Recurrent Neural Networks: Theory and Practice

October 3, 2025

1 RNN Structure and Core Principles

Recurrent neural networks (RNNs) process sequential data by maintaining a hidden state $\mathbf{h}_t$ that aggregates past information. A simple Elman RNN updates

$$\mathbf{a}_t = \mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h, \tag{1}$$

$$\mathbf{h}_t = \phi(\mathbf{a}_t), \tag{2}$$

$$\mathbf{y}_t = \mathbf{W}_{hy}\mathbf{h}_t + \mathbf{b}_y, \tag{3}$$

where ϕ is typically tanh or ReLU. Figure **??** depicts the recurrent computation graph unrolled in time.

1.1 Unrolling and Backpropagation Through Time

Training involves unfolding the network over T steps and applying backpropagation through time (BPTT). Gradients of the loss $\mathcal{L}$ with respect to parameters accumulate contributions from each timestep:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{hh}} = \sum_{t=1}^{T} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{a}_t} \frac{\partial \mathbf{a}_t}{\partial \mathbf{W}_{hh}} \right). \tag{4}$$

Repeated multiplication by $\mathbf{W}_{hh}$ can cause exploding or vanishing gradients. Gradient clipping and orthogonal initialization mitigate instability.

1.2 Variants of Recurrent Connections

Bidirectional RNNs process sequences in both forward and backward directions, concatenating hidden states to capture past and future context. Deep (stacked) RNNs add layers of recurrence for hierarchical representations. Residual and highway connections ease optimization by facilitating gradient flow across layers.

Listing 1: PyTorch Elman RNN cell with gated recurrent unit interface.

```python
import torch
import torch.nn as nn

class SimpleRNNCell(nn.Module):
    def __init__(self, input_size, hidden_size):
        super().__init__()
        self.Wxh = nn.Linear(input_size, hidden_size)
        self.Whh = nn.Linear(hidden_size, hidden_size)

    def forward(self, x_t, h_prev):
        a_t = self.Wxh(x_t) + self.Whh(h_prev)
        h_t = torch.tanh(a_t)
        return h_t
```

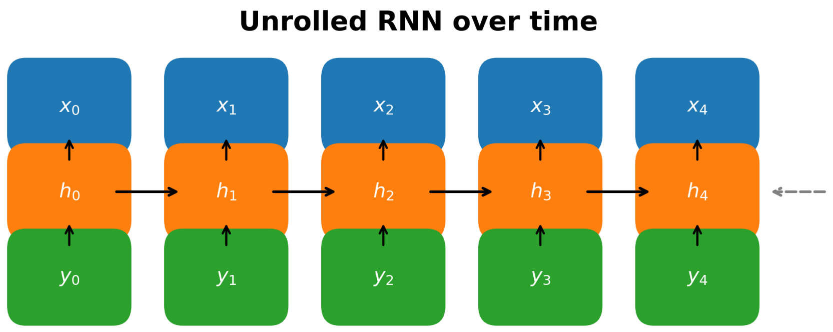

Figure 1: Unrolled recurrent computation showing information flow and hidden-to-hidden transitions.

2 LSTM and GRU Architectures

Long short-term memory (LSTM) and gated recurrent units (GRU) introduce gating mechanisms to alleviate vanishing gradients and capture long-range dependencies. Figure **??** compares their gates.

2.1 LSTM Structure

An LSTM cell maintains a cell state $\mathbf{c}_t$ alongside the hidden state $\mathbf{h}_t$. Gates regulate information flow:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f), \tag{5}$$
$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i), \tag{6}$$
$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_c), \tag{7}$$
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \tag{8}$$
$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o), \tag{9}$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \tag{10}$$

The forget gate $\mathbf{f}_t$ preserves long-term context, while input and output gates control updates and exposure.

2.2 GRU Structure

GRUs simplify LSTMs by combining cell and hidden states. Update and reset gates compute

$$\mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_z), \tag{11}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_r), \tag{12}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h[\mathbf{x}_t, \mathbf{r}_t \odot \mathbf{h}_{t-1}] + \mathbf{b}_h), \tag{13}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t. \tag{14}$$

GRUs require fewer parameters and often converge faster while retaining the ability to model dependencies.

2.3 Gate Dynamics

LSTMs and GRUs both rely on sigmoid gates to scale prior states. Peephole connections allow gates to observe the cell state directly. Layer normalization within gates stabilizes training for long sequences.

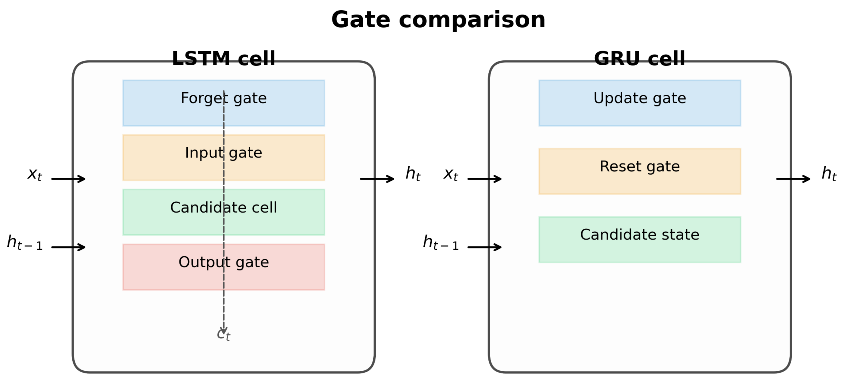

Figure 2: Gating structures of LSTM and GRU cells highlighting information flow.

3 Applications of RNNs

RNNs excel at modeling temporal and sequential data across domains. Figure **??** outlines representative pipelines.

3.1 Time-Series Forecasting

For forecasting, RNNs ingest historical measurements to predict future values. Encoder-decoder architectures with attention capture long-range dependencies. Probabilistic outputs (Gaussian mixtures, quantile losses) quantify uncertainty.

3.2 Text Generation and Language Modeling

Character- or word-level RNNs model conditional distributions $p(w_t \mid w_{<t})$. Techniques such as teacher forcing, scheduled sampling, and temperature sampling influence gener-

ation quality. Combining RNNs with embeddings and subword tokenization improves representation.

3.3 Speech and Audio Modeling

RNNs represent variable-length acoustic sequences for speech recognition or synthesis. Connectionist temporal classification (CTC) aligns RNN outputs to transcriptions without frame-level labels. In text-to-speech, autoregressive vocoders (e.g., WaveRNN) generate waveforms sample by sample.

3.4 Hybrid Architectures

Attention mechanisms and transformers build on RNNs by reweighting sequence elements. Neural ordinary differential equations (Neural ODE) and continuous-time RNNs model irregularly sampled data. Lightweight gated RNNs (SRU, QRNN) reduce sequential dependencies for faster inference.

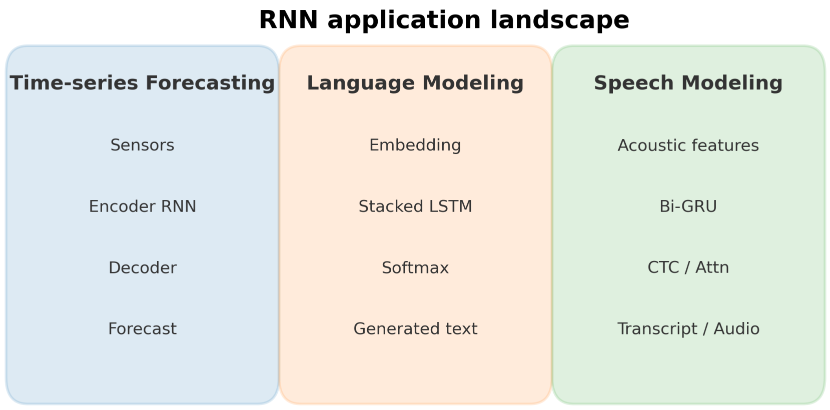

Figure 3: RNN application pipelines: forecasting, language modeling, and speech processing.

4 Practical Considerations

- **Sequence length:** Truncated BPTT balances context coverage and computational cost.
- **Regularization:** Apply dropout on inputs, recurrent dropout (variational dropout), or weight decay to prevent overfitting.
- **Initialization:** Orthogonal initialization for recurrent matrices preserves gradient norms.
- **Optimization:** Momentum SGD or Adam with gradient clipping (e.g., global norm) stabilizes training.
- **Monitoring:** Track perplexity for language models, mean absolute error for forecasting, and character error rate (CER) for speech tasks.