Exp 1 (Keras)

```
import numpy as np
import keras as keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Flatten

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Preprocess the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Defining threshold for Softmax Function
#keras.activations.softmax(x_train, axis=-1)

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# Fit the model
model.fit(
    x_train, y_train,
    epochs=5, batch_size=32,
    validation_data=(x_test, y_test)
)

# Evaluate Model
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print(f'Test loss at each epoch: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')

# Make predictions
```

```
predictions = model.predict(x_test[:5])
predicted_labels = np.argmax(predictions, axis=1)
print('Predicted Labels:', predicted_labels)
```

Exp 2 (Linear)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics

# input dataset
x = np.array([1, 2, 3, 4, 5, 6, 7])
y = np.array([2, 3, 4, 5, 6, 7, 8])


# def linear regression

def linearReg(X, Y):
    x_mean = np.mean(X)
    y_mean = np.mean(Y)

    num = np.dot(X - x_mean, y - y_mean)
    deno = np.dot(x - x_mean, x - x_mean)
    # calculate the slope
    m = num / deno

    #calculate the intercept
    b = y_mean - m * x_mean

    return m, b

def predict(X, m, b):
    y_predicted = m * X + b
    return y_predicted

m, b = linearReg(x, y)

y_pred = predict(x, m, b)

print("Accuracy: ", metrics.accuracy_score(y, y_pred=y_pred))

plt.scatter(x, y, color='blue')
plt.plot(x, y_pred, color='black')
plt.show()
```

Exp 3 (logistic)

```python
import numpy as np
import pandas as pd
from sklearn import metrics

# x1 = [9.2, 3.4, 2.1, 0.3, 5.6]
# x2 = [8.1, 2.4, 1.8, 0.5, 4.3]
# y = [1, 1, 0, 0, 1]

x1 = [9.9, 10.7, 0.34, 7.4, 1.6]
x2 = [8.4, 10.9, 0.2, 8.32, 2.56]
y = [1, 1, 0, 1, 0]

n = 5

b0 = 0
b1 = 0
b2 = 0

s = 0.8

p = []
pc = []

for i in range(n):
    p.append(1 / (1 + np.exp(-(b0 + b1 * x1[i] + b2 * x2[i]))))

    b0 = b0 + s * (y[i] - p[i]) * p[i] * (y[i] - p[i]) * 1
    b1 = b1 + s * (y[i] - p[i]) * p[i] * (y[i] - p[i]) * x1[i]
    b2 = b2 + s * (y[i] - p[i]) * p[i] * (y[i] - p[i]) * x2[i]

    if(p[i] < s):
        pc.append(0)
    else:
        pc.append(1)

print(p)
print(pc)

y_true = np.array(y)
y_pred = np.array(pc)

print("Accuracy: ", metrics.accuracy_score(y_true, y_pred))
```

Exp 4 (SVM)

```
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn import svm

cancer_data = datasets.load_breast_cancer()

print(cancer_data.target)

x_train, x_test, y_train, y_test = train_test_split(cancer_data.data,
cancer_data.target, random_state=42, test_size=0.4)

cls = svm.SVC(kernel='linear')

cls.fit(x_train, y_train)

pred = cls.predict(x_test)

print("accuracy: ", metrics.accuracy_score(y_test, y_pred=pred))
print("precision: ", metrics.precision_score(y_test, y_pred=pred))
print("recall: ", metrics.recall_score(y_test, y_pred=pred))
print("Classification Report: ", metrics.classification_report(y_test,
y_pred=pred))
```

Exp 5 (Hebbian network)

```
import numpy as np

# For OR gate
x1 = [1, 1, -1, -1]
x2 = [1, -1, 1, -1]
y = [1, 1, 1, -1]
b = [1, 1, 1, 1]
# Now calculate the Delta x1, x2 and y

dx1 = []
dx2 = []
db = []

for i in range (0,4):
    dx1.append(x1[i] * y[i])
    dx2.append(x2[i] * y[i])
```

```
    db.append(y[i])


w1 = w2 = b0 = 0

w1n = []
w2n = []
bn = []

for i in range(0,4):
    w1n.append(w1 + dx1[i])
    w1 = w1n[i]
    w2n.append(w2 + dx2[i])
    w2 = w2n[i]
    bn.append(b0 + db[i])
    b0 = bn[i]

print("dw1\tdw2\tdb\t\tw1n\tw2n\tbn")

for i in range(0,4):
    print(str(dx1[i]) + "\t" + str(dx2[i]) + "\t" + str(db[i]) + "\t\t" +
str(w1n[i]) + "\t" + str(w2n[i]) + "\t" + str(bn[i]))
```

Exp 7 (MP neuron)

```
import numpy as np
import pandas as pd

# Inputs
x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
y = [0, 0, 1, 0]

# initialize weights by taking as inpt
w1 = float(input("Enter weigth w1"))
w2 = float(input("Enter weigth w2"))
th = float(input("Enter weigth th"))

# predicted output
y_pred = []

for i in range(0,4):
    yin = x1[i]*w1 + w2 * x2[i]
```

```
    if(yin >= th):
        y_pred.append(1)
    else:
        y_pred.append(0)

table = pd.DataFrame({"x1":x1, "x2": x2, "y": y, "y_pred": y_pred})
print(table)
```

Exp 9 (PCA)

```
import numpy as np
import matplotlib.pyplot as plt

# Define dataset
x = np.array([4, 8, 13, 7])
y = np.array([11, 4, 5, 14])
dataset = np.array([x, y])
print("Define Dataset")
print(dataset)
print()

# Finding mean
xMean = np.mean(x)
yMean = np.mean(y)

# Adjusting the mean obtained
MeanAdjusted = np.array([x - xMean, y - yMean])
print("Mean adjusted:")
print(MeanAdjusted)
print("\n")

# Finding the Covariance
covariance_matrix = np.cov(dataset)
print("Covariance Matrix")
print(covariance_matrix)
print("\n")

# Compute the Eigen Values and Eigen Vectors
eigen_values, eigen_vectors = np.linalg.eig(covariance_matrix)
print("Eigen Values")
print(eigen_values)
```

```
print()

# Sort result in descending order
sorted_indices = np.argsort(eigen_values)[::-1]
sorted_eigen_values = eigen_values[sorted_indices]
sorted_eigen_vectors = eigen_vectors[:, sorted_indices]
print("Sorted Eigen Values")
print(sorted_eigen_values)
print()
print("Sorted Eigen Vectors")
print(sorted_eigen_vectors)
print()

# Perform PCA
PCA = np.dot(sorted_eigen_vectors.T, MeanAdjusted)
print("Principal Component Analysis:")
print(PCA)

# Scatter plot after PCA
plt.subplot(1, 2, 2)
plt.scatter(PCA[0], PCA[1], color='red')
plt.title('After PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')

plt.tight_layout()
plt.show()
```