Data-Driven Techniques: 2nd Homework

Μαρία-Νίκη Ζωγράφου

AM: 1096060

Περιεχόμενα

Ερώτημα 1

Το μοντέλο μαθαίνει να παράγει χειρόγραφα "8" βασιζόμενο στην τυχαία είσοδο Z, χρησιμοποιώντας μη γραμμικούς μετασχηματισμούς (ReLU, sigmoid) και προ-εκπαιδευμένα βάρη (A1,A2,B1,B2) για να αναπαραστήσει τα 8.

Από την θεωρία γνωρίζουμε ότι για ένα $X_i \sim f(x)$, μπορούμε να βρούμε κατάλληλο G(Z) τέτοιο ώστε: $X = G(Z), Z \sim h(Z)$. Σε αυτήν την περίπτωση τα X_i είναι τα χειρόγραφα 8. Το ζητούμενο είναι να χρησιμοποιήσουμε έναν απλό, γνωστό χώρο Z που ακολουθεί την κανονική κατανομή h(Z) και να βρούμε μια κατάλληλη συνάρτηση G(Z) έτσι ώστε: $X = G(Z)$. Η f(x) είναι άγνωστη προφανώς.

Το G(Z) είναι μια συνάρτηση που παίρνει ως είσοδο έναν τυχαίο 10-διάστατο χώρο Z (latent space) και δημιουργεί ένα 784-διάστατο διάνυσμα X το οποίο αντιπροσωπεύει μια εικόνα 28×28 (αφού το παράξουμε θα χωρίσουμε τις τιμές του σε 28 στήλες). Το Z βρίσκεται σε ένα χαμηλής διάστασης χώρο και το G(Z) μαθαίνει να μετασχηματίζει το Z σε ένα υψηλής διάστασης χώρο, όπου ζουν οι χειρόγραφοι αριθμοί "8", εικόνες με 28x28 pixel.

Το Z είναι ένα τυχαίο `10x1` ένα τυχαίο vector που ακολουθεί γκαουσιαννή κατανομή. Το X.Την συνάρτηση G(Z) την προσομοιώνουμε με έναν νευρωνικό δίκτυο της εξής μορφής:

$$W_1 = A_1 Z + B_1$$

$$Z_1 = f_1(W_1), \qquad \text{όπου } f_1(w) = \max(W, 0) = ReLu(W, 0)$$

$$W_2 = A_2 Z_1 + B_2$$

$$X = f_2(W_2), \ \text{όπου } f_2(w) = \frac{1}{1 + e^w}$$

Παράγουμε τα εξής αποτελέσματα:

Ερώτημα 2

Ο σκοπός αυτού του ερωτήματος είναι η χρήση του Generator του Ερωτήματος 2.1 για την ανακατασκευή ("inpainting") εικόνων του αριθμού "8". Αυτό περιλαμβάνει την ανάκτηση των χαμένων στοιχείων των εικόνων και την απομάκρυνση του προστιθέμενου θορύβου.

Θα βελτιστοποιήσουμε το Generator του προηγούμενου ερωτήματος μέσω gradient descent.

Για κάθε **N** ορίζουμε μήτρα Μετασχηματισμού **T** η οποία επιλέγει μόνο τα πρώτα **N** στοιχεία της κάθε εικόνας ($\boldsymbol{X_n}$). Στον υπολογισμό του loss θα χρησιμοποιήσουμε **μόνο** τα πρώτα **N** στοιχεία του **X**, δηλαδή το κομμάτι της εικόνας που δεν χάθηκε. Μόνο αυτό το μέρος έχει νόημα να ληφθεί υπόψιν στο training.

Το νευρωνικό μας δίκτυο όπως και προηγουμένως έχει την εξής μορφή:

$W_1 = A_1 Z + B_1$

$Z_1 = f_1(W_1), \qquad \text{όπου } f_1(w) = \max(W, 0) = ReLu(W, 0)$

$W_2 = A_2 Z_1 + B_2$

$X = f_2(W_2), \ \text{όπου } f_2(w) = \dfrac{1}{1 + e^w}$

Ως συνάρτηση loss χρησιμοποιούμε την: $J(z) = Nlog\left(\left||T * X - X_n|\right|^2\right) + \left||z|\right|^2$ και έχουμε ακόμα ότι: $\nabla_z J(z) = N * \nabla_z \left(\log\left(\left||T * X - X_n|\right|^2\right)\right) + \partial z$

Επομένως για να εκπαιδεύσουμε το δίκτυο θα ακολουθήσουμε την εξής διαδικασία:

$$\varphi(x) = \log\left(\left||T * X - X_n|\right|^2\right)$$

$$u_2 = \nabla_x \varphi(x) = \frac{2 * T^T * (T * X - X_n)}{\left||T * X - X_n|\right|^2}$$

$$\nu_2 = u_2 \cdot f_2'(w_2), \text{όπου } f_2' = -\frac{e^w}{(1 + e^w)^2}$$

$$u_1 = A_2^T * \nu_2$$

$$\nu_1 = u_1 \cdot f_1'(w_1), \text{όπου } f_1'(w_1) = \{1 \ \gamma\iota\alpha \ w_1 > 0, 0 \ \gamma\iota\alpha \ w_1 < 0\}$$

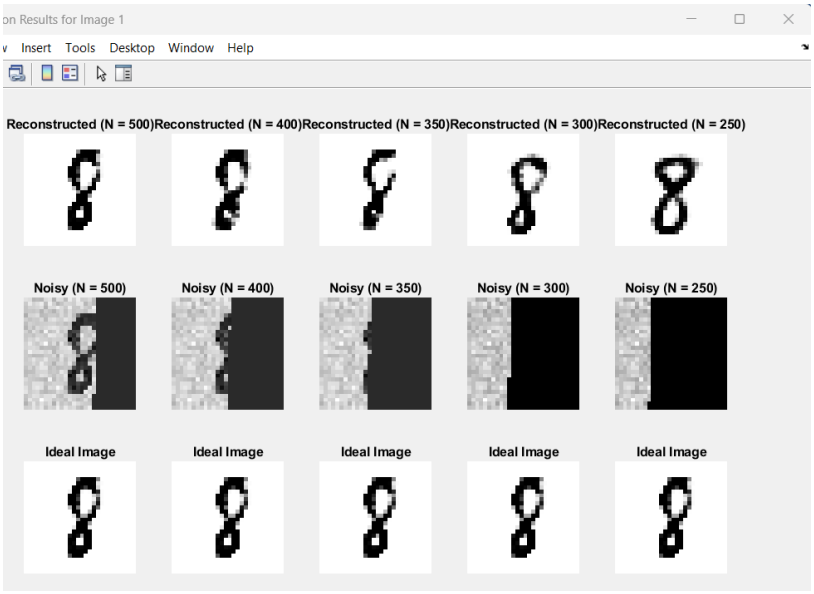
$$u_0 = A_1^T \nu_1 = \nabla_z \varphi(x)$$

Επομένως $\nabla_z \left\{Nlog\left(\left||T * X - X_n|\right|^2\right) + \left||z|\right|^2\right\} = N * u_0 + 2 * \lambda * z$

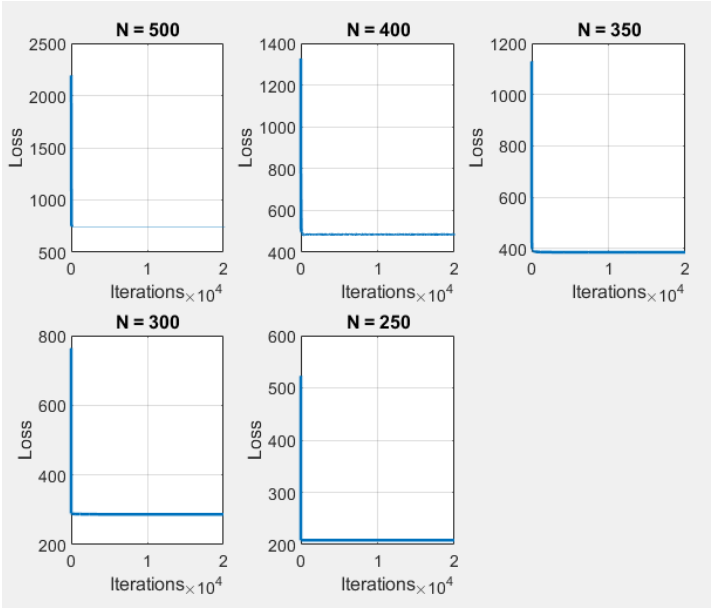
Έτσι εκπαιδεύεται το δίκτυο και αφού ολοκληρωθούν 20.000 επαναλήψεις του gradient descent παίρνουμε την τελική έξοδο του νευρωνικού δικτύου και την μετατρέπουμε σε εικόνα. Αυτή την ονομάζουμε ανακατασκευασμένη εικόνα και την συγκρίνουμε με την ιδανική και με την αλλοιωμένη εικόνα για διαφορετικά N. **Όταν το N πέφτει κάτω από 300 παρατηρούμε ότι η ανακατασκευή απομακρύνεται από το ιδανικό.**

Τέλος θα σχεδιάσουμε και το loss history για να ελέγξουμε την σύγκλιση.

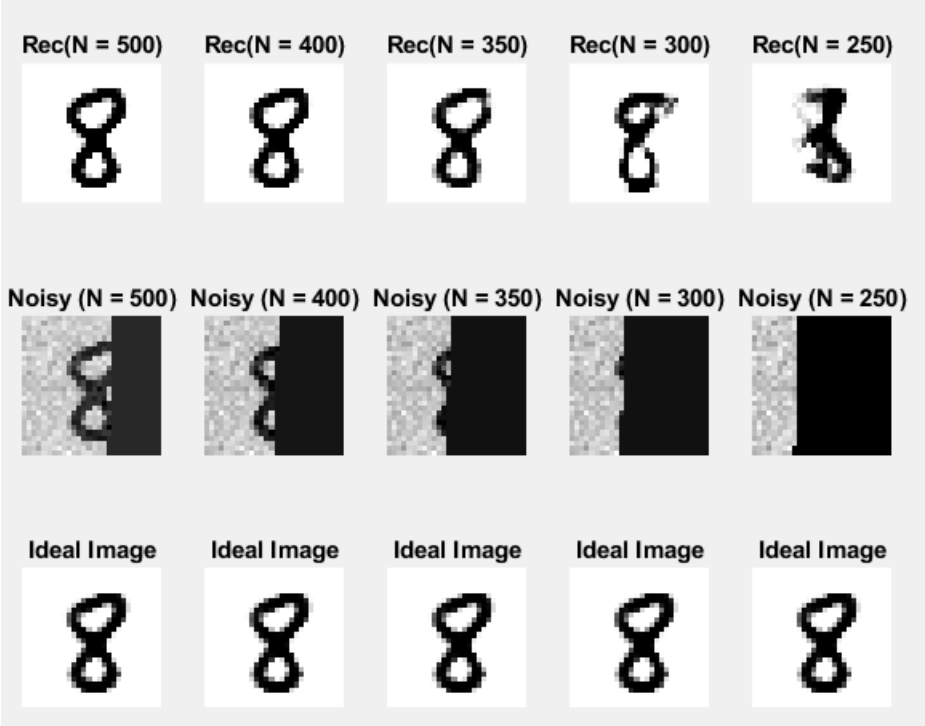
Εικόνα 1:



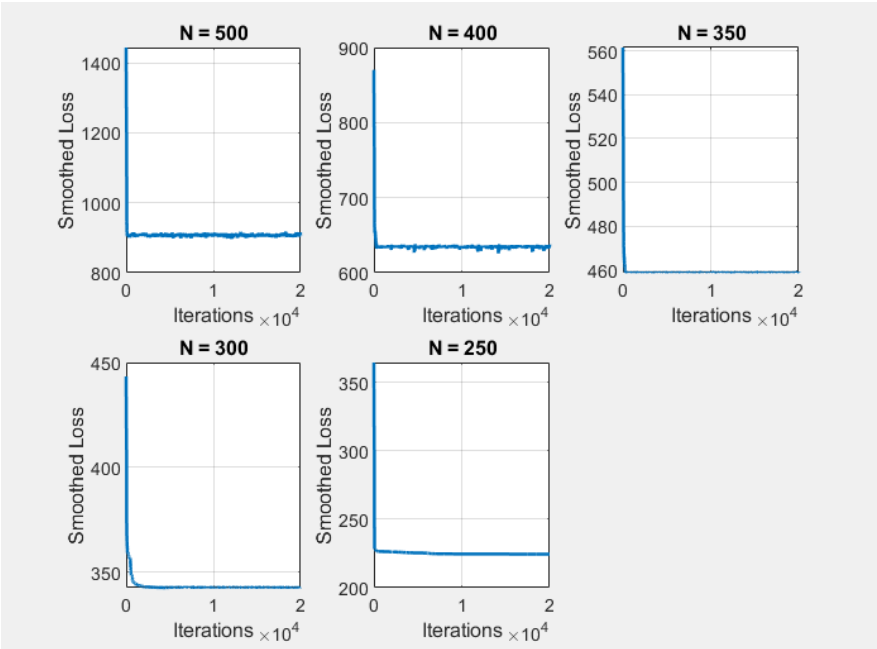
- Για N=500,400 παρατηρούμε σχεδόν τέλεια ανακατασκευή και για N=350 ικανοποιητική.
- Για N=300 η ποιότητα αρχίζει να μειώνεται.
- Για N=250 η ανακατασκευή έχει απομακρυνθεί αρκετά από την ιδανική εικόνα, όπως είναι λογικό.
- Παρατηρούμε επίσης ότι για όλα τα N το loss έχει συγκλίνει.



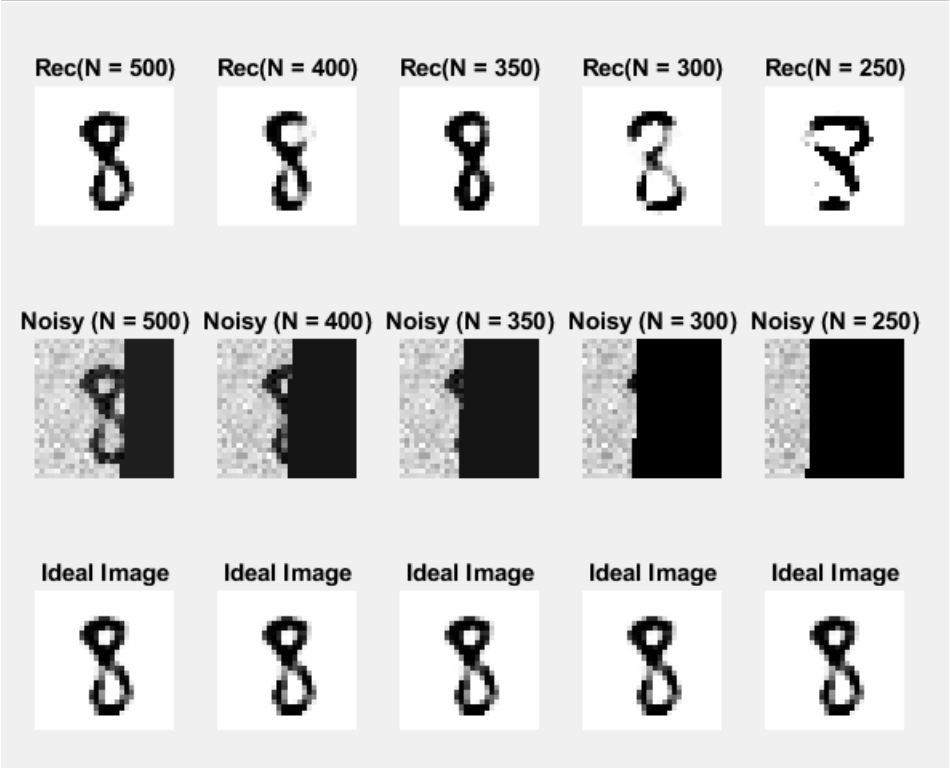
Εικόνα 2:



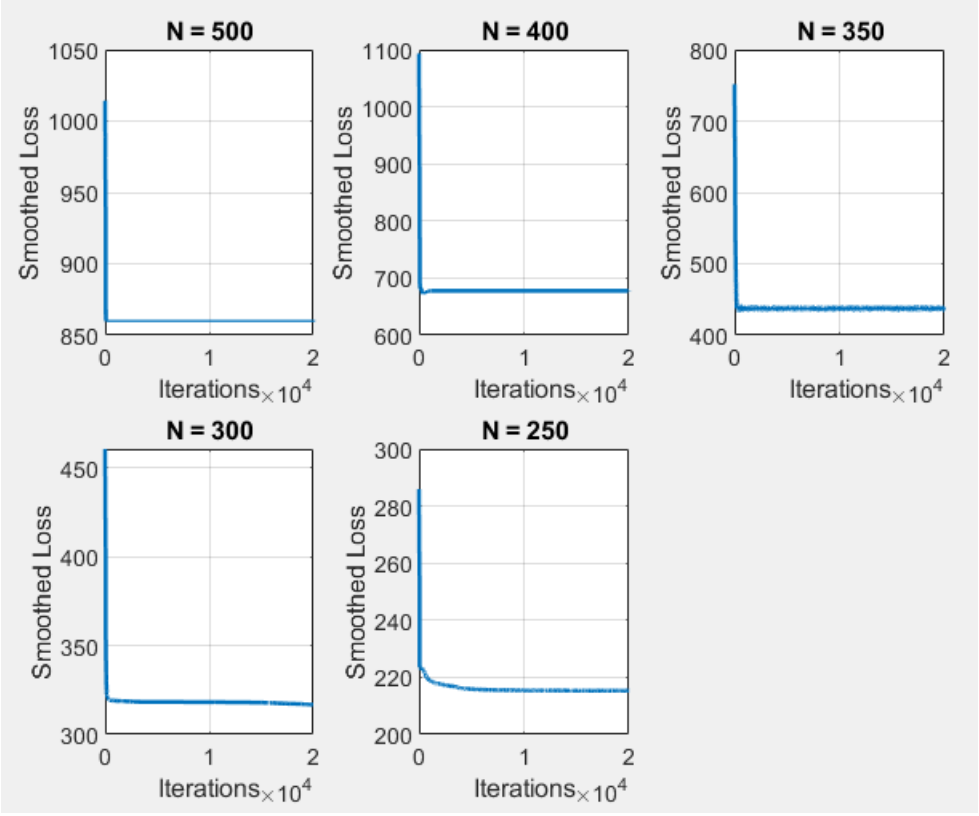
- Για N=500,400,350 παρατηρούμε σχεδόν τέλεια ανακατασκευή και για N=350,300 μειωμένη ποιότητα.
- Για N=300 η ποιότητα αρχίζει να μειώνεται.
- Στο N=250 η ανακατασκευή αποτυγχάνει, παρουσιάζοντας ένα 8 πολύ διαφορετικό από το αρχικό.
- Όσον αφορά την smoothed loss, παρατηρούμε ότι συγκλίνει για όλα τα N.



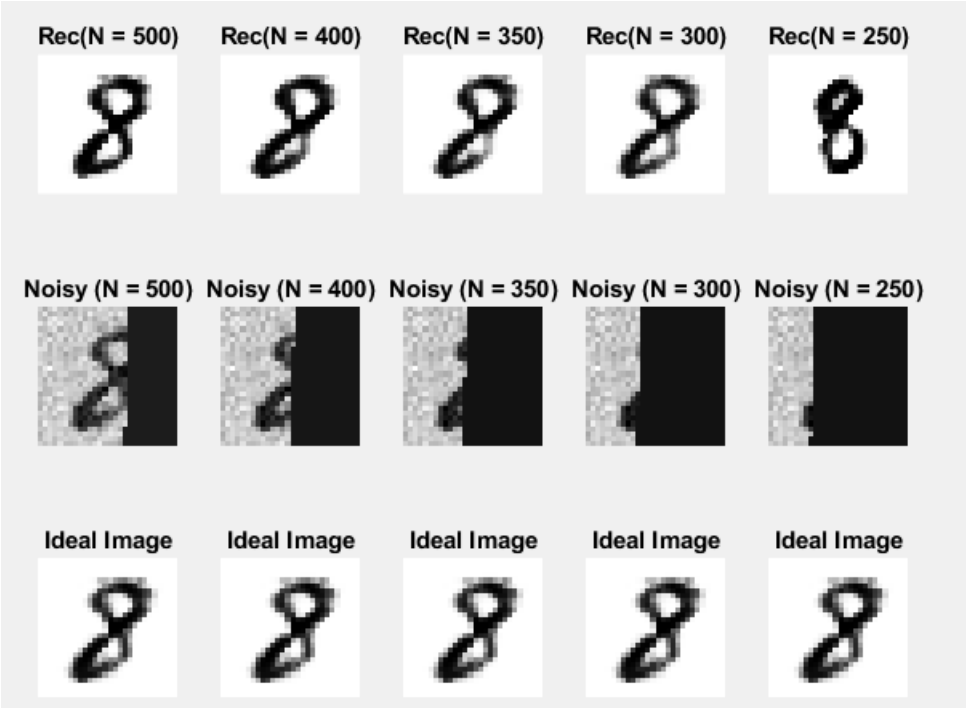
Εικόνα 3:



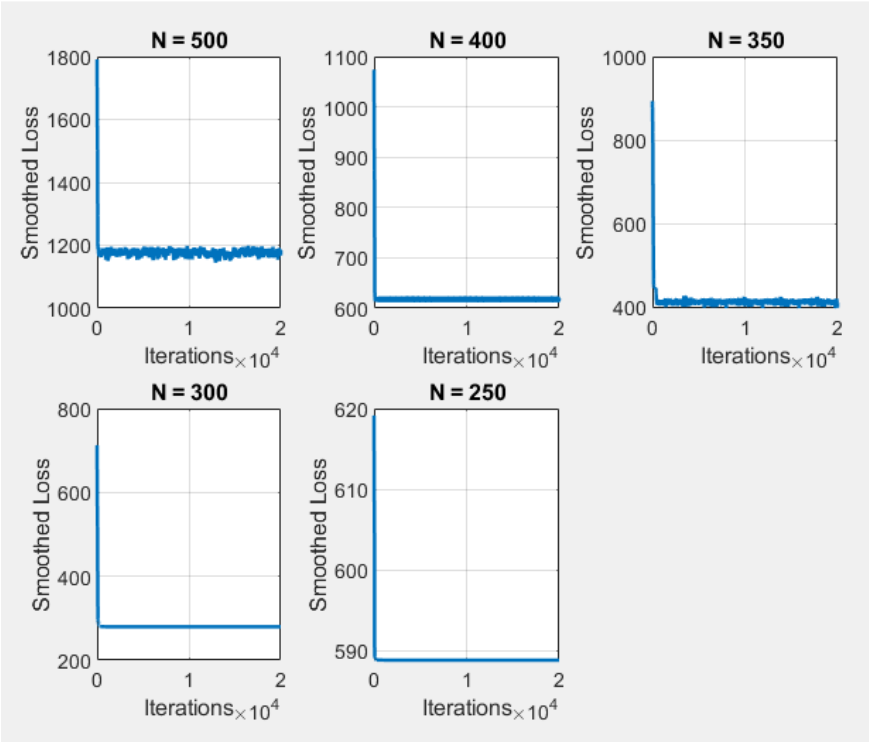
- Για N=500 έως και 350 παρατηρούμε σχεδόν τέλεια ανακατασκευή.
- Για N=300,250 η εικόνα απομακρύνεται από την ιδανική.
- Όλα κάνουν converge.



Εικόνα 4:



- Για N=500 έως και 350 παρατηρούμε σχεδόν τέλεια ανακατασκευή.
- Για N=300 η ποιότητα είναι ικανοποιητική.
- Στο N=250 έχει χαθεί η αρχική πληροφορία.
- Όλα τα losses έχουν συγκλίνει.



Σχολιασμός:

Παρατηρούμε ότι η Ανακατασκευή Λειτουργεί Καλά (συνήθως) με N έως και 300 αλλά αποτυγχάνει με N=250. Όταν N=300 χρησιμοποιεί 300 από τα 784 στοιχεία του Xn (≈40%). Αυτά τα 300 στοιχεία περιέχουν αρκετή πληροφορία. Όταν N μειώνεται στα 250 (≈30%) το μοντέλο λαμβάνει ελάχιστη πληροφορία. Αυτό αφήνει το μοντέλο με λιγότερους περιορισμούς, με αποτέλεσμα η ανακατασκευή να απομακρύνεται από τη σωστή μορφή της εικόνας "8".

Χρειάζεται επίσης να λάβουμε υπόψιν μας τον θόρυβο. Με N= 300 τα δεδομένα είναι αρκετά ώστε να αντιμετωπίσουν τον θόρυβο, ενώ με N=250 ο θόρυβος υπερτερεί, οδηγώντας σε λανθασμένες ενημερώσεις του Z.

Τέλος η συνάρτηση κόστους: : $J(z) = Nlog\left(||T * X - X_n||^2\right) + ||z||^2$ εξαρτάται άμεσα από το N. Όταν το N μειώνεται το gradient του πρώτου όρου (reconstruction loss) μειώνεται.

Ερώτημα 3

Στόχος του ερωτήματος είναι η ανακατασκευή εικόνων του αριθμού "8" από δεδομένα χαμηλής ανάλυσης με θόρυβο, χρησιμοποιώντας το generator του Ερωτήματος 2.1. Οι εικόνες Xn που παρέχονται έχουν ανάλυση 7 × 7 (δηλαδή, 16 φορές μικρότερη από την αρχική 28 × 28) και πρέπει να επαναφερθούν στην αρχική τους ανάλυση.

Νευρωνικό Δίκτυο:

Όπως προηγουμένως χρησιμοποιούμε ένα νευρωνικό δίκτυο με τα εξής επίπεδα:

$W_1 = A_1 Z + B_1$

$Z_1 = f_1(W_1), \qquad \text{όπου } f_1(w) = \max(W, 0) = ReLu(W, 0)$

$W_2 = A_2 Z_1 + B_2$

$X = f_2(W_2), \ \text{όπου } f_2(w) = \frac{1}{1 + e^w}$

Gradient Descent

Loss: $J(z) = log\left(||T * X - X_n||^2\right) + ||z||^2$

Όπως και πριν

$u_2 = \nabla_x \varphi(x) = \frac{2 * T^T * (T * X - X_n)}{||T * X - X_n||^2}, \ \nu_2 = u_2 \cdot f_2'(w_2), \text{όπου } f_2' = -\frac{e^w}{(1+e^w)^2},$

Και $u_1 = A_2^T * \nu_2, \ \ \nu_1 = u_1 \cdot f_1'(w_1), \text{όπου } f_1'(w_1) = \{1 \ για \ w_1 > 0, 0 \ για \ w_1 < 0\}$

Καθώς και $u_0 = A_1^T \nu_1 = \nabla_z \varphi(x)$ με διαφορά ότι δεν χρειάζεται πλέον το N.

Επομένως $\nabla_z \left\{log\left(||T * X - X_n||^2\right) + ||z||^2\right\} = u_0 + 2 * \lambda * z$

Το T στην συγκεκριμένη περίπτωση είναι διαφορετικό της προηγούμενης καθώς δεν χρειάζεται απλώς να κάνουμε mask τα Pixel από N+1 και πάνω, αλλά θα χρειαστεί να μετατρέψουμε την εικόνα **X** 28x28 σε εικόνα μειωμένης ανάλυσης 7x7.

Δημιουργία εικόνων μειωμένης ανάλυσης:

Ο $\boldsymbol{T}$ είναι ένας γραμμικός μετασχηματισμός που αναπαριστά τη διαδικασία μείωσης της ανάλυσης. Μεταφέρει την ανάλυση 28×28 (784 στοιχεία) στην ανάλυση 7×7 (49 στοιχεία) με το να αντικαθιστά 4x4 pixels της αρχικής εικόνας με 1 pixel με τιμή τον μέσο όρο των 4x4 pixels που θα αντικατασταθούν. Ο **T** έχει διαστάσεις 49x784 και ο **X** η εικόνα είναι ένα διάνυσμα **784x1**, οπότε το τελικό αποτέλεσμα είναι 49x1 που

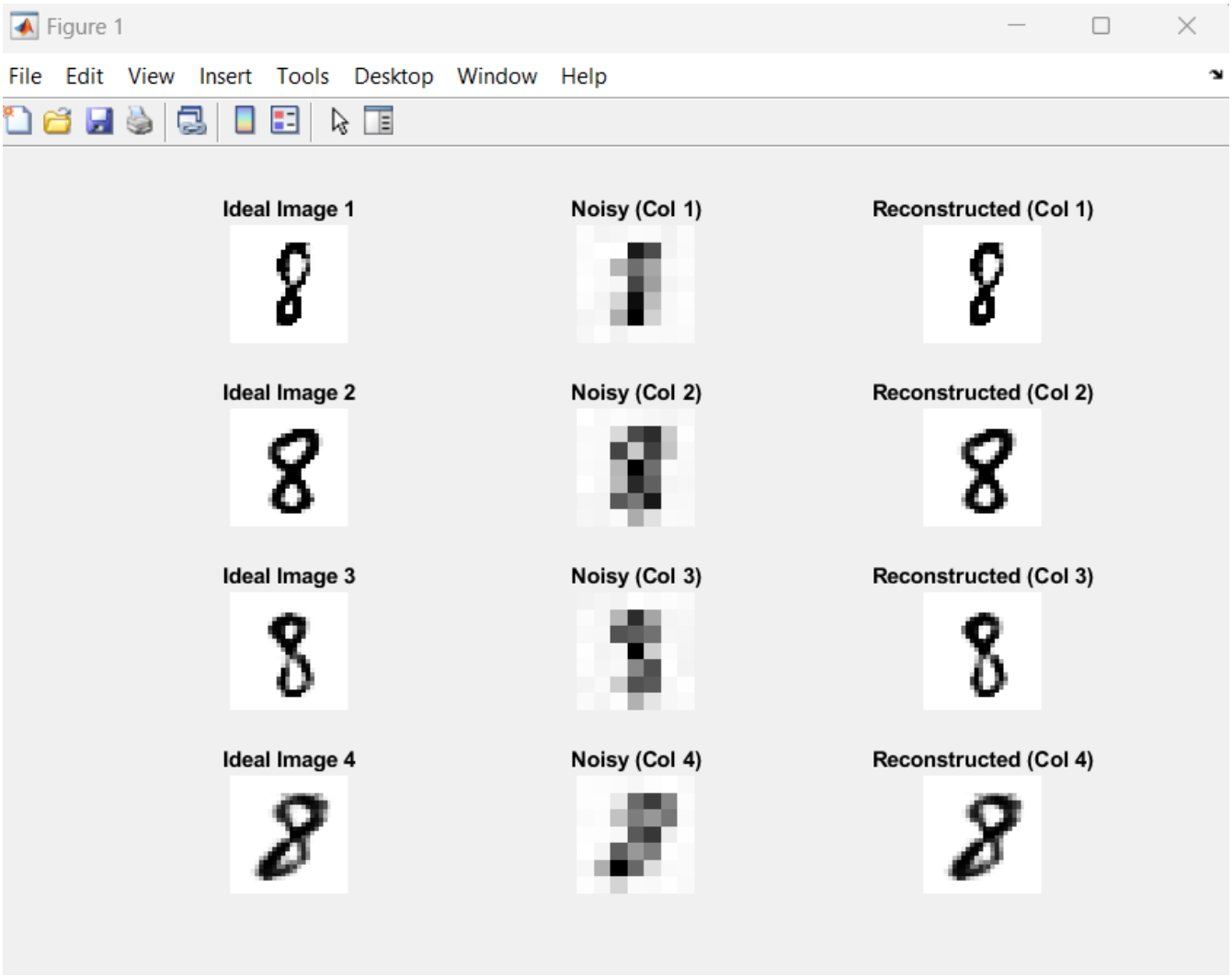
αντιστοιχεί σε μία εικόνα 7x7. Ο **T** θα πολλαπλασιαστεί με το **X** παράγοντας ένα διάνυσμα που **κάθε στοιχείο του είναι ο μέσος όρος 16 pixel του X. Επομένως κάθε σειρά του T θα έχει 16 στοιχεία με τιμή 1/16 και τα υπόλοιπα 0.** Για την πρώτη σειρά τα πρώτα 4 στοιχεία και ύστερα τα 28+(1 έως 4), 28*2+(1 έως 4) και 28*3+(1 έως 4) θα έχουν τιμή 1/16 (ουσιαστικά αυτά αντιστοιχούν στο πάνω αριστερά 4x4 κουτάκι της εικόνας X) και τα υπόλοιπα θα είναι 0. Αυτό θα γίνει η τιμή του πρώτου στοιχείου του **T*X**. Με αυτή την λογική βάζουμε τις κατάλληλες τιμές σε όλες τις σειρές του **T**.

Για κάθε σειρά του T **i**, τα στοιχεία i έως 4, 28 + (i έως 4), 28*2 +(i έως 4) και 28*3+(i έως 4) θα έχουν τιμή 1/16 και τα υπόλοιπα θα έχουν τιμή 0.

Σημείωση: Οι βέλτιστες τιμές για το learning rate, max_iterations, lambda έχουν βρεθεί εμπειρικά, έτσι ώστε να επιτευχθεί το καλύτερο δυνατό αποτέλεσμα και να εξασφαλιστεί σύγκλιση. Έχει επιλεχθεί max_iterations = 150000, learning_rate = 0.0003, και lambda = 0.0007; % Regularization parameter.

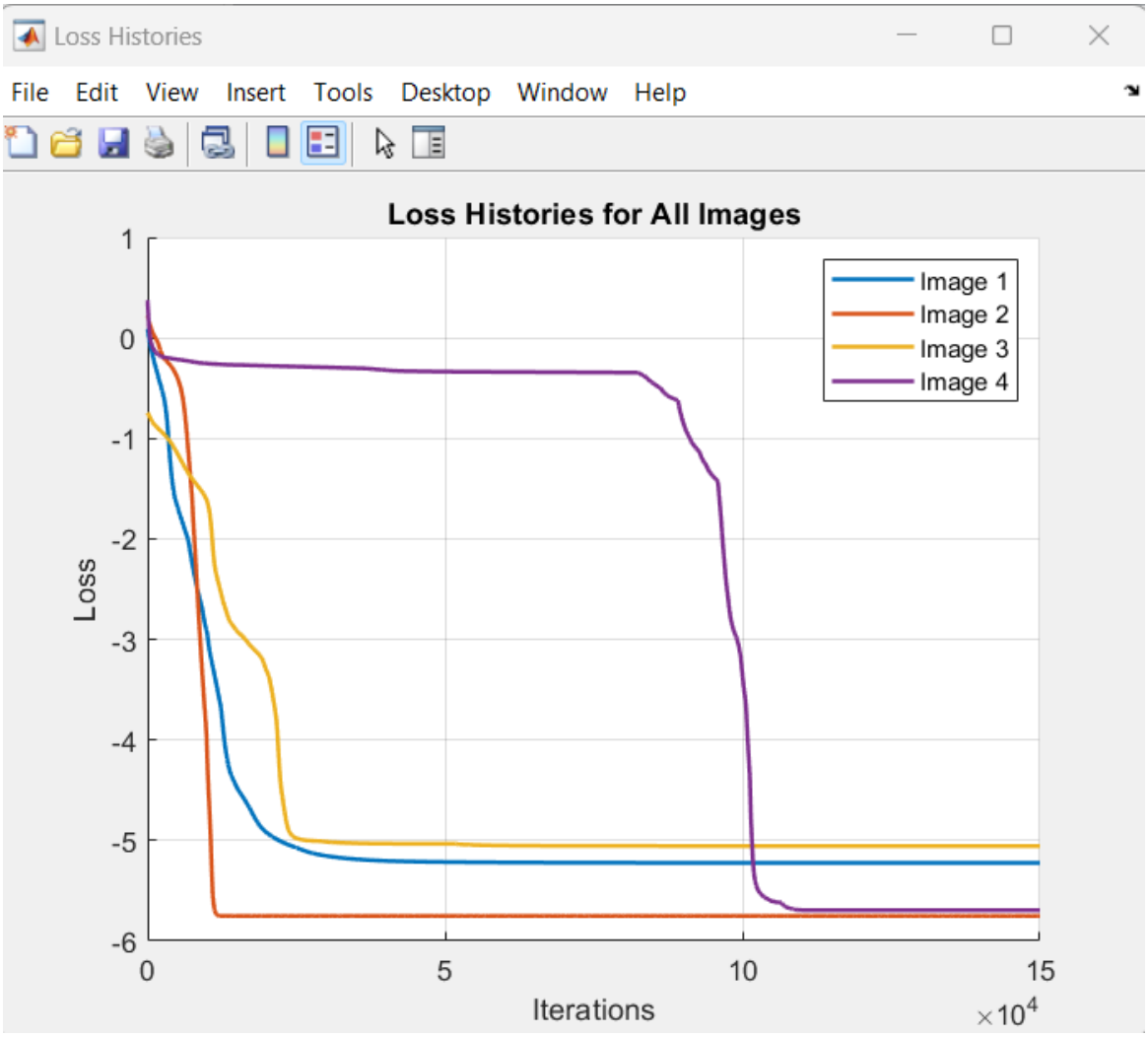
Αποτελέσματα:

Παρατηρούμε ότι ανακτάτε η αρχική εικόνα με πολύ μεγάλη ακρίβεια!



Σύγκλιση:

Φυλάμε το Loss και κάνουμε Plot όλα τα losses στο ίδιο graph. Παρατηρούμε ότι όλες οι εικόνες συγκλίνουν.



Σχολιασμός:

Χάρη στον πίνακα T ο generator μαθαίνει να αντιστρέφει τον μετασχηματισμό υποβάθμισης ανάλυσης που αναπαρίσταται από τον T, ανακατασκευάζοντας εικόνες υψηλής ανάλυσης από δεδομένα χαμηλής ανάλυσης.

Ρύθμιση λάμδα: Η ύπαρξη προσθετικού θορύβου στο Xn σημαίνει ότι μια μεγάλη τιμή του λ θα μπορούσε να οδηγήσει σε υπερπροσαρμογή στον θόρυβο.

Κώδικας

Ερώτημα 1

```
% Load the data21.mat file
clc;
close all;
clear all;
load('data21.mat');

A1 = A_1; % 128 x 10
A2 = A_2; % 784 x 128
B1 = B_1; % 128 x 1
B2 = B_2; % 784 x 1

% Number of samples to generate
n_samples = 100;

% Initialize a container for generated images
images = zeros(28, 28, n_samples);

% Generate 100 samples of Z and corresponding images
for i = 1:n_samples
    % Generate a random Z (10x1 vector, i.i.d. Gaussian)
    Z = randn(10, 1);

    % Compute W1 = A1 * Z + B1
    W1 = A1 * Z + B1;

    % Apply ReLU: Z1 = max(W1, 0)
    Z1 = max(W1, 0);

    % Compute W2 = A2 * Z1 + B2
    W2 = A2 * Z1 + B2;

    % Apply sigmoid activation: X = 1 / (1 + exp(-W2))
    X = 1 ./ (1 + exp(W2));

    % Reshape X into a 28x28 image
    images(:, :, i) = reshape(X, 28, 28);
end

% Create a 10x10 grid of images
grid_size = 10;
image_grid = zeros(28 * grid_size, 28 * grid_size);

for row = 1:grid_size
    for col = 1:grid_size
        idx = (row - 1) * grid_size + col;
        image_grid((row - 1) * 28 + 1:row * 28, (col - 1) * 28 + 1:col *
28) = images(:, :, idx);
    end
end

% Display the grid of images
figure;
imshow(image_grid, []);
title('Generated Handwritten 8s');
```

Ερώτημα 2

```
clc; close all; clear all;
% Load data
load('data21.mat'); % Generative model (A1, A2, B1, B2)
load('data22.mat'); % Matrices Xi and Xn
A1 = A_1; % 128 x 10
A2 = A_2; % 784 x 128
B1 = B_1; % 128 x 1
B2 = B_2; % 784 x 1
Xi = X_i; % Ideal images (784 x 4)
Xn = X_n; % Noisy images (784 x 4)

% Parameters
N_values = [500, 400, 350, 300, 250]; % Different values of N
max_iterations = 20000; % Maximum number of iterations for gradient descent
learning_rate = 0.001; % Learning rate for gradient descent
lambda = 0.001; % Regularization parameter

% Initialize storage for loss histories
loss_histories = zeros(max_iterations, length(N_values), size(Xn, 2));

% Process each column of Xn
for col = 1:size(Xn, 2)
    fprintf('Processing image %d\n', col);

    % Extract the observed noisy column of Xn
    Xn_col = Xn(:, col);

    % Create a figure to display images
    figure('Name', ['Reconstruction Results for Image ', num2str(col)],
'NumberTitle', 'off');
    % Loop over different values of N
    for idx = 1:length(N_values)
        N = N_values(idx);
        T = [eye(N), zeros(N, 784 - N)]; % Downsampling matrix
        fprintf('  Using N = %d\n', N);
        % Extract the noisy partial vector (first N values)
        Xn_partial = Xn_col(1:N);

        % Initialize Z (latent variable)
        Z = randn(10, 1);
        loss_history = zeros(max_iterations, 1);

        % Gradient Descent to optimize Z
        for iter = 1:max_iterations
            % Forward pass through the generative model
            W1 = A1 * Z + B1;
            Z1 = max(W1, 0); % ReLU activation
            W2 = A2 * Z1 + B2;
            X = 1 ./ (1 + exp(W2)); % Sigmoid activation

            % Compute loss
            residual = T * X - Xn_partial; % Residual vector
            norm_squared = norm(residual)^2; % Squared norm of the residual
            loss = N * log(norm_squared) + lambda * norm(Z)^2; % Loss func
            loss_history(iter) = loss; % Store the loss

            % Backpropagation
            gradient_loss = (2  / norm_squared) * (T' * residual); % ∇_x ϕ
```

```
            f2_derivative = -(exp(W2)) ./ ((1 + exp(W2)).^2); %f'2
            v2 = gradient_loss .* f2_derivative; % Backprop through sigmoid
            u1 = A2' * v2; % Backprop through A2
            v1 = u1 .* (W1 > 0); % Backprop through ReLU
            grad = N *  A1' * v1 + 2 * lambda * Z; % Total gradient

            % Update Z
            Z = Z - learning_rate * grad;
        end

        % Save the loss history for this (N, col) combination
        loss_histories(:, idx, col) = loss_history;

        % Reconstruct high-resolution image
        W1 = A1 * Z + B1;
        Z1 = max(W1, 0); % ReLU activation
        W2 = A2 * Z1 + B2;
        X_reconstructed = 1 ./ (1 + exp(W2)); % Reconstructed image
        X_reconstructed_image = reshape(X_reconstructed, [28, 28]);

        % Plot the reconstructed, noisy, and ideal images
        subplot(3, length(N_values), idx); % Row 1: Reconstructed Images
        imshow(X_reconstructed_image, []);
        title(['Rec(N = ', num2str(N), ')']);

        subplot(3, length(N_values), idx + length(N_values)); % Row 2:
Noisy Images
        partial_image = Xn_col;
        partial_image(N+1:end) = 0; % Mask the missing part
        imshow(reshape(partial_image, [28, 28]), []);
        title(['Noisy (N = ', num2str(N), ')']);

        subplot(3, length(N_values), idx + 2*length(N_values)); % Row 3:
Ideal Images
        imshow(reshape(Xi(:, col), [28, 28]), []);
        title('Ideal Image');
    end
end
% Plot smoothed loss histories for visualization
smoothing_window = 100; % Adjust the window size for smoothing
for col = 1:size(Xn, 2)
    figure('Name', ['Smoothed Loss Histories for Image ', num2str(col)],
'NumberTitle', 'off');
    for idx = 1:length(N_values)
        % Smooth the loss history
        smoothed_loss = smooth_loss(squeeze(loss_histories(:, idx, col)),
smoothing_window);

        % Plot the smoothed loss
        subplot(ceil(length(N_values) / 3), 3, idx); % Adjust grid
dynamically
        plot(1:max_iterations, smoothed_loss, 'LineWidth', 1.5);
        xlabel('Iterations');
        ylabel('Smoothed Loss');
        title(['N = ', num2str(N_values(idx))]);
        grid on;
    end
end
```

Ερώτημα 3

```
clc;
close all;
clear all;
% Load data
load('data21.mat'); % Generative model (A1, A2, B1, B2)
load('data23.mat'); % Downsampled and noisy data (Xn, Xi)

% Extract matrices and parameters
A1 = A_1; % 128 x 10
A2 = A_2; % 784 x 128
B1 = B_1; % 128 x 1
B2 = B_2; % 784 x 1
Xi = X_i; % Ideal high-resolution images (784 x 4)
Xn = X_n; % Downsampled noisy images (49 x 4)

% Define T (49 x 784 matrix for downsampling)
T = zeros(49, 784);
for i = 1:7
    for j = 1:7
        row_idx = (i-1)*7 + j;
        col_start = (i-1)*4*28 + (j-1)*4 + 1;
        for x = 0:3
            for y = 0:3
                T(row_idx, col_start + x*28 + y) = 1/16;
            end
        end
    end
end

% Parameters
max_iterations = 150000; % Maximum iterations for gradient descent
learning_rate = 0.0003; % Learning rate
lambda = 0.0007; % Regularization parameter

% Initialize storage for loss histories
loss_histories = zeros(max_iterations, size(Xn, 2)); % Store losses for
each image

% Process each column of Xn
for col = 1:size(Xn, 2)
    fprintf('Processing image %d...\n', col);

    % Extract noisy input
    Xn_col = Xn(:, col);

    % Initialize Z (latent variable)
    Z = randn(10, 1);

    % Gradient Descent
    for iter = 1:max_iterations
        % Forward pass
        W1 = A1 * Z + B1; % Linear transformation
        Z1 = max(W1, 0); % ReLU activation
        W2 = A2 * Z1 + B2; % Linear transformation
        X = 1 ./ (1 + exp(W2)); % Sigmoid activation

        % Compute loss
        residual = T * X - Xn_col; % Residual vector
```

```
        norm_squared = norm(residual)^2; % Squared norm of the residual
        loss = log(norm_squared) + lambda * norm(Z)^2; % Loss function
        loss_histories(iter, col) = loss; % Store the loss

        % Backpropagation
        gradient_loss = (2 / norm_squared) * (T' * residual); % Gradient of
the loss
        f2_derivative = -(exp(W2)) ./ ((1 + exp(W2)).^2); % Derivative of
sigmoid
        v2 = gradient_loss .* f2_derivative; % Backprop through sigmoid
        u1 = A2' * v2; % Backprop through A2
        v1 = u1 .* (W1 > 0); % Backprop through ReLU
        grad = A1' * v1 + 2 * lambda * Z; % Total gradient

        % Update Z
        Z = Z - learning_rate * grad;
    end

    % Reconstruct high-resolution image
    W1 = A1 * Z + B1;
    Z1 = max(W1, 0);
    W2 = A2 * Z1 + B2;
    X_reconstructed = 1 ./ (1 + exp(W2)); % Reconstructed image
        subplot(size(Xn, 2), 3, (col - 1) * 3 + 1); % Ideal Image
    imshow(reshape(Xi(:, col), [28, 28]), []);
    title(['Ideal Image ', num2str(col)]);

    subplot(size(Xn, 2), 3, (col - 1) * 3 + 2); % Noisy Low-Resolution
    imshow(reshape(T' * Xn_col, [28, 28]), []);
    title(['Noisy (Col ', num2str(col), ')']);

    subplot(size(Xn, 2), 3, (col - 1) * 3 + 3); % Reconstructed Image
    imshow(reshape(X_reconstructed, [28, 28]), []);
    title(['Reconstructed (Col ', num2str(col), ')']);
end

% Plot all loss histories on the same graph
figure('Name', 'Loss Histories', 'NumberTitle', 'off');
hold on;
for col = 1:size(Xn, 2)
    plot(1:max_iterations, loss_histories(:, col), 'LineWidth', 1.5);
end
xlabel('Iterations');
ylabel('Loss');
legend(arrayfun(@(x) ['Image ', num2str(x)], 1:size(Xn, 2),
'UniformOutput', false));
title('Loss Histories for All Images');
grid on;
hold off;
```