Name: **Student ID**:

Quiz 10

1. Let $\mathbf{h}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b})$. Given the gradient of the loss w.r.t. $\mathbf{h}_t$, i.e., $\partial L/\partial \mathbf{h}_t$, please show that the gradients can be implemented by

```python
"""
Backward pass for a single timestep of a vanilla RNN.

Inputs:
- dnext_h: Gradient of loss with respect to next hidden state, of shape (N, H)
- cache: Cache object from the forward pass

Returns a tuple of:
- dx: Gradients of input data, of shape (N, D)
- dprev_h: Gradients of previous hidden state, of shape (N, H)
- dWx: Gradients of input-to-hidden weights, of shape (D, H)
- dWh: Gradients of hidden-to-hidden weights, of shape (H, H)
- db: Gradients of bias vector, of shape (H,)
"""

x, prev_h, Wx, Wh, next_h = cache
dout_dnext_h = dnext_h * (1 - torch.pow(next_h, 2))
dx = dout_dnext_h.mm(Wx.T)
dprev_h = dout_dnext_h.mm(Wh.T)
dWx = x.T.mm(dout_dnext_h)
dWh = prev_h.T.mm(dout_dnext_h)
db = dout_dnext_h.sum(axis=0)
```

You only need to sketch the results by using the computational graph and do NOT need to show the derivations.

Hint. It is more convenient to follow the convention "the shape of the gradient equals the shape of the parameter" (see Quiz 3).

2. Please plot the computational graph of the many-to-many RNN (p.33 of Lecture 11) and show that forward of RNN can be implemented by

```
def rnn_step_forward(x, prev_h, Wx, Wh, b):
    next_h = torch.tanh( torch.mm(x, Wx) + torch.mm(prev_h, Wh) + b)
    cache = (x, prev_h, Wx, Wh, next_h)

def rnn_forward(x, h0, Wx, Wh, b):
    """
    Run a vanilla RNN forward on an entire sequence of data. We assume an input
    sequence composed of T vectors, each of dimension D. The RNN uses a hidden
    size of H, and we work over a minibatch containing N sequences. After running
    the RNN forward, we return the hidden states for all timesteps.

    Inputs:
    - x: Input data for the entire timeseries, of shape (N, T, D).
    - h0: Initial hidden state, of shape (N, H)
    - Wx: Weight matrix for input-to-hidden connections, of shape (D, H)
    - Wh: Weight matrix for hidden-to-hidden connections, of shape (H, H)
    - b: Biases, of shape (H,)

    Returns a tuple of:
    - h: Hidden states for the entire timeseries, of shape (N, T, H).
    - cache: Values needed in the backward pass
    """

    _, H = h0.shape
    N, T, D = x.shape
    h = torch.zeros((N, T, H), dtype=x.dtype, device=x.device)
    cache = []

    prev_h = h0
    for i in range(T):
      th, tcache = rnn_step_forward(x[:, i, :], prev_h, Wx, Wh, b)
      prev_h = th
      h[:, i, :] = th
      cache.append(tcache)
```

3. Given the gradient of the loss w.r.t. $\mathbf{h}_t$ (who produce the loss at timestep t), please show that the gradients can be implemented by

```
"""
Compute the backward pass for a vanilla RNN over an entire sequence of data.

Inputs:
- dh: Upstream gradients of all hidden states, of shape (N, T, H).

NOTE: 'dh' contains the upstream gradients produced by the
individual loss functions at each timestep, *not* the gradients
being passed between timesteps (which you'll have to compute yourself
by calling rnn_step_backward in a loop).

Returns a tuple of:
- dx: Gradient of inputs, of shape (N, T, D)
- dh0: Gradient of initial hidden state, of shape (N, H)
- dWx: Gradient of input-to-hidden weights, of shape (D, H)
- dWh: Gradient of hidden-to-hidden weights, of shape (H, H)
- db: Gradient of biases, of shape (H,)
"""
N, T, H = dh.shape
D = cache[0][0].shape[1]
dx = torch.zeros((N, T, D), dtype=dh.dtype, device=dh.device)
dh0 = torch.zeros((N, H), dtype=dh.dtype, device=dh.device)
dWx = torch.zeros((D, H), dtype=dh.dtype, device=dh.device)
dWh = torch.zeros((H, H), dtype=dh.dtype, device=dh.device)
db = torch.zeros((H), dtype=dh.dtype, device=dh.device)
tdprev_h = torch.zeros((N, H), dtype=dh.dtype, device=dh.device)

for i in range(T - 1, -1, -1):
  tdx, tdprev_h, tdWx, tdWh, tdb = rnn_step_backward(dh[:, i, :] + tdprev_h,
    cache[i])
  dx[:, i, :] = tdx
  dWx += tdWx
  dWh += tdWh
  db += tdb

dh0 = tdprev_h
```

You only need to sketch the results by using the computational graph and do NOT need to show the derivations.