Sequence Models II: Encoder–Decoder, Teacher Forcing and Attention

Mapping one sequence to another — and letting the decoder look back

Prof. Nipun Batra

ES 667 · Deep Learning · IIT Gandhinagar

From one sequence to another

Where we are

The last three lectures modeled **one** sequence — next-token prediction over $x_{1:T}$:

Where we are

The last three lectures modeled **one** sequence — next-token prediction over $x_{1:T}$:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

Where we are

The last three lectures modeled **one** sequence — next-token prediction over $x_{1:T}$:

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \mid x_{<t})$$

This lecture changes the **task**: map an **input** sequence to a **different output** sequence — translation, summarization, speech. Input and output can even have **different lengths**.

Tasks that map sequence → sequence

V VISUAL

Many problems are $x_{1:T} \to y_{1:U}$ — a source sequence in, a target sequence out:

Tasks that map sequence → sequence

V VISUAL

Many problems are $x_{1:T} \to y_{1:U}$ — a source sequence in, a target sequence out:

Task	**Input $x_{1:T}$**	**Output $y_{1:U}$**
translation	English sentence	French sentence
summarization	a document	a short summary
speech recognition	audio frames	a transcript
image captioning	image regions	a caption
dialogue	conversation so far	the next reply
forecasting	past observations	future horizon

Tasks that map sequence → sequence

V VISUAL

Many problems are $x_{1:T} \to y_{1:U}$ — a source sequence in, a target sequence out:

Task	**Input $x_{1:T}$**	**Output $y_{1:U}$**
translation	English sentence	French sentence
summarization	a document	a short summary
speech recognition	audio frames	a transcript
image captioning	image regions	a caption
dialogue	conversation so far	the next reply
forecasting	past observations	future horizon

one general shape — **sequence in, sequence out** — behind all of them

The lengths differ

D DERIVATION

The defining difficulty: the output length U is **not** the input length T, and is **not known in advance**.

The lengths differ

D DERIVATION

The defining difficulty: the output length U is **not** the input length T, and is **not known in advance**.

"I like cats" $(T = 3)$ $\rightarrow$ "j'aime les chats" $(U = 3)$

The lengths differ

D DERIVATION

The defining difficulty: the output length U is **not** the input length T, and is **not known in advance**.

"I like cats" $(T = 3)$ $\rightarrow$ "j'aime les chats" $(U = 3)$

A token classifier ($x_{1:T} \rightarrow y_{1:T}$, one label per input) **cannot** do this: it forces $U = T$ and a fixed alignment. We need a model that **generates** an output of **flexible** length.

The conditional sequence model

D DERIVATION

Keep the autoregressive factorization — but now **condition on the whole source** $x_{1:T}$:

The conditional sequence model

D DERIVATION

Keep the autoregressive factorization — but now **condition on the whole source** $x_{1:T}$:

$$p(y_{1:U} \mid x_{1:T}) = \prod_{u=1}^{U} p(y_u \mid y_{<u}, x_{1:T})$$

The conditional sequence model

D DERIVATION

Keep the autoregressive factorization — but now **condition on the whole source** $x_{1:T}$:

$$p(y_{1:U} \mid x_{1:T}) = \prod_{u=1}^{U} p(y_u \mid y_{<u}, x_{1:T})$$

autoregressive in the target: each y_u depends on the tokens generated **before** it.

conditioned on the source: every factor also sees the **entire input** $x_{1:T}$.

The conditional sequence model

D DERIVATION

Keep the autoregressive factorization — but now **condition on the whole source** $x_{1:T}$:

$$p(y_{1:U} \mid x_{1:T}) = \prod_{u=1}^{U} p(y_u \mid y_{<u}, x_{1:T})$$

autoregressive in the target: each y_u depends on the tokens generated **before** it.

conditioned on the source: every factor also sees the **entire input** $x_{1:T}$.

same next-token idea as Lecture 11 — now with a **source** to condition on

The encoder–decoder

Two networks, one bridge

Split the job in two (Sutskever, Vinyals & Le 2014; Cho et al. 2014):

Two networks, one bridge

Split the job in two (Sutskever, Vinyals & Le 2014; Cho et al. 2014):

Encoder — reads the source $x_{1:T}$ and compresses it into a **context vector** c.

Decoder — a conditional language model that **generates** $y_{1:U}$ from c.

Two networks, one bridge

Split the job in two (Sutskever, Vinyals & Le 2014; Cho et al. 2014):

Encoder — reads the source $x_{1:T}$ and compresses it into a **context vector** c.

Decoder — a conditional language model that **generates** $y_{1:U}$ from c.

the context c is the only bridge — everything the decoder knows about the source is in c

The encoder

D DERIVATION

The encoder is an RNN (LSTM/GRU) run over the source; its **final state** is the context:

The encoder

D DERIVATION

The encoder is an RNN (LSTM/GRU) run over the source; its **final state** is the context:

$$h_t = f_{\text{enc}}(h_{t-1}, e(x_t)), \qquad t = 1, \ldots, T$$

The encoder

D DERIVATION

The encoder is an RNN (LSTM/GRU) run over the source; its **final state** is the context:

$$h_t = f_{\text{enc}}(h_{t-1}, e(x_t)), \qquad t = 1, \ldots, T$$

$$c = h_T \qquad \text{(the whole source, summarized in one vector)}$$

The encoder

D DERIVATION

The encoder is an RNN (LSTM/GRU) run over the source; its **final state** is the context:

$$h_t = f_{\text{enc}}(h_{t-1}, e(x_t)), \qquad t = 1, \ldots, T$$

$$c = h_T \qquad \text{(the whole source, summarized in one vector)}$$

read the source left to right; h_T is a fixed-size summary of **all** of it

The decoder

D DERIVATION

A second RNN, **initialized from c**, generates the target one token at a time:

The decoder

D DERIVATION

A second RNN, **initialized from c**, generates the target one token at a time:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c), \qquad s_0 = c$$

The decoder

D DERIVATION

A second RNN, **initialized from c**, generates the target one token at a time:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c), \qquad s_0 = c$$

$$p(y_u \mid y_{<u}, x_{1:T}) = \text{softmax}(W_o\, s_u + b_o)$$

The decoder

D DERIVATION

A second RNN, **initialized from c**, generates the target one token at a time:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c), \qquad s_0 = c$$

$$p(y_u \mid y_{<u}, x_{1:T}) = \text{softmax}(W_o\, s_u + b_o)$$

the decoder is a language model over the target, **conditioned** on the source through c

The architecture

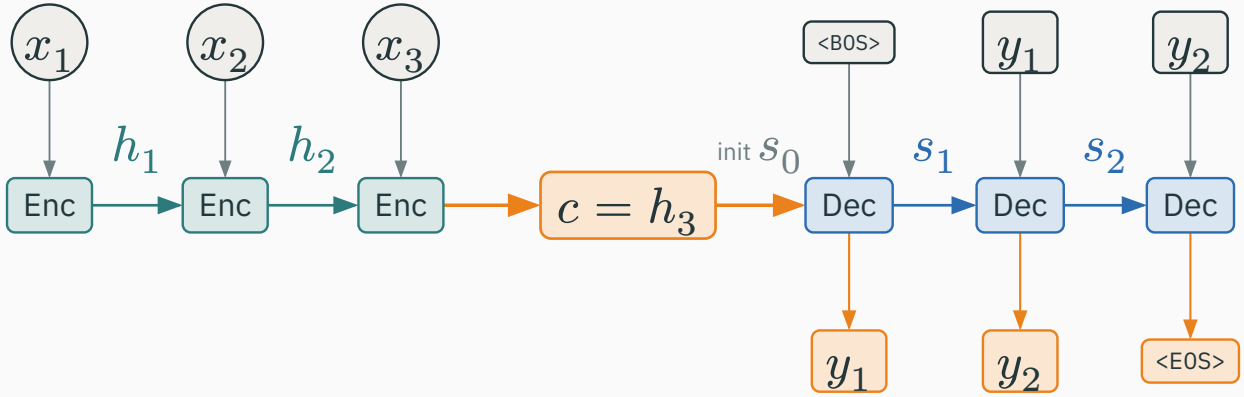
V VISUAL

Encode the source into c; hand c to the decoder; generate the target autoregressively:

The architecture

V VISUAL

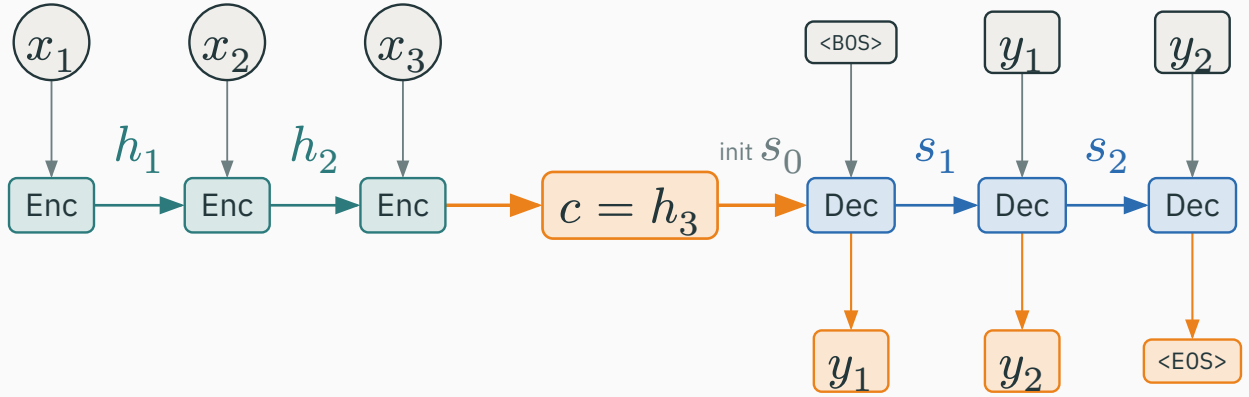
Encode the source into c; hand c to the decoder; generate the target autoregressively:



The architecture

V VISUAL

Encode the source into c; hand c to the decoder; generate the target autoregressively:



<BOS> starts generation; each output feeds the next step; <EOS> stops it

Start and end tokens

D DERIVATION

Two special tokens make **variable-length** generation work:

Start and end tokens

D DERIVATION

Two special tokens make **variable-length** generation work:

<BOS> — the decoder's **first input**, so it can produce y_1 with no target history.

<EOS> — an ordinary vocabulary token; **emitting it** is how the model **chooses to stop**.

Start and end tokens

D DERIVATION

Two special tokens make **variable-length** generation work:

<BOS> — the decoder's **first input**, so it can produce y_1 with no target history.

<EOS> — an ordinary vocabulary token; **emitting it** is how the model **chooses to stop**.

Training a pair shifts the target by one

Start and end tokens

D DERIVATION

input vs **target** of the decoder:

decoder input	<BOS>	y_1	y_2	y_3
decoder target	y_1	y_2	y_3	<EOS>

The training loss

D DERIVATION

Maximize the conditional log-likelihood — sum token cross-entropy over the target:

The training loss

D DERIVATION

Maximize the conditional log-likelihood — sum token cross-entropy over the target:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p(y_u \mid y_{<u}, x_{1:T})$$

The training loss

D DERIVATION

Maximize the conditional log-likelihood — sum token cross-entropy over the target:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p(y_u \mid y_{<u}, x_{1:T})$$

Batches mix target lengths, so pad and **mask** the padded positions out:

$$\mathcal{L} = \frac{\sum_{i,u} m_{iu} \, (-\log p_{iu})}{\sum_{i,u} m_{iu}}, \qquad m_{iu} \in \{0, 1\}$$

The training loss

D DERIVATION

Maximize the conditional log-likelihood — sum token cross-entropy over the target:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p(y_u \mid y_{<u}, x_{1:T})$$

Batches mix target lengths, so pad and **mask** the padded positions out:

$$\mathcal{L} = \frac{\sum_{i,u} m_{iu}\,(-\log p_{iu})}{\sum_{i,u} m_{iu}}, \qquad m_{iu} \in \{0, 1\}$$

$m_{iu} = 0$ on <PAD> — padding is invisible to the gradient

Teacher forcing

Feeding the true previous token

D DERIVATION

At each decoder step the input is the **previous target token**. Which one — the **true** one, or the model's **guess**?

Feeding the true previous token

D DERIVATION

At each decoder step the input is the **previous target token**. Which one — the **true** one, or the model's **guess**?

Teacher forcing: during training, feed the ground-truth y_{u-1}^{true}, not the prediction $\hat{y}_{u-1}$.

Feeding the true previous token

D DERIVATION

At each decoder step the input is the **previous target token**. Which one — the **true** one, or the model's **guess**?

Teacher forcing: during training, feed the ground-truth y_{u-1}^{true}, not the prediction $\hat{y}_{u-1}$.

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}^{\text{true}}), c) \qquad \text{(training)}$$

Why teacher forcing

D DERIVATION

Why teacher forcing

D DERIVATION

Stable, parallel supervision — every step gets a **correct** input, so all U targets can be scored in one pass.

Without it — an early wrong token becomes the **next** step's input, and the error **compounds** down the sequence.

Why teacher forcing

D DERIVATION

Stable, parallel supervision — every step gets a **correct** input, so all U targets can be scored in one pass.

Without it — an early wrong token becomes the **next** step's input, and the error **compounds** down the sequence.

teacher forcing decouples the steps: each is trained as a clean next-token classification

Train ≠ inference: exposure bias

D DERIVATION

The catch — the two regimes feed **different** previous tokens:

Train ≠ inference: exposure bias

D DERIVATION

The catch — the two regimes feed **different** previous tokens:

train: $s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}^{\text{true}}), c)$

infer: $s_u = f_{\text{dec}}(s_{u-1}, e(\hat{y}_{u-1}), c)$

Train ≠ inference: exposure bias

D DERIVATION

The catch — the two regimes feed **different** previous tokens:

train: $s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}^{\text{true}}), c)$

infer: $s_u = f_{\text{dec}}(s_{u-1}, e(\hat{y}_{u-1}), c)$

Exposure bias: the model only ever saw **true** histories in training, but at inference it must consume its **own** (sometimes wrong) outputs — a distribution it was never trained on.

Scheduled sampling

D DERIVATION

★ OPTIONAL

A middle ground (Bengio et al. 2015): **sometimes** feed the model's own prediction during training.

Scheduled sampling

D DERIVATION ★ OPTIONAL

A middle ground (Bengio et al. 2015): **sometimes** feed the model's own prediction during training.

$$\text{decoder input at step } u = \begin{cases} y_{u-1}^{\text{true}} & \text{with prob. } p \\ \hat{y}_{u-1} & \text{with prob. } 1-p \end{cases}$$

Scheduled sampling

D DERIVATION ★ OPTIONAL

A middle ground (Bengio et al. 2015): **sometimes** feed the model's own prediction during training.

$$\text{decoder input at step } u = \begin{cases} y_{u-1}^{\text{true}} & \text{with prob. } p \\ \hat{y}_{u-1} & \text{with prob. } 1-p \end{cases}$$

anneal $p : 1 \to 0$ over training — start fully teacher-forced, gradually expose the model to its own outputs

A worked seq2seq pass

The toy translation

Translate “I like cats” $\rightarrow$ “j’aime les chats”. Set up the source and the shifted target:

The toy translation

Translate “I like cats” $\rightarrow$ “j’aime les chats”. Set up the source and the shifted target:

$x = [\texttt{I}, \texttt{like}, \texttt{cats}, \texttt{<EOS>}]$

The toy translation

Translate “I like cats” $\rightarrow$ “j’aime les chats”. Set up the source and the shifted target:

$x = [\texttt{I}, \texttt{like}, \texttt{cats}, \texttt{<EOS>}]$

$\text{decoder input} = [\texttt{<BOS>}, \texttt{j'aime}, \texttt{les}, \texttt{chats}]$

$\text{decoder target} = [\texttt{j'aime}, \texttt{les}, \texttt{chats}, \texttt{<EOS>}]$

The toy translation

Translate “I like cats” $\rightarrow$ “j’aime les chats”. Set up the source and the shifted target:

$x = [\texttt{I}, \texttt{like}, \texttt{cats}, \texttt{<EOS>}]$

$\text{decoder input} = [\texttt{<BOS>}, \texttt{j'aime}, \texttt{les}, \texttt{chats}]$

$\text{decoder target} = [\texttt{j'aime}, \texttt{les}, \texttt{chats}, \texttt{<EOS>}]$

the target is the input shifted by one — teacher forcing supplies the true tokens

Encode, then start decoding

D DERIVATION

Run the encoder over the four source tokens; take the final state as the context:

Encode, then start decoding

D DERIVATION

Run the encoder over the four source tokens; take the final state as the context:

$$h_1, h_2, h_3, h_4 = f_{\text{enc}(\ldots)}, \qquad c = h_4$$

Encode, then start decoding

D DERIVATION

Run the encoder over the four source tokens; take the final state as the context:

$$h_1, h_2, h_3, h_4 = f_{\text{enc}(\ldots)}, \qquad c = h_4$$

Initialize the decoder and take its **first** step from <BOS>:

$$s_1 = f_{\text{dec}}(s_0, e(\text{<BOS>}), c), \qquad s_0 = c$$

Encode, then start decoding

D DERIVATION

Run the encoder over the four source tokens; take the final state as the context:

$$h_1, h_2, h_3, h_4 = f_{\text{enc}(\ldots)}, \qquad c = h_4$$

Initialize the decoder and take its **first** step from <BOS>:

$$s_1 = f_{\text{dec}}(s_0, e(\texttt{<BOS>}), c), \qquad s_0 = c$$

$$p_1 = \text{softmax}(W_o\, s_1 + b_o) \qquad \text{(a distribution over the French vocabulary)}$$

Step 1 loss

D DERIVATION

The target for step 1 is `j'aime`; the loss is its negative log-probability:

Step 1 loss

D DERIVATION

The target for step 1 is `j'aime`; the loss is its negative log-probability:

$$\mathcal{L}_1 = -\log p_1(\texttt{j'aime})$$

Step 1 loss

D DERIVATION

The target for step 1 is `j'aime`; the loss is its negative log-probability:

$$\mathcal{L}_1 = -\log p_1(\texttt{j'aime})$$

Teacher forcing now feeds the **true** `j'aime` (not $\hat{y}_1$) as the input to step 2:

$$s_2 = f_{\text{dec}}(s_1, e(\texttt{j'aime}), c)$$

Step 1 loss

D DERIVATION

The target for step 1 is `j'aime`; the loss is its negative log-probability:

$$\mathcal{L}_1 = -\log p_1(\texttt{j'aime})$$

Teacher forcing now feeds the **true** `j'aime` (not $\hat{y}_1$) as the input to step 2:

$$s_2 = f_{\text{dec}}(s_1, e(\texttt{j'aime}), c)$$

each step: predict, score against the true next token, then feed that true token onward

The full sequence loss

D DERIVATION

Sum the per-step cross-entropies over the whole target:

The full sequence loss

D DERIVATION

Sum the per-step cross-entropies over the whole target:

$$\mathcal{L} = -\log p_1(\texttt{j'aime}) - \log p_2(\texttt{les}) - \log p_3(\texttt{chats}) - \log p_4(\texttt{<EOS>})$$

The full sequence loss

D DERIVATION

Sum the per-step cross-entropies over the whole target:

$$\mathcal{L} = -\log p_1(\texttt{j'aime}) - \log p_2(\texttt{les}) - \log p_3(\texttt{chats}) - \log p_4(\texttt{<EOS>})$$

$\mathcal{L} = -\sum_{u=1}^{4} \log p_{u(y_u^{\text{true}})}$ **— four next-token classifications, one per target position**

The full sequence loss

D DERIVATION

Sum the per-step cross-entropies over the whole target:

$$\mathcal{L} = -\log p_1(\texttt{j'aime}) - \log p_2(\texttt{les}) - \log p_3(\texttt{chats}) - \log p_4(\texttt{<EOS>})$$

$$\mathcal{L} = -\sum_{u=1}^{4} \log p_{u(y_u^{\text{true}})}$$ **— four next-token classifications, one per target position**

note the last term: the model is **supervised to emit `<EOS>`** — that is how it learns to stop

Inference and decoding

Greedy decoding

D DERIVATION

At inference there is no true history — feed the model's own output back. Simplest rule: take the **arg max** each step:

Greedy decoding

D DERIVATION

At inference there is no true history — feed the model's own output back. Simplest rule: take the **arg max** each step:

$$\hat{y}_u = \arg\max_v p(v \mid \hat{y}_{<u}, x_{1:T})$$

Greedy decoding

D DERIVATION

At inference there is no true history — feed the model's own output back. Simplest rule: take the **arg max** each step:

$$\hat{y}_u = \arg\max_v p(v \mid \hat{y}_{<u}, x_{1:T})$$

Greedy is myopic. The locally most-probable token need not start the globally most-probable **sequence** — one early commitment can doom the rest.

Beam search

D DERIVATION

Keep the B best **partial** sequences (the **beam**) instead of one. Score a hypothesis by its log-probability:

Beam search

D DERIVATION

Keep the B best **partial** sequences (the **beam**) instead of one. Score a hypothesis by its log-probability:

$$\text{score}(y_{1:u}) = \sum_{j=1}^{u} \log p\big(y_j \mid y_{<j}, x_{1:T}\big)$$

Beam search

D DERIVATION

Keep the B best **partial** sequences (the **beam**) instead of one. Score a hypothesis by its log-probability:

$$\text{score}(y_{1:u}) = \sum_{j=1}^{u} \log p\big(y_j \mid y_{<j}, x_{1:T}\big)$$

- **expand** every beam by all vocabulary tokens;
- **score** each extension; **keep** only the top B;
- repeat until enough beams emit <EOS> or a maximum length is reached.

Beam search

D DERIVATION

Keep the B best **partial** sequences (the **beam**) instead of one. Score a hypothesis by its log-probability:

$$\text{score}(y_{1:u}) = \sum_{j=1}^{u} \log p\big(y_j \mid y_{<j}, x_{1:T}\big)$$

- **expand** every beam by all vocabulary tokens;
- **score** each extension; **keep** only the top B;
- repeat until enough beams emit <EOS> or a maximum length is reached.

$B = 1$ is greedy; larger B searches wider (more compute) for a better sequence

Beam search, drawn

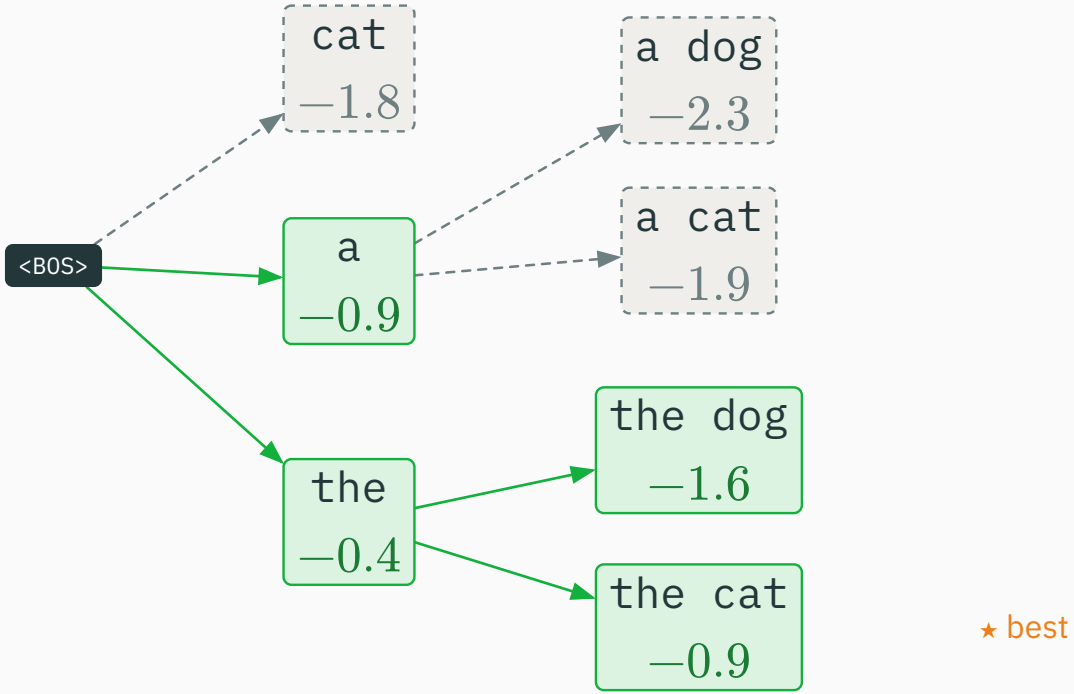
V VISUAL

$B = 2$: at each level, expand the survivors, score, and keep the best two:

Beam search, drawn

V VISUAL

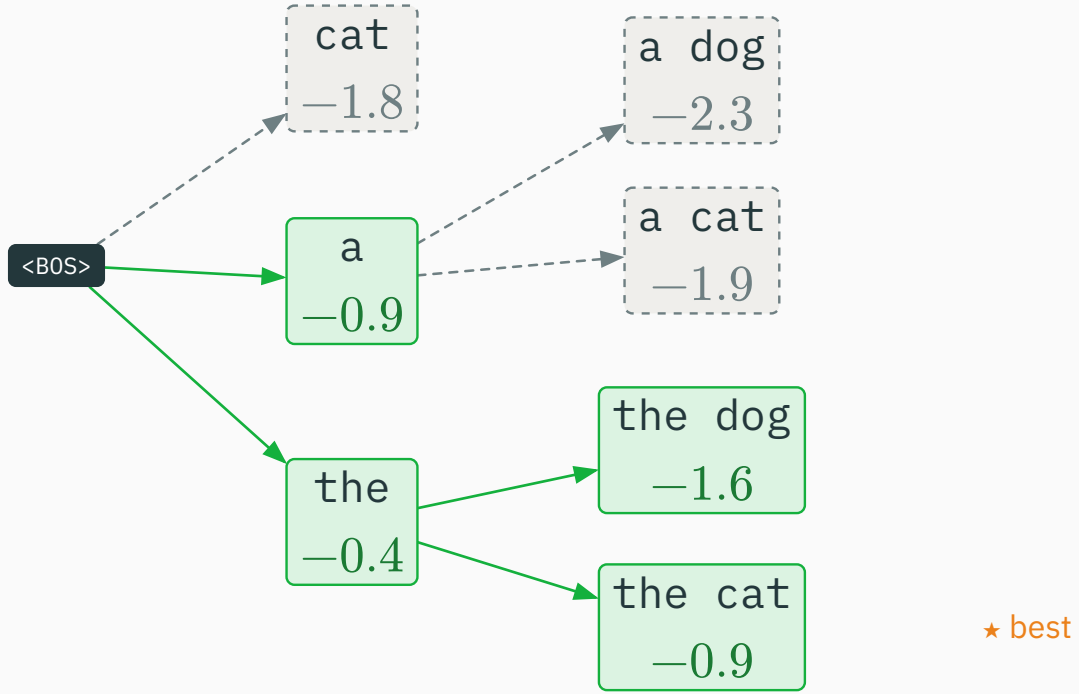
$B = 2$: at each level, expand the survivors, score, and keep the best two:



Beam search, drawn

V VISUAL

$B = 2$: at each level, expand the survivors, score, and keep the best two:



green = kept in the beam; dashed = pruned; the highest-scoring complete sequence wins

The length bias

D DERIVATION

Every factor has $\log p \leq 0$, so a **longer** sequence has a **more negative** raw score:

The length bias

D DERIVATION

Every factor has $\log p \leq 0$, so a **longer** sequence has a **more negative** raw score:

$$\text{score}(y_{1:U}) = \sum_{j=1}^{U} \log p(y_j \mid \ldots) \quad \leq \quad 0 \quad \text{(shrinks with each token)}$$

The length bias

D DERIVATION

Every factor has $\log p \leq 0$, so a **longer** sequence has a **more negative** raw score:

$$\text{score}(y_{1:U}) = \sum_{j=1}^{U} \log p(y_j \mid \ldots) \quad \leq \quad 0 \quad \text{(shrinks with each token)}$$

Raw log-prob **favors short outputs** — the beam is biased toward emitting <EOS> early. Fix it by **length-normalizing** the score:

The length bias

D DERIVATION

Every factor has $\log p \leq 0$, so a **longer** sequence has a **more negative** raw score:

$$\text{score}(y_{1:U}) = \sum_{j=1}^{U} \log p(y_j \mid \ldots) \quad \leq \quad 0 \quad \text{(shrinks with each token)}$$

Raw log-prob **favors short outputs** — the beam is biased toward emitting <EOS> early. Fix it by **length-normalizing** the score:

$$\text{score}_{\text{norm}}(y_{1:U}) = \frac{1}{U^{\alpha}} \sum_{j=1}^{U} \log p(y_j \mid \ldots), \qquad \alpha \in [0.6, 1]$$

The fixed-vector bottleneck

Everything through one vector

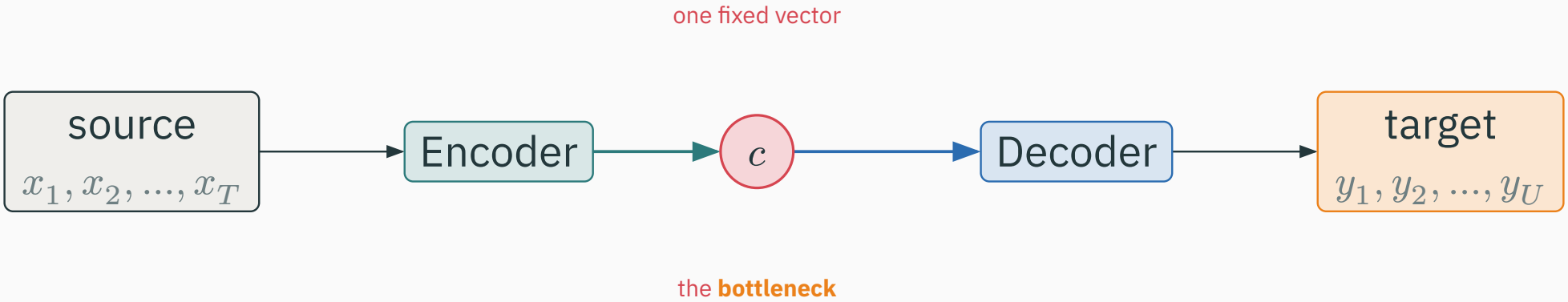
V VISUAL

In the plain encoder–decoder, **all** of the source must pass through a **single** $c = h_T$:

Everything through one vector

V VISUAL

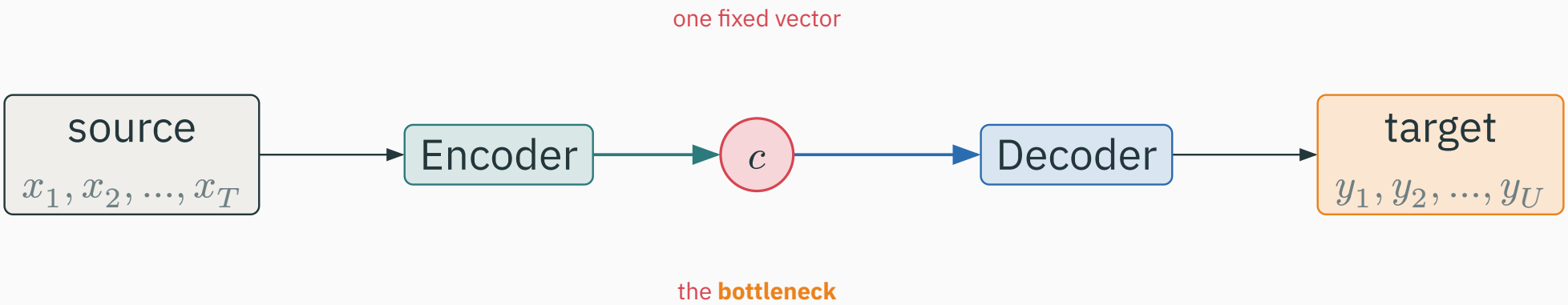
In the plain encoder–decoder, **all** of the source must pass through a **single** $c = h_T$:



Everything through one vector

V VISUAL

In the plain encoder–decoder, **all** of the source must pass through a **single** $c = h_T$:



a 40-word sentence and a 4-word sentence get the **same** fixed-size c

What goes wrong

What goes wrong

D DERIVATION

- for a **long** source, one vector cannot retain every detail — early tokens get **overwritten**;
- the decoder needs **different** parts of the source at **different** output steps — but c is **fixed** for all of them;
- empirically, translation quality **falls off sharply** as the source grows (Bahdanau et al. 2015).

What goes wrong

D DERIVATION

- for a **long** source, one vector cannot retain every detail — early tokens get **overwritten**;
- the decoder needs **different** parts of the source at **different** output steps — but c is **fixed** for all of them;
- empirically, translation quality **falls off sharply** as the source grows (Bahdanau et al. 2015).

The bottleneck is **architectural**, not a training issue: a single fixed vector is simply too small a channel for a long, structured source.

What we actually want

D DERIVATION

At **each** output step, the decoder should look at the **relevant** source positions — not a frozen summary:

What we actually want

D DERIVATION

At **each** output step, the decoder should look at the **relevant** source positions — not a frozen summary:

replace one fixed c with a per-step, source-dependent context c_u

What we actually want

D DERIVATION

At **each** output step, the decoder should look at the **relevant** source positions — not a frozen summary:

replace one fixed c with a per-step, source-dependent context c_u

when translating `chats`, look at `cats`; when translating `j'aime`, look at `like` — this is **attention**

Attention

The idea

D DERIVATION

Keep **all** encoder states $h_1, \ldots, h_T$. Build the context as a **weighted average** of them, recomputed every step:

The idea

D DERIVATION

Keep **all** encoder states $h_1, \ldots, h_T$. Build the context as a **weighted average** of them, recomputed every step:

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t, \qquad \alpha_{ut} \geq 0, \quad \sum_{t=1}^{T} \alpha_{ut} = 1$$

The idea

D DERIVATION

Keep **all** encoder states $h_1, \ldots, h_T$. Build the context as a **weighted average** of them, recomputed every step:

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t, \qquad \alpha_{ut} \geq 0, \quad \sum_{t=1}^{T} \alpha_{ut} = 1$$

α_{ut} = how much output step u **attends to** source position t — a soft, differentiable selection

Scores and weights

D DERIVATION

The weights come from a **score** between the decoder query and each encoder state, passed through a softmax:

Scores and weights

D DERIVATION

The weights come from a **score** between the decoder query and each encoder state, passed through a softmax:

$$e_{ut} = \text{score}(s_{u-1}, h_t) \qquad \text{(one scalar per source position)}$$

Scores and weights

D DERIVATION

The weights come from a **score** between the decoder query and each encoder state, passed through a softmax:

$$e_{ut} = \text{score}(s_{u-1}, h_t) \qquad \text{(one scalar per source position)}$$

$$\alpha_{ut} = \frac{\exp(e_{ut})}{\sum_{j=1}^{T} \exp(e_{uj})} \qquad \text{(softmax over source positions)}$$

Scores and weights

D DERIVATION

The weights come from a **score** between the decoder query and each encoder state, passed through a softmax:

$$e_{ut} = \text{score}(s_{u-1}, h_t) \qquad \text{(one scalar per source position)}$$

$$\alpha_{ut} = \frac{\exp(e_{ut})}{\sum_{j=1}^{T} \exp(e_{uj})} \qquad \text{(softmax over source positions)}$$

high score $\rightarrow$ high weight $\rightarrow$ that source state dominates the context

Scoring functions

D DERIVATION

Three standard ways to score a query s against a key h:

Scoring functions

D DERIVATION

Three standard ways to score a query s against a key h:

Name	score(s, h)	Note
dot	$s^\top h$	no parameters; needs $\dim s = \dim h$ (Luong)
general / bilinear	$s^\top W h$	a learned interaction matrix W (Luong)
additive / MLP	$v^\top \tanh(W_s s + W_h h)$	a small MLP (Bahdanau)

Scoring functions

D DERIVATION

Three standard ways to score a query s against a key h:

Name	score(s, h)	Note
dot	$s^\top h$	no parameters; needs $\dim s = \dim h$ (Luong)
general / bilinear	$s^\top W h$	a learned interaction matrix W (Luong)
additive / MLP	$v^\top \tanh(W_s s + W_h h)$	a small MLP (Bahdanau)

Bahdanau 2015 = **additive**; Luong 2015 = **dot** / **general** — all learned end-to-end

From context to prediction

D DERIVATION

Combine the context with the decoder state, then read out the next token:

From context to prediction

D DERIVATION

Combine the context with the decoder state, then read out the next token:

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t$$

From context to prediction

D DERIVATION

Combine the context with the decoder state, then read out the next token:

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t$$

$$p(y_u \mid y_{<u}, x_{1:T}) = \mathrm{softmax}(W_o\,[s_u; c_u] + b_o)$$

From context to prediction

D DERIVATION

Combine the context with the decoder state, then read out the next token:

$$c_u = \sum_{t=1}^{T} \alpha_{ut}\, h_t$$

$$p(y_u \mid y_{<u}, x_{1:T}) = \mathrm{softmax}(W_o\,[s_u; c_u] + b_o)$$

the readout sees **both** the decoder state s_u **and** the freshly-attended source context c_u

The attention mechanism

V VISUAL

Query the encoder states, softmax the scores into weights, take the weighted sum:

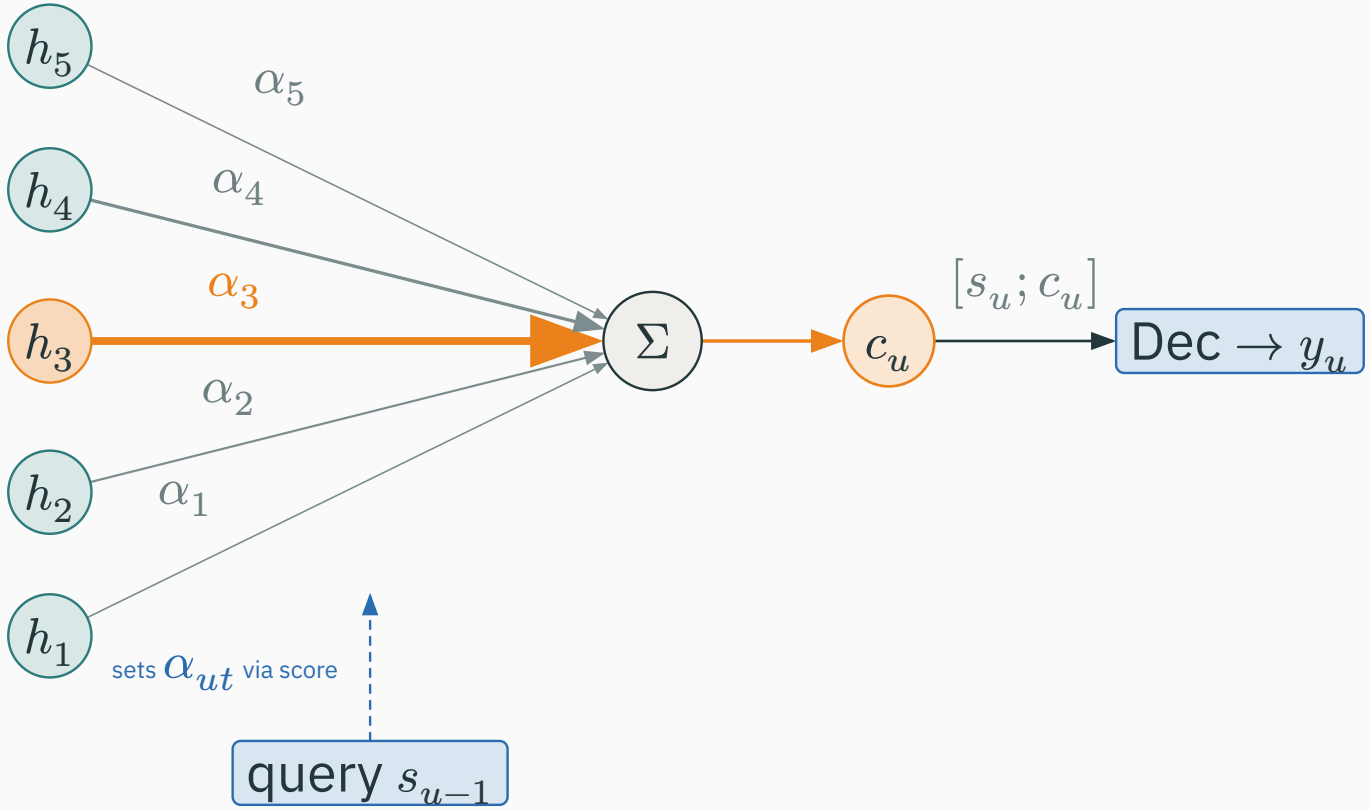
The attention mechanism

V VISUAL

The attention mechanism

V VISUAL

Query the encoder states, softmax the scores into weights, take the weighted sum:



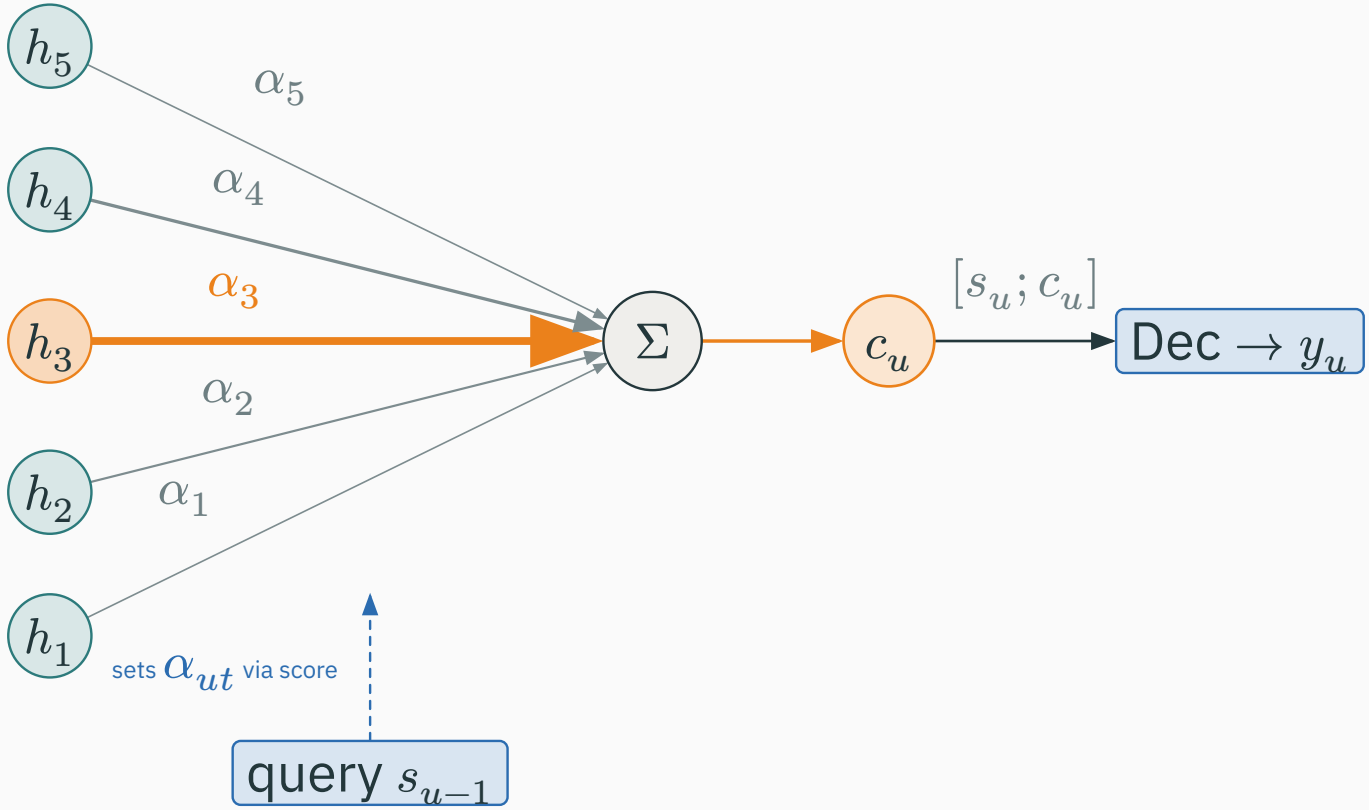
The attention mechanism

V VISUAL

The attention mechanism

V VISUAL

Query the encoder states, softmax the scores into weights, take the weighted sum:



The attention mechanism

V VISUAL

thicker edge = larger α; here step u attends mostly to h_3, so $c_u \approx h_3$

Attention is a differentiable lookup

D DERIVATION

Rename the pieces and attention becomes a **soft dictionary lookup** — the language of Transformers:

Attention is a differentiable lookup

D DERIVATION

Rename the pieces and attention becomes a **soft dictionary lookup** — the language of Transformers:

query $q = s_{u-1}$ — what the decoder is looking for.

keys $k_t = h_t$, **values** $v_t = h_t$ — what the source offers.

Attention is a differentiable lookup

D DERIVATION

Rename the pieces and attention becomes a **soft dictionary lookup** — the language of Transformers:

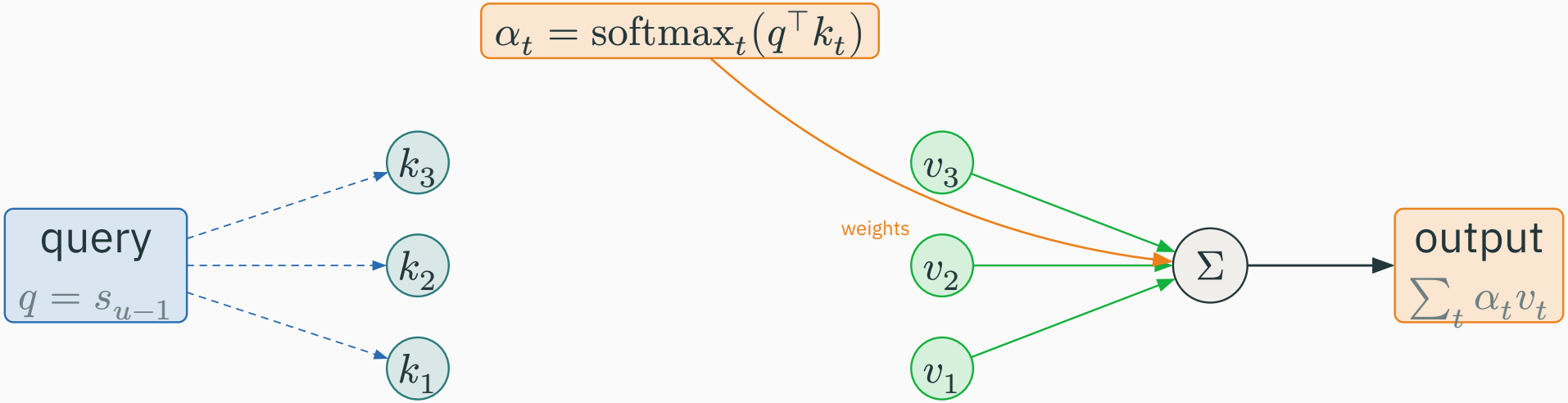
query $q = s_{u-1}$ — what the decoder is looking for.

keys $k_t = h_t$, **values** $v_t = h_t$ — what the source offers.

$$\alpha_t = \mathrm{softmax}_t(q^\top k_t), \qquad \text{output} = \sum_{t=1}^{T} \alpha_t \, v_t$$

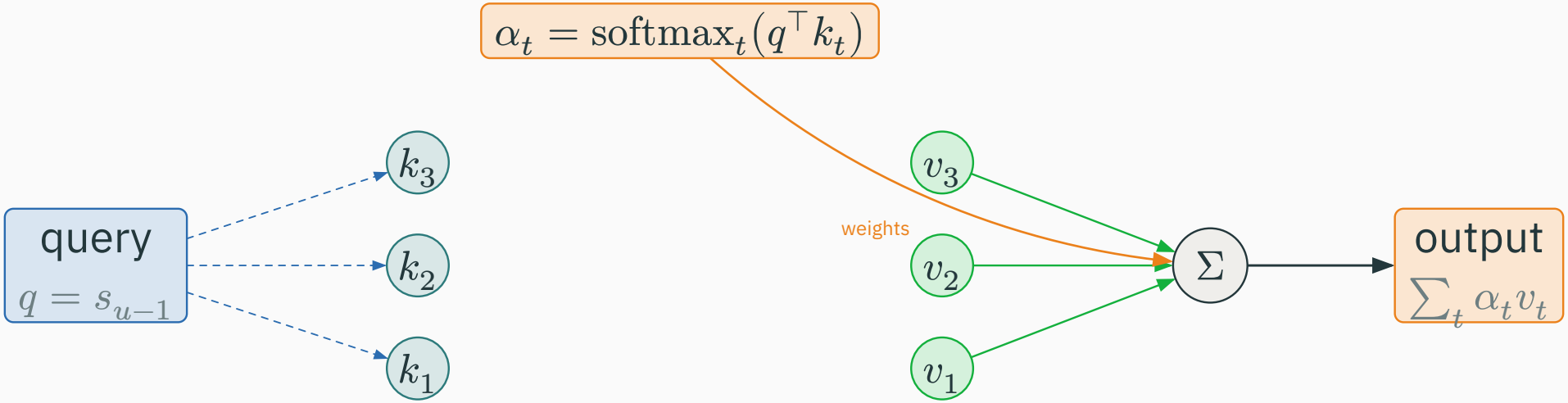
The lookup, drawn

V VISUAL



The lookup, drawn

V VISUAL



a **hard** lookup returns one value; attention returns a **soft, weighted** blend — and is differentiable, so it **learns**

A worked attention step

Scores, then softmax

D DERIVATION

Scalar encoder states $h = [1, 2, 4]$ and a query $q = 2$. Dot-product scores:

Scores, then softmax

D DERIVATION

Scalar encoder states $h = [1, 2, 4]$ and a query $q = 2$. Dot-product scores:

$e_t = q \cdot h_t : \quad e = [2, 4, 8]$

Scores, then softmax

D DERIVATION

Scalar encoder states $h = [1, 2, 4]$ and a query $q = 2$. Dot-product scores:

$$e_t = q \cdot h_t : \quad e = [2, 4, 8]$$

Softmax the scores into weights:

$$\alpha = \mathrm{softmax}([2, 4, 8]) \approx [0.0024, 0.0179, 0.9796]$$

Scores, then softmax

D DERIVATION

Scalar encoder states $h = [1, 2, 4]$ and a query $q = 2$. Dot-product scores:

$$e_t = q \cdot h_t : \quad e = [2, 4, 8]$$

Softmax the scores into weights:

$$\alpha = \mathrm{softmax}([2, 4, 8]) \approx [0.0024, 0.0179, 0.9796]$$

the score gap of 8 vs 4 is **exponentiated** — step attends almost entirely to h_3

The context vector

V VISUAL

Weighted sum of the states — the context lands near the most-attended one:

The context vector

V VISUAL

Weighted sum of the states — the context lands near the most-attended one:

$$c = \sum_t \alpha_t h_t \approx 0.0024(1) + 0.0179(2) + 0.9796(4) \approx 3.96$$

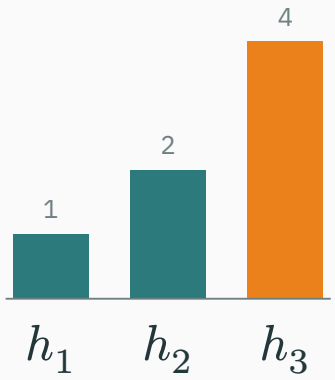
The context vector

V VISUAL

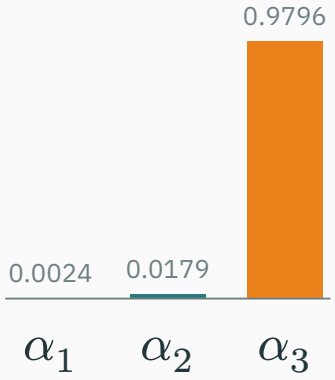
Weighted sum of the states — the context lands near the most-attended one:

$$c = \sum_t \alpha_t h_t \approx 0.0024(1) + 0.0179(2) + 0.9796(4) \approx 3.96$$

encoder states h_t (query $q = 2$)



weights $\alpha = \mathrm{softmax}(qh)$



Temperature: sharp vs diffuse

V VISUAL

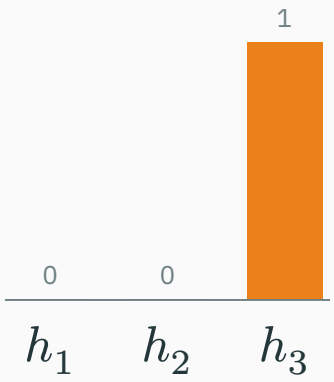
Softmax with temperature τ: $\alpha = \mathrm{softmax}(e/\tau)$ — small τ sharpens, large τ spreads:

Temperature: sharp vs diffuse

V VISUAL

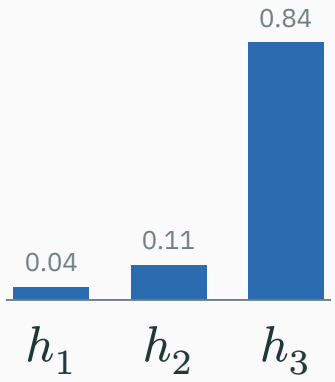
Softmax with temperature τ: $\alpha = \mathrm{softmax}(e/\tau)$ — small τ sharpens, large τ spreads:

sharp $\tau = 0.5$



default $\tau = 1$

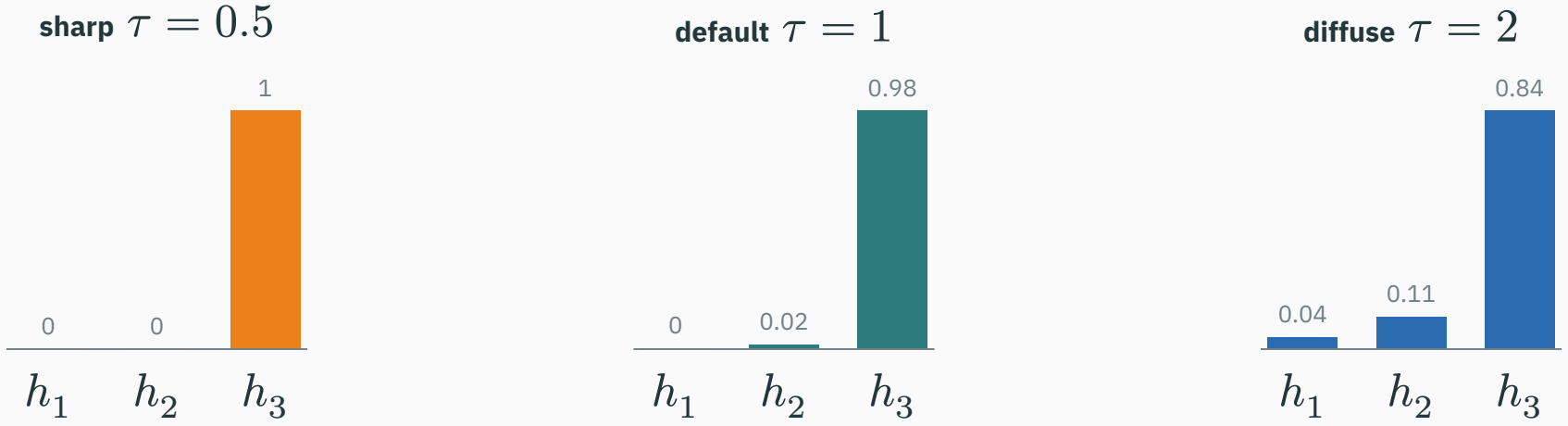
diffuse $\tau = 2$



Temperature: sharp vs diffuse

V VISUAL

Softmax with temperature τ: $\alpha = \mathrm{softmax}(e/\tau)$ — small τ sharpens, large τ spreads:



Transformers use **scaled** dot-product $q^\top k/\sqrt{d}$ — the $\sqrt{d}$ keeps scores from saturating the softmax

Alignment and interpretability

The attention matrix

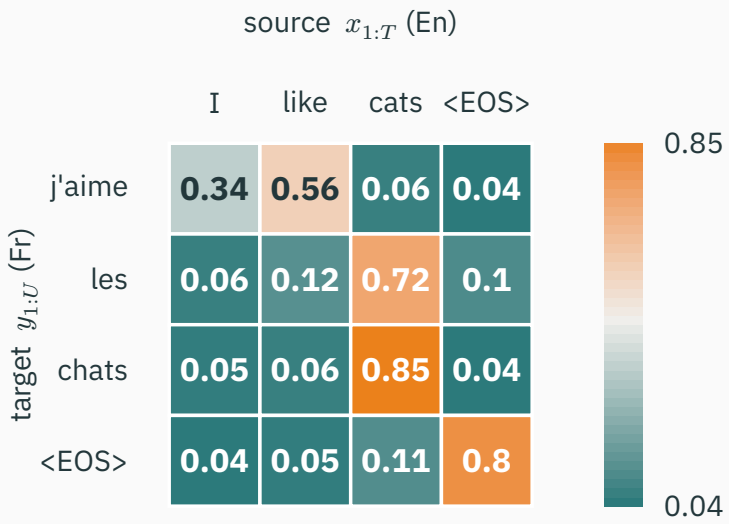
V VISUAL

Stack the weights over all steps: $A \in \mathbb{R}^{U \times T}$, with $A_{ut} = \alpha_{ut}$. Each row is one output step's distribution over the source:

The attention matrix

V VISUAL

Stack the weights over all steps: $A \in \mathbb{R}^{U \times T}$, with $A_{ut} = \alpha_{ut}$. Each row is one output step's distribution over the source:



It learns an alignment

D DERIVATION

The bright cells trace a soft **word alignment** — learned, never supervised:

It learns an alignment

The bright cells trace a soft **word alignment** — learned, never supervised:

target token	attends to source
`j'aime`	`like` (and a little `I`)
`les`	`cats` (the plural determiner)
`chats`	`cats`
`<EOS>`	`<EOS>`

It learns an alignment

The bright cells trace a soft **word alignment** — learned, never supervised:

target token	attends to source
`j'aime`	`like` (and a little `I`)
`les`	`cats` (the plural determiner)
`chats`	`cats`
`<EOS>`	`<EOS>`

attention discovers **which source word each target word corresponds to** — for free

Attention is not always explanation

I INTERACTIVE

Attention is not always explanation

I INTERACTIVE

Caution. Attention weights are a useful **diagnostic** of what the model looked at — but they are **not always a faithful explanation**. Different weightings can give the same output, and high weight $\neq$ causal importance (Jain & Wallace 2019).

Attention is not always explanation

I INTERACTIVE

Caution. Attention weights are a useful **diagnostic** of what the model looked at — but they are **not always a faithful explanation**. Different weightings can give the same output, and high weight $\neq$ causal importance (Jain & Wallace 2019).

INTERACTIVE

↗ nipunbatra.github.io/interactive-articles/attention

Hover the alignment heatmap for “I like cats” $\rightarrow$ “j’aime les chats”: watch each target row light up the source words it attends to, and edit the sentence to see the alignment shift.

The full seq2seq + attention model

Encoder: keep every state

D DERIVATION

The encoder is unchanged — but now we **keep all** the states, not just the last one:

Encoder: keep every state

D DERIVATION

The encoder is unchanged — but now we **keep all** the states, not just the last one:

$$h_t = f_{\text{enc}}(h_{t-1}, e(x_t)), \qquad H = (h_1, ..., h_T)$$

Encoder: keep every state

D DERIVATION

The encoder is unchanged — but now we **keep all** the states, not just the last one:

$$h_t = f_{\text{enc}}(h_{t-1}, e(x_t)), \qquad H = (h_1, \ldots, h_T)$$

H is the memory the decoder attends into — the fixed $c = h_T$ is gone

Decoder with attention

D DERIVATION

Each decoder step: update the state, attend over H, predict from state $+$ context:

Decoder with attention

D DERIVATION

Each decoder step: update the state, attend over H, predict from state + context:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c_{u-1})$$

Decoder with attention

D DERIVATION

Each decoder step: update the state, attend over H, predict from state + context:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c_{u-1})$$

$$e_{ut} = \text{score}(s_u, h_t), \qquad \alpha_{ut} = \text{softmax}_t(e_{ut}), \qquad c_u = \sum_{t=1}^{T} \alpha_{ut} h_t$$

Decoder with attention

D DERIVATION

Each decoder step: update the state, attend over H, predict from state + context:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c_{u-1})$$

$$e_{ut} = \text{score}(s_u, h_t), \qquad \alpha_{ut} = \text{softmax}_t(e_{ut}), \qquad c_u = \sum_{t=1}^{T} \alpha_{ut} h_t$$

$$p(y_u \mid \cdot) = \text{softmax}(W_o\,[s_u; c_u] + b_o)$$

Decoder with attention

D DERIVATION

Each decoder step: update the state, attend over H, predict from state + context:

$$s_u = f_{\text{dec}}(s_{u-1}, e(y_{u-1}), c_{u-1})$$

$$e_{ut} = \text{score}(s_u, h_t), \qquad \alpha_{ut} = \text{softmax}_t(e_{ut}), \qquad c_u = \sum_{t=1}^{T} \alpha_{ut} h_t$$

$$p(y_u \mid \cdot) = \text{softmax}(W_o\,[s_u; c_u] + b_o)$$

this is the Luong **input-feeding** form (query s_u, previous context c_{u-1} fed back in); earlier we scored with the Bahdanau query s_{u-1}

The full model

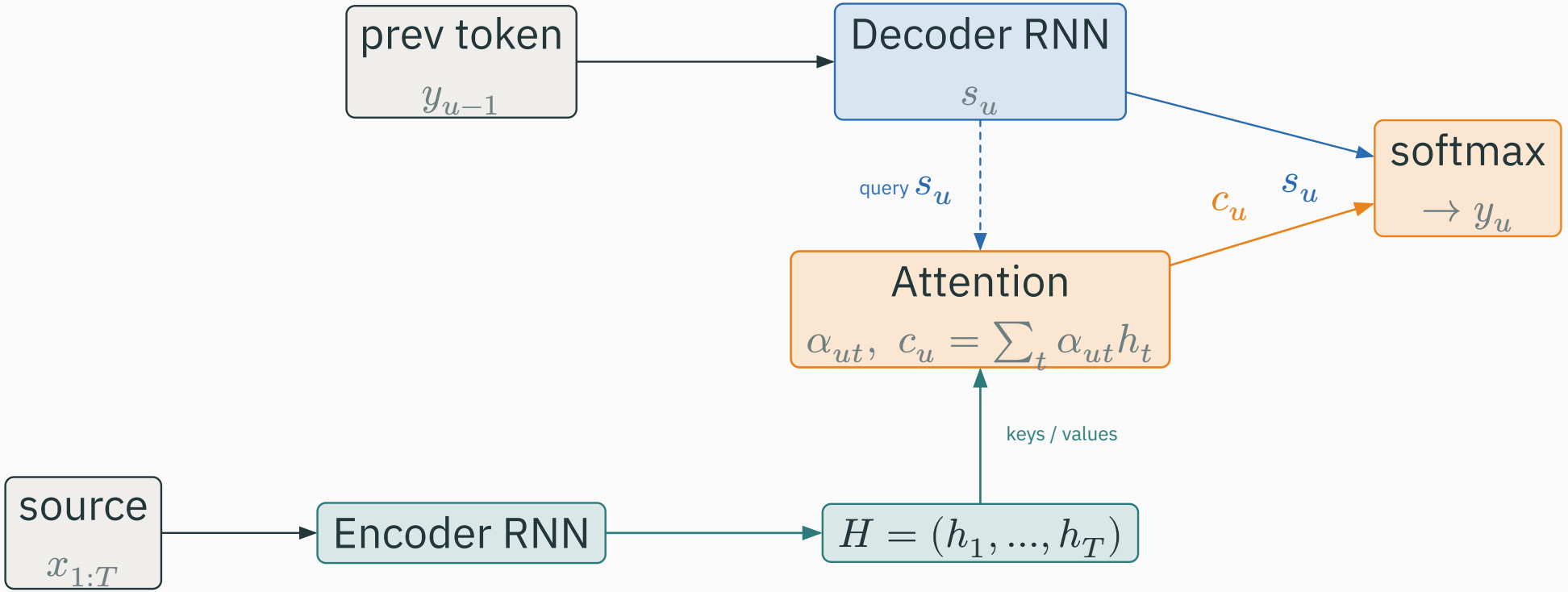
V VISUAL

Encoder produces H; every decoder step queries it for a fresh context c_u:

The full model

V VISUAL

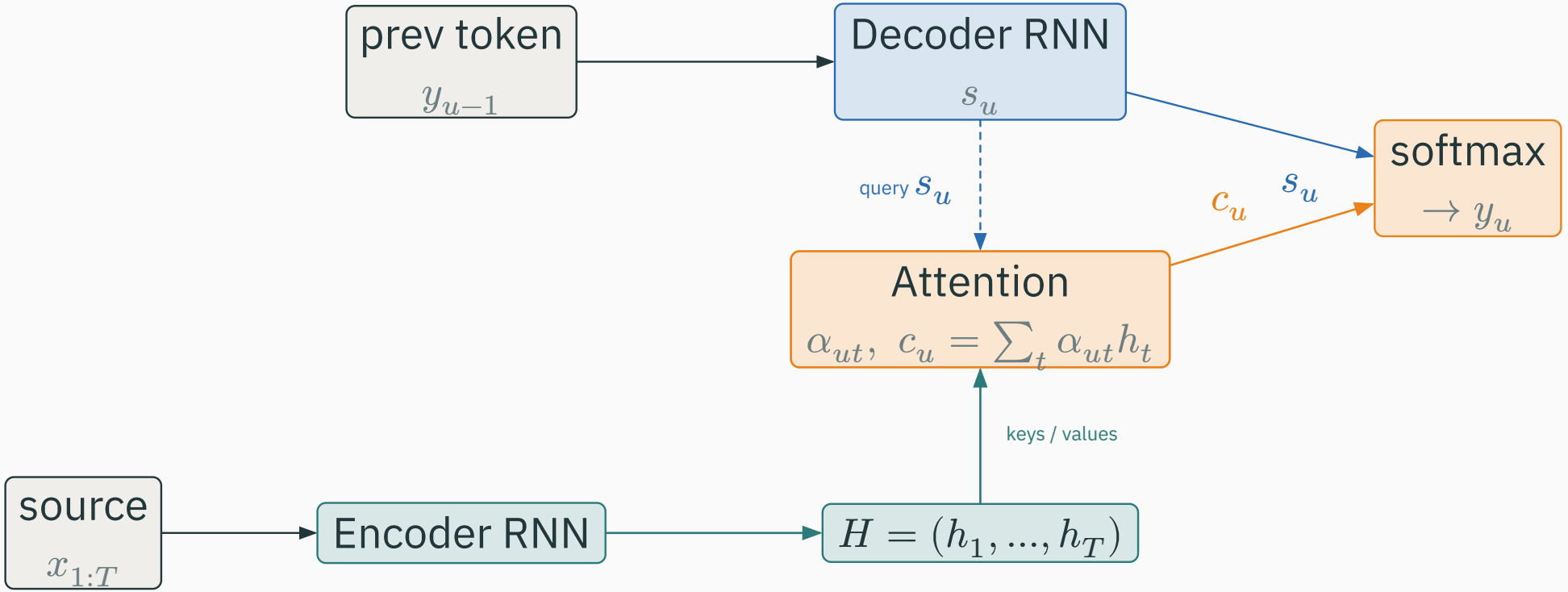
Encoder produces H; every decoder step queries it for a fresh context c_u:



The full model

V VISUAL

Encoder produces H; every decoder step queries it for a fresh context c_u:



the attention block runs **once per output step** — a new c_u each time, chosen by the query s_u

One objective, gradients everywhere

D DERIVATION

Same masked cross-entropy as before — but now the gradient reaches **further**:

One objective, gradients everywhere

D DERIVATION

Same masked cross-entropy as before — but now the gradient reaches **further**:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p(y_u \mid y_{<u}, x_{1:T})$$

One objective, gradients everywhere

D DERIVATION

Same masked cross-entropy as before — but now the gradient reaches **further**:

$$\mathcal{L} = -\sum_{u=1}^{U} \log p(y_u \mid y_{<u}, x_{1:T})$$

Backprop flows: decoder readout $\rightarrow$ attention weights $\alpha_{ut} \rightarrow$ encoder states $h_t \rightarrow$ the encoder RNN. Attention is **fully differentiable**, so the alignment is **learned** end-to-end from the translation loss alone.

Implementation

Batching variable lengths

D DERIVATION

Sequences differ in length, so **pad** to a common T and track a **source mask**:

Batching variable lengths

D DERIVATION

Sequences differ in length, so **pad** to a common T and track a **source mask**:

$$m_t = \begin{cases} 1 & x_t \neq \texttt{<PAD>} \\ 0 & x_t = \texttt{<PAD>} \end{cases}$$

Batching variable lengths

D DERIVATION

Sequences differ in length, so **pad** to a common T and track a **source mask**:

$$m_t = \begin{cases} 1 & x_t \neq \texttt{<PAD>} \\ 0 & x_t = \texttt{<PAD>} \end{cases}$$

the mask marks which source positions are real — attention must not attend to padding

Masked attention

D DERIVATION

Padding positions must get **zero** weight. Set their scores to $-\infty$ **before** the softmax:

Masked attention

D DERIVATION

Padding positions must get **zero** weight. Set their scores to $-\infty$ **before** the softmax:

$$e_{ut} \leftarrow \begin{cases} e_{ut} & m_t = 1 \\ -\infty & m_t = 0 \end{cases} \quad \Rightarrow \quad \alpha_{ut} = 0 \text{ on padding}$$

Masked attention

D DERIVATION

Padding positions must get **zero** weight. Set their scores to $-\infty$ **before** the softmax:

$$e_{ut} \leftarrow \begin{cases} e_{ut} & m_t = 1 \\ -\infty & m_t = 0 \end{cases} \quad \Rightarrow \quad \alpha_{ut} = 0 \text{ on padding}$$

$\exp(-\infty) = 0$, so masked positions drop out of the softmax cleanly and the weights still sum to 1 over the **real** tokens. (The same trick gives the **causal** mask in Transformers.)

Teacher-forcing loop

V VISUAL

Train step: feed the **true** previous target each step, accumulate masked cross-entropy:

Teacher-forcing loop

V VISUAL

Train step: feed the **true** previous target each step, accumulate masked cross-entropy:

```
H, enc_state = encoder(src, src_mask)   # keep ALL states H for attention
dec_in = BOS.expand(batch)              # first decoder input
state = init_from(enc_state); loss = 0
for u in range(U):                      # teacher forcing along the target
    logits, state, alpha = decoder(dec_in, state, H, src_mask)
    loss += cross_entropy(logits, tgt[:, u], ignore_index=PAD)
    dec_in = tgt[:, u]                  # feed the TRUE token, not argmax
loss.backward()
```

Greedy decode & beam

V VISUAL

Inference: feed the model's **own** output back; or keep B hypotheses:

Greedy decode & beam

V VISUAL

Inference: feed the model's **own** output back; or keep B hypotheses:

```
# greedy
dec_in = BOS; out = []
for _ in range(max_len):
  logits, state, _ = decoder(
      dec_in, state, H, src_mask)
  dec_in = logits.argmax(-1)
  if dec_in == EOS: break
  out.append(dec_in)
```

```
# beam search (B hypotheses)
beams = [Hyp(tokens=[BOS], state=s0,
             score=0.0, done=False)]
# each step: expand every beam,
#   score = cum log-prob,
#   keep top-B, stop when all done
```

Limitations and summary

What attention fixed — and did not

D DERIVATION

What attention fixed — and did not

D DERIVATION

Fixed — the bottleneck.
before: $x \to c = h_T \to y$.
after: $x \to H$, and $c_u = \sum_t \alpha_{ut} h_t$ **every step**.

Not fixed — recurrence. The decoder is **still** an RNN: $s_u = f(s_{u-1}, y_{u-1}, c_{u-1})$ — target generation stays **sequential**.

What attention fixed — and did not

D DERIVATION

Fixed — the bottleneck.
before: $x \to c = h_T \to y$.
after: $x \to H$, and $c_u = \sum_t \alpha_{ut} h_t$ **every step**.

Not fixed — recurrence. The decoder is **still** an RNN: $s_u = f(s_{u-1}, y_{u-1}, c_{u-1})$ — target generation stays **sequential**.

attention removed **one** bottleneck; the **sequential** encoder and decoder remain

The remaining costs

The remaining costs

D DERIVATION

- the encoder is $O(T)$ **sequential** — cannot parallelize over source time;
- the decoder is $O(U)$ **sequential** — each s_u needs s_{u-1};
- long-range dependencies still travel through many recurrent steps.

The remaining costs

D DERIVATION

- the encoder is $O(T)$ **sequential** — cannot parallelize over source time;
- the decoder is $O(U)$ **sequential** — each s_u needs s_{u-1};
- long-range dependencies still travel through many recurrent steps.

if attention can access any position directly — do we even need recurrence?

What changed, across three lectures

V VISUAL

Model	Conditional	Context used
RNN LM	$p(y_t \mid y_{<t})$	own past only
seq2seq	$p(y_t \mid y_{<t}, x)$	one fixed $c = h_T$
seq2seq + attention	$p\left(y_t \mid y_{<t}, \sum_t \alpha h_t\right)$	per-step context c_u

What changed, across three lectures

V VISUAL

Model	Conditional	Context used
RNN LM	$p(y_t \mid y_{<t})$	own past only
seq2seq	$p(y_t \mid y_{<t}, x)$	one fixed $c = h_T$
seq2seq + attention	$p\left(y_t \mid y_{<t}, \sum_t \alpha h_t\right)$	per-step context c_u

each row adds access: to the source, then to **every source position, dynamically**

One-slide summary

V VISUAL

Idea	Takeaway
encoder–decoder	encode source $\rightarrow c$; decode target autoregressively from c
teacher forcing	feed true y_{u-1} in training — stable, but **exposure bias**
decoding	greedy / beam search; length-normalize the score
bottleneck	one fixed $c = h_T$ loses long, structured sources
attention	$c_u = \sum_t \alpha_{ut} h_t$ — per-step, source-dependent context
scoring	dot / general / additive; softmax; query–key–value lookup

Final mental model — one idea

The **encoder** represents the source; the **decoder** generates the target.

Teacher forcing trains it; **attention** lets each step read the source **dynamically**.

attention = a differentiable, weighted lookup over all source states

Attention removed the bottleneck — but recurrence remains.

Next: **Transformers** — remove recurrence entirely and make **attention the main computation** (queries, keys, values, self-attention, causal masks, positional encodings).