Midterm Exam I- Solutions
ECE 685D– Introduction to Deep Learning
Fall 2023

Instructor: Prof. Vahid Tarokh
ECE Department, Duke University

Oct 4^{th} 2023
10:05 AM - 11:20 AM
(Exam duration: 75 minutes)

Name: ______________________________
Duke ID: ______________________________

This exam contains 10 pages and 9 questions. This exam has 105points of which 5 are bonus points. This is a closed-book exam. No exam aids are allowed. You are not allowed to communicate with others.

Distribution of Marks

Question	Points	Score
1	2	
2	2	
3	3	
4	3	
5	10	
6	5	
7	15	
8	30	
9	35	
Total:	105	

For each of the following questions, circle the letter of your choice. Each question has AT LEAST one correct option unless explicitly mentioned.

1. (2 points) In a logistic regression model, the decision boundary can be ...
   (a) linear
   (b) non-linear
   (c) both (a) and (b)
   Answer: (a)

2. (2 points) Among these commonly-used CNN layers, what is the least computationally complex in terms of floating point operations?
   (a) Conv layer (convolution operation + bias addition)
   (b) Average pooling
   (c) Max pooling
   (d) Batch Normalization
   Answer: (c)

3. (3 points) Which of the following optimization methods uses first-order momentum?
   (a) RMSProp
   (b) Gauss-Newton
   (c) Adam
   (d) Stochastic Gradient Descent
   Answer: (c)

4. (3 points) Making your network deeper by adding more parametrized layers will always...
   (a) reduce the training loss.
   (b) improve the performance on unseen data.
   (c) slow down training and inference speed.
   (d) both (a) and (b)
   Answer: (c)

5. (10 points) Please complete the Pytorch code below by providing the missing lines necessary to train the CNN model (do not need to worry about syntax)

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data.DataLoader as dataloader

# Load the data
train_data = torch.load("train_data.pt")
trainloader = dataloader(train_data)
test_data = torch.load("test_data.pt")
trainloader = dataloader(test_data)

# Define the model
model = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=3),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Conv2d(64, 128, kernel_size=3),
    nn.ReLU(),
    nn.MaxPool2d(2),
    nn.Flatten(),
    nn.Linear(128 * 7 * 7, 1000),
    nn.ReLU(),
    nn.Linear(1000, 10),
)

# Train the model
# Write in your code here:
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(30):
    for i, data in in enumerate(trainloader):
        input, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

# Evaluate the model
accuracy = model.evaluate(test_data)
print("Accuracy:", accuracy)
```

6. (5 points) A function $f(\cdot) : \mathbb{R} \to \mathbb{R}$ is said to be piece-wise constant if for some non-negative integer n if there exists increasing real numbers

$$-\infty = a_0 < a_1 < a_2 < \cdots < a_n < a_{n+1} = \infty$$

and real numbers $c_0, c_1, \cdots, c_n$ such that $f(x) = c_j$ for $a_j < x \leq a_{j+1}$ for $j = 0, 1, \cdots, n-1$ and $f(x) = c_n$ for $a_n < x$.

Can a piece-wise constant function be suitable for use as an activation function in a neural network? Please explain your answer.

Answer: No, since it is mostly non-differential. Hence, it cannot backpropagate weights to previous layers.

7. Consider a binary logistic regression problem as follows:

$$p_i = p(y_i = 1|\mathbf{x_i}) = \sigma(\mathbf{w}^T\mathbf{x}_i + b),\ \forall i \in \{1, 2\} \tag{1}$$

where $y \in \{1, 0\}$, $\mathbf{x} \in \mathbb{R}^{2\times 1}$, $\mathbf{w} \in \mathbb{R}^{2\times 1}$, $b \in \mathbb{R}$, and $\sigma(\cdot)$ is the sigmoid function, given as: $\sigma(a) = 1/(1 + e^{-a})$. Given a dataset with two data points $\{\mathbf{x}_1, y_1\} = \{(1, 0)^T, 2\}$, $\{\mathbf{x}_2, y_2\} = \{(1, 1)^T, 3\}$.

The loss function is

$$\mathcal{L} = -\sum_{i=1}^{2} y_i \log(p_i) + (1 - y_i)\log(1 - p_i) \tag{2}$$

The initial value of $\mathbf{w}$ is $\mathbf{w}_1 = (1, 1)^T$.

The initial value of b is $b_1 = 0.5$.

(a) (10 points) Write out the gradient of the loss function $\mathcal{L}$ with respect to the weight $\mathbf{w}$ and bias b explicitly.

(b) (5 points) Perform one step of Nesterov's accelerated gradient descent method with $\beta = 0.5$ on $\mathbf{w}$ and b using the dataset formed with two data points $\{\mathbf{x}_i, y_i\}_{i=1}^2$. **Use the definition of Nesterov's Accelerated Gradient Descent in the lecture notes, as shown below** to calculate $t_2, \mathbf{w}_2, b_2$.

Algorithm 1 Nesterov's Accelerated Gradient Descent

First define the following sequences: $\lambda_0 = 0$, $\lambda_k = (1 + \sqrt{1 + 4\lambda_{k-1}^2})/2$, $\gamma_k = (1 - \lambda_k)/\lambda_{k+1}$
for $k = 1, 2, ...$ **do**
  $\mathbf{t_{k+1}} = \mathbf{w_k} - \nabla\mathcal{L}(\mathbf{w_k})/\beta$
  $\mathbf{w_{k+1}} = (1 - \gamma_k)\mathbf{t_{k+1}} + \gamma_k\mathbf{t_k}$
end for

(a)

$$\begin{aligned}
\frac{\partial L}{\partial w} &= -\sum_{i=1}^{2} \frac{y_i}{p_i}\frac{\partial p_i}{\partial w} + \frac{1 - y_i}{1 - p_i}\frac{\partial 1 - p_i}{\partial w} \\
&= -\sum_{i=1}^{2} \frac{y_i}{p_i} p_i(1 - p_i)x_i - \frac{1 - y_i}{1 - p_i} p_i(1 - p_i)x_i \\
&= -\sum_{i=1}^{2} (y_i - p_i)x_i
\end{aligned}$$

$$\begin{aligned}\frac{\partial L}{\partial b} &= -\sum_{i=1}^{2}\frac{y_i}{p_i}\frac{\partial p_i}{\partial b} + \frac{1-y_i}{1-p_i}\frac{\partial 1-p_i}{\partial b} \\ &= -\sum_{i=1}^{2}\frac{y_i}{p_i}p_i(1-p_i) - \frac{1-y_i}{1-p_i}p_i(1-p_i) \\ &= -\sum_{i=1}^{2}(y_i - p_i)\end{aligned}$$

(b) $\lambda_0 = 0, \lambda_1 = 1, \gamma_1 = 0$

We have: $\nabla L(w, b) = \left(\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}\right)$

From Nesterove Algorithm, we obtain: $t_2 = (w, b) - \nabla L(w, b)/\beta$

Hence: $w_2 = (1 - \gamma_1)t_2[0] + \gamma_1 t_1[0] = t_2[0]$ and $b_2 = (1 - \gamma_1)t_2[1] + \gamma_1 t_1[1] = t_2[1]$

8. Consider an RGB image $X = [X_0, X_1, X_2]$ with three channels, and given as follows:

$$X_0 = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 2 & 0 & 1 \\ 2 & 1 & 0 & 1 \end{bmatrix}, X_1 = \begin{bmatrix} 2 & 2 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}, X_2 = \begin{bmatrix} 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 \\ 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}. \tag{3}$$

This image is passed through the convolutional filter with the weights $W = [W_0, W_1, W_2]$ of size $3 \times 3 \times 3$, step size of 1, and is given as follows:

$$W_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -1 \end{bmatrix}, W_1 = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 0 & -1 \\ 0 & -1 & 1 \end{bmatrix}, W_2 = \begin{bmatrix} 0 & 0 & -2 \\ 0 & 1 & 2 \\ -2 & 2 & 0 \end{bmatrix}. \tag{4}$$

The output of the convolutional filter is given as follows:

$$Y = ReLU\left(\sum_{i=0}^{2}(X_i' * W_i) + 2 \times 1_{4\times4}\right) \tag{5}$$

where Y is the output image, X' is the input image after applying zero-padding around the edges (i.e. each channel is converted to a 6×6 matrix such that a row of zeros is added to the top and bottom and a column of zeros is added to the left and right.), $X_i' * W_i$ is the convolution of the i-th channel of X' with the the i-th channel of W, and $1_{4\times4}$ is a 4×4 matrix with all ones.

(a) (20 points) Compute the output Y of the image X.

(b) (5 points) Apply max pooling on non-overlapping 2×2 sub-matrices of the output image and compute the output.

(c) (5 points) Apply average pooling on non-overlapping 2×2 sub-matrices of the output image and compute the output.

Answer:

(a) Here are the results for the non-flipped kernel. Note that we also give full credit for the flipped kernel approach.

$$X_0 * W_0 = \begin{bmatrix} -4 & -4 & -1 & 0 \\ -2 & 2 & -4 & -2 \\ -1 & -4 & -1 & 0 \\ -4 & -2 & 2 & -2 \end{bmatrix} \tag{6}$$

$$X_1 * W_1 = \begin{bmatrix} -2 & 6 & 3 & -1 \\ 4 & 6 & -1 & 4 \\ 1 & -3 & 5 & 7 \\ -1 & 0 & 5 & 2 \end{bmatrix} \tag{7}$$

$$X_2 * W_2 = \begin{bmatrix} 4 & 1 & 4 & -2 \\ 2 & 0 & 4 & 1 \\ 2 & -2 & -4 & 2 \\ 2 & 1 & 2 & 1 \end{bmatrix}. \tag{8}$$

$$Y = ReLU\left(\sum_{i=0}^{2}(X_i' * W_i) + 2 \times 1_{4\times4}\right) = \begin{bmatrix} 0 & 5 & 8 & 0 \\ 6 & 10 & 1 & 5 \\ 4 & 0 & 2 & 11 \\ 0 & 1 & 11 & 3 \end{bmatrix} \tag{9}$$

(b)

$$maxpool(Y) = \begin{bmatrix} 10 & 8 \\ 4 & 11 \end{bmatrix} \tag{10}$$

(c)

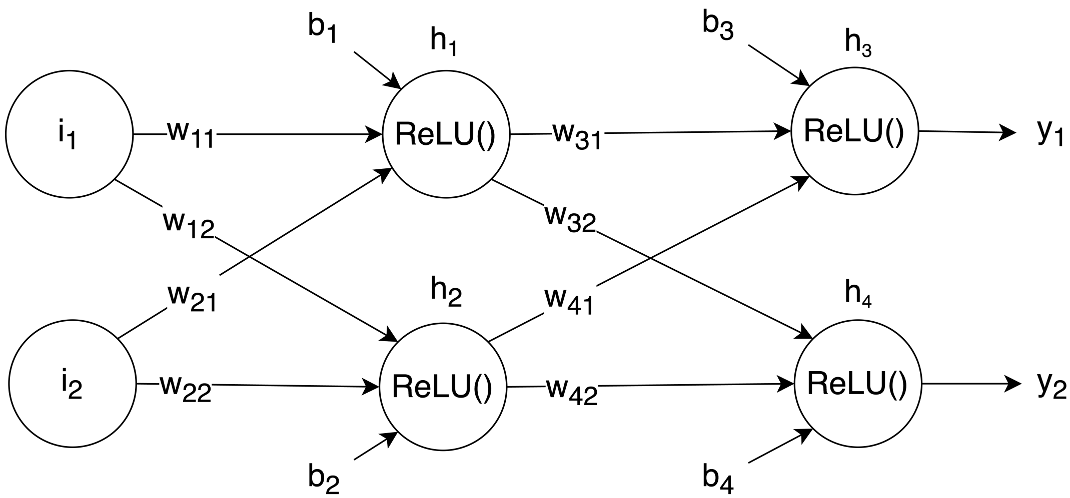
$$avepool(Y) = \begin{bmatrix} 21/4 & 7/2 \\ 5/4 & 27/4 \end{bmatrix} \tag{11}$$

9. Given the following neural network with two input units (i_1, i_2), fully-connected layers and ReLU activations. The weights and bias of hidden units are denoted w and b, with h_1, h_2, h_3, h_4 are ReLU units.

$$h_1 = ReLU(i_1 w_{11} + i_2 w_{21} + b_1) \tag{12}$$

The outputs are denoted as (y_1, y_2), and the ground truth targets are denoted as (t_1, t_2).

$$y_1 = ReLU(h_1 w_{31} + h_2 w_{41} + b_3) \tag{13}$$



The values of the variables are given as follows:

i_1	i_2	w_{11}	w_{12}	w_{21}	w_{22}	w_{31}	w_{32}	w_{41}	w_{42}	b_1	b_2	b_3	b_4	t_1	t_2
1	2	1	0.5	-0.5	1	0.5	-2	-1	0.5	-0.5	-0.5	1	1	2	4

(a) (10 points) Compute the output (y_1, y_2) of the input (i_1, i_2) using the network parameters as specified above (please write down all calculations of the intermediate layers)

(b) (5 points) Compute the mean squared error of the computed output (y_1, y_2) and the target labels (t_1, t_2).

(c) (5 points) Using the calculated MSE above, update the weight w_{31} using gradient descent and backpropagation algorithm with a learning rate of 0.01(write down all your computations).

(d) (5 points) Using the calculated MSE above, update the weight w_{42} using gradient descent and backpropagation algorithm with a learning rate of 0.01(write down all your computations).

(e) (10 points) Using the calculated MSE above, update the weight w_{22} using gradient descent and backpropagation algorithm with a learning rate of 0.01 (write down all your computations).

Answer:

(a) First, $h_1 = 0, h_2 = 2$.
Next, $y_1 = h_3 = ReLU(h_1 w_{31} + h_2 w_{41} + b_3) = 0, y_2 = 2$.

(b) $L_{MSE} = \frac{1}{2}\sum_{i=1}^{2}(y_i - t_i)^2 = 4$

(c) $\frac{\partial L}{\partial w_{31}} = \frac{\partial \frac{1}{2}\sum_{i=1}^{2}(y_i - t_i)^2}{\partial w_{31}} = (y_1 - t_1)\frac{\partial h_3}{w_{31}}$.
Since input of $h_3 < 0$, $\frac{\partial h_3}{w_{31}} = 0$.
Hence, $w_{31} = w_{31} - 0.01\frac{\partial L}{\partial w_{31}} = 0.5 - 0 = 0.5$.

(d) $\frac{\partial L}{\partial w_{42}} = \frac{\partial \frac{1}{2}\sum_{i=1}^{2}(y_i - t_i)^2}{\partial w_{42}} = (y_2 - t_2)\frac{\partial h_4}{w_{42}} = (y_2 - t_2)h_2 = -4$.
Hence, $w_{42} = w_{42} - 0.01\frac{\partial L}{\partial w_{42}} = 0.54$.

(e) $\frac{\partial L}{\partial w_{22}} = (y_1 - t_1)\frac{\partial h_3}{h_2}\frac{\partial h_2}{w_{22}} + (y_2 - t_2)\frac{\partial h_4}{h_2}\frac{\partial h_2}{w_{22}} = -2$.
Hence, $w_{22} = w_{22} - 0.01\frac{\partial L}{\partial w_{22}} = 1.02$.