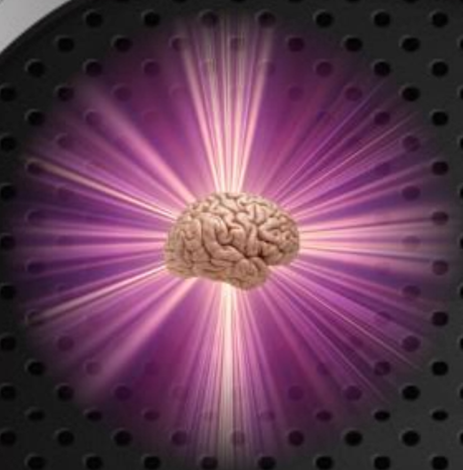
AGH University of Science and Technology
Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering
Department of Biocybernetics and Biomedical Engineering



Computational Intelligence

1D Convolutions and Recurrent Layers vs Transformers for Sequence Processing

Adrian Horzyk
horzyk@agh.edu.pl

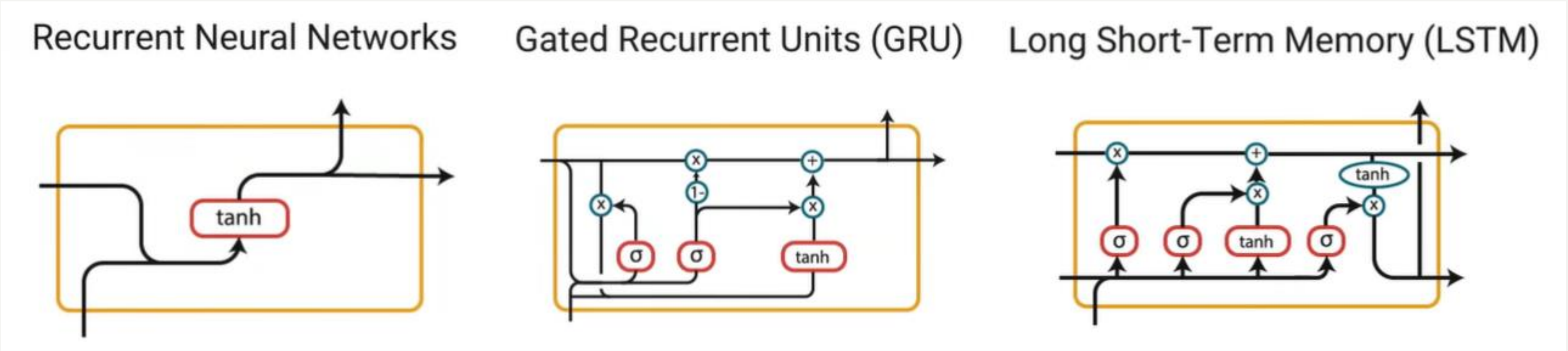
Google: Adrian Horzyk

Sequence Processing

What are the differences in sequence processing using different approaches?

Recurrent Networks for Sequences

Sequencial data are usually processed using recurrent neural networks (RNNs) of various kinds (e.g. GRU or LSTM):



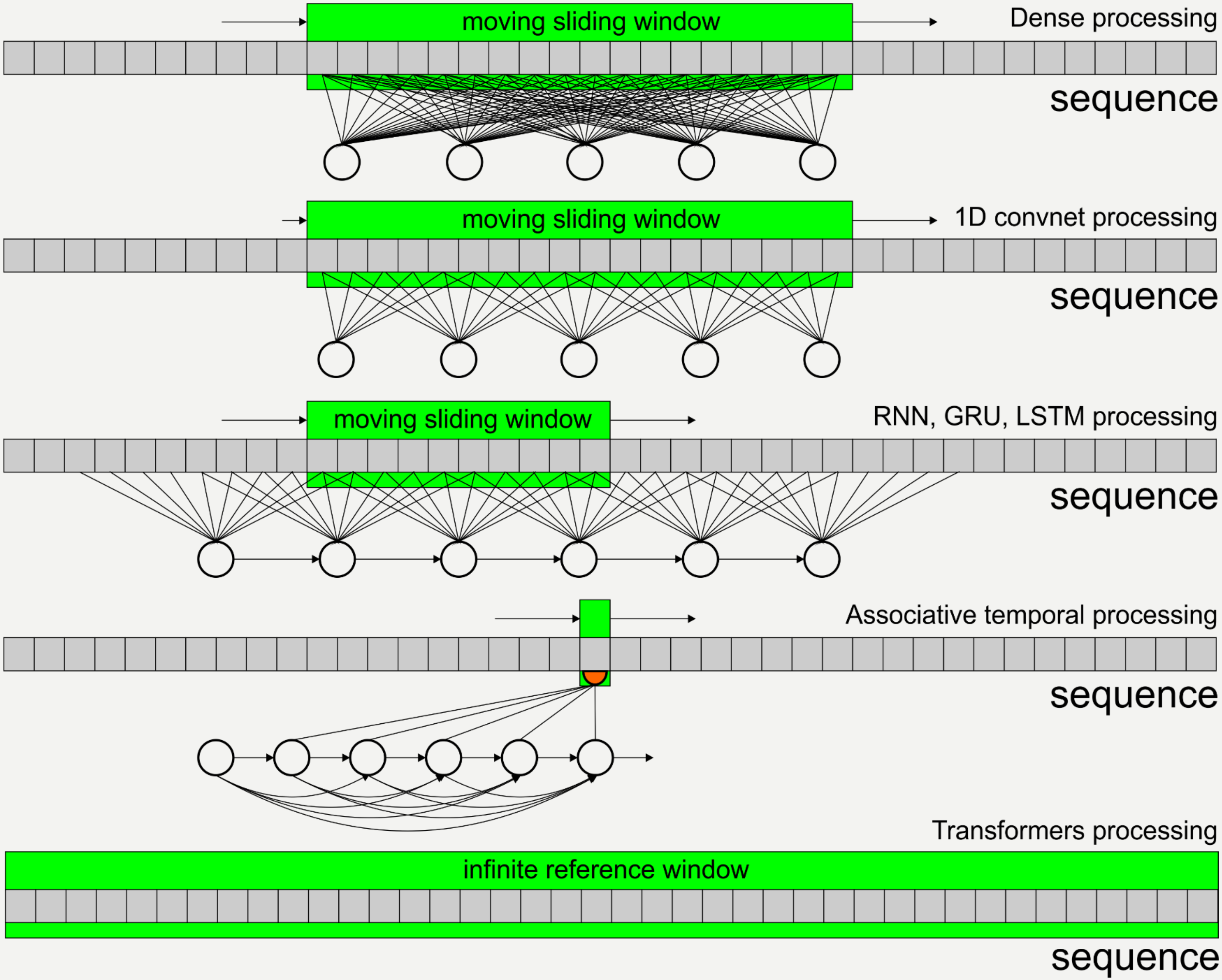
but we can also use many other approaches.

Sometimes we can also use convnets when the data sequence is not so important as the elements used in these sequences are.

One of such problems is the IMDB sentiment classification task where the positive or negative classification depends more on the used words in the sentences than on the sequential relationships.

Sequence Processing

The sequences can be processed using different approaches:



1D Convnets for Sequence Processing

How can we use 1D convnets for sequencial data processing using Keras?

1D Convnet Layers in Keras

In Keras, we use a 1D convnet via the Conv1D layer, which takes as input 3D tensors with shape (samples, time, features) and also returns similarly-shaped 3D tensors.

The convolution window is a 1D window on the temporal axis.

1D convnets are structured in the same way as their 2D counter-parts and have a very similar interface to Conv2D. They consist of a stack of Conv1D and MaxPooling1D layers, eventually ending in either a global pooling layer (GlobalMaxPooling1D) or a Flatten layer, turning the 3D outputs into 2D outputs, allowing to add one or more Dense layers to the model, for classification or regression.

We can afford (taking into account the computing time) to use larger convolution windows with 1D convnets.

Indeed, with a 2D convolution layer, a 3x3 convolution window contains 3*3 = 9 feature vectors, while with a 1D convolution layer, a convolution window of size 3 would only contain 3 feature vectors.

Thus, we can easily afford 1D convolution windows of size 5, 7, 9, or even more.

1D Convnet Network for IMDB

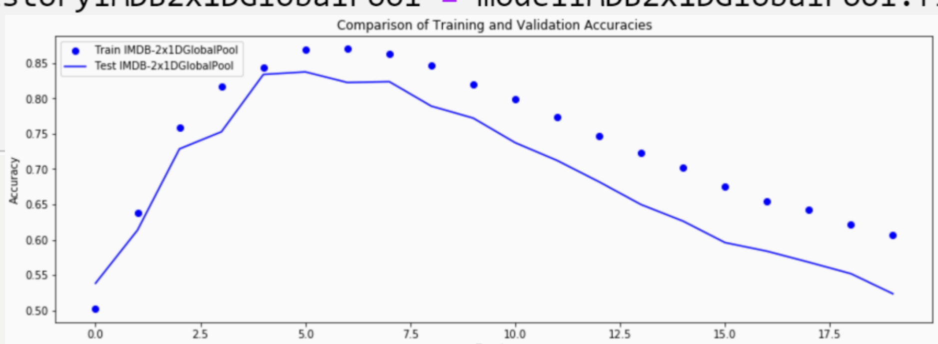
Let's build a simple 2-layer 1D convnet applied to the IMDB sentiment classification task that we are already familiar with:

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

modelIMDB2x1DGlobalPool = Sequential()
modelIMDB2x1DGlobalPool.add(layers.Embedding(max_features, 128, input_length=max_len))
modelIMDB2x1DGlobalPool.add(layers.Conv1D(32, 7, activation='relu'))
modelIMDB2x1DGlobalPool.add(layers.MaxPooling1D(5))
modelIMDB2x1DGlobalPool.add(layers.Conv1D(32, 7, activation='relu'))
modelIMDB2x1DGlobalPool.add(layers.GlobalMaxPooling1D())
modelIMDB2x1DGlobalPool.add(layers.Dense(1))

modelIMDB2x1DGlobalPool.summary()

modelIMDB2x1DGlobalPool.compile(optimizer=RMSprop(lr=1e-4),
                                loss='binary_crossentropy',
                                metrics=['acc'])
historyIMDB2x1DGlobalPool = modelIMDB2x1DGlobalPool.fit(x_train, y_train,
                                                        epochs=20,
                                                        batch_size=128,
                                                        validation_split=0.2)
```



Machine Learning

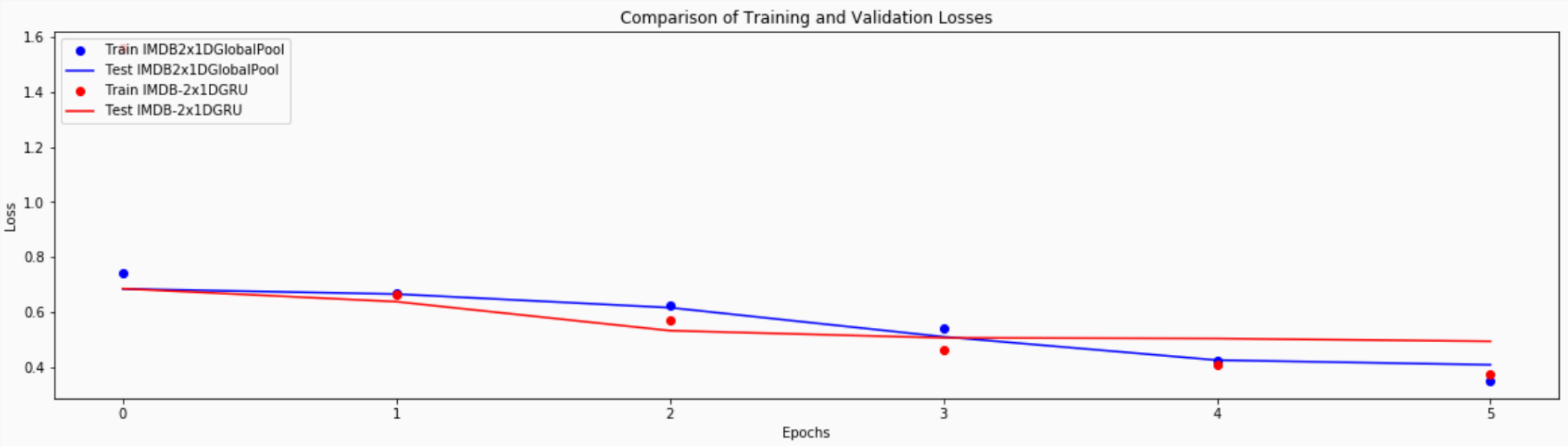
We can combine 1D convolutional layers with GRU or LSTM layers:

```
modelIMDB2x1DGRU = Sequential()
modelIMDB2x1DGRU.add(layers.Embedding(max_features, 128, input_length=max_len))
modelIMDB2x1DGRU.add(layers.Conv1D(32, 7, activation='relu'))
modelIMDB2x1DGRU.add(layers.MaxPooling1D(5))
modelIMDB2x1DGRU.add(layers.Conv1D(32, 7, activation='relu'))
modelIMDB2x1DGRU.add(layers.GRU(32, dropout=0.05, recurrent_dropout=0.05))
modelIMDB2x1DGRU.add(layers.Dense(1))

modelIMDB2x1DGRU.summary()

modelIMDB2x1DGRU.compile(optimizer=RMSprop(lr=1e-4),
                         loss='binary_crossentropy',
                         metrics=['acc'])
historyIMDB2x1DGRU = modelIMDB2x1DGRU.fit(x_train, y_train,
                                          epochs=6, # or 7 dependently on the network initializat
                                          batch_size=128,
                                          validation_split=0.2)

modelIMDB2x1DGRU.save(models_dir + 'IMDB_2x1DGRU.h5')
```



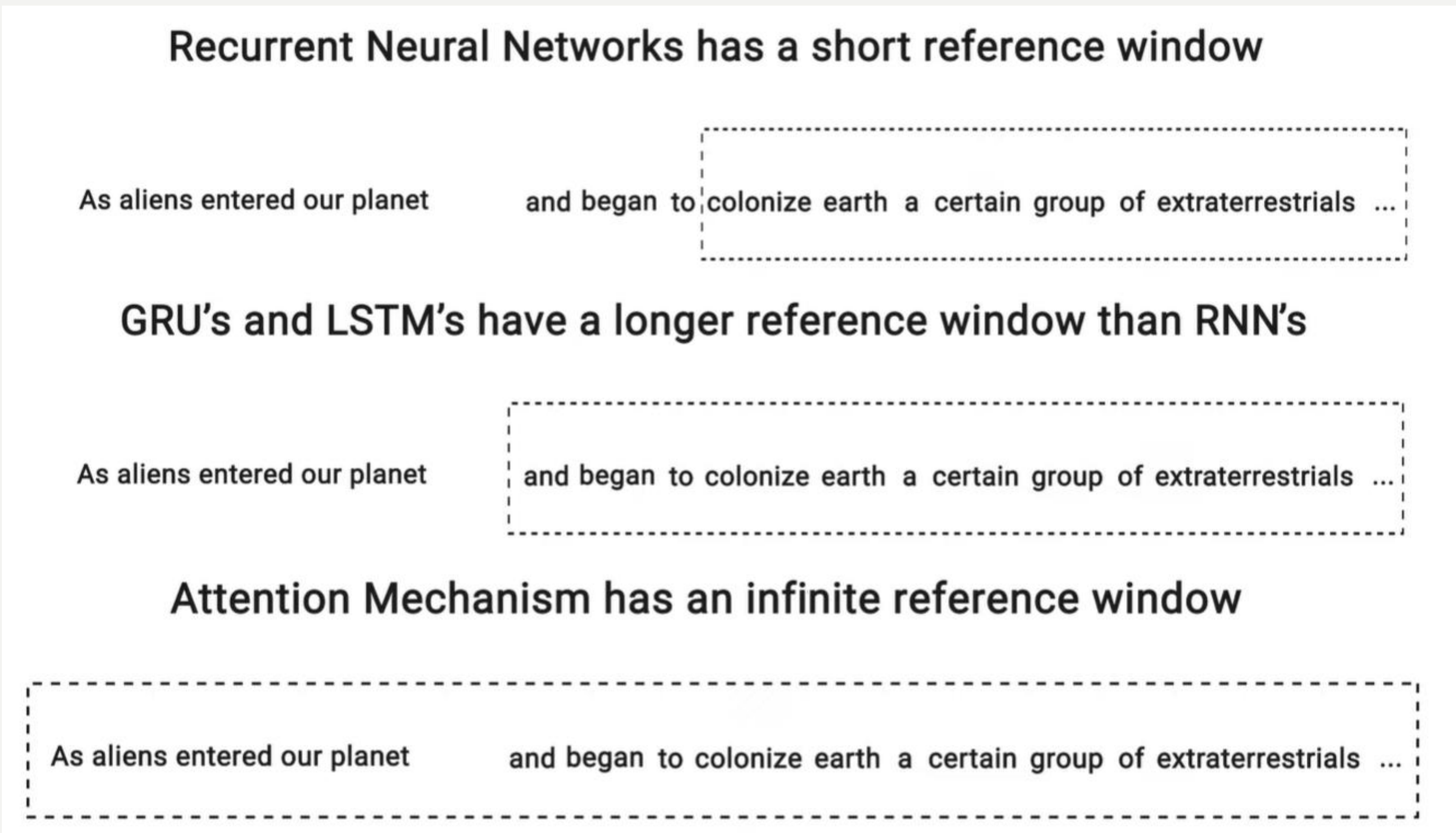
Attention Mechanism and Transformers

Attention is all you need, do you?

Recurrent Memory Limitations

Recurrent memories suffer from a limited size of the used reference windows:

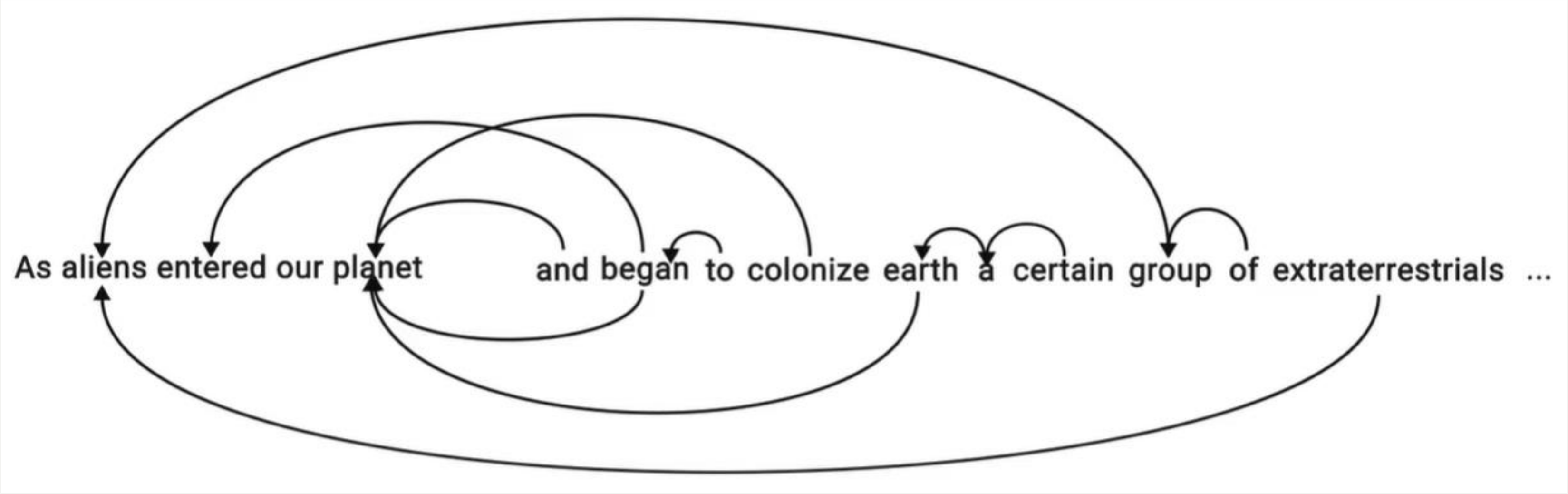
- **RNN can use only short reference windows,**
- **GRU and LSTM can use longer reference windows than RNN, but still limited,**
- **Attention Mechanism uses unlimited reference windows:**



Attention Mechanism

Attention Mechanism used by Transformers:

- **use an infinite reference window, so the context can be take from the entire text, not only from the short reference window as RNN allow for or long reference window as GRU or LSTM allow for.**
- **enables a transformer model to focus on all previous tokens that have been generated, so it does not suffer from short term memory.**

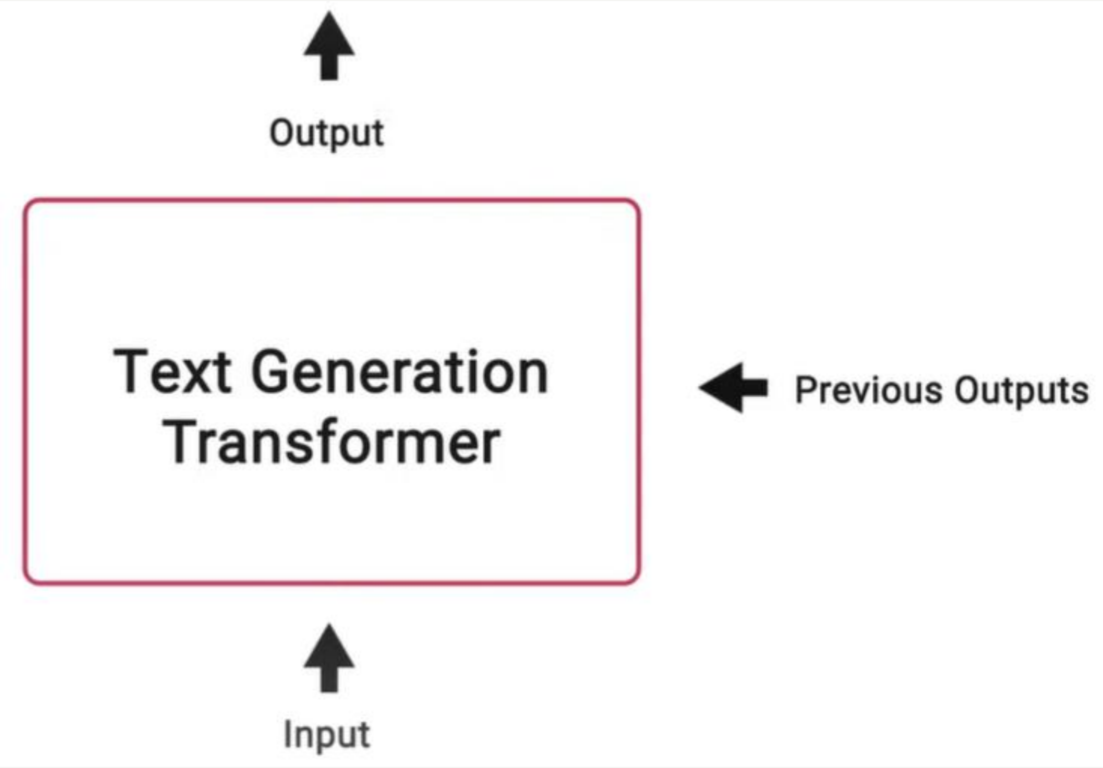


- **Our input:** *"As Aliens entered our planet"*
- **Transformer output:** *"and began to colonized Earth, a certain group of extraterrestrials began to manipulate our society through their influences of a certain number of the elite to keep and iron grip over the populace."*

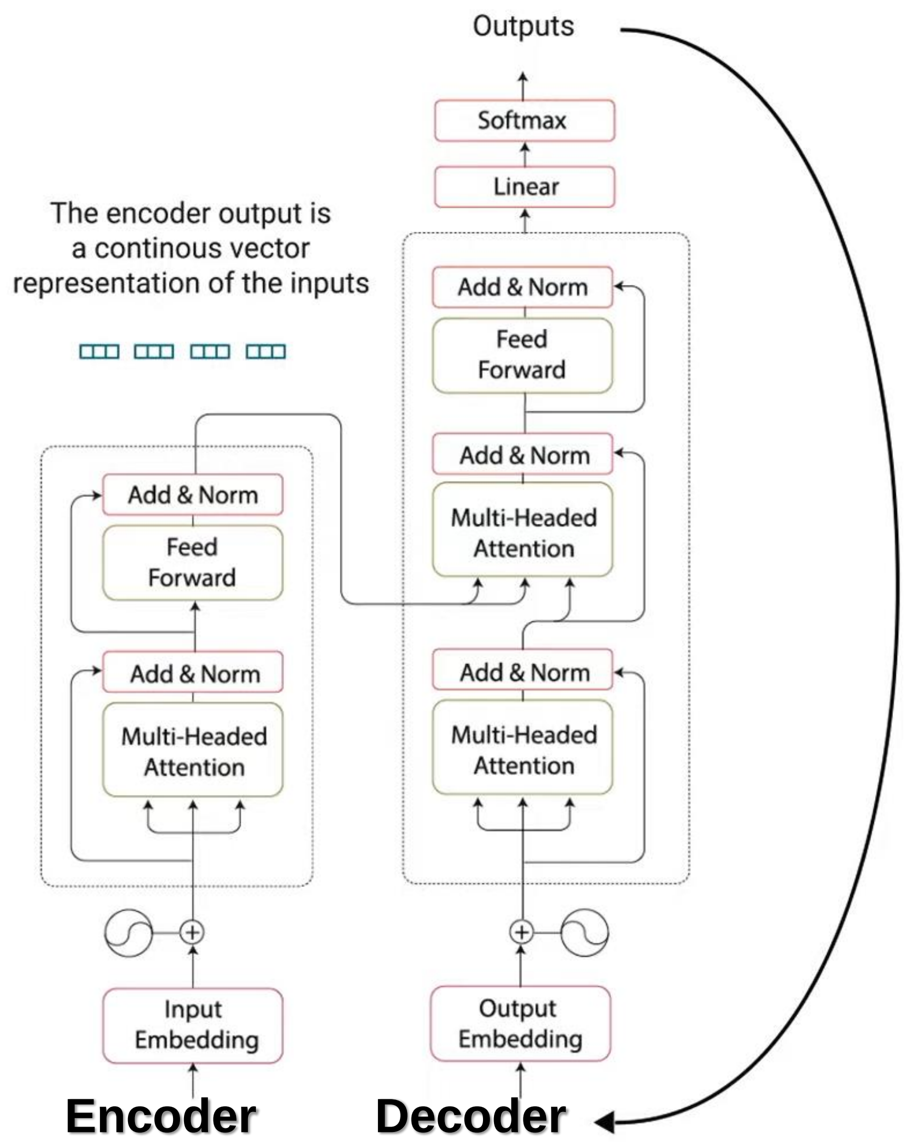
Transformers

[Transformers](#) are taking the NLP world by storm, breaking multiple NLP records and outperforming many previous kings of sequence processing models line RNN, GRU or LSTM:

- **Famous transformers models like BERT (Bidirectional Encoder Representation form Transformers), GPT or GPT2 (Generative Pre-Training).**



Transformers' Network and Model



Network consists of:

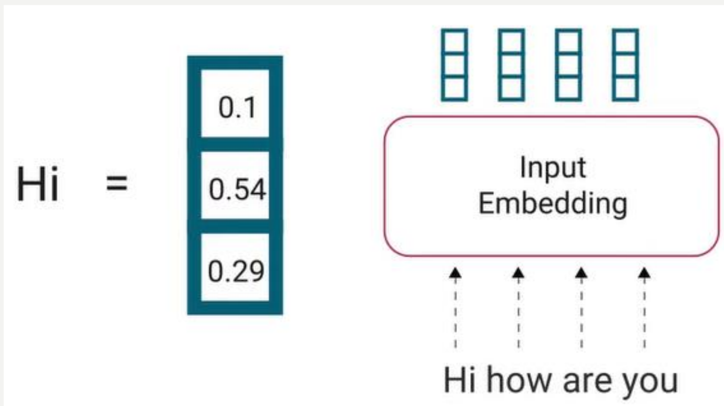
- **Encoder** that maps an input sequence into an abstract continuous representation that holds all the learned information of that input.

Feed previous outputs into the decoder recurrently until an "end of sentence" token, <end> is generated

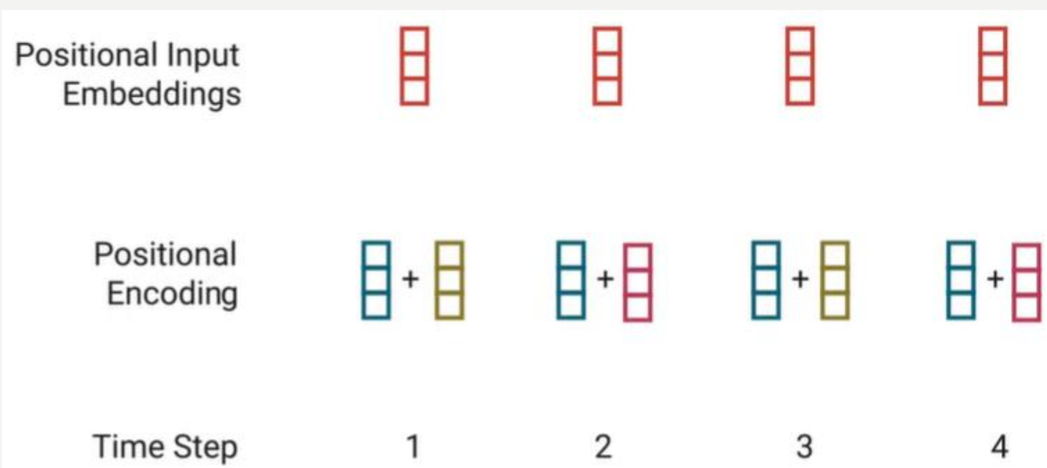
- **Decoder** that takes the continuous representation and step by step generates a single output while also being fed the previous output.

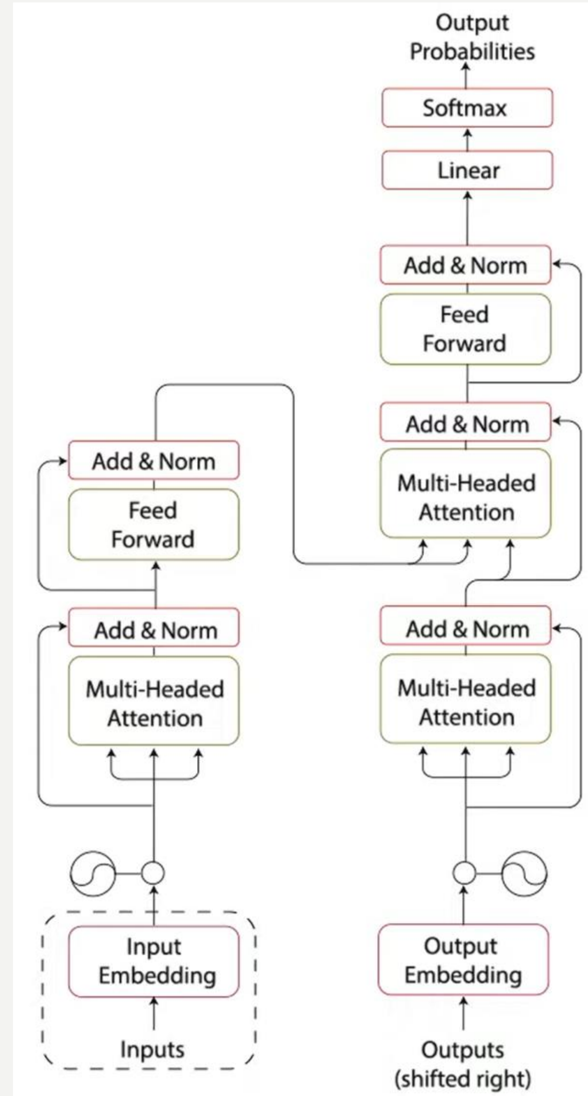
Input Positional Embedding

1. **The input is fed into a word embedding layer, which is like a lookup table to grab a learned continuous-values vector representation of each word:**



2. **The position about the word position is added to the representation of the word embeddings:**

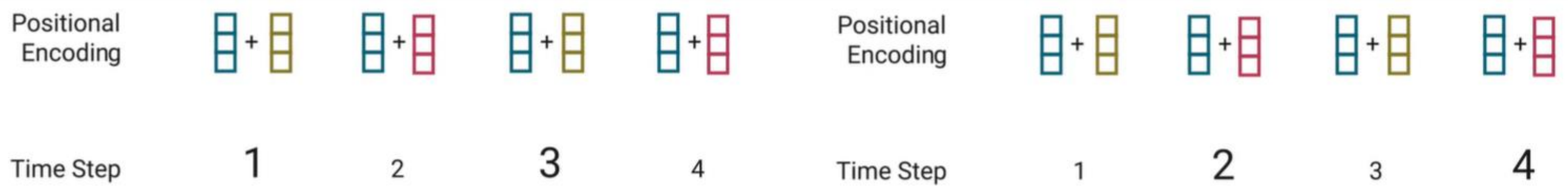




Positional Encoding and Embeddings

Transformers do not use recurrence so the information about the position of words must be added to the word input embeddings:

- For odd positions, we use cosine function
- For even positions, we use sine function



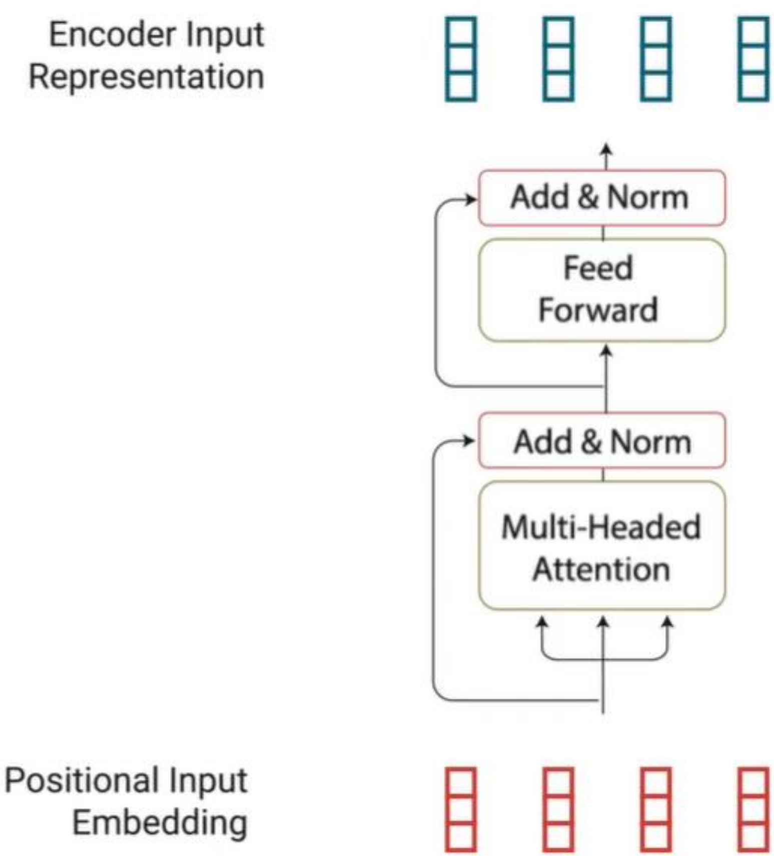
$$PE(pos, 2i+1) = cos(\frac{pos}{10000^{2i/dmodel}})$$

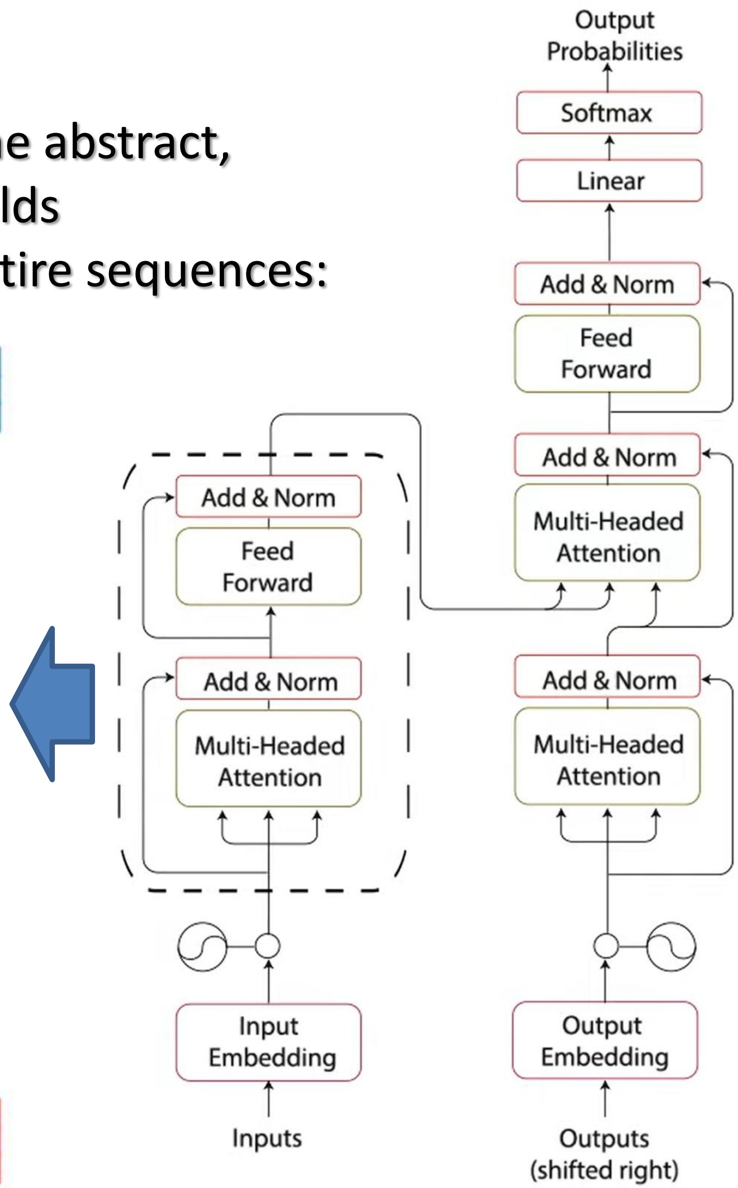
$$PE(pos, 2i) = sin(\frac{pos}{10000^{2i/dmodel}})$$

Encoder Layer Subnetwork

Encoder Layer Subnetwork:

- Maps all the input sequences to the abstract, continuous representation that holds the learned information for the entire sequences:

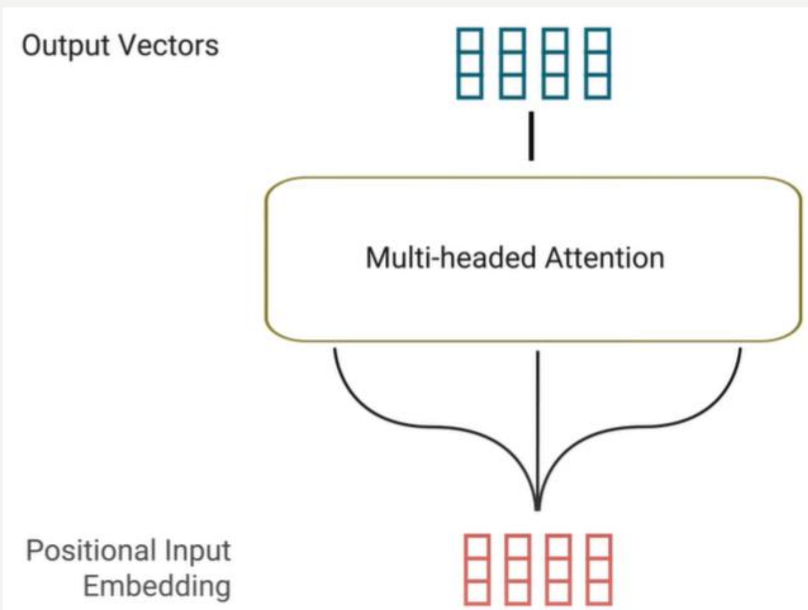


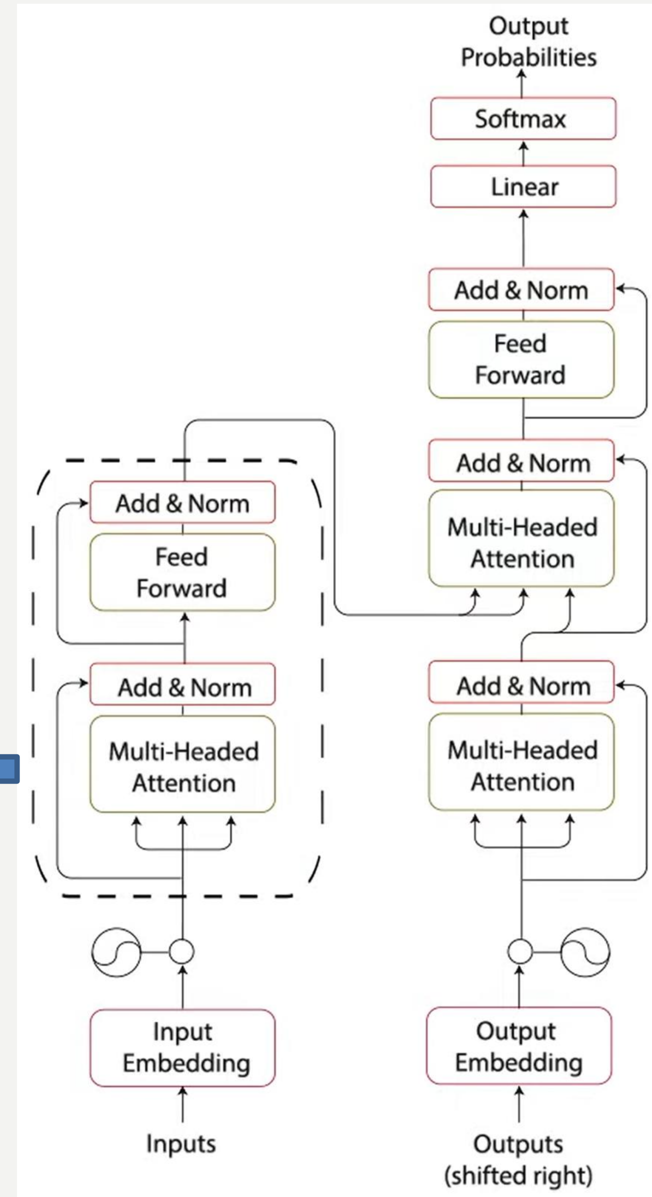


Multi-headed Attention Module

Multi-headed Attention Module:

- Is a network computing the attention weights from the input and producing the output vectors that encoded information on how each word should attend to each word in the input sequence.



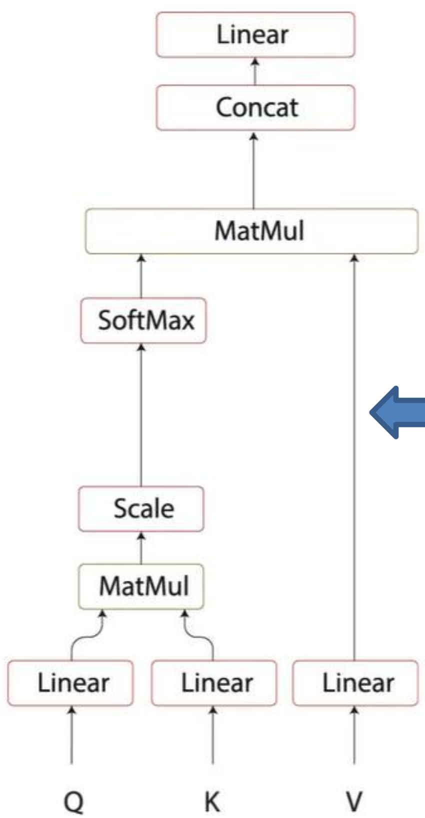


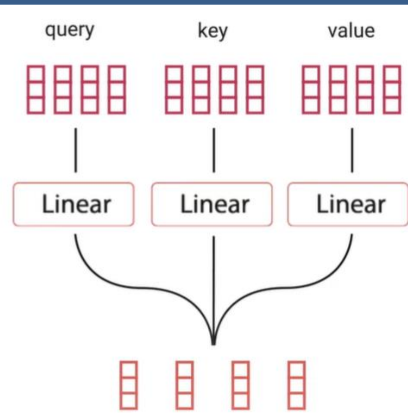
Multi-headed Attention

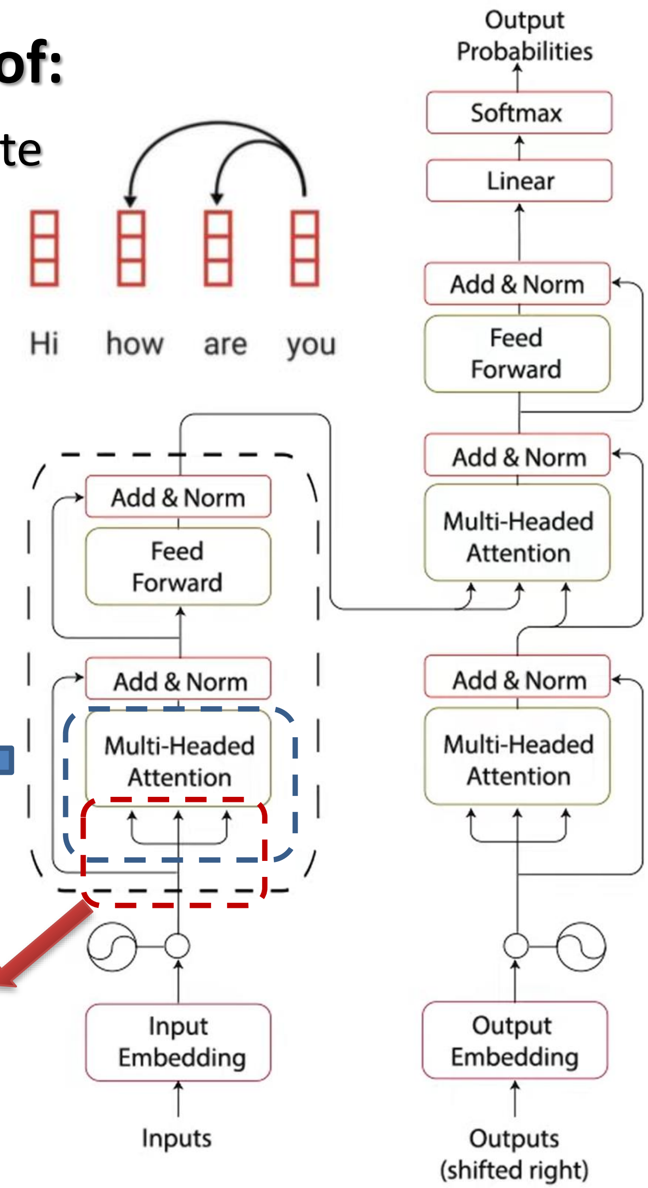
Multi-headed Attention consists of:

- Self-Attention which allows to associate each individual word in the input to the other words in the input:

The search engines usually map the query against a set of keys associated with candidates, from which the best (with the highest matching values) are finally presented:



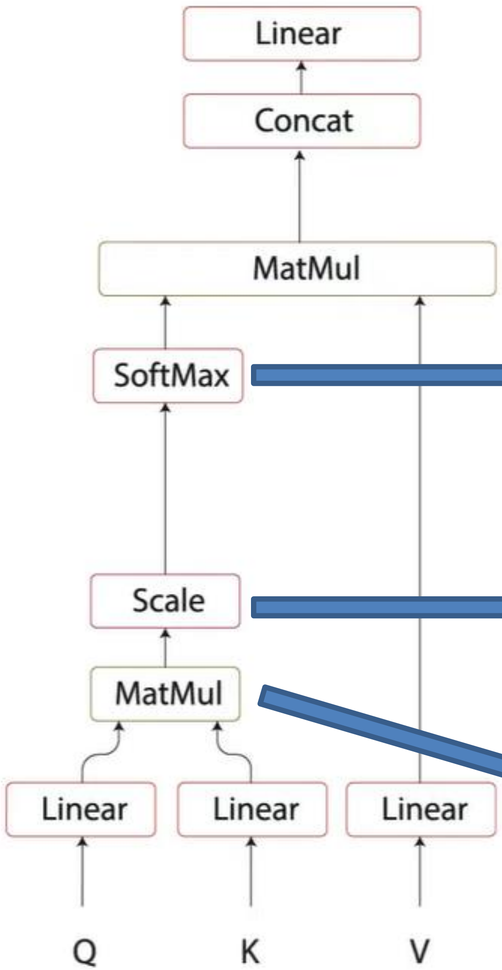


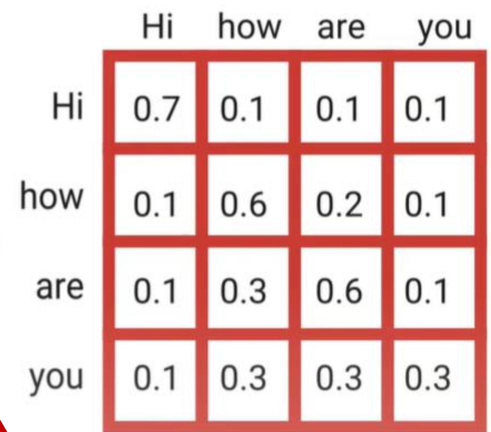


Scoring and Scaling

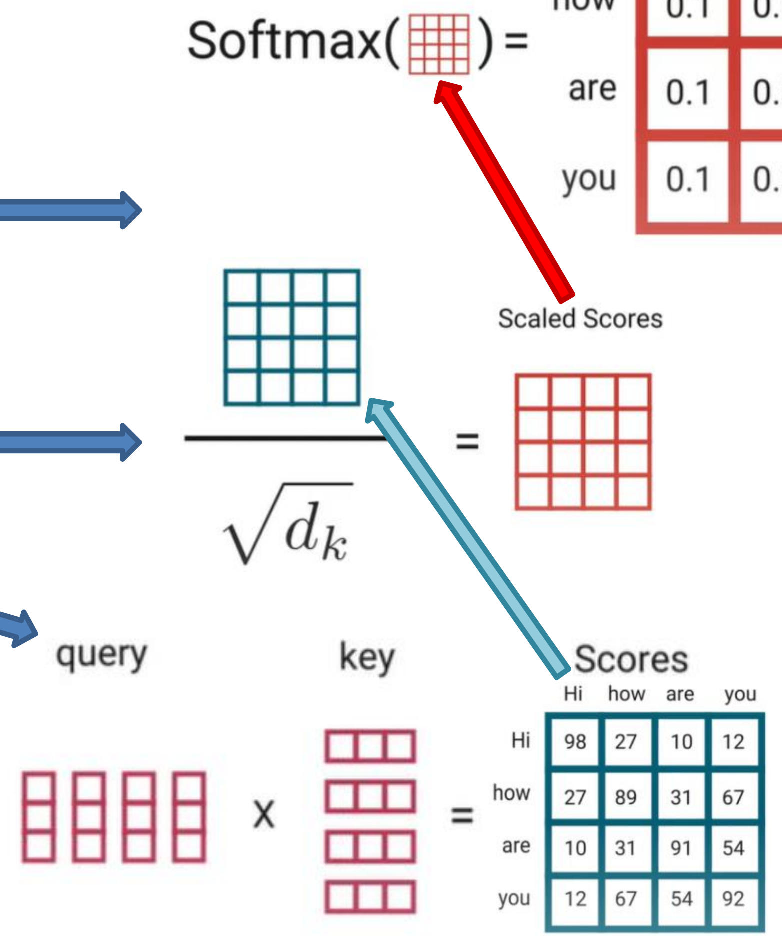
Multiplying → Scaling → SoftMax:

During the SoftMax, the highest scores are heightened and the lowest scores are depressed:



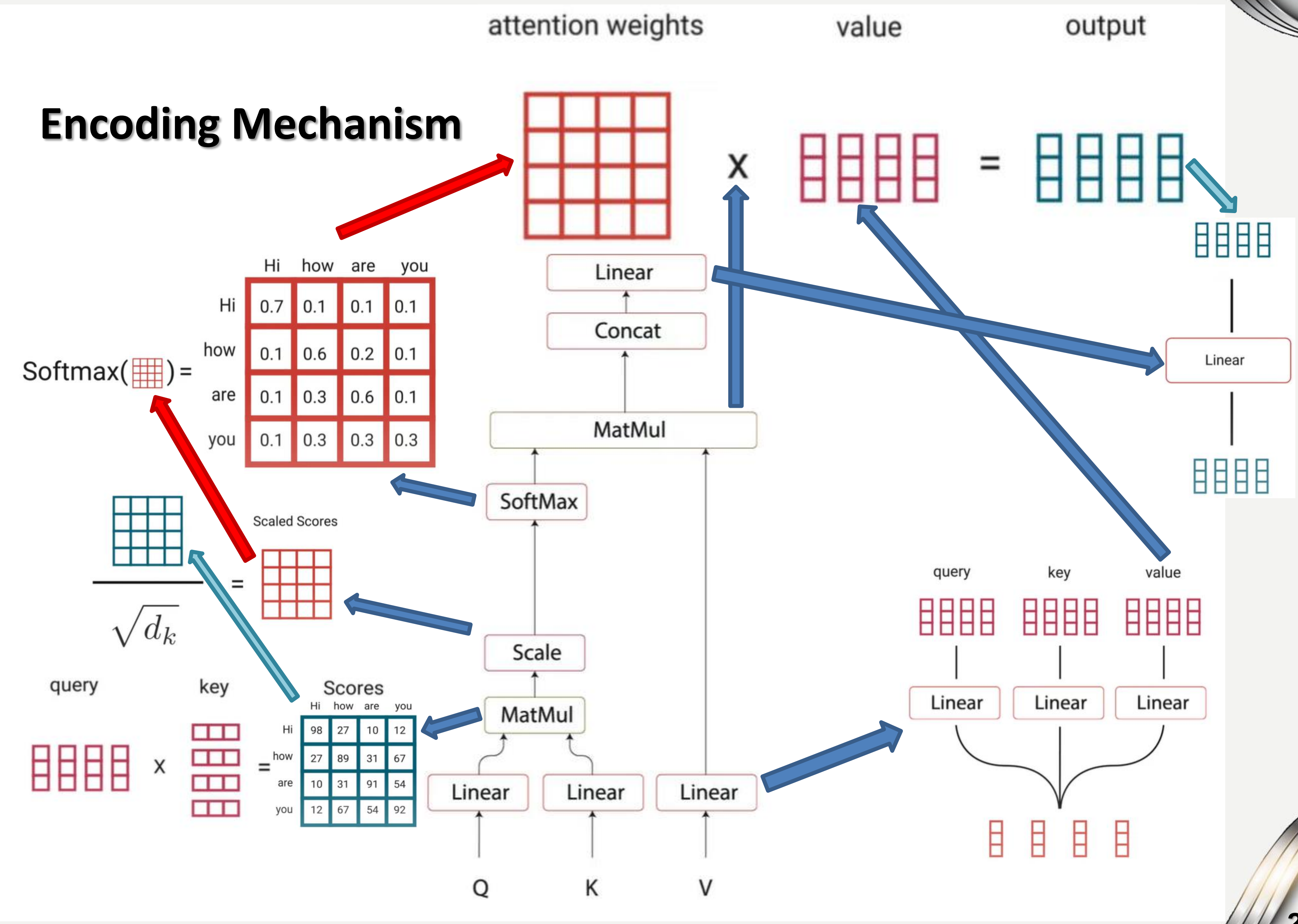


	Hi	how	are	you
Hi	0.7	0.1	0.1	0.1
how	0.1	0.6	0.2	0.1
are	0.1	0.3	0.6	0.1
you	0.1	0.3	0.3	0.3



	Hi	how	are	you
Hi	98	27	10	12
how	27	89	31	67
are	10	31	91	54
you	12	67	54	92

Multiplying the Attention Weights

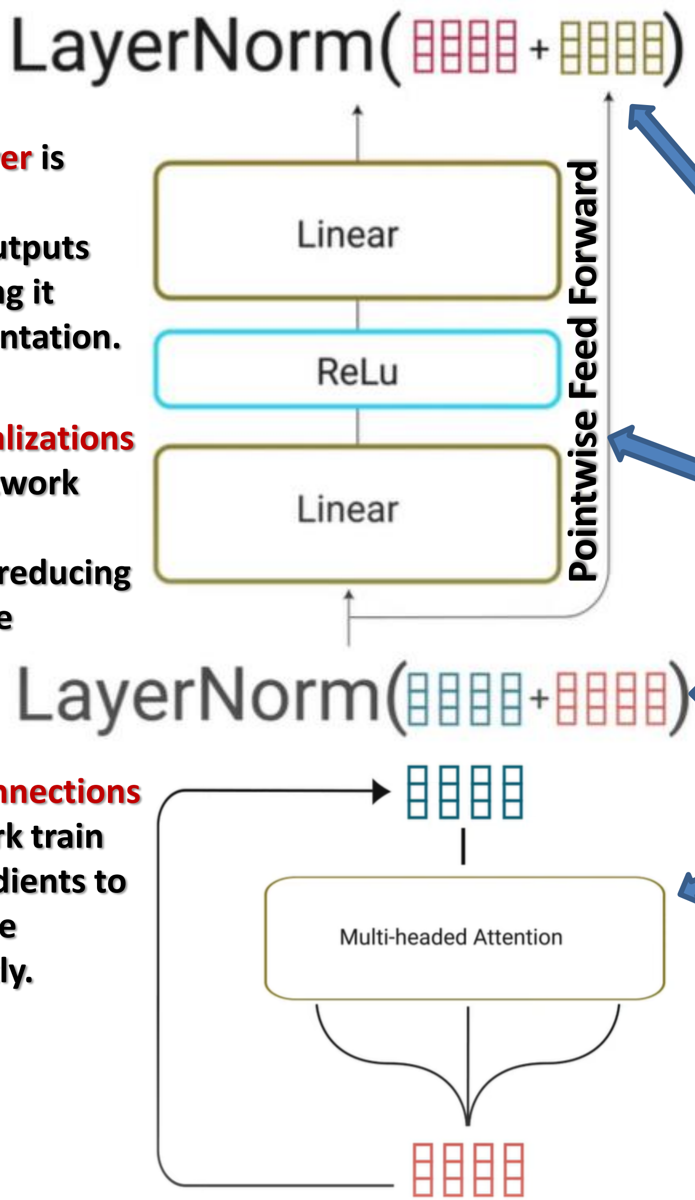


Residual Connections and Normalization

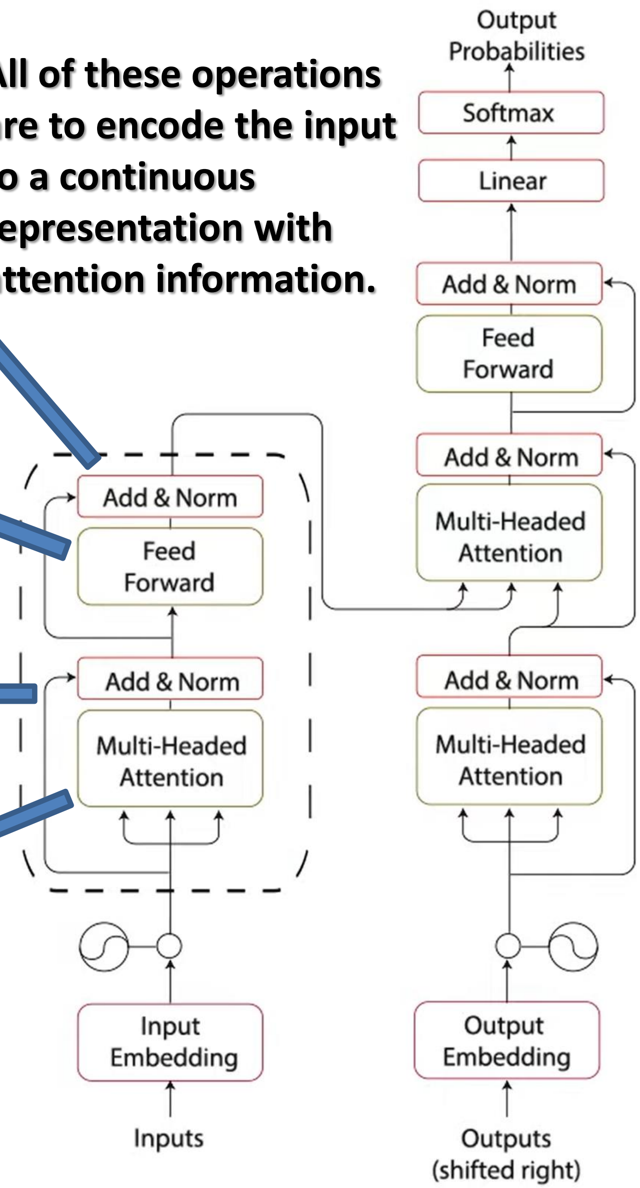
The pointwise feedforward layer is used to project the attention outputs potentially giving it a richer representation.

The layer normalizations stabilize the network which results in substantially reducing the training time necessary.

The residual connections help the network train by allowing gradients to flow through the networks directly.



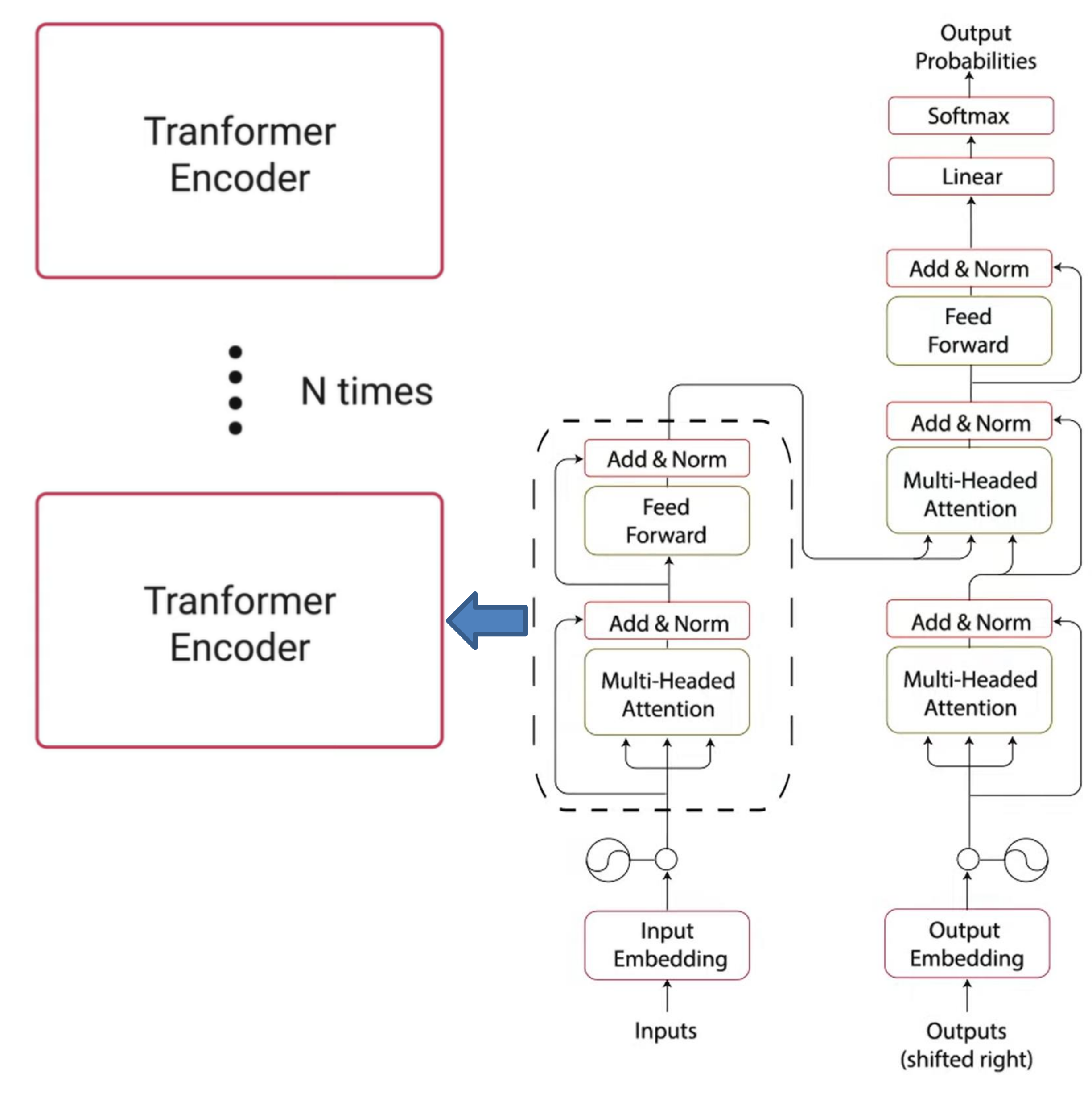
All of these operations are to encode the input to a continuous representation with attention information.



Residual Connections and Normalization

We can also stack encoders N-times to further encode the information.

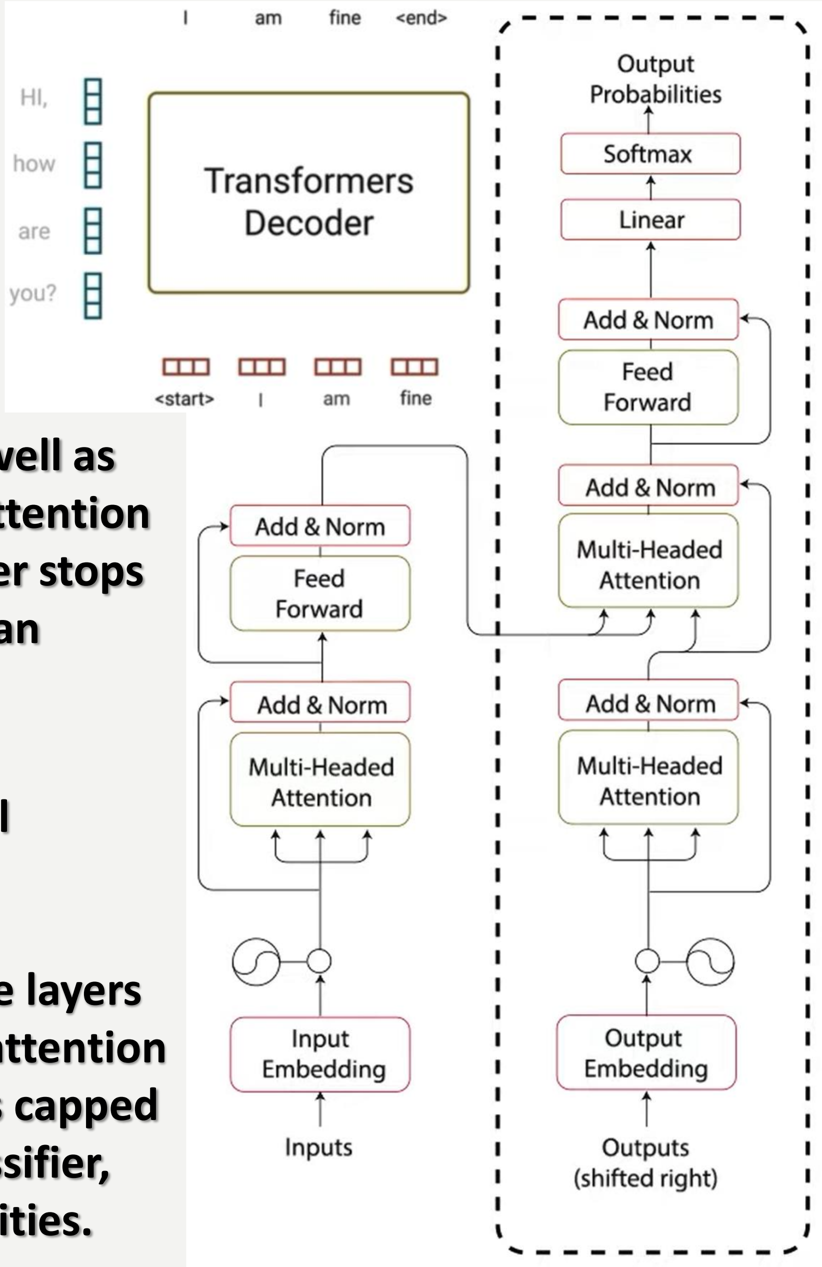
Each layer has the opportunity to learn different attention representations boosting the potential power of the attention network.



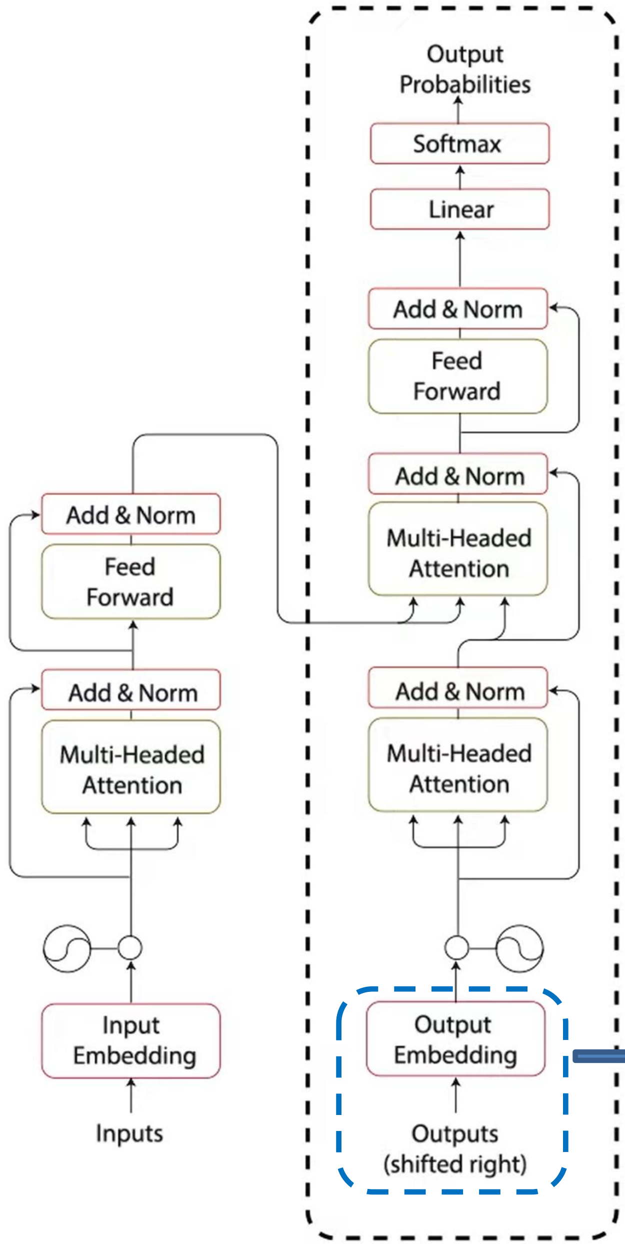
Decoding Transformer's Subnetwork

The decoder

- generates text sequences;
- has a similar sub-layer as the encoder;
- is autoregressive, i.e. it begins with a start token and takes in a list of previous outputs as inputs, as well as the encoder outputs that contain the attention information from the input. The decoder stops decoding when it generates a token as an output.
- has two multi-headed attention layers, a pointwise feed-forward layer, residual connections, and layer normalization after each sub-layer.
  These sub-layers behave similarly to the layers in the encoder but each multi-headed attention layer has a different job. The decoder is capped off with a linear layer that acts as a classifier, and a softmax to get the word probabilities.

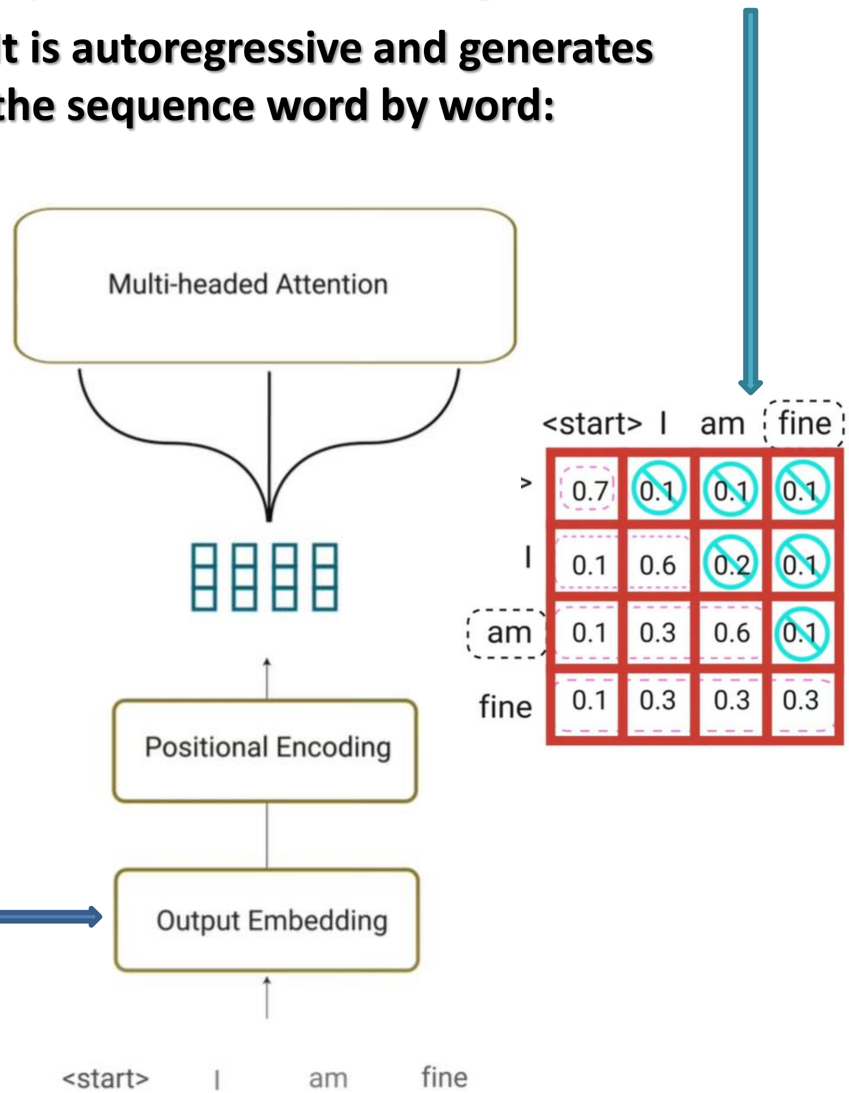


Decoding Layers

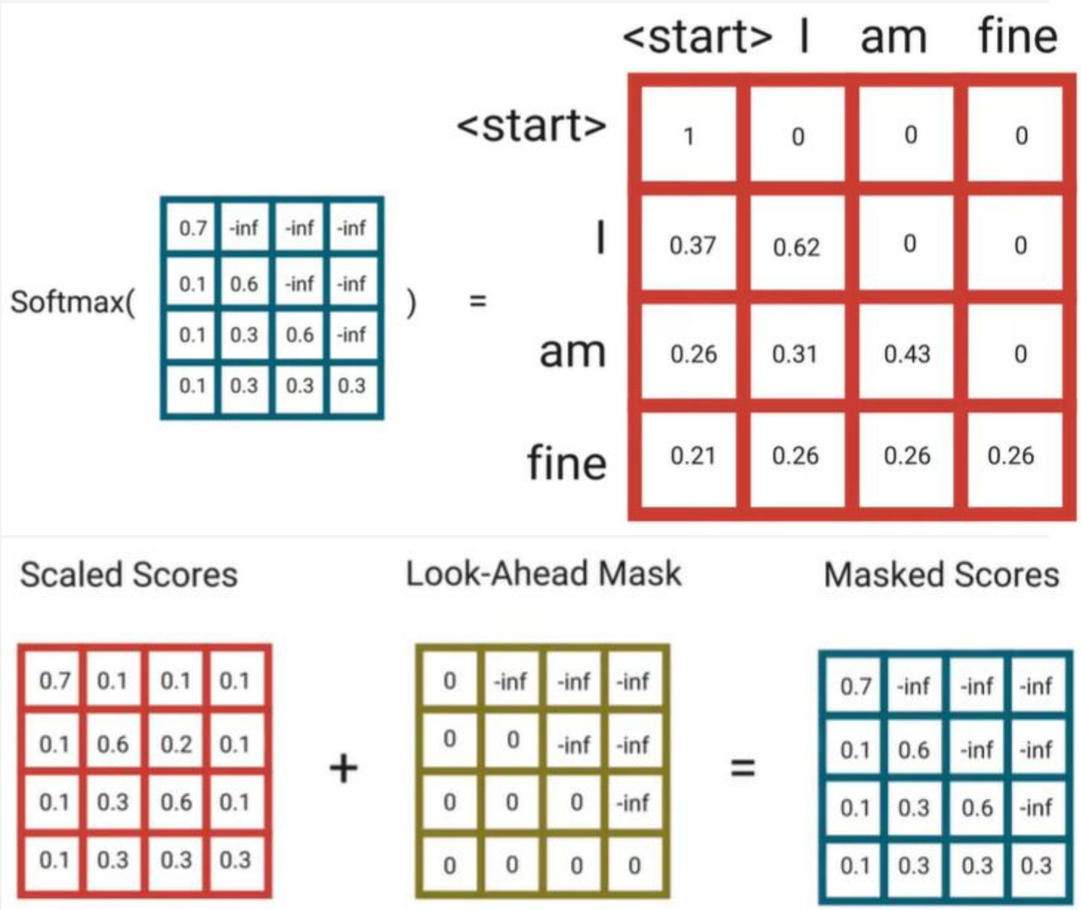


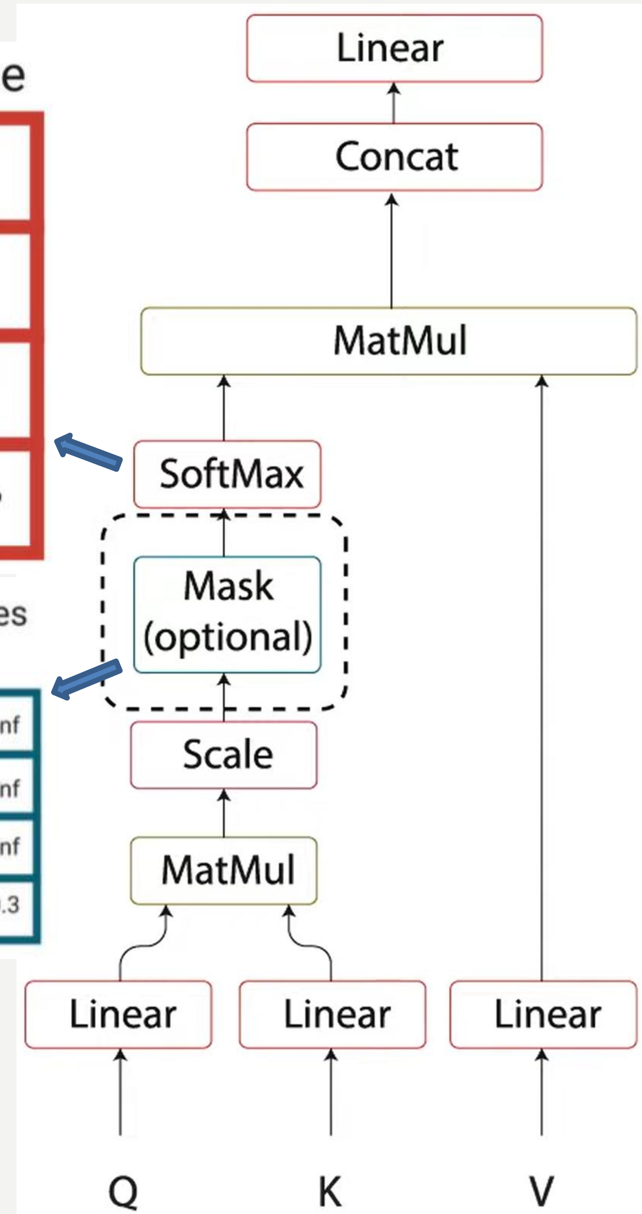
The decoder's multi-headed attention layer cannot conditioning to future tokens.

It is autoregressive and generates the sequence word by word:



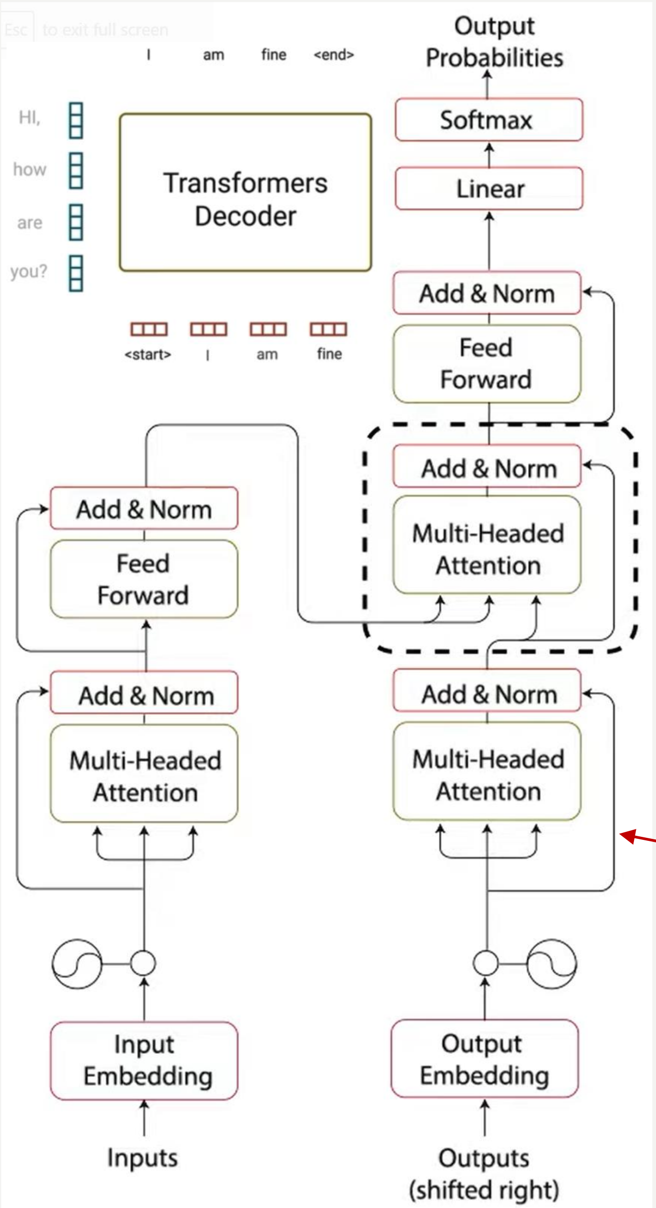
Look-Ahead Masking





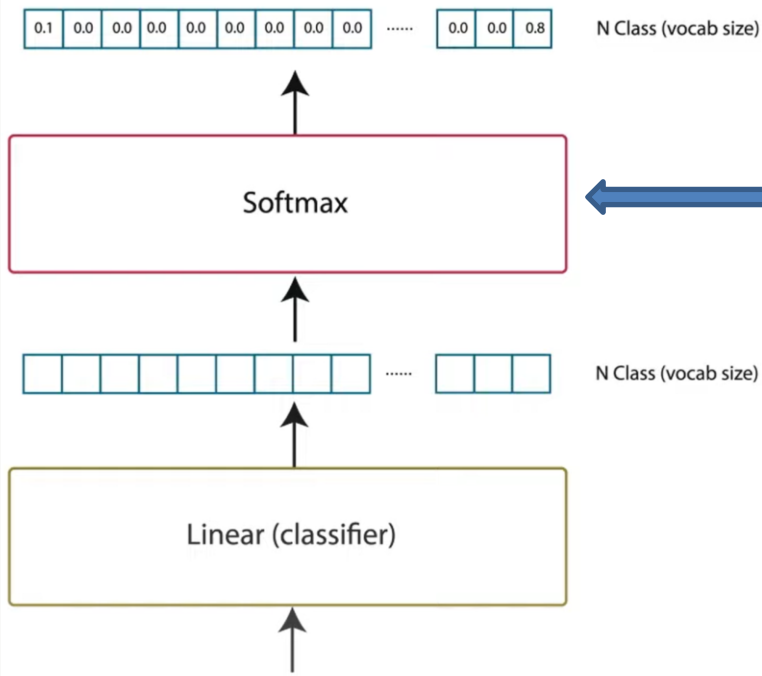
The mask is a matrix of the same size as the attention scores filled with values of 0's and negative infinities. When we add the mask to the scaled attention scores, you get a matrix of the scores, with the top right triangle filled with negativity infinities.

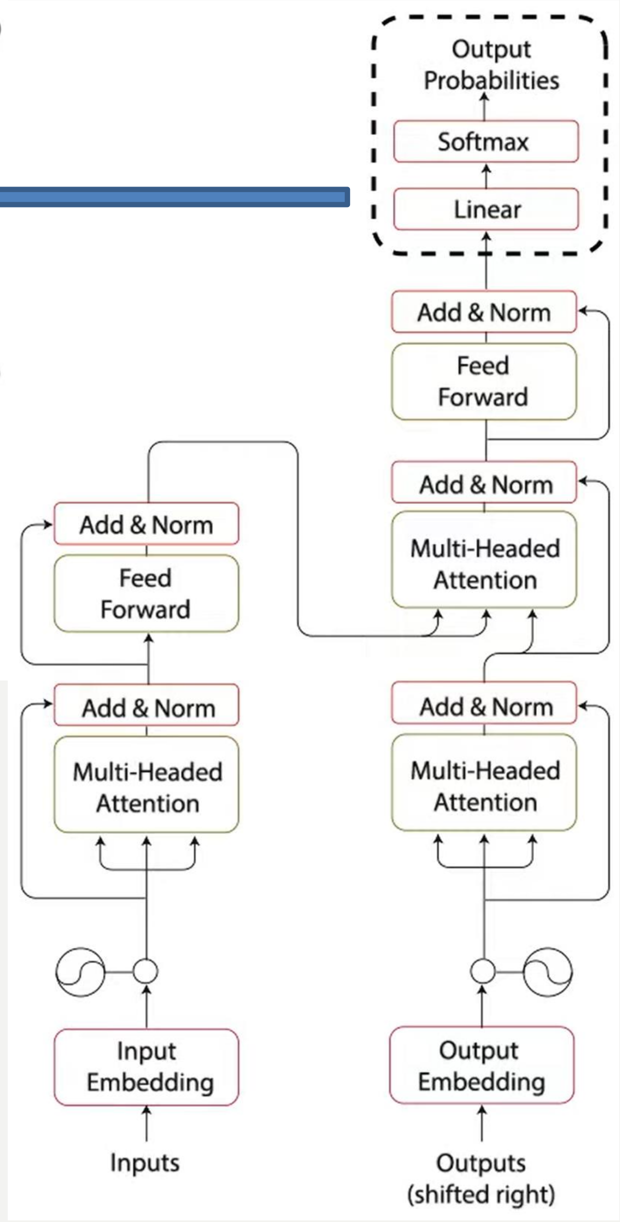
Decoder 2nd Multi-Headed Attention



- The output of the second multi-headed attention goes through a pointwise feedforward layer for further processing.
- For the second multi-headed attention layer, the encoder's outputs are the queries and the keys, and the first multi-headed attention layer outputs of the decoder are the values.
- This process matches the encoder's input to the decoder's input, allowing the decoder to decide which encoder input is relevant to put a focus on.
- The output of the first multi-headed attention is a masked output vector with information on how the model should attend on the decoder's input.
- This layer still has multiple heads that the mask is being applied to, before getting concatenated and fed through a linear layer for further processing.

Decoder's Final Classification



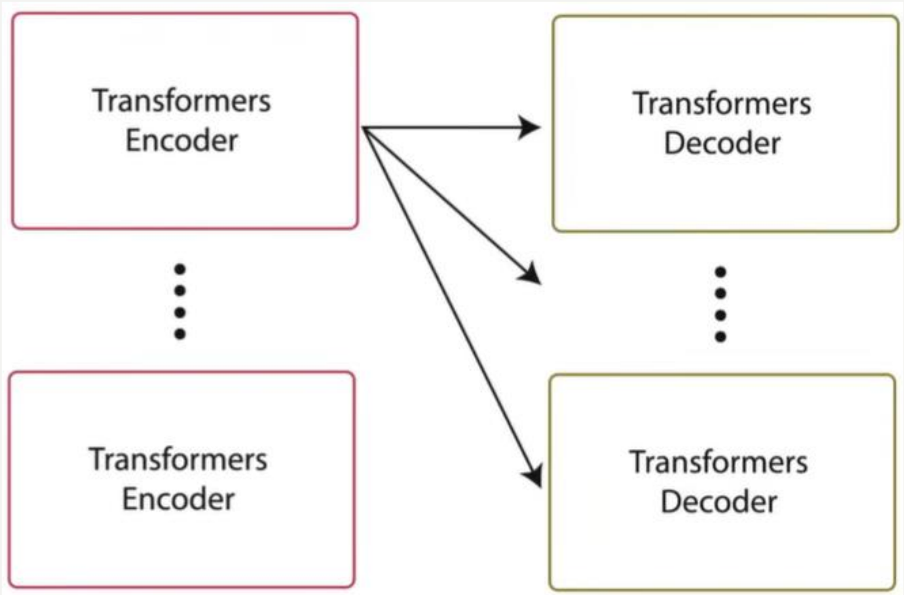


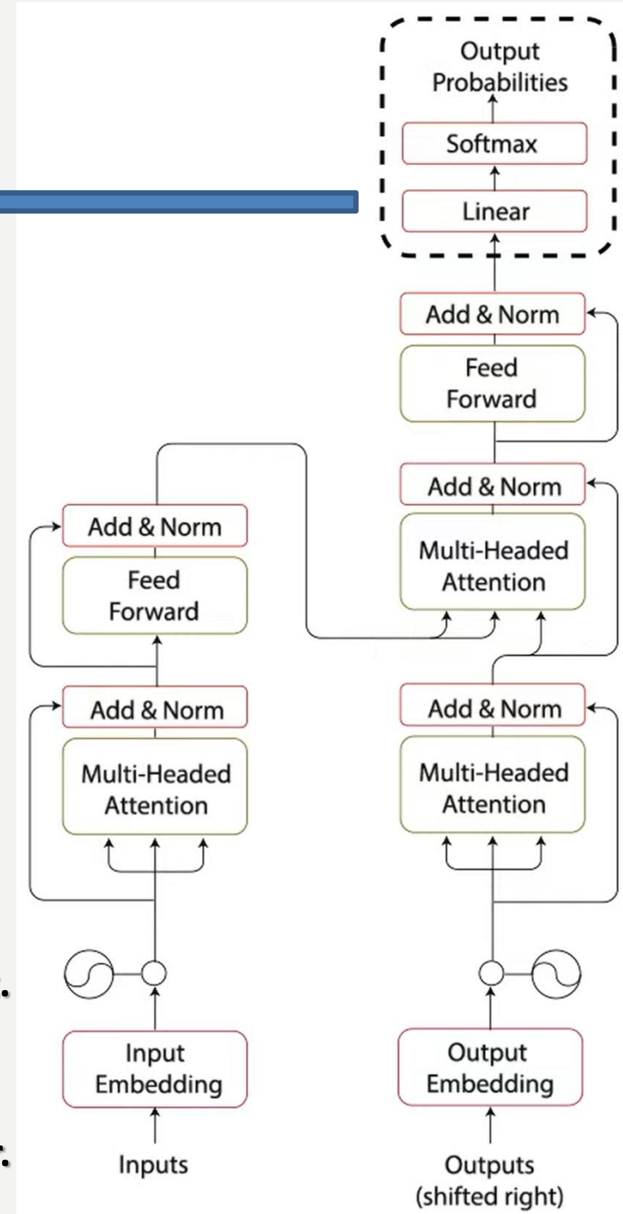
The output of the final pointwise feedforward layer goes through a final linear layer that acts as a classifier. The classifier is as big as the number of classes you have, e.g. 10,000 output for 10,000 words.

Then, the output of the classifier gets fed into a softmax layer, which will produce probability scores between 0 and 1.

Finally, we take the index of the highest probability score, and that equals our predicted word.

Stacking Decoders





The decoder then takes the output, adds it to the list of decoder inputs, and continues decoding again and again until a token is predicted.

For our case, the highest probability prediction is the final class which is assigned to the <end> token.

The decoder can also be stacked N layers high, each layer taking in inputs from the encoder and the layers before it.

By stacking the layers, the model can learn to extract and focus on different combinations of attention from its attention heads, potentially boosting its predictive power.

Final Words

I hope you enjoyed this course

and learned a lot of new things…

Final Project Presentations

It’s time to finish this course…

Final Presentations

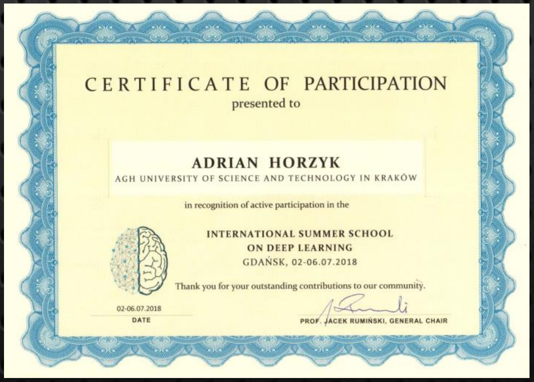
Remarks for the final presentations:

- Each final presentation should be presented in about 5 minutes + 2 minutes for the discussion.
- Try to inspire us and show what you have learned and what was interesting in the problem you solved.
- Share your knowledge and experience gained.
- Focus on the most essential things of your topic, model, results, and solution.
- Show us the difficulties where we could stack when solving similar problems.
- Describe the most important hyperparameters and how you found out those which were finally the most efficient in your case.
- Try to compare your solution and results to the other solutions and results you found on the Internet or research papers.
- Interpret and summarize results and your achievements.
- **Don’t forget to send your final project source codes with or plus presentation using MS Teams for final evaluation!**

BIBLIOGRAPY

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016, ISBN 978-1-59327-741-3 or PWN 2018.
2. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, "Attention Is All You Need", https://arxiv.org/abs/1706.03762
3. Illustrated guide to Transformers: https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0

Home page for this course:
http://home.agh.edu.pl/~horzyk/lectures/ahdydci.php


CERTIFICATE OF PARTICIPATION
presented to

ADRIAN HORZYK
AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY IN KRAKÓW

in recognition of active participation in the

INTERNATIONAL SUMMER SCHOOL
ON DEEP LEARNING
GDAŃSK, 02-06.07.2018

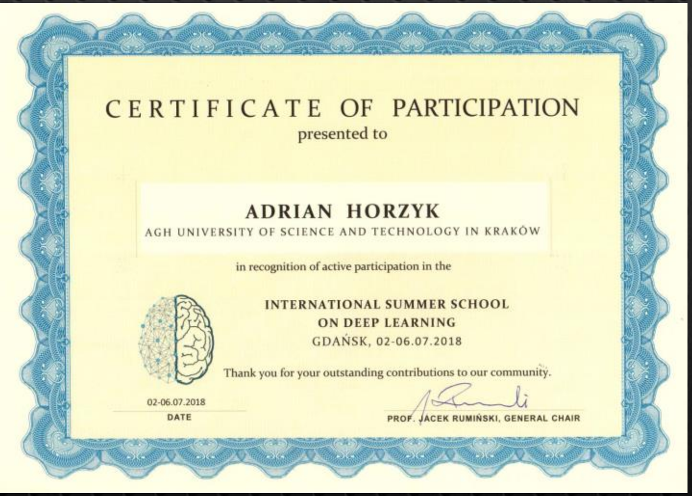
Thank you for your outstanding contributions to our community.

02-06.07.2018
DATE

PROF. JACEK RUMIŃSKI, GENERAL CHAIR

BIBLIOGRAPY

Home page for this course:
http://home.agh.edu.pl/~horzyk/lectures/ahdydci.php


CERTIFICATE OF PARTICIPATION
presented to

ADRIAN HORZYK
AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY IN KRAKÓW

in recognition of active participation in the

INTERNATIONAL SUMMER SCHOOL
ON DEEP LEARNING
GDAŃSK, 02-06.07.2018

Thank you for your outstanding contributions to our community.

02-06.07.2018
DATE

PROF. JACEK RUMIŃSKI, GENERAL CHAIR