1-Multidimensional_Scaling

February 15, 2018

```
In [1]: # Importing libraries
        %matplotlib inline
        import matplotlib.pyplot as plt
        import tensorflow as tf
        import numpy as np
        from tensorflow.examples.tutorials.mnist import input_data
        from scipy.spatial import distance_matrix
        from ipywidgets import FloatProgress
        from IPython.display import display
```

```
/home/marcus/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning: Conversi
  from ._conv import register_converters as _register_converters
```

```
In [2]: # Read data
        data = input_data.read_data_sets("MNIST/", one_hot=True)
```

```
Extracting MNIST/train-images-idx3-ubyte.gz
Extracting MNIST/train-labels-idx1-ubyte.gz
Extracting MNIST/t10k-images-idx3-ubyte.gz
Extracting MNIST/t10k-labels-idx1-ubyte.gz
```

```
In [3]: # Create 10 buckets with 1000 samples of each digit
        samples = [[] for i in range(10)]
        for image, label in zip(data.train.images, data.train.labels):
            label = np.argmax(label)
            if len(samples[label]) < 1000:
                samples[label].append(image * 2 - 1) # Convert [0, 1] -> [-1, 1] range
        samples = [image for s in samples for image in s] # flatten
        N = len(samples)
```

```
In [4]: # Calculate distances between all samples
        from sklearn.metrics.pairwise import euclidean_distances
        D_matrix = euclidean_distances(samples, samples) # calculating the differences of squa
```

```
In [5]: tf.reset_default_graph()
```

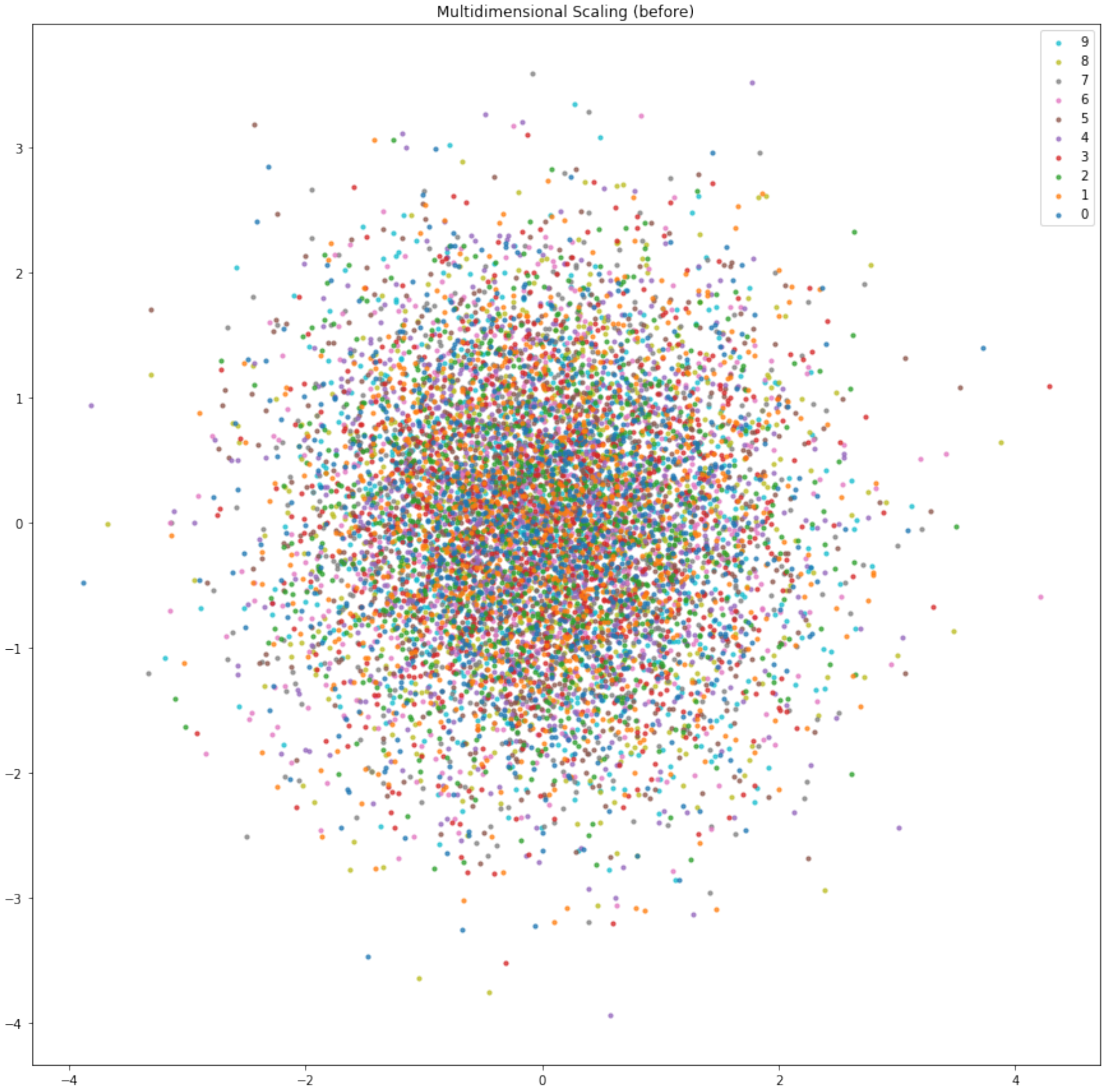
```
#####################################################
## tf_distance_matrix
## Calculates the distance matrix of X (a tf.Tensor)
## Source: https://stackoverflow.com/questions/37009647/compute-pairwise-distance-in-a
#####################################################
def tf_distance_matrix(X):
    r = tf.reduce_sum(X * X, 1)
    r = tf.reshape(r, [-1, 1])
    return tf.sqrt(r - 2 * tf.matmul(X, tf.transpose(X)) + tf.transpose(r) + 0.001)

D = tf.placeholder(tf.float32, [N, N]) # Placeholder for distance matrix D -> [10000, 

X_prime = tf.get_variable("X_prime", initializer=tf.random_normal((N, 2), stddev=1.0))
D_prime = tf_distance_matrix(X_prime) # Distance matrix D' -> [10000, 10000]

# Loss function
loss = tf.reduce_mean(tf.pow(D_prime - D, 2))
```

In [6]:

```
config = tf.ConfigProto(device_count = {'GPU': 0}) # Force CPU
sess = tf.InteractiveSession(config=config)
tf.global_variables_initializer().run()
x_prime = sess.run(X_prime)
```

In [7]:

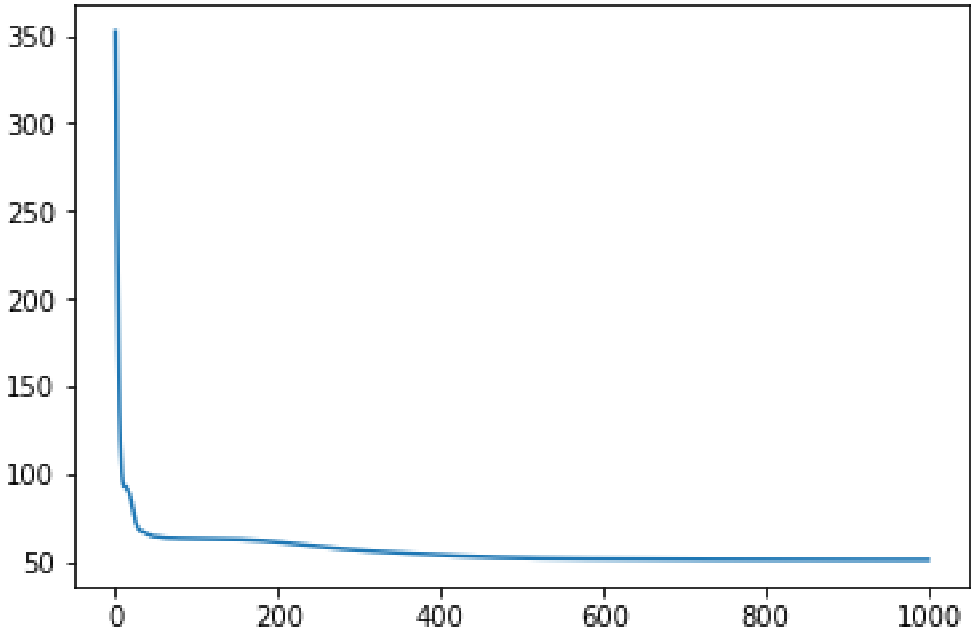
```
def visualize(X, title=""):
    colors = ["C0", "C1", "C2", "C3", "C4", "C5", "C6", "C7", "C8", "C9"]
    x = [x[0] for x in X]
    y = [x[1] for x in X]
    fig, ax = plt.subplots(figsize=(15, 15))
    ax.set_title(title)
    n = len(x_prime) // 10
    for i in reversed(range(10)):
        ax.scatter(x[i*n:(i+1)*n], y[i*n:(i+1)*n],
                   s=10.0, c=colors[i], label=str(i), alpha=0.75)
    ax.legend()

# Show X_prime before optimization
visualize(x_prime, title="Multidimensional Scaling (before)")
```



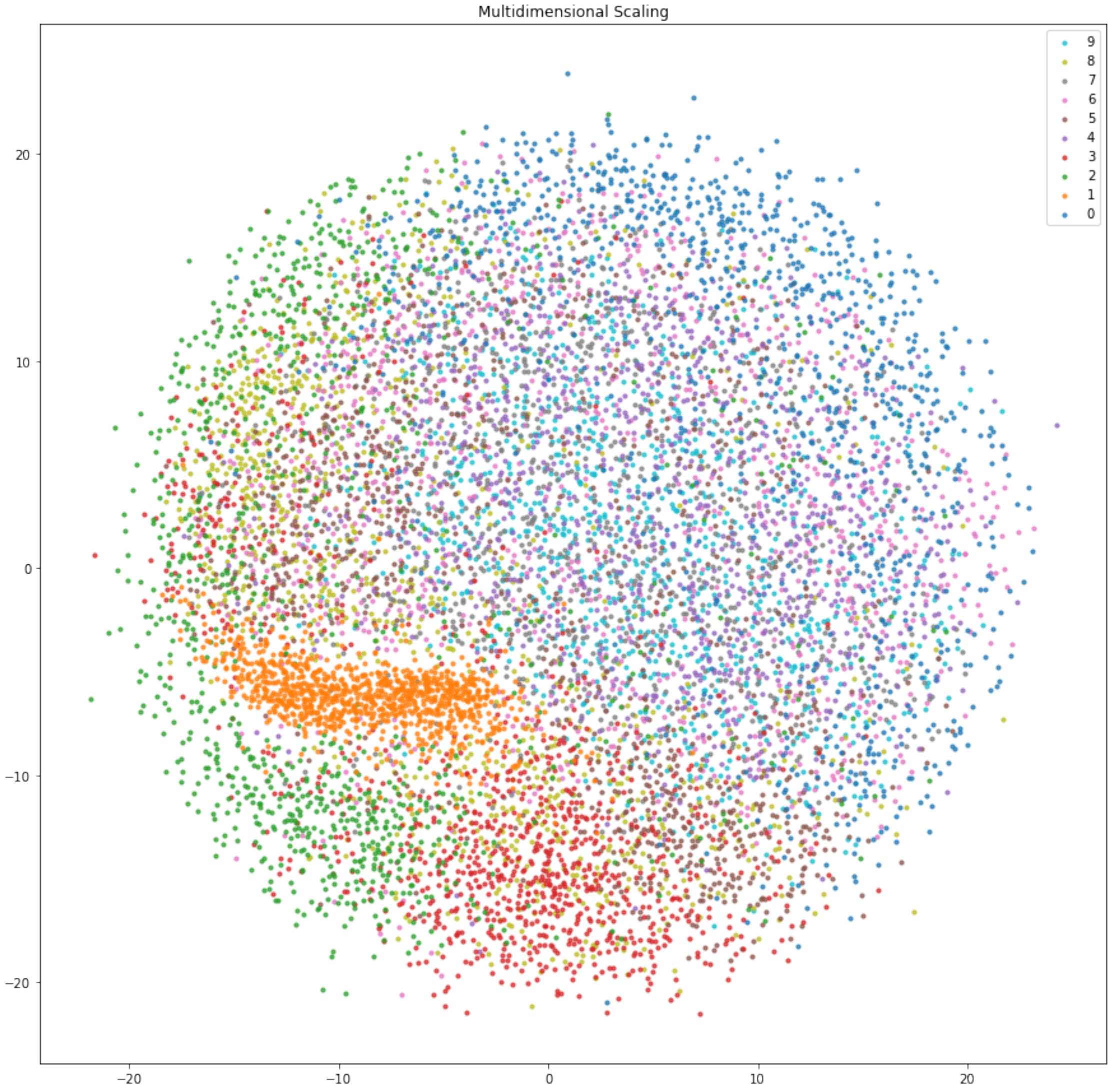
```
In [13]: # Define optimizer
         optim = tf.train.AdamOptimizer(learning_rate=1.0).minimize(loss)
         tf.global_variables_initializer().run()

In [14]: # Minimize the loss
         num_iter = 1000
         loss_values = []
         progress = FloatProgress(min=0, max=num_iter); display(progress)
         for i in range(num_iter):
             progress.description = "Epoch %i/%i" % (i, num_iter)
             _, l = sess.run([optim, loss], feed_dict = { D: D_matrix })
             loss_values.append(l)
             progress.value += 1
```

```
A Jupyter Widget
```

```
In [15]: plt.plot(loss_values)
         plt.show()
```



```
In [16]: x_prime = sess.run(X_prime)
         visualize(x_prime, title="Multidimensional Scaling")
```



We observe that 1s and 0s have little overlap as they are very distinct (circle vs a line). Otherwise, the other numbers seems to group a bit (2s are in the lower-left, and 3s in the lower-middle), but tend to blend together a bit more.

2-Farthest_Point_Sampling

February 15, 2018

```
In [1]: import pymesh
        import numpy as np
        from ipywidgets import FloatProgress
        from IPython.display import display
        from scipy.spatial import distance_matrix

        object_name = "teapot" # object_name = "violin_case"
        mesh = pymesh.load_mesh("%s.obj" % object_name)
```

```
In [2]: ###########################################################################
        # def triangle_area(v0, v1, v2):
        # Returns the area of a triangle given three points (v0, v1, v2)
        # Given by Area = |AB x AC| / 2 (half of the length of the cross product)
        ###########################################################################
        def triangle_area(v0, v1, v2):
            return np.linalg.norm(np.cross(np.array(v1) - np.array(v0),
                                           np.array(v2) - np.array(v0))) * 0.5
```

```
In [3]: # Calculate the total surface area of the mesh
        total_area = 0
        for face in mesh.faces:
            v0 = mesh.vertices[face[0]]
            v1 = mesh.vertices[face[1]]
            v2 = mesh.vertices[face[2]]
            total_area += triangle_area(v0, v1, v2)
```

```
In [4]: # Calculate weight per triangle
        triangle_weights = []
        for face in mesh.faces:
            v0 = mesh.vertices[face[0]]
            v1 = mesh.vertices[face[1]]
            v2 = mesh.vertices[face[2]]
            triangle_weights.append(triangle_area(v0, v1, v2) / total_area)
```

```
In [5]: # Sample points along mesh surface
        num_points = 10000
        point_cloud = []
        for face, weight in zip(mesh.faces, triangle_weights):
            num_points_in_triangle = weight * num_points
            v0 = mesh.vertices[face[0]]
            v1 = mesh.vertices[face[1]]
            v2 = mesh.vertices[face[2]]
            for _ in range(int(np.ceil(num_points_in_triangle))):
                r1 = np.random.rand()
                r2 = np.random.rand()
                D = (1 - np.sqrt(r1)) * v0 + np.sqrt(r1) * (1 - r2) * v1 + np.sqrt(r1) * r2 * v
                point_cloud.append(D)

                if len(point_cloud) >= num_points:
                    break
            if len(point_cloud) >= num_points:
                    break
```

```
In [6]: num_samples = 1000

        i = np.random.randint(0, len(point_cloud))
        S = [point_cloud[i]]
        del point_cloud[i]

        progress = FloatProgress(min=1, max=num_samples); display(progress)

        # Generate new point cloud using farthest point sampling
        while len(S) < num_samples:
            D = distance_matrix(S, point_cloud)
            progress.value = len(S)
            progress.description = "%i/%i" % (len(S), num_samples)
            d_max = 0
            for i in range(len(point_cloud)):
                d_min = float("inf")
                for j in range(len(S)):
                    d_min = min(D[j, i], d_min)
                if d_min > d_max:
                    d_max = d_min
                    q_farthest = i
            S.append(point_cloud[q_farthest])
            del point_cloud[q_farthest]
```

```
FloatProgress(value=1.0, max=1000.0, min=1.0)
```

```
In [7]: import pickle
        def save_object(obj, filename):
            with open(filename, 'wb') as output:
                pickle.dump(obj, output, pickle.HIGHEST_PROTOCOL)
        save_object(S, "%s.cloud" % object_name)
```

2-Farthest_Point_Sampling_Visualization

February 15, 2018

```
In [1]: import pickle
        from pyntcloud import PyntCloud
        import numpy as np
        import pandas as pd

        def load_object(filename):
            with open(filename, "rb") as f:
                return pickle.load(f)
```

```
In [2]: # Load and display pickled point cloud of teapot
        point_cloud = np.array(load_object("teapot.cloud"))

        # Setup point cloud visualization
        points = pd.DataFrame(point_cloud, columns=["x", "y", "z"])
        colors = np.full((point_cloud.shape[0], 3), (255, 255, 255), dtype=np.uint8)
        points[["red", "blue", "green"]] = pd.DataFrame(colors, index=points.index)

        # Show point cloud
        cloud = PyntCloud(points)
        cloud.plot(point_size=1.0, opacity=1.0, lines=[], line_color=[])
```

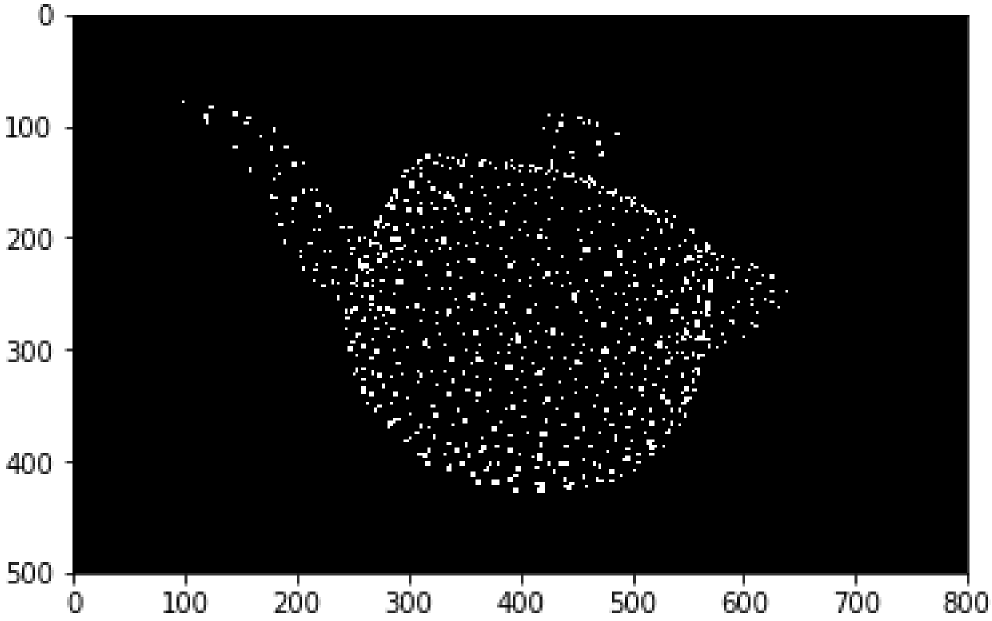
```
Out[2]: <IPython.lib.display.IFrame at 0x7f3e20460588>
```

```
In [3]: # Load and display pickled point cloud of violin case
        point_cloud = np.array(load_object("violin_case.cloud"))

        # Setup point cloud visualization
        points = pd.DataFrame(point_cloud, columns=["x", "y", "z"])
        colors = np.full((point_cloud.shape[0], 3), (255, 255, 255), dtype=np.uint8)
        points[["red", "blue", "green"]] = pd.DataFrame(colors, index=points.index)

        # Show point cloud
        cloud = PyntCloud(points)
        cloud.plot(point_size=0.01, opacity=1.0, lines=[], line_color=[])
```

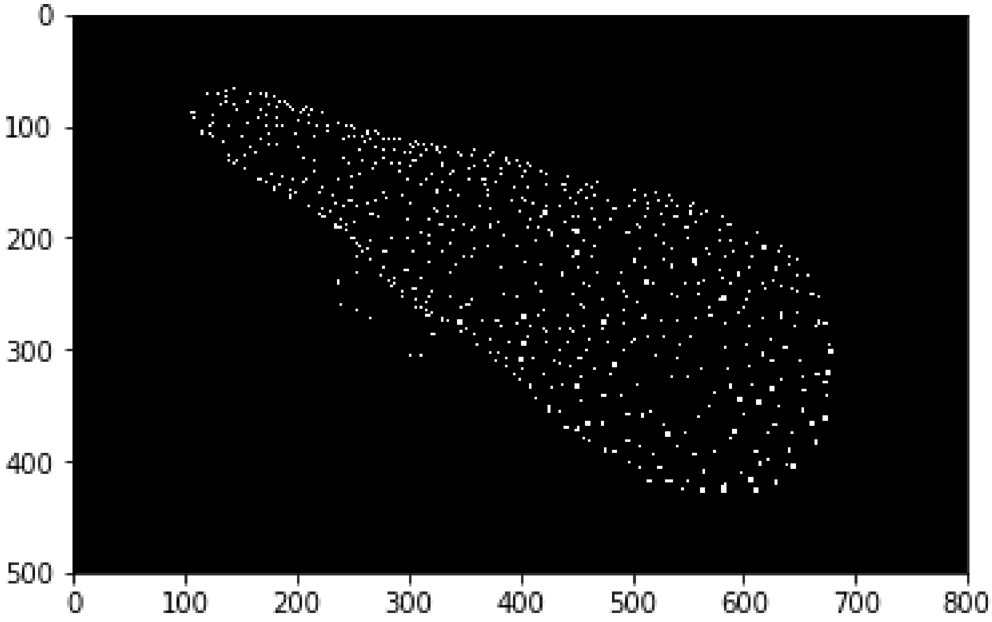
```
Out[3]: <IPython.lib.display.IFrame at 0x7f3e65334080>
```

```
In [4]: import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
```

```
# Plot screenshot for pdf-file
plt.imshow(mpimg.imread("teapot.png"))
plt.show()
```



```
In [5]: plt.imshow(mpimg.imread("violin_case.png"))
        plt.show()
```



3-Earth_Movers_Distance

February 15, 2018

```
In [1]: import tensorflow as tf
        import tf_emddistance
        import numpy as np
        import matplotlib.pyplot as plt
        from ipywidgets import FloatProgress
        from IPython.display import display
```

```
/home/marcus/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning: Convers
  from ._conv import register_converters as _register_converters
```

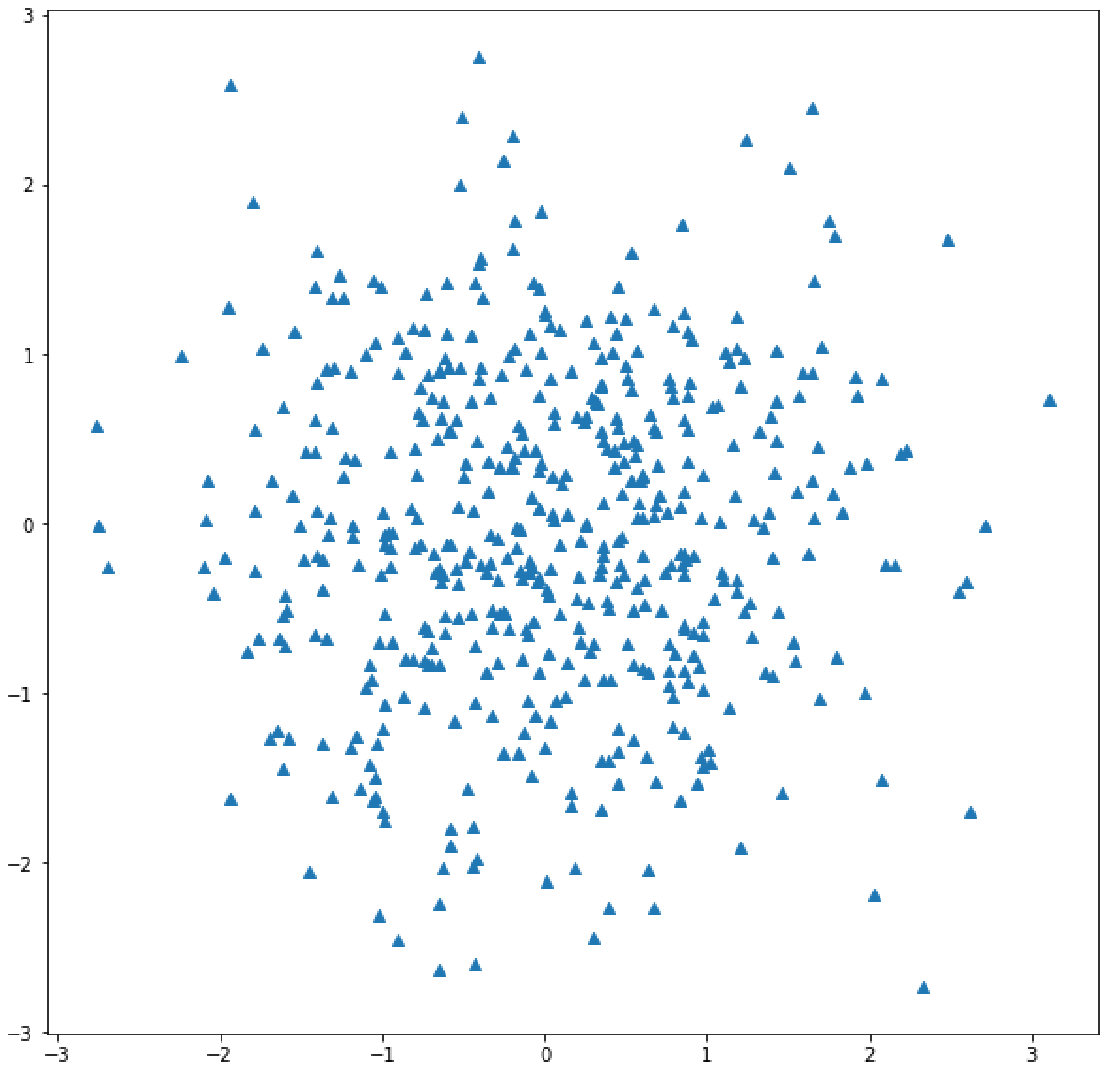
```
In [2]: # Generate 100 circles with radii in the range [0, 10)
        num_samples = 500
        radii = np.arange(0, 10, 0.1)
        num_clouds = len(radii)
        S = np.zeros((num_clouds, num_samples, 3)) # Cricles -> [100, 500, 3]
        for i, r in enumerate(radii):
            # Sample the circumference
            for j in range(num_samples):
                S[i, j, 0] = np.cos(2*np.pi * j / num_samples) * r
                S[i, j, 1] = np.sin(2*np.pi * j / num_samples) * r
                S[i, j, 2] = 0
        S_feed = S
```

```
In [3]: tf.reset_default_graph()

        # Variable X -> [500, 3]
        # Point cloud with which to minimize the EMD distance to
        X = tf.get_variable("X", initializer=tf.random_normal((num_samples, 3), stddev=1.0))

        # Stack X into 100 replicas -> [100, 500, 3]
        X_stacked = tf.stack([X for _ in range(num_clouds)])

        # Placeholder for input point clouds -> [100, 500, 3]
        S = tf.placeholder(tf.float32, [num_clouds, num_samples, 3])

        # Calculate EMD distance
        dist, idx1, idx2 = tf_emddistance.emd_distance(X_stacked, S)
```

```
    # The loss will be the average of all the distances of the points
    loss = tf.reduce_mean(dist)
```

emd

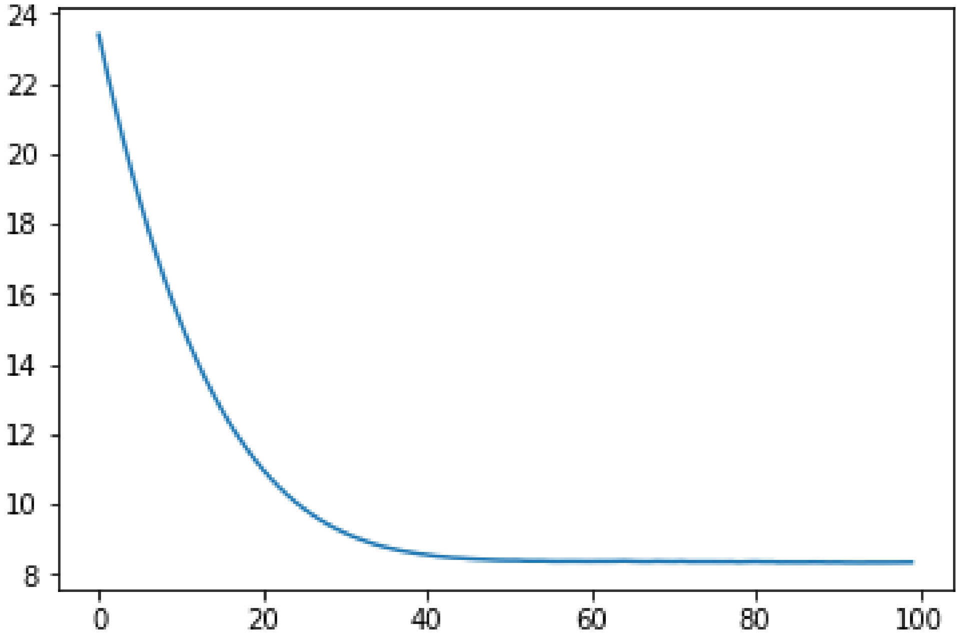
```
In [4]: def visualize(X):
            plt.figure(figsize=(10, 10))
            plt.scatter(X[:, 0], X[:, 1], marker='^', c="C0")
            plt.show()

        sess = tf.InteractiveSession()
        tf.global_variables_initializer().run()
        visualize(sess.run(X))
```



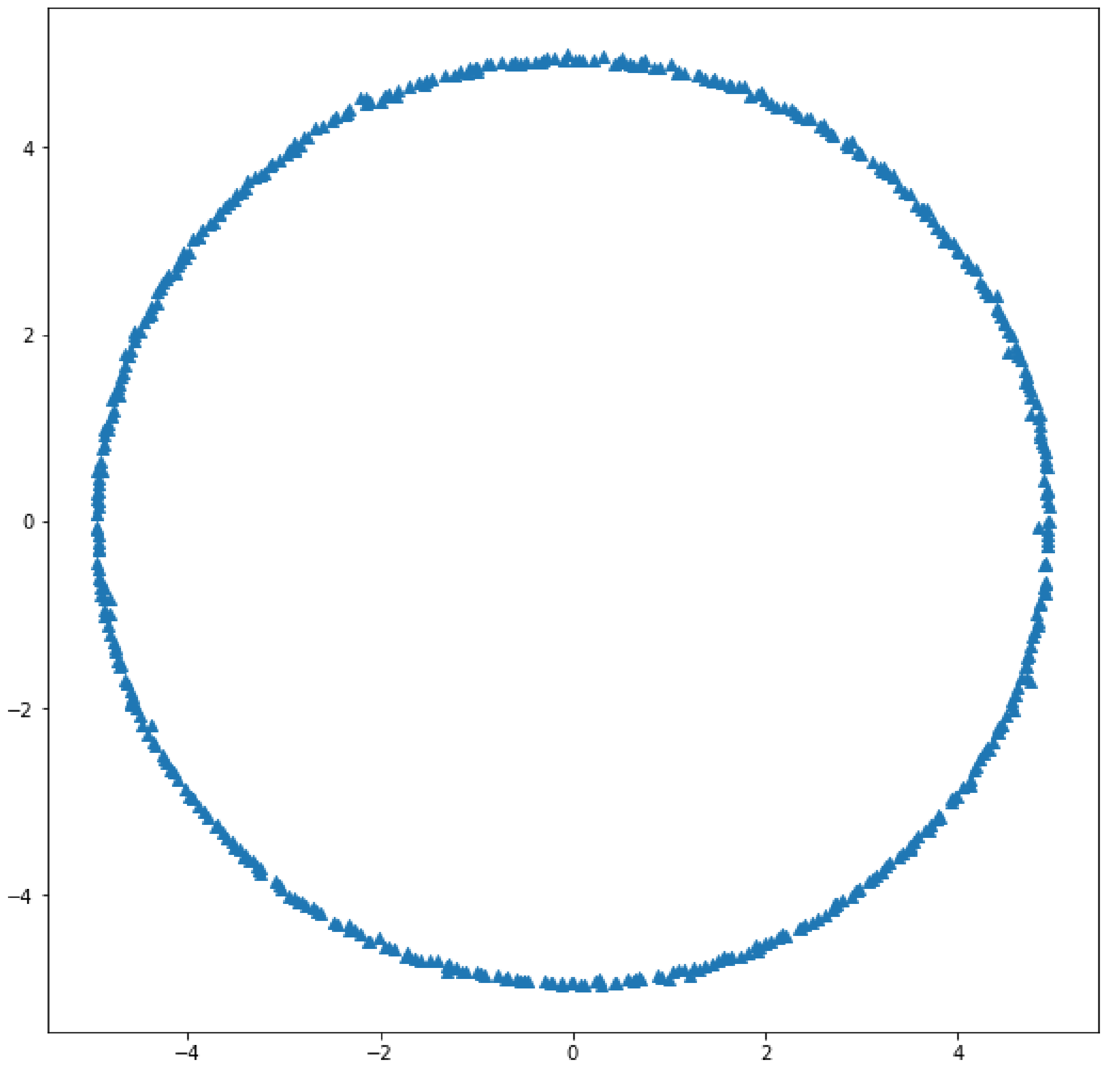
```
In [5]: # Define optimizer
        optim = tf.train.AdamOptimizer(learning_rate=0.1).minimize(loss)
        tf.global_variables_initializer().run()

        # Minimize the loss
        num_iter = 100
        loss_values = []
        progress = FloatProgress(min=0, max=num_iter); display(progress)
        for i in range(num_iter):
            progress.description = "Epoch %i/%i" % (i, num_iter)
            _, l = sess.run([optim, loss], feed_dict = { S: S_feed })
            loss_values.append(l)
            progress.value += 1
```

```
A Jupyter Widget
```

```
In [6]: plt.plot(loss_values)
        plt.show()
```



```
In [7]: visualize(sess.run(X))
```


4
2
0
−2
−4
−4
−2
0
2
4

3-Earth_Movers_Distance-Hungarian

February 15, 2018

```
In [1]: import numpy as np
        from scipy.spatial import distance_matrix
        from scipy.optimize import linear_sum_assignment
        from ipywidgets import FloatProgress
        from IPython.display import display
```

```
In [2]: import pickle
        def load_object(filename):
            with open(filename, 'rb') as f:
                return pickle.load(f)
```

```
In [3]: point_cloud_0 = np.array(load_object("../teapot.cloud"))
        point_cloud_1 = np.array(load_object("../violin_case.cloud"))
        np.random.shuffle(point_cloud_0)
        np.random.shuffle(point_cloud_1)
        point_cloud_0 = point_cloud_0[:10]
        point_cloud_1 = point_cloud_1[:10]
        D = distance_matrix(point_cloud_0, point_cloud_1)
```

```
In [4]: # Source: https://en.wikipedia.org/wiki/Hungarian_algorithm
        def hungarian(cost_matrix):
            # Copy cost matrix
            cost_matrix = np.copy(cost_matrix)
            N = cost_matrix.shape[0]

            # Subtract row minima
            for row in range(N):
                cost_matrix[row, :] -= np.min(cost_matrix[row, :])

            # Subtract column minima
            for col in range(N):
                cost_matrix[:, col] -= np.min(cost_matrix[:, col])

            progress = FloatProgress(min=0, max=N); display(progress)
            #for _ in range(3):
            while True:
                #print("Cost")
                #print(cost_matrix)
```

```
# Find valid row and column assignments
assigned_rows = np.full(N, False, dtype=np.bool)
assigned_columns = np.full(N, False, dtype=np.bool)
zeros_status = np.full((N, N), 0, dtype=np.int8) # 0 = unassigned, 1 = assigne

# For every row, if the row contains only one 0, assign it
for row in range(N):
    column_indices = np.argwhere(np.isclose(cost_matrix[row, :], 0))
    if len(column_indices) == 1: # Only one zero
        col = column_indices[0][0]
        if not assigned_columns[col]: # If column not assigned already
            # Assign column and row
            assigned_columns[col] = True
            assigned_rows[row] = True

            # Cancel other zeroes in the column
            zeros_status[np.where(np.isclose(cost_matrix[:, col], 0)), col] = -
            zeros_status[row, col] = 1

# For every column, if the column contains only one 0, assign it
for col in range(N):
    row_indices = np.argwhere(np.isclose(cost_matrix[:, col], 0))
    if len(row_indices) == 1: # Only one zero
        row = row_indices[0][0]
        if not assigned_rows[row] and not assigned_columns[col]: # If row and
            # Assign column and row
            assigned_columns[col] = True
            assigned_rows[row] = True

            # Cancel other zeroes in the row
            zeros_status[row, np.where(np.isclose(cost_matrix[row, :], 0))] = -
            zeros_status[row, col] = 1

#print("Zeros")
#print(zeros_status)

# Check for single zeroes after cancelation
for row in range(N):
    if not assigned_rows[row]:
        column_index = -1
        for col in range(N):
            if np.isclose(cost_matrix[row, col], 0) and zeros_status[row, col]
                if column_index == -1:
                    column_index = col
                else:
                    column_index = -1
                    break
        if column_index != -1:# and not assigned_columns[column_index]:
            # Assign column and row
            assigned_columns[column_index] = True
            assigned_rows[row] = True
            zeros_status[row, column_index] = 1

for col in range(N):
    if not assigned_columns[col]:
        row_index = -1
        for row in range(N):
            if np.isclose(cost_matrix[row, col], 0) and zeros_status[row, col]
                if row_index == -1:
                    row_index = row
                else:
                    row_index = -1
                    break

        if row_index != -1:# and not assigned_rows[row_index]:
            # Assign column and row
            assigned_columns[col] = True
            assigned_rows[row_index] = True
            zeros_status[row_index, col] = 1

#print("#Assignments")
#print(np.sum(zeros_status == 1))

# If number of assignments == N, we have reached the optimal solution
num_assignments = np.sum(zeros_status == 1)
progress.value = num_assignments
progress.description = "%i/%i" % (num_assignments, N)
if num_assignments == N:
    return list(range(N)), [np.argmax(zeros_status[row]) for row in range(N)]

# Create lines by marking rows and columns
marked_rows = np.full(N, False, dtype=np.bool)
marked_columns = np.full(N, False, dtype=np.bool)
for row in range(cost_matrix.shape[0]):
    # For every row without assignment
    if not assigned_rows[row]:
        # Mark the row
        marked_rows[row] = True

        # Find the zeros in the row
        for col in range(cost_matrix.shape[1]):
            if np.isclose(cost_matrix[row, col], 0):
                # Mark the column
                marked_columns[col] = True
```

```
                                # Find row with assignment in the current column
                                for row2 in range(cost_matrix.shape[0]):
                                    if zeros_status[row2, col] == 1:
                                        marked_rows[row2] = True

                lines = np.full((N, N), 0, dtype=np.uint8)
                for row in range(cost_matrix.shape[0]):
                    for col in range(cost_matrix.shape[1]):
                        if not marked_rows[row]:
                            lines[row, col] += 1
                        if marked_columns[col]:
                            lines[row, col] += 1

                #print("Lines")
                #print(lines)

                min_uncovered = np.min(cost_matrix[np.where(lines == 0)])
                cost_matrix[np.where(lines == 0)] -= min_uncovered # subtract uncovered values
                cost_matrix[np.where(lines == 2)] += min_uncovered # add to intersections


        row_ind, col_ind = linear_sum_assignment(D)
        print("linear_sum_assignment(D) ->")
        print(D[row_ind, col_ind].sum())

        row_ind, col_ind = hungarian(D)
        print("hungarian(D) ->")
        print(D[row_ind, col_ind].sum())
```

```
linear_sum_assignment(D) ->
557.3235828799391


A Jupyter Widget


hungarian(D) ->
556.5343843579753
```

Unfortunately it gets stuck on bigger examples, but here I show it works when I sample 10 points of the point clouds.

4-Denoiser

February 15, 2018

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import tensorflow as tf
        import numpy as np
        from tensorflow.examples.tutorials.mnist import input_data
        from ipywidgets import FloatProgress
        from IPython.display import display
```

```
/home/marcus/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:34: FutureWarning: Convers
  from ._conv import register_converters as _register_converters
```

```
In [2]: # Read data
        data = input_data.read_data_sets("MNIST/", one_hot=True)
```

```
Extracting MNIST/train-images-idx3-ubyte.gz
Extracting MNIST/train-labels-idx1-ubyte.gz
Extracting MNIST/t10k-images-idx3-ubyte.gz
Extracting MNIST/t10k-labels-idx1-ubyte.gz
```

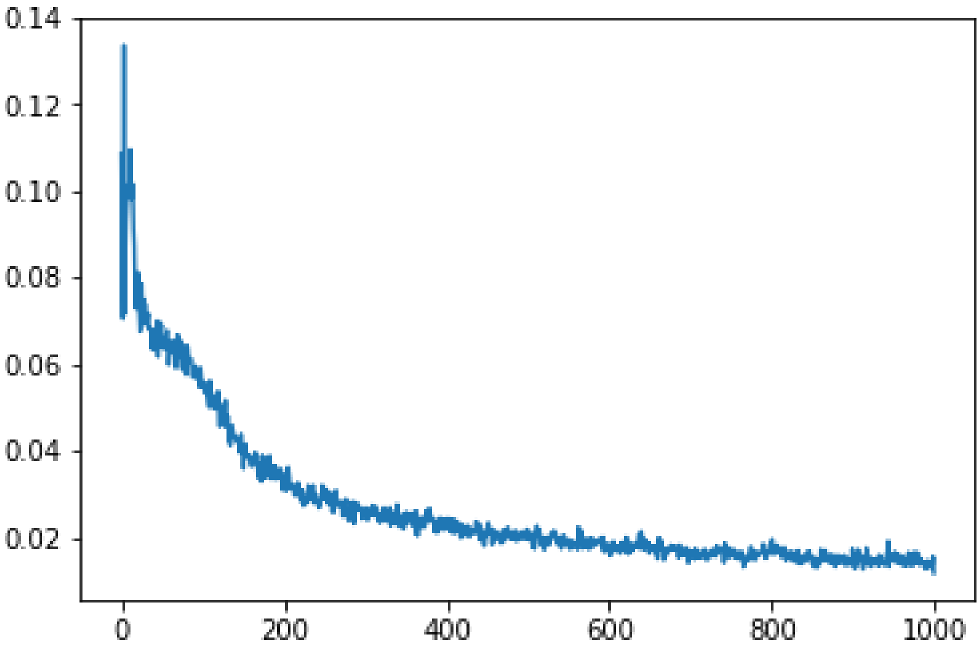
```
In [3]: # Placeholder for noisy image
        img_noisy = tf.placeholder(tf.float32, [None, 784])

        # Placeholder for original image
        img_original = tf.placeholder(tf.float32, [None, 784])

        # Reshape the input -> [?, 28, 28, 1]
        input_layer = tf.reshape(img_noisy, [-1, 28, 28, 1])

        #############################################
        # Encoder
        #############################################

        # Convolutional layer 1 -> [?, 24, 24, 32]
        conv1 = tf.layers.conv2d(inputs=input_layer, filters=32, kernel_size=(5, 5), padding="

        # Max pool -> [?, 12, 12, 32]
```

```
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)

# Convolutional layer 2 -> [?, 8, 8, 64]
conv2 = tf.layers.conv2d(inputs=pool1, filters=64, kernel_size=(5, 5), padding="valid"

# Max pool -> [?, 4, 4, 64]
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)

# Flatten result -> [?, 1024]
conv2_flat = tf.reshape(pool2, [-1, 4 * 4 * 64])

# Feed to dense layer 1 -> [?, 1024] 4*4*64
dense1 = tf.layers.dense(inputs=conv2_flat, units=750, activation=tf.nn.relu)

# Dense layer 2 -> [?, 100]
dense2 = tf.layers.dense(inputs=dense1, units=100, activation=tf.nn.relu)

# Reshape dense into 3D tensor -> [?, 10, 10, 1]
dense_reshaped = tf.reshape(dense2, [-1, 10, 10, 1])

#############################################
# Decoder
#############################################

# Transpose convolusion 1 -> [?, 14, 14, 32]
conv_trans_1 = tf.layers.conv2d_transpose(dense_reshaped,
                                          filters=32,
                                          kernel_size=(5, 5),
                                          strides=(1, 1),
                                          padding="valid",
                                          activation=tf.nn.relu)

# Transpose convolusion 2 -> [?, 18, 18, 64]
conv_trans_2 = tf.layers.conv2d_transpose(conv_trans_1,
                                          filters=64,
                                          kernel_size=(5, 5),
                                          strides=(1, 1),
                                          padding="valid",
                                          activation=tf.nn.relu)

# Transpose convolusion 3 -> [?, 28, 28, 1]
conv_trans_3 = tf.layers.conv2d_transpose(conv_trans_2,
                                          filters=1,
                                          kernel_size=(11, 11),
                                          strides=(1, 1),
                                          padding="valid",
                                          activation=tf.nn.relu)
```

```
        # Calculate loss
        loss = tf.reduce_mean(tf.pow(tf.subtract(tf.reshape(conv_trans_3, [-1, 784]), img_orig

In [4]: optim = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

        sess = tf.InteractiveSession()
        tf.global_variables_initializer().run()

        # Make batches to train
        num_iter = 1000
        batch_size = 64
        loss_values = []
        progress = FloatProgress(min=0, max=num_iter); display(progress)
        for i in range(num_iter):
            progress.description = "Epoch %i/%i" % (i, num_iter)

            # Get next batch
            batch_img, _ = data.train.next_batch(batch_size)

            # Add noise to the batch
            batch_img_noisy = np.copy(batch_img) + np.random.normal(loc=0.0, scale=0.15, size=

            # Do training
            _, l = sess.run([optim, loss], feed_dict={ img_noisy: batch_img_noisy, img_original

            loss_values.append(l)

            # Show progress
            progress.value += 1
```

```
A Jupyter Widget
```

```
In [5]: plt.plot(loss_values)
        plt.show()
```



```
In [6]: batch_img = data.test.images[:10]
        batch_img_noisy = np.copy(batch_img) + np.random.normal(loc=0.0, scale=0.15, size=batc
        output_imgs = sess.run(conv_trans_3, feed_dict={ img_noisy: batch_img_noisy, img_origi

        fig, axes = plt.subplots(3, 10, figsize=(12, 4))
        for i, (img_org, img_noise, img_denoise) in enumerate(zip(batch_img, batch_img_noisy, o
            ax = axes[0, i]
            ax.axis("off")
            ax.imshow(img_org.reshape(28, 28), cmap="binary")

            ax = axes[1, i]
            ax.axis("off")
            ax.imshow(img_noise.reshape(28, 28), cmap="binary")

            ax = axes[2, i]
            ax.axis("off")
            ax.imshow(img_denoise.reshape(28, 28), cmap="binary")

        plt.tight_layout()
```