**Question 2**

(a)

Explain how sequential learning differs from traditional supervised learning. Give a real-world example where sequential learning provides significant advantages.

**Answer**

**Sequential Learning vs. Traditional Supervised Learning:**

* **Input Structure:**
  + *Sequential:* Data has **temporal order** (e.g., time series, text).
  + *Traditional:* Inputs are **independent** (e.g., tabular data).
* **Memory Dependency:**
  + *Sequential:* Current output depends on **past inputs** (e.g., RNNs, LSTMs).
  + *Traditional:* Output depends **only on current input** (e.g., linear regression).

**Example Advantage:**

* **Stock Prediction:** Sequential models (LSTMs) outperform traditional models by learning **price trends** from historical data.

(b)

Summarize the main differences between CNNs and RNNs in terms of input structure, weight sharing, and typical application domains.

**Answer**

**Key Differences Between CNNs and RNNs:**

* **Input Structure:**
  + CNNs: Grid-like data (e.g., images, 2D/3D arrays).
  + RNNs: Sequential data (e.g., time series, text, 1D sequences).
* **Weight Sharing:**
  + CNNs: Kernels shared across spatial dimensions (translation invariance).
  + RNNs: Weights shared across time steps (temporal dependency).
* **Applications:**
  + CNNs: Image classification (ResNet), object detection (YOLO).
  + RNNs: Time-series forecasting (LSTM), NLP (translation, sentiment analysis).
* **Memory**: RNNs retain hidden states; CNNs process inputs independently.

(c)

Why is the tanh or sigmoid function commonly used as the activation function in RNNs? What could go wrong if you used the ReLU activation instead?

**Answer**

**Why tanh/sigmoid in RNNs?**

* Bounded Outputs: tanh (-1 to 1) and sigmoid (0 to 1) prevent exploding gradients in recurrent layers.
* Saturating Gradients: Helps stabilize training by limiting gradient magnitude.

**Issues with ReLU:**

* Exploding Gradients: Unbounded outputs (e.g., ReLU(100)=100) can destabilize RNNs over timesteps.
* Dead Neurons: ReLU’s zero gradient for negative inputs may cause neurons to "die" in recurrent loops.
* Example: A ReLU RNN predicting stock prices might output infinity after prolonged upward trends.

**Exception:** Leaky ReLU can mitigate some issues but is less common in vanilla RNNs.

**Q3. Manual RNN Training on a Simple Dataset**

1. **Forward Pass:** For the input sequence [1, 0, 1], compute the hidden states and outputs for all time steps.

**Ans:** **Initial Parameters:**

W\_xh = 1.0, W\_hh = 0.5, W\_hy = 2.0

b\_h = 0, b\_y = 0, h\_0 = 0

η = 0.1

Sequence: x = [1, 0, 1], y = [0, 1, 1]

**Timestep t=1 (x₁=1, y₁=0):**

h₁ = tanh(W\_xh·x₁ + W\_hh·h₀ + b\_h)

= tanh(1.0·1 + 0.5·0 + 0) = tanh(1) ≈ 0.7616

ŷ₁ = σ(W\_hy·h₁ + b\_y)

= σ(2.0·0.7616 + 0) ≈ σ(1.5232) ≈ 0.821

**Timestep t=2 (x₂=0, y₂=1):**

h₂ = tanh(W\_xh·0 + W\_hh·h₁ + b\_h)

= tanh(0 + 0.5·0.7616 + 0) ≈ tanh(0.3808) ≈ 0.363

ŷ₂ = σ(2.0·0.363 + 0) ≈ σ(0.726) ≈ 0.674

**Timestep t=3 (x₃=1, y₃=1):**

h₃ = tanh(1.0·1 + 0.5·0.363 + 0) ≈ tanh(1.1815) ≈ 0.828

ŷ₃ = σ(2.0·0.828 + 0) ≈ σ(1.656) ≈ 0.840

1. **Loss Calculation:** Use **binary cross-entropy loss** for each step:  
   (where yt is the true target and y^t predicted is the output).

**Ans: Loss Calculation (Epoch 1):**

L₁ = -[0·log(0.821) + (1-0)log(1-0.821)] ≈ 0.197

L₂ = -[1·log(0.674) + 0·log(1-0.674)] ≈ 0.394

L₃ = -[1·log(0.840) + 0·log(1-0.840)] ≈ 0.174

Total Loss = 0.197 + 0.394 + 0.174 = 0.765

1. **Backpropagation Through Time (BPTT):** Manually compute gradients for **one epoch** (all three steps) and update all weights using a learning rate η=0.1\eta = 0.1.  
   (Show step-by-step calculations for the first epoch only.)

Ans: **Gradient for W\_hy:**

∂L/∂W\_hy = Σ(∂L\_t/∂W\_hy)

∂L\_t/∂W\_hy = (ŷ\_t - y\_t)·h\_t

∂L₁/∂W\_hy = (0.821-0)·0.7616 ≈ 0.625

∂L₂/∂W\_hy = (0.674-1)·0.363 ≈ -0.118

∂L₃/∂W\_hy = (0.840-1)·0.828 ≈ -0.132

Total ∂L/∂W\_hy ≈ 0.625 - 0.118 - 0.132 = 0.375

**For W\_hy (Output Weight):**

∂L/∂W\_hy = Σ(ŷ\_t - y\_t)·h\_t

= (0.821-0)·0.7616 + (0.674-1)·0.363 + (0.840-1)·0.828

≈ 0.625 - 0.118 - 0.132 = 0.375

Updated W\_hy = 2.0 - 0.1×0.375 = 1.9625

**For b\_y (Output Bias):**

∂L/∂b\_y = Σ(ŷ\_t - y\_t)

= (0.821-0) + (0.674-1) + (0.840-1)

≈ 0.821 - 0.326 - 0.160 = 0.335

Updated b\_y = 0 - 0.1×0.335 = -0.0335

**For W\_xh (Input Weight):**

∂L/∂W\_xh = Σ(∂L\_t/∂h\_t)·(1-h\_t²)·x\_t

Where ∂L\_t/∂h\_t = (ŷ\_t-y\_t)·W\_hy·(1-h\_t²)

Timestep 3:

= (0.840-1)×2.0×(1-0.828²)×1 ≈ -0.160×2.0×0.314×1 ≈ -0.100

Timestep 2:

= (0.674-1)×2.0×(1-0.363²)×0 ≈ 0 (x₂=0)

Timestep 1:

= (0.821-0)×2.0×(1-0.7616²)×1 ≈ 0.821×2.0×0.420×1 ≈ 0.690

Total ∂L/∂W\_xh ≈ -0.100 + 0 + 0.690 = 0.590

Updated W\_xh = 1.0 - 0.1×0.590 = 0.941

**For W\_hh (Recurrent Weight):**

∂L/∂W\_hh = Σ(∂L\_t/∂h\_t)·(1-h\_t²)·h\_{t-1}

Timestep 3:

= -0.100 × 0.363 ≈ -0.036

Timestep 2:

= -0.326×2.0×(1-0.363²)×0.7616 ≈ -0.326×2.0×0.868×0.7616 ≈ -0.431

Timestep 1:

= 0 (no h\_{-1} exists)

Total ∂L/∂W\_hh ≈ -0.036 - 0.431 = -0.467

Updated W\_hh = 0.5 - 0.1×(-0.467) ≈ 0.5467

**For b\_h (Hidden Bias):**

∂L/∂b\_h = Σ(∂L\_t/∂h\_t)·(1-h\_t²)

Timestep 3:

= -0.100

Timestep 2:

= -0.431/0.7616 ≈ -0.566 (from W\_hh calculation)

Timestep 1:

= 0.690 (from W\_xh calculation)

Total ∂L/∂b\_h ≈ -0.100 - 0.566 + 0.690 = 0.024

Updated b\_h = 0 - 0.1×0.024 = -0.0024

**Final Updated Parameters (After Epoch 1):**

W\_xh ≈ 0.941 (initial: 1.0)

W\_hh ≈ 0.5467 (initial: 0.5)

W\_hy ≈ 1.9625 (initial: 2.0)

b\_h ≈ -0.0024 (initial: 0.0)

b\_y ≈ -0.0335 (initial: 0.0)

1. **Second Epoch:** Repeat the forward pass using the **updated weights** (from Step 3) and report the new outputs for each time step.

Ans: **Forward Pass (Epoch 2 with updated weights):**

**Timestep t=1:**

h₁ ≈ tanh(0.9·1 + 0.45·0 + 0) ≈ 0.716

ŷ₁ ≈ σ(1.96·0.716) ≈ 0.836

**Timestep t=2:**

h₂ ≈ tanh(0.9·0 + 0.45·0.716) ≈ 0.322

ŷ₂ ≈ σ(1.96·0.322) ≈ 0.660

**Timestep t=3:**

h₃ ≈ tanh(0.9·1 + 0.45·0.322) ≈ 0.814

ŷ₃ ≈ σ(1.96·0.814) ≈ 0.852