|  |
| --- |
| import tensorflow as tf |
| import numpy as np |
|  |
| # Definisi model Deep Q-Network (DQN) |
| class DQN(tf.keras.Model): |
| def \_\_init\_\_(self, state\_size, action\_size): |
| super(DQN, self).\_\_init\_\_() |
| self.dense1 = tf.keras.layers.Dense(64, activation='relu') |
| self.dense2 = tf.keras.layers.Dense(64, activation='relu') |
| self.output\_layer = tf.keras.layers.Dense(action\_size, activation=None) |
|  |
| def call(self, inputs): |
| x = self.dense1(inputs) |
| x = self.dense2(x) |
| return self.output\_layer(x) |
|  |
| # Inisialisasi model |
| state\_size = 64 # misalnya, 64 kotak pada papan catur |
| action\_size = 4096 # jumlah langkah yang mungkin dalam permainan catur |
| model = DQN(state\_size, action\_size) |
|  |
| # Fungsi loss dan optimizer |
| optimizer = tf.keras.optimizers.Adam(learning\_rate=0.001) |
| loss\_function = tf.keras.losses.MeanSquaredError() |
|  |
| # Training loop (gunakan environment dan data training) |
| for episode in range(num\_episodes): |
| state = env.reset() |
| done = False |
| while not done: |
| # Pilih aksi berdasarkan kebijakan epsilon-greedy |
| action = epsilon\_greedy\_policy(state) |
|  |
| # Ambil langkah di environment |
| next\_state, reward, done, \_ = env.step(action) |
|  |
| # Hitung target Q-value menggunakan rumus Bellman |
| target = reward + gamma \* np.max(model.predict(next\_state)) |
|  |
| # Hitung loss dan lakukan backpropagation |
| with tf.GradientTape() as tape: |
| predicted\_values = model(state) |
| loss = loss\_function(tf.stop\_gradient(target), predicted\_values[action]) |
| gradients = tape.gradient(loss, model.trainable\_variables) |
| optimizer.apply\_gradients(zip(gradients, model.trainable\_variables)) |
|  |
| state = next\_state |