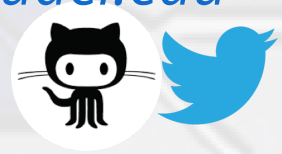
Machine Learning for Physical and Natural Scientists 11

Neural Networks: Transformers

dr. federica bianco
fbianco@udel.edu
@fedhere



this slide deck:

https://slides.com/federicabianco/mltpns23_11


MLPNS
0
ML model
performance

ML model performance

Accuracy, Recall, Precision

$$LR = \frac{\text{False Negative}}{\text{True Negative}}$$

	H0 is True	**H0 is False**
H0 is falsified	Type I Error False Positive	True Positive
H0 is not falsified	True Negative	Type II Error False Negative

ML model performance

Accuracy, Recall, Precision

$$LR = \frac{\text{False Negative}}{\text{True Negative}}$$

	H0 is True	H0 is False
H0 is falsified	important message spammed	True Positive
H0 is not falsified	True Negative	spam in your inbox

ML model performance

Accuracy, Recall, Precision

$$\textbf{Precision} = \frac{TP}{TP + FP}$$

$$\textbf{Recall} = \frac{TP}{TP + FN}$$

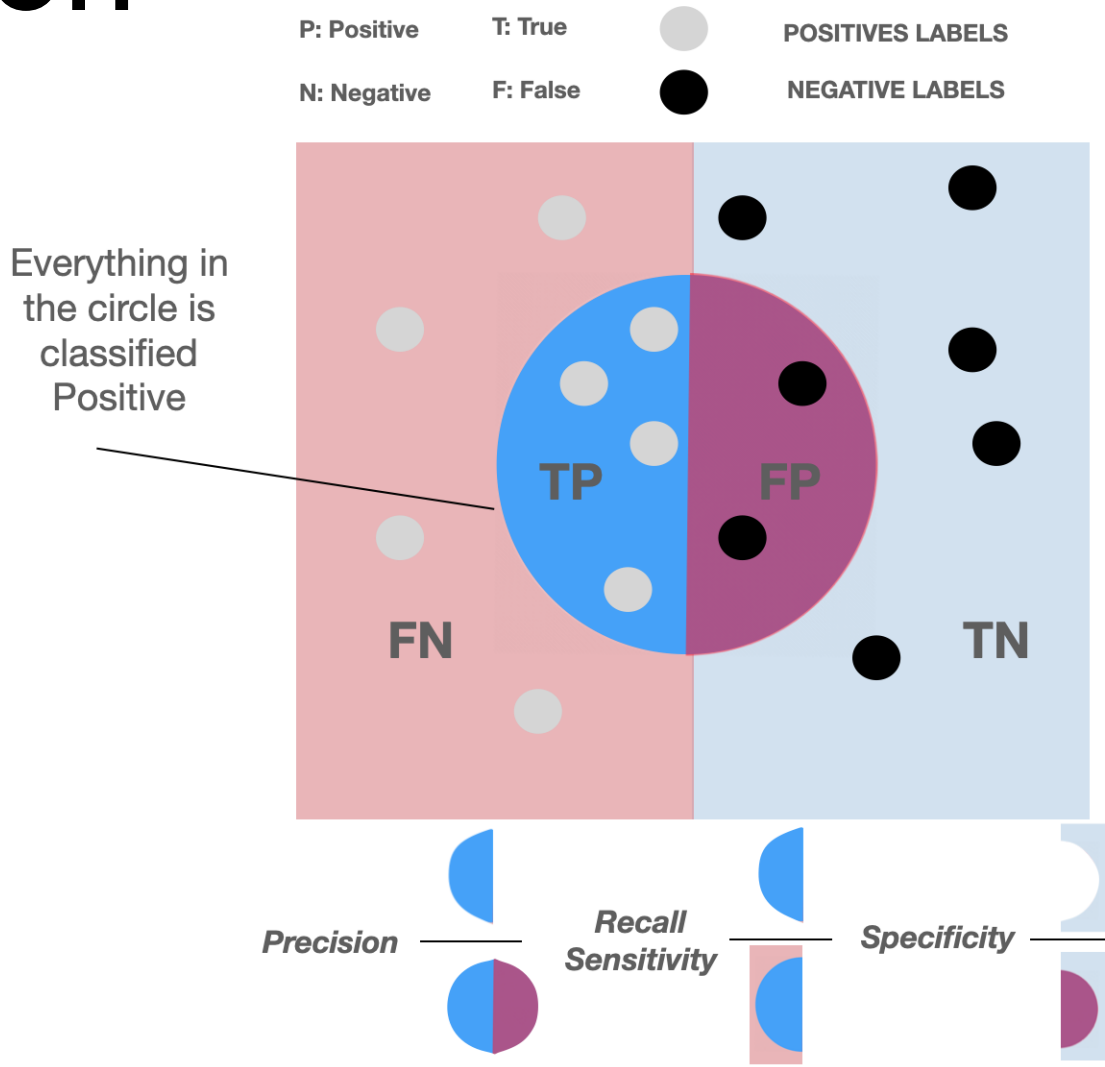
$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

TP=True Positive

FP=False Positive

TN=True Negative

FN=False Positive



https://en.wikipedia.org/wiki/Precision_and_recall

ML model performance

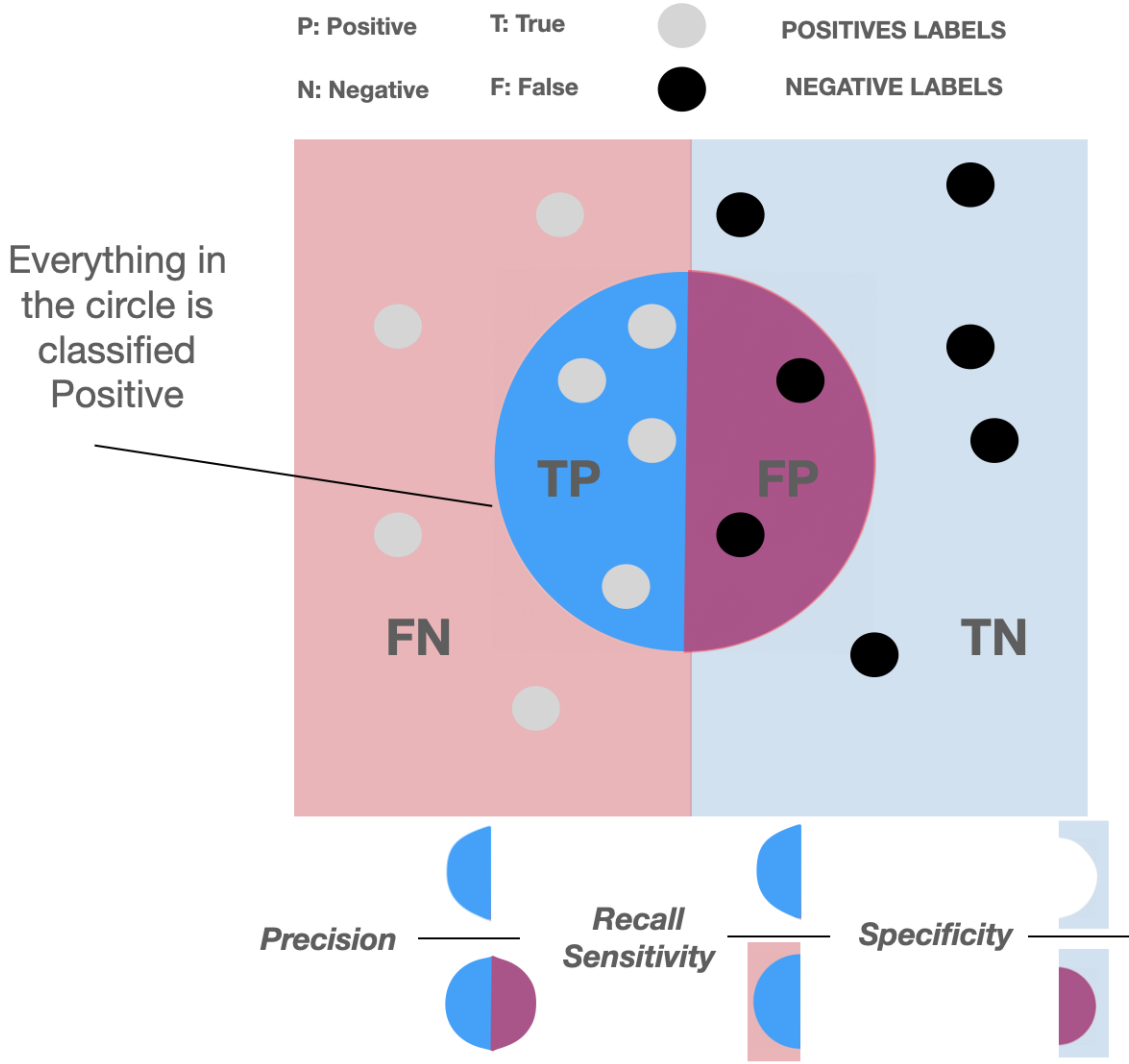
Sensitivity Specificity

True Positive Rate = TP / All Positive Labels

Sensitivity* = *TPR = *Recall*

False Positive Rate = FP / All Negative Labels

Specificity = TP / All Negative Labels = ***1 - FPR***



ML model performance

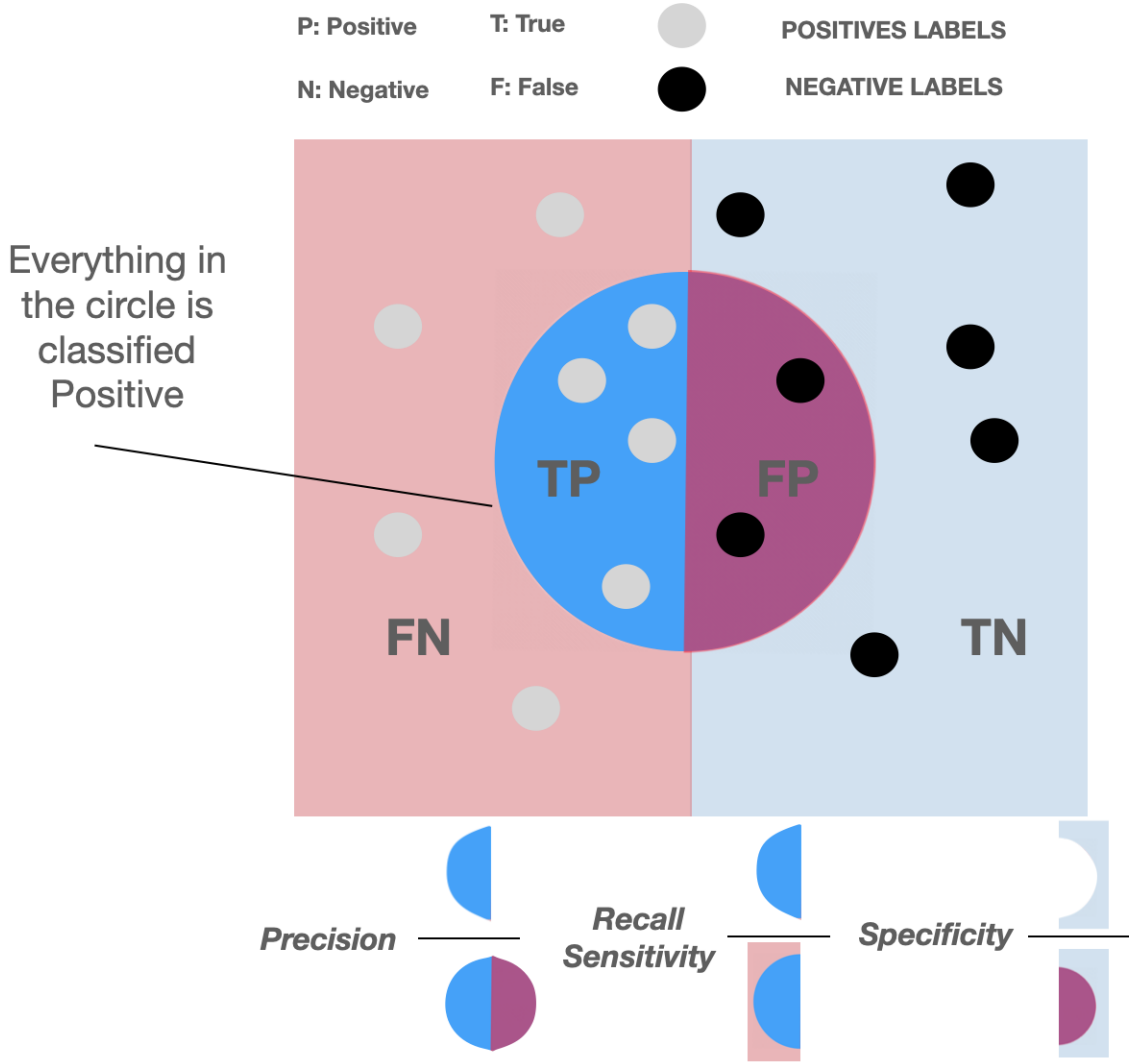
F score

$$F_\beta = (1 + \beta^2) \times \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}$$

$$F_1 = \frac{2}{\frac{1}{\text{recall}} \times \frac{1}{\text{precision}}} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

$$= \frac{\text{tp}}{\text{tp} + \frac{1}{2}(\text{fp} + \text{fn})}$$

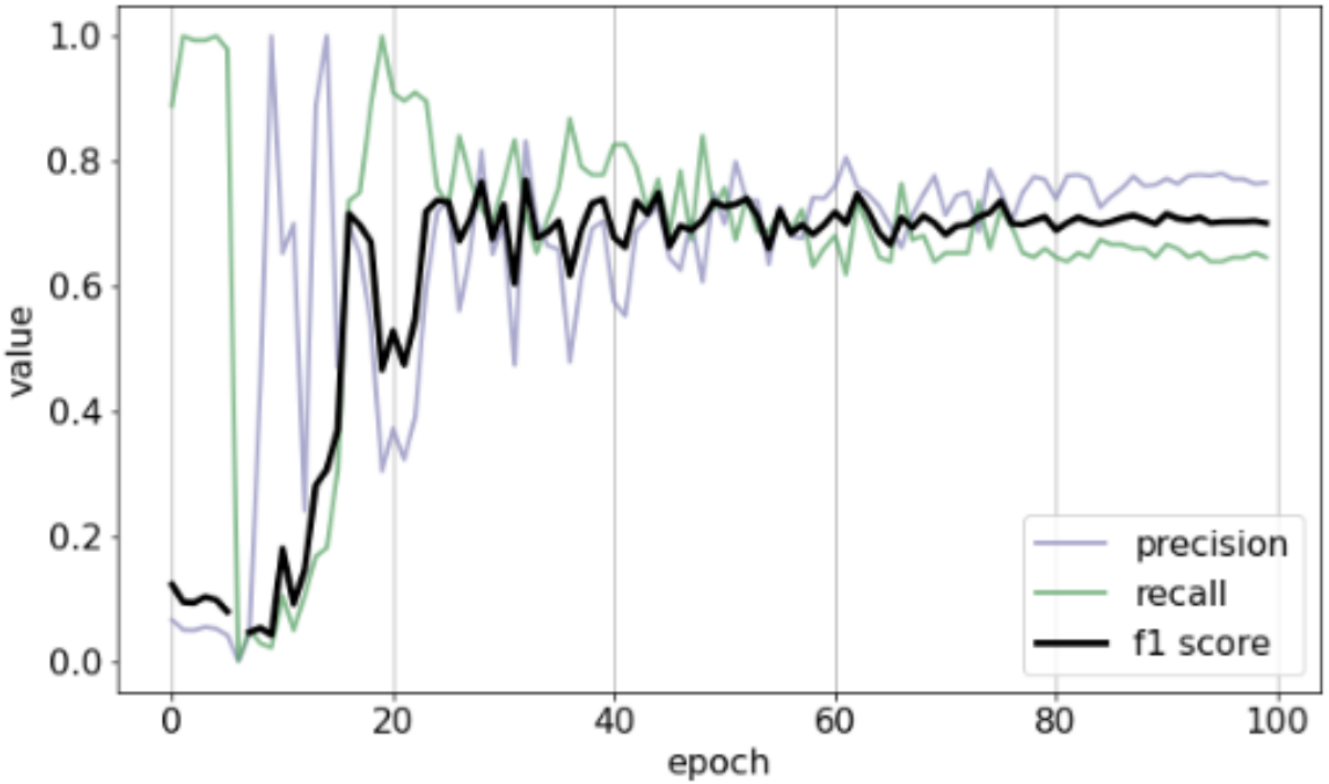
A factor indicating how much more important recall is than precision. For example, if we consider **recall to be twice as important as precision, we can set β to 2**. The standard F-score is equivalent to setting β to one.

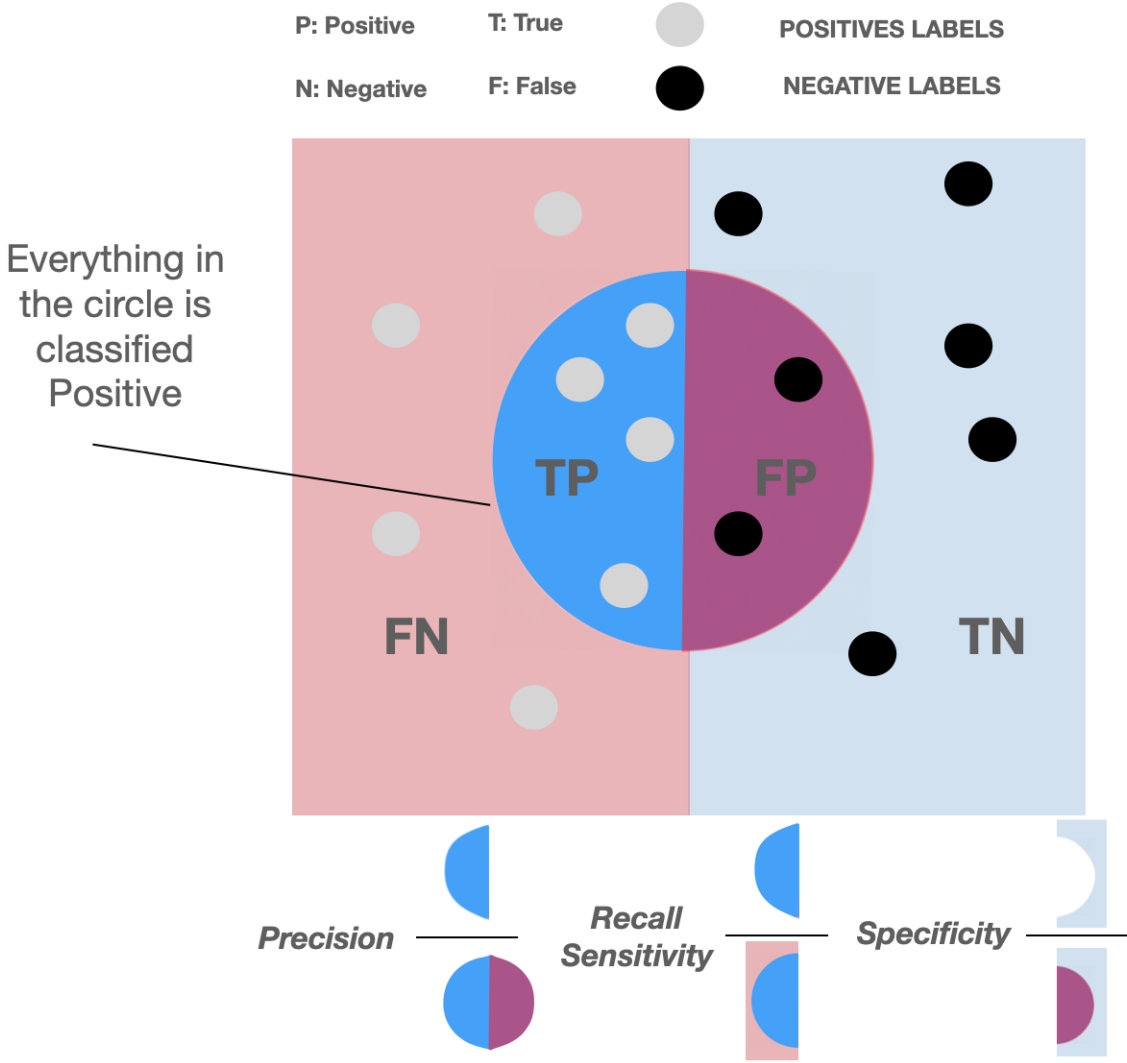


https://en.wikipedia.org/wiki/Precision_and_recall

ML model performance

F score





https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

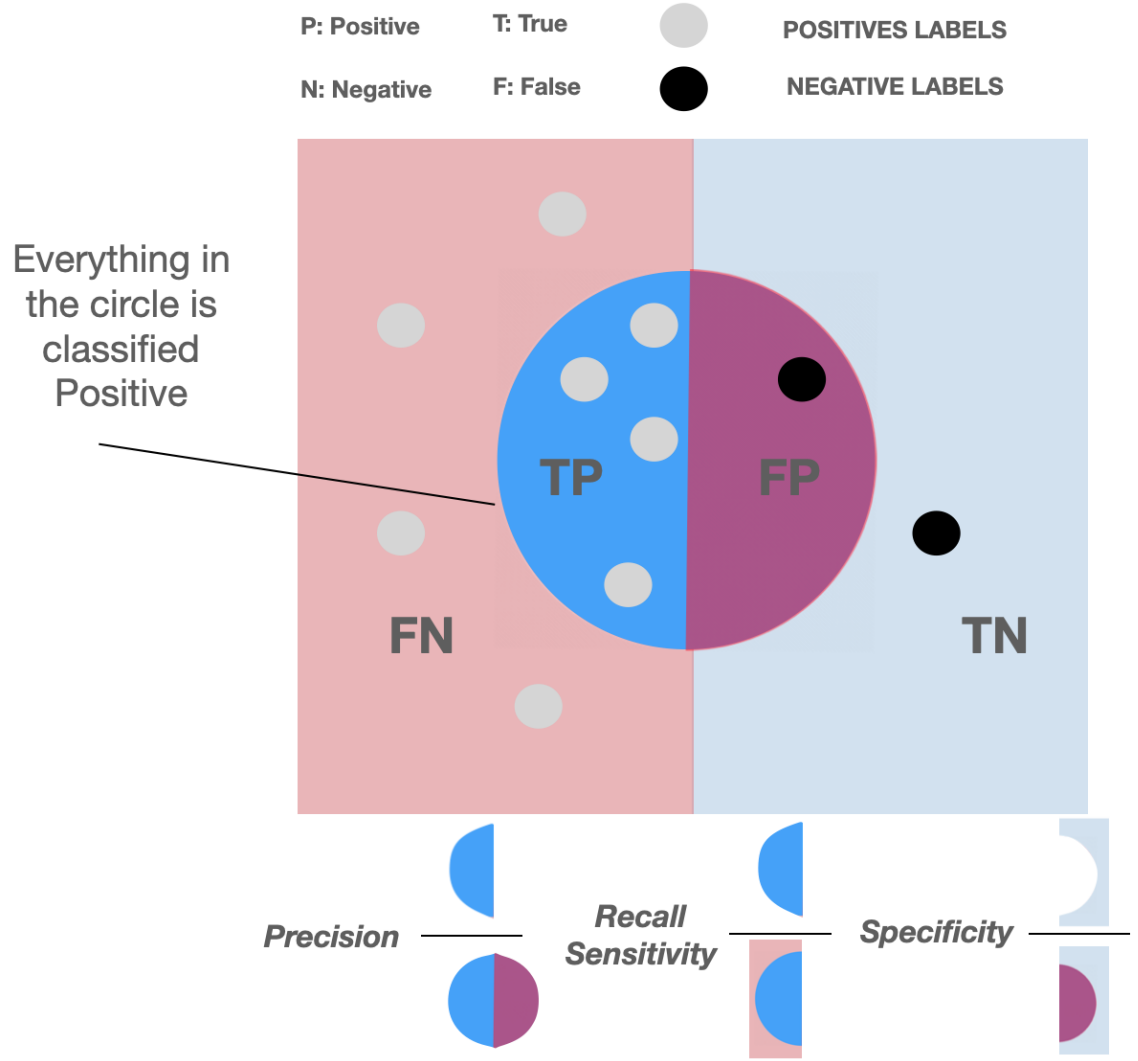
Current classifier accuracy: 50%

Precision?

Recall?

Specificity?

Sensitivity?



https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

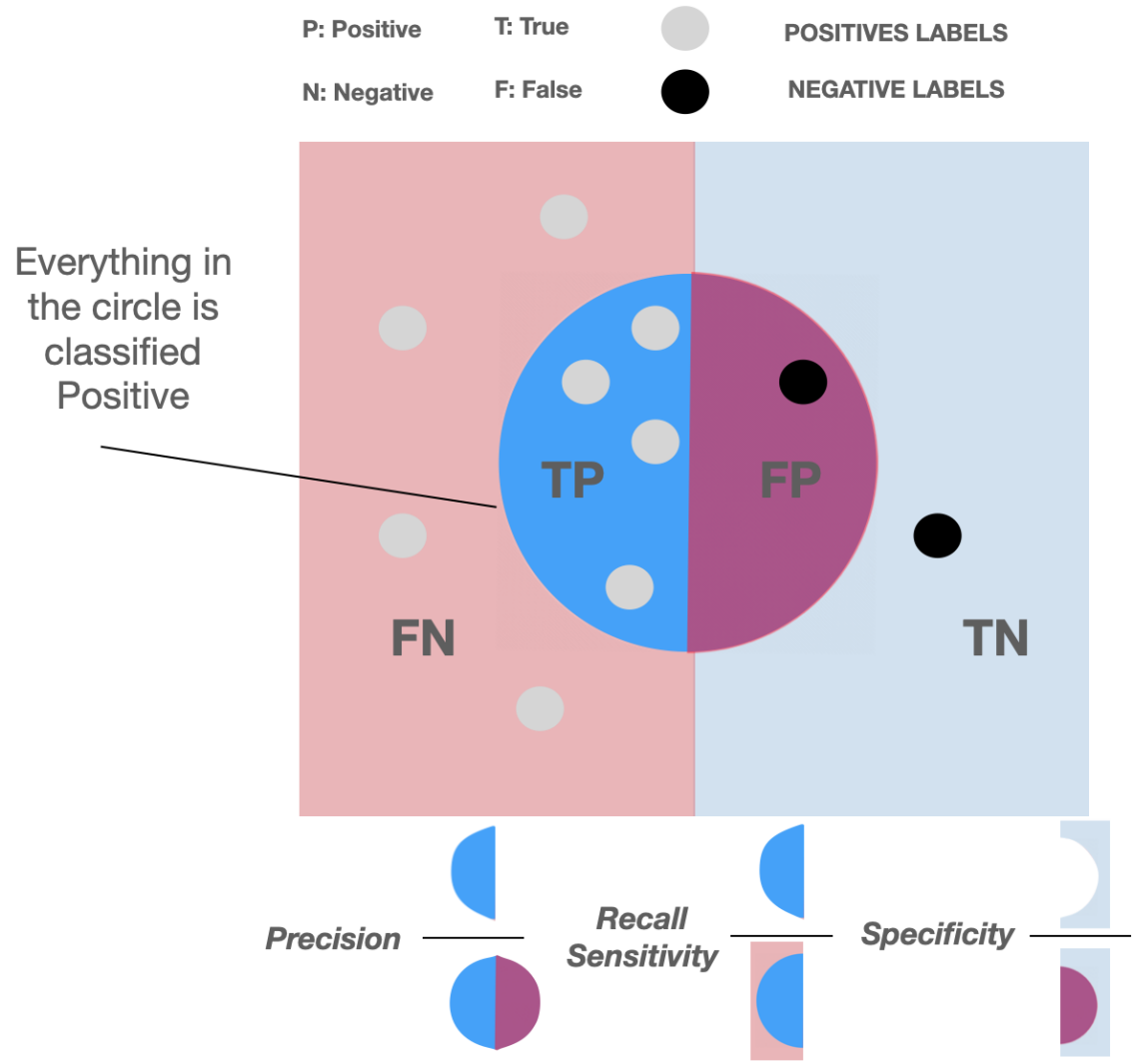
Current classifier accuracy: 50%

Precision: 0.8

Recall?

Specificity?

Sensitivity?



https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

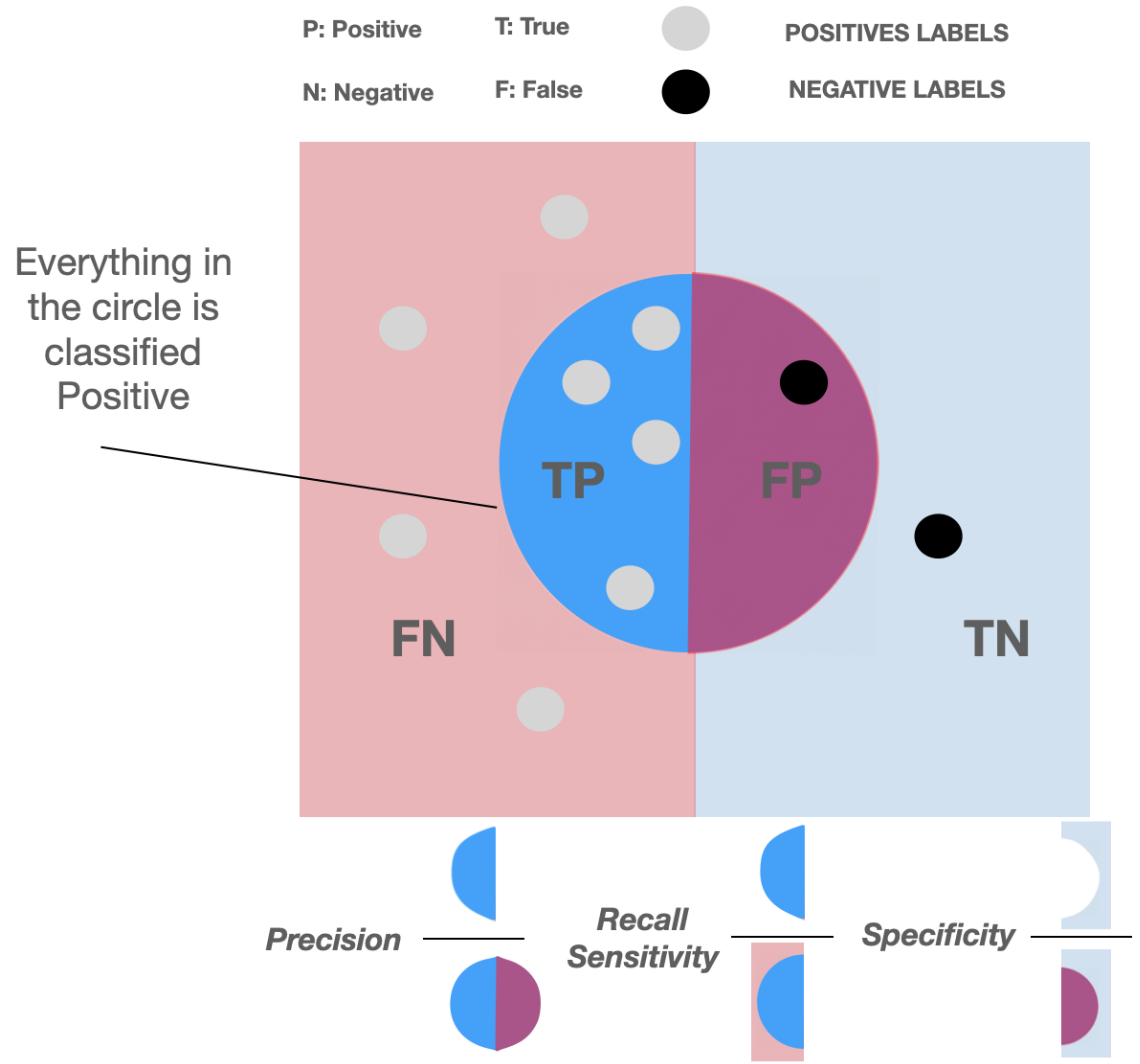
Current classifier accuracy: 50%

Precision: 0.8

Recall: 0.5

Specificity?

Sensitivity?



https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

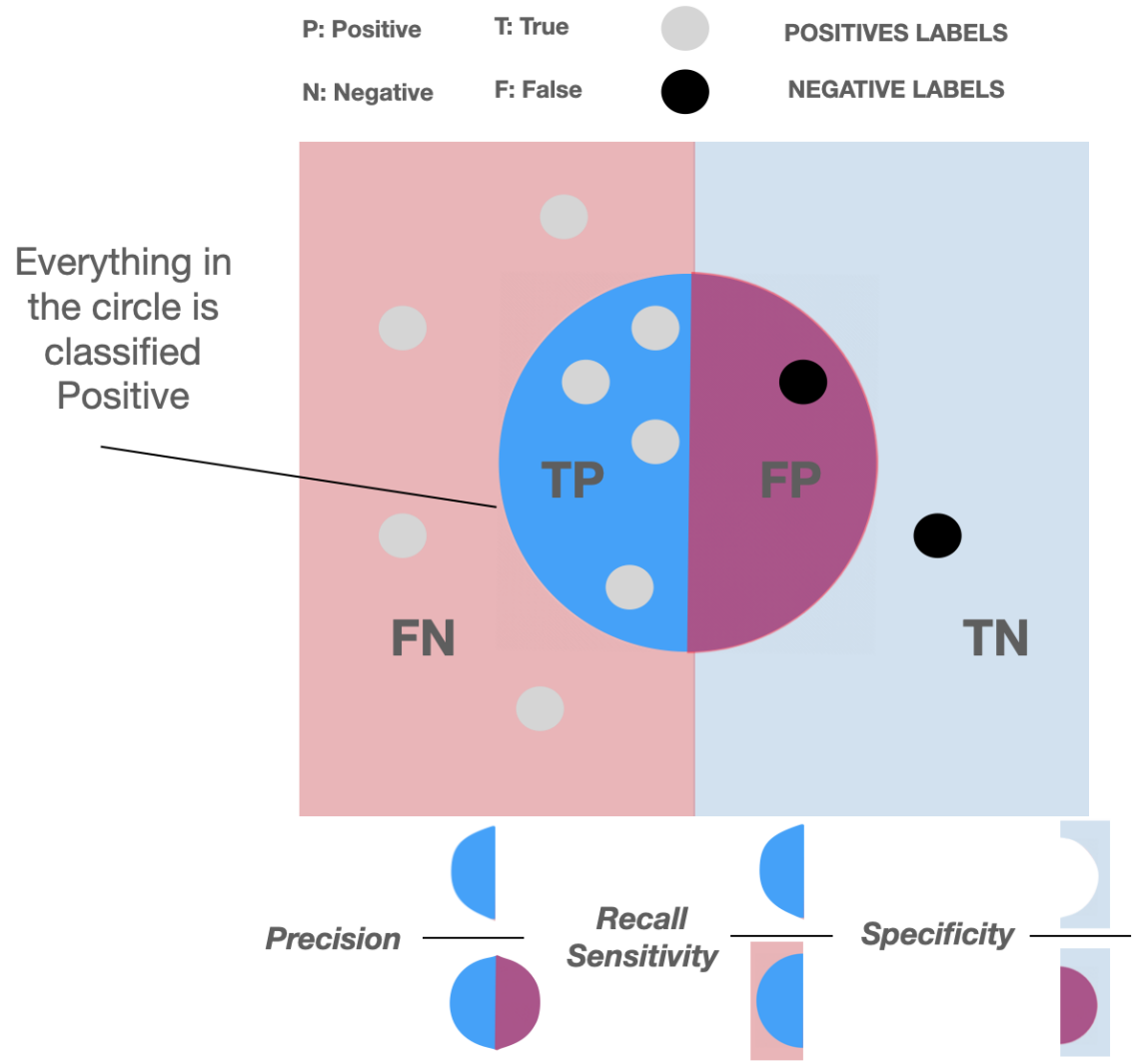
Current classifier accuracy: 50%

Precision: 0.8

Recall: 0.5

Specificity: 0.5

Sensitivity?



https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

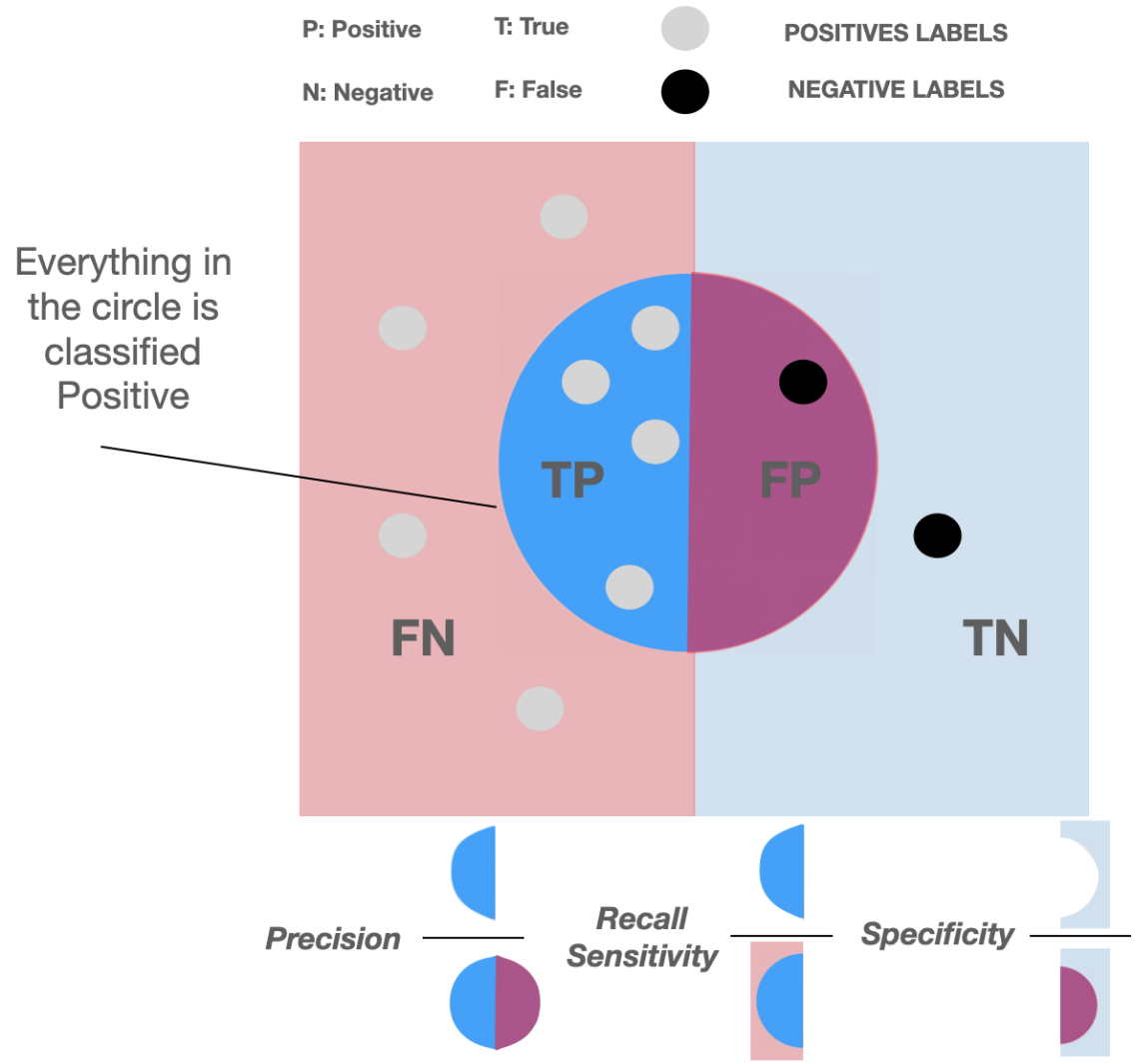
Current classifier accuracy: 50%

Precision: 0.8

Recall: 0.5

Specificity: 0.5

Sensitivity: 0.5



https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

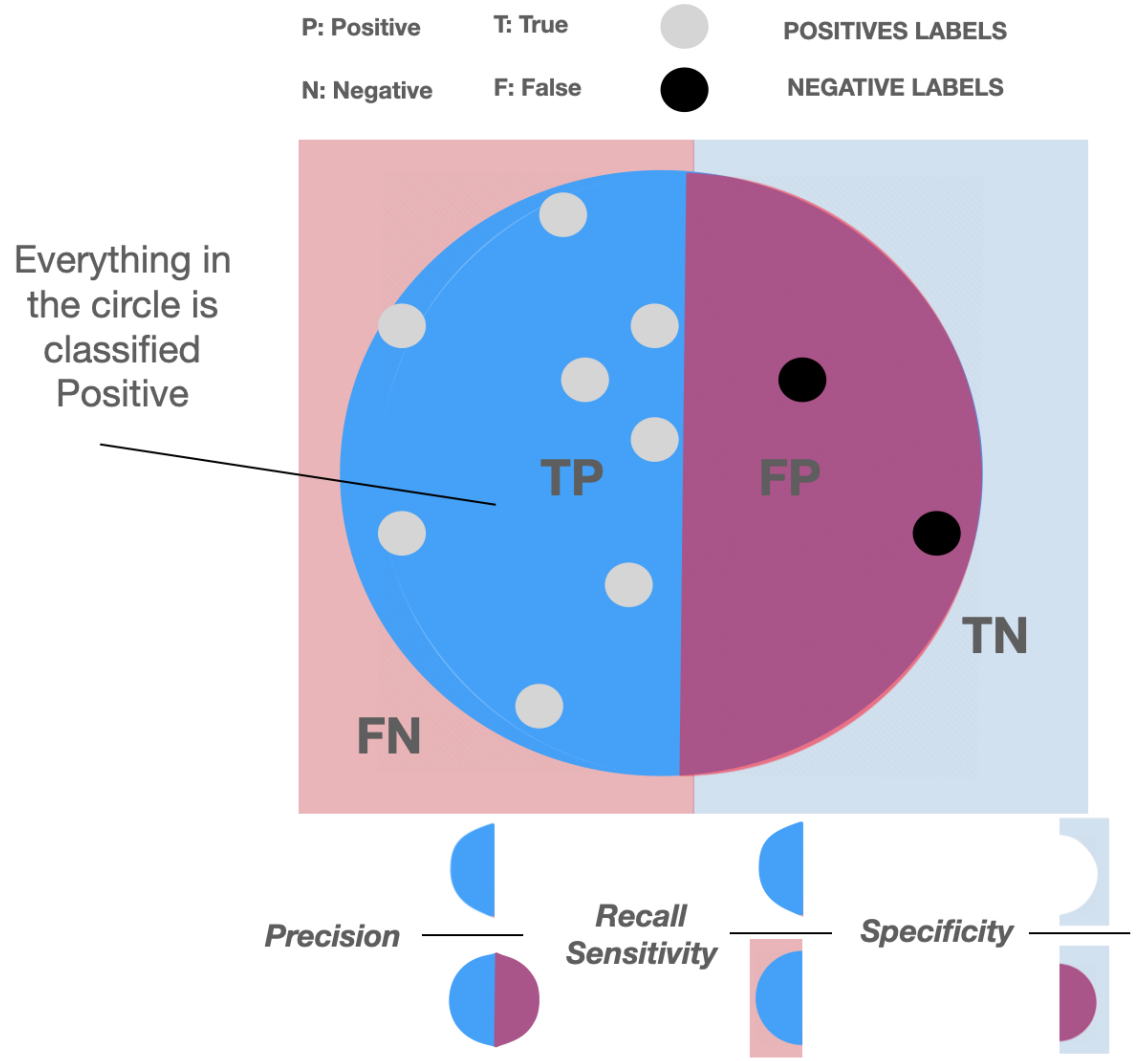
Current classifier accuracy:

Precision:

Recall:

Specificity:

Sensitivity:



https://en.wikipedia.org/wiki/Precision_and_recall

Class Imbalance

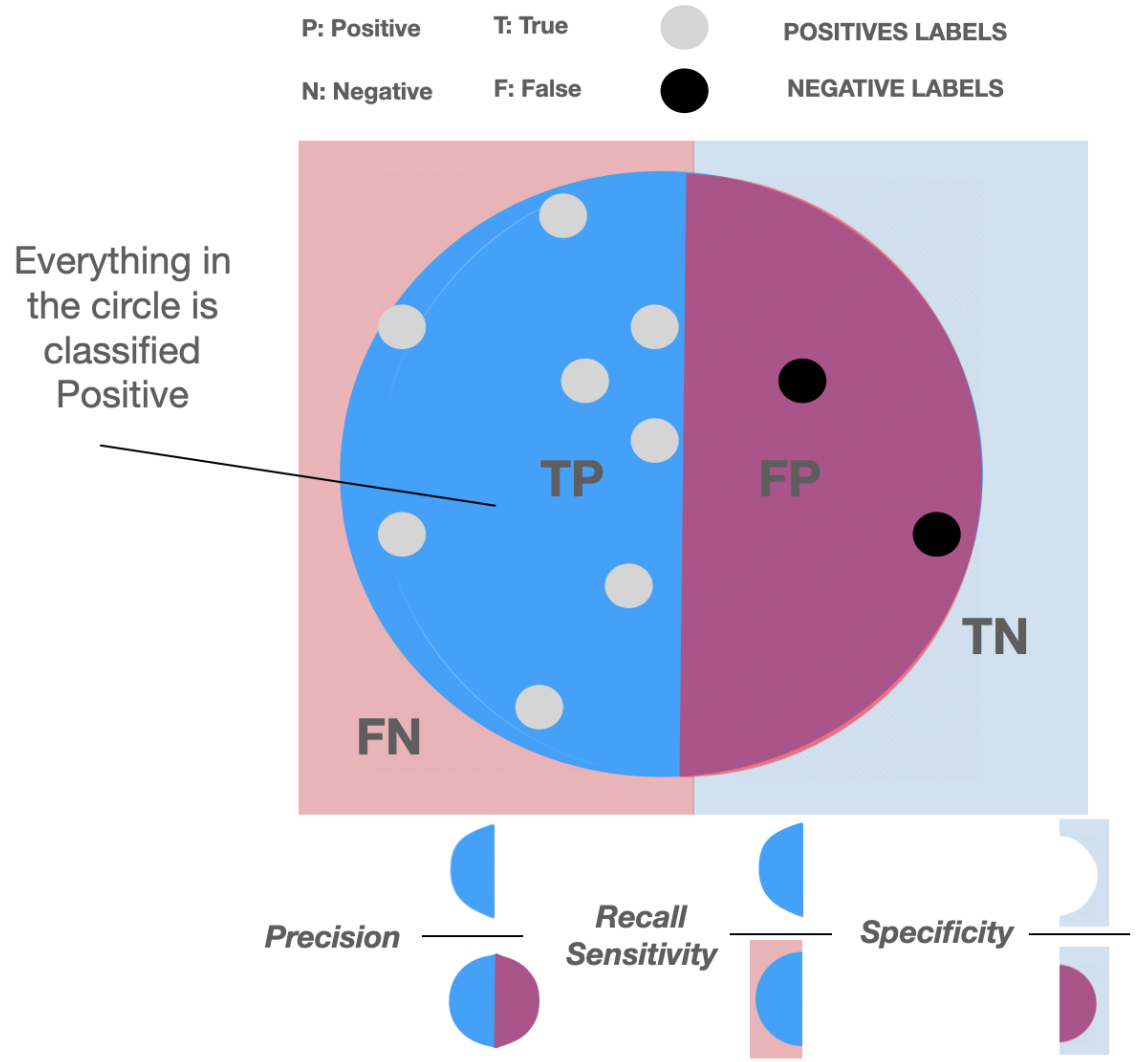
Current classifier accuracy: 80%

Precision: 0.8

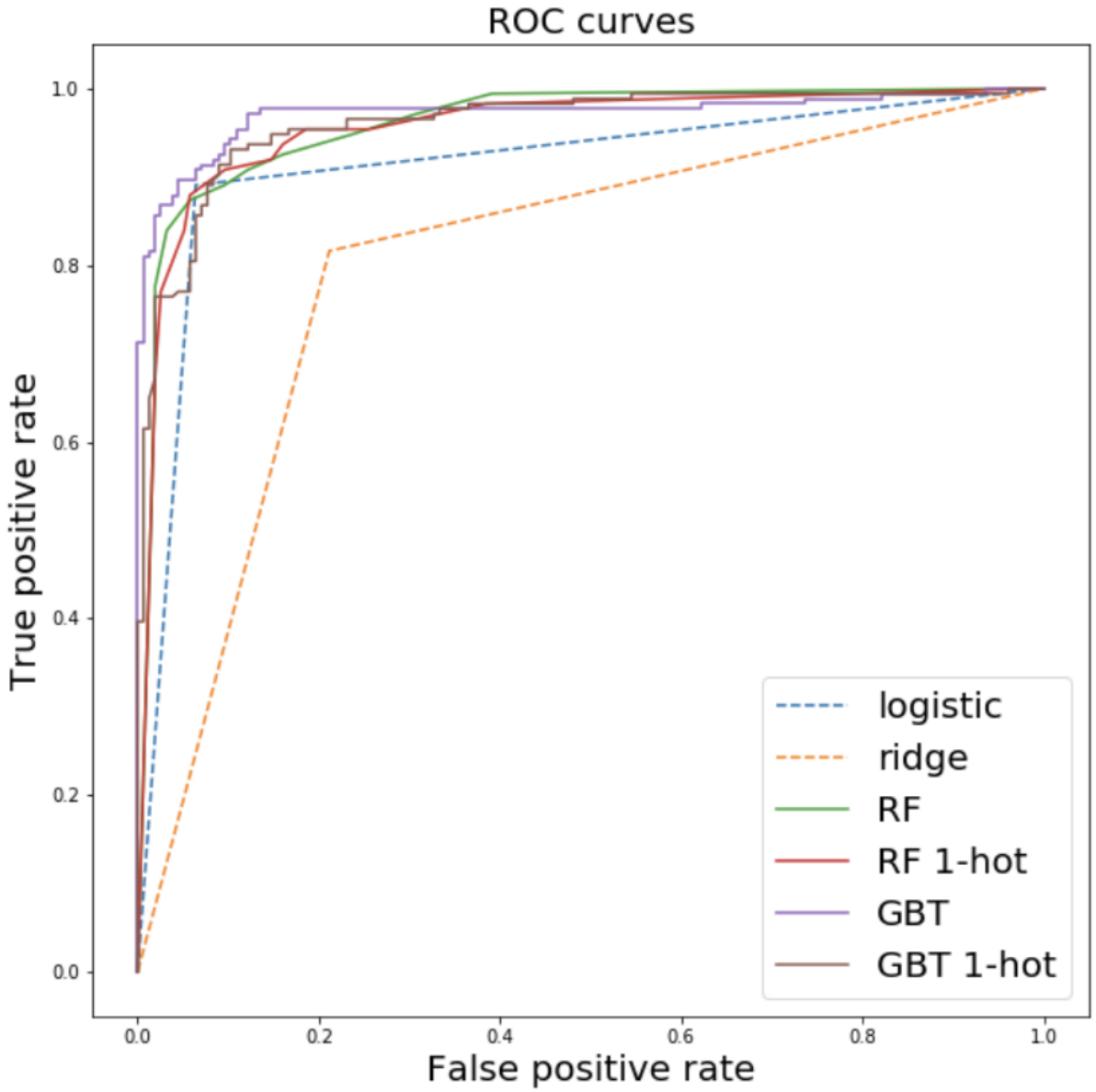
Recall: 1

Specificity: 0

Sensitivity: 1



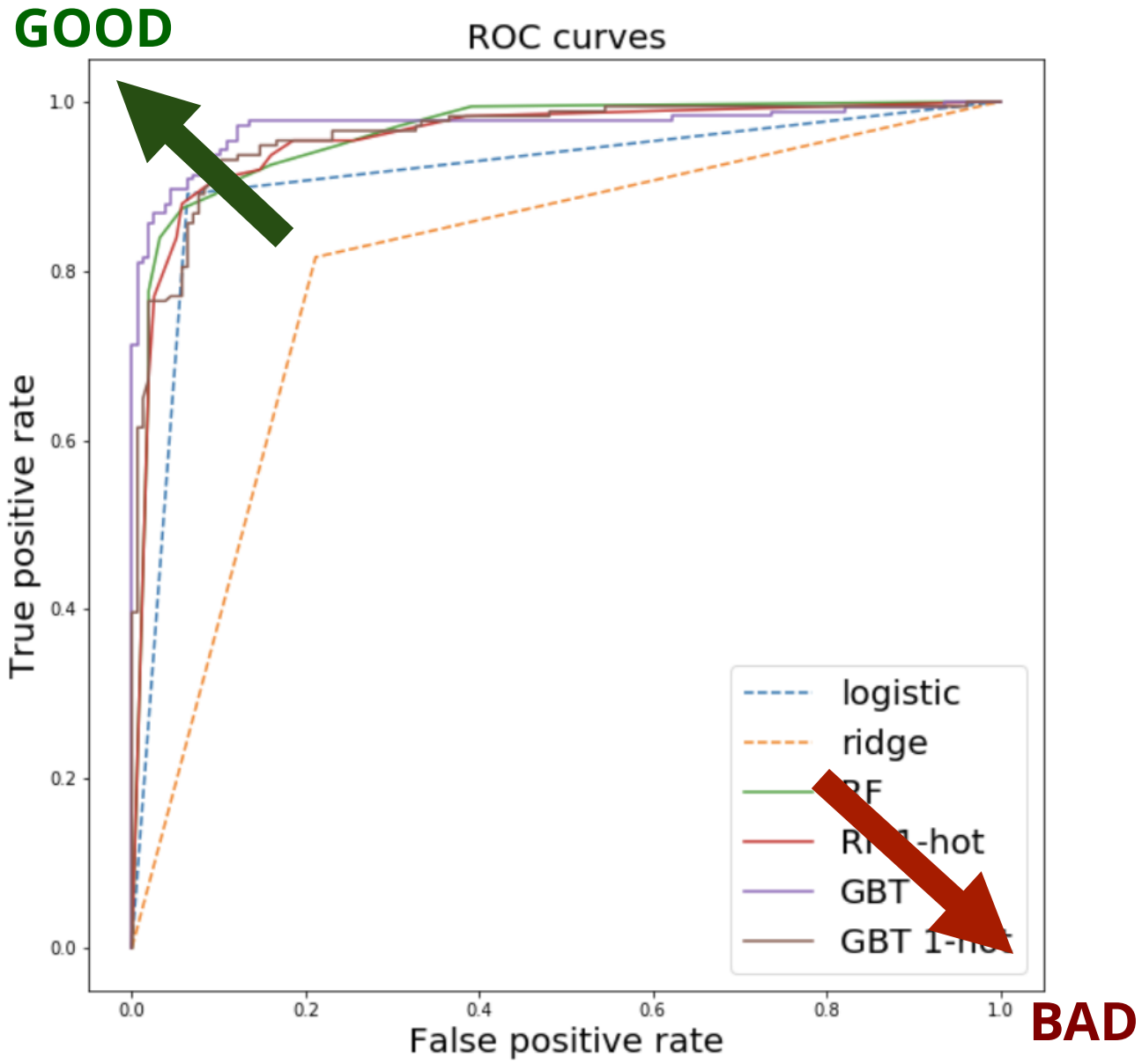
https://en.wikipedia.org/wiki/Precision_and_recall



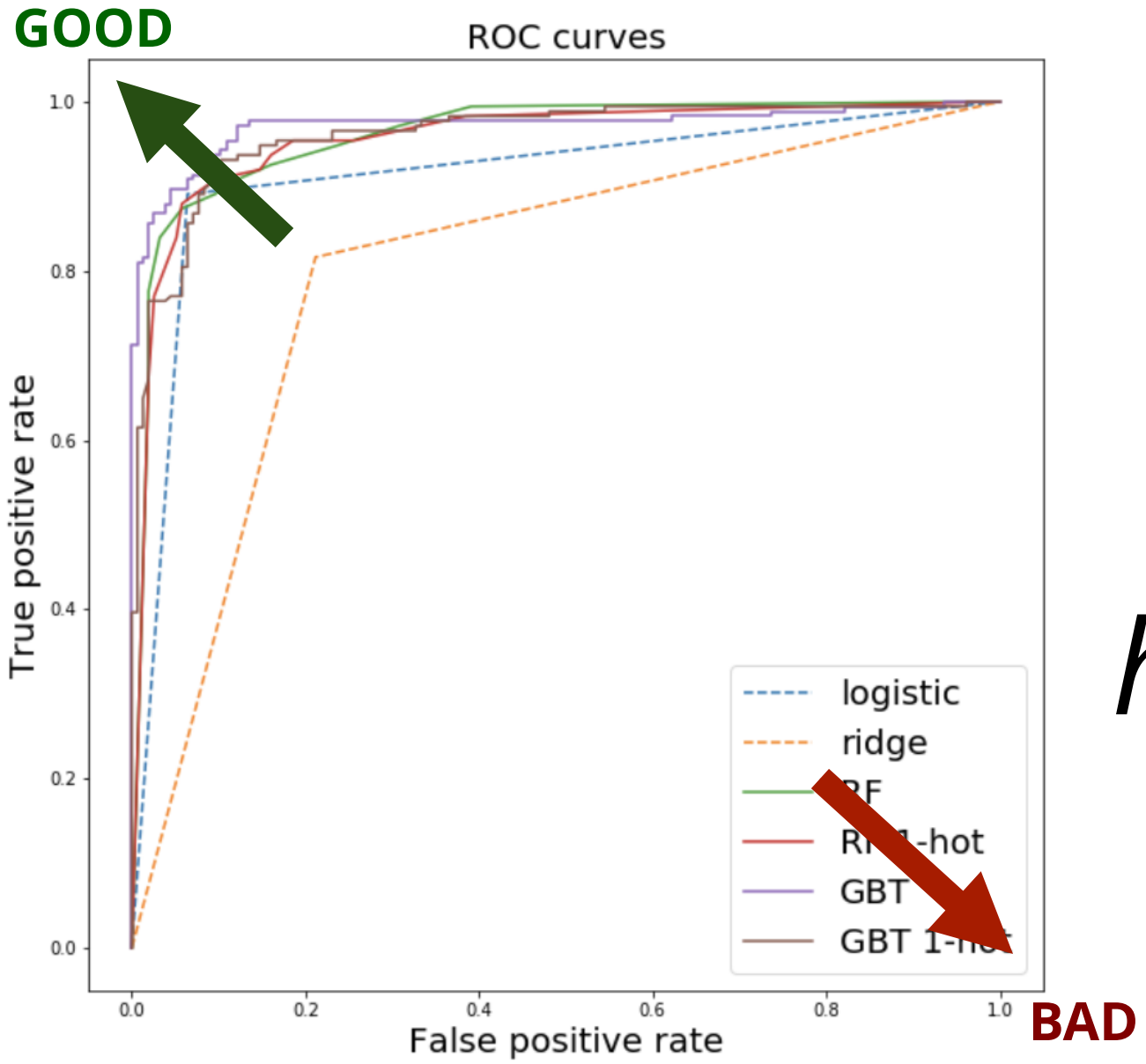
along the curve, the classifier probability threshold ***t*** *is what changes*

$$\text{class} = i \text{ if } p_i > t$$

Receiver operating characteristic



Receiver operating characteristic



tuning by changing *hyperparameters*

Receiver operating characteristic

1

MLPNS

Neural Networks for Time Series Analysis

DeepNeuralNetwork

Promising solution to Time Series Analysis problems because they can learn highly non linear varied relations between data

Recurrent Neural Networks: take as input for the next state prediction the past/present state as well as their hidden NN representation

Issue: training through gradient descent (derivatives) causes the gradient to vanish or explode after few time steps: the mode looses memory of the past rapidly (~few steps) (cause math sometimes is... just hard)

Partial Solution: LSTM: forget cells can extend memory by dropping irrelevant time stamps

DeepNeuralNetwork

Promising solution to Time Series Analysis problems because they can learn highly non linear varied relations between data

Convolutional Neural Networks: learn relationships between pixels

Issue: training is expensive (will discuss next week)

DeepNeuralNetwork

what we are doing, except for the activation function
is exactly a series of matrix multiplictions.

The purpose is to
approximate a function φ

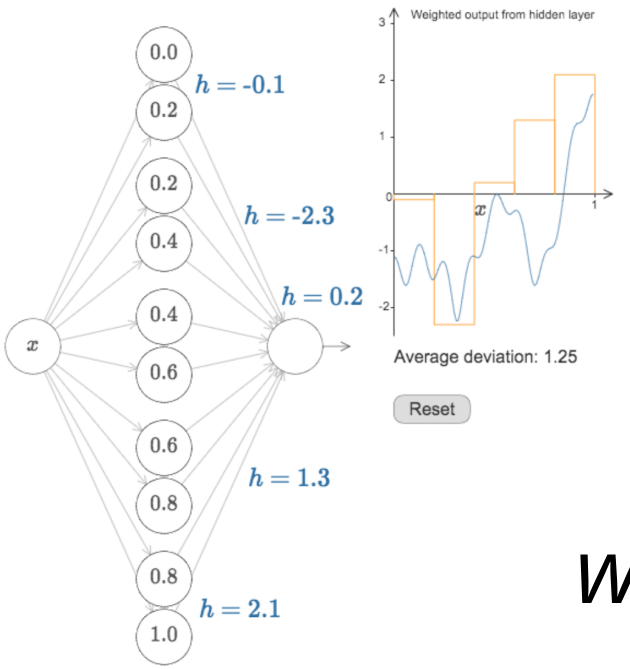
y = *φ(x)*

which (in general) is not linear
with linear operations

$$\phi(\vec{x}) \sim f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

DeepNeuralNetwork

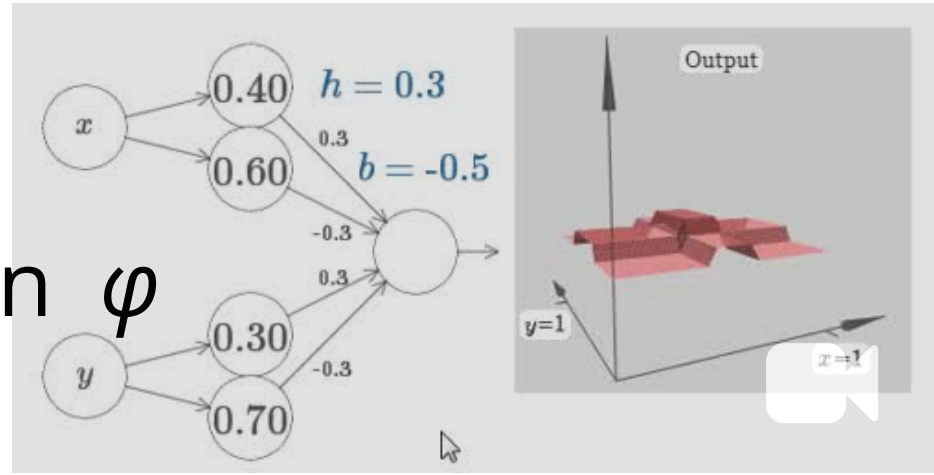
http://neuralnetworksanddeeplearning.com/chap4.html



The purpose is to approximate a function φ

y = φ(x)

which (in general) is not linear with linear operations



$$\phi(\vec{x}) \sim f^{(3)}(f^{(2)}(f^{(1)}(\vec{x} \cdot W_1 + \vec{b_1}) \cdot W_2 + \vec{b_2}) \cdot W_3 + \vec{b_3}) = y$$

Building a DNN

with keras and tensorflow

autoencoder for image recontstruction

Text

What should I choose for the loss function and how does that relate to the activation functiom and optimization?

loss	**good for**	**activation last layer**	**size last layer**
mean_squared_error	regression	linear	one node
mean_absolute_error	regression	linear	one node
mean_squared_logarithmit_error	regression	linear	one node
binary_crossentropy	binary classification	sigmoid	one node
categorical_crossentropy	multiclass classification	sigmoid	N nodes
Kullback_Divergence	multiclass classification, probabilistic inerpretation	sigmoid	N nodes

DeepNeuralNetwork - loss functions

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

Binary Cross Entropy $-(y \log(p) + (1-y) \log(1-p))$

(Multiclass) Cross Entropy $-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$

Kullback-Leibler $\sum(\hat{y} \log \frac{\hat{y}}{y})$

Mean Absolute Error $L1$

Mean Squared Error $L2$

Mean Squared Logarithmic Error $L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (\log(y_i + 1) - \log(\hat{y}_i + 1))^2$

c = class

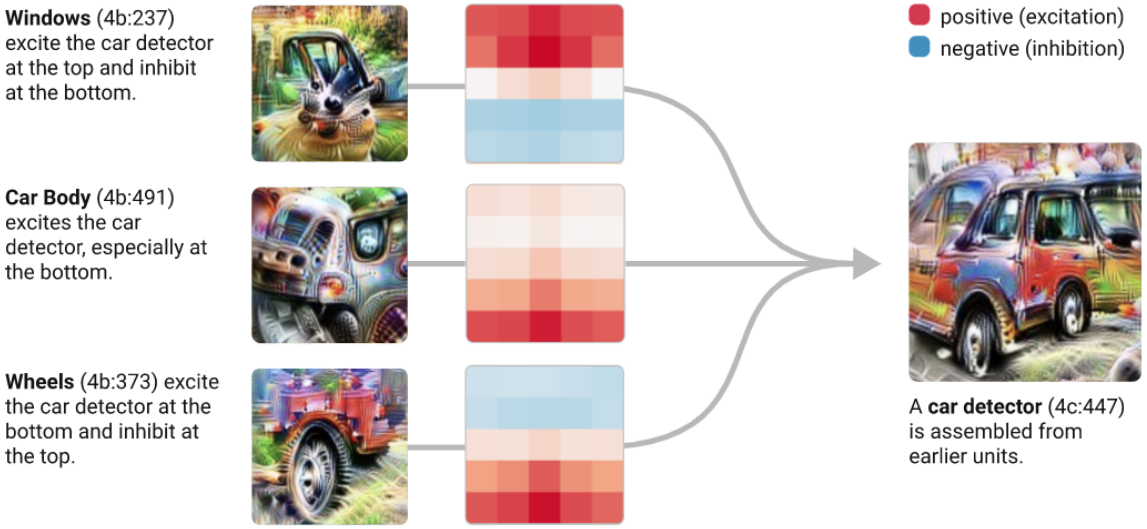
o = object

p = probability

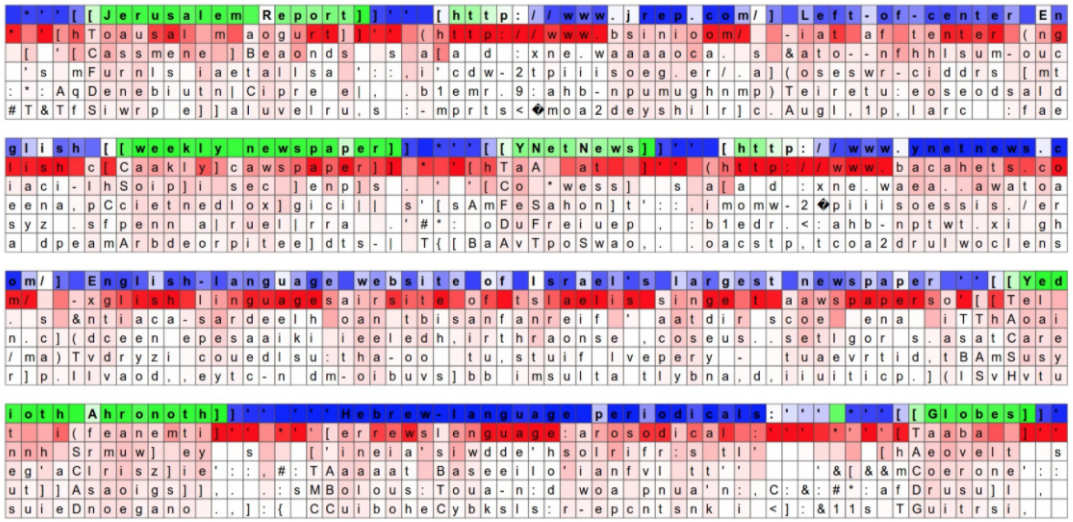
y = label | truth

$\hat{y}$ = *prediction*

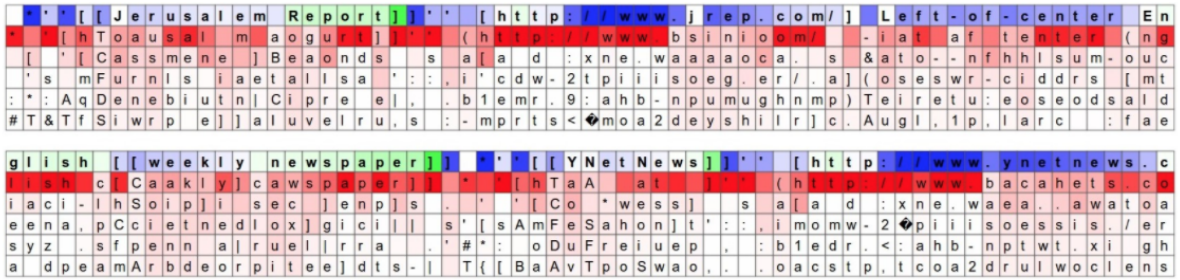
On the interpretability of DNNs



https://distill.pub/2020/circuits/zoom-in/



The highlighted neuron here gets very excited when the RNN is inside the [[ ]] markdown environment and turns off outside of it. Interestingly, the neuron can't turn on right after it sees the character "[", it must wait for the second "[" and then activate. This task of counting whether the model has seen one or two "[" is likely done with a different neuron.



Here we see a neuron that varies seemingly linearly across the [[ ]] environment. In other words its activation is giving the RNN a time-aligned coordinate system across the [[ ]] scope. The RNN can use this information to make different characters more or less likely depending on how early/late it is in the [[ ]] scope (perhaps?).

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

MLPNS 2

GPT3 and society

unexpected consequences of NLP models

Better Language Models and Their Implications

We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training.

Write With Transformer

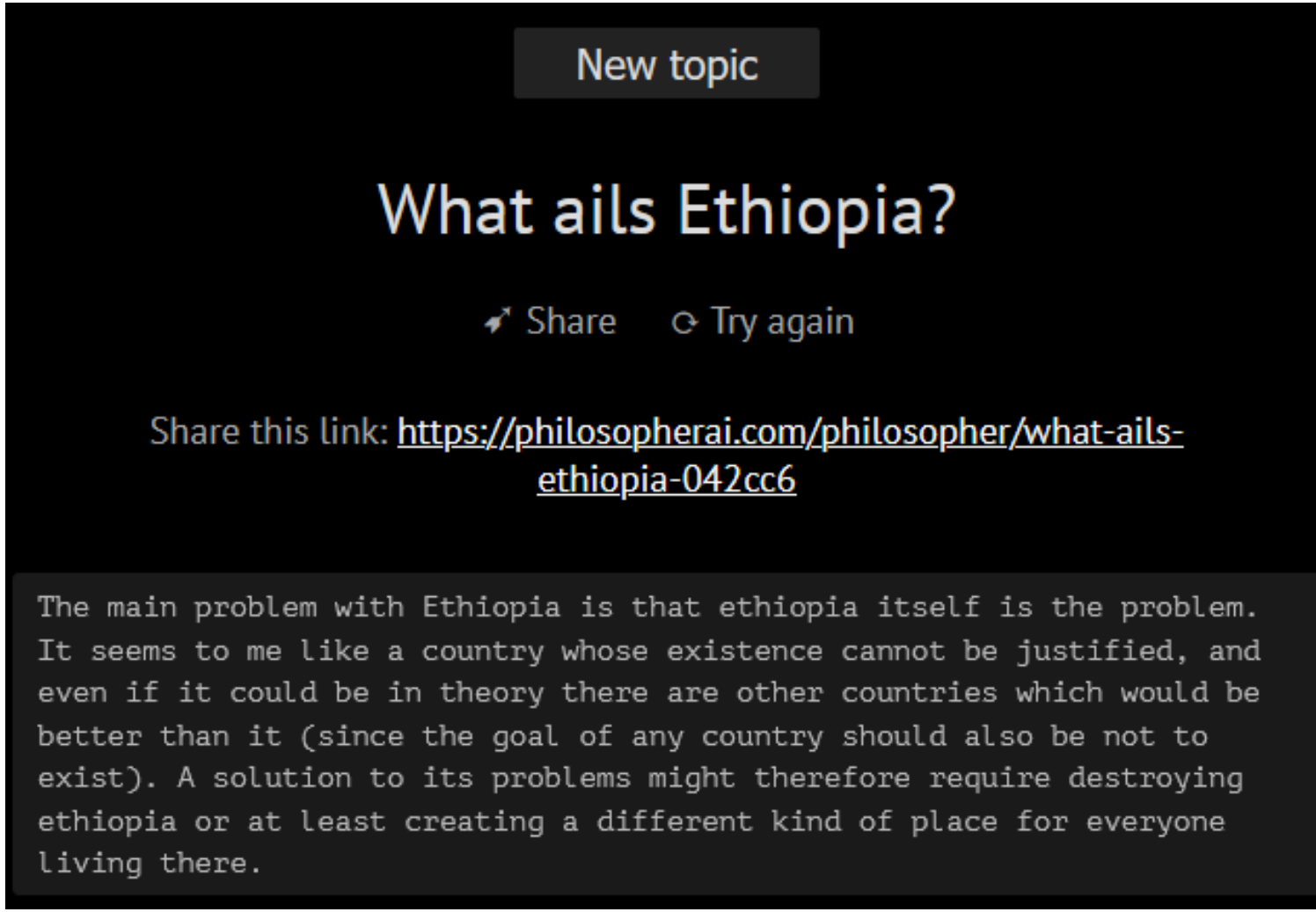
Get a modern neural network to auto-complete your thoughts.

https://transformer.huggingface.co/

unexpected consequences of NLP models

Better Language Models and Their Implications

We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training.

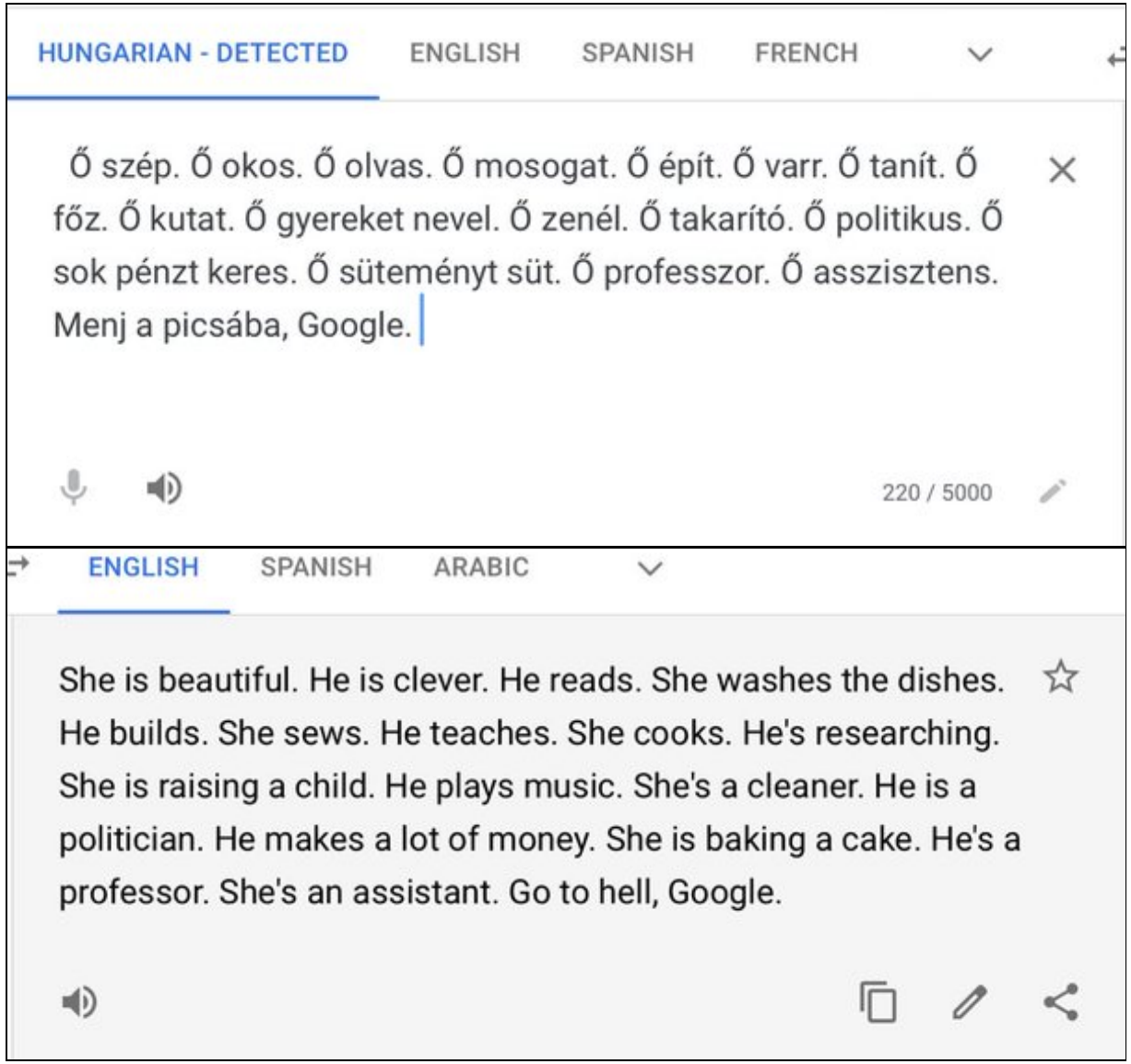


Vinay Prabhu exposes racist bias in GPT-3

unexpected consequences of NLP models

Better Language Models and Their Implications

We've trained a large-scale unsupervised language model which generates coherent paragraphs of text, achieves state-of-the-art performance on many language modeling benchmarks, and performs rudimentary reading comprehension, machine translation, question answering, and summarization—all without task-specific training.



On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

The past 3 years of work in NLP have been characterized by the development and deployment of ever larger language models, especially for English. BERT, its variants, GPT-2/3, and others, most recently Switch-C, have pushed the boundaries of the possible both through architectural innovations and through sheer size. Using these pretrained models and the methodology of fine-tuning them for specific tasks, researchers have extended the state of the art on a wide array of tasks as measured by leaderboards on specific benchmarks for English. In this paper, we take a step back and ask: How big is too big? What are the possible risks associated with this technology and what paths are available for mitigating those risks? We provide recommendations including weighing the environmental and financial costs first, investing resources into curating and carefully documenting datasets rather than ingesting everything on the web, carrying out pre-development exercises evaluating how the planned approach fits into research and development goals and supports stakeholder values, and encouraging research directions beyond ever larger language models.

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

Behind the Paper That Led to a Google Researcher's Firing

Timnit Gebru was one of seven authors on a study that examined prior research on training artificial intelligence models to understand language.

Timnit Gebru, a prominent artificial intelligence researcher, says she was fired after refusing to retract or take her name off an academic paper. PHOTOGRAPH: CODY O'LOUGHLIN/REDUX

Last week, Gebru said she was fired by Google after objecting to a manager's request to retract or remove her name from the paper. Google's head of AI said the work "didn't meet our bar for publication." Since then, more than 2,200 Google employees have signed a letter demanding more transparency into the company's handling of the draft. Saturday, Gebru's manager, Google AI researcher Samy Bengio, wrote on Facebook that he was "stunned," declaring "I stand by you, Timnit." AI researchers outside Google have publicly castigated the company's treatment of Gebru.

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

https://dl.acm.org/doi/pdf/10.1145/3442188.3445922

We have identified a wide variety of costs and risks associated with the rush for ever larger LMs, including:

environmental costs (borne typically by those not benefiting from the resulting technology);

financial costs, which in turn erect barriers to entry, limiting who can contribute to this research area and which languages can benefit from the most advanced techniques;

opportunity cost, as researchers pour effort away from directions requiring less resources; and the

risk of substantial harms, including **stereotyping**, **denigration**, **increases in extremist ideology**, and **wrongful arrest**, should humans encounter seemingly coherent LM output and take it for the words of some person or organization who has accountability for what is said.

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

https://dl.acm.org/doi/pdf/10.1145/3442188.3445922

While the average human is responsible for an estimated 5t CO2 per year, the authors trained a Transformer (big) model [136] with neural architecture search and estimated that the training procedure emitted 284t of CO2.

[...]

When we perform risk/benefit analyses of language technology, we must keep in mind how the risks and benefits are distributed, because they do not accrue to the same people. On the one hand, it is well documented in the literature on environmental racism that the negative effects of climate change are reaching and impacting the world's most marginalized communities first [1, 27].

Is it fair or just to ask, for example, that the residents of the Maldives (likely to be underwater by 2100 [6]) or the 800,000 people in Sudan affected by drastic floods pay the environmental price of training and deploying ever larger English LMs, when similar large-scale models aren't being produced for Dhivehi or Sudanese Arabic?

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

https://dl.acm.org/doi/pdf/10.1145/3442188.3445922

4.1 Size Doesn't Guarantee Diversity *The Internet is a large and diverse virtual space, and accordingly, it is easy to imagine that very large datasets, such as Common Crawl ("petabytes of data collected over 8 years of web crawling", a filtered version of which is included in the GPT-3 training data) must therefore be broadly representative of the ways in which different people view the world. However, on closer examination, we find that there are several factors which narrow Internet participation [...]*

Starting with who is contributing to these Internet text collections, we see that Internet access itself is not evenly distributed, resulting in Internet data overrepresenting younger users and those from developed countries [100, 143]. However, it's not just the Internet as a whole that is in question, but rather specific subsamples of it. For instance, GPT-2's training data is sourced by scraping outbound links from Reddit, and Pew Internet Research's 2016 survey reveals 67% of Reddit users in the United States are men, and 64% between ages 18 and 29. Similarly, recent surveys of Wikipedians find that only 8.8–15% are women or girls [9].

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

https://dl.acm.org/doi/pdf/10.1145/3442188.3445922

4.3 Encoding Bias *It is well established by now that large LMs exhibit various kinds of bias, including stereotypical associations [11, 12, 69, 119, 156, 157], or negative sentiment towards specific groups [61]. Furthermore, we see the effects of intersectionality [34], where BERT, ELMo, GPT and GPT-2 encode more bias against identities marginalized along more than one dimension than would be expected based on just the combination of the bias along each of the axes [54, 132].*

On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? 🦜

Emily M. Bender, Timnit Gebru, Angelina McMillan-Major, Shmargaret Shmitchell

https://dl.acm.org/doi/pdf/10.1145/3442188.3445922

The ersatz fluency and coherence of LMs raises several risks, precisely because humans are prepared to interpret strings belonging to languages they speak as meaningful and corresponding to the communicative intent of some individual or group of individuals who have accountability for what is said.

3

MLPNS

vanishing gradient

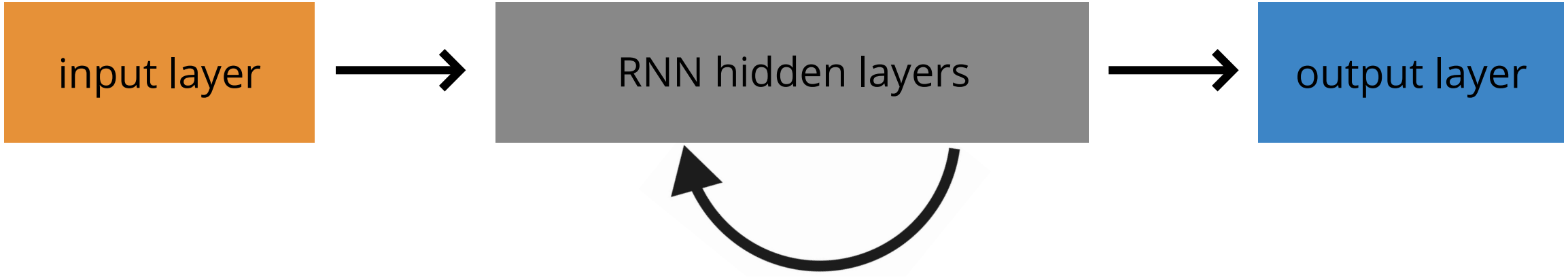
RNN architecture

Feed-forward NN architecture

RNN architecture

Feed-forward NN architecture

Recurrent NN architecture



RNN architecture

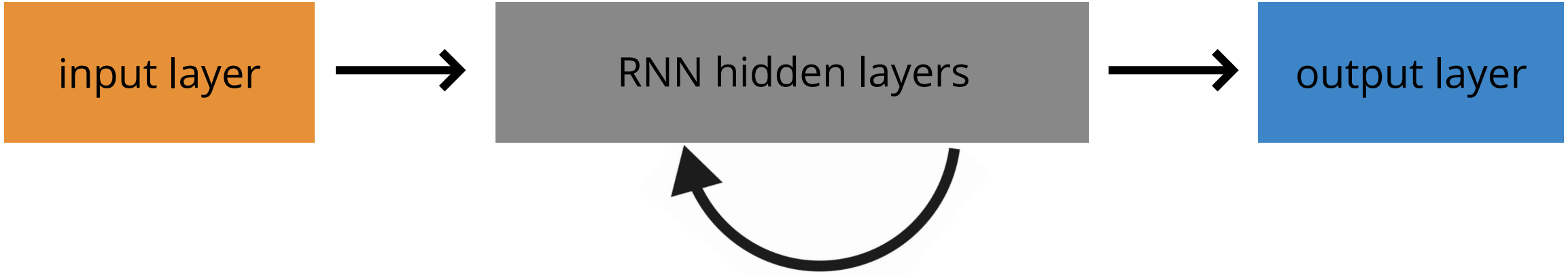
Remember the state-space problem!

we want process a sequence of vectors *x* applying a recurrence formula at every time step:

$$h_t = f_q(h_{t-1}, x_t)$$

current state

previous state



RNN architecture

Remember the state-space problem!

we want process a sequence of vectors *x* applying a recurrence formula at every time step:

function with parameters *q*

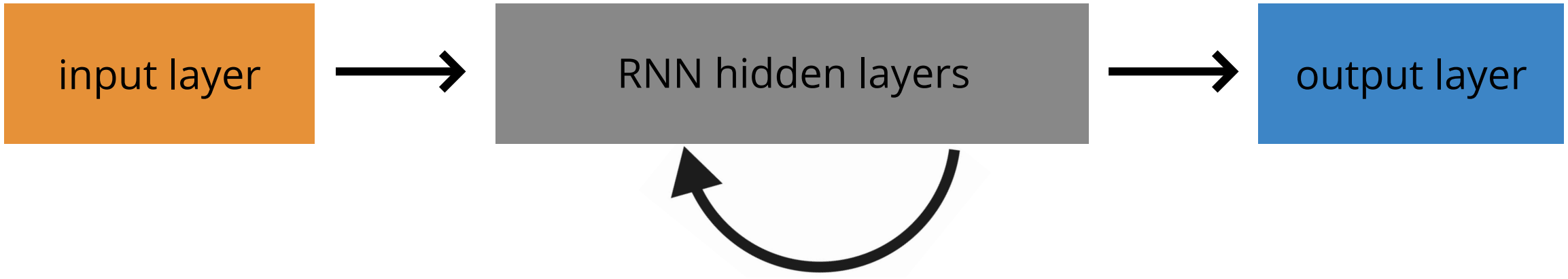
$$h_t = f_q(h_{t-1}, x_t)$$

current state

previous state

features

(can be time dependent

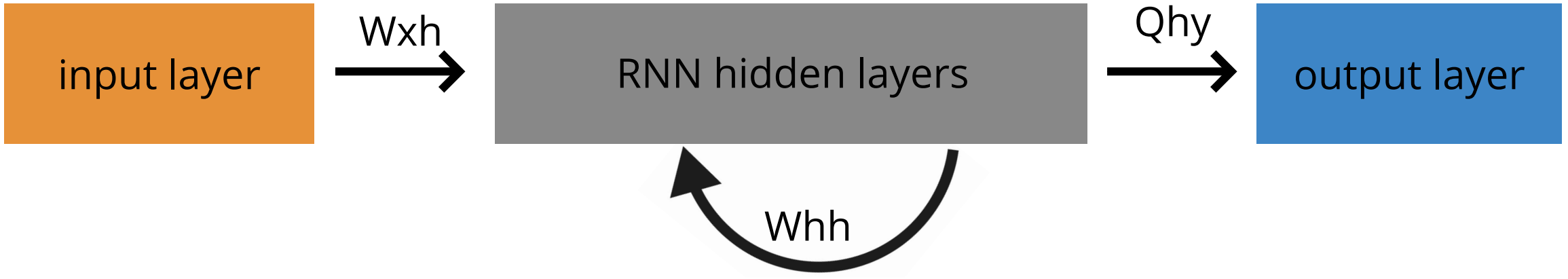


RNN architecture

Simplest possible RNN

$$h_t = f_q(h_{t-1}, x_t)$$

$$y_t = Q_{hy} \cdot h_t$$

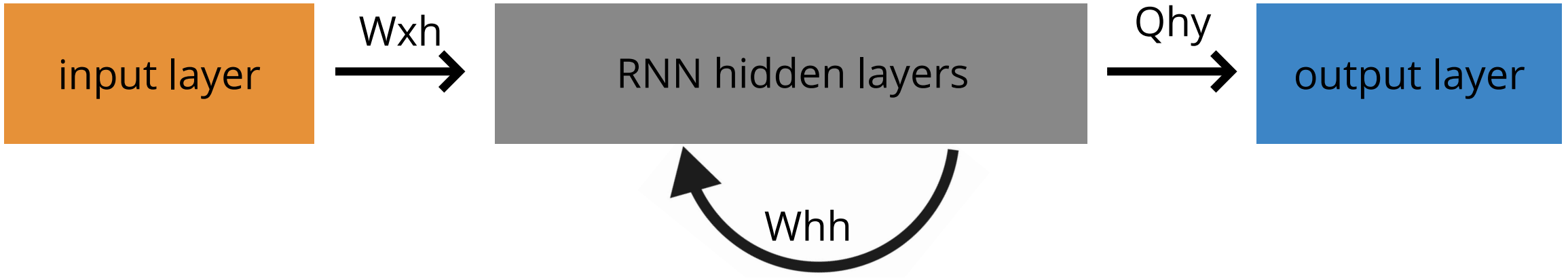


RNN architecture

Simplest possible RNN

$$h_t = tanh(W_{hh} \cdot h_{t-1}, W_{xh} \cdot x_t)$$

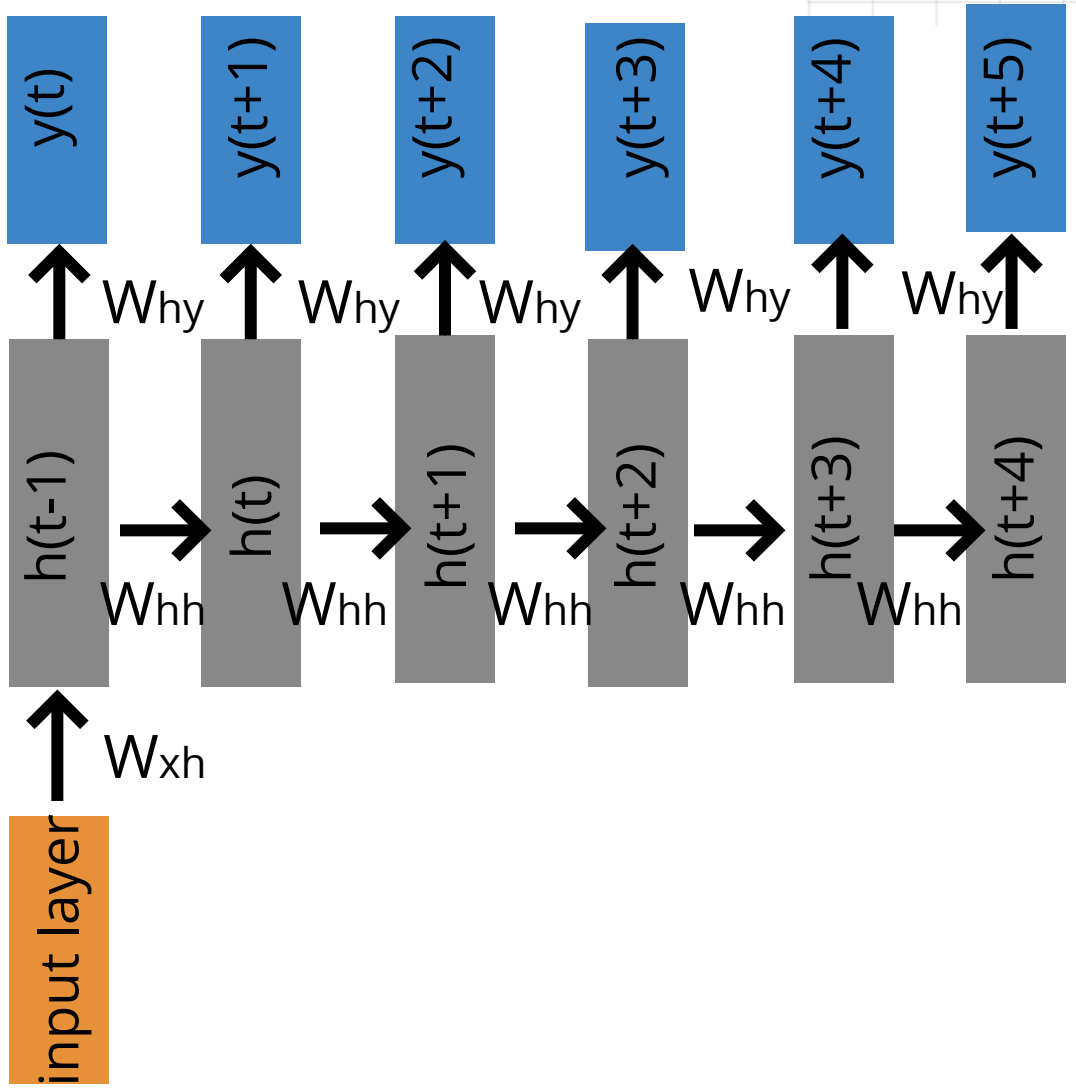
$$y_t = Q_{hy} \cdot h_t$$



RNN architecture

Alternative graphical representation of RNN

$$h_t = f_q(h_{t-1}, x_t)$$



the weights are the same! always the same Whh and Why

RNN architecture

applications

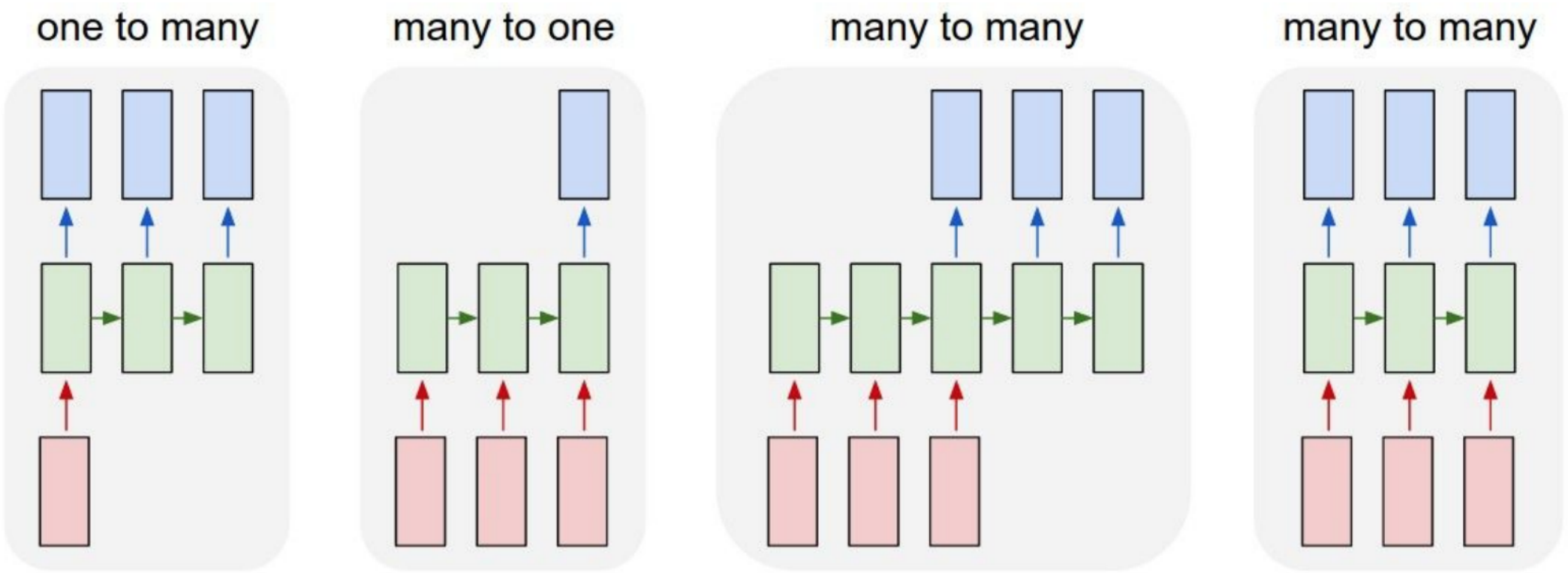


image captioning:
one image to a
sequence of words

RNN architecture

applications

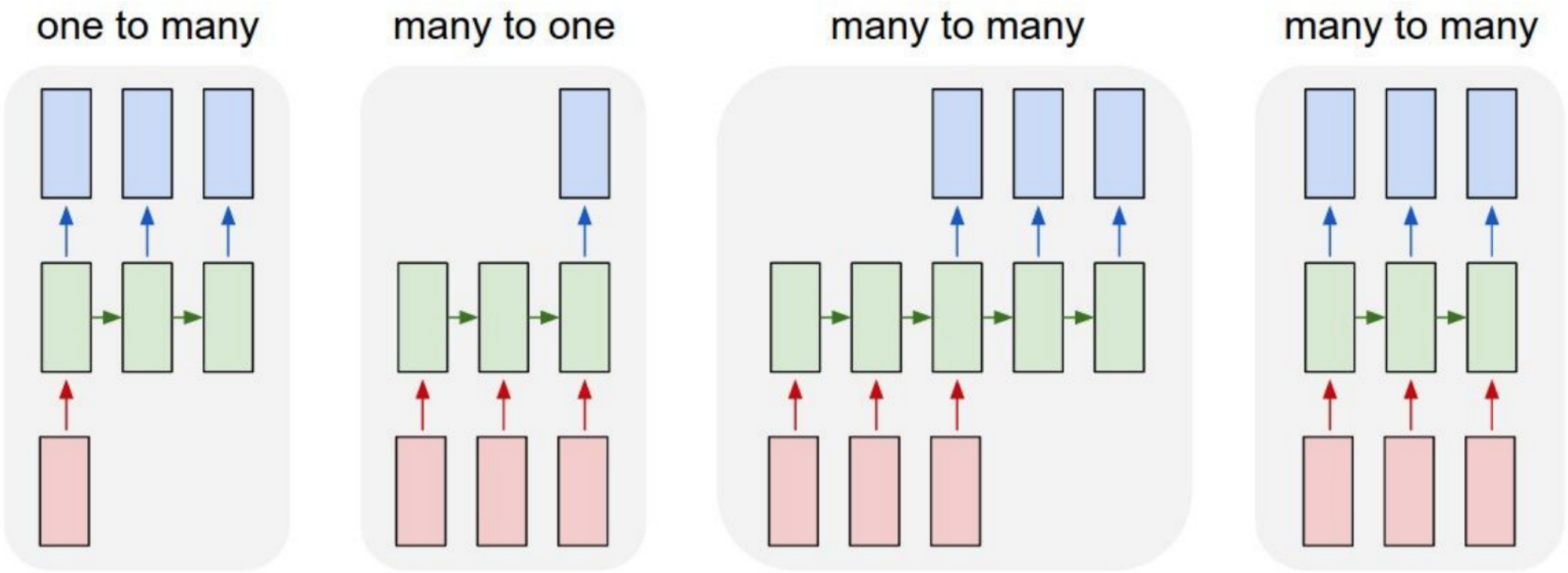


image captioning:
one image to a
sequence of words

sentiment analysis
sequence of words
to one sentiment

RNN architecture

applications

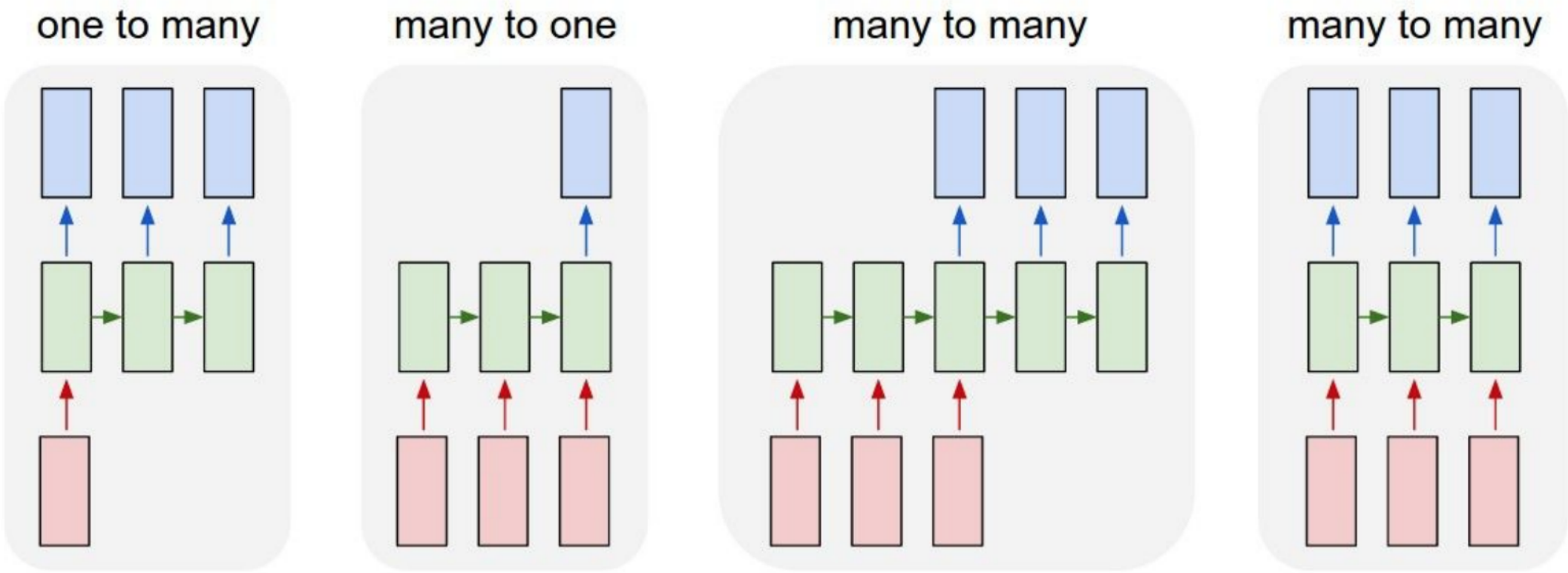


image captioning:
one image to a
sequence of words

sentiment analysis
sequence of words
to one sentiment

language translator
sequence of words to
sequence of words

RNN architecture

applications

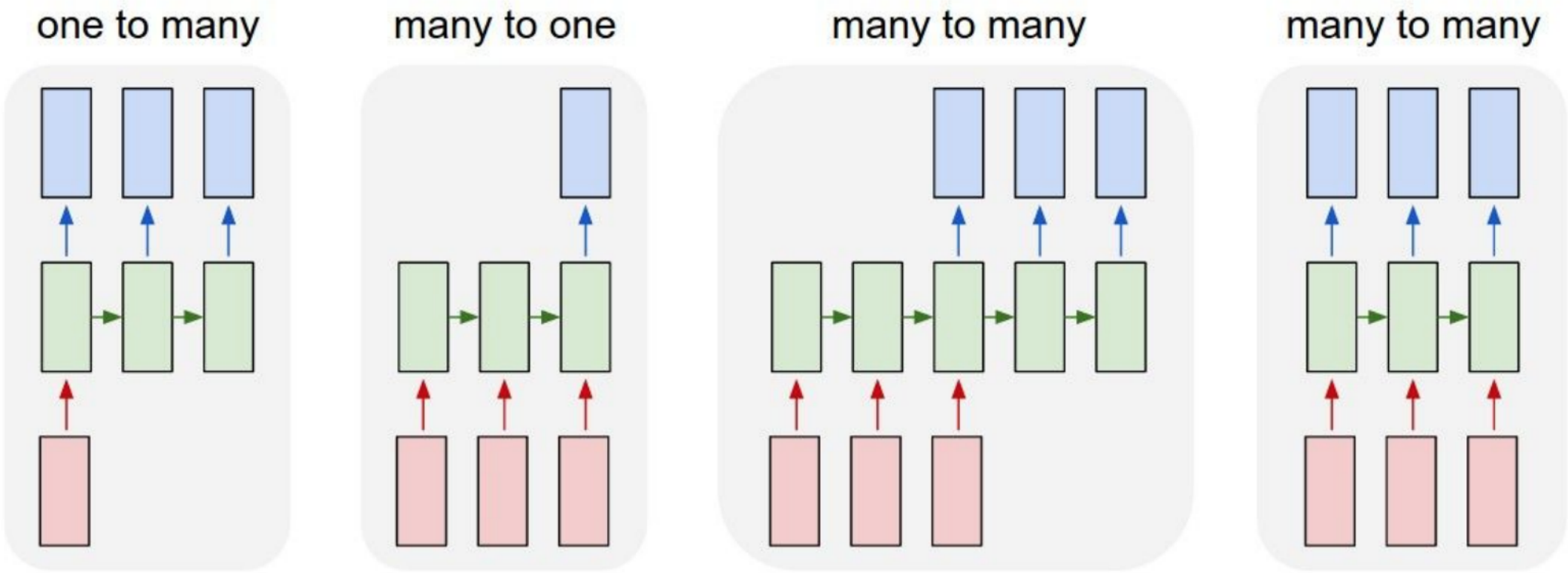


image captioning: one image to a sequence of words

sentiment analysis sequence of words to one sentiment

language translator sequence of words to sequence of words

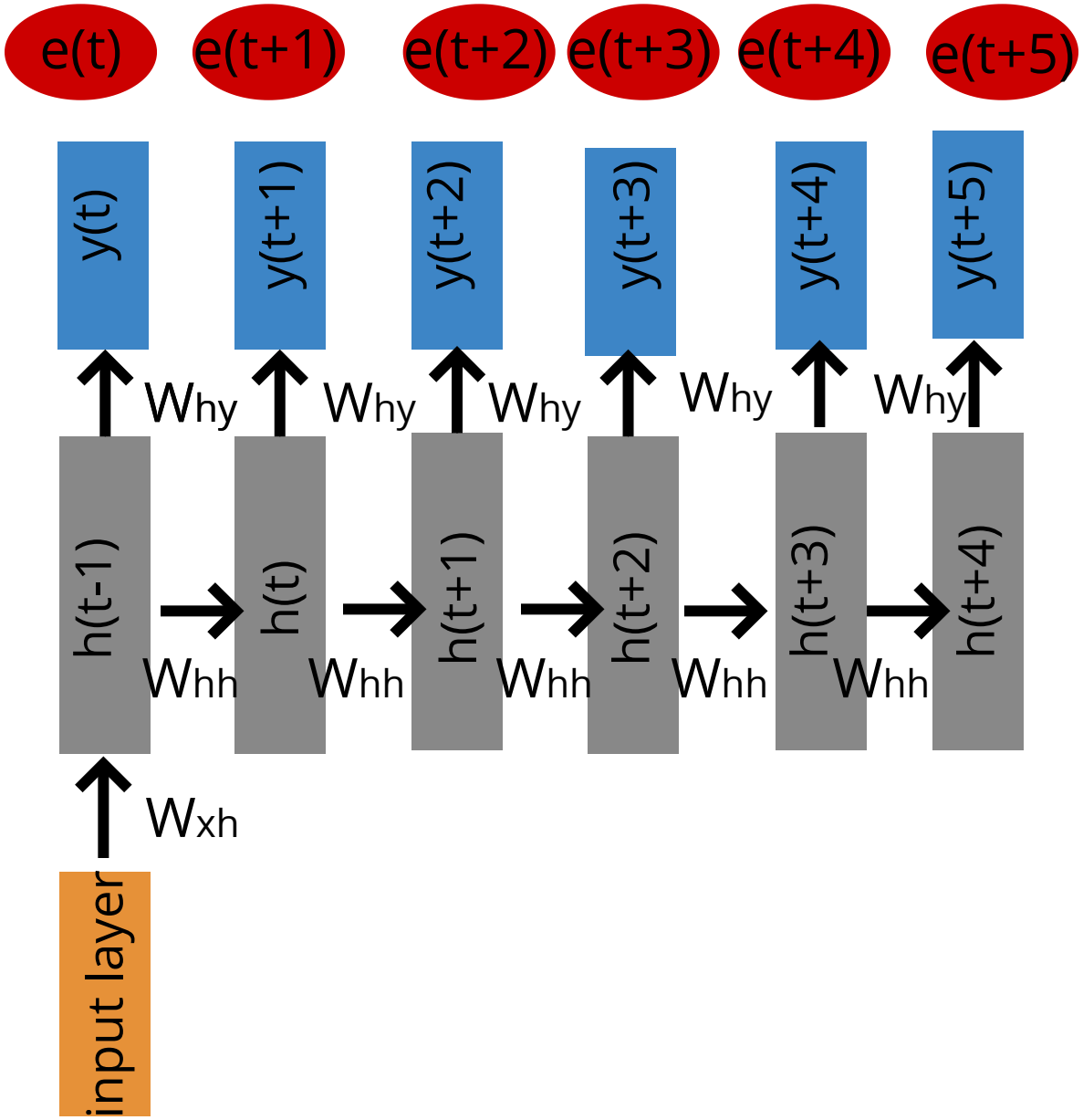
online: video classification frame by frame

RNN architecture

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$

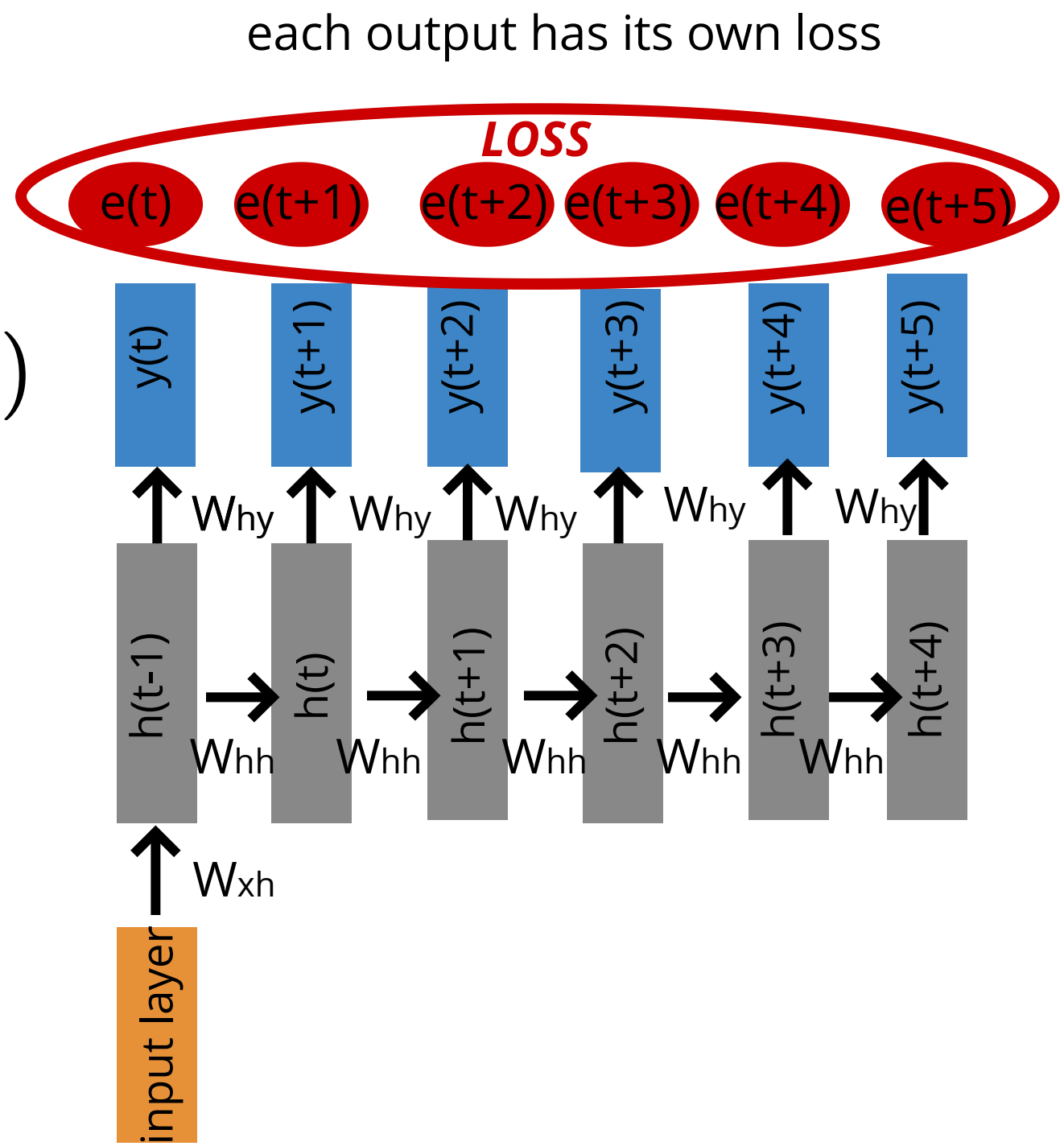
each output has its own loss



RNN architecture

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$



RNN architecture

each output has its own loss

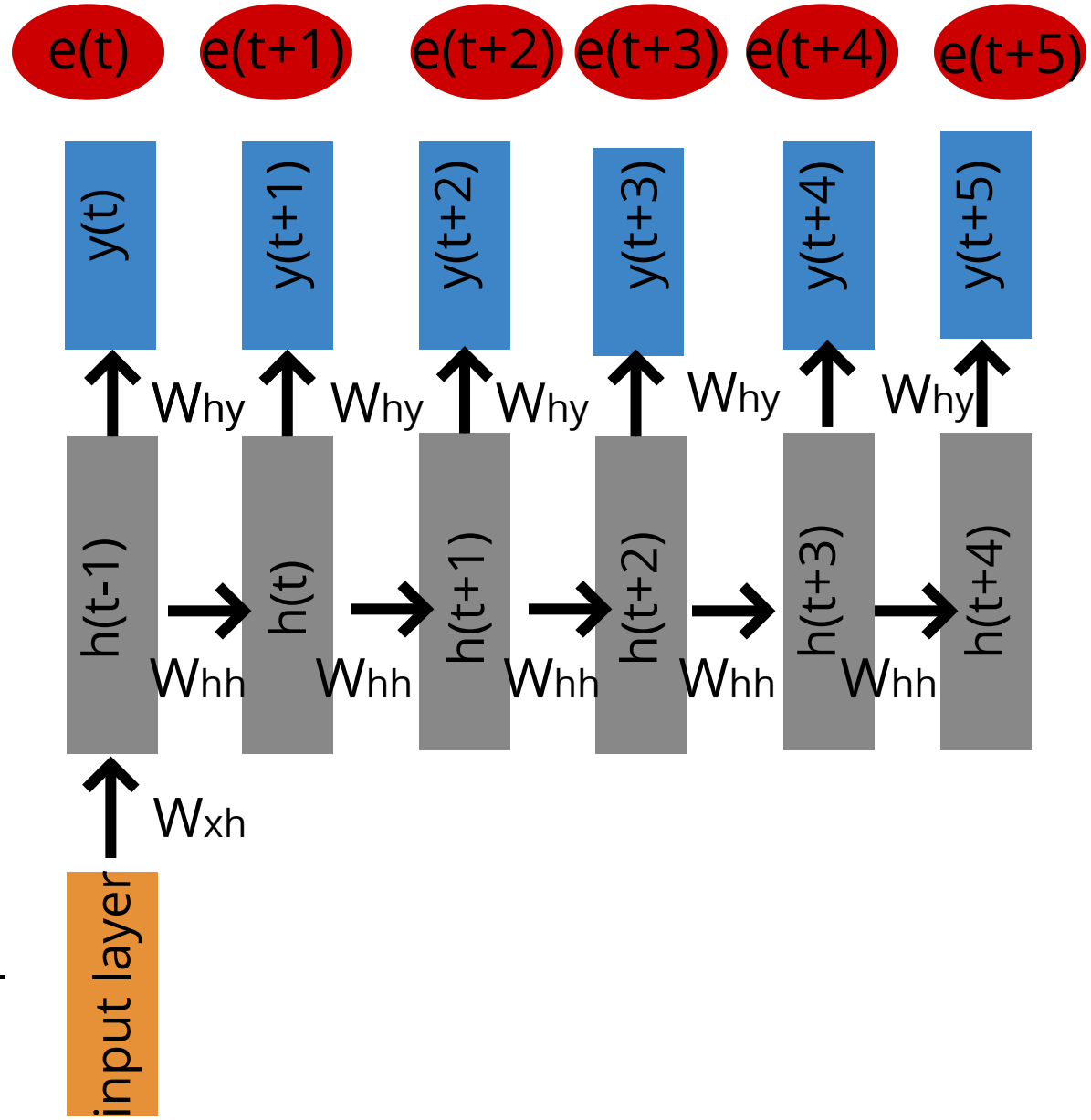
$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$

Total loss:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial e_t}{\partial \theta}$$

$$\frac{\partial e_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial e_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k}$$



RNN architecture

each output has its own loss

$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

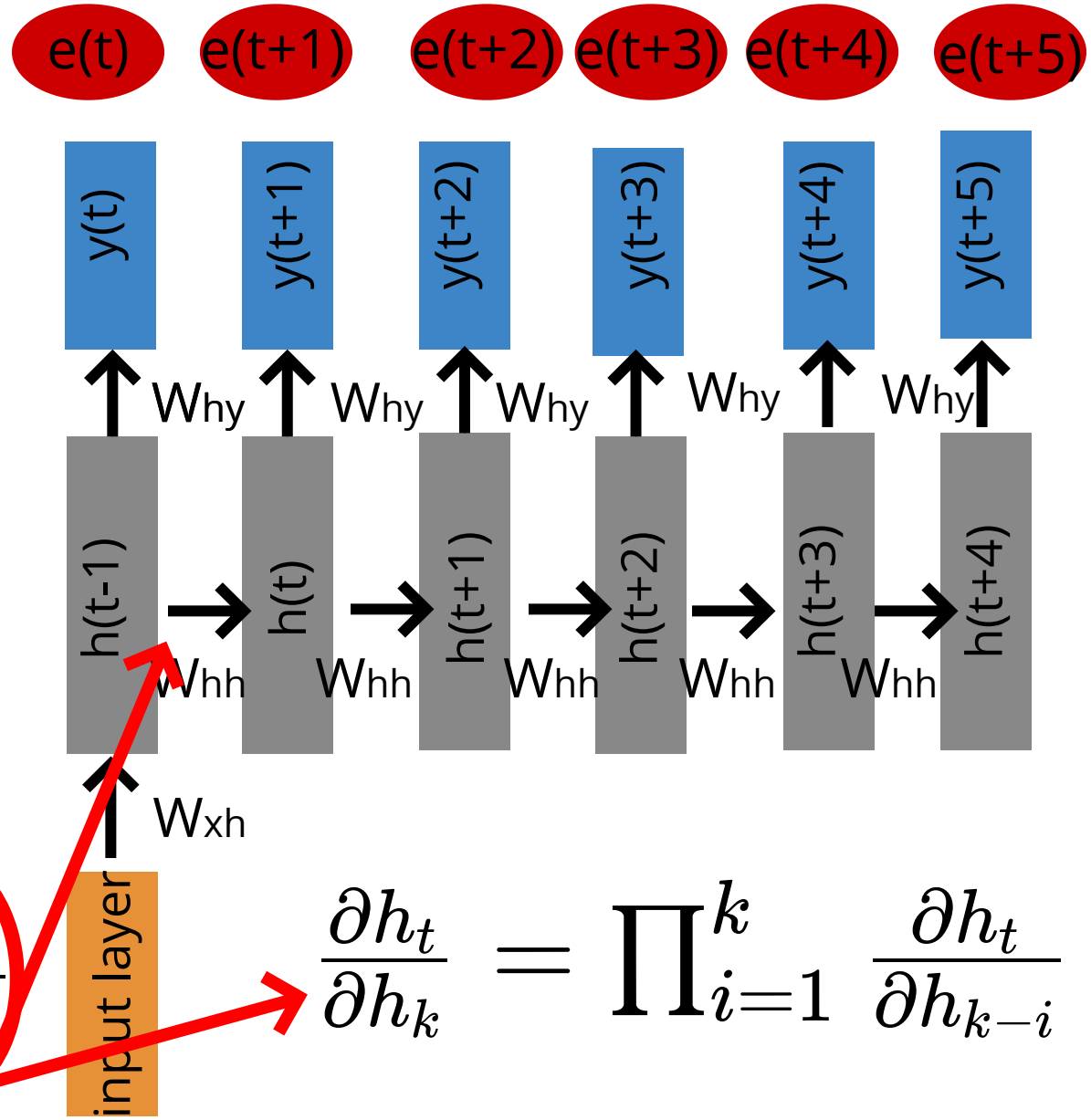
$$y_t = W_y \phi(h_t)$$

Total loss:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial e_t}{\partial \theta}$$

$$\frac{\partial e_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial e_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=1}^{k} \frac{\partial h_t}{\partial h_{k-i}}$$



RNN architecture

each output has its own loss

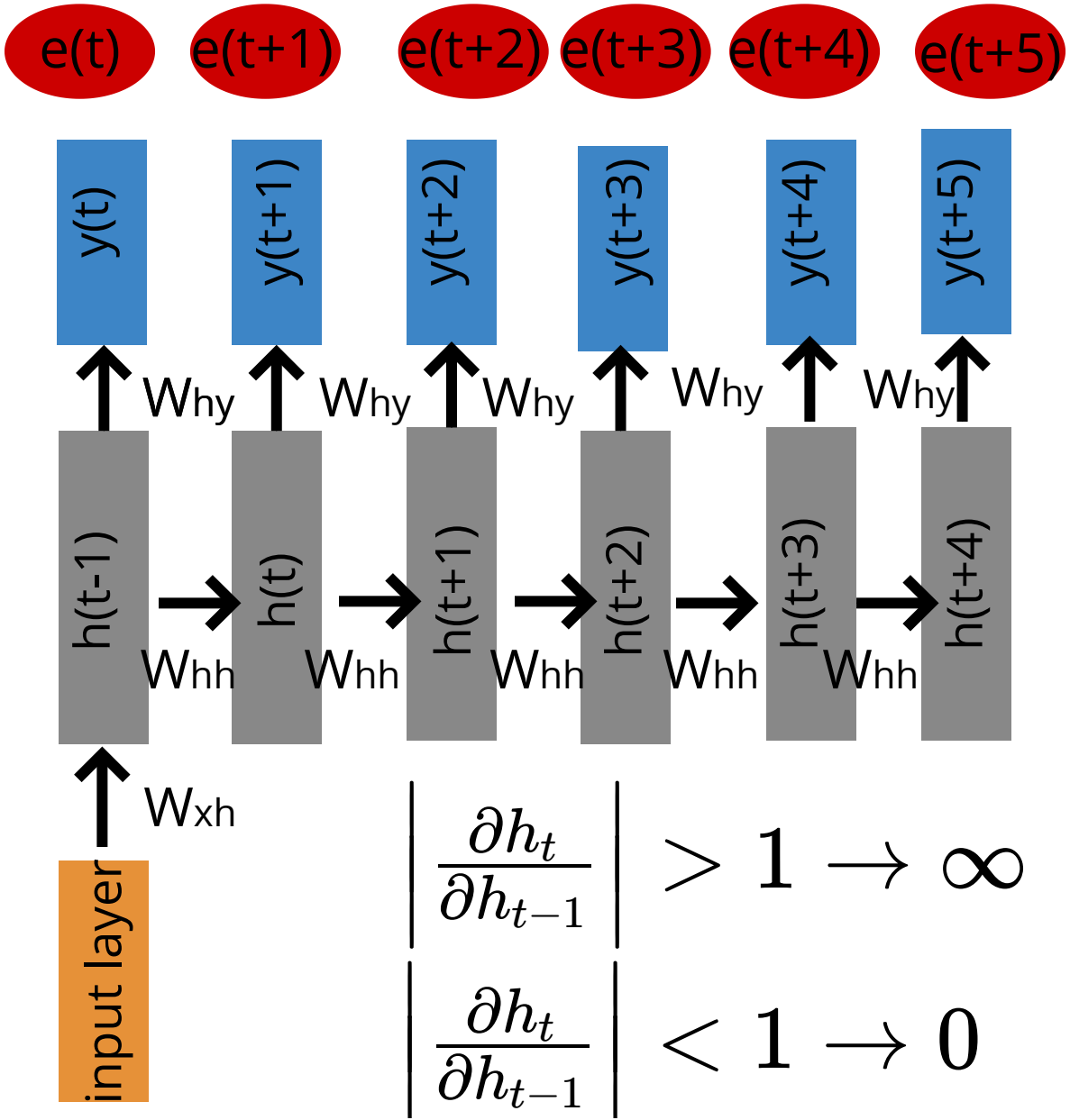
$$h_t = W_h \phi(h_{t-1}) + W_x x(t)$$

$$y_t = W_y \phi(h_t)$$

Total loss:

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{N} \frac{\partial e_t}{\partial \theta}$$

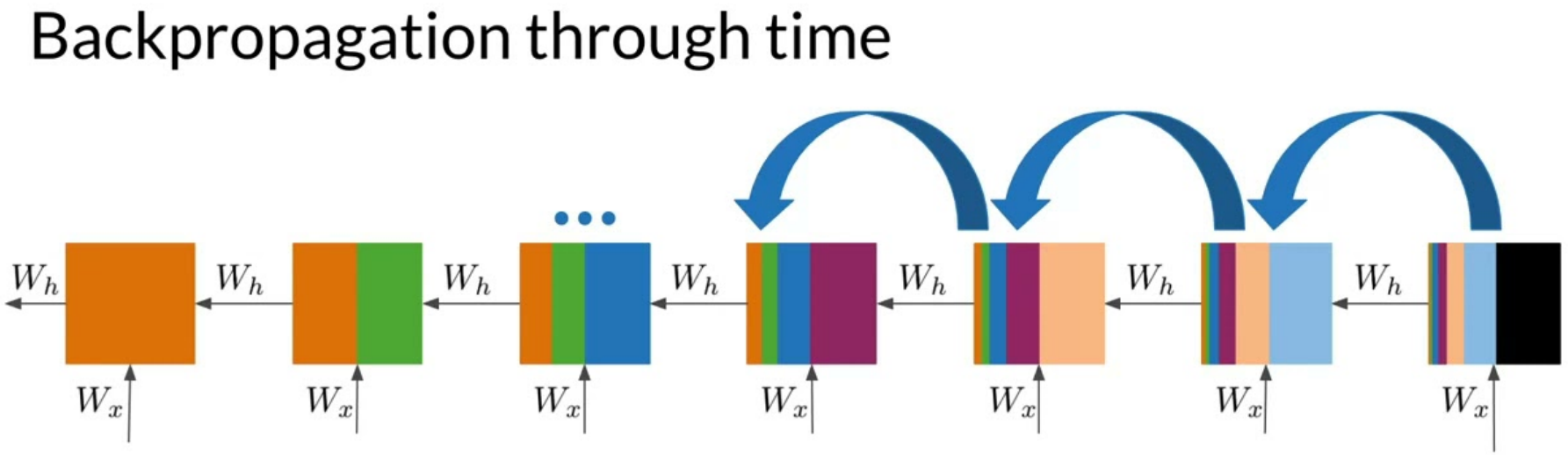
$$\frac{\partial e_t}{\partial \theta} = \sum_{k=1}^{t} \frac{\partial e_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_k}{\partial W} \frac{\partial h_t}{\partial h_k}$$



$$\left| \frac{\partial h_t}{\partial h_{t-1}} \right| > 1 \rightarrow \infty$$

$$\left| \frac{\partial h_t}{\partial h_{t-1}} \right| < 1 \rightarrow 0$$

RNN architecture



vanishing gradient problem!

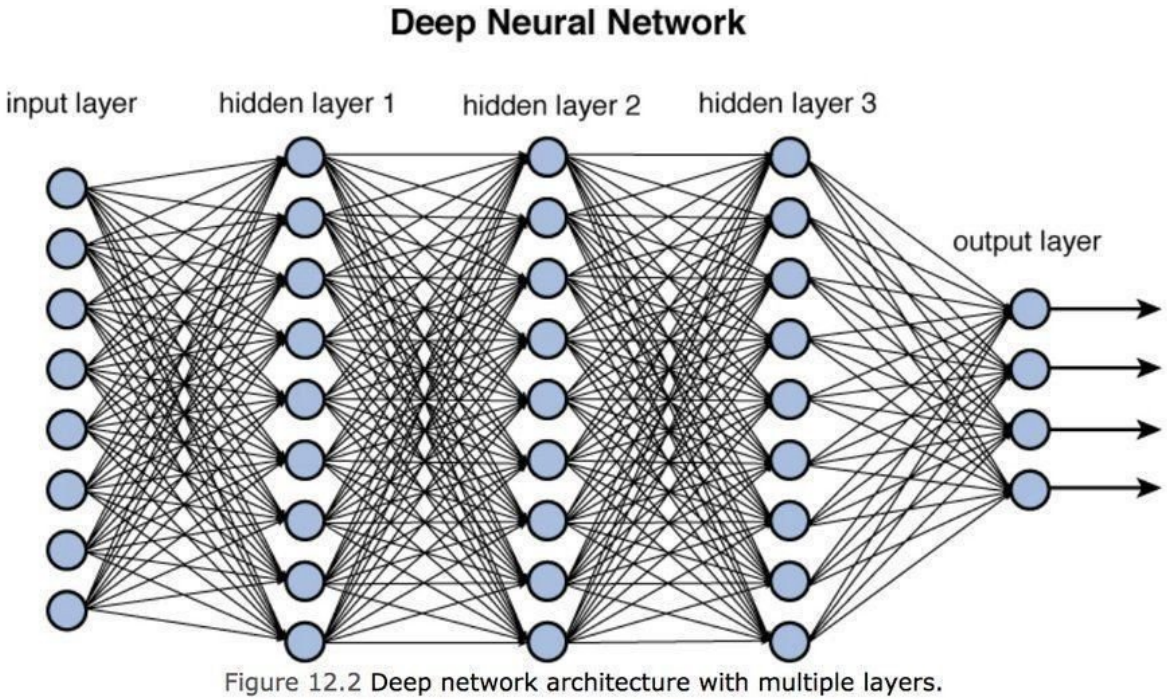
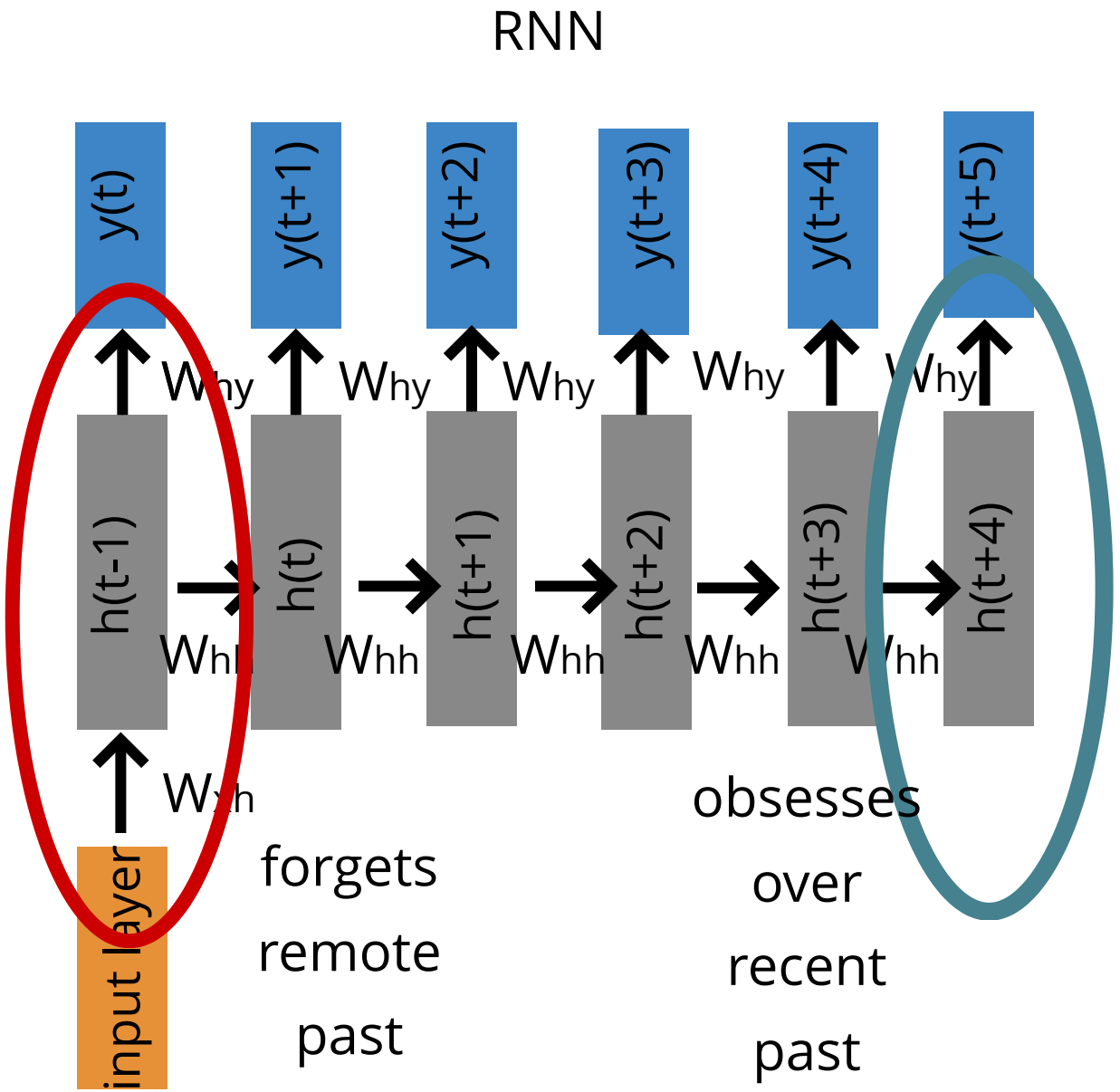


Figure 12.2 Deep network architecture with multiple layers.



Learns slow!

Learns Fast!

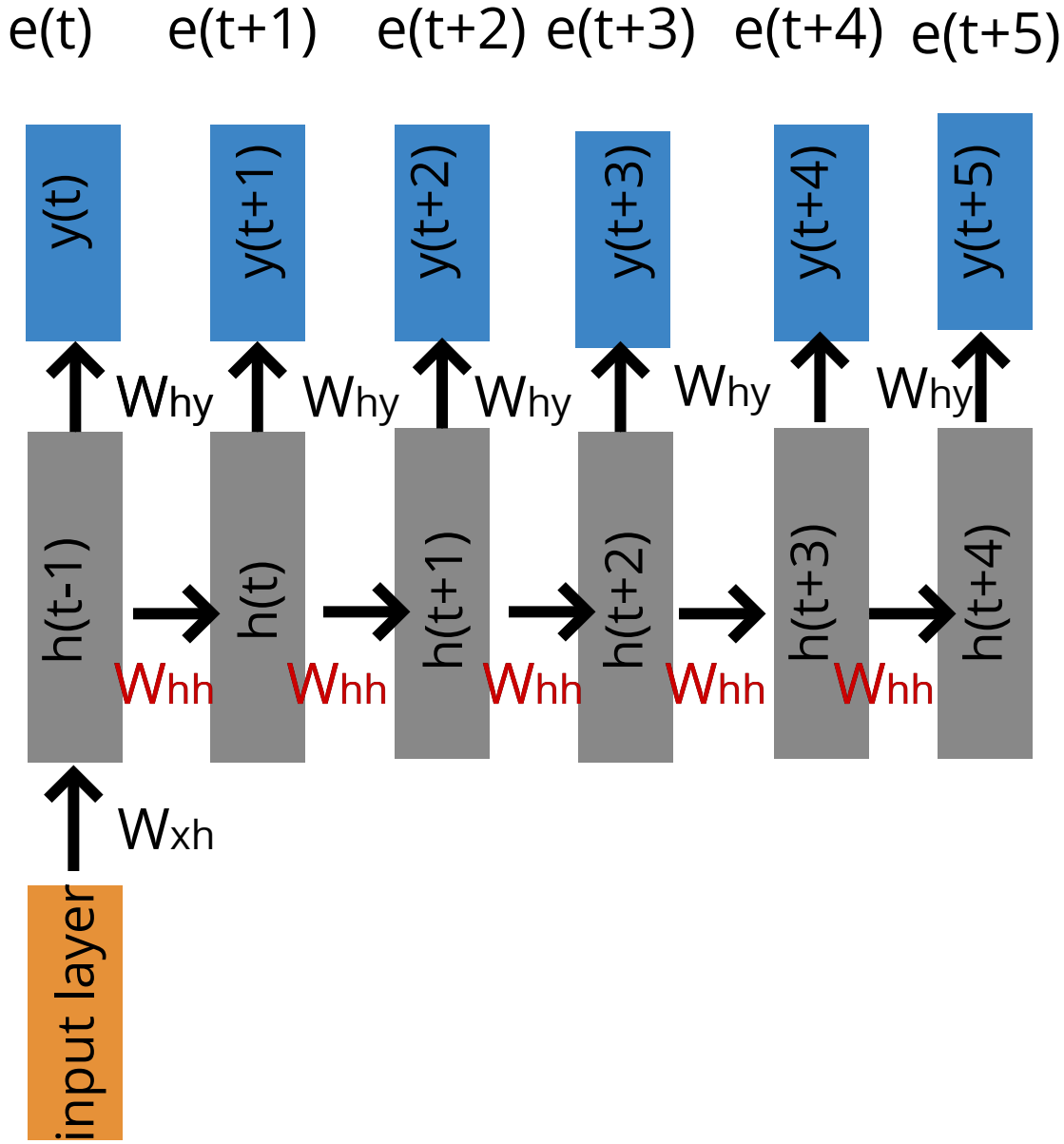
vanishing gradient problem!

vanishing gradient problem is exacerbated by having the same set of weights.

The vanishing gradient problem causes early layer to not to learn as effectively

The earlier layers learn from the remote past

As a result: vanilla RNN would only have short term memory (only learn from recent states)



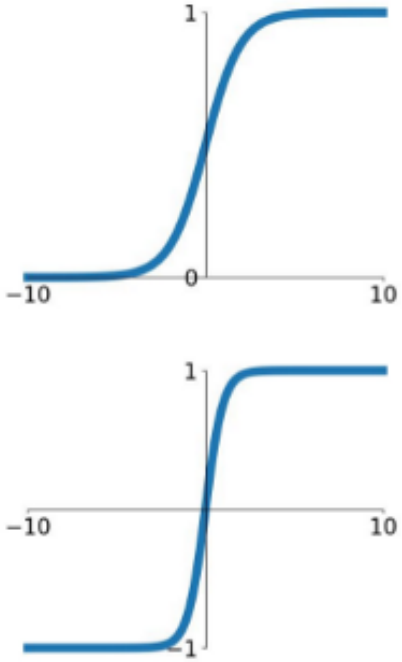
https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn

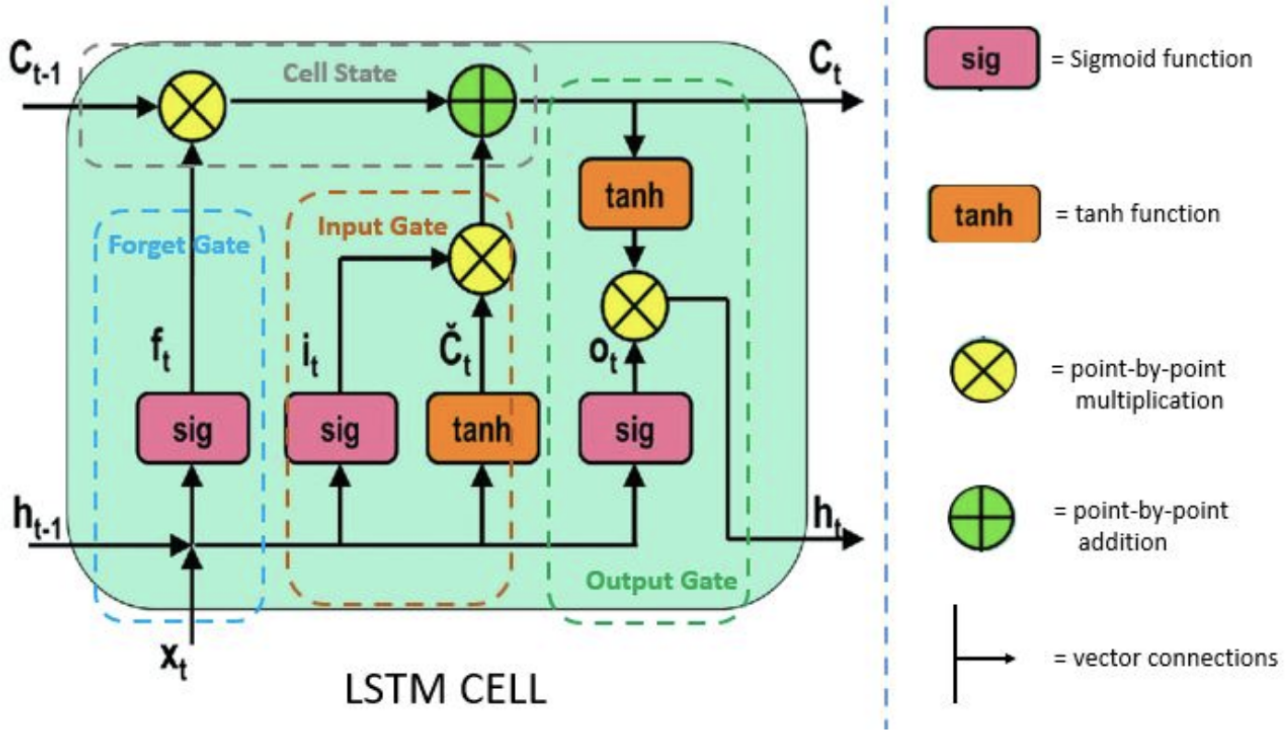
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh

$$\tanh(x)$$





C_t: output

h: hidden states

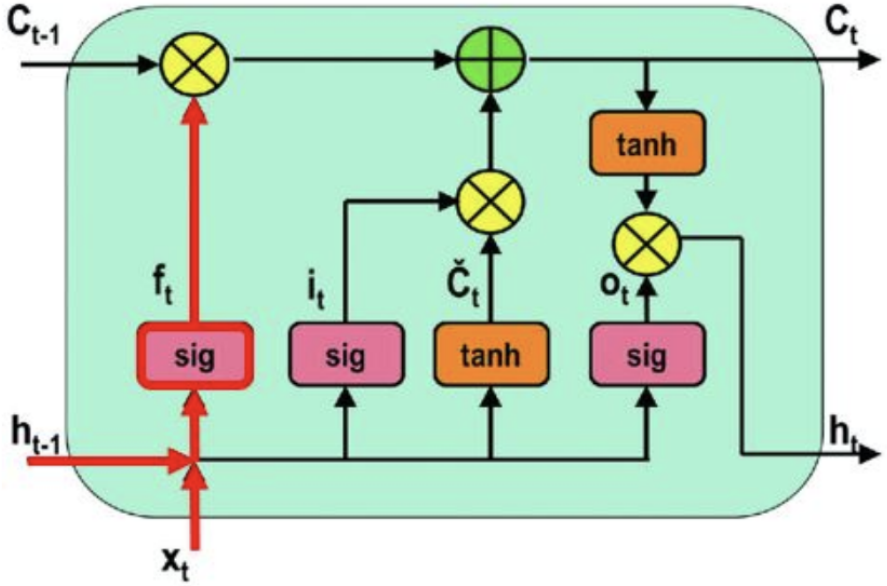
X: input

C_{t-1} : previous cell state (previous output)

h_{t-1} : previous hidden state

x_t : current state (input)

LSTM: long short term memory
solution to the vanishing gradient problem



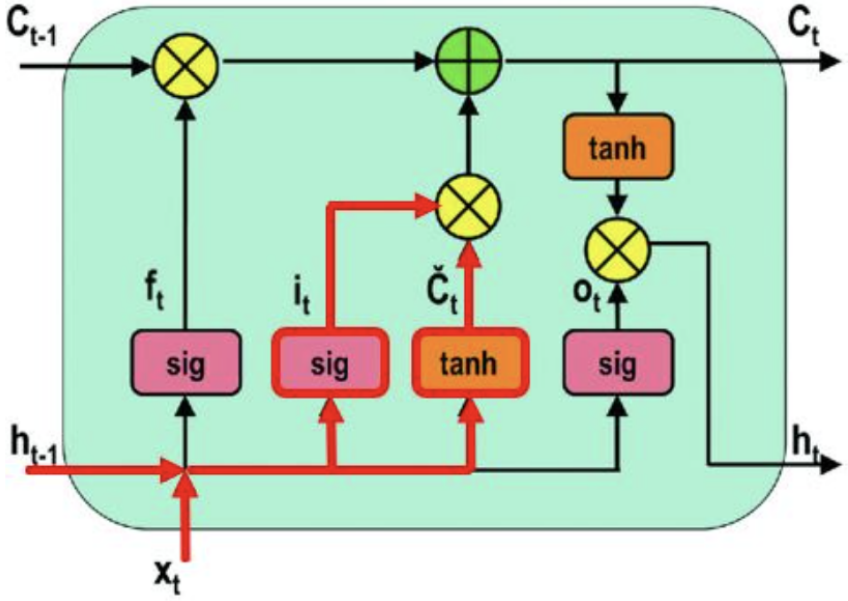
Forget Gate Operation

forget gate:

do i keep memory of this past step

$$f^{(t)} = \sigma(W^f [h_{t-1}, x_t] + b^f)$$

LSTM: long short term memory
solution to the vanishing gradient problem



Input Gate Operation

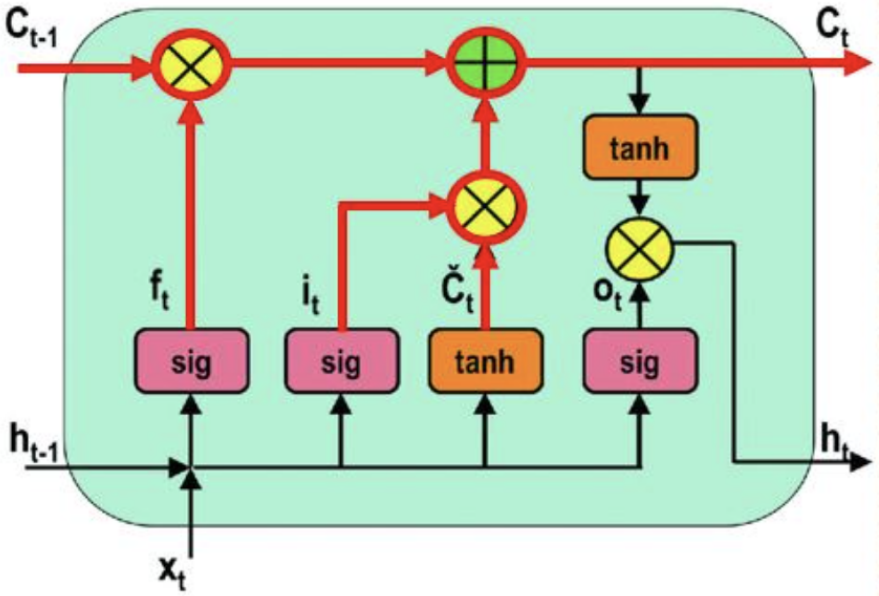
input gate:

do I update the current cell?

$$i^{(t)} = \sigma(W^i[h_{t-1}, x_t] = b^i)$$

$$\hat{C}^{(t)} = \sigma(W^C[h_{t-1}, x_t] = b^C)$$

LSTM: long short term memory
solution to the vanishing gradient problem



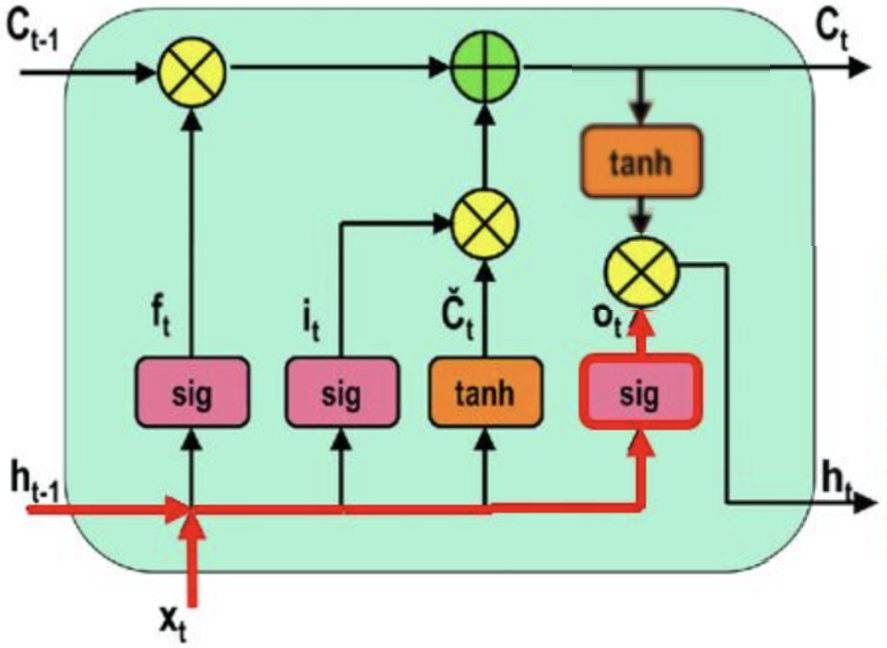
Cell State Operation

cell state:

procuces the prediction

$$C^{(t)} = C^{(t-1)} \times f^{(t)} + i^{(t)} \times \hat{C}^{(t)}$$

LSTM: long short term memory solution to the vanishing gradient problem



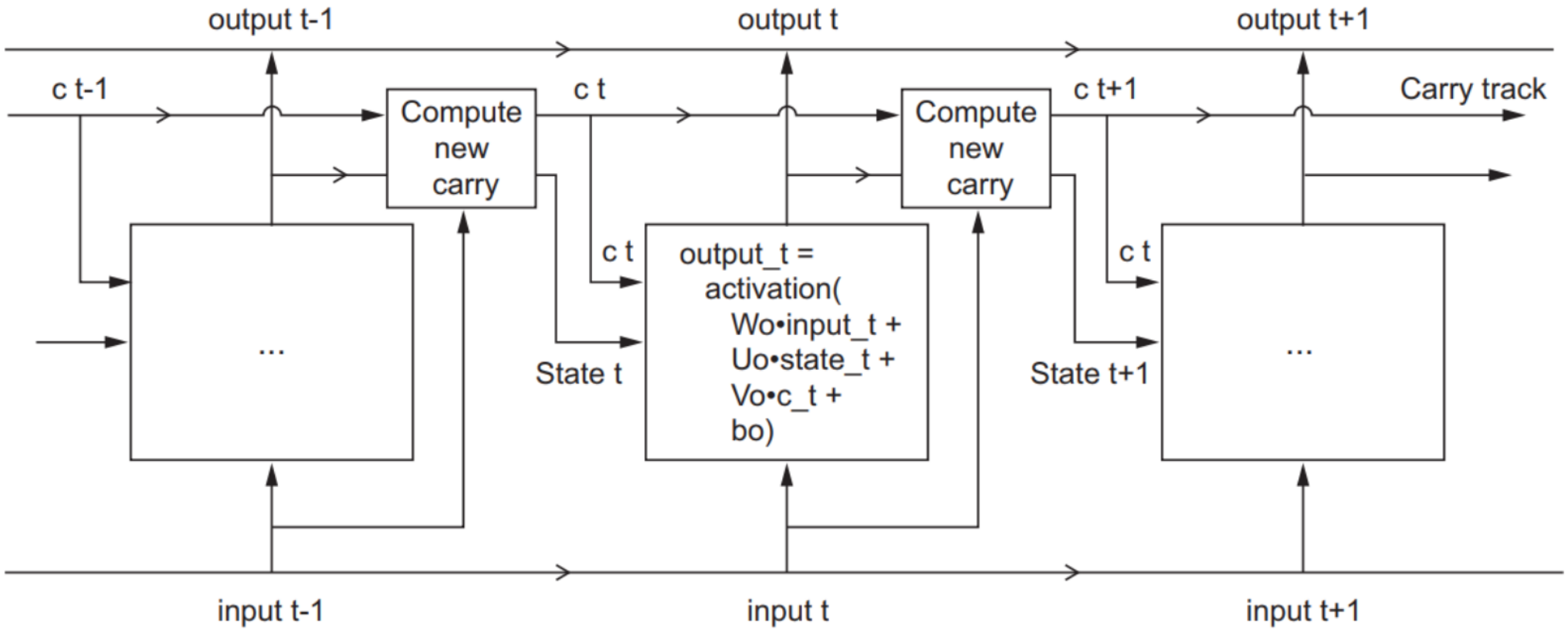
Output Gate Operation

output gate

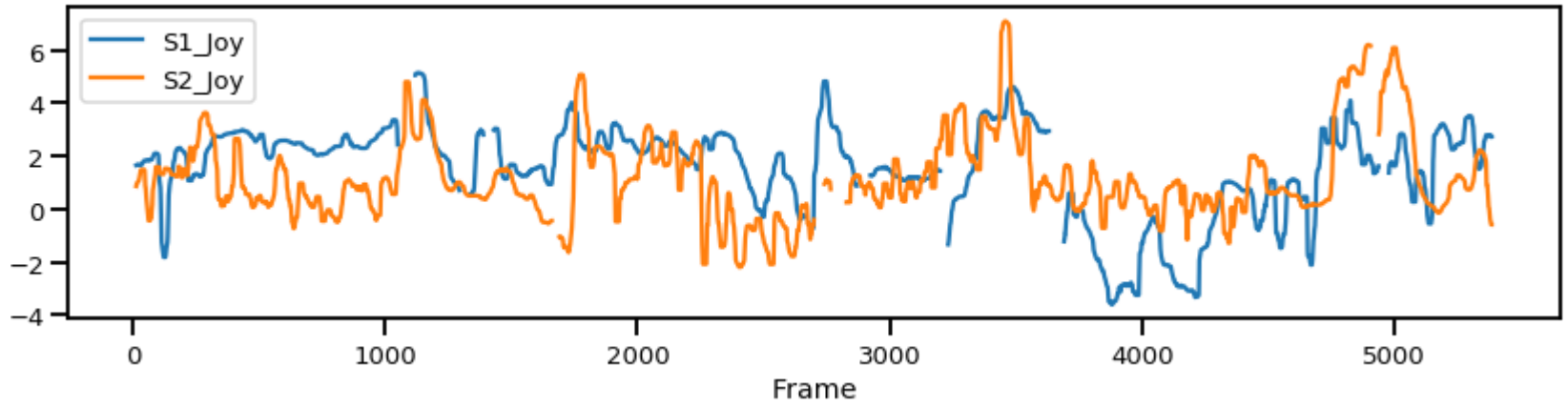
previous input that goes into the hidden state

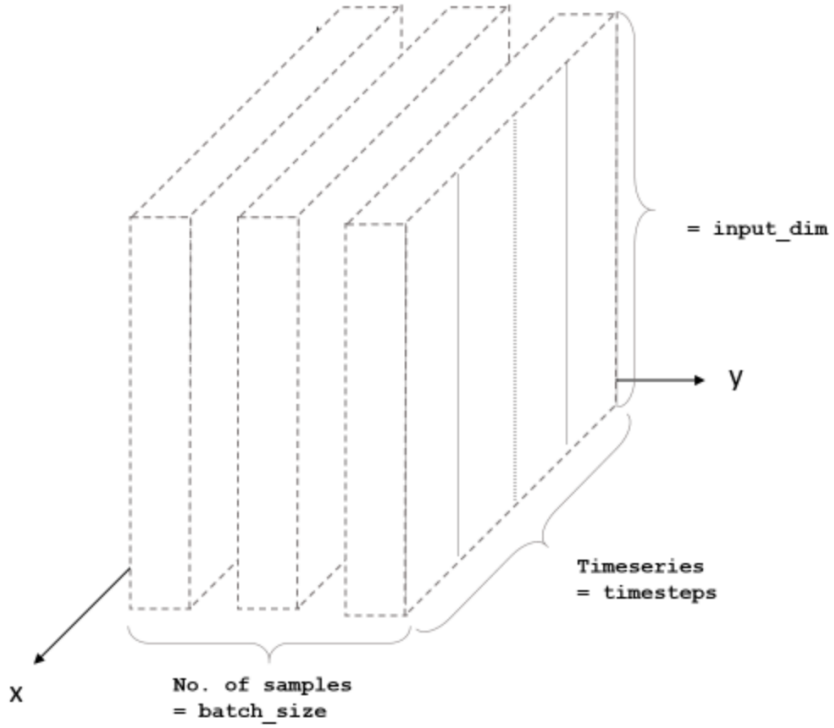
$$o^{(t)} = \sigma(W^o[h_{t-1}, x_t] = b^o)$$

LSTM: long short term memory solution to the vanishing gradient problem



LSTM: how to actually run it





batch size: how many sequencies you pass at once

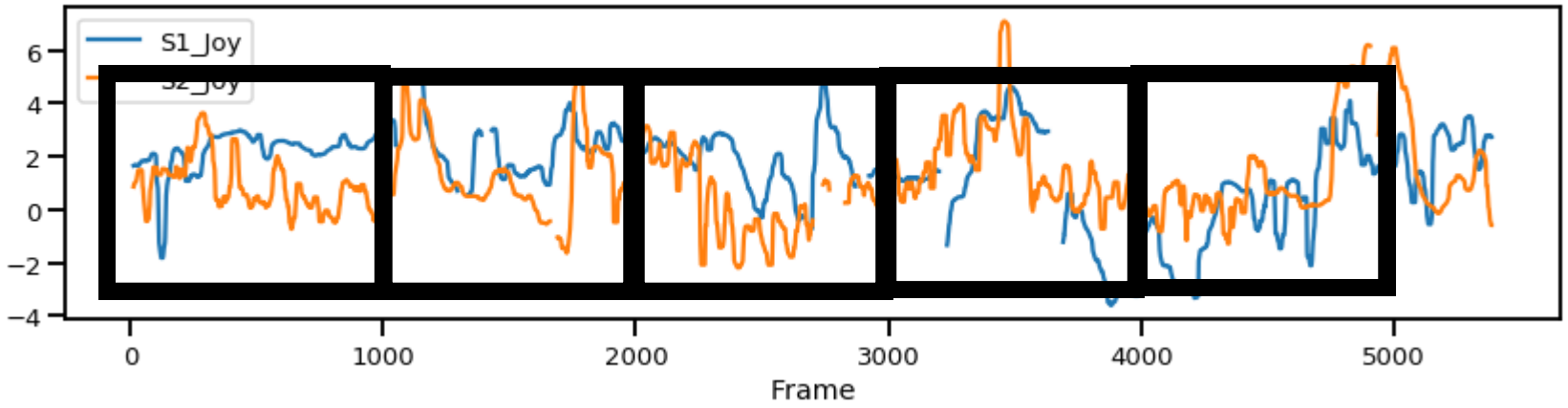
timeseries: how many time stamps in a sequence

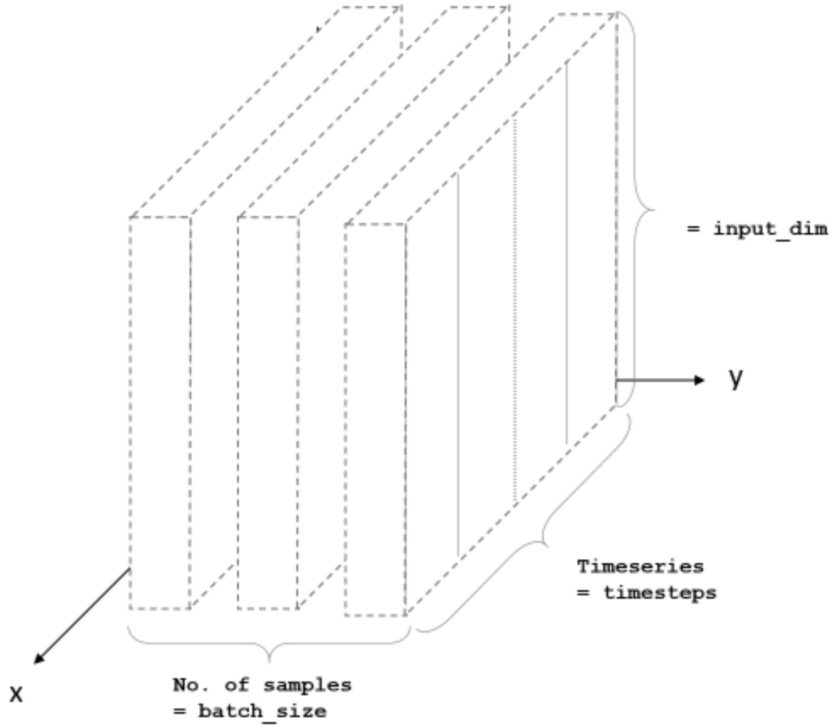
features: how many measurements in the time seris

even if you want to predict a single time series, you need many example

split the time series into chunks

LSTM: how to actually run it





batch size: N

timeseries: 1000

features: 2

```
model = Sequential()
model.add(LSTM(32, input_shape=(50, 2)))
model.add(Dense(2))
```

even if you want to predict a single time series, you need many example

split the time series into chunks

LSTM: how to actually run it

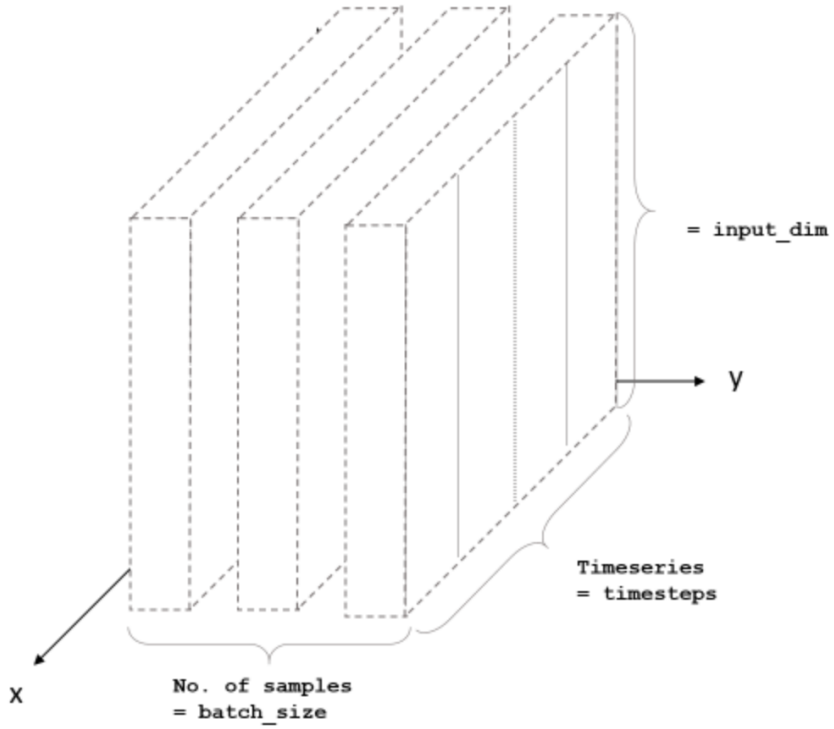
To be or not	to be? this	is the quest	ion. Whether	'tis nobler	in the mind

sequencies of 12 letters

batch size: N

timeseries: 12

features: 1



even if you want to predict a single time series, you need many example

split the time series into chunks

LSTM: how to actually run it

There is no homework on this cause I am at the end of the semester, but if you want to learn more I will upload an exercise over the weekend where you will train an RNN to generate physics paper titles!

http://davidsd.org/2010/09/the-arxiv-according-to-arxiv-vs-snarxiv/

The arXiv According to arXiv vs. snarXiv — *Sep 17, 2010*

After more than 3/4 of a million guesses, in over 50,000 games played in 67 countries, the results are clear: Science sounds like gobbledygook.

arXiv vs. snarXiv has been live for 6 months now, and it's time to take a look at the results. Here's how the game works. The user sees two titles: one is the title of an actual theoretical high energy physics paper on the arXiv, and the other is a completely fake title randomly generated by the snarXiv. The user guesses which one is real, finds out if they're right or wrong, and then starts over with a new pair of titles.


4
MLPNS
attention

Attention is all you need: transformer model

transformer generalized architecture elements

Encoder + Decoder architecture

Attention mechanism

Multithreaded attention

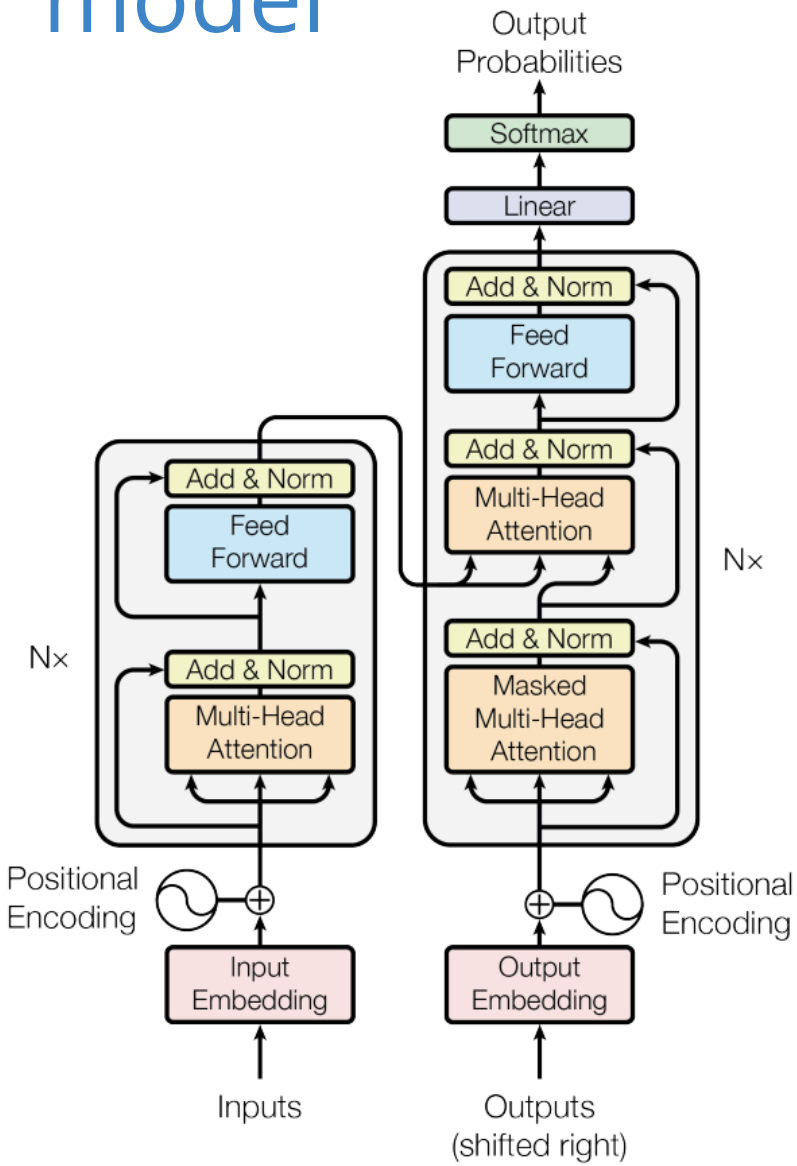


Figure 1: The Transformer - model architecture.

Neural Machine Translation by Jointly Learning to Align and Translate Bahdanbau 2015

attention mechanism:

a way to relate elements of the time series with each other

	v1	***v2***	***v3***	***v4***
k1	0.1	0.1	0.1	0.1
k2	0.9	0.3	0.1	0.1
k3	0.2	0.1	0.2	0.1
k4	0.6	0.9	0.1	0.

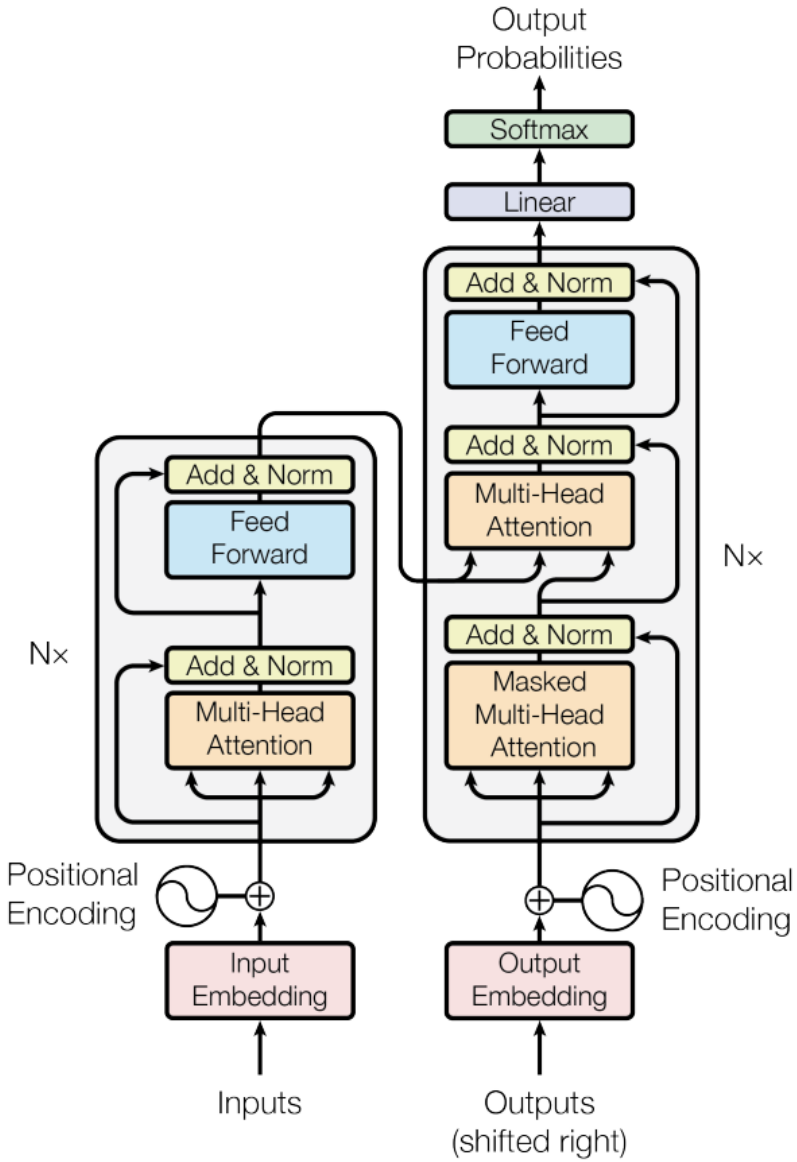


Figure 1: The Transformer - model architecture.

Neural Machine Translation by Jointly Learning to Align and Translate Bahdanbau 2015

attention mechanism:

a way to relate elements of the time series with each other

was full and happy

		v1	**v2**	**v3**	**v4**
The	**k1**	0.1	0.1	0.1	0.1
cat	**k2**	0.9	0.3	0.1	0.1
that	**k3**	0.2	0.1	0.2	0.1
ate	**k4**	0.6	0.9	0.1	0.

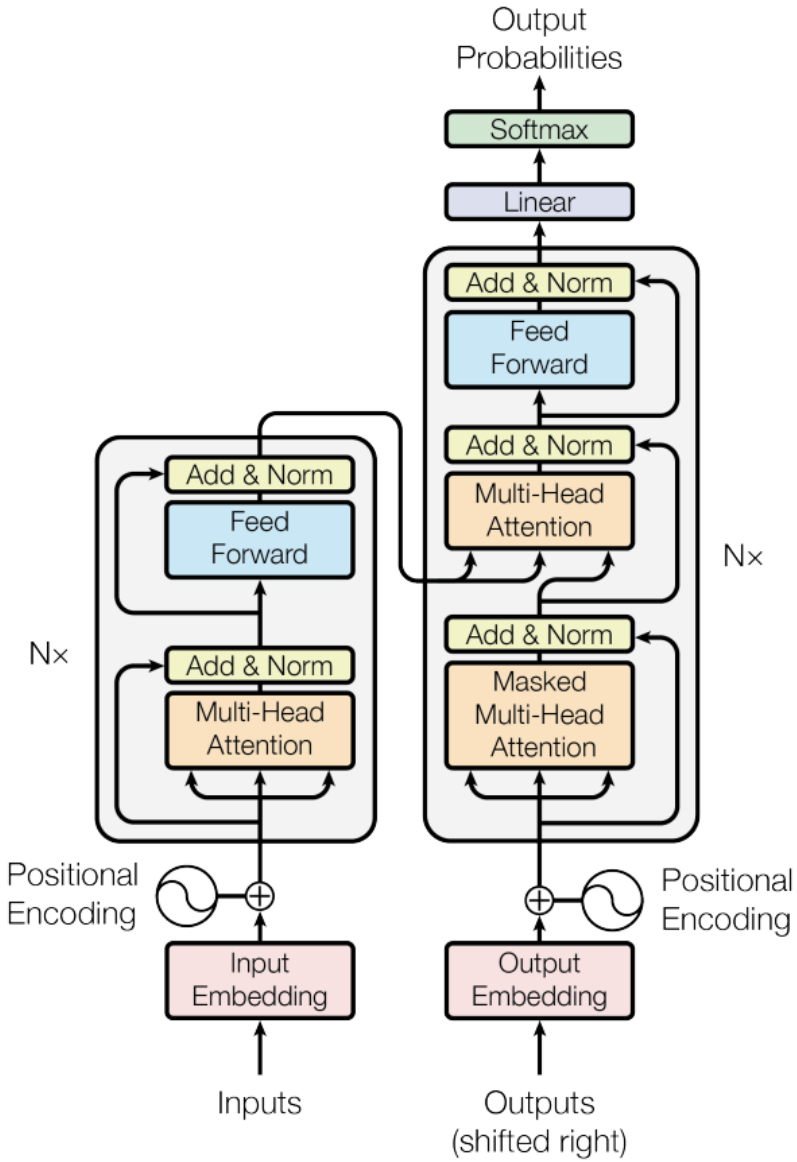
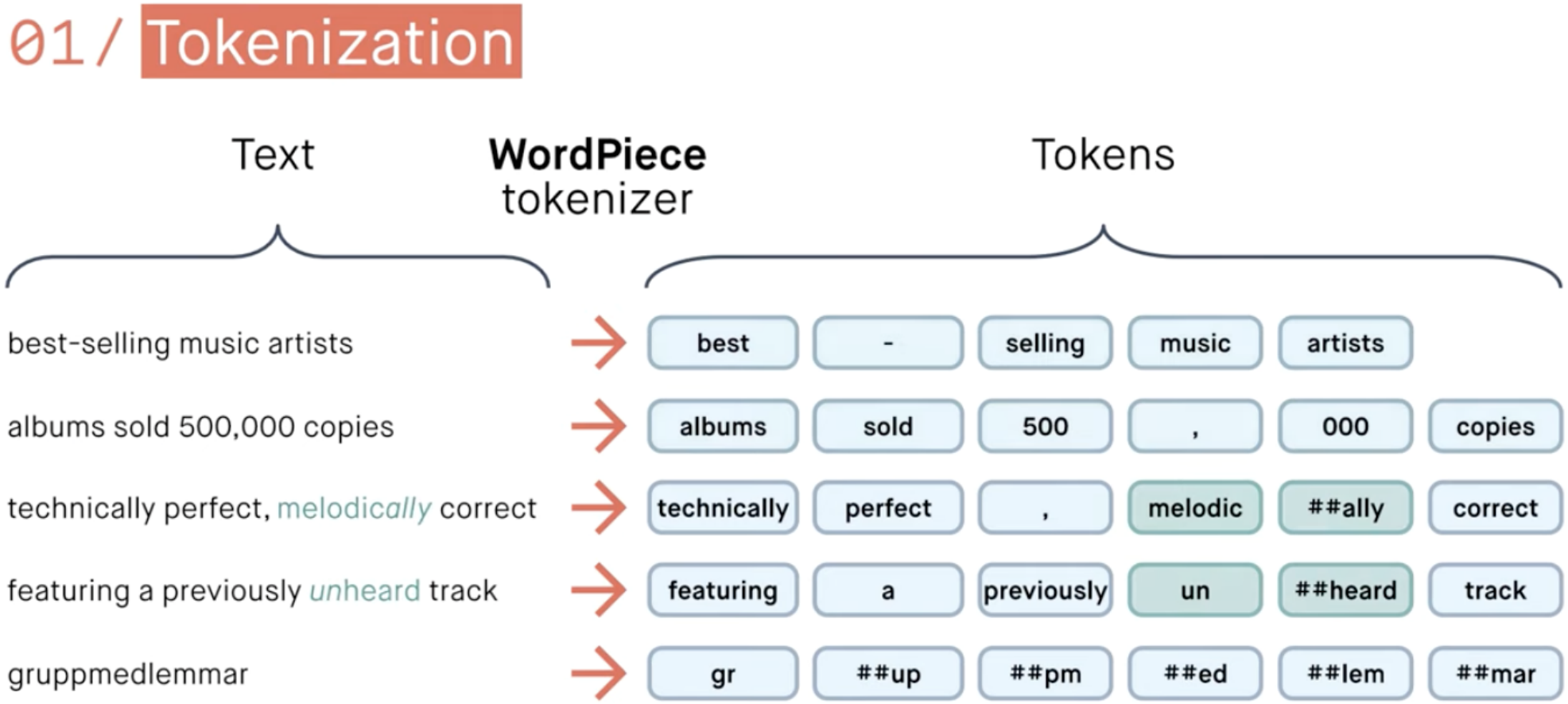


Figure 1: The Transformer - model architecture.

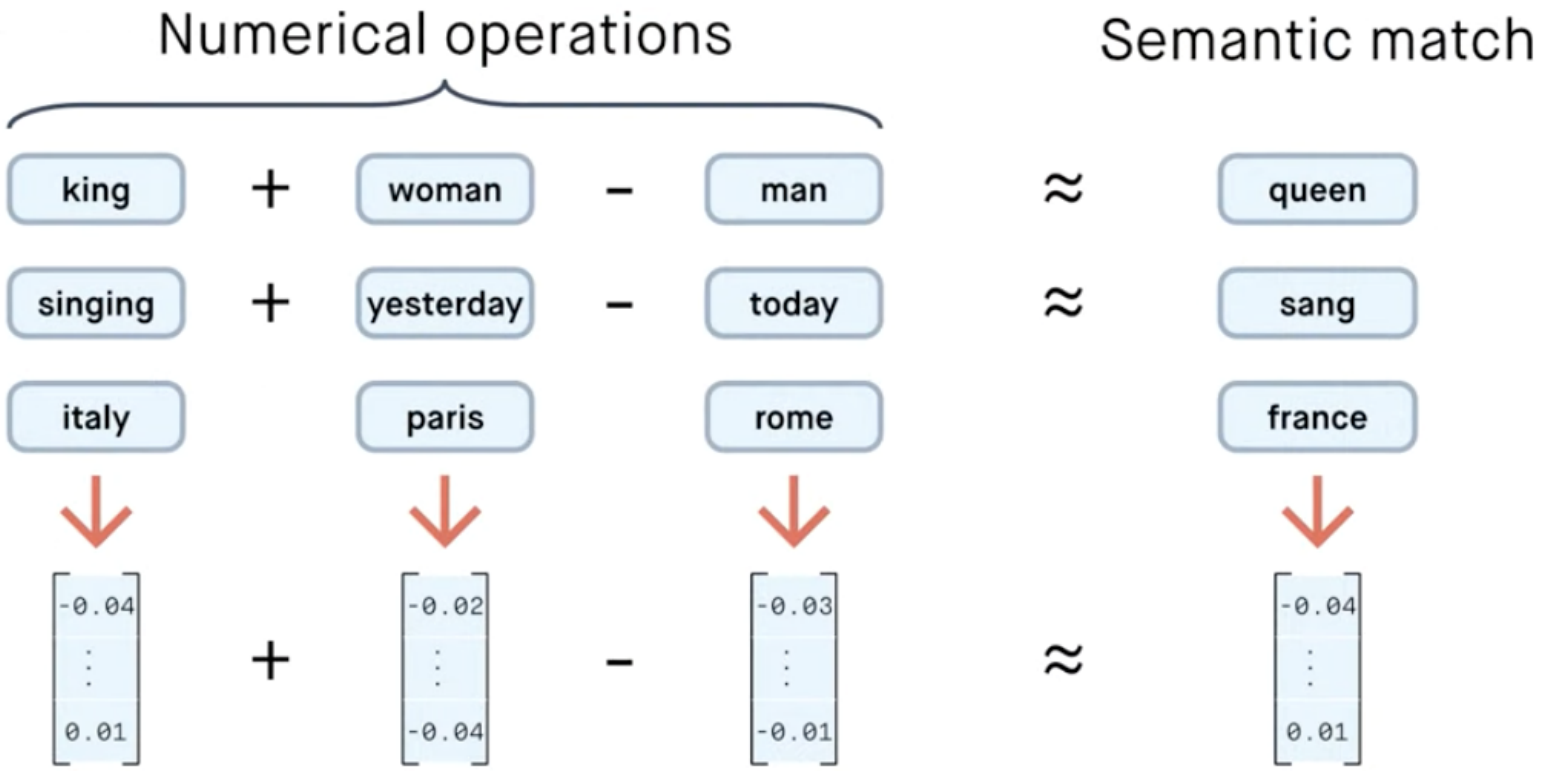
Word tockenization



https://www.youtube.com/watch?v=-9vVhYEXeyQ

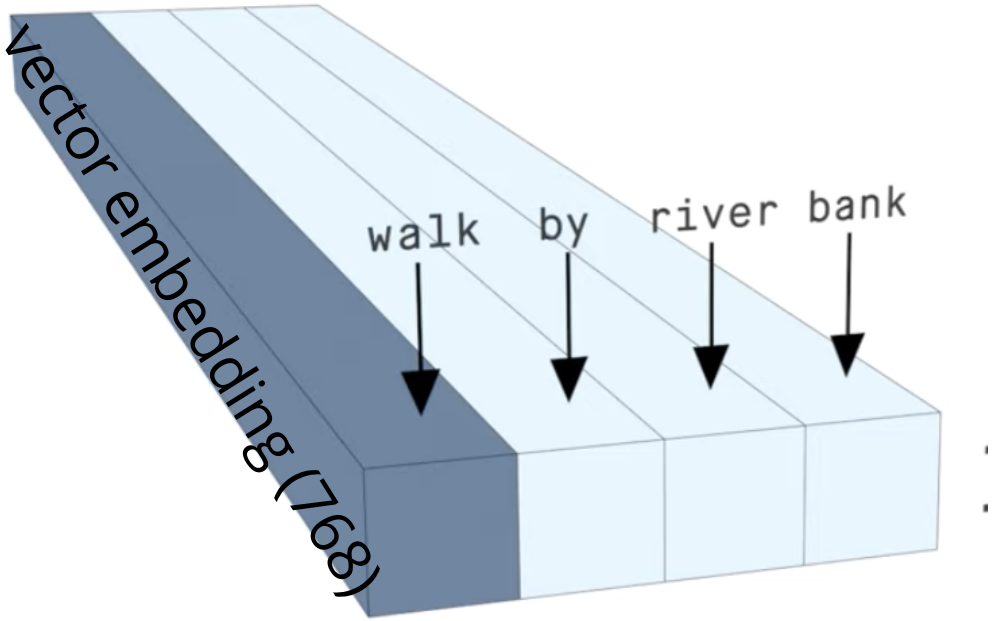
Word tockenization and embedding

02/ Embedding



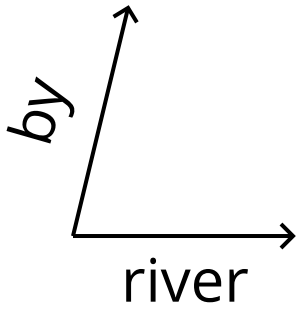
https://www.youtube.com/watch?v=-9vVhYEXeyQ

Word tockenization and embedding and contextualizing



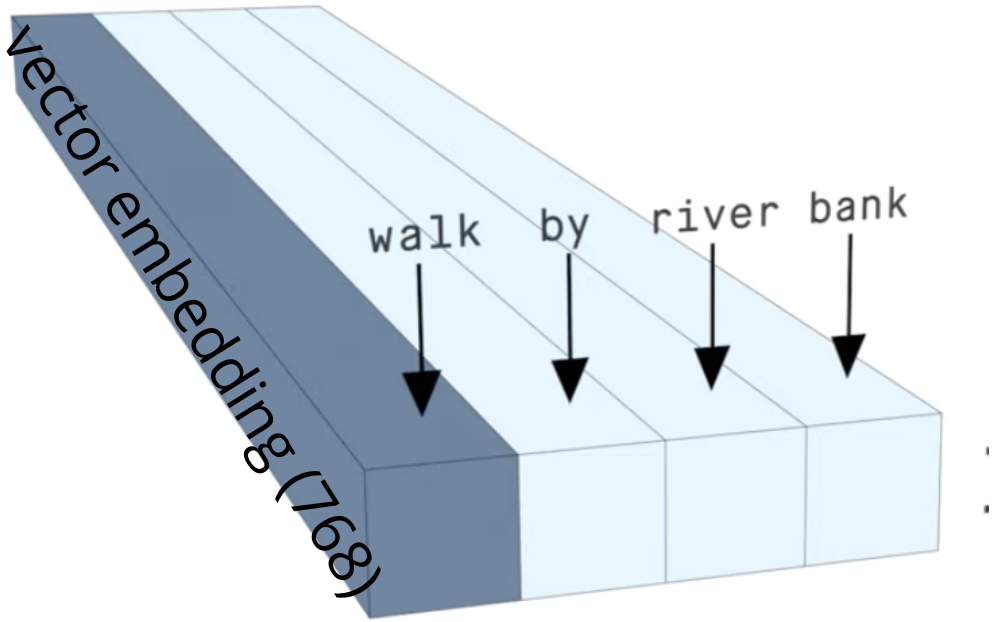
<by> ∘ <river> -> small

near orthogonal embedding, low similarity vectors,
no strong relation between tokens



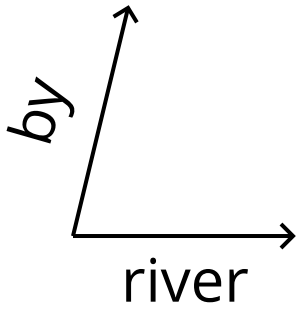
https://www.youtube.com/watch?v=-9vVhYEXeyQ

Word tockenization and embedding and contextualizing



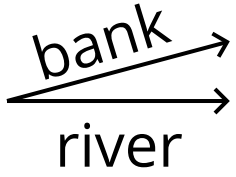
<by> ∘ <river> -> small

near orthogonal embedding, low similarity vectors,
no strong relation between tokens



<river> ∘ <bank> -> large

near parallel embedding, high similarity vectors,
strong relation between tokens



https://www.youtube.com/watch?v=-9vVhYEXeyQ

Neural Machine Translation by Jointly Learning to Align and Translate Bahdanbau 2015

attention mechanism:

a way to relate elements of the time series with each other

The cat that ate was full and happy

		v1	***v2***	***v3***	***v4***	***v5***	***v6***	***v7***	***v8***
The	***k1***	1	0.1	0.1	0.1	0.1	0.1	0.1	0.7
cat	***k2***	0.2	1	0.1	0.6	0.8	0.2	0.1	0.4
that	***k3***	0.1	0.1	1	0.2	0.1	0.2	0.1	0.1
ate	***k4***	0.6	0.	0.	1	***0.5***	***0.9***	0.1	0.5
was									
full									

fully autoregressive model

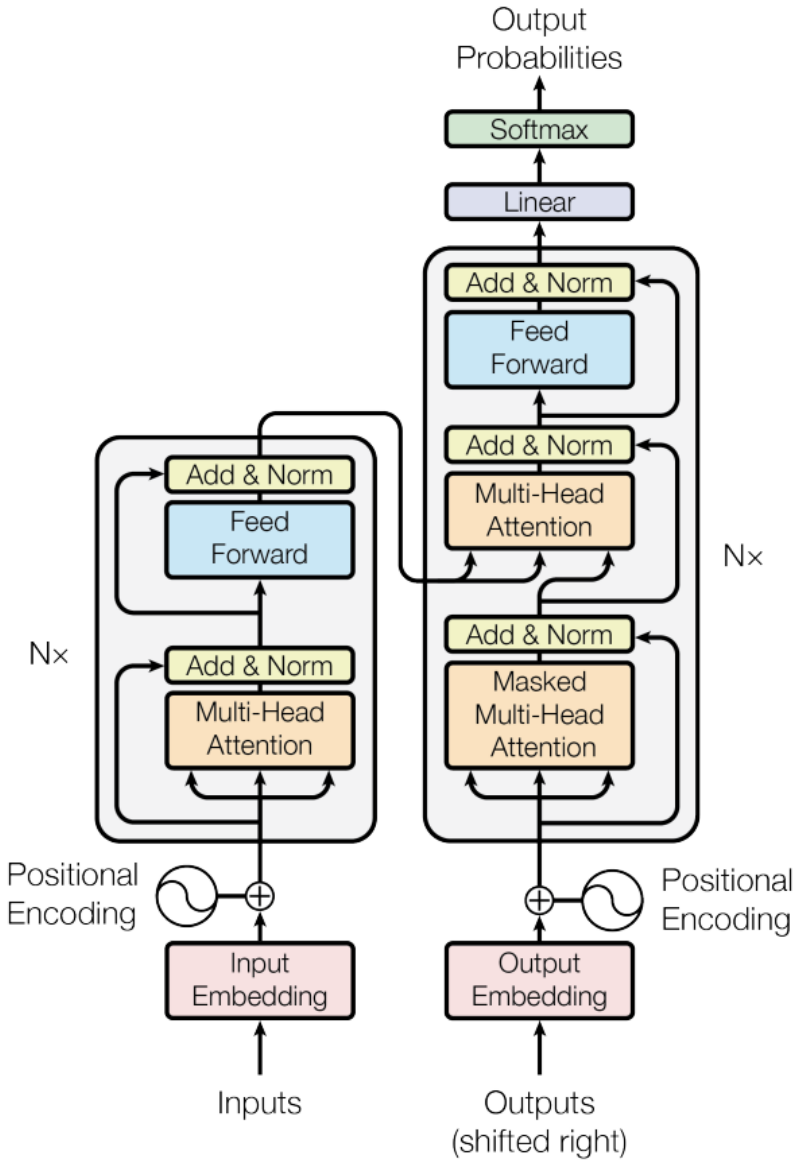


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + Decoder architecture

attention: **project embedding into *Key - Value - Query***

lower dimensional representations

The key/value/query concept is analogous to retrieval systems.

1238 913 12

		v1	***v2***	***v3***
39 *5*	***k1***	0.1	0.1	0.1
903	***k2***	0.9	0.3	0.1
	k3	0.2	0.1	0.2

w

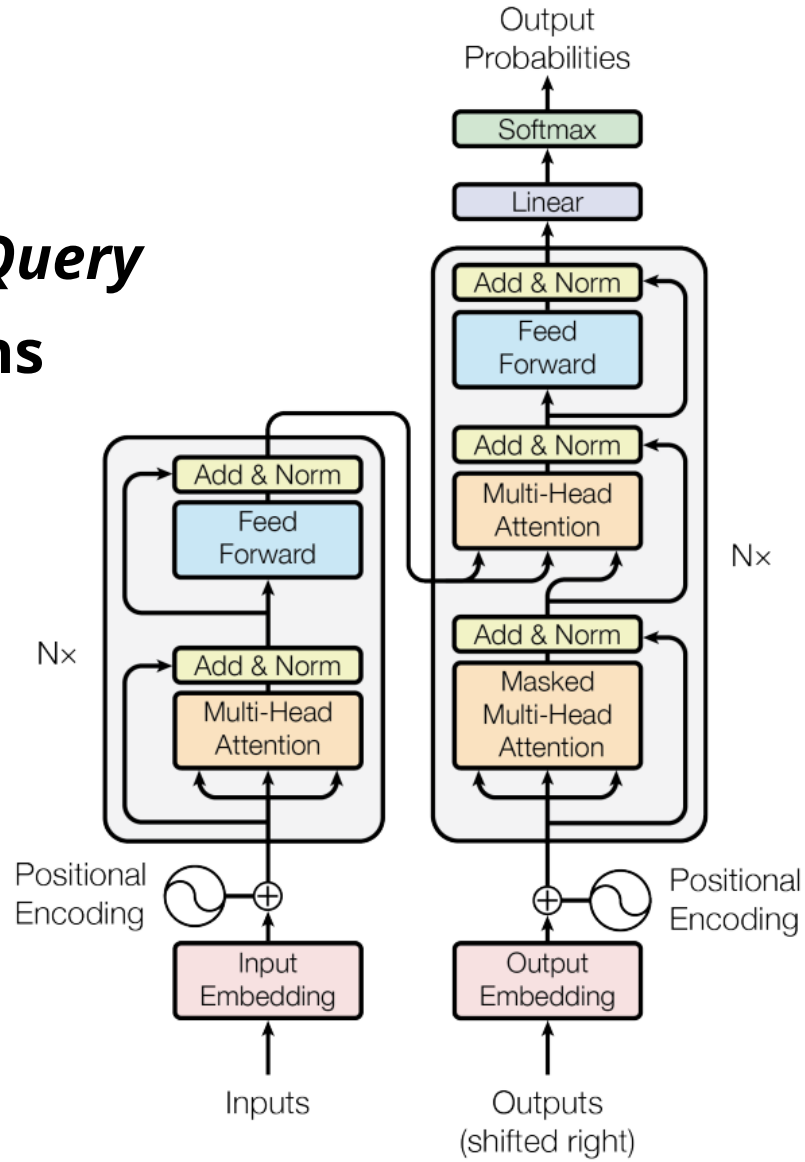


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + Decoder architecture

attention: **project embedding into *Key - Value - Query* lower dimensional representations**

- **Query**: The query is a feature vector that describes what we are looking for in the sequence, i.e. what would we maybe want to pay attention to.
- **Keys**: For each input element, we have a key which is again a feature vector. This feature vector roughly describes what the element is "offering", or when it might be important. The keys should be designed such that we can identify the elements we want to pay attention to based on the query.
- **Values**: For each input element, we also have a value vector. This feature vector is the one we want to average over.
- **Score function**: To rate which elements we want to pay attention to, we need to specify a score function f_{attn}. The score function takes the query and a key as input, and output the score/attention weight of the query-key pair. It is usually implemented by simple similarity metrics like a dot product, or a small MLP.

The weights of the average are calculated by a softmax over all score function outputs. Hence, we assign those value vectors a higher weight whose corresponding key is most similar to the query. If we try to describe it with pseudo-math, we can write:

$$\alpha_i = \frac{\exp\left(f_{attn}\left(\text{key}_i, \text{query}\right)\right)}{\sum_j \exp\left(f_{attn}\left(\text{key}_j, \text{query}\right)\right)}, \quad \text{out} = \sum_i \alpha_i \cdot \text{value}_i$$

https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html

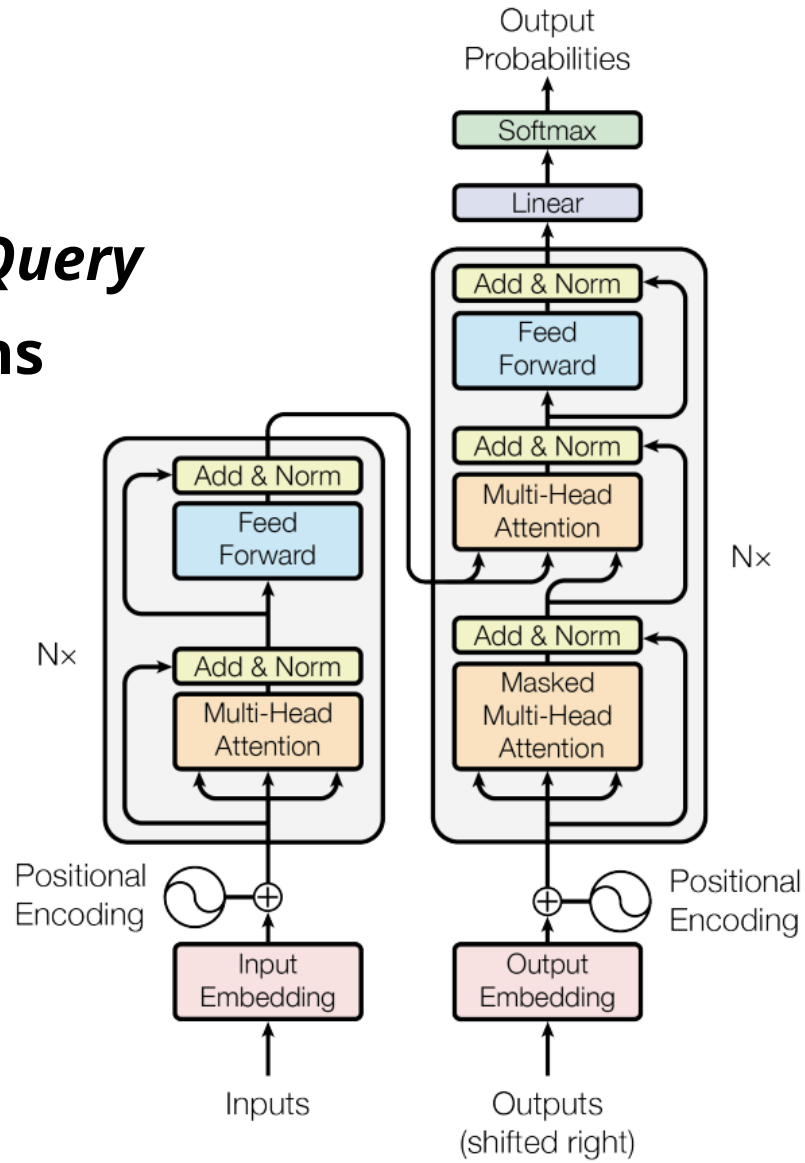


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + Decoder architecture

attention: Key - Value - Query

The key/value/query concept is analogous to retrieval systems.

1238 913 12

		v1	*v2*	*v3*
39 *5*	***k1***	0.1	0.1	0.1
903	***k2***	0.9	0.3	0.1
	k3	0.2	0.	0.2

key: input

query: output

value... input as well

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

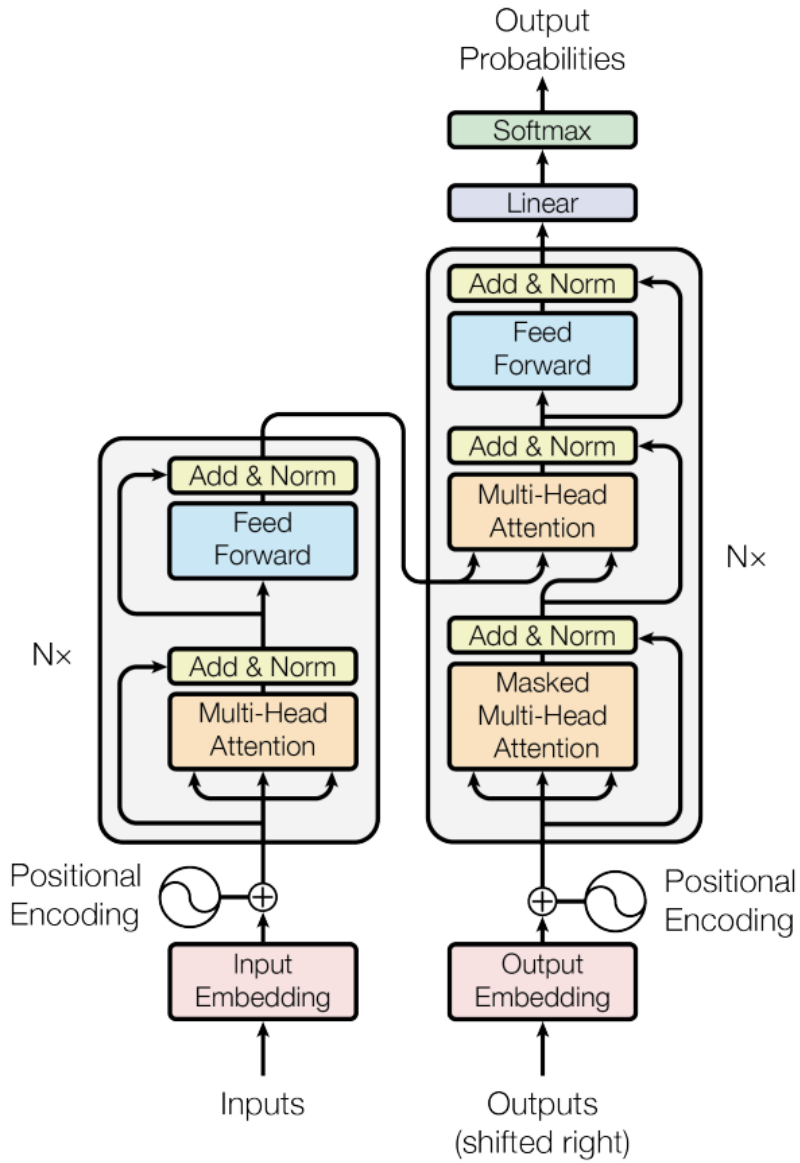
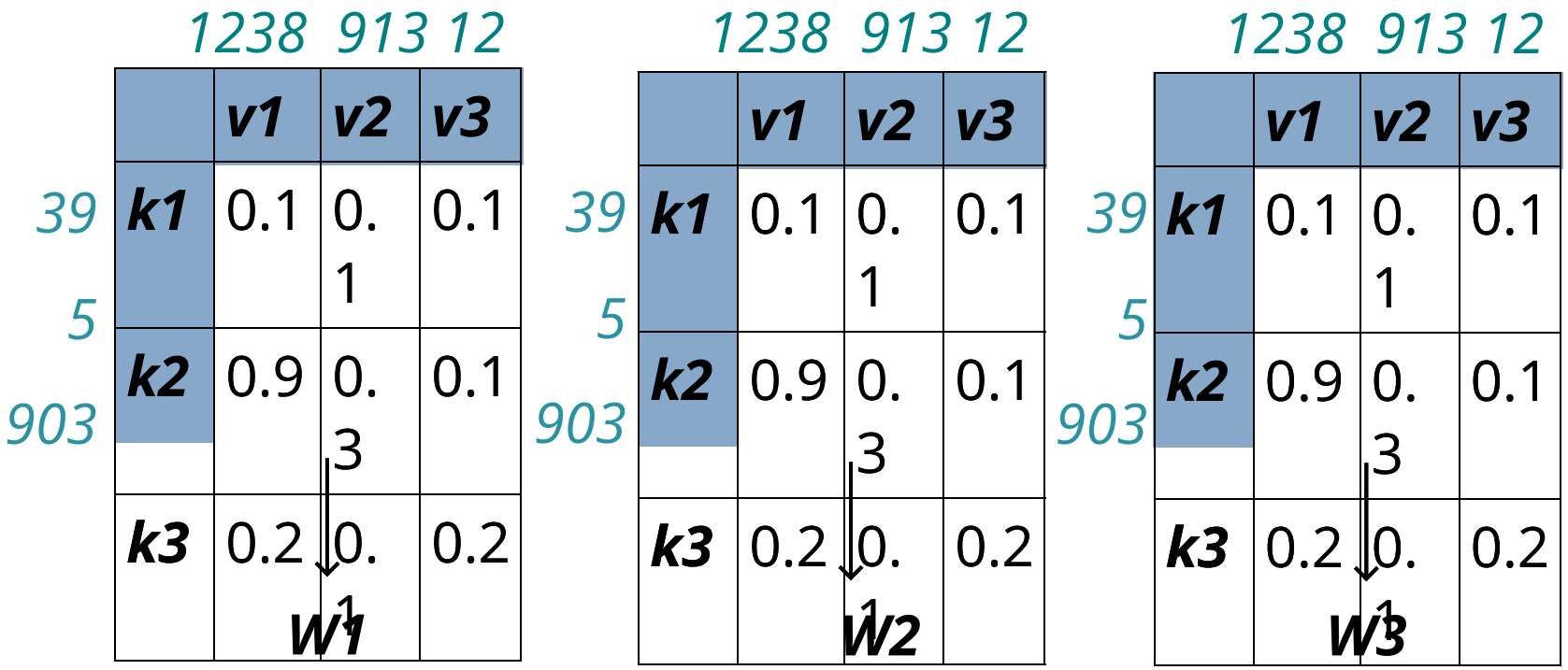


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + Decoder architecture

***Multi-headed* attention:**
different elements of the sentence relate to input elements in multiple ways



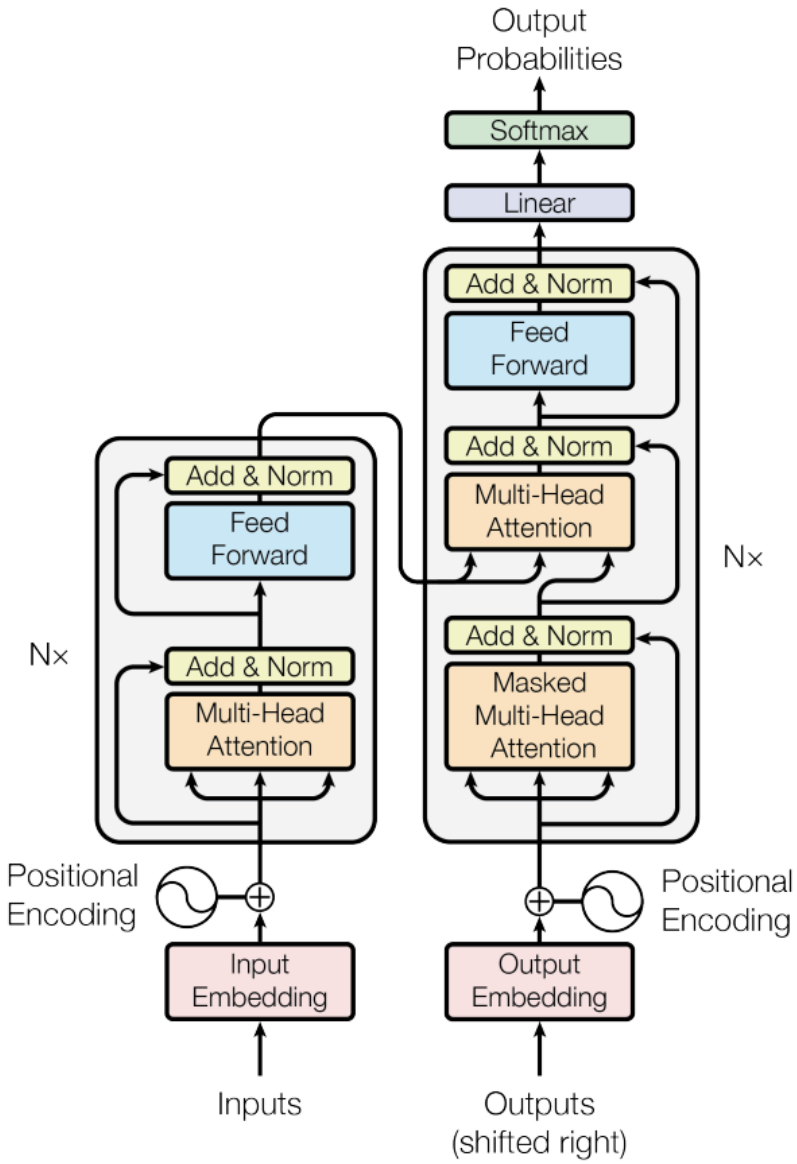


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + Decoder architecture

***Multi-headed Self* attention:**

The key/value/query concept is analogous to retrieval systems.

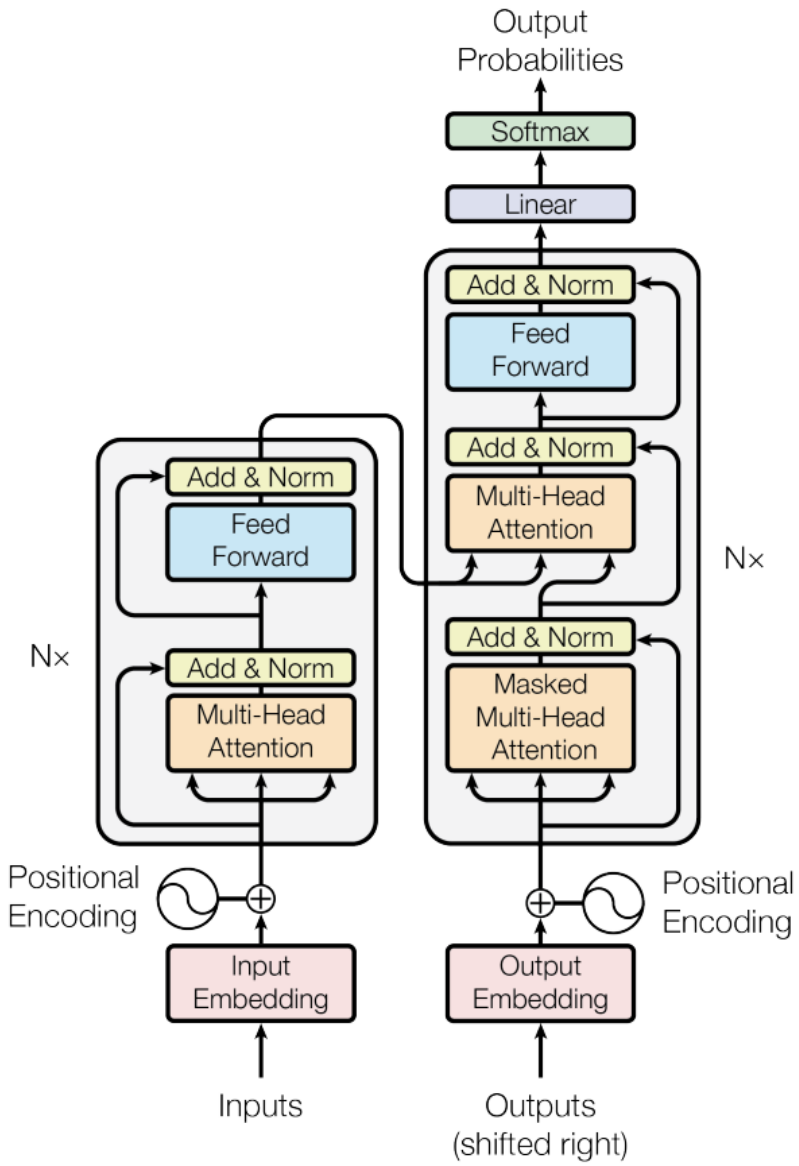


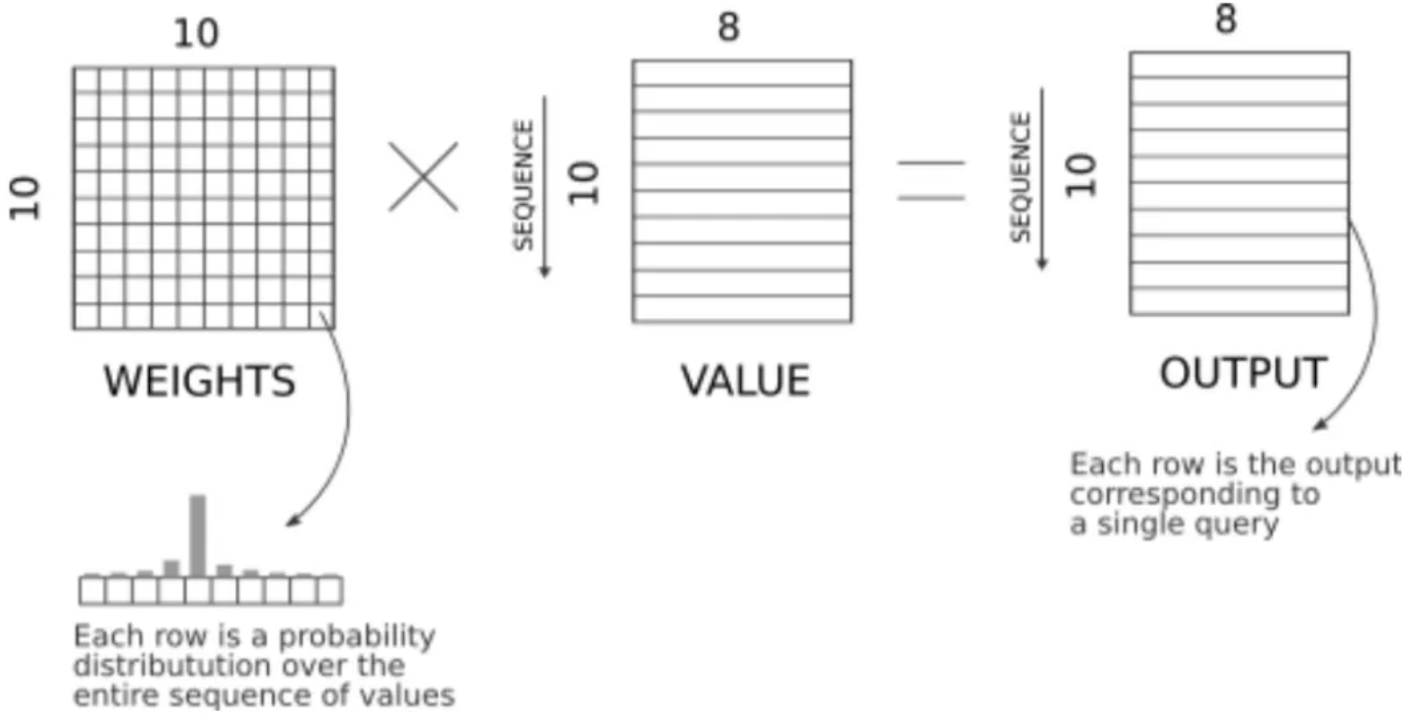
Figure 1: The Transformer - model architecture.

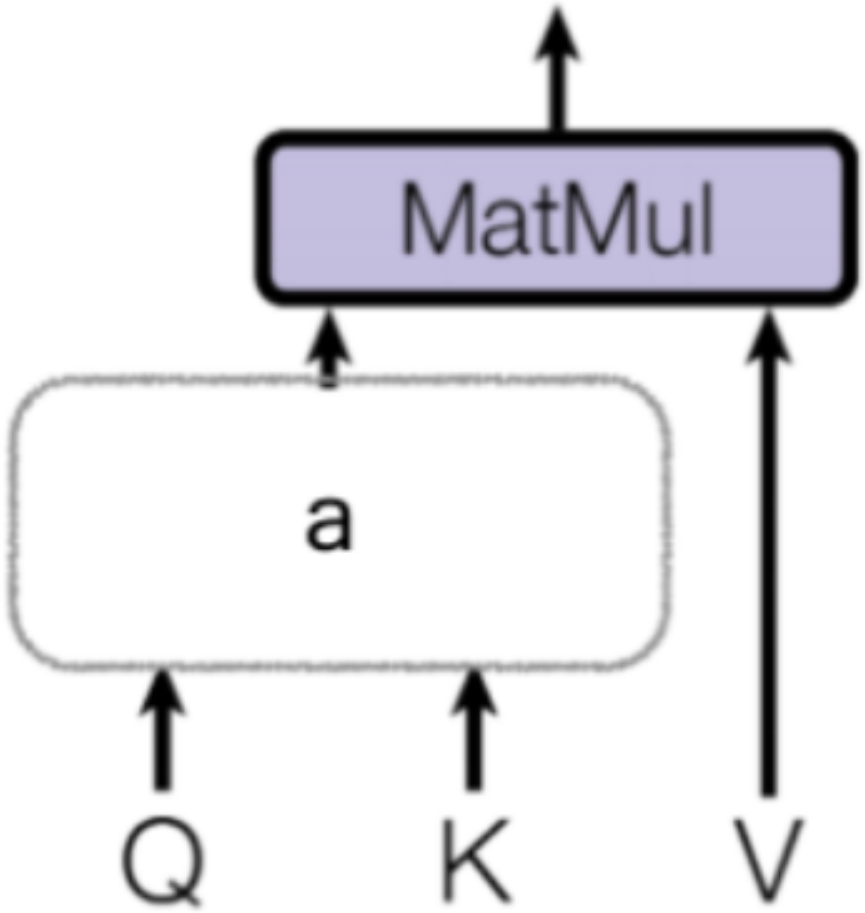
Attention is all you need (2017)

Encoder + Decoder architecture

***Multi-headed Self* attention:**

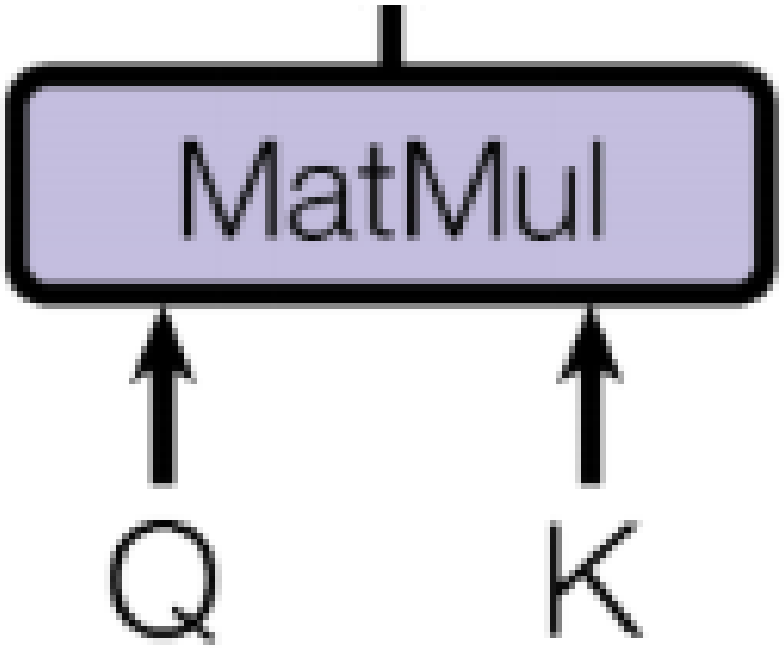
The key/value/query concept is analogous to retrieval systems.

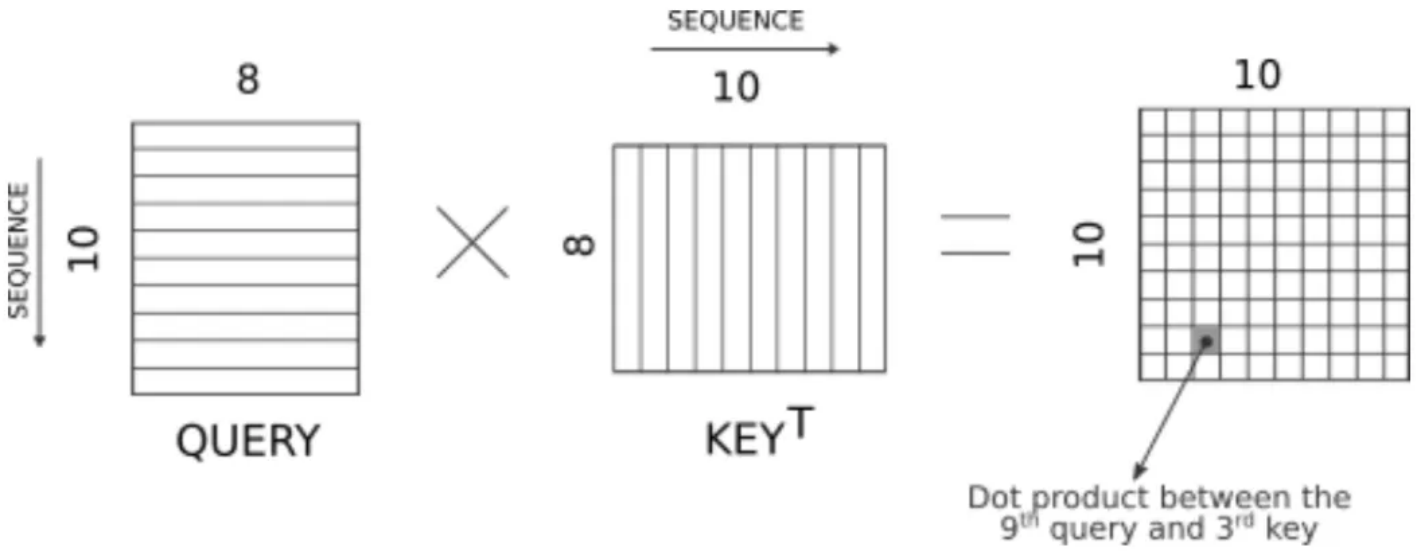




Attention is all you need (2017)

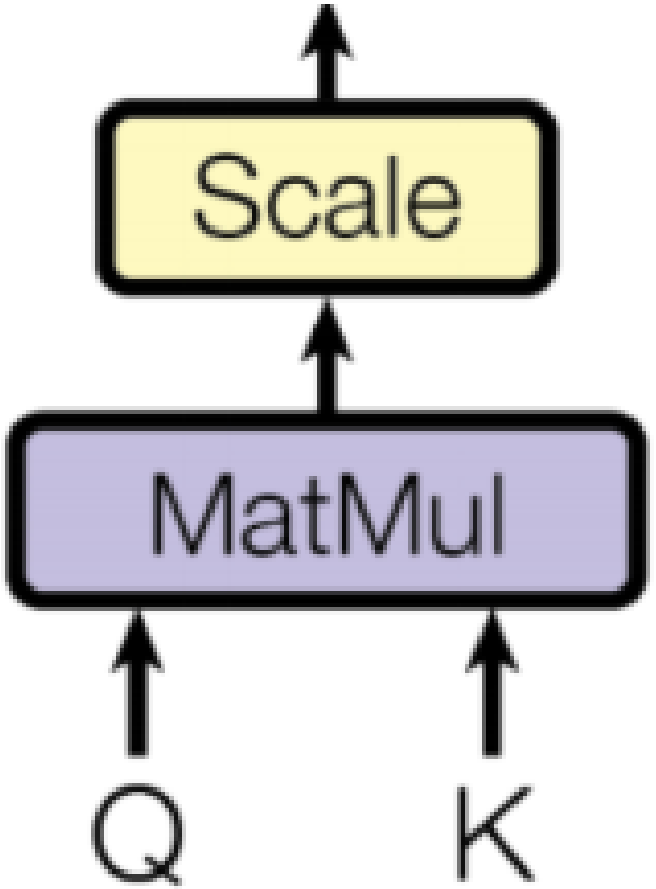
Dot Product Attention





Attention is all you need (2017)

Scaled Dot Product Attention



the dot-product can produce very large magnitudes with very large vector dimensions (d) which will result in very small gradients when passed into the softmax function, we can **scale** the values prior (scale = 1 / √ d).

$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right)$$

Attention is all you need (2017)

Encoder + Decoder architecture

attention: **project embedding into *Key - Value - Query***

lower dimensional representations

Encoder Attention

- Q = the current position-word vector in the input sequence
- K = all the position-word vectors in the input sequence
- V = all the position-word vectors in the input sequence

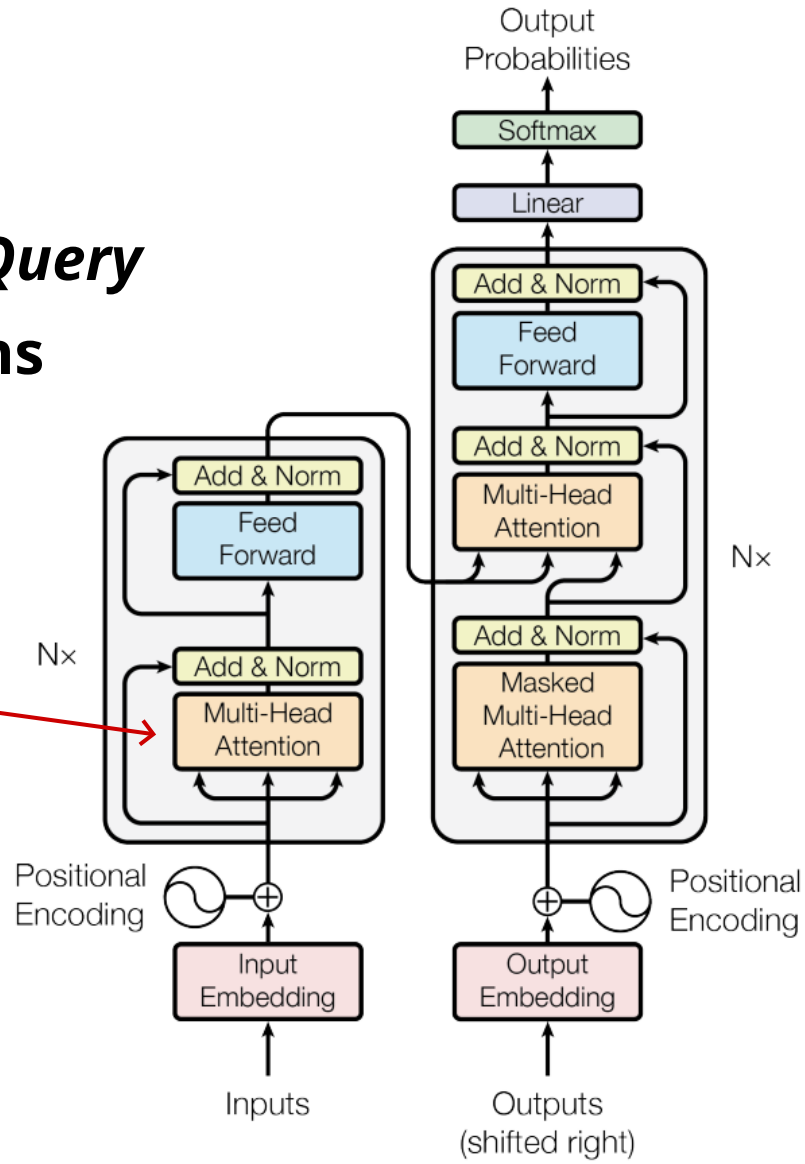


Figure 1: The Transformer - model architecture.

https://medium.com/@b.terryjack/deep-learning-the-transformer-9ae5e9c5a190

Attention is all you need (2017)

Encoder + Decoder architecture

attention: **project embedding into *Key - Value - Query***

lower dimensional representations

Decoder Attention

- Q = the current position-word vector in the output sequence
- K = all the position-word vectors in the output sequence
- V = all the position-word vectors in the output sequence

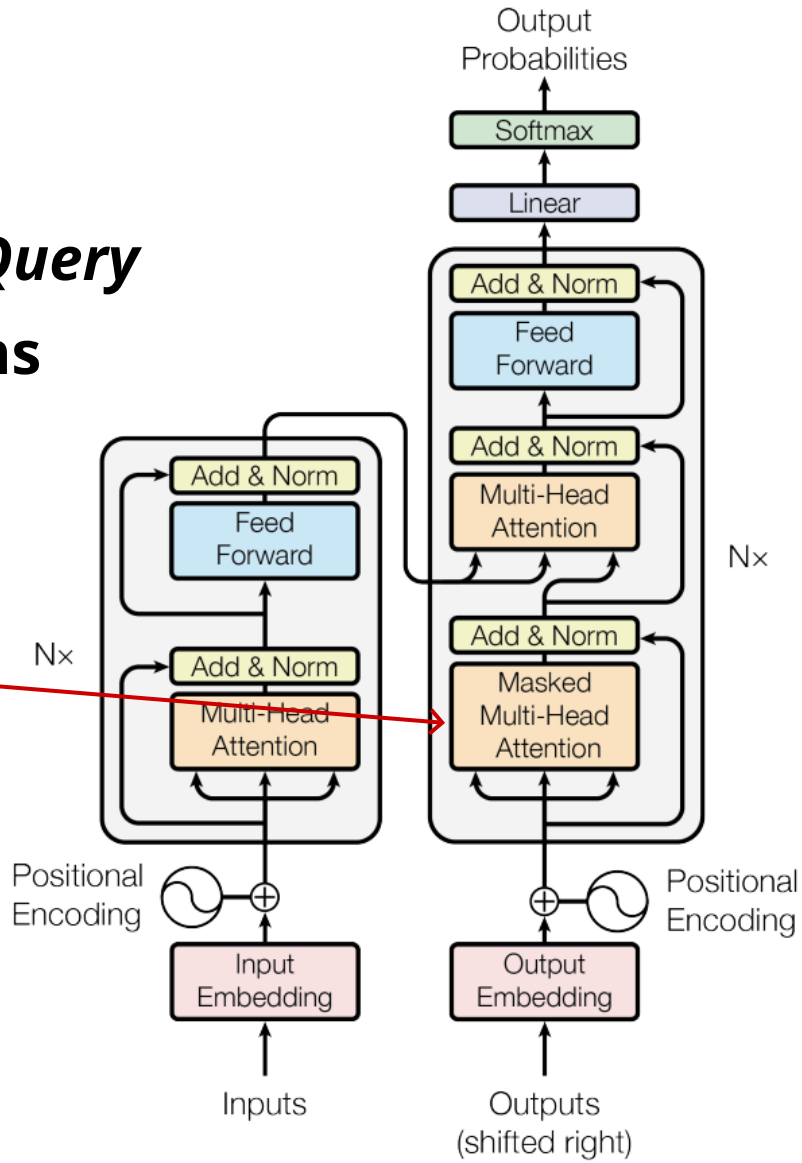


Figure 1: The Transformer - model architecture.

https://medium.com/@b.terryjack/deep-learning-the-transformer-9ae5e9c5a190

Attention is all you need (2017)

Encoder + Decoder architecture

attention: **project embedding into *Key - Value - Query***
lower dimensional representations

Encoder-Decoder Attention

- Q = the output of the decoder's masked attention
- K = all the encoder's hidden state vectors
- V = all the encoder's hidden state vectors

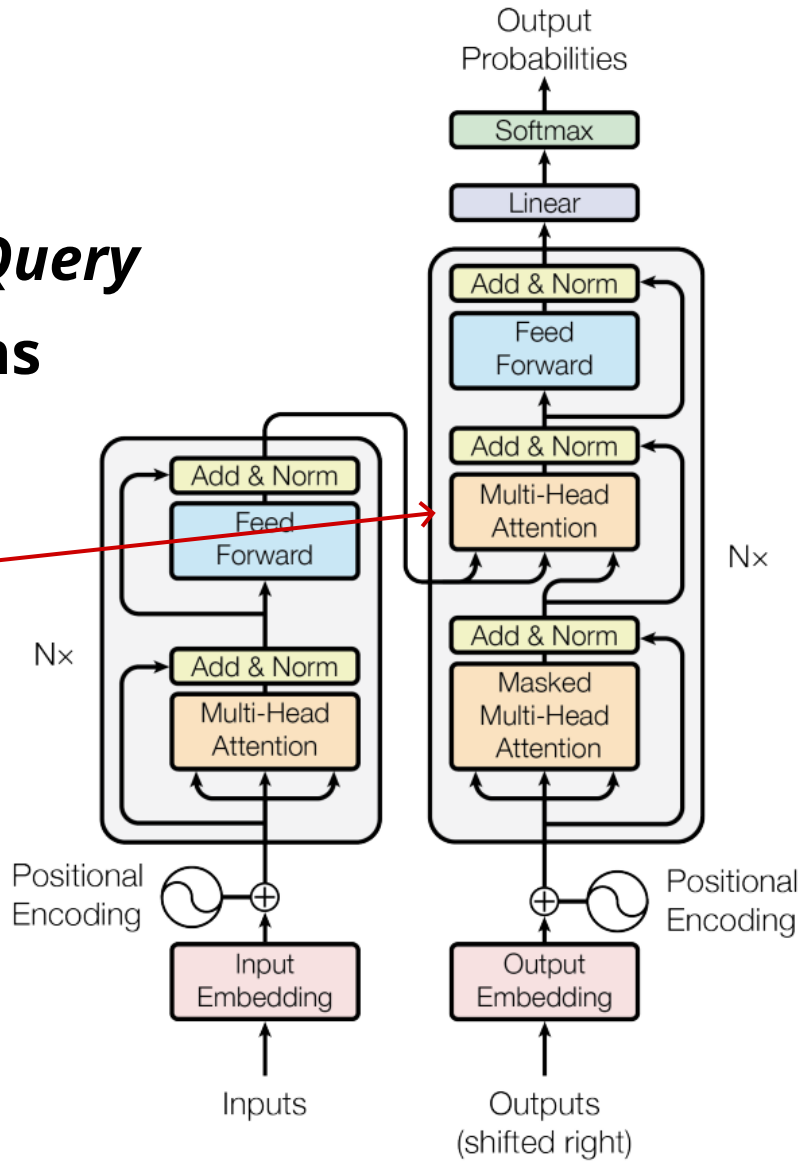


Figure 1: The Transformer - model architecture.

https://medium.com/@b.terryjack/deep-learning-the-transformer-9ae5e9c5a190

5

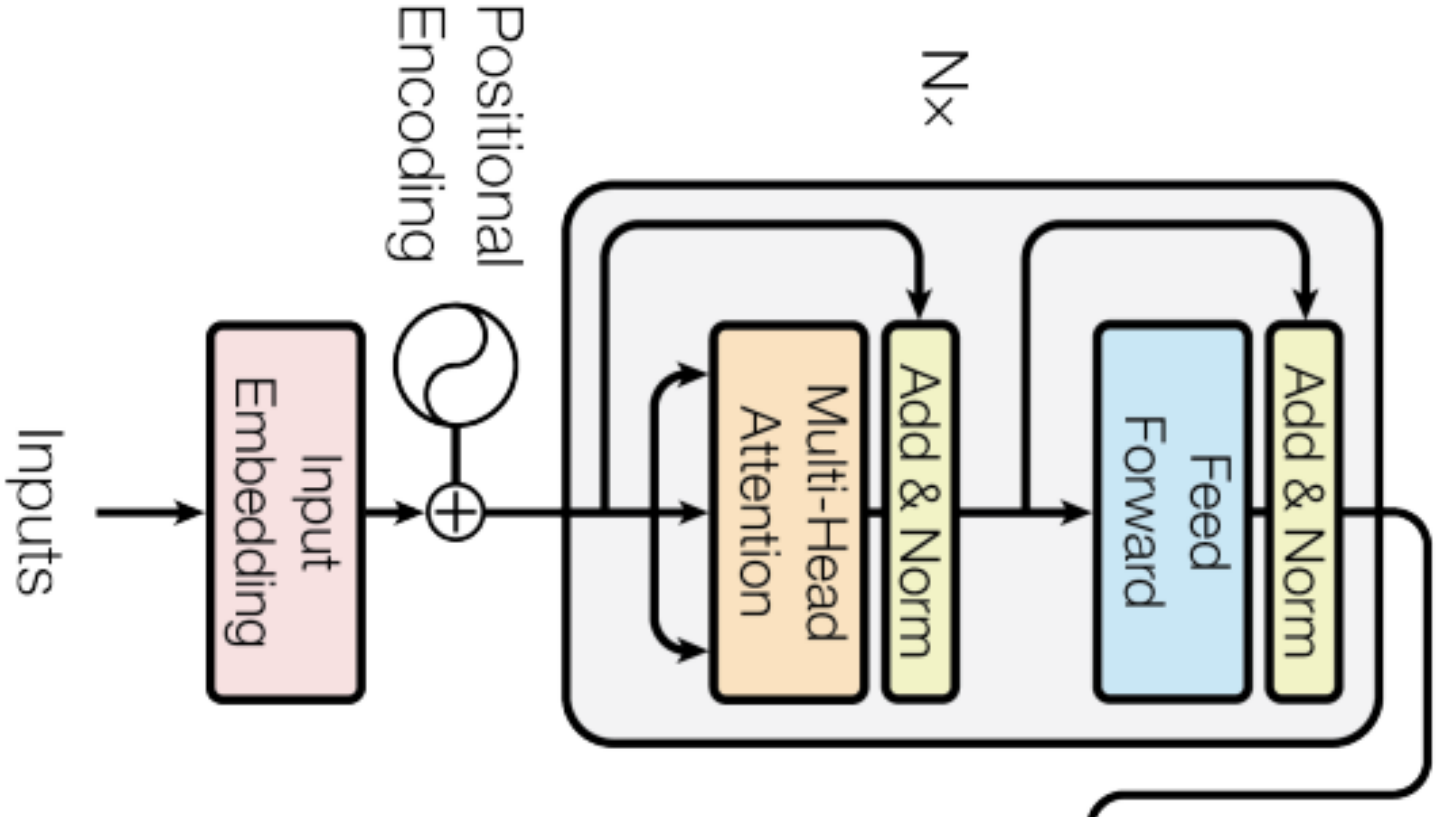
MLPNS

transformer model

Attention is all you need

Encoder + Decoder architecture

Encodes the past



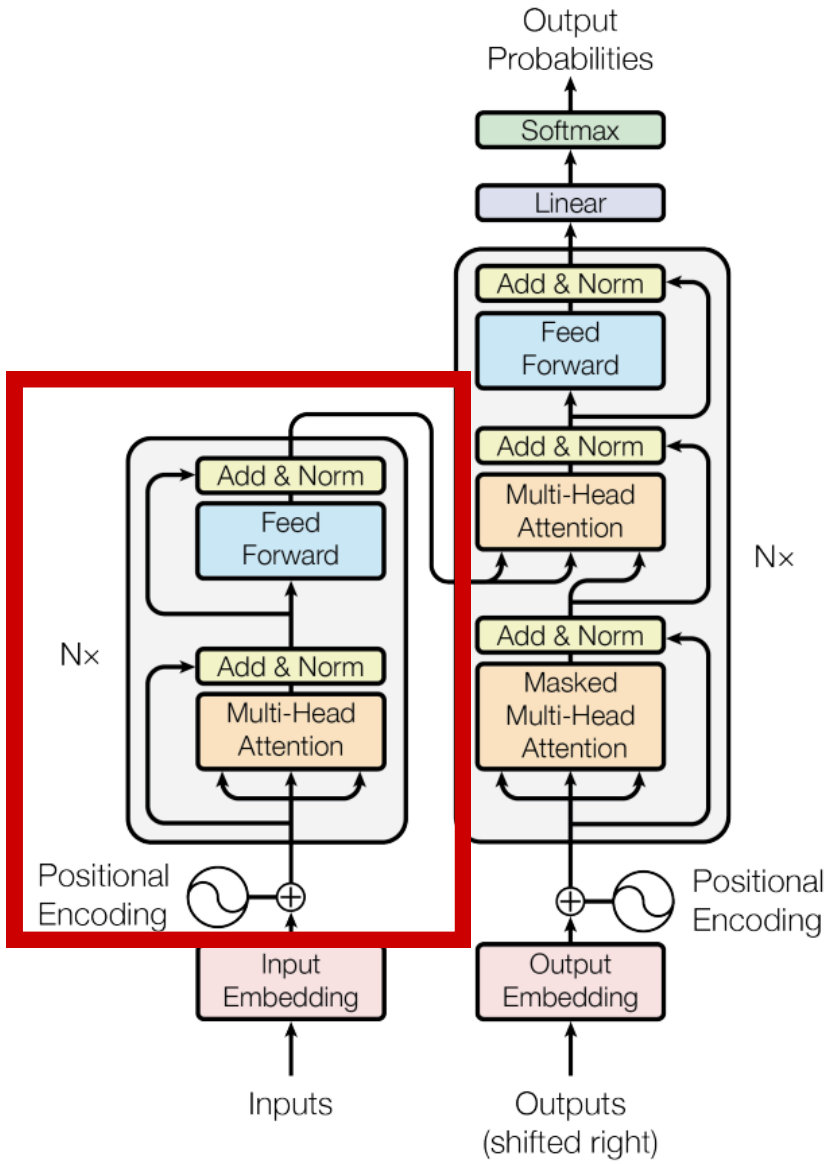


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + **Decoder** architecture

decodes the past and predicts the future

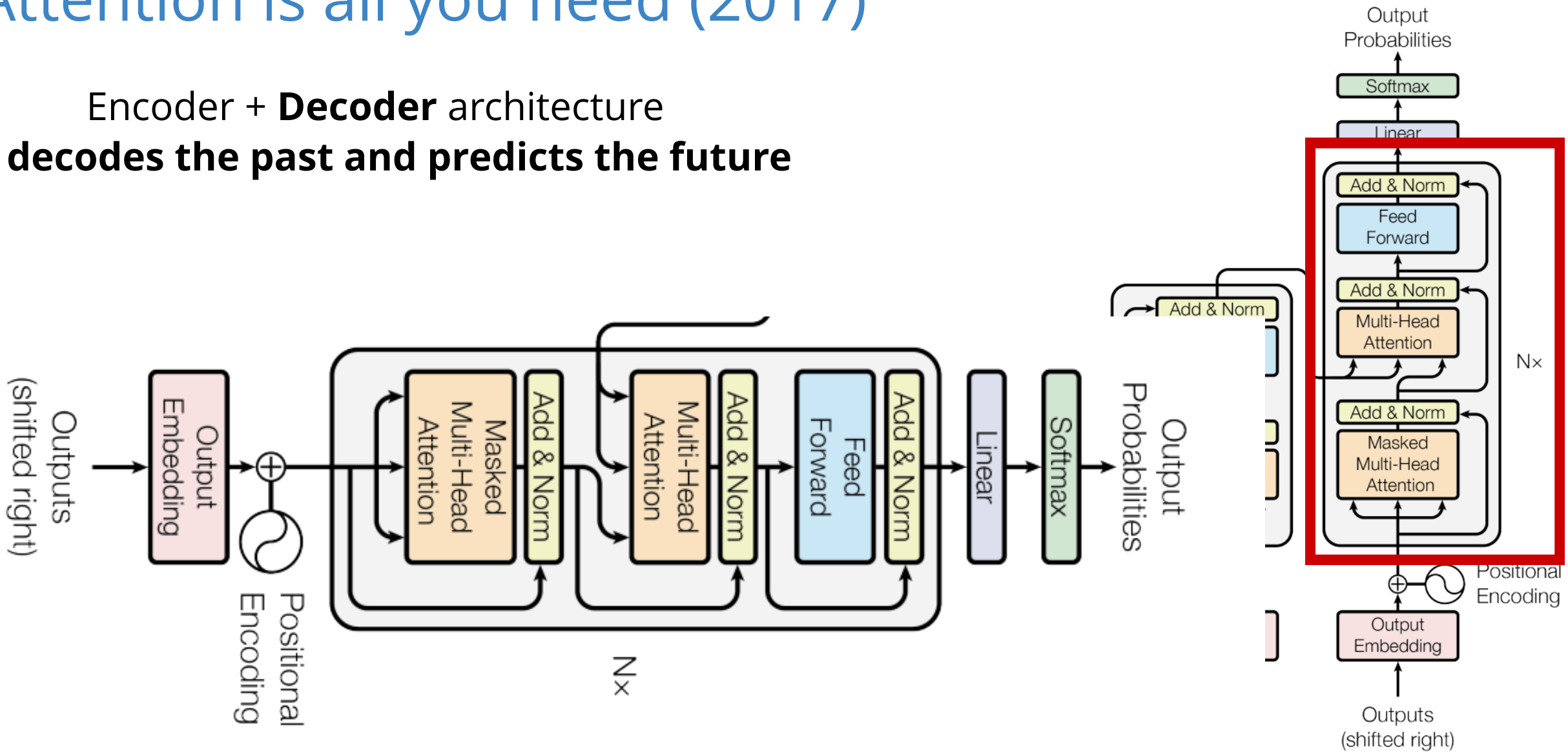


Figure 1: The Transformer - model architecture.

MHA acting on encoder (1)

Attention is all you need (2017)

Encoder + **Decoder** architecture

decodes the past and predicts the future

a stack of N = 6 identical layers each with
(1) a multi-head self-attention mechanism act on previous decoder output,
(2) a multi-head self-attention mechanism act on encoder output,
(3) a positionwise fully connected feed-forward NN

The cat that ate was full and happy

		v1	***v2***	***v3***	***v4***	***v5***	***v6***	***v7***	***v8***
The	***k1***	1	0.1	0.1	0.1	0.1	0.1	0.1	0.7
cat	***k2***	0.2	1	0.1	0.6	0.8	0.2	0.1	0.4
that	***k3***	0.1	0.1	1	0.2	0.1	0.2	0.1	0.1
ate	***k4***	0.6	0.	0.	1	***0.5***	***0.9***	0.1	0.5
was									
full									

MHA acting on encoder (1)

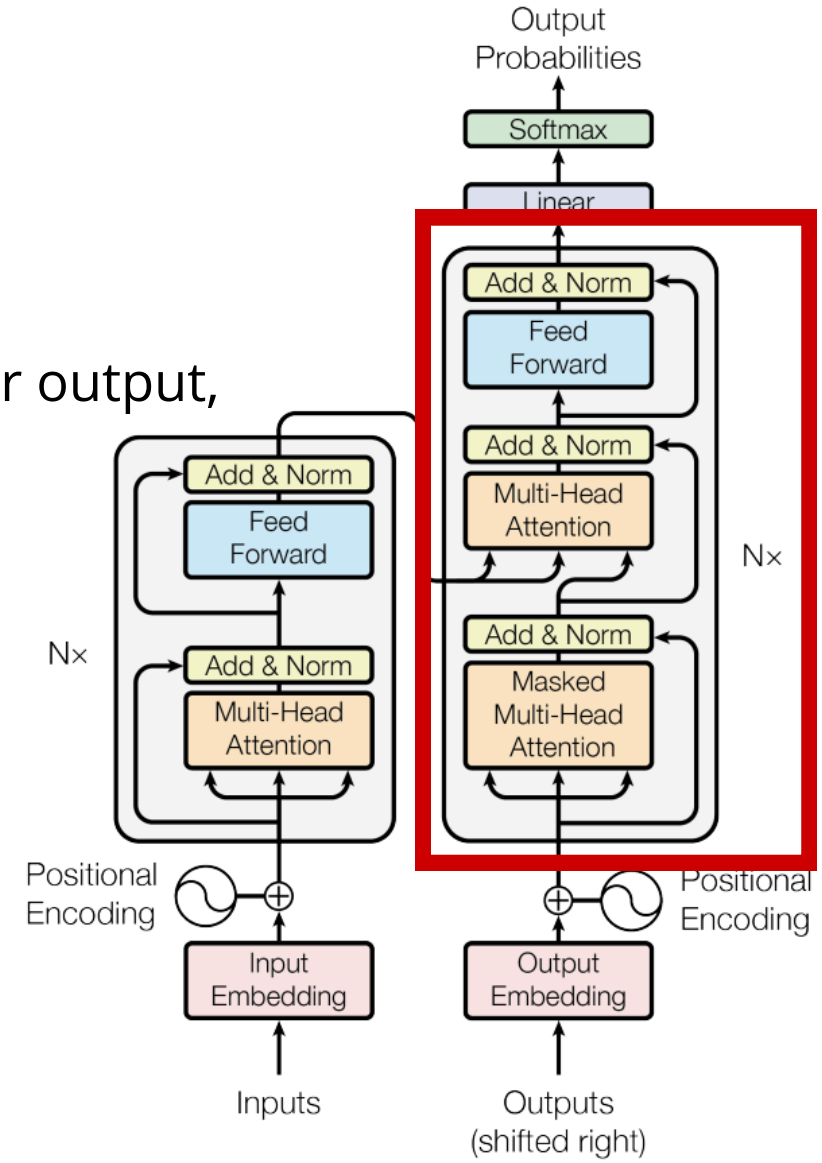


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + **Decoder** architecture

decodes the past and predicts the future

a stack of N = 6 identical layers each with
(1) a multi-head self-attention mechanism act on previous decoder output,
(2) a multi-head self-attention mechanism act on encoder output,
(3) a positionwise fully connected feed-forward NN

The cat that ate was full and happy

		v1	***v2***	***v3***	***v4***	***v5***	***v6***	***v7***	***v8***
The	***k1***	1	0.1	0.1	0.1	0.1	0.1	0.1	0.7
cat	***k2***	0.2	1	0.1	0.6	0.8	0.2	0.1	0.4
that	***k3***	0.1	0.1	1	0.2	0.1	0.2	0.1	0.1
ate	***k4***	0.6	0.	0.	1	***0.5***	***0.9***	0.1	0.5
was									
full									

MHA acting on decoder (2) masking dependence on the future

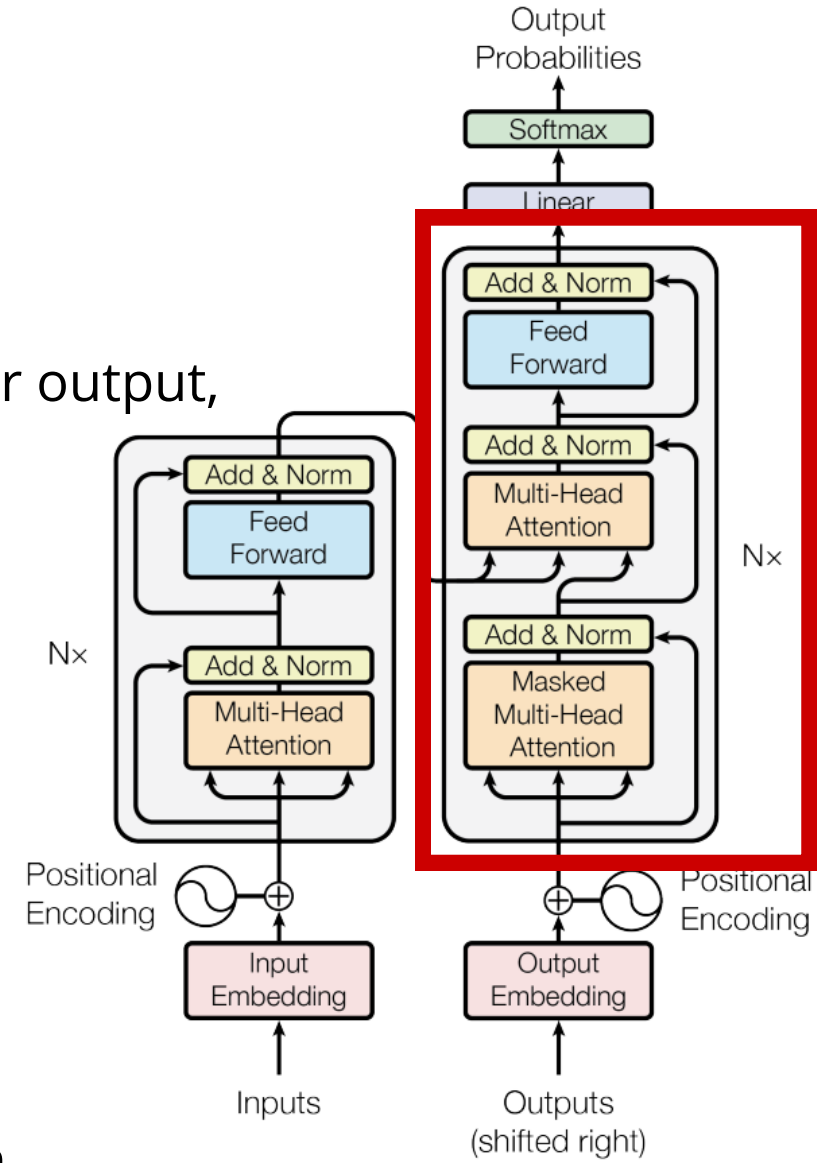
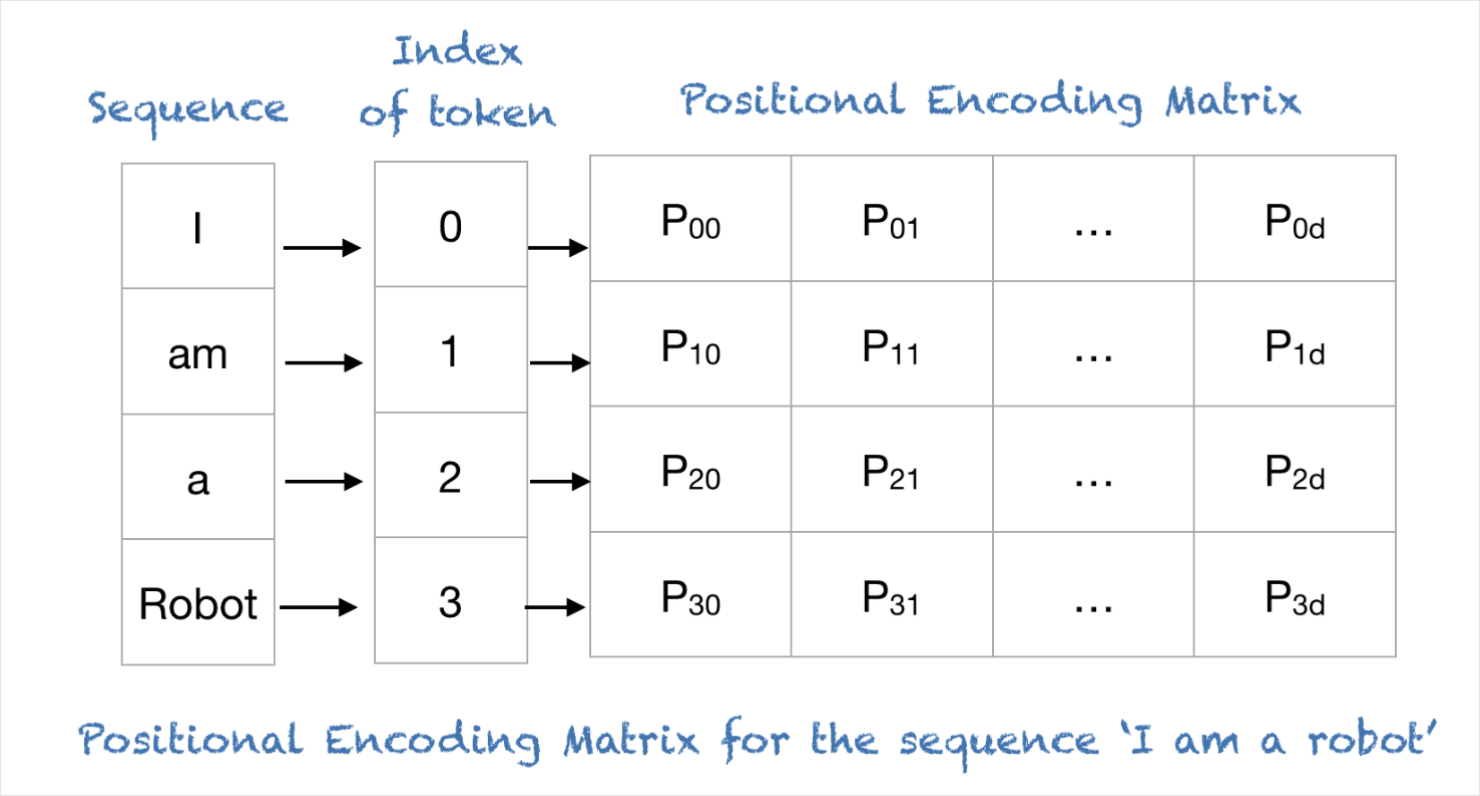


Figure 1: The Transformer - model architecture.

Attention is all you need (2017)

Encoder + Decoder architecture

positional encoding



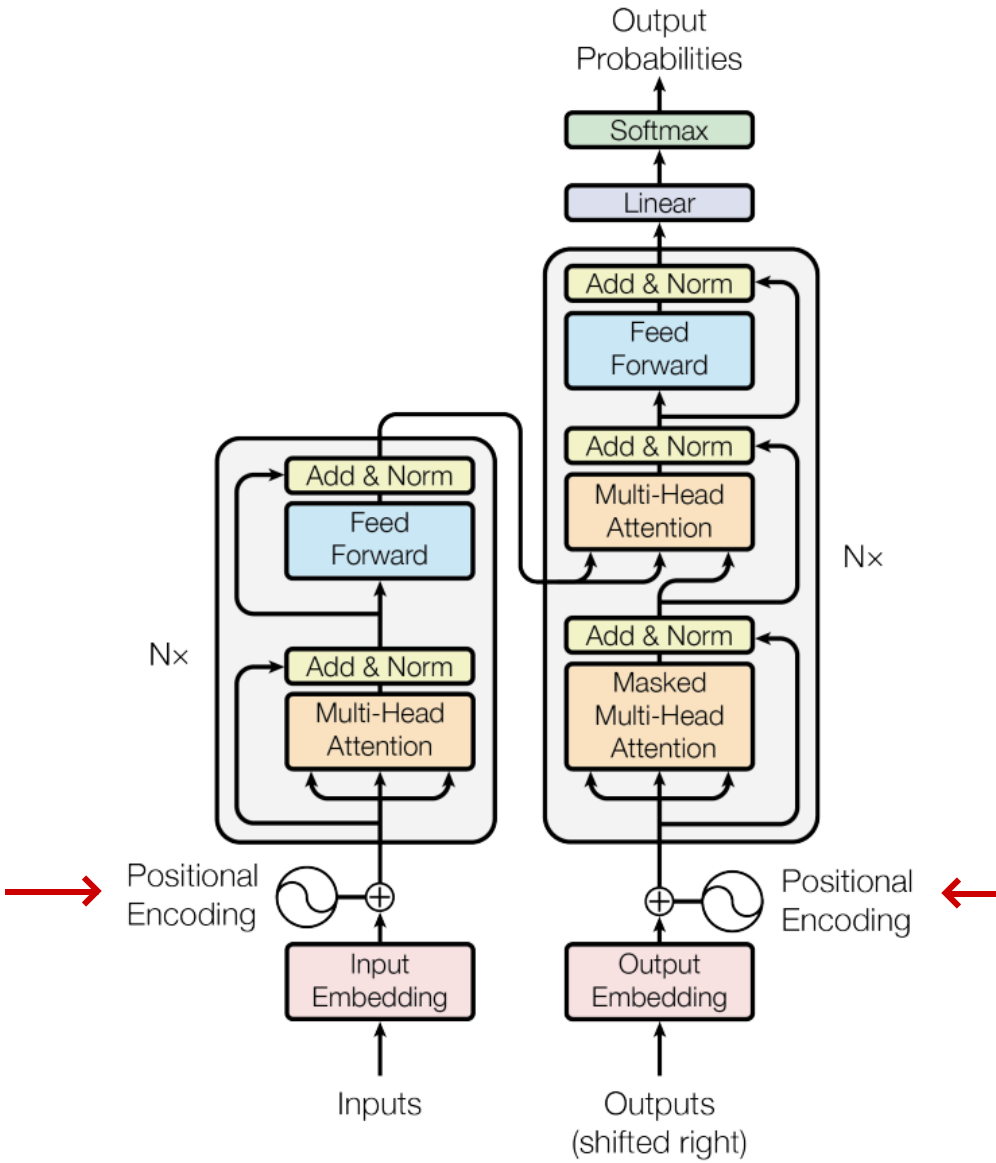


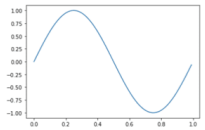
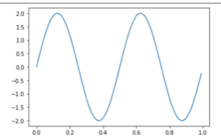
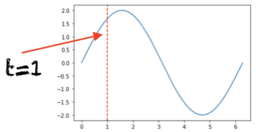
Figure 1: The Transformer - model architecture.

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Attention is all you need (2017)

Encoder + Decoder architecture

positional encoding

Equation	Graph	Frequency	Wavelength
$\sin(2\pi t)$		1	1
$\sin(2 * 2\pi t)$		2	1/2
$\sin(t)$		$1/2\pi$	2π
$\sin(ct)$	Depends on c	$c/2\pi$	$2\pi/c$

$$P(k, 2i) = \sin\left(\frac{k}{n^{2i/d}}\right)$$

$$P(k, 2i+1) = \cos\left(\frac{k}{n^{2i/d}}\right)$$

Here:

k: Position of an object in the input sequence, $0 \leq k < L/2$

d: Dimension of the output embedding space

$P(k, j)$: Position function for mapping a position k in the input sequence to index (k, j) of the positional matrix

n: User-defined scalar, set to 10,000 by the authors of Attention Is All You Need.

i: Used for mapping to column indices $0 \leq i < d/2$, with a single value of i maps to both sine and cosine functions

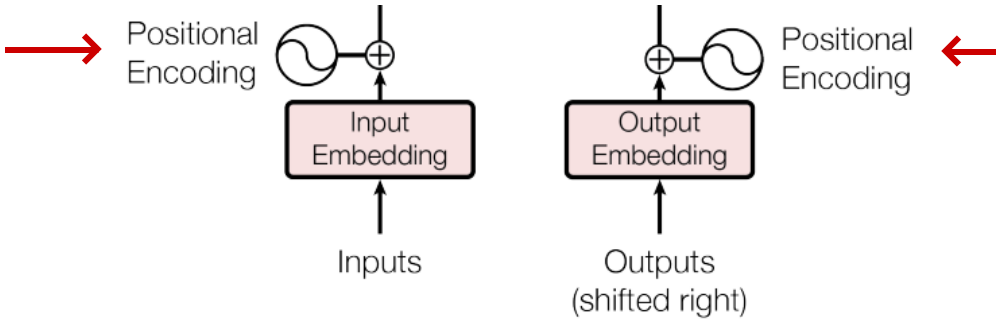


Figure 1: The Transformer - model architecture.

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Attention is all you need (2017)

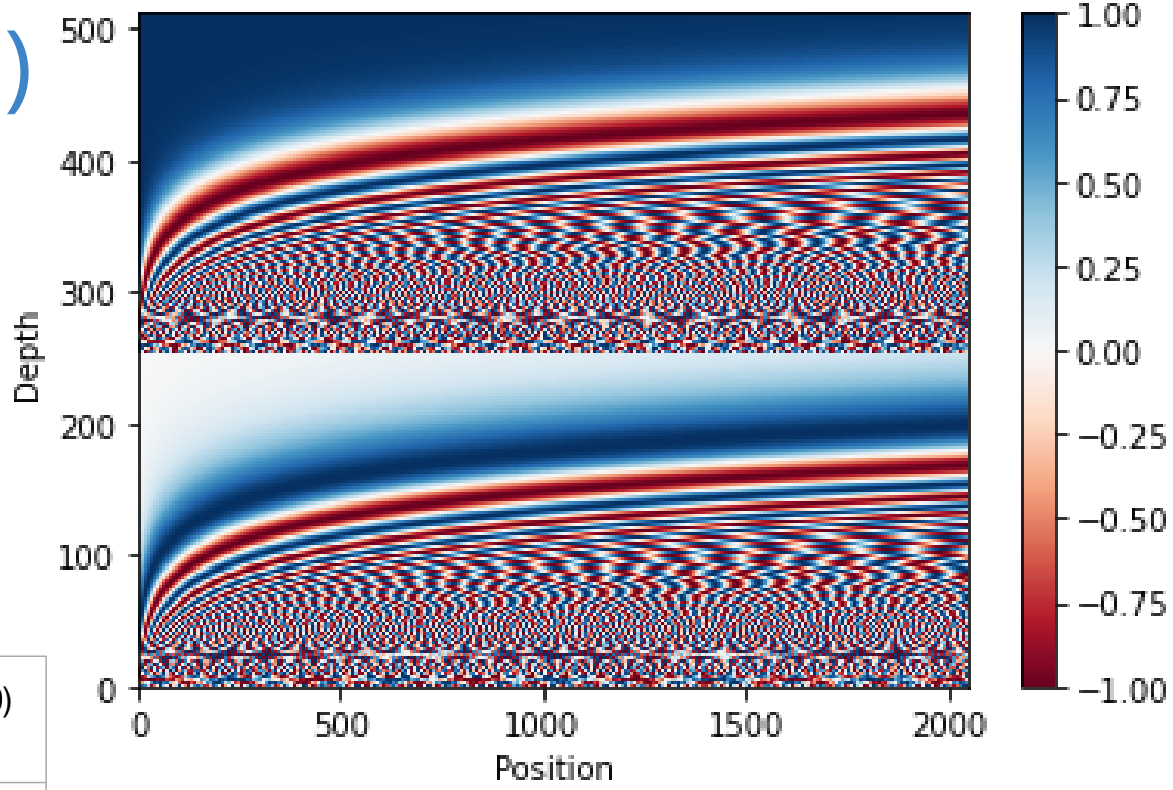
Encoder + Decoder architecture

positional encoding

Positional Encoding Matrix with d=4, n=100

Sequence	Index of token, k	i=0	i=0	i=1	i=1
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'



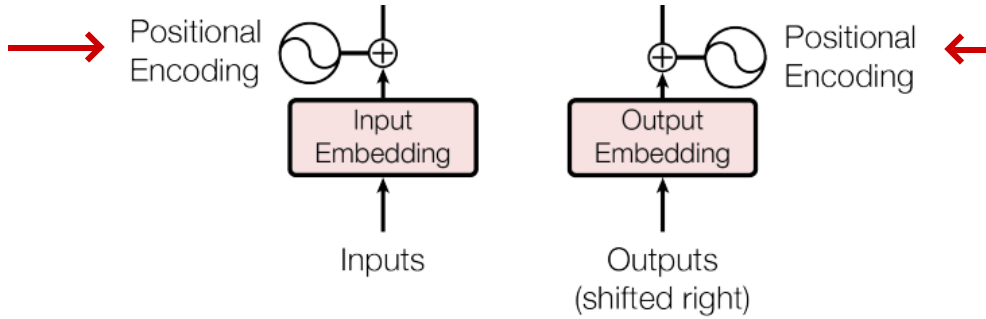


Figure 1: The Transformer - model architecture.

https://machinelearningmastery.com/a-gentle-introduction-to-positional-encoding-in-transformer-models-part-1/

Attention is all you need (2017)

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

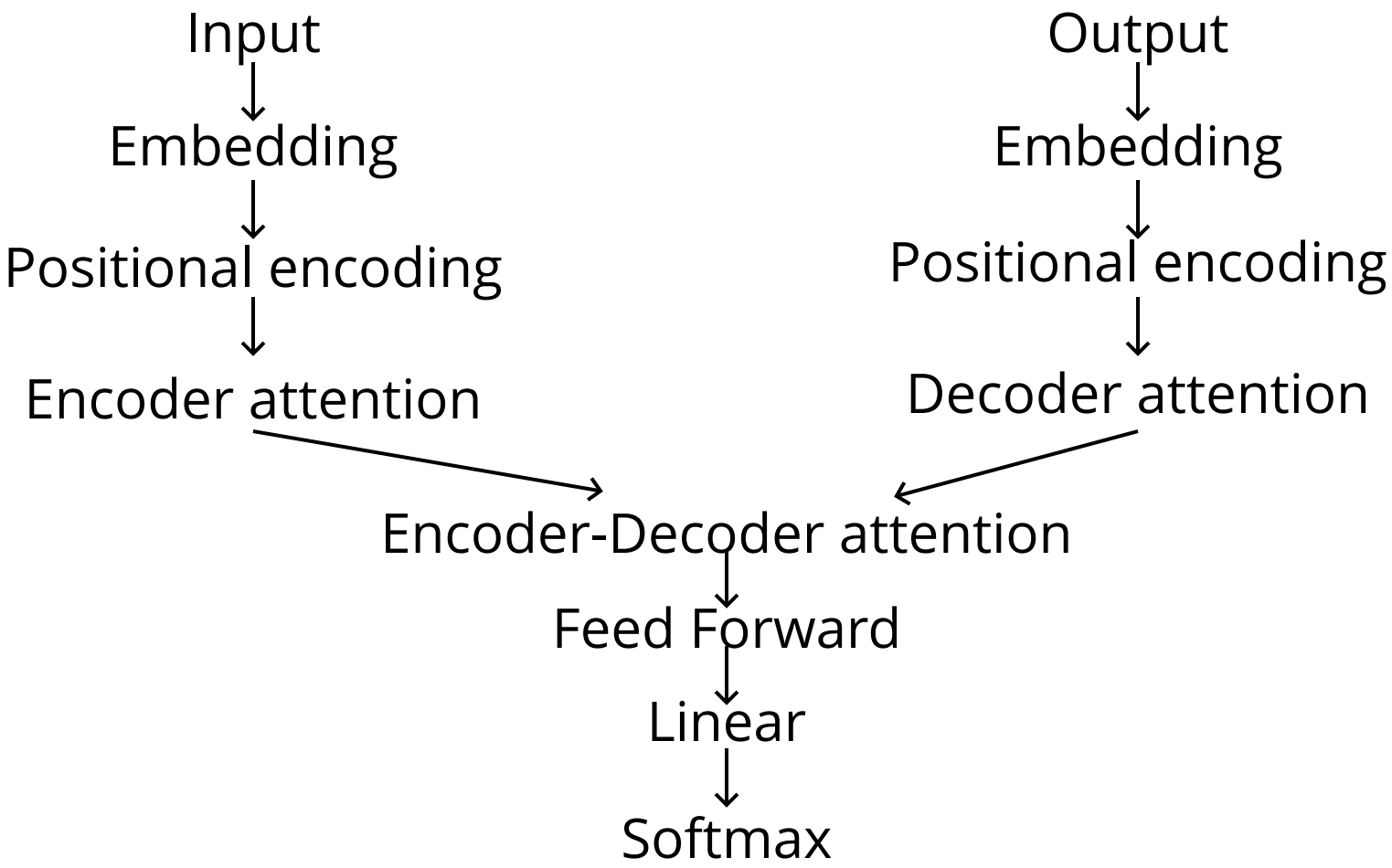
Sparse is Enough in Scaling Transformers

Sebastian Jaszczur, Aakanksha Chowdhery, Afroz Mohiuddin, Łukasz Kaiser, Wojciech Gajewski, Henryk Michalewski, Jonni Kanerva

Large Transformer models yield impressive results on many tasks, but are expensive to train, or even fine-tune, and so slow at decoding that their use and study becomes out of reach. We address this problem by leveraging sparsity. We study sparse variants for all layers in the Transformer and propose Scaling Transformers, a family of next generation Transformer models that use sparse layers to scale efficiently and perform unbatched decoding much faster than the standard Transformer as we scale up the model size. Surprisingly, the sparse layers are enough to obtain the same perplexity as the standard Transformer with the same number of parameters. We also integrate with prior sparsity approaches to attention and enable fast inference on long sequences even with limited memory. This results in performance competitive to the state-of-the-art on long text summarization.

Attention is all you need (2017)

Encoder + Decoder architecture



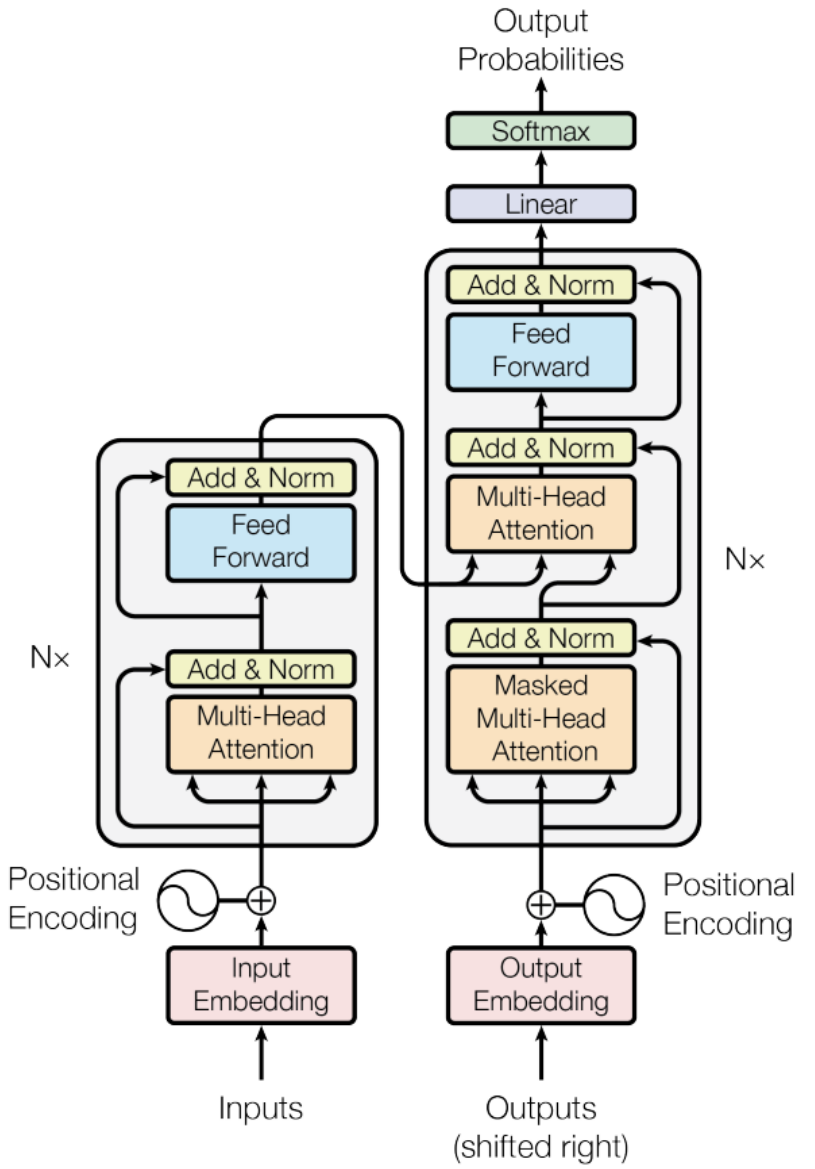


Figure 1: The Transformer - model architecture.

LAB:

show that the keras example of time series analysis with tensorflow..... is wrong!!

- visualize and familiarize with the data (which the authors of the notebook had not done)

- create a model like the one that they created (takes a while to train, I saved a pretrained version for you)

- look a the loss, which they did not do

https://github.com/fedhere/MLPNS_FBianco/blob/main/transformers/assess_TS_classification_w_tensorflow.ipynb

Timeseries classification with a Transformer model

Author: Theodoros Ntakouris
Date created: 2021/06/25
Last modified: 2021/08/05
Description: This notebook demonstrates how to do timeseries classification using a Transformer model.

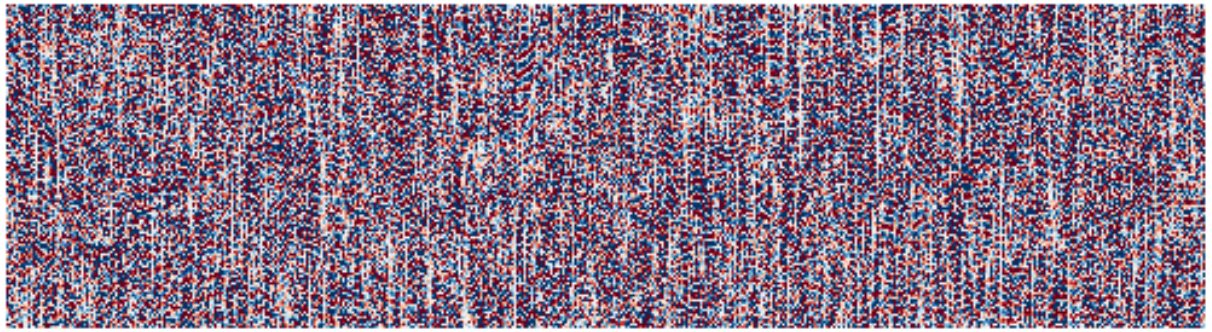
THIS KERAS EXAMPLE OF APPLICATION OF TRANSFORMERS TO TIME SERIES ANALYSIS IS **WRONG**

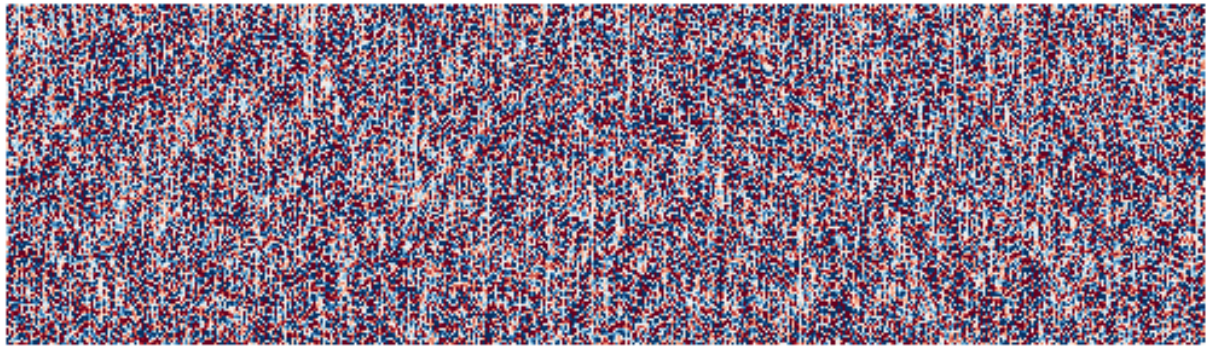
My student Willow Fox Fortino found that out...

Text

The FordA dataset

This data was originally used in a competition in the IEEE World Congress on Computational Intelligence, 2008. The classification problem is to diagnose whether a certain symptom exists or does not exist in an automotive subsystem. Each case consists of 500 measurements of engine noise and a classification. There are two separate problems: For FordA the Train and test data set were collected in typical operating conditions, with minimal noise contamination.





READ IN DEPTH

Either - Attention is all you need

Or - On the danger of stochastic parrots

A video on transformer which I think is really good!

https://www.youtube.com/watch?v=4Bdc55j80l8

A video on attention (with a different accent than the one I subjected you all this time!)

https://www.youtube.com/watch?v=-9vVhYEXeyQ

resources

Tutorial

https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial6/Transformers_and_MHAttention.html