Μηχανική Μάθηση - Εργασία 2

Σγουράκης Δημήτριος (ΑΜ: 1084584)

Πρόβλημα 2.1

Σε αυτό το πρόβλημα δίνεται ένα generative model G(Z) στη μορφή ενός νευρωνικού δικτύου G, με πυκνότητα πιθανότητας του Z: $\mathcal{N}(0,1)$, το οποίο παράγει μια εικόνα ενός χειρόγραφου «8» με ανάλυση 28 × 28 pixels. Πιο συγκεκριμένα, δίνονται τα βάρη (A_1, A_2) και τα offsets (B_1, B_2) του νευρωνικού δικτύου σε μορφή mat αρχείων. Το νευρωνικό δίκτυο έχει ένα hidden layer και είναι διαστάσεων: 10 × 128 × 784. Τα μαθηματικά του νευρωνικού δικτύου είναι τα εξής:

$$W_1 = A_1 \times Z + B_1, \qquad \text{όπου } Z = \text{διάνυσμα εισόδου } (10 \times 1)$$

$$Z_1 = ReLU(W_1) = \max(W_1, 0)$$

$$W_2 = A_2 \times Z_1 + B_2$$

$$X = sig(W_2), \qquad \text{όπου } X = \text{διάνυσμα εξόδου } (784 \times 1)$$

Σκοπός αυτού του προβλήματος είναι η παραγωγή 100 τυχαίων εικόνων χειρόγραφων «8» χρησιμοποιώντας το generative model G(Z). Αρχικά διαβάστηκαν οι πίνακες A1, A2, B1, B2 από το αρχείο datamat21.mat μέσω του αρχείου **read_mat.py** και υλοποιήθηκε το νευρωνικό δίκτυο που αντιπροσωπεύει το G στο αρχείο **neural_net2.py**. Έπειτα δημιουργήθηκαν 100 σετ των 10 δειγμάτων του Z, τα οποία μπήκαν ως είσοδος στο νευρωνικό δίκτυο στο αρχείο **gen_img_1.py**, και έτσι παράχθηκαν οι 100 ζητούμενες εικόνες ως διανύσματα μήκους 784. Τέλος έγινε αλλαγή διαστάσεων των παραγόμενων πινάκων σε 28 × 28 και με κατάλληλη επεξεργασία τα αποτελέσματα παρουσιάστηκαν ως μία εικόνα 10 γραμμών και 10, η οποία περιέχει όλα τα αποτελέσματα:

Εικόνα 1: Αποτελέσματα προβλήματος 2.1

Όπως φαίνεται στην εικόνα 1 δεν είναι τέλεια όλα τα αποτελέσματα, καθώς υπάρχει τυχαιότητα στα δέκα αρχικά δείγματα της Z κατανομής.

Πρόβλημα 2.2

Σε αυτό το πρόβλημα ζητείται η ανάκτηση τεσσάρων εικόνων χειρόγραφων «8», στις οποίες έχει προστεθεί θόρυβος με μέση τιμή 0 και άγνωστη διασπορά. Επίσης είναι διαθέσιμα μόνο N από τα 784 pixels. Για να υλοποιηθεί η διαδικασία της ανάκτησης πρέπει να εφαρμοστούν τα παρακάτω βήματα:

1. Παραγωγή 10 τυχαίων δειγμάτων της κατανομής Z, τα οποία θα χρησιμοποιηθούν ως αρχική είσοδος του νευρωνικού δικτύου.
2. Forward propagation του διανύσματος Z μέσω του νευρωνικού δικτύου, ώστε να υπολογιστεί η έξοδός του X.
3. Υπολογισμός της συνάρτησης κόστους J(Z). Εδώ η συνάρτηση κόστους που χρησιμοποιείται είναι η:

   $$J(Z) = N \times \log(\|T \times X - X_n\|^2) + \|Z\|^2$$

   Η πρώτη νόρμα υπολογίζει το μέτρο της διαφοράς των πρώτων N στοιχείων του διανύσματος X με το γνωστό διάνυσμα X_n (Είναι το διάνυσμα της αλλοιωμένης εικόνας μήκους N). Για να επιτευχθεί αυτό, το διάνυσμα X πολλαπλασιάζεται με τον πίνακα T, ο οποίος είναι διαστάσεων N × 784 με όλα τα στοιχεία στην κύρια διαγώνιο να είναι ίσα με τη μονάδα και τα υπόλοιπα στοιχεία να είναι μηδενικά:

Πίνακας T για N = 350

Η δεύτερη νόρμα υπάρχει ώστε να προστεθεί στο κόστος το τετράγωνο του μέτρου του στοχαστικού διανύσματος εισόδου Z. Η συνάρτηση κόστους πρέπει να υπολογιστεί, ώστε να ελεγχθεί εάν συγκλίνουν οι τιμές της.

4. Backward propagation για τον υπολογισμό του της μερικής παραγώγου του Z ως προς τη συνάρτηση J(Z):

$$u_2 = 2T^T \times \frac{(T \times X - X_n)}{\|T \times X - X_n\|^2}$$

$$v_2 = u_2 \times sig'(W_2) = -sig(W_2) \times \left(1 - sig(W_2)\right)$$

$$u_1 = {A_2}^T \times v_2$$

$$v_1 = u_1 \times ReLU'(W_1) = \begin{cases} 0, & W_1 \leq 0 \\ 1, & W_1 > 0 \end{cases}$$

$$u_0 = {A_1}^T \times v_1$$

$$\boldsymbol{\nabla_Z J(Z) = \frac{\partial J}{\partial Z} = N \times u_0 + 2Z}$$

5. Κανονικοποίηση παραγώγων για την παραγωγή πιο σταθερών λύσεων. Σε αυτή την περίπτωση χρησιμοποιήθηκε η κανονικοποίηση ADAM, οπότε η μερική εκτίμηση ισχύος της παραγώγου είναι:

$$P_Z(t) = (1 - \lambda)P_Z(t-1) + \lambda \left(\nabla_{Z_{t-1}} J(Z_t)\right)^{.2}, \text{όπου } \lambda \ll 1$$

Και η αρχική τιμή της είναι:

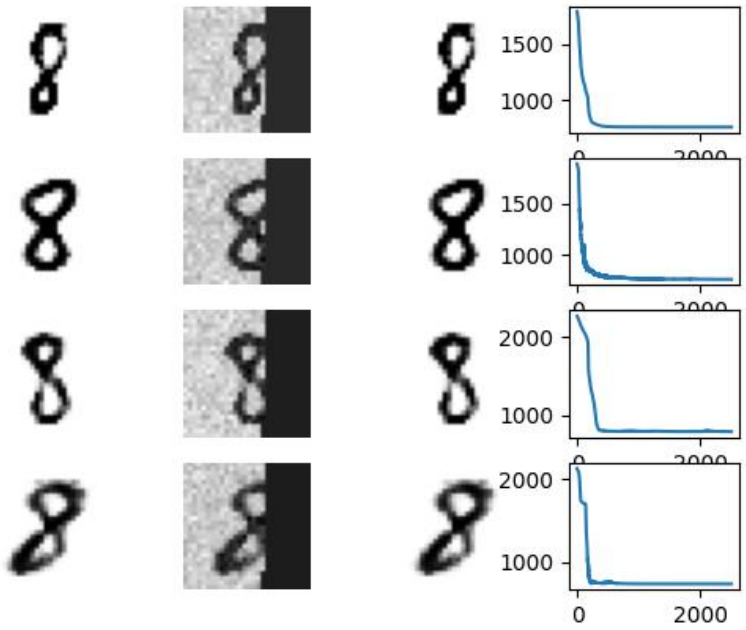
$$P_Z(1) = \left(\nabla_{Z_0} J(Z_1)\right)^{.2}$$

6. Υπολογισμός της καινούργιας εισόδου Z μέσω του παρακάτω τύπου:

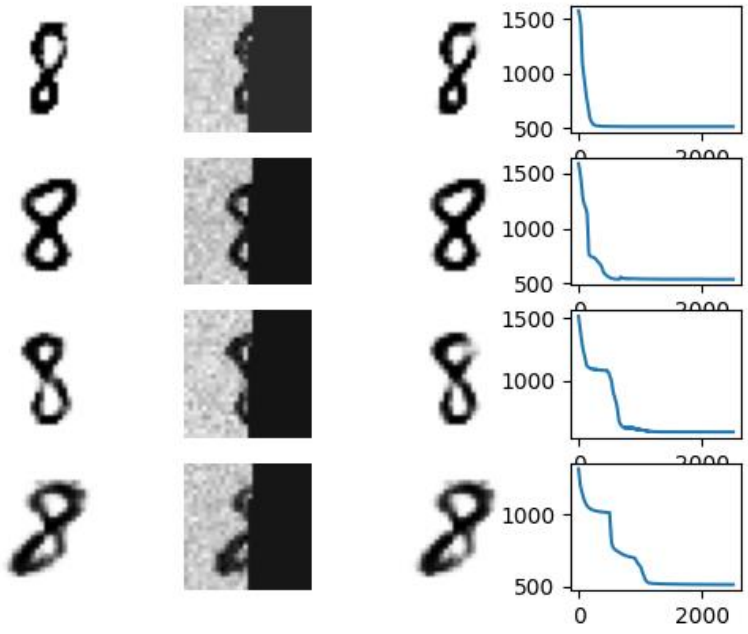
$$Z_t = Z_{t-1} - {\mu \nabla_{Z_{t-1}} J(Z_t)} \Big/ {.\sqrt{c + P_Z(t)}}, \qquad \text{όπου } \mu = \rho\upsilon\theta\mu\acute{o}\varsigma\ \mu\alpha\theta\acute{\eta}\sigma\varepsilon\omega\varsigma$$

7. Επανάληψη των βημάτων 2 έως 6 μέχρι το κόστος να συγκλίνει. Τα βήματα 4 έως 6 αποτελούν την υλοποίηση του αλγορίθμου Stochastic Gradient Descent.
8. Το διάνυσμα X μήκους 784 που παράγεται επεξεργάζεται κατάλληλα, ώστε να παρουσιαστεί ως εικόνα ανάλυσης 28 × 28.

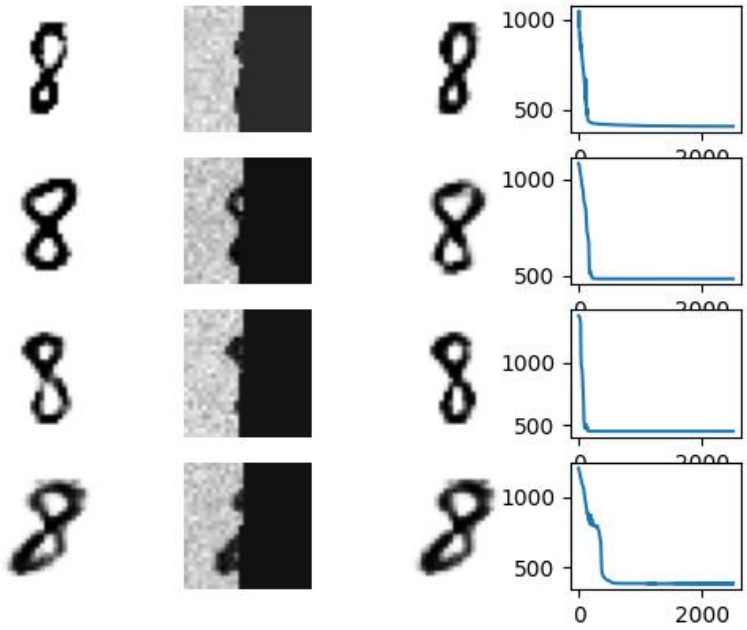
Το βήμα 8 υλοποιείται στο αρχείο **gen_img_2.py** και τα βήματα 1 έως 7 υλοποιούνται στο αρχείο **neural_net2.py**. Τα παραπάνω βήματα επαναλαμβάνονται 4 φορές (μία φορά για κάθε εικόνα).



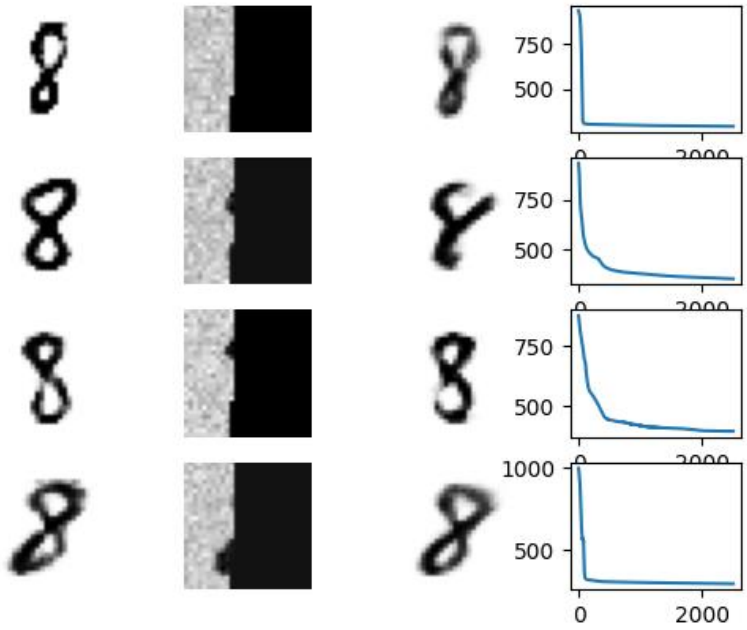
Εικόνα 2: Αρχική εικόνα, αλλοιωμένη εικόνα, παραγόμενη εικόνα, σύγκληση συνάρτησης κόστους (N = 500)



Εικόνα 3: Αρχική εικόνα, αλλοιωμένη εικόνα, παραγόμενη εικόνα, σύγκληση συνάρτησης κόστους (N = 400)



Εικόνα 4: Αρχική εικόνα, αλλοιωμένη εικόνα, παραγόμενη εικόνα, σύγκληση συνάρτησης κόστους (N = 350)



Εικόνα 5: Αρχική εικόνα, αλλοιωμένη εικόνα, παραγόμενη εικόνα, σύγκληση συνάρτησης κόστους (N = 300)

Όπως φαίνεται από τις εικόνες 2 έως 5, ενώ για ψηλότερες τιμές του N (500, 400, 350), τα αποτελέσματα είναι πολύ καλά, για χαμηλότερες τιμές (300) τα αποτελέσματα χαλάνε, καθώς δεν υπάρχει επαρκής πληροφορία στην αλλοιωμένη εικόνα για την ανάκτηση της αρχικής εικόνας.

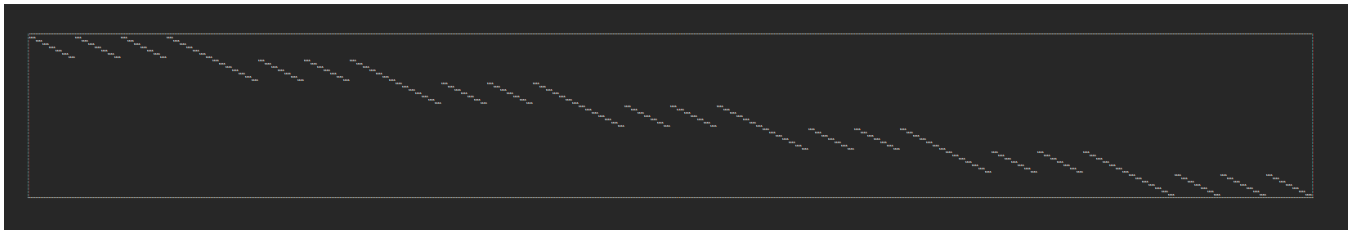
Πρόβλημα 2.3

Σε αυτό το πρόβλημα ζητείται η ανάκτηση των ίδιων τεσσάρων εικόνων με το προηγούμενο πρόβλημα, στις οποίες αυτή τη φορά έχει προστεθεί θόρυβος με μέση τιμή 0 και άγνωστη διασπορά και η ανάλυση έχει χαμηλωθεί στα 7×7 pixels.

Το συγκεκριμένο πρόβλημα έχει παρόμοια λύση με το προηγούμενο. Εδώ το N είναι 49 και ο μέσος όρος από κάθε 16 pixel συνιστά ένα pixel στην αλλοιωμένη εικόνα. Οπότε εδώ ο πίνακας T θα πρέπει να είναι διαστάσεων 49×784, ώστε ο TX να είναι διαστάσεων 49×1 και να μπορεί να γίνει η πράξη $TX - X_n$. Επίσης για κάθε 4 στήλες του X θα πρέπει να βρεθεί ο μέσος όρος των 4 σειρών που αντιστοιχούν στο κάθε pixel. Άρα ο T θα έχει παντού μηδενικά στοιχεία εκτός από τα στοιχεία που αντιστοιχούν στα κατάλληλα στοιχεία του πίνακα, τα οποία θα παίρνουν τιμή 1/16 = 0.0625. Για παράδειγμα, για να υπολογιστεί το 1ο pixel του TX θα πρέπει να ισχύει για τον T:

- 1η σειρά, στοιχεία 1 × (1-4, 29-32, 57-60, 85-88) + 0×4: 0.0625
- 2η σειρά, στοιχεία 1 × (1-4, 29-32, 57-60, 85-88) + 1×4: 0.0625
- 3η σειρά, στοιχεία 1 × (1-4, 29-32, 57-60, 85-88) + 2×4: 0.0625
- 4η σειρά, στοιχεία 1 × (1-4, 29-32, 57-60, 85-88) + 3×4: 0.0625
- Τα υπόλοιπα στοιχεία των πρώτων 4 σειρών είναι 0

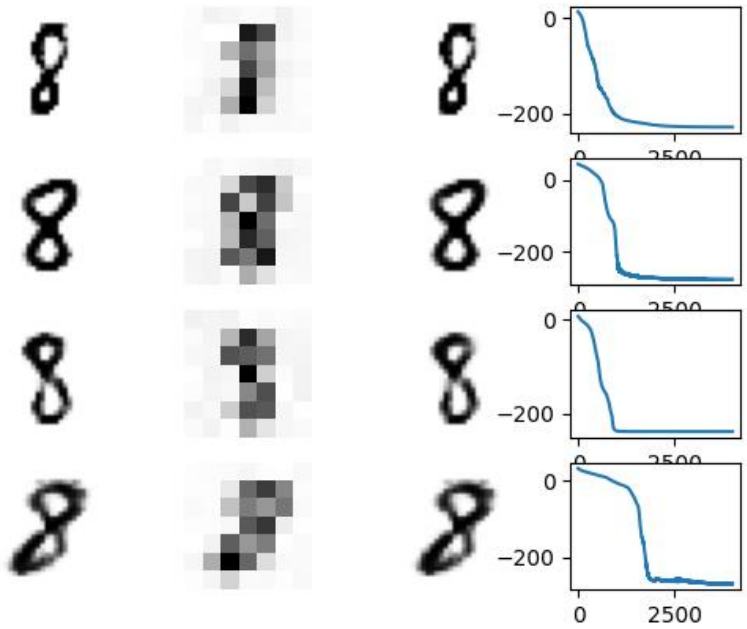
Οι παραπάνω σειρές σχηματίζουν τα πρώτα 4 pixels της πρώτης στήλης της εικόνας 7×7. Τέλος για τη δημιουργία κάθε στήλης της εικόνας 7×7 απαιτείται μετακίνηση της αντίστοιχης σειράς κατά 28×4=112 pixels. Τελικά ο πίνακας T παίρνει την παρακάτω μορφή:



Πίνακας T

Από αυτό το σημείο και πέρα ο αλγόριθμος είναι ο ίδιος με το πρόβλημα 2.2 μόνο που τώρα το N είναι πάντα 49. Ο πίνακας T υλοποιείται στο αρχείο **neural_net2.py** και η επεξεργασία του διανύσματος X που παράγεται, ώστε να προβληθούν οι εικόνες υλοποιείται στο αρχείο **gen_img_3.py**.

Παρακάτω φαίνονται τα αποτελέσματα:



Εικόνα 6: Αρχική εικόνα, αλλοιωμένη εικόνα, παραγόμενη εικόνα, σύγκληση συνάρτησης κόστους

Αρχεία υλοποίησης Python

read_mat.py

```
import numpy as np
import scipy.io
import json

mat21 = scipy.io.loadmat("data21.mat")
mat22 = scipy.io.loadmat("data22.mat")
mat23 = scipy.io.loadmat("data23.mat")

matA1 = mat21["A_1"]
matA2 = mat21["A_2"]
matB1 = mat21["B_1"]
matB2 = mat21["B_2"]

matXi_2 = mat22["X_i"]
matXn_2 = mat22["X_n"]

matXi_3 = mat23["X_i"]
matXn_3 = mat23["X_n"]

open("A1.json", 'w').close()
open("A2.json", 'w').close()
open("B1.json", 'w').close()
open("B2.json", 'w').close()

open("Xi2.json", 'w').close()
open("Xn2.json", 'w').close()

open("Xi3.json", 'w').close()
open("Xn3.json", 'w').close()

with open("A1.json", 'a') as f: json.dump(np.array(matA1).tolist(), f)
with open("A2.json", 'a') as f: json.dump(np.array(matA2).tolist(), f)
with open("B1.json", 'a') as f: json.dump(np.array(matB1).tolist(), f)
with open("B2.json", 'a') as f: json.dump(np.array(matB2).tolist(), f)

with open("Xi2.json", 'a') as f: json.dump(np.array(matXi_2).tolist(), f)
with open("Xn2.json", 'a') as f: json.dump(np.array(matXn_2).tolist(), f)

with open("Xi3.json", 'a') as f: json.dump(np.array(matXi_3).tolist(), f)
with open("Xn3.json", 'a') as f: json.dump(np.array(matXn_3).tolist(), f)
```

neural_net2.py

```
import numpy as np
import json

def ReLU(x: np.ndarray) -> np.ndarray:
    return np.maximum(0, x)

def der_ReLU(x: np.ndarray) -> int:
    return np.where(x > 0, 1, 0)

def sigmoid(x: np.ndarray) -> np.ndarray:
    return 1 / (1 + np.exp(x))

def der_sigmoid(x: np.ndarray) -> np.ndarray:
    return - sigmoid(x) * (1 - sigmoid(x))

class NeuralNetwork:
    def __init__(self, A1: np.ndarray, A2: np.ndarray, B1: np.ndarray, B2: np.ndarray,
in_size = 10,  hid_size = 128, out_size = 784):
        """A Neural Network consisting of three layers to be used as a generative
model.\n
        Since this is an already trained generative model the parameters (weights,
offsets) must be given.
        in_size, hid_size, out_size are the sizes of each of the three layers (n, m,
k).\n
        It is advised that n, m, k remain at their default values since this is a
generative model and the
        paramteters of the neural network which represent the generative function
shouldn't change."""

        # Layer sizes
        self.in_size = in_size      # n
        self.hid_size = hid_size    # m
        self.out_size = out_size    # k

        # Weights and offsets
        self.A1 = A1    # m x n
        self.A2 = A2    # k x m
        self.B1 = B1    # m x 1
        self.B2 = B2    # k x 1

        # Power estimation and constants
        self.P_dZ = 0
        self.lamda = 0.000001 # 0.000001
```

```
        self.c = 0.00000001
        self.N = 0

        self.cost = np.array([])

    def forward(self, Z: np.ndarray):
        """Propagate forward through the neural network. This generative model uses as
input a normal
        distribution with mean 0 and covariance 1.\n
        Uses a ReLU activation function for the hidden layer and a sigmoid activation
        function for the output layer.\n
        Returns the output layer of the network. Z must be an array of shape (n, 1)."""

        # W1 = A1 x Z + B1
        self.W1 = np.dot(self.A1, Z) + self.B1          # m x 1
        # Z1 = ReLU(W1)
        self.Z1 = ReLU(self.W1)                         # m x 1
        # W2 = A2 x Z1 + B2
        self.W2 = np.dot(self.A2, self.Z1) + self.B2    # k x 1
        # X = sig(W2)
        self.X = sigmoid(self.W2)                       # k x 1

    def T_matrix_create(self, problem: str):
        if problem == "2.2":
            self.T_mat = np.eye(self.N, self.out_size) # N x k
        elif problem == "2.3": # Note that this only works for 784 pixels to 49 pixels
            self.T_mat = np.array([])
            for i in range(7):
                for j in range(7):
                    self.T_mat = np.append(self.T_mat, np.zeros(112 * i))
                    for l in range(4):
                        self.T_mat = np.append(self.T_mat, np.zeros(4 * j))
                        self.T_mat = np.append(self.T_mat, [0.0625, 0.0625, 0.0625,
0.0625])
                        self.T_mat = np.append(self.T_mat, np.zeros(24 - 4 * j))
                    self.T_mat = np.append(self.T_mat, np.zeros(672 - 112 * i))
            self.T_mat = self.T_mat.reshape(49, 784)

    def calc_cost(self, Xn: np.ndarray, Z: np.ndarray):
        """Calculate the cost value of the output based on the correct value.
        Cost function is:\n
        J(Z) = N*log(||T*X - Xn||^2) + ||Z||^2"""

        self.TX_gen = np.dot(self.T_mat, self.X) # N x 1
```

```
        self.J = self.N * np.log(np.square(np.linalg.vector_norm(self.TX_gen - Xn))) +
np.square(np.linalg.vector_norm(Z))
        self.cost = np.append(self.cost, self.J)

    def backward(self, Xn: np.ndarray, Z: np.ndarray):
        """Propagate backwards through the neural network calculating the gradient
        of the input vector for the cost function."""

        # u2 = 2T^T * (TX - Xn) / ||TX - Xn||^2
        u2 = 2 * np.dot(self.T_mat.T, (self.TX_gen - Xn)) /
np.square(np.linalg.vector_norm(self.TX_gen - Xn)) # k x 1
        v2 = u2 * der_sigmoid(self.W2)      # k x 1
        u1 = np.dot(self.A2.T, v2)          # m x 1
        v1 = u1 * der_ReLU(self.W1)         # m x 1
        u0 = np.dot(self.A1.T, v1)          # n x 1
        self.dJ_dZ = self.N * u0 + 2 * Z    # n x 1

    def power_init(self):
        """Compute the power estimate of the Z gradient for the first iteration."""

        self.P_dZ = np.square(self.dJ_dZ)

    def power_get(self):
        """Compute the power estimate of the Z gradient after the first iteration."""

        self.P_dZ = (1 - self.lamda) * self.P_dZ + self.lamda * np.square(self.dJ_dZ)

    def updated_input(self, Z: np.ndarray, m: float) -> np.ndarray:
        """Return the updated input vector to minimize the cost function"""

        return Z - (m * self.dJ_dZ) / (self.c + np.sqrt(self.P_dZ))

    def prop_vec_bunch(self, Z: np.ndarray):
        """Apply the neural network as a vector tranformation G(Z) of a generative
model
        to the vectors of list Z and present the results."""

        results = np.array([])

        for array in Z:
            self.forward(array)
            results = np.append(results, np.array(self.X))
        self.out = results.reshape(len(Z), self.out_size, 1)
```

```
    def image_proc(self, Xn: np.ndarray, N: int, reps: int, m: float, problem: str) ->
np.ndarray:
        """Restore a noisy image that has either pixels missing or resolution lowered
        \nreps is the number of repetitions of the SGD algorithm.
        \nm is the SGD algorithm learning rate.
        \nproblem is the exercise problem number"""

        self.N = N
        self.T_matrix_create(problem)
        input_Z = np.random.normal(0, 1, self.in_size).reshape(self.in_size, 1)

        if problem == "2.2":
            Xn = Xn[:N]

        for i in range(reps - 1):
            self.forward(input_Z)
            self.calc_cost(Xn, input_Z)
            self.backward(Xn, input_Z)
            if i == 0:
                self.power_init()
            else:
                self.power_get()
            input_Z = self.updated_input(input_Z, m)

        self.forward(input_Z)
        self.calc_cost(Xn, input_Z)
        return self.X
```

gen_img_1.py

```
from neural_net2 import NeuralNetwork
import numpy as np
import json
from matplotlib import pyplot as plt

k = 784
pixels_hor = 28
pixels_ver = 28
z = np.array([])
image = np.array([])

# Load the generative model's neural network parameters
with open("A1.json", "r") as f: a1 = np.array(json.load(f))
with open("A2.json", "r") as f: a2 = np.array(json.load(f))
```

```
with open("B1.json", "r") as f: b1 = np.array(json.load(f))
with open("B2.json", "r") as f: b2 = np.array(json.load(f))

# Load the generative model's pdf samples
for i in range(100): z = np.append(z, np.random.normal(0, 1, 10))
z = z.reshape(100, 10, 1)

# Create the neural network and apply it to the samples
nn = NeuralNetwork(a1, a2, b1, b2)
nn.prop_vec_bunch(z)
array = nn.out.reshape(10, 10, k)

# Process the results to display images correctly (10 x 10)
for i in range(10):
    for j in range(int(k/pixels_hor)):
        for l in range(10):
            image = np.append(image, array[i, l, j*pixels_hor:(j+1)*pixels_hor])

# Display the images
image = image.reshape(10 * pixels_ver, 10 * pixels_hor).T
plt.imshow(image, cmap="gray")
plt.axis("off")
plt.show()
```

gen_img_2.py

```
from neural_net2 import NeuralNetwork
import numpy as np
import json
from matplotlib import pyplot as plt

k = 784
N = 300
m = 0.005
images = 4
pix_ver = 28
pix_hor = 28
iterations = 2500

# Load the generative model's neural network parameters
with open("A1.json", "r") as f: a1 = np.array(json.load(f))
with open("A2.json", "r") as f: a2 = np.array(json.load(f))
with open("B1.json", "r") as f: b1 = np.array(json.load(f))
with open("B2.json", "r") as f: b2 = np.array(json.load(f))
```

```
# Load the images
with open("Xi2.json", 'r') as f: X_ideal = np.array(json.load(f))
with open("Xn2.json", 'r') as f: X_noisy = np.array(json.load(f))

im_ideal = []
im_noisy = []
im_rec = []
cost_trend = []

# Create the original image, corrupted image and generated image from the neural
network
for i in range(images):
    nn = NeuralNetwork(a1, a2, b1, b2)
    im_ideal.append(X_ideal[:, i].reshape(pix_ver, pix_hor).T)
    im_noisy.append(np.concatenate((X_noisy[:N, i], np.zeros(k - N))).reshape(pix_ver,
pix_hor).T)
    im_rec.append(nn.image_proc(X_noisy[:, i].reshape(k, 1), N, iterations, m,
"2.2").reshape(pix_ver, pix_hor).T)
    cost_trend.append(nn.cost)

# Display the images and the cost convergance
figure , axis = plt.subplots(images, 4)
for i in range(images):
    axis[i, 0].imshow(im_ideal[i], cmap="gray")
    axis[i, 1].imshow(im_noisy[i], cmap="gray")
    axis[i, 2].imshow(im_rec[i], cmap="gray")
    for j in range(3): axis[i, j].axis("off")
    axis[i, 3].plot(np.linspace(0, len(cost_trend[i]), len(cost_trend[i])),
cost_trend[i])
plt.show()
```

gen_img_3.py

```
from neural_net2 import NeuralNetwork
import numpy as np
import json
from matplotlib import pyplot as plt

N = 49
m = 0.001
images = 4
pix_ver_lres = 7
pix_hor_lres = 7
```

```
pix_ver_hres = 28
pix_hor_hres = 28
iterations = 4000

# Load the generative model's neural network parameters
with open("A1.json", "r") as f: a1 = np.array(json.load(f))
with open("A2.json", "r") as f: a2 = np.array(json.load(f))
with open("B1.json", "r") as f: b1 = np.array(json.load(f))
with open("B2.json", "r") as f: b2 = np.array(json.load(f))

# Load the images
with open("Xi3.json", 'r') as f: X_ideal = np.array(json.load(f))
with open("Xn3.json", 'r') as f: X_lowres = np.array(json.load(f))

im_ideal = []
im_lowres = []
im_rec = []
cost_trend = []

# Create the original image, low res image and generated image from the neural network
for i in range(images):
    nn = NeuralNetwork(a1, a2, b1, b2)
    im_ideal.append(X_ideal[:, i].reshape(pix_ver_hres, pix_hor_hres).T)
    im_lowres.append(X_lowres[:, i].reshape(pix_ver_lres, pix_hor_lres).T)
    im_rec.append(nn.image_proc(X_lowres[:, i].reshape(N, 1), N, iterations, m,
"2.3").reshape(pix_ver_hres, pix_hor_hres).T)
    cost_trend.append(nn.cost)

# Display the images and the cost convergance
figure , axis = plt.subplots(images, 4)
for i in range(images):
    axis[i, 0].imshow(im_ideal[i], cmap="gray")
    axis[i, 1].imshow(im_lowres[i], cmap="gray")
    axis[i, 2].imshow(im_rec[i], cmap="gray")
    for j in range(3): axis[i, j].axis("off")
    axis[i, 3].plot(np.linspace(0, len(cost_trend[i]), len(cost_trend[i])),
cost_trend[i])
plt.show()
```