HW1

January 14, 2025

[1]:
```
"Tom Deng, RIN 662007936"
```

[1]: 'Tom Deng, RIN 662007936'

[11]:
```
#Problem 1
"a) x1 has 5 features, x2 has 4 features"
```

[11]: 'a) x1 has 5 features, x2 has 4 features'

[3]:
```
#b)
import numpy as np

# Define the vectors
x1 = np.array([2, 3, 4, 8, 9])
x2 = np.array([2, -3, -4, 89])

# Calculate norms for x1
L1_norm_x1 = np.sum(np.abs(x1))
L2_norm_x1 = np.sqrt(np.sum(x1**2))
L_inf_norm_x1 = np.max(np.abs(x1))

# Calculate norms for x2
L1_norm_x2 = np.sum(np.abs(x2))
L2_norm_x2 = np.sqrt(np.sum(x2**2))
L_inf_norm_x2 = np.max(np.abs(x2))

# Print results
print("Vector x1:")
print(f"L1 Norm: {L1_norm_x1}")
print(f"L2 Norm: {L2_norm_x1}")
print(f"LâĹđ Norm: {L_inf_norm_x1}")

print("\nVector x2:")
print(f"L1 Norm: {L1_norm_x2}")
print(f"L2 Norm: {L2_norm_x2}")
print(f"LâĹđ Norm: {L_inf_norm_x2}")
```

```
Vector x1:
```

```
L1 Norm: 26
L2 Norm: 13.19090595827292
LâĹđ Norm: 9

Vector x2:
L1 Norm: 98
L2 Norm: 89.16277250063504
LâĹđ Norm: 89
```

[4]:
```
#Problem 2

input_height, input_width, input_channels = 1024, 1024, 3
output_height, output_width, output_channels = 64, 64, 3

# Calculate input vector length
input_vector_length = input_height * input_width * input_channels

# Calculate output vector length
output_vector_length = output_height * output_width * output_channels

# Matrix W dimensions and vector b size
W_elements = output_vector_length * input_vector_length
b_elements = output_vector_length

# Print results
print("Input Vector Length:", input_vector_length)
print("Output Vector Length:", output_vector_length)
print("Number of elements in Matrix W:", W_elements)
print("Number of elements in Vector b:", b_elements)
```

```
Input Vector Length: 3145728
Output Vector Length: 12288
Number of elements in Matrix W: 38654705664
Number of elements in Vector b: 12288
```

[5]:
```
#Problem 3
import numpy as np

# Define the matrix W
W = np.array([[1, -1], [2, 0]])

# Calculate norms
L1_norm = np.max(np.sum(np.abs(W), axis=0))  # Maximum column sum
L2_norm = np.linalg.norm(W, 2)              # Spectral norm (2-norm)
L_inf_norm = np.max(np.sum(np.abs(W), axis=1))  # Maximum row sum
Frobenius_norm = np.sqrt(np.sum(W**2))      # Frobenius norm
```

```
# Print results
print(f"L1 Norm: {L1_norm}")
print(f"L2 Norm: {L2_norm}")
print(f"LâĹđ Norm: {L_inf_norm}")
print(f"Frobenius Norm: {Frobenius_norm}")
```

```
L1 Norm: 3
L2 Norm: 2.2882456112707374
LâĹđ Norm: 2
Frobenius Norm: 2.449489742783178
```

[6]:
```
#Problem 4
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data  # Features (4 columns: sepal length, sepal width, petal length,␣
↪petal width)
y = iris.target  # Target (class labels)

# Choose 3 out of the 4 features (e.g., 0, 1, 2)
X_selected = X[:, :3]  # Selecting the first 3 features

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.
↪2, random_state=42)

# Initialize and train a Logistic Regression model
model = LogisticRegression(max_iter=200)  # Increase max_iter for convergence
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f"Accuracy using 3 features: {accuracy * 100:.2f}%")
```

```
Accuracy using 3 features: 100.00%
```

[8]:
```
#Problem 5
import numpy as np
```

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Generate data: integers from 1 to 100 and their 7th root
X = np.arange(1, 101).reshape(-1, 1)  # Input data (1 to 100)
y = X**(1/7)  # Target data (7th root)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Define the neural network model
model = Sequential([
    Input(shape=(1,)),                    # Input layer explicitly defined
    Dense(10, activation='relu'),         # Hidden layer with 10 neurons
    Dense(10, activation='relu'),         # Another hidden layer
    Dense(1)                              # Output layer
])

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
history = model.fit(X_train, y_train, epochs=500, verbose=0)

# Evaluate the model on the test data
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on Test Data: {mse:.4f}")

# Test the model with a few examples
test_values = np.array([[10], [50], [100]])
predictions = model.predict(test_values)
for i, val in enumerate(test_values.flatten()):
    print(f"7th root of {val}: True = {val**(1/7):.4f}, Predicted =␣
 ↪{predictions[i][0]:.4f}")
```

```
1/1 âŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄ 0s 142ms/step
Mean Squared Error on Test Data: 0.0863
1/1 âŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄ 0s 158ms/step
7th root of 10: True = 1.3895, Predicted = 0.9867
7th root of 50: True = 1.7487, Predicted = 1.5464
7th root of 100: True = 1.9307, Predicted = 2.2461
```

[10]:
```
#Problem 6
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from sklearn.metrics import mean_squared_error

# Load the Iris dataset
iris = load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
data['target'] = iris.target

# Extract input (sepal length) and output (petal length)
X = data['sepal length (cm)'].values.reshape(-1, 1)  # Input: Sepal length
y = data['petal length (cm)'].values  # Output: Petal length

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Scale the input features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the neural network model
model = Sequential([
    Input(shape=(1,)),                      # Input layer explicitly defined
    Dense(10, activation='relu'),           # Hidden layer with 10 neurons
    Dense(10, activation='relu'),           # Hidden layer with 10 neurons
    Dense(1)                                # Output layer with 1 neuron␣
↪(regression)
])

# Compile the model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Train the model
history = model.fit(X_train, y_train, epochs=300, verbose=0)

# Evaluate the model
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error on Test Data: {mse:.4f}")
```

```
# Test the model with a few examples
test_values = np.array([[5.0], [6.5], [7.2]])
test_values_scaled = scaler.transform(test_values)
predictions = model.predict(test_values_scaled)
for i, val in enumerate(test_values.flatten()):
    print(f"Sepal length: {val:.1f}, Predicted Petal length: {predictions[i][0]:.
↪4f}")
```

```
1/1 âŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄ 0s 173ms/step
Mean Squared Error on Test Data: 0.4782
1/1 âŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄâŤĄ 0s 147ms/step
Sepal length: 5.0, Predicted Petal length: 1.9362
Sepal length: 6.5, Predicted Petal length: 5.0833
Sepal length: 7.2, Predicted Petal length: 5.8491
```

[ ]:

[ ]: