Homework #3

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

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: Decrease the bandwidth σ to get a better fit because you are decreasing the bias. Bias-variance trade off means you need to be careful how much to decrease the bias before we start overfitting.

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, the gradient can either blow up or go to zero, so we use SGD, but we could use for something else too.

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, we initialize the weights with a random generator.

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 use non-linear activtion in between linear hidden layers to improve the model's accuracy by learning non-linear boundaries as well.

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 complexity of backwards and forward pass is the same up to a constant.

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

Solution: False, never say never. While NN are very extensible and a good choice for most circumstances, there are some circumstanes in which another type of Machine Learning may be better used - something like determining the price of a house given paramters such as # of bedrooms, square footage, # of bathrooms, etc. We want these paramters to be defined by us not some mystery parameter defined by the NN.

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: When $w = \begin{bmatrix} -1 & 2 \end{bmatrix}^\top$ and $b = 2$ then

$$w^\top x + b = 0$$

$$\begin{bmatrix} -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -2$$

$$-x_1 + 2x_2 = -2$$

$$x_2 = \frac{1}{2}x_1 - 1$$

Thus, the hyperplane looks like:

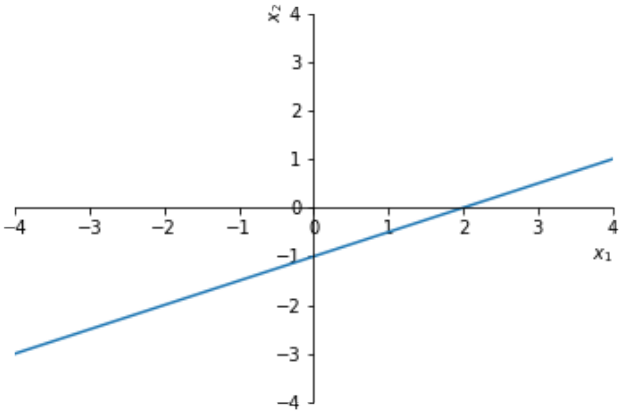

Figure 1: hyperplane x when $w = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

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: When $w = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ and $b = 0$, then the hyperplane x is:

$$x_1 + x_2 + x_3 = 0$$

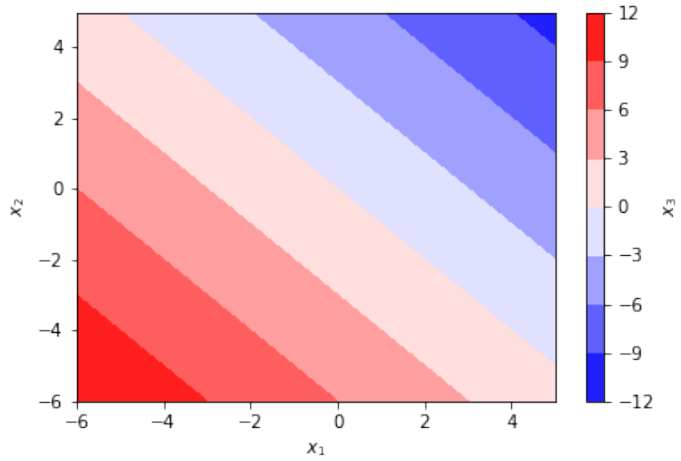

Figure 2: hyperplane x when $w = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$

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:

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

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 $w^\top \widetilde{x}_0$?)

Proof. Let $\tilde{x_0}$ be a minimizer of x_0, or the orthogonal projection of x_0 onto the hyperplane x which is parallel to w. Then,

$$\begin{aligned} x_0 - \widetilde{x_0} &= \frac{w}{\|w\|_2}\|x_0 - \widetilde{x_0}\| \\ |w^\top(x_0 - \widetilde{x_0})| &= \left|\frac{w^\top w}{\|w\|_2}\|x_0 - \widetilde{x_0}\|\right| \text{ multiplying both sides by } w^\top \\ \|x_0 - \widetilde{x_0}\| &= \frac{\|w\|_2}{\|w\|_2^2}\left|w^\top(x_0 - \widetilde{x_0})\right| \\ &= \frac{w^\top x_0 - w^\top \widetilde{x_0}}{\|w\|_2} \\ &= \frac{w^\top x_0 + b}{\|w\|_2} \text{ since } (w^\top \widetilde{x_0} + b = 0) \end{aligned}$$

Since $\widetilde{x_0}$ was the minimizer, the for all $x \in \mathbb{R}^n$ the minimum is

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

□

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:

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

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

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

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

Proof. We will show

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

Let's begin with the LHS:

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

Recall the Taylor series expansion of $e^y = \sum_{i=0}^{\infty} \frac{y^i}{i!}$. So we can use this rule to substitute the summation out (let $y = xx'$).

$$\begin{aligned}
&= e^{-(x^2+x'^2)/2} e^{xx'} \\
&= e^{-(x^2-2xx'+x'^2)/2} \\
&= e^{-(x-x')^2/2}
\end{aligned}$$

Thus we have shown that LHS = RHS. And, the Kernel, $K(x, x') = e^{-\frac{(x-x')^2}{2}}$ is the kernel function for this feature map. □

B1.

Intro to sample complexity

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 $1 - \epsilon \leq \mathrm{e}^{-\epsilon}$.)

Proof. For some $f \in \mathcal{F}$, we have $R(f) > \epsilon$. We will show that

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

We will show the probability of the empirical risk, $\widehat{R}$ being zero is small. The following lines are equivalent:

$$\widehat{R}_n(f) = 0$$

$$0 = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(f(x_i) \neq y_i)$$

$$\forall i \in \{1, 2, \ldots, n\}, \ f(x_i) = y_i$$

Then,

$$\begin{aligned}
\mathbb{P}\left[\widehat{R}_n(f)=0\right] &= \mathbb{P}\left[f(x_i)=y_i \mid \forall i \in \{1,2,\ldots,n\}\right] \\
&= \mathbb{P}\left[(f(x_1)=y_1)\cdots(f(x_n)=y_n)\right] \\
&= \mathbb{P}[(f(x_1)=y_1)]\cdots\mathbb{P}[(f(x_n)=y_n)] \\
&= \Pi_{i=1}^{n}\mathbb{P}(f(x_i)=y_i) \\
&= \Pi_{i=1}^{n}(1-\mathbb{P}(f(x_i)\neq y_i)) \\
&= \mathbb{P}(1-(f(x_i)\neq y_i))^n \text{ since i.i.d.} \\
&= (1-\mathbb{E}\mathbf{1}(f(x_i)\neq y_i))^n \\
&\leq (1-\epsilon)^n \text{ where } \epsilon \leq R(f) \\
&\leq 1-\epsilon^n \leq \mathrm{e}^{-n\epsilon} \text{ from hint}
\end{aligned}$$

Hence, we have shown that

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

□

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

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

Proof. Recall,

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

and

$$\widehat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(f(x_i) \neq y_i) = 0$$

□

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

Skipped.

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

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

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

Context: Note that among a larger number of functions $\mathcal{F}$ there is more likely to exist an $\widehat{f}$ such that $\widehat{R}_n(\widehat{f}) = 0$. However, this increased flexibility comes at the cost of a worse guarantee on the true error reflected in the larger $|\mathcal{F}|$. This trade-off quantifies how we can choose function classes $\mathcal{F}$ that over fit. This sample complexity result is remarkable because it depends just on the number of functions in $\mathcal{F}$, not what they look like. This is among the simplest results among a rich literature known as 'Statistical Learning Theory'. Using a similar strategy, one can use Hoeffding's inequality to obtain a generalization bound when $\widehat{R}_n(\widehat{f}) \neq 0$.
Skipped

B2.

Perceptron

... skipping this problem...

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?

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)?

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.

What to Submit:

- For (a) a single step of subgradient descent
- An answer to the question proposed in part (b).
- An explanation as described in part (c).

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.

   See code.

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: Figure 3 is the plot of loss vs epochs for each neural network defined above for the training (solid) and test (dashed) sets using Mean Square Error (MSE).

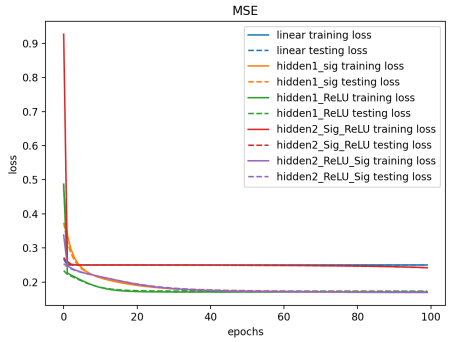

Figure 3: MSE training loss (solid) and test loss (dashed) vs. epochs.

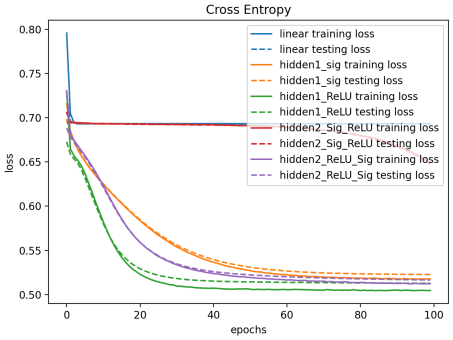

Figure 4: Cross Entropy training loss (solid) and test loss (dashed) vs. epochs.

Figure 4 is the plot of loss vs epochs for each neural network defined above for the training (solid) and test (dashed) sets using cross entropy.

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: The best model according to MSE is ReLU Sigmoid with two hidden layers, while for the CE loss says that ReLU is the best model (one hidden layer). This actually changes when I played with the learning rate (for smaller learning rates < 0.1, I got ReLU; but for 0.1, I got ReLU Sigmoid, like the MSE). So there is that small caveat but the loss function for larger learning rates was not pretty - I decided to go with a smaller learning rate since we give it 100 epochs Figure 5 is the result of `plot_model_guesses` for MSE and Fig. 6 for cross entropy. Either way both CE and MSE tell us that ReLU is the best to use whether after the first hidden layer or alone.

Accuracy MSE = 0.748 Accuracy CE = 0.7478

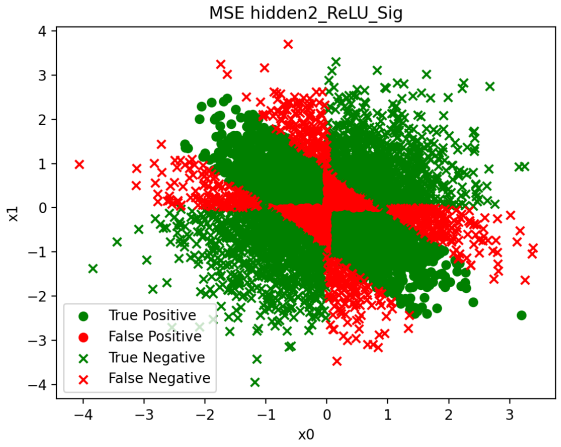

Figure 5: MSE model guesses

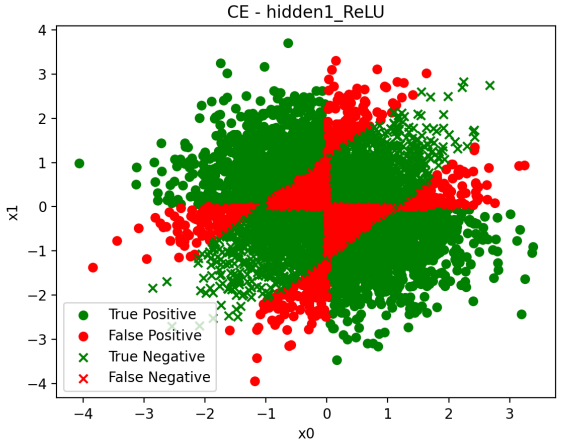

Figure 6: Cross Entropy model guesses

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: Yes, there is a gap in the loss functions of MSE and CE. In general we should be using cross entropy for classification problems like this but MSE is fine and still gives us high accuracy (75%, similar to CE) and acceptable loss function. I played with the learning rate and the gap still existed just at different losses. The difference may arise from the difference in the definition of MSE and CE (and that MSE does better with Gaussian distributions) because the optimizer and the models are otherwise the same (except that cross entropy uses softmax for the models).

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: Training plot of loss vs. epoch for F1 model in Figure 7.

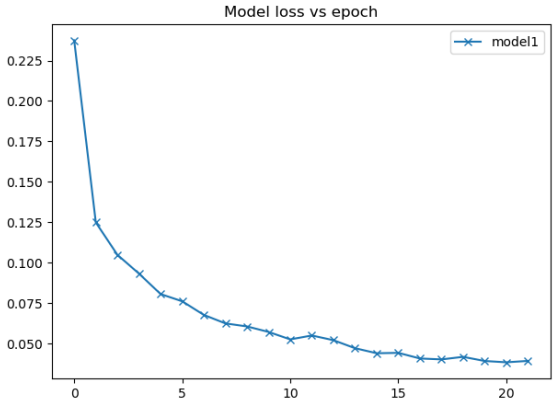

Figure 7: Loss vs epoch for the training set for F1 model.

For the test set:
F1 model: Loss = 6.3526e-08, Accuracy = 0.97

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: Training plot of loss vs. epoch for F2 model in Figure 8.

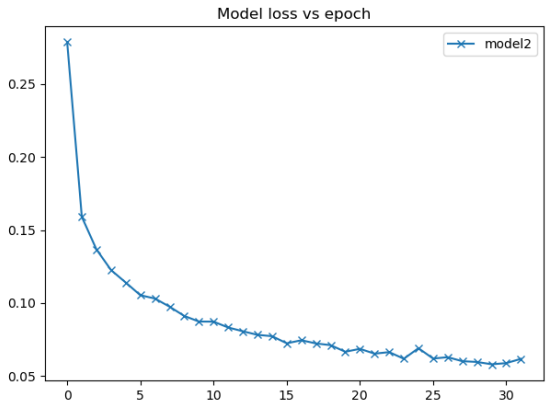

Figure 8: Loss vs epoch for the training set for F2 model.

For the test set:
F2 model: Loss = 0.00089, Accuracy = 0.96

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: Number of parameters: F1 = 50890; F2 = 26506.
Test accuracy: F1 = 0.97; F2 = 0.96

Wide and shallow is better. I guess maybe the deep but narrow network might be missing out on some semi-significant parameters. Even though it is not as good as the wide and shallo (but still pretty good), it is simpler to understand because it has less variables or parameters to consider.

Administrative

A6.

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

24 hours - three straight days of doing no other work :(, too long - code took forever because I always make a thousand dumb mistakes as well as struggling to figure out what to do... but I really enjoyed A2 and A3.