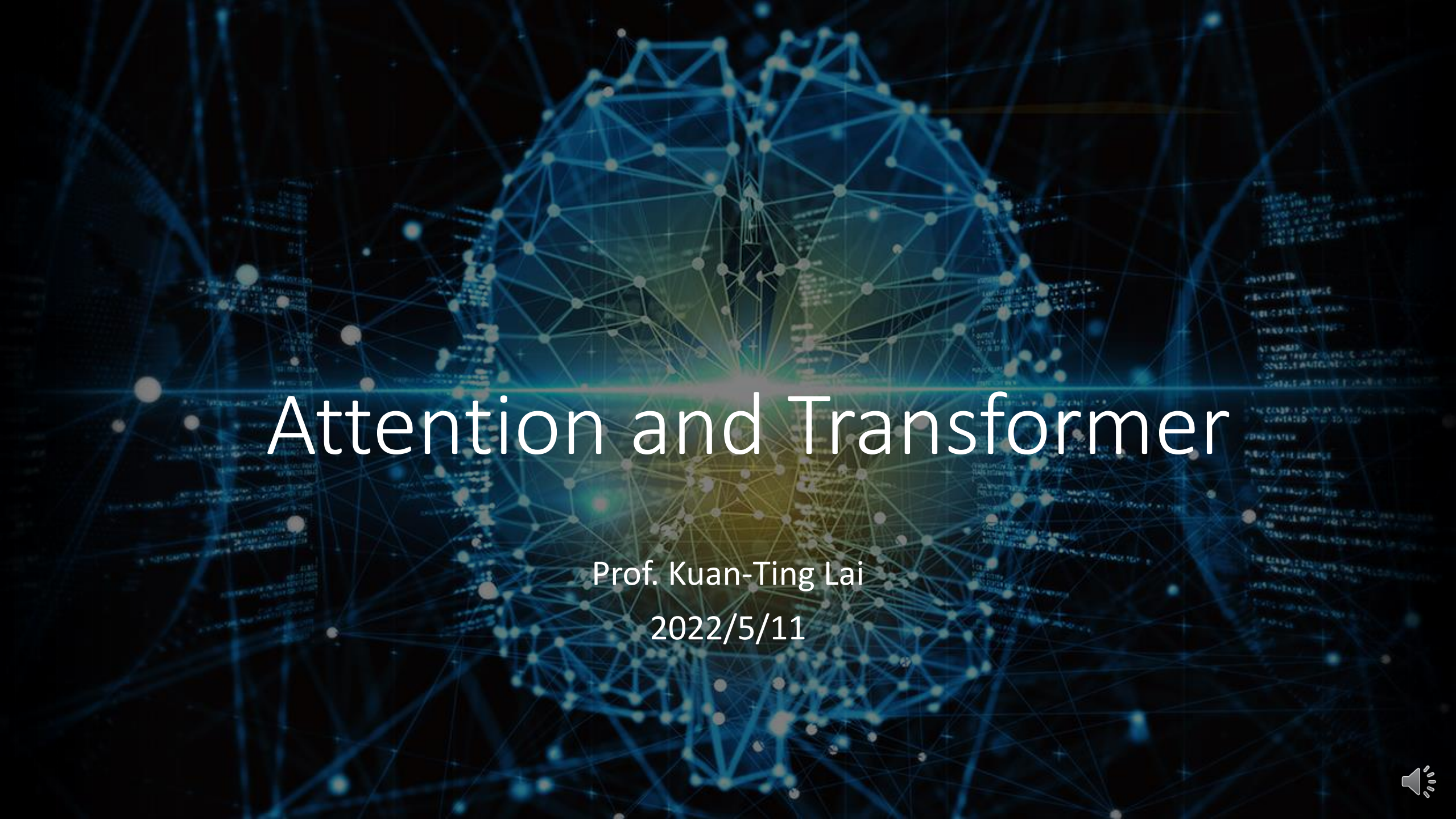
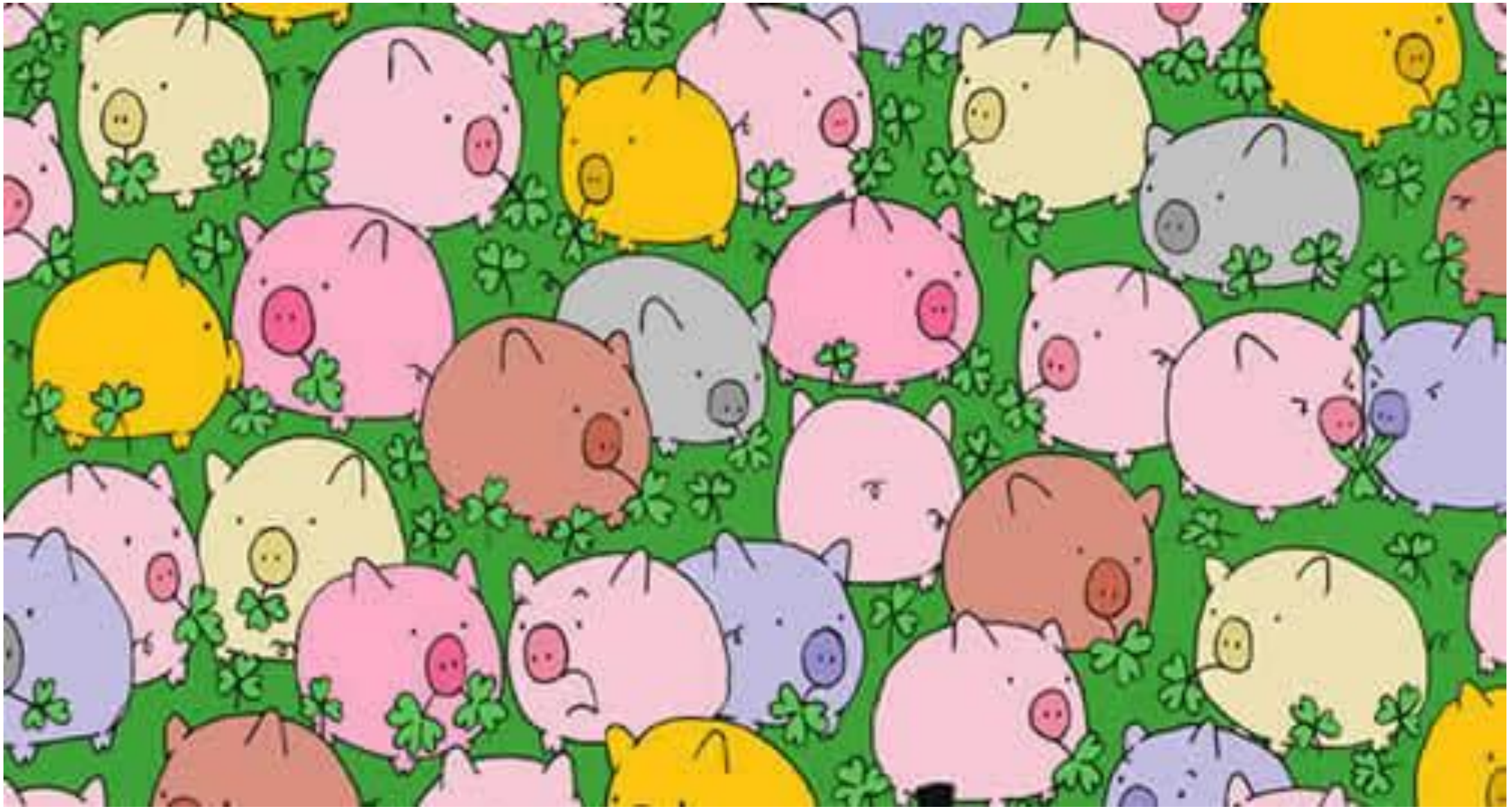


Attention and Transformer

Prof. Kuan-Ting Lai

2022/5/11

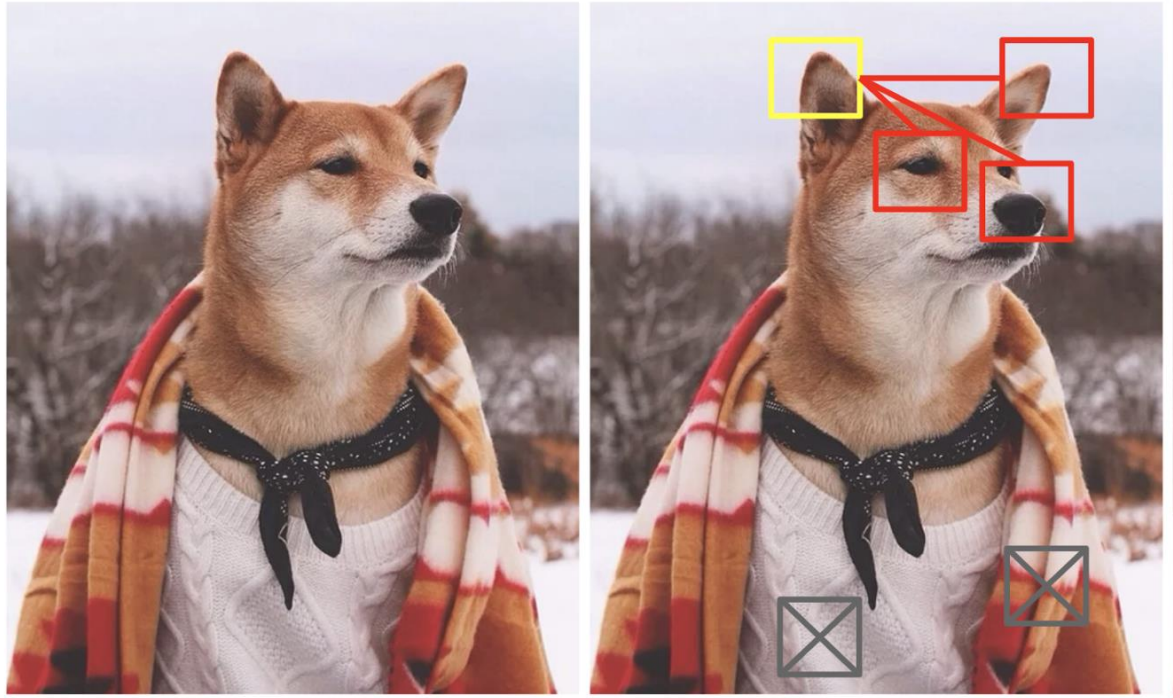
Attention Test: Where is the 4-leaf clover?

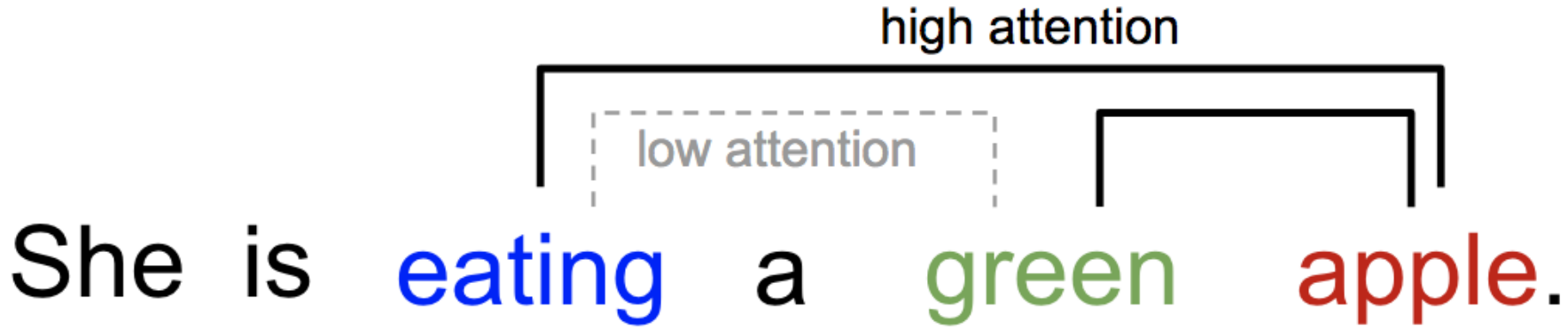


https://www.mentalup.co/blog/concentration-test

Attention

- Visual attention and textual attention

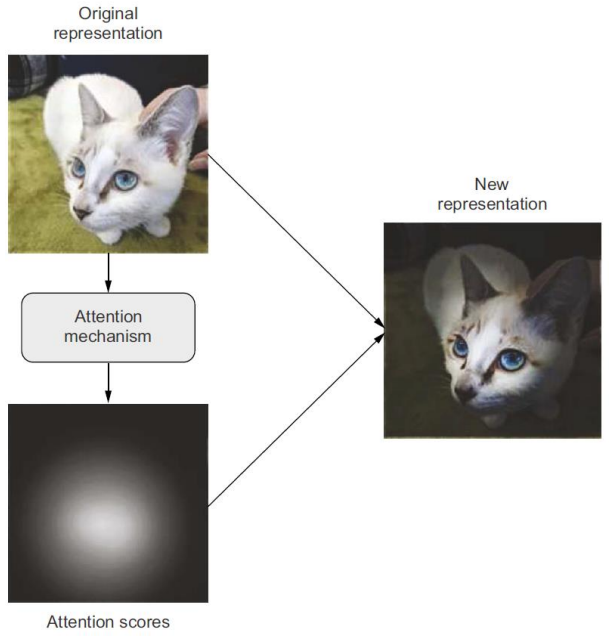




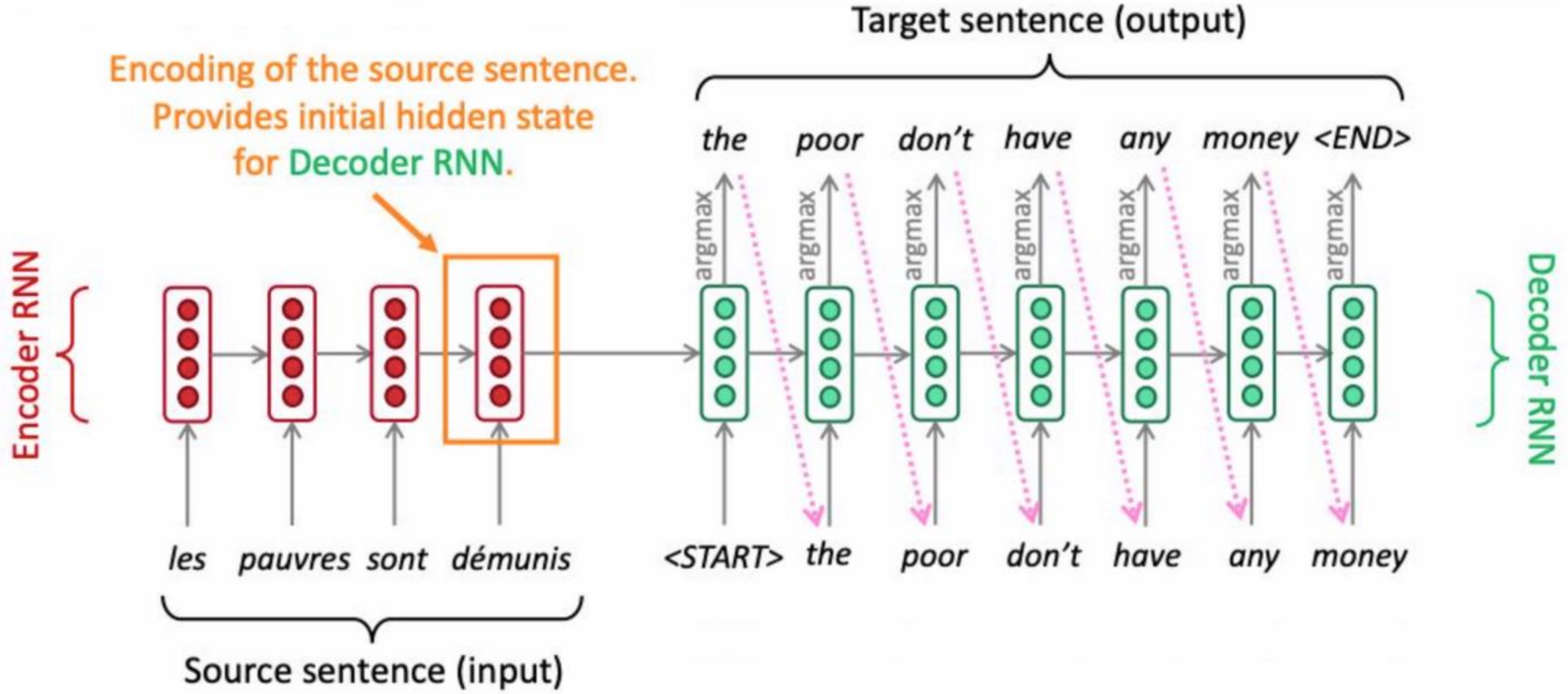
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention = Vector of Importance Weights

- Use weights to assign importance of input data
- Types of attention
  - –Implicit vs. Explicit
    - Max pooling is implicit attention
  - –Hard vs. Soft
    - Soft attention is described by continuous variables
    - Hard attention is described by discrete variables
  - –Additive vs. Multiplicative

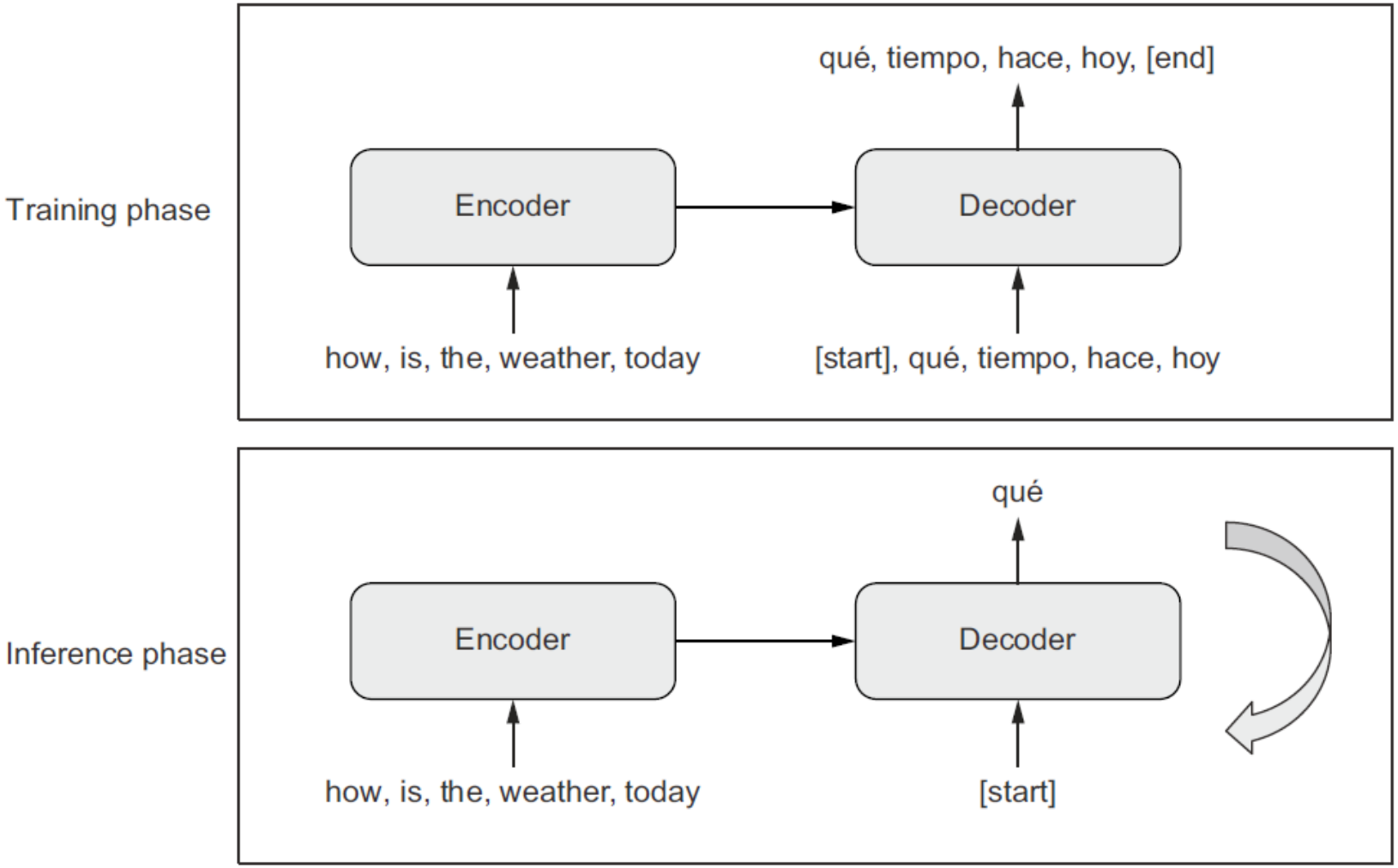


Seq2seq model (Language Translation)



https://towardsdatascience.com/introduction-to-rnns-sequence-to-sequence-language-translation-and-attention-fc43ef2cc3fd

Training and Inference of Seq-to-Seq



Homonyms: Multiple-meaning Words

- Words are "Context-aware".

- **date**

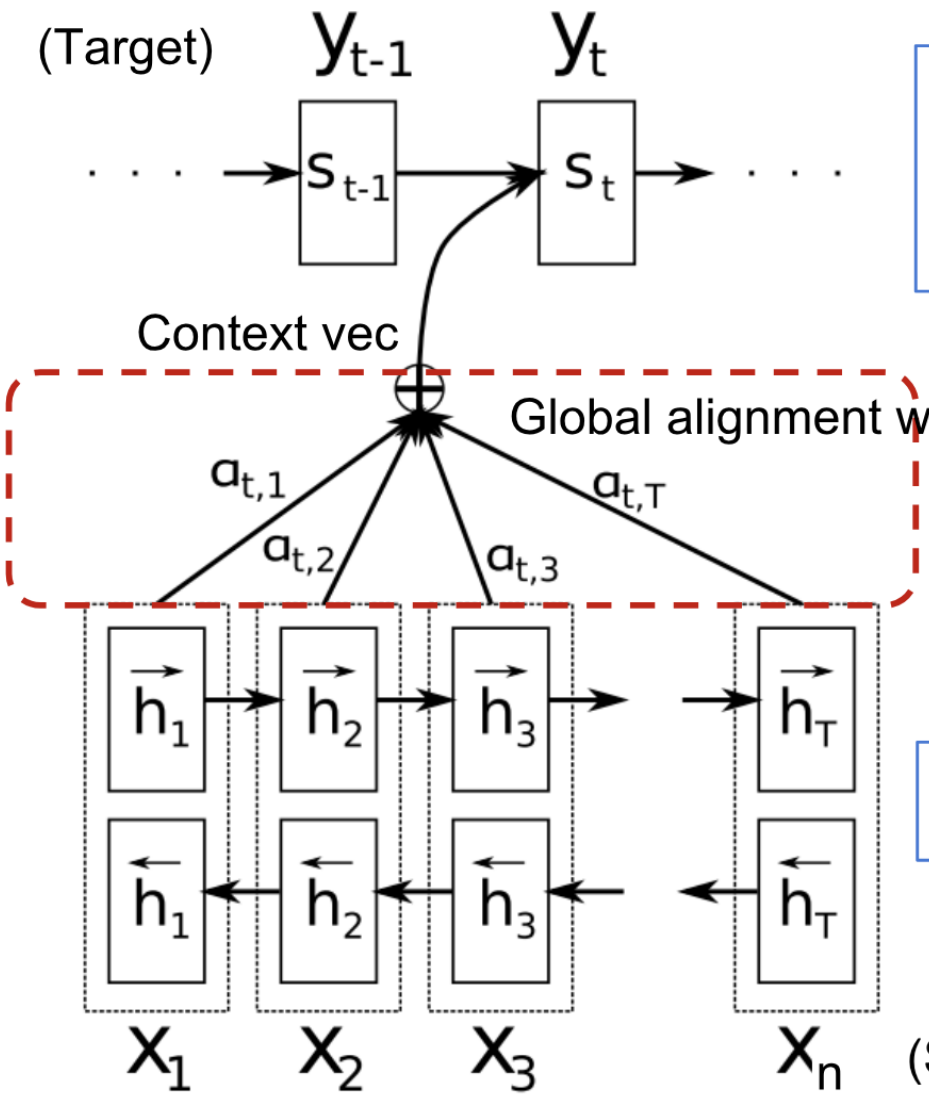
Her favorite fruit to eat is a **date**.
John took Mary out on a **date**.
What is your **date** of birth?

- **fall**

I love cool, crisp **fall** weather.
Don't **fall** on your way to the gym.

https://grammar.yourdictionary.com/for-students-and-parents/words-with-multiple-meanings.html

Additive Attention in RNN



Decoder: RNN with input from previous state + dynamic context vector.

Attention layer: parameterized by a simple feed-forward network

Additive Attention

Encoder: bidirectional RNN

Attention is All You Need!

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

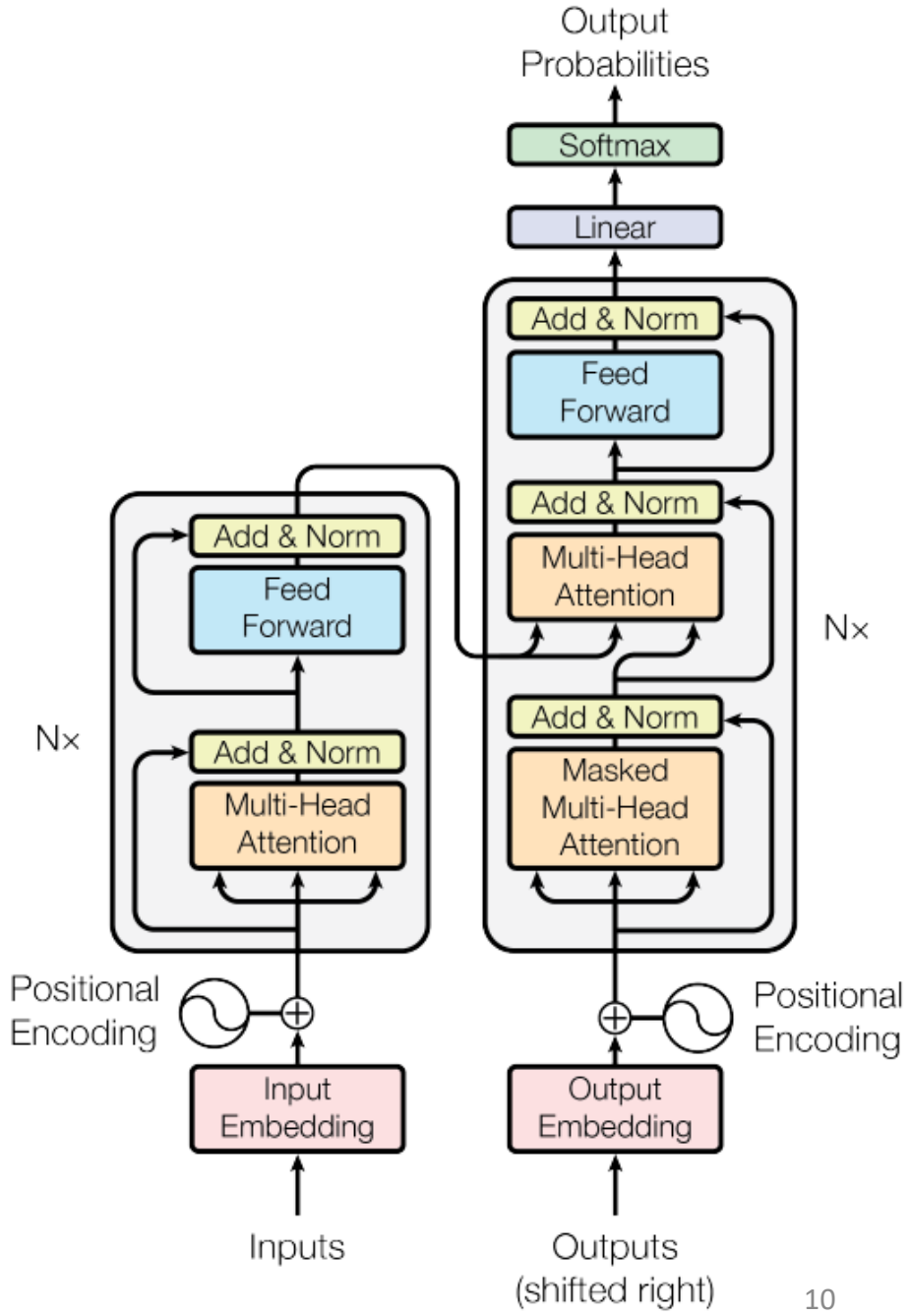
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

NIPS, 2017

Google Brain & University of Toronto

The Transformer Model

- Multi-head attention
  - Self-attention in encoders
  - Masked Self-attention in decoders
  - Encoder-decoder attention
- Positional encoding

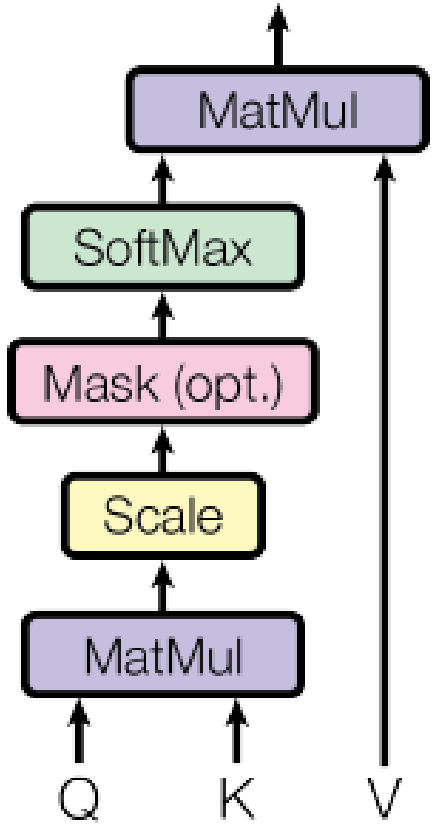


A. Waswani et al., "Attention is All You Need," NIPS, 2017

Attention Module in Transformer

- Query (Q), Key (K), Value (V) attention

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



A. Waswani et al., "Attention is All You Need," NIPS, 2017

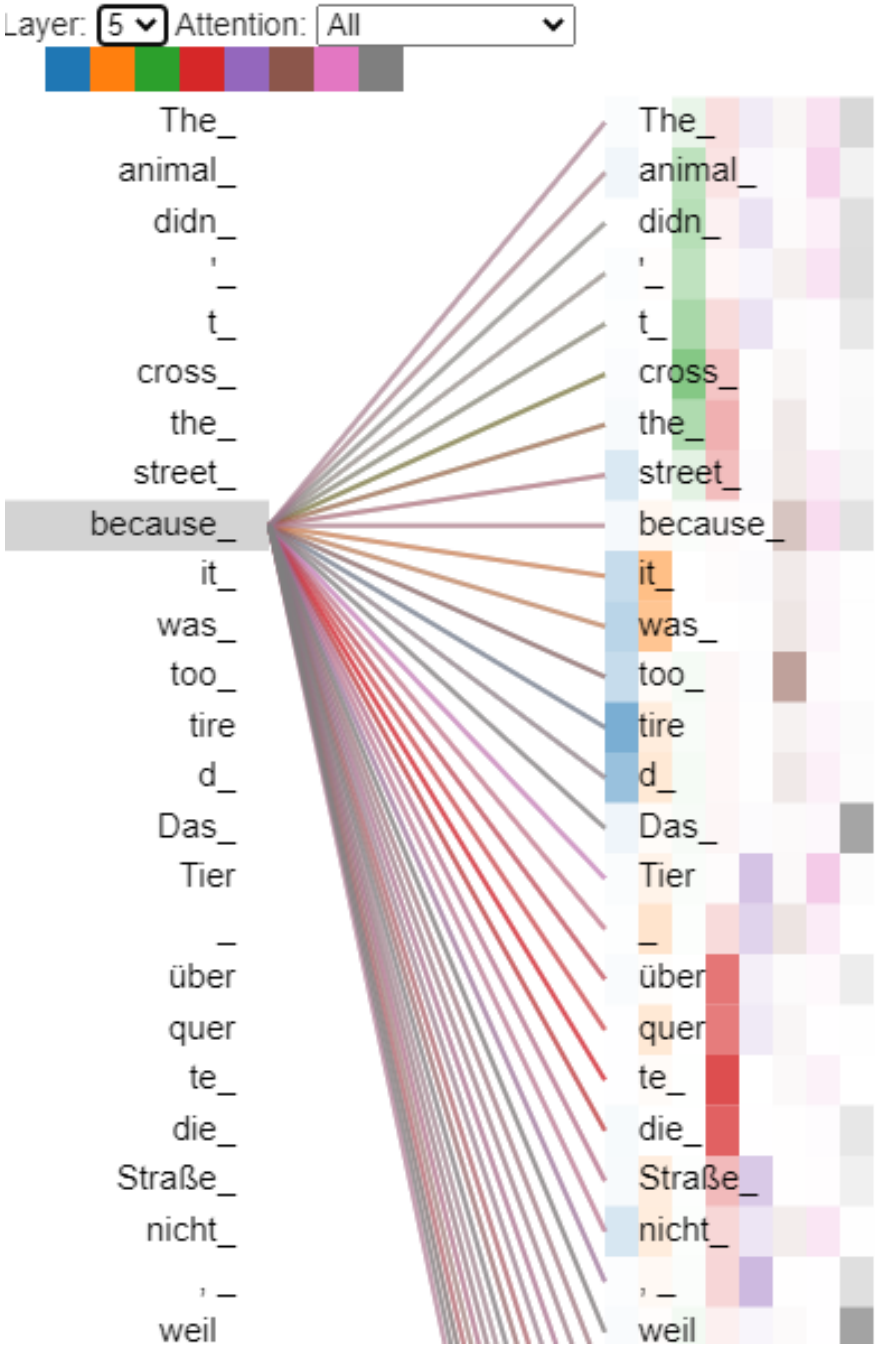
Visualizing Attention

- Tensor2Tensor Notebook

https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb

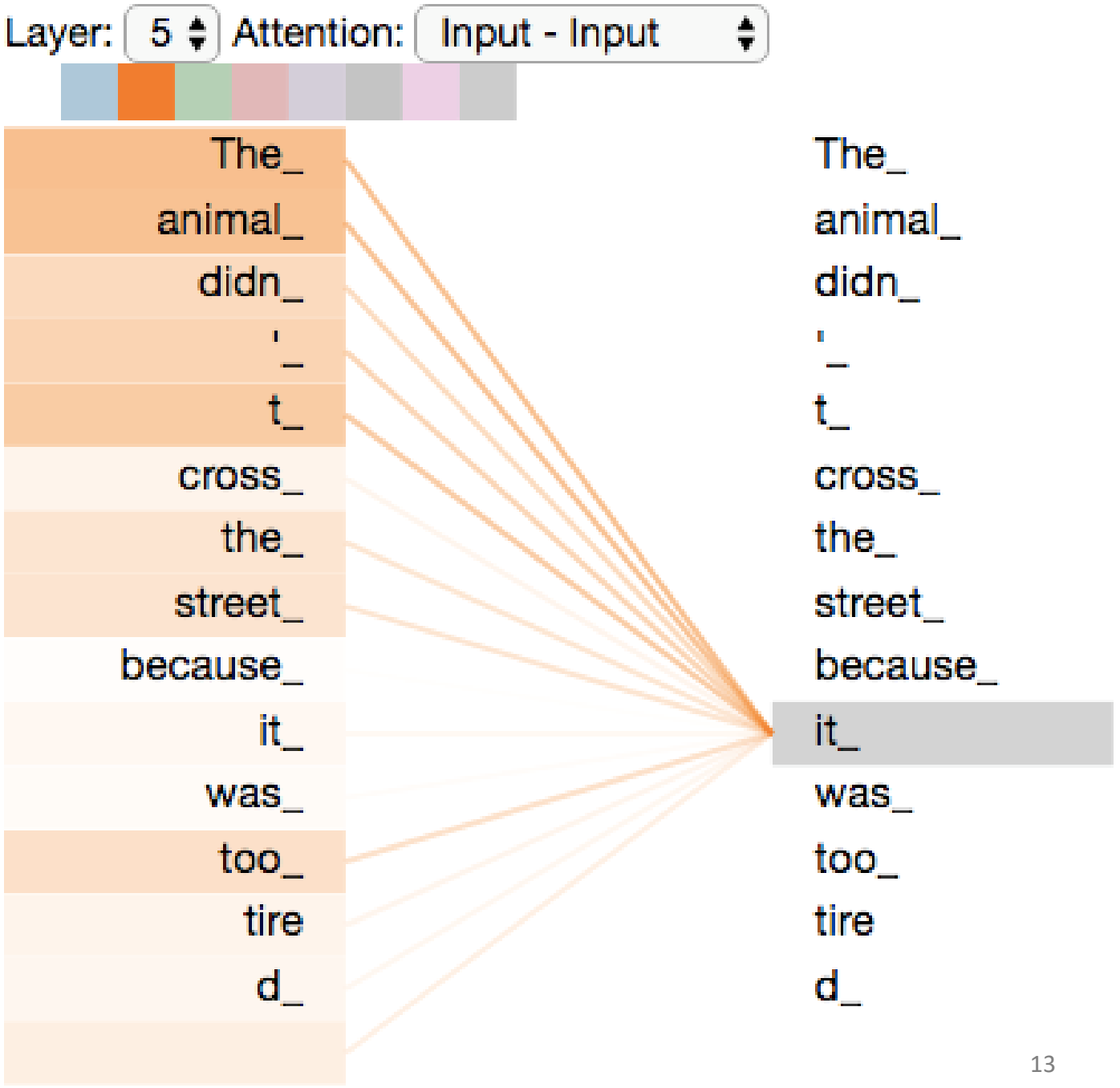
Inputs: The animal didn't cross the street because it was too tired

Outputs: Das Tier überquerte die Straße nicht, weil es zu müde war, weil es zu müde war.



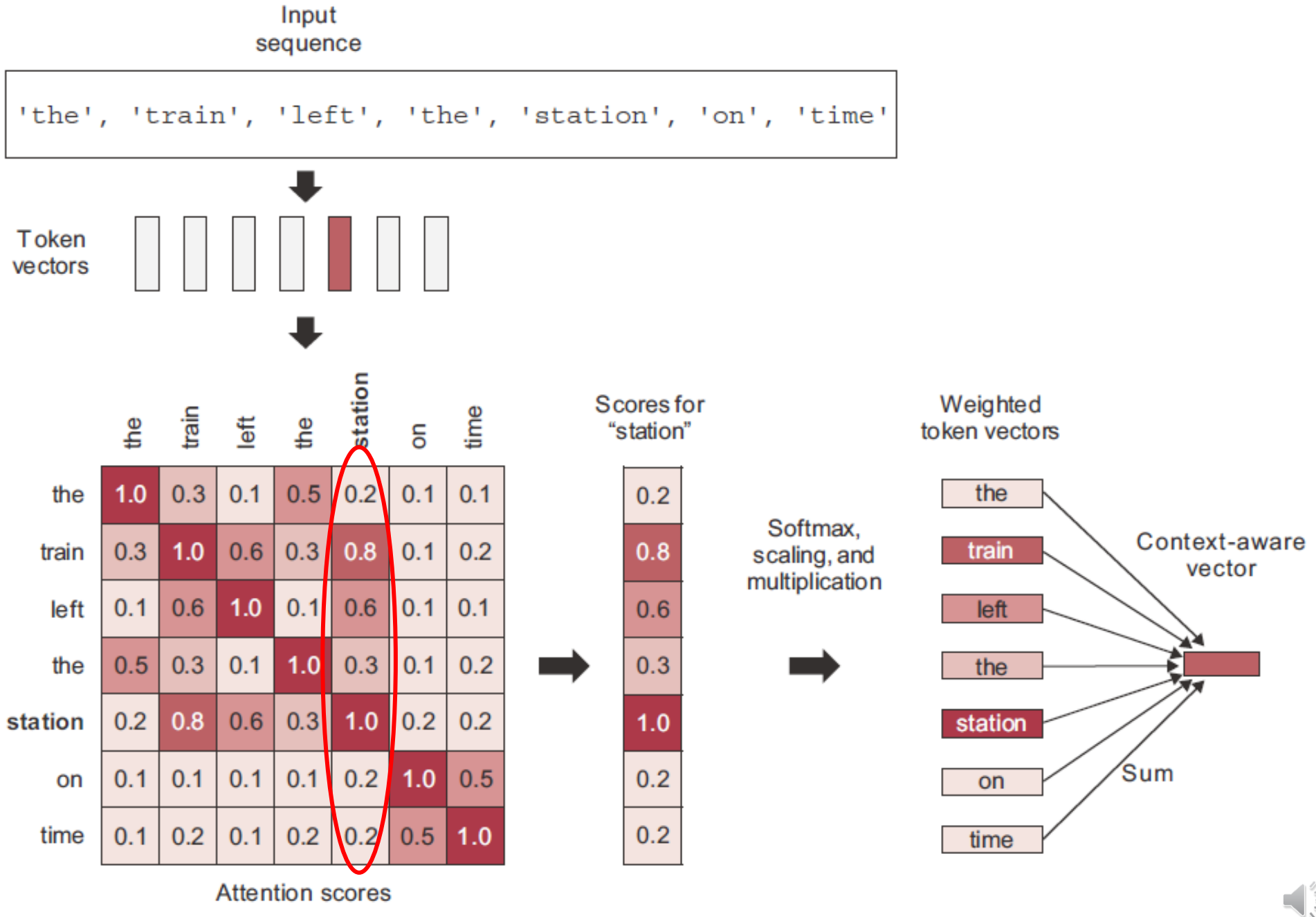
Self-attention Example

- “animal” and “it” have a high relevancy score



Self Attention

Simple Self Attention



Self-attention Pseudocode

```
def self_attention(input_sequence):
    output = np.zeros(shape=input_sequence.shape)
    for i, pivot_vector in enumerate(input_sequence):
        scores = np.zeros(shape=(len(input_sequence),))
        for j, vector in enumerate(input_sequence):
            scores[j] = np.dot(pivot_vector, vector.T)
        scores /= np.sqrt(input_sequence.shape[1])
        scores = softmax(scores)
        new_pivot_representation = np.zeros(shape=pivot_vector.shape)
        for j, vector in enumerate(input_sequence):
            new_pivot_representation += vector * scores[j]
        output[i] = new_pivot_representation
    return output
```

Iterate over each token in the input sequence.

Compute the dot product (attention score) between the token and every other token.

Scale by a normalization factor, and apply a softmax.

Take the sum of all tokens weighted by the attention scores.

That sum is our output.

Query, Keys, Values

```
outputs = sum(inputs * pairwise_scores(inputs, inputs))
```

```
outputs = sum(values * pairwise_scores(query, keys))
```

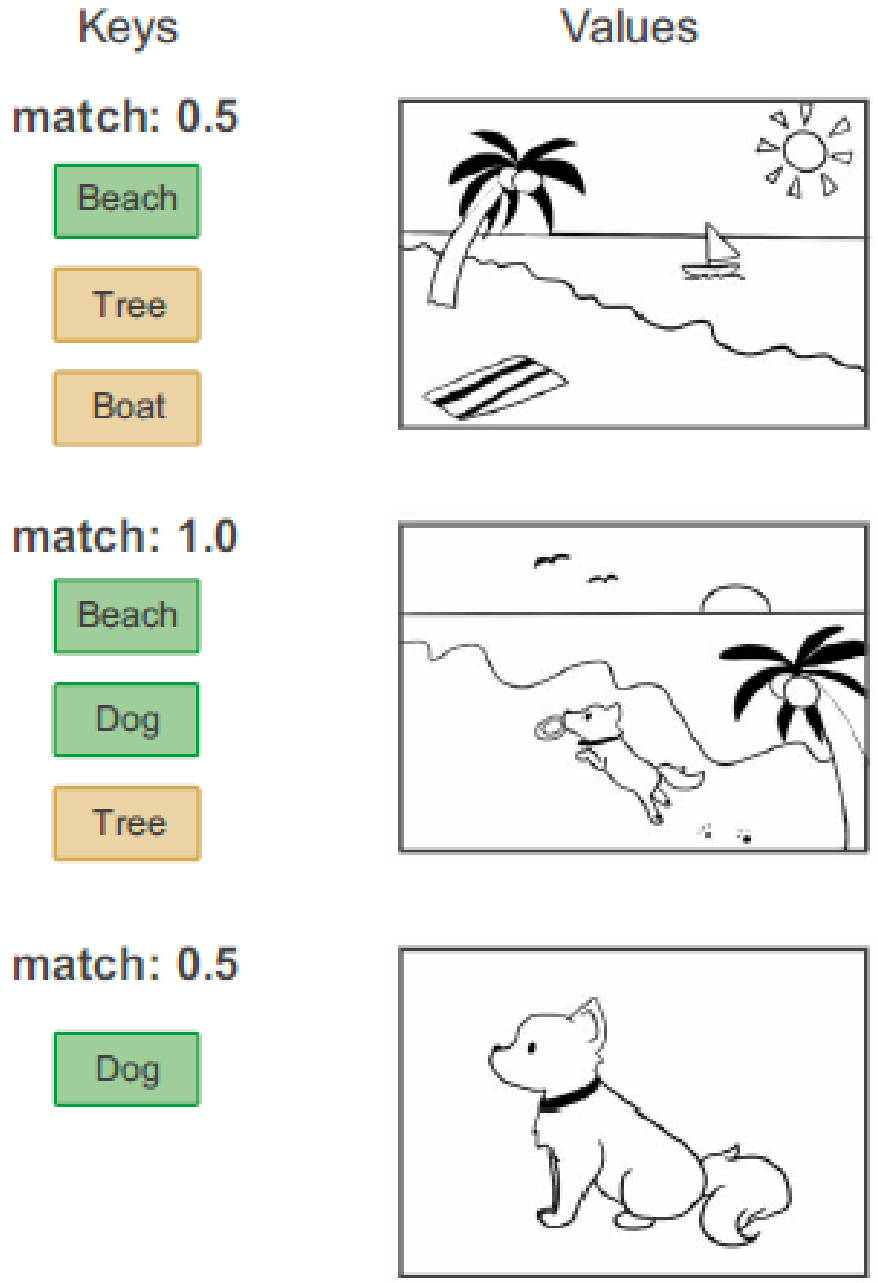
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Query, Keys, Values

Query

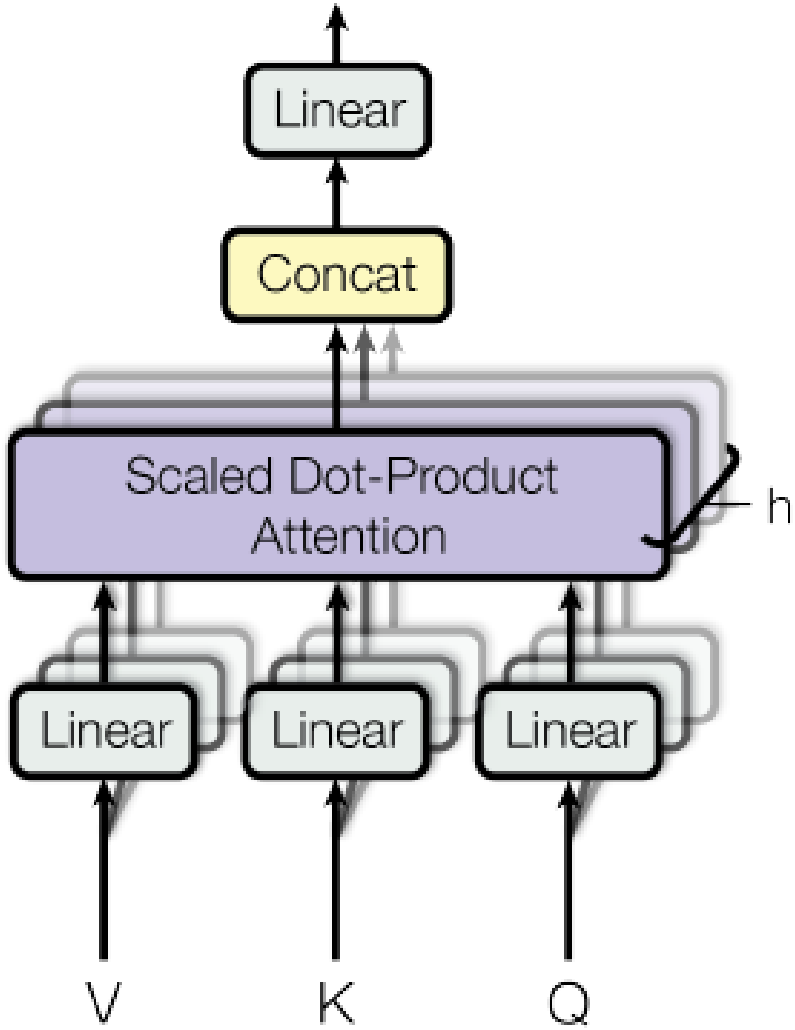
"dogs on the beach"

Retrieving images from a database



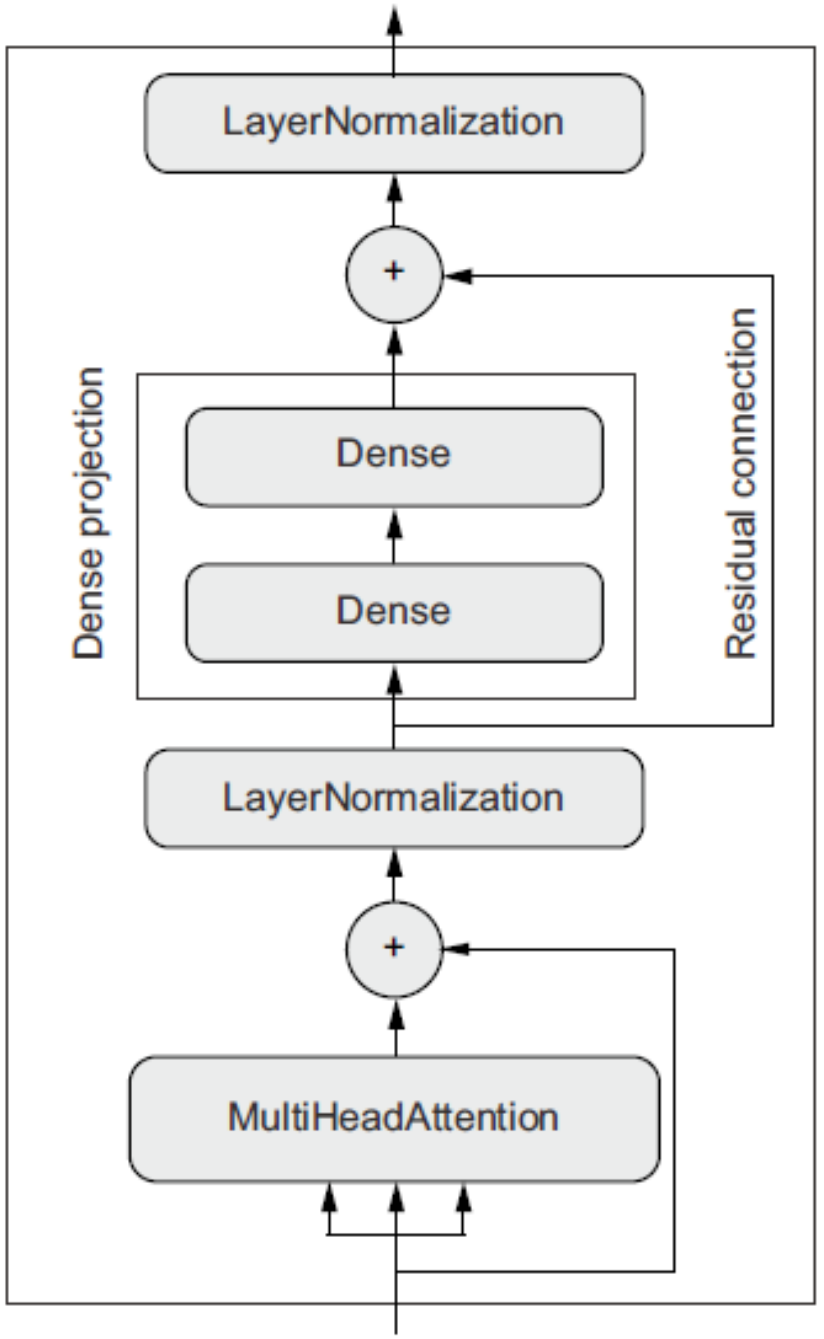
Multi-head Attention

- Use multiple attention modules and concatenate outputs
- Like depthwise separable convolution



Transformer Encoder

- Multi-head attention
- Dense network
- Residual connection
- Layer normalization



Transformer Encoder

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),]
        )
        self.layernorm_2 = layers.LayerNormalization()

    def call(self, inputs, mask=None):
        if mask is not None:
            mask = mask[:, tf.newaxis, :]
        attention_output = self.attention(
            inputs, inputs, attention_mask=mask)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)

    def get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config
```

Size of the input token vectors

Size of the inner dense layer

Number of attention heads

Computation goes in call().

The mask that will be generated by the Embedding layer will be 2D, but the attention layer expects to be 3D or 4D, so we expand its rank.

Implement serialization so we can save the model.

Layer Normalization

```
def layer_normalization(batch_of_sequences):
    mean = np.mean(batch_of_sequences, keepdims=True, axis=-1)
    variance = np.var(batch_of_sequences, keepdims=True, axis=-1)
    return (batch_of_sequences - mean) / variance
```

Input shape: (batch_size, sequence_length, embedding_dim)

To compute mean and variance, we only pool data over the last axis (axis -1).

Compare to `BatchNormalization` (during training):

```
def batch_normalization(batch_of_images):
    mean = np.mean(batch_of_images, keepdims=True, axis=(0, 1, 2))
    variance = np.var(batch_of_images, keepdims=True, axis=(0, 1, 2))
    return (batch_of_images - mean) / variance
```

Input shape: (batch_size, height, width, channels)

Pool data over the batch axis (axis 0), which creates interactions between samples in a batch.

Transformer Encoder for Text Classification

```
vocab_size = 20000
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
x = layers.Embedding(vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()
```

Since TransformerEncoder returns full sequences, we need to reduce each sequence to a single vector for classification via a global pooling layer.

Different Types of NLP Models

	Word order awareness	Context awareness (cross-words interactions)
Bag-of-unigrams	No	No
Bag-of-bigrams	Very limited	No
RNN	Yes	No
Self-attention	No	Yes
Transformer	Yes	Yes

Positional Encoding

A downside of position embeddings is that the sequence length needs to be known in advance.

```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=input_dim, output_dim=output_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)

    def get_config(self):
        config = super().get_config()
        config.update({
            "output_dim": self.output_dim,
            "sequence_length": self.sequence_length,
            "input_dim": self.input_dim,
        })
        return config
```

Prepare an Embedding layer for the token indices.

And another one for the token positions

Add both embedding vectors together.

Like the Embedding layer, this layer should be able to generate a mask so we can ignore padding 0s in the inputs. The compute_mask method will called automatically by the framework, and the mask will get propagated to the next layer.

Implement erialization so we can save the model.

Transformer Encoder with Positional Embedding

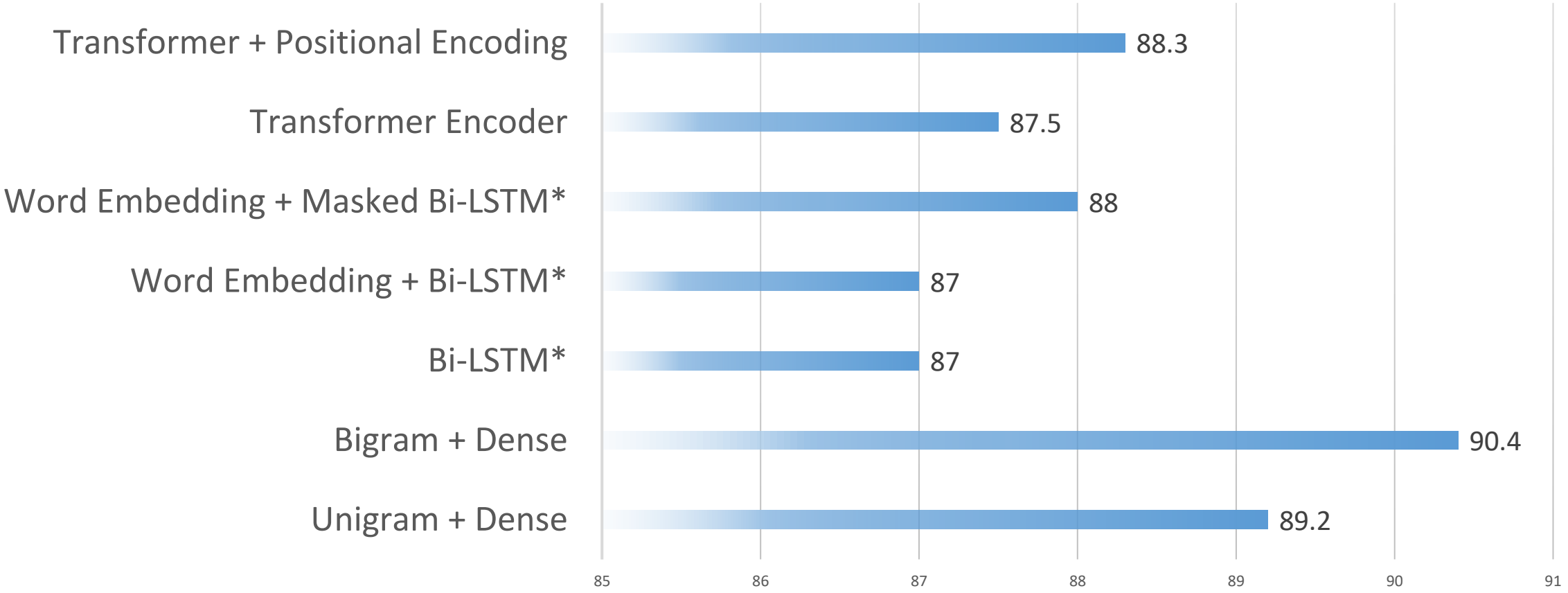
```
vocab_size = 20000
sequence_length = 600
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(optimizer="rmsprop",
              loss="binary_crossentropy",
              metrics=["accuracy"])
model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint("full_transformer_encoder.keras",
                                    save_best_only=True)
]
model.fit(int_train_ds, validation_data=int_val_ds, epochs=20,
     callbacks=callbacks)
model = keras.models.load_model(
    "full_transformer_encoder.keras",
    custom_objects={"TransformerEncoder": TransformerEncoder,
                    "PositionalEmbedding": PositionalEmbedding})
print(f"Test acc: {model.evaluate(int_test_ds)[1]:.3f}")
```

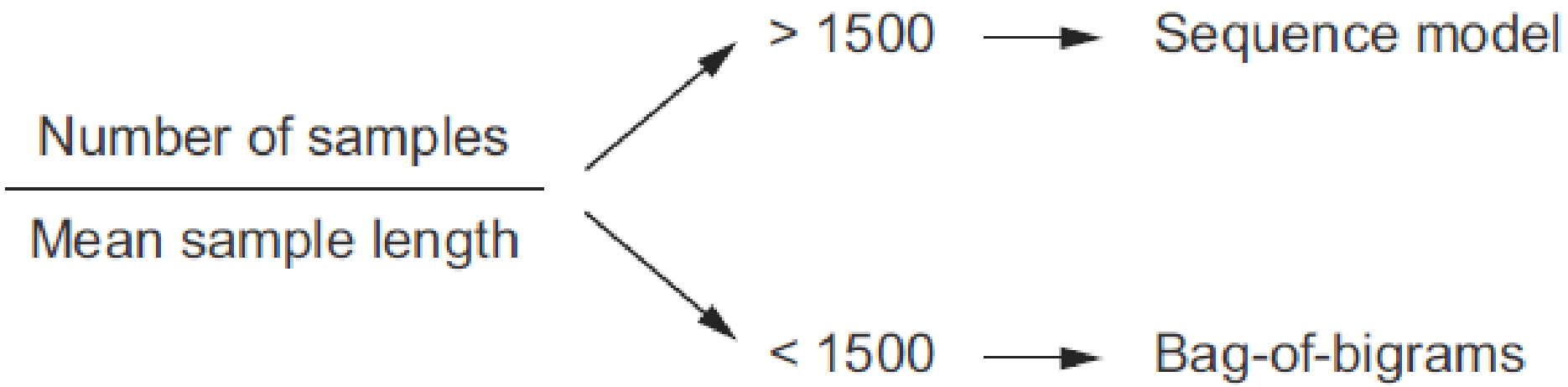
Look here!

IMDB Movie Review Classification

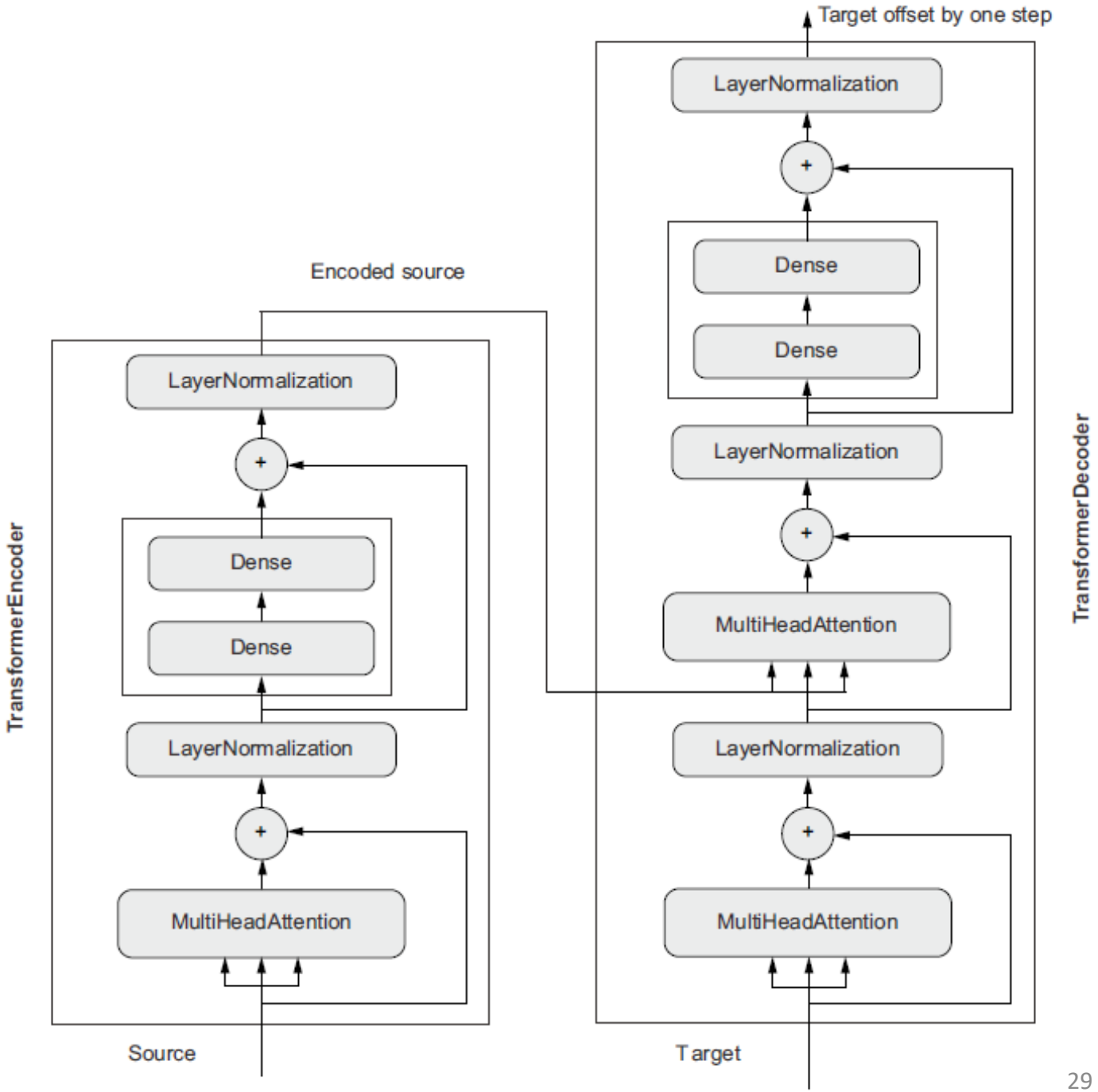


Bag-of-Words or Sequence Models?

- Chollet, et al. (http://mng.bz/AOzK)



Transformer Encoder + Decoder



References

1. François Chollet, “Deep Learning with Python, 2nd edition”, Chapter 11, 2021
2. https://towardsdatascience.com/introduction-to-rnns-sequence-to-sequence-language-translation-and-attention-fc43ef2cc3fd
3. Vaswani et al., “Attention is All You Need,” NIPS, 2017