From N-grams to LSTMs

N-grams · Word2Vec · RNNs · LSTMs · Motivation for Attention

The core problem

"The cat sat on the mat"

?

Representation

How do we turn words into numbers that capture meaning?

Sequence modeling

How do we capture context and word order?

N-grams (1990s) — **Word2Vec (2013)** — **RNN/LSTM** — **Transformer (2017)**

N-gram language models

$$P(w_t \mid w_{t-n+1}, \ldots, w_{t-1}) = \frac{\text{count}(w_{t-n+1} \cdots w_t)}{\text{count}(w_{t-n+1} \cdots w_{t-1})}$$

The cat sat on the ???

bigram context

trigram context

Bigram:

$$P(\text{mat} \mid \text{the}) = \frac{\text{count(“the mat”)}}{\text{count(“the”)}}$$

Trigram:

$$P(\text{mat} \mid \text{on, the}) = \frac{\text{count(“on the mat”)}}{\text{count(“on the”)}}$$

Idea: estimate probabilities by counting how often word sequences appear in a large corpus

N-gram limitations

$\overbrace{\text{"The cat that sat on the mat near the door by the window was \_\_\_\_\_\_"}}^{\text{12 words apart — trigram window can't see "cat"}}$

1. Fixed context window

A trigram only sees 2 previous words. Long-range dependencies are invisible.

2. Curse of dimensionality

Vocabulary $V = 50\text{k} \Rightarrow$ possible 5-grams: $50\text{k}^5 = 3 \times 10^{23}$. Most never observed.

3. Data sparsity

"armadillo aardvark" has count 0 in most corpora. Smoothing helps but doesn't solve it.

4. No generalization

Knowing "dog sat on" doesn't help predict after "cat sat on." Words are discrete symbols.

We need a way to represent words so that similar words share information.

One-hot encoding

Vocabulary

1. cat
2. dog
3. sat
4. the

$\vdots$

50k. zoo

One-hot vectors

$$\text{cat} = \underbrace{[\,1 \mid 0 \mid 0 \mid 0 \mid \cdots \mid 0\,]}$$

$$\text{dog} = [\,0 \mid 1 \mid 0 \mid 0 \mid \cdots \mid 0\,]$$

$$\text{sat} = \underbrace{[\,0 \mid 0 \mid 1 \mid 0 \mid \cdots \mid 0\,]}_{\text{50,000 dimensions}}$$

No similarity

$$\text{cat} \cdot \text{dog} = 0$$

$$\text{cat} \cdot \text{banana} = 0$$

Every pair is equally different!

Huge & sparse

50k-dim vectors with exactly one non-zero entry

We need: dense, low-dimensional vectors
where similar words have similar representations

Word2Vec — the distributional hypothesis

"You shall know a word by the company it keeps"

— J. R. Firth, 1957

Words appearing in similar contexts have similar meanings:

"The movie was surprisingly **good** and the audience loved it"

"The movie was surprisingly **great** and the audience loved it"

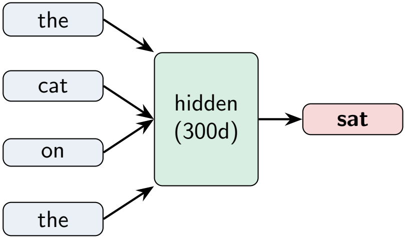
"The movie was surprisingly **excellent** and the audience loved it"

good, **great**, and **excellent** share context $\Rightarrow$
their vectors should be close in embedding space

Word2Vec — CBOW & Skip-gram

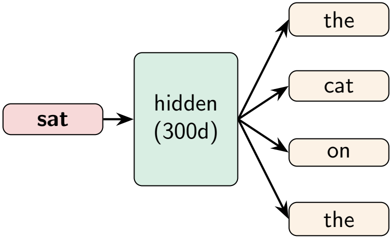
CBOW

Context → Center word



Skip-gram

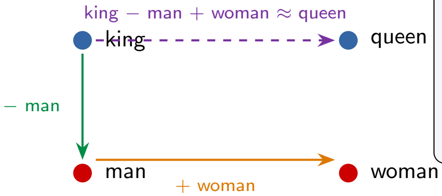
Center word → Context



Result: the hidden layer weights *are* the word embeddings — dense vectors (100–300 dims)

Word2Vec — emergent properties

Vector arithmetic in embedding space



More analogies:

Paris − France + Italy ≈ Rome

bigger − big + small ≈ smaller

walking − walk + swim ≈ swimming

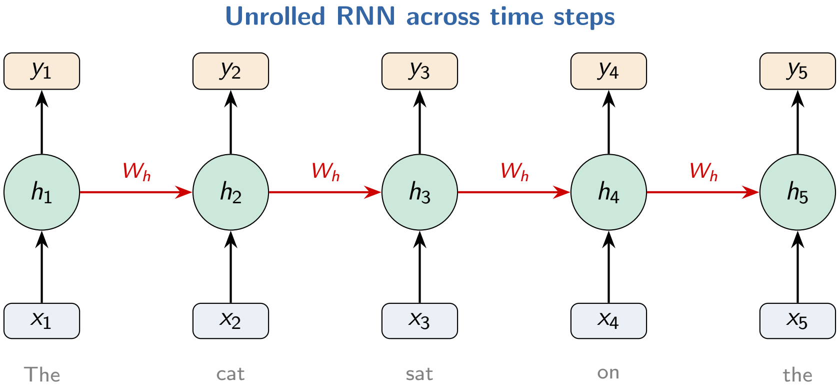
Critical limitation: one vector per word

"I went to the river **bank**" vs. "I deposited money at the **bank**"

Both map to the *same* vector — no polysemy, no context-dependence

We need representations that change based on context. Enter: recurrent neural networks.

Recurrent Neural Networks (RNNs)

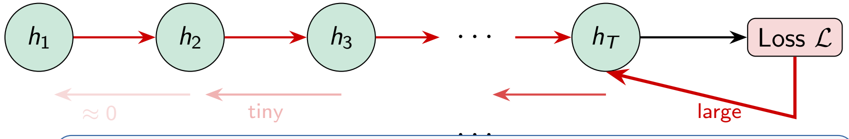


$$h_t = \tanh\left(W_h \cdot h_{t-1} + W_x \cdot x_t + b\right)$$

Key: hidden state h_t is a "memory" — no fixed window!

The vanishing gradient problem

Backpropagation through time



$$\frac{\partial \mathcal{L}}{\partial h_1} = \frac{\partial \mathcal{L}}{\partial h_T} \cdot \prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}$$

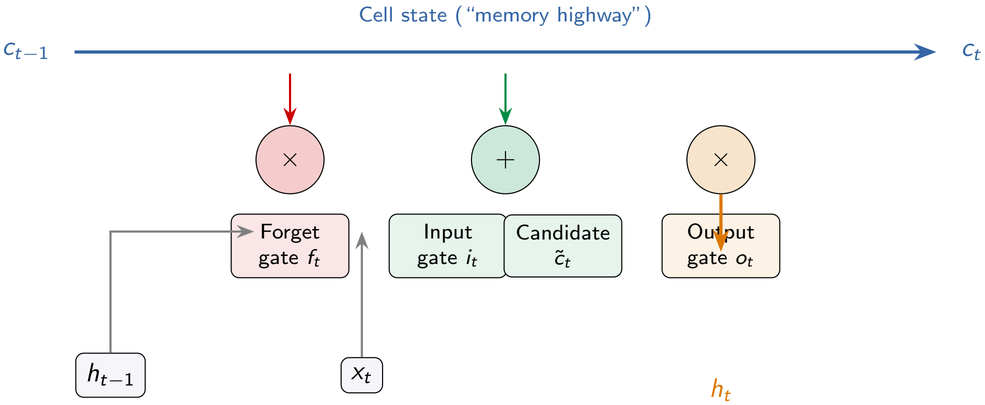
Product of many values $< 1 \Rightarrow$ gradient vanishes

"**The cat** that I saw yesterday in the garden near the old oak tree **was** cute"

The gradient from "was" can't reach "cat" — the RNN *forgets* early tokens

Solution attempt: LSTM (1997) and GRU (2014)

LSTM — Long Short-Term Memory

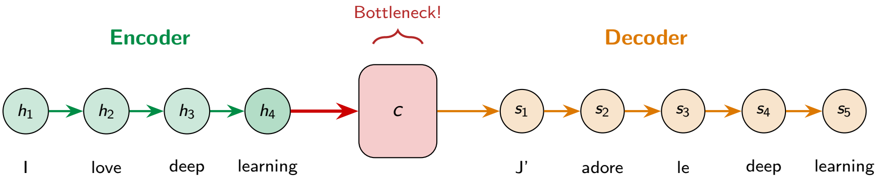


Cell state flows with minimal modification — gradient highway

Gates learn what to forget, store, and output at each step

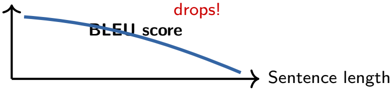
Sequence-to-sequence (Encoder–Decoder)

Machine translation with LSTMs (Sutskever et al., 2014)



The entire input sentence is compressed into a single fixed-size vector c!

Works for short sentences, but performance degrades for long ones.

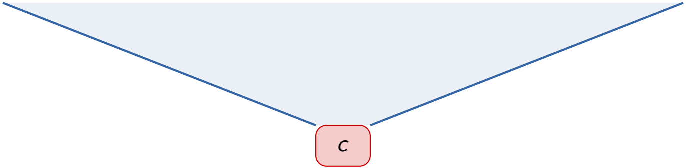


What if the decoder could **look back** at all encoder states, not just c?

→ This is attention!

Why LSTMs are not enough — the bottleneck

"The quick brown fox jumped over the lazy dog that
was sleeping by the fireplace in the old cottage"



512 or 1024 dims

Information lost:

Early tokens get overwritten by later ones as h_t is updated sequentially

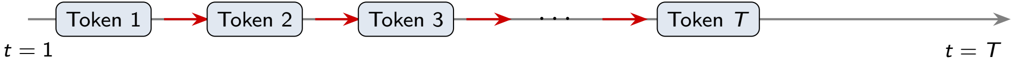
Fixed capacity:

Whether the input is 5 tokens or 500, it must fit in the same vector

Fundamental issue: you can't losslessly compress arbitrary-length sequences into a fixed-size vector

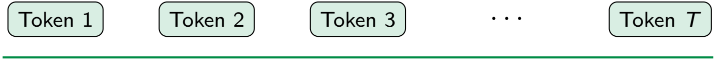
Why LSTMs are not enough — sequential processing

LSTM: must process tokens one at a time



Each step **waits** for the previous one

What if we could process all tokens in parallel?



All processed **simultaneously** on GPU

LSTM: $O(T)$ sequential steps
Can't utilize GPU parallelism
Training is *slow*

Transformer: $O(1)$ parallel steps
Fully utilizes GPU cores
Training is *fast*

Why LSTMs are not enough — long-range dependencies

"The **trophy** didn't fit in the **suitcase** because **it** was too ________"

If **it** = **trophy**:
"it was too **big**"

If **it** = **suitcase**:
"it was too **small**"

Path length from "it" to its referent:

LSTM: $O(n)$ steps

Information must pass through every intermediate hidden state

Signal degrades over distance

Attention: $O(1)$ steps

Direct connection between any two tokens

No degradation over distance

Attention: "let me directly look at any token I need" — regardless of distance

The stage is set

N-grams

Fixed window

No generalization

Data sparsity

Can't scale

Word2Vec

Good embeddings

But static (one vector per word)

No sequence modeling

RNN / LSTM

Sequential ($\times$ parallel)

Bottleneck vector

Long-range deps still hard

We need: **parallel processing**, **direct access to all positions**, **contextual representations**

$\Rightarrow$ **Self-Attention** and the **Transformer** (Vaswani et al., 2017)

Word2Vec — training tricks

Skip-gram objective: $$\max_{\theta} \sum_{t=1}^{T} \sum_{-c \le j \le c,\, j \ne 0} \log P(w_{t+j} \mid w_t;\, \theta)$$

Problem: full softmax

$$P(w_O \mid w_I) = \frac{\exp(v'_{w_O} \cdot v_{w_I})}{\sum_{w=1}^{V} \exp(v'_w \cdot v_{w_I})}$$

Sum over $V = 50\text{k}$ words per update!

$\longrightarrow$

Negative sampling

Instead of all V words, sample k random "negative" words

Binary classification: real context vs. noise

$$\log \sigma(v'_{w_O} \cdot v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n}\left[\log \sigma(-v'_{w_i} \cdot v_{w_I})\right]$$

$f(w)^{3/4})$

Subsampling

Downsample frequent words ("the", "a")

Window size

Larger window = more semantic; smaller = more syntactic

Dimensionality

Typical: 100–300 dims; more data $\rightarrow$ more dims

Beyond Word2Vec — GloVe & FastText

GloVe (Pennington et al., 2014)

Idea: combine count-based and prediction-based methods

Build word co-occurrence matrix X_{ij}, then factorize:

$$w_i^T \tilde{w}_j + b_i + \tilde{b}_j = \log X_{ij}$$

Global statistics + local context

Often matches or beats Word2Vec

FastText (Bojanowski et al., 2017)

Idea: represent words as bags of character n-grams

"where" $\rightarrow$ {⟨wh, whe, her, ere, re⟩}

Word vector = sum of n-gram vectors

Handles unseen words & morphology

Great for morphologically rich languages

Comparison

	Word2Vec	**GloVe**	**FastText**
Training signal	Local context	Global co-occurrence	Local + subword
OOV words	No	No	Yes
Morphology	No	No	Yes
Speed	Fast	Fast	Medium

All three share the same limitation: static embeddings — one vector per word regardless of context

Further reading

N-grams & Statistical NLP

- Jurafsky & Martin, *Speech and Language Processing*, Ch. 3 — N-gram Language

Word Embeddings

- Mikolov et al. (2013), "Efficient Estimation of Word Representations in Vector Space"
- Mikolov et al. (2013), "Distributed Representations of Words and Phrases" (negative sampling)

RNNs & Sequence Models

- Hochreiter & Schmidhuber (1997), "Long Short-Term Memory"
- Cho et al. (2014), "Learning Phrase Representations using RNN Encoder-Decoder" (GRU)
- Sutskever et al. (2014), "Sequence to Sequence Learning with Neural Networks"
- Bahdanau et al. (2015), "Neural Machine Translation by Jointly Learning to Align and Translate"

Questions?

Next: The Transformer Architecture