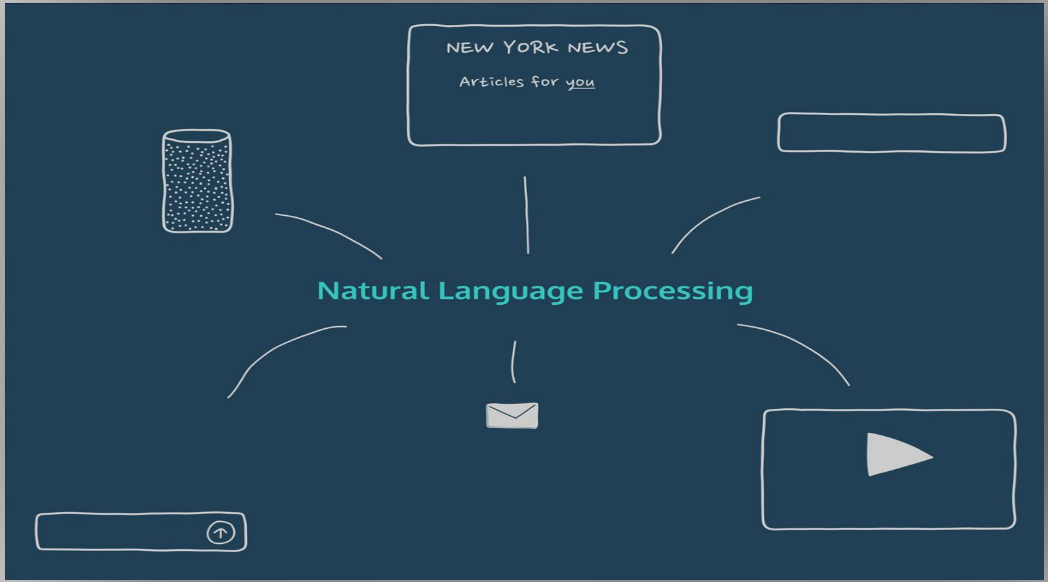
# Neural Machine Translation using Transformers



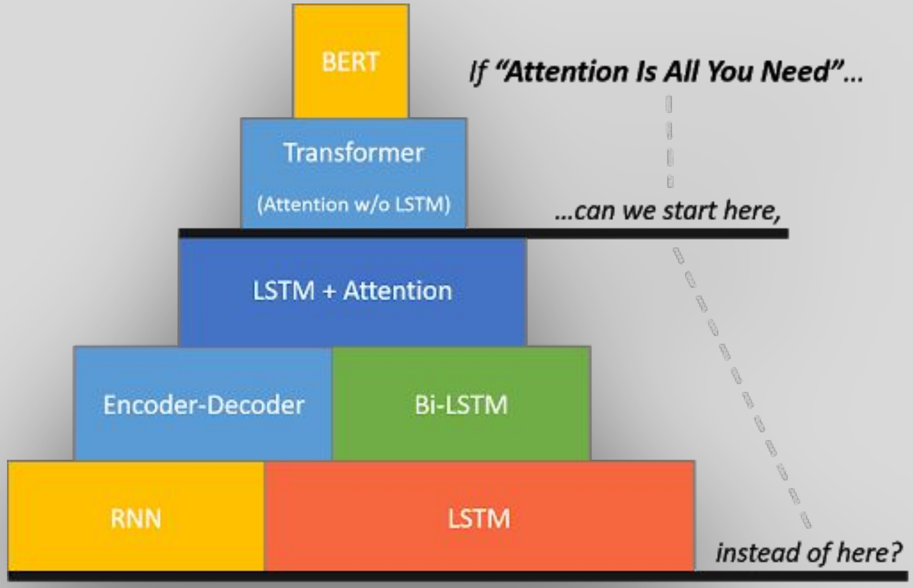

Group 4 : Apurva Shekhar , Harsh Tandon, Jaskaran Singh Kohli
Ritu Ranjani Ravi Shankar , Suchita Negi

# TABLE OF CONTENTS

# 01

# Introduction

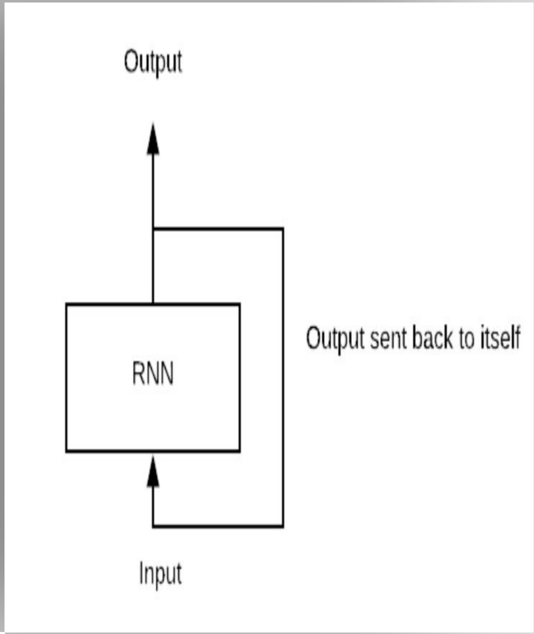
Attention is all we need

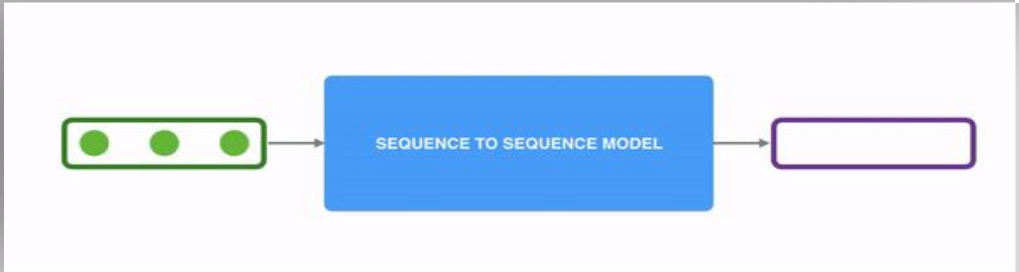


# Traditional RNNs

- Origin of Transformer Model - Traditional RNN

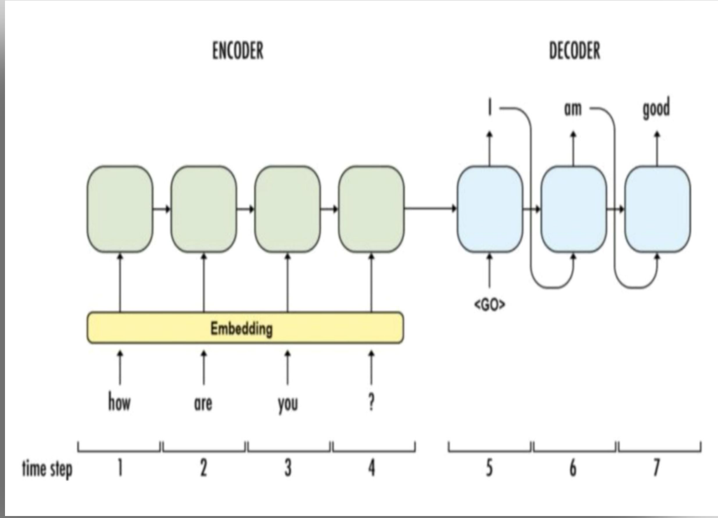
- Type of RNN Model:
- Vector-Sequence Model
- Sequence- Vector Model
- **Sequence-Sequence Model (Language Translator)**



# Sequence- Sequence Model

- Sequential Processing
- Dependency on previous state
- **Weakness:**
    - Slow to train
    - Loss of information in long sentence
- ***Long Short- Term Memory***
    - Helped with information loss
    - Slower than RNN
- How can we use parallelization for sequential data?



# Attention Is All You Need

- Transformer NN architecture was introduced by Google in 2017 in their paper ‘*Attention is all you need*’.
- Transformers try to solve the problem of parallelization by using a concept called ‘**Attention**’
- How Transformer is better than RNN -
    - No sequential input -Input is one whole sentence at a time
    - Faster to train

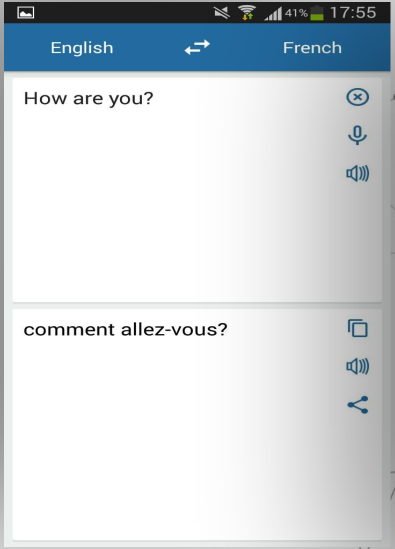
# 02

# Problem Statement

Language Translation

# Problem Statement

- Implement Google Language Translation using Transformer Neural Networks/Attention.
- We would be performing language translation from English to French for the ease of depiction.

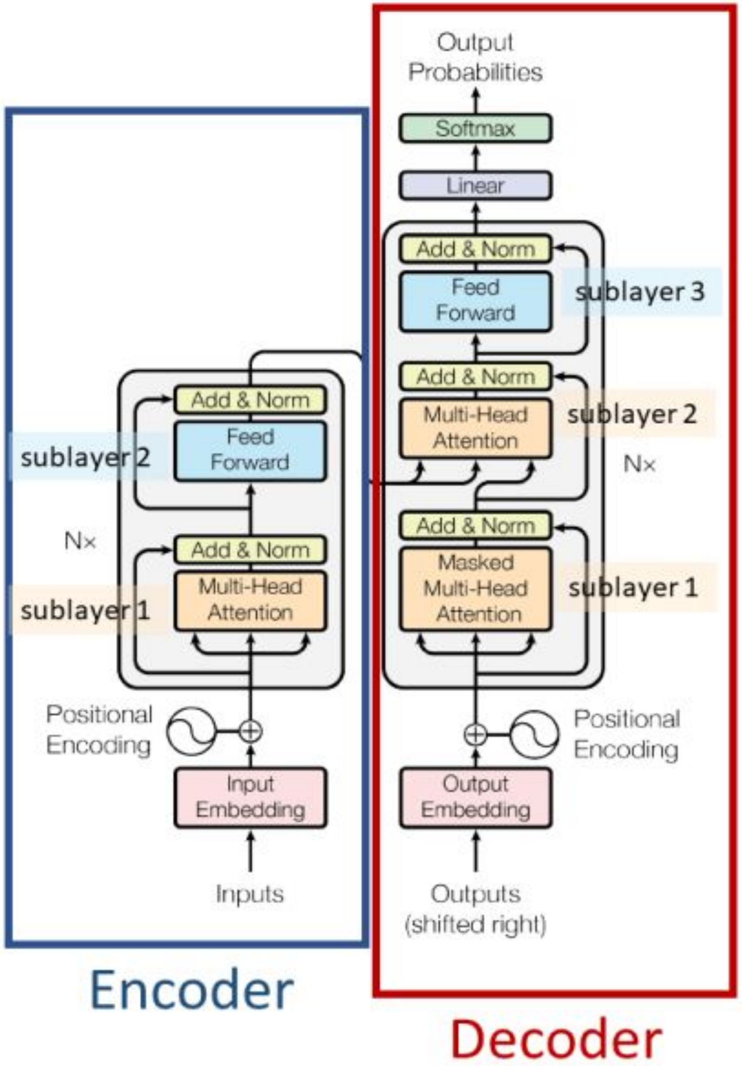


# 03

# Architecture Overview

# Overview

- Transformer architecture consists of:
  - Encoder
  - Decoder

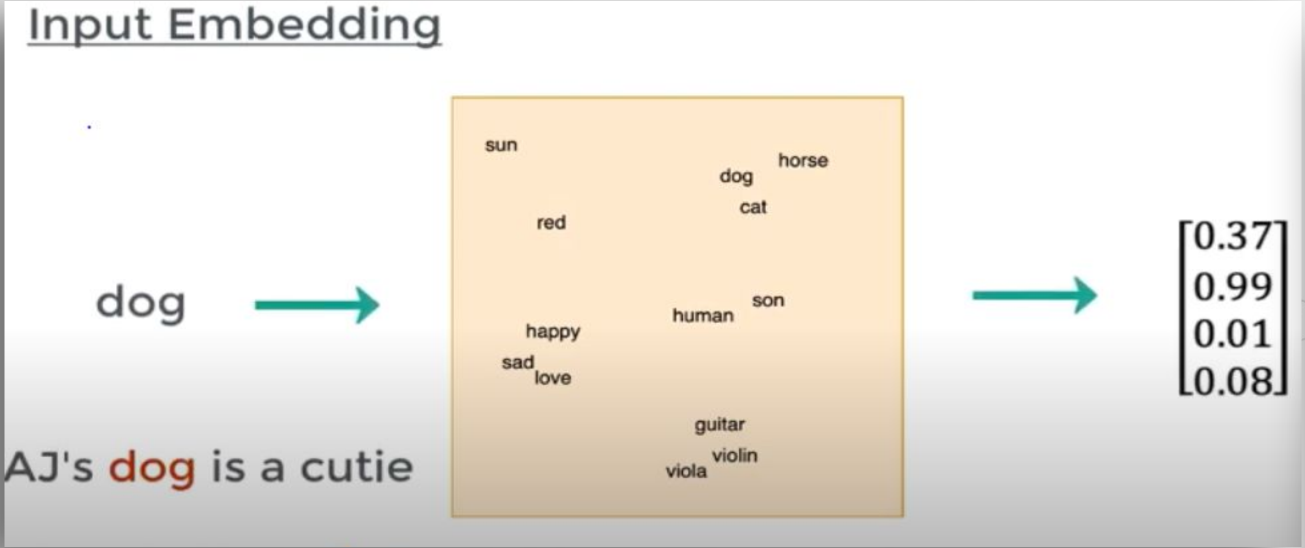


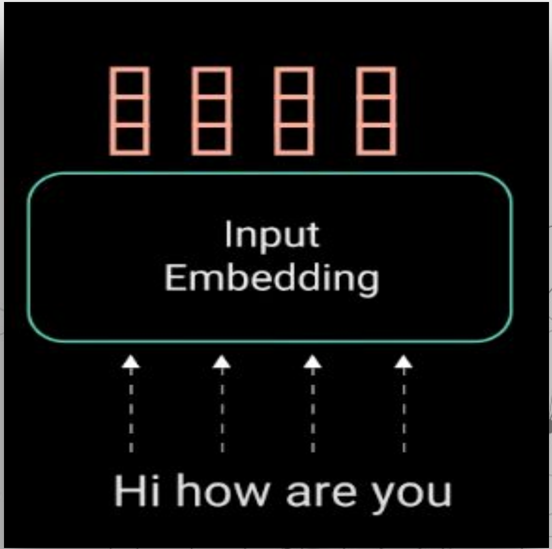
# 04

# Architecture In Depth

# Input Embedding:

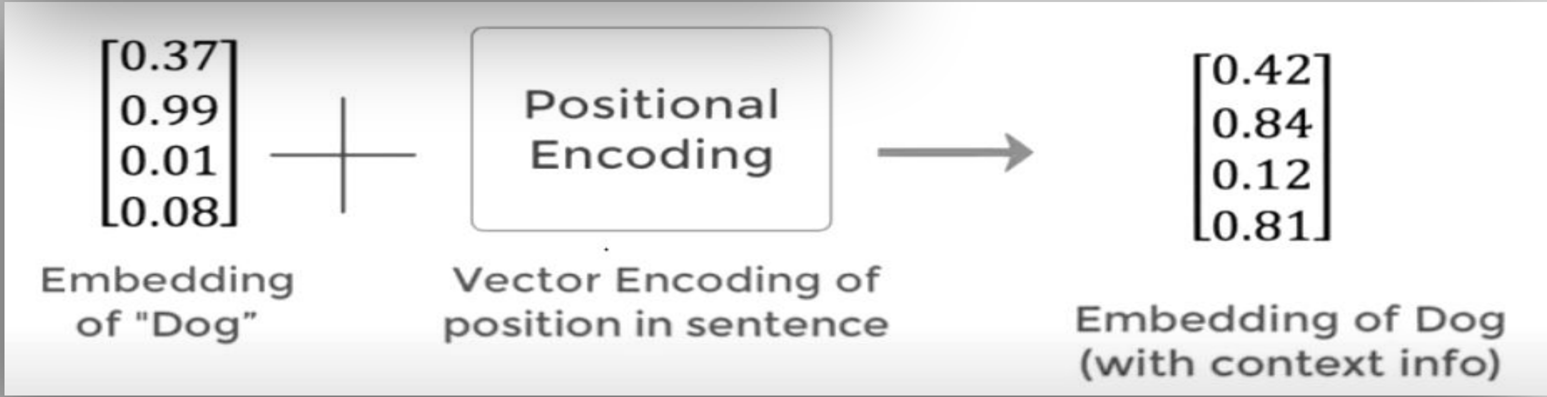
- Computers don’t understand words
    - Solution: Map words to vectors
- Words occurring nearby in text are closer in vector space.



# Positional Encoding

- Same words have different context in different sentence.
- Positional encoders gives context to words in a sentence.

# Positional Encoding

Positional encoding formulae:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/dmodel})$$

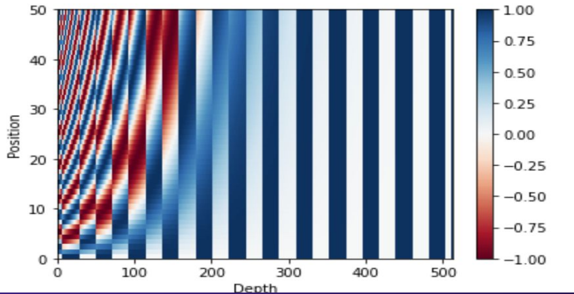
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/dmodel})$$

```
def get_angles(pos, i, d_model):
  angle_rates = 1 / np.power(10000, (2 * (i//2)) / np.float32(d_model))
  return pos * angle_rates
```

```
def positional_encoding(position, d_model):
  angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                          np.arange(d_model)[np.newaxis, :],
                          d_model)

  # apply sin to even indices in the array; 2i
  angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

  # apply cos to odd indices in the array; 2i+1
  angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

  pos_encoding = angle_rads[np.newaxis, ...]

  return tf.cast(pos_encoding, dtype=tf.float32)
```

```
pos_encoding = positional_encoding(50, 512)
print (pos_encoding.shape)

plt.pcolormesh(pos_encoding[0], cmap='RdBu')
plt.xlabel('Depth')
plt.xlim((0, 512))
plt.ylabel('Position')
plt.colorbar()
plt.show()
```

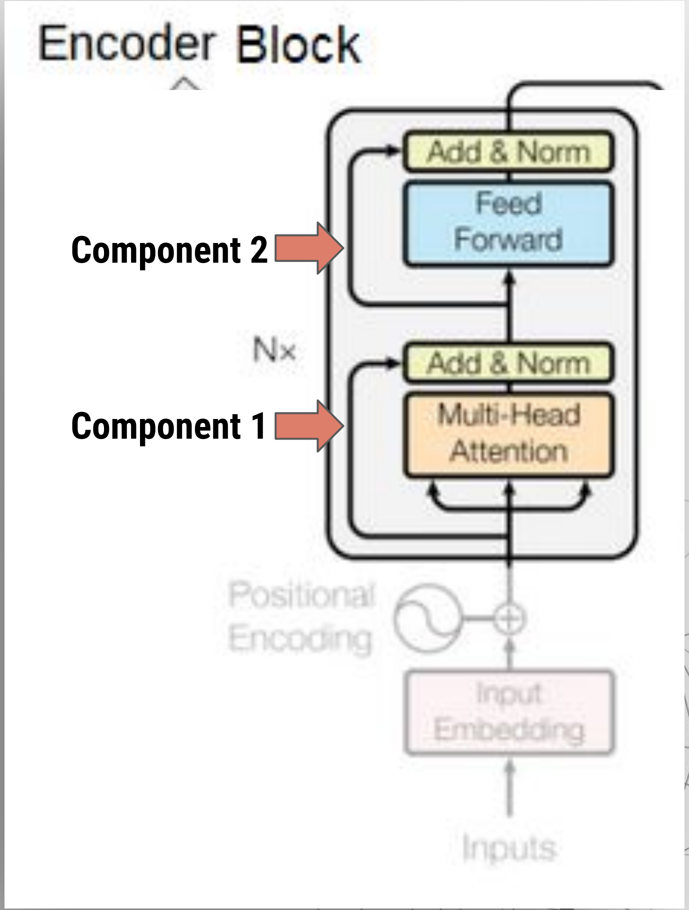
```
(1, 50, 512)
```



# Encoder Block

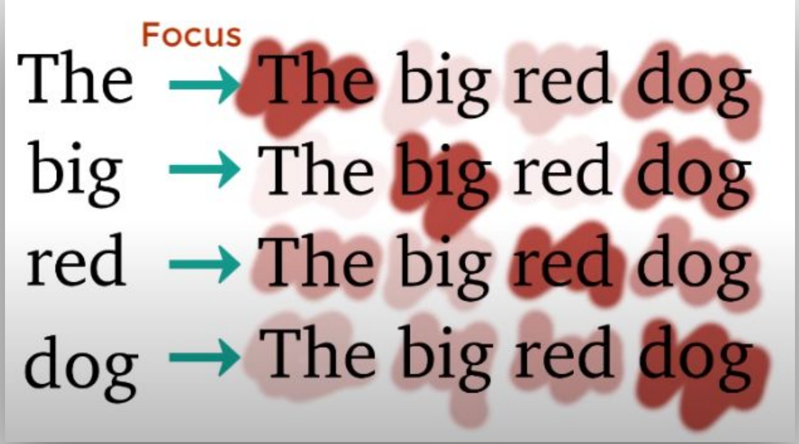
Encoder block consists of:

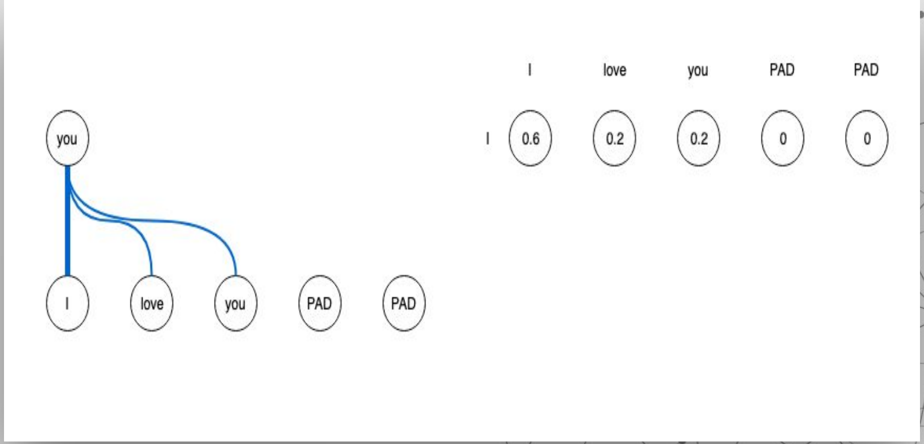
- Attention layer and
- Feed Forward Network.



# Component 1: Attention Layer

- Answers which part of the input should we focus on.
- For every word, generate 'attention vector' capturing the contextual relationship between words in a sentence.



# Attention

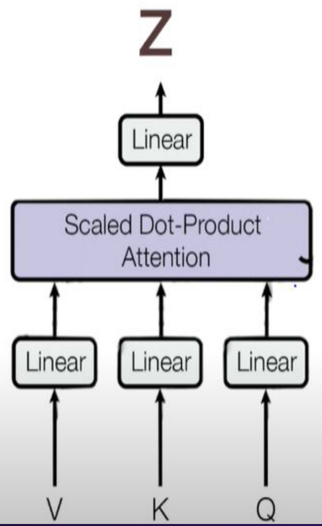
$$Attention(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

```
def scaled_dot_product_attention(queries, keys, values, mask):
    product = tf.matmul(queries, keys, transpose_b=True)

    keys_dim = tf.cast(tf.shape(keys)[-1], tf.float32)
    scaled_product = product / tf.math.sqrt(keys_dim)

    if mask is not None:
        scaled_product += (mask * -1e9)

    attention_weights = tf.nn.softmax(scaled_product, axis=-1)

    attention = tf.matmul(attention_weights, values)

    return attention, attention_weights
```

```
temp_k = tf.constant([[10,0,0],
                      [0,10,0],
                      [0,0,10],
                      [0,0,10]], dtype=tf.float32)  # (4, 3)

temp_v = tf.constant([[   1,0],
                      [  10,0],
                      [ 100,5],
                      [1000,6]], dtype=tf.float32)  # (4, 2)

# This `query` aligns with the second `key`,
# so the second `value` is returned.
temp_q = tf.constant([[0, 10, 0]], dtype=tf.float32)  # (1, 3)
print_out(temp_q, temp_k, temp_v)
```

```
Attention weights are:
tf.Tensor([[0. 1. 0. 0.]], shape=(1, 4), dtype=float32)
Output is:
tf.Tensor([[10.  0.]], shape=(1, 2), dtype=float32)
```

# Attention contd..

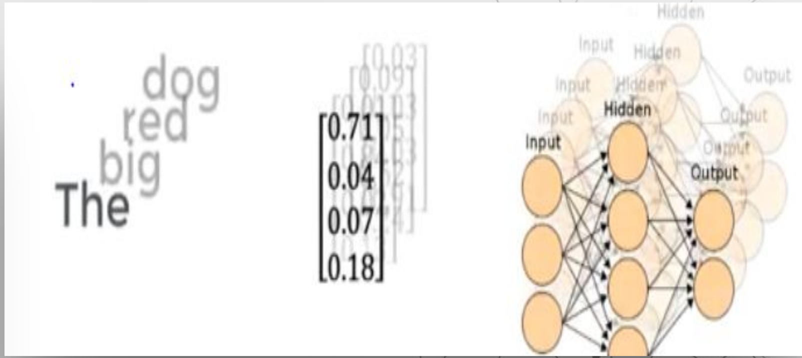



$$Z = softmax\left(\frac{Q.K^T}{\sqrt{Dimension\ of\ vector\ Q,K\ or\ V}}\right).V$$

# Component 2 : Feed Forward Layer

- Each attention vector is fed to Feed Forward Net.
- Feed Forward Net converts the attention vectors into an understandable form for next stages.

Wait..Weren't FF nets slow?



# Encoder Block - Code

```python
class EncoderLayer(layers.Layer):

    def __init__(self, d_model, FFN_units, nb_proj, dropout_rate):
        super(EncoderLayer, self).__init__()
        self.d_model = d_model
        self.FFN_units = FFN_units
        self.nb_proj = nb_proj
        self.dropout_rate = dropout_rate

    #def build(self, input_shape):
        #self.d_model = input_shape[-1]

        self.multi_head_attention = MultiHeadAttention(self.d_model, self.nb_proj)
        self.ffn = point_wise_feed_forward_network(self.d_model, self.FFN_units)

        self.norm_1 = layers.LayerNormalization(epsilon=1e-6)
        self.norm_2 = layers.LayerNormalization(epsilon=1e-6)

        self.dropout_1 = layers.Dropout(rate=self.dropout_rate)
        self.dropout_2 = layers.Dropout(rate=self.dropout_rate)

        #self.dense_1 = layers.Dense(units=self.FFN_units, activation="relu")
        #self.dense_2 = layers.Dense(units=self.d_model)

    def call(self, inputs, training, mask):
        attention, _ = self.multi_head_attention(inputs,
                                                 inputs,
                                                 inputs,
                                                 mask)
        attention = self.dropout_1(attention, training=training)
        out1 = self.norm_1(inputs + attention)

        ffn_output = self.ffn(out1)  # (batch_size, input_seq_len, d_model)
        ffn_output = self.dropout_2(ffn_output, training=training)
        out2 = self.norm_2(out1 + ffn_output)
        #outputs = self.dense_1(attention)
        #outputs = self.dense_2(outputs)
        #outputs = self.dropout_2(outputs, training=training)
        #outputs = self.norm_2(outputs + attention)

        return out2
```

```python
def point_wise_feed_forward_network(d_model, dff):
  return tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),  # (batch_size, seq_len, dff)
      tf.keras.layers.Dense(d_model)  # (batch_size, seq_len, d_model)
  ])
```

```python
sample_encoder_layer = EncoderLayer(d_model=512, nb_proj=8, FFN_units=2048, dropout_rate=0.1)

sample_encoder_layer_output = sample_encoder_layer(
    tf.random.uniform((64, 43, 512)), False, None)

sample_encoder_layer_output.shape  # (batch_size, input_seq_len, d_model)
```

```
TensorShape([64, 43, 512])
```

# Decoder Block

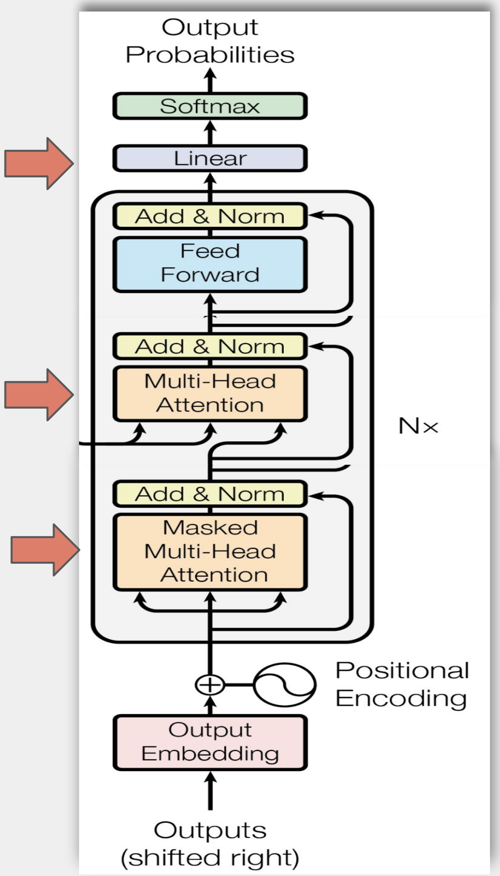
Decoder Block consists of:

- Self Attention Layers
- Encoder-Decoder Attention Layer
- Linear Layer and Softmax Layer

**Component 3**

**Component 2**

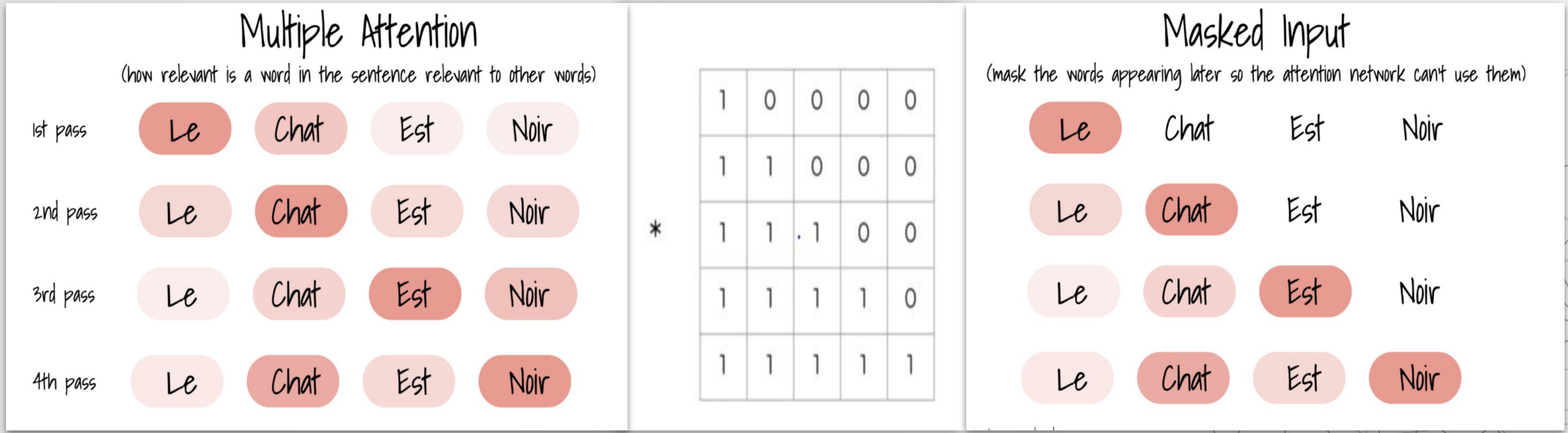
**Component 1**




- To train decoder, we feed French sentences to the decoder.
- Remember, computer don’t understand words?
    - Use Embedding and Positional Encoding

# Component 1: Masked Attention

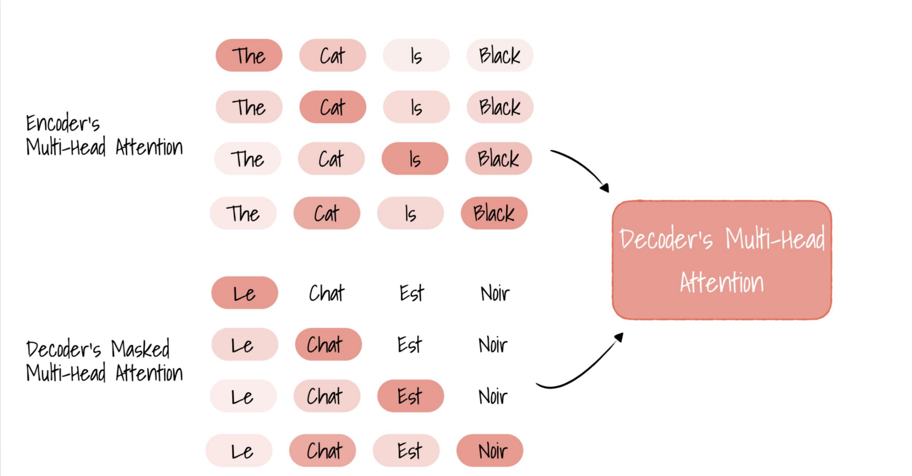
- Generates Attention Vectors
- Masking forbids the decoder to have access to the future words.



# Component 2: Encoder Decoder Attention

- Inputs: Attention Vectors from
  - Encoder’s multi-head Attention
  - Decoder’s Masked Multi-Head Attention.
- Output: Attention vectors for every word in English and French sentences

This is where the main English to French word mapping happens!

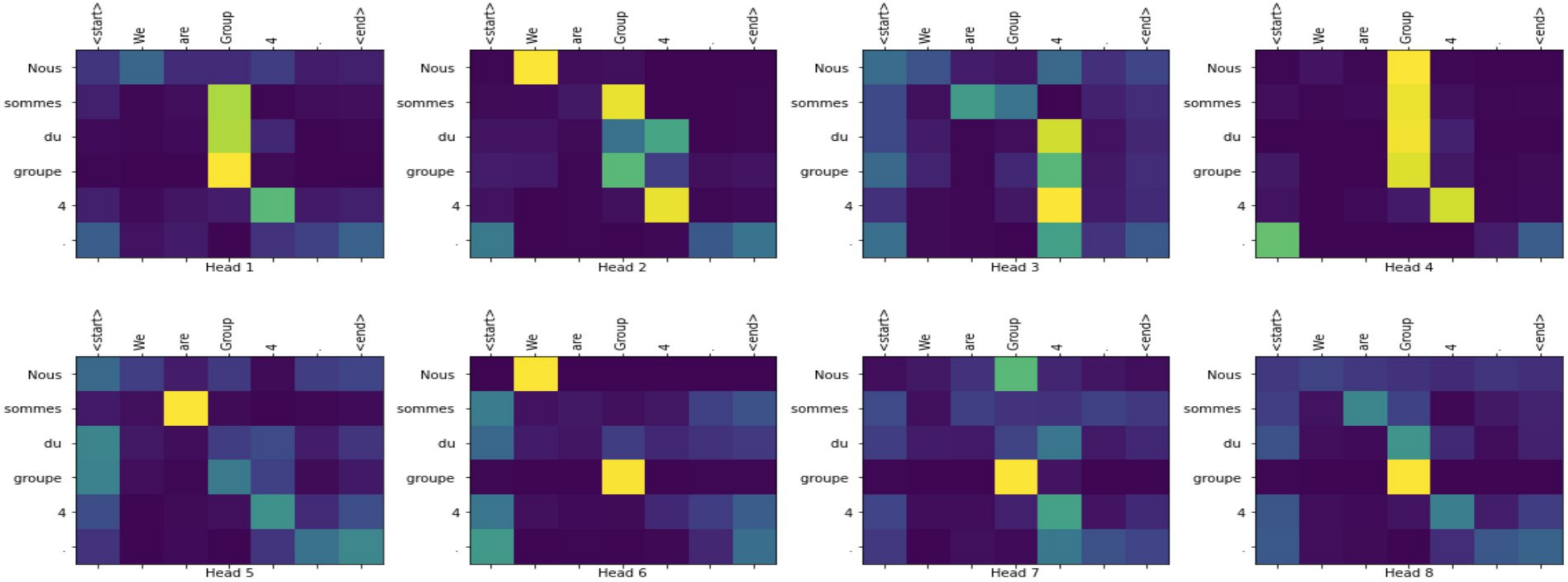



Input Attention vectors into the Decoder’s Multi-Head Attention

# Encoder - Decoder Attention

```
translate("We are Group 4.", plot='decoder_layer    ck2')
```

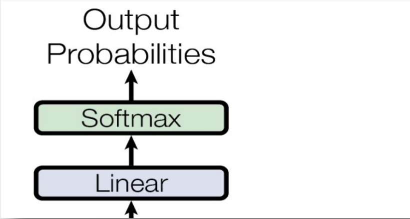
```
Input: We are Group 4.
Predicted translation: Nous sommes du groupe 4.
```

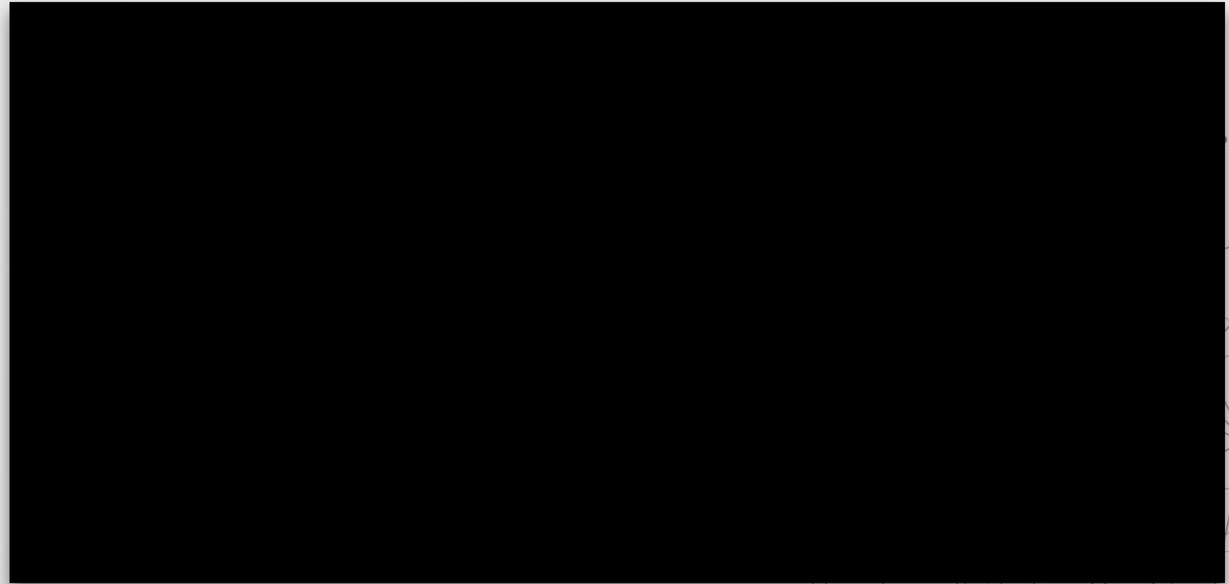


# Component 3: Linear and Softmax Layer

To convert the final decoder vector into words we have linear layer and SoftMax layer

- Linear layer generates logit for SoftMax
- SoftMax gives probability to all vocabularies.

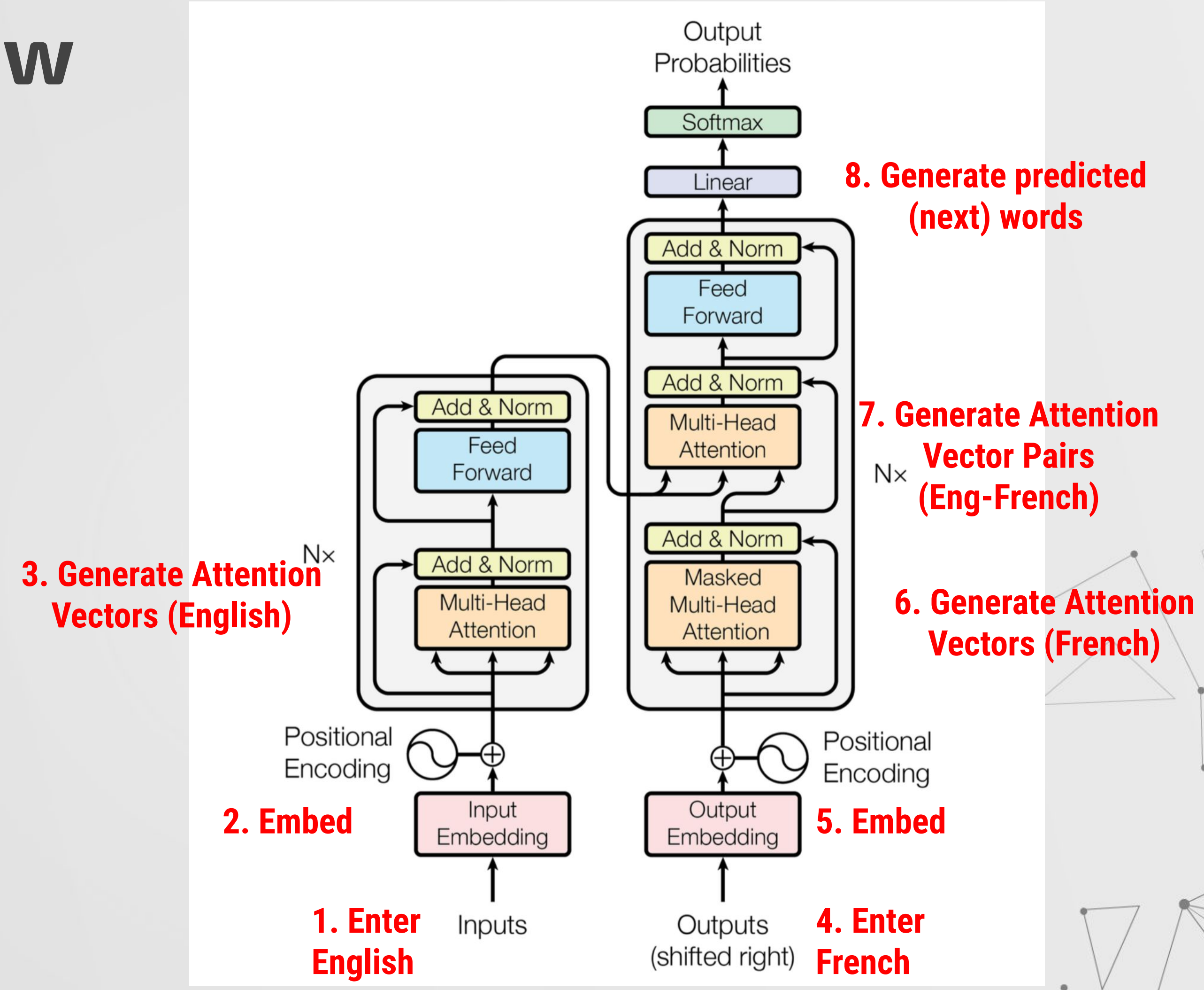


# 05

# Bringing it together

# Review



Output Probabilities
Softmax
Linear
8. Generate predicted (next) words
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
7. Generate Attention Vector Pairs (Eng-French)
Nx
Add & Norm
Masked Multi-Head Attention
6. Generate Attention Vectors (French)
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
3. Generate Attention Vectors (English)
Positional Encoding
Positional Encoding
2. Embed
Input Embedding
Output Embedding
5. Embed
1. Enter English
Inputs
Outputs (shifted right)
4. Enter French

# DEMO

# Output - Translation

```
translate("Do you speak French ?")

Input: Do you speak French ?
Predicted translation: Vous parlez de français?
```

```
translate("No, I don't speak French,I speak English.")

Input: No, I don't speak French,I speak English.
Predicted translation: Non, je ne parle pas de français, je parle anglais.
```

```
translate("I would like to reserve a flight for US from Paris.")

Input: I would like to reserve a flight for US from Paris.
Predicted translation: Je voudrais réserver un vol pour les États-Unis de Paris.
```

```
translate("The departure is at 12:00 hours.")

Input: The departure is at 12:00 hours.
Predicted translation: Le départ est à 12 heures.
```

```
translate("It's good.")

Input: It's good.
Predicted translation: C'est bien le bien.
```

```
translate("Your reservation for US is confirmed.")

Input: Your reservation for US is confirmed
Predicted translation: Votre réserve pour les États-Unis est confirmée
```

```
translate("Ok.")

Input: Ok.
Predicted translation: D'accord.
```

```
translate("Please give me five Stars.")

Input: Please give me five Stars.
Predicted translation: Je vous prie de donner cinq Staes.
```

```
translate("HAHAHAHAHAHAHAHAHA")

Input: HAHAHAHAHAHAHAHAHA
Predicted translation: HUUUUUUUUHAHAHAHAUUU
```

Merci

# Credits and References


- https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf
- https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html
- https://www.tensorflow.org/tutorials/text/transformer#setup_input_pipeline
- https://www.youtube.com/watch?v=TQQlZhbC5ps
- https://towardsdatascience.com/illustrated-guide-to-transformer-cf6969ffa067