```python
import numpy as np
import scipy.io.wavfile
import os
from numpy import linalg as LA

def update_W(W, x, learning_rate):
    """
    Perform a gradient ascent update on W using data element x and the provided learning rate.

    This function should return the updated W.

    Args:
        W: The W matrix for ICA
        x: A single data element
        learning_rate: The learning rate to use

    Returns:
        The updated W
    """

    # *** START CODE HERE ***
    laplace, logistic = True, False

    if logistic:
        g = 1./(1 + np.exp(-W.dot(x)))
        updated_W = W + learning_rate * (np.outer(1 - 2*g, x) + np.linalg.inv(W.T))

    if laplace:
        g = np.sign(W.dot(x))
        updated_W = W + learning_rate * (-np.outer(g, x) + np.linalg.inv(W.T))

    # *** END CODE HERE ***

    return updated_W


def unmix(X, W):
    """
    Unmix an X matrix according to W using ICA.

    Args:
        X: The data matrix
        W: The W for ICA

    Returns:
        A numpy array S containing the split data
    """

    S = np.zeros(X.shape)

    # *** START CODE HERE ***
    S = X.dot(W.T)
    # *** END CODE HERE ***

    return S


Fs = 11025

def normalize(dat):
    return 0.99 * dat / np.max(np.abs(dat))

def load_data():
    mix = np.loadtxt('./mix.dat')
    return mix

def save_W(W):
    np.savetxt('./W.txt',W)

def save_sound(audio, name):
    scipy.io.wavfile.write('./{}.wav'.format(name), Fs, audio)

```

```python
def unmixer(X):
    M, N = X.shape
    W = np.eye(N)

    anneal = [0.1 , 0.1, 0.1, 0.05, 0.05, 0.05, 0.02, 0.02, 0.01 , 0.01, 0.005, 0.005, 0.002, 0.002, 0.001, 0.001]
    print('Separating tracks ...')
    for lr in anneal:
        print(lr)
        rand = np.random.permutation(range(M))
        for i in rand:
            x = X[i]
            W = update_W(W, x, lr)

    return W

def main():
    # Seed the randomness of the simulation so this outputs the same thing each time
    np.random.seed(0)
    X = normalize(load_data())

    print(X.shape)

    for i in range(X.shape[1]):
        save_sound(X[:, i], 'mixed_{}'.format(i))

    W = unmixer(X)
    print(W)
    save_W(W)
    S = normalize(unmix(X, W))
    assert S.shape[1] == 5
    for i in range(S.shape[1]):
        if os.path.exists('split_{}'.format(i)):
            os.unlink('split_{}'.format(i))
        save_sound(S[:, i], 'split_{}'.format(i))

if __name__ == '__main__':
    main()
```