# Welcome to SEAS Online at George Washington University

## Class will begin shortly

**Audio:** To eliminate background noise, please be sure your audio is muted. To speak, please click the hand icon at the bottom of your screen (**Raise Hand**). When instructor calls on you, click microphone icon to unmute. When you've finished speaking, ***be sure to mute yourself again***.

**Chat:** Please type your questions in Chat.

**Recordings:** As part of the educational support for students, we provide downloadable recordings of each class session to be used exclusively by registered students in that particular class. Releasing these recordings is strictly prohibited.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

# SEAS 8520 – Lecture 8: NLP Transformers

Walid Hassan, M.B.A, D.Eng.

# BERT Embedding…

Using BERT , the embedding for Bank differs between the two sentences…

sentence1 = "I sat by the river bank."

sentence2 = "I deposited money in the bank."

embedding1 = get_word_embedding(sentence1, "bank")

embedding2 = get_word_embedding(sentence2, "bank")

BertRiverBankEmbedding.txt

BertCashBank.txt

```
similarity = torch.nn.functional.cosine_similarity(
    torch.tensor(embedding1).unsqueeze(0), torch.tensor(embedding2).unsqueeze(0)
)
```

**Cosine similarity between the embeddings: 0.525728702545166**

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: https://statquest.org/transformer-neural-networks-chatgpts-foundation-clearly-explained/

Hands On…..

# BERT

- BERT: Bidirectional Encoder Representations from Transformers
- Training Data: BooksCorpus (800 million words) and English Wikipedia (2,500 million words)
- Model Sizes: BERT-Base: 12 layers (transformer blocks) 110 million parameters

  BERT-Large: 24 layers (transformer blocks) 340 million parameters
- **Key Features**:
  - Bidirectional context understanding
  - Pre-trained on masked language modeling
  - Fine-tuned for various tasks (classification, QA, etc.)
- **Innovations**: Introduced "Attention Masking" for dynamic input lengths
- Captures deep contextual relations between words

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: https://statquest.org/transformer-neural-networks-chatgpts-foundation-clearly-explained/

# BERT

- BERT utilizes the Transformer architecture and was originally trained on vast amounts of text, including the entire English Wikipedia and BooksCorpus.
- While pre-trained on general data, BERT can be fine-tuned on a specific dataset to capture domain-specific nuances and context. This adaptability has made it a powerful tool for various NLP tasks.
- The Hugging Face Model Hub hosts thousands of BERT variants and models fine-tuned for diverse tasks and languages.
- https://huggingface.co/models?sort=trending

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: https://statquest.org/transformer-neural-networks-chatgpts-foundation-clearly-explained/

# BERT - Masked Language Model

- MLM enables/enforces bidirectional learning from text by masking (hiding) a word in a sentence and forcing BERT to bidirectionally use the words on either side of the covered word to predict the masked word.
- **Examples:**
- Dang! I'm out fishing and a huge trout just [blank] my line!
- What could a "good" prediction for that interruption … Broke.
- A random 15% of tokenized words are hidden during training and BERT's job is to correctly predict the hidden words. Thus, directly teaching the model about the English language.
- 3.3 Billion words has contributed to BERT's continued success

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

https://towardsdatascience.com/a-gentle-introduction-to-natural-language-processing-e716ed3c0863

# BERT – Next Sentence Validity/Prediction

- NSP/ NSV(Next Sentence Prediction/ Validity ) is used to help BERT learn about relationships between sentences by predicting if a given sentence follows the previous sentence or not.
- **Examples:**
- Paul went shopping. He bought a new shirt. (correct sentence pair)
- Ramona made coffee. Vanilla ice cream cones for sale. (incorrect sentence pair).
- In training, 50% correct sentence pairs are mixed in with 50% random sentence pairs to help BERT increase next sentence prediction accuracy.
- BERT is trained on both MLM (50%) and NSP (50%) at the same time

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

https://huggingface.co/blog/bert-101#2-how-does-bert-work

# Transformers and Attention Mechanism

Era: Late 2010s onwards

Description: Transformers use self-attention mechanisms for parallel processing of sequences.

Impact: Led to models like BERT, GPT, revolutionizing NLP across numerous benchmarks.

.

Attention is all you need….

# Transformers

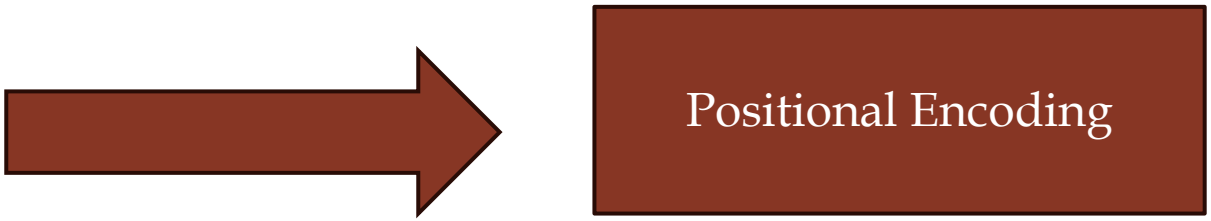
What is the first thing to do?

Digitize our sentences i.e. Tokenization + Embedding…

But that only gets us a very rich meaning of each word. That is as far as it gets us (It is great, but not a fully “human like” model)

# Transformers

First, It is important to know where the word or token is in the sequence.
For example.
We need to know the difference between, I ate the apple, apple ate I the, ate apple the I



# Positional Encoding

1. **Purpose of Positional Encoding**:
    1. Introduces sequence information to the model, enabling the differentiation of word positions within the input sentence.

2. **Mathematical Formulas**:
    1. For each position(pos) and dimension(i), the positional encoding is calculated as:
        1. $PE(pos,2i)=\sin(pos/10000^{2i/d\text{model}})$
        2. $PE(pos,2i+1)=\cos(pos/10000^{2i/d\text{model}})$

    Here, $d$model represents the dimensionality of the embeddings.

3. **Incorporation into Embeddings**:
    1. The resulting positional encoding is added element-wise to the word embedding vector.
    2. This combination ensures that the model not only captures the semantic meaning of each word but also understands the syntax and structure of the sentence.

# Positional Encoding -- Example

- **Objective**: Demonstrating the calculation of positional encodings for the example sentences.

- **Sentences for Illustration**: "I love to play soccer“.

- **Understanding Positional Encoding**: Explaining the need for positional encoding in transformers.
- **Tokenizing the Sentence**: Breaking the sentence into tokens: ["I", "love", "to", "play", "soccer"].

- **Assigning Positions**: Assigning positional values starting from 0: I(0), love(1), to(2), play(3), soccer(4).

# Positional Encoding -- Example

- **Sentence**: "I love to play soccer" - focusing on the word "I" (position 0).
- **d_model = 4**: Four dimensions for illustrative purposes. GPT 4 has d_model of 2048.

- **Calculation for pos=0**:
  - Dimension 0: $\sin(0/10000^{0/4}) = \sin(0) = 0$
  - Dimension 1: $\cos(0/10000^{0/4}) = \cos(0) = 1$
  - Dimension 2: $\sin(0/10000^{2/4}) = \sin(0) = 0$
  - Dimension 3: $\cos(0/10000^{2/4}) = \cos(0) = 1$

- **Result**: Positional encoding for "I" is [0, 1, 0, 1].

# Positional Encoding -- Example

- **Word**: "love" (position 1).

- **Calculation for pos=1**:
  - Dimension 0: $\sin(1/10000^{0/4}) = \sin(1)$
  - Dimension 1: $\cos(1/10000^{0/4}) = \cos(1)$
  - Dimension 2: $\sin(1/10000^{2/4}) = \sin(1/10000^{.5})$
  - Dimension 3: $\cos(1/10000^{2/4}) = \cos(1/10000^{.5})$

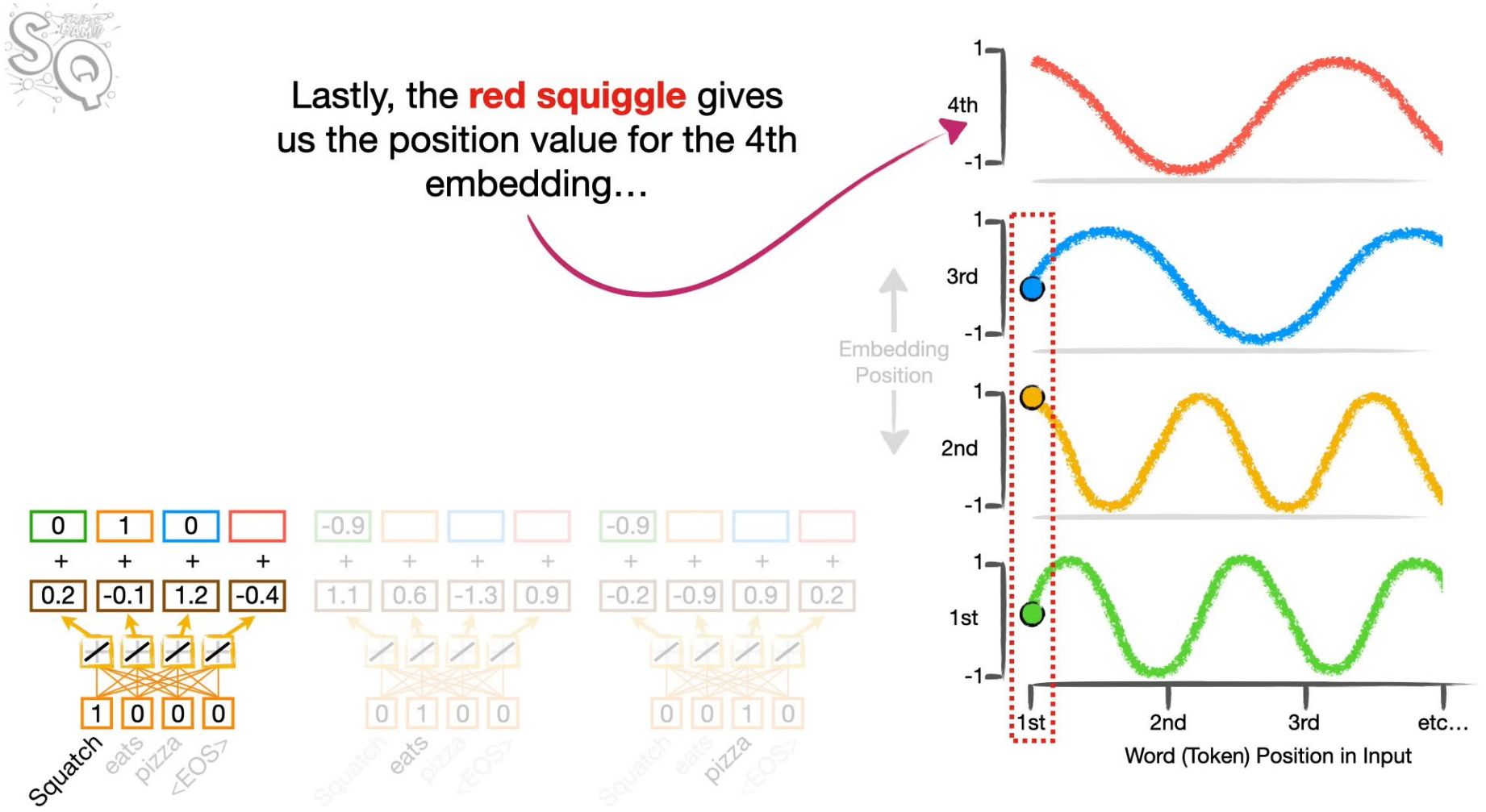
- **Result**: Positional encoding for "love" shows variation across dimensions.

# Positional Encoding -- Example

- Note that each word leverages the position "tensor" has the full encoding of where it is at in the sentence.

- So the position encoding is not one single value for each word.

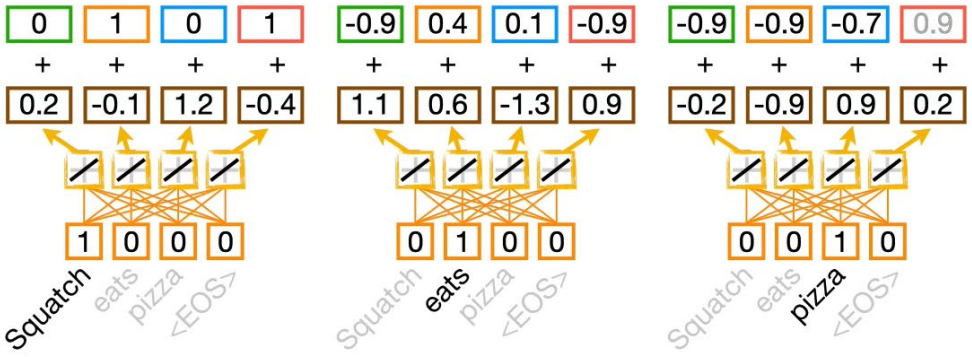
# Positional Encoding -- Visually

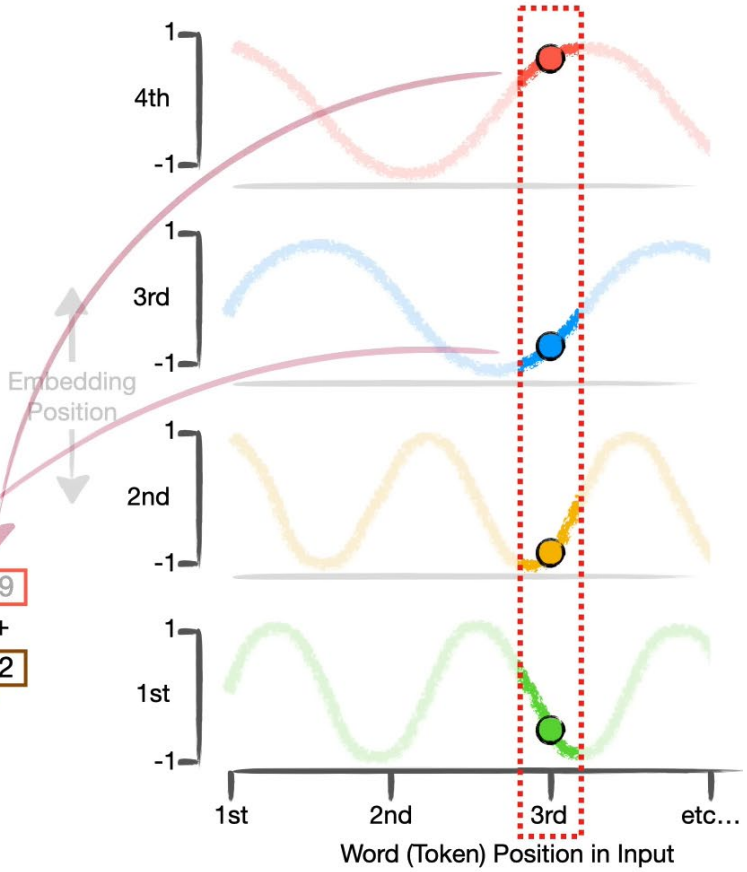



Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Positional Encoding -- Visually

…we use the y-axis coordinates on the squiggles that correspond to the x-axis coordinate for the third word.

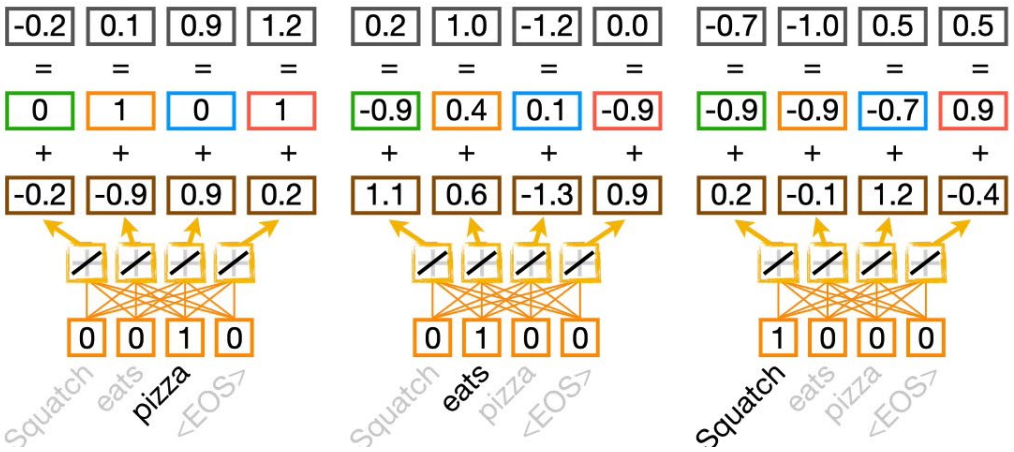


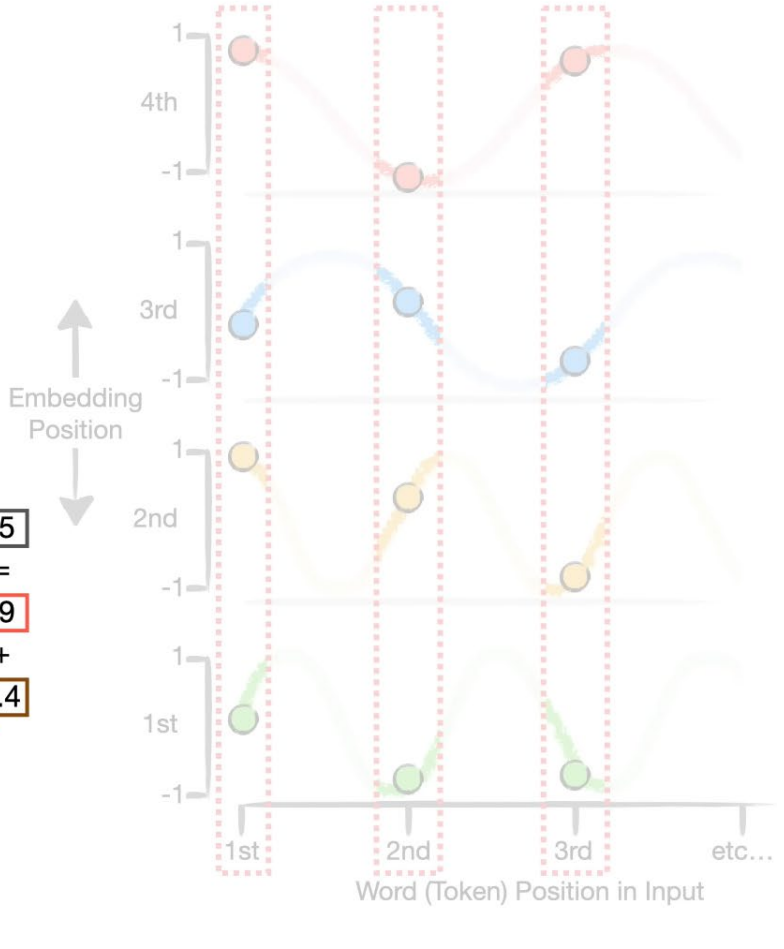




Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Positional Encoding -- Visually

Thus, **Position Encoding** allows a **Transformer** to keep track of word order.

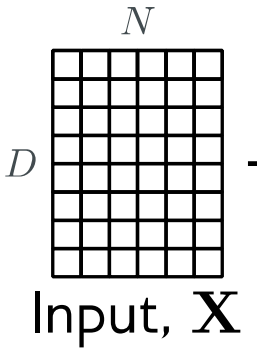


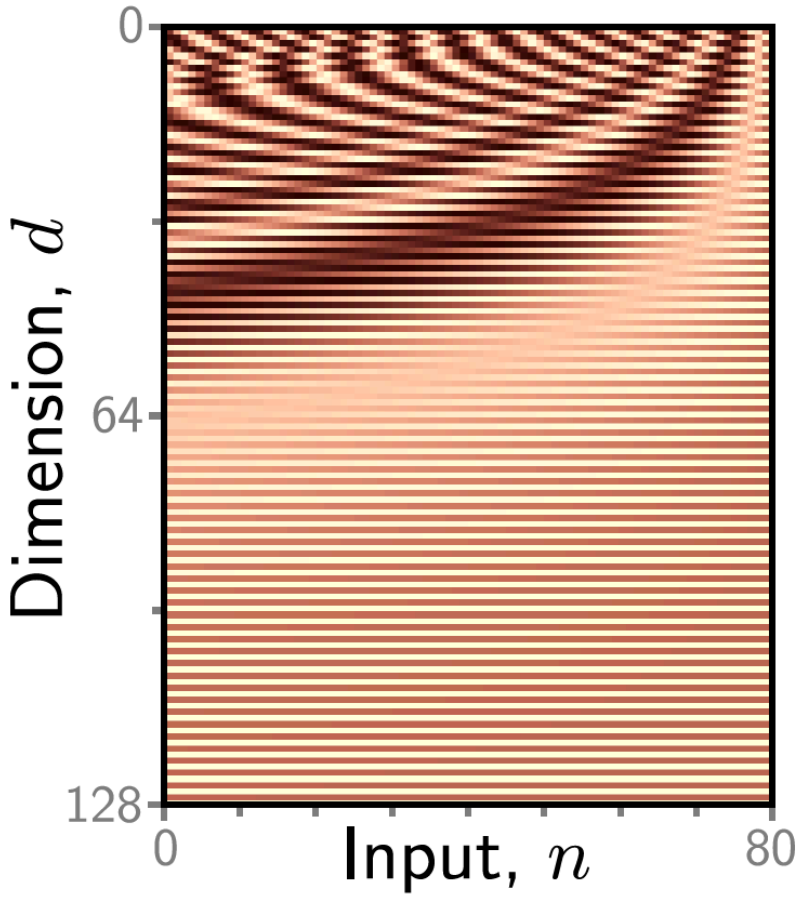

Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

Hands On….. Positional encoding

# Positional Encoding




This "Matrix" is pre-generated.




Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Is positional Encoding Enough?

sentence1 = "I sat by the river bank."

sentence2 = "I deposited money in the bank."

The fact that position alone, does not give you what the word “Bank” in a sentence means. So, we need something else??? →

Its relationship to the different words. Hence the attention mechanisms. But How?

# Attention Mechanism in Transformers

•An attention mechanism allows the model to focus on different parts of the input sequence when making predictions.

•Self-attention enables each token to attend to all tokens in the sequence.

•The model computes a weighted sum of all input representations based on their relevance.

•Attention helps in understanding context and relationships between words in a sentence.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

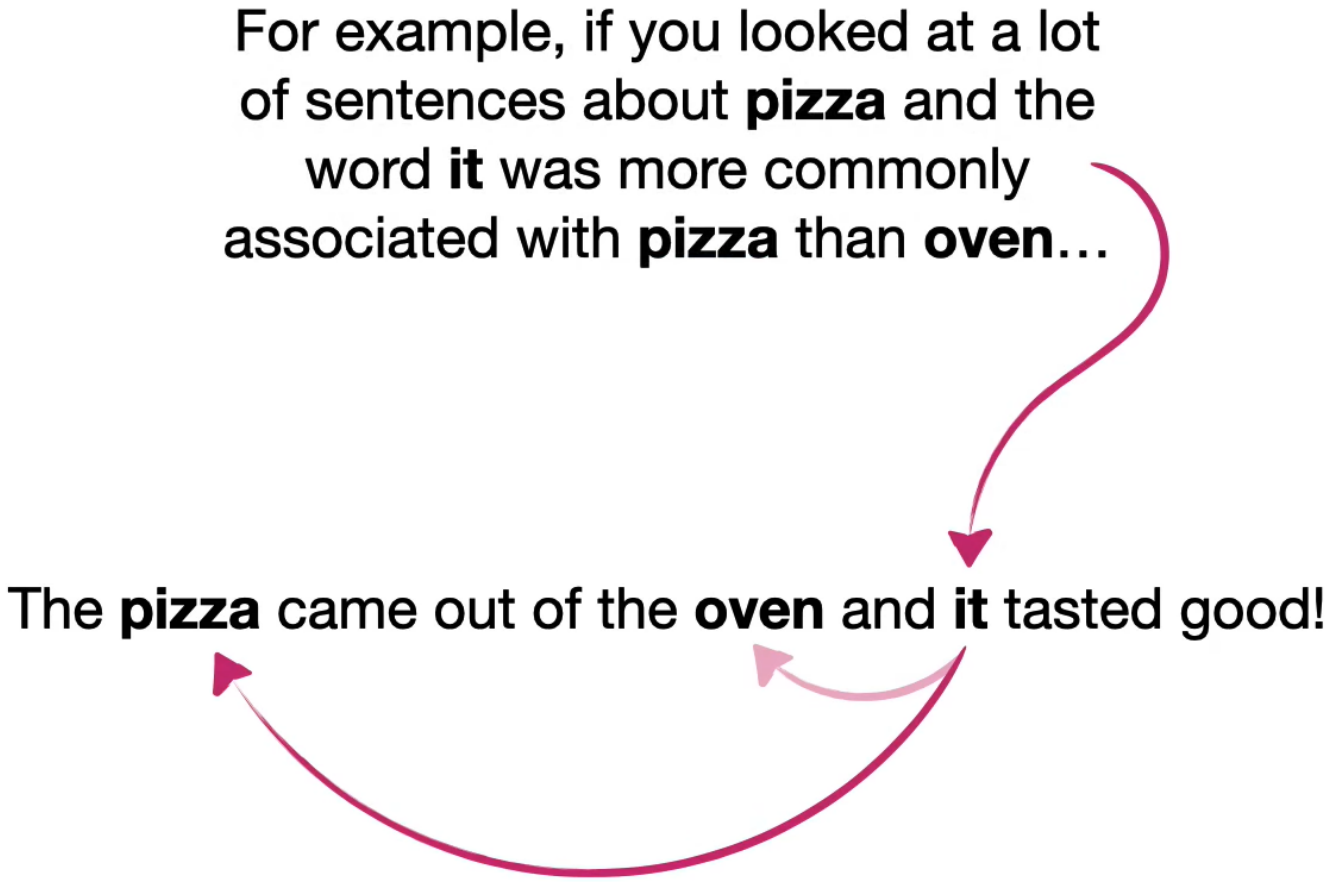
# Attention Mechanism -- Visually

In general terms, **Self-Attention** works by seeing how similar each word is to all of the words in the sentence, including itself.

The **pizza** came out of the **oven** and **it** tasted good!

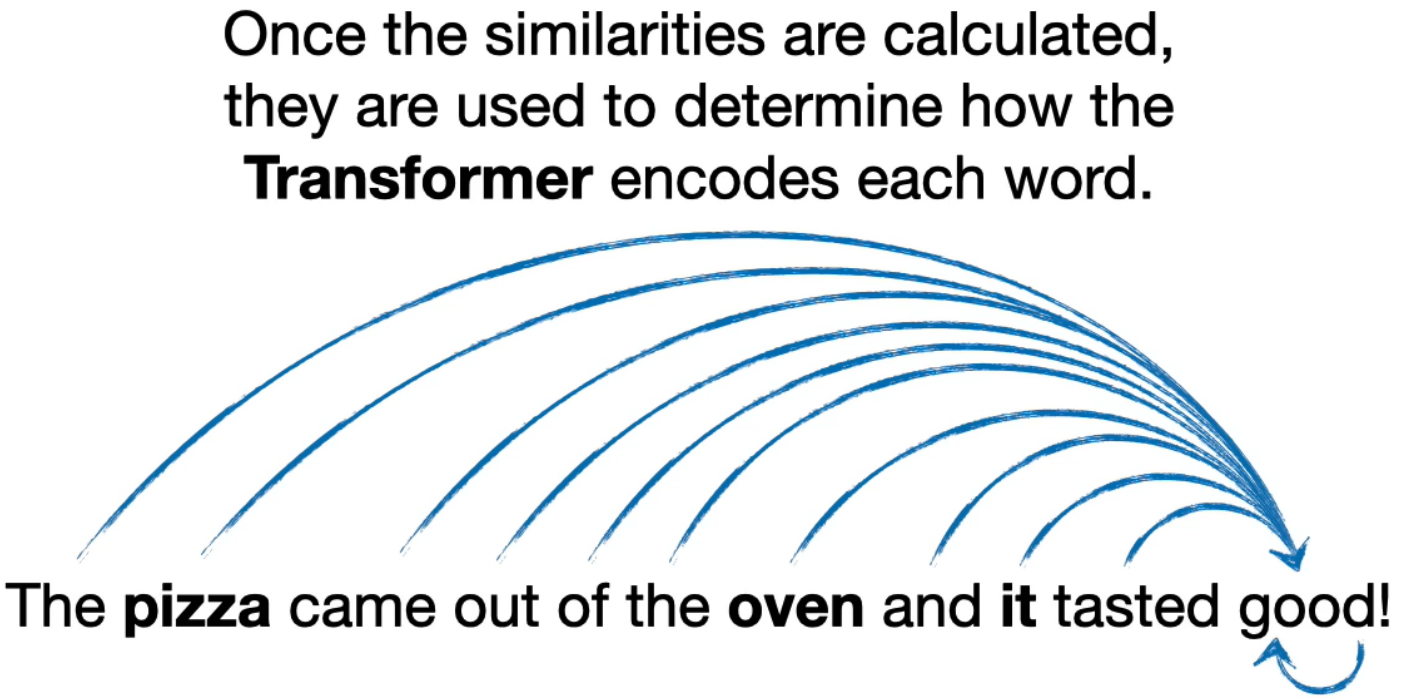
Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Attention Mechanism – Visual Example



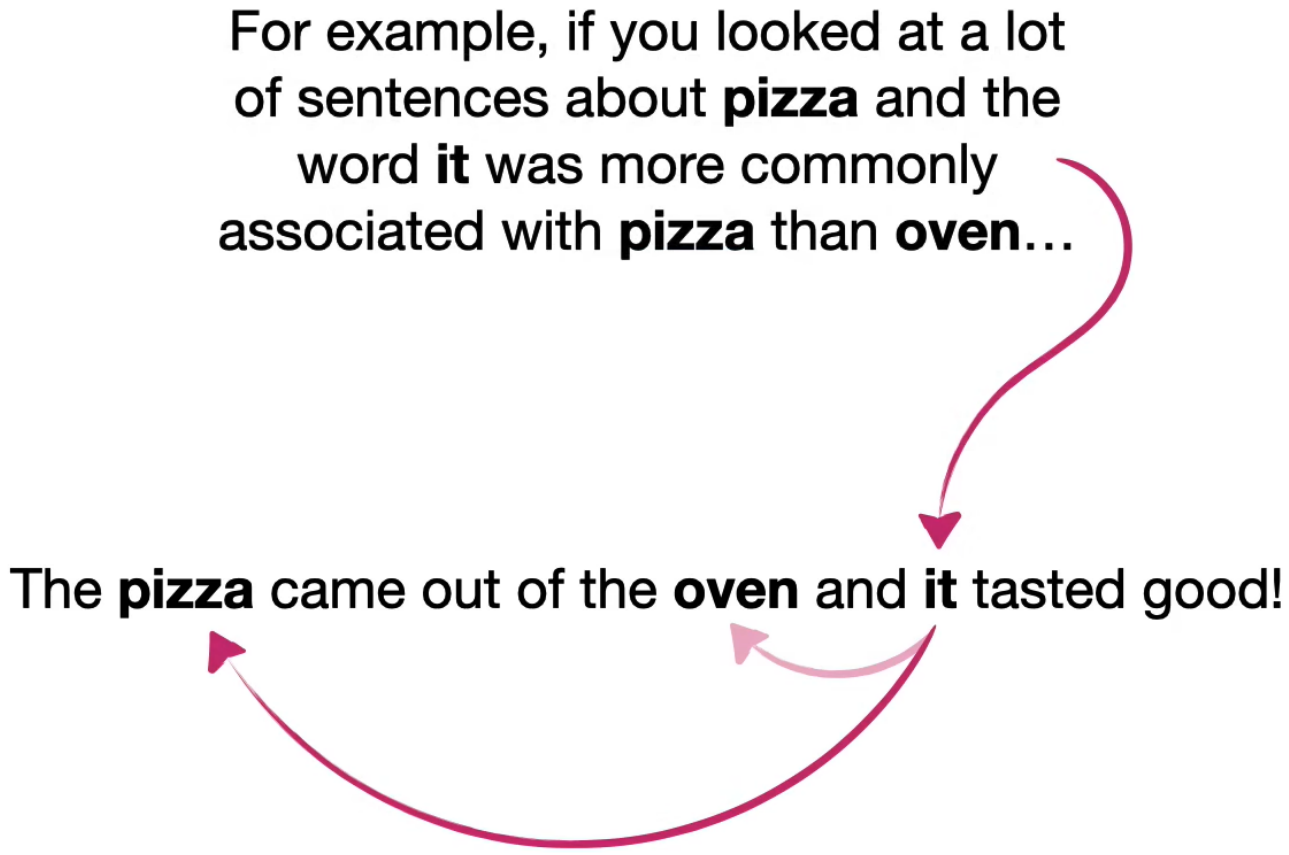

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Attention Mechanism -- Example




Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

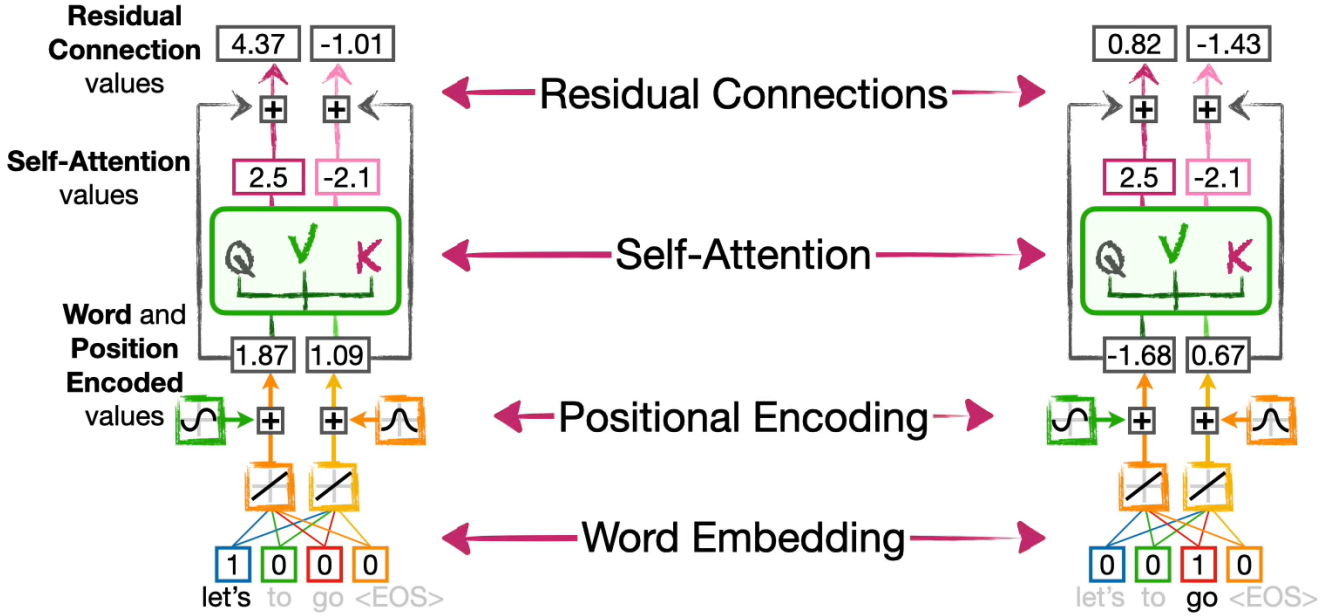
# Attention Mechanism -- Example




Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

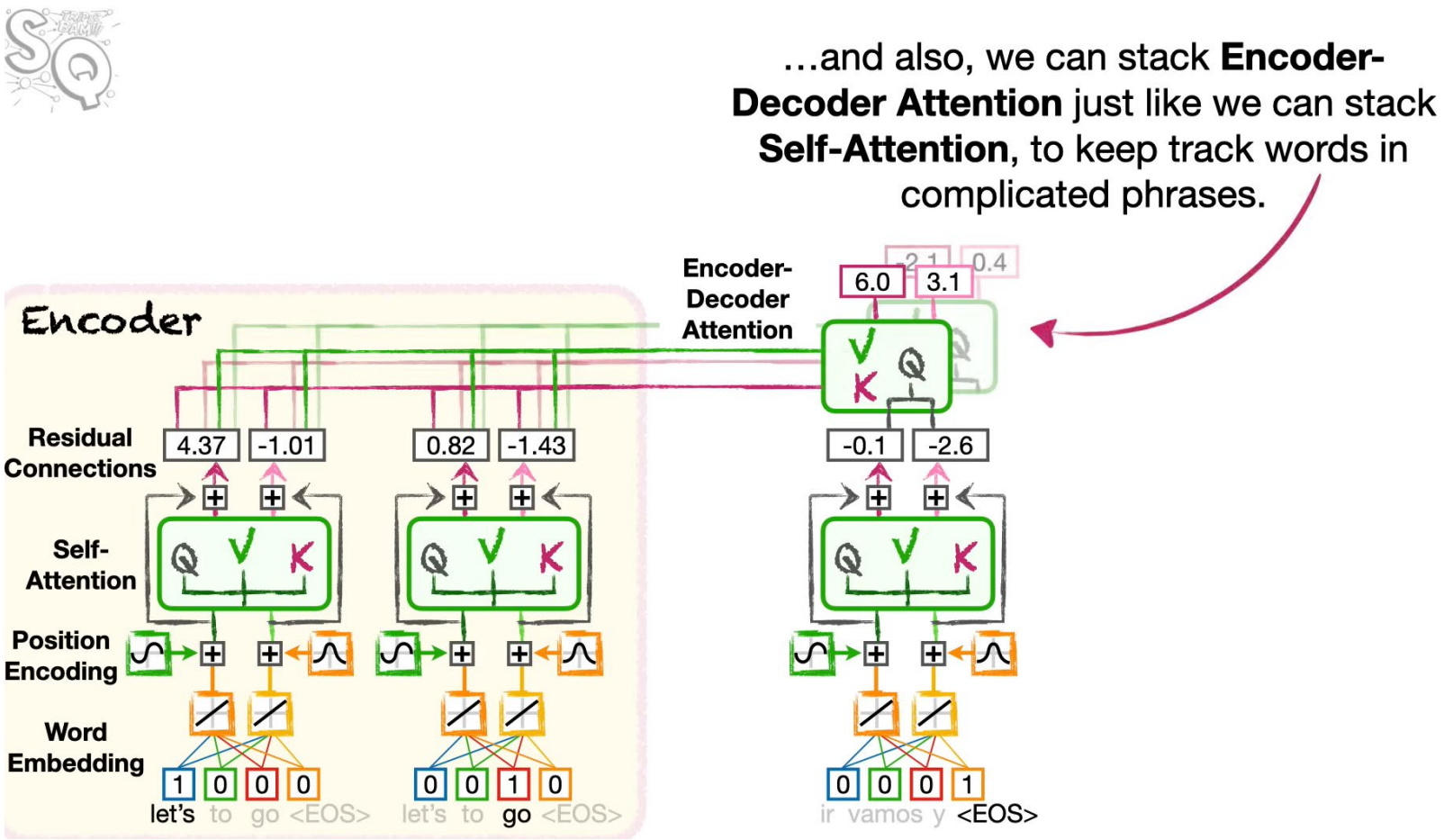
# Attention Mechanism -- Example

These **4** features allow the **Transformer** to: encode words into numbers, encode the positions of the words, encode the relationships among the words, and relatively easily and quickly train in parallel.




Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Attention Mechanism -- Example

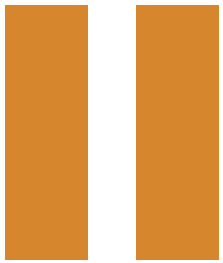



Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Demo of Attention

https://github.com/jessevig/bertviz?tab=readme-ov-file

Src: Jessy Vig (GitHub: https://github.com/jessevig/bertviz?tab=readme-ov-file)



# BREAK

**Please come back @**

**2:20 PM EST**

**1:20 PM CST**

# Motivation

Design neural network to encode and process text:

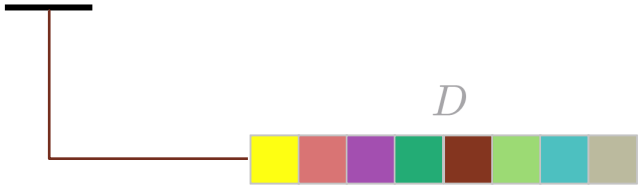
The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

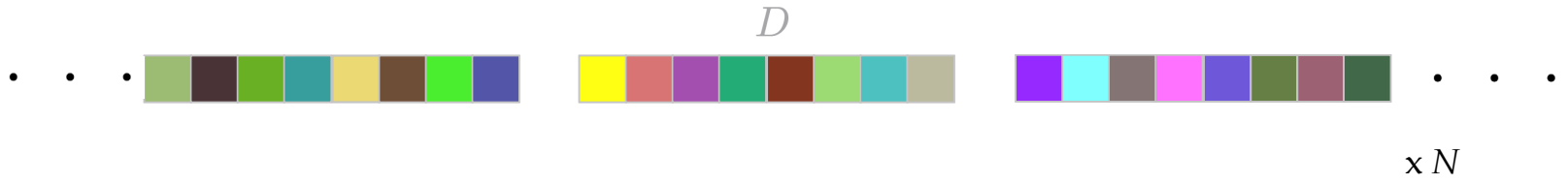



Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Motivation

Design neural network to encode and process text:

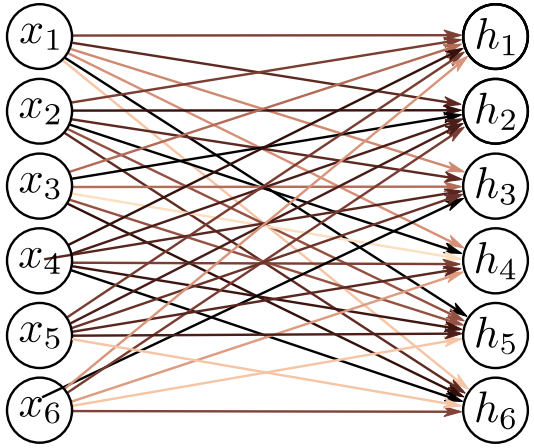
The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.




Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Standard fully-connected layer

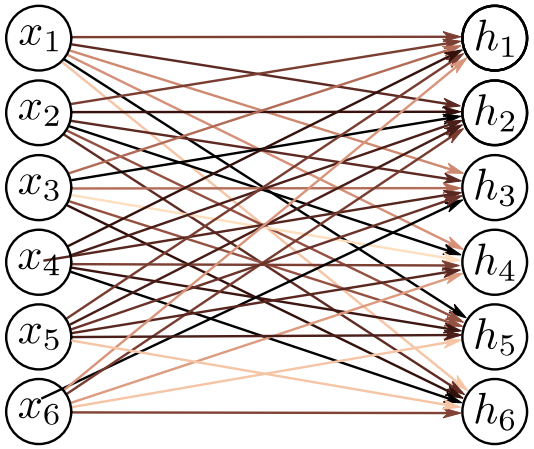
$$\mathbf{h} = \mathbf{a}[\boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{x}]$$




Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Standard fully-connected layer

$$\mathbf{h} = \mathbf{a}[\boldsymbol{\beta} + \boldsymbol{\Omega}\mathbf{x}]$$




$\boldsymbol{\Phi}$ contains

$D^2$ connections

$D^2$ connections

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Standard fully-connected layer

Problem:

- A very large number of parameters
- Can't cope with text of different lengths

Conclusion:

- We need a model where parameters don't increase with input length

Similar to CNNs & Filters

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

The word their must "attend to" the word restaurant.

In this case, each of the 37 words might be represented by an embedding vector of length 1024, so the encoded input would be of length 37 × 1024 = 37888 even for this small passage

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Motivation

Design neural network to encode and process text:

The restaurant refused to serve me a ham sandwich, because it only cooks vegetarian food. In the end, they just gave me two slices of bread. Their ambience was just as good as the food and service.

The word their must "attend to" the word restaurant.

Conclusions:

- There must be connections between the words.
- The strength of these connections will depend on the words themselves.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention Enablement

How do we enable such a functionality?

1. We need to do a dot product between the words to measure the “relationship” between the words.
2. Create a “learnable” Copy of the word… to add flexibility to the network.
3. Use softmax to create percentage attention (so they all sum up to 1).

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Attention Enablement

To enable that, from the word+position embedding, you create "copies" of the input by multiplying them by a trainable parameters (but are the same all across the attention head) so that the model can learn different relationships from the positional and word embedding… This allows the model to learn the relationship between these words without actually impacting the original values.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Dot-product self attention

Takes N inputs of size Dx1 and returns N inputs of size Dx1
Computes N values (no ReLU)

$$\mathbf{v}_n = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_n$$

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Dot-product self attention

**The scalar weight *a*[x*m*, x*n*] is the *attention* that the *nth* output pays to input x*m*. (will get to how *a*[x*m*, x*n*] is obtained in a bit)**

The *N* weights *a*[•, **x***n*] are non-negative and sum to one.

Then the $n^{th}$ output $\mathbf{sa}_n[\mathbf{x}_1, \ldots, \mathbf{x}_N]$ is a weighted sum of all the values $\mathbf{v}_1, \ldots, \mathbf{v}_N$:

$$\mathbf{sa}_n[\mathbf{x}_1, \ldots, \mathbf{x}_N] = \sum_{m=1}^{N} a[\mathbf{x}_m, \mathbf{x}_n]\mathbf{v}_m. \quad (12.3)$$

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Dot-product self attention

Takes N inputs of size Dx1 and returns N inputs of size Dx1
Computes N values (no ReLU)

$$\mathbf{v}_n = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_n$$

N outputs are weighted sums of these values

$$\mathbf{sa}[\mathbf{x}_n] = \sum_{m=1}^{N} a[\mathbf{x}_n, \mathbf{x}_m]\mathbf{v}_m$$

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Dot-product self attention

Takes N inputs of size Dx1 and returns N inputs of size Dx1
Computes N values (no ReLU)

$$\mathbf{v}_n = \boldsymbol{\beta}_v + \boldsymbol{\Omega}_v \mathbf{x}_n$$

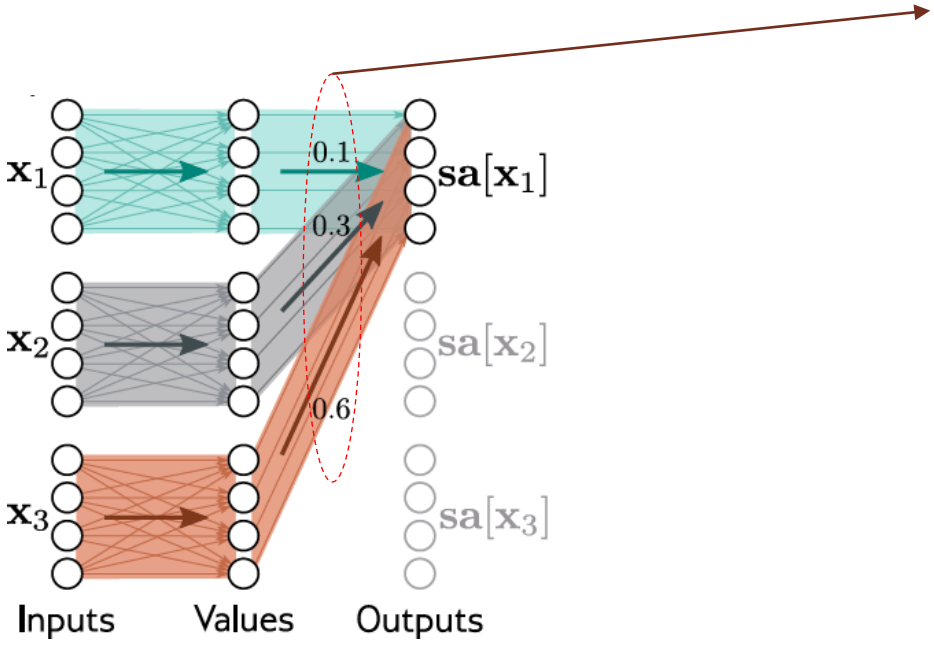
N outputs are weighted sums of these values

$$\mathbf{sa}[\mathbf{x}_n] = \sum_{m=1}^{N} a[\mathbf{x}_n, \mathbf{x}_m]\mathbf{v}_m$$

Weights depend on the inputs themselves

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

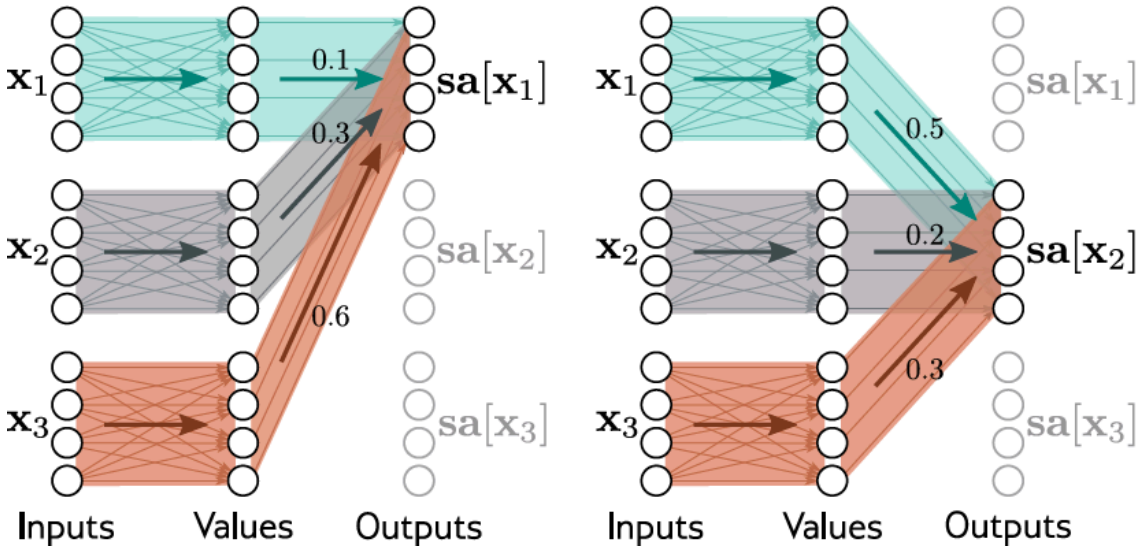
# Attention as routing




These values is how do other words related to word 1(including itself)
Their sum is 1. sort of "partitioning" the attention between the words as they relate to word 1

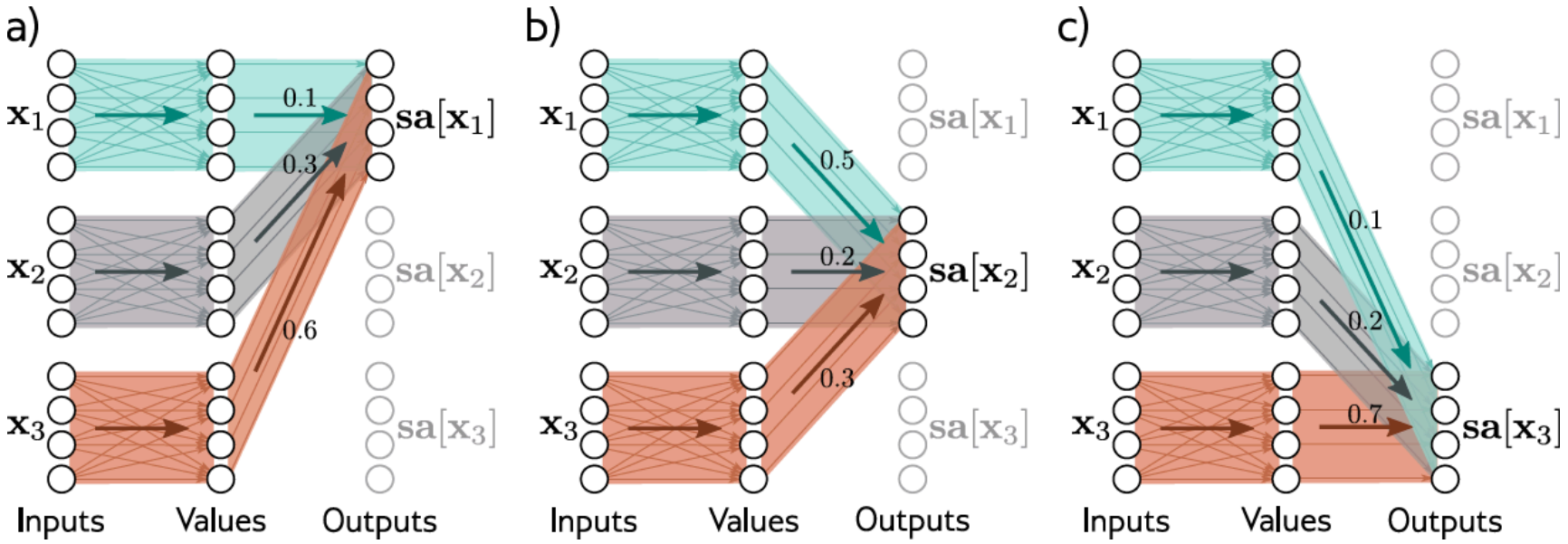
Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention as routing




Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention as routing




self-attention can be thought of as *routing* the values in different proportions to create each output

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention as routing

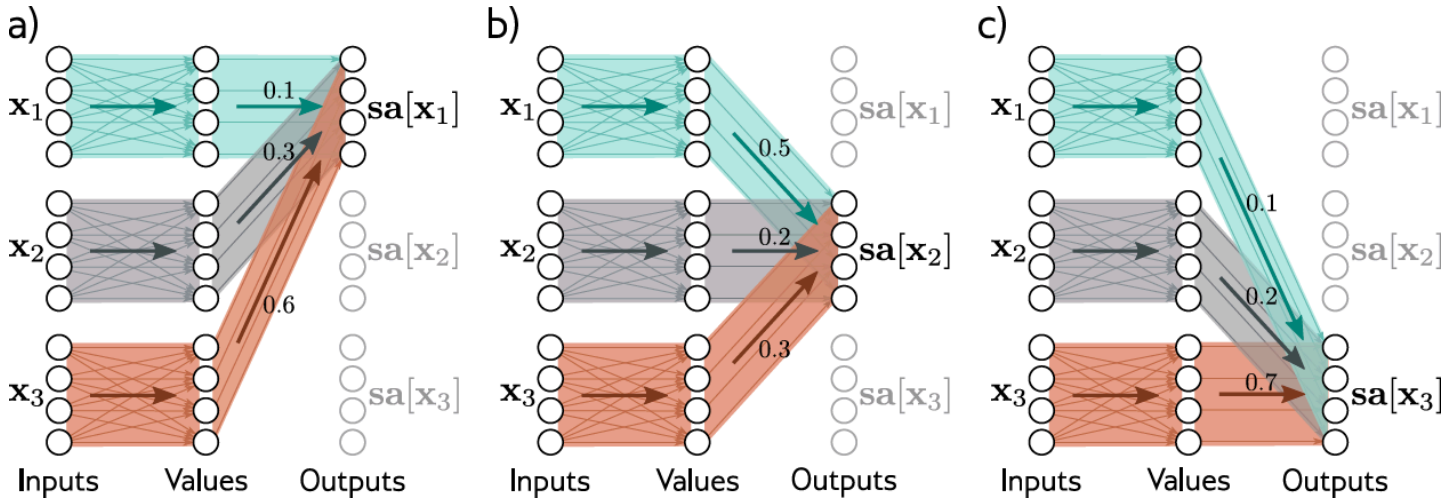



**Figure 12.1** Self-attention as routing. The self-attention mechanism takes $N$ inputs $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$ (here $N = 3$ and $D = 4$) and processes each separately to compute $N$ value vectors. The $n^{th}$ output $\mathbf{sa}_n[\mathbf{x}_1, \ldots \mathbf{x}_N]$ (written as $\mathbf{sa}_n[\mathbf{x}_\bullet]$ for short) is then computed as a weighted sum of the $N$ value vectors, where the weights are positive and sum to one. a) Output $\mathbf{sa}_1[\mathbf{x}_\bullet]$ is computed as $a[\mathbf{x}_1, \mathbf{x}_1] = 0.1$ times the first value vector, $a[\mathbf{x}_2, \mathbf{x}_1] = 0.3$ times the second value vector, and $a[\mathbf{x}_3, \mathbf{x}_1] = 0.6$ times the third value vector. b) Output $\mathbf{sa}_2[\mathbf{x}_\bullet]$ is computed in the same way, but this time with weights of 0.5, 0.2, and 0.3. c) The weighting for output $\mathbf{sa}_3[\mathbf{x}_\bullet]$ is different again. Each output can hence be thought of as a different routing of the $N$ values.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention

This is Great, but where do the "a"s come from. They are at the core of this.

How does the "attention" of a word to another get computed?

If you have two values and you want to "measure" how they relate "without" impacting them, take a "copy" of them by multiplying them with some parameter. Note that that parameter has to be the same within the attention head (or else, you will be not measuring/listening equally to the impact of the words to each other).

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention weights

we apply two more linear transformations to the inputs:

Compute N "queries" and N "keys" from input

$$\mathbf{q}_n = \boldsymbol{\beta}_q + \boldsymbol{\Omega}_q \mathbf{x}_n$$
$$\mathbf{k}_n = \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{x}_n,$$

Calculate similarity (dot product) and pass through softmax:

$$a[\mathbf{x}_n, \mathbf{x}_m] = \text{softmax}_m\left[\mathbf{k}_m^T \mathbf{q}_n\right]$$
$$= \frac{\exp\left[\mathbf{k}_m^T \mathbf{q}_n\right]}{\sum_{m'=1}^{N} \exp\left[\mathbf{k}_{m'}^T \mathbf{q}_n\right]}$$

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention weights

$$a[\mathbf{x}_n, \mathbf{x}_m] = \text{softmax}_m\left[\mathbf{k}_m^T \mathbf{q}_n\right]$$
$$= \frac{\exp\left[\mathbf{k}_m^T \mathbf{q}_n\right]}{\sum_{m'=1}^{N} \exp\left[\mathbf{k}_{m'}^T \mathbf{q}_n\right]}$$

a is nothing more than a % which was created from the similarity (dot product) of "copies" of the input.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention weights

While these are learnable, each is the same across the attention head.

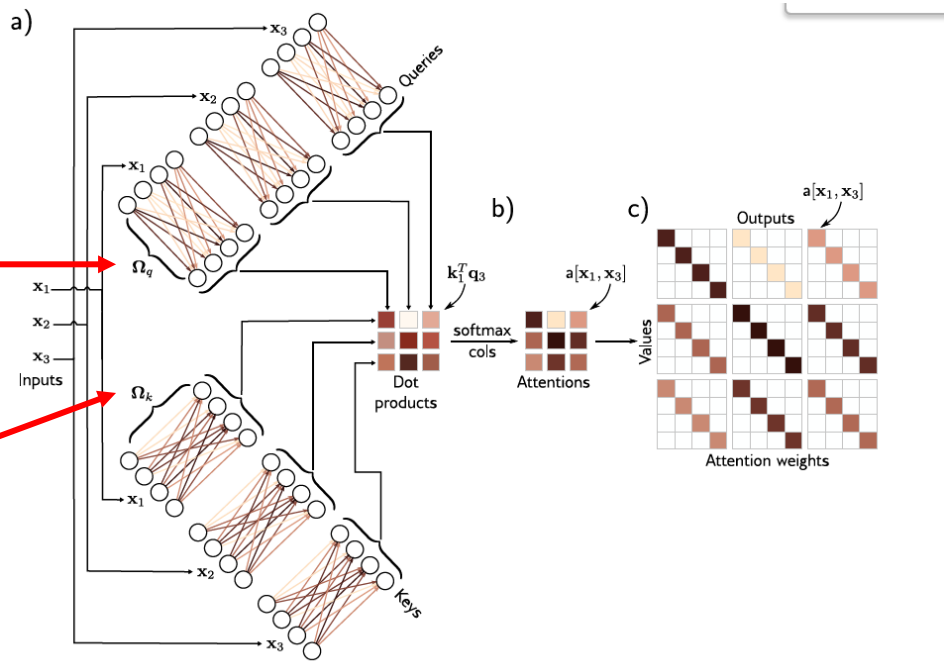
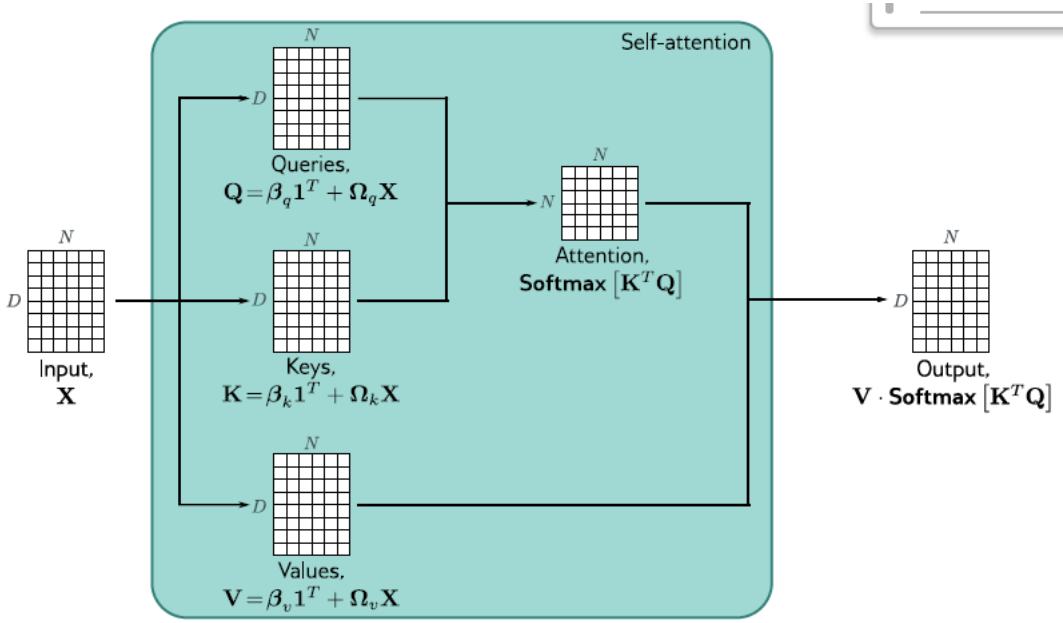



**Figure 12.3** Computing attention weights. a) Query vectors $\mathbf{q}_n = \boldsymbol{\beta}_q + \boldsymbol{\Omega}_q \mathbf{x}_n$ and key vectors $\mathbf{k}_n = \boldsymbol{\beta}_k + \boldsymbol{\Omega}_k \mathbf{x}_n$ are computed for each input $\mathbf{x}_n$. b) The dot products between each query and the three keys are passed through a softmax function to form non-negative attentions that sum to one. c) These route the value vectors (figure 12.1) via the sparse matrix from figure 12.2c.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention Summary




Self-attention in matrix form. Self-attention can be implemented efficiently if we store the *N* input vectors $\mathbf{x}_n$ in the columns of the *D*×*N* matrix **X**. The input **X** is operated on separately by the query matrix **Q**, key matrix **K**, and value matrix **V**. The dot products are then computed using matrix multiplication, and a softmax operation is applied independently to each column of the resulting matrix to calculate the attentions. Finally, the values are post-multiplied by the attentions to create an output of the same size as the input

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Attention Summary

The model parameters don't increase with input length. So we can apply the parameters to different sentence length

$$\phi = \{\boldsymbol{\beta}_v, \boldsymbol{\Omega}_v, \boldsymbol{\beta}_q, \boldsymbol{\Omega}_q, \boldsymbol{\beta}_k, \boldsymbol{\Omega}_k\}$$

- There are connections between the words.
- The strength of these connections will depend on the words themselves via the attention weights.
- Note: During training, All these parameters are learnt. During inference, they need to be computed.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

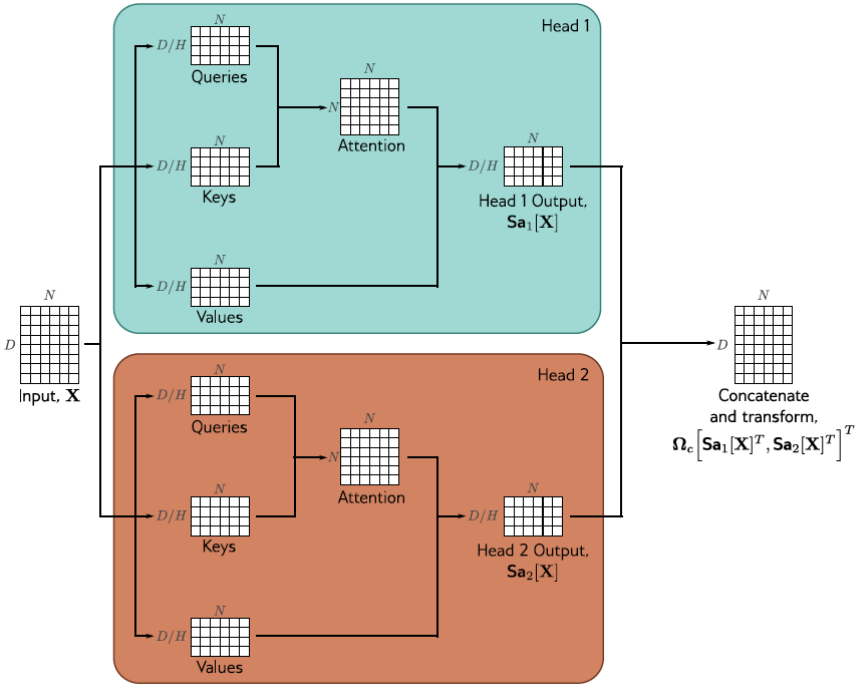
# More than one Attention Head….

Multiple self-attention mechanisms are usually applied in parallel, and this is known as *multi-head self-attention*. Now *H* different sets of values, keys, and queries are computed

Typically, if the dimension of the inputs $\mathbf{x}_m$ is *D* and there are *H* heads, the values, queries, and keys will all be of size *D/H*, as this allows for an efficient implementation. The outputs of these self-attention mechanisms are vertically concatenated, and another linear transform $\mathbf{\Omega}_c$ is applied to combine them

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

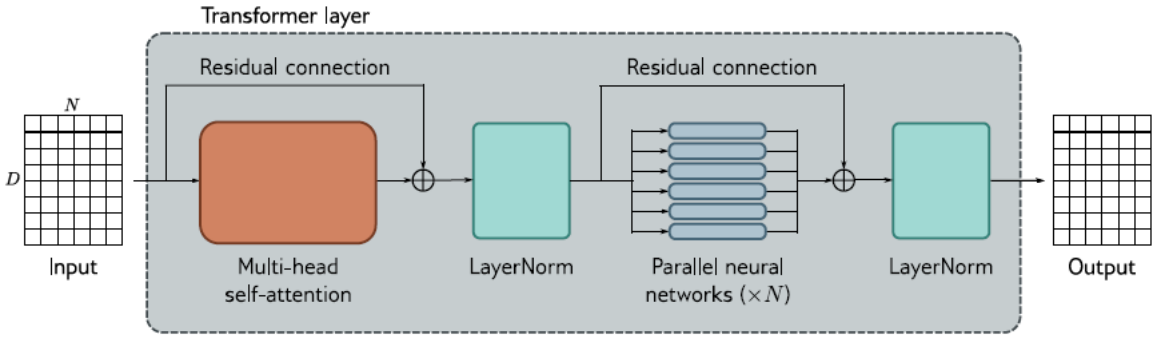
# More than one Attention Head….




Multi-head self-attention. Self-attention occurs in parallel across multiple "heads." Each has its own queries, keys, and values. Here two heads are depicted, in the cyan and orange boxes, respectively. The outputs are vertically concatenated, and another linear transformation $\boldsymbol{\Omega}_c$ is used to recombine them.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

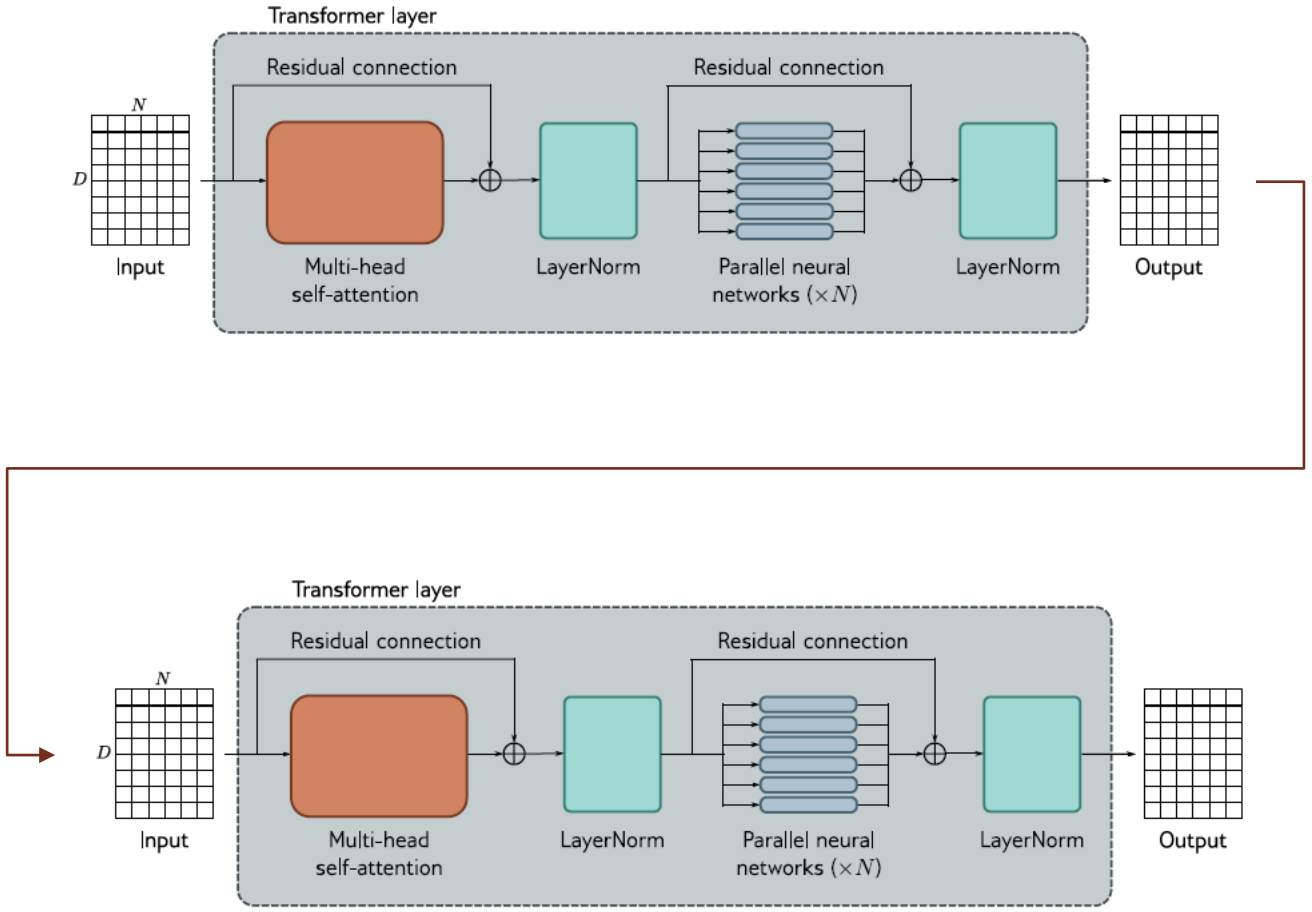
# Transformer Layer




Transformer layer. The input consists of a $D \times N$ matrix containing the $D$-dimensional word embeddings for each of the $N$ input tokens. The output is a matrix of the same size. The transformer layer consists of a series of operations. First, there is a multi-head attention block, allowing the word embeddings to interact with one another. This forms the processing of a residual block, so the inputs are added back to the output. Second, a LayerNorm operation is applied. Third, there is a second residual layer where the same fully connected neural network is applied separately to each of the $N$ word representations (columns). Finally, LayerNorm is applied again.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Transformer Layer




Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Transformer Layer: FFN

- The attention layers enrich the word embedding with relationships to other words in the sentence, but does not provide  enable an expansion of the word itself. i.e. the attention mechanism does not inherently increase the dimensionality or the informational capacity of the word embedding. It's about the context and not about expanding the representational space
- → The need for a Feed Forward network
- The FFN has 4 X D  i.e. we take the word embedding into higher dimension to allow further fine graining the word itself.
- Then it is “compressed back into D”. To bring it back inline with the model for easier processing.
- Note that all the words share the same weights and are processed in parallel. This is core to the gain in efficiency (and why GPUs are super critical in the NLP space).

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Transformer Training

- BERT initial release was trained on mainly two tasks
  - MASK word prediction
  - This is done by feeding the final “super rich” embedding sequences into a predicting head which uses the vocabulatory tokens and a softmax to pick one of them as the most likely for the [Mask].
  - During training, the loss function is computed and network is tuned using backpropagation (ANNs).
  - Next Sentence Prediction
    - In the next sentence prediction, the “prediction” of that head is whether “isnext sentence” is correct or not.

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# BERT

- BERT has 12 of these layers, each with 12 attention heads!!.
- Each Head is connected parallelly to the next head in the same transformer layer, whereas layers are connected in series.
- Residual connections are done within the layer (i.e. adding back the input is within the layer, but not across layers).

Src: Prince, S. J. (January 28, 2024). *Understanding Deep Learning.* MIT Press.

# Library Analogy- Keys, Values & Query

•Imagine you're in a large library, and you're looking for books on a particular topic, say "Artificial Intelligence." Here's how the analogy maps to the keys, queries, and values in the context of self-attention:

•Query: This is like you, the searcher, having a specific question or topic in mind — "I want books on Artificial Intelligence." The query represents what you are looking for.

# Library Analogy- Keys, Values & Query

•Key: Each book in the library has a unique title on its spine, acting as an identifier or "key." In our analogy, the key is the title of the book, which you use to decide if the book might contain the information you're seeking. In the self-attention mechanism, the key helps determine how much attention (relevance) a particular piece of information (in our case, a book) should get in relation to the query.

•Value: Assuming the book's title (key) matches your interest, the "value" would be the content or subject matter of the book itself — the information you gain by selecting this book. In the library, if the title (key) aligns well with your query, the value is what you take away from the book.

# Attention- Keys, Values & Query

•Keys (K), Queries (Q), and Values (V) are vectors derived from the input data.

•Each input token is transformed into these vectors through trainable linear transformations.

•Queries represent the part of the data that we're currently focusing on.

•Keys correspond to parts of the input we want to draw information from.

•Values hold the actual information of the input that we want to use for creating the output.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Creation of Keys, Values & Query

- Embed input tokens into a high-dimensional space.
- Apply learned weight matrices to these embeddings to create keys, queries, and values.
- Dimensionality of keys (and queries) determines the depth of the attention mechanism.
- These vectors are essential for computing the attention scores.
- The process is learned end-to-end through backpropagation and gradient descent.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Scaled Dot Product Attention mechanism

- Attention(Q, K, V) = softmax(($QK^T$)/sqrt($d_k$))V, where $d_k$ is the dimension of keys.
- Dot products of queries with keys determine relevance scores.
- Scores are scaled to avoid vanishing gradients during training.
- Softmax function turns scores into probabilities, normalizing them.
- These probabilities are used to create a weighted sum of values.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

•Each Transformer layer contains two main sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network.
•**Normalization and Feed-Forward**: After attention calculation, the output typically undergoes layer normalization and then is passed through a feed-forward neural network. This network is the same across different positions but has different parameters from layer to layer.

•**Residual Connections**: Each of these two sub-layers has a residual connection around it, followed by layer normalization. This means the output of each sub-layer is LayerNorm($x$+Sublayer($x$)), where Sublayer($x$) is the function implemented by the sub-layer itself. This helps in mitigating the vanishing gradient problem and allows for deeper models.

•**Processing "cat" Example**: The representation of "cat" is refined through each layer, considering its context (informed by attention scores with other words like "sat"). After several layers, this representation is much richer and context-aware compared to the initial embedding

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

In the case of generative tasks like text completion:
•The final output of the Transformer layers (after the last layer) is a set of vectors representing each input token in a contextually enriched form.
•These vectors are then passed through a final linear layer, which is often followed by a softmax layer to generate a probability distribution over possible next tokens.

•**Next Word Prediction for "The cat sat on..."**: If our input sequence were "The cat sat on...", the model would use the contextual information encoded in the vectors to predict the next token. For instance, the model might predict "the" with high probability, followed by lower probabilities for other tokens.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

If generating text:
•Once a token is predicted, it is added to the sequence, and the entire process starts over for the next token.

•**Example with "the" as Next Token**:

- Suppose the model predicts "the" as the next token after "The cat sat on..."
- The new sequence "The cat sat on the" is re-tokenized, and the process (steps 1-8) is repeated to predict the subsequent word.
- This iterative process continues until the model generates an end-of-sequence token or reaches a predefined maximum length.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

let's go through the process of translating the English sentence "The cat sat on the mat" into Russian using the self-attention mechanism in a Transformer model. The Russian translation we're aiming for could be "Кот сел на коврик" (Kot sel na kovrik).

Step 1: Input Embedding
•First, each word in the English sentence is converted into an embedded vector representing that word in a high-dimensional space.

Step 2: Generating Keys, Queries, and Values
•The embedded vectors are transformed into keys (K), queries (Q), and values (V) for each word.
•For instance, "cat" would get its own set of K, Q, and V.

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

Step 3: Attention Calculation for "cat"

•As the model prepares to translate "cat", it generates a query (Q) for this word, considering the entire sentence's context.

•Keys (K) are generated for every word in the English sentence to evaluate their relevance to "cat".

•Values (V) hold the semantic content for each word, which will be used to form the Russian sentence

Step 4: Dot Product and Softmax

•The dot product is computed between the query for "cat" and all keys, assigning a relevance score to each English word.

•Softmax normalizes these scores to a probability distribution, indicating the importance of each word's value in relation to "cat".

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

Step 5: Weighted Sum of Values
The model takes a weighted sum of the values based on the softmax probabilities, which encapsulates the contextual information needed for translation.

Step 6: Generating the Russian Word
Using the gathered context, the Transformer's decoder generates the Russian word "Кот" as the translation for "cat".
Attention weights might emphasize the verbs and prepositions related to "cat" to grasp the action and location, crucial for an accurate translation

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# Foundations of Language Modeling – Layer-wise Processing

Step 7: Continuing the Process
This process is repeated for each word in the sentence, with the self-attention mechanism ensuring contextual relevance for the translation.
The model translates "sat" to "сел" (sel), "on" to "на" (na), and "the mat" to "коврик" (kovrik), considering the context provided by the attention scores.

**Final Output:**
The full sentence in English "The cat sat on the mat" is translated into Russian as "Кот сел на коврик" (Kot sel na kovrik). Throughout the translation process, the self-attention mechanism dynamically adjusts to focus on the appropriate words

Src:Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.
Src: Josh Strommer, Statquestt: Word Embedding and Word2Vec clearly explained

# References

In addition to the references in each slide the below are leveraged throughout this course.


Gareth James, D. W. (2023). An Introduction to Statistical Learning: with Applications in Python. Springer.

VanderPlas, J. (2017). Python Data Science Handbook. O'Reilly.
Wolff, S. G. (2018). Less is more: optimizing classification performance through feature selection in a very-high-resolution remote sensing object-based urban application. GIScience & Remote Sensing. doi:10.1080/15481603.2017.1408892

Prince, S. J. (January 28, 2024). Understanding Deep Learning. MIT Press.
Josh Stormer, https://statquest.org/