BiRNN Formulas and Operations

📝 Notation Guide

x_t Input at time t	$\vec{h}_t$ Forward hidden state	$\overleftarrow{h}_t$ Backward hidden state
W_f, U_f Forward weights	W_b, U_b Backward weights	y_t Output at time t

→ Forward RNN (Left to Right)

Hidden State:

$$\vec{h}_t = \tanh(W_f \cdot x_t + U_f \cdot \vec{h}_{t-1} + b_f)$$

Processes sequence from **start to end**, capturing past context. Each hidden state depends on current input and previous hidden state.

← Backward RNN (Right to Left)

Hidden State:

$$\overleftarrow{h}_t = \tanh(W_b \cdot x_t + U_b \cdot \overleftarrow{h}_{t+1} + b_b)$$

Processes sequence from **end to start**, capturing future context. Each hidden state depends on current input and next hidden state.

Output Combination

Concatenation: $h_t = [h_t^{\rightarrow} ; h_t^{\leftarrow}]$

Output Layer: $y_t = W_o \cdot h_t + b_o = W_o \cdot [h_t^{\rightarrow} ; h_t^{\leftarrow}] + b_o$

Combines both directions by **concatenating** forward and backward hidden states, creating a representation with **full context** (past + future).

🎯 Key Computational Points

Independent Processing

Forward and backward RNNs have separate parameters and process independently

Double Dimensionality

Output dimension is 2× hidden size due to concatenation: [h→; h←]

Parallel Computation

Full Sequence Required

Forward and backward passes can be computed in parallel for efficiency

Complete sequence must be available before backward RNN can start processing

Implementation Note

In practice, frameworks like PyTorch and TensorFlow handle BiRNN efficiently with built-in functions (e.g., `nn.LSTM(bidirectional=True)`). The framework automatically manages the forward and backward passes, parameter initialization, and output concatenation.