Sofia Lima

15-686 Neural Computation

Dr. Tai-Sing Lee

12 April, 2023

Problem Set 5: Deep Belief Net

Introduction A deep belief net is a stack of Restricted Boltzmann Machine (RBM). While ICA and LCA compute and represent information using firing rates of neurons, RBM computes using "spikes" with each spike indicating the presence of a particular proposition, with the spike count over time proportional to the probability of the proposition represented by the neuron.

```
initialize_rbm(n_visible, n_hidden, batch_size)
```

The probability of a hidden unit is defined by $p(h_j = 1|V) = sigmoid(b_j + \sum_i v_i w_{i,j})$ where h_j is independent of other hidden units, allowing us to compute the probability of all hidden units simultaneously.

`rbm_ph_given_v(rbm, V)` computes $p(h_j = 1|V = v)$ for all j for all data examples.

2a: **Computing the probability of visible units given the hidden units** (2 pts)

The first task is sampling. Compute the probability of the visible units being ON given the state of the hidden units (ON or OFF)

`rbm_pv_given_h(rbm, V)` → $p(v_i = 1|H = h) = sigmoid(c_i + \sum_j h_j w_{i,j})$

Hint: Note that forward computation is the sum of i and feedback computation is the sum over j. What is the implication of this?

Shown below are the MATLAB codes to solve this:

```
p_v = sigmoid(h*rbm.w' + repmat(rbm.vb, size(h,1), 1));
```

The implication for summing over the columns vs rows is in which axis the normalization is computed.

2b: **Synthesizing an image by sampling** (2 pts)

`rbm_reconstruct`→ $p(h_j|V = v)$

Reconstructing or synthesizing an image begins with sampling the hidden layer given the visible layer (or image). A sample of the visible layer is then computed given the sample of the hidden layer.

```
H_sample = sample(rbm_ph_given_v(rbm, V))
```

Shown below are the MATLAB codes to solve this:

```
H_sample = sample(rbm_ph_given_v(rbm, original_V));
recon = sample(rbm_pv_given_h(rbm, H_sample));
```

2c: **The RBM learning rule** (2 pts)

Learning an RBM consists of finding a set of parameters (W, B, C) such that the log probability of the model generating the data is maximized:

$$\underset{W,B,C}{\operatorname{argmax}} \log\big(P(V = v)\big)$$

Use gradient ascent to search for the maximum log probability.

$$\Delta w_{i,j} = E_{data}[v_i h_j] - E_{model^1}[v_i h_j]$$

$$\Delta c_i = E_{data}[v_i] - E_{model^1}[v_i]$$

$$\Delta b_j = E_{data}[h_j] - E_{model^1}[h_j]$$

E_{data} is the expectation given the visible units set by the data. E_{model^1} is the expectation computed from samples drawn from 1 round of alternately sampling the visible and hidden layers

```
train_rbm
```

Shown below are the MATLAB codes to solve this:

```
clamped_expectation =  (data_batch' * p_h_clamped)./rbm.batch_size;
model_expectation =  (p_v_model'*p_h_model)/rbm.batch_size
```

2d: **Visualizing the weights of a hidden unit** (2 pts)

Plot 'receptive field' of hidden units. Each hidden unit has a connection to a pixel in the data images.

```
rbm_visualize_weights
```

Shown below are the MATLAB codes to solve this:

```
receptive_fields = w(:,1:100);
receptive_fields = reshape(receptive_fields, dim1,dim2,100);
```

2e: **Learning hand written digits** (2 pt)

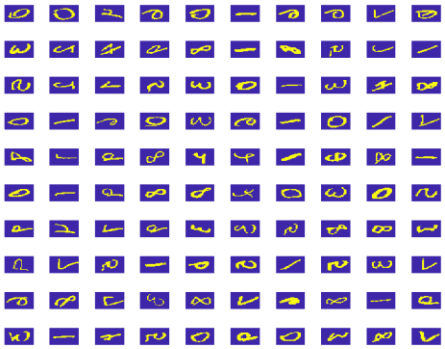
Use `data_1000` 1000 randomly selected examples from MNIST dataset in `data.mat` to train RBM.

```
rbm = initialize_rbm(784, 500, 100)
```

```
rbm_n = rbm_train(rbm, data_1000, n)
```

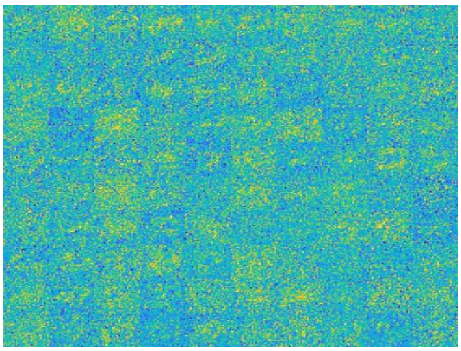
Train an RBM with $n = \{1,10,50,200\}$ iterations. For each, visualize the weights of the first 100 hidden units. Reconstruct with `data_100`. Do these match the test samples in the sense that they are the same digit?

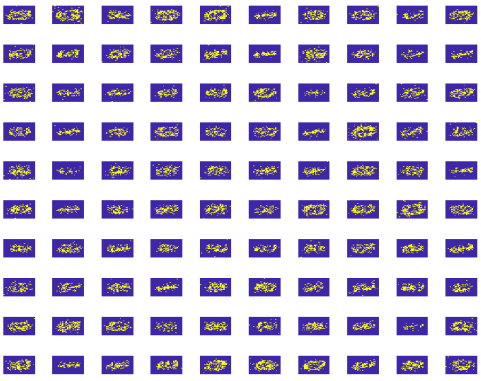
Shown below are the original images in the testing dataset stored in the data_100 matrix.



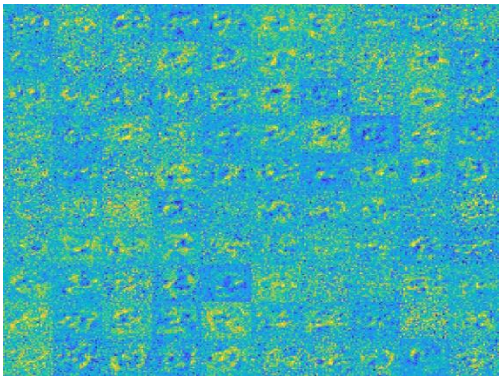
In the codes, we initialize an RBM with 784 visible units and 500 hidden units. Shown below are the results from training such an RBM with 1000 images for different number of epochs and testing with 100 test images. On the left are the first 100 hidden unit weights learned by sampling, whereas on the right are the reconstructed images from 100 test images in data_100 using the learned receptive fields (weights).

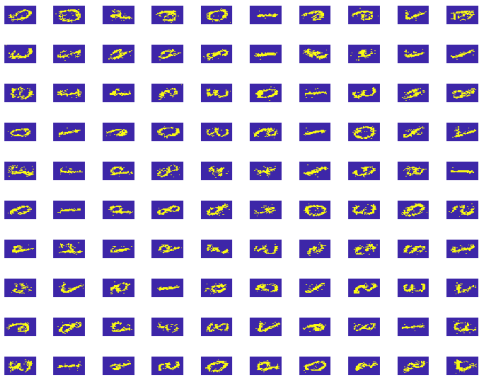
- Below is n=1



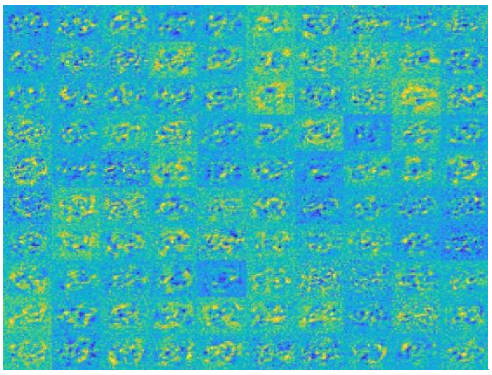


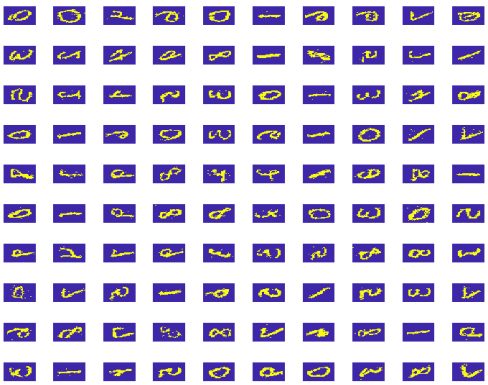
- Below is n=10



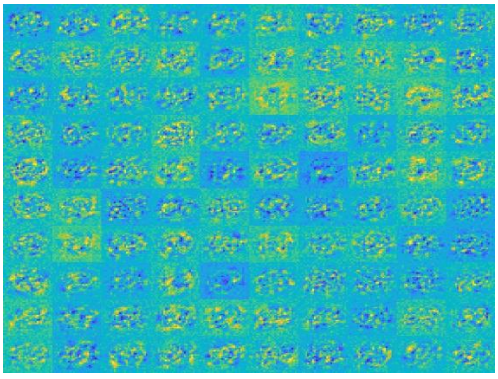


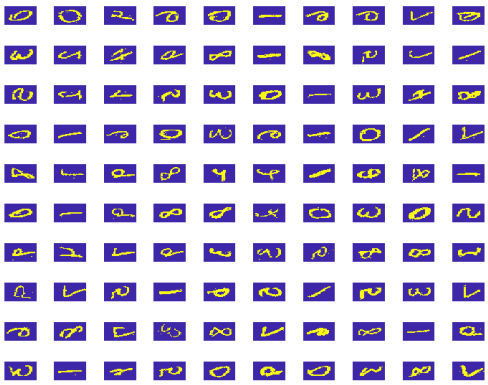
- Below is n=50





- Below is n=200





In this problem, we can see that with more epochs, the model is able to get closer to the correct weight values (i.e. receptive fields) in order to correctly reconstruct the image. With only 10 iterations, some of

the reconstructions resemble the original test samples in the sense that they would be categorized into the same digits. Notably, there are some receptive fields that do not have distinct patterns. With 50 epochs, there are more "mature" receptive fields. The results with 200 epochs are similar to 50 epochs, where the reconstructions resemble well the original test samples.

2f: **Adding sparsity constraint** (2 pt)

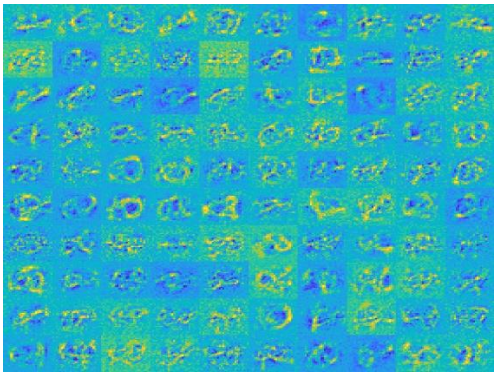
Initialize to have target sparsity `p`.

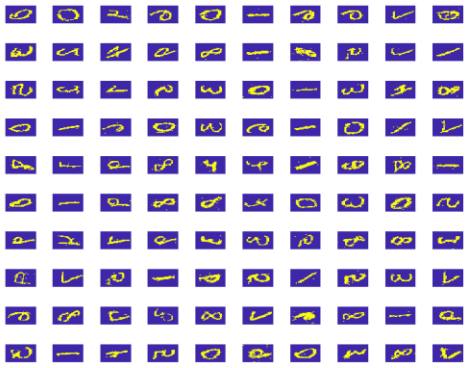
```
rbm = initialize_rbm(784, 500, 100, p)
```

Train an RBM with target sparsity of $p = 0.1$ for 200 iterations. Visualize the weights. Repeat with $p = 0.01$. Try lambda=0.1 or 1.0.

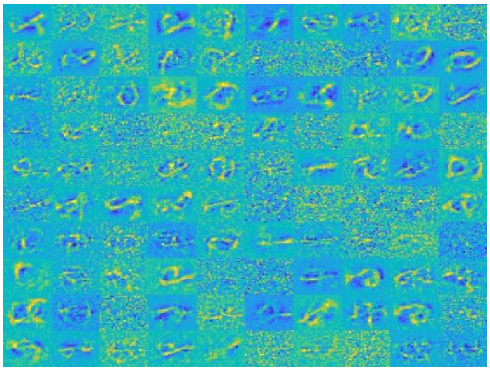
In the codes, we initialize an RBM with 784 visible units and 500 hidden units. We initialize with different sparsity conditions (e.g. decrease sparsity means fewer neurons firing for each input). Shown below are the results from training an RBM with 1000 images for 200 epochs and testing with 100 test images. On the left are the first 100 hidden unit weights learned by sampling, whereas on the right are the reconstructed images from 100 test images in data_100 using the learned receptive fields (weights).

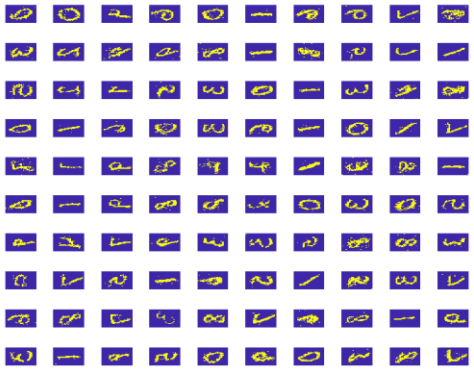
- p=0.1, λ=0.1



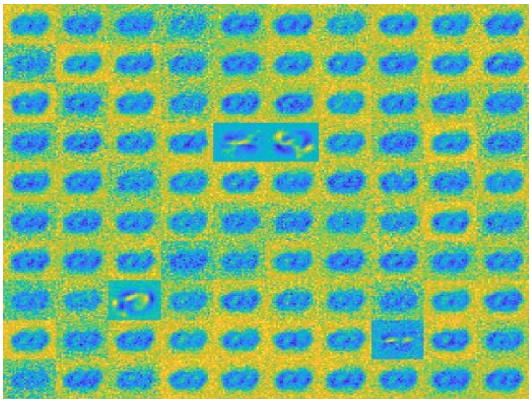


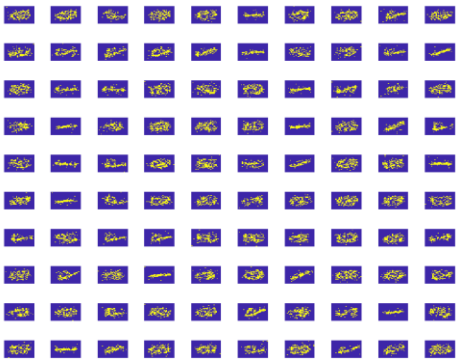
- p=0.01, λ=0.1



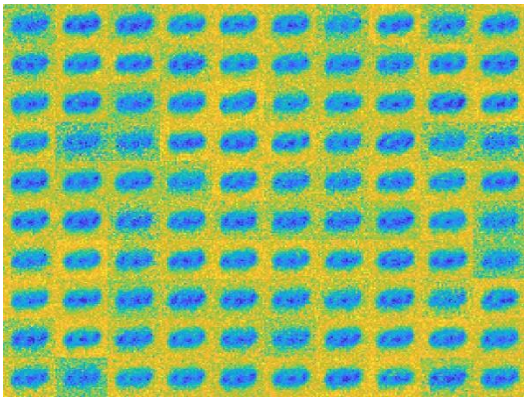


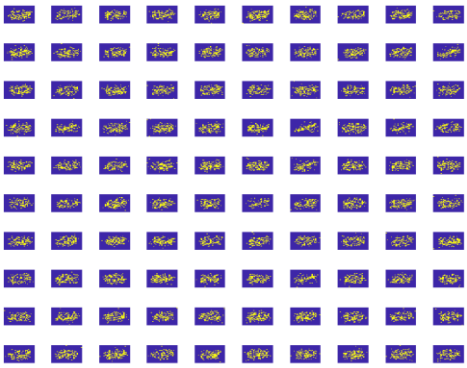
- p=0.1, λ=1.0





- p=0.01, λ=1.0





In this problem, we see a balance in the effects of sparsity. With the lambda parameter set to 0.1, the number of “mature” receptive fields increases with higher sparsity p value 0.1 compared to 0.01. With lambda parameter set to 1.0, we see that there is a point where sparsity is too extreme and the receptive fields become too “general”.