

```

1  import numpy as np
2
3  """
4  This file implements various first-order update rules that are commonly used for
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 of
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,
16        momentum, etc. If the update rule requires caching values over many
17        iterations, then config will also hold these cached values.
18
19  Returns:
20      - next_w: The next point after the update.
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 perform
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 sgd(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.
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     v = config['momentum'] * v - config['learning_rate'] * dw
69     next_w = v + w
70     # ===== #
71     # END YOUR CODE HERE
72     # ===== #
73
74     config['velocity'] = v
75
76     return next_w, config
77
78 def sgd_nesterov_momentum(w, dw, config=None):
79     """
80     Performs stochastic gradient descent with Nesterov momentum.
81
82     config format:
83     - learning_rate: Scalar learning rate.
84     - momentum: Scalar between 0 and 1 giving the momentum value.
85       Setting momentum = 0 reduces to sgd.
86     - velocity: A numpy array of the same shape as w and dw used to store a moving
87       average of the gradients.
88     """
89     if config is None: config = {}
90     config.setdefault('learning_rate', 1e-2)
91     config.setdefault('momentum', 0.9) # set momentum to 0.9 if it wasn't there
92     v = config.get('velocity', np.zeros_like(w)) # gets velocity, else sets it to zero.
93
94     # ===== #
95     # YOUR CODE HERE:
96     #   Implement the momentum update formula. Return the updated weights
97     #   as next_w, and the updated velocity as v.
98     # ===== #
99     v_prev = v
100    v = config['momentum'] * v_prev - config['learning_rate'] * dw
101    next_w = v + w + config['momentum'] * (v - v_prev)
102    # ===== #
103    # END YOUR CODE HERE
104    # ===== #
105
106    config['velocity'] = v
107
108    return next_w, config
109
110 def rmsprop(w, dw, config=None):
111     """
112     Uses the RMSProp update rule, which uses a moving average of squared gradient
113     values to set adaptive per-parameter learning rates.
114
115     config format:
116     - learning_rate: Scalar learning rate.
117     - decay_rate: Scalar between 0 and 1 giving the decay rate for the squared
118       gradient cache.
119     - epsilon: Small scalar used for smoothing to avoid dividing by zero.
120     - beta: Moving average of second moments of gradients.
121     """
122     if config is None: config = {}
123     config.setdefault('learning_rate', 1e-2)
124     config.setdefault('decay_rate', 0.99)
125     config.setdefault('epsilon', 1e-8)
126     config.setdefault('a', np.zeros_like(w))
127
128     next_w = None
129
130     # ===== #
131     # YOUR CODE HERE:
132     #   Implement RMSProp. Store the next value of w as next_w. You need
133     #   to also store in config['a'] the moving average of the second
134     #   moment gradients, so they can be used for future gradients. Concretely,

```

```

135 # config['a'] corresponds to "a" in the lecture notes.
136 # ===== #
137 config['a'] = config['decay_rate'] * config['a'] + (1 - config['decay_rate']) * dw *
dw
138 c = 1 / (np.sqrt(config['a']) + config['epsilon'])
139 next_w = w - config['learning_rate'] * c * dw
140 # ===== #
141 # END YOUR CODE HERE
142 # ===== #
143
144 return next_w, config
145
146
147 def adam(w, dw, config=None):
148     """
149     Uses the Adam update rule, which incorporates moving averages of both the
150     gradient and its square and a bias correction term.
151
152     config format:
153     - learning_rate: Scalar learning rate.
154     - beta1: Decay rate for moving average of first moment of gradient.
155     - beta2: Decay rate for moving average of second moment of gradient.
156     - epsilon: Small scalar used for smoothing to avoid dividing by zero.
157     - m: Moving average of gradient.
158     - v: Moving average of squared gradient.
159     - t: Iteration number.
160     """
161     if config is None: config = {}
162     config.setdefault('learning_rate', 1e-3)
163     config.setdefault('beta1', 0.9)
164     config.setdefault('beta2', 0.999)
165     config.setdefault('epsilon', 1e-8)
166     config.setdefault('v', np.zeros_like(w))
167     config.setdefault('a', np.zeros_like(w))
168     config.setdefault('t', 0)
169
170     next_w = None
171
172     # ===== #
173     # YOUR CODE HERE:
174     # Implement Adam. Store the next value of w as next_w. You need
175     # to also store in config['a'] the moving average of the second
176     # moment gradients, and in config['v'] the moving average of the
177     # first moments. Finally, store in config['t'] the increasing time.
178     # ===== #
179
180     config['t'] += 1
181     config['v'] = config['beta1'] * config['v'] + (1 - config['beta1']) * dw
182     config['a'] = config['beta2'] * config['a'] + (1 - config['beta2']) * dw * dw
183
184     v_tld = config['v'] / (1 - config['beta1'] ** config['t'])
185     a_tld = config['a'] / (1 - config['beta2'] ** config['t'])
186
187     c = 1 / (np.sqrt(a_tld) + config['epsilon'])
188     next_w = w - config['learning_rate'] * v_tld * c
189
190     # ===== #
191     # END YOUR CODE HERE
192     # ===== #
193
194     return next_w, config
195

```