PEGASUS Model For Text Summarization

(Pre-training with Extracted Gap-sentences for Abstractive Summarization Sequence-to-sequence models)

Submitted by
Name : Dakshita Arora
ID : BTBTC22171
Roll Number : 2216258
Course : BTech-CS
Section : A

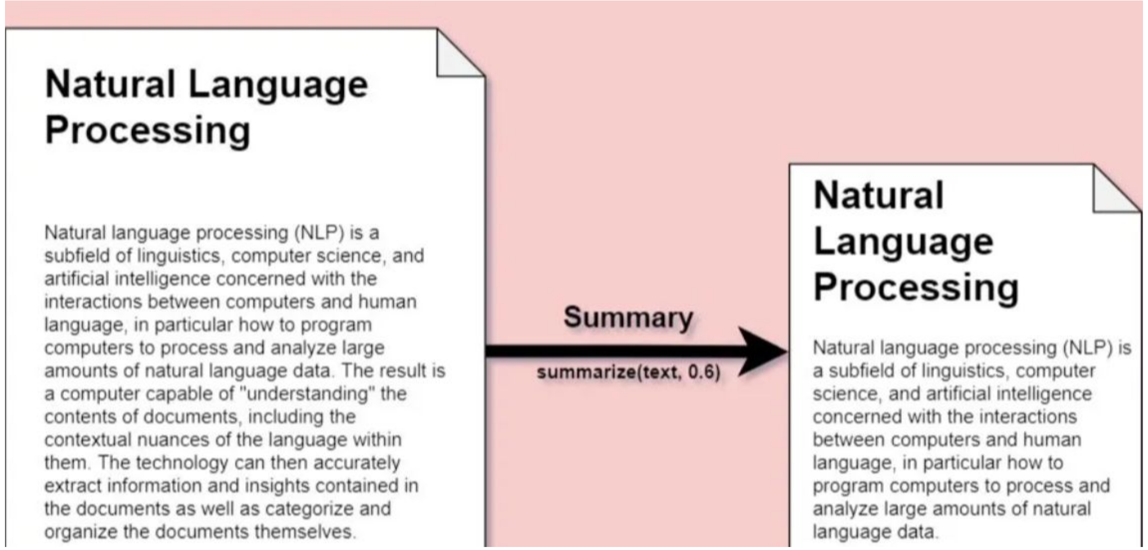
Outlines

- Overview of the PEGASUS Model
- Architecture of Pegasus Model
- Gap-Sentences Generation (GSG) Pretraining
- Working of Pegasus Model
- Applications
- Advantages and Disadvantages of PEGASUS Model
- Conclusion
- References

Overview of the Pegasus Model

Introduction to Pegasus:

- Pegasus is a state-of-the-art model specifically designed for abstractive text summarization.
- PEGASUS, based on the powerful **Transformer architecture**, brings significant advancements in how machines summarize text by pretraining on **gap-sentence prediction**.



Architecture of Pegasus Model

1.Input Embedding & Positional Encoding

2.Encoder

- ❖ Multi-Head Attention (Self-Attention)
- ❖ Feed-Forward Layer

3.Decoder

- ❖ Masked Multi-Head Attention(Self Attention)
- ❖ Multi-Head Attention (Cross-Attention)
- ❖ Feed-Forward Layer

4.Output Probabilities

5.Add & Norm

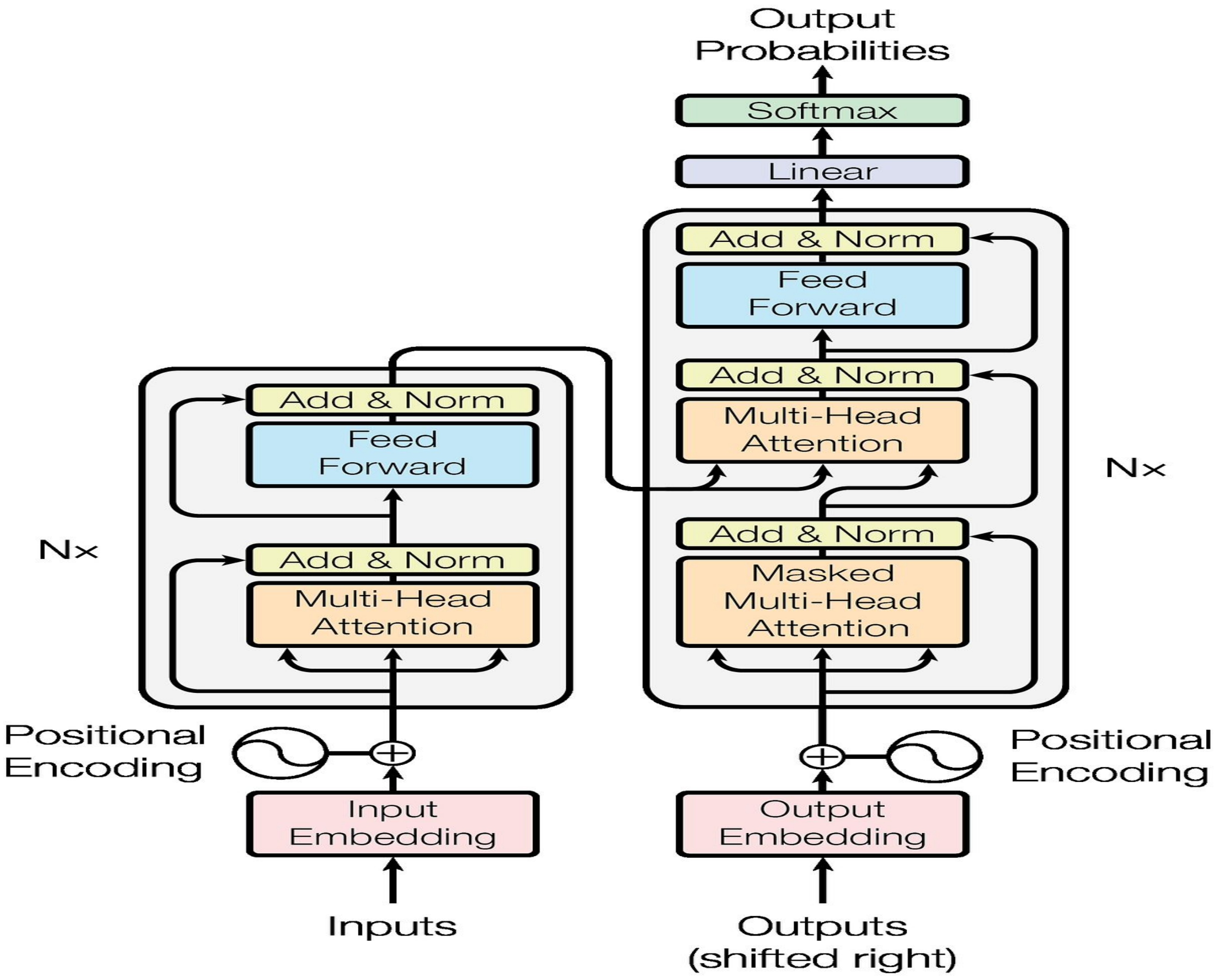


Fig : Transformer Architecture of PEGASUS Model

Gap-sentences Generation (GSG) Pretraining

Gap-sentences Generation (GSG) is a novel pre-training strategy used in the PEGASUS model, where entire sentences are masked and the model is tasked with predicting these sentences.

Working Of GSG

It involves two steps:

- Sentence Masking
- Prediction Task

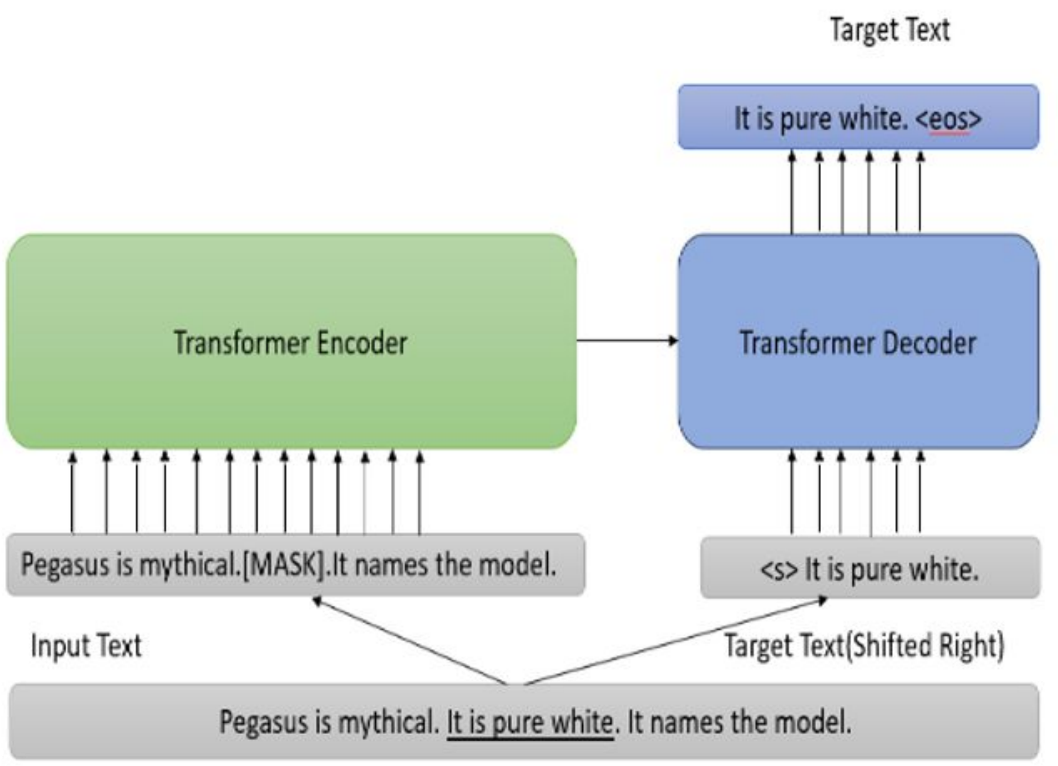
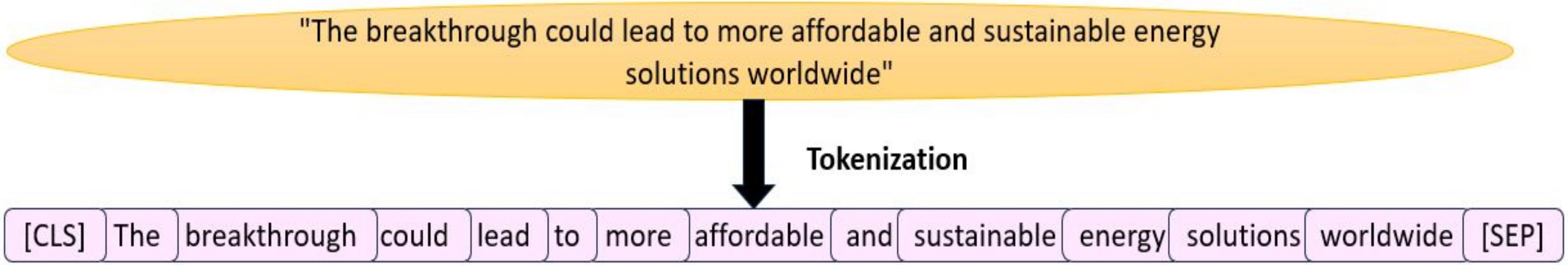


Fig 2. Gap Sentence Generation (GSG)

WORKING OF PEGASUS MODEL

1st Step : Input Processing

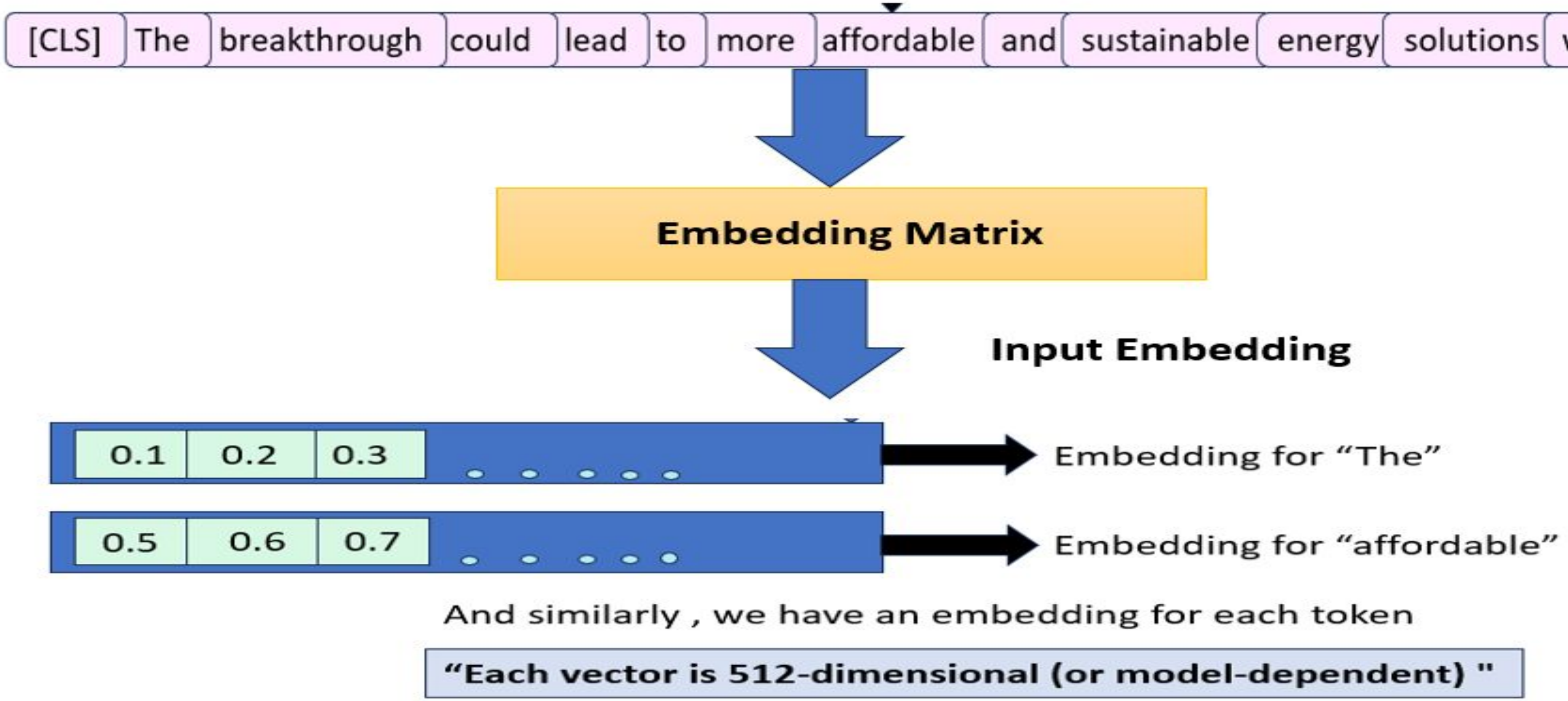
- **Tokenization**
  - Tokenization is the process of converting raw text into smaller, manageable units called tokens.



[CLS] (Classification) token is added at the beginning of the input sequence. It is a placeholder that is used for capturing the overall representation of the sequence.

[SEP] (Separation) token is used to separate distinct portions of the input. It is added at the end of the sequence or between sentence pairs.

- **<u>Input Embedding</u>**
  - Converts tokens into dense vector representations or embeddings, capturing semantic meaning.
  - Each token is mapped to a high-dimensional vector from a pre-trained embedding matrix.
  - These embeddings are learned during model training and refined for specific tasks.



- **<u>Positional Encoding</u>**
  - Adds information about the position of each token in the sequence since Transformers don't inherently understand sequence order.

0.1	0.2	0.3	. . . . .

→ Embedding for "The"

0.5	0.6	0.7	. . . . .

→ Embedding for "affordable"

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

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

-0.28	0.96	-0.64	. . . .

→ Positional Encoding for "affordable"

- **<u>Addition of Input Embedding and Positional Encoding</u>**

Final Vector for "affordable" :

0.5	0.6	0.7	. . . .

+

-0.28	0.96	-0.64	. . . .

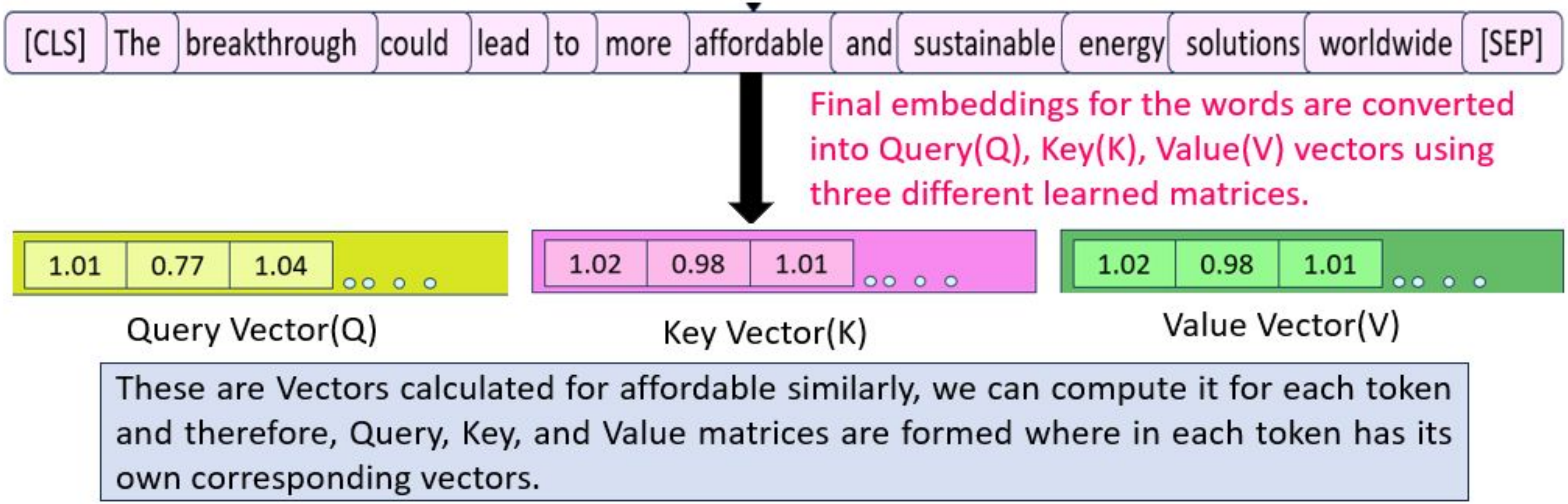
=

0.22	1.56	0.06	. . . .

2nd Step: Encoding the input text

- **Multi-head self-attention**
  - It enables each token to attend to all other tokens, capturing relationships between words.
  - For each token in the sequence, the model calculates three vectors:

Query (Q) => $E(t_i)W_Q$ Key (K) => $E(t_i)W_K$ Value(V) => $E(t_i)W_V$



- Attention Mechanism

$$\text{Attention}(Q_i, K_j, V) = \text{Softmax}\left(\frac{Q_i \cdot K_j^T}{\sqrt{d_k}}\right) \cdot V$$

"The" → 0.1 breakthrough → 0.2 lead → 0.3 ………… and so on

If these are the normalized attention scores of each word w.r.t affordable. After applying self-attention, the final vector for "affordable" will be something like this.

Final Vector for affordable = 0.1 × [1.05 | 0.92 | 1.01 | …] + 0.2 × [0.19 | 0.03 | 1.07 | …]

…………

= [1.02 | 1.24 | 1.19 | …]

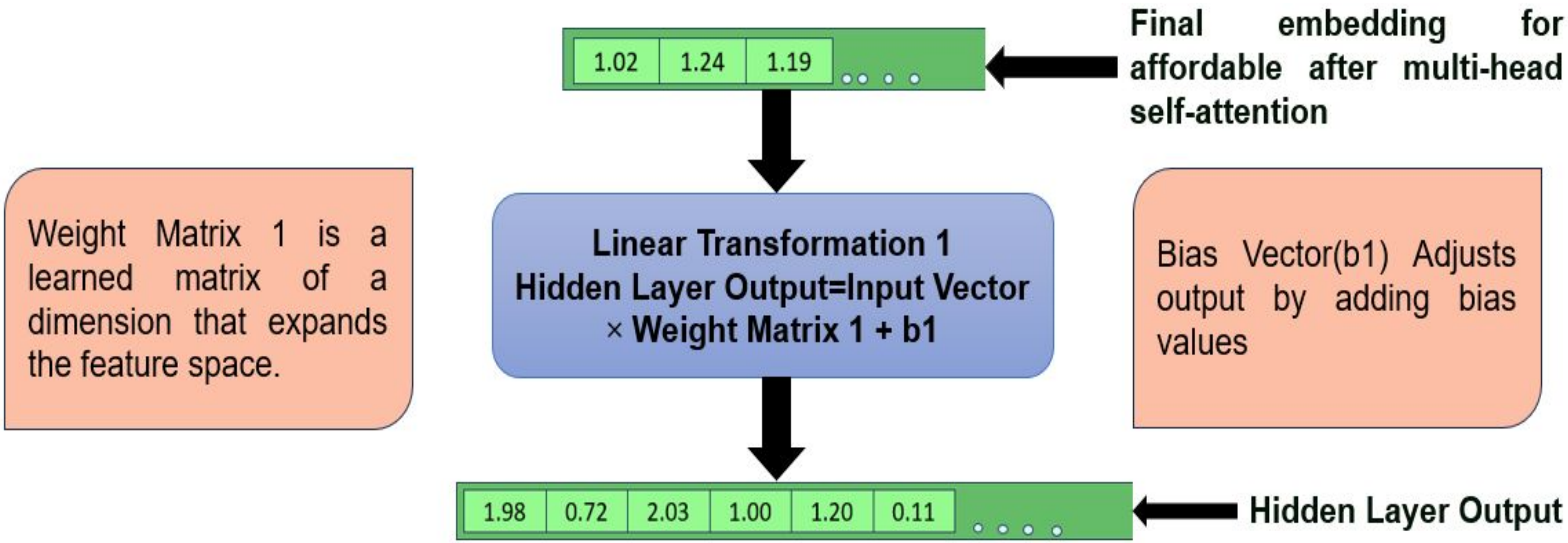
Similarly, we will have this weighted sum vector for each token in the sequence.

- Instead of computing a single attention score, Pegasus uses multiple attention heads (h), allowing the model to focus on different parts of the sequence.
- The outputs from all heads are concatenated and passed through a linear transformation.

- **<u>Feed-Forward Network</u>**
  - A simple feed-forward neural network applied to each position separately, allowing the model to capture more complex transformations.

    Structure:

  - Two linear transformations with a ReLU activation in between.



1.98	0.72	2.03	1.00	1.20	0.11	. . . .

Hidden Layer Output

Non-Linear Activation (ReLU):
ReLU(x) = max (0, x)

Since, there are no negative Values ,Output after applying non-linear activation remains the same

1.98	0.72	2.03	1.00	1.20	0.11	. . . .

ReLU Output

Learned weight matrix W_2, matrix maps the hidden layer back to the original input dimension

Linear Transformation 2:
ReLU Output× Weight Matrix 2+b2

Bias Vector(b2) Adjusts output by adding bias values

1.31	0.26	1.97	. . . .

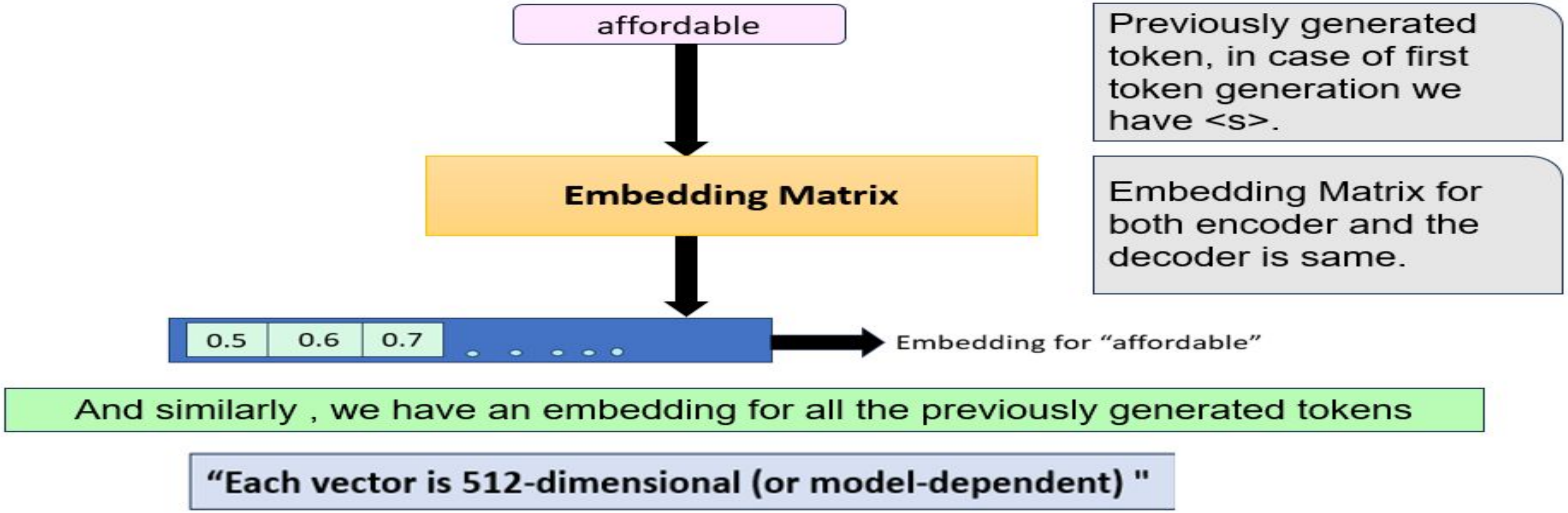
Final Output

This process occurs for every token in the input. A sequence of contextually rich vectors representing the input is ready for the Pegasus model decoder.

3rd Step: Decoding

- **Input to Decoder**
  - Decoder operates in an autoregressive manner, generating one token at a time.
  - It uses the previously generated token as input along with context from the encoder's output vectors.

    In this example,we have assumed that decoder has already generated a sequence "< s > affordable".



- **<u>Positional Encoding</u>**
  - Similar to the encoder, the decoder adds positional encoding to the token embeddings to account for the token's position in the sequence.

0.5	0.6	0.7	. . . . .

→ Embedding for “affordable”

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

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

0.84	0.54	0.82	. . . .

→ Positional Encoding for “affordable”

- **<u>Addition of Input Embedding and Positional Encoding</u>**

Final Vector for “affordable” :

0.5	0.6	0.7	. . . .

+

0.84	0.54	0.82	. . . .

=

1.34	1.14	1.52	. . . .

- **Masked Multi-Head Attention(Self Attention)**

  - It helps the decoder capture relationships between previously generated tokens, ensuring coherent and contextually aware output.It prevents the decoder from attending to future tokens
  - **Masking:** Blocks future tokens by assigning low values to prevent attending to them.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q_{i-1} \cdot K_{i-1}^T}{\sqrt{d_k}} + M\right) \cdot V_{i-1}$$

We have Q, K, and V are matrices where each previously generated token has its own corresponding query, key, and value vectors. These Query(Q),Key(K) and Value(V) vectors are computed using learned weight matrices

Previously generated tokens

<s>	affordable

Decoder

Here, the decoder can only attend to the tokens that were generated previously and masks the future tokens

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{Q_{i-1} \cdot K_{i-1}^{T}}{\sqrt{d_k}} + M\right) \cdot V_{i-1}$$

The attention scores are computed between the current token i.e. affordable and all the previously generated tokens including itself

These are the normalized attention scores, i.e.

Between affordable and <s>= 0.1

Between affordable and affordable=0.2

A mask is applied which ensures that attention score will be 0 for all the future tokens, due to the softmax function.

Final Output vector after masked self-attention for generating the third token will be as follows:

$$0.1 \times \begin{bmatrix} 0.01 & 0.12 & 1.49 & \cdots \end{bmatrix} + 0.2 \times \begin{bmatrix} 1.08 & 0.20 & 1.89 & \cdots \end{bmatrix} = \begin{bmatrix} 0.217 & 0.052 & 0.527 & \cdots \end{bmatrix}$$

- Cross-Attention Mechanism
    - The decoder uses cross-attention to focus on relevant parts of the encoder output while generating the next token.

The	breakthrough	could	lead	to	more	affordable	and	sustainable	energy	solutions	worldwide

Key(K) and Value(V) Vectors are computed for each token using the learned matrices just like in the encoder.

The cross-attention score is calculated between the decoder's query vector and the encoder's key vectors

affordable → Query of the Current token in the decoder

$$\text{Cross-Attention Score} = \frac{Q_{i-1} \cdot K_{\text{enc}}^T}{\sqrt{d_k}} \qquad z_{\text{cross}} = \sum \left(\text{softmax}(\text{Cross-Attention Score}) \cdot V_{\text{enc}}\right)$$

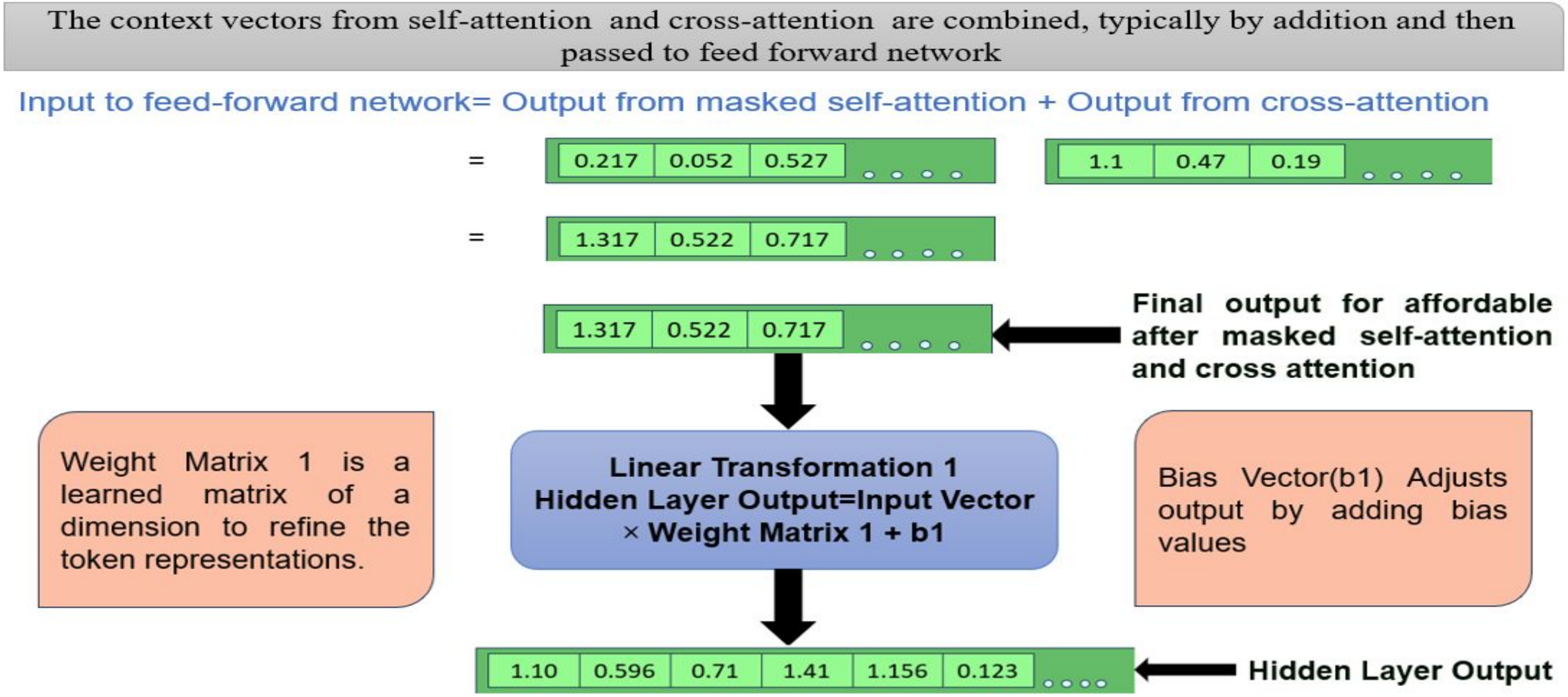
The → 0.1 lead → 0.2 energy → 0.3 could → 0.03

and so on.......

Final Output = 0.1 X [0.36 | 0.1 | 0.5 | . . . .] + 0.2 X [1.2 | 0.6 | 0.57 | . . . .] ....

= [1.1 | 0.47 | 0.19 | . . . .]

- <u>Feed-Forward Network</u>
  - This enhances the representation of the current token and helps model to refine the token representations.



1.10	0.596	0.71	1.41	1.156	0.123	....

Hidden Layer Output

Non-Linear Activation (ReLU):
ReLU(x) = max (0, x)

Since, there are no negative Values ,Output after applying non-linear activation remains the same

1.10	0.596	0.71	1.41	1.156	0.123	....

ReLU Output

Learned weight matrix W_2, matrix maps the hidden layer back to the original input dimension

Linear Transformation 2:
ReLU Output× Weight Matrix 2+b2

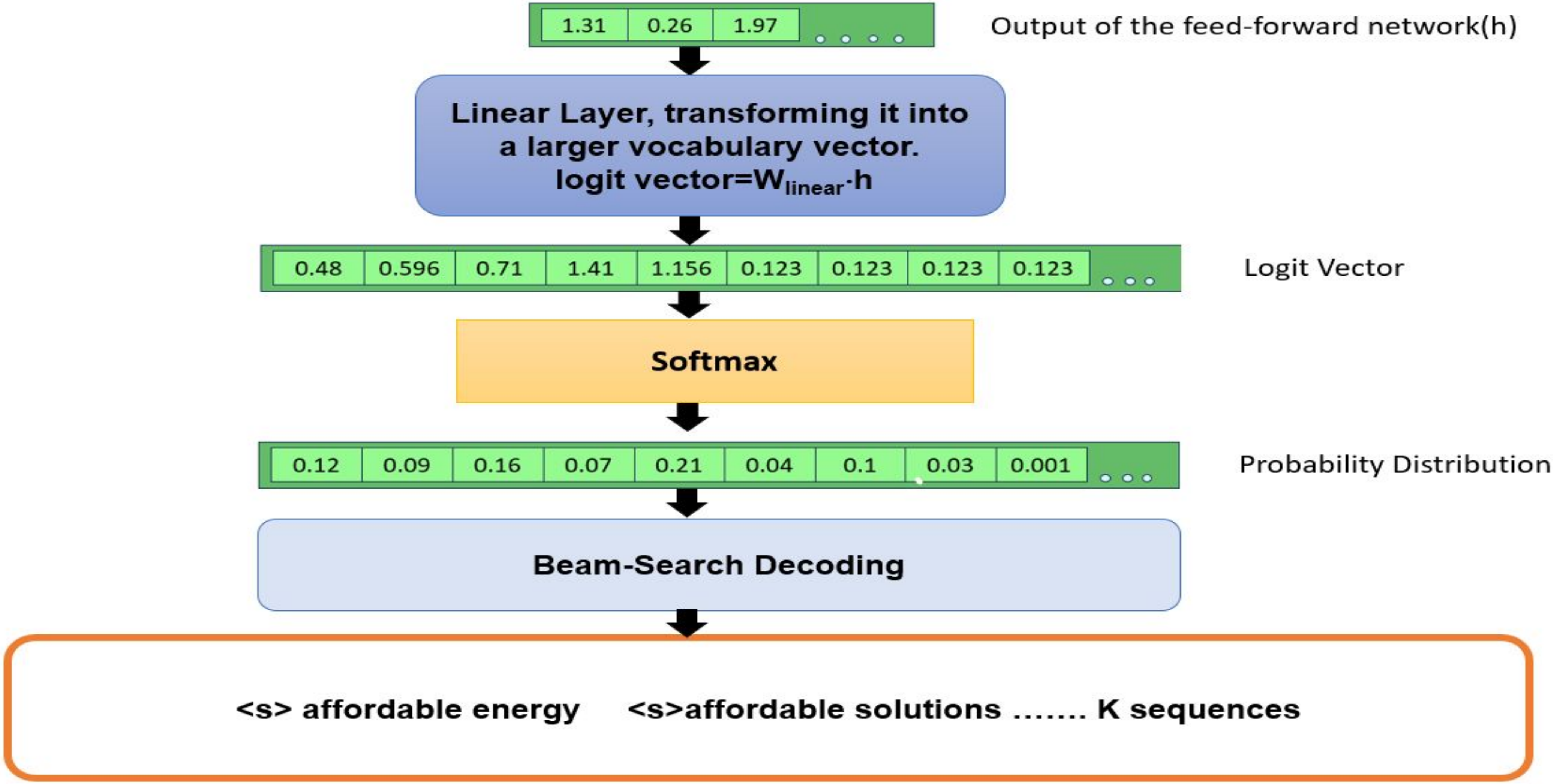
Bias Vector(b2) Adjusts output by adding bias values

1.31	0.26	1.97	....

Final Output

Thus ,we have a vector which will be used to predict the next word.

- <u>Token Generation</u>
    - The refined vector from the FFN is used to predict the next token.



Applications of pegasus model

- News Article Summarization
- Scientific Paper Summarization
- Healthcare Summarization
- Legal Document Summarization

Advantages and Disadvantages

Advantages :

- State-of-the-art Performance: PEGASUS excels on summarization benchmarks, outperforming previous models.
- Flexibility: The model can be fine-tuned for various domains, making it versatile for different tasks.
- Efficiency: It efficiently generates summaries by focusing on key sentences during pre-training.

Disadvantages :

- Complexity: Requires significant computational resources for pretraining and fine-tuning.
- Data Dependency: The quality of summaries depends on the datasets used.
- Ethical Considerations: Potential for generating biased or inappropriate content.

Conclusion

- **Recap:** We explored the **Pegasus** model, focusing on its innovative **Gap-Sentence Generation (GSG)** method, and how it effectively predicts and summarizes key parts of text using **Transformer architecture** and **multi-head attention**.
- Pegasus addresses challenges in **text summarization** by capturing contextual importance, making it a powerful tool for processing large amounts of information across various domains.
- **Looking Ahead:** In the future, the Pegasus model will likely play a key role in improving automatic text summarization, making it easier to process large amounts of information across different fields.

References

1. Zhang, J., Zhao, Y., Saleh, M., Liu, P., Com>, M., Saleh, Com>, P., & Liu. (2020). *PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization*. https://arxiv.org/pdf/1912.08777
2. *Self -attention in NLP*. (2020, September 4). GeeksforGeeks. https://www.geeksforgeeks.org/self-attention-in-nlp/
3. *Feedforward neural network*. (2024, June 20). GeeksforGeeks. https://www.geeksforgeeks.org/feedforward-neural-network/
4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., Kaiser, Ł., & Polosukhin, I. (2017). *Attention Is All You Need*. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf