```
1 import numpy as np
 2
  1111111
 3
4 This file implements various first-order update rules that are commonly used
 5 training neural networks. Each update rule accepts current weights and the
6 gradient of the loss with respect to those weights and produces the next set
7
  weights. Each update rule has the same interface:
8
9 def update(w, dw, config=None):
10
11 Inputs:
12
    - w: A numpy array giving the current weights.
13
     - dw: A numpy array of the same shape as w giving the gradient of the
14
       loss with respect to w.
15
    - config: A dictionary containing hyperparameter values such as learning
   rate,
       momentum, etc. If the update rule requires caching values over many
16
17
       iterations, then config will also hold these cached values.
18
19 Returns:
    - next_w: The next point after the update.
20
21
    - config: The config dictionary to be passed to the next iteration of the
22
       update rule.
23
24 NOTE: For most update rules, the default learning rate will probably not
25 well; however the default values of the other hyperparameters should work
  well
26 for a variety of different problems.
27
28 For efficiency, update rules may perform in-place updates, mutating w and
29 setting next w equal to w.
30 """
31
32
33 def sqd(w, dw, config=None):
34
35
     Performs vanilla stochastic gradient descent.
36
37
     config format:
38

    learning rate: Scalar learning rate.

39
40
     if config is None: config = {}
41
     config.setdefault('learning_rate', 1e-2)
42
43
    w -= config['learning_rate'] * dw
44
     return w, config
45
46
47 def sgd_momentum(w, dw, config=None):
48
49
     Performs stochastic gradient descent with momentum.
50
51
     config format:
52
    - learning_rate: Scalar learning rate.
53
     - momentum: Scalar between 0 and 1 giving the momentum value.
54
       Setting momentum = 0 reduces to sgd.
```

localhost:4649/?mode=python 1/5

```
- velocity: A numpy array of the same shape as w and dw used to store a
   moving
56
      average of the gradients.
57
58
    if config is None: config = {}
    config.setdefault('learning_rate', 1e-2)
59
     config.setdefault('momentum', 0.9) # set momentum to 0.9 if it wasn't there
60
     v = config.get('velocity', np.zeros_like(w)) # gets velocity, else sets
61
   it to zero.
62
63
    # ============= #
64
    # YOUR CODE HERE:
      Implement the momentum update formula. Return the updated weights
65
        as next w, and the updated velocity as v.
66
    67
68
69
    alpha = config['momentum']
70
    eps = config['learning_rate']
71
72
    v = alpha * v - eps * dw
73
    w += v
74
75
    next_w = w
76
77
    78
    # END YOUR CODE HERE
79
80
81
    config['velocity'] = v
82
83
     return next_w, config
84
85 def sgd_nesterov_momentum(w, dw, config=None):
86
87
    Performs stochastic gradient descent with Nesterov momentum.
88
89
    config format:
90
    learning_rate: Scalar learning rate.
91
    - momentum: Scalar between 0 and 1 giving the momentum value.
92
      Setting momentum = 0 reduces to sgd.
    - velocity: A numpy array of the same shape as w and dw used to store a
93
   moving
94
      average of the gradients.
95
96
     if config is None: config = {}
    config.setdefault('learning_rate', 1e-2)
97
     config.setdefault('momentum', 0.9) # set momentum to 0.9 if it wasn't there
98
     v = config.get('velocity', np.zeros_like(w)) # gets velocity, else sets
99
   it to zero.
100
101
    # YOUR CODE HERE:
102
103
        Implement the momentum update formula. Return the updated weights
104
        as next_w, and the updated velocity as v.
105
    # =========== #
106
107
    alpha = config['momentum']
    eps = config['learning_rate']
108
109
110
    v old = v
```

localhost:4649/?mode=python 2/5

```
2022/2/23 清晨7:44
                                        optim.py
111
     v = alpha * v - eps * dw
112
     w += (v + alpha * (v - v old))
113
114
     next_w = w
115
116
     # ============= #
117
     # END YOUR CODE HERE
118
     # ============= #
119
120
     config['velocity'] = v
121
122
     return next_w, config
123
124 def rmsprop(w, dw, config=None):
125
126
     Uses the RMSProp update rule, which uses a moving average of squared
     values to set adaptive per-parameter learning rates.
127
128
129
     config format:
130
     learning_rate: Scalar learning rate.
131
     - decay_rate: Scalar between 0 and 1 giving the decay rate for the squared
132
       gradient cache.
133
     - epsilon: Small scalar used for smoothing to avoid dividing by zero.
134

    beta: Moving average of second moments of gradients.

135
136
     if config is None: config = {}
     config.setdefault('learning_rate', 1e-2)
137
138
     config.setdefault('decay_rate', 0.99)
139
     config.setdefault('epsilon', 1e-8)
     config.setdefault('a', np.zeros_like(w))
140
141
142
     next_w = None
143
     144
145
     # YOUR CODE HERE:
     # Implement RMSProp. Store the next value of w as next_w. You need
146
147
         to also store in config['a'] the moving average of the second
148
         moment gradients, so they can be used for future gradients. Concretely,
         config['a'] corresponds to "a" in the lecture notes.
149
150
151
     a = config['a']
152
153
     beta = config['decay_rate']
     eps = config['learning rate']
154
155
     nu = config['epsilon']
156
157
     a = beta * a + (1 - beta) * dw * dw
158
     w = (eps * dw) / (np.sqrt(a) + nu)
159
160
     config['a'] = a
161
     next_w = w
162
163
     # ============ #
164
     # END YOUR CODE HERE
165
     166
167
      return next_w, config
168
169
```

localhost:4649/?mode=python 3/5

```
170 def adam(w, dw, config=None):
171
172
     Uses the Adam update rule, which incorporates moving averages of both the
173
     gradient and its square and a bias correction term.
174
175
     config format:
176
     - learning_rate: Scalar learning rate.
     - beta1: Decay rate for moving average of first moment of gradient.
177
178
     - beta2: Decay rate for moving average of second moment of gradient.
179
     - epsilon: Small scalar used for smoothing to avoid dividing by zero.
180
     - m: Moving average of gradient.
181
     - v: Moving average of squared gradient.
182
     - t: Iteration number.
183
184
     if config is None: config = {}
     config.setdefault('learning rate', 1e-3)
185
186
     config.setdefault('beta1', 0.9)
187
     config.setdefault('beta2', 0.999)
     config.setdefault('epsilon', 1e-8)
188
     config.setdefault('v', np.zeros_like(w))
189
     config.setdefault('a', np.zeros_like(w))
config.setdefault('t', 0)
190
191
192
193
     next w = None
194
195
196
     # YOUR CODE HERE:
         Implement Adam. Store the next value of w as next_w. You need
197
         to also store in config['a'] the moving average of the second
198
199
         moment gradients, and in config['v'] the moving average of the
         first moments. Finally, store in config['t'] the increasing time.
200
201
     202
203
     t = config['t']
204
     v = config['v']
205
     a = config['a']
     eps = config['learning_rate']
206
207
     nu = config['epsilon']
208
     beta1 = config['beta1']
209
     beta2 = config['beta2']
210
211
     t += 1
212
     v = beta1 * v + (1 - beta1) * dw
213
     a = beta2 * a + (1 - beta2) * dw * dw
     v u = v / (1 - beta1**t)
214
     a u = a / (1 - beta2**t)
215
216
     w = (eps * v_u) / (np.sqrt(a_u) + nu)
217
     config['t'] = t
218
     config['v'] = v
219
220
     config['a'] = a
221
     next w = w
222
223
     224
     # END YOUR CODE HERE
225
     # ============= #
226
227
     return next_w, config
228
229
```

localhost:4649/?mode=python 4/5

localhost:4649/?mode=python 5/5