# 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:**

$$P(\text{mat} \mid \text{the}) = \frac{\text{count}(\text{``the mat''})}{\text{count}(\text{``the''})}$$

**Trigram:**

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

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

# N-gram limitations

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

**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

⋮

50k. zoo

**One-hot vectors**

cat $= \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$

dog $= \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$

sat $= \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}}_{\text{50,000 dimensions}}$

**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:

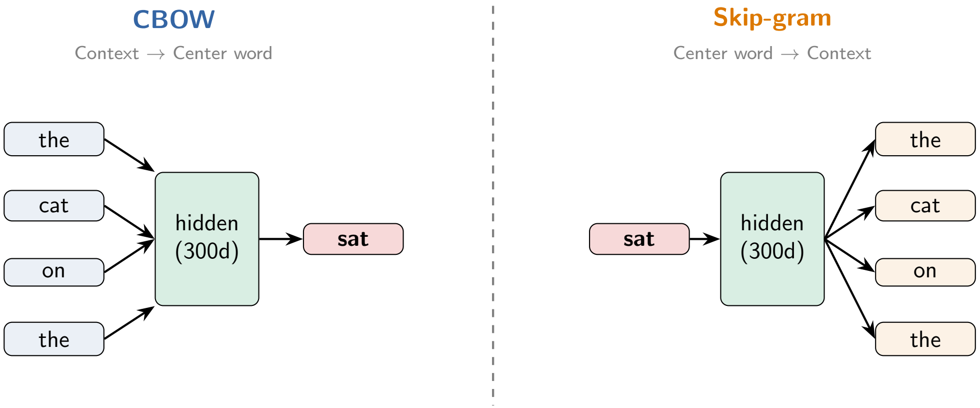
"The movie was surprisingly **good** and the audience loved it"

"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 ⇒
their vectors should be close in embedding space

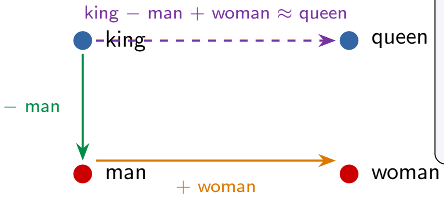
# Word2Vec — CBOW & Skip-gram




**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**

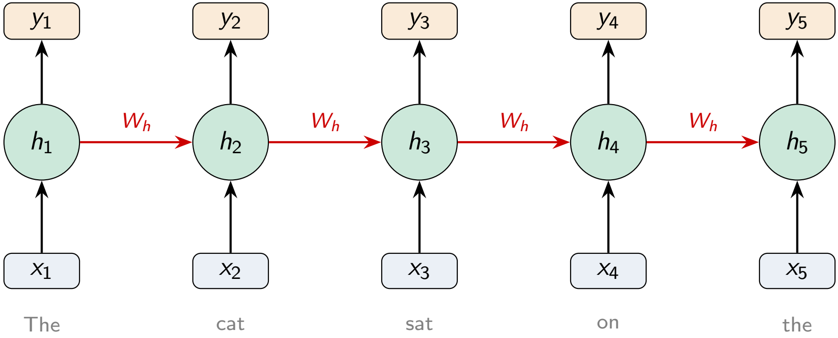
"I went to the river **bank**" vs. "I deposited money at the **bank**"

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)

**Unrolled RNN across time steps**

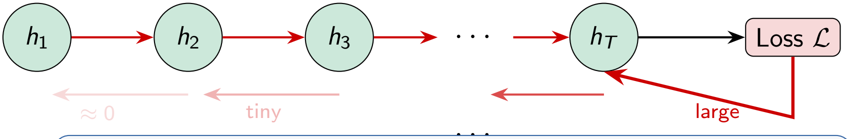



$$h_t = \tanh\left(W_h \cdot h_{t-1} + W_x \cdot x_t + b\right)$$

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

# The vanishing gradient problem

### Backpropagation through time




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

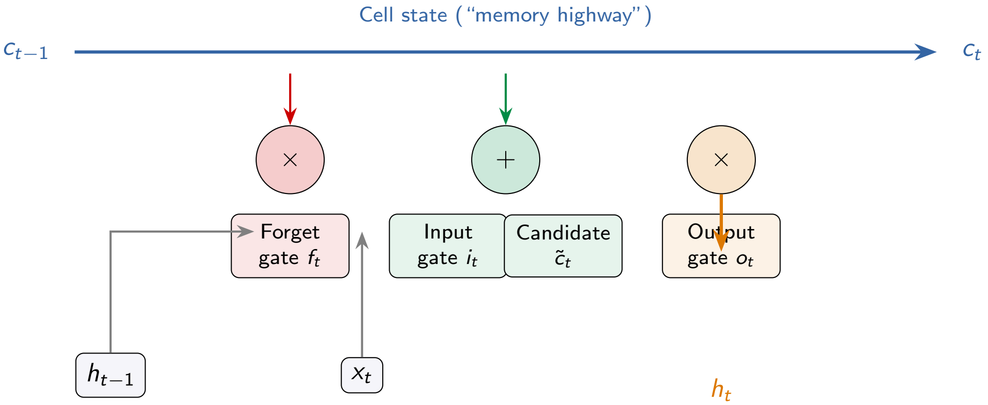
Product of many values $< 1 \Rightarrow$ gradient vanishes

"**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

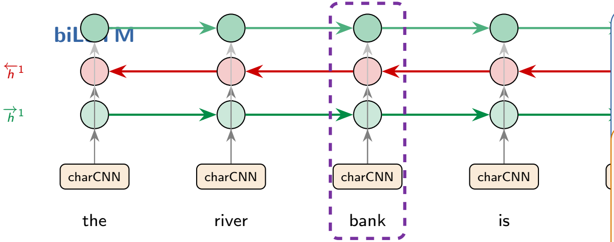
**Gates** learn what to forget, store, and output at each step

# ELMo — context changes meaning

**Peters et al. (2018): "Deep contextualized word representations"**

**Problem:** Word2Vec gives "bank" **one** vector, always

**ELMo:** run sentence through biLSTM → **different** vector each time




**ELMo representation:**

$$\text{ELMo}_k = \gamma \sum_{j=0}^{L} s_j\, h_{k,j}$$

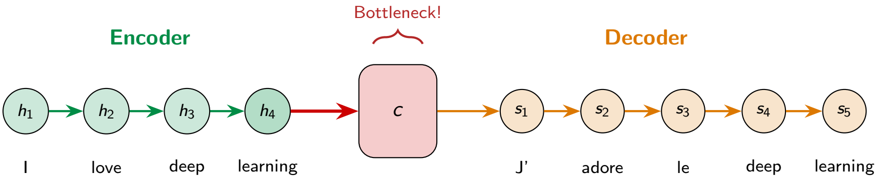
**Impact:** Pre-train biLSTM on large corpus, then **fine-tune** on downstream task:

- SQuAD: +4.7% F1
- Sentiment, NER, coref: new SOTA across 6 tasks

→ The **pre-train + fine-tune** paradigm is born

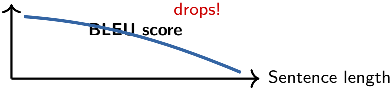
# Sequence-to-sequence (Encoder–Decoder)

**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.

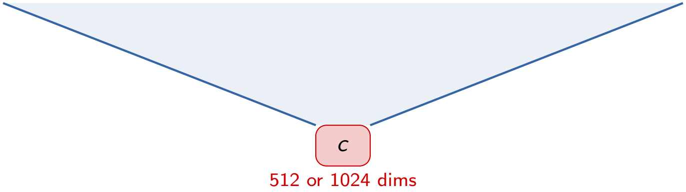



What if the decoder could **look back** at all encoder states, not just $c$?

$\rightarrow$ 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"




**Information lost:**

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

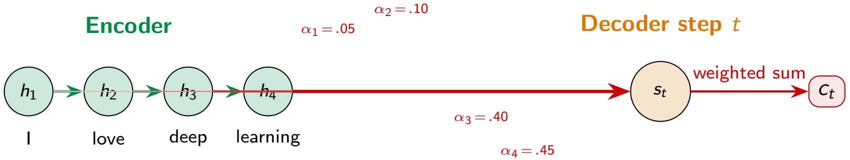
**Fixed capacity:**

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

# Attention — looking back at the source

**Bahdanau et al. (2015): "Jointly Learning to Align and Translate"**




$$\alpha_{ti} = \mathrm{softmax}(e_{ti}) \quad \text{where} \quad e_{ti} = a(s_{t-1}, h_i) \qquad c_t = \sum_i \alpha_{ti} h_i$$

**No more bottleneck!**

Each decoder step gets a **different** weighted view of all encoder states

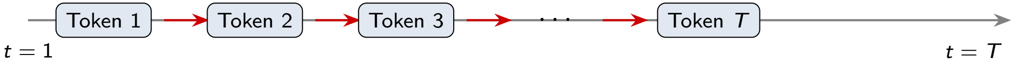
**Alignment is learned**

The model learns which source words to focus on for each output word

What if we applied this idea to *every* token attending to *all other* tokens? → **Self-attention**

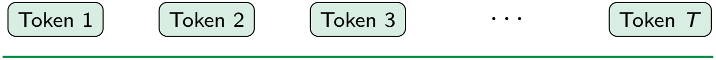
# Why LSTMs are not enough — sequential processing

**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

→

**Word2Vec**

Good embeddings

But static: one vector per word

No sequences

→

**RNN / LSTM**

Sequential ($\times$ parallel)

Long-range deps still hard

→

**Attention**

Fixes bottleneck

But still built on top of RNNs

Still sequential

↓

What if attention is **all you need**? Drop the RNN entirely.

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

**Problem: full softmax**

Normalizing over all $V$ words per update — too slow!

**Negative sampling**

Sample $k$ random "negatives"
Binary: real context vs. noise

$\log \sigma(v'_{w_O} \cdot v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n}\left[\log \sigma(-v'_{w_i} \cdot v_{w_I})\right] \quad k = 5\text{–}20, \; P_n \propto f(w)^{3/4}$

**Subsampling**

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

**Window size**

Large → semantic similarity
Small → syntactic similarity

**Dimensionality**

Typical: 100–300 dims
More data → more dims

Word2Vec trained on Google News: 3M vocabulary, 300-dim vectors, 100B tokens

# Beyond Word2Vec — GloVe & FastText

**GloVe** (Pennington et al., 2014)

Co-occurrence matrix $X_{ij}$ + factorize:

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

Global statistics + local context

**FastText** (Bojanowski et al., 2017)

Words as bags of char n-grams:

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

Handles OOV words & morphology

| | Word2Vec | GloVe | FastText |
|---|---|---|---|
| Training signal | Local context | Global co-occur. | 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 Models

### Word Embeddings

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

### RNNs, Attention & Sequence Models

- Hochreiter & Schmidhuber (1997), "Long Short-Term Memory"
- 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"
- Luong et al. (2015), "Effective Approaches to Attention-based Neural Machine Translation"
- Peters et al. (2018), "Deep contextualized word representations" (ELMo)

# Questions?

Next: The Transformer Architecture