```
function [W5, Wo] = OneEpochUpdate(W1, W5, Wo, X, D)
// updates W5 and Wo after one epoch
// X is the set of 10000 training data each of size 28 x 28 = 784; D is the set of labels of size 10000
// W1 contains the filter coefficients - W1[1], W1[2], ... W1[20] form the 20 filters
// W5 is the set of weights associated with a hidden layer of 100 neurons. Dimension of W5 is 2000 by 100
// Wo is the set of weights associated with the output layer neurons. Dimension of Wo is 100 by 10
N = length(D)
alpha = 0.01; // learning rate, you can try changing this, but not too much
// training data divided into 100 batches of 100 rows
bsize = 100;
blist = [1, 101, 201, ..., 7901]
% a single epoch loop
%
for batch = 1 to 100 // length(blist)
  dW5 = matrix of order 2000 by 100 initialized to 0.0
  dWo = matrix of order 100 by 10 initialized to 0.0
  % Mini-batch loop
  %
  begin = blist[batch];
  for k = begin to (begin+bsize-1)
    // Forward pass - evaluate the output for input using current weights
    x  = X[k]                     // x is the k-th row of size 28x28
    y1 = Conv(x, W1);             // applying all 20 filters to generate a 20x20x20 matrix
    y2 = ReLU(y1);                // y2 is also a 20 x 20 x 20 matrix
    y3 = Pool(y2);                // Mean-Pooling results in 10x10x20 matrix
    y4 = reshape(y3);             // converts y3 into a 1-dim matrix of order 1 x 2000
    v5 = y4*W5;                   // multiplying by hidden-layer matrix gives a 1 x 100 matrix
    y5 = ReLU(v5);                // just replace -ve values in v5 by 0; v5 is also 1 x 100
    v  = y5*Wo;                   // multiplying by output-layer matrix gives a 1 x 10 matrix
    y  = Softmax(v);              // apply Softmax function to turn v into a vector that sums to 1

    d = correct output; // d is a 1 by 10 matrix in which d[i] = 1 where i is the label associated with the input; 0 else
    // Thus, if the D[k] = i, then d[i] = 1, and for all j != i, d[j] = 0

    //Backpropagation
    delta  = d - y;               // error at the output layer; delta is 1 x 10
    e5     = delta * Wo';         // Wo' is just the transpose of Wo. e5 is 1 x 100
    delta5 = dotProduct(y5,  e5);
    dW5 = dW5 + y4' * delta5;
    dWo = dWo + y5' * delta ;
  end-for (inner)

  % Update weights for the mini-match
  dW5 = dW5 / bsize;
  dWo = dWo / bsize;
  W5  = W5 + alpha*dW5;
  Wo  = Wo + alpha*dWo;
return W5,
end-for (outer)
end (of function)
```