INTELLIGENT SYSTEMS FOR
PATTERN RECOGNITION
A.Y. 2020/2021

MIDTERM 2
ASSIGNMENT 3

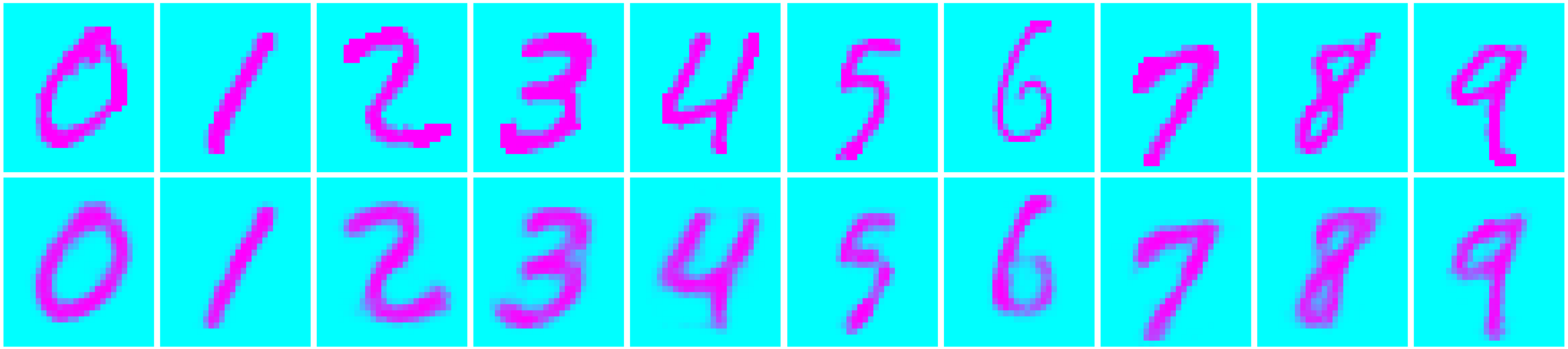
ELIA PICCOLI

621332

[HTTPS://GITHUB.COM/ELIAPICCOLI/ISPR-PROJECTS](https://github.com/eliapiccoli/ISPR-projects)

ORIGINAL

RBM RECONSTRUCTION



RBM - CODE

```python
class RBM:
    def __init__(self, num_visible, num_hidden, W=None, vb=None, hb=None, k=None):
        self.n_visible = num_visible
        self.n_hidden = num_hidden

        self.W = W if W is not None else np.random.uniform(-1, 1, (num_hidden, num_visible))
        self.vb = vb if vb is not None else np.zeros(num_visible)
        self.hb = hb if hb is not None else np.zeros(num_hidden)
        self.k = k if k is not None else 1

    def hidden_expectation(self, V):
        return sigmoid(self.hb + np.dot(V, self.W.T))

    def visible_expectation(self, H):
        return sigmoid(self.vb + np.dot(H, self.W))

    def foward(self, V):
        hp = self.hidden_expectation(V)
        hs = np.random.binomial(1, hp, size=hp.size)
        return hp, hs

    def backward(self, H):
        vp = self.visible_expectation(H)
        vs = np.random.binomial(1, vp, size=vp.size)
        return vp, vs

    def gibbs_sampling(self, V):
        vs = V
        for i in range(self.k):
            hp, hs = self.foward(vs)
            vp, vs = self.backward(hs)
        return hp, hs, vp, vs

    def reconstruct(self, V):
        hp, hs = self.foward(V)
        vp, vs = self.backward(hp)
        return vp, vs
```

$$p(h_j = 1 \mid \mathbf{v}) = \sigma(b_j + \sum_i v_i w_{ij})$$

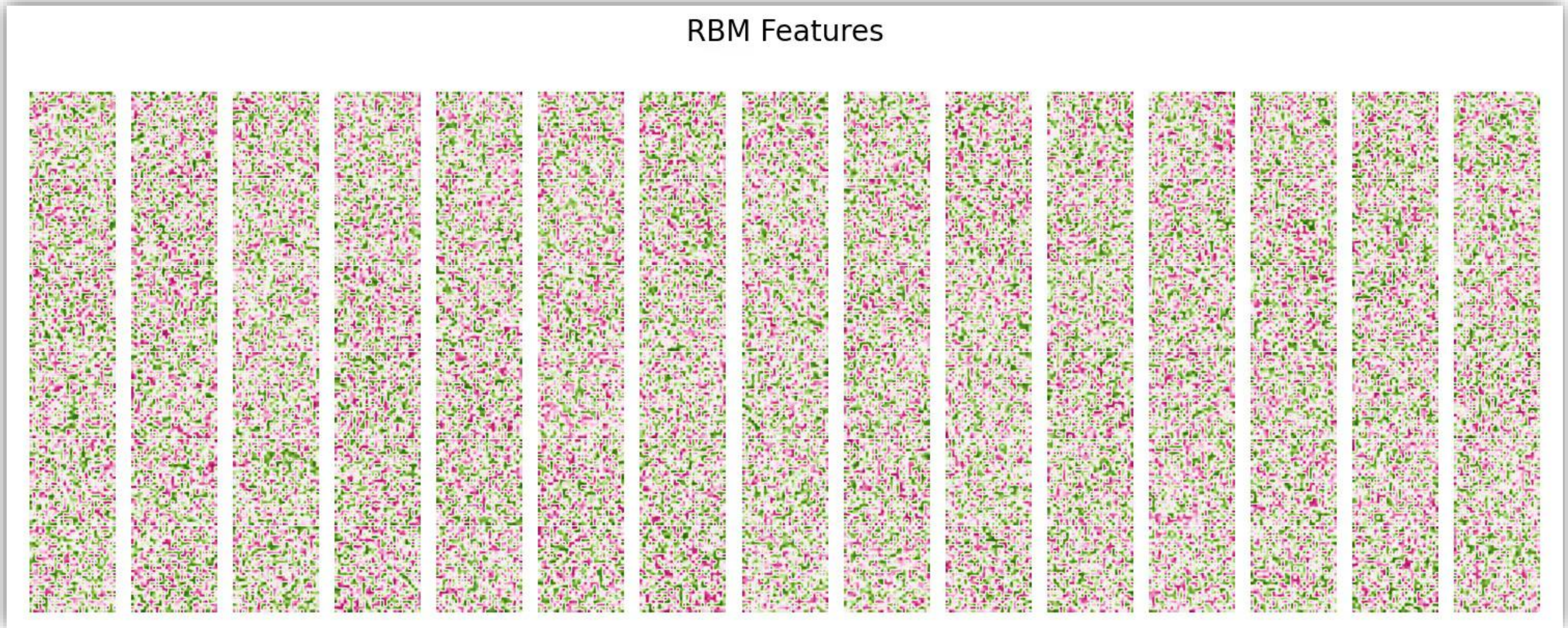
$$p(v_i = 1 \mid \mathbf{h}) = \sigma(a_i + \sum_j h_j w_{ij})$$

CONTRASTIVE DIVERGENCE - K

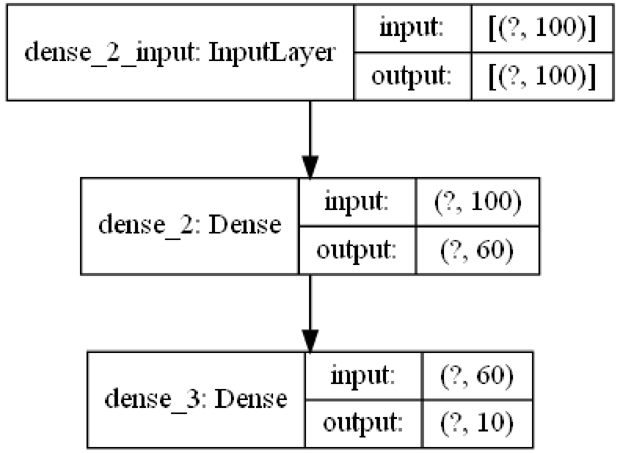
```python
def cd(self, X, epoch=1, batch_size=10, k=1, learning_rate=0.01, verbose=False):
    self.k = k
    n_sample, size = X.shape
    for e in range(epoch):
        for i in range(0, n_sample, batch_size):
            if verbose and i%5000==0:
                print(f"Epoch: {e} - batch: {i/batch_size}")
            j=i
            batch_W = np.empty((batch_size, self.n_hidden, self.n_visible))
            batch_hb = np.empty((batch_size, self.n_hidden))
            batch_vb = np.empty((batch_size, self.n_visible))
            while j < n_sample and j-i < batch_size:
                V = X[j]
                hp, hs, vp_r, vs_r = self.gibbs_sampling(V)
                hp_r, hs_r = self.foward(vs_r)
                E_data = np.outer(hp, V)
                E_model = np.outer(hp_r, vs_r)
                batch_W[j%batch_size] = E_data - E_model
                batch_hb[j%batch_size] = hp - hp_r
                batch_vb[j%batch_size] = V - vs_r
                j+=1
            # avg gradient over batch
            delta_W = np.mean(batch_W, axis=0)
            delta_hb = np.mean(batch_hb, axis=0)
            delta_vb = np.mean(batch_vb, axis=0)

            self.W += learning_rate*(delta_W)
            self.hb += learning_rate*(delta_hb)
            self.vb += learning_rate*(delta_vb)
```

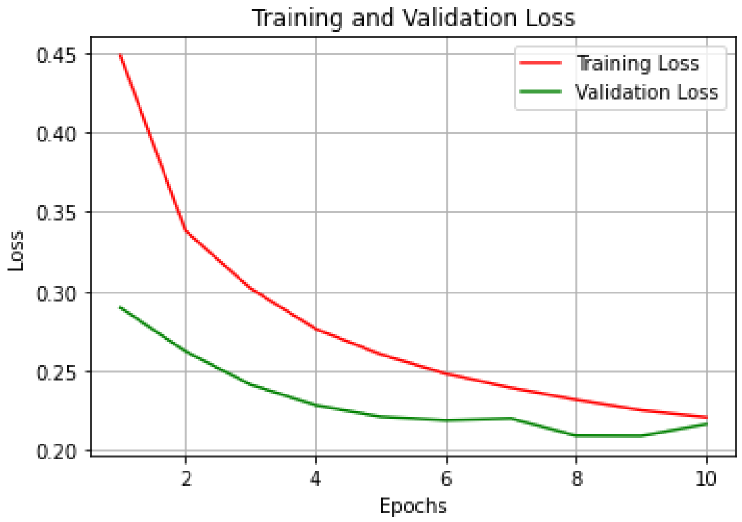
Average per-case gradient computed on a mini-batch

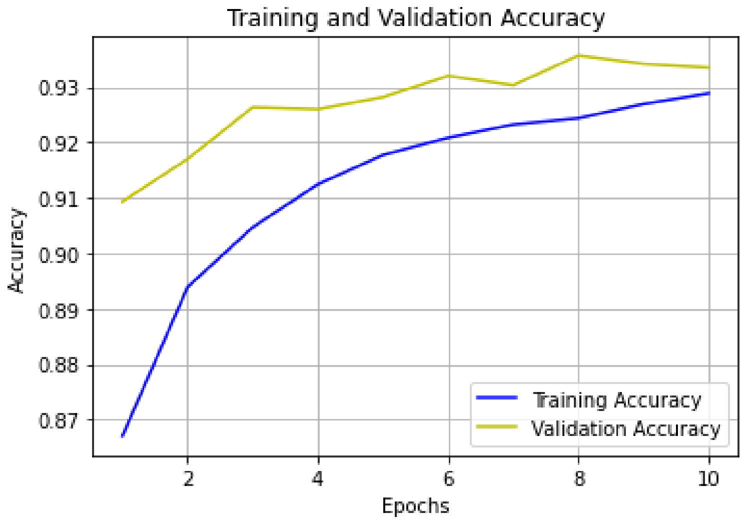
RBM - HIDDEN UNITS EVOLUTION



RBM - CLASSIFIER



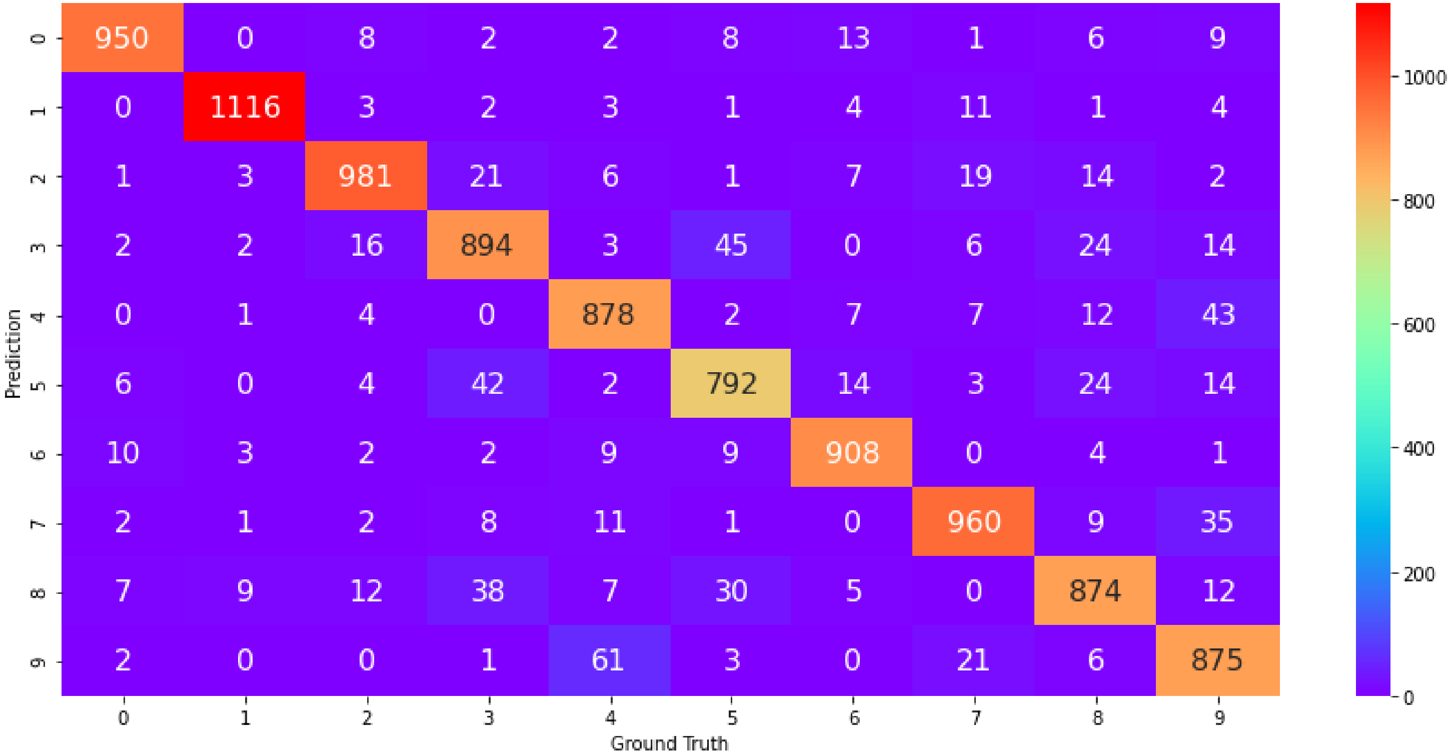




TEST ACCURACY: 92.31% ± 0.16

[OVER 20 TEST]

CONFUSION MATRIX

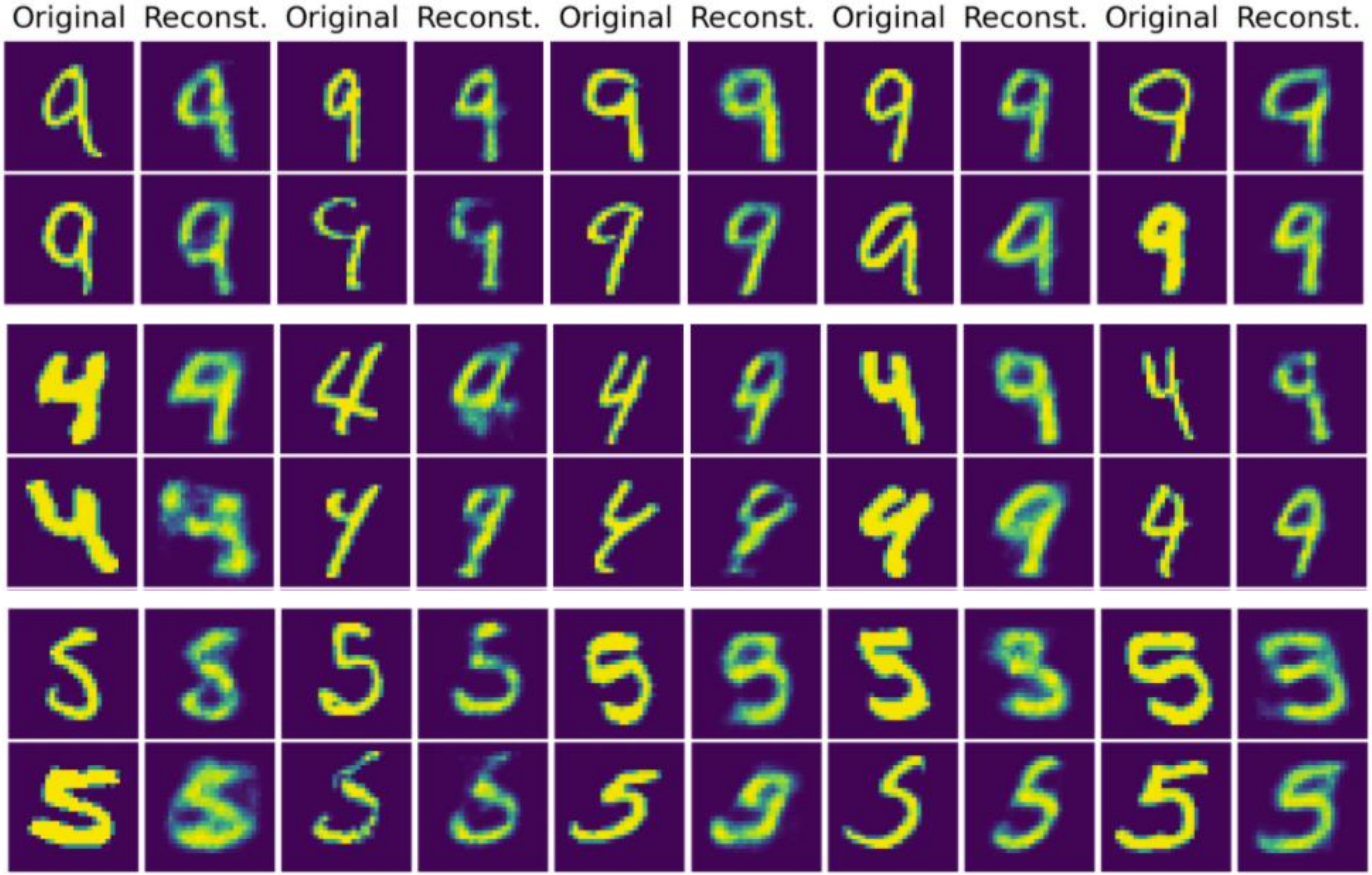


RBM - CLASSIFIER RESULT ANALYSIS

PREDICTION: 4
LABEL: 9

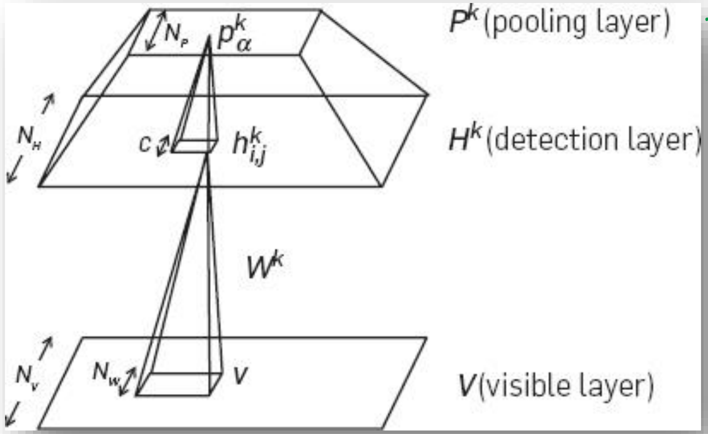
PREDICTION: 9
LABEL: 4

PREDICTION: 3
LABEL: 5



CRBM - CONVOLUTIONAL RBM WITH PROBABILISTIC MAX-POOLING [1]

ARCHITECTURE




```python
class CRBM:
    def visible_expectation(self, H):
        x = sum(ss.convolve(self.W[k], H[k]) for k in range(self.num_filters))
        x += self.vb
        return sigmoid(x)

    def hidden_expectation(self, V):
        x = np.exp(
            np.array([
                ss.convolve(self.W[k, ::-1, ::-1], V, 'valid') + self.hb[k]
                for k in range(self.num_filters)
            ])
        )
        return x / (1. + self.pooling_group_weight(x))

    def pooling_expectation(self, V):
        x = np.exp(
            np.array([
                ss.convolve(self.W[k, ::-1, ::-1], V, 'valid') + self.hb[k]
                for k in range(self.num_filters)
            ])
        )
        return 1 - 1. / (1 + self.pool(x))
```

$$P(v_{ij}=1|\mathbf{h}) = \sigma\left(\left(\sum_k W^k * h^k\right)_{ij} + c\right)$$

$$P(h_{i,j}^k = 1|\mathbf{v}) = \frac{\exp(I(h_{i,j}^k))}{1+\sum_{(i',j')\in B_\alpha}\exp(I(h_{i',j'}^k))}$$

WHERE $I(h_{ij}^k) \triangleq b_k + (\tilde{W}^k * v)_{ij}$

$$P(p_\alpha^k = 0|\mathbf{v}) = \frac{1}{1+\sum_{(i',j')\in B_\alpha}\exp(I(h_{i',j'}^k))}$$

Algorithm 1 A training algorithm for the convolutional RBM

repeat {over the training data (e.g., a set of training images)}

Set $V^{(0)} := V$ (e.g., set the current image as a mini-batch)

Compute the posterior $Q^{(0)} \triangleq P(H|V^{(0)})$ (Equations 14 and 15).

Sample $H^{(0)}$ from $Q^{(0)}$.

for $n = 1$ to N_{cd} **do**

Sample V^n from $P(V|H^{(n-1)})$ (Equation 10 or 11).[c]

Compute the posterior $Q^{(n)} \triangleq P(H|V^n)$ (Equations 14 and 15).

Sample $H^{(n)}$ from $Q^{(n)}$.

end for

Update weights and biases with contrastive divergence and sparsity regularization:

$$\Delta W^k \propto \frac{1}{N_H^2}(\tilde{Q}^{(0),k} * V^{(0)} - \tilde{Q}^{(n),k} * V^{(n)}) \quad (17)$$

$$\Delta b_k \propto \frac{1}{N_H^2}\sum_{ij}(Q_{ij}^{(0),k} - Q_{ij}^{(n),k}) + \Delta b_k^{sparsity} \quad (18)$$

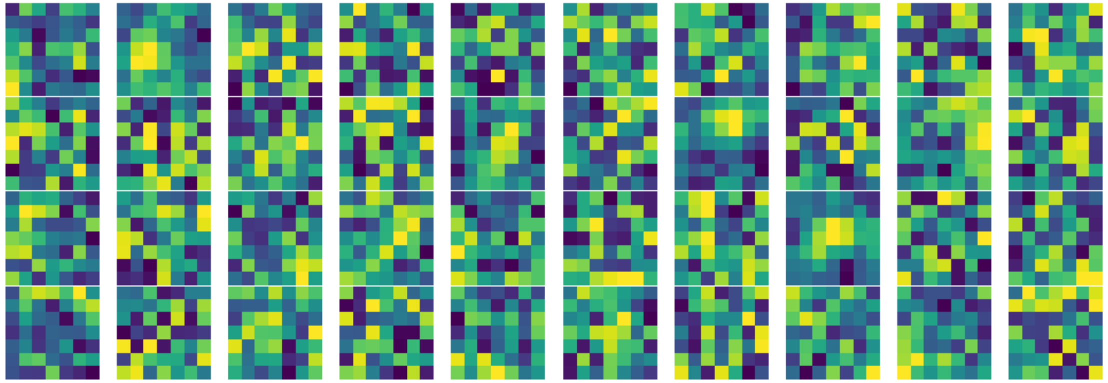
$$\Delta c \propto \frac{1}{N_V^2}\sum_{ij}(V_{ij}^{(0)} - V_{ij}^{(n)}) \quad (19)$$

until convergence

[1] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. 2011. Unsupervised learning of hierarchical representations with convolutional deep belief networks. Commun. ACM 54, 10 (October 2011), 95-103. DOI:https://doi.org/10.1145/2001269.2001295

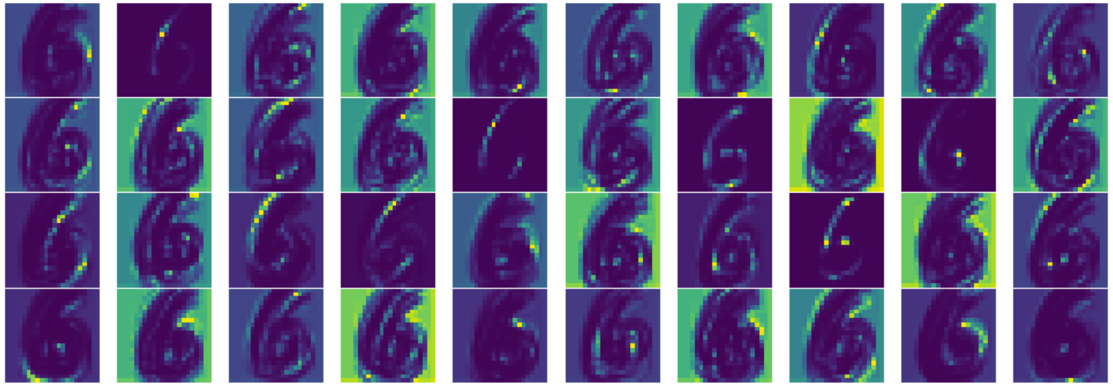
CRBM - CONVOLUTIONAL RBM RESULTS

FILTERS

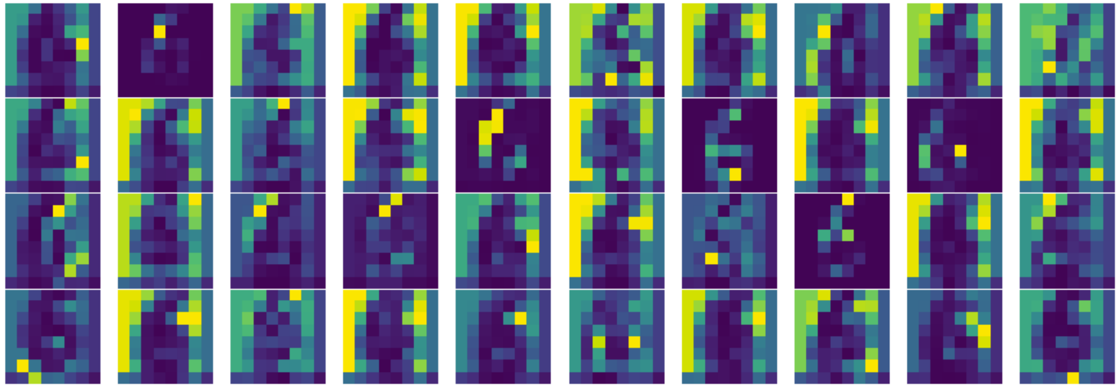


APPLY TO A 6 IMAGE

HIDDEN UNITS



POOLING UNITS



WHAT IF WE BUILD A CLASSIFIER OVER THE POOLING UNITS?

IT OUT-PERFORMS THE ONE BUILT OVER RBM'S HIDDEN UNITS WITH AN ACCURACY OF 98.4%!