LAMPIRAN

Url Gambar Dataset Hardhat

https://images.cdn3.stockunlimited.net/thumb450/portrait-of-a-happy-young-woman-architect-wearing-hardhat-at-construction-site_1918949.jpg

https://www.handymanclevelandoh.com/uploads/1/3/1/5/131540157/cleveland-oh-handyman-service-about-orig_orig.jpg

https://st.depositphotos.com/1010683/4099/i/450/depositphotos_40996081-stock-photo-construction-worker.jpg

https://1.bp.blogspot.com/-vzVGWX5WdSg/Xm-8EQ5T5BI/AAAAAAAABNY/fF6aq3H9rGksxBf0FKuNfC6QdzqZ8OH_ACLcBGAsYHQ/s1600/5529419-male-worker-wearing-hardhat-ear-protection-and-dust-mask.jpg

https://previews.123rf.com/images/rido/rido1201/rido120100035/12155596-happy-manual-worker-presenting-your-text-isolated-on-white-background.jpg

https://previews.123rf.com/images/quicklyfy/quicklyfy1802/quicklyfy180200531/96128577-portrait-of-architect-wearing-suit-and-yellow-hardhat-holding-project-plan-on-white-background.jpg

https://images.squarespace-cdn.com/content/v1/5aec90113e2d09a9e028a9b2/1525715473497-SYW5SW3OHFOC4SBYF5IC/ke17ZwdGBTоddI8pDm48kHTHJlACqy9PR67J39ATHOp7gQa3H78H3Y0txjaiv_0fDoOvxcdMmMKkDsyUqMSsMWxHk725yiiHCCLfrh8O1z5QPOohDIaIeljMHgDF5CVlOqpeNLcJ80NK65_fV7S1UfZ1qQlCBU8D8HwMSx19XWWhZWoAraqJt_ybhixA0kzVDVfRxgAIsQ7eQiOnQS3C_Q/bigstock-Warehouse-team-with-arms-cross-81266576.jpg?format=1500w

http://2.bp.blogspot.com/-uFOdI9ZKdzA/Vc7grTLp6DI/AAAAAAAAAng/GqcGrA562ps/s1600/534485906-blueprint-construction-engineer-indian-india-arms-crossed.jpg

https://www.skyluxeroofing.com/wp-content/uploads/2018/12/commercial-siding-contractor.jpg

https://thumbs.dreamstime.com/z/%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE-%D0%B7%D1%80%D0%B5%D0%BB%D1%8B%D0%BC-%D1%87%D0%B5%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA-%D0%B2-%D1%81%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%B8%D0

%B8-
%D0%B7%D0%B0%D1%89%D0%B8%D1%82%D0%BD%D0%BE%D0%B3%D0%BE-%D1%88%D0%BB%D0%B5%D0%BC%D0%B0-
%D0%B8%D0%B7%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D0%B8-167922585.jpg

https://res.cloudinary.com/fleetnation/image/private/c_fit,w_1120/g_south,l_text:style_gothic2:%C2%A9%20Tetra%20Images,o_20,y_10/g_center,l_watermark4,o_25,y_50/v1515946218/d7gu0pc0gaebbmohuiji.jpg

https://cdn7.dissolve.com/p/D25_135_911/D25_135_911_1200.jpg

https://s.alicdn.com/@sc01/kf/Hb937bace700f49adaf8fc727dd7f4b75y.jpg_220x220.jpg

https://media.istockphoto.com/photos/hispanic-woman-at-construction-site-wearing-hardhat-picture-id513761954?b=1&k=6&m=513761954&s=170667a&h=8InEDfARWESRylCNSpAQflHdLzSick782JSwFosvGA4=

https://i.pinimg.com/600x315/f9/eb/1b/f9eb1b5d079fc6b3ba950c0a4e359535.jpg

https://i.pinimg.com/564x/39/43/a4/3943a45f9ea7dc302ec772279678ec23--wavy-hair-her-hair.jpg

https://c8.alamy.com/comp/2AMF4MJ/head-and-shoulders-portrait-of-beautiful-female-worker-wearing-hardhat-and-looking-at-camera-while-posing-confidently-against-pale-pink-background-copy-space-2AMF4MJ.jpg

https://dispartilaw.com/wp-content/uploads/2017/02/hurt-on-construction-site-01-e1487093772104.jpg

https://plusrh.com.mx/gallery_gen/b5405bcbbca6b1ac87213e741822f543_3000x2160.jpg

https://www.etteplan.com/sites/default/files/header-images/Engineer-at-the-industrial-plant_0.jpg

https://lh3.googleusercontent.com/hfrYnyPUjVbejSwdmrDlr2FK14oqzJ6wcU5ZoaEhg31Wq9aZOF0ymgb-NoVgGJ4zZxtmmjw=s116

https://thumbs.dreamstime.com/b/man-clipboard-wearing-hardhat-man-clipboard-wearing-hardhat-man-122033980.jpg

https://image.shutterstock.com/image-photo/man-hard-hat-260nw-63941569.jpg

https://thumbs.dreamstime.com/b/foreman-7961915.jpg

https://pbs.twimg.com/media/CjQKlELUkAAEMnP.jpg:large

https://image.shutterstock.com/image-photo/man-wearing-blue-hardhat-using-600w-546698692.jpg

https://cdn.xl.thumbs.canstockphoto.com/man-wearing-hard-hat-and-construction-vest-stock-photo_csp45383744.jpg

https://thumbs.dreamstime.com/b/maskinarkitekt-med-h%C3%A5rd-hatt-som-inspekterar-nya-byggnader-171099972.jpg

https://thumbs.dreamstime.com/b/men-hardhats-9617544.jpg

https://ak.picdn.net/offset/photos/5f88a7b06f52af4409e4bc42/medium/offset_1019720.jpg?DFghwDcb?DFghwDcb

https://img.freepik.com/foto-gratis/mujer-ingeniero-vistiendo-hardhat-sostener-tableta-digital_8595-11583.jpg?size=626&ext=jpg

https://w7.pngwing.com/pngs/345/827/png-transparent-bicycle-helmets-ski-snowboard-helmets-equestrian-helmets-hard-hats-construction-foreman-bicycle-helmets-hat-engineer-sports.png

https://img2.pngdownload.id/20190801/fos/kisspng-construction-worker-laborer-industry-carpenter-urcae-universal-research-of-civil-amp-architec-5d437be621e2e5.188615941564703718 1388.jpg

https://c8.alamy.com/comp/J2N5E6/portrait-of-a-young-engineer-wearing-a-helmet-against-metal-background-J2N5E6.jpg

https://www.karebasulsel.id/wp-content/uploads/2020/12/Pengertian-Penting-Keselamatan-Saat-Kerja.jpg

https://cdn.redshift.autodesk.com/2020/06/smart-hard-hat-construction-site.jpg

https://images1.westend61.de/0001391225l/thoughtful-mature-male-manager-wearing-yellow-hardhat-in-factory-RORF02202.jpg

https://mh-1-stockagency.panthermedia.net/media/previews/0015000000/15477000/15477023_high.jpg

https://img3.stockfresh.com/files/e/elnur/m/41/6078285_stock-photo-woman-wearing-hard-hat-isolated-on-white.jpg

https://thumbs.dreamstime.com/z/young-worker-wearing-hard-hat-10560915.jpg

Mnistdeepauto.py

```
from rbm import rbm
from backprop import backprop
from rbmhidlinear import rbmhidlinear
import scipy.io as sio
from mnistdisp import mnistdisp

MAX_EPOCH = 10
NUM_HID = 1000 # n^1
NUMPEN = 500 # n^2
NUMPEN2 = 250 # n^3
NUMOPEN = 30 # n^4
# W^l
RANDOMIN_ = sio.loadmat("randomin_poshidstates.mat", verify_compressed_
data_integrity=False)
RANDOMIN = RANDOMIN_['randomin']
# W_ij
VISHID_ = sio.loadmat("vis_try___.mat", verify_compressed_data_integrit
y=False)
VISHID = VISHID_['vishid_']

# p̂
BATCH_DATA_ = sio.loadmat("batchdata_py.mat", verify_compressed_data_in
tegrity=False)
BATCH_DATA = BATCH_DATA_['batchdata']
NUM_CASES, NUM_DIMS, NUM_BATCHES = BATCH_DATA.shape
# print("BATCH_DATA.shape= ", BATCH_DATA.shape) #100, 784, 600

print('Pretraining Layer 1 with RBM: {}-{}'.format(NUM_DIMS, NUM_HID))
RESTART = 1

BATCHPOSHIDPROBS, VISHID, HIDBIASES, VISBIASES = rbm(BATCH_DATA, RESTAR
T, NUM_HID, NUM_DIMS, MAX_EPOCH, RANDOMIN, VISHID)
print("BATCHPOSHIDPROBS.shape= ", BATCHPOSHIDPROBS.shape) #100, 1000
HIDRECBIASES = HIDBIASES
VISHID__= VISHID
VISBIASES__ = VISBIASES

print('Pretraining Layer 2 with RBM: {}-{}'.format(NUM_HID, NUMPEN))
BATCH_DATA = BATCHPOSHIDPROBS
NUM_HID = NUMPEN
RESTART = 1
BATCHPOSHIDPROBS, VISHID, HIDBIASES, VISBIASES = rbm(BATCH_DATA, RESTAR
T, NUM_HID, NUM_DIMS, MAX_EPOCH, RANDOMIN, VISHID)
```

```
HIDPEN = VISHID
PENRECBIASES = HIDBIASES
HIDGENBIASES = VISBIASES

print('Pretraining Layer 3 with RBM: {}-{}'.format(NUMPEN, NUMPEN2))
BATCH_DATA = BATCHPOSHIDPROBS
NUM_HID = NUMPEN2
RESTART = 1
# # RBMUpdate(p̂,n^1,W^l,
BATCHPOSHIDPROBS, VISHID, HIDBIASES, VISBIASES = rbm(BATCH_DATA, RESTAR
T, NUM_HID, NUM_DIMS, MAX_EPOCH, RANDOMIN, VISHID)
HIDPEN2 = VISHID
PENRECBIASES2 = HIDBIASES
HIDGENBIASES2 = VISBIASES

print('Pretraining Layer 4 with RBM: {}-{}'.format(NUMPEN2, NUMOPEN))
BATCH_DATA = BATCHPOSHIDPROBS
NUM_HID = NUMOPEN
RESTART = 1
VISHID, HIDBIASES, VISBIASES = rbmhidlinear(BATCH_DATA, RESTART, NUM_HI
D, NUM_DIMS, MAX_EPOCH, RANDOMIN, VISHID)
HIDTOP = VISHID
TOPRECBIASES = HIDBIASES
TOPGENBIASES = VISBIASES

print('END of Pretraining Layer 4 with RBM: {}-
{}'.format(NUMPEN2, NUMOPEN))

ERR = backprop(VISHID__, VISBIASES__, PENRECBIASES, PENRECBIASES2, HIDR
ECBIASES, HIDPEN, HIDPEN2, HIDGENBIASES, HIDGENBIASES2, HIDTOP, TOPRECB
IASES, TOPGENBIASES)
```

rbm.py

```
    # ϵ_W
    EPSILON_W = 0.1
    # ϵ_VB
    EPSILON_VB = 0.1
    # ϵ_HB
    EPSILON_HB = 0.1
    # β
    WEIGHT_COST = 0.0002
    # θ_1
```

```
    INITIAL_MOMENTUM = 0.5
    # θ_t
    FINAL_MOMENTUM = 0.9

    NUM_CASES, NUM_DIMS, NUM_BATCHES = BATCH_DATA.shape

    if RESTART==1:
        RESTART = 0
        EPOCH = 0
        # w_ij
        VISHID = 0.1 * np.random.randn(NUM_DIMS, NUM_HID)
        # b
        HIDBIASES = np.zeros(NUM_HID)
        # c
        VISBIASES = np.zeros(NUM_DIMS)
        p(h_j = 1)
        POSHIDPROBS = np.zeros((NUM_CASES, NUM_HID))
        Q(h_j = 1)
        NEGHIDPROBS = np.zeros((NUM_CASES, NUM_HID))
        # g^l' x p(h_j = 1)
        POSPRODS = np.zeros((NUM_DIMS, NUM_HID))
        # Q(h_j = 1)' x σ(x)
        NEGPRODS = np.zeros((NUM_DIMS, NUM_HID))
        # Δw_ij
        VISHIDINC = np.zeros((NUM_DIMS, NUM_HID))
        # Δb
        HIDBIASINC = np.zeros(NUM_HID)
        # Δc
        VISBIASINC = np.zeros(NUM_DIMS)
        BATCHPOSHIDPROBS = np.zeros((NUM_CASES, NUM_HID, NUM_BATCHES))

    for epoch in range(EPOCH, MAX_EPOCH):
        print('epoch {}'.format(epoch))
        ERR_SUM = 0
        # for l = 0 to L = 599
        for batch in range(NUM_BATCHES):
            print('epoch {} batch {}'.format(epoch,batch))
            # g^l
            data = BATCH_DATA[:,:,batch]
            # p(h_j = 1) = 1/(1 + e^(v_i w_ij))
            POSHIDPROBS = 1.0/(1 + (np.exp(np.matmul(-
data,VISHID)- np.tile(HIDBIASES, (NUM_CASES, 1)))))
            BATCHPOSHIDPROBS[:,:,batch] = POSHIDPROBS
```

```
            # g^l' x p(h_j = 1)
            POSPRODS = np.matmul(data.T,POSHIDPROBS)
            # ∑p(h_j = 1)
            POSHIDACT = np.sum(POSHIDPROBS, axis=0)
            # ∑g^l
            POSVISACT = np.sum(data, axis=0)
            # σ(x) = σ(b_j + ∑iv_i w_ij) = 1/[1 + exp(−(b_j + ∑iv_i w_ij))]
```

```
            # page 5, practical guide
            # hidden unit turns on if this probability is greater than
a random number uniformly distributed between 0 and 1
            # h_j = {0, σ(x) < random
                     1, σ(x) > random
            POSHIDSTATES = np.where(POSHIDPROBS > np.random.randn(NUM_C
ASES, NUM_HID), 1, 0)

            # Q(h_j = 1) = 1/(1 + e^(v_i w_ij))
            NEGDATA = 1.0/(1 + np.exp(np.matmul(-
POSHIDSTATES, VISHID.T) - np.tile(VISBIASES, (NUM_CASES, 1))))
            # σ(x) = σ(b_j + ∑iv_i w_ij) = 1/[1 + exp(−x)]
            NEGHIDPROBS = 1.0/(1 + np.exp(np.matmul(-
NEGDATA, VISHID) - np.tile(HIDBIASES, (NUM_CASES,1))))
            # Q(h_j = 1)' x σ(x)
            NEGPRODS = np.matmul(NEGDATA.T, NEGHIDPROBS)
            ∑σ(x)
            NEGHIDACT = np.sum(NEGHIDPROBS, axis=0)
            ∑p(h_j = 1)
            NEGVISACT = np.sum(NEGDATA, axis=0)
            # C = ∑∑ (g^l − Q(h_j = 1))^2
            ERR = np.sum(np.sum(np.square(data-NEGDATA)))
            ERR_SUM += ERR

            if epoch>5:
                MOMENTUM = FINAL_MOMENTUM
            else:
                MOMENTUM = INITIAL_MOMENTUM


            # Δw_ij = ϵ(⟨v_i h_j⟩_data − ⟨v_i h_j⟩_recon)
            VISHIDINC = MOMENTUM* VISHIDINC + EPSILON_W  * ((POSPRODS-
NEGPRODS)/NUM_CASES - WEIGHT_COST*VISHID)
            # Δc^i = ϵ(⟨v_i h_j⟩_data − ⟨v_i h_j⟩_recon)
```

```
            VISBIASINC = MOMENTUM* VISBIASINC + (EPSILON_VB/NUM_CASES)
*(POSVISACT-NEGVISACT)
            # Δb^i = ϵ(⟨v_i h_j⟩_data − ⟨v_i h_j⟩_recon)
            HIDBIASINC = MOMENTUM * HIDBIASINC + (EPSILON_HB/NUM_CASES)
* (POSHIDACT-NEGHIDACT)
            # W^l = W^(l−1) + ΔW^l
            VISHID += VISHIDINC
            # c^l = c^(l−1) + Δc^i
            VISBIASES += VISBIASINC
            # b^l = b^(l−1) + Δb^i
            HIDBIASES += HIDBIASINC
        print('epoch {} error {}'.format(epoch, ERR_SUM))
    return BATCHPOSHIDPROBS, VISHID, HIDBIASES, VISBIASES
```

rbmlinear.py

```
def rbmhidlinear(BATCH_DATA, RESTART, NUM_HID, NUM_DIMS, MAX_EPOCH, RAN
DOMIN, VISHID):
    import scipy.io as sio
    import numpy as np

    # ϵ_W
    EPSILON_W = 0.001
    # ϵ_VB
    EPSILON_VB = 0.001
    # ϵ_HB
    EPSILON_HB = 0.001
    # β
    WEIGHT_COST = 0.0002
    # θ_1
    INITIAL_MOMENTUM = 0.5
    # θ_t
    FINAL_MOMENTUM = 0.9

    print(BATCH_DATA.shape)
    NUM_CASES, NUM_DIMS, NUM_BATCHES = BATCH_DATA.shape
    RANDOMIN = np.random.randn(NUM_CASES, NUM_HID)

    if RESTART==1:
        RESTART = 0
        EPOCH = 1
        # w_ij
        VISHID = 0.1 * np.random.randn(NUM_DIMS, NUM_HID)
```

```
        # b
        HIDBIASES = np.zeros(NUM_HID)
        # c
        VISBIASES = np.zeros(NUM_DIMS)
        p(h_j = 1)
        POSHIDPROBS = np.zeros((NUM_CASES, NUM_HID))
        Q(h_j = 1)
        NEGHIDPROBS = np.zeros((NUM_CASES, NUM_HID))
        # g^l' x p(h_j = 1)
        POSPRODS = np.zeros((NUM_DIMS, NUM_HID))
        # Q(h_j = 1)' x σ(x)
        NEGPRODS = np.zeros((NUM_DIMS, NUM_HID))
        # Δw_ij
        VISHIDINC = np.zeros((NUM_DIMS, NUM_HID))
        # Δb
        HIDBIASINC = np.zeros(NUM_HID)
        # Δc
        VISBIASINC = np.zeros(NUM_DIMS)
        BATCHPOSHIDPROBS = np.zeros((NUM_CASES, NUM_HID, NUM_BATCHES))

    for epoch in range(EPOCH, MAX_EPOCH):
        print('epoch {}'.format(epoch))
        ERR_SUM = 0
        # for l = 0 to L = 599
        for batch in range(NUM_BATCHES):
            print('epoch {} batch {}'.format(epoch,batch))
            # g^l
            data = BATCH_DATA[:,:,batch]
            # p(h_j = 1) = 1/(1 + e^(v_i w_ij))
            POSHIDPROBS = 1.0/(1 + (np.exp(np.matmul(-
data,VISHID)- np.tile(HIDBIASES, (NUM_CASES, 1))
)))
            BATCHPOSHIDPROBS[:,:,batch] = POSHIDPROBS
            # g^l' x p(h_j = 1)
            POSPRODS = np.matmul(data.T,POSHIDPROBS)
            # ∑p(h_j = 1)
            POSHIDACT = np.sum(POSHIDPROBS, axis=0)
            # ∑g^l
            POSVISACT = np.sum(data, axis=0)
            # h_j = p(h_j = 1) + randomin
            POSHIDSTATES = POSHIDPROBS + RANDOMIN
            # Q(h_j = 1) = 1/(1 + e^(v_i w_ij))
```

```
            NEGDATA = 1.0/(1 + np.exp(np.matmul(-
POSHIDSTATES, VISHID.T) - np.tile(VISBIASES, (NUM_CASES, 1))))
            # σ(x) = σ(b_j + ∑ iv_i w_ij = 1/[1 + exp(−x)]
            NEGHIDPROBS = 1.0/(1 + np.exp(np.matmul(-
NEGDATA, VISHID) - np.tile(HIDBIASES, (NUM_CASES,1))))
            # Q(h_j = 1)' x σ(x)
            NEGPRODS = np.matmul(NEGDATA.T, NEGHIDPROBS)
            ∑ σ(x)
            NEGHIDACT = np.sum(NEGHIDPROBS, axis=0)
            ∑ p(h_j = 1)
            NEGVISACT = np.sum(NEGDATA, axis=0)
            # C = ∑∑ (g^l − Q(h_j = 1))^2
            ERR = np.sum(np.sum(np.square(data-
NEGDATA), axis=0), axis=0)
            ERR_SUM += ERR

            if epoch>5:
                MOMENTUM = FINAL_MOMENTUM
            else:
                MOMENTUM = INITIAL_MOMENTUM
            # Δw_ij = θ * Δw_ij + ε_W * (
            VISHIDINC = MOMENTUM* VISHIDINC + EPSILON_W  * ((POSPRODS-
NEGPRODS)/NUM_CASES - WEIGHT_COST*VISHID)
            # Δc^i = θ * Δc^i + (ε_VB/N) * (∑ g^l − ∑ p(h_j = 1))
            VISBIASINC = MOMENTUM* VISBIASINC + (EPSILON_VB/NUM_CASES)
*(POSVISACT-NEGVISACT)
            # Δb^i = θ * Δb^i + (ε_HB/N) * ( ∑ p(h_j = 1) − ∑ σ(x))
            HIDBIASINC = MOMENTUM * HIDBIASINC + (EPSILON_HB/NUM_CASES)
* (POSHIDACT-NEGHIDACT)
            # W^l = W^(l−1) + ΔW^l
            VISHID += VISHIDINC
            # c^l = c^(l−1) + Δc^i
            VISBIASES += VISBIASINC
            # b^l = b^(l−1) + Δb^i
            HIDBIASES += HIDBIASINC
        print('epoch {} error {}'.format(epoch, ERR_SUM))
    return VISHID, HIDBIASES, VISBIASES
```

backprop.py

```
def backprop(VISHID, VISBIASES, PENRECBIASES, PENRECBIASES2, HIDRECBIAS
ES, HIDPEN, HIDPEN2, HIDGENBIASES, HIDGENBIASES2, HIDTOP, TOPRECBIASES,
 TOPGENBIASES):
# def backprop():
    import numpy as np
    import scipy.io as sio
    from makebatches import makebatches
    from mnistdisp import mnistdisp
    from minimize import minimize
    from CG_MNIST import CG_MNIST

    MAX_EPOCH = 200
    print('Fine-
tuning deep autoencoder by minimizing cross entropy error.')
    print('60 batches of 1000 cases each.')

    BATCH_DATA_ = sio.loadmat("batchdata_py.mat", verify_compressed_dat
a_integrity=False)
    BATCHDATA = BATCH_DATA_['batchdata']
    TEST_BATCH_DATA_ = sio.loadmat("testbatchdata_py.mat", verify_compr
essed_data_integrity=False)
    TESTBATCHDATA = TEST_BATCH_DATA_['testbatchdata']

    VISHID_ = sio.loadmat("vishid_mnistvh.mat", verify_compressed_data_
integrity=False)
    VISHID = VISHID_['vishid_']
    # w1
    W1 = np.append(VISHID, HIDRECBIASES.reshape(1, -1), axis = 0)
    # w2
    W2 = np.append(HIDPEN, PENRECBIASES.reshape(1, -1), axis = 0)
    # w3
    W3 = np.append(HIDPEN2, PENRECBIASES2.reshape(1, -1), axis = 0)
    # w4
    W4 = np.append(HIDTOP, TOPRECBIASES.reshape(1, -1), axis = 0)
    # w5
    W5 = np.append(HIDTOP.T, TOPGENBIASES.reshape(1, -1), axis = 0)
    # w6
    W6 = np.append(HIDPEN2.T, HIDGENBIASES2.reshape(1, -1), axis = 0)
    # w7
    W7 = np.append(HIDPEN.T, HIDGENBIASES.reshape(1, -1), axis = 0)
    # w8
    W8 = np.append(VISHID.T, VISBIASES.reshape(1, -1), axis = 0)
```

```
    L1 = W1.shape[0]-1
    L2 = W2.shape[0]-1
    L3 = W3.shape[0]-1
    L4 = W4.shape[0]-1
    L5 = W5.shape[0]-1
    L6 = W6.shape[0]-1
    L7 = W7.shape[0]-1
    L8 = W8.shape[0]-1
    L9 = L1

    TEST_ERR=[]
    TRAIN_ERR=[]

    for epoch in range(1, MAX_EPOCH):
        ERR = 0
        NUM_CASES, NUM_DIMS, NUM_BATCHES = BATCHDATA.shape
        N = NUM_CASES
        for batch in range(1, NUM_BATCHES):
            # g^l
            data = BATCHDATA[:,:,batch]
            data = np.append(data, np.ones((N, 1)), axis=1)
            # W1_PROBS = 1/(1 + exp(−g^l x W1)
            W1_PROBS = 1.0/(1 + np.exp(np.matmul(-data, W1)))
            W1_PROBS = np.append(W1_PROBS, np.ones((N, 1)), axis=1)
            # W2_PROBS = 1/(1 + exp(−W1_PROBS x W2)
            W2_PROBS = 1.0/(1 + np.exp(np.matmul(-W1_PROBS, W2)))
            W2_PROBS = np.append(W2_PROBS, np.ones((N, 1)), axis=1)
            # W3_PROBS = 1/(1 + exp(−W2_PROBS x W3)
            W3_PROBS = 1.0/(1 + np.exp(np.matmul(-W2_PROBS,W3)))
            W3_PROBS = np.append(W3_PROBS, np.ones((N, 1)), axis=1)
            # W4_PROBS = 1/(1 + exp(−W3_PROBS x W4)
            W4_PROBS = np.matmul(W3_PROBS, W4)
            W4_PROBS = np.append(W4_PROBS, np.ones((N, 1)), axis=1)
            # W5_PROBS = 1/(1 + exp(−W4_PROBS x W5)
            W5_PROBS = 1.0/(1 + np.exp(np.matmul(-W4_PROBS,W5)))
            W5_PROBS = np.append(W5_PROBS, np.ones((N, 1)), axis=1)
            # W6_PROBS = 1/(1 + exp(−W5_PROBS x W6)
            W6_PROBS = 1.0/(1 + np.exp(np.matmul(-W5_PROBS, W6)))
            W6_PROBS = np.append(W6_PROBS, np.ones((N, 1)), axis=1)
            # W7_PROBS = 1/(1 + exp(−W6_PROBS x W7)
            W7_PROBS = 1.0/(1 + np.exp(np.matmul(-W6_PROBS, W7)))
            W7_PROBS = np.append(W7_PROBS, np.ones((N, 1)), axis=1)
            # g^l_recon = 1/(1 + exp(−W7_PROBS x W8)
```

```
            DATAOUT = 1.0/(1 + np.exp(np.matmul(-W7_PROBS, W8)))
            ERR += 1/N*np.sum(np.sum(np.square(data[:,:-1]-
DATAOUT), axis=0), axis=0)
        TRAIN_ERR = ERR/NUM_BATCHES

        print('Displaying in figure 1: Top row - real data, Bottom row
-- reconstructions')
        OUTPUT = np.array([])
        for ii in range(15):
            A = np.append(data[ii, :-1].T, DATAOUT[ii, :].T)
            A = A.reshape(784, 2)
            OUTPUT = np.append(OUTPUT, A)
        # mnistdisp(OUTPUT)
        # plt.show()

        TESTNUMCASES, TESTNUMDIMS, TESTNUMBATCHES = TESTBATCHDATA.shape
        N = TESTNUMCASES
        ERR = 0
        for batch in range(1, TESTNUMBATCHES):
            data = TESTBATCHDATA[:,:,batch]
            data = np.append(data, np.ones((N, 1)), axis=1)

            # W1_PROBS = 1/(1 + exp(-g^l x W_1)
            W1_PROBS = 1.0/(1 + np.exp(np.matmul(-data, W1)))
            W1_PROBS = np.append(W1_PROBS, np.ones((N, 1)), axis=1)
            # W2_PROBS = 1/(1 + exp(-W1_PROBS x W_2)
            W2_PROBS = 1.0/(1 + np.exp(np.matmul(-W1_PROBS, W2)))
            W2_PROBS = np.append(W2_PROBS, np.ones((N, 1)), axis=1)
            # W3_PROBS = 1/(1 + exp(-W2_PROBS x W_3)
            W3_PROBS = 1.0/(1 + np.exp(np.matmul(-W2_PROBS,W3)))
            W3_PROBS = np.append(W3_PROBS, np.ones((N, 1)), axis=1)
            # W4_PROBS = 1/(1 + exp(-W3_PROBS x W_4)
            W4_PROBS = np.matmul(W3_PROBS, W4)
            W4_PROBS = np.append(W4_PROBS, np.ones((N, 1)), axis=1)
            # W5_PROBS = 1/(1 + exp(-W4_PROBS x W_5)
            W5_PROBS = 1.0/(1 + np.exp(np.matmul(-W4_PROBS,W5)))
            W5_PROBS = np.append(W5_PROBS, np.ones((N, 1)), axis=1)
            # W6_PROBS = 1/(1 + exp(-W5_PROBS x W_6)
            W6_PROBS = 1.0/(1 + np.exp(np.matmul(-W5_PROBS, W6)))
            W6_PROBS = np.append(W6_PROBS, np.ones((N, 1)), axis=1)
            # W7_PROBS = 1/(1 + exp(-W6_PROBS x W_7)
            W7_PROBS = 1.0/(1 + np.exp(np.matmul(-W6_PROBS, W7)))
            W7_PROBS = np.append(W7_PROBS, np.ones((N, 1)), axis=1)
```

```
            # g^l_recon = 1/(1 + exp(−W_7PROBS x W_8)
            DATAOUT = 1.0/(1 + np.exp(np.matmul(-W7_PROBS, W8)))
            ERR += 1/N*np.sum(np.sum(np.square(data[:,:-1]-
DATAOUT), axis=0), axis=0)
        TEST_ERR = ERR/NUM_BATCHES
        print('Before epoch {} Train squared error: {} Test squared err
or: {}'.format(epoch,TRAIN_ERR,TEST_ERR))

        TT = 0
        for batch in range(int(NUM_BATCHES/10)):
            print('epoch {} batch {}'.format(epoch,batch))
            TT+=1
            data=np.empty((100,784), int)
            for kk in range(10):
                data = np.append(data, BATCHDATA[:,:,((TT-
1)*10+kk)], axis=0)

            MAX_ITER = 3
            VV = np.concatenate((W1.reshape(1,-1), W2.reshape(1,-
1), W3.reshape(1,-1),
                    W4.reshape(1,-1), W5.reshape(1,-1), W6.reshape(1, -
1), W7.reshape(1, -1),
                    W8.reshape(1, -1)), axis=1)
            DIM = np.array([L1, L2, L3, L4, L5, L6, L7, L8, L9]).reshap
e(1, -1).T

            f, df = CG_MNIST(VV, DIM, data)
            X, fX, i = minimize(VV,f, df, MAX_ITER, DIM, data, 1.0, Tru
e)

            W1 = X[0][0:(L1+1)*L2].reshape(L1+1,L2)
            X3 = (L1+1)*L2
            W2 = X[0][X3:X3+(L2+1)*L3].reshape(L2+1, L3)
            X3 = X3+(L2+1)*L3
            W3 = X[0][X3:X3+(L3+1)*L4].reshape(L3+1, L4)
            X3 = X3+(L3+1)*L4
            W4 = X[0][X3:X3+(L4+1)*L5].reshape(L4+1, L5)
            X3 = X3+(L4+1)*L5
            W5 = X[0][X3:X3+(L5+1)*L6].reshape(L5+1, L6)
            X3 = X3+(L5+1)*L6
            W6 = X[0][X3:X3+(L6+1)*L7].reshape(L6+1, L7)
            X3 = X3+(L6+1)*L7
            W7 = X[0][X3:X3+(L7+1)*L8].reshape(L7+1, L8)
```

```
            X3 = X3+(L7+1)*L8
            W8 = X[0][X3:X3+(L8+1)*L9].reshape(L8+1, L9)
    return ERR


# if __name__ == "__main__":
#     backprop()
```

CG_MNIST.py

```
def CG_MNIST(VV, Dim, X2):
    import numpy as np
    L1 = Dim[0]
    L2 = Dim[1]
    L3 = Dim[2]
    L4 = Dim[3]
    L5 = Dim[4]
    L6 = Dim[5]
    L7 = Dim[6]
    L8 = Dim[7]
    L9 = Dim[8]
    N = X2.shape[0]

    # Do decomversion
    W1 = VV[0][0:(L1[0]+1)*L2[0]].reshape(L1[0]+1,L2[0])
    X3 = (L1[0]+1)*L2[0]
    W2 = VV[0][X3:X3+(L2[0]+1)*L3[0]].reshape(L2[0]+1, L3[0])
    X3 = X3+(L2[0]+1)*L3[0]
    W3 = VV[0][X3:X3+(L3[0]+1)*L4[0]].reshape(L3[0]+1, L4[0])
    X3 = X3+(L3[0]+1)*L4[0]
    W4 = VV[0][X3:X3+(L4[0]+1)*L5[0]].reshape(L4[0]+1, L5[0])
    X3 = X3+(L4[0]+1)*L5[0]
    W5 = VV[0][X3:X3+(L5[0]+1)*L6[0]].reshape(L5[0]+1, L6[0])
    X3 = X3+(L5[0]+1)*L6[0]
    W6 = VV[0][X3:X3+(L6[0]+1)*L7[0]].reshape(L6[0]+1, L7[0])
    X3 = X3+(L6[0]+1)*L7[0]
    W7 = VV[0][X3:X3+(L7[0]+1)*L8[0]].reshape(L7[0]+1, L8[0])
    X3 = X3+(L7[0]+1)*L8[0]
    W8 = VV[0][X3:X3+(L8[0]+1)*L9[0]].reshape(L8[0]+1, L9[0])

    X2 = np.append(X2, np.ones((N, 1)), axis=1)
    # W1_PROBS = 1/(1 + exp(-X2 x W1)
    W1_PROBS = 1.0/(1 + np.exp(np.matmul(-X2, W1)))
```

```
    W1_PROBS = np.append(W1_PROBS, np.ones((N, 1)), axis=1)
    # W2_PROBS = 1/(1 + exp(−W1_PROBS x W1)
    W2_PROBS = 1.0/(1 + np.exp(np.matmul(-W1_PROBS, W2)))
    W2_PROBS = np.append(W2_PROBS, np.ones((N, 1)), axis=1)
    # W3_PROBS = 1/(1 + exp(−W2_PROBS x W1)
    W3_PROBS = 1.0/(1 + np.exp(np.matmul(-W2_PROBS,W3)))
    W3_PROBS = np.append(W3_PROBS, np.ones((N, 1)), axis=1)
    # W4_PROBS = 1/(1 + exp(−W3_PROBS x W1)
    W4_PROBS = np.matmul(W3_PROBS, W4)
    W4_PROBS = np.append(W4_PROBS, np.ones((N, 1)), axis=1)
    # W5_PROBS = 1/(1 + exp(−W4_PROBS x W1)
    W5_PROBS = 1.0/(1 + np.exp(np.matmul(-W4_PROBS,W5)))
    W5_PROBS = np.append(W5_PROBS, np.ones((N, 1)), axis=1)
    # W6_PROBS = 1/(1 + exp(−W5_PROBS x W1)
    W6_PROBS = 1.0/(1 + np.exp(np.matmul(-W5_PROBS, W6)))
    W6_PROBS = np.append(W6_PROBS, np.ones((N, 1)), axis=1)
    # W7_PROBS = 1/(1 + exp(−W6_PROBS x W1)
    W7_PROBS = 1.0/(1 + np.exp(np.matmul(-W6_PROBS, W7)))
    W7_PROBS = np.append(W7_PROBS, np.ones((N, 1)), axis=1)
    # g_RECON = 1/(1 + exp(−W7_PROBS x W1)
    X2_OUT = 1.0/(1 + np.exp(np.matmul(-W7_PROBS, W8)))

    f = -1/N*np.sum(np.sum(X2[:,:-1]*np.log(X2_OUT)+ (1-X2[:, :-
1])*np.log(1-X2_OUT)))
    IO = 1/N*(X2_OUT-X2[:,:-1])
    Ix8 = IO
    # W7_PROBS x IX
    DW8 = np.matmul(W7_PROBS.T,Ix8)

    Ix7 = (np.matmul(Ix8,W8.T))*W7_PROBS*(1-W7_PROBS)
    Ix7 = Ix7[:, :-1]
    DW7 = np.matmul(W6_PROBS.T,Ix7)

    Ix6 = (np.matmul(Ix7, W7.T))*W6_PROBS*(1-W6_PROBS)
    Ix6 = Ix6[:, :-1]
    DW6 = np.matmul(W5_PROBS.T,Ix6)

    Ix5 = (np.matmul(Ix6, W6.T))*W5_PROBS*(1-W5_PROBS)
    Ix5 = Ix5[:, :-1]
    DW5 = np.matmul(W4_PROBS.T,Ix5)

    Ix4 = np.matmul(Ix5, W5.T)
    Ix4 = Ix4[:, :-1]
```

```
    DW4 = np.matmul(W3_PROBS.T,Ix4)

    Ix3 = np.matmul(Ix4, W4.T)*W3_PROBS*(1-W3_PROBS)
    Ix3 = Ix3[:, :-1]
    DW3 = np.matmul(W2_PROBS.T,Ix3)

    Ix2 = np.matmul(Ix3, W3.T)*W2_PROBS*(1-W2_PROBS)
    Ix2 = Ix2[:, :-1]
    DW2 = np.matmul(W1_PROBS.T, Ix2)

    Ix1 = np.matmul(Ix2, W2.T)*W1_PROBS*(1-W1_PROBS)
    Ix1 = Ix1[:, :-1]
    DW1 = np.matmul(X2.T, Ix1)

    # df = lst.extend([DW1[:].T, DW2[:].T, DW3[:].T, DW4[:].T, DW5[:].T
, DW6[:].T, DW7[:].T, DW8[:].T]).T
    df = np.concatenate((DW1.reshape(1,-1), DW2.reshape(1,-
1), DW3.reshape(1,-1),
                DW4.reshape(1,-1), DW5.reshape(1,-1), DW6.reshape(1, -
1), DW7.reshape(1, -1),
                DW8.reshape(1, -1)), axis=1)
    return f, df
```

minimize.py

```
from math import *
from numpy import dot, isinf, isnan, any, sqrt, isreal, real, nan, inf
from CG_MNIST import CG_MNIST

def minimize(X, f0, df0, length, dim, data, red=1.0, verbose=True):
    INT = 0.1
    EXT = 3.0
    MAX = 20
    RATIO = 10
    SIG = 0.1
    RHO = SIG/2

    SMALL = 10.**-16

    i = 0                                           # zero the run length
 counter
    ls_failed = 0                             # no previous line search ha
s failed
```

```
    # f0 = f(X, *args)                          # get function value an
d gradient
    # df0 = grad(X, *args)
    fX = [f0]
    i = i + (length<0)                                      # count
epochs?!
    s = -df0; d0 = -
dot(s,s.T)    # initial search direction (steepest) and slope
    x3 = red/(1.0-
d0)                            # initial step is red/(|s|+1)

    while i < abs(length):                                  # while not
finished
        i = i + (length>0)                                  # count iter
ations?!

        X0 = X; F0 = f0; dF0 = df0              # make a copy of curren
t values
        if length>0:
            M = MAX
        else:
            M = min(MAX, -length-i)
        ME = 1
        while ME!=0:                       # keep extrapolating as long
as necessary
            x2 = 0; f2 = f0; d2 = d0; f3 = f0; df3 = df0
            success = 0
            while (not success) and (M > 0):
                try:
                    M = M - 1; i = i + (length<0)              # count
epochs?!
                    # f3 = f(X+x3*s, *args)
                    # df3 = grad(X+x3*s, *args)
                    f3, df3 = CG_MNIST(X+x3*s, dim, data)
                    if isnan(f3) or isinf(f3) or any(isnan(df3)+isinf(d
f3)):
                        print("error")
                    success = 1
                except:                     # catch any error which occu
red in f
                    x3 = (x2+x3)/2                      # bisect and t
ry again
            if f3 < F0:
```

```
                X0 = X+x3*s; F0 = f3; dF0 = df3   # keep best values
            d3 = dot(df3,s.T)                                          #
 new slope
            if (d3 > SIG*d0)[0][0] or (f3 > f0+x3*RHO*d0)[0][0] or M ==
 0:
                ME = 0
            x1 = x2; f1 = f2; d1 = d2                 # move point 2 to
 point 1
            x2 = x3; f2 = f3; d2 = d3                 # move point 3 to
 point 2
            A = 6*(f1-f2)+3*(d2+d1)*(x2-
x1)          # make cubic extrapolation
            B = 3*(f2-f1)-(2*d1+d2)*(x2-x1)
            Z = B+sqrt(complex(B*B-A*d1*(x2-x1)))
            if Z != 0.0:
                x3 = x1-d1*(x2-
x1)**2/Z             # num. error possible, ok!
            else:
                x3 = inf
            if (not isreal(x3)) or isnan(x3) or isinf(x3) or (x3 < 0):
                                                  # num prob | wro
ng sign?
                x3 = x2*EXT                         # extrapolate maximu
m amount
            elif x3 > x2*EXT:           # new point beyond extrapolatio
n limit?
                x3 = x2*EXT                         # extrapolate maximu
m amount
            elif x3 < x2+INT*(x2-
x1):  # new point too close to previous point?
                x3 = x2+INT*(x2-x1)
            x3 = real(x3)

        while (abs(d3) > -SIG*d0 or f3 > f0+x3*RHO*d0) and M > 0:
                                                          # keep inter
polating
            if (d3 > 0) or (f3 > f0+x3*RHO*d0):            # choose sub
interval
                x4 = x3; f4 = f3; d4 = d3             # move point 3 to
 point 4
            else:
                x2 = x3; f2 = f3; d2 = d3             # move point 3 to
 point 2
```

```
            if f4 > f0:
                x3 = x2-(0.5*d2*(x4-x2)**2)/(f4-f2-d2*(x4-x2))
                                                    # quadratic inter
polation
            else:
                A = 6*(f2-f4)/(x4-
x2)+3*(d4+d2)           # cubic interpolation
                B = 3*(f4-f2)-(2*d2+d4)*(x4-x2)
                if A != 0:
                    x3=x2+(sqrt(B*B-A*d2*(x4-x2)**2)-B)/A
                                                   # num. error possi
ble, ok!
                else:
                    x3 = inf
            if isnan(x3) or isinf(x3):
                x3 = (x2+x4)/2      # if we had a numerical problem the
n bisect
            x3 = max(min(x3, x4-INT*(x4-x2)),x2+INT*(x4-x2))
                                                     # don't accept t
oo close
            f3, df3 = CG_MNIST(X+x3*s, dim, data)
            if f3 < F0:
                X0 = X+x3*s; F0 = f3; dF0 = df3              # keep bes
t values
            M = M - 1; i = i + (length<0)                      # count
epochs?!
            d3 = dot(df3,s.T)                                          #
 new slope

        if abs(d3) < -
SIG*d0 and f3 < f0+x3*RHO*d0:  # if line search succeeded
            X = X+x3*s; f0 = f3; fX.append(f0)               # update v
ariables
            if verbose: print('%s %6i;  Value %4.6e\r' % (S, i, f0))
            s = (dot(df3,df3)-dot(df0,df3))/dot(df0,df0)*s - df3
                                                  # Polack-
Ribiere CG direction
            df0 = df3                                        # swap der
ivatives
            d3 = d0; d0 = dot(df0,s)
            if d0 > 0:                                # new slope must be
negative
```

```
                s = -df0; d0 = -
dot(s,s)     # otherwise use steepest direction
            x3 = x3 * min(RATIO, d3/(d0-
SMALL))     # slope ratio but max RATIO
            ls_failed = 0                       # this line search did
not fail
        else:
            X = X0; f0 = F0; df0 = dF0              # restore best poin
t so far
            if ls_failed or (i>abs(length)):# line search failed twice
in a row
                break                       # or we ran out of time, so we
 give up
            s = -df0; d0 = -
dot(s,s.T)                                  # try steepest
            x3 = 1/(1-d0)
            ls_failed = 1                               # this line searc
h failed
    if verbose: print("\n")
    return X, fX, i
```