Sequences and Language Basics

Words come in order, and order changes meaning. The RNN idea, why it forgets, and

Introduction

A photo is one frozen moment, but language is a sequence: words arrive one after another, and order changes meaning. "The dog bit the man" and "The man bit the dog" use the same words and mean opposite things. A model that treats a sentence as a bag of words — counting words but ignoring order — cannot tell them apart. Sequence models exist to remember what came before while reading what comes next.

The first popular idea was the **recurrent neural network (RNN)**: it reads one word at a time and carries a **hidden state** — a small summary of everything read so far — from step to step. At each new word it mixes the word with the hidden state and updates the summary. In theory this remembers the whole sentence. In practice, the summary fades: by the end of a long sentence the network has almost forgotten the beginning. This is the same vanishing-signal problem from DL.3, now stretched across time instead of layers.

Two fixes changed the field. **LSTM** (long short-term memory) gave the network gates — little doors that decide what to keep, what to forget, and what to add to the summary — so important information survives long sentences. Then the **transformer** dropped step-by-step reading altogether: it looks at the whole sentence at once and uses **attention** to decide, for each word, which other words matter most to it. Transformers power today's chatbots and translators, and they train far faster because nothing has to wait for the previous step.

In this lesson you will trace a sentence through an RNN, see why it forgets, and understand attention with one small example.

Where beginners get stuck: they treat the hidden state like a perfect memory, as if the network writes down every word and reads it back later. It does not. The hidden state is a fixed-size bundle of numbers that gets rewritten at every step — more like a summary you keep re-writing in your head than a recording. Each rewrite keeps the gist and drops detail, which is why the beginning fades on long sentences. LSTM gates and attention are both tricks for deciding what survives each rewrite. *(Roman Urdu: Jumlay main lafzon ki tarteeb matlab badal deti hai — is liye model ko pehle wale lafz yaad rakhne partay hain; transformer aik sath poora jumla dekh kar faisla karta hai ke kis lafz par zyada dhyan de.)*

Key points

- **Order matters in sequences.** "Dog bites man" vs "man bites dog" — same words, opposite meaning. A model must remember order, not just count words.
- **An RNN reads one step at a time and carries a hidden state.** The hidden state is the network's running summary. Each new word updates it: new summary = bend(old summary + new word). The final summary drives the prediction.

- **RNNs forget long sentences.** The learning signal must travel backwards through every step, and it fades — the network effectively remembers only the last few words. Long-range links (like a pronoun referring to a name ten words back) get lost.
- **LSTM adds gates to the summary.** Three learned doors: forget (throw away stale summary), input (add the new word's information), output (decide what to pass on). Gates let important facts survive many steps, which is why LSTMs beat plain RNNs on longer text.
- **Attention asks: which words matter for this word?** For each word, the model scores every other word by relevance and takes a weighted mix. In "the animal crossed the street because it was tired", attention lets "it" pull strongly from "animal", not from "street".
- **A transformer reads everything at once.** No step-by-step waiting, so training parallelises beautifully on GPUs. Stacked attention layers plus position information (so order is not lost) make transformers the standard for language today.
- **The price of transformers is size.** They need huge data and compute to shine — the same hunger from DL.1, at a bigger scale. For short sequences and small data, simpler models still win.
- **Words must become numbers before any of this works.** Each word is first turned into a vector (a list of numbers) called an embedding. Similar words get similar vectors — "cat" and "dog" sit closer together than "cat" and "car" — so the network starts with a sense of word meaning, not just word identity.
- **Transformers still need order information.** Because a transformer reads all words at once, it cannot tell word order unless order is added back in. Position information (positional encoding) is added to each word's vector, so "dog bites man" and "man bites dog" still get different representations.

Examples

Example 1 — An RNN reading a sentence

Sentence: "The cat sat on the mat." Hidden state starts empty.

- Step 1 "The": summary = "a sentence started".
- Step 2 "cat": summary = "about a cat".
- Step 3 "sat": summary = "cat sat".
- Step 4 "on": summary = "cat sat on something".
- Step 5 "the mat": summary = "cat sat on the mat" → prediction: "a sitting cat scene".

Each step's summary is built from the previous one. Now imagine a 40-word sentence: by step 40 the summary has been rewritten 40 times, and the first words' influence has faded to almost nothing. That fading is the RNN's core weakness.

Example 2 — Attention resolving "it"

Sentence: "The trophy did not fit in the suitcase because it was too big." Question: what does "it" mean? A human says "the trophy". Attention does the same mechanically: when processing "it", the model scores "trophy" high and "suitcase" low, and mixes mostly "trophy" into "it"'s representation. Change the ending to "because it was too small" and the scores flip — "it" now pulls from "suitcase". One mechanism, both readings, decided by context.

Example 3 — When to use what

- **Tiny data, short sequences (a few words):** a simple model or a small RNN is enough, and it trains in minutes.
- **Medium text, clear structure:** an LSTM handles longer sentences without forgetting the important parts.
- **Big text, best quality:** a transformer, ideally a pre-trained one reused for your task (transfer learning again — the same trick as DL.4, now for language).

Example 4 — One word, two meanings (worked by context)

The word "bank" means different things in these sentences:

- "She sat by the river bank." → bank = the side of a river.
- "She deposited money in the bank." → bank = a financial institution.

A bag-of-words model sees the same word "bank" in both and cannot tell them apart. A sequence model can: in the first sentence, the nearby words "river" and "sat" pull "bank" toward the river meaning; in the second, "deposited" and "money" pull it toward the money meaning. That pull is attention in action — the meaning of a word is decided by the words around it.

Example 5 — LSTM gates saving a name across a sentence

Sentence: "Ali put his keys on the table, walked to the kitchen, made tea, and then he picked them up." To resolve "them", the network must still know "keys" many words later.

- **Forget gate at "walked to the kitchen":** the summary about "put" and "table" is stale, but "keys" is still needed → forget gate keeps the keys information, drops the rest.
- **Input gate at "made tea":** tea-making is new information, but it does not replace the keys → input gate adds only a little, and does not overwrite the keys.
- **Output gate at "them":** the network reads out the surviving keys information to resolve "them" = the keys.

Without gates, each rewrite would wash "keys" away. With gates, the important fact survives the whole sentence — that is the LSTM's whole advantage, shown in one sentence.

Example 6 — Real world: translation and subtitles you use every day

Type a sentence into a translation app, or turn on auto-subtitles for a video, and a transformer is doing the work. It reads the whole sentence at once, uses attention to link pronouns and verbs to the right words (the "it" trick from Example 2), and produces the translation in the other language — all in under a second, because nothing waits step by step. The same machinery powers chatbots answering questions and email apps finishing your sentences. Language models stopped being research demos when transformers made them fast enough to run for everyone.

Quick recap

- Sequences need order memory: the same words in a different order mean different things.
- An RNN carries a hidden-state summary from step to step, but it fades on long sequences.

- LSTM gates decide what to keep, forget and add, so key facts survive longer.
- Attention scores which words matter for each word; transformers read the whole sequence at once and train fast on GPUs.
- Bigger sequence models need bigger data and compute; match the model to your data size.

Common mistakes

1. **Treating a sentence as a bag of words** — counting words and ignoring order destroys meaning. Fix: use a sequence model, or at least word-order features, for any text task.
2. **Using a plain RNN for long documents** — the beginning will be forgotten. Fix: LSTM for moderate lengths, transformers (or chunking) for long ones.
3. **Training a transformer from scratch on a tiny dataset** — it will memorise and fail. Fix: reuse a pre-trained language model and fine-tune it on your small data.

1. **Feeding a whole long document in one go** — a transformer has a limit on how many words it reads at once, and anything past it is cut off or forgotten. Fix: split long documents into chunks, process each chunk, and combine the results.
2. **Training only on one style of text and expecting it to generalise** — a model trained only on formal news text will stumble on casual chat messages, slang and short forms. Fix: match your training text to the text your users actually write.

Practice

1. Write two sentences with the same words in a different order and different meanings. Explain in one line why a bag-of-words model fails on them.
2. In "Sara told Ayesha that she won the prize", who does "she" most likely mean? Which word should attention score highest for "she", and why?
3. You have 500 short customer reviews to classify as happy or unhappy. Would you train a transformer from scratch? Give your answer and a better plan.

1. In "Ayesha gave Bilal her book because she trusted him", who does "she" mean, and which word should attention score highest for "she"? Give one reason.
2. A plain RNN must read a 200-word product description and answer a question about its first sentence. Explain in two sentences why it will struggle, and name the better model type.
3. You have 1,000 short Roman Urdu comments to sort as positive or negative. Would you train a transformer from scratch? Write your answer and your better plan in two sentences.

Practice quiz

1. Why can't a bag-of-words model tell 'dog bites man' from 'man bites dog'?

- It counts words but ignores their order
- It is too slow

- It needs more words
- It cannot read

Answer: It counts words but ignores their order

2. What is an RNN's hidden state?

- A secret password
- A running summary of everything read so far, carried from step to step
- The last word only
- The training data

Answer: A running summary of everything read so far, carried from step to step

3. What does attention do in a transformer?

- It reads faster
- It scores which other words matter most for each word
- It forgets old words
- It translates languages

Answer: It scores which other words matter most for each word

---

Watch the video lesson: https://youtu.be/ka_57VI6oFI

Read online: https://scopetreasure.com/notes/dl-sequences-language · More free tools and notes at https://scopetreasure.com