

# Restricted Boltzmann Machine

Eric Blohm

October 7, 2024

Here is the Python code for the Restricted Boltzmann Machine:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 def sample_patterns_equal_prob(batch_size, patterns):
6     pattern_indices = np.random.choice(len(
7         patterns), size=batch_size, p=[0.25, 0.25,
8         0.25, 0.25])
9     mini_batch = np.array([patterns[idx] for idx
10         in pattern_indices])
11     return mini_batch
12
13
14 def p_boltzmann(b):
15     denominator = 1+np.exp(-2*b)
16     return 1/denominator
17
18
19 def compute_delta_w(eta, b_i_h_0, v_0, b_i_h, v, w):
20     delta_w = w.copy()
21     for m in range(0, len(w)):
22         for n in range(0, len(w[0])):
23             term1 = np.tanh(b_i_h_0[m])*v_0[n]
24             term2 = np.tanh(b_i_h[m])*v[n]
25             delta_w[m][n] = eta*(term1 - term2)
26     return delta_w
27
28
29 def compute_delta_theta_v(eta, v_0, v, theta_v):
30     delta_theta_v = np.zeros(len(theta_v))
31     for n in range(0, len(theta_v)):
32         delta_theta_v[n] = -eta*(v_0[n]-v[n])
33     return delta_theta_v
```

```

31
32
33 def compute_delta_theta_h(eta, b_i_h_0, b_i_h, theta_h
    ):
34     delta_theta_h = np.zeros(len(theta_h))
35     for m in range(0, len(theta_h)):
36         term1 = np.tanh(b_i_h_0[m])
37         term2 = np.tanh(b_i_h[m])
38         delta_theta_h[m] = -eta*(term1-term2)
39     return delta_theta_h
40
41
42
43 def train_RBM(M, eta, k, epochs, batch_size): #, patterns)
44
45     #PD = np.array([0.25, 0, 0, 0.25, 0, 0.25, 0.25,
46                     0])
47
48     ### Init ###
49     #patterns that have a 1/4 probability to be
50     #sampled, used for training
51     patterns = np.array([[ -1, -1, -1],
52                         [ -1, 1, 1],
53                         [ 1, -1, 1],
54                         [ 1, 1, -1]])
55
56     variance = 1/ np.maximum(3, M)
57     std = np.sqrt(variance)
58     w = np.random.normal(0, std, size=(M, 3))
59
60     theta_v = np.zeros(3)
61     theta_h = np.zeros(M)
62     v = np.zeros(3)
63     h = np.zeros(M)
64     #####
65
66     energies = []
67     all_samples = []
68
69     for epoch in range(0, epochs):
70
71         ### Init weights and thresholds ###
72         delta_w = np.zeros((M, 3))
73         delta_theta_v = np.zeros(3)
74         delta_theta_h = np.zeros(M)
75         #####

```

```

74
75     ### Sample from patterns with equal prob ###
76     mini_batch = sample_patterns_equal_prob(
77         batch_size, patterns)
78     #####
79     for sample in mini_batch:
80         all_samples.append(sample)
81
82     for mu, pattern in enumerate(mini_batch):
83         v = pattern.copy()
84         v_0 = v.copy()
85         b_i_h_0 = np.zeros(M)
86
87         ### Calculate  $b_i^h(0)$  and update all
88             hidden neurons  $h_i(0)$  ###
89         for i in range(0, len(b_i_h_0)):
90             b_i_h_0[i] = np.dot(w[i], v_0) - theta_h[
91                 i]
92             r = np.random.rand()
93             p_B = p_boltzmann(b_i_h_0[i])
94             if(r < p_B):
95                 h[i] = 1
96             else:
97                 h[i] = -1
98             #
99             #####
100
101         b_i_h = b_i_h_0.copy()
102         b_j_v = np.zeros(3)
103         for step in range(0, k):
104             ### update visible neurons ###
105             for j in range(0, 3):
106                 b_j_v[j] = np.dot(h, w[:, j]) -
107                     theta_v[j]
108                 p_B_v = p_boltzmann(b_j_v[j])
109                 r = np.random.rand()
110                 if(r < p_B_v):
111                     v[j] = 1
112                 else:
113                     v[j] = -1
114             #####
115
116         ### update hidden neurons ###
117         for i in range(0, M):
118             b_i_h[i] = np.dot(w[i], v) - theta_h[

```

```

114         i]
115         p_B_h = p_boltzmann(b_i_h[i])
116         r = np.random.rand()
117         if(r < p_B_h):
118             h[i] = 1
119         else:
120             h[i] = -1
121         #####
122         ### calculate delta_w ###
123         for m in range(0,M):
124             for n in range(0,3):
125                 delta_w[m][n] += eta* ( np.tanh(
126                     b_i_h_0[m])*v_0[n] - np.tanh(
127                     b_i_h[m])*v[n] )
128             #####
129             ### calculate delta_theta_v ###
130             for n in range(0,3):
131                 delta_theta_v[n] -= eta*(v_0[n]-v[n])
132             #####
133             ## calculate delta_theta_h ###
134             for m in range(0, M):
135                 delta_theta_h[m] -= eta*(np.tanh(
136                     b_i_h_0[m])-np.tanh(b_i_h[m]))
137             #####
138             ### update values ###
139             w += delta_w
140             theta_h += delta_theta_h
141             theta_v += delta_theta_v
142             #####
143             ### Monitor energy function ###
144             H = compute_energy_function(w, h, v , theta_v,
145                 theta_h)
146             energies.append(H)
147
148     """
149     ### Monitor energy function ###
150     epoch_range = np.arange(epochs)
151     plt.plot(epoch_range,energies,marker='o', color='
152         green', markersize=1, linestyle='-')
153     plt.title(f'Energy of training, M={M}')
154     plt.xlabel('Epochs')
155     plt.ylabel('Energy')

```

```

154     plt.grid()
155     plt.tight_layout()
156     plt.show()
157     """
158     return w, theta_h, theta_v
159
160 def D_kl_bound(M):
161     #number of inputs
162     N=3
163     expression = 3 - int(np.log2(M+1)) - ((M+1)/(2**
164         int(np.log2(M+1))))
165     if M < (2**(N-1) - 1):
166         return np.log(2)*expression
167
168     elif M >= (2**(N-1) - 1):
169         return np.log(2)*0
170
171 def D_kl(PD,PB):
172     sum= 0
173     for mu in range(0,len(PD)):
174         if(PB[mu] > 0 and PD[mu]>0):
175             #print("PB^mu: ", PB[mu], ", ", "PD^mu: ",
176                 PD[mu])
177             sum += PD[mu] * np.log(PD[mu]/PB[mu])
178     return sum
179
180 def compute_energy_function(w, h, v , theta_v, theta_h
181 ):
182     term1_sum1 = 0
183     for i in range(0,len(h)):
184         term1_sum2 = 0
185         for j in range(0,len(v)):
186             term1_sum2 += w[i][j]*h[i]*v[j]
187         term1_sum1 += term1_sum2
188
189     term2_sum = 0
190     for j in range(0,len(theta_v)):
191         term2_sum += theta_v[j]*v[j]
192
193     term3_sum = 0
194     for i in range(0,len(theta_h)):
195         term3_sum += theta_h[i]*h[i]
196
197     return -term1_sum1 + term2_sum + term3_sum

```

```

197
198 def main():
199
200     ### Init ###
201     # pattern = 0.25 for index 0,3,5,6, used when
        sampling
202     all_patterns = np.array([[ -1, -1, -1],
203                               [ -1, -1,  1],
204                               [ -1,  1, -1],
205                               [ -1,  1,  1],
206                               [  1, -1, -1],
207                               [  1, -1,  1],
208                               [  1,  1, -1],
209                               [  1,  1,  1]])
210
211     M_values = [1, 2, 4, 8]
212     d_kl_bound_values = []
213     d_kl_values = []
214     eta = 0.004
215     k = 10
216     batch_size = 100
217     epochs = 1500
218
219     # sample using the dynamincs in the CD_k algorithm
220     #
221     num_iterations = 10000
222     max_T = 10
223     print(f"Sampling: num_iterations={num_iterations},
224           T={max_T}")
225     #####
226
227     for M in M_values:
228         print("-----")
229         print(f"Training configuration: M={M} | eta={
230               eta} | k={k} | batch_size={batch_size} |
231               epochs={epochs}")
232
233         w, theta_h, theta_v = train_RBM(M,eta,k,epochs
234                                         ,batch_size)
235         print(f"\n w={w} | theta_h={theta_h} |
236               theta_v={theta_v}")
237
238         PD = np.array([0.25, 0, 0, 0.25, 0, 0.25,
239                        0.25, 0])
240         PB = np.zeros(8)
241
242         h = np.zeros(M)

```

```

235
236     for step in range(0,num_iterations):
237         v = all_patterns[np.random.randint(
238             all_patterns.shape[0])].copy()
239         b_j_v = np.zeros(3)
240         b_i_h = np.zeros(M)
241
242         for T in range(0,max_T):
243             ### update hidden neurons ###
244             for i in range(0,M):
245                 b_i_h[i] = np.dot(w[i],v)-theta_h[
246                     i]
247                 p_B_h = p_boltzmann(b_i_h[i])
248                 r = np.random.rand()
249                 if(r < p_B_h):
250                     h[i] = 1
251                 else:
252                     h[i] = -1
253             #####
254             ### update visible neurons ###
255             for j in range(0,3):
256                 b_j_v[j] = np.dot(h,w[:,j])-
257                     theta_v[j]
258                 p_B_v = p_boltzmann(b_j_v[j])
259                 r = np.random.rand()
260                 if(r < p_B_v):
261                     v[j] = 1
262                 else:
263                     v[j] = -1
264             #####
265             for idx in range(0,len(PB)):
266                 if(np.array_equal(v,all_patterns[idx])
267                     ):
268                     PB[idx]+=1
269
270             if(step % 10 == 0 and step >0):
271                 tmp = PB/step#compute_frequencies(
272                     samples)
273                 print(f"runtime PD: {tmp}, iteration:
274                     {step}")
275
276     print(f"Results:\nM={M}")
277     PB= PB/num_iterations

```

```

275     print("PB: ", PB, "sum =", np.sum(PB))
276     print("PD: ", PD)
277
278     d_kl_bound = D_kl_bound(M)
279     d_kl_bound_values.append(d_kl_bound)
280
281     d_kl = D_kl(PD,PB)
282     d_kl_values.append(d_kl)
283     print(f"D_KL: {d_kl}, D_KL_bound: {d_kl_bound}
284           ")
285     print("-----")
286
287     print("D_KL: " ,d_kl_values," ", "D_KL_bound: ",
288           d_kl_bound_values)
289
290     plt.plot(M_values,d_kl_bound_values,marker='o',
291             color='green', markersize=4, linestyle='--',
292             label='Bound')
293     plt.plot(M_values,d_kl_values,marker='o', color='
294             orange', markersize=4, linestyle='--',label='
295             True value')
296     plt.title('Kullback-Leibler divergence bound vs
297             true value')
298     plt.xlabel('Number of hidden neurons (M)')
299     plt.ylabel('D_KL')
300     plt.grid()
301     plt.legend()
302     plt.tight_layout()
303     plt.show()
304
305 if __name__ == "__main__":
306     main()

```