Seq2Seq & the Motivation for Attention

Lecture 11 · ES 667: Deep Learning

Prof. Nipun Batra
IIT Gandhinagar · Aug 2026

Learning outcomes

By the end of this lecture you will be able to:

1. Describe **encoder-decoder** RNN architecture for variable-length tasks.
2. Implement **teacher forcing** and explain exposure bias.
3. Contrast **greedy / beam / top-k / nucleus** decoding.
4. Apply **length normalization** in beam search.
5. Diagnose the **fixed-length bottleneck** that killed pre-attention Seq2Seq.
6. Motivate **attention** (L12) as the fix for the bottleneck.

Recap · where we are

Last lecture: **LSTMs** solve the vanishing-gradient problem in RNNs via gated cell states.

But solving depth is not enough. Many tasks need to map **an input sequence** to **a different output sequence**:

- Machine translation — English to French
- Summarization — article to abstract
- Speech recognition — audio to text
- Program synthesis — comment to code

> REFERENCE
>
> Today maps to **Bishop Ch 12** (Seq2Seq). UDL treats Transformers directly; we cover the encoder-decoder precursor because it *motivates* attention.

Four questions

1. How do we map a variable-length input to a variable-length output?
2. What is **teacher forcing** and why does everyone use it?
3. How do we **decode** at inference — greedy, beam, nucleus?
4. What's wrong with Seq2Seq — and why did attention become inevitable?

PART 1

The encoder-decoder architecture

Read all, then generate

Seq2Seq · the 2014 breakthrough

> REFERENCE
>
> Sutskever, Vinyals, Le 2014 · *"Sequence to Sequence Learning with Neural Networks"* — achieved BLEU 34 on English→French, within striking distance of phrase-based MT.

Two separate RNNs:

1. **Encoder** · reads the source sequence $(x_1, \ldots, x_T)$, updates its hidden state.
2. **Context vector $\mathbf{c}$** · encoder's final hidden state — a compressed summary of the whole source.
3. **Decoder** · starts from $\mathbf{c}$, generates target tokens $(y_1, y_2, \ldots)$ one at a time.

The whole idea in one sentence

Compress source into a vector · decompress into target.

KEY IDEA

Two unrolled RNNs, back to back, trained end-to-end. No grammar rules, no alignment dictionaries, no phrase tables — the representations are learned from parallel corpus data alone. This was radically new in 2014; by 2016 it was state-of-the-art in production MT.

The same encoder-decoder pattern returns in T5 (L14), Stable Diffusion (L22), and every modular ML system that maps between domains.

Shared vs separate vocabularies

Two design choices:

- **Separate vocab** · source is 40k English tokens, target is 40k French tokens, each with its own embedding matrix. Clean; embeddings can specialize.
- **Shared vocab** · one vocabulary for both, one embedding matrix. Saves parameters; lets the model see "Paris" as the same token in both languages.

> IN PRACTICE
>
> Modern multilingual models (mT5, NLLB, Whisper) share vocab via SentencePiece — a single token stream covers 100+ languages. Today's LLMs do the same.

The translator analogy

KEY IDEA

A human translator faced with a long German sentence does not translate word-by-word. They **read the whole sentence**, pause to grasp the meaning, **form a mental summary**, then start composing the English translation by unpacking that summary.

Seq2Seq does exactly this · the encoder reads, builds a context vector (the "mental summary"), and the decoder unpacks it into the target language.

The problem · for *long* sentences, even a great human's mental summary fails. So does Seq2Seq · this is why we'll add attention in L12.

The big picture · encoder–decoder

INTUITION

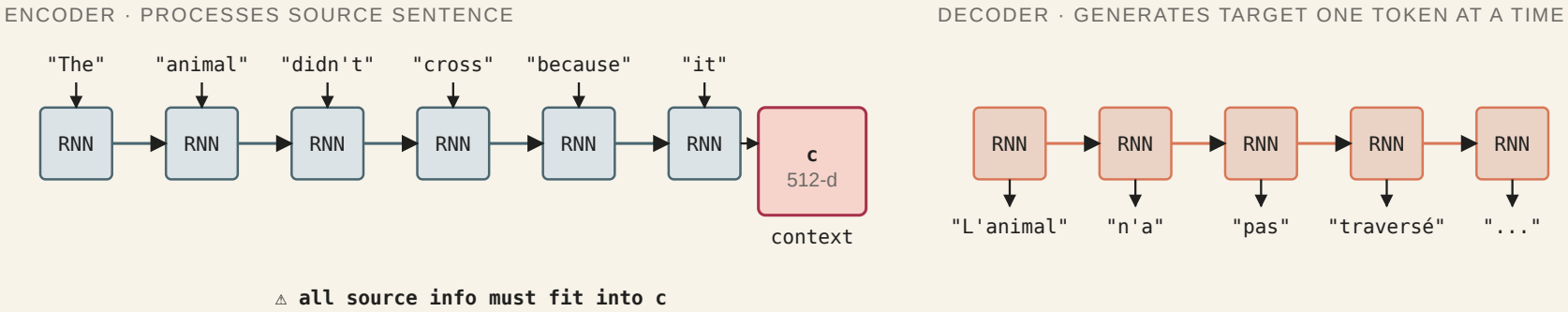
Forget RNNs for a second. Two black boxes:

1. **Encoder** · reads the entire English sentence and squishes its meaning into a single **thought vector $\mathbf{c}$**.
2. **Decoder** · takes $\mathbf{c}$ and unpacks it, word by word, into French.

Like reading a sentence, thinking *"got it"*, and explaining it in another language.

The architecture

Seq2Seq — encoder compresses everything into one vector; decoder generates from it



⚠ **all source info must fit into c**

THE BOTTLENECK PROBLEM

Every word of the source must be encoded into a `512-dim vector c`.

Short sentences — fine. 50-word sentences — you lose detail. 500-word paragraphs — forget it.

Performance of Seq2Seq degrades roughly linearly with source length.

IN PRACTICE

▶

Interactive: see BLEU curves fall as source length grows — seq2seq-bottleneck.

A quick look inside `nn.LSTM`

An `nn.LSTM` layer is a function with specific I/O:

- **Input** · sequence of embedded tokens (e.g. 10-word sentence → `(10, 256)`).
- **Output** · returns **two** things:
  i. `outputs` — hidden state at *every* time step (e.g. `(10, 512)`).
  ii. `(h_n, c_n)` — *final* hidden + cell state (each `(1, 512)`).

  For the encoder we want the **final** state — that's our context vector $\mathbf{c}$. We discard `outputs` (using `_` in Python) and keep the tuple `(h, c)`.

Seq2Seq in PyTorch · skeleton

```
class Seq2Seq(nn.Module):
    def __init__(self, src_vocab, tgt_vocab, d_emb=256, d_h=512):
        super().__init__()
        self.src_emb = nn.Embedding(src_vocab, d_emb)
        self.tgt_emb = nn.Embedding(tgt_vocab, d_emb)
        self.encoder = nn.LSTM(d_emb, d_h, batch_first=True)
        self.decoder = nn.LSTM(d_emb, d_h, batch_first=True)
        self.output  = nn.Linear(d_h, tgt_vocab)

    def forward(self, src, tgt):
        _, (h, c) = self.encoder(self.src_emb(src))     # context = (h, c)
        dec_out, _ = self.decoder(self.tgt_emb(tgt), (h, c))
        return self.output(dec_out)
```

PART 2

Teacher forcing

How we train sequence generators

The problem with training auto-regressively

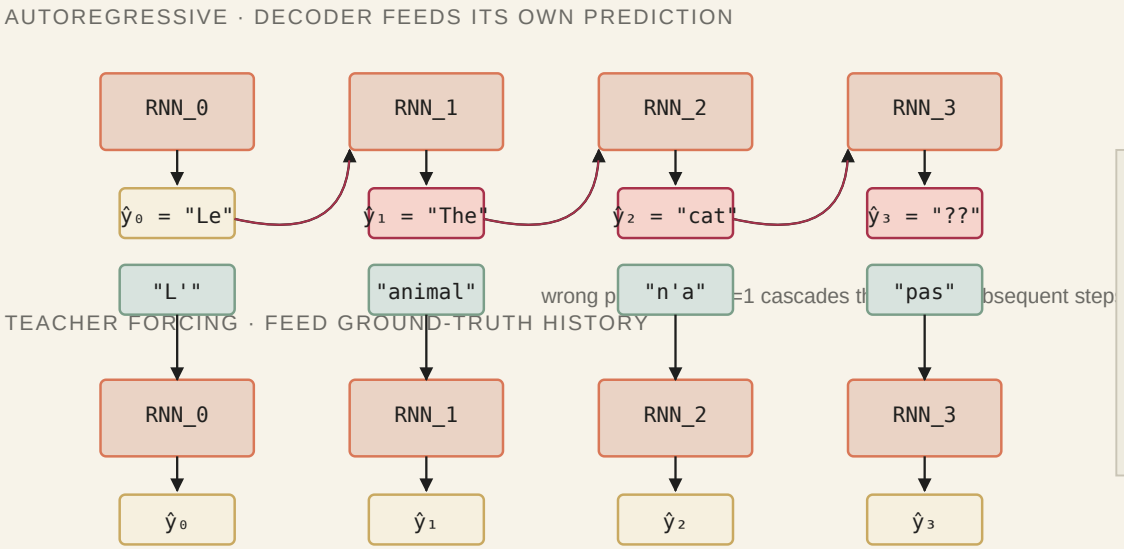
At **inference** the decoder feeds its own predictions back in:

> *decoder sees "<start>" → emits "The" → sees "The" → emits "animal" → …*

But at **training**, if the decoder's first prediction is wrong, the error compounds — all subsequent steps condition on bad inputs. Training becomes painfully slow and unstable.

Teacher forcing · the fix

Teacher forcing · feed ground-truth history during training · no error cascade



TRADE-OFF · EXPOSURE BIAS

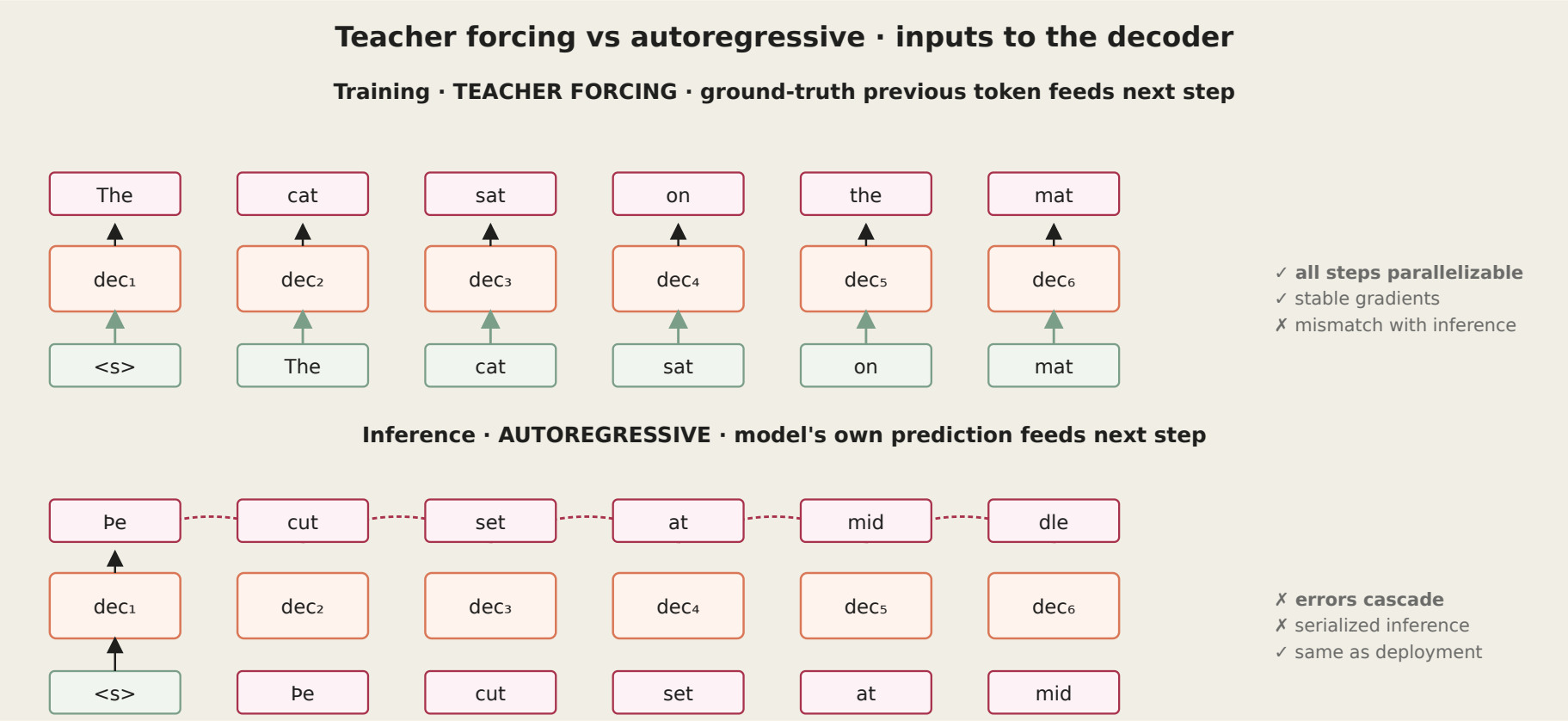
Training · decoder always sees truth.
Inference · decoder sees its own wrong predictions it was never trained on.

→ distribution shift at test time

Mitigations:
- scheduled sampling (Bengio 2015)
- Transformer-scale just hides it

every step conditioned on correct prefix · stable parallel training · the standard

Teacher forcing · detailed flow



Teacher forcing · the training-wheels analogy

INTUITION

Like learning to ride a bike with a parent holding the seat. You still pedal and steer (predict the next word). But when you wobble (make a mistake), the parent keeps you on the right path (feeds you the ground-truth word).

You learn the core motion much faster and more safely. At inference, the training wheels come off.

Two regimes side-by-side

Autoregressive (inference)

$$y_{t-1}^{\text{pred}} \rightarrow \text{decoder} \rightarrow y_t^{\text{pred}}$$

The decoder feeds **its own** prediction back in. One error at step 1 → wrong context for *every* later step.

Teacher forcing (training)

$$y_{t-1}^{\text{truth}} \rightarrow \text{decoder} \rightarrow y_t^{\text{pred}}$$

The decoder sees the **ground-truth** previous token. Every step is a clean, independent prediction problem.

Why teacher forcing? · speed

The biggest reason for teacher forcing isn't safety — it's **parallelism**.

- **Autoregressive** · compute step 1 → step 2 → step 3 → … sequential.
- **Teacher forcing** · all decoder inputs `<s>, "The", "cat", …` are known up-front → feed them in **all at once** → one big matrix multiply.

  A slow sequential loop becomes a fast parallel computation — **10–100× speedup** on training. Empirically tolerated despite the "you train on a distribution you won't see at inference" critique.

Teacher forcing · why it's OK pragmatically

Even though training ≠ inference, teacher forcing works because:

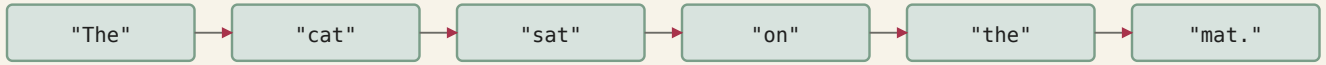
1. **Loss is averaged over all time steps** · bad predictions at step 10 aren't penalized harder than bad predictions at step 1. Every position gets balanced gradient.
2. **The model learns conditional distributions** $P(y_t|y_{<t})$ · if you give it the right $y_{<t}$ at inference (as in a perfect rollout), it generalizes.
3. **Huge speedup from parallelization** · with ground truth, all decoder steps can be computed in parallel (one big matrix multiply) instead of sequentially.

> INTUITION
>
> The "you're training on a distribution you won't see at inference" critique is real (exposure bias, next slide). But empirically it's tolerated because the alternative — fully autoregressive training — is 10-100× slower.

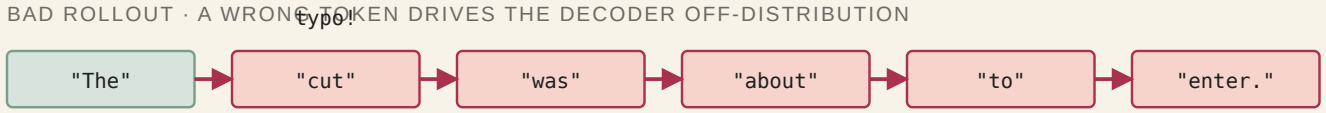
Exposure bias · the price you pay

Exposure bias · one bad token at inference cascades into gibberish

GOOD ROLLOUT · EVERY STEP STAYS ON THE DATA MANIFOLD



BAD ROLLOUT · A WRONG TOKEN DRIVES THE DECODER OFF-DISTRIBUTION



model has never seen "The cut" in training · subsequent tokens are sampled under distribution shift

MITIGATIONS

- Scheduled sampling · probabilistically feed model's own predictions during late training (Bengio 2015)
- Large-scale pretraining · GPT-3+ rarely exhibits exposure bias because the training distribution is that broad

Exposure bias · concrete cascade

Source: "Le chien a chassé le chat." Target: "The dog chased the cat."

Training (teacher forcing).

1. Input `<s>` → predicts "A" (logP −1.2). Truth is "The". Loss is computed.
2. Next input is the ground truth `"The"`. **Back on track.**

Inference (autoregressive).

1. Input `<s>` → predicts "A".
2. Next input is **"A"** (its own mistake).
3. Conditioned on "A", the model now predicts "dog" (probable continuation of "A").
4. Sequence so far · "A dog…". The model has **never seen** its own mistakes during training, so it has no idea how to recover. Continues "A dog ran away" — completely diverged.

The model trained in a perfect world, tested in a messy one.

Mitigations · scheduled sampling (Bengio 2015), noisy data augmentation, or just sidestep with massive-scale Transformers (2020+).

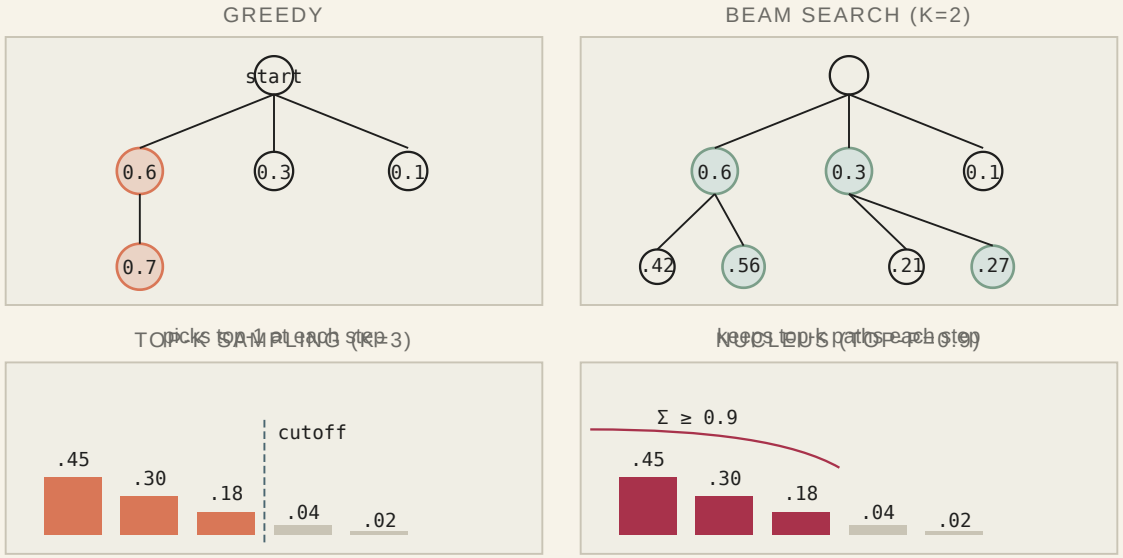
PART 3

Decoding strategies

How to generate at inference

Four common strategies

Decoding — four ways to pick the next token



keeps top-k paths at each step

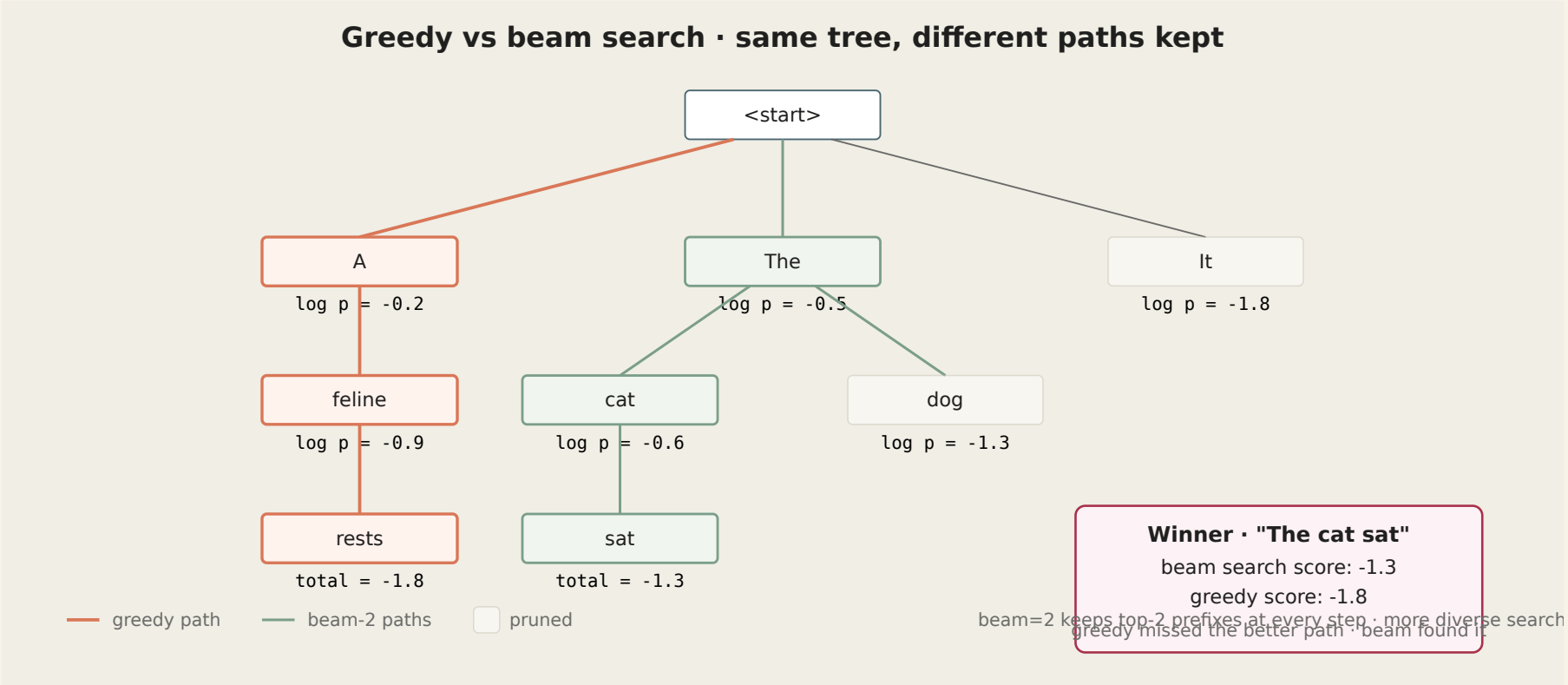
picks top-1 at each step

sample from top 3 only

adaptive cutoff by cumulative probability

Greedy/beam *— deterministic, safe, can be repetitive.* ***Top-p*** *(nucleus) is the 2026 LLM default.*

Decoding · the search tree



Greedy · pick top-1 every step

Simplest: at each step, pick $\arg\max_y P(y \mid \text{history})$.

Fast. Deterministic. Usually **suboptimal** — a slightly-less-likely next token can lead to a much-more-likely full sequence.

Example (simplified):

> *greedy: "The dog is running" (but doesn't quite fit context)*
> *better: "The puppies are running" (total prob higher)*

Greedy fails · the canonical example

Imagine the true best translation is "The cat sits on the mat" (probability 0.7).

At step 1, probabilities are:

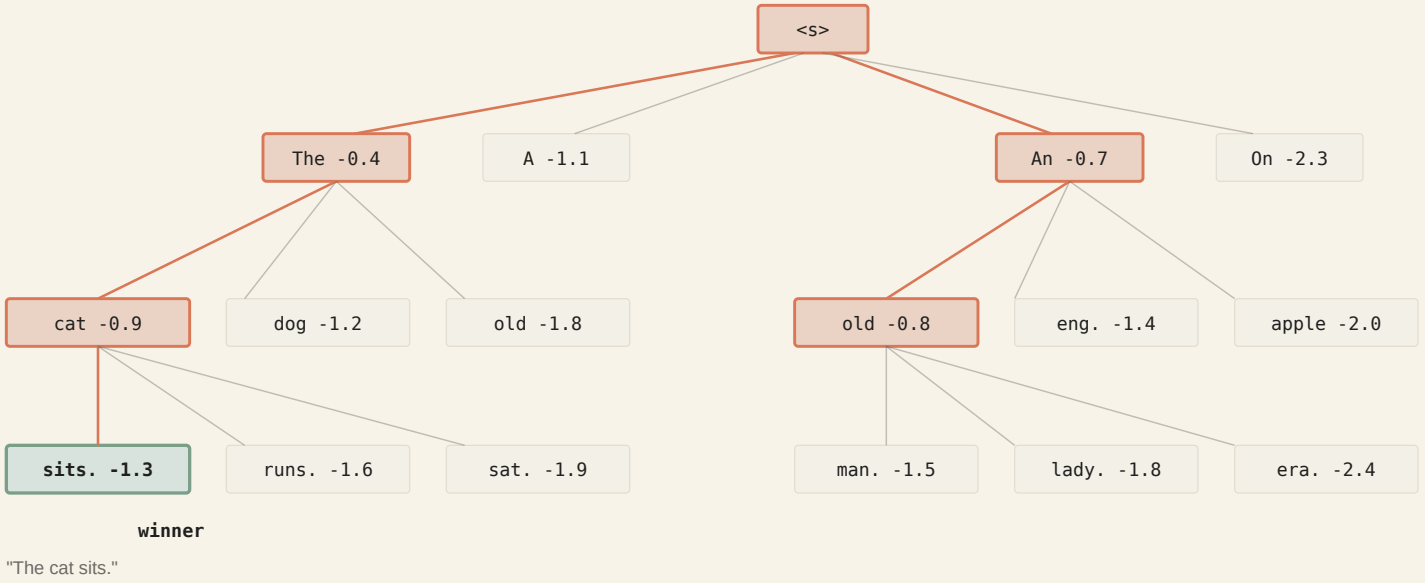
- "The" · 0.6 (leads to the correct sequence)
- "A" · 0.8 (leads to "A feline rests...", probability 0.5)

Greedy picks "A" because it's locally higher. But the full-sequence score is lower than starting with "The".

WATCH OUT

Local optima are not global optima. Greedy decoding is a greedy search on the product of conditional probabilities — it commits at every step. Beam search (next) mitigates by keeping multiple candidates alive until the sequence ends.

Beam search · the tree



Beam search · the team-of-hikers analogy

KEY IDEA

Greedy decoding is **one hiker** taking the steepest step at every fork. They miss bigger peaks that require a less steep early path.

Beam search sends out k **hikers** (the "beam"). At every fork, they explore in parallel. We rank all paths globally and keep the k most-promising survivors. Repeat.

The team's collective best end-of-path score is far closer to the global maximum than greedy's. Cost · $k\times$ compute. Quality · 2-5 BLEU points typically.

From probabilities to log-probabilities

We want the sentence with highest $P(y_1, \ldots, y_T)$:
$P = P(y_1)\, P(y_2|y_1)\, \cdots$

For a 20-word sentence we'd multiply 20 small probabilities → **numerical underflow** (rounded to 0).

Fix · take logs. $\log(ab) = \log a + \log b$. Multiplying probabilities becomes **adding log-probabilities** — much more stable.

$$\text{score}(y) = \sum_{t=1}^{T} \log P(y_t \mid y_{<t})$$

Each $\log P$ is a *negative* number. Higher (less negative) = better.

The length-bias problem

Every $\log P$ is negative → longer sentences accumulate **more** negatives → lower scores → **always preferred shorter**.

SENTENCE	LOGP
"I am"	$-0.5 + (-0.8) = -1.3$

Plain log-prob picks "I am". Wrong.

Length normalization · divide by sentence length raised to α:

$$\text{score}(y) = \frac{1}{T^{\alpha}} \sum_{t=1}^{T} \log P(y_t \mid y_{<t})$$

- $\alpha = 0$ · no penalty (broken).
- $\alpha = 1$ · simple average per token.
- $\alpha \approx 0.6$–0.7 · typical compromise (a soft penalty).

  It makes finished, longer hypotheses **comparable** to short ones — the goal is fair comparison, not always preferring long.

Beam search · keep top-k paths

At each step, maintain k candidate prefixes. Expand each by all next tokens, keep the top k by **length-normalized** log-prob score.

Typical $k = 4$ to 10. More = better quality, slower. Still deterministic given k.

Beam search · worked example with $k = 2$

Vocab · {The, A, cat, dog, sat, ran}. Decoding "The cat sat".

DERIVATION

Step 1. Beams · `[<s>]`. Top-2 next-token log-probs:

TOKEN	LOGP
The	-0.5
A	-0.7

Keep both. Beams · `[<s>, The]` (-0.5), `[<s>, A]` (-0.7).

Step 2. Expand each. For brevity, top extensions:

SEQUENCE	LOGP
`<s> The cat`	-0.5 + -0.6 = **-1.1**
`<s> The dog`	-0.5 + -1.3 = -1.8
`<s> A feline`	-0.7 + -0.9 = -1.6
`<s> A dog`	-0.7 + -1.0 = -1.7

Keep the top 2 · `<s> The cat` (-1.1) and `<s> A feline` (-1.6).

Top-k and nucleus · the improv-comedian analogy

INTUITION

- **Greedy / beam** · a predictable comedian who always says the most obvious next line. Safe, but boring and repetitive.
- **Sampling** · a creative comedian who knows the top 5–10 good responses and sometimes picks the 3rd or 4th — leads to a funnier story.

Top-k and nucleus are two ways to define the **pool of good options** for the comedian to sample from.

Top-k vs nucleus · how the pool is chosen

For open-ended generation (story writing, chat) beam is *too* deterministic — everything sounds the same.

- **Top-k** · pool = the k most-likely tokens. Fixed size. Renormalize, sample.
- **Top-p (nucleus)** · pool = smallest set with cumulative prob $\geq p$. **Dynamic size** — small when the model is confident, large when uncertain.

  When the next-token distribution has one tall bar, nucleus narrows automatically; when it's flat, nucleus widens. Top-k ignores this and uses the same k everywhere.

> IN PRACTICE
>
> **2026 LLM default** — nucleus with $p = 0.9$ or 0.95, plus a temperature knob.

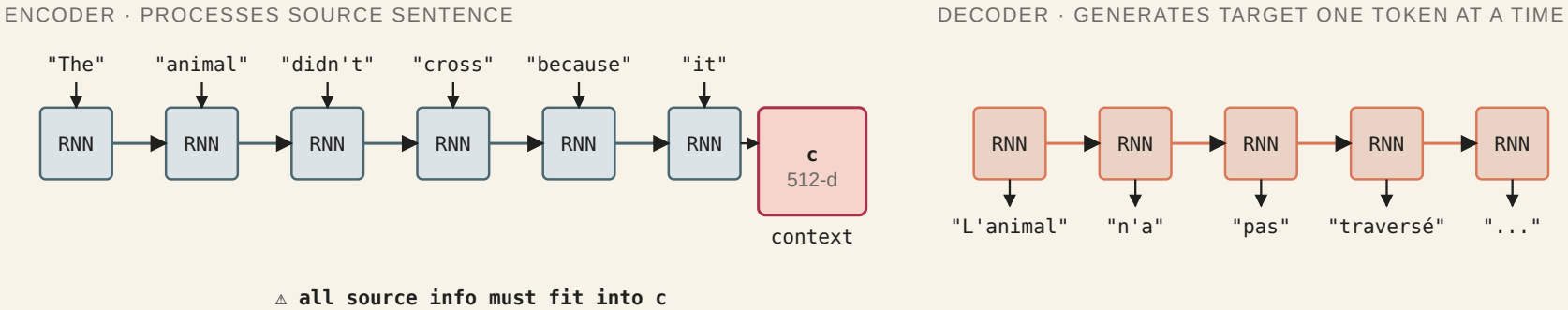
PART 4

The bottleneck that killed Seq2Seq

Why attention became inevitable

The failure mode · source length

Seq2Seq — encoder compresses everything into one vector; decoder generates from it



⚠ **all source info must fit into c**

THE BOTTLENECK PROBLEM

Every word of the source must be encoded into a `512-dim vector c`.

Short sentences — fine. 50-word sentences — you lose detail. 500-word paragraphs — forget it.

Performance of Seq2Seq degrades roughly linearly with source length.

The bottleneck in one sentence

KEY IDEA

The entire source — 5 words or 500 — must compress into one fixed-size context vector. The decoder then generates from that single vector.

For short sentences, fine. For long sentences, the encoder **forgets** the beginning by the time it reaches the end. The decoder has no way to recover what was lost.

Sutskever's own fix · reverse the input

The 2014 paper itself found that **reversing the source** improved BLEU by several points:

`"I am happy"` → *encoded in order*
`"happy am I"` → *encoded reversed*

Why? The last source words (now first) are closest to where the decoder begins generating — less path length for that information to travel through the hidden-state chain.

INTUITION

A hack that works is a sign of a problem waiting to be solved properly. Reversing the input "fixed" the bottleneck by shifting where the leakage happens, not by removing it.

The obvious next step

If one context vector can't hold all source info, **don't use one context vector**.

Instead, let the decoder *look at* all the encoder hidden states — and decide which ones to focus on for each target step.

KEY IDEA

That is **attention**. Bahdanau et al. 2014 — the paper that launched a decade of NLP. Next lecture.

PART 5

Applications of classic Seq2Seq

Even in 2026, some parts survive

Where Seq2Seq-like ideas still ship

- **Speech recognition** · Whisper uses encoder-decoder with attention (the full recipe).
- **Machine translation** · Transformer encoder-decoder is *literally* Seq2Seq with attention instead of a context vector.
- **Code generation** · same family, different data.
- **Summarization** · T5, BART — encoder-decoder Transformers.

> IN PRACTICE
>
> The Seq2Seq **pattern** (encoder → context → decoder) is everywhere. Only the *implementation* of "context" changed: fixed vector (2014) → attention (2015) → self-attention (2017) → … → your favorite 2026 LLM.

Summary · Lecture 11 — summary

- **Seq2Seq** · encoder-decoder for variable-length input → variable-length output.
- **Context vector** is the encoder's final hidden state — a compressed summary.
- **Teacher forcing** trains the decoder with ground-truth history; **exposure bias** is the price.
- **Decoding** — greedy, beam (length-normalized), top-k, **nucleus (2026 default)**.
- **The bottleneck** — one fixed vector for all source info; BLEU collapses past ~30 tokens.
- **Attention** is the fix · next lecture opens Module 6.

Read before Lecture 12

Prince Ch 12 (early sections on attention and QKV).

Next lecture

The Attention Mechanism — Bahdanau additive · Luong multiplicative · the query-key-value abstraction · scaled dot-product and why $\sqrt{d_k}$ · self-attention.

NOTEBOOK

Notebook 11 · `11-seq2seq-nmt.ipynb` — tiny English→French translator with an LSTM encoder-decoder; no attention yet. Next notebook adds attention and you'll see the BLEU gap close.