

CHAPTER 15

Processing Sequences Using RNNs and CNNs

Chapter 15 introduces **recurrent neural networks (RNNs)** and 1D CNNs for processing sequential data such as time series, audio, and text, and shows how to train them, handle long sequences, and forecast future values.

RNN basics and sequence mappings

RNNs add **recurrent connections** so each time step's computation depends on both the current input x_t and previous hidden state h_{t-1} , giving a form of **memory** over the sequence. A simple recurrent layer updates h_t via a nonlinearity (e.g., \tanh) applied to a linear combination of x_t and h_{t-1} , and can be stacked into **deep RNNs** when multiple recurrent layers are placed on top of each other. Different I/O setups are possible:

- **Sequence \rightarrow sequence:** output at every time step (e.g., next-step prediction).
- **Sequence \rightarrow vector:** use only the last output (e.g., sentiment analysis).
- **Vector \rightarrow sequence:** generate a sequence from a fixed vector (e.g., captioning).
- **Encoder–decoder:** sequence \rightarrow vector \rightarrow sequence, for tasks like machine translation.

Training uses **backpropagation through time (BPTT)**: unroll the RNN over time, compute a loss over some or all outputs, and backpropagate gradients along the time axis.

Time series forecasting with simple & deep RNNs

The chapter builds a synthetic time series dataset by summing two sine waves with random frequencies/phases plus noise, producing univariate sequences of length 50. Baselines for one-step-ahead forecasting:

- **Naive:** predict last value; $\text{MSE} \approx 0.020$.
- **Dense linear model** (Flatten + Dense(1)): $\text{MSE} \approx 0.004$, much better than naive.

A **simple RNN** (Single SimpleRNN(1)) on the same task performs worse than the linear model ($\text{MSE} \approx 0.014$), illustrating that naive RNNs can underperform strong classical baselines if too small or poorly designed. A **deep RNN** with several SimpleRNN(20, return_sequences=True) layers and a final recurrent or dense output improves accuracy, beating the linear baseline ($\text{MSE} \approx 0.003$) on this dataset. The chapter also covers forecasting **multiple time steps ahead** (direct multi-step outputs vs recursive predictions) and discusses handling trend/seasonality via differencing or other preprocessing if needed.

Handling long sequences and unstable gradients

RNNs suffer from **vanishing and exploding gradients** along the time dimension, limiting how far back they can effectively “remember.” The chapter presents several mitigation strategies:

- **Gradient clipping** by norm or by value in the optimizer to prevent explosions.
- **Recurrent dropout**: dropout applied to recurrent connections (with fixed masks over time) to regularize without breaking temporal structure.
- **Layer normalization** inside recurrent cells to stabilize activations across time.
- **Careful initialization** and using non-saturating activations where appropriate.

Even with these, basic RNN cells tend to have limited **short-term memory**, motivating more advanced cells.

LSTM and GRU memory cells

To extend memory, the chapter introduces **LSTM** and **GRU** cells, which add gating mechanisms to control information flow:

- **LSTM** has a cell state c_t and hidden state h_t , with input, forget, and output gates deciding what to write, erase, and expose at each time step; this architecture helps gradients flow over many more steps.
- **GRU** simplifies LSTM with fewer gates and merges cell and hidden state, often performing similarly with slightly less computation and fewer parameters.

In Keras, they are used via `keras.layers.LSTM` and `keras.layers.GRU`, with options like `return_sequences`, `dropout`, `recurrent_dropout`, and `stateful`. These cells generally outperform SimpleRNN on longer sequences and more complex tasks, including multistep forecasting.

1D CNNs for sequences and WaveNet

The chapter shows that **1D convolutional networks** (Conv1D) can process sequences efficiently and capture local temporal patterns with receptive fields that grow with depth and kernel size. Key points:

- 1D CNNs slide temporal kernels over the sequence, often combined with pooling and residual connections; they parallelize well and avoid BPTT over long horizons.
- With appropriate architecture (stacked/dilated convolutions), they can handle **very long sequences** (e.g., tens of thousands of time steps).

The chapter culminates with a simplified **WaveNet-style model** for sequence modeling and generation:

- Uses **causal** convolutions (no access to future time steps) and **dilated** convolutions (skipping time steps) to exponentially enlarge the receptive field without huge depth.

- Employs gated activations and residual/skip connections to enable deep stacks and stable training.

On the time-series task, a small 1D CNN can match or beat RNN variants, illustrating that CNNs are a strong alternative for sequence problems, especially where long-range dependencies and high throughput are important