```python
import numpy as np


"""
This file implements various first-order update rules that are commonly used for
training neural networks. Each update rule accepts current weights and the
gradient of the loss with respect to those weights and produces the next set of
weights. Each update rule has the same interface:

def update(w, dw, config=None):

Inputs:
  - w: A numpy array giving the current weights.
  - dw: A numpy array of the same shape as w giving the gradient of the
    loss with respect to w.
  - config: A dictionary containing hyperparameter values such as learning rate,
    momentum, etc. If the update rule requires caching values over many
    iterations, then config will also hold these cached values.

Returns:
  - next_w: The next point after the update.
  - config: The config dictionary to be passed to the next iteration of the
    update rule.

NOTE: For most update rules, the default learning rate will probably not perform
well; however the default values of the other hyperparameters should work well
for a variety of different problems.

For efficiency, update rules may perform in-place updates, mutating w and
setting next_w equal to w.
"""


def sgd(w, dw, config=None):
  """
  Performs vanilla stochastic gradient descent.

  config format:
  - learning_rate: Scalar learning rate.
  """
  if config is None: config = {}
  config.setdefault('learning_rate', 1e-2)

  w -= config['learning_rate'] * dw
  return w, config


def sgd_momentum(w, dw, config=None):
  """
  Performs stochastic gradient descent with momentum.

  config format:
  - learning_rate: Scalar learning rate.
  - momentum: Scalar between 0 and 1 giving the momentum value.
```

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

```
111   v_old = v
112   v = alpha * v - eps * dw
113   w += (v + alpha * (v - v_old))
114
115   next_w = w
116
117   # ================================================================ #
118   # END YOUR CODE HERE
119   # ================================================================ #
120
121   config['velocity'] = v
122
123   return next_w, config
124
125 def rmsprop(w, dw, config=None):
126   """
127   Uses the RMSProp update rule, which uses a moving average of squared
    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
```

```python

def adam(w, dw, config=None):
  """
  Uses the Adam update rule, which incorporates moving averages of both the
  gradient and its square and a bias correction term.

  config format:
  - learning_rate: Scalar learning rate.
  - beta1: Decay rate for moving average of first moment of gradient.
  - beta2: Decay rate for moving average of second moment of gradient.
  - epsilon: Small scalar used for smoothing to avoid dividing by zero.
  - m: Moving average of gradient.
  - v: Moving average of squared gradient.
  - t: Iteration number.
  """
  if config is None: config = {}
  config.setdefault('learning_rate', 1e-3)
  config.setdefault('beta1', 0.9)
  config.setdefault('beta2', 0.999)
  config.setdefault('epsilon', 1e-8)
  config.setdefault('v', np.zeros_like(w))
  config.setdefault('a', np.zeros_like(w))
  config.setdefault('t', 0)

  next_w = None

  # ================================================================ #
  # YOUR CODE HERE:
  #   Implement Adam.  Store the next value of w as next_w.  You need
  #   to also store in config['a'] the moving average of the second
  #   moment gradients, and in config['v'] the moving average of the
  #   first moments.  Finally, store in config['t'] the increasing time.
  # ================================================================ #

  t = config['t']
  v = config['v']
  a = config['a']
  eps = config['learning_rate']
  nu = config['epsilon']
  beta1 = config['beta1']
  beta2 = config['beta2']

  t += 1
  v = beta1 * v + (1 - beta1) * dw
  a = beta2 * a + (1 - beta2) * dw * dw
  v_u = v / (1 - beta1**t)
  a_u = a / (1 - beta2**t)
  w -= (eps * v_u) / (np.sqrt(a_u) + nu)

  config['t'] = t
  config['v'] = v
  config['a'] = a
  next_w = w

  # ================================================================ #
  # END YOUR CODE HERE
  # ================================================================ #

  return next_w, config

```

```
230
231
232
233
234
```