HW 3 on kernel methods and neural network

3.1 Direct Construction of Valid Kernels

In class, we saw that by choosing a kernel $k(x, z) = \phi(x)^\top \phi(z)$, we can implicitly map data to a high-dimensional space, and have the SVM algorithm work in that space. **One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding k.**

However, in this question, we are interested in **direct construction of kernels**. Suppose we have a function $k(x, z)$ that gives an appropriate similarity measure (similar to inner product) for our learning problem, and consider plugging k into a kernelized algorithm (like SVM) as the kernel function. In order for $k(x, z)$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. In addition, Mercer's theorem states that $k(x, z)$ is a (Mercer) kernel if and only if for any finite set

$$\{x^{(1)}, \ldots, x^{(N)}\},$$

the matrix $K \in \mathbb{R}^{N \times N}$ given by

$$K_{ij} = k(x^{(i)}, x^{(j)})$$

is symmetric and positive semi-definite.

Now comes the questions.

Let k_1, k_2 be kernels over $\mathbb{R}^D \times \mathbb{R}^D$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^D \to \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^D \to \mathbb{R}^M$ be a function mapping from $\mathbb{R}^D$ to $\mathbb{R}^M$, let k_3 be a kernel over $\mathbb{R}^M \times \mathbb{R}^M$, and let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial with positive coefficients.

For each of the functions k below, **state whether it is necessarily a kernel**. If you think it is a kernel, please prove it; if you think it is not, please prove it or give a counterexample. You can use both definitions (**inner product of feature map or PSD**) when proving the validity of the kernel. If you once proved a property in one sub-question (e.g., you proved Q1.(a)), then it is okay to use it in other questions, e.g., Q1.(f), by mentioning that you proved that property in Q1.(a); please make sure to provide the proof at least once. However, you are **not** allowed to use the 'results' from the 'Constructing kernels' slide without proving them yourself.

(Recall **Feature-map definition**: $k(\mathbf{x}, \mathbf{z})$ is a kernel if there exists a (possibly high-dimensional) mapping $\phi(\mathbf{x})$ such that

$$k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$$

PSD definition: k **is a kernel iff for any finite set of points** $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$**, the** $N \times N$ **matrix**

$$K \quad \text{with} \quad K_{ij} = k\big(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\big)$$

is symmetric and p.s.d. (i.e., $\mathbf{v}^\top K \mathbf{v} \geq 0$ for all $\mathbf{v} \in \mathbb{R}^N$).)

1. **[(a)]** [2 points] $k(x, z) = k_1(x, z) + k_2(x, z)$ **Yes**, this is a kernel.

Proof Since k_1 is a kernel, then there exists a $\phi_1(\mathbf{x})$ such that

$$k_1(\mathbf{x}, \mathbf{z}) = \langle \phi_1(\mathbf{x}),\, \phi_1(\mathbf{z}) \rangle$$

Similarly, there exists a $\phi_2(\mathbf{x})$ such that

$$k_2(\mathbf{x}, \mathbf{z}) = \langle \phi_2(\mathbf{x}),\, \phi_2(\mathbf{z}) \rangle$$

Define a combined feature map

$$\phi(\mathbf{x}) = \big(\, \phi_1(\mathbf{x}),\ \phi_2(\mathbf{x}) \big)$$

concatenatint the two vectors. Then

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \langle \phi_1(\mathbf{x}), \phi_1(\mathbf{z}) \rangle + \langle \phi_2(\mathbf{x}), \phi_2(\mathbf{z}) \rangle = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$$

Hence $k_1 + k_2$ is a kernel.

2. **[(b)]** [2 points] $k(x, z) = k_1(x, z) - k_2(x, z)$ **No**, this is not necessarily a valid kernel.

   Proof Consider the followng counterex. Take linear and const kernel.

   $$k_1(x, z) = xz$$

   This is a valid kernel because it corresponds to the standard dot product.

   $$k_2(x, z) = c$$

   where constant $c > 0$. This is a valid kernel because it produces a rank-1 positive semi-definite matrix. Then for $k = k_1 - k_2$ as the difference, we have the Gram matrix:

   $$K := K_1 - K_2 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} - \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} -1 & -3 \\ -3 & -1 \end{bmatrix}$$

   with the eigenvalues ok K

   $$\lambda = -1 \pm 3 = \{-4, 2\}$$

   Thus the matrix is not positive semi-definite, meaning K is not a valid kernel.

3. **[(c)]** [2 points] $k(x, z) = -ak_1(x, z)$ **No**, this is not necesssarily a valid kernel.

   Proof If k_1 is PSD, then for any finite set $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, the Gram matrix K_1 of k_1 is PSD, so all of its eigenvalues are ≥ 0. Since **positive** a, then multiplying by $-a$ simply multiplies all eigenvalues by $-a$. That makes the resulting matrix have nonpositive eigenvalues and thus not be PSD unless it is the all-zero matrix.

4. **[(d)]** [2 points] $k(x, z) = f(x)f(z)$ **Yes**, this is necessarily a kernel.

   Proof Consider the 1-dim feature map

   $$\phi(\mathbf{x}) \ = \ f(\mathbf{x})$$

   Then

   $$\langle \phi(\mathbf{x}),\, \phi(\mathbf{z}) \rangle = f(\mathbf{x}) \cdot f(\mathbf{z})$$

   is precisely $k(\mathbf{x}, \mathbf{z})$. As long as $f(\mathbf{x})$ is a well-defined real-valued function, this construction is valid, so $f(\mathbf{x})\, f(\mathbf{z})$ is a kernel.

5. **[(e)]** [3 points] $k(x, z) = k_3(\phi(x), \phi(z))$ **Yes**, this is necessarily a kernel.

Proof Since k_3 is a kernel on $\mathbb{R}^M$, there is some feature map ψ s.t.

$$k_3(\mathbf{u}, \mathbf{v}) \;=\; \langle \psi(\mathbf{u}), \psi(\mathbf{v}) \rangle \quad \text{for all } \mathbf{u}, \mathbf{v} \in \mathbb{R}^M$$

Now let $\mathbf{u} = \phi(\mathbf{x})$ and $\mathbf{v} = \phi(\mathbf{z})$. Define

$$\tilde{\phi}(\mathbf{x}) := \psi\big(\phi(\mathbf{x})\big)$$

Then

$$k(\mathbf{x}, \mathbf{z}) = k_3\big(\phi(\mathbf{x}), \phi(\mathbf{z})\big) \langle \psi(\phi(\mathbf{x})),\, \psi(\phi(\mathbf{z})) \rangle = \langle \tilde{\phi}(\mathbf{x}),\, \tilde{\phi}(\mathbf{z}) \rangle$$

Thus k is a kernel.

6. **[(f)]** [3 points] $k(x, z) = p(k_1(x, z))$ Yes, this is necessarily a kernel.

   Proof For the polynomial p,

   $$p(t) \;=\; \sum_{m=0}^{M} c_m\, t^m \quad \text{each } c_m \geq 0$$

   Then

   $$k(x, z) = p\big(k_1(\mathbf{x}, \mathbf{z})\big) = \sum_{m=0}^{M} c_m \big(k_1(\mathbf{x}, \mathbf{z})\big)^m$$

   From the conclusion of (g) we know that product of two kernels is a kernel. Thus

   $$\big(k_1(\mathbf{x}, \mathbf{z})\big)^m$$

   is a kernel. And multiplying a matrix by a potive number multiply all its eigenvalues by it, thus keeping its positive definiteness. Thus each $c_m \big(k_1(\mathbf{x}, \mathbf{z})\big)^m$ is also a kernel.

   Also, from the conclusion of (a) we know that addition of two kernels is a kernel. $k(x, z) = p\big(k_1(\mathbf{x}, \mathbf{z})\big) = \sum_{m=0}^{M} c_m \big(k_1(\mathbf{x}, \mathbf{z})\big)^m$ is a kernel.

7. **[(g)]** [3 points] $k(x, z) = k_1(x, z)k_2(x, z)$ **Yes**. This is necessarily a valid kernel.

   Proof We have

   $$\begin{aligned}
   k(x, z) &= k_1(x, z)k_2(x, z) && (3.1)\\
   &= \langle \phi_1(\mathbf{x}), \phi_1(\mathbf{z}) \rangle \cdot \langle \phi_2(\mathbf{x}), \phi_2(\mathbf{z}) \rangle && (3.2)\\
   &= \sum_i \Big(f_i(\mathbf{x})\, f_i(\mathbf{z}) \Big) \times \sum_j \Big(g_j(\mathbf{x})\, g_j(\mathbf{z}) \Big) && (3.3)\\
   &= \sum_i \sum_j \Big(f_i(\mathbf{x})\, f_i(\mathbf{z}) \Big) \Big(g_j(\mathbf{x})\, g_j(\mathbf{z}) \Big) && (3.4)\\
   &= \big\langle \phi_1(\mathbf{x}) \otimes \phi_2(\mathbf{x}),\; \phi_1(\mathbf{z}) \otimes \phi_2(\mathbf{z}) \big\rangle && (3.5)
   \end{aligned}$$

   Therefore by defefing:

   $$\phi(\mathbf{x}) := \phi_1(\mathbf{x}) \otimes \phi_2(\mathbf{x}) \tag{3.6}$$

   We shall have:

   $$k(x, z) = \big\langle \phi(\mathbf{x}), \phi(\mathbf{x}) \big\rangle$$

8. **[(h)]** [4 points] For this sub-problem, you are not required to check if k is a valid kernel. Instead, given a kernel $k(x, z) = (x^\top z + 1)^2$, find a feature map ϕ associated with $k(x, z)$ such that $k(x, z) = \phi(x)^\top \phi(z)$. You may assume $D = 2$ for this subproblem.

Proof Assume $D = 2$. We expand:

$$(\mathbf{x}^\top \mathbf{z} + 1)^2 = (x_1 z_1 + x_2 z_2 + 1)^2 = x_1^2 z_1^2 + 2\, x_1 z_1 x_2 z_2 + x_2^2 z_2^2 + 2\, x_1 z_1 + 2\, x_2 z_2 + 1$$

We want to write this as an inner product of feature maps. Consider:

$$\phi(\mathbf{x}) := \left(\sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1, \sqrt{2}\, x_2, x_1^2, x_2^2, 1\right)$$

Indeed, check the dot product:

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = (\sqrt{2}x_1x_2)(\sqrt{2}z_1z_2) + (\sqrt{2}x_1)(\sqrt{2}z_1) \tag{3.7}$$

$$+ (\sqrt{2}x_2)(\sqrt{2}z_2) + (x_1^2)(z_1^2) + (x_2^2)(z_2^2) + 1 \tag{3.8}$$

$$= 2\, x_1x_2\, z_1z_2 + 2\, x_1\, z_1 + 2\, x_2\, z_2 + x_1^2\, z_1^2 + x_2^2\, z_2^2 + 1 \tag{3.9}$$

which matches the polynomial expansion exactly. Thus

$$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$$

9. **[(i)]** [3 points, extra credit] Prove that the Gaussian Kernel,

$$k(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

can be expressed as $k(x, z) = \phi(x)^\top \phi(z)$, where $\phi(\cdot)$ is an infinite-dimensional vector. Specifically, find -the explicit closed form of an infinite dimensional feature vector ϕ. (Note: you cannot directly apply Mercer's theorem here as ϕ is an infinite dimensional vector.) **Hints:**

- $\|x - z\|^2 = x^\top x + z^\top z - 2x^\top z$. It might be useful to consider Power Series: $\exp(x) = \sum_{n=0}^{\infty} \frac{1}{n!} x^n$.
- You might find the formula in slide page 16 of Lecture 8 helpful.
- Each element of $\phi(x)$ can be written as an explicit formula over x without any limits or infinite summations.

Proof We have

$$k(\mathbf{x}, \mathbf{z}) := \exp\left(-\tfrac{\|\mathbf{x}-\mathbf{z}\|^2}{2\sigma^2}\right)$$

Rewrite

$$\|\mathbf{x} - \mathbf{z}\|^2 = \mathbf{x}^\top \mathbf{x} + \mathbf{z}^\top \mathbf{z} - 2\mathbf{x}^\top \mathbf{z}$$

Hence

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\tfrac{\mathbf{x}^\top \mathbf{x}}{2\sigma^2}\right) \exp\left(-\tfrac{\mathbf{z}^\top \mathbf{z}}{2\sigma^2}\right) \exp\left(\tfrac{\mathbf{x}^\top \mathbf{z}}{\sigma^2}\right)$$

Expand the exponential of the dot product by Taylor series:

$$\exp\left(\tfrac{\mathbf{x}^\top \mathbf{z}}{\sigma^2}\right) = \sum_{n=0}^{\infty} \frac{1}{n!} \left(\tfrac{\mathbf{x}^\top \mathbf{z}}{\sigma^2}\right)^n$$

And we also take the multinomial expansion of $\left(\mathbf{x}^\top \mathbf{z}\right)^n$:

$$(\mathbf{x}^\top \mathbf{z})^n \;=\; \sum_{|\alpha|=n} \frac{n!}{\alpha_1! \cdots \alpha_D!}\, x_1^{\alpha_1} \cdots x_D^{\alpha_D}\; z_1^{\alpha_1} \cdots z_D^{\alpha_D}$$

(where the sum is over all multi-indices $\alpha = (\alpha_1, \ldots, \alpha_D)$ with $\alpha_1 + \cdots + \alpha_D = n$.) Therefore we have:

$$\left(\ldots, \frac{\mathbf{x}^\alpha}{\sigma^{|\alpha|}\sqrt{\alpha!}}, \ldots\right)^\top_{\alpha \in \mathbb{N}^D} \left(\ldots, \frac{\mathbf{z}^\alpha}{\sigma^{|\alpha|}\sqrt{\alpha!}}, \ldots\right)_{\alpha \in \mathbb{N}^D} = \sum_{n=0}^{\infty} \frac{1}{n!} \left(\tfrac{\mathbf{x}^\top \mathbf{z}}{\sigma^2}\right)^n = \exp\left(\tfrac{\mathbf{x}^\top \mathbf{z}}{\sigma^2}\right)$$

(where $\mathbf{x}^\alpha = x_1^{\alpha_1} \cdots x_D^{\alpha_D}$ and $\alpha! = \alpha_1! \cdots \alpha_D!$)

Thus by taking

$$\phi(\mathbf{x}) := \exp\left(-\tfrac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\left(\ldots, \frac{\mathbf{x}^\alpha}{\sigma^{|\alpha|}\sqrt{\alpha!}}, \ldots\right)_{\alpha\in\mathbb{N}^D}$$

We finally have:

$$k(x,z) = \exp\left(-\frac{\|x-z\|^2}{2\sigma^2}\right) = \phi(\mathbf{x})^\top\phi(\mathbf{z})$$

This proves that the Gaussian kernel is the inner product in an infinite(countable)-dimensional feature space.

3.2 Implementing Soft Margin SVM by Optimizing Primal Objective

Support Vector Machines (SVM) is a discriminative model for classification. Although it is possible to develop SVMs that do K-class classifications, we will restrict ourselves to binary classification in this question, where the class label is either $+1$ (positive) or -1 (negative). **SVM is not a probabilistic algorithm**; in other words, in its usual construction, it does not optimize a probability measure as a likelihood. SVM tries to find the "best" hyperplane that maximally separates the positive class from the negative class.

Recall that the objective function for maximizing the soft margin is equivalent to the following minimization problem:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi^{(i)} \tag{3.10}$$

$$\text{subject to} \quad y^{(i)}(w^T\phi(x^{(i)}) + b) \geq 1 - \xi^{(i)}, \quad \forall i = 1,\ldots,N \tag{3.11}$$

$$\xi^{(i)} \geq 0, \quad \forall i = 1,\ldots,N \tag{3.12}$$

The above is known as the primal objective of SVM. Notice the two constraints on the slack variables $\xi^{(i)}$, which must satisfy both of these conditions while minimizing the sum of $\xi^{(i)}$ times C. The constrained minimization is equivalent to the following minimization involving the hinge loss term:

$$\min_{w,b} E(w,b) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\max(0, 1 - y^{(i)}(w^T\phi(x^{(i)}) + b)) \tag{3.13}$$

3.2.1 Derivatives of the Loss Function

Find the derivatives of the loss function $E(w,b)$ with respect to the parameters w, b. Show that:

$$\nabla_w E(w,b) = w - C\sum_{i=1}^{N}\mathbb{I}[y^{(i)}(w^T\phi(x^{(i)}) + b) < 1]y^{(i)}\phi(x^{(i)}) \tag{3.14}$$

$$\frac{\partial}{\partial b}E(w,b) = -C\sum_{i=1}^{N}\mathbb{I}[y^{(i)}(w^T\phi(x^{(i)}) + b) < 1]y^{(i)} \tag{3.15}$$

where $\mathbb{I}[\cdot]$ is the indicator function. If $f(x) = \max(0, x)$, then assume that

$$\frac{\partial f}{\partial x} = \begin{cases} 1, & x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.16}$$

Proof

$$\nabla_w E(w,b) = \nabla_w \tfrac{1}{2}\|w\|_2 + C\sum_{i=1}^{N}\nabla_w \max\left(0, 1-y^{(i)}(w^T\varphi(x^{(i)})+b)\right)$$

$$= w$$

where $\nabla_w \max\left(0, 1-y^{(i)}(w^T\varphi(x^{(i)})+b)\right)$

$$= \begin{cases} \nabla_w -y^{(i)}(w^T\varphi(x^{(i)})+b) = -y^{(i)}\varphi(x^{(i)}), & \text{if } y^{(i)}(w^T\varphi(x^{(i)})+b) < 1 \\ \nabla_w 0 = 0, & \text{otherwise} \end{cases}$$

Thus $\nabla_w E(w,b) = w - C\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}(w^T\varphi(x^{(i)})+b) < 1\} y^{(i)}\varphi(x^{(i)})$

Similarly, $\nabla_b \max\left(0, 1-y^{(i)}(w^T\varphi(x^{(i)})+b)\right)$

$$= \begin{cases} \nabla_b -y^{(i)}(w^T\varphi(x^{(i)})+b) = -y^{(i)}, & \text{if } y^{(i)}(w^T\varphi(x^{(i)})+b) < 1 \\ \nabla_b 0 = 0, & \text{otherwise} \end{cases}$$

Thus $\nabla_b E(w,b) = -C\sum_{i=1}^{N} \mathbb{I}\{y^{(i)}(w^T\varphi(x^{(i)})+b) < 1\} y^{(i)}$

3.2.2 SVM Batch Gradient Descent Implementation

Implement the SVM algorithm using batch gradient descent, following part 1 of 'soft margin svm.ipynb'. In previous assignments, you have implemented gradient descent while minimizing over one variable. Minimizing over two variables (w, b) is not different. Both gradients are computed from current parameter values, and then parameters are simultaneously updated. (Note: it is a common mistake to implement gradient descent over multiple parameters by updating the first parameter, then computing the derivative w.r.t second parameter using the updated value of the first parameter. In fact, updating one parameter then computing the gradient of the second parameter using the updated value of the first parameter, is a different optimization algorithm, known as Coordinate Descent.)

Example pseudocode for optimizing w and b is given by Algorithm 2.

Algorithm 2 SVM Batch Gradient Descent

$w^* \leftarrow 0$
$b^* \leftarrow 0$
for $j = 1$ to NumEpochs **do**
  $w_{\text{grad}} \leftarrow \nabla_w E(w^*, b^*)$
  $b_{\text{grad}} \leftarrow \frac{\partial}{\partial b} E(w^*, b^*)$
  $w^* \leftarrow w^* - \eta(j) w_{\text{grad}}$
  $b^* \leftarrow b^* - \eta(j) b_{\text{grad}}$
end for
return w^*, b^*

Throughout this question, we set $C = 5$, NumEpochs = 100, and the learning rate η as a constant, i.e., $\eta(j) = 1e^{-3}$. (Remarks: In this problem, we only asked you to implement batch gradient descent for primal SVM, but it' s quite straightforward to implement stochastic gradient descent in a very similar way (although you may need to decay the learning rate and set the initial learning rate properly). Unlike stochastic gradient descent, we don' t need to decay the learning rate for convergence.)

report parameters

Run gradient descent over the training data 5 times, once for each of the NumEpochs = {1, 3, 10, 30, 100}. For each run, report the trained parameters (w, b) and the test classification accuracy.

```
[NumEpochs: 1] Accuracy: 54.17%
b: [0.01], W: [[ 0.224  -0.0855  0.545   0.206 ]]
[NumEpochs: 3] Accuracy: 54.17%
b: [0.02], W: [[ 0.44848122 -0.17019759  1.08918105  0.41163421]]
[NumEpochs: 10] Accuracy: 95.83%
b: [-0.14], W: [[-0.1648026  -0.80606447  1.37816462  0.57445096]]
[NumEpochs: 30] Accuracy: 95.83%
b: [-0.175], W: [[-0.20885861 -0.69979483  1.30489009  0.56605151]]
[NumEpochs: 100] Accuracy: 95.83%
b: [-0.315], W: [[-0.28240917 -0.77529188  1.75856715  0.82441652]]
```

3.3 Asymmetric Cost SVM

Consider applying an SVM to a supervised learning problem where the cost of a false positive (mistakenly predicting $+1$ when the label is -1) is different from the cost of a false negative (predicting -1 when the label is $+1$). The asymmetric cost SVM models these asymmetric costs by posing the following optimization problem:

$$\min_{w,b,\xi} \quad \frac{1}{2}w^\top w + C_0 \sum_{i:y^{(i)}=-1} \xi^{(i)} + C_1 \sum_{i:y^{(i)}=1} \xi^{(i)} \tag{3.17}$$

$$\text{s.t.} \quad y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi^{(i)}, \quad \forall i = 1, \ldots, N \tag{3.18}$$

$$\xi^{(i)} \geq 0, \quad \forall i = 1, \ldots, N \tag{3.19}$$

Here, C_0 is the cost of a false positive; C_1 is the cost of a false negative. (Both C_0 and C_1 are fixed, known constants.)

3.3.1 Lagrangian Formulation [4 points]

We will find the dual optimization problem. First, write down the Lagrangian. Use $\alpha^{(i)}$ and $\mu^{(i)}$ to denote the Lagrange multipliers corresponding to the two constraints ($\alpha^{(i)}$ for the first constraint, and $\mu^{(i)}$ for the second constraint (slack variables)) in the primal optimization problem above.

Sol. Write the problem into the standard form:

$$\min_{w,b,\xi} \quad \frac{1}{2}w^\top w + C_0 \sum_{i:y^{(i)}=-1} \xi^{(i)} + C_1 \sum_{i:y^{(i)}=1} \xi^{(i)} \tag{3.20}$$

$$\text{s.t.} \quad -\left(y^{(i)}(w^\top x^{(i)} + b) - 1 + \xi^{(i)}\right) \leq 0 \quad \forall i = 1, \ldots, N \tag{3.21}$$

$$-\xi^{(i)} \leq 0, \quad \forall i = 1, \ldots, N \tag{3.22}$$

Thus the Lagrangian is:

$$\mathcal{L}(w, b, \xi, \alpha, \mu) = \frac{1}{2}w^\top w + \sum_{i=1}^{N} C^{(i)}\xi^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)}(w^\top x^{(i)} + b) - 1 + \xi^{(i)}\right] - \sum_{i=1}^{N} \mu^{(i)}\xi^{(i)}$$

where

$$C^{(i)} = C_0 \mathbb{I}[y^{(i)} = -1] + C_1 \mathbb{I}[y^{(i)} = 1]$$

3.3.2 Derivatives for Dual Optimization [6 points]

Calculate the following derivatives with respect to the primal variables:

Sol.

$$\nabla_w \mathcal{L}(w, b, \xi, \alpha, \mu) = w - \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \nabla_w (w^\top x^{(i)} + b) \tag{3.23}$$

$$= w - \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)} \tag{3.24}$$

$$\frac{\partial \mathcal{L}(w, b, \xi, \alpha, \mu)}{\partial b} = -\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \frac{\partial (w^\top x^{(i)} + b)}{\partial b} \tag{3.25}$$

$$= -\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} (0 + \frac{\partial b}{\partial b}) \tag{3.26}$$

$$= -\sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \tag{3.27}$$

$$\nabla_{\xi^{(i)}} \mathcal{L}(w, b, \xi, \alpha, \mu) = \sum_{j=1}^{N} C^{(j)} \nabla_{\xi^{(i)}} \xi^{(j)} - \sum_{j=1}^{N} \alpha^{(j)} \nabla_{\xi^{(i)}} \left(-1 + \xi^{(j)}\right) - \sum_{j=1}^{N} \mu^{(j)} \nabla_{\xi^{(i)}} \xi^{(j)} \tag{3.28}$$

$$= C^{(i)} - \alpha^{(i)} - \mu^{(i)} \quad (\text{since } \nabla_{\xi^{(i)}} \xi^{(j)} = 1 \text{ if } i = j; = 0 \text{ otherwise}) \tag{3.29}$$

3.3.3 Dual Optimization Problem [10 points]

Find the dual optimization problem. You should write down the dual optimization problem in the following form. Try to simplify your answer as much as possible. In particular, to obtain full credit, the Lagrange multipliers $\mu^{(i)}$ should not appear in your simplified form of the dual.

Sol. Here the dual optimization problem is:

$$\max_{\mu, \alpha: \alpha_i \geq 0 \ \forall i} \min_{w,b,\xi} \mathcal{L}(w, b, \xi, \alpha, \mu)$$

So we need to first find $\min_{w,b,\xi} \mathcal{L}(w, b, \xi, \alpha, \mu)$. This can be done by setting the gradients to 0. For fixed μ, α,

taking the stationary conditions:

$$\nabla_w \mathcal{L}(w, b, \xi, \alpha, \mu) = 0 \implies w^* = \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)}$$

$$\frac{\partial \mathcal{L}(w, b, \xi, \alpha, \mu)}{\partial b} = 0 \implies \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0 \text{ as constraint}$$

$$\text{each } \nabla_{\xi^{(i)}} \mathcal{L}(w, b, \xi, \alpha, \mu) = 0 \implies \alpha^{(i)} + \mu^{(i)} = C^{(i)} \ \forall i$$

Since $\mu^{(i)} \geq 0$, this imposes the constraint:

$$0 \leq \alpha^{(i)} \leq C^{(i)} \ \ \forall i$$

Thus

$$\mathcal{L}(w^*, b^*, \xi^*, \alpha, \mu) = \frac{1}{2} w^\top w + \sum_{i=1}^{N} C^{(i)} \xi^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)} (w^\top x^{(i)} + b) - 1 + \xi^{(i)} \right] - \sum_{i=1}^{N} \mu^{(i)} \xi^{(i)} \tag{3.30}$$

$$= \frac{1}{2} w^\top w + \sum_{i=1}^{N} \left(C^{(i)} - \mu^{(i)} \right) \xi^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)} (w^\top x^{(i)} + b) - 1 + \xi^{(i)} \right] \tag{3.31}$$

$$= \frac{1}{2} w^\top w + \sum_{i=1}^{N} \alpha^{(i)} \xi^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)} (w^\top x^{(i)} + b) \right] + \sum_{i=1}^{N} \alpha^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} \xi^{(i)} \tag{3.32}$$

$$= \frac{1}{2} w^\top w + \sum_{i=1}^{N} \alpha^{(i)} - \sum_{i=1}^{N} \alpha^{(i)} \left[y^{(i)} (w^\top x^{(i)}) \right] - b \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} \tag{3.33}$$

$$= \frac{1}{2} w^\top w + \sum_{i=1}^{N} \alpha^{(i)} - w^\top \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} x^{(i)} - 0 \tag{3.34}$$

$$= \frac{1}{2} w^\top w + \sum_{i=1}^{N} \alpha^{(i)} - w^\top w \tag{3.35}$$

$$= \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} w^\top w \tag{3.36}$$

$$= \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} x^{(i)\top} x^{(j)} \tag{3.37}$$

Thus the dual optimization problem is:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{N} \alpha^{(i)} - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha^{(i)} \alpha^{(j)} y^{(i)} y^{(j)} x^{(i)\top} x^{(j)} \tag{3.38}$$

$$\text{s.t.} \quad \sum_{i=1}^{N} \alpha^{(i)} y^{(i)} = 0, \tag{3.39}$$

$$0 \leq \alpha^{(i)} \leq C^{(i)}, \quad \forall i = 1, \ldots, N \tag{3.40}$$

3.4 SVMs with Convex Optimization

In this problem, you will practice training SVMs on toy data. You will learn how to use the popular Scikit-Learn library's SVM implementation and how to use the convex optimization library CVXOPT to train an SVM by

directly solving the dual optimization problem.

3.4.1 write the dual of SVM into quadratic programming [4 points]

Recall from lecture that the (kernelized) dual optimization problem of SVMs can be written as follows:

$$\max_{\alpha} \quad \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m k(x_n, x_m) \tag{3.41}$$

$$\text{subject to} \quad 0 \leq \alpha_n \leq C, \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \tag{3.42}$$

where there are N training examples, $x^{(i)} \in \mathbb{R}^D$, $y^{(i)} \in \{-1, 1\}$, k is a kernel function $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, $\alpha \in \mathbb{R}^N$, and $C \in \mathbb{R}$.

To solve this problem using CVXOPT, we will use the quadratic programming solver `cvxopt.solvers.qp`. Given matrices P, G, A and vectors q, h, b, CVXOPT can solve the following optimization problem (quadratic programming):

$$\min_{v} \quad \frac{1}{2} v^\top P v + q^\top v \tag{3.43}$$

$$\text{subject to} \quad Gv \preceq h, \quad Av = b \tag{3.44}$$

Find values for matrices P, G, A and vectors q, h, b in terms of $x^{(i)}, y^{(i)}, k(x^{(i)}, x^{(j)})$ and C such that the solution of the SVM dual problem (α in Equation 5) is equal to the solution of the CVXOPT equation (v in Equation 6).

Hint 1: A maximization problem can be the same as a minimization when applying a sign change:

$$\min_{x} f(x) = \max_{x} -f(x). \tag{3.45}$$

Hint 2: The constraint $0 \leq \alpha_n \leq C$ can be separated into two constraints:

$$-\alpha_n \leq 0, \quad \alpha_n \leq C. \tag{3.46}$$

Sol. the maximization problem is the same as the below minimization:

$$\min_{v} \quad -\sum_{n=1}^{N} \alpha_n + \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n P_{mn} \alpha_m \tag{3.47}$$

$$\text{subject to} \quad 0 \leq \alpha_n \leq C, \quad \sum_{n=1}^{N} \alpha_n y_n = 0 \tag{3.48}$$

Consider let $(P_{nm}) := y_n y_m k(x_n, x_m)$, then

$$-\sum_{n=1}^{N} \alpha_n + \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m k(x_n, x_m) = -\sum_{n=1}^{N} \alpha_n + \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n P_{mn} \alpha_m$$

Thus for by defining the input vector $v := (\alpha_1, \cdots, \alpha_N)^\top$, we have:

$$-\sum_{n=1}^{N} \alpha_n + \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m k(x_n, x_m) = -\mathbf{1}_N \cdot v - \frac{1}{2} v^\top P v$$

For $\alpha_n \geq 0$, i.e. $-\alpha_n \leq 0$, we take $G_1 = -I$ (the negative identity matrix) and vector $h_1 = \mathbf{0}_N$; and for $\alpha_n \leq C$, we take $G_2 = I$ (the identity matrix) and vector $h_2 = C\mathbf{1}_N$, concatenate the two matrices and vectors

as:

$$G = \begin{pmatrix} -I_N \\ I_N \end{pmatrix}, \quad h = \begin{pmatrix} \mathbf{0}_N \\ C\mathbf{1}_N \end{pmatrix}$$

, then we can transform the constraint $0 \leq \alpha_n \leq C$ into:

$$Gv \preceq h$$

This is because $-\alpha_n \leq 0$ and $\alpha_n \leq C$ for all α_n is imposed by the positive semi-definiteness of this matrix. And the constraint $\sum_{n=1}^{N} \alpha_n y_n = 0$ can be imposed by $A\alpha = 0$ by taking

$$A := [y_1, y_2, \ldots, y_N]$$

as a $1 \times$ N row vector. (and $b := 0$)

Then we can write it into:

$$\min_v \quad \frac{1}{2} v^\top P v + q^\top v \tag{3.49}$$

$$\text{subject to} \quad Gv \preceq h, \quad Av = b \tag{3.50}$$

where $(P_{nm}) := y_n y_m k(x_n, x_m)$, $G = \begin{pmatrix} -I_N \\ I_N \end{pmatrix}$, $h = \begin{pmatrix} \mathbf{0}_N \\ C\mathbf{1}_N \end{pmatrix}$, $A := [y_1, y_2, \ldots, y_N]$, $q = -\mathbf{1}_N$, $b = 0$

3.4.2 implementing SVM with cvxopt [10 points]

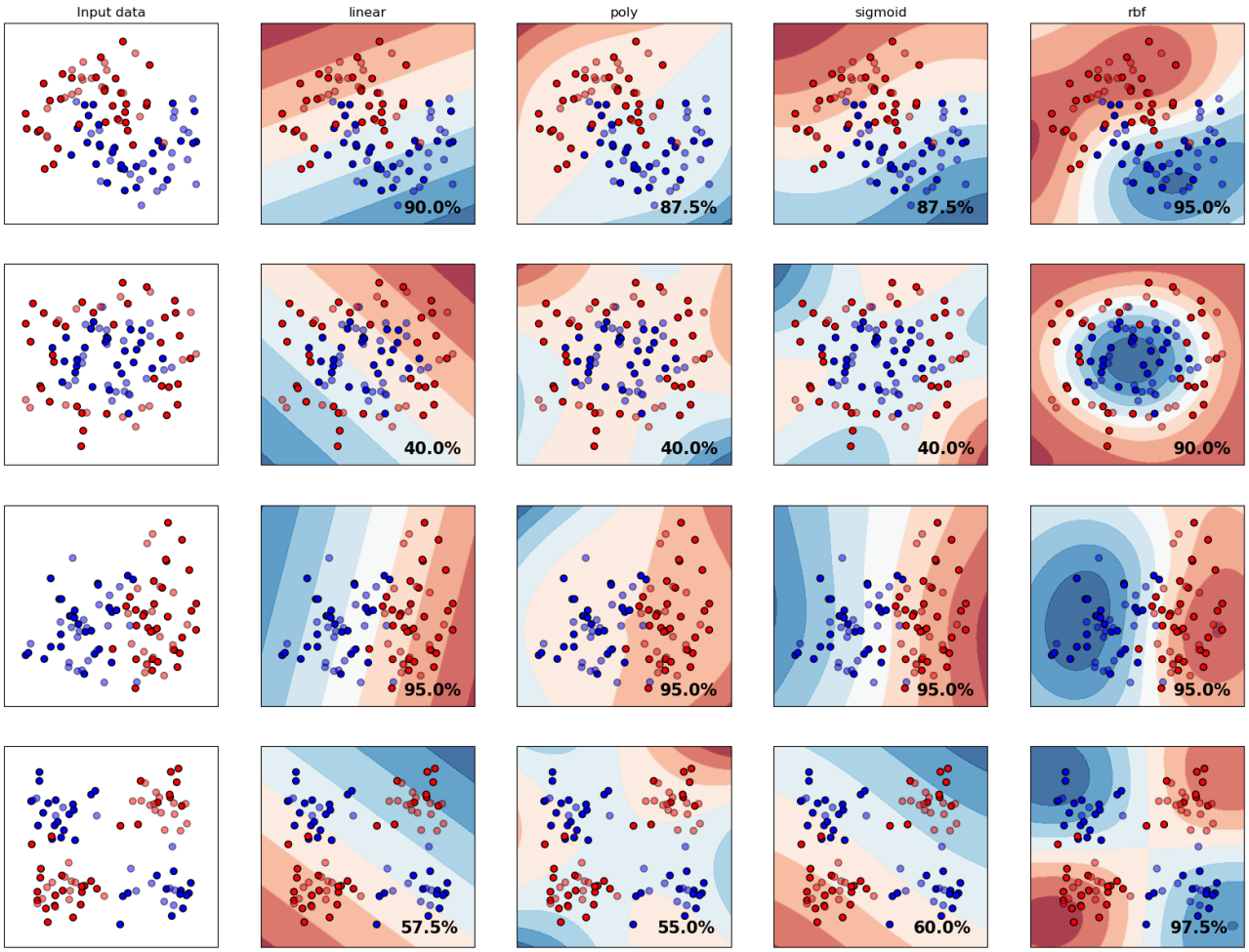
(Autograder) Use your derivation above to complete the code in the `svm.py` file.

3.4.3 draw decision boundary [4 points]

(Autograder) In the notebook, you will implement drawing a decision boundary on a simple dataset and identify support vectors from the solutions to the optimization problems. Include the test performance on each dataset and the generated visualizations of the CVXOPT SVMs in your submission writeup (generated in `svm.ipynb`).

CVXOPTSVM performance

	linear	poly	sigmoid	rbf
dataset 0	90.00	87.50	87.50	95.00
dataset 1	40.00	40.00	40.00	90.00
dataset 2	95.00	95.00	95.00	95.00
dataset 3	57.50	55.00	60.00	97.50



3.5 Neural Network Layer Implementation

In this problem, you will implement various neural network layers. Let X, Y represent the input and output of the layers, respectively. We use row-major notation (i.e., each row of X and Y corresponds to one data sample) to align with the multi-dimensional array structure of NumPy. Furthermore, L is the scalar-valued loss function of Y.

For the Fully-Connected Layer and ReLU, you should include your derivation of the gradient in your written solution.

Important Note: Any indices involved (i, j, etc.) in the mathematical notation start from 1, not 0. However, in your code, you should use 0-based indexing (standard NumPy notation).

3.5.1 Fully-Connected Layer [8 points]

In this question, you will be deriving the gradient of a fully-connected layer. For this problem, consider:

- $X \in \mathbb{R}^{N \times D_{in}}$, where N is the number of samples in a batch.
- A dense layer with weight parameters $W \in \mathbb{R}^{D_{in} \times D_{out}}$ and bias parameters $b \in \mathbb{R}^{1 \times D_{out}}$.

As we saw in the lecture, we can compute the output of the layer through a simple matrix multiplication operation:

$$Y = XW + B, \tag{3.51}$$

where $Y \in \mathbb{R}^{N \times D_{out}}$ is the output matrix and $B \in \mathbb{R}^{N \times D_{out}}$ is the bias matrix where each row of B is the vector b. (Please note that we are using row-vector notation here to make it compatible with NumPy/PyTorch, but the lecture slides used column-vector notation, which is standard notation for most ML methods. We hope that the distinction is clear from the context.)

Please note that the matrix multiplication operation stated above is generally useful for computing batch outputs (i.e., simultaneously computing the output of N samples in the batch). However, for the purposes of computing the gradient, you might first want to consider the single-sample output operation of this fully-connected layer, which can be stated in row-major notation as:

$$y^{(n)} = x^{(n)}W + b, \tag{3.52}$$

where $y^{(n)} \in \mathbb{R}^{1 \times D_{out}}$ and $x^{(n)} \in \mathbb{R}^{1 \times D_{in}}$ for $1 \leq n \leq N$. Here, the index (n) in the superscript denotes the n-th example, not the layer index, and $X_i^{(n)} = X_{n,i}$ denotes the i-th dimension of the n-th example $x^{(n)}$.

Note: It might be fruitful for you to consider this expression as a summation and then calculate the gradient for individual parameters for a single-example case. After this, you can try to extend it to a vector/matrix format for the involved parameters.

Now, compute the partial derivatives (in matrix form):

$$\frac{\partial L}{\partial W} := \nabla_W L \in \mathbb{R}^{D_{in} \times D_{out}}, \tag{3.53}$$

$$\frac{\partial L}{\partial b} := \nabla_b L \in \mathbb{R}^{1 \times D_{out}}, \tag{3.54}$$

$$\frac{\partial L}{\partial X} := \nabla_X L \in \mathbb{R}^{N \times D_{in}}, \tag{3.55}$$

in terms of:

$$\frac{\partial L}{\partial Y} := \nabla_Y L \in \mathbb{R}^{N \times D_{out}}. \tag{3.56}$$

For technical correctness, you might want to start with writing the gradient with a non-vectorized form involving:

$$\frac{\partial L}{\partial Y_j^{(n)}} \in \mathbb{R}, \tag{3.57}$$

where $\frac{\partial L}{\partial Y_j^{(n)}}$ is the gradient with respect to the j-th element of the n-th sample in Y with $1 \leq n \leq N$ and $1 \leq j \leq D_{out}$.

calculate $\frac{\partial L}{\partial W}$

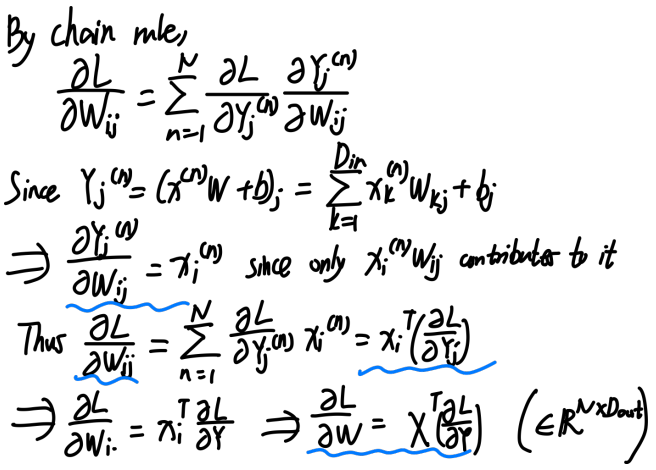
Hint: Please note that we use a "matrix/vector" notation $\frac{\partial L}{\partial W}$ to denote a matrix in $\mathbb{R}^{D_{in} \times D_{out}}$, where the element in the i-th row and j-th column is:

$$\frac{\partial L}{\partial W_{i,j}}, \tag{3.58}$$

and $W_{i,j}$ is the (i,j)-th element of W with $1 \leq i \leq D_{in}$ and $1 \leq j \leq D_{out}$. Here you may want to calculate the gradient $\frac{\partial L}{\partial W_{i,j}}$ using the formula:

$$\frac{\partial L}{\partial W_{i,j}} = \sum_{n=1}^{N} \frac{\partial L}{\partial Y_j^{(n)}} \frac{\partial Y_j^{(n)}}{\partial W_{i,j}}$$

Sol.



Thus we conclude:

$$\frac{\partial L}{\partial W} = X^\top \left(\frac{\partial L}{\partial Y}\right)$$

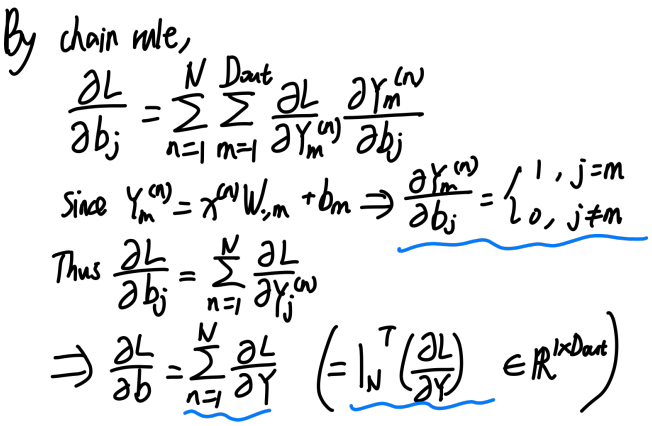
calculate $\frac{\partial L}{\partial b}$

Hint: Please note that $\frac{\partial L}{\partial b}$ is a vector in $\mathbb{R}^{1 \times D_{out}}$. You may want to start by using the formula:

$$\frac{\partial L}{\partial b_j} = \sum_{n=1}^{N} \sum_{m=1}^{D_{out}} \frac{\partial L}{\partial Y_m^{(n)}} \frac{\partial Y_m^{(n)}}{\partial b_j}$$

and then move on to derive $\frac{\partial L}{\partial b}$.

Sol.



Thus we conclude:

$$\frac{\partial L}{\partial b} = 1_N^\top \left(\frac{\partial L}{\partial Y}\right)$$

calculate $\frac{\partial L}{\partial X}$

Hint: Please note that $\frac{\partial L}{\partial X}$ is a matrix in $\mathbb{R}^{N \times D_{in}}$. In order to derive this gradient, you can start by using the formula:

$$\frac{\partial L}{\partial X_i^{(n)}} = \sum_{n'=1}^{N} \sum_{m=1}^{D_{out}} \frac{\partial L}{\partial Y_m^{(n')}} \frac{\partial Y_m^{(n')}}{\partial X_i^{(n)}}$$

Following this, you can say that:

$$\frac{\partial L}{\partial X_i^{(n)}} = \sum_{m=1}^{D_{out}} \frac{\partial L}{\partial Y_m^{(n)}} \frac{\partial Y_m^{(n)}}{\partial X_i^{(n)}}$$

because the n-th sample in X is only related to the n-th sample in Y and:

$$\frac{\partial Y_m^{(n')}}{\partial X_i^{(n)}} = 0 \quad \text{for all } n' \neq n.$$

Sol.

By chain rule,

$$\frac{\partial L}{\partial X_i^{(n)}} = \sum_{n'=1}^{N} \sum_{m=1}^{D_{out}} \frac{\partial L}{\partial Y_m^{(n')}} \frac{\partial Y_m^{(n')}}{\partial X_i^{(n)}}$$

Since only $n'=n$ matters,

$$\frac{\partial L}{\partial X_i^{(n)}} = \sum_{m=1}^{D_{out}} \frac{\partial L}{\partial Y_m^{(n)}} \frac{\partial Y_m^{(n)}}{\partial X_i^{(n)}}$$

Since $Y_m^{(n)} = (X^{(n)}W + b)_m = \sum_{k=1}^{D_{in}} X_k^{(n)} W_{k,m} + b_m$

$$\Rightarrow \frac{\partial Y_m^{(n)}}{\partial X_i^{(n)}} = W_{i,m}$$

Thus $\frac{\partial L}{\partial X_i^{(n)}} = \sum_{m=1}^{D_{out}} \frac{\partial L}{\partial Y_m^{(n)}} W_{i,m}$

$$\Rightarrow \frac{\partial L}{\partial X^{(n)}} = \frac{\partial L}{\partial Y^{(n)}} W^T \quad (\in \mathbb{R}^{1 \times D_{in}})$$

$$\Rightarrow \frac{\partial L}{\partial X} = \left(\frac{\partial L}{\partial Y}\right) W^T \quad (\in \mathbb{R}^{N \times D_{in}})$$

Thus we conclude:

$$\frac{\partial L}{\partial X} = (\frac{\partial L}{\partial Y}) W^\top$$

3.5.2 ReLU [3 points]

Let $X \in \mathbb{R}^{N \times D}$ be a 2-D array and $Y = \text{ReLU}(X)$. In this case, ReLU is applied to X in an element-wise fashion. For an element $x \in X$, the corresponding output is:

$$y = \text{ReLU}(x) = \max(0, x).$$

You can observe that Y will have the same shape as X.

Express $\frac{\partial L}{\partial X}$ in terms of $\frac{\partial L}{\partial Y}$, where $\frac{\partial L}{\partial X}$ has the same shape as X.

Hint: You may need to use the element-wise product to express $\frac{\partial L}{\partial X}$. Please feel free to introduce the indicator function $\mathbb{I}[\cdot]$.

Sol.

By chain rule,

$$\frac{\partial L}{\partial X_{n,d}} = \sum_{n'=1}^{N} \sum_{d'=1}^{D} \frac{\partial L}{\partial Y_{n',d'}} \frac{\partial Y_{n',d'}}{\partial X_{n,d}}$$

Since $Y_{n,d} = \max(0, X_{n,d})$

$$\Rightarrow \frac{\partial Y_{n',d'}}{\partial X_{n,d}} = \begin{cases} 1, & \text{if } n'=n,\ d'=d \text{ and } X_{n,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$\Rightarrow \frac{\partial L}{\partial X_{n,d}} = \frac{\partial L}{\partial Y_{n,d}} \frac{\partial Y_{n,d}}{\partial X_{n,d}} = \frac{\partial L}{\partial Y_{n,d}} \cdot \mathbb{I}\{X_{n,d} > 0\}$$

Thus $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot \mathbb{I}\{X > 0\}$

where $\odot$ is elementwise multiplication.

Thus we conclude:

$$\frac{\partial L}{\partial X} = (\frac{\partial L}{\partial Y}) \odot \mathbb{I}\{X > 0\}$$

3.5.3 Autograder Implementation [14 points]

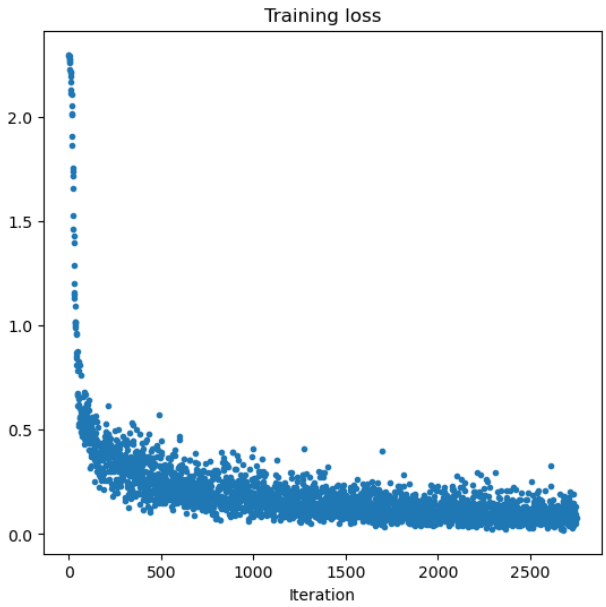
Now, we will implement the corresponding forward and backward passes in `two_layer_net.py`. More importantly, please ensure that you use vectorization to make your layer implementations as efficient as possible. You should use `two_layer_net.ipynb` to help you check the correctness of your implementation. Note that for each layer, the iPython notebook just checks the result for one specific example. Passing these tests does not necessarily mean that your implementation is 100% correct. Once you implement the layers properly, please implement a two fully-connected layer-based classifier with a Softmax loss in `two_layer_net.py` and test it on the MNIST dataset, following the guide in `two_layer_net.ipynb`. All your implementations should go into `two_layer_net.py`.

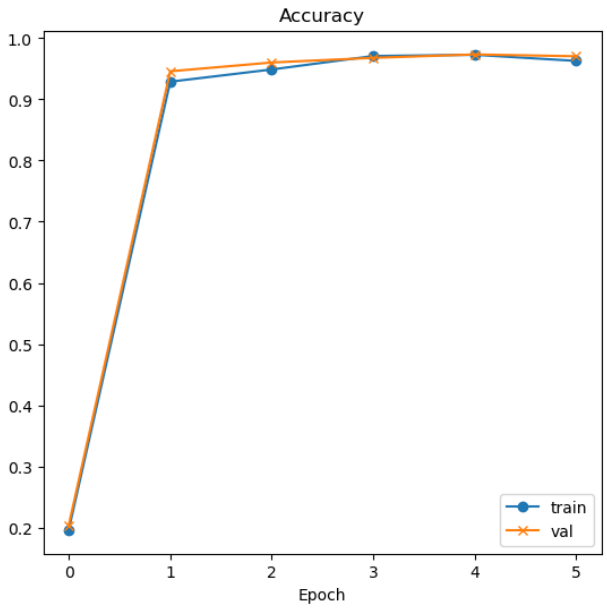
Report Training Loss and Accuracy Plot [1 point]

Please report the training loss with train/test accuracy plot of the MNIST dataset, stored as `two_layer_net.png` from `two_layer_net.ipynb`, in your writeup.

Sol.

```
(Epoch 0 / 5) train acc: 19.60%    val_acc: 20.36%
(Epoch 1 / 5) train acc: 92.90%    val_acc: 94.60%
(Epoch 2 / 5) train acc: 94.90%    val_acc: 96.04%
(Epoch 3 / 5) train acc: 97.10%    val_acc: 96.80%
(Epoch 4 / 5) train acc: 97.30%    val_acc: 97.36%
(Epoch 5 / 5) train acc: 96.30%    val_acc: 97.06%
```





Report Test Set Accuracy [1 point]

Please report the accuracy obtained on the test set of the MNIST dataset in your writeup.

Sol.

```
Test accuracy: 96.31%
```