Encoder–Decoder with Attention in TensorFlow (Hands-on Practical)

This practical guides you through building and training a simple Encoder–Decoder (Seq2Seq) model with Attention in TensorFlow/Keras. You’ ll implement a small synthetic sequence task (reverse a character sequence) so it runs fast on CPU and lets you focus on the mechanics of attention. You’ ll also find a breakdown of the key parameters, and ideas for other practicals to try.

What you’ ll do:

1. Build a character-level dataset (synthetic) for sequence reversal.
2. Implement an encoder (GRU) and a decoder (GRU) with Additive (Bahdanau) attention.
3. Train with teacher forcing and greedy-decode at inference time.
4. Inspect shapes and learn the purpose of each parameter.

Files

- practical_seq2seq_attention_tf.py – Minimal runnable script with dataset, model, training, and inference.
- README.md – This tutorial and parameter explanations.

1. Quick start

- Requirements: TensorFlow 2.x (CPU is fine). No extra packages needed.
- Run: python3 practical_seq2seq_attention_tf.py
- Expected: Training loss decreasing, and sample predictions printed (model learns to reverse strings like “abcde” -> “edcba”).

1. Architecture overview

- Encoder: Embedding -> GRU produces a sequence of hidden states (one per source time step) and a final state.
- Attention (Bahdanau/Additive): At each decoder step, compute a context vector as a weighted sum over encoder outputs using current decoder state as query.
- Decoder: At each step, embed previous target token, combine with context vector, process with GRU to produce next hidden state and a vocabulary distribution.
- Training: Teacher forcing – feed the true previous token to predict the next token.
- Inference: Start with [START], repeatedly predict next token until [END] or max length.

1. Key parameters and what they do

- vocab_size: Size of the target/source symbol set including special tokens ([PAD], [START], [END]). Affects embedding and final softmax sizes.
- embedding_dim: Size of the learned dense vector for each token. Larger can capture more semantics, but slower and more prone to overfit.
- units (a.k.a. hidden size): Number of GRU units for encoder/decoder, and the attention hidden size (using AdditiveAttention aligns to GRU size). Larger improves capacity but increases compute.
- max_src_len / max_tgt_len: Maximum time steps. Longer sequences need more compute and memory.
- batch_size: Number of sequences processed together. Larger is faster but uses more memory.
- learning_rate: Step size for optimizer (Adam). Too high diverges; too low slows learning.
- teacher forcing ratio (implicit here 100% during training): How often you feed the ground-truth previous token vs. the model’ s previous prediction.

1. Training loop explained (high level)
   For each batch:

- Encode source tokens: enc_outputs (batch, src_len, units), enc_state (batch, units)
- Initialize dec_state = enc_state and dec_input = [START] (batch,)
- For each target time step t:
  - Compute attention(context) from dec_state over enc_outputs
  - Concatenate context with embedding(dec_input)
  - Run decoder GRU one step to get new dec_state and output
  - Project to logits (vocab) and compute loss against target[t]
  - Set dec_input = target[t] (teacher forcing)

1. Practical variations you can try (types of practicals)

- Neural Machine Translation (word-level) on small spa↔eng dataset (like the official TF tutorial). Swap synthetic data for real text, increase vocab, and train longer.
- Text Summarization: Use sentences/paragraphs as input and shorter target summaries; identical architecture.
- Image Captioning: CNN encoder (e.g., ResNet features per image region) + attention + RNN decoder.
- Speech Recognition (toy): MFCC feature frames as encoder input, character decoder with attention.

- Date Normalization / Slot Filling: Convert textual dates (“March 5, 2021”) to canonical format (“2021-03-05”) or extract entities.
- Copy/Sort/Reverse: Synthetic tasks to validate your implementation quickly before moving to real data.

1. Glossary of important tensors

- enc_inputs: (batch, src_len) source token IDs
- enc_outputs: (batch, src_len, units) sequence of encoder hidden states
- enc_state: (batch, units) final encoder hidden state
- dec_input (per step): (batch,) single token ID (previous target)
- context: (batch, units) attention-weighted sum of enc_outputs
- dec_state: (batch, units) current decoder hidden state
- logits: (batch, vocab_size) unnormalized scores for next token

1. Tips and pitfalls

- Shape mismatches usually come from mixing per-step (batch,) tokens with per-sequence (batch, T) tensors. For embedding and decoder GRU, pass a single-step input as shape (batch, 1, feat).
- Mask padding tokens when computing loss to avoid penalizing the model for padded positions.
- Start simple (synthetic) before training on real datasets; iterate on shapes and correctness first.
- For Luong attention (dot/general), you can replace AdditiveAttention with a dot-product mechanism.

1. What next

- Scale up: Use wordpieces (TextVectorization) and a parallel text dataset.
- Experiments: Vary embedding_dim, units, optimizer, and teacher forcing.
- Visualization: Save attention weights per step to plot heatmaps (decoder step × source positions).
- Transition to Transformers: Replace GRU with self-attention blocks.

If you want, I can also produce a Jupyter notebook version of this tutorial.