Homework #3

CSE 446/546: Machine Learning
Profs. Jamie Morgenstern and Simon Du
Due: **Wednesday** November 17, 2021 11:59pm
A: 71 points, **B:** 15 points

Please review all homework guidance posted on the website before submitting to GradeScope. Reminders:

- Make sure to read the "What to Submit" section following each question and include all items.
- Please provide succinct answers and supporting reasoning for each question. Similarly, when discussing experimental results, concisely create tables and/or figures when appropriate to organize the experimental results. All explanations, tables, and figures for any particular part of a question must be grouped together.
- For every problem involving generating plots, please include the plots as part of your PDF submission.
- When submitting to Gradescope, please link each question from the homework in Gradescope to the location of its answer in your homework PDF. Failure to do so may result in deductions of up to *[5 points]* . For instructions, see `https://www.gradescope.com/get_started#student-submission`.
- Please recall that B problems, indicated in boxed text, are only graded for 546 students, and that they will be weighted at most 0.2 of your final GPA (see the course website for details). In Gradescope, there is a place to submit solutions to A and B problems separately. You are welcome to create a single PDF that contains answers to both and submit the same PDF twice, but associate the answers with the individual questions in Gradescope.
- If you collaborate on this homework with others, you must indicate who you worked with on your homework. Failure to do so may result in accusations of plagiarism.
- For every problem involving code, please submit your code to the separate assignment on Gradescope created for code. Not submitting all code files will lead to a deduction of *[1 point]* .
- Please indicate your final answer to each question by placing a box around the main result(s). To do this in LaTeX, one option is using the `boxed` command.

Not adhering to these reminders may result in point deductions.

Short Answer and "True or False" Conceptual Questions

A1. The answers to these questions should be answerable without referring to external materials. Briefly justify your answers with a few words.

a. *[2 points]* Say you trained an SVM classifier with an RBF kernel ($K(u,v) = \exp\left(-\frac{\|u-v\|_2^2}{2\sigma^2}\right)$). It seems to underfit the training set: should you increase or decrease σ?

Solution: We should **decrease** σ. From class, we know that the bandwidth σ affects fit. It essentially controls the bias-variance tradeoff. If we decrease σ, it decreases the bias which should improve the fit of the training set. However, it will increase the variance and make the fit less smooth.

---

b. *[2 points]* True or False: Training deep neural networks requires minimizing a non-convex loss function, and therefore gradient descent might not reach the globally-optimal solution.

Solution: True. Training deep neural networks do require minimizing a non-convex loss function. Since we are minimizing a non-convex loss function, it may have many local minima.Therefore, gradient descent might converge to one of these local minima which may not be globally optimal. The solution reached by gradient descent will be highly dependent on the starting weights.

---

c. *[2 points]* True or False: It is a good practice to initialize all weights to zero when training a deep neural network.

Solution: False. This is not a good practice. If we consider the sigmoid activation function as an example, initialization all weights to zero leads to zero derivatives and perfect symmetry, and the algorithm never moves. Usually starting values for weights are chosen to be random values near zero. Hence models start out nearly linear, and become nonlinear as the weights increase.

---

d. *[2 points]* True or False: We use non-linear activation functions in a neural network's hidden layers so that the network learns non-linear decision boundaries.

Solution: True. We need non-linear activation functions in a neural network's hidden layers so that the network learns non-linear decision boundaries. We use non linear activation functions so that neural networks can approximate non linear functions by making non linear transformation of inputs to match non linear phenomenon. It can also predict class of a function that does not have linear decision boundaries. Without non linear activation functions, out output is just going to be a linear combination of the inputs and we will not be able to make the network learn non-linear decision boundaries.

---

e. *[2 points]* True or False: Given a neural network, the time complexity of the backward pass step in the backpropagation algorithm can be prohibitively larger compared to the relatively low time complexity of the forward pass step.

Solution: False. The time complexity of the backward pass step in the backpropagation algorithm is the same up to a constant as the time complexity of the forward pass step in a neural network. However, this would not be true if we used a naive approach.

---

f. *[2 points]* True or False: Neural Networks are the most extensible model and therefore the best choice for every circumstance.

Solution: False. Overfitting is an issue with neural networks. Neural networks provides the ability to pick from a large number of models, so the probability that a particular model performs well on test and validation is relatively high. If we have a linear problem, we would rather just use a linear method, since the extra fitting of a neural network is not useful and can be actively harmful. Also, neural networks are computationally expensive and requires a large amount of data which might not be available. There is also an interpretability issue which might make it not ideal for certain circumstances. This means that it is hard to interpret why neural network classifies things the way it does which might actually be important to know in certain circumstances.

Support Vector Machines

A2. Assume w is an n-dimensional vector and b is a scalar. A hyperplane in $\mathbb{R}^n$ is the set $\{x \in \mathbb{R}^n \mid w^\top x + b = 0\}$.

a. *[1 point]* ($n = 2$ example) Draw the hyperplane for $w = \begin{bmatrix} -1 & 2 \end{bmatrix}^\top$, $b = 2$? Label your axes.

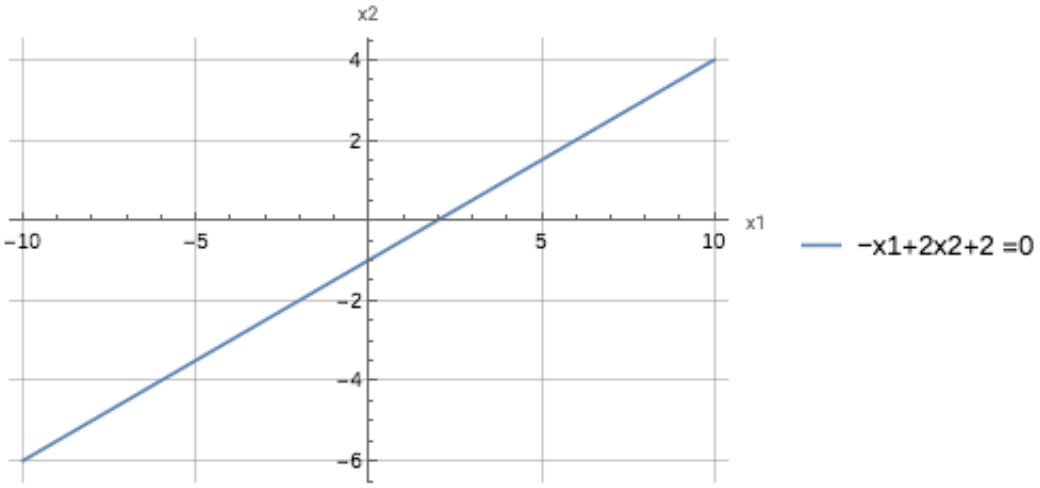
Solution:

Let $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$.

$$\begin{aligned} & w^\top x + b = 0 \\ \implies & -x_1 + 2x_2 + 2 = 0 \end{aligned}$$

The hyperplane can then be expressed by:

$$\{x \in \mathbb{R}^n : x_2 = \frac{x_1}{2} - 1\}$$



The hyperplane was generated using the following piece of code in **Mathematica**:

```
Plot[x/2-1,{x, -10, 10}, AxesLabel->{x1, x2}, GridLines->Automatic,
PlotLegends ->{"-x_1+2x_2+2 =0"}]
```

---

b. *[1 point]* ($n = 3$ example) Draw the hyperplane for $w = \begin{bmatrix}1 & 1 & 1\end{bmatrix}^\top$, $b = 0$? Label your axes.

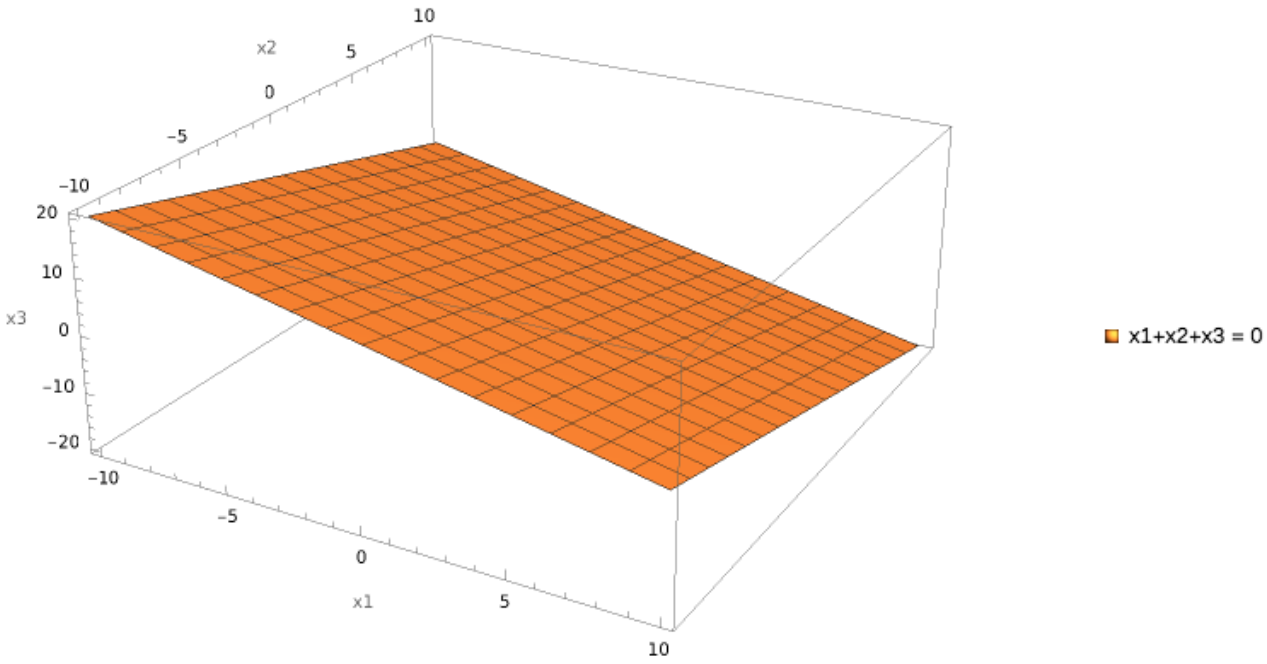
Solution:

Let $x = \begin{bmatrix}x_1 \\ x_2 \\ x_3\end{bmatrix}$.

$$\begin{aligned} & w^\top x + b = 0 \\ \implies & x_1 + x_2 + x_3 = 0 \end{aligned}$$

The hyperplane can then be expressed by:

$$\{x \in \mathbb{R}^n : x_3 = -x_1 - x_2\}$$



The hyperplane was generated using the following piece of code in **Mathematica**:

```
Plot3D[-x-y,{x,-10,10},{y,-10,10}, AxesLabel->{x1, x2, x3}, PlotLegends ->{"x1+x2+x3 = 0"}]
```

---

c. *[2 points]* Let $w^\top x + b = 0$ be the hyperplane generated by a certain SVM optimization. Given some point $x_0 \in \mathbb{R}^n$, Show that the minimum *squared distance* to the hyperplane is $\frac{|x_0^\top w + b|}{\|w\|_2}$. In other words, show the following:

$$\min_x \|x_0 - x\|_2^2 = \frac{|x_0^\top w + b|}{\|w\|_2}$$
$$\text{subject to: } w^\top x + b = 0\,.$$

Hint: Think about projecting x_0 onto the hyperplane. (Hint: If $\widetilde{x}_0$ is the minimizer of the above problem, show that $\|x_0 - \widetilde{x}_0\|_2 = \left|\frac{w^\top(x_0 - \widetilde{x}_0)}{\|w\|_2}\right|$. What is $w^\top \widetilde{x}_0$?)

Solution: We want to show that:

$$\min_x \|x_0 - x\|_2 = \frac{|x_0^\top w + b|}{\|w\|_2}$$
$$\text{subject to: } w^\top x + b = 0\,.$$

Suppose $\widetilde{x}_0$ is the minimizer of the above problem. We know that orthogonal projection will minimize the distance and since w is orthogonal to the hyperplane we have that $x_0 - \widetilde{x}_0$ is parallel to w. We can also observe that $\widetilde{x}_0$ satisfies $w^\top \widetilde{x}_0 + b = 0$. Using the fact that w and $x_0 - x_0$ are parallel:

$$\begin{aligned}
x_0 - \widetilde{x}_0 &= \frac{w}{\|w\|_2}\|x_0 - \widetilde{x}_0\|_2 \\
\implies |w^\top(x_0 - \widetilde{x}_0)| &= \left|\frac{w^\top w}{\|w\|_2}\|x_0 - \widetilde{x}_0\|_2\right| \\
\implies \|x_0 - \widetilde{x}_0\|_2 &= \frac{\|w\|_2}{\|w\|_2^2}|w^\top(x_0 - \widetilde{x}_0)| \\
&= \frac{|w^\top x_0 - w^\top \widetilde{x}_0|}{\|w\|_2} \\
\implies \|x_0 - \widetilde{x}_0\|_2 &= \frac{|w^\top x_0 + b|}{\|w\|_2} \quad (\text{since } w^\top \widetilde{x}_0 + b = 0 \implies w^\top \widetilde{x}_0 = -b)
\end{aligned}$$

where $\widetilde{x}_0$ minimizes $\|x_0 - x\|_2$ for all points on the hyperplane. We can also note that $w^\top x_0 = \sum w_i x_i = \sum = x_i w_i = x_0^\top w$. Thus:

$$\|x_0 - \widetilde{x}_0\|_2 = \boxed{\frac{|x_0^\top w + b|}{\|w\|_2}}$$

$$\implies \min_x \|x_0 - x\|_2 = \frac{|x_0^\top w + b|}{\|w\|_2}$$
$$\text{subject to: } w^\top x + b = 0\,.$$

Kernels

A3. *[5 points]* Suppose that our inputs x are one-dimensional and that our feature map is infinite-dimensional: $\phi(x)$ is a vector whose ith component is:

$$\frac{1}{\sqrt{i!}}\mathrm{e}^{-x^2/2}x^i \ ,$$

for all nonnegative integers i. (Thus, ϕ is an infinite-dimensional vector.) Show that $K(x, x') = \mathrm{e}^{-\frac{(x-x')^2}{2}}$ is a kernel function for this feature map, i.e.,

$$\phi(x) \cdot \phi(x') = \mathrm{e}^{-\frac{(x-x')^2}{2}} \ .$$

Hint: Use the Taylor expansion of $z \mapsto \mathrm{e}^z$. (This is the one dimensional version of the Gaussian (RBF) kernel).

Solution: Notice

$$\begin{aligned}
\phi(x) \cdot \phi(x') &= \sum_{i=0}^{\infty} \left(\frac{1}{\sqrt{i!}} e^{-\frac{x^2}{2}} x^i \frac{1}{\sqrt{i!}} e^{-\frac{x'^2}{2}} x'^i \right) \\
&= \sum_{i=0}^{\infty} \frac{1}{i!} e^{\frac{-x^2-x'^2}{2}} x^i x'^i \\
&= e^{\frac{-x^2-x'^2}{2}} \sum_{i=0}^{\infty} \frac{1}{i!} x^i x'^i \\
&= e^{\frac{-x^2-x'^2}{2}} \sum_{i=0}^{\infty} \frac{1}{i!} (xx')^i
\end{aligned}$$

Recall: The Taylor expansion of e^x is given by

$$\begin{aligned}
e^x &= 1 + x + \frac{x^2}{2!} + \dots \\
&= \sum_{i=0}^{\infty} \frac{x^i}{i!}
\end{aligned}$$

Using the Taylor expansion of e^x, we have the following:

$$\sum_{i=0}^{\infty} \frac{1}{i!} (xx')^i = e^{xx'}$$

Hence:

$$\begin{aligned}
\phi(x) \cdot \phi(x') &= e^{\frac{-x^2-x'^2}{2}} e^{xx'} \\
&= e^{\frac{-x^2-x'^2+2xx'}{2}} \\
&= e^{-\frac{(x^2-2xx'+x'^2)}{2}} \\
&= \boxed{e^{-\frac{(x-x')^2}{2}}}
\end{aligned}$$

We have shown that $K(x, x') = e^{-\frac{(x-x')^2}{2}}$ is a kernel function for this feature map.

Intro to sample complexity

B1. For $i = 1, \ldots, n$ let $(x_i, y_i) \overset{\text{i.i.d.}}{\sim} P_{X,Y}$ where $y_i \in \{-1, 1\}$ and x_i lives in some set $\mathcal{X}$ (x_i is not necessarily a vector). The 0/1 loss, or *risk*, for a deterministic classifier $f \colon \mathcal{X} \to \{-1, 1\}$ is defined as:

$$R(f) = \mathbb{E}_{X,Y}[\mathbf{1}(f(X) \neq Y)]$$

where $\mathbf{1}(\mathcal{E})$ is the indicator function for the event $\mathcal{E}$ (the function takes the value 1 if $\mathcal{E}$ occurs and 0 otherwise). The expectation is with respect to the underlying distribution $P_{X,Y}$ on (X, Y). Unfortunately, we don't know $P_{X,Y}$ exactly, but we do have our i.i.d. samples $\{(x_i, y_i)\}_{i=1}^n$ drawn from it. Define the *empirical risk* as

$$\widehat{R}_n(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(f(x_i) \neq y_i) \ ,$$

which is just an empirical estimate of our risk. Suppose that a learning algorithm computes the empirical risk $R_n(f)$ for all $f \in \mathcal{F}$ and outputs the prediction function $\widehat{f}$ which is the one with the smallest empirical risk. (In this problem, we are assuming that $\mathcal{F}$ is finite.) Suppose that the best-in-class function f^* (i.e., the one that minimizes the true 0/1 loss) is:

$$f^* = \arg\min_{f \in \mathcal{F}} R(f) \, .$$

a. *[2 points]* Suppose that for some $f \in \mathcal{F}$, we have $R(f) > \epsilon$. Show that

$$\mathbb{P}\left[\widehat{R}_n(f) = 0\right] \leq \mathrm{e}^{-n\epsilon} \ .$$

(You may use the fact that $1 - \epsilon \leq \mathrm{e}^{-\epsilon}$.)

Solution:

Notice:

$$\begin{aligned} \widehat{R}_n(f) = 0 &\iff \sum_{i=1}^{n} \mathbf{1}(f(x_i) \neq y_i) = 0 \\ &\iff f(x_i) = y_i \ \forall \ i \in \{1, 2, \ldots, n\} \end{aligned}$$

Thus:

$$\begin{aligned} \mathbb{P}\left[\widehat{R}_n(f) = 0\right] &= \mathbb{P}\left[f(x_i) = y_i \ \forall \ i \in \{1, 2, \ldots, n\}\right] \\ &= \mathbb{P}\left[f(x_1) = y_1, \ldots, f(x_n) = y_n\right] \\ &= \mathbb{P}\left[f(x_1) = y_1\right] \ldots \mathbb{P}\left[f(x_n) = y_n\right] \ \text{(by i.i.d)} \\ &= \prod_{i=1}^{n} \mathbb{P}\left[f(x_i) = y_i\right] \\ &= \prod_{i=1}^{n} \Big(1 - \mathbb{P}\left[f(x_i) \neq y_i\right]\Big) \\ &= \Big(1 - \mathbb{P}\left[f(x_i) \neq y_i\right]\Big)^n \ \text{(by i.i.d)} \\ &= \Big(1 - \mathbb{E}_{X,Y}[\mathbf{1}(f(X) \neq Y)]\Big)^n \ \text{(expectation of indicator of an event is the probability of the event)} \\ &= \Big(1 - R(f)\Big)^n \\ &\leq (1 - \epsilon)^n \\ &\leq (e^{-\epsilon})^n \end{aligned}$$

$$\implies \boxed{\mathbb{P}\left[\widehat{R}_n(f) = 0\right] \leq e^{-n\epsilon}}$$

---

b. *[2 points]* Use the *union bound* to show that

$$\mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right] \leq |\mathcal{F}|\mathrm{e}^{-\epsilon n}.$$

Recall that the union bound says that if $A_1, \ldots, A_k$ are events in a probability space, then

$$\mathbb{P}\left[A_1 \cup A_2 \cup \ldots \cup A_k\right] \leq \sum_{1 \leq i \leq k} \mathbb{P}(A_i).$$

Solution: We are interested in finding the probability that at least one $f \in \mathcal{F}$ satisfies the event above. This is equivalent to finding the probability that either $f_1 \in \mathcal{F}$ satisfies the event or $f_2 \in \mathcal{F}$ satisfies the event and so on. Let us define $A_i = R(f_i) > \epsilon$ and $\widehat{R}_n(f_i) = 0$ for a particular $f_i \in \mathcal{F}$. We can consider this for every $f_i \in \mathcal{F}$ and we want to find the probability that A_1 is true or A_2 is true or A_3 is true and so on. For any $f_i \in \mathcal{F}$, we have that:

$$\begin{aligned}
\mathbb{P}\left[R(f_i) > \epsilon \text{ and } \widehat{R}_n(f_i) = 0\right] &= \mathbb{P}(A_i) \\
&= \mathbb{P}\left[\widehat{R}_n(f_i) = 0 | R(f_i) > \epsilon\right] \times \mathbb{P}\left[R(f_i) > \epsilon\right] \\
&\leq \mathbb{P}\left[\widehat{R}_n(f_i) = 0 | R(f_i) > \epsilon\right] \text{ (since } \mathbb{P}\left[R(f_i) > \epsilon\right] \leq 1) \\
&= e^{-\epsilon n} \text{ (from part a)}
\end{aligned}$$

Thus, the probability of event A_i for all $i \in \{1, \ldots, |\mathcal{F}|\}$ is less than or equal to $e^{-\epsilon n}$. Using the union bound:

$$\begin{aligned}
\mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right] &= \mathbb{P}(A_1 \cup A_2 \cup \cdots \cup A_{|\mathcal{F}|}) \\
&\leq \sum_{i=1}^{|\mathcal{F}|} \mathbb{P}(A_i) \\
&= \mathbb{P}(A_i) \sum_{i=1}^{|\mathcal{F}|} 1 \\
&= |\mathcal{F}| \mathbb{P}(A_i) \\
&\leq |\mathcal{F}| e^{-\epsilon n}
\end{aligned}$$

Therefore, we have shown the following:

$$\boxed{\mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right] \leq |\mathcal{F}| e^{-\epsilon n}}.$$

---

c. *[2 points]* Solve for the minimum ϵ such that $|\mathcal{F}|\mathrm{e}^{-\epsilon n} \leq \delta$.

Solution:

We want to find the minimum ϵ such that $|\mathcal{F}|\mathrm{e}^{-\epsilon n} \leq \delta$. Notice:

$$
\begin{aligned}
& |\mathcal{F}|e^{-\epsilon n} \leq \delta \\
\iff & e^{\epsilon n} \geq \frac{|\mathcal{F}|}{\delta} \\
\iff & \epsilon n \geq \log\left(\frac{|\mathcal{F}|}{\delta}\right) \qquad \text{(taking log on both sides)} \\
\iff & \epsilon \geq \frac{1}{n}\log\left(\frac{|\mathcal{F}|}{\delta}\right)
\end{aligned}
$$

Therefore, $\boxed{\epsilon = \frac{1}{n}\log\left(\frac{|\mathcal{F}|}{\delta}\right)}$ is the minimum ϵ such that $|\mathcal{F}|\mathrm{e}^{-\epsilon n} \leq \delta$.

---

d. *[4 points]* Use this to show that with probability at least $1-\delta$

$$\widehat{R}_n(\widehat{f}) = 0 \quad \implies \quad R(\widehat{f}) - R(f^*) \leq \frac{\log(|\mathcal{F}|/\delta)}{n}$$

where $\widehat{f} = \arg\min_{f \in \mathcal{F}} \widehat{R}_n(f)$.

Solution: Suppose $\widehat{R}_n(\widehat{f}) = 0$ where $\widehat{f} = \arg\min_{f \in \mathcal{F}} \widehat{R}_n(f)$.
We want to show the following:

$$\mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \implies R(\widehat{f}) - R(f^*) \leq \frac{\log(|\mathcal{F}|/\delta)}{n}\right) \geq 1-\delta.$$

Notice: The event $\widehat{R}_n(\widehat{f}) = 0 \implies R(\widehat{f}) - R(f^*) \leq \frac{\log(|\mathcal{F}|/\delta}{n}$ is equivalent to the event $\widehat{R}_n(\widehat{f}) \neq 0$ or $R(\widehat{f}) - R(f^*) \leq \frac{\log(|\mathcal{F}|/\delta}{n}$. If we negate this, we get the event:

$$\widehat{R}_n(\widehat{f}) = 0 \text{ and } R(\widehat{f}) - R(f^*) > \frac{\log(|\mathcal{F}|/\delta}{n}$$

Using this fact we can see the following:

$$\mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \implies R(\widehat{f}) - R(f^*) \leq \frac{\log(|\mathcal{F}|/\delta)}{n}\right) = 1 - \mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \text{ and } R(\widehat{f}) - R(f^*) > \frac{\log(|\mathcal{F}|/\delta)}{n}\right)$$

Notice: If $R(\widehat{f}) - R(f^*) > \frac{\log(|\mathcal{F}|/\delta}{n}$, then $R(\widehat{f}) > \frac{\log(|\mathcal{F}|/\delta}{n}$ since $R(f^*)) \geq 0$. Thus, the first event is a subset of the second event and we have that:

$$\mathbb{P}\left(R(\widehat{f}) - R(f^*) > \frac{\log(|\mathcal{F}|/\delta}{n}\right) \leq \mathbb{P}\left(R(\widehat{f}) > \frac{\log(|\mathcal{F}|/\delta}{n}\right)$$

Thus:

$$\begin{aligned} 1 - \mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \text{ and } R(\widehat{f}) - R(f^*) > \frac{\log(|\mathcal{F}|/\delta)}{n}\right) &\geq 1 - \mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \text{ and } R(\widehat{f}) > \frac{\log(|\mathcal{F}|/\delta)}{n}\right) \\ &= 1 - \mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \text{ and } R(\widehat{f}) > \epsilon\right) \end{aligned}$$

where $\epsilon = \frac{\log(|\mathcal{F}|/\delta)}{n}$. Since $\widehat{f} = \arg\min_{f \in \mathcal{F}} \widehat{R}_n(f)$, the above is equivalent to:

$$1 - \mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \text{ and } R(\widehat{f}) > \epsilon\right) \geq 1 - \mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right]$$

This is because if there exists an $f \in \mathcal{F}$ such that $\widehat{R}_n(f) = 0$ it will be the argmin by default as $\widehat{R}_n(f) \geq 0$. So, the first event implies the second event, that is, there exists an $f \in \mathcal{F}$ such that $\widehat{R}_n(f) = 0 \implies \widehat{R}_n(\widehat{f}) = 0$. Therefore, the first event is a subset of the second event. Finally:

$$\begin{aligned} \mathbb{P}\left(\widehat{R}_n(\widehat{f}) = 0 \implies R(\widehat{f}) - R(f^*) \leq \frac{\log(|\mathcal{F}|/\delta)}{n}\right) &\geq 1 - \mathbb{P}\left[\exists f \in \mathcal{F} \text{ s.t. } R(f) > \epsilon \text{ and } \widehat{R}_n(f) = 0\right] \\ &\geq \boxed{1-\delta} \text{ (from part c)} \end{aligned}$$

as desired.

Perceptron

B2. One of the oldest algorithms used in machine learning (from early 60's) is an online algorithm for learning a linear threshold function called the Perceptron Algorithm:

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = 0$, and initialize t to 1. Also let's automatically scale all examples $\mathbf{x}$ to have (Euclidean) norm 1, since this doesn't affect which side of the plane they are on.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

If we make a mistake on a positive $\mathbf{x}$ we get $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t + \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} + 1$, and similarly if we make a mistake on a negative $\mathbf{x}$ we have $\mathbf{w}_{t+1} \cdot \mathbf{x} = (\mathbf{w}_t - \mathbf{x}) \cdot \mathbf{x} = \mathbf{w}_t \cdot \mathbf{x} - 1$. So, in both cases we move closer (by 1) to the value we wanted. Here is a link if you are interested in more details.

Now consider the linear decision boundary for classification (labels in $\{-1, 1\}$) of the form $\mathbf{w} \cdot \mathbf{x} = 0$ (i.e., no offset). Now consider the following loss function evaluated at a data point $(\mathbf{x}, y)$ which is a variant on the hinge loss.

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max\{0, -y(\mathbf{w} \cdot \mathbf{x})\}.$$

a. *[2 points]* Given a dataset of $(\mathbf{x}_i, y_i)$ pairs, write down a single step of subgradient descent with a step size of η if we are trying to minimize

$$\frac{1}{n} \sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w})$$

for $\ell(\cdot)$ defined as above. That is, given a current iterate $\widetilde{\mathbf{w}}$ what is an expression for the next iterate?

Solution:

Let $\mathbf{w}_t$ be the current iterate. We are minimizing

$$L((\mathbf{x}_i, y_i), \mathbf{w}) = \frac{1}{n} \sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w})$$

and so the next iterate is going to be:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \Big[L((\mathbf{x}_i, y_i), \mathbf{w}_t) \Big] \\ &= \mathbf{w}_t - \eta \Big(\frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{w}} \big[\ell((\mathbf{x}_i, y_i), \mathbf{w}_t) \big] \Big) \end{aligned}$$

where

$$\nabla_{\mathbf{w}} \big[\ell((\mathbf{x}_i, y_i), \mathbf{w}) \big] = \begin{cases} -y_i \mathbf{x}_i & \text{if } -y_i(\mathbf{w}_t \cdot \mathbf{x}_i) \geq 0 \\ 0 & \text{if } -y_i(\mathbf{w}_t \cdot \mathbf{x}_i) < 0 \end{cases}$$

---

b. *[2 points]* Use what you derived to argue that the Perceptron can be viewed as implementing SGD applied to the loss function just described (for what value of η)?

Solution: First, we will consider a batch size of 1. This means that the expression we derived above reduces to:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \Big[\ell((\mathbf{x}_i, y_i), \mathbf{w}_t) \Big]$$

We will consider two cases: when we predict correctly and when we make a mistake. Let us consider when we predict correctly. If $\mathbf{w}_t \cdot \mathbf{x}_i < 0$ and we have predicted correctly, then $y_i(\mathbf{w}_t \cdot \mathbf{x}_i) > 0$. Similarly, if $\mathbf{w}_t \cdot \mathbf{x}_i > 0$ and we predicted correctly then $y_i(\mathbf{w}_t \cdot \mathbf{x}_i) > 0$. Thus, correct prediction means that $-y_i(\mathbf{w}_t \cdot \mathbf{x}_i) < 0$. As we can see above, the SGD step in this case becomes:

$$\mathbf{w}_t = \mathbf{w}_t.$$

Now, suppose we made a mistake. This means that $\mathbf{w}_t \cdot \mathbf{x}_i$ has the opposite sign of $y_i \implies y_i(\mathbf{w}_t \cdot \mathbf{x}_i) > 0$. Then, the SGD update becomes:

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta(-y_i \mathbf{x}_i) \\ &= \mathbf{w}_t + \eta(y_i \mathbf{x}_i) \end{aligned}$$

If we had made a mistake on positive $y_i = 1$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta(\mathbf{x}_i)$$

If we had made a mistake on negative $y_i = -1$:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta(\mathbf{x}_i)$$

Therefore, we can see that if we set $\eta = 1$ the above becomes:

Mistake on positve: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}_i$

Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}_i$

This is the same as what we had in the Perceptron algorithm. So, Perceptron can be viewed as implementing SGD applied to the loss function for $\eta = 1$. We would also have to use a batch size of 1.

---

c. *[1 point]* Suppose your data was drawn i.i.d. and that there exists a $\mathbf{w}^*$ that separates the two classes perfectly. Provide an explanation for why hinge loss is generally preferred over the loss given above.

Solution:

Recall: The hinge loss is given by the following:

$$\ell((\mathbf{x}, y), \mathbf{w}) = \max\{0, 1 - y(\mathbf{w} \cdot \mathbf{x})\}.$$

The difference comes from this addition of 1 in the hinge loss. If we make a correct prediction, that is, $y(\mathbf{w} \cdot \mathbf{x}) > 0$, then $1 - y(\mathbf{w} \cdot \mathbf{x}) > 0$ could still be greater than zero. In the loss given above, this would be strictly negative and so there will be no update made. However, in the hinge loss case there is a possibility of a non-zero loss even in the case of a correct prediction. This means that these data points will be used to update the weights in the hinge loss case but not in the case of the loss given above. This should help to increase the margin of the decision boundary.

Coding

Introduction to PyTorch

A4. PyTorch is a great tool for developing, deploying and researching neural networks and other gradient-based algorithms. In this problem we will explore how this package is built, and re-implement some of its core components. Firstly start by reading `README.md` file provided in `intro_pytorch` subfolder. A lot of problem statements will overlap between here, readme's and comments in functions.

a. *[10 points]* You will start by implementing components of our own PyTorch modules. You can find these in folders: `layers`, `losses` and `optimizers`. Almost each file there should contain at least one problem function, including exact directions for what to achieve in this problem. Lastly, you should implement functions in `train.py` file.

Solution: Code submitted on Gradescope.

---

b. *[5 points]* Next we will use the above module to perform hyperparameter search. Here we will also treat loss function as a hyper-parameter. However, because cross-entropy and MSE require different shapes we are going to use two different files: `crossentropy_search.py` and `mean_squared_error_search.py`. For each you will need to build and train (in provided order) 5 models:

- Linear neural network (Single layer, no activation function)
- NN with one hidden layer (2 units) and sigmoid activation function after the hidden layer
- NN with one hidden layer (2 units) and ReLU activation function after the hidden layer
- NN with two hidden layer (each with 2 units) and Sigmoid, ReLU activation functions after first and second hidden layers, respectively
- NN with two hidden layer (each with 2 units) and ReLU, Sigmoid activation functions after first and second hidden layers, respectively

For each loss function, submit a plot of losses from training and validation sets. All models should be on the same plot (10 lines per plot), with two plots total (1 for MSE, 1 for cross-entropy).

Solution:

Mean Squared Error:

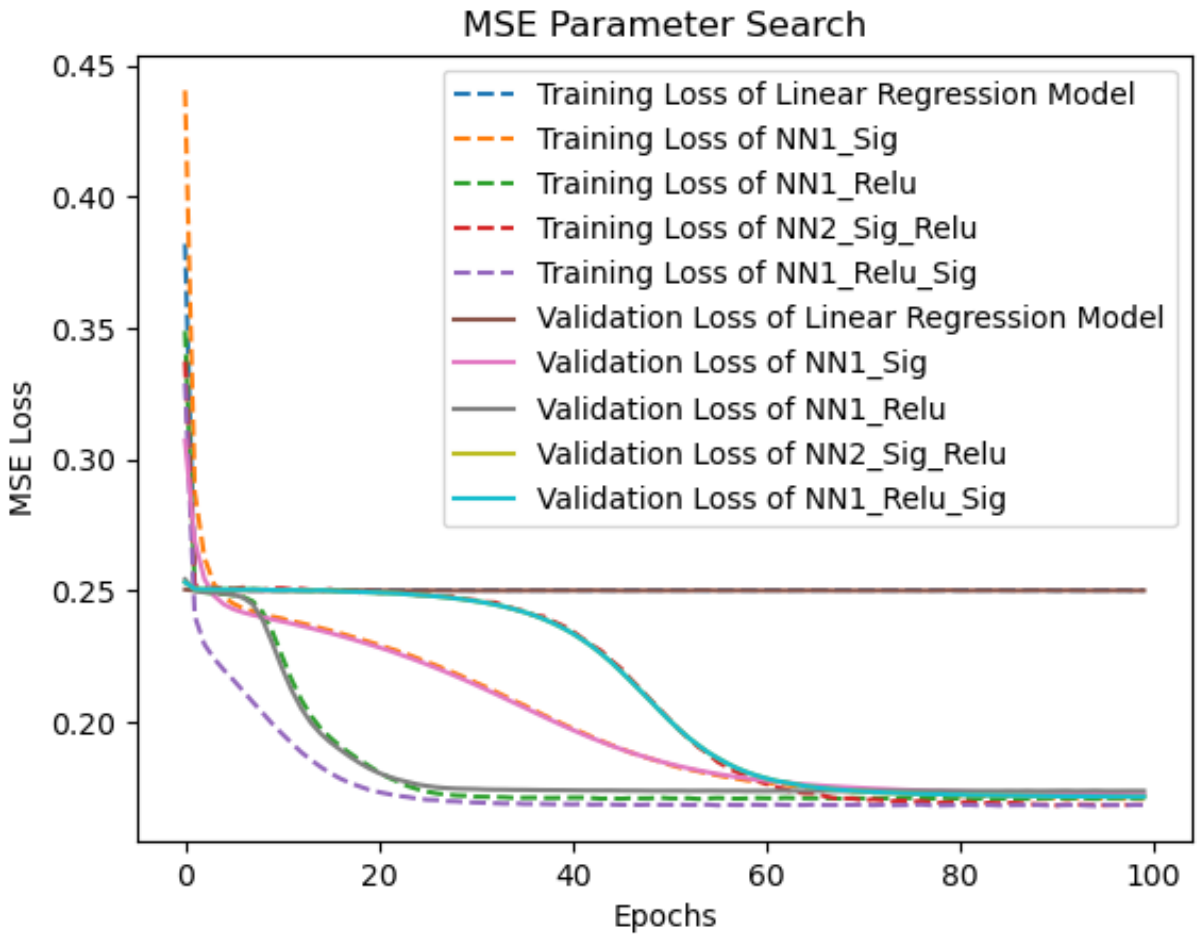


Figure 1: Plot of losses for Mean Squared Error Loss

Cross Entropy Loss:

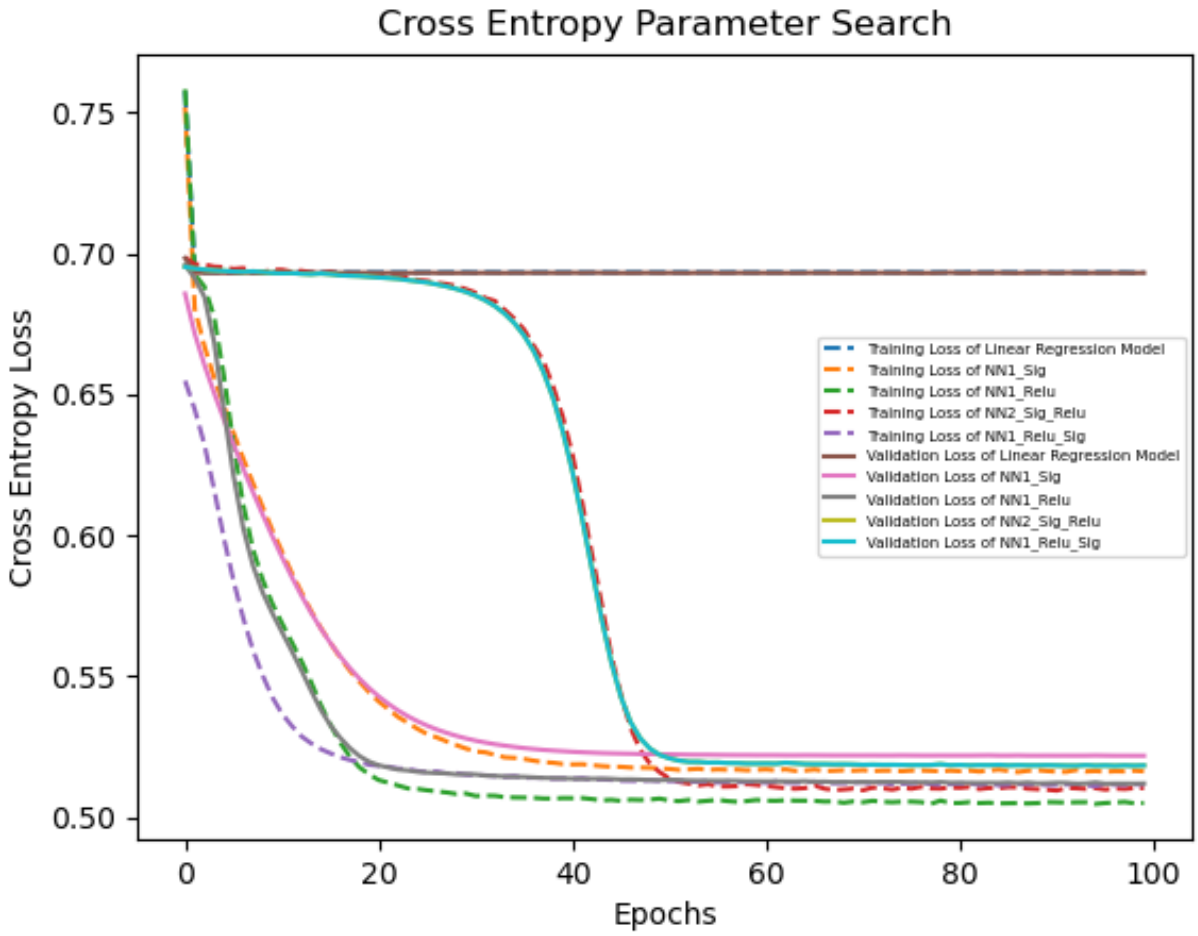


Figure 2: Plot of losses for Cross Entropy Loss

c. *[5 points]* For each loss function, report the best performing architecture (best performing is defined here as achieving the lowest validation loss at any point during the training), and plot it's guesses on test set. You should use function `plot_model_guesses` from `train.py` file. Lastly, report accuracy of that model on a test set.

Solution:

Mean Squared Error:

The best performing architecture is: **NN with two hidden layer (each with 2 units) and ReLU, Sigmoid activation functions after first and second hidden layers, respectively.**

Test Accuracy: **0.7502** $\approx$ **75**%

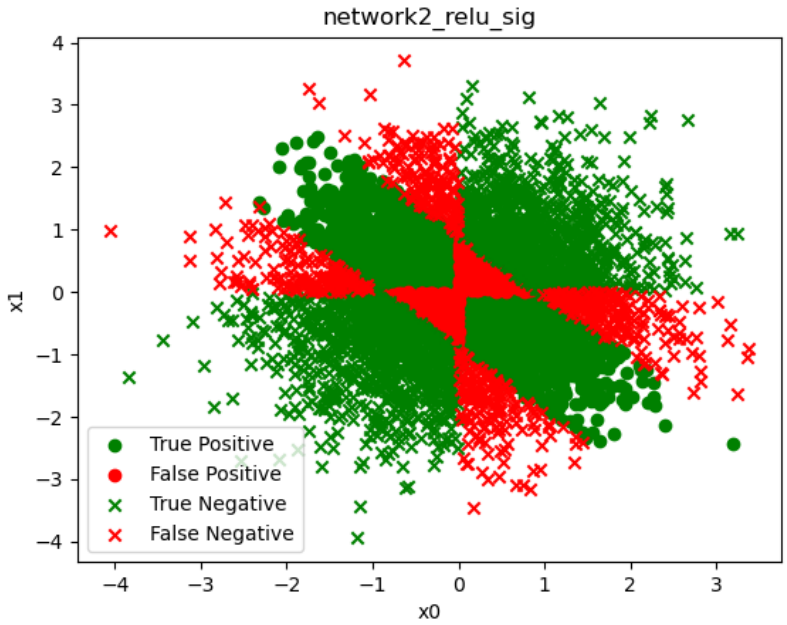


Figure 4: Plot of guesses using best performing architecture for Mean Squared Error Loss

Cross Entropy Loss:

The best performing architecture is: **NN with one hidden layer (2 units) and ReLU activation function after the hidden layer**

Test accuracy: $\mathbf{0.7468} \approx \mathbf{75}\%$

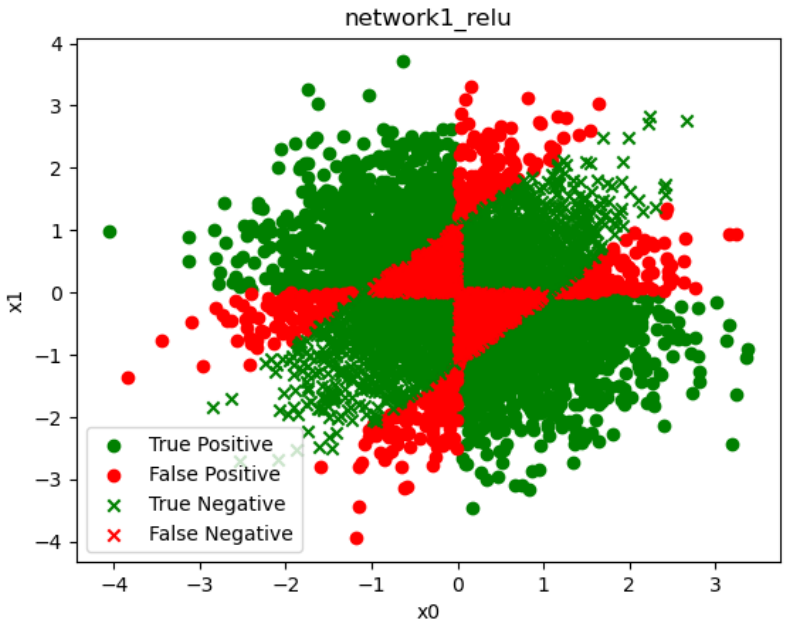


Figure 4: Plot of guesses using best performing architecture for Cross Entropy Loss

d. *[3 points]* Is there a big gap in performance between between MSE and Cross-Entropy models? If so, explain why it occurred? If not explain why different loss functions achieve similar performance? Answer in 2-4 sentences.

Solution:

The training losses have a big gap but the exact value of losses is not really relevant for comparison against one other in classification. If we compare the best performing architectures' test accuracies between the two losses, we can notice that they are pretty much the same at roughly 75%. Since this is a classification problem we should mainly be concerned about accuracy and not loss. In that regard, the two loss functions achieve similar performance. In general, we should be using Cross Entropy for classification problems but I am not sure why they have similar performance here. It maybe because we only have two classes. It could also be due to the fact that we are using a normal distribution initialize our parameters.

Neural Networks for MNIST

Resources

For next question you will use a lot of PyTorch. In Section materials (Week 6) there is a notebook that you might find useful. Additionally make use of PyTorch Documentation, when needed.
If you do not have access to GPU, you might find Google Colaboratory useful. It allows you to use a cloud GPU for free. To enable it make sure: "Runtime" -¿ "Change runtime type" -¿ "Hardware accelerator" is set to "GPU". When submitting please download and submit a `.py` version of your notebook.

A5. In previous homeworks, we used ridge regression for training a classifier for the MNIST data set. Similarly in previous homework, we used logistic regression to distinguish between the digits 2 and 7.
In this problem, we will use PyTorch to build a simple neural network classifier for MNIST to further improve our accuracy.

We will implement two different architectures: a shallow but wide network, and a narrow but deeper network. For both architectures, we use d to refer to the number of input features (in MNIST, $d = 28^2 = 784$), h_i to refer to the dimension of the i-th hidden layer and k for the number of target classes (in MNIST, $k = 10$). For the non-linear activation, use ReLU. Recall from lecture that

$$\text{ReLU}(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}.$$

Weight Initialization

Consider a weight matrix $W \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^n$. Note that here m refers to the input dimension and n to the output dimension of the transformation $x \mapsto Wx + b$. Define $\alpha = \frac{1}{\sqrt{m}}$. Initialize all your weight matrices and biases according to $\text{Unif}(-\alpha, \alpha)$.

Training

For this assignment, use the Adam optimizer from `torch.optim`. Adam is a more advanced form of gradient descent that combines momentum and learning rate scaling. It often converges faster than regular gradient descent in practice. You can use either Gradient Descent or any form of Stochastic Gradient Descent. Note that you are still using Adam, but might pass either the full data, a single datapoint or a batch of data to it. Use cross entropy for the loss function and ReLU for the non-linearity.

Implementing the Neural Networks

a. *[10 points]* Let $W_0 \in \mathbb{R}^{h \times d}$, $b_0 \in \mathbb{R}^h$, $W_1 \in \mathbb{R}^{k \times h}$, $b_1 \in \mathbb{R}^k$ and $\sigma(z) \colon \mathbb{R} \to \mathbb{R}$ some non-linear activation function applied element-wise. Given some $x \in \mathbb{R}^d$, the forward pass of the wide, shallow network can be formulated as:

$$\mathcal{F}_1(x) \coloneqq W_1 \sigma(W_0 x + b_0) + b_1$$

Use $h = 64$ for the number of hidden units and choose an appropriate learning rate. Train the network until it reaches 99% accuracy on the training data and provide a training plot (loss vs. epoch). Finally evaluate the model on the test data and report both the accuracy and the loss.

Solution:

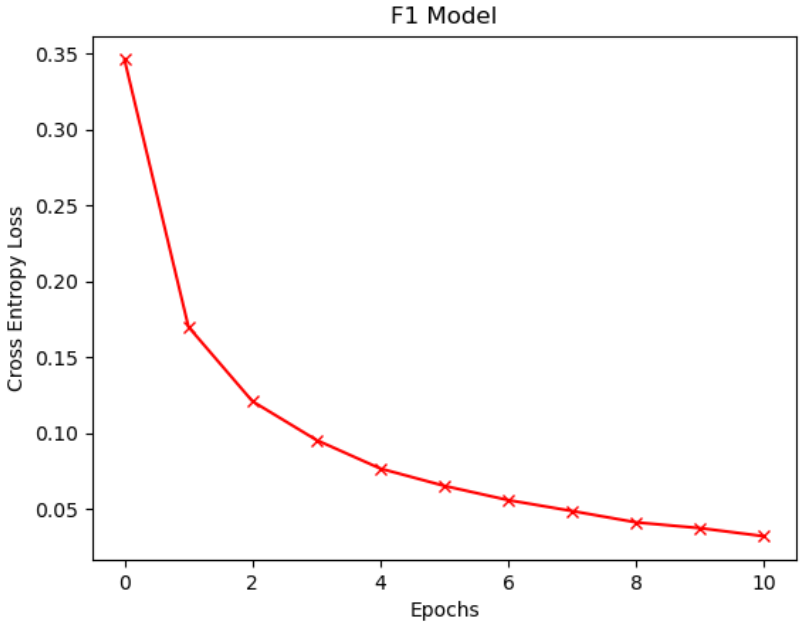


Figure: Training plot of $\mathcal{F}_1$ (loss vs epoch)

Training loss of F1 model: **0.03207075647345434**

Test loss of F1 model: **0.0001**

Test accuracy of F1 model: **0.9747**$\approx$ **97.5**%

b. *[10 points]* Let $W_0 \in \mathbb{R}^{h_0 \times d}$, $b_0 \in \mathbb{R}^{h_0}$, $W_1 \in \mathbb{R}^{h_1 \times h_0}$, $b_1 \in \mathbb{R}^{h_1}$, $W_2 \in \mathbb{R}^{k \times h_1}$, $b_2 \in \mathbb{R}^k$ and $\sigma(z) : \mathbb{R} \to \mathbb{R}$ some non-linear activation function. Given some $x \in \mathbb{R}^d$, the forward pass of the network can be formulated as:

$$\mathcal{F}_2(x) \coloneqq W_2\sigma(W_1\sigma(W_0x + b_0) + b_1) + b_2$$

Use $h_0 = h_1 = 32$ and perform the same steps as in part a.

Solution:

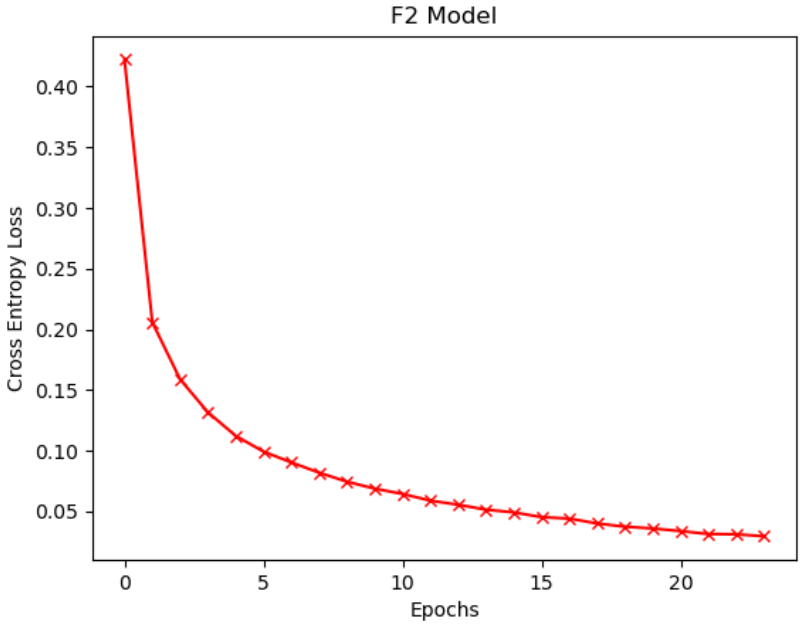


Figure: Training plot of $\mathcal{F}_1$ (loss vs epoch)

Training loss of F2 model: **0.029572819789908437**

Test loss of F2 model: **0.0002**

Test accuracy of F2 model: **0.9665** $\approx$ **96.7**%

---

c. *[5 points]* Compute the total number of parameters of each network and report them. Then compare the number of parameters as well as the test accuracies the networks achieved. Is one of the approaches (wide, shallow vs. narrow, deeper) better than the other? Give an intuition for why or why not.

Solution: There are $\boxed{50890}$ parameters in the $\mathcal{F}_1$ model and $\boxed{26506}$ parameters in the $\mathcal{F}_2$ model. There is very little difference in the test accuracies the networks achieved with $\mathcal{F}_1$ slightly outperforming the $\mathcal{F}_2$ model. The deep network ($\mathcal{F}_2$) has roughly half the number of parameters so it achieves same level of test accuracy while using fewer parameters and so it seems to be the better one. The deep network has more hidden layers which allows it to learn more from the data and so it takes less time to train. The deep model is less computationally expensive and achieves same level of performance.

Administrative

A6.

a. *[2 points]* About how many hours did you spend on this homework? There is no right or wrong answer :)

---

15 hours
