Alberto Marinelli (638283)

Midterm 2 (assignment 3)
Restricted Boltzmann Machines

Implement from scratch an RBM and apply it to DSET3. The RBM should be implemented fully by you (both CD-1 training and inference steps) but you are free to use library functions for the rest (e.g. image loading and management, etc.).

1. Train an RBM with a number of hidden neurons selected by you (single layer) on the MNIST data (use the training set split provided by the website).

2. Use the trained RBM to encode all the images using the corresponding activation of the hidden neurons.

3. Train a simple classifier (e.g. any simple classifier in scikit) to recognize the MNIST digits using as inputs their encoding obtained at step 2. Use the standard training/test split. Show a performance metric of your choice in the presentation/handout.

Initialization and parameters

v_units: number of visible units

h_units: number of hidden units

weights: weights

v_bias: biases of visible units

h_bias: biases of hidden units

path_directory: string that identify the directory of dataset

[Code]

```
def __init__(self, v_units, h_units=1, weights=None, v_bias=None, h_bias=None, path_directory=None):

    self.tr_imgs, self.tr_labels, self.ts_imgs, self.ts_labels = load_data(path_directory)

    self.v_units = v_units # Number of visible units
    self.h_units = h_units # Number of hidden units

    self.weights = weights if weights is not None else np.random.uniform(-1, 1, size=(h_units, v_units)) # Weights matrix
    self.v_bias = v_bias if v_bias is not None else np.zeros(v_units) # Bias vector for visible layer
    self.h_bias = h_bias if h_bias is not None else np.zeros(h_units) # Bias vector for hidden layer

    self.classifier = tf.keras.models.Sequential([  # Set classifier
        Dense(units=200, activation='relu', input_dim=h_units),
        Dense(units=100, activation='relu'),
        Dense(units=50, activation='relu'),
        Dense(units=10, activation='softmax')
    ])
```

Training – Contrastive divergence

[Code]
v_probs: visible units activations
k: order of the Gibbs sampling

```
def contrastive_divergence(self, v_probs, k):
    # COMPUTE WAKE
    v_sample = np.random.binomial(n=1, p=v_probs, size=len(v_probs))
    h_probs, h_sample = self.hidden_visible(v_sample)
    wake = np.outer(h_probs, v_probs)

    # COMPUTE DREAM
    v_probs_gibbs, v_sample_gibbs, h_probs_gibbs, h_sample_gibbs = self.gibbs_sampling(h_sample, k)
    dream = np.outer(h_probs_gibbs, v_probs_gibbs)

    deltaW = np.subtract(wake, dream)
    deltaBv = np.subtract(v_sample, v_sample_gibbs)
    deltaBh = np.subtract(h_sample, h_sample_gibbs)

    return deltaW, deltaBv, deltaBh
```

Training – Gibbs Sampling

[Code]
h_sample: binary vector of the hidden activation
k: order of the Gibbs sampling

```
def hidden_visible(self, v_sample):
    h_probs = sigmoid(np.add(np.matmul(self.weights, v_sample), self.h_bias))
    h_samples = np.random.binomial(n=1, p=h_probs, size=len(h_probs))
    return h_probs, h_samples

def visible_hidden(self, h_sample):
    v_probs = sigmoid(np.add(np.matmul(h_sample, self.weights), self.v_bias))
    v_samples = np.random.binomial(n=1, p=v_probs, size=len(v_probs))
    return v_probs, v_samples

def gibbs_sampling(self, h_sample, k):
    v_prob, v_sample, h_prob = None, None, None
    for i in range(k):
        v_prob, v_sample = self.visible_hidden(h_sample)
        h_prob, h_sample = self.hidden_visible(v_sample)
    return v_prob, v_sample, h_prob, h_sample
```

Trained RBM and results

(1) **Hidden neurons**: **10**

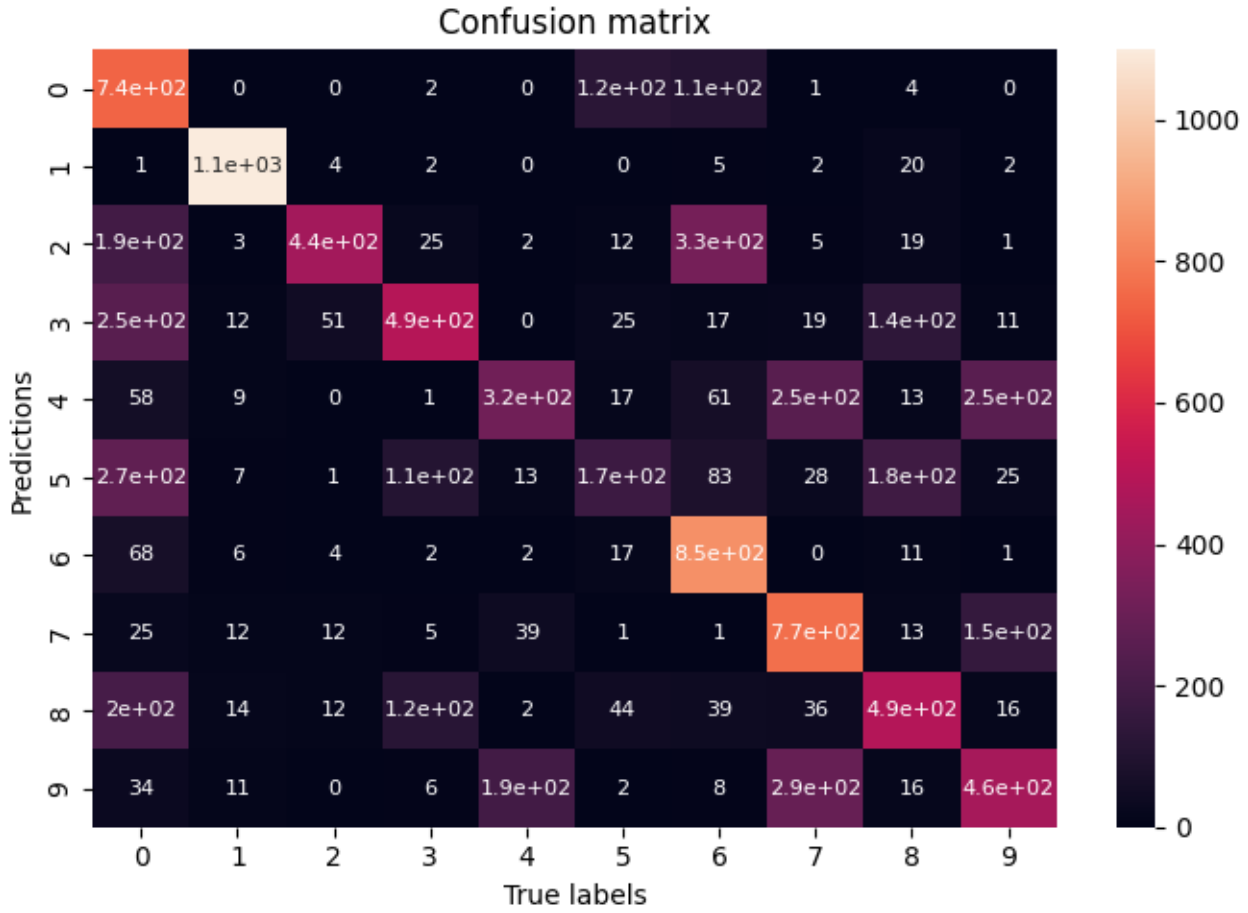
Epochs: 1

Learning rate: 0.1

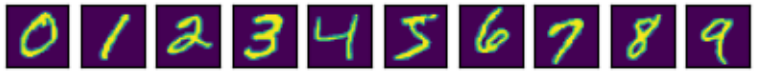
Order of the Gibbs sampling (k): 1

- - -

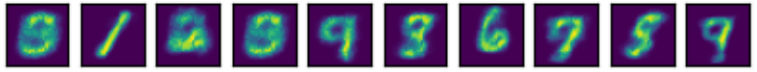
Accuracy: **59.01 %**



[Original images]



[Reconstructions]



Trained RBM and results

(2) **Hidden neurons**: **50**

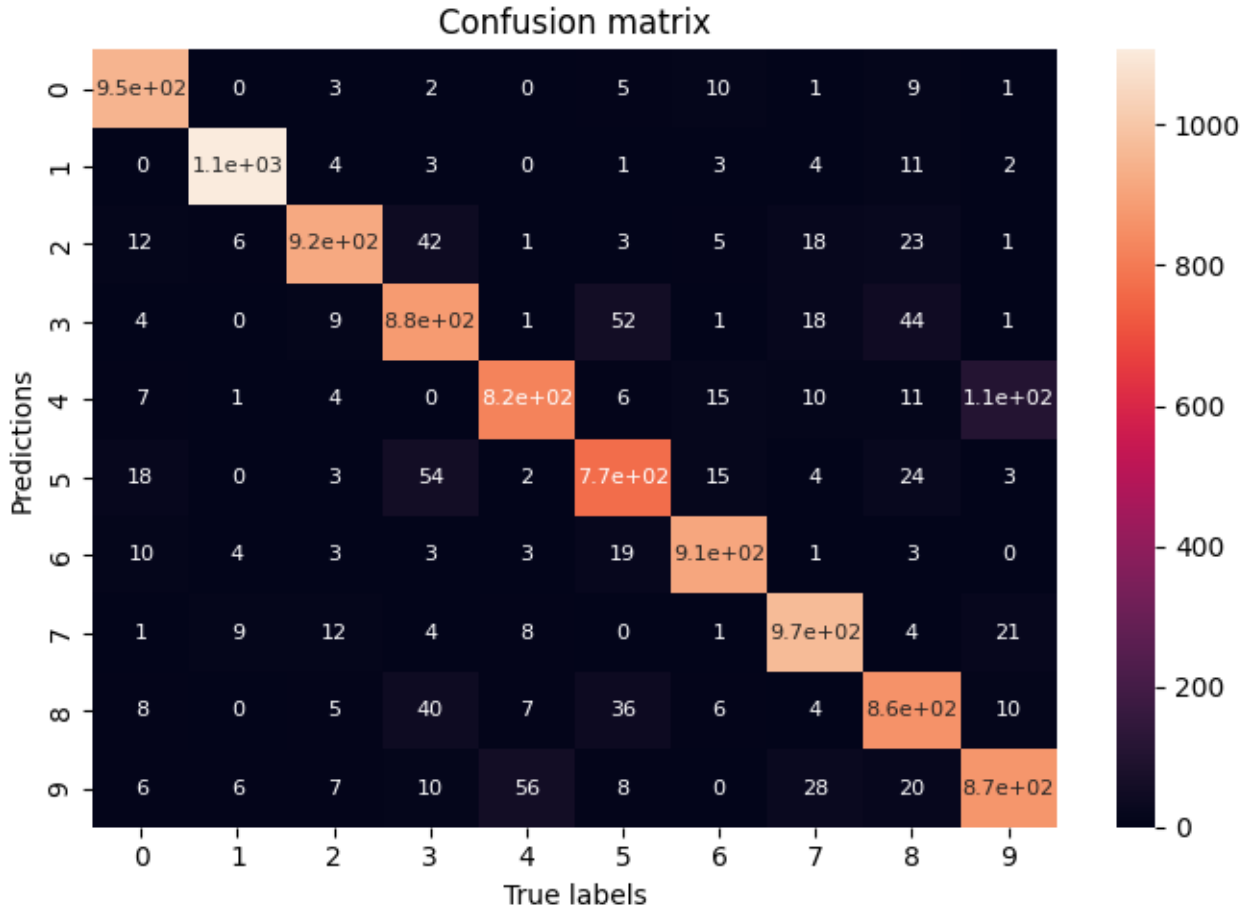
Epochs: 1

Learning rate: 0.1

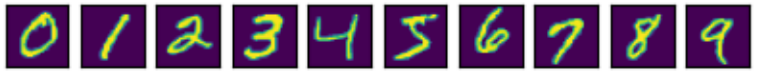
Order of the Gibbs sampling (k): 1

- - -

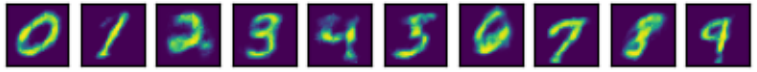
Accuracy: **90.49 %**



[Original images]



[Reconstructions]



Trained RBM and results

(3) **Hidden neurons**: **100**

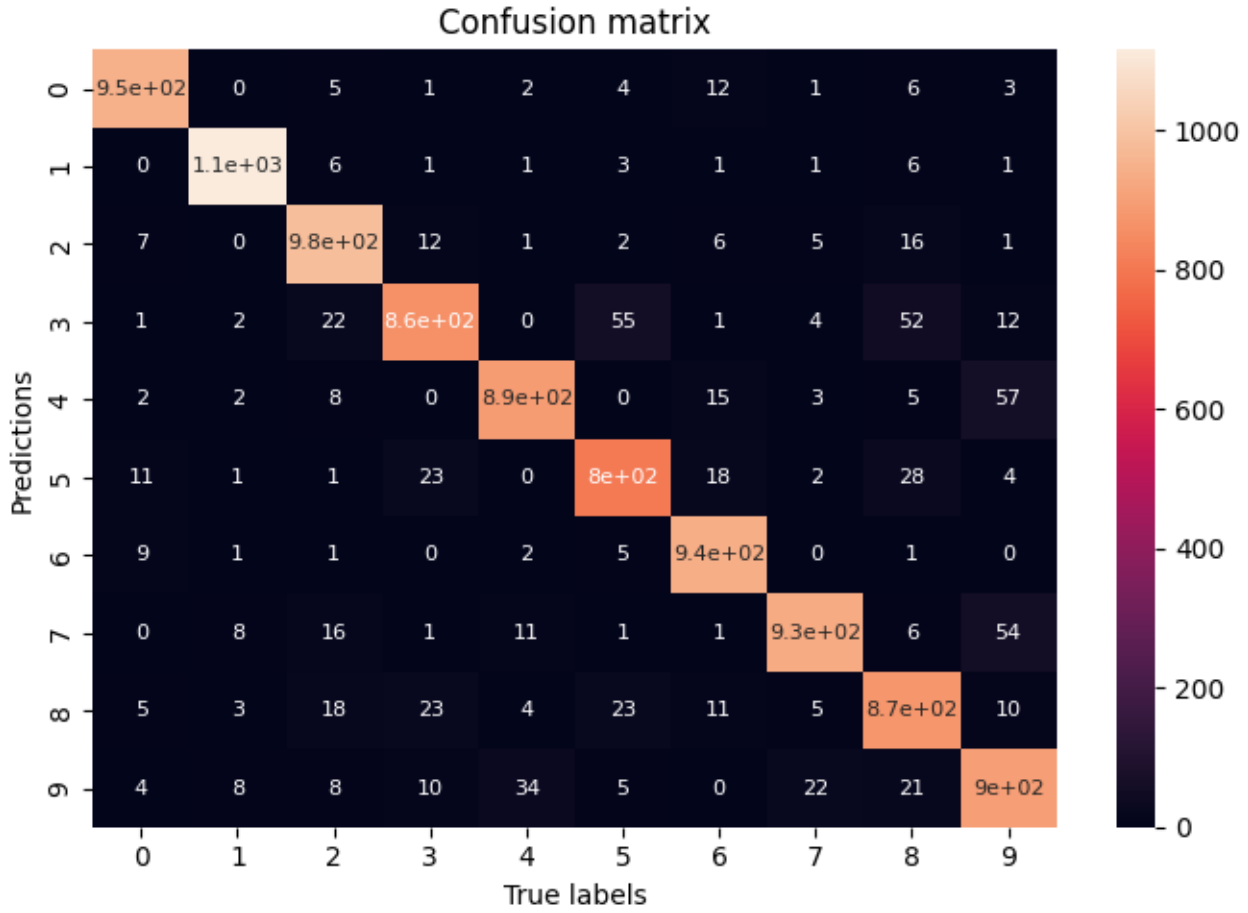
Epochs: 1

Learning rate: 0.1

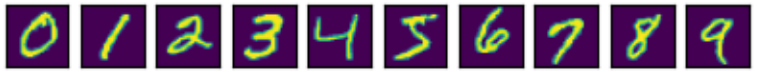
Order of the Gibbs sampling (k): 1

- - -

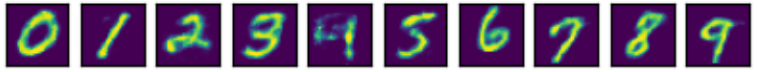
Accuracy: **92.29 %**



[Original images]



[Reconstructions]



Thanks for attention!

Alberto Marinelli