Code HW3 part 3

Jens Ifver

September 2021

```
    import numpy as np

### FUNCTIONS ###

def initialize_weights_thresholds(M):
    W_mat = np.random.normal(loc = 0, scale = 1, size = (M,3))
    theta_v = np.zeros(3)
    theta_h = np.zeros(M)
    return W_mat, theta_v, theta_h
def choose_pattern(X, n):
    p0 = np.random.randint(0,n)
    return np.copy(X[p0,:])
def update_hidden_n(W_mat, theta, state):
    bx = np.subtract(np.matmul(W_mat,state), theta)
    return np.copy(bx)
def update_visable_n(W_mat, theta, state):
    bx = np.subtract( np.matmul(np.transpose(state), W_mat), theta)
    return np.copy(bx)
def prob(bx):
    return 1/(1 + np.exp(-2*bx))
def mccullock_pitts(bx):
    px = prob(bx)
    rx = np.random.rand(1)
    if ( rx < px): return 1
    else: return -1
def change_w(b_vec_0, b_vec_k, visable_0, visable_k, eta):
    hard_brack = np.subtract( np.outer( np.tanh(b_vec_0), visable_0 ),
        np.outer( np.tanh(b_vec_k), visable_k ) )
    return eta * hard_brack
def change_theta_v(eta, visable_0, visable_k):
    return -eta * np.subtract( visable_0, visable_k )
def change_theta_h(eta, b_vec_0, b_vec_k):
    return -eta * np.subtract( np.tanh(b_vec_0), np.tanh(b_vec_k) )

# CD-k algorithm
```

```
def run_CD_k(epochs, patterns, M, k, eta, batch_size):
    W_mat, theta_v, theta_h = initialize_weights_thresholds(M)
    for ep in range(epochs):
        delta_w = np.zeros((M,3))
        delta_theta_v = np.zeros(3)
        delta_theta_h = np.zeros(M)
        for mu in range(batch_size):
            # Choose pattern
            choosen_pattern = choose_pattern(patterns, 4)
            visable_0 = np.copy(choosen_pattern)
            visable_n = np.copy(visable_0)

            # Update hidden neurons
            b_v_t = np.zeros(3)
            b_h_0 = np.zeros(M)
            b_h_t = np.zeros(M)
            b_h_0 = np.add( b_h_0,update_hidden_n(W_mat, theta_h, visable_0))
            hidden_n = np.zeros(M)
            for it in range(M):
                hidden_n[it] = mccullock_pitts(b_h_0[it])
            # Time loop
            for tx in range(k):
                # Update visable neurons
                b_v_t = update_visable_n(W_mat, theta_v, hidden_n)
                for jx in range(3):
                    visable_n[jx] = mccullock_pitts(b_v_t[jx])
                # Update hidden neurons
                b_h_t =  update_hidden_n(W_mat, theta_h,  visable_n)
                for ix in range(M):
                    hidden_n[ix] = mccullock_pitts(b_h_t[ix])

            # Compute weight and threshold increments
            delta_w = np.add(delta_w, change_w(b_h_0, b_h_t, visable_0, visable_n, eta))
            delta_theta_v = np.add(delta_theta_v , change_theta_v(eta, visable_0,visable
            delta_theta_h = np.add(delta_theta_h , change_theta_h(eta, b_h_0, b_h_t))
        # Adjust weights and thresholds
        W_mat = np.add(delta_w, W_mat)
        theta_v = np.add(theta_v, delta_theta_v)
        theta_h = np.add(theta_h, delta_theta_h)
    return W_mat, theta_v, theta_h

# Feed patterns
def feed_patterns(Xi, N_o, N_i, W_mat, theta_h, theta_v):
    M, v = W_mat.shape
    P_B = np.zeros(8)
    for ix in range(N_o):
```

```
        mu = choose_pattern(Xi, 8)
        visable_n = mu
        # Update hidden neurons
        b_h_0 = np.zeros(M)
        b_h_t = np.zeros(M)
        b_v_t = np.zeros(v)
        b_h_0 = update_hidden_n(W_mat, theta_h, visable_n)
        hidden_n = np.zeros(M)
        for hx in range(M):
            hidden_n[hx] = mccullock_pitts(b_h_0[hx])
        # Time loop
        for tx in range(N_i):
            # Update visable neurons
            b_v_t = update_visable_n(W_mat, theta_v, hidden_n)
            for jx in range(3):
                visable_n[jx] = mccullock_pitts(b_v_t[jx])
            # Update hidden neurons
            b_h_t = update_hidden_n(W_mat, theta_h, visable_n)
            for kx in range(M):
                hidden_n[kx] = mccullock_pitts(b_h_t[kx])
            for lx in range(8):
                if (np.sum (visable_n ==  Xi[lx,:]) == 3):
                    P_B[lx] += 1/(N_o * N_i)
    return P_B

# KL divergence
def KL(P_B):
    sm = 0
    P_data = np.array([0.25, 0.25, 0.25, 0.25])
    for it in range(4):
            sm += P_data[it] * np.log(P_data[it]/P_B[it])
    return sm
def KL_M(P_B_mat):
    KL_vec = np.zeros(4)
    for it in range(4):
        KL_vec[it] = KL(P_B_mat[it,:])
    return KL_vec

# Main function
def run_for_all_M(X_all):
    batch_size = np.array([1, 2, 30, 30])
    time = np.array([25, 50, 75, 100])
    Ms = np.array([1, 2, 4, 8])
    epochs = 200
    eta = 0.1
    k = 100
```

```
    N_outer = 1000
    N_inner = 1000
    P_B_mat = np.zeros((4,8))
    for m in range(4):
        W_mat, theta_v, theta_h = run_CD_k(epochs, X_all[:4,:], Ms[m], k, eta,batch_size
        P_B_mat[m,:] = feed_patterns(X_all, N_outer, N_inner, W_mat, theta_h, theta_v)
        print(time[m])
    KL_vec = KL_M(P_B_mat)
    return KL_vec

### CALCULATIONS ###

# Declare patterns
x1 = np.array([-1, -1, -1])
x2 = np.array([1, -1, 1])
x3 = np.array([-1, 1, 1])
x4 = np.array([1, 1, -1])
X = np.array([x1,x2,x3,x4])
x5 = np.array([1, 1, 1])
x6 = np.array([-1, 1, -1])
x7 = np.array([1, -1, -1])
x8 = np.array([-1, -1, 1])
X_all = np.array([x1, x2, x3, x4, x5, x6, x7, x8])

# Run for all M
KL_out1 = run_for_all_M(X_all)

# Plot result
import matplotlib.pyplot as plt
M = [1, 2, 4, 8]
plt.plot(M, KL_out1, 'p', markersize = 10)
plt.plot(M, KL_out1)
plt.xlabel('# hidden neurons')
plt.ylabel('$KL_D$')
plt.title('The Kullback-Leible divergence vs number of hidden neurons')
```