CS 446 / ECE 449 — Homework 2

your NetID here

Version 1.2

Instructions.

- Homework is due **Tuesday, March 2, at noon CST**; no late homework accepted.
- 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.
- The list of library routines with the coding problems are only suggestive.
- 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).
- 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.

1.1. SVM implementation wording tweaked.

1.2. Initialize $\boldsymbol{\alpha}$ to zeros in 1b; clarified wording of 4e.

1. SVM Implementation.

Recall that the dual problem of an SVM is

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

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 this as the minimization problem

$$\min_{\boldsymbol{\alpha}\in\mathcal{C}} f(\boldsymbol{\alpha}) := \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i\alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{n} \alpha_i.$$

This can be solved by projected gradient descent, which starts from some $\boldsymbol{\alpha}_0 \in \mathcal{C}$ (e.g., $\mathbf{0}$) and updates via

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

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

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

If $\mathcal{C}$ is convex, the projection is uniquely defined.

(a) Prove that

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

and

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

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

Library routines: `torch.empty.`

(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.

2. RBF kernel and nearest neighbors.

(a) Recall that given data examples $((\boldsymbol{x}_i, y_i))_{i=1}^n$ and an optimal dual solution $(\hat{\alpha}_i)_{i=1}^n$, the RBF kernel SVM makes a prediction as follows:

$$f_\sigma(\boldsymbol{x}) = \sum_{i=1}^n \hat{\alpha}_i y_i \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2}{2\sigma^2}\right) = \sum_{i \in S} \hat{\alpha}_i y_i \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|_2^2}{2\sigma^2}\right),$$

where $S \subset \{1, 2, \ldots, n\}$ is the set of indices of support vectors.

Given an input $\boldsymbol{x}$, let $T := \arg\min_{i \in S} \|\boldsymbol{x} - \boldsymbol{x}_i\|_2$ denote the set of closest support vectors to $\boldsymbol{x}$, and let $\rho := \min_{i \in S} \|\boldsymbol{x} - \boldsymbol{x}_i\|_2$ denote this smallest distance. (In other words, $T := \{i \in S : \|\boldsymbol{x} - \boldsymbol{x}_i\|_2 = \rho\}$.) Prove that

$$\lim_{\sigma \to 0} \frac{f_\sigma(\boldsymbol{x})}{\exp\left(-\rho^2/2\sigma^2\right)} = \sum_{i \in T} \hat{\alpha}_i y_i.$$

Hint: Split up the sum over elements of S into two sums: one over $i \in T$ and one over $i \in S \setminus T$.
Remark: In other words, when the bandwidth σ becomes small enough, the RBF kernel SVM is almost the 1-nearest neighbor predictor with the set of support vectors as the training set. Note that while nearest neighbors will not be introduced until Lecture 11, solving this problem does not require knowledge of them.
Remark 2: The prediction function, $f_\sigma(\boldsymbol{x}) : \boldsymbol{x} \to \phi(\boldsymbol{x})^\top \overline{\mathbf{w}}$, depends only on the labels of the closest support vectors of $\boldsymbol{x}$, i.e., the constituents of the set T.

(b) 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

$$A_{ij} = y_i y_j \exp\left(-\frac{\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2}{2\sigma^2}\right).$$

(c) Show that an optimal dual solution $\hat{\boldsymbol{\alpha}}$, satisfies the equation $\boldsymbol{A}\hat{\boldsymbol{\alpha}} = \mathbf{1}$.
Hint: A function's gradient is zero at an extremum.

Solution.

3. Neural Networks for Emotion Classification

In this problem you will build a single-layer neural network that classifies pictures into one of six categories: anger, disgusted, happy, maudlin, fear, and surprise. The CAFE [1] dataset included in your directory provides a set of grayscale facial images expressing the described emotions. This will also serve as an introduction to writing your own neural networks in PyTorch! Consider the single layer neural network below

$$\boldsymbol{x} \mapsto \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}),$$

where σ is the softmax activation and we use cross entropy loss to train the network.

(a) Implement your network in the class CAFENet. You will need to modify the `__init__` and `forward` methods. Due to numerical issues, we won't explicitly add a softmax layer to our network. Instead, output the raw logits at this step and use `torch.nn.CrossEntropyLoss` in part (b), which performs the softmax and cross entropy calculation in log-space. Refer to `IMAGE_DIMS`, `load_cafe`, and `get_cafe_data` in `hw2_utils.py`, using `get_cafe_data` to obtain the images and labels as tensors.

Library routines: `torch.nn.Linear, torch.nn.Module.forward.`

(b) Implement `fit` to train the input network for `n_epochs` epochs. Use cross entropy loss and an Adam optimizer.

Library routines: `torch.nn.Module.forward, torch.nn._Loss.backward, torch.optim.Adam, torch.optim.Optimizer.step, torch.optim.Optimizer.zero_grad, torch.nn.CrossEntropyLoss.`

(c) Implement the `plot_cafe_loss` function. Using your `fit` function, train a CAFENet on the CAFE dataset for 5000 epochs. Use `hw_2_utils.get_cafe_data()` to load the dataset. Plot the initial loss and the losses after the first 200 epochs and include this plot in your writeup. Save the model for use in the next part.

Library routines: `plt.plot, torch.save.`

(d) Now let's visualize the model's weights by implementing the `visualize_weights` method. For each of the 91,200-dimensional weights of your CAFENet's six output nodes, linearly map them to the grayscale range [0, 255] by performing the following transformation

i. Compute the minimum and maximum weights across all six output nodes, denoted `min_weight, max_weight` respectively.

ii. Transform the weights `w` by `w = (w - min_weight) * 255 / (max_weight - min_weight)` to linearly map `w` into the range [0, 255].

iii. Cast the weights to integers.

Then, reshape the weights to the image dimensions `380 x 240` and plot them in grayscale. Include all six plots in your writeup. What do you see? Why might the weights appear this way?

Library routines: `torch.load, torch.nn.Module.parameters, torch.nn.tensor.min, torch.nn.tensor.max, torch.nn.tensor.int, torch.nn.tensor.reshape, torch.tensor.detach, plt.imshow(..., cmap='gray').`

Note: In practice, for simple neural networks like this we would use `torch.nn.Sequential`.

Solution.

[1] Inspiration for this problem from Garrison Cottrell's neural networks course. See Dailey et al. (2001) for more info on the CAFE dataset.

4. 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, $\eta_{i+1} \coloneqq \eta_i$ 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 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 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. However, comments can be made on the general behavior of each of these losses.
Library routines: `torch.optim.SGD, torch.nn.CrossEntropyLoss, plt.plot, plt.legend, torch.load.`

Solution.

5. On Initialization.

Consider a 2-layer network

$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{v}) = \sum_{j=1}^{m} v_j \sigma\left(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle\right),$$

where $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{W} \in \mathbb{R}^{m \times d}$ with rows $\boldsymbol{w}_j^\top$, and $\boldsymbol{v} \in \mathbb{R}^m$. For simplicity, the network has a single output, and bias terms are omitted.

Given a data example $(\boldsymbol{x}, y)$ and a loss function ℓ, consider the empirical risk

$$\widehat{\mathcal{R}}(\boldsymbol{W}, \boldsymbol{v}) = \ell\left(f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{v}), y\right).$$

Only a single data example will be considered in this problem; the same analysis extends to multiple examples by taking averages.

(a) For each $1 \le j \le m$, derive $\partial \widehat{\mathcal{R}} / \partial v_j$ and $\partial \widehat{\mathcal{R}} / \partial \boldsymbol{w}_j$. Note that the first is a derivative with respect to a scalar (so the answer should be a scalar), and the second is a derivative with respect to a vector (so the answer should be a vector).

(b) Consider gradient descent which starts from some $\boldsymbol{W}^{(0)}$ and $\boldsymbol{v}^{(0)}$, and at step $t \ge 0$, updates the weights for each $1 \le j \le m$ as follows:

$$\boldsymbol{w}_j^{(t+1)} = \boldsymbol{w}_j^{(t)} - \eta \frac{\partial \widehat{\mathcal{R}}}{\partial \boldsymbol{w}_j^{(t)}}, \qquad \text{and} \qquad v_j^{(t+1)} = v_j^{(t)} - \eta \frac{\partial \widehat{\mathcal{R}}}{\partial v_j^{(t)}}.$$

Suppose there exist two hidden units $p, q \in \{1, 2, \ldots, m\}$ such that $\boldsymbol{w}_p^{(0)} = \boldsymbol{w}_q^{(0)}$ and $v_p^{(0)} = v_q^{(0)}$. Prove by induction that for any step $t \ge 0$, it holds that $\boldsymbol{w}_p^{(t)} = \boldsymbol{w}_q^{(t)}$ and $v_p^{(t)} = v_q^{(t)}$.

Remark: As a result, if the neural network is initialized symmetrically, then such a symmetry may persist during gradient descent, and thus the representation power of the network will be limited.

(c) Random initialization is a good way to break symmetry. Moreover, proper random initialization also preserves the squared norm of the input, as formalized below.

Consider the identity activation $\sigma(z) = z$. For each $1 \le j \le m$ and $1 \le k \le d$, initialize $w_{j,k}^{(0)} \sim \mathcal{N}(0, 1/m)$ (i.e., normal distribution with mean 0 and variance $1/m$). Prove that

$$\mathbb{E}\left[\left\|\boldsymbol{W}^{(0)} \boldsymbol{x}\right\|_2^2\right] = \|\boldsymbol{x}\|_2^2.$$

Hint: Note that $\left\|\boldsymbol{W}^{(0)} \boldsymbol{x}\right\|_2^2$ can be written as a summation of $\boldsymbol{w}_j^\top x$, then use linearity of expectation. It may be helpful to recall that for independent random variables X, Y $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ and that $\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

Remark: A similar property holds with the ReLU activation.

Solution.

References

Matthew N. Dailey, Garrison W. Cottrell, and Judith Reilly. CAlifornia Facial Expressions (CAFE), 2001. URL `http://www.cs.ucsd.edu/users/gary/CAFE/`.