05010106DS06 - Soft Computing-Lab Manual

Requirements (install once):

```
pip install numpy scipy scikit-learn matplotlib tensorflow==2.12 scikit-
fuzzy pandas
```

(We use only widely-available packages. Where a specialized package would simplify an experiment — e.g., ANFIS libraries — I include a small simplified implementation or guidance.)

---

1. Implementation of Fuzzy Logic Operations (using `scikit-fuzzy`)

```
# file: fuzzy_operations.py
import numpy as np
import skfuzzy as fuzz

# Example fuzzy sets on universe 0..10
x = np.linspace(0, 10, 101)
mf_low = fuzz.trimf(x, [0, 0, 5])
mf_med = fuzz.trimf(x, [2.5, 5, 7.5])
mf_high = fuzz.trimf(x, [5, 10, 10])

# Fuzzy operations: union (max), intersection (min), complement (1 - mu)
union = np.fmax(mf_low, mf_med)
intersection = np.fmin(mf_med, mf_high)
complement_low = 1 - mf_low

if __name__ == "__main__":
    import matplotlib.pyplot as plt
    plt.plot(x, mf_low, label='Low')
    plt.plot(x, mf_med, label='Medium')
    plt.plot(x, mf_high, label='High')
    plt.plot(x, union, '--', label='Union(Low,Med)')
    plt.plot(x, intersection, ':', label='Intersection(Med,High)')
    plt.plot(x, complement_low, '-.', label='Complement(Low)')
    plt.legend(); plt.title("Fuzzy Set Operations"); plt.show()
```

---

2. Design of Fuzzy Inference System (FIS) — Mamdani

```
# file: simple_fis.py
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
```

```
# Define fuzzy variables
temp = ctrl.Antecedent(np.arange(0, 41, 1), 'temperature')
fan = ctrl.Consequent(np.arange(0, 11, 1), 'fan')

# Memberships
temp['cold'] = fuzz.trimf(temp.universe, [0, 0, 15])
temp['warm'] = fuzz.trimf(temp.universe, [10, 20, 30])
temp['hot']  = fuzz.trimf(temp.universe, [25, 40, 40])

fan['low'] = fuzz.trimf(fan.universe, [0, 0, 4])
fan['med'] = fuzz.trimf(fan.universe, [3, 5, 7])
fan['high']= fuzz.trimf(fan.universe, [6, 10, 10])

# Rules
rule1 = ctrl.Rule(temp['cold'], fan['low'])
rule2 = ctrl.Rule(temp['warm'], fan['med'])
rule3 = ctrl.Rule(temp['hot'],  fan['high'])

fan_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])
fan_sim  = ctrl.ControlSystemSimulation(fan_ctrl)

if __name__ == "__main__":
    # Example input
    fan_sim.input['temperature'] = 28
    fan_sim.compute()
    print("Fan speed (crisp):", fan_sim.output['fan'])
```

3. Defuzzification Techniques (Centroid, Bisector, MOM, LOM, SOM demo)

```
# file: defuzz_demo.py
import numpy as np
import skfuzzy as fuzz

x = np.linspace(0, 10, 1001)
mf = fuzz.gaussmf(x, 6, 1.2)  # example aggregated output

centroid = fuzz.defuzz(x, mf, 'centroid')
bisector  = fuzz.defuzz(x, mf, 'bisector')
mom       = fuzz.defuzz(x, mf, 'mom')   # mean of maximum
lom       = fuzz.defuzz(x, mf, 'lom')   # largest of maximum
som       = fuzz.defuzz(x, mf, 'som')   # smallest of maximum

print("Centroid:", centroid)
print("Bisector:", bisector)
print("Mean of maxima:", mom)
print("Largest of maxima:", lom)
print("Smallest of maxima:", som)
```

4. Implementation of Single-Layer Perceptron (binary classification)

```
# file: perceptron.py
import numpy as np

class Perceptron:
    def __init__(self, n_inputs, lr=0.1, epochs=100):
        self.w = np.zeros(n_inputs + 1)
        self.lr = lr
        self.epochs = epochs

    def predict(self, x):
        s = np.dot(x, self.w[1:]) + self.w[0]
        return 1 if s >= 0 else 0

    def fit(self, X, y):
        for _ in range(self.epochs):
            for xi, target in zip(X, y):
                pred = self.predict(xi)
                error = target - pred
                self.w[1:] += self.lr * error * xi
                self.w[0] += self.lr * error

if __name__ == "__main__":
    # Simple OR dataset
    X = np.array([[0,0],[0,1],[1,0],[1,1]])
    y = np.array([0,1,1,1])
    p = Perceptron(2)
    p.fit(X, y)
    print("Weights:", p.w)
    for x in X:
        print(x, "->", p.predict(x))
```

---

5. Multilayer Perceptron using Backpropagation (using Keras)

```
# file: mlp_backprop.py
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD

if __name__ == "__main__":
    # XOR dataset
    X = np.array([[0,0],[0,1],[1,0],[1,1]])
    y = np.array([[0],[1],[1],[0]])

    model = Sequential([
        Dense(4, input_dim=2, activation='tanh'),
        Dense(1, activation='sigmoid')
    ])
```

```
    model.compile(loss='binary_crossentropy',
optimizer=SGD(learning_rate=0.2), metrics=['accuracy'])
    model.fit(X, y, epochs=500, verbose=0)
    print("Predictions:")
    print(model.predict(X).round(3))
```

---

6. Simple Neural Network (McCulloch–Pitts neuron)

```
# file: mcculloch_pitts.py
import numpy as np

def mcnp(inputs, weights, threshold):
    s = np.dot(inputs, weights)
    return 1 if s >= threshold else 0

if __name__ == "__main__":
    # AND gate as McCulloch-Pitts
    w = np.array([1,1])
    thresh = 2
    print([mcnp(x, w, thresh) for x in [(0,0),(0,1),(1,0),(1,1)]])  #
[0,0,0,1]
```

---

7. Implementation of Genetic Algorithm (simple maximization example)

```
# file: simple_ga.py
import random
import math

def fitness(x):
    # Example: maximize f(x) = x*sin(10*pi*x) + 1.0
    return x * math.sin(10*math.pi*x) + 1

def mutate(x, rate=0.1):
    if random.random() < rate:
        x += random.uniform(-0.1, 0.1)
        x = max(0.0, min(1.0, x))
    return x

def crossover(a, b):
    alpha = random.random()
    return alpha*a + (1-alpha)*b

def ga(pop_size=30, generations=50):
    pop = [random.random() for _ in range(pop_size)]
    for g in range(generations):
        pop.sort(key=fitness, reverse=True)
        next_gen = pop[:int(0.2*pop_size)]  # elitism
        while len(next_gen) < pop_size:
```

```python
            a, b = random.choices(pop[:15], k=2)
            child = crossover(a,b)
            child = mutate(child, rate=0.2)
            next_gen.append(child)
        pop = next_gen
    best = max(pop, key=fitness)
    return best, fitness(best)

if __name__ == "__main__":
    best_x, best_fit = ga()
    print("Best x:", best_x, "fitness:", best_fit)
```

---

8. Solving Travelling Salesman Problem using GA (simple representation)

```python
# file: tsp_ga.py
import random
import math

def tour_length(tour, dist_matrix):
    return sum(dist_matrix[tour[i]][tour[(i+1)%len(tour)]] for i in
range(len(tour)))

def create_population(n_cities, pop_size):
    base = list(range(n_cities))
    return [random.sample(base, n_cities) for _ in range(pop_size)]

def pmx_crossover(a, b):
    size = len(a)
    p1, p2 = sorted(random.sample(range(size), 2))
    child = [None]*size
    child[p1:p2+1] = a[p1:p2+1]
    for i in range(p1, p2+1):
        if b[i] not in child:
            pos = i
            val = b[i]
            while True:
                pos = a.index(val)
                if child[pos] is None:
                    child[pos] = b[i]
                    break
                val = b[pos]
    for i in range(size):
        if child[i] is None:
            child[i] = b[i]
    return child

def mutate_swap(tour, rate=0.02):
    if random.random() < rate:
        i,j = random.sample(range(len(tour)), 2)
        tour[i], tour[j] = tour[j], tour[i]

def tsp_ga_demo():
    # sample cities (coordinates)
    coords = [(random.random(), random.random()) for _ in range(12)]
```

```
    n = len(coords)
    # distance matrix
    dist = [[math.dist(coords[i], coords[j]) for j in range(n)] for i in
range(n)]
    pop = create_population(n, 100)
    for gen in range(300):
        pop.sort(key=lambda t: tour_length(t, dist))
        next_pop = pop[:10]
        while len(next_pop) < 100:
            a,b = random.sample(pop[:30],2)
            child = pmx_crossover(a,b)
            mutate_swap(child)
            next_pop.append(child)
        pop = next_pop
    best = min(pop, key=lambda t: tour_length(t, dist))
    print("Best tour length:", tour_length(best, dist))
    print("Tour:", best)

if __name__ == "__main__":
    tsp_ga_demo()
```

---

9. Design of Adaptive Neuro-Fuzzy Inference System (ANFIS) — simplified demo

ANFIS is involved. Below is a compact *conceptual* implementation illustrating an ANFIS-like two-input Sugeno model with gradient descent for consequent parameters. This is intentionally simplified for lab purposes.

```
# file: simple_anfis.py
import numpy as np

def gauss(x, c, sigma):
    return np.exp(-((x-c)**2)/(2*sigma**2))

class SimpleANFIS:
    def __init__(self):
        # two inputs, each with 2 Gaussian MFs -> 4 rules
        # MF params (centers and sigmas)
        self.c = np.array([[0.2, 0.8], [0.2, 0.8]])  # input1 centers,
input2 centers
        self.sigma = np.array([[0.2, 0.2], [0.2, 0.2]])
        # consequent parameters for linear function: p1*x + p2*y + p3
        self.P = np.random.randn(4, 3)

    def forward(self, x, y):
        # firing strengths
        w = []
        for i in range(2):
            for j in range(2):
                mu1 = gauss(x, self.c[0,i], self.sigma[0,i])
                mu2 = gauss(y, self.c[1,j], self.sigma[1,j])
                w.append(mu1*mu2)
```

```
        w = np.array(w)
        W = w / (np.sum(w) + 1e-9)
        # rule outputs
        outs = np.array([self.P[k,0]*x + self.P[k,1]*y + self.P[k,2] for k
in range(4)])
        return np.dot(W, outs), W, outs

    def train(self, data, epochs=200, lr=0.01):
        for ep in range(epochs):
            for x,y,t in data:
                y_pred, W, outs = self.forward(x,y)
                err = (t - y_pred)
                # update consequent params (simple gradient step)
                for k in range(4):
                    grad = -2*err*W[k]*np.array([x,y,1.0])
                    self.P[k] -= lr * grad

if __name__ == "__main__":
    # synthetic target function: t = x + y
    data = [(np.random.rand(), np.random.rand(), None) for _ in range(200)]
    data = [(x,y,x+y) for x,y,_ in data]
    anfis = SimpleANFIS()
    anfis.train(data, epochs=200, lr=0.02)
    print("Test:", anfis.forward(0.2, 0.3)[0], "expected 0.5")
```

Note: For a full-featured ANFIS lab, consider using specialized packages (e.g., `anfis` on PyPI) or extend the above to tune MF parameters with gradient descent.

---

10. Comparison of Hard and Soft Computing Techniques

This is a **report-style** exercise. Below is a short Python script that generates a printable comparison table (CSV) you can include in the lab manual.

```
# file: compare_hard_soft.py
import pandas as pd

rows = [
    ("Paradigm","Hard Computing","Soft Computing"),
    ("Philosophy","Precise models, exact algorithms","Approximate, tolerant
to uncertainty"),
    ("Examples","Classical algorithms, exact optimization","Fuzzy logic,
neural networks, GA, PSO"),
    ("Uncertainty Handling","Poor","Good"),
    ("Adaptivity","Low","High"),
    ("Typical Use","Deterministic problems","Noisy, uncertain or complex
problems"),
]

df = pd.DataFrame(rows[1:], columns=["Aspect","Hard Computing","Soft
Computing"])
df.to_csv("comparison_hard_vs_soft.csv", index=False)
```

```
print("Saved comparison_hard_vs_soft.csv")
```

---

11. Implement PSO and find global minimum of Rastrigin function

```
# file: pso_rastrigin.py
import random, math
import numpy as np

def rastrigin(x):
    A = 10
    return A*len(x) + sum([(xi**2 - A*math.cos(2*math.pi*xi)) for xi in x])

def pso(n_dim=2, swarm_size=40, iters=200):
    w = 0.7; c1 = 1.5; c2 = 1.5
    lb, ub = -5.12, 5.12
    pos = [np.array([random.uniform(lb,ub) for _ in range(n_dim)]) for _ in
range(swarm_size)]
    vel = [np.zeros(n_dim) for _ in range(swarm_size)]
    pbest = pos.copy()
    pbest_val = [rastrigin(p) for p in pbest]
    gbest = pbest[np.argmin(pbest_val)]
    gbest_val = min(pbest_val)

    for _ in range(iters):
        for i in range(swarm_size):
            r1, r2 = random.random(), random.random()
            vel[i] = w*vel[i] + c1*r1*(pbest[i] - pos[i]) + c2*r2*(gbest -
pos[i])
            pos[i] += vel[i]
            pos[i] = np.clip(pos[i], lb, ub)
            fitness = rastrigin(pos[i])
            if fitness < pbest_val[i]:
                pbest[i] = pos[i].copy(); pbest_val[i] = fitness
                if fitness < gbest_val:
                    gbest, gbest_val = pos[i].copy(), fitness
    return gbest, gbest_val

if __name__ == "__main__":
    best, val = pso()
    print("Best position:", best, "Rastrigin value:", val)
```

---

12. Use PSO to solve a simple load balancing problem (minimize makespan)

We assign `N` tasks each with processing times `t_i` to `M` machines. Represent a particle as task-to-machine continuous vector; decoding maps each task to machine index by rounding.

```
# file: pso_load_balance.py
import numpy as np, random
```

```
def makespan(assignments, times, M):
    load = [0]*M
    for task_idx, machine in enumerate(assignments):
        load[machine] += times[task_idx]
    return max(load)

def decode(position, M):
    # position entries in [0,1] -> machine 0..M-1
    return [int(min(M-1, int(p*M))) for p in position]

def pso_balance(times, M=3, swarm_size=40, iters=200):
    N = len(times)
    lb, ub = 0.0, 1.0
    pos = [np.random.rand(N) for _ in range(swarm_size)]
    vel = [np.zeros(N) for _ in range(swarm_size)]
    pbest = pos.copy()
    pbest_val = [makespan(decode(p, M), times, M) for p in pbest]
    gbest = pbest[np.argmin(pbest_val)]
    gbest_val = min(pbest_val)

    for _ in range(iters):
        for i in range(swarm_size):
            vel[i] = 0.7*vel[i] + 1.5*random.random()*(pbest[i]-pos[i]) +
1.5*random.random()*(gbest-pos[i])
            pos[i] += vel[i]
            pos[i] = np.clip(pos[i], lb, ub)
            val = makespan(decode(pos[i], M), times, M)
            if val < pbest_val[i]:
                pbest[i] = pos[i].copy(); pbest_val[i] = val
                if val < gbest_val:
                    gbest, gbest_val = pos[i].copy(), val
    return decode(gbest, M), gbest_val

if __name__ == "__main__":
    times = [random.randint(1,20) for _ in range(20)]
    assignment, val = pso_balance(times, M=4)
    print("Processing times:", times)
    print("Assignment (task->machine):", assignment)
    print("Makespan:", val)
```

---

13. Build & train LSTM to predict next value in univariate time series

```
# file: lstm_predict_univariate.py
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

def create_sine_series(n=1000):
    t = np.arange(n)
    series = np.sin(0.02*t) + 0.3*np.random.randn(n)
    return series

def create_dataset(series, lookback=10):
```

```
    X, y = [], []
    for i in range(len(series)-lookback):
        X.append(series[i:i+lookback])
        y.append(series[i+lookback])
    X = np.array(X)[:, :, None]
    y = np.array(y)
    return X, y

if __name__ == "__main__":
    s = create_sine_series(1000)
    X, y = create_dataset(s, lookback=20)
    split = int(0.8*len(X))
    model = Sequential([LSTM(32, input_shape=(X.shape[1],1)), Dense(1)])
    model.compile(optimizer='adam', loss='mse')
    model.fit(X[:split], y[:split], epochs=20, batch_size=32, verbose=1)
    preds = model.predict(X[split:split+10]).flatten()
    print("Real:", y[split:split+10])
    print("Pred:", preds.round(3))
```

14. LSTM for Sentiment Classification (small synthetic dataset)

```
# file: lstm_sentiment.py
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense

if __name__ == "__main__":
    texts = ["good movie", "excellent", "bad movie", "not good", "i loved
it", "i hate it", "very nice", "poor quality"]
    labels = [1,1,0,0,1,0,1,0]
    tok = Tokenizer(num_words=100)
    tok.fit_on_texts(texts)
    seq = tok.texts_to_sequences(texts)
    X = pad_sequences(seq, maxlen=5)
    y = np.array(labels)
    model = Sequential([Embedding(100, 16, input_length=5), LSTM(16),
Dense(1, activation='sigmoid')])
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(X, y, epochs=30, verbose=0)
    print("Preds:", (model.predict(X)>0.5).astype(int).flatten())
```

15. Implement SOM + clustering/visualization on Iris

We'll write a simple Kohonen SOM implementation (2D grid) for visualization.

```python
# file: som_iris.py
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

class SimpleSOM:
    def __init__(self, m, n, dim, lr=0.1, sigma=None):
        self.m, self.n, self.dim = m, n, dim
        self.weights = np.random.rand(m*n, dim)
        self.lr = lr
        self.sigma = sigma if sigma else max(m,n)/2
        self.locations = np.array([[i,j] for i in range(m) for j in
range(n)])

    def winner(self, x):
        dists = np.linalg.norm(self.weights - x, axis=1)
        return np.argmin(dists)

    def train(self, data, epochs=100):
        for t in range(epochs):
            for x in data:
                w_idx = self.winner(x)
                w_loc = self.locations[w_idx]
                dists = np.linalg.norm(self.locations - w_loc, axis=1)
                h = np.exp(-(dists**2)/(2*(self.sigma**2)))
                self.weights += (self.lr * h[:,None]) * (x - self.weights)

if __name__ == "__main__":
    iris = datasets.load_iris()
    X = iris.data
    X = MinMaxScaler().fit_transform(X)
    som = SimpleSOM(10,10,4)
    som.train(X, epochs=50)
    # Map points to neurons
    winners = [som.winner(x) for x in X]
    plt.scatter([loc[0] for loc in som.locations[winners]], [loc[1] for loc
in som.locations[winners]], c=iris.target)
    plt.title("SOM mapping of Iris (color = true class)")
    plt.show()
```

---

16. Hopfield Network to store & recall binary patterns

```python
# file: hopfield.py
import numpy as np

class Hopfield:
    def __init__(self, n):
        self.n = n
        self.W = np.zeros((n,n))

    def train(self, patterns):
        for p in patterns:
            p = p*2 - 1  # convert 0/1 to -1/+1
```

```
            self.W += np.outer(p, p)
        np.fill_diagonal(self.W, 0)

    def recall(self, pattern, steps=10):
        p = pattern.copy()
        for _ in range(steps):
            for i in range(self.n):
                s = np.dot(self.W[i], p*2-1)
                p[i] = 1 if s >= 0 else 0
        return p

if __name__ == "__main__":
    patterns = [np.array([1,0,1,0,1]), np.array([0,1,0,1,0])]
    net = Hopfield(5); net.train(patterns)
    noisy = np.array([1,0,0,0,1])
    print("Recalled:", net.recall(noisy))
```

---

17. Bidirectional Associative Memory (BAM) for pattern association

```
# file: bam.py
import numpy as np

class BAM:
    def __init__(self):
        self.W = None

    def train(self, A, B):
        # A: list of binary vectors shape (k, na), B: (k, nb)
        na = A.shape[1]; nb = B.shape[1]
        self.W = np.zeros((na, nb))
        for a,b in zip(A, B):
            a0 = a*2-1; b0 = b*2-1
            self.W += np.outer(a0, b0)

    def recall_B(self, a, steps=10):
        x = a*2-1
        for _ in range(steps):
            y = np.sign(np.dot(x, self.W))
            y[y==0] = 1
            x = np.sign(np.dot(self.W, y))
        return ((y+1)//2).astype(int)

if __name__ == "__main__":
    A = np.array([[1,0,1],[0,1,0]])
    B = np.array([[0,1],[1,0]])
    bam = BAM(); bam.train(A,B)
    test = np.array([1,0,1])
    print("Associated B:", bam.recall_B(test))
```

How to use / print

- Copy each code block into its own `.py` file (I suggested filenames).
- Run `python filename.py` to execute demos.
- Use `matplotlib` plots to add figures to your printed manual.
- For the lab manual, include: objective, theory, code, sample run & observations for each experiment. I can generate a formatted PDF combining explanation + code + sample outputs if you want.

If you'd like, I can now:

- Produce a **single ZIP** with all `.py` files (ready to download), or
- Create a **print-ready PDF lab manual** with objectives, theory, code, and sample outputs for each experiment, or
- Expand any experiment into a longer step-by-step lab sheet with questions and expected results.

Which would you like next?