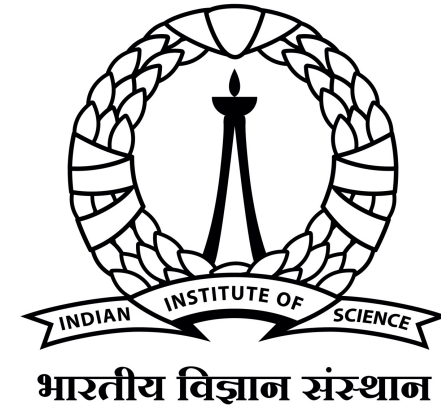




Department of Computational and Data Sciences

# AI Module: Transformer Models



Deepak Subramani

Assistant Professor

Dept. of Computational and Data Science

Indian Institute of Science Bengaluru

# Outline for Week 02

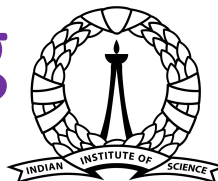
- Part 01: Sequence Modeling
  - NLP as a sequence modelling task
  - Names of recurrent architectures and introduction
  - Details in an additional slide deck on RNN – not included in the main content
    - Other Seq Tasks (Time Series Modeling; Not NLP)
    - Additional notebooks on RNN and LSTM (not compulsory)
- Part 02: Attention Mechanism and Transformer Encoders
  - Key, Query, Value
  - Multi Head Attention
  - MHA as text representation learners for use in discriminative tasks
  - Positional Embedding
  - Trick of the trade: Layer Normalization
  - Assignment on Attention Mechanism

# Outline for Week 02

- **Part 01: Sequence Modeling**
  - NLP as a sequence modelling task
  - Names of recurrent architectures and introduction
  - Details in an additional slide deck on RNN – not included in the main content
    - Other Seq Tasks (Time Series Modeling; Not NLP)
    - Additional notebooks on RNN and LSTM (not compulsory)
- **Part 02: Attention Mechanism and Transformer Encoders**
  - Key, Query, Value
  - Multi Head Attention
  - MHA as text representation learners for use in discriminative tasks
  - Positional Embedding
  - Trick of the trade: Layer Normalization
  - Assignment on Attention Mechanism

# Sequence Model Approach

1. Represent input text as a sequence of integers (1 integer per word)
2. Map it into a vector space (one-hot embedding in vocab dim space projected to embed dimensional space)
3. Feed this sequence of vectors to a stack of layers capable of handling sequences (Transformer or RNN/LSTM/Bi-LSTM)



# Neural Network Layers for Sequence Modeling Approach

- Recurrent Layers
  - LSTM
  - GRU
- Transformer Layers
  - Attention
  - Multi Head Attention (MHA) – workhorse of modern NLP incl GPT4, Llama, Copilot, Mistral, Gemini, Gemma, Claude etc

# RNN vs Transformers: When to use?

- RNN/LSTM are almost obsolete in modern NLP pipelines
- Some use cases still relevant for RNN/LSTM in 2024:
  1. Low data, low compute settings where training a heavy transformer model is expensive
  2. As a baseline for larger transformer models
  3. Application is a legacy code from 2014-2018 period

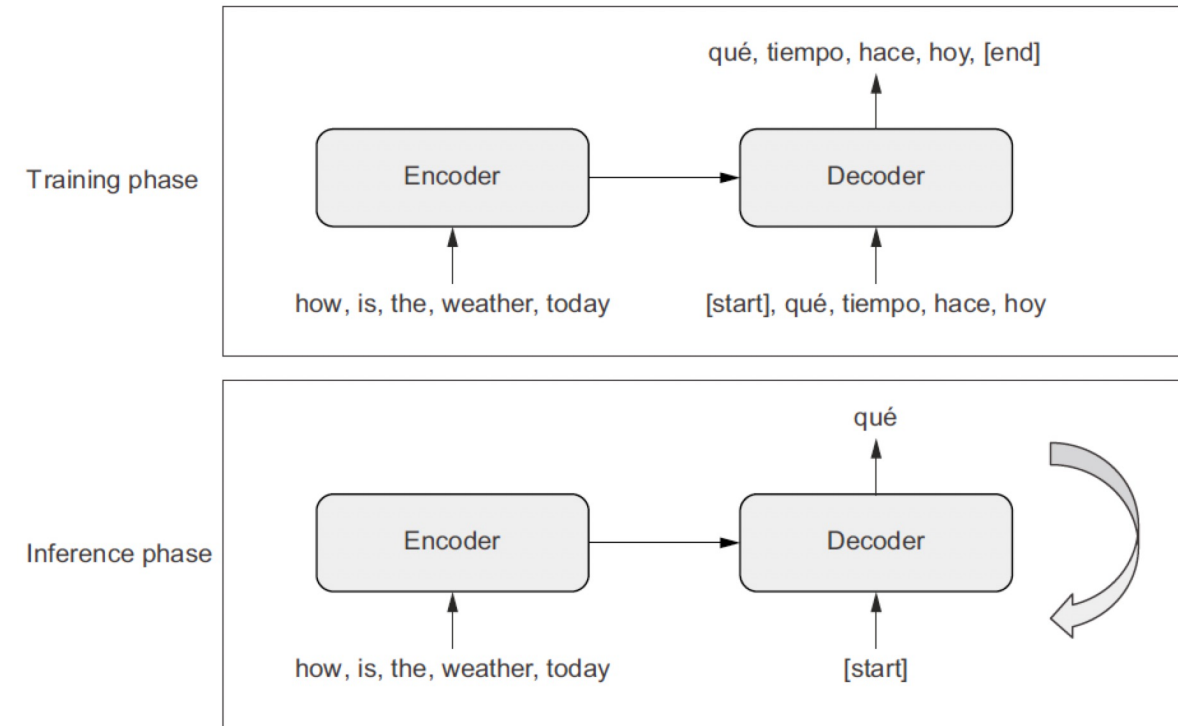
# Sequence-to-Sequence NLP Tasks

- A seq-to-seq model takes a sequence as input and provides another sequence
- Examples
  - Machine Translation
  - Text Summarization
  - Question Answering
  - Chat bot
  - Text generation



# Seq-to-Seq: General Template

- Encoder turns an input sequence to an intermediate representation
- Decoder is trained to predict the next token  $i$  from the previous token (0 to  $i - 1$ ) and the encoded source sequence
- During inference:
  - Encoder encodes the source sequence
  - Decoder uses a seed token (such as [start]) and the encoded source to produce the first token of the target
  - Predicted sequence so far is fed back into the decoder that predicts the next token
  - Repeat



Neural Machine Translation is the birthplace of Transformers



# Transformers

- Transformer Encoder
  - Converts a sequence of words to a vector representation
  - This vector representation can be used for text-understanding tasks
- Transformer Decoder
  - Uses the context of the sequence of words so far (sometimes with an additional context from encoder or retrieval) to predict next token in the sequence

# Outline for Week 02

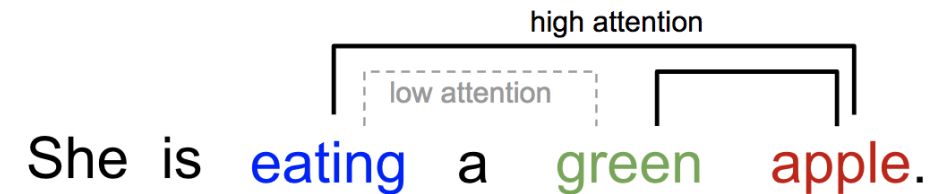
- Part 01: Sequence Modeling
  - Essential tasks in sequence modeling
  - When to use a seq model?
  - Names of recurrent architectures and introduction (details in additional slide deck on RNN – not included in the main content)
  - Additional notebooks on RNN and LSTM (not compulsory)
- Part 02: Attention Mechanism and Transformer Encoders
  - Key, Query, Value
  - Multi Head Attention
  - MHA as text representation learners for use in discriminative tasks
  - Positional Embedding
  - Trick of the trade: Layer Normalization
  - Assignment on Attention Mechanism

# Transformer

- Introduced in the seminal paper Vaswani et al (2017) – Attention is all you need
- This paper introduced transformer as an Encoder-Decoder Neural Machine Translation model
- It was a sequence-to-sequence model for language translation
- The encoder part can be used for other language tasks such as text classification, sentiment analysis
- Gist of the paper: “Neural Attention” can be used for building sequential models without any recurrent or convolutional layers

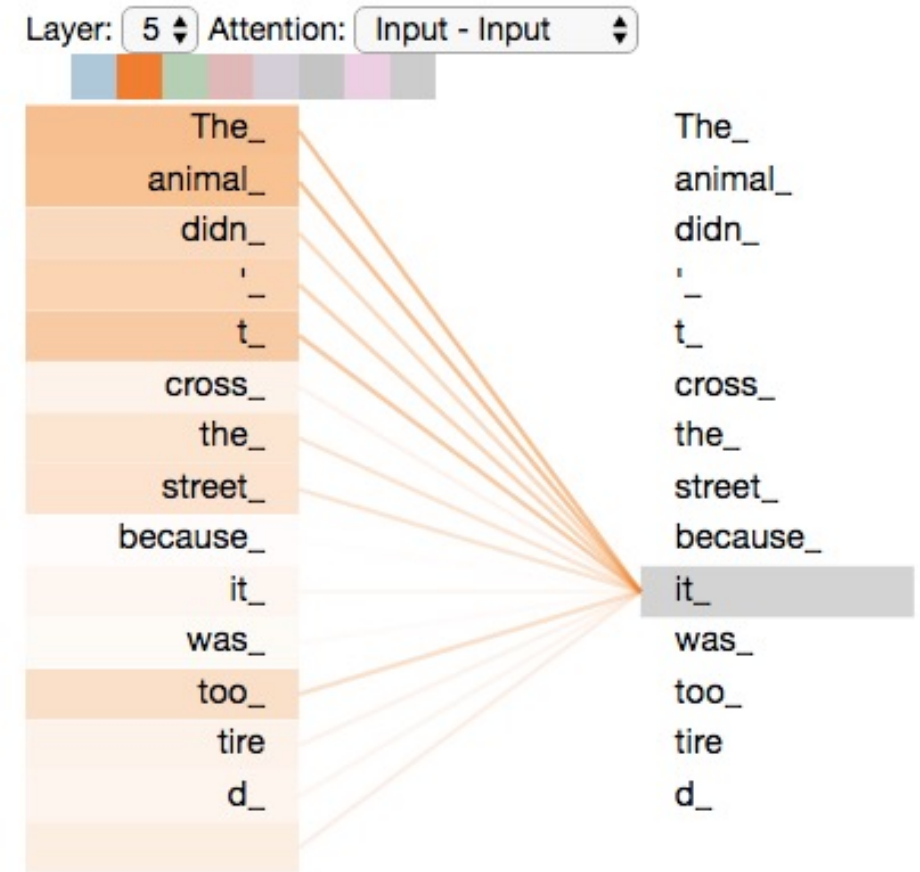
# Self Attention

- Key idea: Some part of the input is more important than other parts
- Model must pay “more attention” to important parts and “less attention” to other parts – A simple mechanism
- Consider the word “station”
  - Are we talking about train station, pizza station, police station or space station?
- Context determines the meaning
- So we need to adaptively change the word vector in the embedding space to represent the meaning of the word in its context
- Self attention provides this mechanism to modulate the representation of a token using the representation of tokens nearby



# Self-Attention

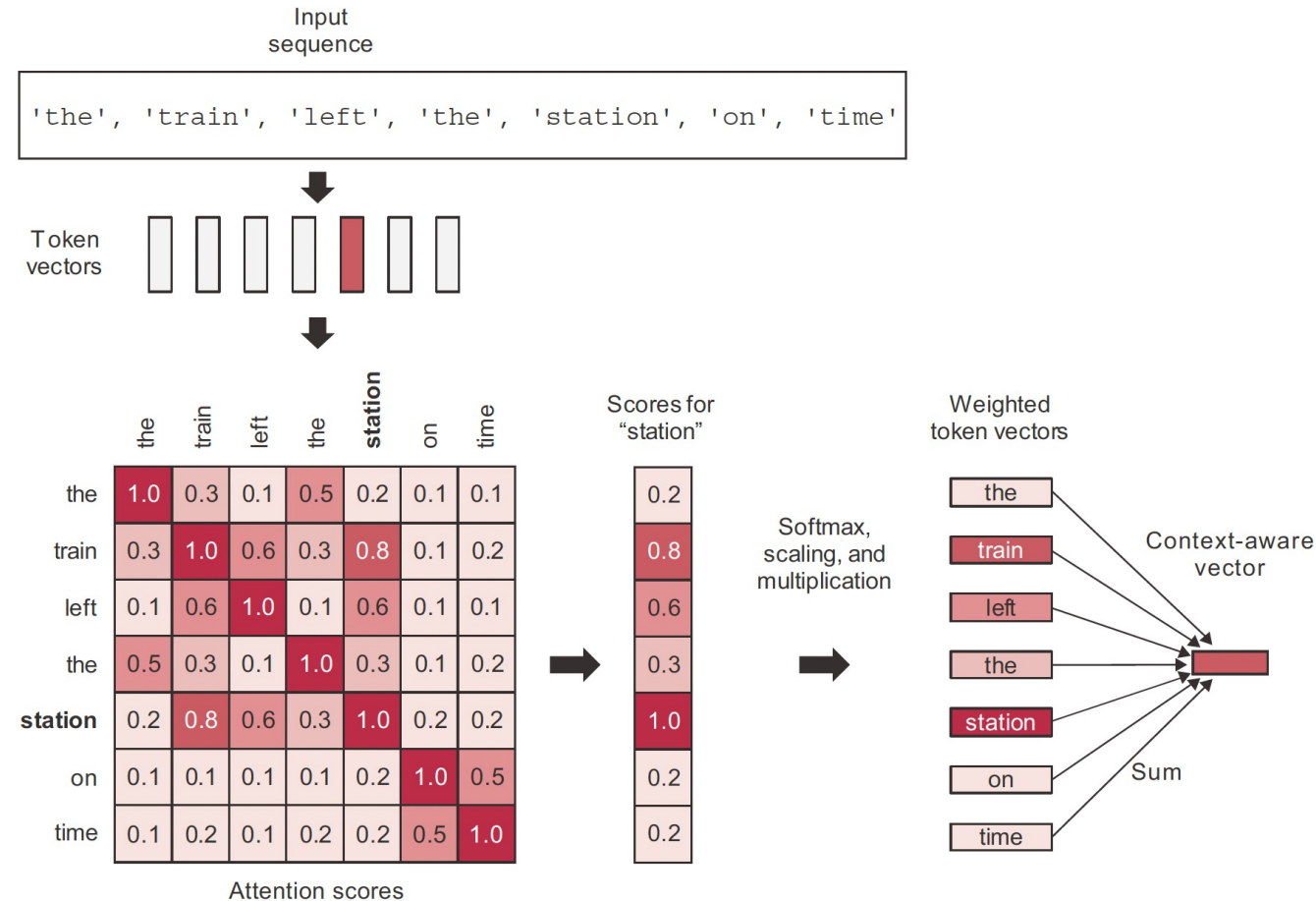
- Consider the sentence: 'The animal didn't cross the street because it was too tired'.
- What does 'it' refer to?
- While performing self attention, the association of 'it' is made with 'animal'.





# Self Attention for Context Aware Vector

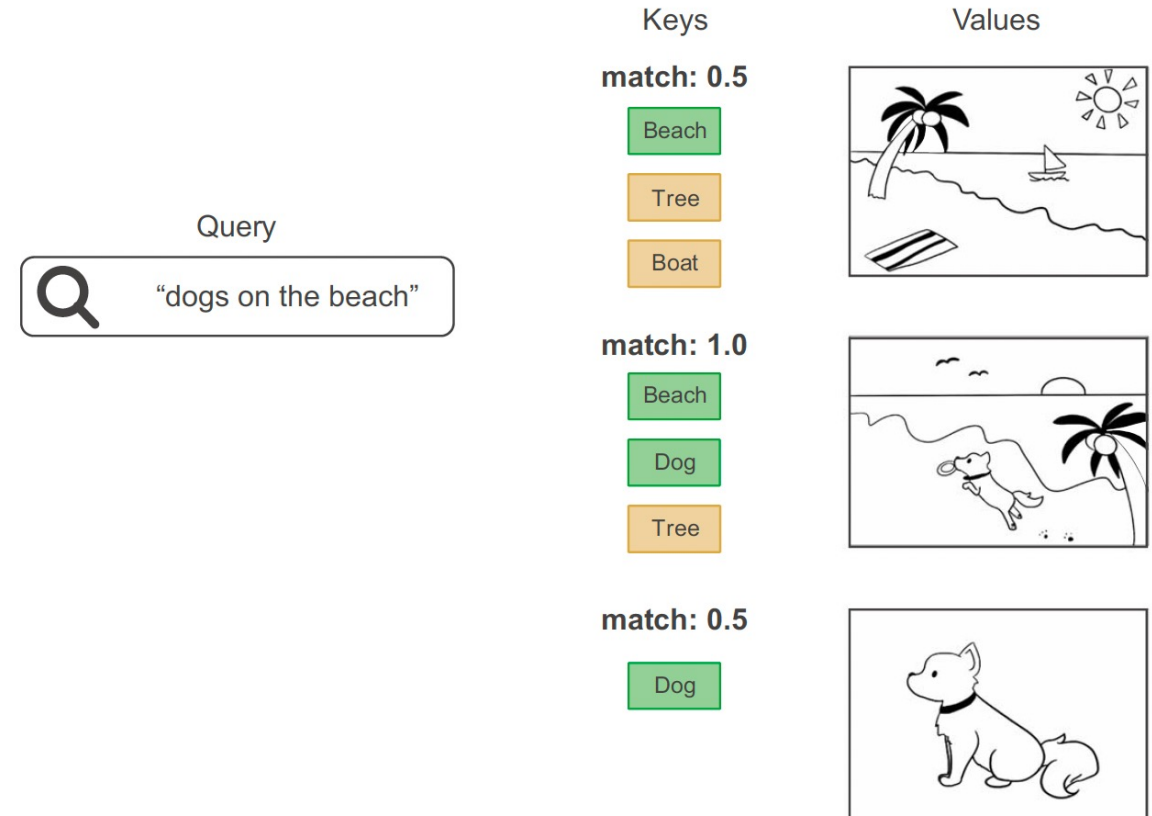
- Step 1: Compute the relevancy score of “station” with every other word in the sequence
- This score (just a dot product, cosine distance) scaled and softmaxed is our attention score
- Step 2: Compute sum of all word vectors weighted with attention scores to replace the representation of station
- Here the word station to itself is also used, so everything works well!
- Repeat the process for every word in the sequence and get a new sequence of vectors encoding the sentence
- $\text{output} = \text{sum}(\text{input} * \text{pairwise\_score}(\text{input}, \text{input}))$



**Figure 11.6** Self-attention: attention scores are computed between “station” and every other word in the sequence, and they are then used to weight a sum of word vectors that becomes the new “station” vector.

# Generalized Self Attention – Query-Key-Value Model

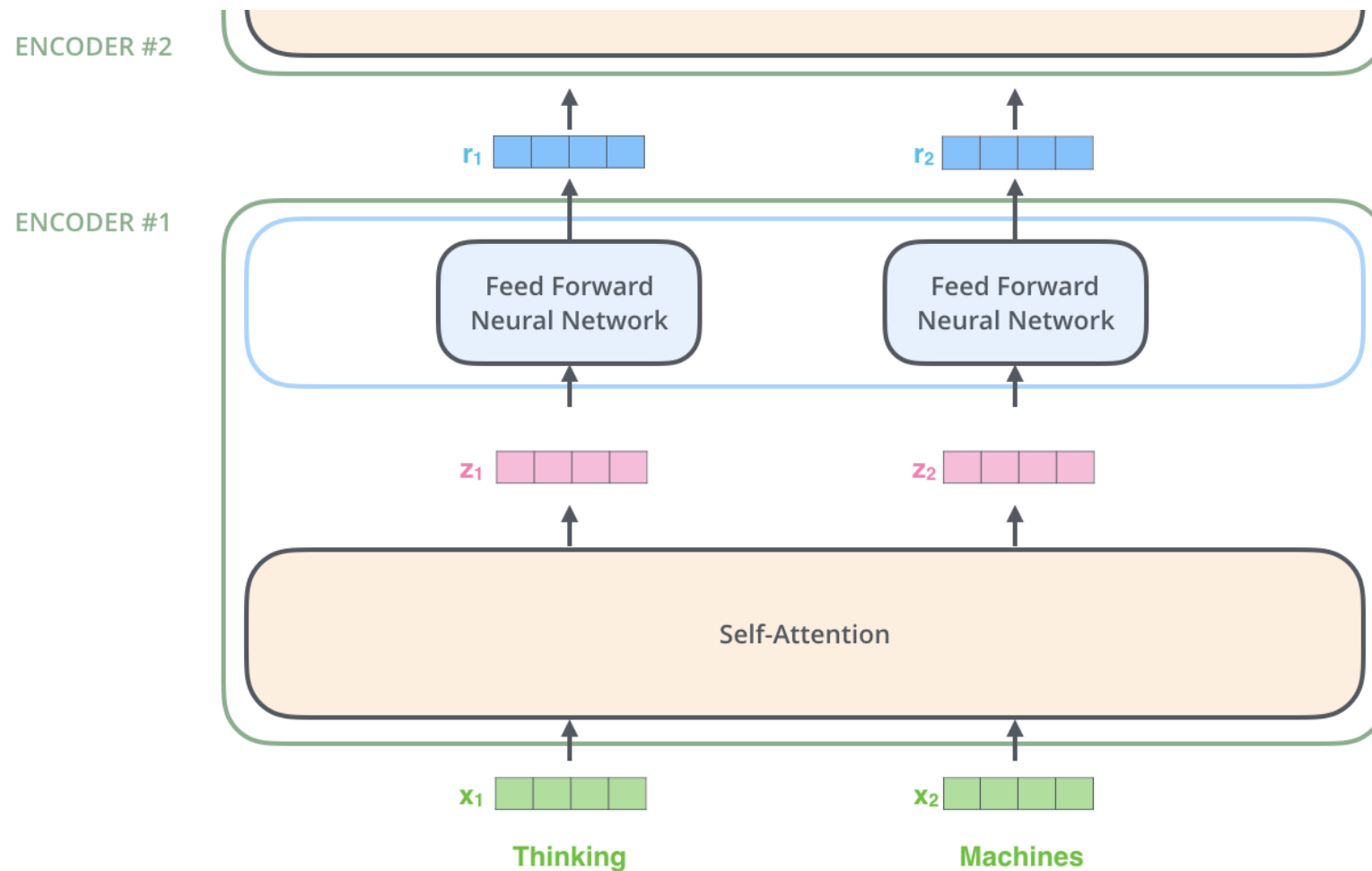
- Consider a search engine example
- Write the Self Attention of the station example as a schematic
- $\text{output} = \text{sum}(\text{input} * \text{pairwise\_score}(\text{input}, \text{input}))$
- Generalize it to a QKV Model
- $\text{output} = \text{sum}(\text{values} * \text{pairwise\_score}(\text{query}, \text{key}))$



**Figure 11.7** Retrieving images from a database: the “query” is compared to a set of “keys,” and the match scores are used to rank “values” (Images).



# Attention through figures







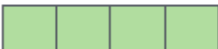
# 1<sup>st</sup> Step: Key, Query, Value

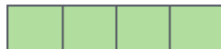
Input

Thinking

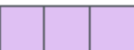
Machines


Embedding

$X_1$  

$X_2$  

Queries


$q_1$  

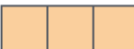
$q_2$  



$W^Q$

Keys

$k_1$  

$k_2$  



$W^K$

Values

$v_1$  

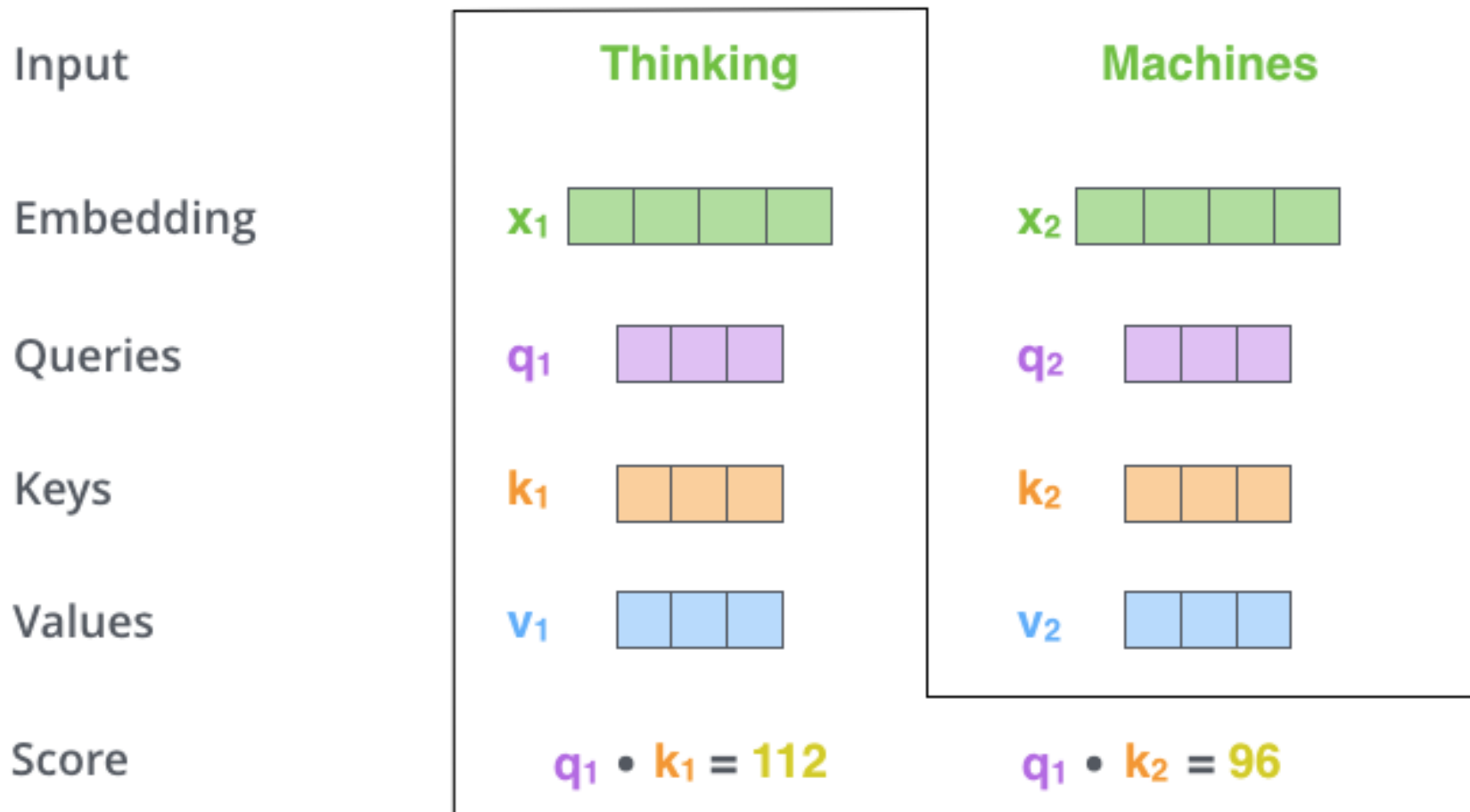
$v_2$  



$W^V$

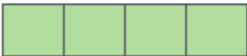
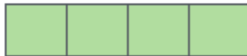


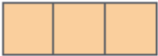
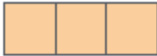
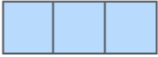
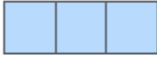


# 2<sup>nd</sup> Step: q.k pairwise score



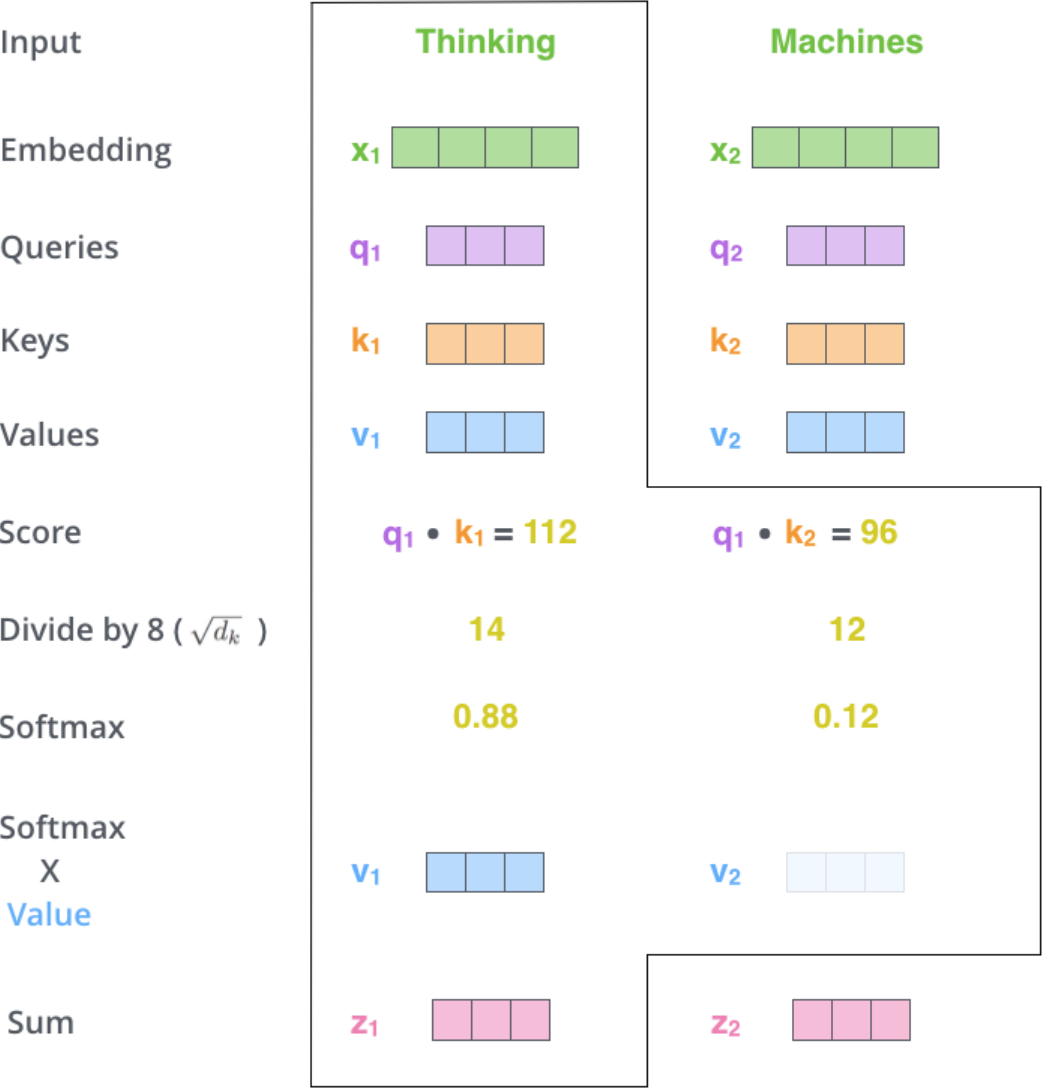


# 3,4: Divide and Normalize

Input	Thinking	Machines
Embedding	$x_1$ 	$x_2$ 
Queries	$q_1$ 	$q_2$ 
Keys	$k_1$ 	$k_2$ 
Values	$v_1$ 	$v_2$ 
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$
Divide by 8 ( $\sqrt{d_k}$ )	14	12
Softmax	0.88	0.12

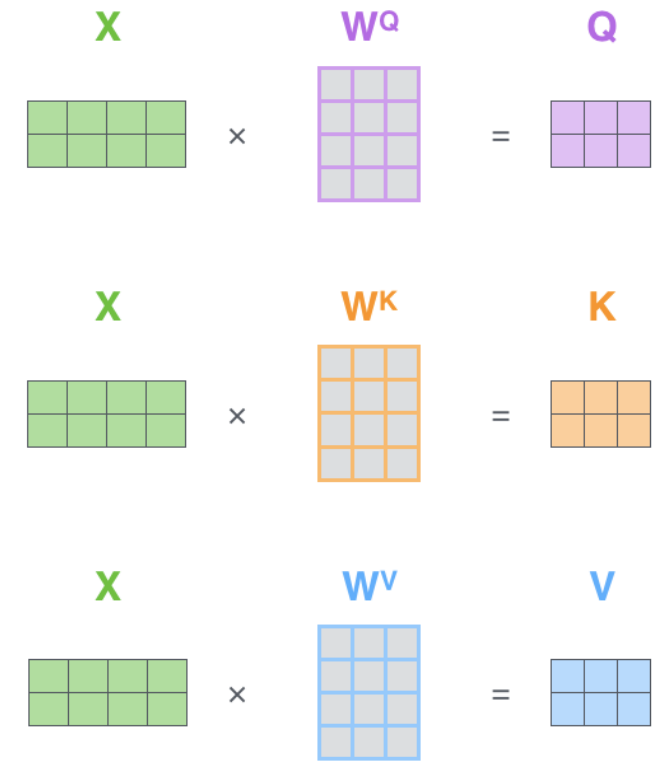


# 5,6: Multiply and Add



# Query-Key-Value as Matrices

- How is self attention calculated?
- Attention operates on **queries, keys and values**.
- They are generated from the same source in case of self-attention.
- The weight matrices are learned during training.
- $q_i = W^Q x_i$
- $k_i = W^K x_i$
- $v_i = W^V x_i$

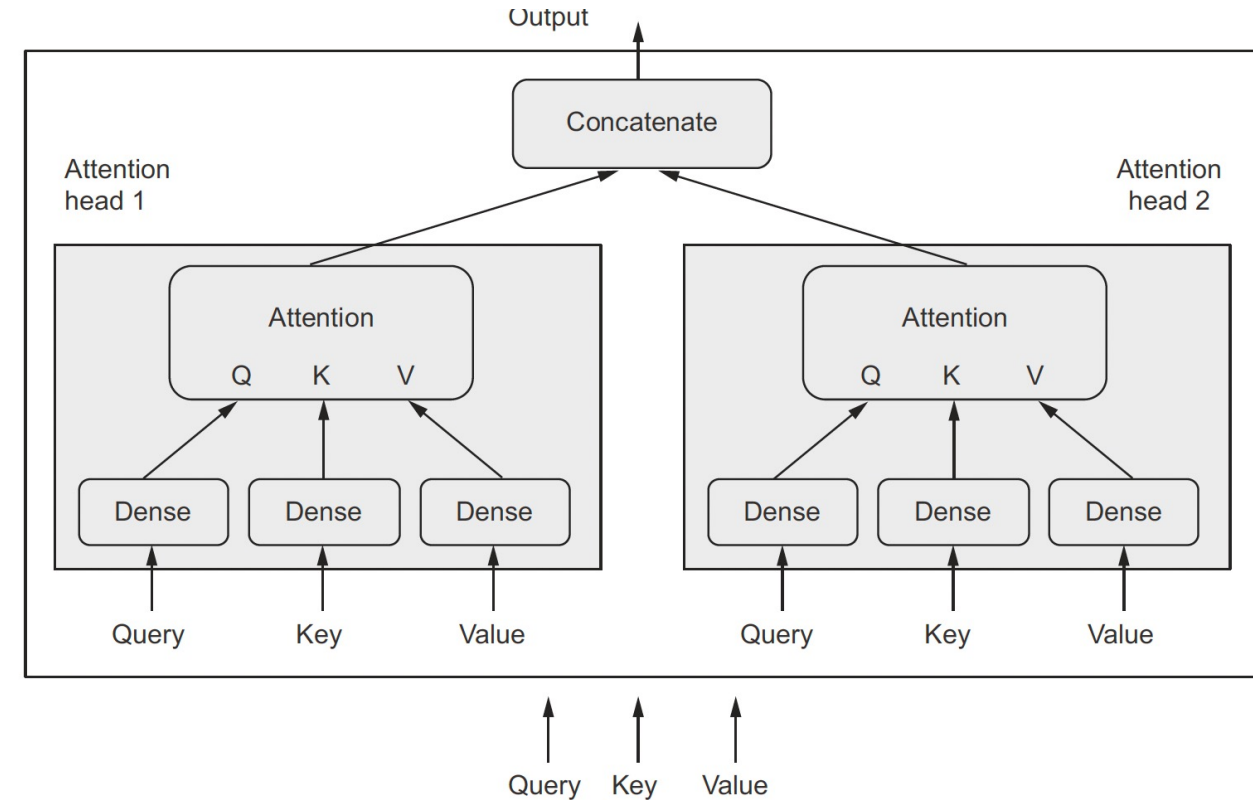


- $Attention(Q, K, V) = softmax\left(\frac{QK^T}{scaling}\right) V$



# Multi Head Attention

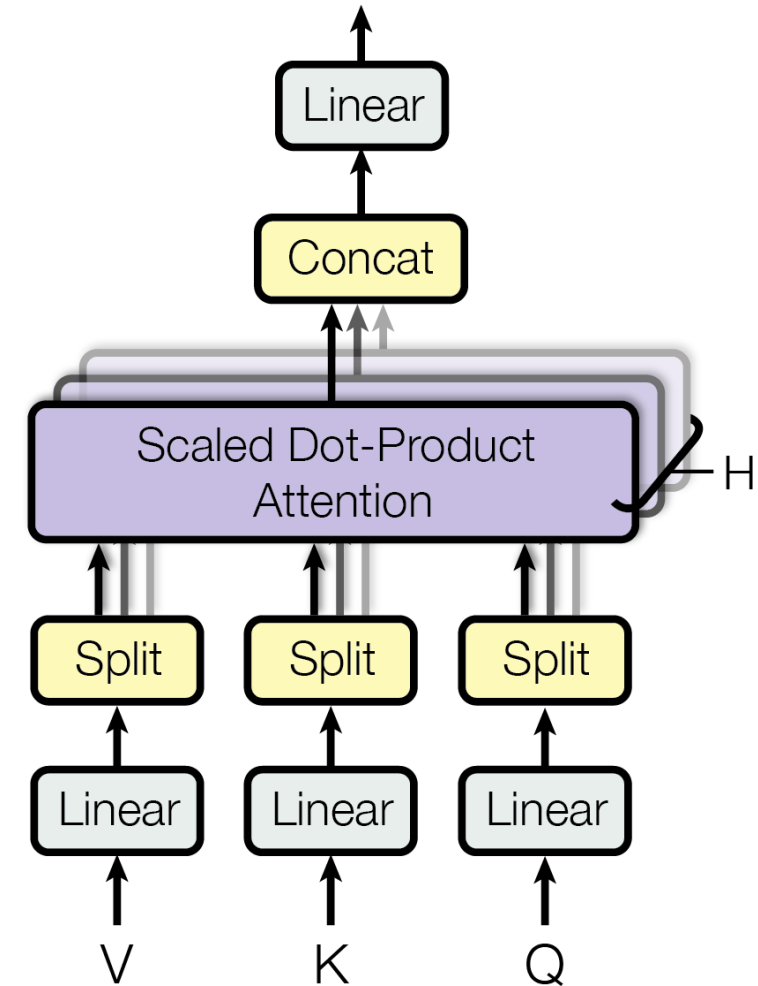
- In an attention head: The initial query, key and value are sent through three separate dense layers before self attention is applied
- Several heads are applied in parallel to an input sequence, making it a multi-head attention layer





# Benefit of Dense Layers in MHA

- This mechanism has two benefits
  - Dense layers allows learning within the attention layer
  - The output space is factored into three subspaces much like the depthwise separable convolution





# All Calculations: Summary

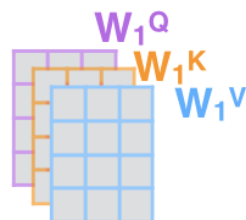
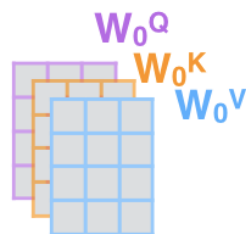
1) This is our input sentence\*

Thinking  
Machines

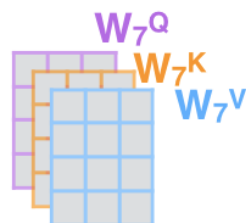
2) We embed each word\*



3) Split into 8 heads.  
We multiply  $X$  or  $R$  with weight matrices



...



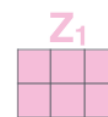
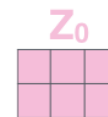
4) Calculate attention using the resulting  $Q/K/V$  matrices



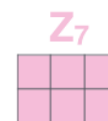
...



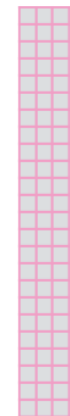
5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer



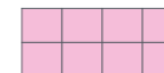
...



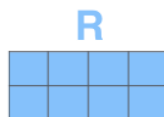
$W^O$



$Z$



\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one





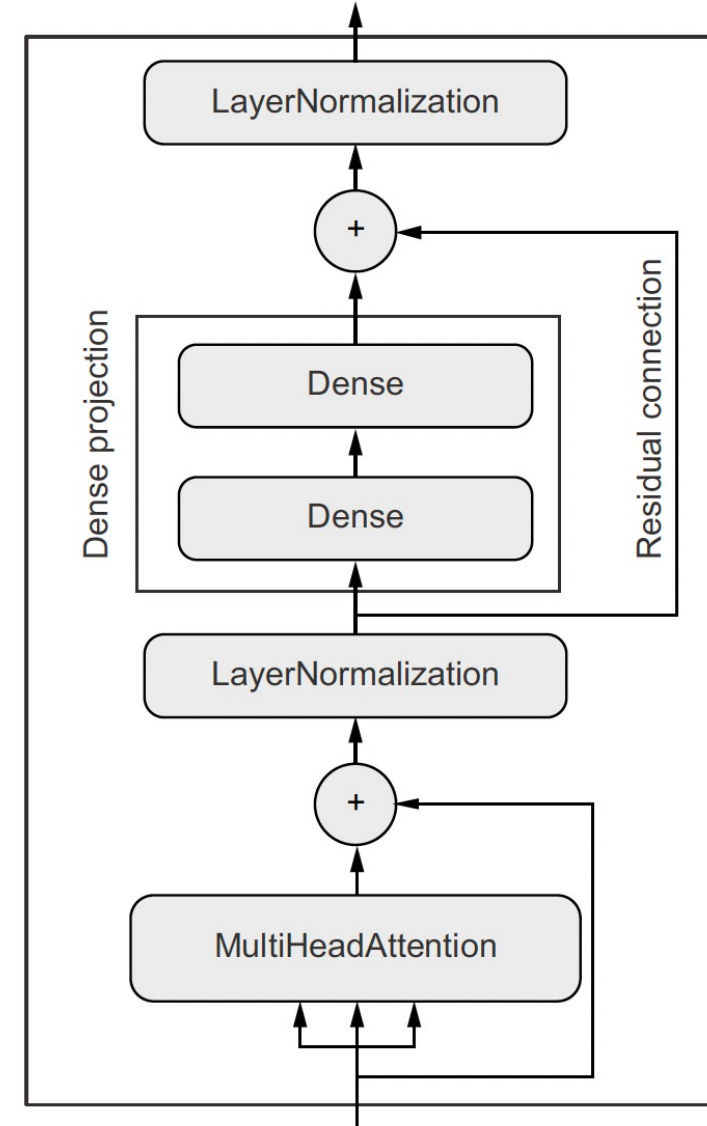
# Poll

## 1. Which of the following is FALSE

- Self Attention scores modify the word embeddings to be more context aware
- Attention scores can be calculated only for one document with itself
- Attention mechanism tells a model to pay more importance to certain parts of the input sequence
- For a sequence with  $n$  tokens,  $n^2$  self attention scores are needed.

# Transformer Encoder

- Adding dense layers before the attention mechanism helped, so why not after?
- Yes, this leads to what is called as the transformer encoder
- While at it, add in layer normalization and residual connections
- Adding a dense layer is also called as a Dense projection
- This architecture is called as a transformer encoder that allows learning from text corpora well
- Transformer Encoder has to be written as a subclassed Layer in Keras
- Keras NLP has an inbuilt layer [https://keras.io/api/keras\\_nlp/layers/transformer\\_encoder/](https://keras.io/api/keras_nlp/layers/transformer_encoder/)





# Batch vs 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.



# The Missing Ingredient

- Wait a minute for one minute – Aren't we learning deep sequential models? Where is the multi head attention layer using sequence information?
- Well, it does not
- Models with Multi Head Attention (as seen so far) will process text data as a set of words
- Now we add in the missing ingredient – Positional Encoding to inject order information to the transformer architecture

	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

- Add the word's position in the sentence to each of the word embeddings
- Word Embeddings = Usual word vector (without seq info) + position of the word in the sequence
- Simple approach is to give an integer, but neural networks like to have normalized numbers, so a transformation must take place
- The original paper proposed a sinusoidal position embedding
- A simpler and more powerful approach is to make the position embedding similar to word embedding and make it learnable
- After that let the Multi Head Attention act on the combined word and position embedding

# Positional Encoding

- RNNs processed words at each time step.
- How to we add the notion of position in case of transformers?
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.

$\mathbf{p}_i \in \mathbb{R}^d$ , for  $i \in \{1, 2, \dots, n\}$  are position vectors

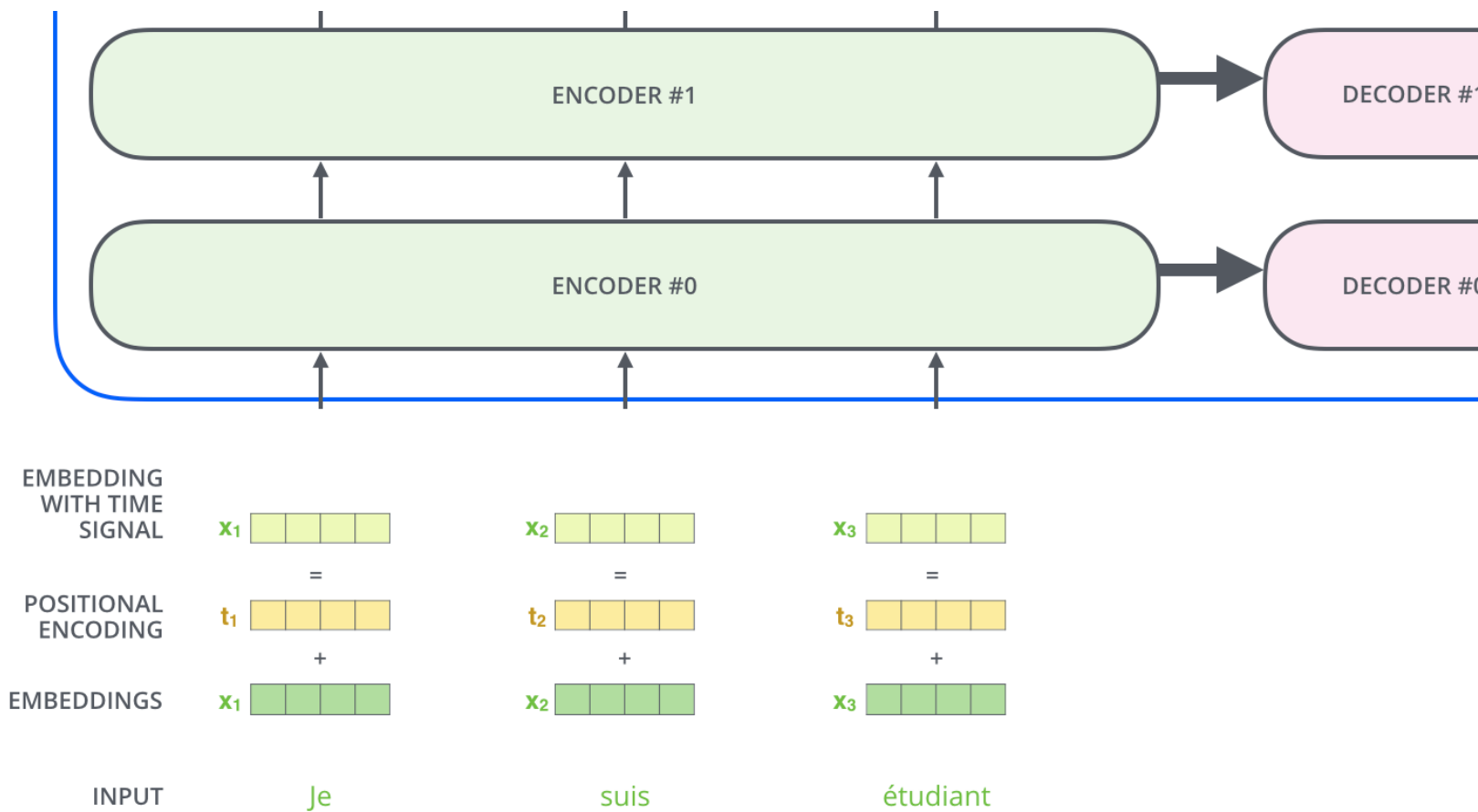
- $\mathbf{x}_i$  is the embedding of the word  $i$ . The position information is added with the embedding.

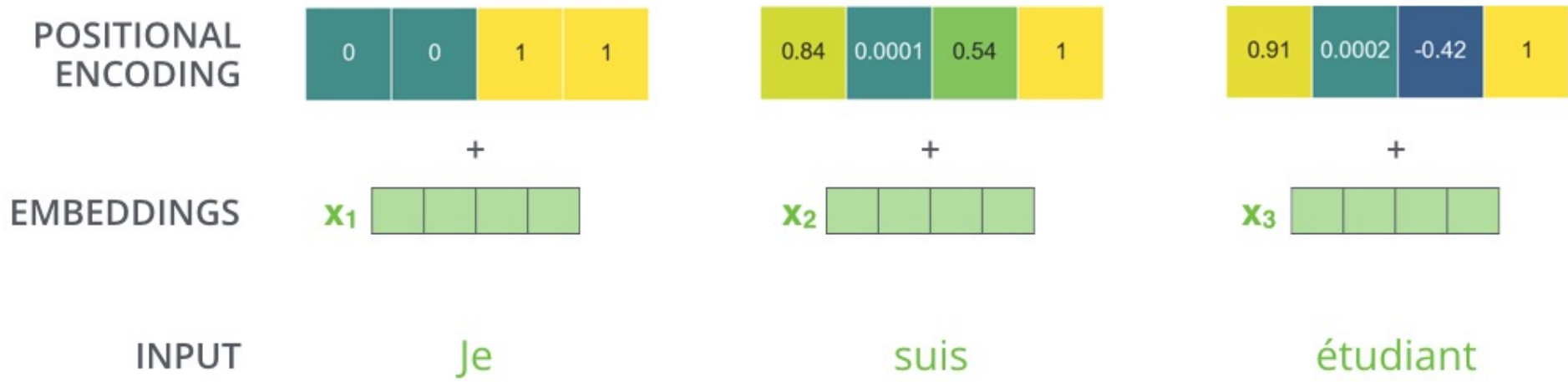
$$\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{p}_i$$

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$



# Positional Embedding

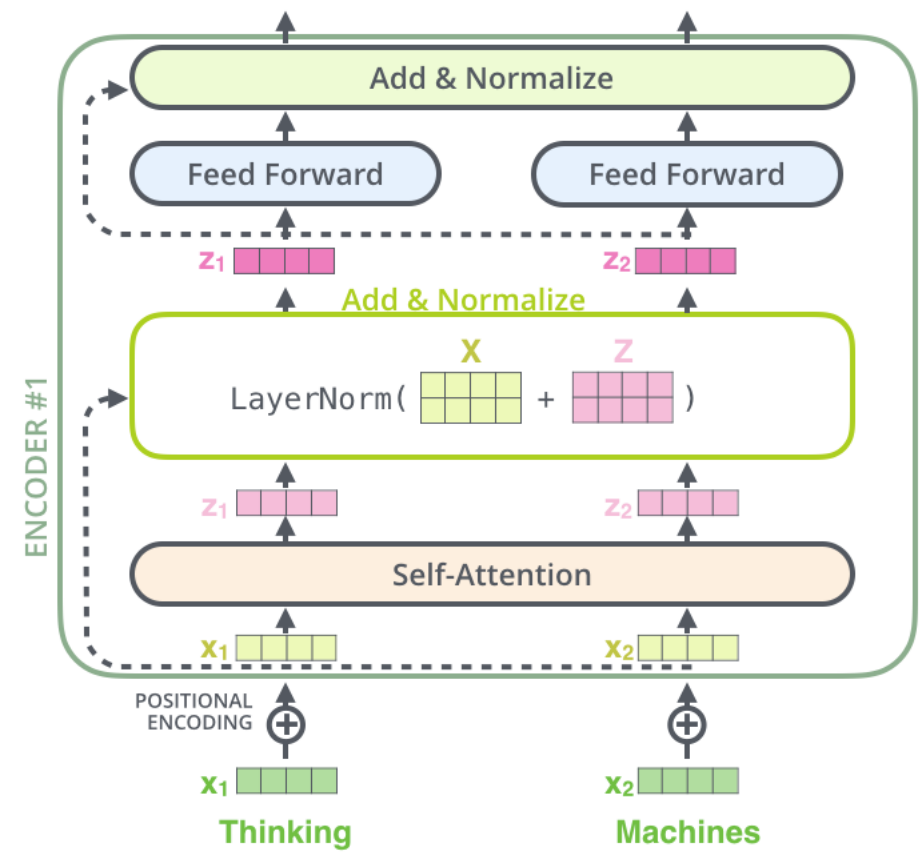




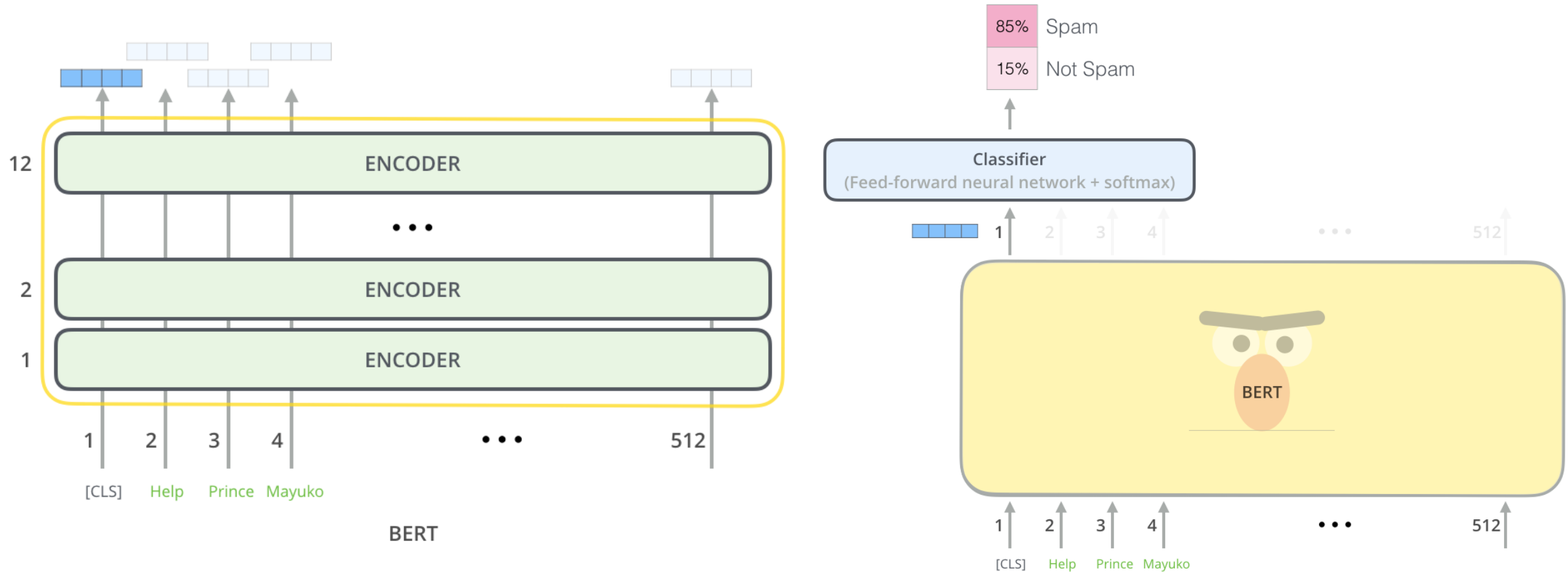




# Visualization of Residual Connection and Layer Normalization – Full Transformer Encoder



# Bidirectional Encoder Representation with Transformers (BERT) for Text Classification





# Poll

1. Layer Normalization is not used in Transformer Encoder
  - True, False
2. In Batch Normalization, normalization is done across the features
  - True, False
3. Positional Encoding is learnt as an embedding layer
  - True, False



# Outline for Week 02

- Part 01: Sequence Modeling
  - NLP as a sequence modelling task
  - Names of recurrent architectures and introduction
  - Details in an additional slide deck on RNN – not included in the main content
    - Other Seq Tasks (Time Series Modeling; Not NLP)
    - Additional notebooks on RNN and LSTM (not compulsory)
- Part 02: Attention Mechanism and Transformer Encoders
  - Key, Query, Value
  - Multi Head Attention
  - MHA as text representation learners for use in discriminative tasks
  - Positional Embedding
  - Trick of the trade: Layer Normalization
  - Assignment on Attention Mechanism

# Keras Code

```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(1)
])

model = keras.models.Sequential([
    keras.layers.LSTM(20, return_sequences=True, input_shape=[None, 1]),
    keras.layers.LSTM(20, return_sequences=True),
    keras.layers.TimeDistributed(keras.layers.Dense(10))
])
```

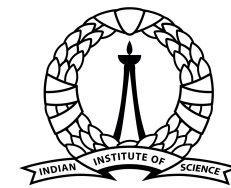
# Timeseries or Sequential Modeling

- Timeseries: Data obtained by measurements in regular intervals
  - Sequential data indexed by time
  - Regular – time difference between two measurements is same
    - Daily stock price, hourly electricity, weekly sales
  - Irregular – time difference between two measurements is not the same
    - ATM withdrawal
- Arises in both natural and socio-technical systems
- In the natural sciences, these are the outcome of a dynamical system
- A dynamical system usually has its own periodic cycles, trends, regimes
  - Study of chaos and models for the same are important subjects in natural sciences!



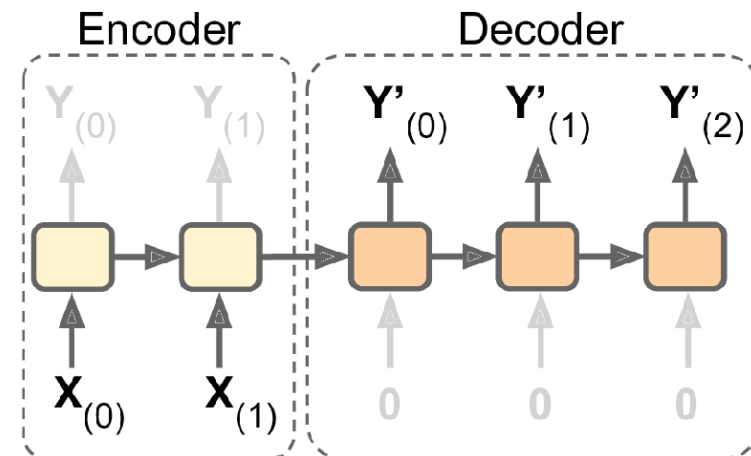
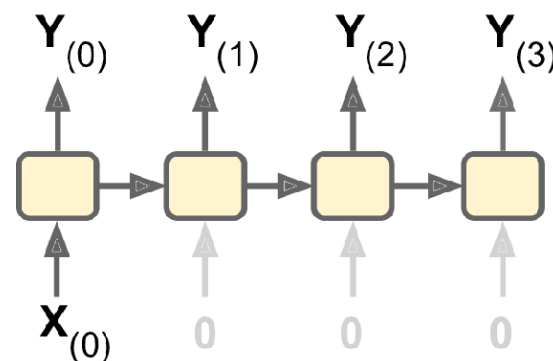
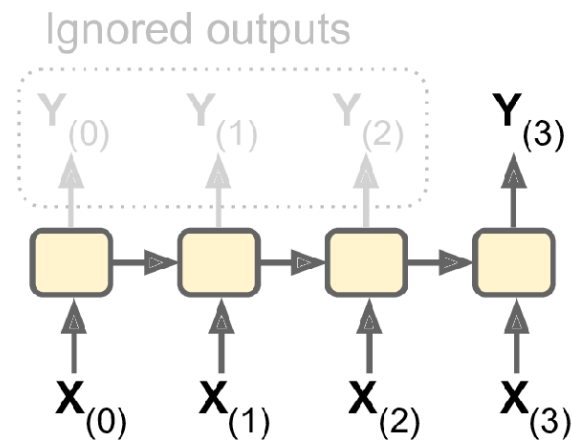
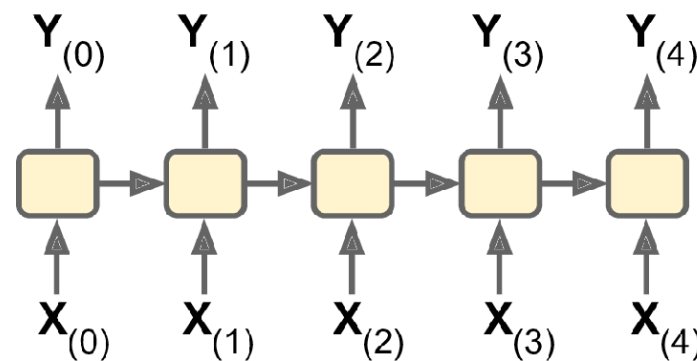
# Essential Sequence Modelling Tasks

- Forecasting – What happens next in a sequence
  - Next season's sale
  - Weather for the weekend
  - Electricity consumption at 8 pm
  - Supervised Regression
- Classification – Assign a categorical label to a timeseries
  - TS of visitor activity on a website – Bot or Human?
  - Is the rainfall timeseries El Nino or La Nina?
  - Supervised Classification [Sequence to Label]
- Event Detection – Identify a specific expected (known) event within a continuous data stream
  - “Ok Google” from an ambient listener on mobile
  - Onset of summer monsoon
  - Supervised Classification [Sequence to Sequence-Label]
- Anomaly Detection – Identify any unusual activity in a stream of data
  - Attacker on the network?
  - Unsupervised learning
  - Cannot train on specific anomalies (then it is event detection, done via supervised learning)



# Sequence Modeling Tasks: Data view

- Seq-to-Seq
  - Input: Sequence of words
  - Output: Context Modified Sequence of Words
- Seq-to-Vec
  - Input: sequence of words in a movie review
  - Output: sentiment [positive, neutral, negative]; [+1,0,-1]
- Vec-to-Seq
  - Input: An image
  - Output: Caption [sequence of words describing the image]
- Seq-to-Vec + Vec-to-Seq
  - Input: English
  - Output: Hindi







# Timeseries Analysis: Data Cleaning

- Timeseries data representation is highly domain specific
- The whole field of signal processing is in a sense timeseries analysis
- The whole field of geoscience research is in a sense timeseries analysis
- The mathematical tools remain the same for modelling
- But data representation varies widely
- Fourier transform is common in several tasks
- Aside: Statistical timeseries models (SARIMAX and variants) are also powerful tools in your arsenal. Often these are the baseline models that allows us to justify use (or no use) of deep models

# Hands-On Tutorial

- `keras.utils.timeseries_dataset_from_array()`
  - [https://www.tensorflow.org/api\\_docs/python/tf/keras/utils/timeseries\\_dataset\\_from\\_array](https://www.tensorflow.org/api_docs/python/tf/keras/utils/timeseries_dataset_from_array)
- `sklearn.model_selection.TimeSeriesSplit`
  - [https://scikit-learn.org/stable/modules/generated/sklearn.model\\_selection.TimeSeriesSplit.html](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html)
- Before building deep learning models it is important to build a baseline
  - Non-ML
    - Predict the mean of the timeseries!
    - Always predict class A
    - Often these common-sense non-ML baselines are difficult to beat!
  - Basic ML
    - ARIMA
    - Simple Dense Network, 1D convolutional network
    - Many times, simple ML models do worse than non-ML baselines – Beware!

# Word of Caution

- Many will be tempted to use the sequential modeling approach to the stock market
- Beware – trading is all about information arbitrage. Past is not a good indicator of the future
- Neural Networks (and ML) are good for situations where past is a good indicator of the future
- We are not responsible if you lose money by using a stock market predictor made with RNNs!