CHAPTER 16

Natural Language Processing with RNNs and Attention

Chapter 16 covers **natural language processing (NLP)** with RNNs, attention, and Transformers, moving from character-level text generation to NMT and modern attention-only architectures.

Character RNN and text generation

The chapter starts with a **character-level RNN** trained on Shakespeare to generate fake text. Steps:

- Build a character vocabulary, map chars ↔ IDs, create overlapping windows from the text, and prepare (input_seq, target_seq) pairs shifted by one character.
- Train a stacked RNN (e.g., Embedding → GRU/LSTM → Dense over vocab with softmax), then generate text **one character at a time**, feeding each sampled output back as next input.
- Use **temperature sampling** to control randomness: low temperature → conservative, repetitive text; high temperature → diverse but error-prone text.

A **stateful RNN** variant keeps hidden state across batches/windows to better model long-range dependencies, but requires careful manual handling of resets and batch ordering.

Sentiment analysis with word embeddings and RNNs/CNNs

For **sentiment analysis** (e.g., IMDb reviews), text is processed at the word level:

- Tokenize reviews and map words to integer IDs, then use:
    - A learned **Embedding** layer, or
    - Pretrained embeddings (e.g., GloVe), loaded as a fixed or fine-tunable embedding matrix.
- Build sequence models such as:
    - 1D CNNs over word embeddings with global max pooling, or
    - RNNs (LSTM/GRU, possibly bidirectional) optionally followed by dense layers.

Keras supports **masking** via Embedding(mask_zero=True) or Masking layers so that padding tokens do not influence recurrent computations or attention.

Encoder–decoder for neural machine translation

For **NMT**, the chapter builds an **Encoder–Decoder** with RNNs:

- **Encoder**: recurrent stack (e.g., bidirectional GRUs/LSTMs) maps source sentence to a sequence of hidden states (and final state).
- **Decoder**: RNN initialized from encoder state; at each step it receives the previous target token (teacher forcing during training) and outputs next-token probabilities via a softmax layer.
- Inference uses greedy or **beam search** decoding, feeding back predicted tokens until an end-of-sequence token is produced.

Plain encoder–decoder RNNs struggle with **long sentences** due to limited memory, which motivates **attention mechanisms**.

Attention mechanisms and their uses

Attention lets the decoder **focus on specific encoder states** at each time step instead of compressing the entire source sentence into one vector:

- At decoder step t, scores $e_{t,i}$ are computed between decoder state (e.g., h_t or h_{t-1}) and each encoder output h_i, then normalized with softmax to get weights $\alpha_{t,i}$.
- The **context vector** $c_t = \sum_i \alpha_{t,i} h_i$ summarizes relevant source positions and is used along with decoder state to predict the next token.

Two classic attention types:

- **Bahdanau (additive) attention**: MLP over [decoder_state; encoder_output] then softmax; more expressive but somewhat heavier.
- **Luong (multiplicative) attention**: dot-product (or general dot-product) between encoder outputs and decoder state (often current h_t), faster and widely used.

Attention is also applied to **visual tasks** like image captioning (decoder RNN attends over CNN feature maps), providing both better performance and **explainability** via attention heatmaps.

Transformers and "Attention Is All You Need"

The chapter culminates with the **Transformer** architecture, which dispenses with recurrence and convolutions, using only attention plus feedforward layers. Core ideas:

- **Positional encodings** (sinusoidal or learned) are added to word embeddings to encode order.
- The **encoder** stack (repeated N times) has:
    - **Multi-head self-attention**: each word attends to all words in the input sentence using scaled dot-product attention in multiple heads, capturing diverse relations.

        - Position-wise feedforward network (two dense layers with ReLU), residual connections, and layer normalization.
- The **decoder** stack has:
    - Masked self-attention over previously generated target tokens (no peeking ahead).
    - Encoder–decoder attention (target words attend over encoder outputs).
    - Position-wise feedforward layers with residuals and normalization.

Training uses full input–output sentence pairs; inference generates tokens step-by-step, typically with beam search. Transformers achieved state-of-the-art NMT with **better quality, faster training, and more parallelism** than RNN-based models, and became the foundation for large language models and modern NLP