EECS 442 Practice Exam, Winter 2025

Disclaimer: this exam was previously released as a take-home exam with 6 problems. We include the 4 problems that are relevant here. Thus, some aspects of it may be harder than our in-class version, and length may not be applicable.

Problem 1 *Machine Learning Fundamentals* **(16 points)**

(a) **(4 points)** List two methods to prevent underfitting and overfitting respectively.

Overfitting: inrcease regularization, decrease training time
Underfitting. decrease regularization, increase model complexity, increase training data

(b) **(4 points)** How does increasing or decreasing a linear model's regularization parameter (the λ term in L2 regularization) affect the learned model's complexity?

Higher regularization makes model simpler (low variance, high bias)
Lower regularization makes model more complex (high variance, low bias)

(c) **(4 points)** Explain the roles of training, validation, and testing datasets in the machine learning workflow. Why is it important to separate these datasets?

Training set: train parameters of the model
Validation set: tune hyper-parameters of the model
Testing set: Doing final evaluation of the model. Only use once
It's is important to seperate data to prevent the possible overfit in training data.

(d) **(4 points)** Explain the purpose of cross-entropy loss in classification tasks. Why is it preferred over MSE (mean-squared-error) loss in classification?

Purpose: measue the difference between the predicted probility distributuion and the actual class labes
Why is it prefered:
1. In probaitliy interpretation, it's mathematically equivalent to maximum likelihood estimation
2. Gradient behavior: MSE sufferes from slow learning, while corss-entropy loss gives larger gradients when prediction are incorrect

Problem 2 *Image Filtering* **(16 points)**

Given the original 5×5 grayscale image matrix:

$$\text{Input Matrix} = \begin{bmatrix} 10 & 20 & 20 & 20 & 10 \\ 20 & 30 & 40 & 30 & 20 \\ 20 & 40 & 200 & 40 & 20 \\ 20 & 30 & 40 & 30 & 20 \\ 10 & 20 & 20 & 20 & 10 \end{bmatrix}$$

A noise spike is present at the center of the matrix (200). Your goal is to reduce this noise while preserving other details.

(a) **(4 points) Median Filtering** (3×3 filter window with zero padding) Apply a 3×3 median filter to the entire image using zero-padding to keep the output matrix the same size (5×5) as the input. Please provide the filtered output matrix.

```
0  20 20 20 0
20 20 30 20 20
20 30 40 30 20
20 20 30 20 20
0  20 20 20 0
```

(b) **(4 points) Gaussian Filtering** (3×3 filter window without padding) Apply a Gaussian filter with a 3×3 kernel using the following values (not normalized), without any padding, so that the output matrix size is reduced to 3×3:

$$\text{Gaussian Kernel} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

After filtering, normalize by dividing each pixel value by 16. Please provide the resulting 3×3 output matrix.

10 20 20 20 10		
20 30 40 30 20	610 840 610	305/8 105/2 305/8
20 40 200 40 20	840 1240 610	105/2 155/2 105/2
20 30 40 30 20	610 840 610	305/8 105/2 305/8
10 20 20 20 10		

(c) **(2 points) Is the above Gaussian filter kernel separable?** If so, pleased write the two $1D$ rectangular filter kernels that can be used to reconstruct the 3×3 Gaussian filter. If not, please give the reason.

Yes
v = [1 2 1]^T, u = [1 2 1]
gaussian filter = vu

(d) **(4 points) Bilateral Filtering** (3×3 filter window without padding) Apply a bilateral filter with a 3×3 window centered on each pixel. Use the Gaussian kernel from Question (b) for spatial weighting. For intensity weighting, we will use a simplified scheme that makes computation easier: we will use an *intensity threshold* of 30, which means that pixels with at most an intensity difference of 30 from the center pixel would contribute. For example, if the center pixel is 50, only neighboring pixels within the intensity range of $[20, 80]$ will contribute to the filtered results. More precisely, the equation for this simplified bilateral filter is as follows:

$$I_{\text{filtered}}(x) = \frac{1}{W(x)} \sum_{x' \in \mathcal{N}(x)} I(x') \cdot G_{\sigma_s}(\|x - x'\|) \cdot \delta(I(x), I(x'))$$

where:

- $I_{\text{filtered}}(x)$ is the intensity of the filtered image at pixel x;
- $\mathcal{N}(x)$ is the spatial neighborhood of pixel x;
- $I(x')$ is the intensity of the neighboring pixel x';
- G_{σ_s} is the spatial Gaussian kernel with spatial standard deviation σ_s. For simplification, we just use the Gaussian kernel from Question (b) as an approximation for G_{σ_s};
- $W(x)$ is the normalization factor:

$$W(x) = \sum_{x' \in \mathcal{N}(x)} G_{\sigma_s}(\|x - x'\|) \cdot \delta(I(x), I(x'))$$

- $\delta(I(x), I(x'))$ is the indicator function enforcing the intensity difference threshold:

$$\delta(I(x), I(x')) = \begin{cases} 1 & \text{if } |I(x) - I(x')| \leq 30 \\ 0 & \text{otherwise} \end{cases}$$

This equation combines both spatial and intensity constraints for bilateral filtering, using the given Gaussian kernel and intensity threshold. Please provide the resulting 3×3 output matrix after applying this bilateral filtering using a 3×3 filter window without padding.

27.33	31.43	27.33
31.43	200	31.43
27.33	31.43	27.33

(e) **(2 points)** Compare the output matrices from the three filters. Discuss in a few sentences: which filter best reduced the noise spike while preserving details and explain why.

Median filtering best reduced the noise spike while preserving details. because it completely removed outlier (200) while preserving original structure of surring pixels.
On the other hand, bilateral filtering and Gaussian filtering failed to remove outlier 200.

Problem 3 *Fourier Transform* **(16 points)**

(a) **(2 points)** If we rotate an image clockwise by 90°, what effect does this have on its Fourier transform?

[✓ a) The Fourier transform rotates 90° clockwise.]

b) The Fourier transform rotates 90° counter-clockwise.

c) The Fourier transform rotates 180°.

d) No general statement can be made about the Fourier transform.

(b) **(2 points)** If we scale an image by a factor of 2, doubling its size, what effect does this have on its Fourier transform?

a) The Fourier transform scales by a factor of 2.

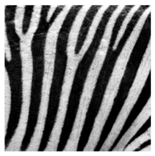
[✓ b) The Fourier transform scales by a factor of $\frac{1}{2}$.]

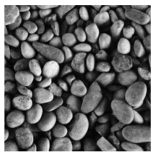
c) The Fourier transform is unaffected.

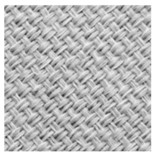
d) No general statement can be made about the Fourier transform.

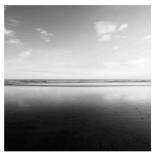
(c) **(12 points)** Please match the following images to their respective Fourier transforms.

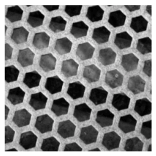
a) 3 b) 1 c) 6 d) 2 e) 4 f) 5

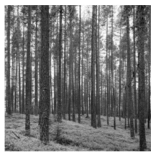

(a)

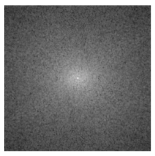

(b)

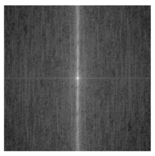

(c)

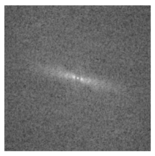

(d)

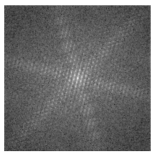

(e)

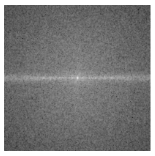

(f)

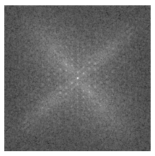

(1)


(2)


(3)


(4)


(5)


(6)

Problem 4 *Backpropagation* **(16 points)**

Recall that a neural network can be represented as a computation graph, enabling us to systematically compute its gradients. For example, Figure 1 is an example of the equation $f(x, y) = x + y$. The corresponding code for the forward and backward of this diagram is also shown below.

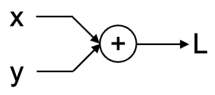


Figure 1: Computation graph for $f(x, y) = x + y$

```
def f(x, y):
    ###forward pass###
    L = x + y

    ###backward pass###
    grad_L = 1
    grad_x = 1 * grad_L
    grad_y = 1 * grad_L
    return L, (grad_x, grad_y)
```

Figure 2 is a computation graph for function $f(a, b, c, d)$.

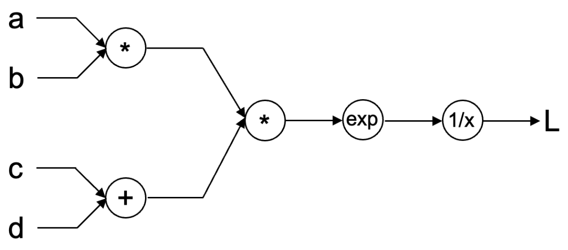


Figure 2: Computation graph

(a) **(2 points)** Please write down the mathematical formula for $f(a, b, c, d)$.

f(a,b,c,d) = 1/exp(ab*(c+d))

(b) **(4 points)** Please implement the code for forward and backward pass of computation graph in (a).

```
def f(a,b,c,d):
        # Forward propogation
        ab = a*b
        c_d = c+d
        ab_cd = ab*cd
        exp_term = np.exp(ab_cd)
        L = 1/exp_term

        # Backward propogation
        grad_exp_term = -1/(exp_term**2)
        grad_ab_cd = grad_exp_term * np.exp(ab_cd)

        grad_ab = grad_ab_cd * c_d
        grad_c_d = grad_ab_cd * ab

        grad_a = grad_ab * b
        grad_b = grad_ab * a

        grad_c = grad_c_d
        grad_d = grad_c_d

        return L, (grad_a, grad_b, grad_c, grad_d)
```

(c) **(7 points)** Please draw the computation graph and implement the code for forward and backward pass of function $f(x, y, z) = \frac{1}{e^{2x-1/x+3yz}+1}$.

Note: Please use the following operations: $+, \times, -, +1, \times(-1), exp, \frac{1}{x}$.

```
def f(x,y,z):

#forward pass
x_2 = 2*x
one_x = 1/x
neg_one_x = -one_x

yz3 = 3*y*z
sums = x_2 + neg_one_x + yz3

exp_term = np.exp(sums)
exp_term_1 = exp_term + 1

L = 1/exp_term_1

#backward pass

grad_L = 1
grad_exp_term_1 = -1/(exp_term_1 ** 2) * grad_L
grad_exp_term = grad_exp_term_1 * 1

grad_sums = grad_exp_term * np.exp(sums)

grad_yz3 = grad_sums * 1
grad_neg_one_x = grad_sums * 1
grad_x_2 = grad_sums * 1

grad_one_x = grad_neg_one_x * (-1)
grad_x = grad_one_x * (-1) / (x ** 2) + grad_x_2 * 2

grad_y = grad_yz3 * 3 * z

grad_z = grad_yz3 * 3 * y
return L, (grad_x, grad_y, grad_z)
```

(d) **(3 points)** Why might Stochastic Gradient Descent (SGD) be more effective than Batch Gradient Descent (BGD) when training neural networks?

Stochastic Gradient Descent might be more effective than Batch Gradient Descent because:
1. Since it update gradient after each data point instead of waiting for the entire dataset (as BGD), it can start making progress toward optimal solution more quickly
2. The randomness in SGD helps escape local minima and saddle points, leading to a more robuts model that generalizes better to unseen data
3. SGD requires less memory because it update parameters using a single sample at a time, instead of using entire data set at a same time as BGD
4. In real-world applications, such as recommendation system in Netflix or Amazon, data arrives in a continuous stream. SGD allows training to be preformed online without need to store and process entire dataset at once