Attention-Based Architectures: Recurrent Encoder–Decoder

Handout Notes - Generative and Deep Learning (GDL)

Davide Bacciu - University of Pisa

Notation. An input sequence is denoted by $\mathbf{x} = (x_1, \ldots, x_n)$ and an output sequence by $\mathbf{y} = (y_1, \ldots, y_m)$, where input and output lengths need not coincide. Encoder hidden states are denoted by $h_1, \ldots, h_n$, decoder hidden states by $s_1, \ldots, s_m$. A global context vector is denoted by c; when attention is used, the time-dependent context is written c_t. Attention weights are $\alpha_{t,i}$, where i indexes encoder positions and t decoder steps. Alignment scores are denoted by $e_{t,i}$. Model parameters are generically denoted by θ. A dataset is denoted by $\mathcal{D}$ with size $N = |\mathcal{D}|$.

1 Why sequence-to-sequence learning is harder than standard sequence modeling

Many recurrent models studied earlier solve tasks in which the output is either a single item or a sequence aligned rather closely with the input. However, many important problems have a more general structure:

> *the input is a sequence, the output is also a sequence, and the two may have different lengths and different internal alignments.*

This is the sequence-to-sequence setting.

Machine translation is the classical example. Given an input sentence

$$x_1, \ldots, x_n,$$

we want to generate an output sentence

$$y_1, \ldots, y_m,$$

where in general $m \neq n$. The output is not a simple pointwise transformation of the input. It is a structured object whose generation depends on both local choices and global context.

This creates two major challenges:

- the model must compress or access the relevant information from a variable-length input;
- the model must generate a variable-length structured output, one symbol at a time, while conditioning on both the past output and the input representation.

A plain unfolded RNN with "blank" outputs and inputs is not a satisfactory solution. Instead, one needs an explicit mechanism for separating *encoding* of the source from *generation* of the target.

2 Encoder–decoder principle

The encoder–decoder architecture is the canonical solution to sequence transduction with recurrent models. It separates computation into two stages.

2.1 Encoder

The encoder reads the input sequence

$$x_1, \dots, x_n$$

and transforms it into hidden states

$$h_1, \dots, h_n.$$

In the simplest form, the encoder then summarizes the entire input into a fixed-size vector

$$c.$$

Historically, one common choice was

$$c = h_n,$$

the final hidden state of the encoder.

2.2 Decoder

The decoder is another recurrent network that generates the output sequence autoregressively. At step t, it updates its internal state using:

- the previous decoder state,
- the previous output token,
- the context information extracted from the input.

It then predicts the next output token.

Thus the decoder models a conditional distribution of the form

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{m} p(y_t \mid y_{<t}, \mathbf{x}),$$

where

$$y_{<t} = (y_1, \dots, y_{t-1}).$$

This autoregressive factorization is central: it turns structured prediction into a sequence of conditional one-step predictions.

3 Basic recurrent encoder–decoder

A simple recurrent encoder may be written as

$$h_t = f_{\text{enc}}(h_{t-1}, x_t), \qquad t = 1, \dots, n,$$

and the simplest context vector is

$$c = h_n.$$

The decoder then evolves as

$$s_t = f_{\text{dec}}(s_{t-1}, y_{t-1}, c),$$

with output distribution

$$p(y_t \mid y_{<t}, \mathbf{x}) = p(y_t \mid s_t).$$

In practice, the decoder is often initialized from the context vector or receives it as an additional input at every step. A conceptually clean decoder recurrence is therefore

$$s_t = f_{\text{dec}}(s_{t-1}, y_{t-1}, c).$$

This makes clear that the decoder is conditioned both on the output history and on an input-derived summary.

Worked example — Autoregressive decoding in an encoder–decoder

Suppose the encoder has already produced a context vector c from the source sentence. The decoder begins from an initial state s_0 (or from c directly) and a start-of-sequence symbol $\langle \text{bos} \rangle$. Then:

$$s_1 = f_{\text{dec}}(s_0, \langle \text{bos} \rangle, c), \qquad p(y_1 \mid \mathbf{x}) = p(y_1 \mid s_1).$$

After generating or observing y_1, the next step is

$$s_2 = f_{\text{dec}}(s_1, y_1, c), \qquad p(y_2 \mid y_1, \mathbf{x}) = p(y_2 \mid s_2).$$

Continuing in this way yields

$$p(\mathbf{y} \mid \mathbf{x}) = p(y_1 \mid \mathbf{x})\, p(y_2 \mid y_1, \mathbf{x}) \cdots p(y_m \mid y_{<m}, \mathbf{x}).$$

Thus the decoder implements a conditional language model whose state is informed by the encoded input.

4 Teacher forcing and autoregressive generation

During training, it is common to use *teacher forcing*. Instead of feeding the decoder its own previous prediction, we feed it the true previous target token. That is, at training time the decoder update uses

$$y_{t-1}^{\text{true}}$$

rather than the sampled or argmax prediction from the previous step.

This is helpful because it stabilizes training and gives the model direct access to the correct output history. However, at prediction time the target sequence is not known, so the model must feed back its own generated tokens. This creates the usual train–test discrepancy of autoregressive sequence models.

5 The bottleneck of fixed-length context

The basic encoder–decoder architecture has a serious limitation. If the whole input sequence must be compressed into a single fixed-size vector c, then the decoder has access only to that compressed summary.

This creates a *representation bottleneck*:

- the encoder must pack all relevant information into one vector,
- the decoder must recover all needed source-side detail from that same vector,
- longer or more complex inputs are harder to summarize adequately.

There is also a directional bias. If $c = h_n$, then the representation is naturally influenced most strongly by the latest encoder states. Even in gated recurrent encoders, distant source positions can be harder to preserve faithfully in a single summary vector.

For tasks like translation, this is particularly problematic. Different target words often need to focus on different parts of the input. A single global summary is too crude.

6 Why attention is needed

The core motivation for attention is simple:

When generating the next output symbol, the decoder should be able to look back at the most relevant parts of the input, rather than rely only on one fixed global vector.

Instead of compressing the source into one vector once and for all, attention lets the decoder compute a *dynamic context* at each output step. This context is built from the entire set of encoder states

$$h_1, \ldots, h_n,$$

but weighted according to their relevance to the current decoder state.

This is why attention is often described as a soft alignment mechanism. For each decoder step t, the model decides which source positions matter most for predicting y_t.

7 Soft attention: black-box view

At decoder step t, suppose the decoder has a current state s_t. The encoder provides a collection of source-side representations

$$h_1, \ldots, h_n.$$

An attention module takes these as input and returns a context vector

$$c_t,$$

which is then used together with s_t to predict the next output.

At a high level:

$$(h_1, \ldots, h_n,\ s_t) \quad \longrightarrow \quad c_t.$$

The context c_t is not arbitrary. It is a learned weighted combination of encoder states. The weights tell us how much attention the decoder gives to each source position at that step.

8 Soft attention: score, normalization, aggregation

Soft attention is most naturally described in three stages.

8.1 Score computation

For each encoder state h_i and current decoder state s_t, compute a scalar relevance score

$$e_{t,i} = a(s_t, h_i),$$

where $a(\cdot, \cdot)$ is a trainable compatibility function.

A common classical choice is additive (Bahdanau-style) attention:

$$e_{t,i} = v^\top \tanh(W_s s_t + W_h h_i + b),$$

where v, W_s, W_h, and b are learned parameters.

8.2 Normalization

Convert scores into attention weights using a softmax:

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_{j=1}^{n} \exp(e_{t,j})}.$$

Thus

$$\alpha_{t,i} \geq 0, \qquad \sum_{i=1}^{n} \alpha_{t,i} = 1.$$

The weights therefore form a probability distribution over source positions.

8.3 Aggregation

The context vector is a weighted sum of encoder states:

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i.$$

Hence the decoder receives a source summary specialized to the current output step.

Worked example — Why soft attention is a differentiable selector

Suppose at decoder step t the alignment scores are

$$e_{t,1} = 0, \qquad e_{t,2} = 2, \qquad e_{t,3} = 1.$$

Then the attention weights are

$$\alpha_{t,1} = \frac{e^0}{e^0 + e^2 + e^1}, \qquad \alpha_{t,2} = \frac{e^2}{e^0 + e^2 + e^1}, \qquad \alpha_{t,3} = \frac{e^1}{e^0 + e^2 + e^1}.$$

Numerically,

$$e^0 = 1, \qquad e^1 \approx 2.718, \qquad e^2 \approx 7.389,$$

so

$$\alpha_{t,1} \approx 0.09, \qquad \alpha_{t,2} \approx 0.67, \qquad \alpha_{t,3} \approx 0.24.$$

Thus the context vector

$$c_t = 0.09\, h_1 + 0.67\, h_2 + 0.24\, h_3$$

focuses primarily on h_2, but still retains some contribution from the others. This is why soft attention can be interpreted as a differentiable version of "choosing where to look."

9 Cross-attention in sequence-to-sequence models

In sequence transduction, the attention just described is more precisely *cross-attention*. The reason is that it relates two different sequences:

- the source-side sequence of encoder states $\{h_i\}_{i=1}^{n}$,
- the target-side decoder state s_t at each output step.

The decoder state acts as a query: it asks which source positions are most relevant right now. The encoder states serve as the items to be scored and aggregated.

In the classical recurrent encoder–decoder with attention, the decoder update can be written conceptually as

$$s_t = f_{\text{dec}}(s_{t-1}, y_{t-1}, c_t),$$

where

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$$

and

$$\alpha_{t,i} = \text{softmax}_i\big(a(s_{t-1}, h_i)\big)$$

or a closely related variant depending on the specific implementation. The essential point is unchanged: the decoder no longer depends on one frozen source summary, but on a dynamically recomputed context.

10 Attention as learned alignment

One of the most important conceptual interpretations of attention is that it learns an approximate alignment between source and target positions. When translating a sentence, for instance, the model often assigns high weight to the source word or phrase most relevant to the current target word.

This can be visualized as a matrix of weights:

$$A = [\alpha_{t,i}] \in \mathbb{R}^{m \times n},$$

where rows correspond to target positions and columns to source positions. Each row sums to one. If a row has a sharp peak, the decoder is focusing strongly on a single source position. If it is diffuse, the decoder is combining several source elements.

This alignment interpretation is one reason attention became so influential: it improves predictive performance while also giving a degree of interpretability.

Worked example — Attention matrix interpretation in translation

Suppose the source sentence has four tokens and the target sentence has five tokens. Then the attention matrix is

$$A = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} \\ \alpha_{5,1} & \alpha_{5,2} & \alpha_{5,3} & \alpha_{5,4} \end{bmatrix}.$$

If row $t = 3$ places most of its mass on column $i = 2$, then the third target word is generated mainly by attending to the second source word. If many rows follow a roughly diagonal pattern, this suggests monotone alignment between source and target. More irregular patterns indicate reordering, which is common in translation.

11 Hard attention

Soft attention produces a weighted average over all encoder states. An alternative is *hard attention*, where the model selects one encoder state stochastically according to the attention distribution.

Instead of computing

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i,$$

hard attention samples an index

$$I_t \sim \text{Categorical}(\alpha_{t,1}, \ldots, \alpha_{t,n}),$$

and then sets

$$c_t = h_{I_t}.$$

This is a sharper form of focus, but it comes at a cost. Because sampling is discrete, the resulting mechanism is no longer straightforwardly differentiable. Standard backpropagation cannot be applied through the sampling operation without additional estimators or relaxations.

For this reason, soft attention became far more widely used in mainstream neural sequence models: it is fully differentiable and easy to train end-to-end.

12 Local attention

Local attention is a compromise between soft and hard attention. Instead of attending over the entire source sequence, the model predicts a target-dependent location or neighborhood and performs soft attention only within that local window.

This has two motivations:

- it reduces computational cost relative to full attention,
- it approximates the behavior of hard focus while remaining differentiable.

Conceptually, local attention introduces an alignment signal p_t that predicts where the decoder should look, and then restricts or biases the softmax to encoder positions near p_t.

This makes local attention a useful bridge: it preserves differentiability like soft attention, but enforces more localized focus patterns.

13 Generalizing attention score functions

The attention mechanism described above depends critically on the compatibility function

$$a(s_t, h_i).$$

Different choices of this function lead to different attention variants.

A widely used taxonomy includes:

- **content-based / cosine-style scoring**, based on vector similarity,
- **additive attention**, e.g.

$$a(s_t, h_i) = v^\top \tanh(W_s s_t + W_h h_i + b),$$

- **general bilinear attention**,

$$a(s_t, h_i) = s_t^\top W_a h_i,$$

- **dot-product attention**,

$$a(s_t, h_i) = s_t^\top h_i,$$

- **scaled dot-product attention**,

$$a(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{d}},$$

where d is the dimensionality of the compared vectors.

In recurrent encoder–decoder models, additive attention played a historically central role. Dot-product and scaled dot-product scoring later became especially important in Transformer architectures.

14 Training the recurrent encoder–decoder with attention

Let $\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^N$ be a dataset of paired input-output sequences. The model defines a conditional distribution

$$p(\mathbf{y} \mid \mathbf{x}; \theta) = \prod_{t=1}^{m} p(y_t \mid y_{<t}, \mathbf{x}; \theta).$$

Training is performed by maximizing conditional log-likelihood:

$$\max_\theta \sum_{n=1}^{N} \log p(\mathbf{y}^{(n)} \mid \mathbf{x}^{(n)}; \theta).$$

Equivalently, we minimize the negative log-likelihood:

$$\mathcal{L}(\theta) = -\sum_{n=1}^{N}\sum_{t=1}^{m_n} \log p\big(y_t^{(n)} \mid y_{<t}^{(n)}, \mathbf{x}^{(n)}; \theta\big).$$

Because soft attention is differentiable, all parameters of the encoder, attention module, and decoder can be trained end-to-end by backpropagation through time.

Worked example — Why soft attention is easy to train end-to-end

The attention weights are

$$\alpha_{t,i} = \frac{\exp(e_{t,i})}{\sum_j \exp(e_{t,j})},$$

and the context vector is

$$c_t = \sum_i \alpha_{t,i} h_i.$$

Both operations are differentiable:

- the softmax is differentiable with respect to the scores $e_{t,i}$,
- the weighted sum is differentiable with respect to both the weights $\alpha_{t,i}$ and the encoder states h_i.

Therefore, when the loss gradient reaches c_t, it can flow backward:

- into the attention scores and their parameters,
- into the decoder state that produced the query,
- into all encoder states h_i.

This is the main optimization advantage of soft attention over hard attention.

15 Attention beyond text: heterogeneous encoder–decoder models

A major strength of the encoder–decoder framework is that the encoder and decoder need not be of the same type. This makes the architecture highly compositional.

For example:

- a convolutional network can encode an image into a set of regional feature vectors,
- an RNN or LSTM decoder can then generate a caption word by word,
- attention selects which image regions are most relevant for each generated word.

In this setting, the encoder outputs are no longer temporal hidden states from a source sentence, but spatial features from image regions. The attention mechanism remains conceptually the same:

$$c_t = \sum_i \alpha_{t,i} h_i,$$

but now the h_i are visual-region encodings instead of source-word encodings.

This is why the encoder–decoder abstraction is so important. It is not limited to homogeneous sequence models. It provides a general interface for combining different model families and data modalities.

16 Interpretability of attention

Attention maps are often useful for interpretation. In translation, they can reveal approximate word alignments. In image captioning, they can highlight which image regions influenced each generated word.

This does not mean attention is a perfect explanation of the model's reasoning. However, it does provide a structured signal that is often much more informative than a raw hidden-state vector.

In practice, attention visualizations help:

- understand what the model is focusing on,
- diagnose errors,
- inspect whether learned alignments are sensible,
- compare different attention mechanisms.

17 From recurrent attention to Transformers

The recurrent encoder–decoder with attention is historically important because it introduced the core idea that changed modern sequence modeling:

> *instead of compressing everything into one vector, let the model compute dynamic context by weighted interaction among representations.*

In the recurrent seq2seq setting, this appears as cross-attention from decoder states to encoder states. In the Transformer, the idea is generalized and extended:

- self-attention replaces recurrence as the main representation-building mechanism,
- cross-attention remains crucial in encoder–decoder Transformers,
- scaled dot-product attention becomes the standard compatibility form.

Thus recurrent attention-based encoder–decoder models are best understood as the conceptual precursor to the Transformer era.

18 Conceptual summary

The main ideas of recurrent encoder–decoder attention models can be summarized as follows:

- sequence-to-sequence problems require mapping variable-length structured inputs to variable-length structured outputs;
- the encoder–decoder architecture separates source encoding from target generation;
- a fixed-length context vector creates a bottleneck, especially for long or complex inputs;
- attention removes this bottleneck by building a target-step-specific context vector from all encoder states;
- soft attention works by *scoring*, *normalizing*, and *aggregating* encoder representations;
- in sequence transduction this is cross-attention, because decoder states attend over encoder states;
- soft attention is fully differentiable and trainable end-to-end, whereas hard attention involves stochastic selection;
- the encoder–decoder perspective generalizes beyond text, enabling heterogeneous multimodal architectures such as image captioning.

More broadly, attention-based recurrent encoder–decoders mark the transition from purely compressed-sequence representations toward dynamically routed information access. That transition set the stage for the next major architectural leap: the Transformer.