

```
1 import numpy as np
2
3 """
4 This file implements various first-order update rules that are commonly used
5 for
6 training neural networks. Each update rule accepts current weights and the
7 gradient of the loss with respect to those weights and produces the next set
8 of
9 weights. Each update rule has the same interface:
10
11 def update(w, dw, config=None):
12
13 Inputs:
14 - w: A numpy array giving the current weights.
15 - dw: A numpy array of the same shape as w giving the gradient of the
16 loss with respect to w.
17 - config: A dictionary containing hyperparameter values such as learning
18 rate,
19 momentum, etc. If the update rule requires caching values over many
20 iterations, then config will also hold these cached values.
21
22 Returns:
23 - next_w: The next point after the update.
24 - config: The config dictionary to be passed to the next iteration of the
25 update rule.
26
27 NOTE: For most update rules, the default learning rate will probably not
28 perform
29 well; however the default values of the other hyperparameters should work
30 well
31 for a variety of different problems.
32
33 For efficiency, update rules may perform in-place updates, mutating w and
34 setting next_w equal to w.
35 """
36
37 def sgd(w, dw, config=None):
38     """
39     Performs vanilla stochastic gradient descent.
40
41     config format:
42     - learning_rate: Scalar learning rate.
43     """
44     if config is None: config = {}
45     config.setdefault('learning_rate', 1e-2)
46
47     w -= config['learning_rate'] * dw
48     return w, config
49
50 def sgd_momentum(w, dw, config=None):
51     """
52     Performs stochastic gradient descent with momentum.
53
54     config format:
55     - learning_rate: Scalar learning rate.
56     - momentum: Scalar between 0 and 1 giving the momentum value.
57     Setting momentum = 0 reduces to sgd.
```

```

55     - 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 = {}
59     config.setdefault('learning_rate', 1e-2)
60     config.setdefault('momentum', 0.9) # set momentum to 0.9 if it wasn't there
61     v = config.get('velocity', np.zeros_like(w)) # gets velocity, else sets
it to zero.
62
63     # ===== #
64     # YOUR CODE HERE:
65     #     Implement the momentum update formula. Return the updated weights
66     #     as next_w, and the updated velocity as v.
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.
93     - velocity: A numpy array of the same shape as w and dw used to store a
moving
94     average of the gradients.
95     """
96     if config is None: config = {}
97     config.setdefault('learning_rate', 1e-2)
98     config.setdefault('momentum', 0.9) # set momentum to 0.9 if it wasn't there
99     v = config.get('velocity', np.zeros_like(w)) # gets velocity, else sets
it to zero.
100
101     # ===== #
102     # YOUR CODE HERE:
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']
108     eps = config['learning_rate']
109
110     v_old = v

```

```

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
127     gradient
128     values to set adaptive per-parameter learning rates.
129
130     config format:
131     - learning_rate: Scalar learning rate.
132     - decay_rate: Scalar between 0 and 1 giving the decay rate for the squared
133       gradient cache.
134     - epsilon: Small scalar used for smoothing to avoid dividing by zero.
135     - beta: Moving average of second moments of gradients.
136     """
137     if config is None: config = {}
138     config.setdefault('learning_rate', 1e-2)
139     config.setdefault('decay_rate', 0.99)
140     config.setdefault('epsilon', 1e-8)
141     config.setdefault('a', np.zeros_like(w))
142
143     next_w = None
144
145     # ===== #
146     # YOUR CODE HERE:
147     # Implement RMSProp. Store the next value of w as next_w. You need
148     # to also store in config['a'] the moving average of the second
149     # moment gradients, so they can be used for future gradients. Concretely,
150     # config['a'] corresponds to "a" in the lecture notes.
151     # ===== #
152
153     a = config['a']
154     beta = config['decay_rate']
155     eps = config['learning_rate']
156     nu = config['epsilon']
157
158     a = beta * a + (1 - beta) * dw * dw
159     w -= (eps * dw) / (np.sqrt(a) + nu)
160
161     config['a'] = a
162     next_w = w
163
164     # ===== #
165     # END YOUR CODE HERE
166     # ===== #
167
168     return next_w, config
169

```

```

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.
177     - beta1: Decay rate for moving average of first moment of gradient.
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 = {}
185     config.setdefault('learning_rate', 1e-3)
186     config.setdefault('beta1', 0.9)
187     config.setdefault('beta2', 0.999)
188     config.setdefault('epsilon', 1e-8)
189     config.setdefault('v', np.zeros_like(w))
190     config.setdefault('a', np.zeros_like(w))
191     config.setdefault('t', 0)
192
193     next_w = None
194
195     # ===== #
196     # YOUR CODE HERE:
197     # Implement Adam. Store the next value of w as next_w. You need
198     # to also store in config['a'] the moving average of the second
199     # moment gradients, and in config['v'] the moving average of the
200     # first moments. Finally, store in config['t'] the increasing time.
201     # ===== #
202
203     t = config['t']
204     v = config['v']
205     a = config['a']
206     eps = config['learning_rate']
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
214     v_u = v / (1 - beta1**t)
215     a_u = a / (1 - beta2**t)
216     w -= (eps * v_u) / (np.sqrt(a_u) + nu)
217
218     config['t'] = t
219     config['v'] = v
220     config['a'] = a
221     next_w = w
222
223     # ===== #
224     # END YOUR CODE HERE
225     # ===== #
226
227     return next_w, config
228
229

```

230
231
232
233