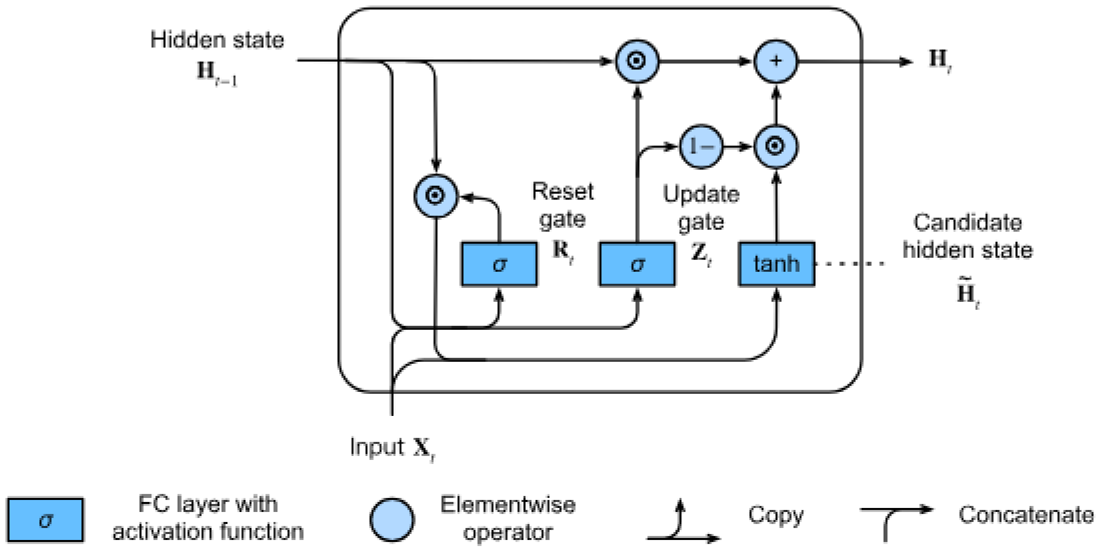

Gated Recurrent Unit (GRU) is a type of Recurrent Neural Network (RNN) that was introduced to solve the **vanishing gradient** problem and improve long-term dependency learning, similar to LSTMs, but with a simpler architecture.

🧠 1. What is GRU?

GRU introduces two gates:

- **Update Gate** ztz_t
- **Reset Gate** rtr_t

These gates control the flow of information, allowing the network to retain relevant information and forget irrelevant data.

🧮 2. Mathematical Formulation

Let:

- xtx_t: input at time step tt
- hth_t: hidden state at time step tt
- ht−1h_{t-1}: hidden state at previous time step
- σ\sigma: sigmoid function
- tanh\tanh: hyperbolic tangent function

2.1 Update Gate (z_t)

Determines how much of the previous memory to keep.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$

2.2 Reset Gate (r_t)

Controls how much of the past information to forget.

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$

2.3 Candidate Activation ($\tilde{h}_t$)

Generates new candidate memory.

$$\tilde{h}_t = \tanh(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$

Here, $\odot$ represents element-wise multiplication.

2.4 Final Memory at Time t

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

This mixes the old memory and the new candidate based on the update gate.

📊 3. Intuition

- If $z_t \approx 1$: keep new candidate $\tilde{h}_t$
- If $z_t \approx 0$: retain old state h_{t-1}
- If $r_t \approx 0$: ignore previous hidden state when generating $\tilde{h}_t$

📈 4. Simple Example

Suppose we're trying to predict the next number in a sequence like:

Input: [0, 1, 0, 1, 0]

Target: [1, 0, 1, 0, 1]

This is a pattern learning task.

🧑‍💻 5. Code Example (PyTorch)

```
import torch

import torch.nn as nn


# Sample data
```

```python
X = torch.tensor([[0], [1], [0], [1], [0]], dtype=torch.float32).view(1, 5, 1)

Y = torch.tensor([[1], [0], [1], [0], [1]], dtype=torch.float32).view(1, 5, 1)


# Define GRU model

class GRUNet(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):

        super(GRUNet, self).__init__()

        self.hidden_size = hidden_size

        self.gru = nn.GRU(input_size, hidden_size, batch_first=True)

        self.fc = nn.Linear(hidden_size, output_size)


    def forward(self, x):

        h0 = torch.zeros(1, x.size(0), self.hidden_size)

        out, _ = self.gru(x, h0)

        out = self.fc(out)

        return out


# Instantiate model

model = GRUNet(input_size=1, hidden_size=8, output_size=1)


# Loss and optimizer

criterion = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)


# Training loop

for epoch in range(300):

    output = model(X)

    loss = criterion(output, Y)
```

```
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    if epoch % 50 == 0:
        print(f"Epoch {epoch}, Loss: {loss.item():.4f}")

# Predict
with torch.no_grad():
    prediction = model(X)
    print("Prediction:", prediction.view(-1).numpy())
```

✅ 6. Why GRU over LSTM?

Feature	GRU	LSTM
Gates	2 (Update, Reset)	3 (Input, Forget, Output)
Simpler?	✅ Yes	❌ No (more parameters)
Performance	Similar	Similar
Training Speed	Faster	Slower

🧠 7. Use Cases

- Text Generation
- Time Series Prediction
- Machine Translation
- Speech Recognition
