```
import numpy as np
from scipy.special import digamma as dga
from scipy.special import gamma as ga
from scipy.special import loggamma as lga
```

```
eps=1e-10
def log(x):
    return np.log(x + eps)

def digamma(x):
    return dga(x + eps)

def loggamma(x):
    return lga(x + eps)
```

```
def init(data):
    vocab = np.array([i for i in range(100)])

    num_doc = data.shape[0]
    num_vocab = vocab.shape[0]
    len_doc = data.shape[1]
    num_topic = 10

    w = np.zeros([num_doc, len_doc, num_vocab])
    for d in range(num_doc):
        for n in range(len_doc):
            w[d, n, data[d, n]] = 1

    alpha = np.ones(shape=num_topic)
    eta = np.ones(shape=num_vocab)

    phi = np.random.rand(num_doc, len_doc, num_topic)
    for d in range(num_doc):
        for n in range(len_doc):
            phi[d, n] /= np.sum(phi[d, n])

    gam = np.random.rand(num_doc, num_topic)
    gam /= np.sum(gam, axis=1)[:, np.newaxis]

    lam = np.random.rand(num_topic, num_vocab)
    lam /= np.sum(lam, axis=1)[:, np.newaxis]
    return lam, gam, phi, w, num_doc, num_topic, num_vocab, len_doc, alpha, eta
```

```
def one_step(lam, gam, phi, w, num_doc, num_topic, num_vocab, len_doc, alpha, eta):
    #print(num_doc, num_topic, num_vocab)
    for k in range(num_topic):
        lam[k] = eta
        for d in range(num_doc):
            for n in range(len_doc):
                lam[k] += phi[d, n, k] * w[d, n]
    #lam /= np.sum(lam, axis=1)[:, np.newaxis]

    gam = alpha + np.sum(phi, axis=1)
    #gam /= np.sum(gam, axis=1)[:, np.newaxis]

    def get_single_doc(lam, gam, phi, w, d):
        for n in range(len_doc):
            #phi[d, n, :] = np.exp(digamma(gam[d, :]) + digamma(lam[:, data[d, n]]) - digamma(np.sum(lam, axis=1)))
            for k in range(num_topic):
                phi[d, n, k] = np.exp(digamma(lam[k, data[d, n]]) - digamma(np.sum(lam[k])) + digamma(gam[d, k]) - digamma(np.sum(gam[d])))
            phi[d, n, :] /= np.sum(phi[d, n, :])
        return phi[d], d

    with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
        future_list = [executor.submit(get_single_doc, lam, gam, phi, w, d) for d in range(num_doc)]
        for future in concurrent.futures.as_completed(future_list):
            phi_d, d = future.result()
            phi[d] = phi_d

    return lam, gam, phi, w, num_doc, num_topic, num_vocab, len_doc, alpha, eta
```

```
import concurrent.futures

def get_res1(lam, gam, phi, w):
    res_1 = 0.0
    res_1 += num_topic * loggamma(np.sum(eta))
    res_1 -= num_topic * np.sum(loggamma(eta))
    '''
    for k in range(num_topic):
        for i in range(num_vocab):
            res_1 += (eta[i] - 1) * (digamma(lam[k, i]) - digamma(np.sum(lam[k])))
    '''
    return res_1


def get_res2(lam, gam, phi, w):
    res_2 = 0.0
    for n in range(len_doc):
        for k in range(num_topic):
            res_2 += phi[:, n, k] * (digamma(gam[:, k]) - digamma(np.sum(gam, axis=1)))
    #res_2 -= digamma(np.sum(gam, axis=1))
    res_2 = np.sum(res_2)
    return res_2


def get_res3(lam, gam, phi, w):
    res_3 = 0.0
    res_3 += loggamma(np.sum(alpha))
    res_3 -= np.sum(loggamma(alpha))
    '''
    for k in range(num_topic):
        res_3 += (alpha[k] - 1) * (digamma(gam[:, k] - digamma(np.sum(gam[:, k]))))
    '''
    res_3 = np.sum(res_3)
    return res_3

def get_res4(lam, gam, phi, w):
    res_4 = 0.0
    def get_res4_single_loc(lam, gam, phi, w, n):
        res_loc = 0.0
        for k in range(num_topic):
            sum_lam_k = np.sum(lam[k])
            for i in range(num_vocab):
                res_loc += phi[:, n, k] * w[:, n, i] * (digamma(lam[k, i]) -
digamma(sum_lam_k))
        res_loc = np.sum(res_loc)
        return res_loc

    with concurrent.futures.ThreadPoolExecutor(max_workers=32) as executor:
        future_list = [executor.submit(get_res4_single_loc, lam, gam, phi, w, n) for n
in range(len_doc)]
        for future in concurrent.futures.as_completed(future_list):
            res_4 += future.result()
    return res_4

def get_res5(lam, gam, phi, w):
    res_5 = 0.0
    for k in range(num_topic):
        res_5 += loggamma(np.sum(lam[k])) - np.sum(loggamma(lam[k]))
    for k in range(num_topic):
        sum_lam_k = np.sum(lam[k])
        #'''
        res_5 += np.sum((lam[k] - 1) * (digamma(lam[k]) - digamma(sum_lam_k)))
        '''
        for i in range(num_vocab):
            res_5 += (lam[k, i] - 1) * (digamma(lam[k, i]) - digamma(sum_lam_k))
        '''
    return -res_5

def get_res6(lam, gam, phi, w):
    res_6 = 0.0
    res_6 += np.sum(phi * log(phi))
    return -res_6

def get_res7(lam, gam, phi, w):
    res_7 = 0.0
    res_7 += loggamma(np.sum(gam, axis=1)) - np.sum(loggamma(gam), axis=1)
    #print(res_7)
    res_7 = np.sum(res_7)
    for d in range(num_doc):
        res_7 += np.sum((gam[d] - 1) * (digamma(gam[d]) - digamma(np.sum(gam[d]))))
    return -res_7

def elbo(lam, gam, phi, w):
    res = 0.0
    func_list = [get_res1, get_res2, get_res3, get_res4, get_res5, get_res6, get_res7]
    with concurrent.futures.ThreadPoolExecutor(max_workers=8) as executor:
        future_list = [executor.submit(func, lam, gam, phi, w) for func in func_list]
        for future in concurrent.futures.as_completed(future_list):
            res += future.result()

    return res
```

```
elbo_list = []
if True:
    data = np.load("mcs_hw4_p1_lda.npy")
    lam, gam, phi, w, num_doc, num_topic, num_vocab, len_doc, alpha, eta = init(data)
    for i in range(100):
        lam, gam, phi, w, num_doc, num_topic, num_vocab, len_doc, alpha, eta =
one_step(lam, gam, phi, w, num_doc, num_topic, num_vocab, len_doc, alpha, eta)
        #print("iteration " + str(i) + " done")
        elbo_per_point = elbo(lam, gam, phi, w)
        elbo_list.append(elbo_per_point)
        print(elbo_per_point)
```