Language Models
How Machines Understand Text

From Bigrams to Transformers

Nipun Batra | IIT Gandhinagar

The Story So Far

Week	Domain	Key Insight
6	Vision	Images = grids of pixels → CNNs
7	Language	Text = sequences of tokens → ?

Today's Agenda

1. **The Core Idea** - Next token prediction
2. **The Counting Era** - Bigrams and N-grams
3. **Word Embeddings** - Words as vectors
4. **The Memory Problem** - RNNs
5. **The Attention Revolution** - Transformers
6. **Modern LLMs** - From GPT to ChatGPT

Part 1: The Core Idea

It's All About Prediction

The One Question

Every language model answers **one simple question**:

> **"Given what I have seen so far, what word comes next?"**

Example: "The capital of France is ___" → **"Paris"**

That's it. **Predict the next word. Repeat until done.**

You Already Use This!

Application	You type...	Suggestion
Phone Keyboard	"I'm running ___"	`late`
Google Search	"how to make ___"	`money`, `pancakes`
Gmail	"Thanks for the ___"	`quick response!`

> All of these are next-word prediction models!

The Mathematical View

"The capital of France is ___" → Probability distribution:

Word	P(word | context)
Paris	0.85
the	0.02
London	0.01
beautiful	0.01
...	0.11

All probabilities sum to 1.0

ChatGPT: Just Prediction!

Prompt	Prediction	Appears to Know
"F = m"	"a"	Physics
"To be or not to"	"be"	Shakespeare
"E = mc"	"²"	Einstein
"print('Hello"	"')"	Python

> If you predict well enough, you **appear** to understand everything.

The Generation Algorithm

```
def generate_text(prompt, model):
    tokens = tokenize(prompt)

    while not done:
        # Step 1: Predict probabilities for ALL possible next tokens
        probs = model(tokens)

        # Step 2: Sample one token
        next_token = sample(probs)

        # Step 3: Add to sequence and repeat
        tokens.append(next_token)

    return tokens
```

That's ALL ChatGPT does!

Part 2: The Counting Era

Bigrams: The Simplest Model

The Simplest Language Model

Idea: Count what letter usually follows each letter.

Training: `aabid`, `priya`, `zeel`, `nipun`

After	Saw	Probability
`a`	`a` once, `b` once	P(a
`z`	`e` once	P(e

This is a **Bigram** model (pairs of 2 characters).

Generating with Bigrams

```
Step 1: Start with "." (beginning token)
        Sample from row "." → Got 'a'

Step 2: Current = 'a'
        Sample from row "a" → Got 'b'

Step 3: Current = 'b'
        Sample from row "b" → Got 'i'

Step 4: Current = 'i'
        Sample from row "i" → Got 'd'

Step 5: Current = 'd'
        Sample from row "d" → Got "." (DONE!)

Result: "abid" ← Looks like a real name!
```

Why Bigrams Fail

Sentence: "Alice picked up the golden key. She walked to the door and tried to open it with the ___"

Model	What It Sees
Bigram	Only "the" (previous word)
Human	"golden key" (from earlier)

> Bigrams have NO MEMORY of earlier context!

N-grams: More Context

Model	Context	Limitation
Bigram	1 word	Too little
Trigram	2 words	Still limited
4-gram	3 words	Better
10-gram	9 words	Storage explodes!

Problem: With vocabulary 50K, 10-gram needs $50K^{10}$ entries!

Part 3: Word Embeddings

Words as Vectors

The Representation Problem

How do we represent words for a neural network?

Bad idea: One-hot encoding

```
"cat" → [1, 0, 0, 0, ... , 0]  (50,000 zeros!)
"dog" → [0, 1, 0, 0, ... , 0]

cat · dog = 0  (orthogonal = unrelated!)
```

But cats and dogs ARE related!

Word Embeddings

Idea: Learn a dense vector for each word!

```
"cat" → [0.2, -0.5, 0.8, 0.1, ...]  (maybe 300 dims)
"dog" → [0.3, -0.4, 0.7, 0.2, ...]

cat · dog = 0.9  (similar!)
```

Embeddings Capture Meaning

Famous example from Word2Vec (2013):

$$\text{king} - \text{man} + \text{woman} \approx \text{queen}$$

The vector arithmetic works because embeddings capture semantic relationships!

Embedding in PyTorch

```
import torch.nn as nn

# Create embedding layer
vocab_size = 50000
embed_dim = 256

embedding = nn.Embedding(vocab_size, embed_dim)

# Get vector for word index 42
word_idx = torch.tensor([42])
vector = embedding(word_idx)  # shape: [1, 256]
```

Part 4: The Memory Problem

RNNs: Passing the Baton

Fixed Windows Aren't Enough

Story: "Alice picked up the golden key. She walked to the door..."

Model Type	Sees	Missing
Fixed window (3 words)	"to the door"	"key"
We need	Everything	-

RNNs: The Relay Race

Idea: Pass information forward like a baton.

Hidden state `h` carries "memory" of previous words.

RNN: The Telephone Game Problem

Sequence Length	Memory Quality
10 words	Clear
50 words	Fuzzy
100+ words	Lost!

> RNNs suffer from "vanishing gradients" — they forget old information!

LSTM and GRU help but don't fully solve this.

Part 5: The Attention Revolution

"Just Look Back!"

The Brilliant Idea (2017)

What if, instead of compressing everything...
We could just **look back** at everything directly?

Approach	Sees	Limitation
Fixed window	Last few words	Very limited
RNN	Blurry summary	Degrades over time
ATTENTION	Any word directly!	None!

Attention: The Library Analogy

You're at a library with a question:

Step	Action
1. Query	"What opens doors?"
2. Scan Keys	"key" (relevant!), "door" (related), "Alice" (not relevant)
3. Read Values	Mostly from "key"!

$$\text{Attention} = \text{softmax}(QK^T/\sqrt{d}) \cdot V$$

Why Attention is Powerful

Text: "The animal didn't cross the street because **it** was too tired."

What does "it" refer to?

Word	Attention Score
animal	0.75
street	0.15
other	0.10

The model **learns** to connect "it" to "animal"!

Self-Attention

Every word attends to **every other word**:

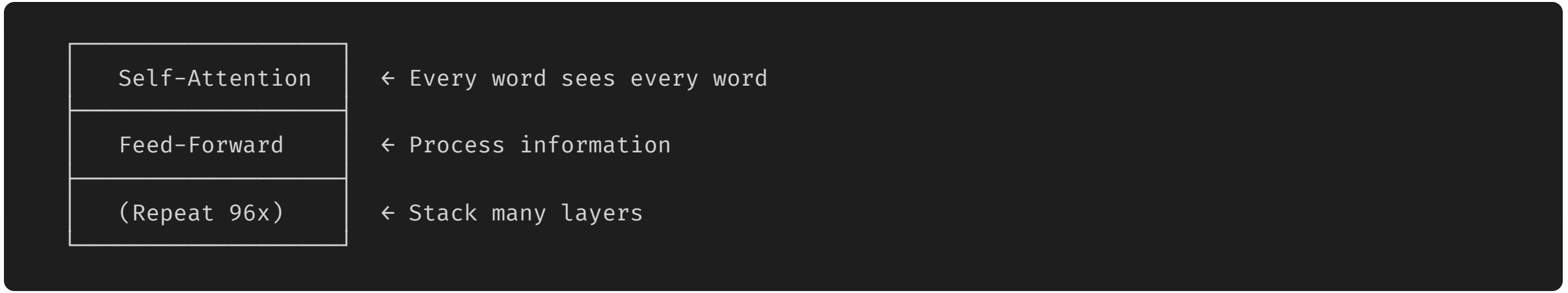
```
"The cat sat on the mat"

"cat" attends to: "The"(0.1), "cat"(0.2), "sat"(0.4), "on"(0.1)...
"mat" attends to: "on"(0.2), "the"(0.3), "cat"(0.3)...
```

All in parallel — no sequential bottleneck!

The Transformer (2017)

"Attention Is All You Need"



GPT-4 has ~120 transformer layers!

Part 6: Modern LLMs

From GPT to ChatGPT

Scaling Up

Feature	Toy Model	GPT-4
Vocabulary	27 (letters)	100,000 (tokens)
Embedding size	2 dims	12,288 dims
Layers	1	~120
Parameters	~1,000	175+ BILLION
Training data	1,000 names	500B+ tokens
Context	3 chars	128K tokens

Same algorithm. Just MUCH bigger.

Tokenization: Not Words, Not Characters

Approach	Example	Problem
Characters	"hello" → 5 tokens	Too slow
Words	"unhappiness" = 1 token	Millions needed
Subwords	"un" + "happiness"	Best of both!

LLMs use ~50K-100K tokens (subwords).

Tokenization Examples

Text	Tokens
"Hello world"	["Hello", " world"]
"ChatGPT"	["Chat", "G", "PT"]
"unhappiness"	["un", "happiness"]

"How many r's in strawberry?" fails because the model sees ["str", "aw", "berry"]!

Training an LLM

```
1. COLLECT DATA
   - Web crawl (trillions of tokens)
   - Books, Wikipedia, code

2. PRE-TRAINING
   - Objective: Predict next token
   - Massive compute (thousands of GPUs)

3. FINE-TUNING
   - Instruction following
   - RLHF (Reinforcement Learning from Human Feedback)
```

RLHF: Making ChatGPT Helpful

Pre-trained model: Great at next-word prediction
Problem: Doesn't follow instructions well

Solution: RLHF

1. Humans rank model responses
2. Train reward model on rankings
3. Fine-tune LLM to maximize reward

This is what makes ChatGPT **conversational**!

Temperature: Creativity Knob

When sampling the next token:

Temperature	Effect	Use Case
0.0	Always pick most likely	Facts, code
0.7	Some randomness	Balanced
1.0+	Very random	Creative writing

```
response = client.chat.completions.create(
    model="gpt-4",
    messages=[ ... ],
    temperature=0.7
)
```

Emergent Abilities

As models get bigger, **new abilities emerge**:

Size	Capabilities
Small (100M)	Grammar, simple completion
Medium (1B)	Factual Q&A, basic reasoning
Large (100B+)	Complex reasoning, code, creativity

All from the same objective: **predict the next token!**

Key Takeaways

1. **LLMs predict the next token** — that's it!
2. **Embeddings** represent words as vectors
3. **Attention** lets models look at ALL context
4. **Transformers** stack attention + feed-forward layers
5. **Scale matters** — same algorithm, more parameters
6. **RLHF** makes models follow instructions

You Now Understand LLMs!

Next: Generative AI - How Machines Create

Lab: Build a small language model, use GPT API

"The question is no longer whether machines can think, but what will they think about."

Questions?