Recurrent Neural Network from Scratch

Bhanu Prasanna Koppolu

Table of contents

Recurrent Neural Network 3
- Architecture 3
- Loss Function 3
- Backpropagation Through Time (BPTT) 4
- Gradient Clipping 4

Implementation 4
- Training Example 6
- Vanishing/Exploding Gradients 7

Source Code

Full implementation available at DL_from_scratch/rnn.py

Recurrent Neural Network

RNNs process sequential data by maintaining a hidden state that captures information from previous time steps.

Architecture

For each time step t:

$$a_t = x_t W_{xh} + h_{t-1} W_{hh} + b_h$$

$$h_t = \tanh(a_t)$$

$$\hat{y} = h_T W_{hy} + b_y$$

Where:

- x_t - input at time t
- h_t - hidden state at time t
- W_{xh} - input-to-hidden weights (D $\times$ H)
- W_{hh} - hidden-to-hidden weights (H $\times$ H)
- W_{hy} - hidden-to-output weights (H $\times$ O)
- $h_0 = 0$ - initial hidden state

Loss Function

For sequence prediction (predicting next number):

$$L = \frac{1}{2}(\hat{y} - y)^2$$

Backpropagation Through Time (BPTT)

Gradients flow backwards through all time steps:

$$\delta_t = \frac{\partial L}{\partial h_t} \odot (1 - h_t^2)$$

$$\frac{\partial L}{\partial W_{xh}} = \sum_t x_t^T \delta_t$$

$$\frac{\partial L}{\partial W_{hh}} = \sum_t h_{t-1}^T \delta_t$$

$$\frac{\partial L}{\partial h_{t-1}} = \delta_t W_{hh}^T$$

Gradient Clipping

To prevent exploding gradients:

$$g = \begin{cases} g \cdot \frac{\text{clip\_norm}}{\|g\|} & \text{if } \|g\| > \text{clip\_norm} \\ g & \text{otherwise} \end{cases}$$

Implementation

```
class RNN:
    """Vanilla RNN for sequence prediction.

    Trained to predict the next number in a sequence.
    """

    def __init__(self,
                 input=1,      # Input dimension (D)
                 hidden=100,   # Hidden dimension (H)
                 seq_len=50):  # Sequence length (T)

        self.output = 1  # Predicting single value
```

```
        # Weight matrices (small random initialization)
        self.w_xh = np.random.randn(input, hidden) * 0.1    # D × H
        self.w_hh = np.random.randn(hidden, hidden) * 0.1   # H × H
        self.w_hy = np.random.randn(hidden, self.output) * 0.1  # H × O

        self.b_h = np.random.rand(hidden)  # Hidden bias
        self.b_y = np.random.rand(input)   # Output bias

    def forward(self, x):
        """Forward pass through sequence.

        h_t[0] = 0 (initial hidden state)
        For t = 1 to T:
            a_t = x_t @ W_xh + h_{t-1} @ W_hh + b_h
            h_t = tanh(a_t)
        y_hat = h_T @ W_hy + b_y
        """
        self.x = x
        self.h_t = np.zeros((self.seq_len + 1, self.hidden))
        self.a_t = np.zeros((self.seq_len, self.hidden))

        for t in range(1, self.seq_len + 1):
            self.a_t[t-1] = (self.x[t-1] @ self.w_xh) + \
                            (self.h_t[t-1] @ self.w_hh) + self.b_h
            self.h_t[t] = np.tanh(self.a_t[t-1])

        self.y_hat = (self.h_t[self.seq_len] @ self.w_hy + self.b_y).item()
        return self.y_hat

    def backward(self, e, learning_rate=0.001):
        """Backpropagation through time (BPTT).

        1. Compute output layer gradients
        2. For t = T down to 1:
           - Compute delta_t = dL/dh_t * tanh'(a_t)
           - Accumulate weight gradients
           - Propagate gradient to h_{t-1}
        3. Clip gradients to prevent explosion
        4. Update weights

        Args:
            e: Error (y_hat - y_true)
```

```
        """
        # Output layer gradients
        dl_dw_hy = self.h_t[self.seq_len][:, None] * e
        dl_db_y = np.array([e])

        # Gradient flowing into final hidden state
        dl_dh_t = (self.w_hy[:, 0] * e)

        # Accumulate gradients over time
        dl_dw_xh = np.zeros_like(self.w_xh)
        dl_dw_hh = np.zeros_like(self.w_hh)
        dl_db_h = np.zeros_like(self.b_h)

        for t in range(self.seq_len, 0, -1):
            # Tanh derivative: 1 - h_t^2
            delta_t = dl_dh_t * (1.0 - self.h_t[t] ** 2)

            # Accumulate weight gradients
            dl_dw_xh += np.outer(self.x[t-1], delta_t)
            dl_dw_hh += np.outer(self.h_t[t-1], delta_t)
            dl_db_h += delta_t

            # Propagate to previous hidden state
            dl_dh_t = delta_t @ self.w_hh.T

        # Gradient clipping
        ...

        # Weight updates
        ...
```

Training Example

The model is trained to predict the next number in a sequence:

```
# Example: Given [1, 2, 3, ..., 50], predict 51
rnn = RNN(input=1, hidden=100, seq_len=50)

for epoch in range(epochs):
    start = np.random.randint(1, 901)
    inp = (np.arange(start, start + 50) / 1000).reshape(50, 1)
```

```
    tar = (start + 50) / 1000

    y_hat = rnn.forward(inp)
    loss = 0.5 * (y_hat - tar) ** 2
    e = y_hat - tar
    rnn.backward(e, learning_rate=0.005)
```

Vanishing/Exploding Gradients

RNNs suffer from:

- **Vanishing gradients**: Gradients shrink exponentially over long sequences
- **Exploding gradients**: Gradients grow exponentially (mitigated by clipping)

This is why LSTMs and GRUs were developed to handle long-term dependencies better.