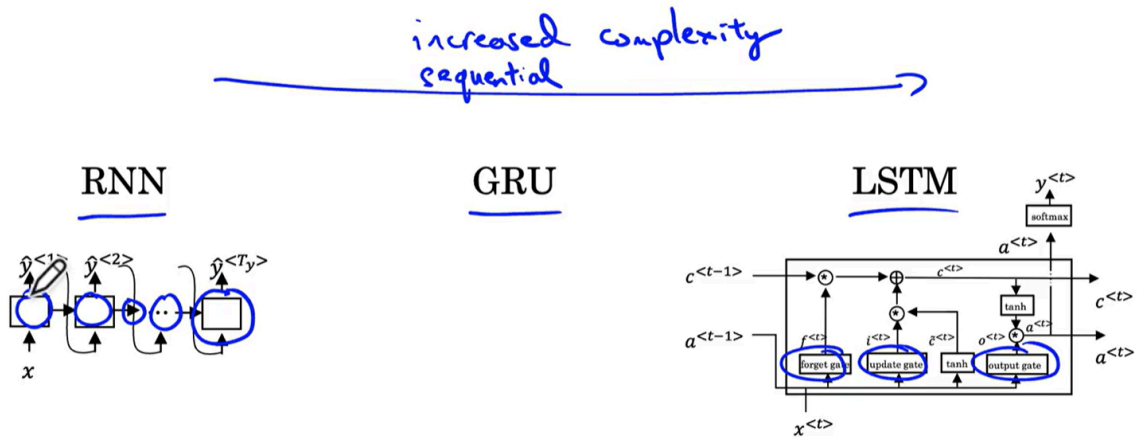
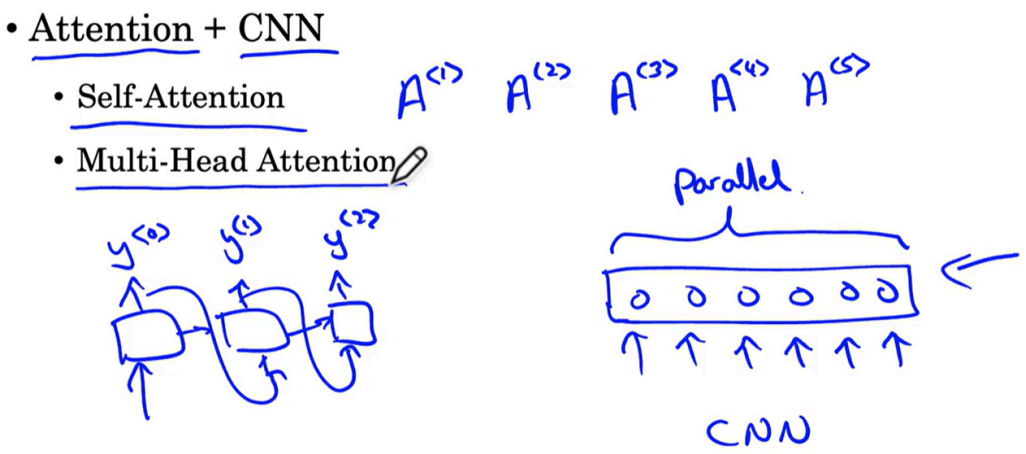
WEEK 4: Transformer Network

Transformer Network Intuition (Transformers)





1. Motivation & Evolution:

- **RNNs/LSTMs/GRUs:** Sequential architectures; struggle with long-term dependencies, vanishing gradients, and slow training due to their step-by-step nature.
- **Transformers:** Address these limitations by enabling *parallel* sequence processing.

2. Core Technical Concepts:

- **Self-Attention:**

    - Computes interactions between every word/token in a sequence.
    - Enables richer context; every output can attend to all inputs.
    - Example: For an input sequence of n words, self-attention generates n output representations simultaneously.
    - Formula (intuition, not explicit):

      Each output O_i= weighted sum of all input representations, where weights are learned via attention scores.

- **Multi-Headed Attention:**
    - Runs multiple self-attention processes in parallel.
    - Each head learns different types of relationships.
    - Final output is a concatenation or combination of these heads.

3. Transformer Architecture Highlights:

- Consists of *encoder* and *decoder* blocks. (Encoder for input understanding; decoder for output generation.)
- Layers rely heavily on the *attention mechanism*; minimizes reliance on recurrence and convolution.

4. Advantages for NLP:

- Faster training (due to parallelism).
- Superior long-range dependency modeling.
- Foundation for state-of-the-art models (BERT, GPT, etc.)

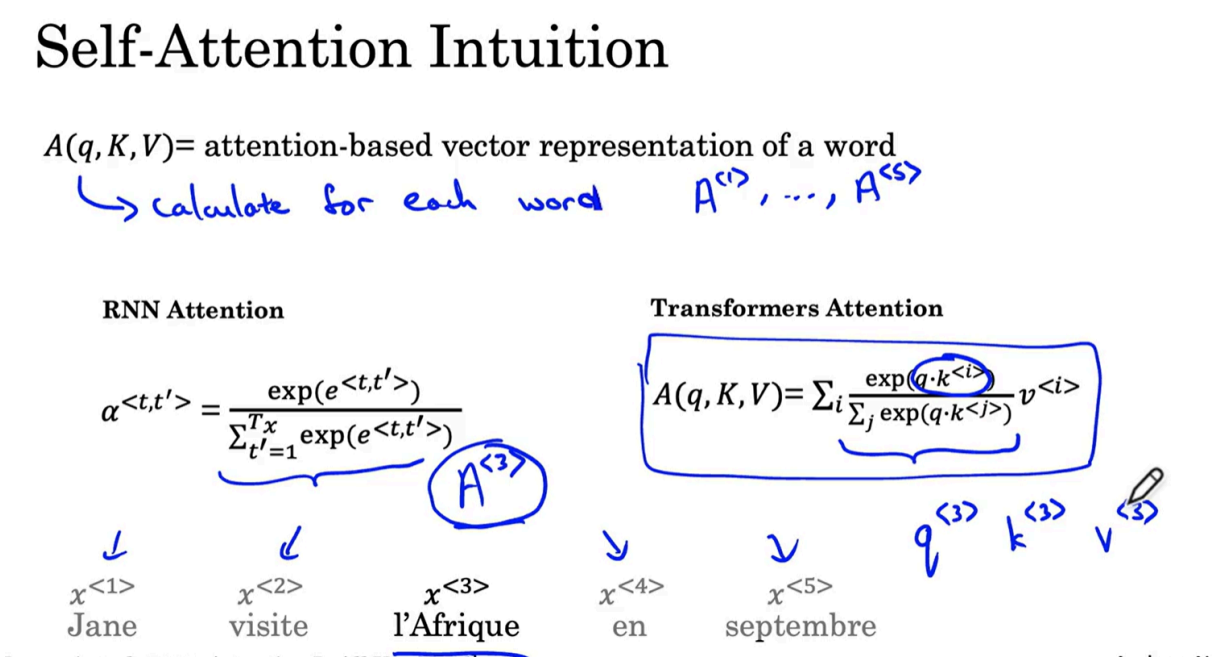
5. Real-World Impact:

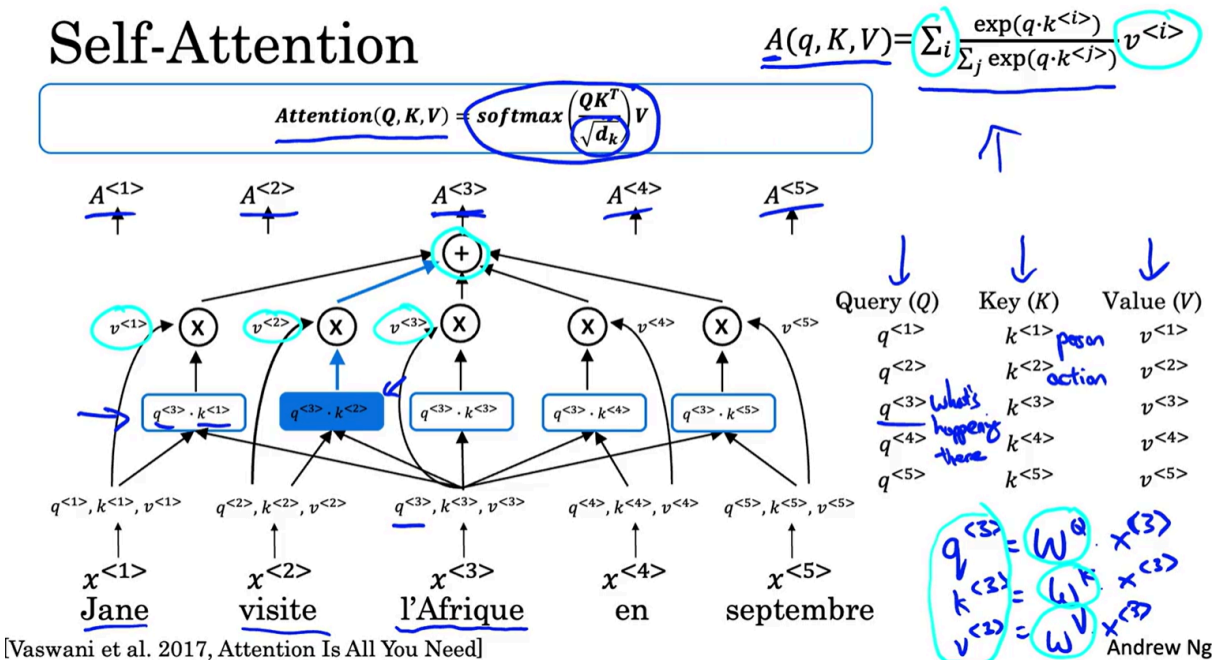
- **Machine Translation:** Dramatic quality improvements.
- **Text Summarization, Question Answering:** Now use transformer-based methods for best results.

6. Key References:

- 2017 Paper “Attention is All You Need” (Ashish Vaswani et al.)
- Lukasz Kaiser: Both a co-author and course instructor.

Self-Attention Mechanism of Transformers





- **Purpose:** Builds context-based word representations by letting every word "look at" every other word in a sentence.

- **Key Operations:**
  - For each word, generate three vectors using trained weights:
    - **Query (Q)**: asks a question about the meaning
    - **Key (K)**: offers information to answer questions
    - **Value (V)**: contains the data to be shared
  - Compute **attention score** for each word pair: dot product Q·K
  - Apply **softmax**: turns scores into weights (probabilities)
  - **Weighted sum**: For each word, new vector is a sum of all Value (V) vectors weighted by attention scores
- **Mathematical formula:**

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

- **Result:** Each word's new vector fully captures keyword relationships and context in the sentence.
- **Why powerful?** Parallel, captures long-range dependencies, foundation for all transformer models.

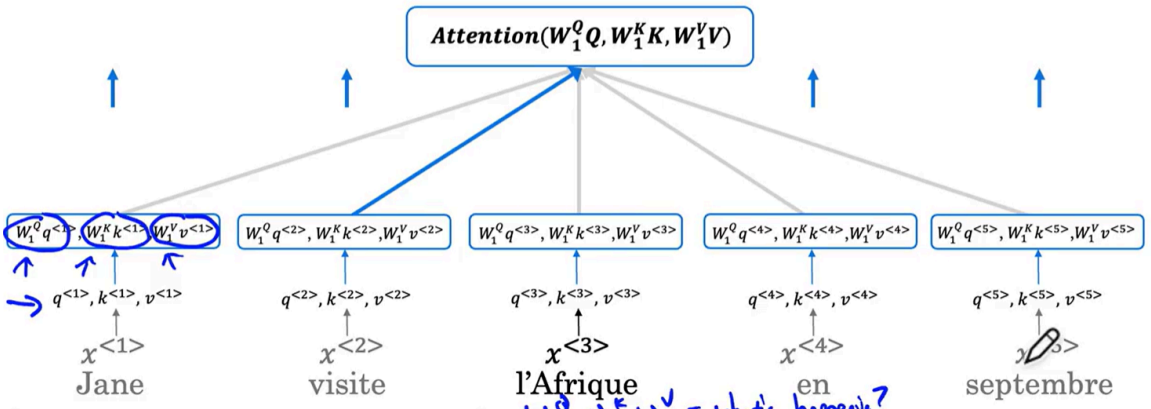
Multi-Head Attention Mechanism

Multi-Head Attention

"head"

$$MultiHead(Q,K,V)$$

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



[Vaswani et al. 2017, Attention Is All You Need]

W_1^Q, W_1^K, W_1^V - what's happening?

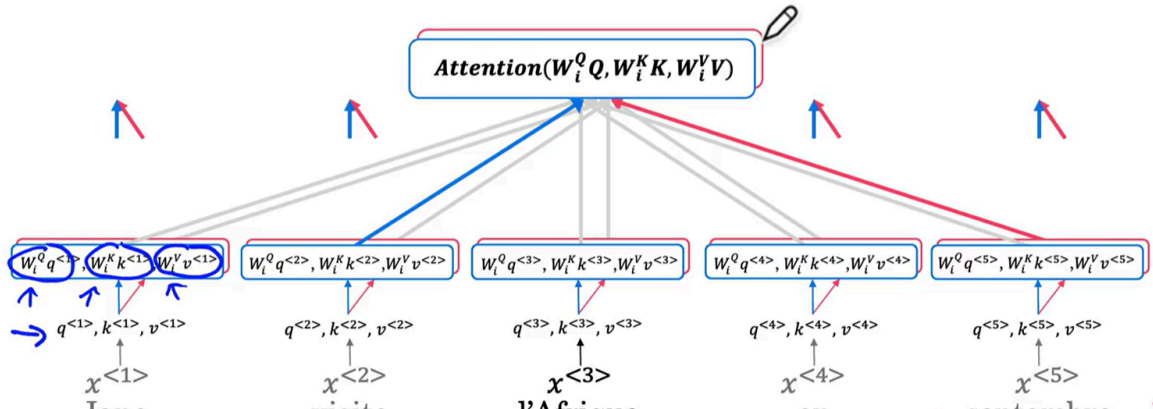
Andrew Ng

Multi-Head Attention

"head"

$$MultiHead(Q,K,V)$$

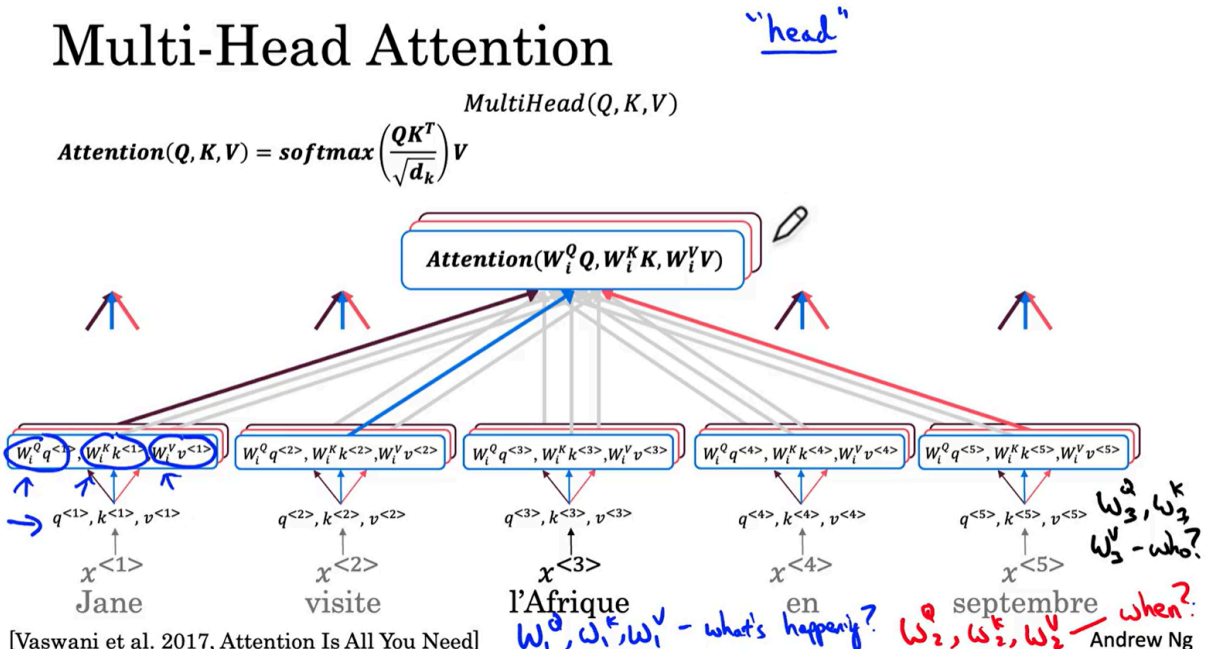
$$\textbf{Attention}(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

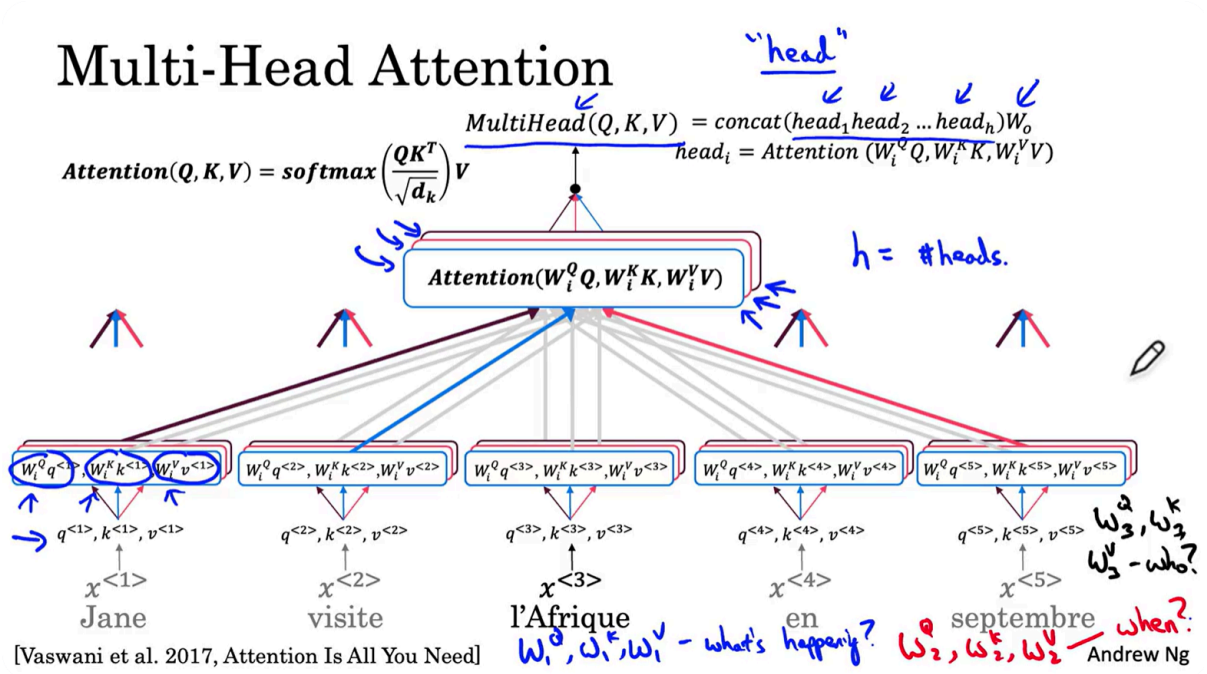


[Vaswani et al. 2017, Attention Is All You Need]

W_1^Q, W_1^K, W_1^V - what's happening? W_2^Q, W_2^K, W_2^V — when?

Andrew Ng





1. Core Idea

- **Multi-head attention** is an extension of self-attention, a key part of the Transformer architecture.
- Instead of running self-attention once, we run it **multiple times in parallel**—each run is called a "head."

- Each head can focus on different relationships or aspects in the input sequence, allowing the model to learn richer representations.

---

2. Step-by-Step Process

a) Input Preparation

- For each word (or token) in the input, we create three vectors:
  - **Query (Q)**
  - **Key (K)**
  - **Value (V)**
- These are computed by multiplying the input by learned weight matrices: W^Q, W^K, W^V

b) Multiple Heads

- In multi-head attention, we use **different sets of weights** for each head:
  - For head 1: W_1^Q, W_1^K, W_1^V
  - For head 2: W_2^Q, W_2^K, W_2^V
  - ... up to head h
- Each head computes its own self-attention using its unique Q, K, V.
- **Intuition:** Each head can "ask" a different question about the input (e.g., "what's happening?", "when?", "who?").

c) Self-Attention Calculation (per head)

- For each head, self-attention is computed as:

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

  where d_k is the dimension of the key vectors.
- This gives a new set of vectors for each word, per head.

d) Concatenation and Final Linear Layer

- The outputs from all heads are **concatenated** (joined together).
- This concatenated vector is then passed through a final linear layer (weight matrix W^O) to produce the final output:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

---

3. Why Use Multiple Heads?

- **Diversity:** Each head can focus on different types of relationships (e.g., syntax, position, meaning).
- **Richer Representations:** Combining multiple heads gives a more comprehensive understanding of the input.
- **Parallelism:** All heads can be computed at the same time (parallel computation), making it efficient.

4. Implementation Details

- **Number of heads (h):** Commonly 8 or 12 in practice.
- **Dimension per head:** The model's total dimension is split among the heads (e.g., 512-dim model, 8 heads → 64-dim per head).
- **Parallel Computation:** Although described as a for-loop, in practice all heads are computed in parallel since they are independent.

5. Summary Table

Step	What Happens?
Linear Projection	Input → multiple Q, K, V sets (one per head)
Attention Heads	Each head computes self-attention independently
Concatenation	Outputs from all heads are concatenated
Final Projection	Concatenated output is projected back to model dimension with W^O

---

6. Intuitive Analogy

- Think of each head as a different "expert" looking at the sentence from a unique perspective.

- By combining their insights, the model gets a much deeper understanding of the input.

7. Key Takeaways

- **Multi-head attention** = multiple self-attentions in parallel, each with its own "view" of the data, combined for a richer output.
- Enables the model to "ask" several questions about each word at once, leading to better representations for tasks like translation, summarization, etc.

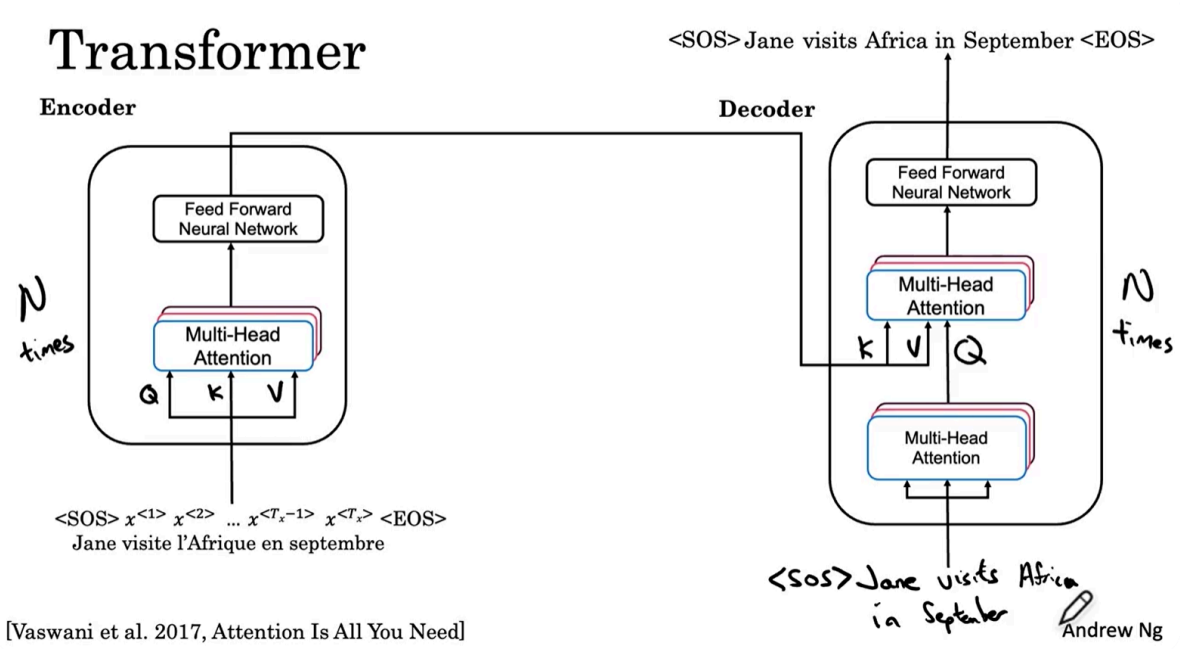
8. Summary

- The multi-head attention block takes in Q, K, V and outputs a combined result.
- In diagrams, it's often shown as a single block with multiple heads inside, representing the parallel attention computations.

9. Formulae Recap

- **Scaled Dot-Product Attention:** $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
- **Multi-Head Attention:**
  $\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$
  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Transformer Network



[Vaswani et al. 2017, Attention Is All You Need]

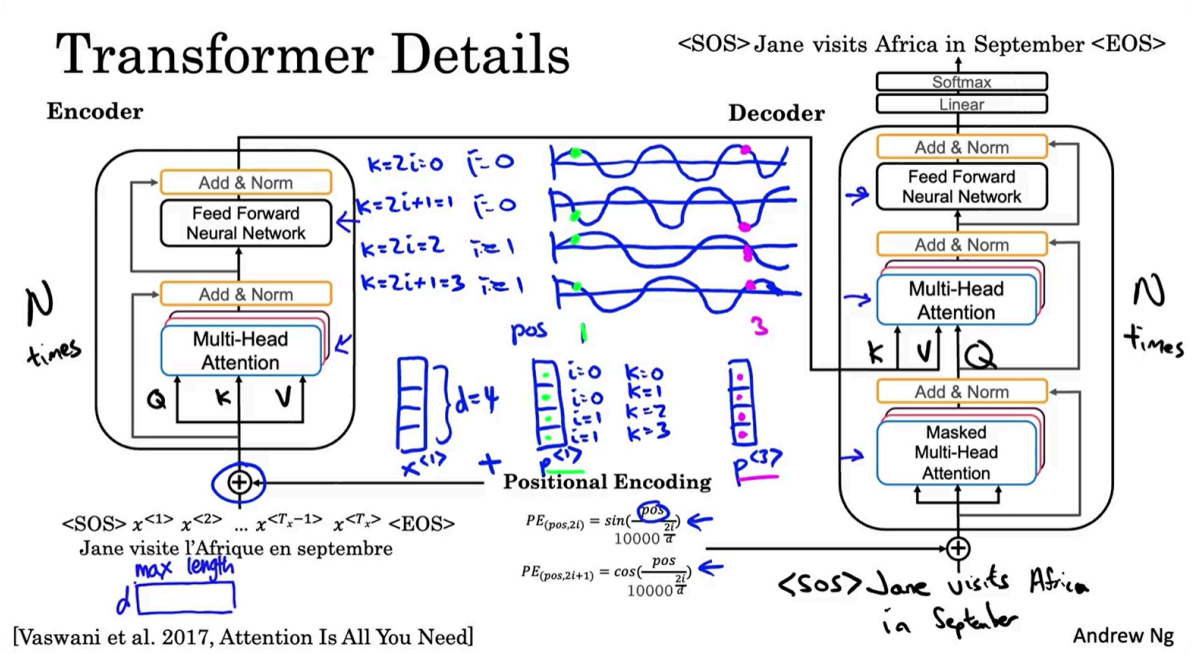
Note that in the video every value of i repeats twice. This is because you use each value of i to encode two dimensions using the sine and cosine functions (same i is used both for sine and cosine). To calculate i, you can use a helper index k, which is simply counting over the dimensions of the word embedding from 0 to $d-1$ and then calculate i as integer division of $k//2$. You will see this in practice during this weeks assignment.

$$k = 2i = 0$$

$$k = 2i + 1 = 1$$

$$k = 2i = 2$$

$$k = 2i + 1 = 3$$



1. Input Embeddings & Special Tokens

- **Input**: Sequence of words (e.g., "Jane visite l'Afrique en septembre").
- **Special tokens**: Add Start-of-Sentence (SOS) and End-of-Sentence (EOS) tokens to mark boundaries.
- **Word Embeddings**: Each word (including SOS/EOS) is mapped to a vector (embedding) of fixed dimension d.

Illustration:

```
text[SOS]  Jane  visite  l'Afrique  en  septembre  [EOS]
    |       |     |        |       |     |        |
 [vec]    [vec] [vec]    [vec]   [vec] [vec]    [vec]
```

Why?

- Embeddings let the model work with numbers instead of words.
- SOS/EOS help the model know where sentences start and end.

---

2. Positional Encoding

- **Problem**: Self-attention doesn't know word order.
- **Solution**: Add a *positional encoding* vector to each word embedding.

- **How**: Use sine and cosine functions to generate unique vectors for each position.

Formula:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

Where:

- pos: position in the sequence
- i: dimension index
- d: embedding dimension

Illustration:

```
textWord Embedding: [0.2, 0.5, -0.1, 0.7]
Positional Encoding: [0.8, -0.3, 0.1, 0.6]
Combined: [1.0, 0.2, 0.0, 1.3]
```

Why?

- This lets the model distinguish between, for example, "Jane visited Africa" and "Africa visited Jane".

---

3. Encoder Block

Each encoder block has:

- **Multi-Head Self-Attention**
- **Add & Norm** (residual connection + layer normalization)
- **Feed-Forward Neural Network (FFN)**
- **Add & Norm** again

3.1. Multi-Head Self-Attention

- **Purpose**: Lets each word "look at" other words in the sentence to gather context.
- **How**: For each word, compute Query (Q), Key (K), and Value (V) vectors using learned weight matrices.
- **Multiple heads**: Run several attention mechanisms in parallel, then concatenate results.

Illustration:

```
Jane ← attends to → visite, l'Afrique, en, septembre
```

3.2. Add & Norm

- **Residual connection**: Add input to output of attention layer.
- **Layer normalization**: Stabilizes and speeds up training.

3.3. Feed-Forward Neural Network (FFN)

- **Purpose**: Applies a small neural network to each position separately.
- **Structure**: Usually two linear layers with a ReLU in between.

3.4. Stacking

- Repeat the encoder block n times (commonly 6).

4. Decoder Block

Each decoder block has:

- **Masked Multi-Head Self-Attention** (prevents looking ahead)
- **Add & Norm**
- **Multi-Head Attention over Encoder Output**
- **Add & Norm**
- **Feed-Forward Neural Network (FFN)**
- **Add & Norm**

4.1. Masked Multi-Head Self-Attention

- **Purpose**: At each step, the decoder can only attend to previous words (not future ones).
- **How**: Mask out future positions in the attention calculation.

4.2. Multi-Head Attention over Encoder Output

- **Purpose**: Lets the decoder attend to the encoder's output (i.e., the input sentence's context).

- **How**: Q comes from the decoder, K and V come from the encoder output.

4.3. Feed-Forward Neural Network (FFN)

- Same as in the encoder.

4.4. Stacking

- Repeat the decoder block n times (commonly 6).

Illustration:

```
Decoder input: [SOS] → predicts "Jane"
Decoder input: [SOS, Jane] → predicts "visits"
```

5. Linear + Softmax Output Layer

- **Purpose**: Converts decoder output to probabilities for each word in the vocabulary.
- **How**: Linear layer projects to vocabulary size, softmax picks the most likely next word.

6. Training with Masked Attention

- **During training**: The model has access to the full correct output sentence.
- **Masking**: Hides future words so the model learns to predict the next word based only on previous ones.

Illustration:

```
Jane visits Africa in September [EOS]
 ^
Predict this word, mask the rest
```

7. Residual Connections & Layer Normalization

- **Residual connections**: Help gradients flow and preserve information.
- **Layer normalization**: Stabilizes and speeds up training.
- **Add & Norm**: Used after each attention and FFN block.

8. Summary Table: Transformer Blocks

Block	Main Components	Purpose
Input Embedding	Word vectors + positional encoding	Represent words & their positions
Encoder Block	Self-attention, FFN, Add & Norm	Encode input context
Decoder Block	Masked self-attention, encoder attention, FFN	Generate output sequence
Output Layer	Linear + Softmax	Predict next word
Residual & Norm	Add & Norm layers throughout	Stabilize and speed up training

9. Key Intuitions

- **Self-attention**: Lets the model focus on relevant words anywhere in the sentence.
- **Multi-head**: Each head can focus on different types of relationships.
- **Positional encoding**: Adds order information.
- **Masking**: Ensures the model doesn't "cheat" during training.
- **Residuals & Norms**: Make deep networks trainable.

10. Quick Recap: Data Flow

1. Input sentence → Embedding + Positional Encoding
2. Pass through stacked encoder blocks
3. Decoder takes previous outputs + encoder output
4. Pass through stacked decoder blocks
5. Linear + softmax layer predicts next word

11. Practice Questions

- Can you explain why positional encoding is needed?
- What is the difference between encoder and decoder attention?
- Why do we use masking in the decoder?

References

Week 1:

- Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy (GitHub: karpathy)
- The Unreasonable Effectiveness of Recurrent Neural Networks (Andrej Karpathy blog, 2015)
- deepjazz (GitHub: jisungk)
- Learning Jazz Grammars (Gillick, Tang & Keller, 2010)
- A Grammatical Approach to Automatic Improvisation (Keller & Morrison, 2007)
- Surprising Harmonies (Pachet, 1999)

Week 2:

- Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings (Bolukbasi, Chang, Zou, Saligrama & Kalai, 2016)
- GloVe: Global Vectors for Word Representation (Pennington, Socher & Manning, 2014)
- Woebot.

Week 4:

- Natural Language Processing Specialization (by DeepLearning.AI)
- Attention Is All You Need (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser & Polosukhin, 2017)