solution.

3. Neural Networks

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

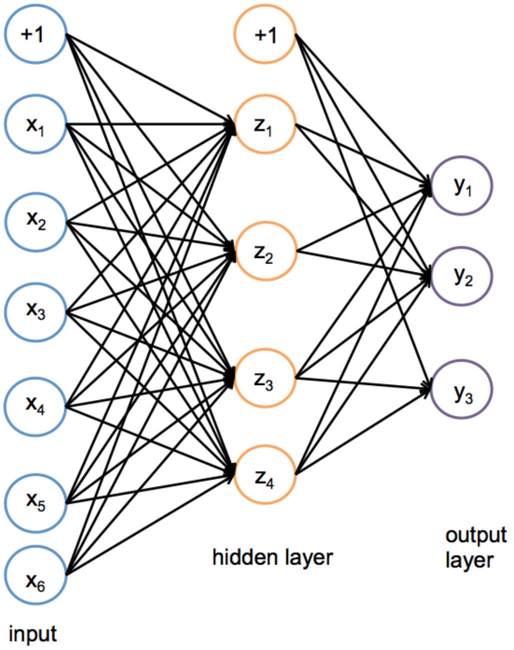

Figure 1: A One Hidden Layer Neural Network

Network Overview

Consider the neural network with one hidden layer shown in Figure 1. The input layer consists of 6 features $\mathbf{x} = [x_1, ..., x_6]^T$, the hidden layer has 4 nodes $\mathbf{z} = [z_1, ..., z_4]^T$, and the output layer is a probability distribution $\mathbf{y} = [y_1, y_2, y_3]^T$ 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:

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5, x_6]^T \tag{1}$$

Linear combination at the first (hidden) layer:

$$a_j = \alpha_{j,0} + \sum_{i=1}^{6} \alpha_{j,i} x_i, \quad j \in \{1, \dots, 4\} \tag{2}$$

Activation at the first (hidden) layer:

$$z_j = \sigma(a_j) = \frac{1}{1 + \exp(-a_j)}, \quad j \in \{1, \dots, 4\} \tag{3}$$

Linear combination at the second (output) layer:

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

Activation at the second (output) layer:

$$\hat{y}_k = \frac{\exp(b_k)}{\sum\limits_{l=1}^{3} \exp(b_l)}, \quad k \in \{1, \dots, 3\} \tag{5}$$

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:

$$\ell(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_{k=1}^{3} y_k \log(\hat{y}_k) \tag{6}$$

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** $\mathbf{y}$ 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?

(c) Let us now introduce regularization into our neural network. For this question, we will incorporate L2 regularization into our loss function $\ell(\hat{\mathbf{y}}, \mathbf{y})$, with the parameter λ controlling the weight given to the regularization term.

i. Write the expression for the regularized loss function of our network after adding L2 regularization (**Hint:** Remember that bias terms should not be regularized!)

ii. Compute the regularized loss for training example $\mathbf{x}^{(1)}$ (assume $\lambda = 0.01$ and use the weights before backpropagation)

iii. For a network which uses the regularized loss function, write the gradient update equation for $\alpha_{j,i}$. You may use $\frac{\partial \ell(\hat{\mathbf{y}}, \mathbf{y})}{\partial \alpha_{j,i}}$ to denote the gradient update w.r.t non-regularized loss and η to denote the learning rate.

iv. Based on your observations from previous questions, **select all statements which are true**: (**Hint:** Use “$\blacksquare$” (■) to indicate your choice(s).)

□ The non-regularized loss is always higher than the regularized loss

□ As weights become larger, the regularized loss increases faster than non-regularized loss

□ On adding regularization to the loss function, gradient updates for the network become larger

□ When using large initial weights, weight values decrease more rapidly for a network which uses regularized loss

□ None of the above

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. The layers of your neural network should be:

- A 2D convolutional layer with 1 input channel and 8 output channels, with a kernel size of 3
- A 2D maximimum pooling layer, with kernel size 2
- A 2D convolutional layer with 8 input channels and 4 output channels, with a kernel size of 3
- A fully connected (torch.nn.Linear) layer with 4 inputs and 10 outputs

Apply the ReLU activation function to the output of each of your convolutional layers before inputting them to your next layer. For both of the convolutional layers of the network, use the default settings parameters (stride=1, padding=0, dilation=1, groups=1, bias=True).

(a) Implement the class `DigitsConvNet`. Please refer to the dosctrings in hw2.py for details.

Library routines: `torch.nn.Conv2d, torch.nn.MaxPool2D, and torch.nn.Linear.`

(b) 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 hw2.py and hw2_util.py for details.

Library routines: `torch.no_grad.`

(c) Fit a `DigitsConvNet` on the train dataset from `torch_digits` in hw2_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 1. Keep track of your training and test loss for part (e).

Library routines: `torch.optim.SGD, torch.nn.CrossEntropyLoss, torch.save.`

(d) Fit another `DigitsConvNet` with `torch.nn.CrossEntropyLoss` and `torch.optim.SGD` with learning rate 0.005, no momentum, and a batch size of 1 for 30 epochs. This time we will adjust the learning rate so that it decreases at each epoch. Recall the gradient descent update step

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \eta_i \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_i).$$

where $\mathcal{L}$ is the loss function and i is the step index. We will update the learning rate only at the end of each epoch. Therefore, $\eta_{i+1} := \eta_i$ within each epoch and $\eta_{i+1} = \gamma \cdot \eta_i$ at the end of each epoch. You should use `torch.optim.lr_scheduler.ExponentialLR`. Use a decay rate of $\gamma = 0.95$ and start the learning rate at 0.005. You may find it useful to temporarily modify your `fit_and_evaluate` function to include a scheduler for this part. Keep track of your training and test loss for part (e).

Library routines: `torch.optim.lr_scheduler.ExponentialLR, torch.nn.CrossEntropyLoss, torch.save.`

(e) Fit a third `DigitsConvNet`, again with `torch.nn.CrossEntropyLoss` and `torch.optim.SGD` with learning rate 0.005 and no momentum for 30 epochs. However, this time use a batch size of 16.

Plot the epochs vs training and test losses for parts (c), (d), and (e) (you should have six plots on the same figure, include this figure in your written submission). For plot legend, please name the two plots in (c) as ”train batch = 1”, ”test batch = 1”, (d) as ”train decayed_lr = 1”, ”test decayed_lr = 1”, (e) as ”train batch = 16”, ”test batch = 16”.

You do **not need to include the plot generation code in your code submission.**

Include the figure and your assessment of the impact the exponentially decayed learning rate has on training speed (the behavior of the loss, not CPU time). Additionally, comment on the impact that increasing the batch size has on loss.

In your report, you may leave parts (c) and (d) blank and include all comments in part (e). Note that it is normal to get different plots for separate runs as PyTorch randomly initializes the weights. We added ‘torch.manual_seed(0)’ in the coding template to make everyone’s plot look similar.

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

Solution.