

Sequence Models

Study Guide

by Richard J. Young, Ph.D.

richard@deepneuro.ai

Chapter 14 of "The NeuroAI Handbook"

How models learned to handle order, memory, and long-range meaning: from RNNs through attention to the Transformer.

One-sentence thesis. The whole history of sequence models is one long fight against distance, getting information from early in a stream to influence a decision made much later, which recurrence solved partially and attention solved directly.

PART 1: The Sequence Problem

Why Order and Distance Are Hard

Start with the data, not the network. Language, speech, and a patient's record all arrive as an *ordered stream* where meaning depends both on sequence ("the dog bit the man" versus "the man bit the dog") and on elements that sit far apart ("the cat chased the mouse because *it* was fast"). A model that treats a sentence as a bag of words throws away exactly the structure that carries the meaning.

The brain faces the same puzzle first at the level of *segmentation*. Continuous speech has no reliable silences between words; the pauses you seem to hear are largely constructed by your brain. Infants solve this with **statistical regularities**, tracking transitional probabilities between syllables, and adults segment at roughly **150 ms** per word. Tokenization is the computational version: chop the stream into discrete units a network can consume.

Two empirical laws govern those units. **Zipf's law**: a handful of words dominate ("the" is about 7 percent of English text). **Heaps' law**: vocabulary grows sublinearly with corpus size, $|V| = kN^\beta$ with $\beta \approx 0.5$ to 0.7, so new words keep appearing but ever more slowly.

The N-gram Baseline, and Why It Broke

Before neural nets, a **language model** was a counting exercise. It answers one question, how likely is the

next word given the ones before it, and the *Markov assumption* truncates the history to the last $n - 1$ words. In symbols,

$$P(w_i | w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i)}{\text{Count}(w_{i-1})}$$

estimated by tallying a corpus. It works until it doesn't. A trigram model over a 50,000-word vocabulary has $50,000^3 \approx 125$ trillion possible entries; no corpus fills that table. So unseen phrases get probability **zero**, and, worse, the model cannot *generalize*: knowing "the cat sat" tells it nothing about "the dog sat," because "cat" and "dog" are just different symbols. Smoothing patches the zeros but never the generalization.

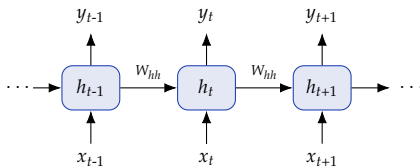
The neural fix. Replace discrete symbols with dense **embeddings**, where similar words sit near each other (famously, king - man + woman $\approx$ queen). Now "cat" and "dog" share structure, and the model generalizes by similarity. That single move, from counting to continuous representation, is what made everything after it possible. Model quality is scored by **perplexity**, the weighted average number of choices the model is torn between (lower is better, and Perplexity = $2^{\text{cross-entropy}}$).

The Idea: a Running Summary

How can a fixed-size network read an input of any length while remembering what came before? A **Recurrent Neural Network** answers with a *hidden state*: a running summary that is updated at every step and fed back into the next one. Read a token, fold it into the summary, repeat. In symbols the update is a single recurrent equation,

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t + b_h)$$

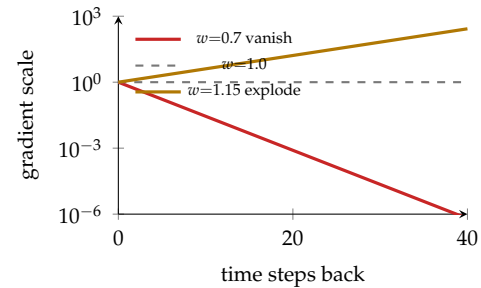
where the *same* weight matrix W_{hh} is reused at every time step. This is a stripped-down model of **persistent activity** in the prefrontal cortex, where neurons keep firing after a stimulus is gone, holding information “online” through local recurrent loops.



Unrolled through time: one shared weight W_{hh} carries the summary forward, the source of both its memory and its trouble.

When Memory Fades

Because that one matrix is applied over and over, training by backpropagation-through-time multiplies many copies of it together. If its influence is a bit less than one, the gradient shrinks exponentially and long-range signals **vanish**; a bit more than one, and it **explodes**. Bengio and colleagues (1994) and Pascanu, Mikolov, and Bengio (2013) made this precise: vanilla RNNs simply cannot connect events far apart.



Repeated multiplication is unforgiving: weights off 1.0 send gradients to zero or infinity.

Gates: Learning What to Keep

The **LSTM** (Hochreiter and Schmidhuber, 1997) fixes this by adding a protected *cell state* plus three learned gates, a **forget** gate (what to discard), an **input** gate (what to write), and an **output** gate (what to expose). The gates act like valves, letting relevant information ride the cell state across long spans untouched. The **GRU** (Cho et al., 2014) merges these into update and reset gates, cheaper and often just as good.

ANALOGY

The LSTM cell state is a conveyor belt running the length of the sequence. Information rides along largely undisturbed; the gates are workers who decide what to place on the belt, what to take off, and what to read, so a fact from step 2 can still be intact at step 200.

Brain parallel. These gates mirror the **basal ganglia**, which gate what enters and leaves prefrontal working memory, opening for relevant input and clearing stale content. A **bidirectional RNN** (Schuster and Paliwal, 1997) adds a second pass from right to left, so a decision at each point can also use context from the future, essential when a word’s role depends on what follows.

The Bottleneck, and the Fix

Even an LSTM must squeeze the entire input into one fixed-size vector before the output can use it, an information bottleneck that gets tighter as sequences grow. The question that cracked the field open: why force the model to *remember* everything when it could just *look back* at the original? **Attention**, introduced by Bahdanau, Cho, and Bengio (2015) for translation, lets the model soft-search the whole input and pull out whatever is relevant right now, regardless of distance.

ANALOGY

The cocktail-party effect: in a noisy room you lock onto one voice and suppress the rest. Your fronto-parietal networks amplify the neural signal you care about. Attention is that spotlight, computed rather than felt.

Query, Key, Value

Here is the mechanism without the algebra first. Each position emits a **Query** (what am I looking for?); every position also advertises a **Key** (what do I offer?) and carries a **Value** (my actual content). Match a query against all keys, turn the match scores into weights that sum to one, and return the weighted blend of values. When the model processes “it” in “the cat chased the mouse because it was fast,” the query for

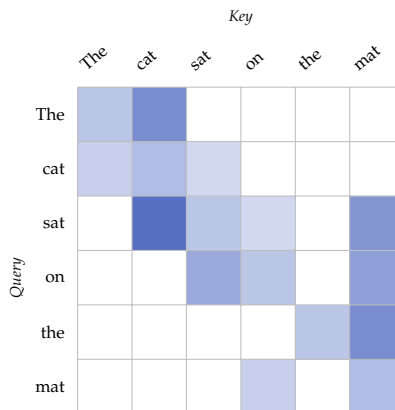
“it” matches the key for “mouse,” and the value for “mouse” flows in. Formally,

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

The dot product $QK^\top$ scores relevance; dividing by $\sqrt{d_k}$ keeps those scores from growing so large that softmax saturates and its gradient dies. The brain analogies are unusually clean: the query is a top-down goal, keys are sensory features, values are the representations, softmax is **lateral inhibition** (winners suppress losers), and multiplying a value by its weight is **gain modulation**, the same multiplicative boost attention gives sensory neurons.

Many Heads

One set of weights can track only one kind of relationship. **Multi-head attention** runs the mechanism several times in parallel with different learned projections, so one head can follow subject-verb links while another resolves pronouns and a third tracks position. In the original Transformer, d_{model} is 512, split across 8 heads of $d_k = 64$ each, roughly the cost of one head but far richer. This echoes the brain’s parallel visual streams, V4 for color, MT for motion, IT for identity, all reading the same input for different features.



A self-attention matrix for “The cat sat on the mat.” Each row (a query word) attends to columns (key words); darker means higher weight. Note “sat” reaching back to its subject “cat” and forward to its location “mat,” plus the elevated diagonal.

Dropping Recurrence Entirely

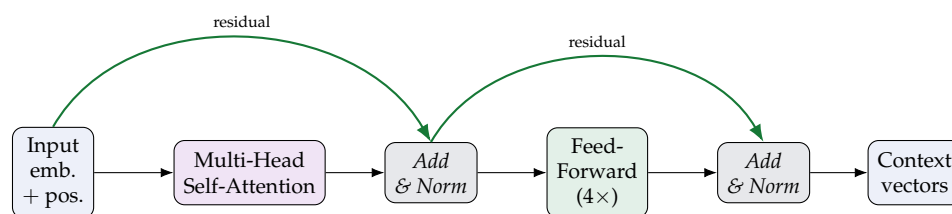
If attention already gives every position direct access to every other, do we still need the recurrent loop at all? Vaswani et al. (2017), in *Attention Is All You Need*, answered no. Removing recurrence removes the sequential dependency that forced RNNs to process one step at a time, so a Transformer can compute all positions **in parallel**, which is what let models scale to billions of parameters. It reached **41.0** BLEU on English-to-French translation in far less training time than recurrent rivals. The architecture pairs an **encoder** (builds a context-rich representation) with a **decoder** (generates output one token at a time).

Inside a Block

Each layer stacks two ideas, both stabilized by two tricks. First a multi-head self-attention sub-layer gathers context; then a position-wise **feed-forward network** transforms each position independently, expanding the width about 4x (512 to 2048) with a GELU nonlinearity before projecting back. Around each sub-layer sit a **residual connection** and **layer normalization**. The residual is the quiet hero: writing

$$x_{l+1} = x_l + f_l(x_l)$$

gives gradients a direct path around every layer (so



One Transformer encoder block. Self-attention mixes context, the feed-forward net transforms each position, and residual + layer-norm keep gradients flowing. Stack these, and depth builds abstraction.

Model	Long-range memory	Parallel training	Core mechanism	Watch out
Vanilla RNN	Weak	No (serial)	Reused hidden state	Vanishing gradients
LSTM / GRU	Moderate	No (serial)	Gated cell state	Still a fixed bottleneck
Transformer	Strong	Yes	Self-attention	$O(n^2)$ cost, no built-in order

100-plus-layer models still train) and lets a layer *refine* rather than replace its input. Layer norm rescales across features per token, which, unlike batch norm, is indifferent to batch size and sequence length.

Telling Position Without Order

Parallelism costs the model its sense of sequence, so order must be injected. **Positional encodings** add sine and cosine signals of different frequencies to each embedding, giving every position a unique, smoothly varying fingerprint the model can use to reason about relative distance and to extrapolate to lengths it never saw in training.

The Brain as a Prediction Machine

Zoom out and a theme emerges. Training a language model to predict the next token is exactly **predictive coding**: generate an expectation, measure the mismatch, update. The brain's **N400** response, a spike when an unexpected word arrives, is its prediction-error signal. And just as auditory, phonemic, and semantic processing span a hierarchy of timescales in cortex, lower Transformer layers capture local syntax while higher layers integrate long-range, abstract meaning. This same machinery now reads clinical time series, learning disease progression from EHR sequences to flag events like sepsis hours early, though irregular sampling and missing values remain open challenges.

1. Why do n-gram models fail to generalize from “the cat sat” to “the dog sat,” and what representational change fixes it?
2. Explain in one sentence why the *same* recurrent weight matrix is the cause of the vanishing gradient problem.
3. In the attention formula $\text{softmax}(QK^\top / \sqrt{d_k})V$, what breaks if you drop the $\sqrt{d_k}$ term?
4. What single architectural change let Transformers train in parallel, and what did the model lose as a result that positional encodings restore?
5. Name the brain analogue the chapter draws for each of: the RNN hidden state, LSTM gates, and softmax over attention scores.

If You Remember One Thing

Recurrence remembers the past by *compressing* it into a single evolving state, and that compression is both its memory and its ceiling. Attention refuses to compress: it keeps the whole input available and lets each position reach directly for whatever it needs. The Transformer is what you get when you trust attention completely and throw the recurrence away.

Richard J. Young, Ph.D. · richard@deepneuro.ai · Senior AI Research Scientist, UnitedHealth Group · UNLV Lee Business School · ORCID 0000-0002-1109-7552