Predictive Analytics World / Deep Learning World
Exercises - Adam Optimization

1. Locate the research paper "Adam: A Method for Stochastic Optimization", by D. P. Kingma and J. L. Bas (2015). Using the paper's Algorithm 1, write a program that implements the Adam (adaptive moment estimation) algorithm and use it to minimize the function f(W) = sin(W0) + cos(W1).

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

2. Modify your program to minimize the function f(W) = tan(W0) + (W1)^3 + e^(W2).

3. Which statement is most accurate?

a.) The Adam algorithm is based on the earlier AdaGrad and RMSProp algorithms.
b.) The Adam algorithm is based on Nesterov momentum.
c.) The Adam algorithm is not closely related to any previous optimization algorithm.

4. Which statement is most accurate?

a.) The Adam algorithm uses a different learning rate for different weights.
b.) The Adam algorithm uses the same learning rate for all weights, but different rates for the biases.
c.) The Adam algorithm uses the same learning rate for all nodes in a neural layer, but different rates for different layers.

```python
# adam_demo.py
# see https://arxiv.org/pdf/1412.6980v8.pdf

import numpy as np

def main():
  print("\nBegin Adam optimization algorithm demo \n")
  print("Goal is to minimize E = sin(W0) + cos(W1)")
  print("Solution is W0 = -pi/2 W1 = pi (-1.5708, 3.1416) \n")

  W = np.array([0.1, 0.2], dtype=np.float32)    # theta
  g = np.zeros(shape=[2], dtype=np.float32)     # gradients
  a = np.float32(0.001)                         # alpha
  b1 = np.float32(0.9)                          # beta1
  b2 = np.float32(0.999)                        # beta2
  b1t = np.float32(0.9)                         # b1^t
  b2t = np.float32(0.999)                       # b2^t
  eps = np.float32(1.0e-8)                      # epsilon
  m = np.zeros(shape=[2], dtype=np.float32)     # first moment
  v = np.zeros(shape=[2], dtype=np.float32)     # second moment
  mhat = np.zeros(shape=[2], dtype=np.float32)  # corrected m
  vhat = np.zeros(shape=[2], dtype=np.float32)  # corrected v

  t = np.int(0)
  while t <= 5000:
    t += 1

    g[0] = np.cos(W[0])   # compute partial derivatives
    g[1] = -np.sin(W[1])

    m[0] = (b1 * m[0]) + ((1 - b1) * g[0])  # biased first moment est
    m[1] = (b1 * m[1]) + ((1 - b1) * g[1])

    v[0] = (b2t * v[0]) + ((1 - b2t) * (g[0] * g[0]))  # second moment
    v[1] = (b2t * v[1]) + ((1 - b2t) * (g[1] * g[1]))

    b1t = b1t * b1
    b2t = b2t * b2
    mhat[0] = m[0] / (1 - b1t)  # bias-corrected first moment
    mhat[1] = m[1] / (1 - b1t)

    vhat[0] = v[0] / (1 - b2t)  # bias-corrected second moment
    vhat[1] = v[1] / (1 - b2t)

    W[0] = W[0] - ((a * mhat[0]) / np.sqrt(vhat[0] + eps))
    W[1] = W[1] - ((a * mhat[1]) / np.sqrt(vhat[1] + eps))

    if t % 500 == 0:
      print("t = %6d W[0] = % 0.4f  W[1] = % 0.4f " % (t, W[0], W[1]))

if __name__ == "__main__":
  main()
```