

Workshop: Introduction to Deep Learning

Lecture 3 : Deep Learning (YOLO, RNN, LSTM, GRU, Transformers)

Petia Georgieva

Department of Electronics Telecommunications and Informatics (DETI)
Institute of Electronics and Informatics Engineering of Aveiro (IEETA)
Intelligent Robotics and Systems Lab
University of Aveiro, Portugal (<https://www.ua.pt/>)
Email: petia@ua.pt

European Union-NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project N0 BG-RRP-2.004-0005.

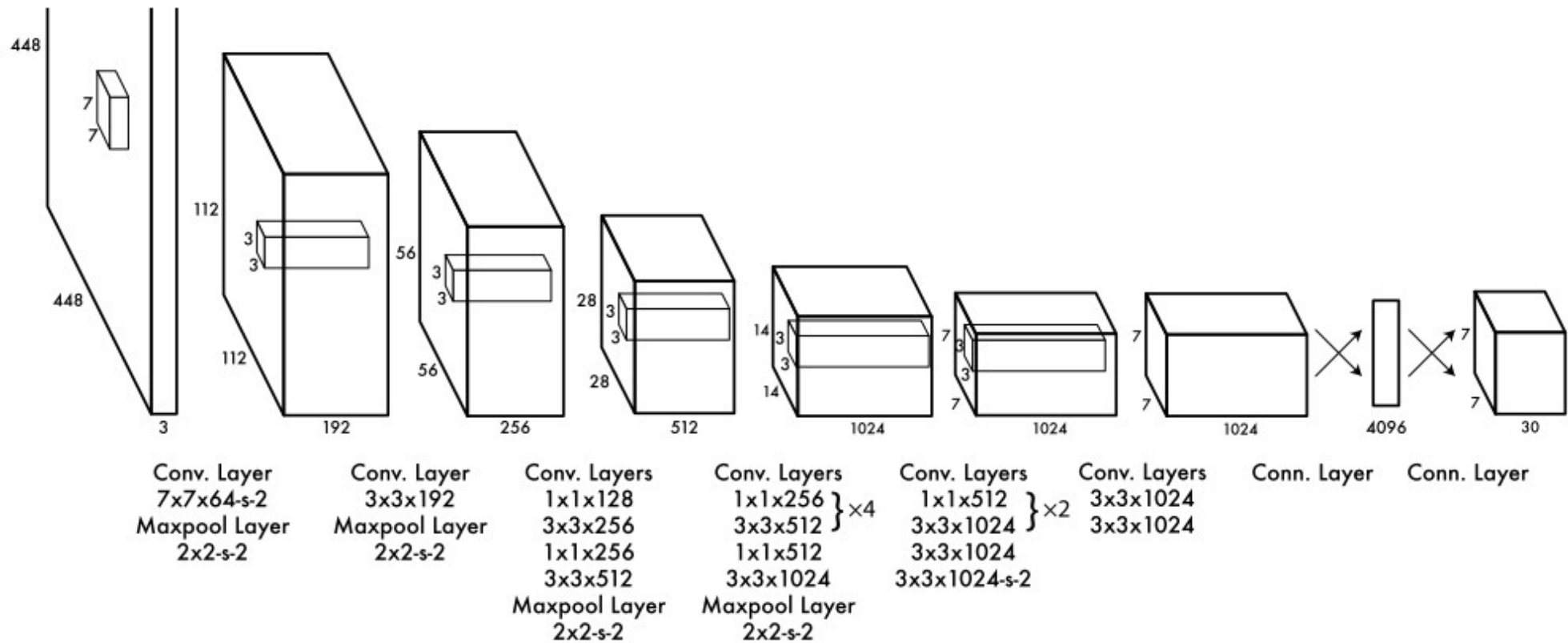


Outline

- 1. Object detection with YOLO**
- 2. Sequential data –
Recurrent Neural Networks (RNN):
Long-Short Term Memory (LSTM);
Gated Recurrent Unit (GRU)**
- 3. Time Series Forecasting**
- 4. Transformers**

1. Object detection with YOLO

Object detection with YOLO (You Only Look Once)



YOLO - CNN network for both classification and localising the object using bounding boxes.

24 convolutional layers + 2 fully connected layers.

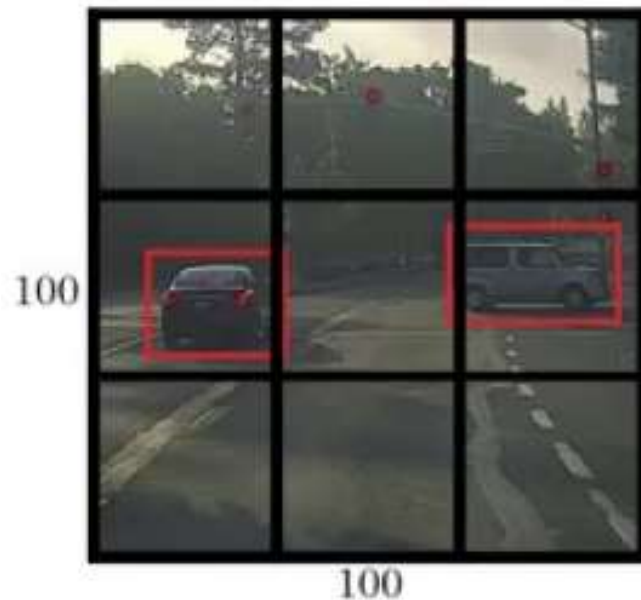
Conv layers pretrained on ImageNet dataset.

*Redmon et al, 2015, “You Only Look Once: Unified, Real-Time Object Detection”

(<https://arxiv.org/abs/1506.02640>)

Redmon & Farhadi, 2016 (<https://arxiv.org/abs/1612.08242>).

YOLO (You Only Look Once) algorithm



\leq cell label: $[0, ?, ?, ?, ? \cdot ?, ?; ?]$
? – “don’t care”

\leq cell label: $[1, b_x, b_y, b_h, b_w, 0, 0, 1]$

\leq cell label: $[0, ?, ?, ?, ? \cdot ?, ?; ?]$
? – “don’t care”

Let we have 100x100 pixels input image. We place a grid on this image.
In the original paper the grid is 19x19, here for illustration is 3x3 grid (9 cells).
Apply object classification and localization to each cell.

How to define the labels used for training:

For each cell, the label is 8 dimensional vector (if we have 3 classes)

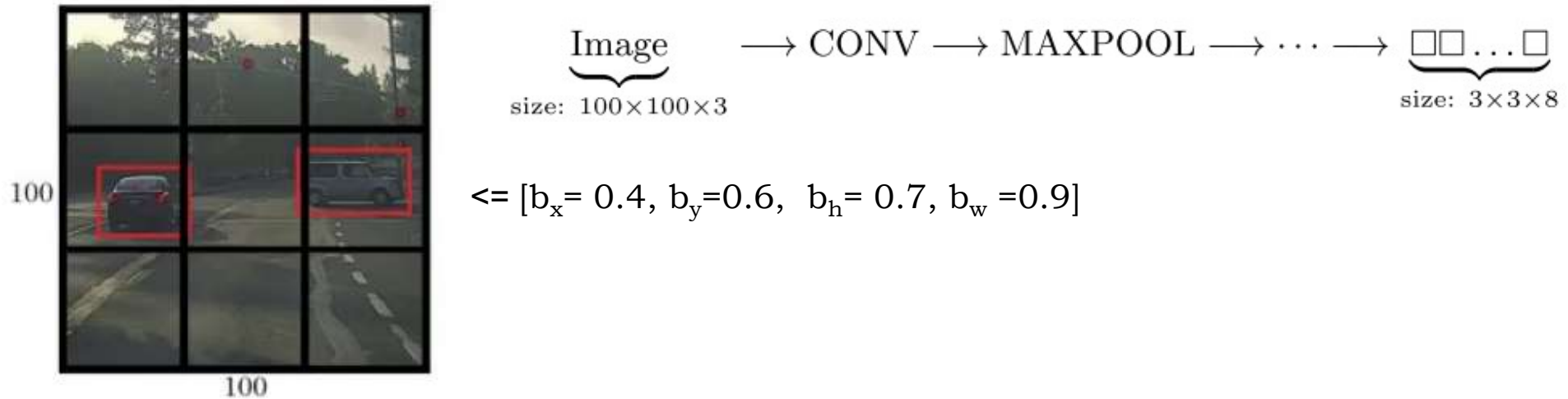
$y = [p_c, b_x, b_y, b_h, b_w, c1, c2, c3]$, where

- p_c (0 or 1) specifies if there is or not an object in that cell.
- $[b_x, b_y, b_h, b_w]$ specify the bounding box if there is an object in that cell.
- $c1; c2; c3$ (0 or 1) - to assign the class (e.g. person, bicycle, car)

YOLO algorithm assigns the object only to the cell containing the midpoint of the object.

Since we have 3x3 grid, the total volume of the target output Y is 3x3x8.

YOLO (You Only Look Once) algorithm



To train YOLO, input e.g. 100x100x3 original image (X).

The network has the usual conv layers, max pool layers, and so on.

It has to end up with 3x3x8 output volume that is compared with the target labels Y (also 3x3x8).

If the grid is much finer (e.g. 19x19), it reduces the chance of multiple objects assigned to the same grid cell.

How to specify the bounding boxes $[b_x, b_y, b_h, b_w]$?

YOLO assumes the upper left point of the cell is (0,0), the lower right point is (1,1).

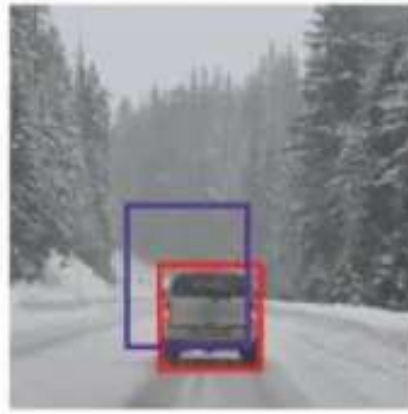
The bounding boxes are specified relative to the grid cell.

$[b_x, b_y]$ represent the midpoint of the object, have to be <1 and >0 .

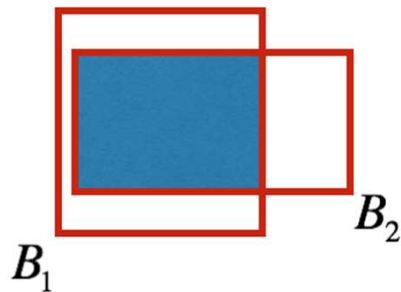
$[b_h, b_w]$ represent the height and width of the bounding box, they could be >1 if the object is bigger than the cell.

There are other ways to specify the bounding boxes.

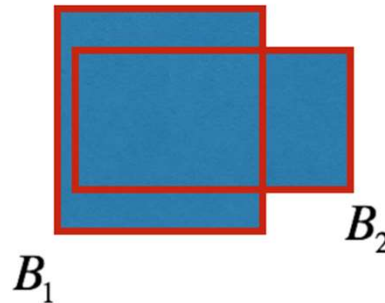
Intersection Over Union



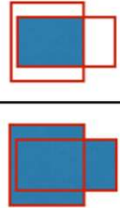
Intersection



Union



Intersection over Union

$$IoU = \frac{B_1 \cap B_2}{B_1 \cup B_2} = \frac{\text{Intersection}}{\text{Union}}$$


How to tell if the object detection algorithm is working well?

We use the function called intersection over union (IoU) .

IoU computes the intersection over union of the two bounding boxes (red is the ground true, purple is the YOLO prediction).

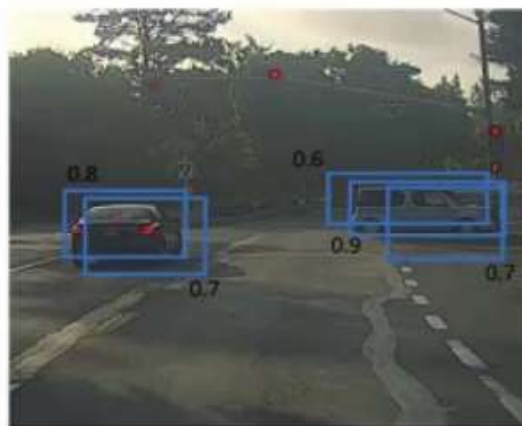
IoU is a measure of the overlap between two bounding boxes (measure of similarity).

Higher IoU, better prediction:

IoU=1 => perfect detection

IoU >=0.5 => “correct” (by convention)

Non-max Suppression



Common object detection problem: the algorithm finds multiple detections of same object =>

Non-max suppression is a way to guarantee that each object is detected only once.

Let's say we want to detect 3 classes - persons, bikes, cars.

We placed 19x19 grid. Technically each car has just one midpoint, so it should be assigned just to one grid cell.

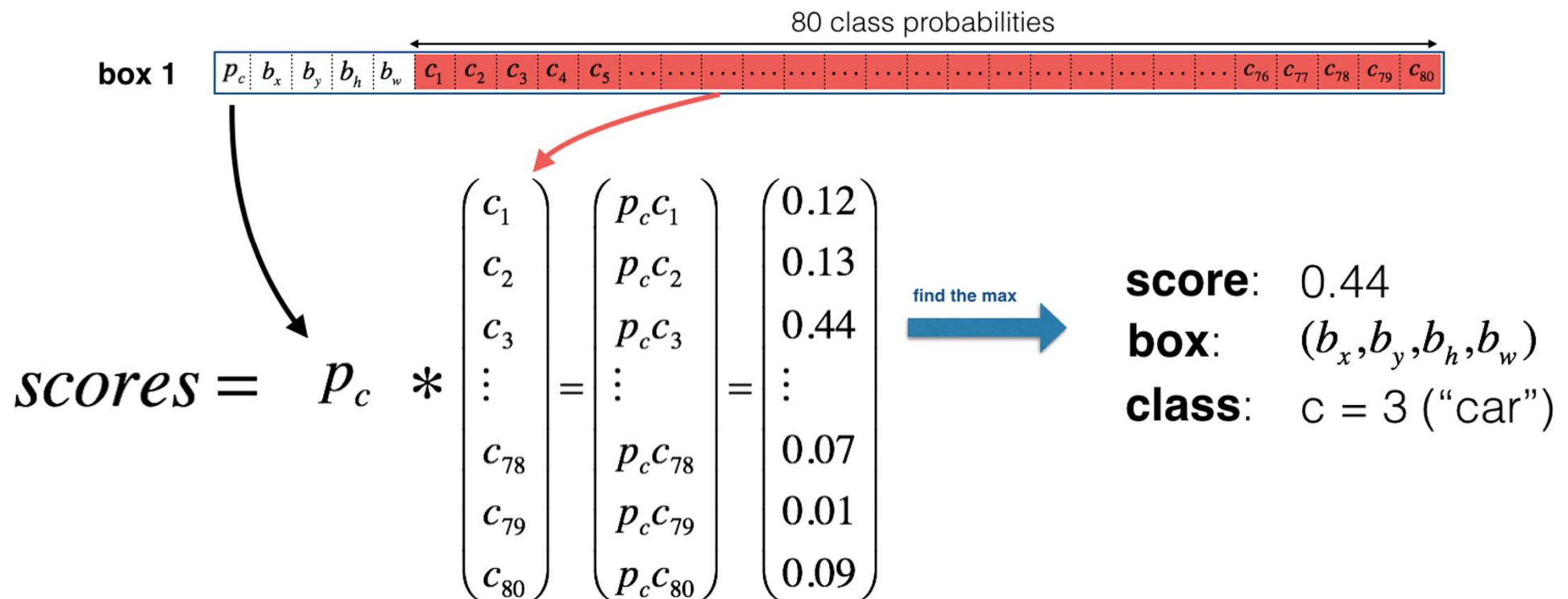
In practice, we run object classification & localization algorithm for every cell.

It's possible that many neighbour cells claim they found a car.

Non-max suppression cleans up these detections independently for each class (persons, bikes, cars).

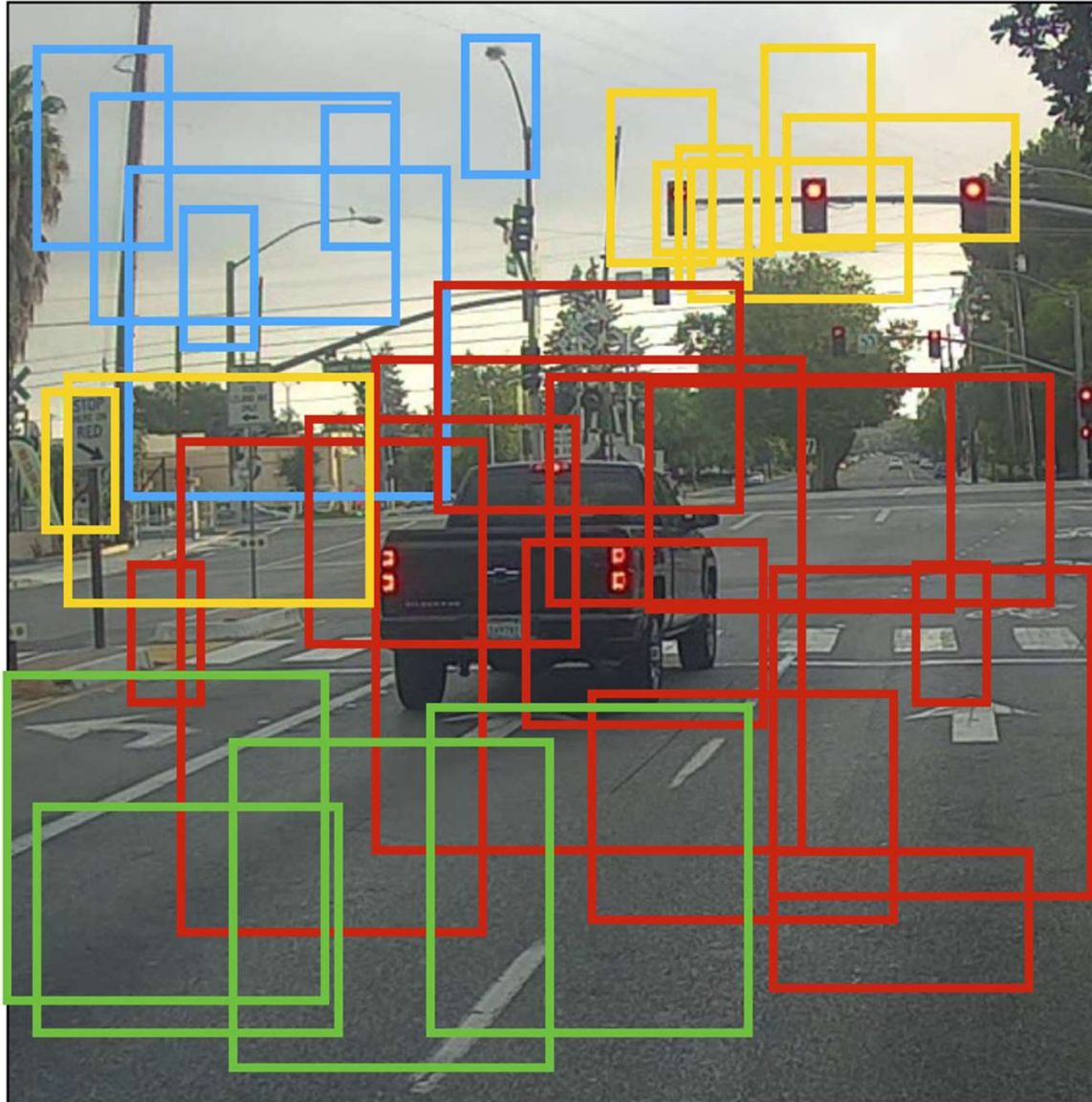
- 1) Discard all boxes with low probability (let say $p_c \leq 0.6$)
- 2) Pick the box with the largest p_c . Output that as a prediction.
- 3) Discard any remaining box with high overlap (e.g. $\text{IoU} \geq 0.5$) with the box picked in step 2).

CNN output processing & filtering



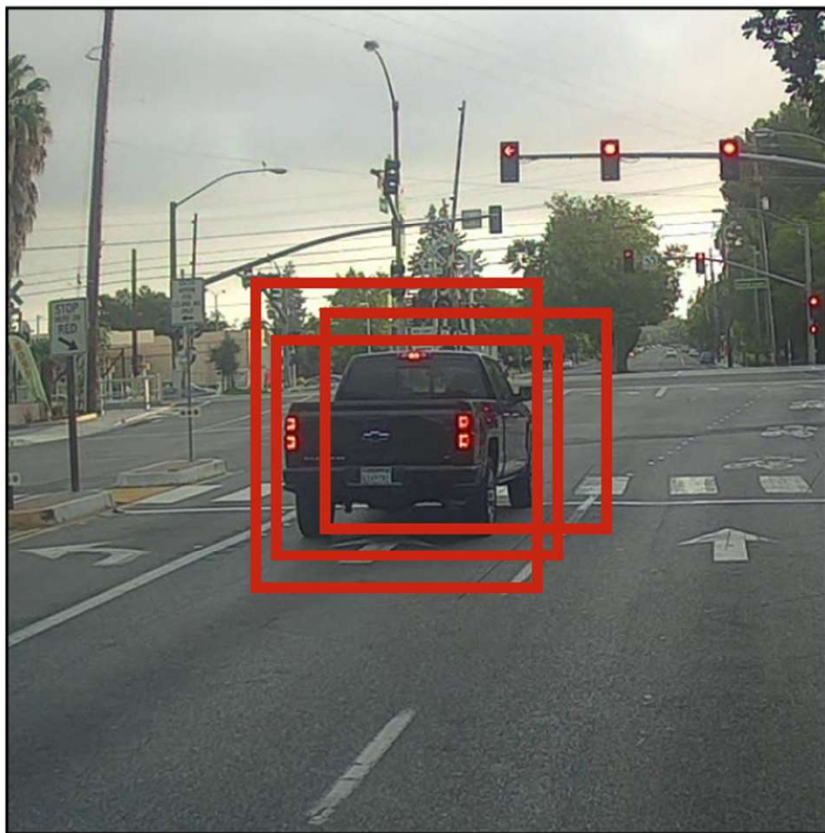
the box (b_x, b_y, b_h, b_w) has detected $c = 3$ ("car") with probability score: 0.44

Remove boxes with low probability (scores)

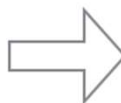


Non Max Supression (NMS)

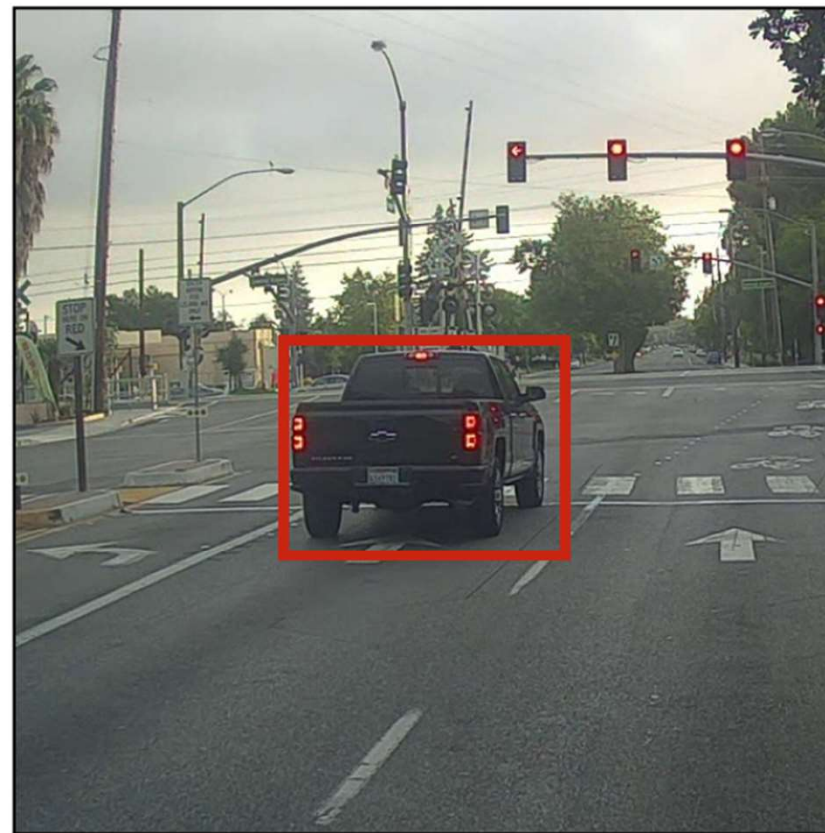
Before non-max suppression



Non-Max
Suppression



After non-max suppression



2. Sequential data –
Recurrent Neural Networks (RNN)
Long-Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)

Examples of sequence data

x (model input)

y (model output)

Speech recognition



=> "The quick brown fox jumped over the lazy dog."

Sentiment classification

"There is nothing to like in this movie."



DNA sequence analysis

AGCCCCTGTGAGGAACTAG

=> AG**CCCCTGTGAGGAACTAG**

Machine translation

Voulez-vous chanter avec moi?

=> Do you want to sing with me?

Video activity recognition



=> Running

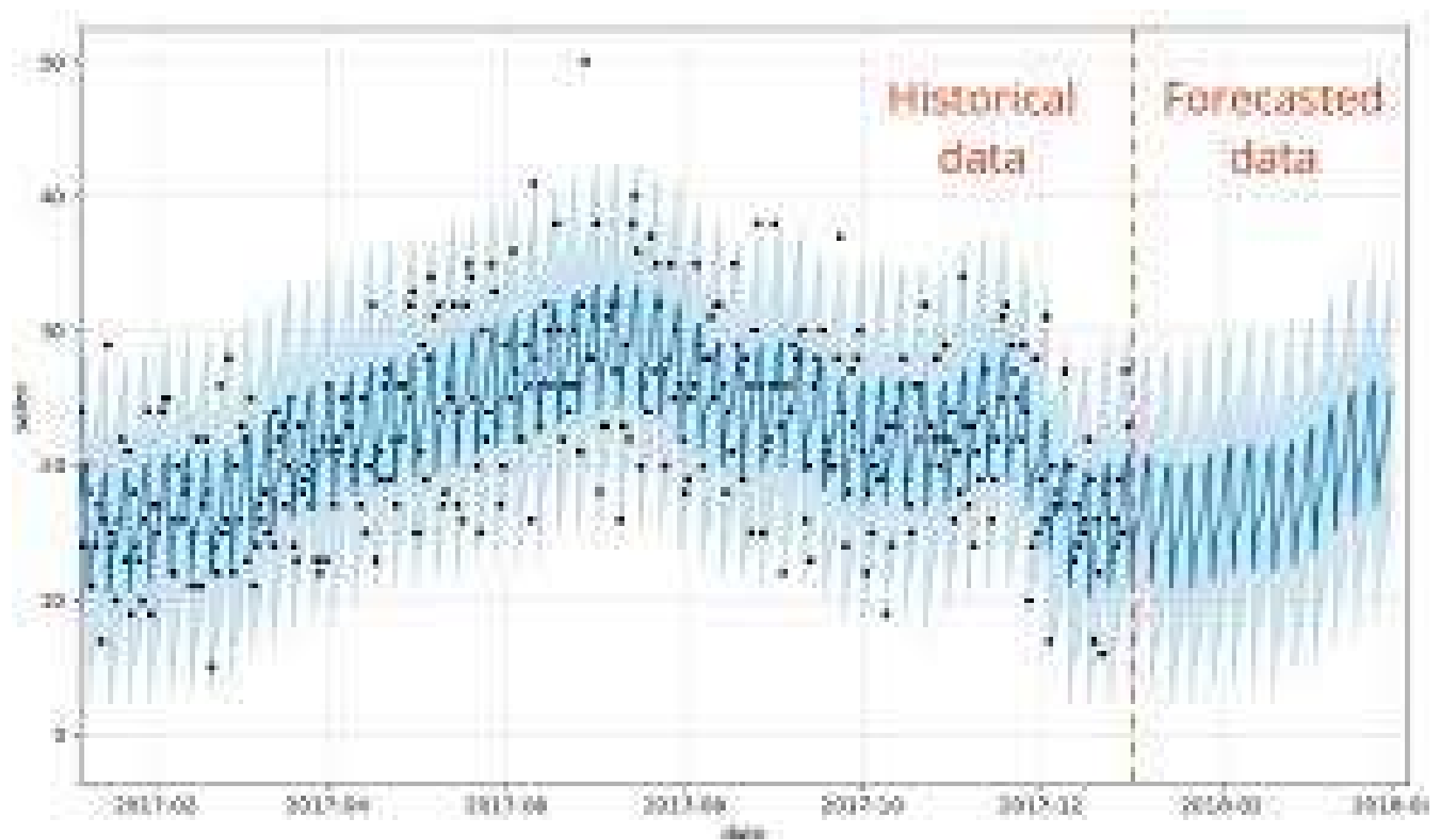
Named entity recognition

Yesterday, Harry Potter met Hermione Granger.

=> Yesterday, **Harry Potter** met **Hermione Granger**.

x and y are both sequences, or only x is a sequence, or only y is a sequence.

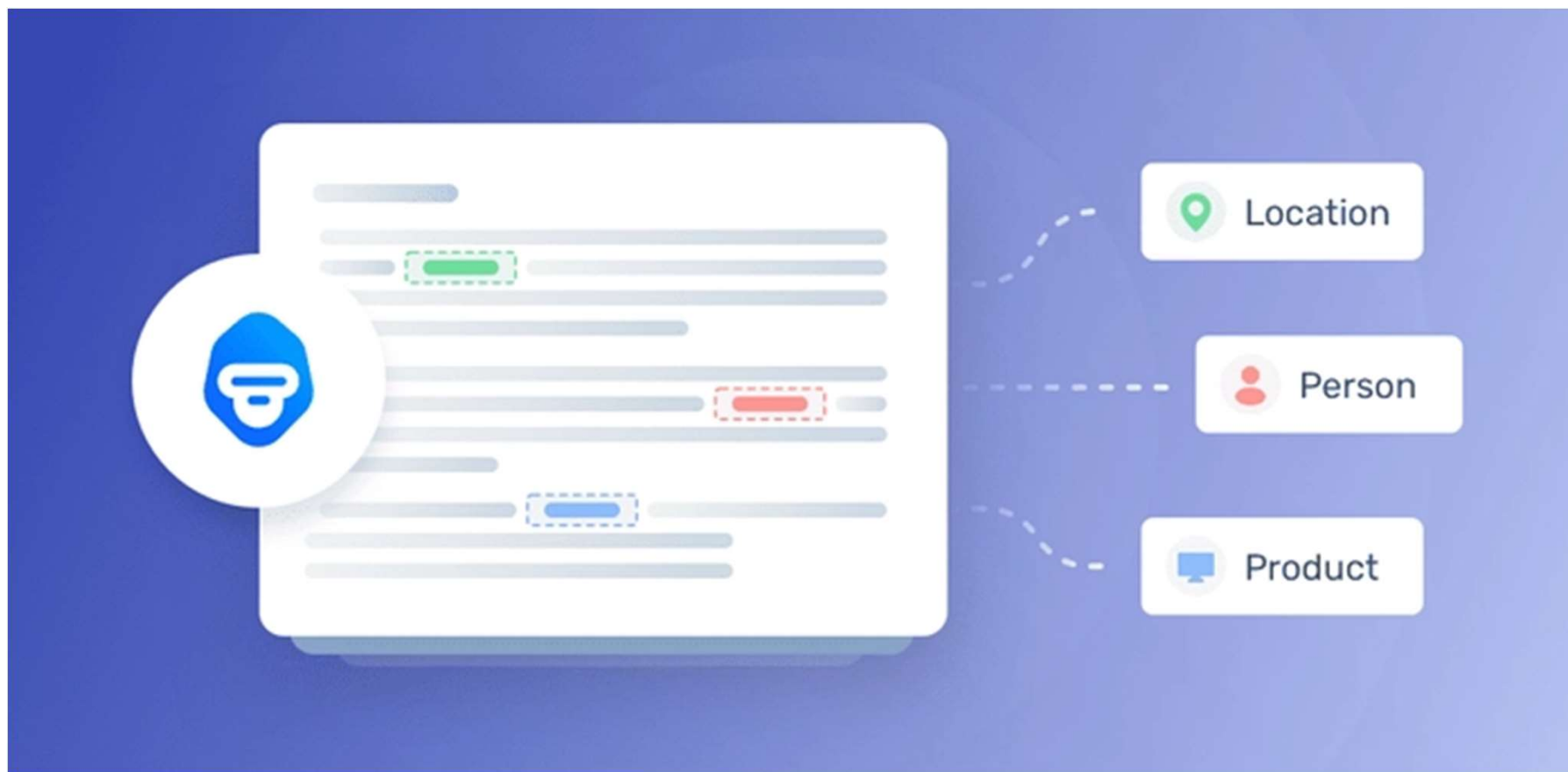
Sequence data – sensor time series



Named Entity Recognition (NER)

Named Entity Recognition is to find people's names, companies names, locations, countries names, currency names, etc. in text.

Given an input sequence of words (X), the NER model has to automatically identifies named entities and classifies them into predefined categories.



NER example

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$... $x^{<9>}$

For this example the sequence length is 9 words (features $x^{<1>}$ $x^{<2>}$ $x^{<T_x>}$) $\Rightarrow T_x=9$

Target output y is binary vector with same length as the input
(1 if the word is name; 0 if not)

$y = [\begin{matrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{matrix}]$
 $\begin{matrix} y^{<1>} & y^{<2>} & y^{<3>} & & y^{<t>} & & & & y^{<T_y>} \end{matrix}$
 $T_y=9$

In this problem every input $x^{<i>}$ has an output $y^{<i>}$ \Rightarrow **length $T_y=T_x$**

Each sentence (example) can have different sequence length.

NLP - representing individual words

Natural Language Processing (NLP). Create vocabulary or download existing dictionary. For modern NLP, 10000 words is a small dictionary, 30-50 thousand is more common. Large internet companies use 1 million words dictionary.

Each word is represented by a binary vector with # elements = dimension of dictionary where only the index of the word in the dictionary =1, all others are 0 (**one-hot vector**).

word index (in dictionary)

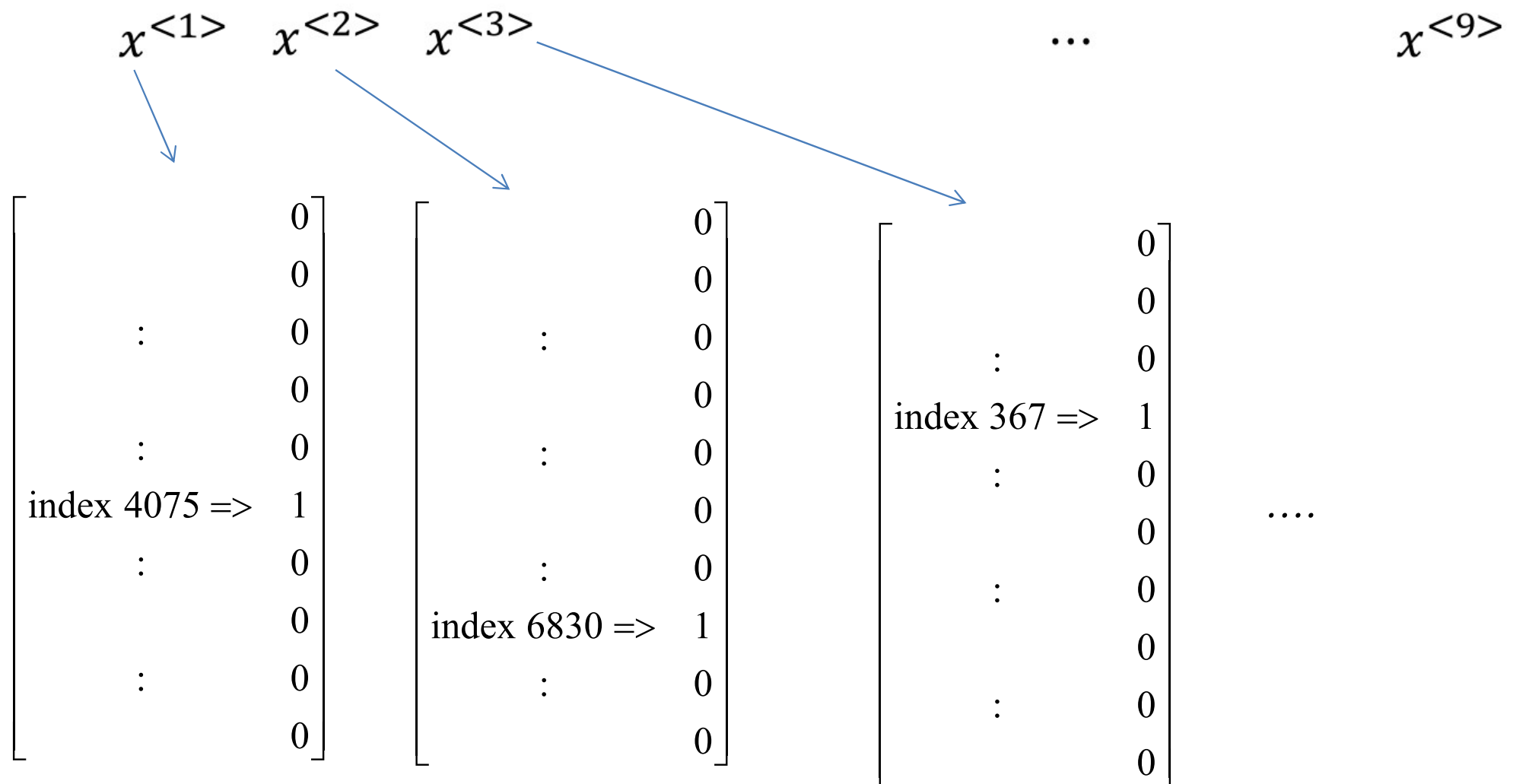
one-hot encoding

$a \Rightarrow$	1
$aaron \Rightarrow$	2
:	
$and \Rightarrow$	367
:	
$Harry \Rightarrow$	4075
:	
$Potter \Rightarrow$	6830
:	
$zulu \Rightarrow$	10000

$$Harry = \begin{bmatrix} 0 \\ 0 \\ : \\ 0 \\ : \\ 1 \\ : \\ 0 \\ : \\ 0 \end{bmatrix}$$

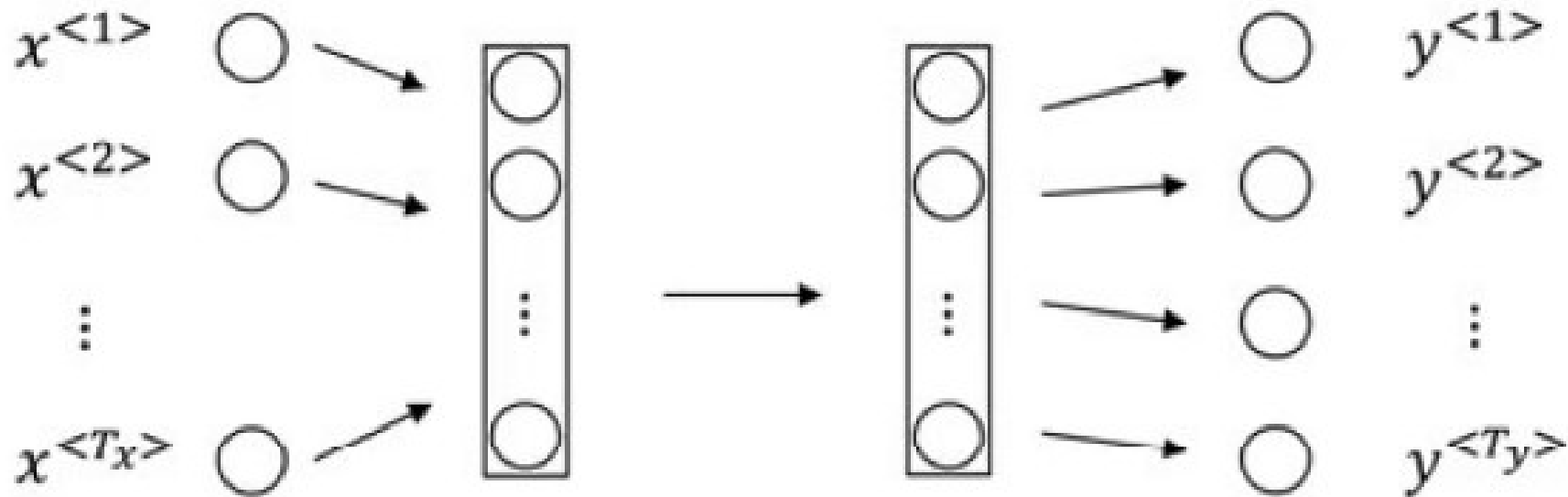
NLP – one hot encoding

x: Harry Potter and Hermione Granger invented a new spell.



<unk> - notation for words not in the dictionary. It can be added to encode all missing words that may appear in sentences.

Why not a standard neural network?



Problems:

- 1) Inputs and outputs can have different lengths in different sentences.
- 2) High input dimension (e.g. $T_x \times$ dimension of dictionary) \Rightarrow too many trainable parameters.

Recurrent Neural Networks (RNN) – better solution

Recurrent Neural Network (RNN)

Read the sentence word by word (one time step per word).

Activation produced by 1st word is taken into account when read 2nd word.

Notation: inputs - $x^{<t>}$; outputs - $y^{<t>}$; hidden states - $a^{<t>}$

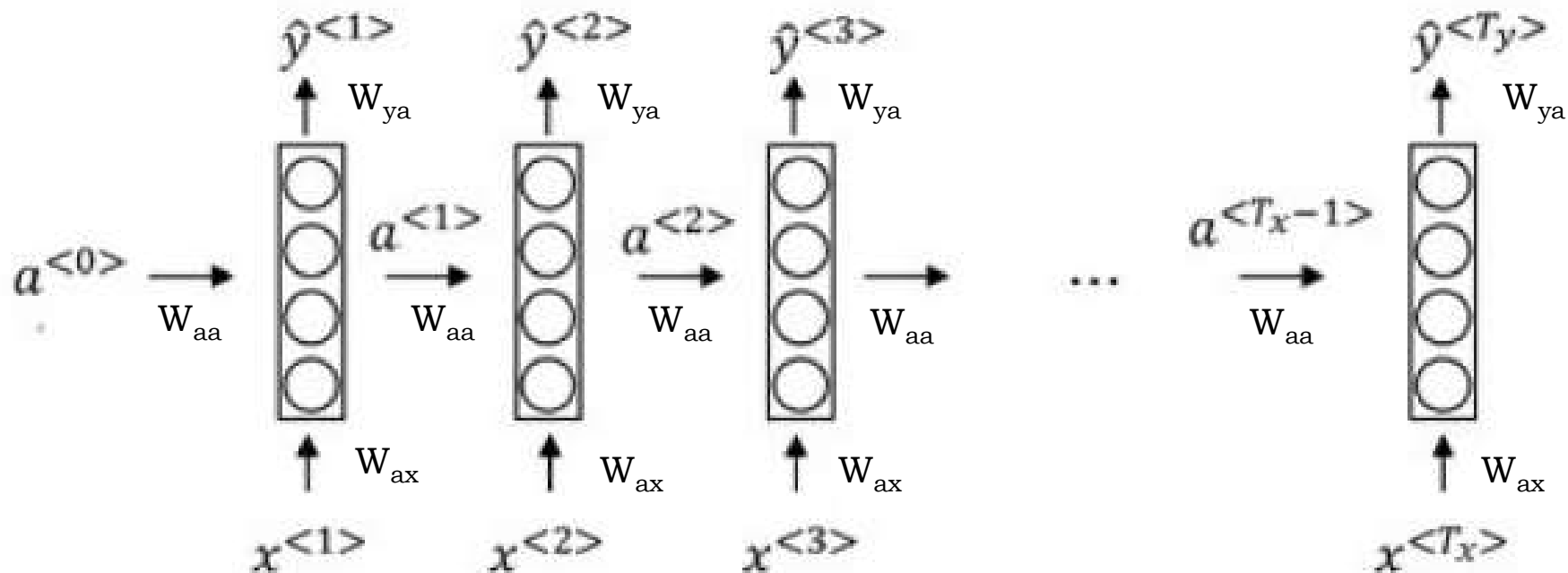
$a^{<0>}$ - initial state at time step 0, usually vector of zeros.

Previous results are passed as inputs => we get context !

Uni-directional RNN predicts current or future output based only on past information in the sequence.

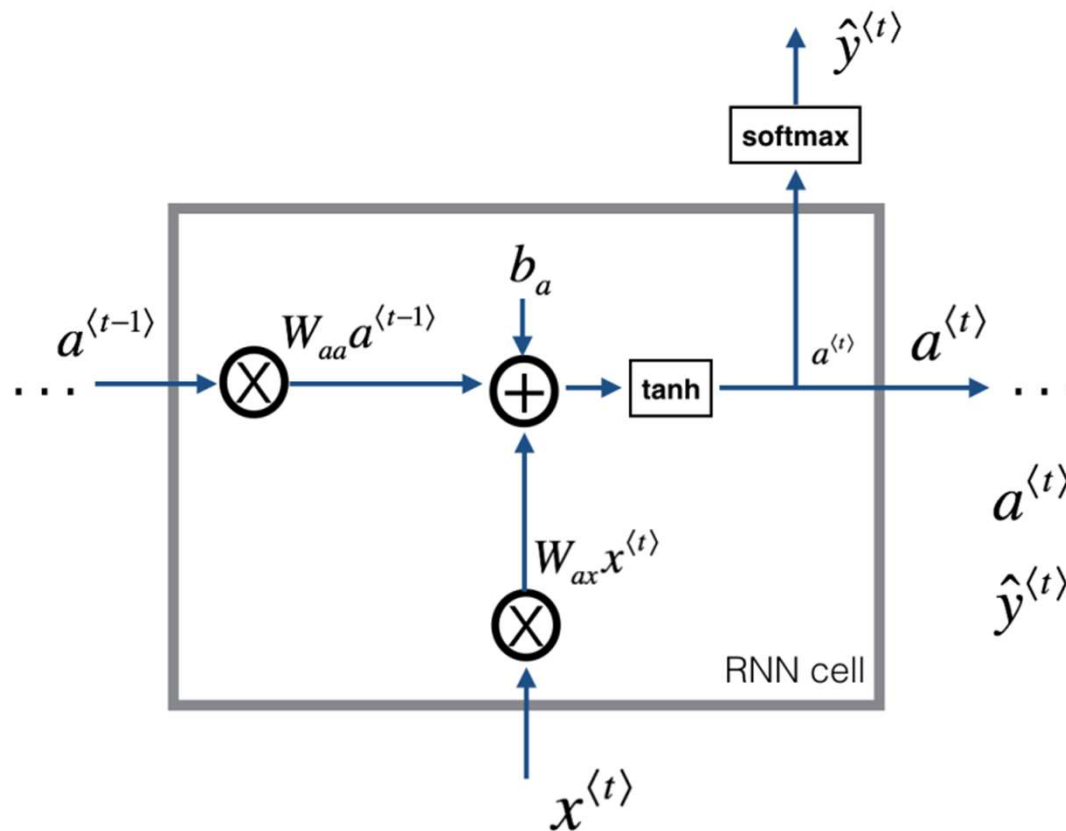
Bidirectional RNN (BRNN) predicts outputs based on the whole sentence.

Unrolled/unfolded RNN diagram



Basic RNN unit

Takes $x^{(t)}$ (current input) and $a^{(t-1)}$ (activation from previous step) as inputs, and computes $a^{(t)}$ (current activation). $a^{(t)}$ is then used to predict $y^{(t)}$ (current output) and passed forward to the next RNN unit (next time step). W_{aa} , W_{ax} , W_{ya} , b_a , b_y – the same RNN weights used in all time steps.



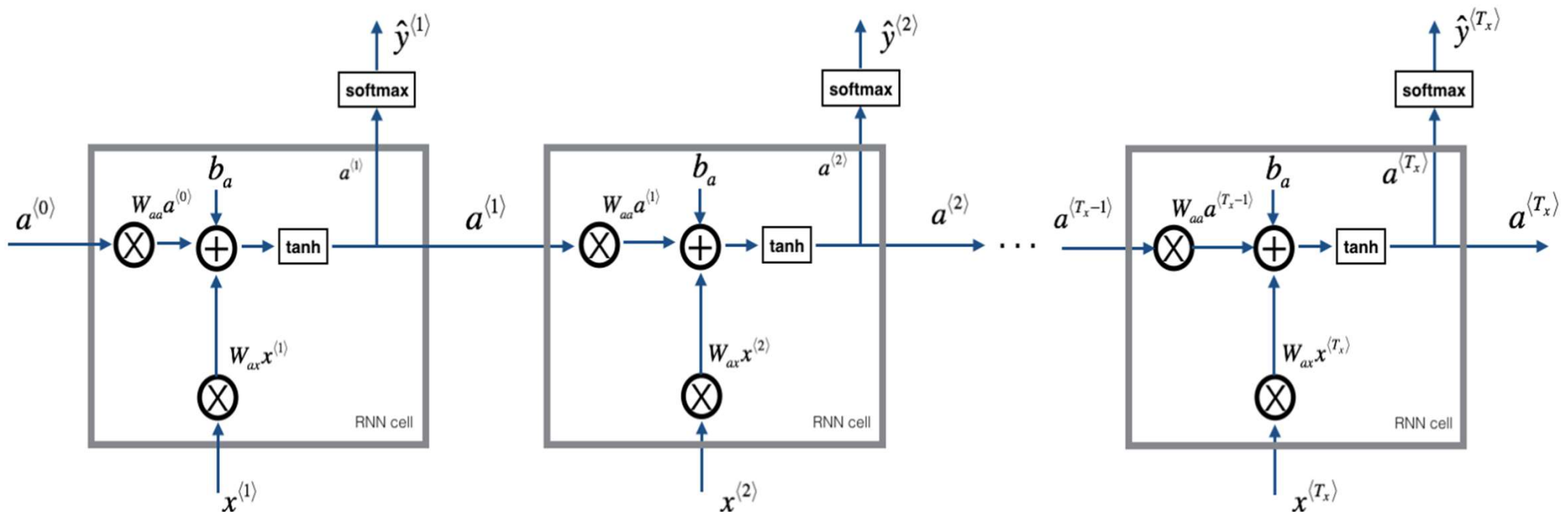
$$a^{(t)} = \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b_a)$$
$$\hat{y}^{(t)} = \text{softmax}(W_{ya}a^{(t)} + b_y)$$

RNN architecture

RNN is a sequence of basic RNN cells.

If input sequence is $\mathbf{x}=[\mathbf{x}^{<1>}, \mathbf{x}^{<2>}, \dots, \mathbf{x}^{<T_x>}] \Rightarrow$ RNN cell is copied T_x times.

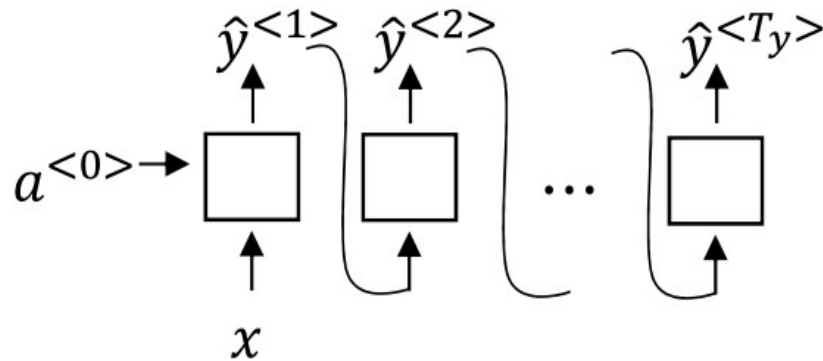
RNN outputs $\mathbf{y}=[\mathbf{y}^{<1>}, \mathbf{y}^{<2>}, \dots, \mathbf{y}^{<T_x>}]$



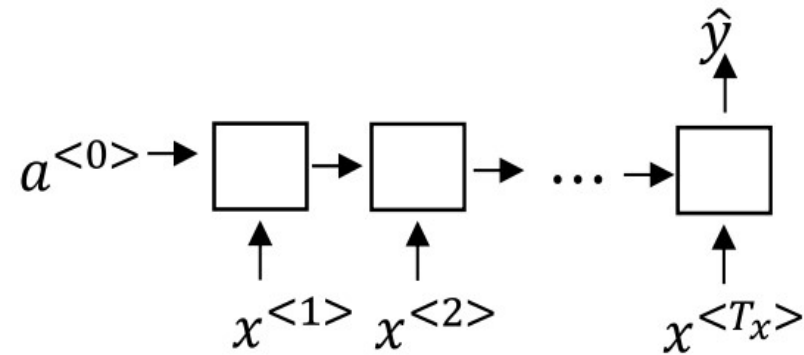
$$\mathbf{w}_{new} = \mathbf{w}_{old} - \alpha \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$$

Different Types of RNN

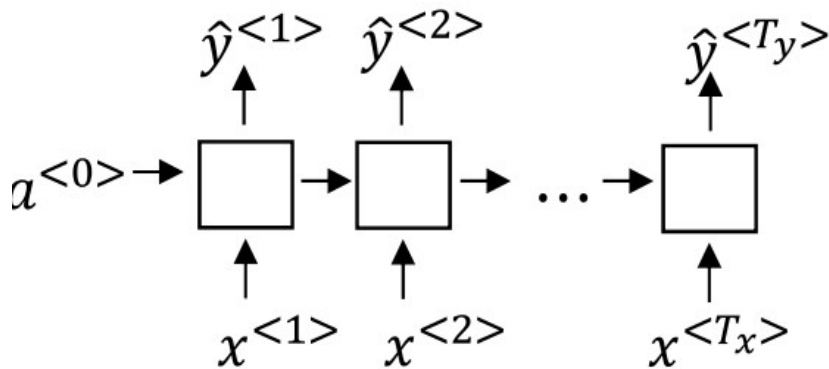
Input X and output Y may not be the same length



One to many (e.g. Music generation)

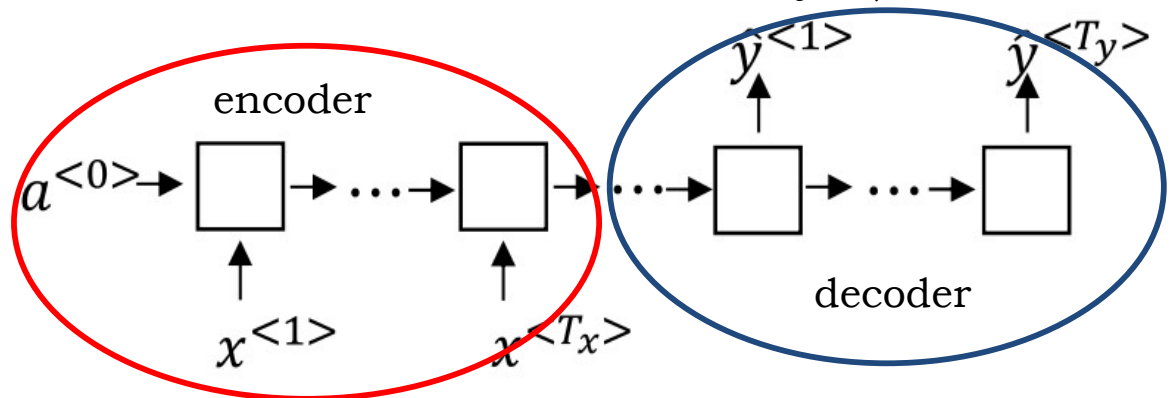


Many to one (e.g. Sentiment analysis)



Many to many

$T_x = T_y$ (e.g. Named entity rec.)



Many to many

T_x different from T_y (e.g. machine translation)

Sensor time series data forecasting is an example of Many to One.

RNN – vanishing/exploding gradients

RNN works well when each output $y^{(t)}$ can be estimated using "local" context (i.e. inf from inputs $x^{(t')}$ where t' is not too far from t).

But **RNN's suffers from vanishing gradient !!!**

$$w_{new} = w_{old} - \alpha \frac{\partial J(w)}{\partial w}$$

Vanishing gradient - gradient values become very small. Layers that get small gradients stops learning. Those are usually the earlier layers in the sequence model. Because these layers don't learn, RNN's can forget what it seen in longer sequences, thus having a short-term memory.

Standard RNN is not very good in capturing long-range dependencies, ex.:

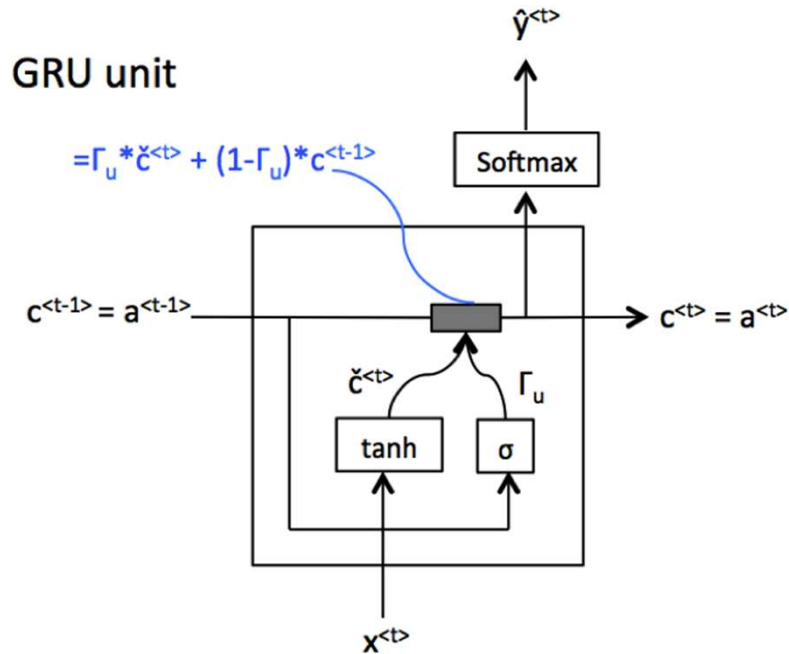
*The **girl** that entered the coffee shop with many friends **was** happy.*

*The **girls** that entered the coffee shop with many friends **were** happy.*

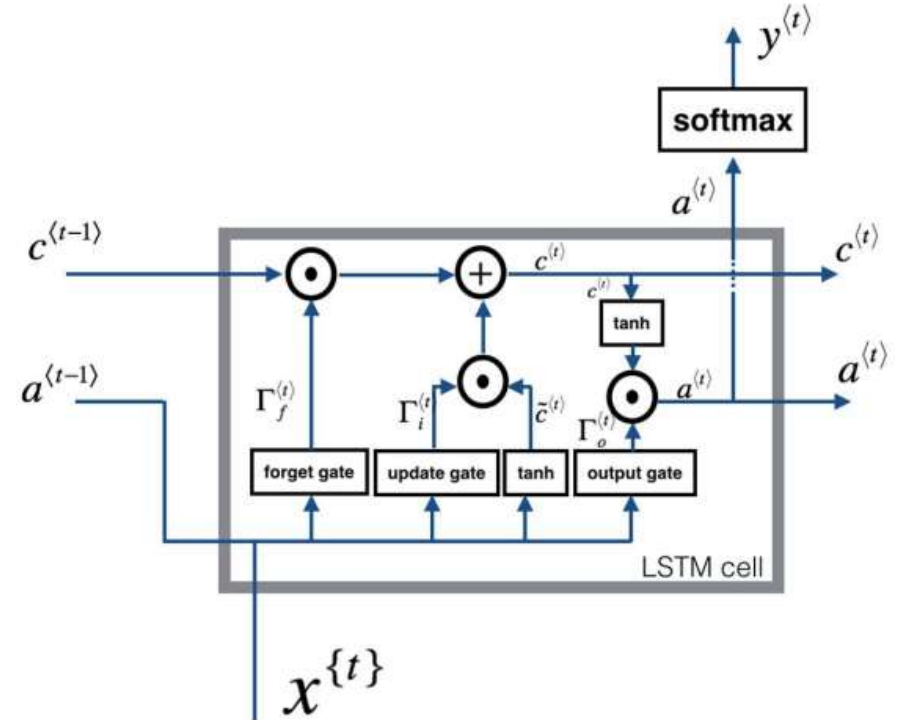
Need to remember “**girl-was**” or “**girls-were**”.

Long Short-Term Memory * (LSTM) and **Gated Recurrent Units** (GRU) – Better architectures to deal with vanishing gradients.

GRU



LSTM units



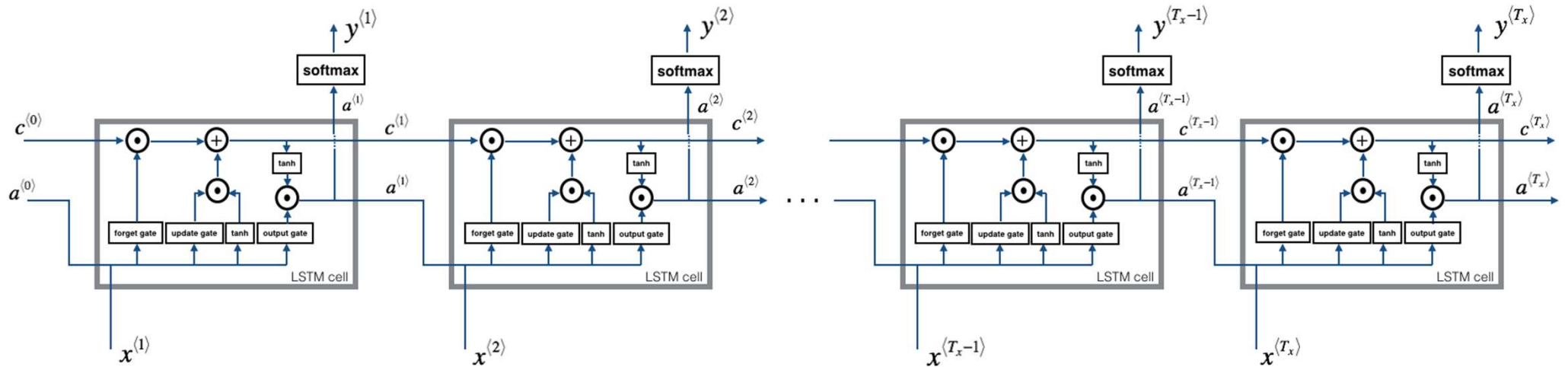
LSTM and GRU are variations of RNNs to capture better long range dependencies (connections) in sequences.

They mitigate short-term memory using mechanisms called memory cell and gates.

Gates remember (keep saved) bits of inf for many time steps.

Ex.: Read text and use LSTM/GRU to keep track of grammatical structures
=> if the subject is singular or plural.

LSTM network



LSTM/GRU network is a sequence of LSTM/GRU units.

LSTM's and GRU's are successfully applied in speech recognition, speech synthesis, natural language understanding, time series forecasting, etc.

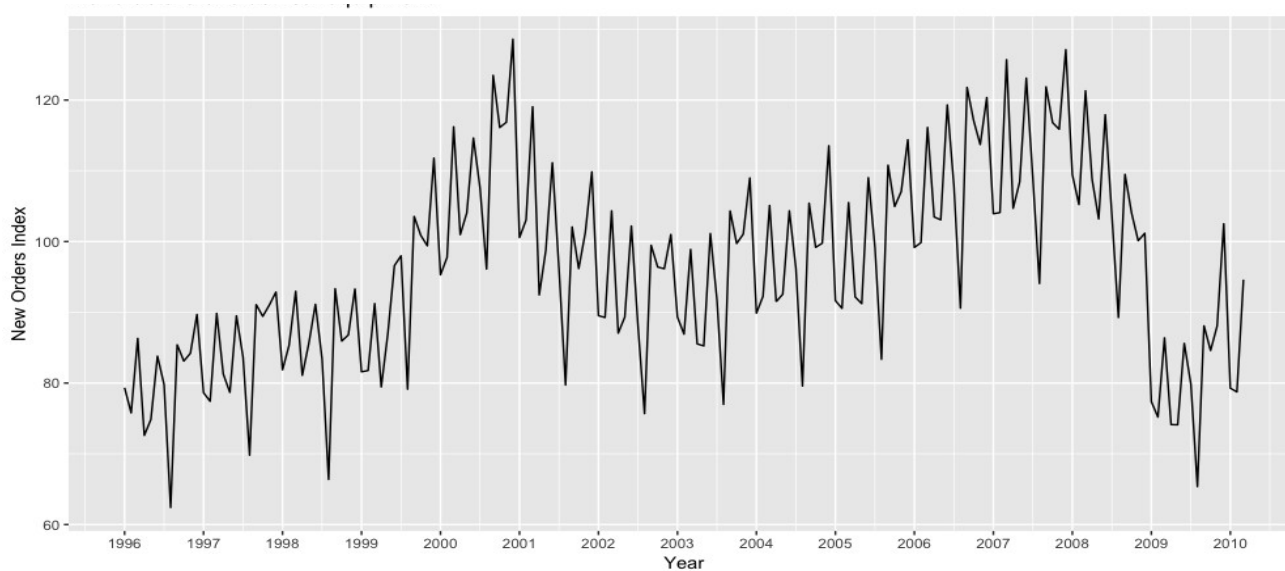
When to use LSTM or GRU ? - there is not a consensus.

LSTM appeared first (1997), GRU (2014) is a simplification of LSTM.
LSTM is more powerful (3 gates in LSTM, 1 gate in GRU).

* *ref. Hochreiter and Schmidhuber 1997, "Long short-term Memory".*

3. Time Series Forecasting

Time Series Data

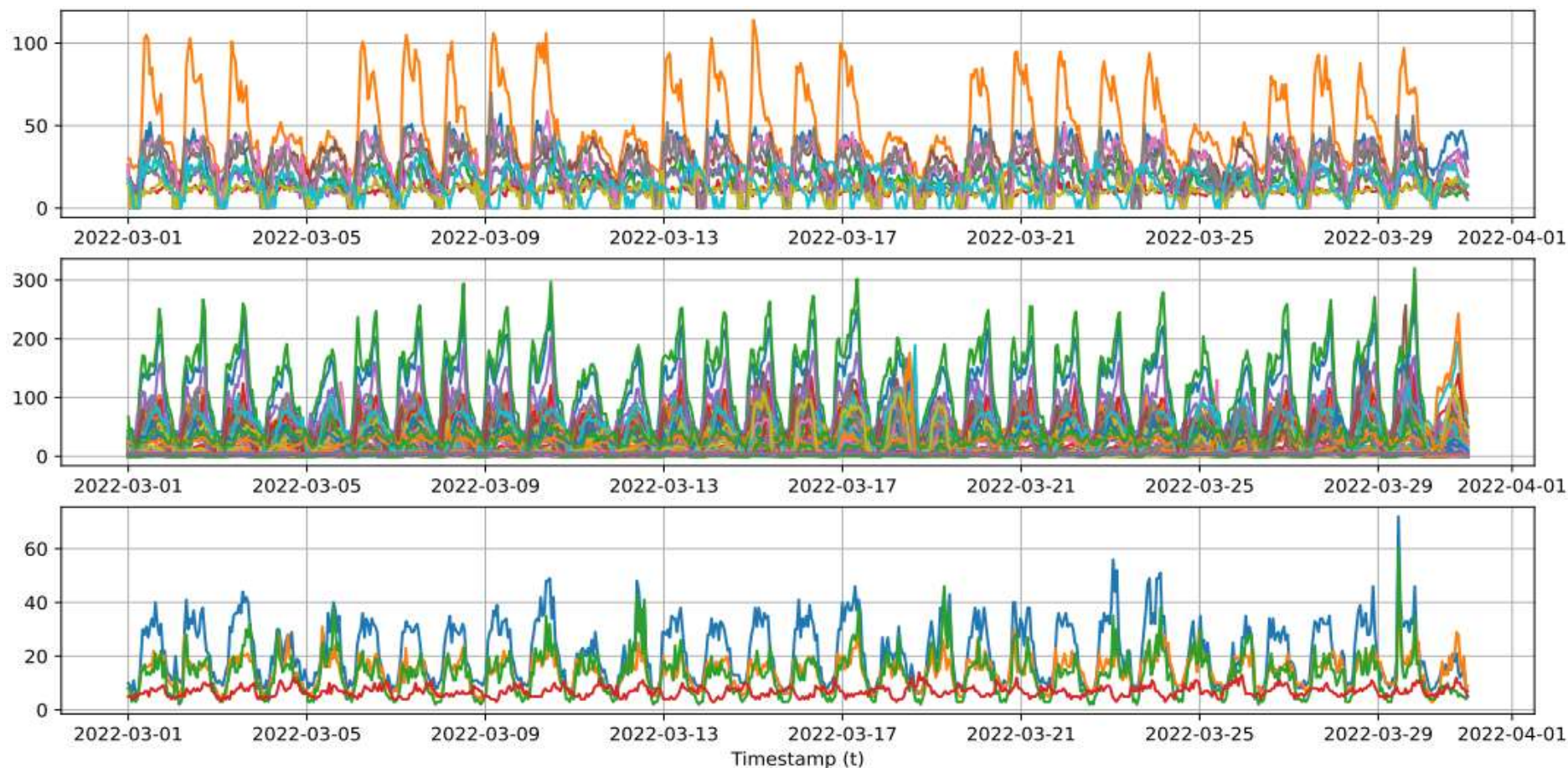


Time Series (TS) - collection of samples recorded at a sequence of time intervals

TS forecasting (prediction) => based on past samples, predict future trends, seasonality, anomalies, etc. Many applications:

- Key Performance Indicators (KPIs) in networks: e.g. traffic prediction
- Smart Homes – indoor temperature prediction
- Weather forecasting
- Financial data forecasting – e.g. bitcoin price
- Industrial sensor measurements / (wireless) sensor networks

Time Series Properties

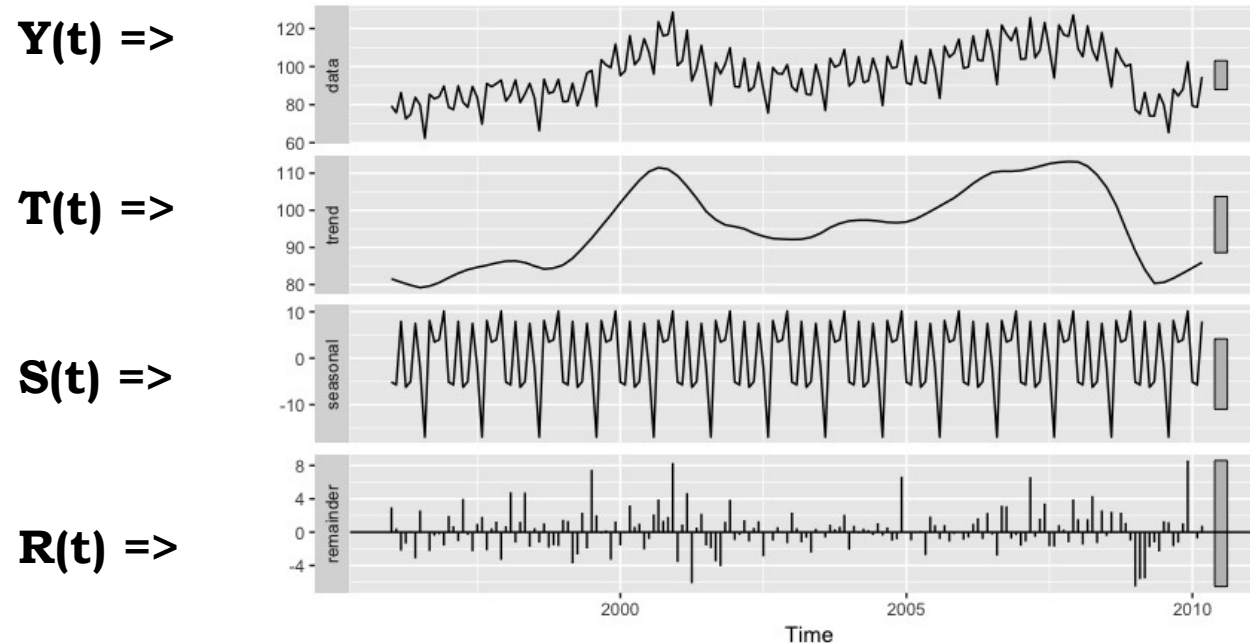


Stationarity - TS is stationary when the mean and variance remain constant over time.

Seasonality – TS variations at specific time-frames (i.e. people buy more Christmas trees during Christmas, people eat more ice-cream in summer).

Auto-correlation - correlation between the current value and the value from previous times (lags).

Time Series Decomposition



If TS $Y(t)$ shows some seasonality (e.g. daily, weekly, yearly) it can be decomposed into 3 components:

$$Y(t) = S(t) + T(t) + R(t)$$

$T(t)$ - trend, estimated through the mean of last n samples (rolling mean)

$S(t)$ - seasonal component

$R(t)$ - remaining component

Time Series Forecasting – linear models

1) Naïve model (persistence) : future value $(t+h)$ = last observed value (t)

$$\hat{Y}(t+h | t) = Y(t)$$

2) Exponential smoothing

Weighted past samples, weights decrease exponentially as go back in time.

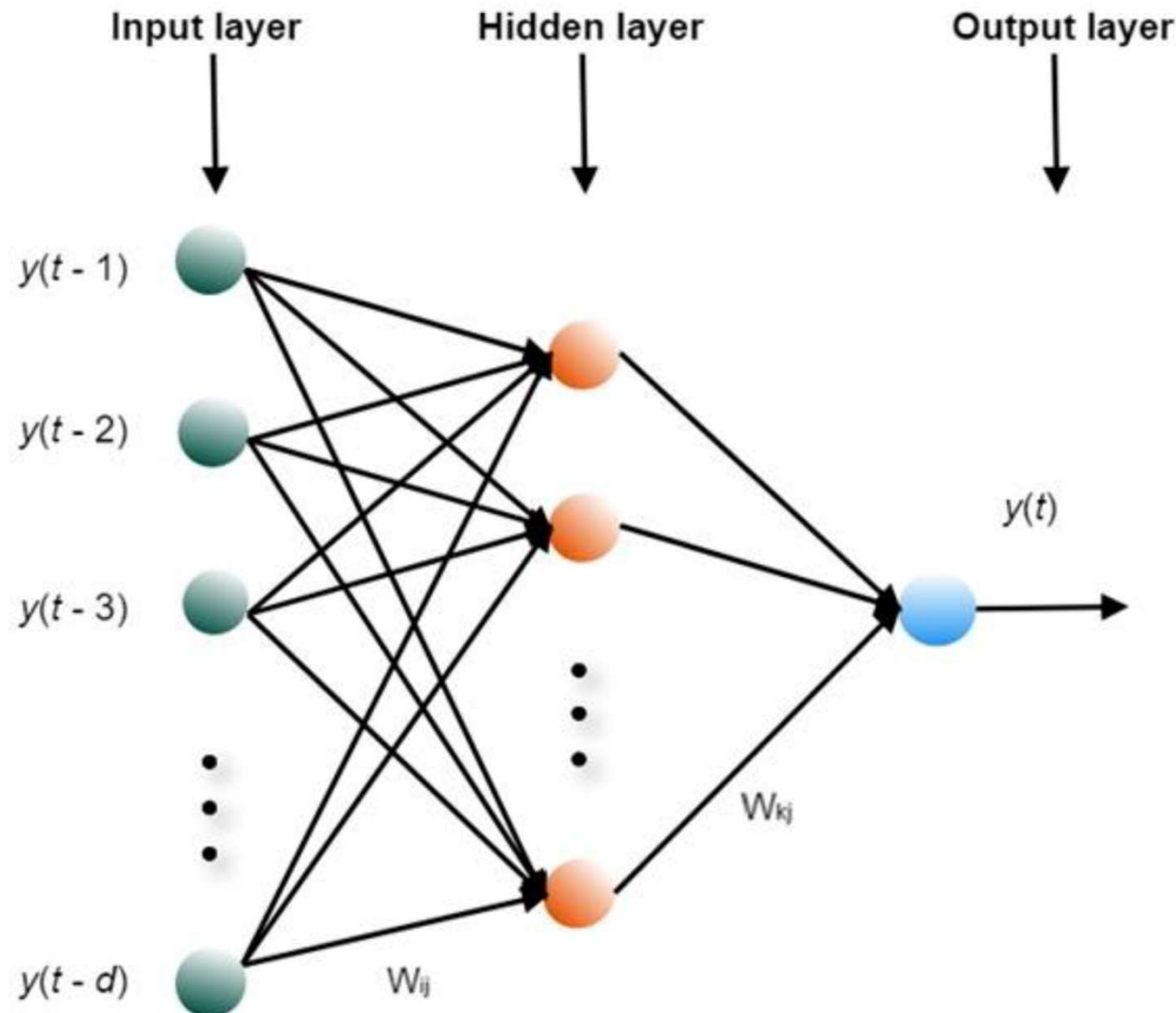
$$\hat{Y}(t+h | t) = \alpha Y(t) + \alpha(1-\alpha)Y(t-1) + \alpha(1-\alpha)^2Y(t-2) + \dots, 0 < \alpha < 1$$

3) ARIMA /SARIMA - Auto-Regressive (Integrated) Moving Average /Seasonal

$$\hat{Y}(t+h | t) = a_1Y(t) + a_2Y(t-1) + \dots a_pY(t-p) + b_1E(t) + b_2E(t-1) + \dots b_qE(t-q)$$

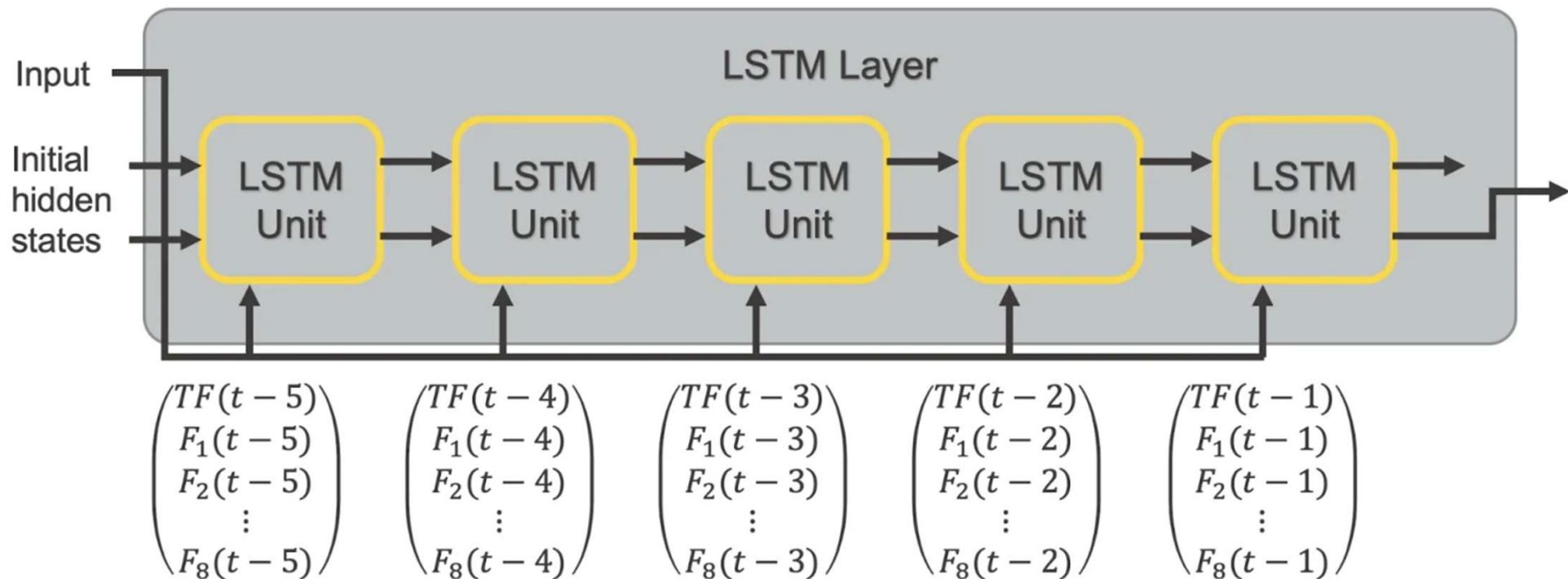
Time Series Forecasting – classical NN

Non-linear Auto Regressive Neural Network (NARN)
Classical fully connected shallow (3 layers) ANN
d- lag hyperparameter (time lag, model memory)



Time Series Forecasting – LSTM network

**Long Short Term Memory (LSTM) architecture
or GRU (Gated Recurrent Unit) architecture**



Multivariate prediction model

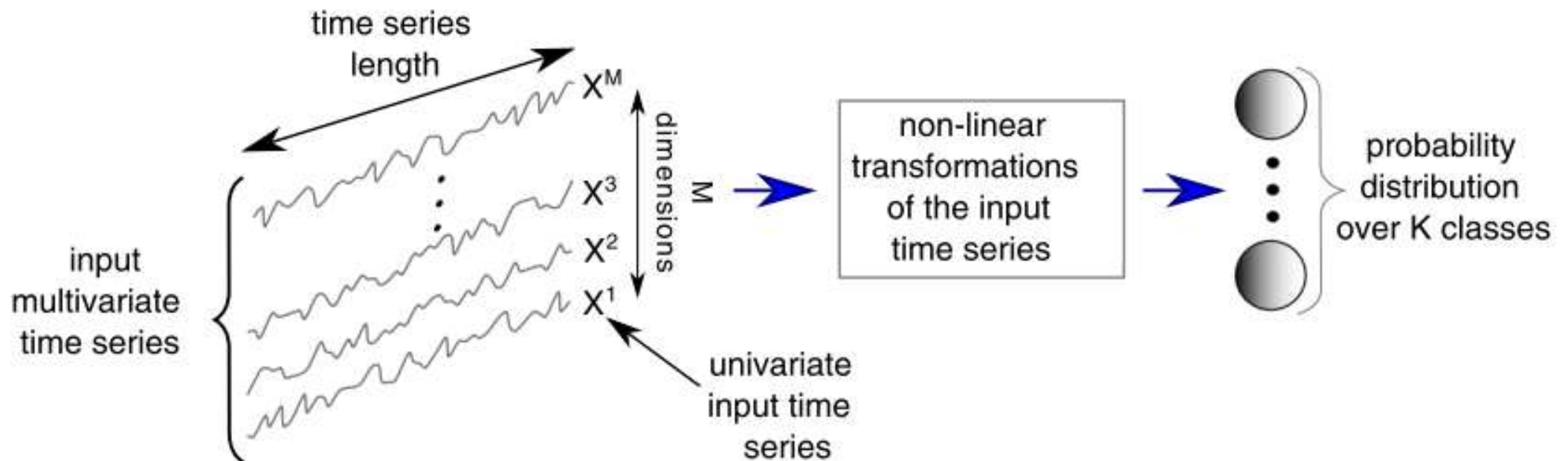
Predict $TF(t)$ – the output

TF, F_1, F_2, \dots, F_8 – predictors/features

ex. lag $d=5$

Time Series classification - CNN

Convolutional Neural Network (CNN)



4. Transformers

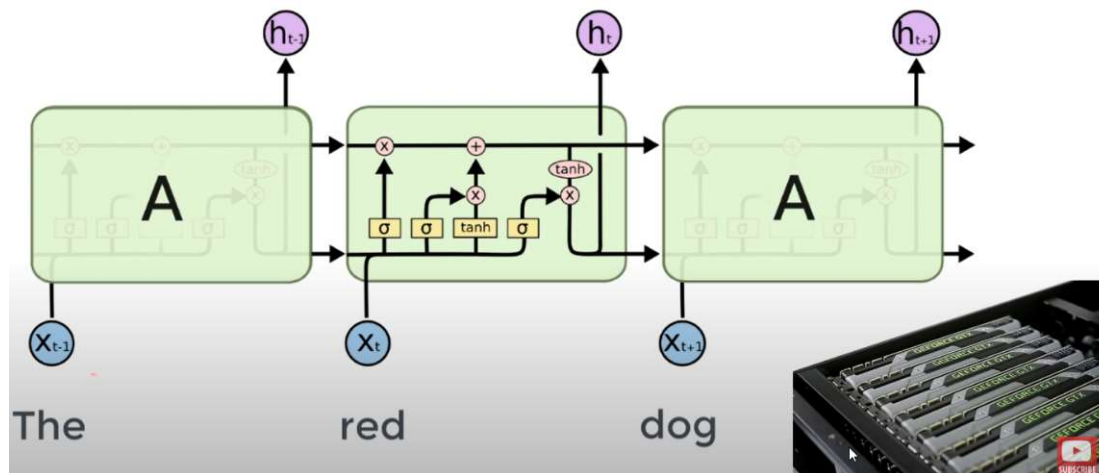
RNN – disadvantages

- RNN are slow to train
- Long sequences => lead to vanishing/exploding gradients
- LSTM/GRU – some improvements to long sequence training but even slower to train than RNN.

Ex. The most likely meaning of the word “bank” in the sentence “I arrived at the bank after crossing the...” requires knowing if the sentence ends in “... road.” or “... river.”

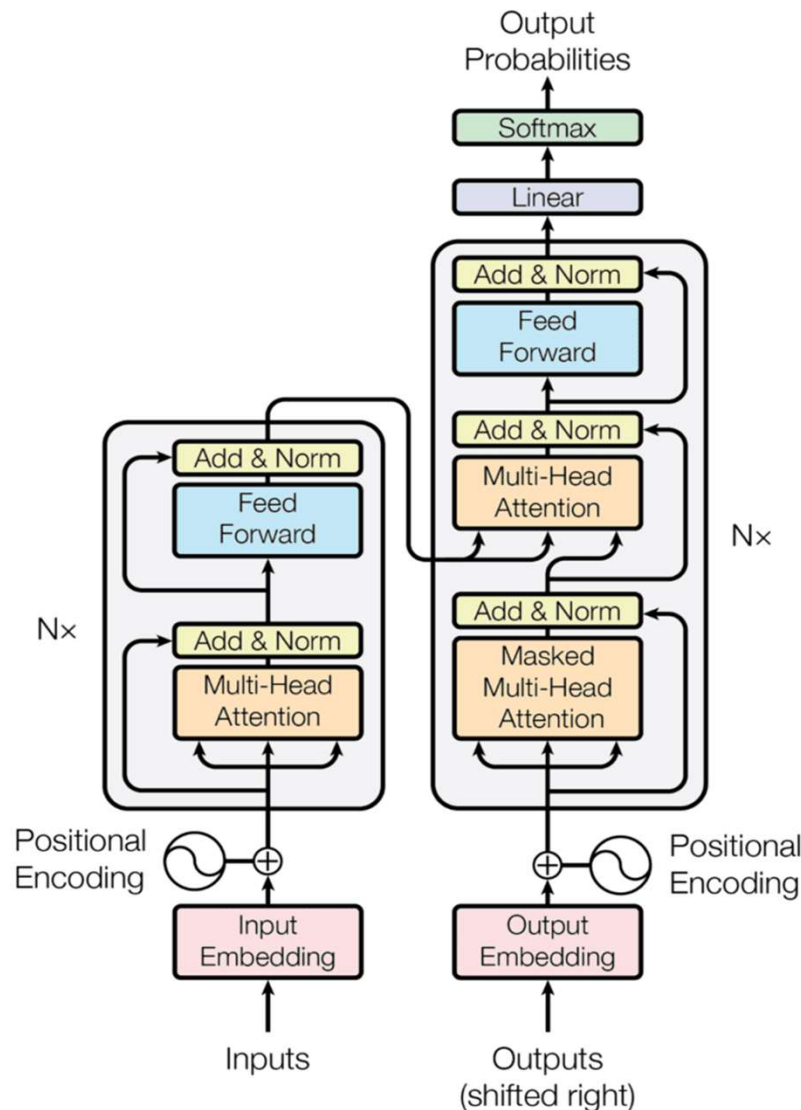
- Input data is passes sequentially (one word/token at a time).
- Need inputs from previous state to make operation on the current state.
RNNs do not make good use of GPUs designed for parallel computation.

How can parallelize sequential data ? => **Transformers (2017)**



Transformer architecture (Attention)

It follows an encoder-decoder structure but does not rely on recurrence and convolutions in order to generate an output.



Ashish Vaswani et al., Attention Is All You Need, 2017

Encoder (left half) maps an input sequence to a representation, which is then fed into a decoder.

Decoder (right half) receives the output of the encoder together with the decoder output at the previous time step to generate an output sequence.

NLP – One Hot Encoding

Word embedding is a way of representing words.

Vocabulary = {a, aaron, ...zulu, <UNK>} - e.g. 10000 words

One hot encoding vector. This representation does not relate the words. The inner product between one-hot vectors is 0. The distances between them are the same. It cannot generalize across words:

Example: “**Ana wants a glass of orange ...?.**”, let assume the model learned to predict the next word as **juice**.

A similar example “**Ana wants a glass of apple ...?.**”, the model won't easily predict juice if it was not trained on it.

The two examples are not related although orange and apple are similar.

Man	Woman	King	Queen	Apple	Orange
(5391)	(9853)	(4914)	(7157)	(456)	(6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

Word Embedding (featurized representation)

Instead of one hot vector, each word is represented with a number of features (semantic meaning of the words)– gender, royal, age, food,etc.
e.g. 300-dimensional vector.

It reduces the size of the input (ex. from 10000 to 300).

Example: Orange and apple now share many similar features which makes it easier for the model to generalize between them.

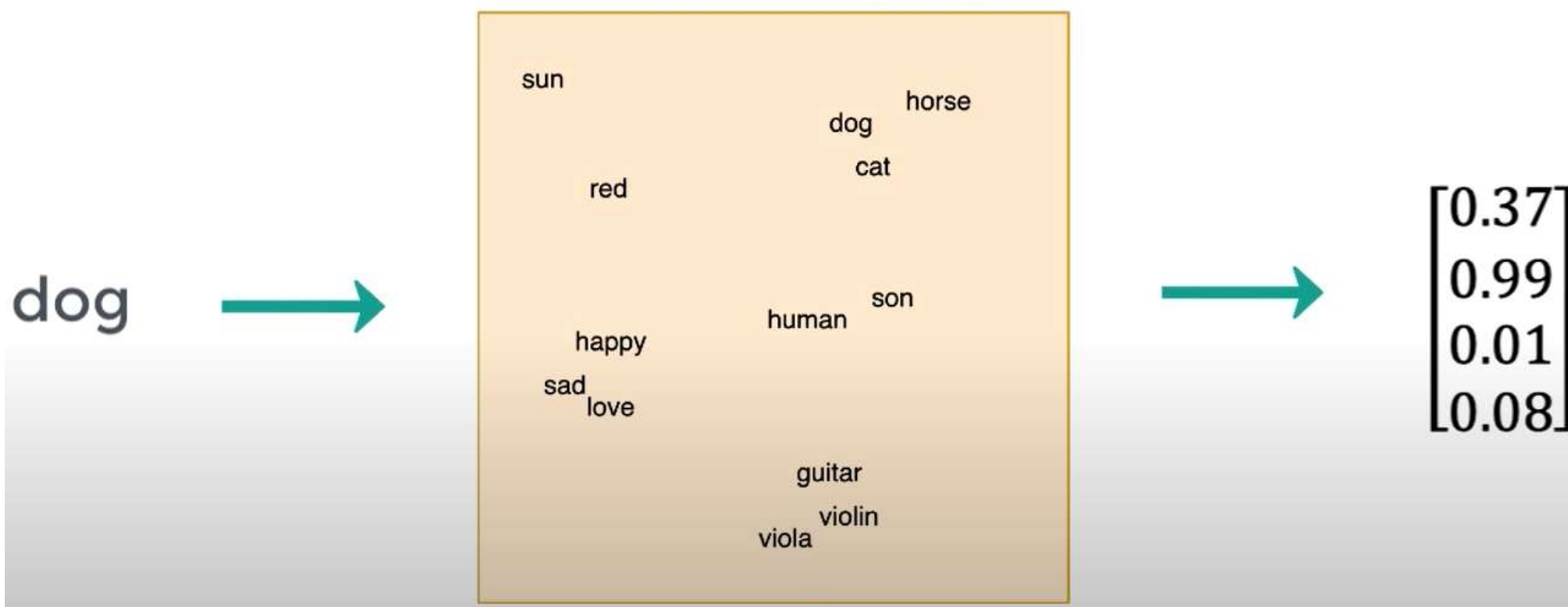
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

e Man *e* Woman *e* King *e* Queen

Visualizing word embeddings

Embedding space : map every word to a point in space where similar words in meaning are physically closer to each other.

t-distributed Stochastic neighbour embedding (t-SNE) reduces the features to 2D . t-SNE is a complex non-linear mapping. Related words are closer to each other.



Positional Encoding

The same word in different sentences may have different meaning =>

Positional Encoding (PE) gives context based on position of word in text.

His dog is cute => pos = 2;

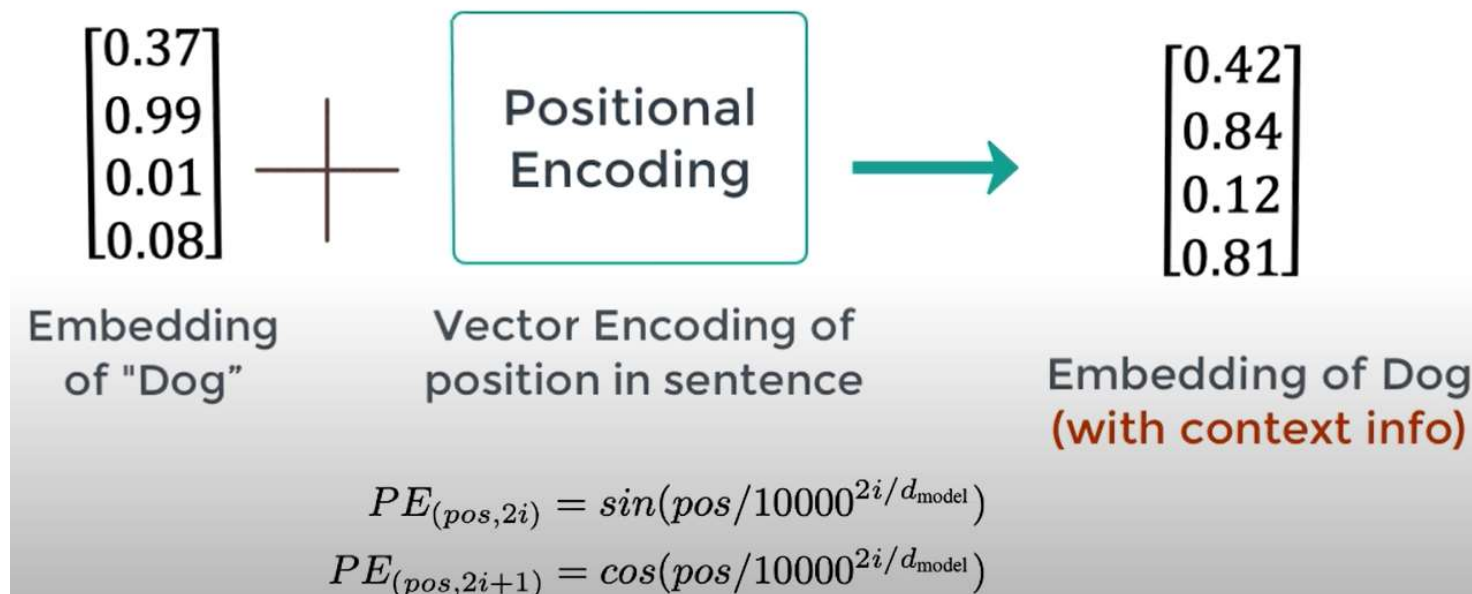
He looks like a dog => pos = 5

pos – numerical position of the word.

i – index in the embedding vector

d – dimension of the embedding vector (d=4)

Original paper uses sin & cos but it can be other function.



Attention

What part of the input should we focus ?

How relevant is one word to the other words in this sentence ?

Several attention vectors.

Encoder:

		Attention Vectors
The	→ The big red dog	$[0.71 \quad 0.04 \quad 0.07 \quad 0.18]^T$
big	→ The big red dog	$[0.01 \quad 0.84 \quad 0.02 \quad 0.13]^T$
red	→ The big red dog	$[0.09 \quad 0.05 \quad 0.62 \quad 0.24]^T$
dog	→ The big red dog	$[0.03 \quad 0.03 \quad 0.03 \quad 0.91]^T$

Transformers way to compute Attention

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} v^{<i>}$$

Query (Q)	Key (K)	Value (V)
$q^{<1>}$	$k^{<1>}$	$v^{<1>}$
$q^{<2>}$	$k^{<2>}$	$v^{<2>}$
$q^{<3>}$	$k^{<3>}$	$v^{<3>}$
$q^{<4>}$	$k^{<4>}$	$v^{<4>}$

Every word, is associated with 3 values (q- the query; k-key; v- value pairs).

$X^{<i>}$ - word embedding of i word =>

$$q^{<i>} = W^Q X^{<i>}, \quad k^{<i>} = W^K X^{<i>}, \quad v^{<i>} = W^V X^{<i>};$$

W^Q , W^K , W^V – parameters to be learned.

Intuition: each Attention will be an answer to a different query, e.g. what happens (concert, Celine Dion) ? When (time, July 2024) ? Where (city, Paris) ?, etc.

The goal of Attention is to compute more useful/rich representation (features) $A^{<i>}$ for each word.

Energy/storage/computational issues

Modern AI architectures face those problems : energy consumption, storage space, computational recourses.

These issues are open research problems with great interest and dynamics.

Transformer (65 million parameters): needs 12h for training, 1 server with 8 Nvidia Tesla P1001. Consume 27kW/hour, produce 1 CO2 lbs (pounds).

BERT (language models based on Transformers): needs 79h for training, with 64 Nvidia Tesla V100. Consume 1.507 kWh, produce 1.438 CO2 lbs.

DALL-E (12-billion parameters): needs 40-45 days for training on TPUs (Tensor Processing Units) v3-256. Needs about 24 GB of memory when stored in 16-bit precision, which exceeds the memory of a 16 GB NVIDIA V100 GPU.

ChatGPT (175 billions parameters) created to produce text similar to a human, needs 800 GB memory.