CS 446 / ECE 449 — Homework 2

ivette2

Version 1.0

Instructions.

- Homework is due **Tuesday, February 22, 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 select pages for each problem; 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).
- Coding problems come with suggested "library routines"; we include these to reduce your time fishing around APIs, but you are free to use other APIs.
- When submitting to `hw2code`, only upload the two python files `hw2.py` and `hw2_utils.py`. Don't upload a zip file or additional files.

Version history.

1.0. Initial version.

1. SVM with Biases.

This problem is about SVMs over $\mathbb{R}^d$ with linearly separable data (i.e., the hard margin SVM).

Our formulation of SVM required separators to pass through the origin, which does not provide a geometrically pleasing notion of maximum margin direction.

A first fix is provided by lecture 4: by appending a 1 to the inputs, we obtain the convex program

$$\begin{aligned} \min_{\boldsymbol{u}} \quad & \frac{1}{2}\|\boldsymbol{u}\|^2 \\ \text{subject to} \quad & \boldsymbol{u} \in \mathbb{R}^{d+1} \\ & y_i \begin{bmatrix} \boldsymbol{x}_i \\ 1 \end{bmatrix}^\intercal \boldsymbol{u} \geq 1 \qquad \forall i, \end{aligned}$$

and let $\bar{\boldsymbol{u}}$ denote the optimal solution to this program.

A second standard fix is to incorporate the bias directly into the optimization problem:

$$\begin{aligned} \min_{\boldsymbol{v},b} \quad & \frac{1}{2}\|\boldsymbol{v}\|^2 \\ \text{subject to} \quad & \boldsymbol{v} \in \mathbb{R}^d, b \in \mathbb{R} \\ & y_i(\boldsymbol{v}^\intercal \boldsymbol{x}_i + b) \geq 1 \qquad \forall i, \end{aligned}$$

and let $(\bar{\boldsymbol{v}}, \bar{b}) \in \mathbb{R}^d \times \mathbb{R}$ denote an optimal solution to this program. This second version is standard, but we do not use it in lecture for various reasons.

(a) In lecture, we stated that the first formulation is a *convex program* (formally defined in lecture 5). Show that the second formulation is also a convex program.

To be considered a convex program, we must show that the second formulation is minimizing a convex function over a convex set. $f : \mathbb{R}^d \to \mathbb{R}$ is convex if for any $x, x' \in \mathbb{R}^d$ and $\alpha \in [0,1]$, $f\left((1-\alpha)\,x + \alpha x'\right) \leq (1-\alpha)\,f\left(x\right) + \alpha f\left(x'\right)$. For the second formulation, we are minimizing $\frac{1}{2}\left\|v^2\right\|$, which gives us the inequality $\frac{1}{2}\left\|\left((1-\alpha)\,v + \alpha v'\right)^2\right\| \leq (1-\alpha)\,\frac{1}{2}\left\|v\right\|^2 + \alpha\frac{1}{2}\left\|v\right\|^2$. Given the triangle equality for real norms, $\|x+y\| \leq \|x\| + \|y\|$, and the fact that $\alpha \in [0,1]$, we can say that $\left\|\left((1-\alpha)\,v + \alpha v'\right)\right\| \leq (1-\alpha)\left\|v\right\| + \alpha\left\|v\right\|$ and $\left\|\left((1-\alpha)\,v + \alpha v'\right)\right\|^2 \leq \left((1-\alpha)\left\|v\right\| + \alpha\left\|v\right\|\right)^2$. This becomes $\left\|\left((1-\alpha)\,v + \alpha v'\right)\right\|^2 \leq \left((1-\alpha)^2\left\|v\right\|^2 + 2\left(\alpha - \alpha^2\right)\left\|v\right\| + \alpha^2\left\|v\right\|^2\right)$, which proves the initial inequality and the function to be convex.

To prove the set convex, we can use the definition: For every pair of points $\left\{x, x'\right\} \in S$, $\left[x, x'\right] = \left\{\alpha x + (1-\alpha)\,x' : \alpha \in [0,1]\right\} \subseteq S$. We can use the subjected condition and prove that
$y_i\left(\left(\alpha v + (1-\alpha)v'\right)^T x_i + (\alpha b + (1-\alpha))b'\right) \geq 1$ has $(v,b) \subseteq S$. This expands out to
$y_i\left(\alpha v^T x_i + (1-\alpha)\left(v'\right)^T x_i + \alpha b + (1-\alpha)b'\right)$, where we can look at each term and say $\alpha v^T x_i \geq 0$, $(1-\alpha)\left(v'\right)^T x_i \geq (v')^T x_i$, $\alpha b \geq 0$ and $(1-\alpha)b' \geq b'$, all given that $\alpha \in [0,1]$. This proves that
$y_i\left(\alpha v^T x_i + (1-\alpha)\left(v'\right)^T x_i + \alpha b + (1-\alpha)b'\right) \geq y_i\left((v')^T x_i + b'\right) \geq 1$, and we have a convex set.

(b) Suppose there is only one datapoint: $\boldsymbol{x}_1 = \boldsymbol{e}_1$, the first standard basis vector, with label $y_1 = +1$. The first formulation will have a unique solution $\bar{\boldsymbol{u}}$, as discussed in lecture. Show that the second formulation does not have a unique solution.

If we have only one data point with $x_1 = e_1, y_1 = +1$, we are subject to condition $y_i \left(v^T x_i + b\right) \longrightarrow v_1 + b \geq 1$. With this, $\min_{v,b} \frac{1}{2} \left\|v^2\right\|$ will be minimized for any $v = 0$ and $b \geq 1$, resulting in infinite solutions.

(c) Let's add another datapoint: $\boldsymbol{x}_2 = -a\boldsymbol{e}_1$ for some $a \geq 3$, with label $y_2 = -1$. Now that we have two data points, both of the convex programs now have two constraints. Write out the explicit constraints to the first convex program.

Adding another data point, $x_2 = -ae_1, a \geq 3, y_1 = -1$. The constraints for the first program are $u \in \mathbb{R}^{d+1}, y_i \begin{bmatrix} x_i \\ 1 \end{bmatrix}^T u \geq 1$. Writing these out for the two data points, we have $1 \begin{bmatrix} e_1 \\ 1 \end{bmatrix}^T u \geq 1$ and $-1 \begin{bmatrix} -ae_1 \\ 1 \end{bmatrix}^T u \geq 1$. Knowing that e_1 is a standard basis vector, we can explicitly write these constraints as $u_1 + u_{d+1} \geq 1$ and $au_1 - u_{d+1} \geq 1$, respectively.

(d) Using these two constraints, show that the first coordinate $\bar{u}_1$ of the optimal solution $\bar{\boldsymbol{u}}$ satisfies $\bar{u}_1 \geq \frac{2}{a+1}$.

If we simply combine the two constraint inequalities on an optimal solution $\bar{u}_1$, we have $u_1 + au_1 \geq 2$. This is the same as saying $u_1(1 + a) \geq 2$, and thus $u_1 \geq \frac{2}{a+1}$.

(e) Using parts (c) and (d), find optimal solutions $\bar{\boldsymbol{u}}$ and $(\bar{\boldsymbol{v}}, \bar{b})$, and prove they are in fact optimal.
Hint: If you are stuck, first try the case $d = 1$. Then study what happens for $d = 2, d = 3, \ldots$
Hint: $(\bar{\boldsymbol{v}}, \bar{b})$ will be unique.

The optimal solutions $\bar{\boldsymbol{u}}$ and $(\bar{\boldsymbol{v}}, \bar{b})$ can be determined using the constraints and the minimization problems themselves, $\min_{\boldsymbol{u}} \quad \frac{1}{2}\|\boldsymbol{u}\|^2$ and $\min_{\boldsymbol{v},b} \quad \frac{1}{2}\|\boldsymbol{v}\|^2$. For $\bar{\boldsymbol{u}}$, the norm is minimized with the smallest possible components $\bar{\boldsymbol{u}}_i$, so $\bar{\boldsymbol{u}} = \left(\frac{2}{a+1}, 0, ..., \frac{a-1}{a+1}\right)$, as these are the smallest components that still satisfy the constraints.

For $(\bar{\boldsymbol{v}}, \bar{b})$, we can take the same approach. The components that minimize the norm of v go as $(\bar{\boldsymbol{v}}) = \left(\frac{1}{1+a}, 0, ...\right)$, and $(\bar{b}) = \left(1 - \frac{a}{1+a}\right)$.

(f) Now we will consider the behavior of $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ as a increases; to this end, write $\bar{\boldsymbol{u}}_a$ and $\bar{\boldsymbol{v}}_a$, and consider $a \to \infty$. Determine and formally prove the limiting behavior of $\lim_{a\to\infty} \frac{1}{2}\|\bar{\boldsymbol{u}}_a\|^2$ and $\lim_{a\to\infty} \frac{1}{2}\|\bar{\boldsymbol{v}}_a\|^2$.
Hint: The two limits will not be equal.

$\lim_{a\to\infty} \frac{1}{2}\|\bar{\boldsymbol{u}}_a\|^2 = (0, 0, ..., 1)$, and
$\lim_{a\to\infty} \frac{1}{2}\|\bar{\boldsymbol{v}}_a\|^2 = \big(0, 0, ...\big)\big)$.

(g) Between the two versions of SVM with bias, which do you prefer? Any answer which contains at least one complete sentence will receive full credit.
Remark: Initially it may have seemed that both optimization problems have the same solutions; the purpose of this problem was to highlight that small differences in machine learning methods can lead

to observably different performance.

I prefer the second convex program, obtained by incorporating the bias into the optimization problem. I thought the notion of infinite solutions for insufficient data (like the one datapoint proposed in 1b) felt more intuitive than the first formulation with appended ones.

Solution.

2. 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 ith component of any other $\boldsymbol{\alpha}' \in \mathcal{C}$ is further from the ith component of $\boldsymbol{\alpha}$ than the ith 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 $\boldsymbol{\alpha}' \in [0, C]^n$.

Looking at $\left|\alpha_i' - \alpha_i\right| \geq \left|\max\{0, \alpha_i\} - \alpha_i\right|$ for $\boldsymbol{\alpha}' \in [0,\infty)^n$, we know that $\alpha_i' \geq 0$ for any $\alpha_i \geq 0$ We can say that $\left(\Pi_{[0,\infty)^n}[\boldsymbol{\alpha}]\right)_i = \left\{ \begin{array}{ccc} \alpha_i & : & \alpha_i \geq 0 \\ 0 & : & \alpha_i < 0 \end{array} \right\} = \max\{0, \alpha_i\}$
We can say that $\left|\alpha_i' - \alpha_i\right| \geq \left|\max\{0, \alpha_i\} - \alpha_i\right|$ for $\boldsymbol{\alpha}' \in [0,\infty)^n$, because $\alpha_i' \geq 0$.
The same argument goes for $\left|\alpha_i' - \alpha_i\right| \geq \left|\min\left\{\max\{0, \alpha_i\}, C\right\} - \alpha_i\right|$ 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.`

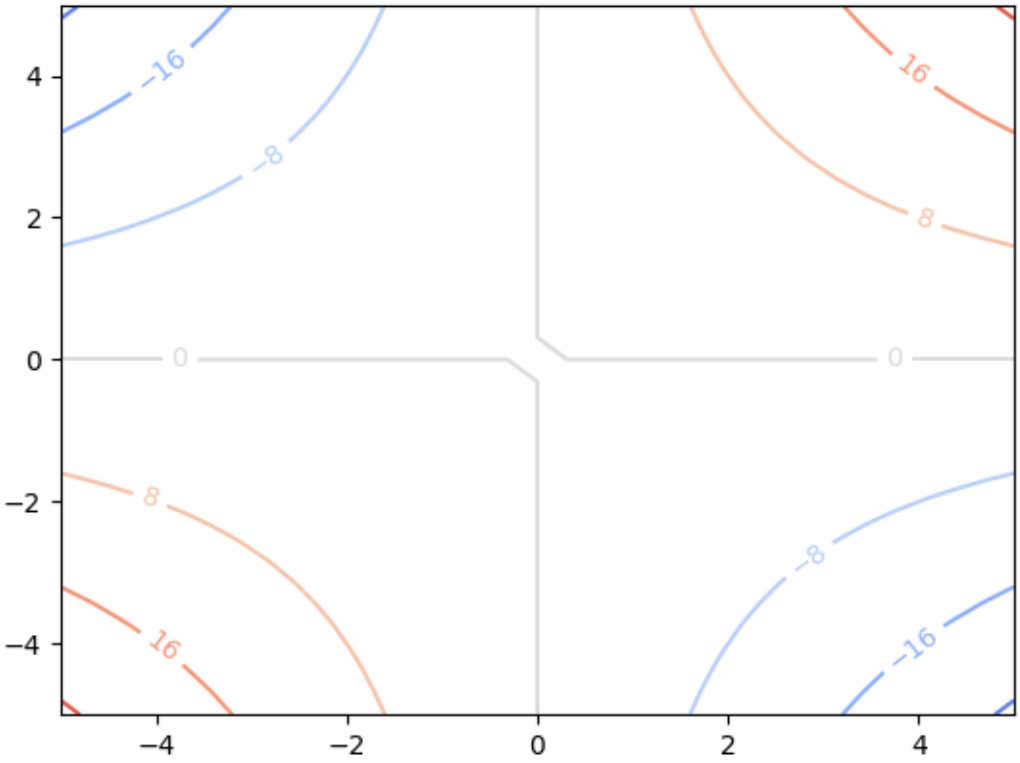
(c) Implement an `svm_predictor()`, using an optimal dual solution, the training set, and an input. See the docstrings in `hw2.py` for details.
Library routines: `torch.empty.`

(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()`.

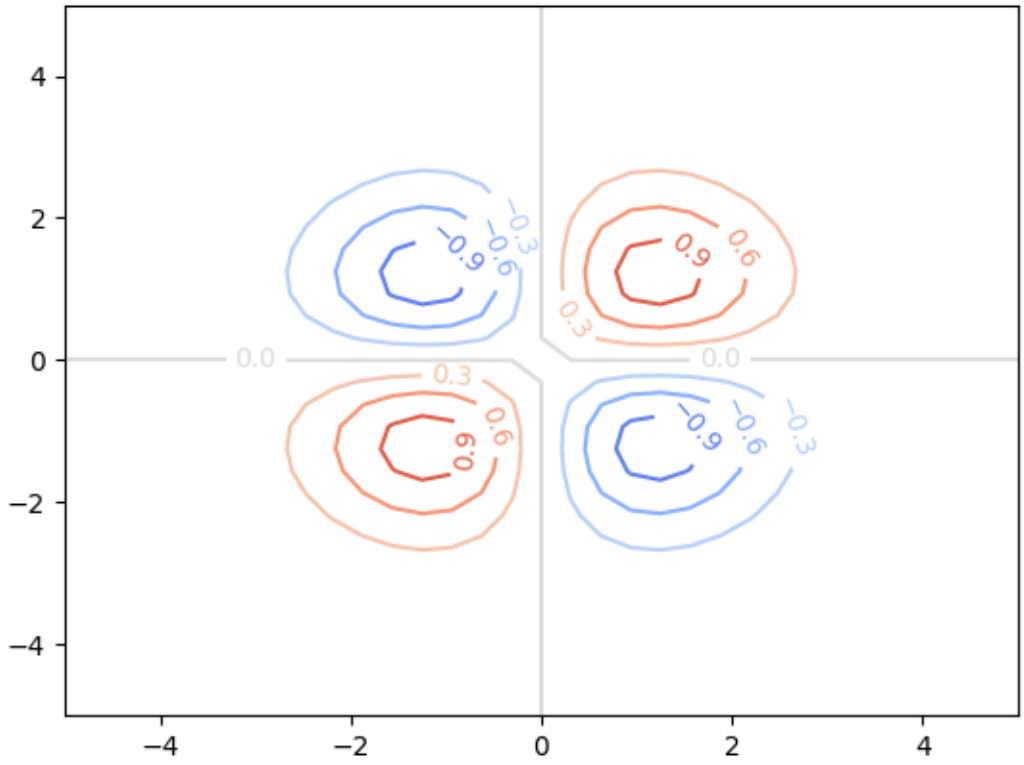
- The polynomial kernel with degree 2.
- The RBF kernel with $\sigma = 1$.
- The RBF kernel with $\sigma = 2$.
- The RBF kernel with $\sigma = 4$.

Include these four plots in your written submission.

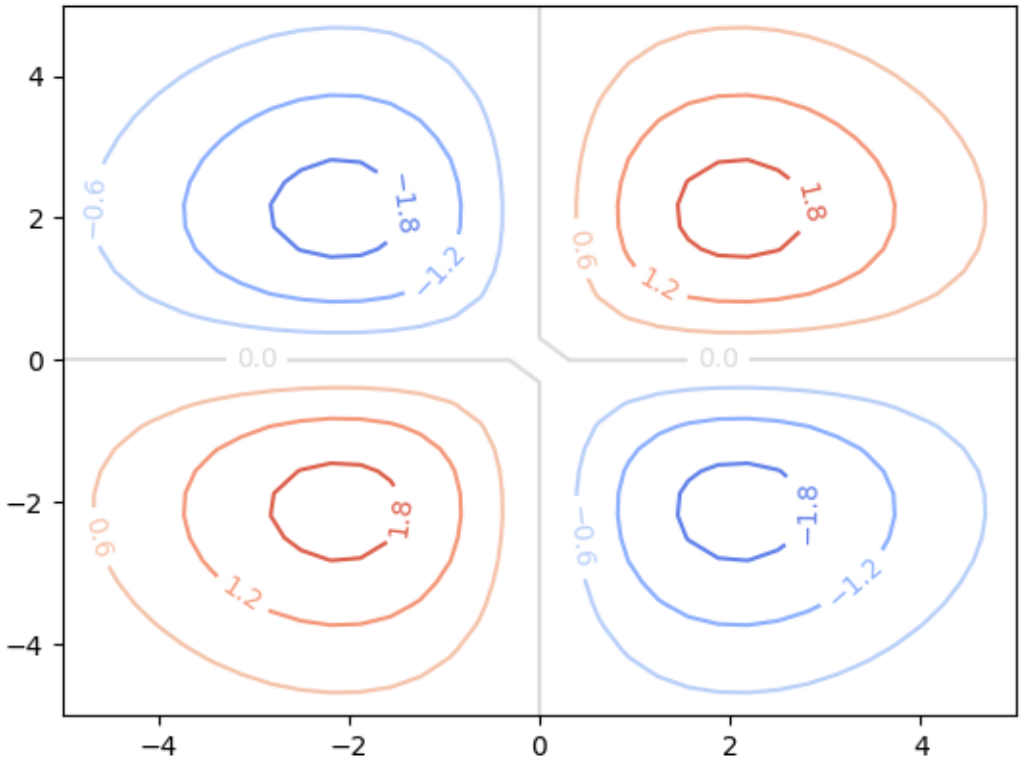
Polynomial kernel with degree 2



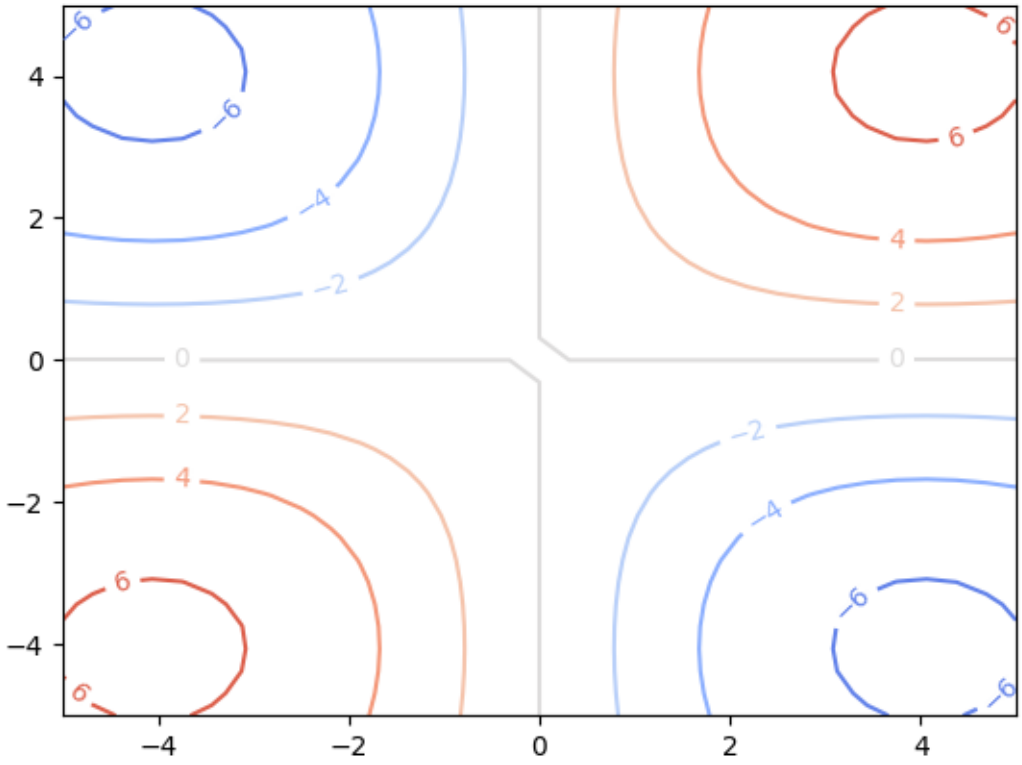
RBF kernel with $\sigma = 1$



RBF kernel with $\sigma = 2$



RBF kernel with $\sigma = 4$



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 this homework's zip file 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, do not include an explicit softmax layer in your network. Instead, your implementation should output the raw logits (meaning $\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$); in part (b), the network will be fit to data with `torch.nn.CrossEntropyLoss`, which implicitly applies a softmax as discussed in lecture. Refer to `IMAGE_DIMS`, `load_cafe`, and `get_cafe_data` in `hw2_utils.py` for how the images and labels will be passed to the network as tensors.

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

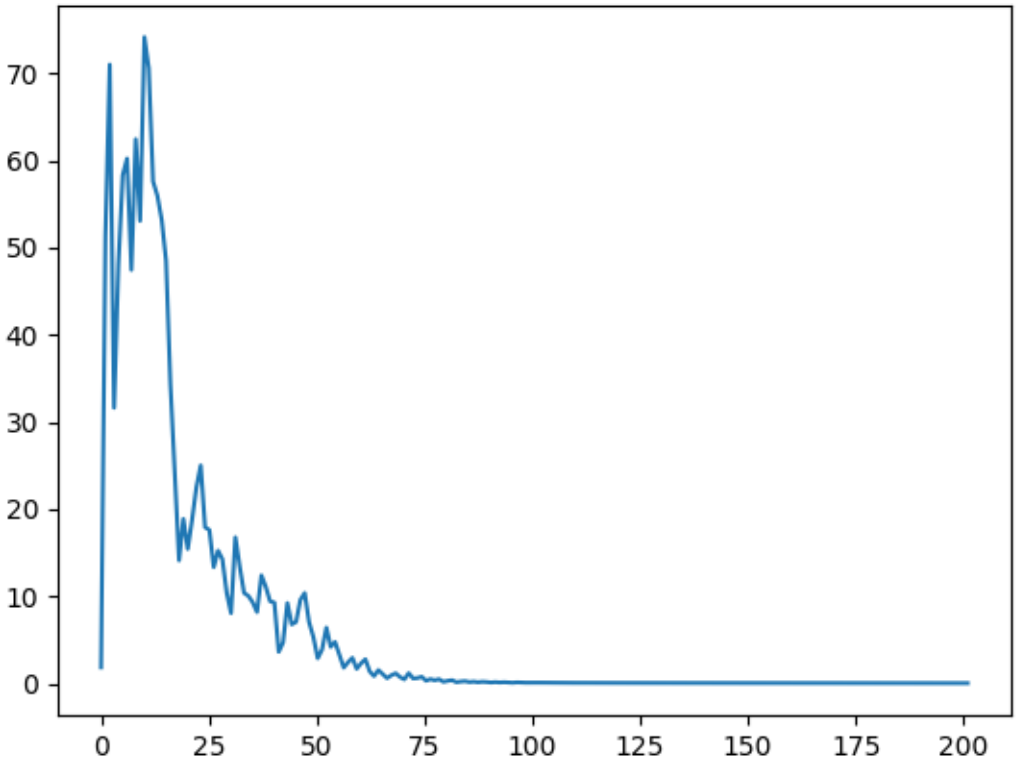
(b) Implement `fit` to train the input network for `n_epochs` epochs. Use cross entropy loss and an Adam optimizer.

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 and run the `plot_cafe_loss` function. Specifically, use `hw2_utils.get_cafe_data()` to load the training set, then train a CAFENet via your `fit` function for 201 epochs. Plot the empirical risk (in terms of cross entropy loss) across these first 201 epochs, and include the resulting plot in your written handin. Lastly, use `torch.save` to save your model in order to use it in the next two problem parts.

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

[1] Inspiration for this problem from Garrison Cottrell's neural networks course. See **?** for more info on the CAFE dataset.



(d) Let's see how well our model predicts labels. We will use a confusion matrix to visualize how well it does for each category. Implement `print_confusion_matrix` to print out two confusion matrices for your model, one on the training set and one on the test set. Use `hw2_utils.get_cafe_data("test")` to load the test set. Include both matrices in your writeup, along with 1-3 sentences discussing differences in the matrices and what might cause them.
Library routines: `torch.load, torch.argmax, sklearn.metrics.confusion_matrix.`

$$
\begin{bmatrix}
\begin{bmatrix} 6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 \end{bmatrix} \\
\begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 1 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{bmatrix}
\end{bmatrix}
$$

We can see that the first confusion matrix on the training set has all positives, as each diagonal entry $C_{i,i}$ is fully confident in the 6 classes. This makes sense given that the model was trained on the training data, so it performs well on predictions it has already seen. The second matrix for the test set represents significantly less confidence, as expected with data it hasn't seen before.

(e) 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 transformations:

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.

4. Shallow Network Random 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).

$\widehat{\mathcal{R}}(\boldsymbol{W}, \boldsymbol{v}) = \ell\left(f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{v}), y\right) = \ell\left(\sum_{j=1}^m v_j \sigma(\langle \omega_j, x \rangle), y\right)$, and we can differentiate using chain rule and linearity. $\partial \widehat{\mathcal{R}} / \partial v_j = \left(\partial \ell / \partial f\right)\left(\partial / \partial v_j\right) = \left(\partial \ell / \partial f\right)\left(\sum_{j=1}^m \sigma(\langle \omega_j, x \rangle)\right)$
Similarly, and assuming σ is an arbitrary activation function,
$\partial \widehat{\mathcal{R}} / \partial \boldsymbol{w}_j = \left(\partial \ell / \partial f\right)\left(\partial f / \partial \sigma\right)\left(\partial \sigma / \partial \langle \omega_j, x \rangle\right)\left(\partial \langle \omega_j, x \rangle\right) / \partial \omega_j\right) = \left(\partial \ell / \partial f\right)\left(\partial \sigma / \partial \langle \omega_j, x \rangle\right)\right)\left(\sum_{j=1}^m v_j\right)\left(x^T\right)$

(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 $p, q \in \{1, 2, \ldots, m\}$ and t such that $\boldsymbol{w}_p^{(t)} = \boldsymbol{w}_q^{(t)}$ and $v_p^{(t)} = v_q^{(t)}$. Show that $\boldsymbol{w}_p^{(t+1)} = \boldsymbol{w}_q^{(t+1)}$ and $v_p^{(t+1)} = v_q^{(t+1)}$.

$\boldsymbol{w}_p^{(t+1)} = \boldsymbol{w}_p^{(t)} - \eta \frac{\partial \widehat{\mathcal{R}}}{\partial \boldsymbol{w}_p^{(t)}} = \boldsymbol{w}_p^{(t)} - \eta\left(\left(\partial \ell / \partial f\right)\left(\partial \sigma / \partial \langle \omega_p, x \rangle\right)\right)\left(\sum_{j=1}^m v_j\right)\left(x^T\right)\right)$
$\boldsymbol{w}_q^{(t+1)} = \boldsymbol{w}_q^{(t)} - \eta \frac{\partial \widehat{\mathcal{R}}}{\partial \boldsymbol{w}_q^{(t)}} = \boldsymbol{w}_q^{(t)} - \eta\left(\left(\partial \ell / \partial f\right)\left(\partial \sigma / \partial \langle \omega_q, x \rangle\right)\right)\left(\sum_{j=1}^m v_j\right)\left(x^T\right)\right)$

The updates $\boldsymbol{w}_p^{(t+1)}$ and $\boldsymbol{w}_q^{(t+1)}$ will be equal so long that $\boldsymbol{w}_p^{(t)} = \boldsymbol{w}_q^{(t)}$

$v_p^{(t+1)} = v_p^{(t)} - \eta \frac{\partial \widehat{\mathcal{R}}}{\partial v_p^{(t)}} = v_p^{(t)} - \eta\left(\left(\partial \ell / \partial f\right)\left(\sum_{j=1}^m \sigma(\langle \omega_j, x \rangle)\right)\right) = v_q^{(t+1)}$

Similarly, v_p and v_q will follow the same update as long as $v_p^{(t)} = v_q^{(t)}$.

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

For the base case of $t = 0$, we're given that $\boldsymbol{w}_p^{(0)} = \boldsymbol{w}_q^{(0)}$. For any step $t \geq 0$, we retain the update rule derived in 4b such that $\boldsymbol{w}_p^{(t+1)} = \boldsymbol{w}_q^{(t+1)}$ and $v_p^{(t+1)} = v_q^{(t+1)}$, so we can say that $\boldsymbol{w}_p^{(t)} = \boldsymbol{w}_q^{(t)}$ and $v_p^{(t)} = v_q^{(t)}$, $\forall t \geq 0$.

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 $1 \leq j \leq m$ and $1 \leq k \leq d$, initialize $w_{j,k}^{(0)} \sim \mathcal{N}(0, 1/m)$ (i.e., normal distribution with mean 0 and variance $1/m$). We will show that

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

For convenience, define $\boldsymbol{W} := \boldsymbol{W}^{(0)}$.

(d) Let $\boldsymbol{w}^\top$ be an arbitrary row of $\boldsymbol{W}$. Prove that

$$\mathbb{E}\left[(\boldsymbol{w}^\top \boldsymbol{x})^2\right] = \mathbb{E}\left[\sum_{i=1}^d w_i^2 x_i^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^d w_i w_j x_i x_j\right].$$

We can prove this knowing that $(\boldsymbol{w}^\top \boldsymbol{x})$ is simply the inner product, where $(\boldsymbol{w}^\top \boldsymbol{x}) = \langle w, x \rangle = \sum_{i=1}^d w_i x_i$. With this, the result becomes trivial as $(\boldsymbol{w}^\top \boldsymbol{x})^2 = \left(\sum_{i=1}^d w_i x_i\right)^2 = \sum_{i=1}^d w_i^2 x_i^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^d w_i w_j x_i x_j$

(e) Using linearity of expectation, prove that

$$\mathbb{E}\left[\sum_{i=1}^d w_i^2 x_i^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^d w_i w_j x_i x_j\right] = \frac{1}{m}\|\boldsymbol{x}\|^2.$$

Hint: It may be helpful to recall that for independent random variables X, Y, we have $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ and $\mathrm{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$.

Knowing that $(\boldsymbol{w}^\top \boldsymbol{x})^2 = \sum_{i=1}^d w_i^2 x_i^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^d w_i w_j x_i x_j$, we can rewrite as $\mathbb{E}\left[\sum_{i=1}^d w_i^2 x_i^2 + \sum_{\substack{i,j=1 \\ i \neq j}}^d w_i w_j x_i x_j\right] = \mathbb{E}\left[(\boldsymbol{w}^\top \boldsymbol{x})^2\right]$. We can also say that $\mathbb{E}[w_i] = 0$ and thus $\mathbb{E}[w_i^2 = 0]$ and the first summation goes to zero. By the remaining summation and given means, we have $\mathbb{E}\left[\sum_{\substack{i,j=1 \\ i \neq j}}^d w_i w_j x_i x_j\right] = \frac{1}{m}\|\boldsymbol{x}\|^2$

(f) Using parts (d) and (e), prove that

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

Remark: A similar property holds with the ReLU activation.

With the same reasoning as in 4e, $\mathbb{E}\left[\|\boldsymbol{W}\boldsymbol{x}\|^2\right] = \|\boldsymbol{x}\|^2$ is simply equivalent to $\mathbb{E}\left[\|\boldsymbol{W}\boldsymbol{x}\|^2\right] = \sum_i^d x_i^2$, as all $\boldsymbol{W}$ components will cancel and the squared norm of $\boldsymbol{x}$ remains.

Solution.