Brief intro to Large Language Models.

Part 2: internals.

What does this stuff mean?

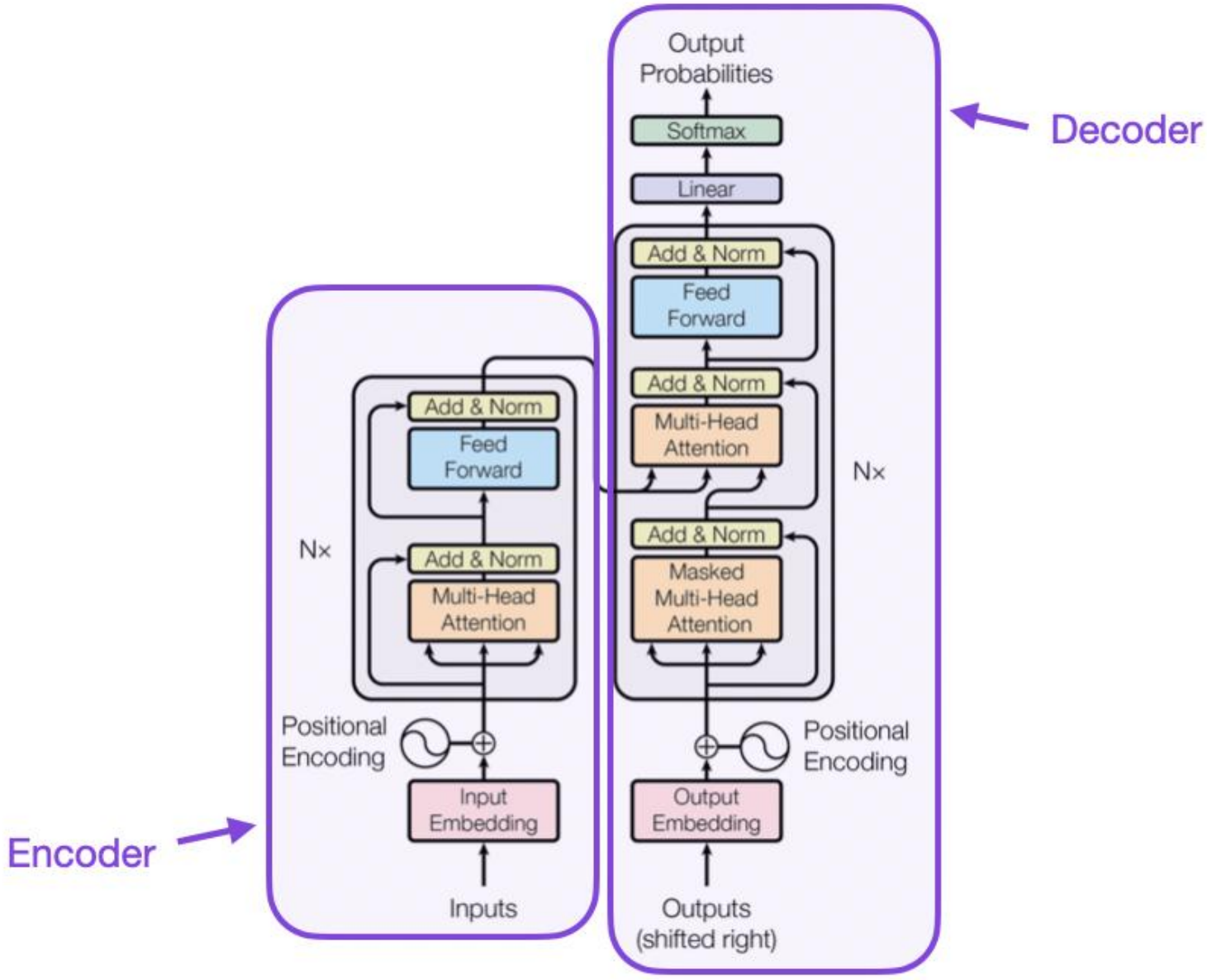


Figure 1: The Transformer - model architecture.

From Jurafsky & Martin:

Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention

---

Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

From Jurafsky & Martin:

1990 Static Word Embeddings

2003 Neural Language Model

2008 Multi-Task Learning

2015 Attention

2017 Transformer

2018 Contextual Word Embeddings and Pretraining

2019 Prompting

Attention as transforming information of word vectors: example for GPT2 (predecessor of GPT3)

When Mary and John went to the store, John gave a drink to … GPT2 predicts: Mary.

Three types of attention heads contributed to this prediction:

- Three heads copied information from the Mary vector to the final input vector (for the word to). GPT-2 uses the information in this rightmost vector to predict the next word.
- How did the network decide Mary was the right word to copy? A group of four attention heads marked the second John vector in a way that blocked the first three heads from copying the name John.
- How did the second head group decide John shouldn't be copied? Two attention heads marked the second John vector as a duplicate of the first John vector, which helped the second head group to decide that John shouldn't be copied.

These nine attention heads enabled GPT-2 to figure out that "John gave a drink to John" doesn't make sense and choose "John gave a drink to Mary" instead

First: word vectors aka embeddings

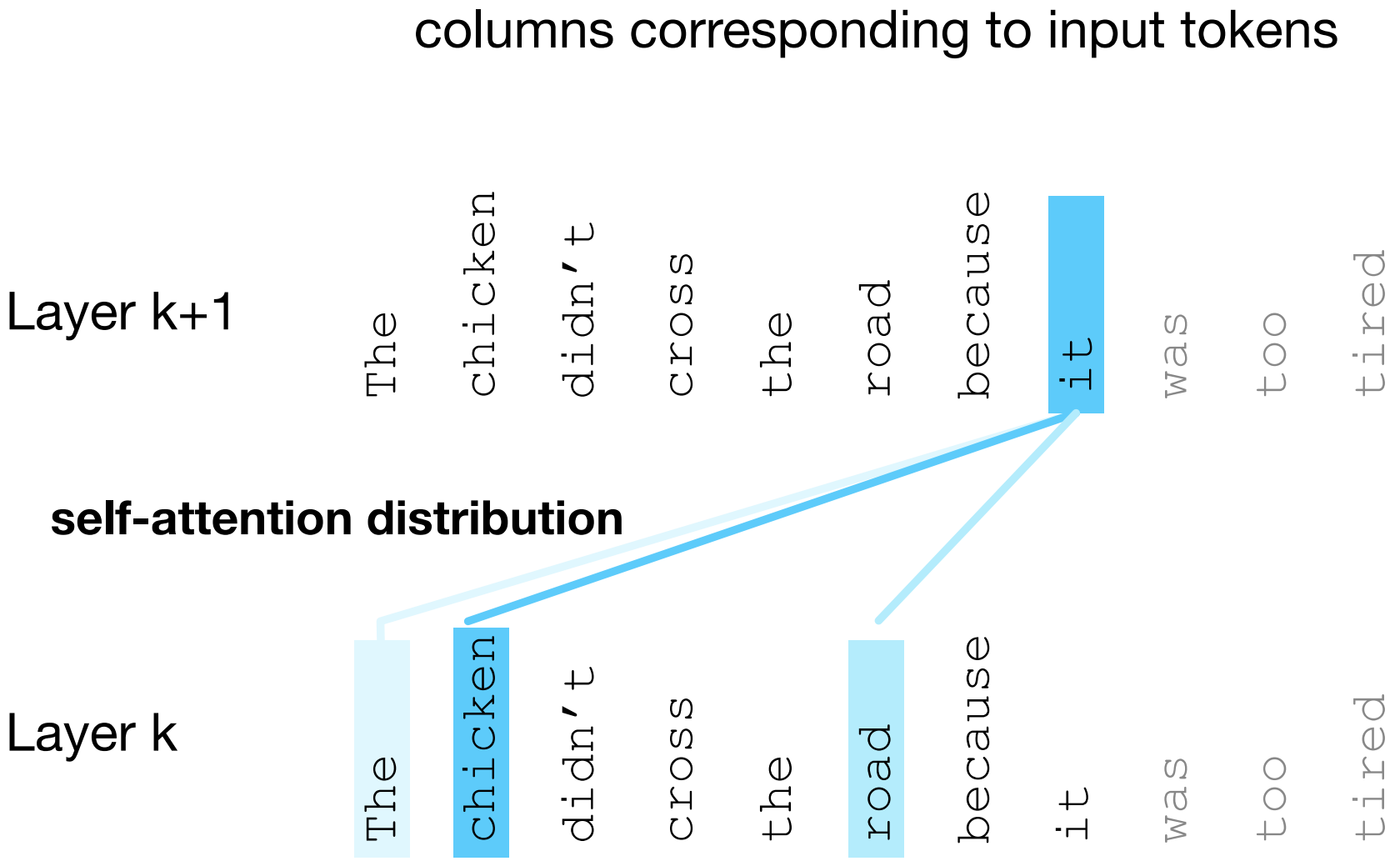
We could use some existing word vectors (embeddings), but LLMs commonly produce their own embeddings that are part of the input layer and are updated during training.

Intuition of attention

Build up the contextual embedding from a word by selectively integrating information from all the neighboring words

We say that a word "attends to" some neighboring words more than others

Intuition of attention:



Next stage: word x word "mattering matrix"

Each word looks at all other words and decides how much each one matters: this is called "attention weights".

Example: "*The animal didn't cross the street because it was too tired*" When the model processes "it": it attends strongly to "animal", not much to "street"

Behind each “mattering” arrow each word creates:

Query (Q) = what I'm looking for

Key (K) = what I offer

Value (V) = information I carry

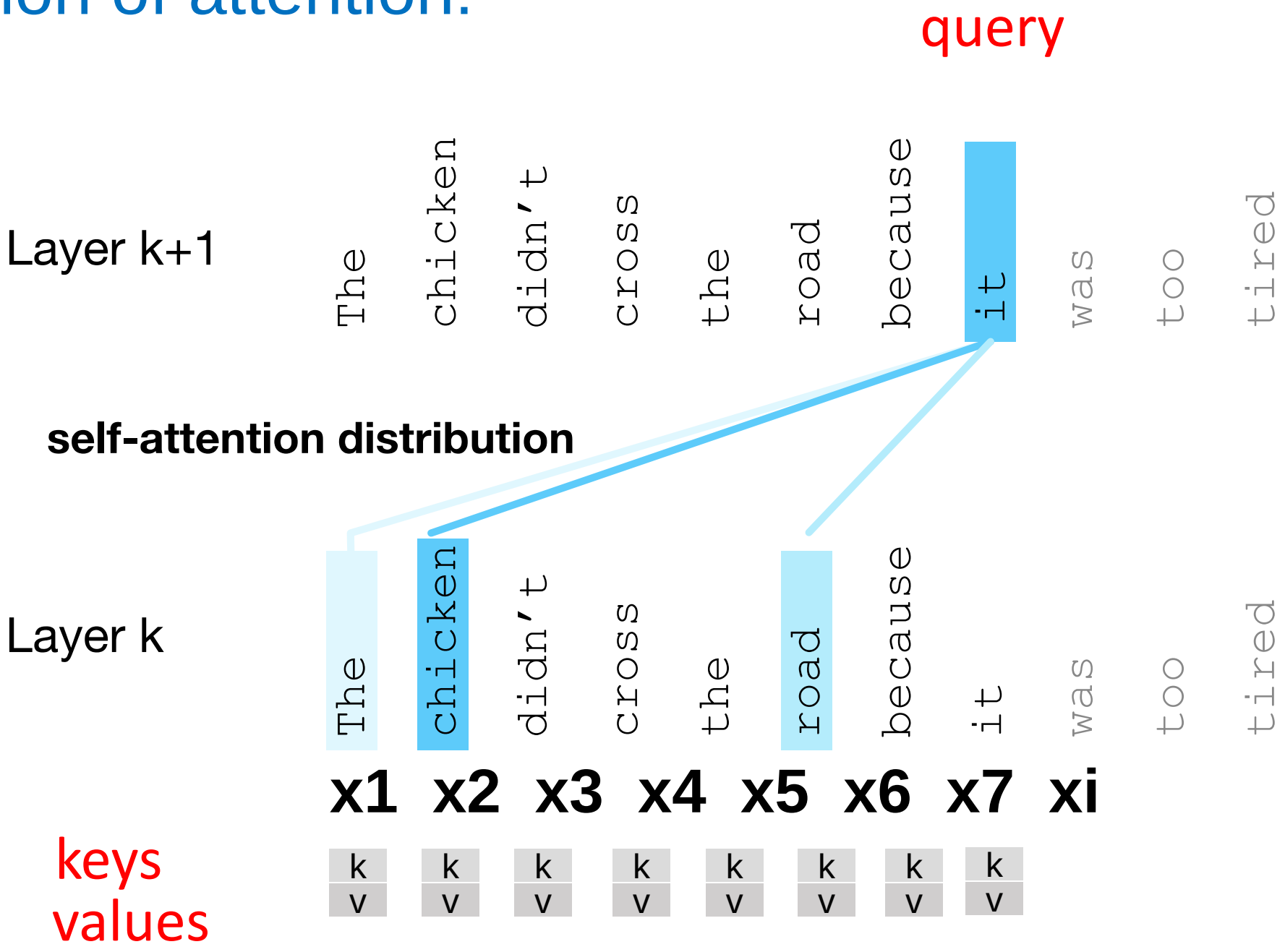
Then:

Compare Q with all K -> scores

Softmax -> attention weights

Weighted sum of V -> new representation

Intuition of attention:



Next stage: we need to turn the "attention weights" into a better representation of each word, and eventually into a prediction.

After attention, you have something like:

For each word -> a list of weights over all words (the attention matrix).

Example (for word “it”):

[it] -> 0.7 animal, 0.1 street, 0.2 others

NB! So far, this is just importance scores.

Now we use the weights to mix information from other words

Each word has a vector (its embedding or hidden state):

V_1, V_2, …., V_n

Now for word *i*, we compute the weighted sum of value vectors

*h_i = sum for all j: matrix_{ij} * V_j*

Where

- *matrix_{ij}* = attention weight (how much word *i* cares about word *j*)
- *V_j* = information (vector representation) from word *j*

Intuition:

Each word becomes *a mixture of other words*, based on how much it cares about them.

For example, *"it"* becomes mostly *"animal",* plus a bit of everything else. This gives a new representation for each word.

After attention, every word gets a new vector: no longer just its own meaning now includes context This is called: **contextual embedding**

Next, more processing: attention is just one sub-step inside a transformer block.

After self-attention, for each token *i*, you have:

- A new vector *h~i* : this is the context-mixed version ("what I learned from others").
- But you also still have the original vector *hi*: this is the original identity of the word.

The model computes: *hi_new = h~i + hi* (called a **residual connection**)

"Take what you learned from others, but don't lose your own identity."

This turns the model into "iterative refinement" instead of "recompute from scratch"

After adding, we apply **layer normalization**: make sure all word vectors stay on a comparable scale.

Feed-forward network stage (FFN)

Now each token is processed independently: same small neural network applied to each token, no interaction between tokens here.

This reshapes and enriches the representation

More concretely:

Step 1 -- Expand vector goes from size d -> larger size (e.g. 4 bigger).

Step 2 -- Nonlinearity / apply activation (ReLU / GELU).

Step 3 -- Compress back: return to original size.

Intuition: "Take the gathered information and think about it."

After FFN: token = interpreted version of the mixture

FFN can do feature detection and learn things like:

- "Is this a subject?"
- "Is this a number?"
- "Is this a verb?"
- "Does this look like code?"

It turns raw vectors into useful internal features.

In real models, FFNs often:

- act like pattern detectors
- specialize neurons (some detect syntax, some semantics)
- create sparse activations (only some neurons fire)

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

Attention: "it" -> looks at "animal"

FFN: transforms that into: "it refers to a tired entity"

Intuition:

- Attention = communication
- Feed-forward = thinking

Each transformer layer does:

For each token:

- gather information (attention)
- keep original + add new info (residual)
- stabilize (normalization)
- process locally (feed-forward)

What happens across many layers? Each layer refines the representation a little more, like:

- Layer 1: basic relationships (syntax)
- Layer 5: phrase structure
- Layer 10+: meaning, reasoning patterns

The full pipeline

First, input embeddings

Then repeat many times (layers):

- self-attention (mix information)
- feed-forward network (process)

Finally:

- Take the last position in the sequence ("The animal didn't cross the street because it was **too** …"
- Map that vector to vocabulary scores (next-token probabilities): P(“tired”), P(“fast”), …