LITRA model.

Transformer model for translation (The state of the art)

_Summarize the paper ("attention is all you Need"):

The paper titled "Attention Is All You Need" introduces a new network architecture called the Transformer, which is based solely on attention mechanisms and eliminates the need for recurrent or convolutional neural networks. The authors propose this architecture as an alternative to existing sequence transduction models. The Transformer model demonstrates superior quality, parallelizability, and reduced training time compared to traditional models.

The paper begins by discussing the prominence of recurrent neural networks (RNNs) in sequence modeling and transduction tasks such as language modeling and machine translation. However, RNNs suffer from sequential computation and limited parallelization, especially with longer sequences. Attention mechanisms have been used in conjunction with RNNs to model dependencies between positions in sequences. The authors propose the Transformer as a model architecture that relies solely on attention mechanisms, allowing for increased parallelization and improved performance.

The paper presents the Transformer model's architecture and its components, including the encoder and decoder stacks. The encoder consists of multiple layers with self-attention mechanisms and position-wise fully connected networks. The decoder, in addition to the encoder layers, incorporates multi-head attention over the encoder's output. The attention function used in the Transformer is described as scaled dot-product attention, which computes the compatibility between queries and keys to obtain weighted values.

The authors highlight the advantages of the Transformer model over existing approaches, such as reduced computational complexity and increased parallelization. They conduct experiments on machine translation tasks, demonstrating that the Transformer outperforms previous models in terms of translation quality. The Transformer achieves state-of-the-art results on the WMT 2014 English-to-German and English-to-French translation tasks while requiring significantly less training time.

Furthermore, the authors show that the Transformer model generalizes well to other tasks by successfully applying it to English constituency parsing with both large and limited training data. They provide detailed descriptions of the model's architecture, training process, and evaluation metrics.

In conclusion, the paper introduces the Transformer model as a novel architecture for sequence transduction tasks. By leveraging attention mechanisms and eliminating the reliance on recurrent or convolutional networks, the Transformer achieves improved performance, parallelizability, and training efficiency. The experimental results validate the effectiveness of the Transformer on machine translation tasks, setting new state-of-the-art results.

Explanation:

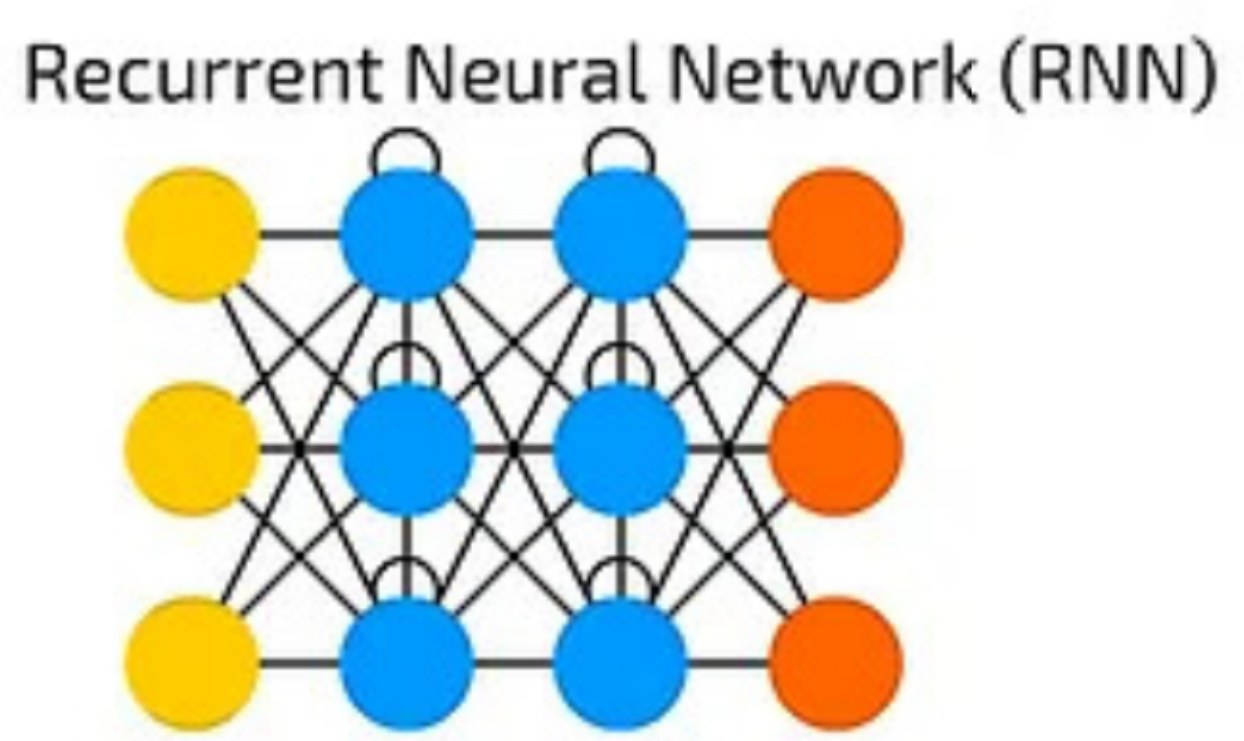
The Transformer neural network is a powerful deep learning model that was introduced in a landmark paper titled "attention is all you need..pdf" by Vaswani et al. in 2017. It revolutionized the field of natural language processing (NLP) and has since found applications in various other domains. The Transformer architecture is based on the concept of attention, enabling it to capture long-range dependencies and achieve state-of-the-art performance on a wide range of tasks.

Applications of the Transformer neural network:

1. **Machine Translation**: The Transformer has achieved impressive results in machine translation tasks, such as translating text from one language to another. Its ability to capture long-range dependencies and handle variable-length input sequences makes it well-suited for this task.
2. **Text Generation**: The Transformer can be used for generating coherent and contextually relevant text. It has been applied to tasks such as generating news articles, dialogue systems, and story generation.
3. **Summarization and Document Understanding**: The Transformer's attention mechanism enables it to focus on important parts of a document or text, making it effective for tasks like text summarization, document classification, and sentiment analysis.
4. **Speech Recognition**: The Transformer has also been applied to automatic speech recognition tasks, where it converts spoken language into written text. It has shown promising results in this domain as well.
5. **Question Answering**: The Transformer's ability to understand and generate text has made it useful in question answering systems. It can process a question and a context paragraph and generate relevant answers.
6. **Image Recognition**: While primarily designed for NLP tasks, the Transformer has also found applications in computer vision tasks. It can be adapted for image recognition tasks by treating images as sequences of patches.

Before the emergence of Transformer Neural Networks (TNNs), Recurrent Neural Networks (RNNs) were commonly employed for sequential processing tasks, including machine translation. However, RNNs were characterized by slow processing speeds, limited accuracy, and challenges in handling large datasets.

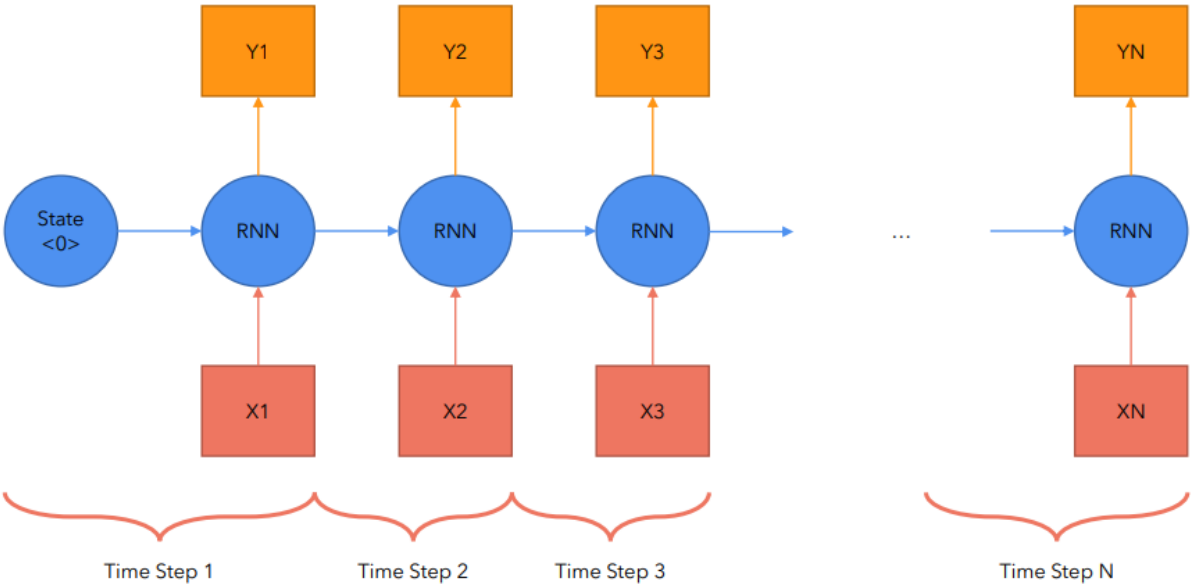
here how RNN woks:
is designed to process sequential data, where the current input not only depends on the current state but also on the previous inputs and states.



suppose we have this sentence "I work at the university.", and we want to translate it to Arabic "انا اعمل في الجامعة" .

In the translation task, the RNN analyzes each word ('I', 'work', 'at', 'the', 'university') one by one, updating the hidden state at each step. The output at each time step is influenced by the current word and the hidden state, which captures the historical information from previous words. The final output is a sequence of translated words

('انا', 'اعمل', 'في', 'الجامعة') in Arabic.



problems with RNN:

1. Slow computation for long sequences
2. Vanishing or exploding gradients
3. Difficulty in accessing information from long time ago

Indeed, RNNs tend to be slow and can struggle with handling large datasets, which can lead to potential confusion or difficulties in processing extensive data. However,

the Transformer Neural Network (TNN) introduced a breakthrough solution called "Self-Attention" in the paper "Attention is all you Need." This innovation addressed these issues and paved the way for subsequent advancements such as GPT, Bert, LLama, stable diffusion, and more.

In this section, I will delve into the details of the Transformer Neural Network as described in the paper:

> The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these

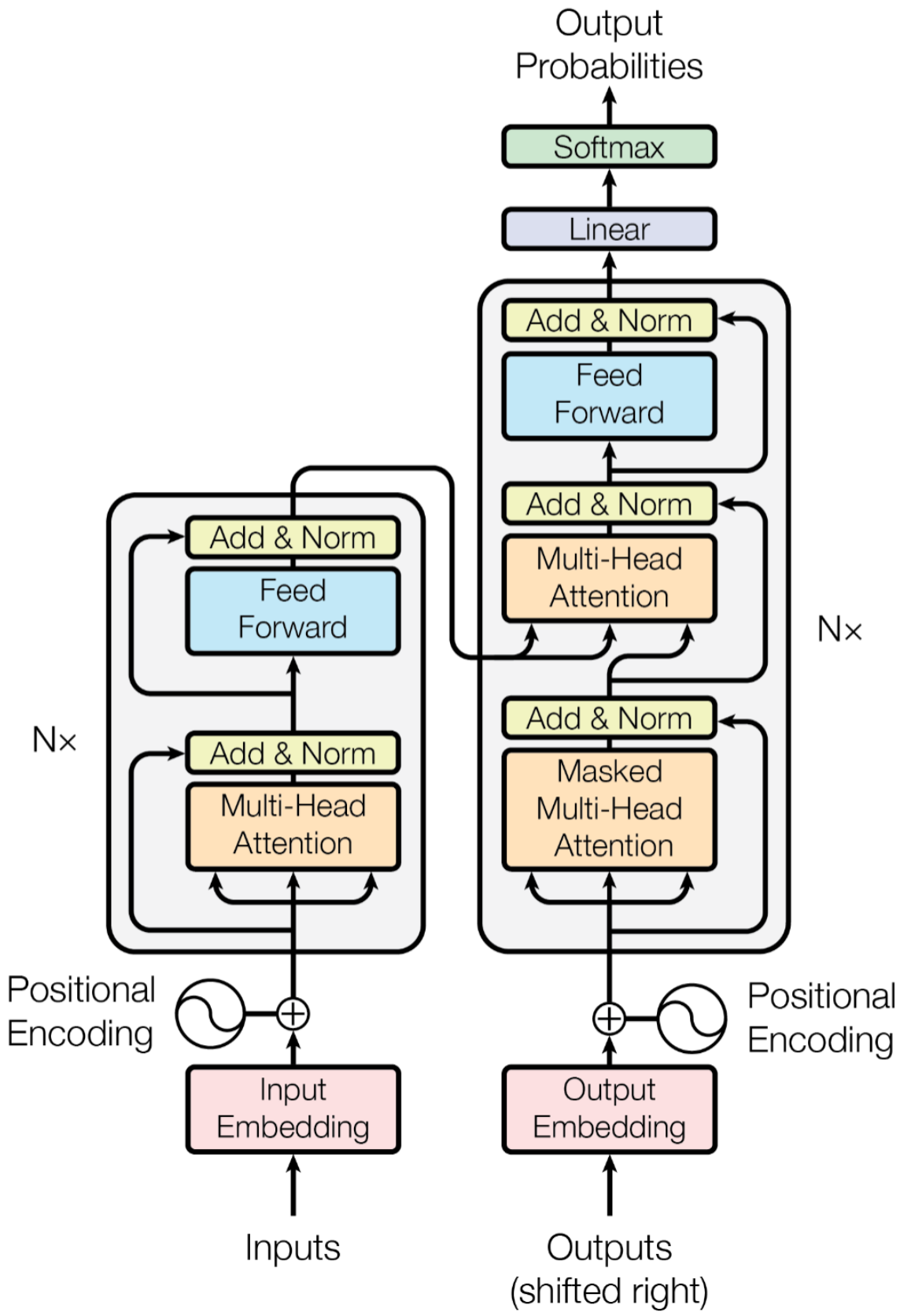
models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data

1 Introduction Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and transduction problems such as language modeling and machine translation [35, 2, 5]. Numerous efforts have since continued to push the boundaries of recurrent language models and encoder-decoder architectures [38, 24, 15]. Recurrent models typically factor computation along the symbol positions of the input and output sequences. Aligning the positions to steps in computation time, they generate a sequence of hidden states ht, as a function of the previous hidden state ht−1 and the input for position t. This inherently sequential nature precludes parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples. Recent work has achieved significant improvements in computational efficiency through factorization tricks [21] and conditional computation [32], while also improving model performance in case of the latter. The fundamental constraint of sequential computation, however, remains. Attention mechanisms have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [2, 19]. In all but a few cases [27], however, such attention mechanisms are used in conjunction with a recurrent network. In this work we propose the Transformer, a model architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between input and output. The Transformer allows for significantly more parallelization and can reach a new state of the art in translation quality after being trained for as little as twelve hours on eight P100 GPUs.

Model Architecture:

Most competitive neural sequence transduction models have an encoder-decoder structure [5, 2, 35]. Here, the encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, ..., ym) of

> symbols one element at a time. At each step the model is auto-regressive [10], consuming the previously generated symbols as additional input when generating the next.

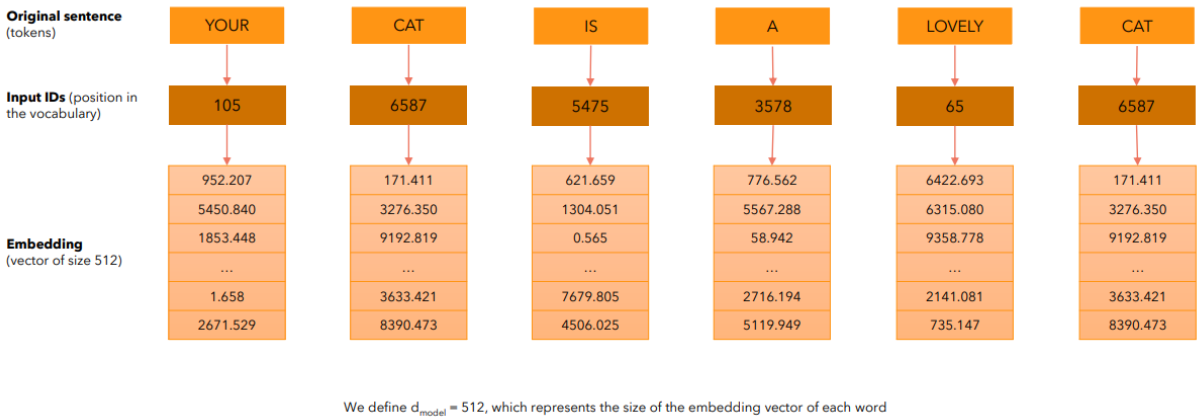


Detailed explanation:

Please check this canvas [Transformer model.] to observe how the peculiar words function, and then proceed to read the following section to understand how the model itself operates.

so first we have the left architecture which is the "encoder" and the right is the "decoder":

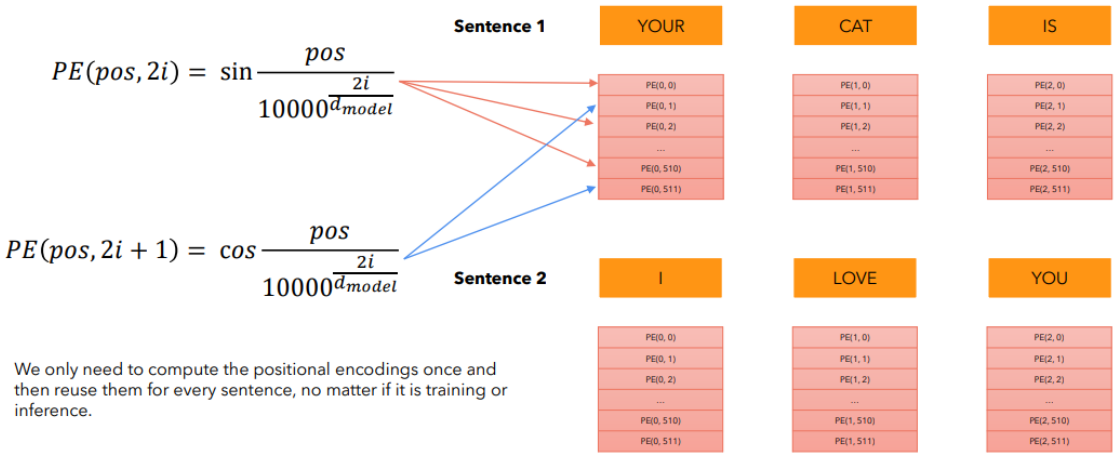
1. **Input Embeddings:**
   The input sequence is transformed into fixed-dimensional embeddings, typically composed of word embeddings and positional encodings. Word embeddings capture the semantic meaning of each word.



2. while **positional encodings** indicate the word's position in the sequence using the sin and cos waves.

$$PE(pos, 2i) = sin(pos/100002i/dmodel)$$

$$PE(pos, 2i + 1) = cos(pos/100002i/dmodel)$$

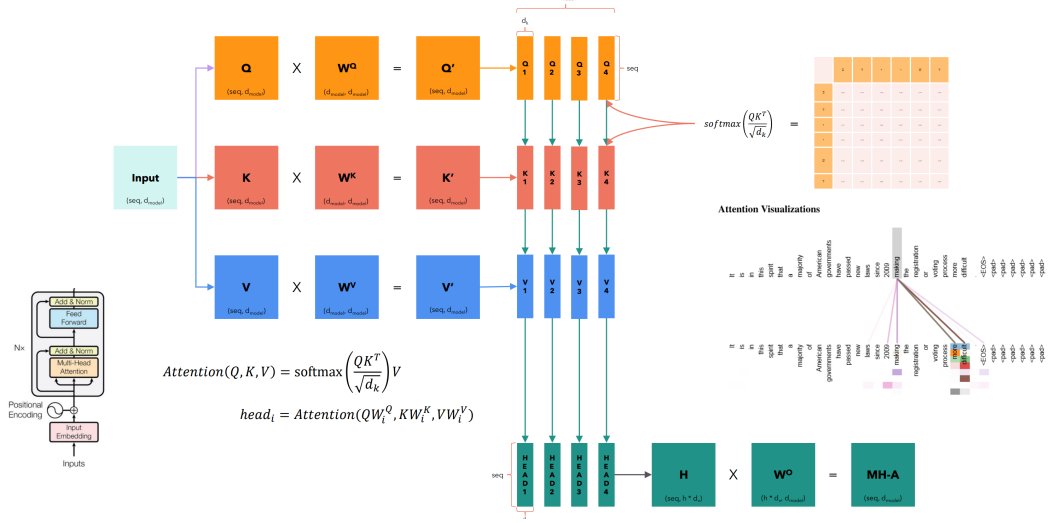


Why trigonometric functions?

Trigonometric functions like cos and sin naturally represent a pattern that the model can recognize as continuous, so relative positions are easier to see for the model. By watching the plot of these functions, we can also see a regular pattern, so we can hypothesize that the model will see it too.

3. **Encoder and Decoder:**
   The Transformer model consists of an encoder and a decoder. Both the encoder and decoder are composed of multiple layers. Each layer has two sub-layers: a multi-head self-attention mechanism and a feed-forward neural network.
   - ***Encoder:*** The encoder takes the input sequence and processes it through multiple layers of self-attention and feed-forward networks. It captures the contextual information of each word based on the entire sequence.
   - ***Decoder:*** The decoder generates the output sequence word by word, attending to the encoded input sequence's relevant parts. It also includes an additional attention mechanism called "encoder-decoder attention" that helps the model focus on the input during decoding.
4. **Self-Attention Mechanism:**
   - **firs what is self attention:** it is the core of the Transformer model is the self-attention mechanism. It allows each word in the input sequence to attend to all other words, capturing their relevance and influence, works by seeing how similar and important each word is to all of the words in a sentence, including itself.
   - **Second the Mechanism:**
     - ***Multi-head attention in the encoder block*:** plays a crucial role in capturing different types of information and learning diverse relationships between words. It allows the model to attend to different

parts of the input sequence simultaneously and learn multiple representations of the same input.



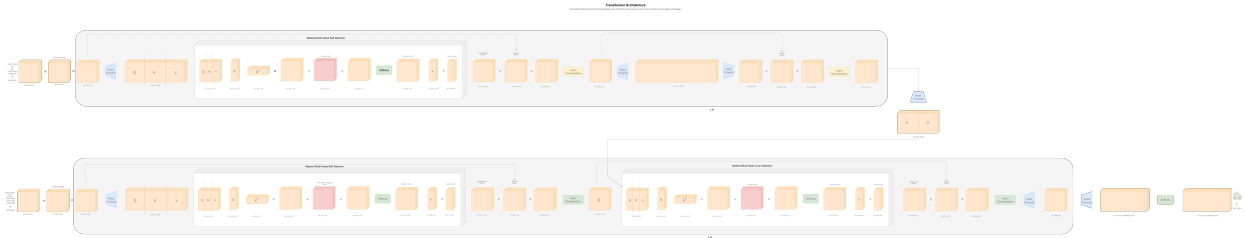
- ***Masked Multi-head attention in the decoder block*:** the same as Multi-head attention in the encoder block but this time for the translation sentence, is used to ensure that during the decoding process, each word can only attend to the words before it. This masking prevents the model from accessing future information, which is crucial for generating the output sequence step by step.
- ***Multi-head attention in the decoder block*:** do the same as the Multi-head attention in the encoder block but between the input sentence and the translation sentence, is employed to capture different relationships between the input sequence and the generated output sequence. It allows the decoder to attend to different parts of the encoder's output and learn multiple representations of the context.

5. **Feed Forward in two blocks:** it is just feed forward neural network but in this paper the neurons are 2048.
6. **Add & Normalization.**

$$Z_{\text{norm}}^{(i)} = Z^{(i)} - \frac{\mu}{\sqrt{\sigma^2 + \epsilon}}$$

Optional using Learnable parameters:

$$Z^{(\tilde{i})} = \gamma Z_{\text{norm}}^{(i)} + \beta$$

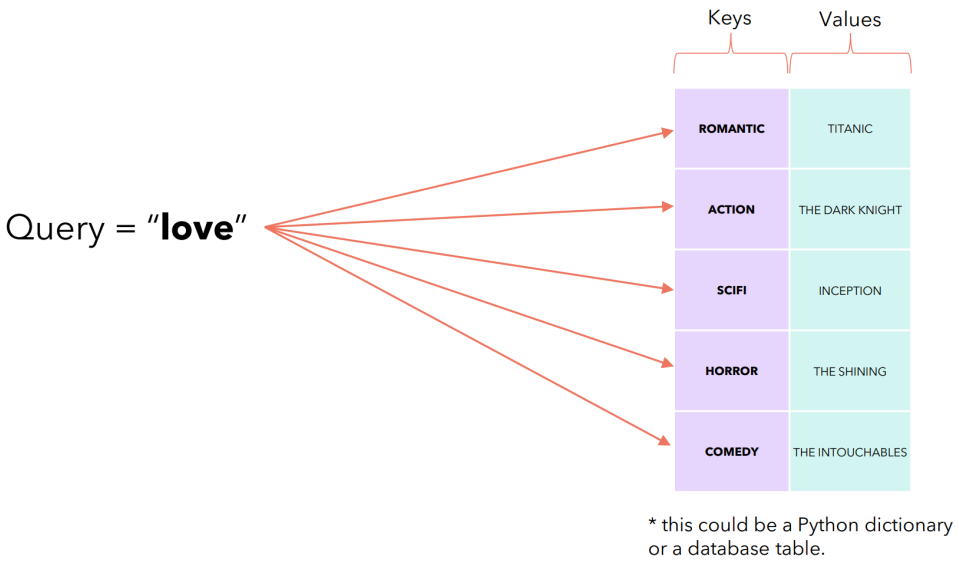


self attention mechanism:

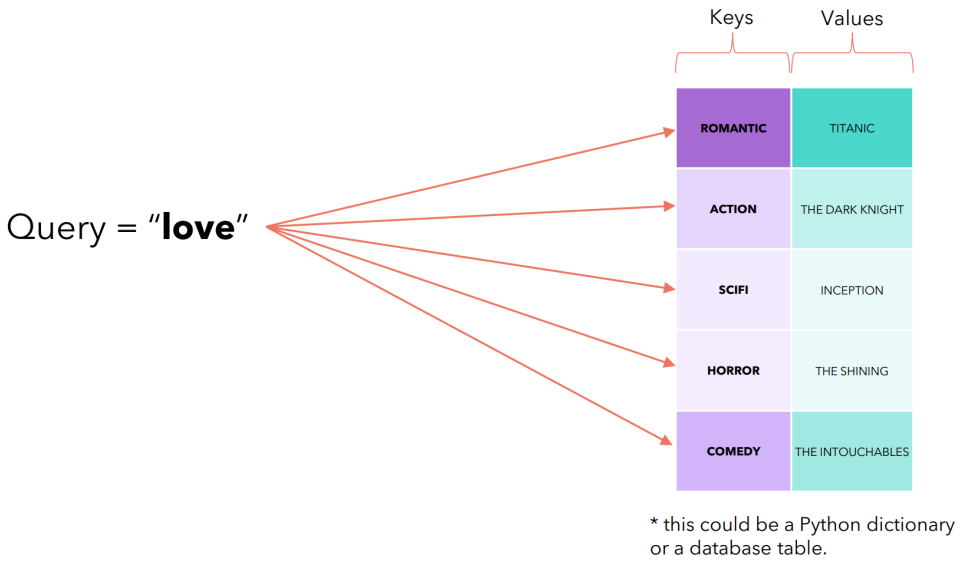
The core of the Transformer model is the self-attention mechanism. It allows each word in the input sequence to attend to all other words, capturing their relevance and influence. Self-attention computes three vectors for each word: Query, Key, and Value.

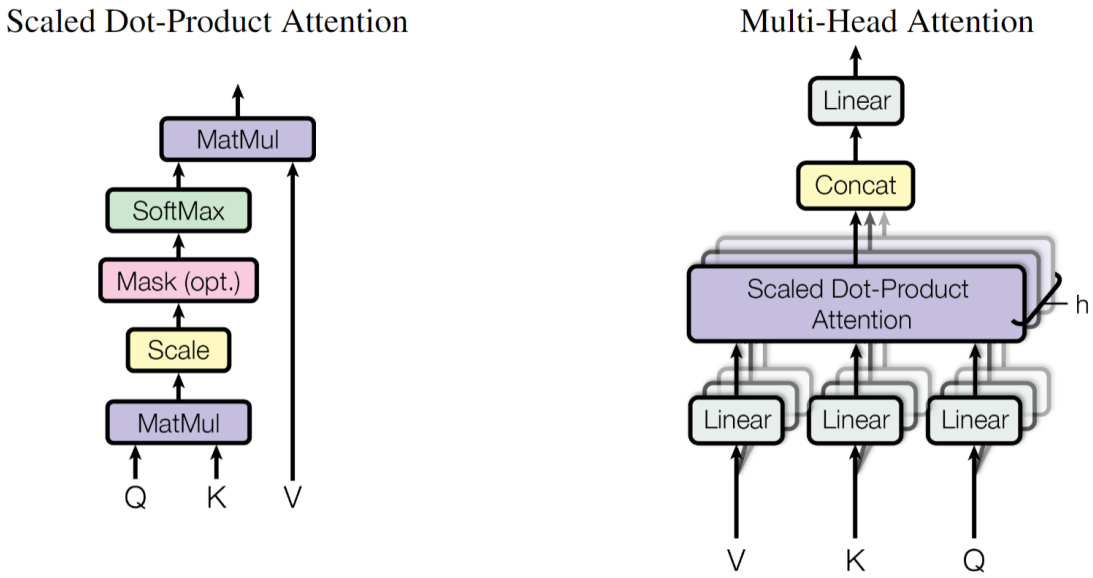
- Query (Q): Each word serves as a query to compute the attention scores.
  - Q: what I am looking for.
- Key (K): Each word acts as a key to determine its relevance to other words.
  - K: what I can offer.
- Value (V): Each word contributes as a value to the attention-weighted sum.
  - what I actually offer.

The Internet says that these terms come from the database terminology or the Python-like dictionaries.



The Internet says that these terms come from the database terminology or the Python-like dictionaries.





Attention vector for every word using this formula:

$$Z = \text{softmax}\left(\frac{QK^T}{\sqrt{\text{Dimension of vector } Q, K \text{ or } V}}\right)V$$

Self-attention is calculated by taking the dot product of the query and key, scaled by a factor, and applying a softmax function to obtain attention weights. These attention weights determine the importance of each word's value for the current word.

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

$$\text{self attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{\text{model}}}} + M\right)$$

Output:

The final layer of the decoder is a linear projection followed by a softmax activation function. It produces a probability distribution over the vocabulary, allowing the model to generate the output word by sampling from this distribution.

Softmax:

The softmax function is a mathematical function that converts a vector of K real numbers into a probability distribution of K possible outcomes. It is a generalization of the logistic function to multiple dimensions, and used in multinomial logistic regression. The softmax function is often used as the last activation function of a neural network to

normalize the output of a network to a probability distribution over predicted output classes. The formula for the standard (unit) softmax function is as follows:

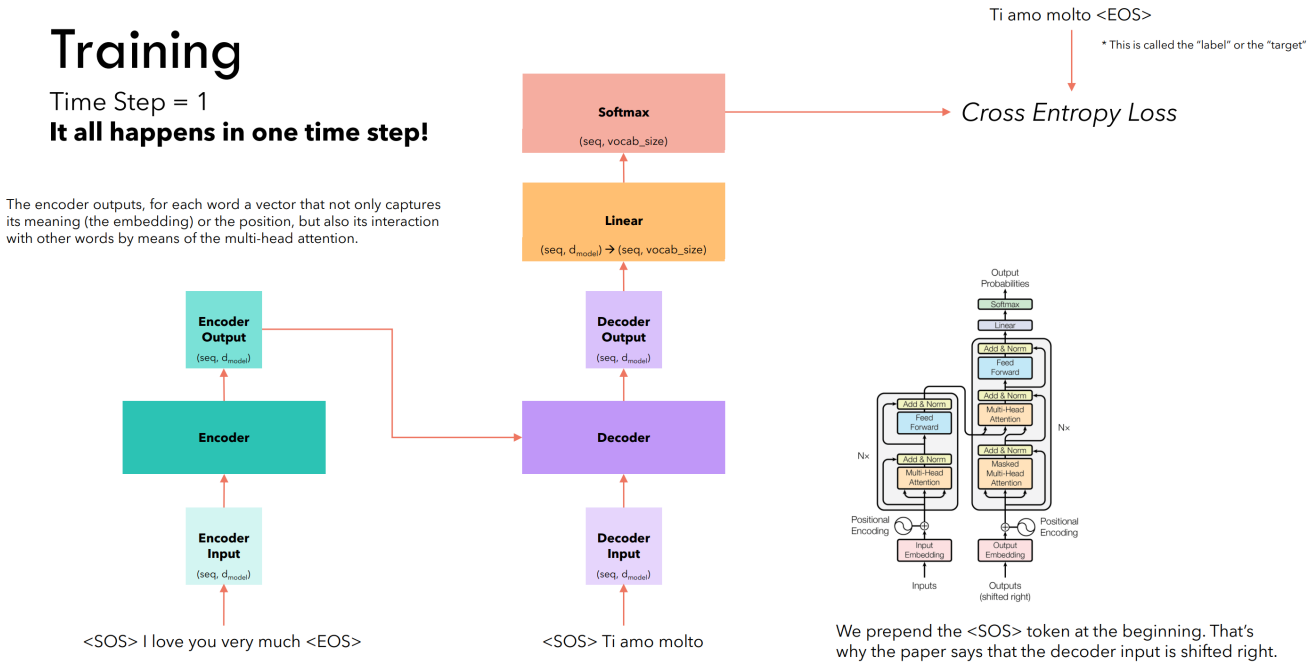
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Linear

convert the embeddings to word again (***it just has weights not biases.***)

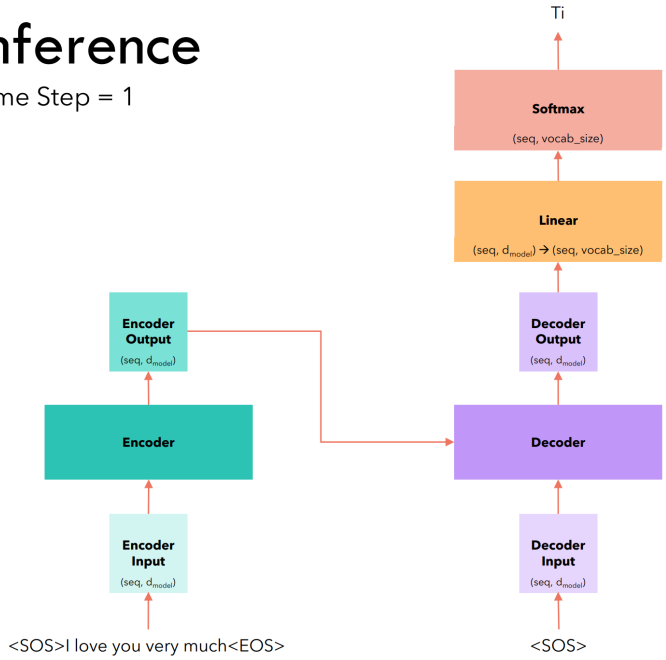
Training:

$$H(P^*|P) = -\sum P^*(i)\log(P(i))$$



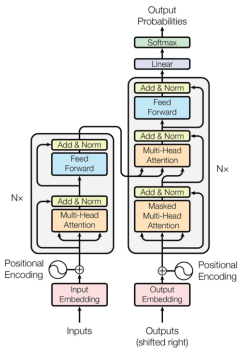
Inference

Time Step = 1



We select a token from the vocabulary corresponding to the position of the token with the maximum value.

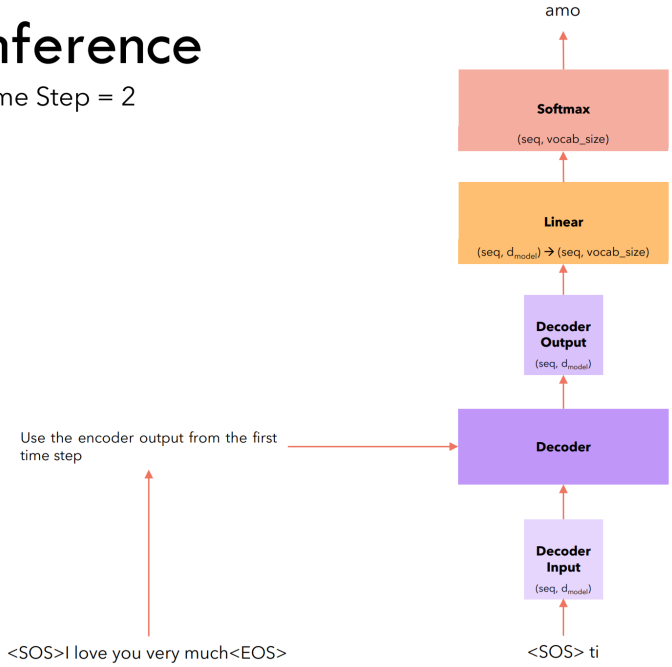
The output of the last layer is commonly known as **logits**



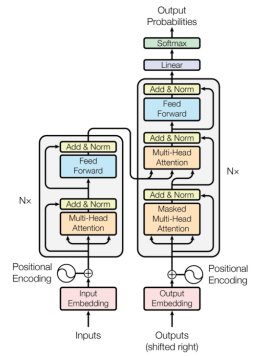
* Both sequences will have same length thanks to padding

Inference

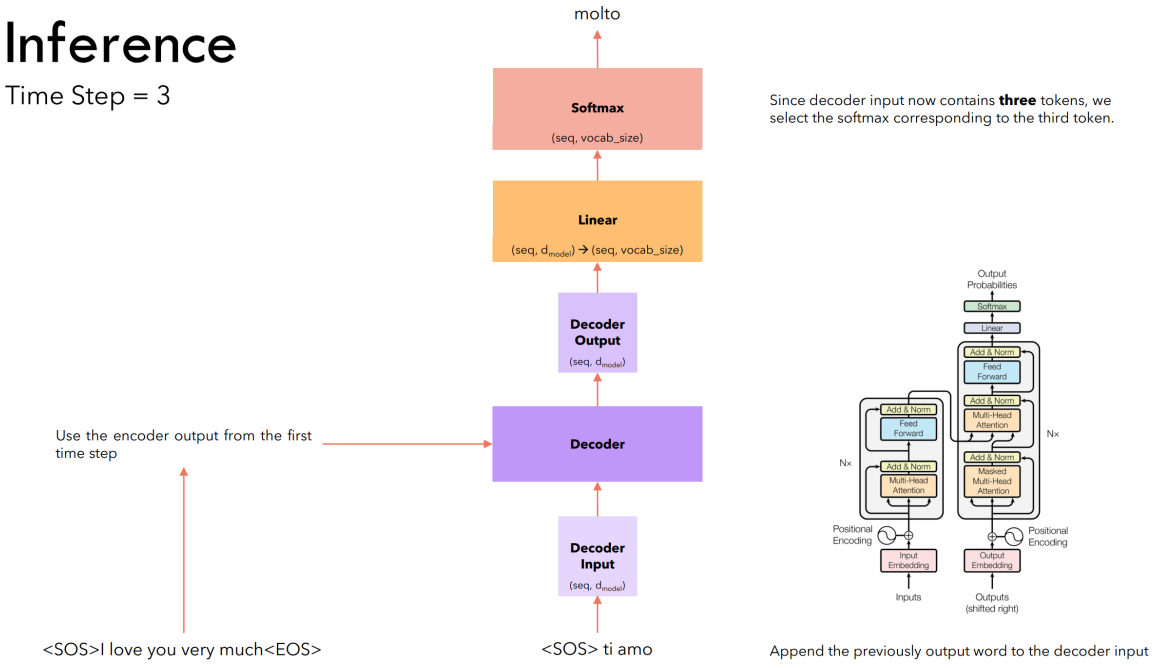
Time Step = 2

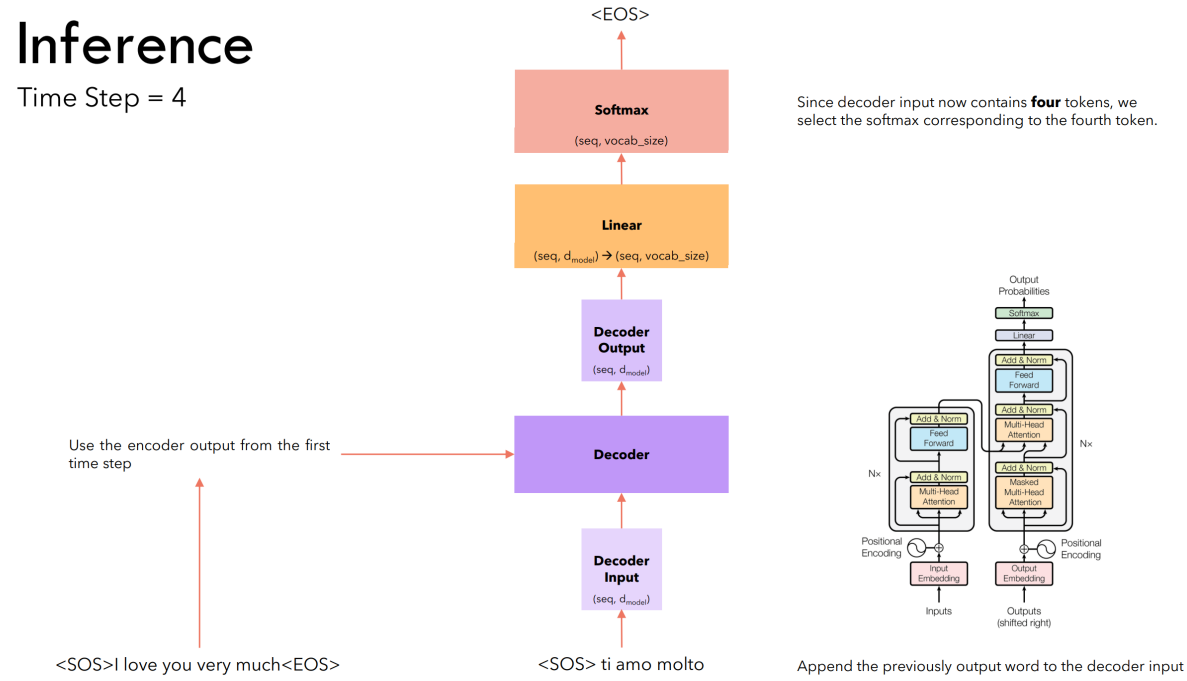


Since decoder input now contains **two** tokens, we select the softmax corresponding to the second token.



Append the previously output word to the decoder input





Project definition:

LITRA stands for: Language to Language Transformer model from the paper "Attention is all you Need", building transformer model:[Transformer model]() from scratch and use it for translation using pytorch.

Problem Statement:

In the rapidly evolving landscape of natural language processing (NLP) and machine translation, there exists a persistent challenge in achieving accurate and contextually rich language-to-language transformations. Existing models often struggle with capturing nuanced semantic meanings, context preservation, and maintaining grammatical coherence across different languages. Additionally, the demand for efficient cross-lingual communication and content generation has underscored the need for a versatile language transformer model that can seamlessly navigate the intricacies of diverse linguistic structures.

Goal:

Develop a specialized language-to-language transformer model that accurately translates from the Arabic language to the English language, ensuring semantic fidelity, contextual awareness, cross-lingual adaptability, and the retention of grammar and style. The model should provide efficient training and inference processes to make it practical and accessible for a wide range of applications, ultimately contributing to the advancement of Arabic-to-English language translation capabilities.

Dataset used:

from hugging Face huggingface/opus_infopankki

Search algorithm used:

Greedy Algorithm for finding which token has the maximum probability.

Loss function used:

cross entropy function.

Optimization algorithms used:

Adam algorithm: Adam (short for Adaptive Moment Estimation) is an optimization algorithm used for training deep neural networks. It combines ideas from two other popular optimization algorithms: RMSprop (Root Mean Square Propagation) and momentum.

Training:

I used my drive to upload the project and then connected it to the Google Collab to train it:

Collab training connected with my drive.

```
from google.colab import drive

drive.mount('/content/drive')

import os

os.chdir('/content/drive/MyDrive/TrainModel')

%run train.py
```

hours of training in 20 epochs: 4 hours

Some Results:

```
SOURCE: العائلات الناطقة بلغة أجنبية لديها الحق في خدمات الترجمة عند اللزوم.
TARGET: A foreign-language family is entitled to interpreting services as necessary.
PREDICTED: in a native language is provided by the services of the services for the elderly .
--------------------------------------------------------------------------------
SOURCE: قمت بارتكاب جرائم وتُعتبر بأنك خطير على النظام أو الأمن العام.
TARGET: you have committed crimes and are considered a danger to public order or safety
PREDICTED: you have committed crimes and are considered a danger to public order or safety
--------------------------------------------------------------------------------
SOURCE: عندما تلتحق بالدراسة، فستحصل على الحق في إنجاز كلتا الدرجتين العلميتين.
TARGET: When you are accepted into an institute of higher education, you receive the right to complete both degrees.
PREDICTED: When you have a of residence , you will receive a higher education degree .
--------------------------------------------------------------------------------
SOURCE: اللجنة لا تتداول حالات التهميش والتمييز المتعلقة بالعمل.
TARGET: The Tribunal does not handle cases of employment-related discrimination.
PREDICTED: The does not have to pay and the work .
```

```
--------------------------------------------------------------------------------
SOURCE: يجب عليك أيضاً أن تستطيع إثبات على سبيل المثال بالوصفة الطبية أو بالتقرير الطبي بأن الغرض من الدواء هو استخدامك أنت الشخصي.
TARGET: In addition, you must be able to prove with a prescription or medical certificate, for example, that the medicine is intended for your personal use.
PREDICTED: You must also have to prove your identity with a friend or friend , for example , that the medicine is intended for your personal use .
--------------------------------------------------------------------------------
SOURCE: إذا كان لديك ترخيص إقامة في فنلندا، ولكن لم تُمنح ترخيص إقامة استمراري، فسوف تصدر دائرة شؤون الهجرة قراراً بالترحيل.
TARGET: If you already have a residence permit in Finland but are not granted a residence permit extension, the Finnish Immigration Service makes a deportation decision.
PREDICTED: If you have a residence permit in but are not granted a residence permit , the Service makes a decision .
```

_Project components:

1. model.py:

The provided Transformer model is a flexible and general-purpose architecture that can be used for various natural language processing tasks by modifying the input and output layers. It offers a powerful mechanism for capturing dependencies in sequential data and has been widely adopted for tasks like machine translation and text generation.

1. InputEmbeddingsLayer:
    - This module represents the input embeddings layer, which maps the input tokens to continuous vector representations.
    - It uses an embedding layer to perform the mapping.
2. PositionalEncodingLayer:
    - This module adds positional information to the input embeddings to capture the order of the tokens in the sequence.
    - It uses sinusoidal functions to encode the position information.
3. NormalizationLayer:

- This module performs layer normalization on the input tensor to stabilize the learning process.
- It normalizes the input tensor by subtracting the mean and dividing by the standard deviation.

4. FeedForwardBlock:
    - This module represents the feed-forward block in the Transformer model.
    - It consists of two linear layers with a ReLU activation function and dropout in between.
5. MultiHeadAttentionBlock:
    - This module implements the multi-head attention mechanism, a key component of the Transformer model.
    - It performs attention computations using linear transformations of the input tensor.
6. ResidualConnection:
    - This module combines the output of a sub-layer with its input using residual connections.
    - It applies dropout and layer normalization to the input tensor before adding it to the sub-layer's output.
7. EncoderBlock:
    - This module represents a single block in the encoder part of the Transformer model.
    - It consists of a self-attention block and a feed-forward block, both with residual connections.
8. Encoder:
    - This module represents the encoder part of the Transformer model, composed of multiple encoder blocks.
    - It applies the encoder blocks sequentially to the input sequence.
9. DecoderBlock:
    - This module represents a single block in the decoder part of the Transformer model.
    - It consists of a self-attention block, a cross-attention block, and a feed-forward block, all with residual connections.
10. Decoder:
    - This module represents the decoder part of the Transformer model, composed of multiple decoder blocks.
    - It applies the decoder blocks sequentially to the target sequence, using the encoder output for cross-attention.
11. LinearLayer:

- This module performs a linear transformation of the decoder output to obtain the final logits for each target token.

12. TransformerBlock:
    - This module combines the encoder, decoder, and linear layers to form a complete Transformer model.
    - It provides methods for encoding, decoding, and obtaining the linear output.
13. TransformerModel:
    - This function creates an instance of the TransformerBlock with the specified dimensions and hyperparameters.
    - It initializes the model's parameters and returns the TransformerBlock object.

2. dataset.py:

The BilingualDataset class provides a convenient way to process and prepare bilingual data for training the Transformer model. It handles tokenization, padding, and masking of the input sequences, and provides the necessary tensors and masks for training.

1. Initialization:
    - The constructor takes several parameters, including the dataset, source and target tokenizers, source and target languages, and sequence length.
    - It initializes the dataset and other necessary attributes.
2. Length:
    - The **len** method returns the length of the dataset, which is the number of samples in the dataset.
3. Get Item:
    - The **getitem** method retrieves a specific sample from the dataset at the given index.
    - It retrieves the source and target texts from the dataset dictionary.
    - The source and target texts are tokenized using the source and target tokenizers, respectively.
4. Padding:
    - The method calculates the padding required for the source and target sequences to match the specified sequence length.
    - It ensures that the total length of the encoded sequences is equal to the sequence length.
5. Encoding:
    - The method constructs the encoder input, decoder input, and target tensors.
    - It concatenates the special tokens (SOS, EOS, PAD) with the encoded source and target sequences.

    - The encoder input includes the SOS token, source tokens, EOS token, and padding tokens.
    - The decoder input includes the SOS token, target tokens, and padding tokens.
    - The target tensor includes the target tokens, padding tokens, and EOS token.
6. Masking:
    - The method creates masks for the encoder and decoder inputs.
    - The encoder input mask marks the positions of the padding tokens as 0 and the rest as 1.
    - The decoder input mask marks the positions of the padding tokens and future tokens (after the current position) as 0 and the rest as 1.
    - The casual_mask function is used to create the decoder input mask.
7. Return:
    - The method returns a dictionary containing various elements:
        - "encoder_input": The tensor representing the encoder input sequence.
        - "decoder_input": The tensor representing the decoder input sequence.
        - "encoder_input_mask": The mask for the encoder input.
        - "decoder_input_mask": The mask for the decoder input.
        - "Target": The tensor representing the target sequence.
        - "source_text": The original source text.
        - "target_text": The original target text.

3. train.py:

The provided `train.py` file is responsible for training a model by building tokenizers and feeding it with a dataset. Overall, this script builds tokenizers for the source and target languages using the dataset, creates a transformer model, trains the model using the dataset, and saves the model's weights after each epoch.

Let's go through the different parts of the file to understand its functionality:

1. Importing libraries: The script starts by importing the necessary libraries and modules, including PyTorch components, dataset-related modules, tokenizers, and other utility modules.
2. Utility functions (commented out): The file contains two utility functions, `greedy_search` and `run_validation`, which are currently commented out. These functions are likely used for generating predictions and evaluating the model's performance during validation.

3. Get_All_Sentences: This function retrieves sentences from the dataset for a given language. It utilizes a generator to yield the sentences.
4. Build_Tokenizer: This function builds a tokenizer for a specific language using the dataset. It first checks if the tokenizer file exists for the given language. If not, it creates a new tokenizer using the WordLevel model and trains it on the sentences from the dataset. The trained tokenizer is then saved to the tokenizer file. If the tokenizer file already exists, it loads the tokenizer from the file.
5. Get_dataset: This function loads the dataset using the `load_dataset` function from the `datasets` module. It then calls the `Build_Tokenizer` function to build the tokenizers for the source and target languages. The function splits the dataset into training and validation sets and creates instances of the `BilingualDataset` class for both sets, passing the tokenizers and other configuration parameters. It also determines the maximum source and target sequence lengths in the dataset. Finally, it creates data loaders for the training and validation datasets.
6. Get_model: This function creates an instance of the `TransformerModel` class, which represents the model to be trained. It takes the source and target vocabulary sizes, sequence lengths, and the dimensionality of the model as input.
7. train_model: This is the main training function. It checks the availability of a CUDA-enabled GPU and sets the device accordingly. It creates the necessary directories and loads the dataset and model. It also initializes the optimizer, loss function, and other variables for training.
8. Training loop: The script enters a loop that iterates over the specified number of epochs. Inside the epoch loop, it iterates over the batches of the training data. For each batch, it performs the following steps:
    - Moves the input tensors to the device (GPU).
    - Passes the input through the model's encoder and decoder.
    - Computes the loss between the predicted output and the target output.
    - Backpropagates the gradients and updates the model's parameters.
    - Logs the training loss and updates the global step counter.
9. Saving the model: After each epoch, the script saves the model's weights, optimizer state, epoch number, and global step to a file using the `torch.save` function.
10. Main section: The script includes a main section that loads the configuration settings, ignores warnings, and calls the `train_model` function with the configuration.

4. Configuration.py:

this is the settings of the model, You can customize the source and target languages, sequence lengths for each, the number of epochs, batch size, and more.

```python
def Get_configuration():
    return {
        "batch_size": 8,
        "num_epochs": 30,
        "lr": 10**-4,
        "sequence_length": 100,
        "d_model": 512,
        "datasource": 'opus_infopankki',
        "source_language": "ar",
        "target_language": "en",
        "model_folder": "weights",
        "model_basename": "tmodel_",
        "preload": "latest",
        "tokenizer_file": "tokenizer_{0}.json",
        "experiment_name": "runs/tmodel"
    }
```

Notes and Terms in the research:

1. Input matrix (sequence, dmodel)
2. 1. Embedding words.
3. positional Encoding.
4. self attention.
5. Query, key values.
6. sequence to sequence(seq2seq).
7. Recurrent neural network.
8. Convolutional neural network.
9. LTM.
10. GNN.
11. Encoder.
12. Decoder.
13. Multi head attention.
14. auto regression.
15. seq2seq(Translation).
16. seq2vec(sentence classify).
17. Natural language processing.
18. Stable Diffusion.
19. Translator.

20. N: number of layers.
21. dmodel: length of word.
22. dff: number of neurons in ffn.
23. h: multi-head attention .
24. dk: length of Keys, quers, vlaues.
25. Pdrop: probability of droping one block .
26. BLEU: مقیاس کلما زاد کان افضل
27. PPL: مقیاس کلما زاد کان اخس.

---

links :

1. coding the transformers from scratch
2. Explain the transformers
3. explain the transformers and beyond the mathematical things
4. Explain the transformer, the paper itself
5. my pre trained model
6. my model in drive