

Departamento de Eletrónica, Telecomunicações e
Informática

SEQUENCE MODELS 🗨️
&
TIME SERIES FORECASTING

Author: Petia Georgieva
Edited by: Susana Brás (Susana.bras@ua.pt)

Outline

1. Sequence models
2. Time Series Forecasting
3. Recurrent Neural Networks (RNN)
4. Backpropagation through time
5. Long-Short Term Memory (LSTM)
6. Gated Recurrent Unit

Sequence models

Sequence Models

- Sequence models are machine learning models that process data organized as sequences: text, audio, video, or time series data.
- They are designed to capture the dependencies between elements within a sequence.
- The key point for sequence models is that the data we are processing are no longer independently and identically distributed (i.i.d.) samples; the data carry some dependency due to their sequential order.
- **Sequential Data:** the order of elements matters, and the past influences the future.

Examples of sequence data

x (input)

y (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

Name 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 Model Notation

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

Given an input sequence of words (X), the sequence model has to automatically tell where are the proper names in this sentence.

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

$$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad \dots \quad x^{<9>}$$

For this example the sequence length is 9 words (features) => $\mathbf{T_x=9}$

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

$$\mathbf{y} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{y}^{<1>} \quad \mathbf{y}^{<2>} \quad \mathbf{y}^{<3>} \quad \dots \quad \mathbf{y}^{<t>} \quad \mathbf{y}^{<Ty>}$$

Ty=9

In this problem every input $\mathbf{x}^{(i)}$ has an output $y^{(i)}$ \Rightarrow **length $\mathbf{T}\mathbf{y}=\mathbf{T}\mathbf{x}$**

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

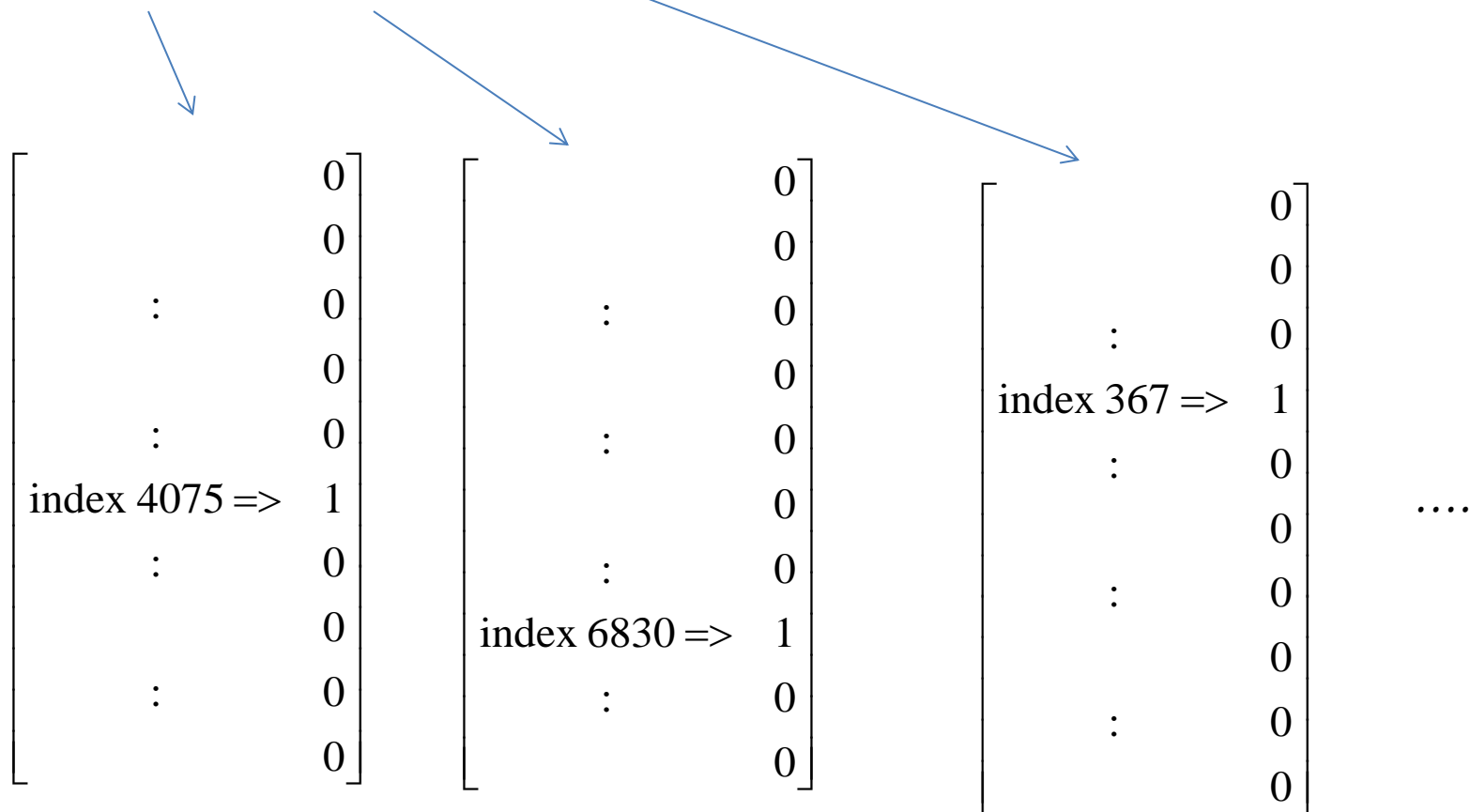
$$\begin{bmatrix} a \Rightarrow & 1 \\ aaron \Rightarrow & 2 \\ : & \\ and \Rightarrow & 367 \\ : & \\ Harry \Rightarrow & 4075 \\ : & \\ Potter \Rightarrow & 6830 \\ : & \\ zulu \Rightarrow & 10000 \end{bmatrix}$$

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

NLP – one hot encoding

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

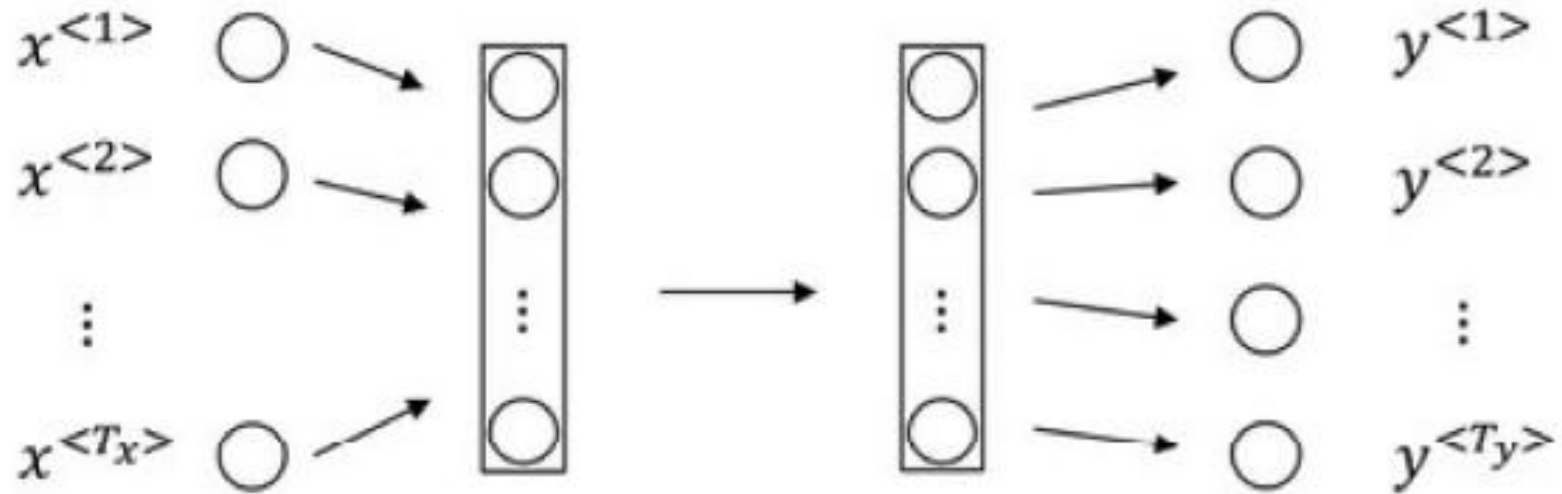
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$... $x^{<9>}$



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

The problem is formulated as supervised learning with labelled data (x,y).

Why not a standard neural network? 🗨️



Problems:

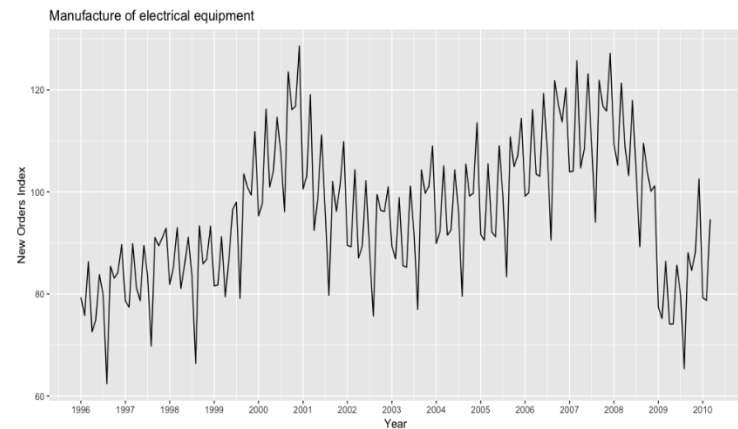
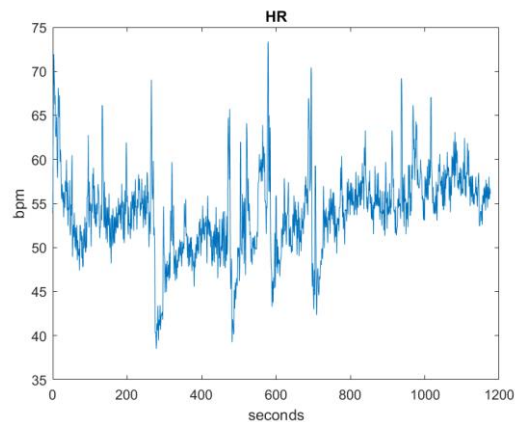
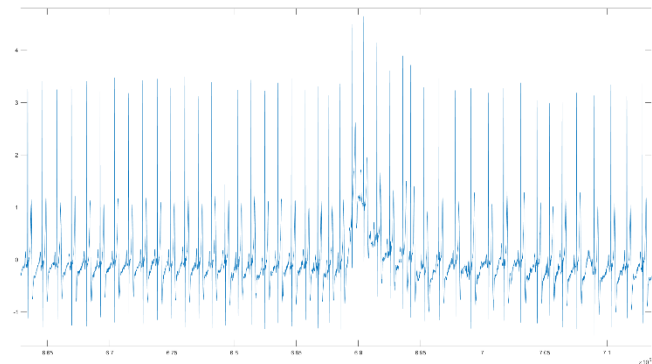
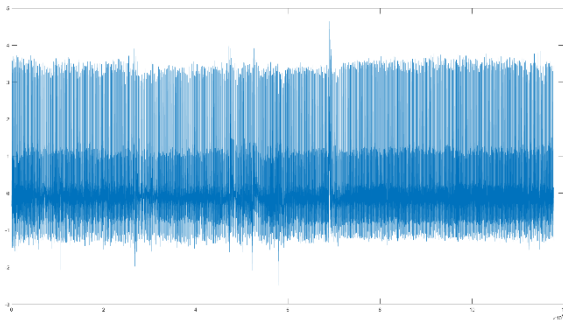
- 1) Inputs and outputs can have different lengths in different examples.
- 2) Doesn't share features learned across different positions of text.
- 3) High input dimension (e.g. $T_x \times$ dimension of dictionary) \Rightarrow too many trainable parameters.

Time Series Forecasting



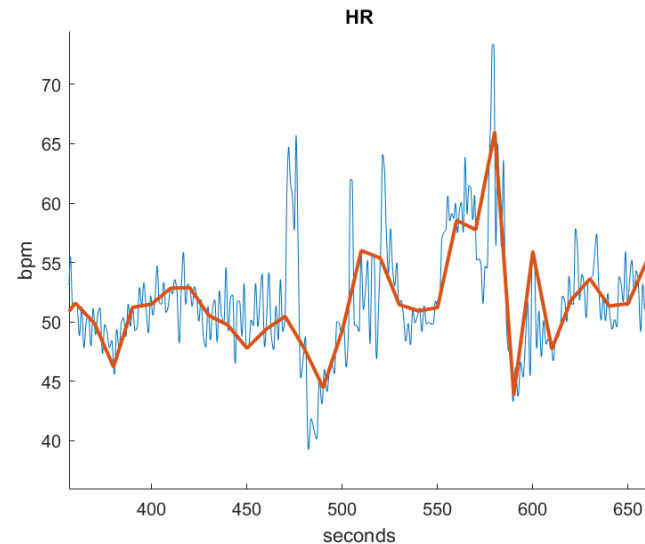
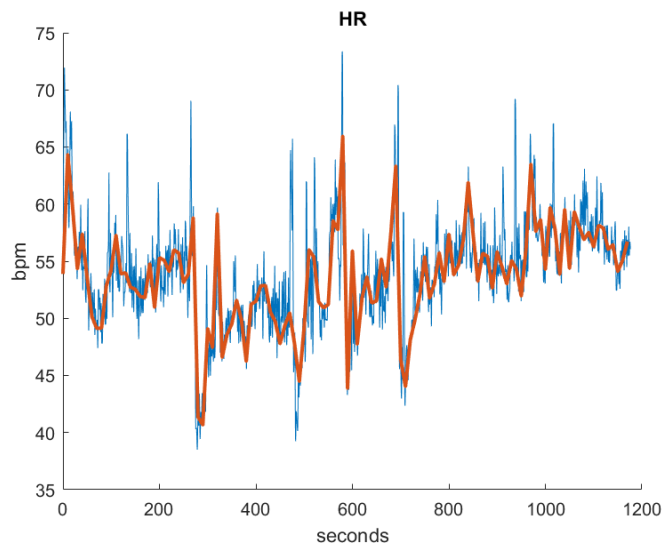
Time Series

- A sequence of random variables defined at fixed sampling intervals is sometimes referred as a discrete-time stochastic process.
- This is also known as a time series.
- A time series represents a variable that is indexed by time.

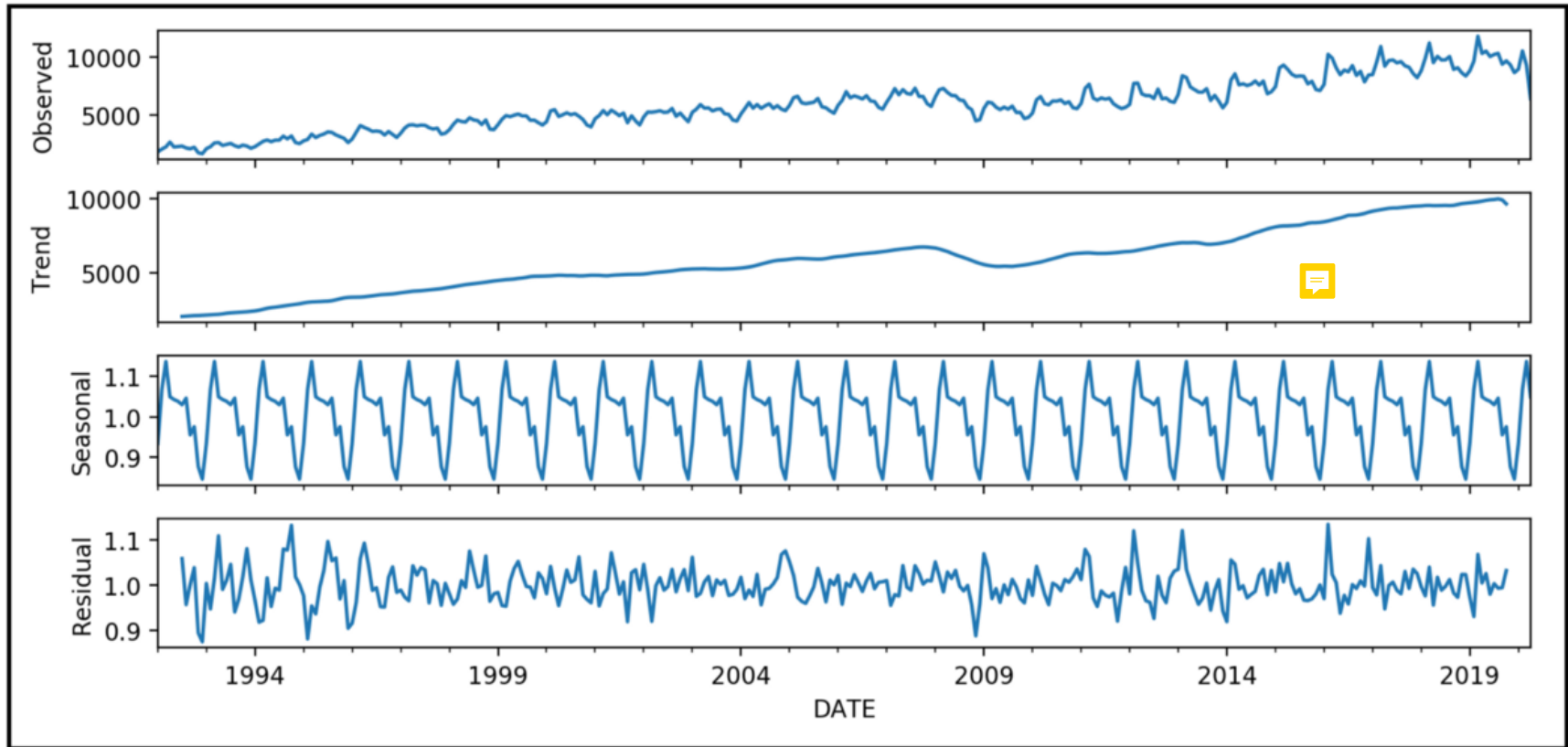


Sampling in Time Series

- The variable can be measured over time or at a fixed interval (sampling interval).
- If the data represented in the time series is sampled, it is important that the interval is sufficiently small to allow for a proper approximation of the continuous signal when the data is interpolated.

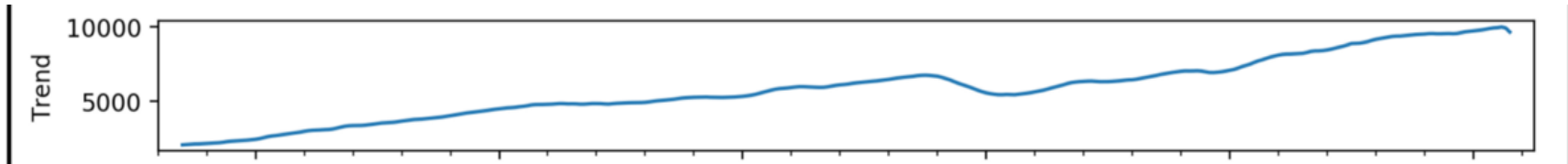


Time Series Decomposition



By breaking down the data into its constituent parts, such as trend, seasonality, and residual, analysts can gain valuable insights into the patterns and behavior of the series

Time Series Decomposition



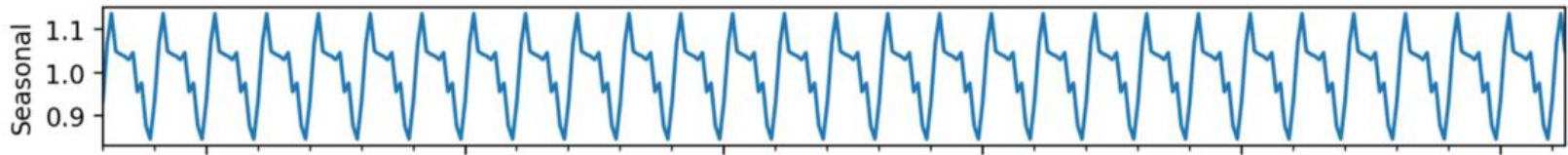
Trend - represents the long-term movement or direction of the data. It captures the persistent behavior of the series, which may be:

- Increase
- Decrease
- Stability.

Key characteristics:

- **Long-Term Behavior:** Trends are typically observed over a long period.
- **Smooth and Gradual Changes:** Trend components are often relatively smooth, exhibiting gradual changes over time.
- **Ignore Short-Term Fluctuations:** The trend component filters out short-term, noise-like fluctuations.

Time Series Decomposition



Seasonality - regular, recurring patterns evidenced at fixed time intervals.

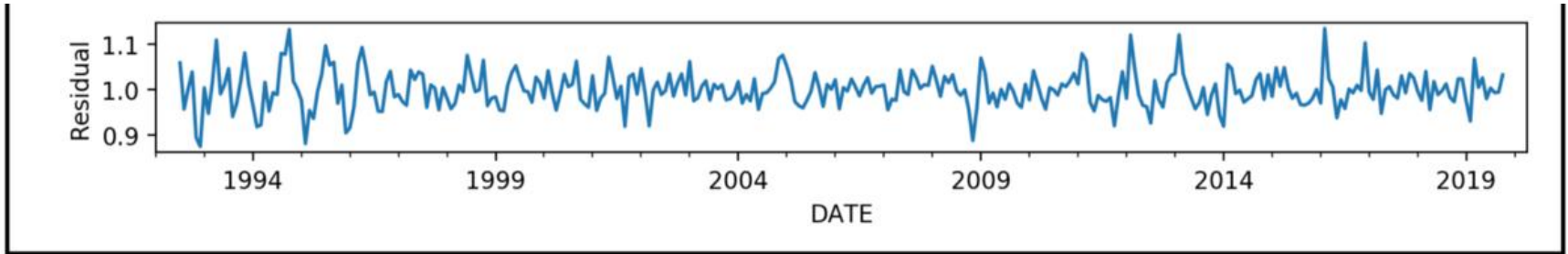
Patterns are often associated with calendar time:

- Daily
- Weekly
- Monthly
- Yearly

Key characteristics of the seasonality component include:

- **Regular Periodicity:** Seasonal patterns occur at consistent time intervals.
- **Predictable Patterns:** Seasonality can be anticipated and may exhibit similar patterns in different periods, making it valuable for forecasting.

Time Series Decomposition



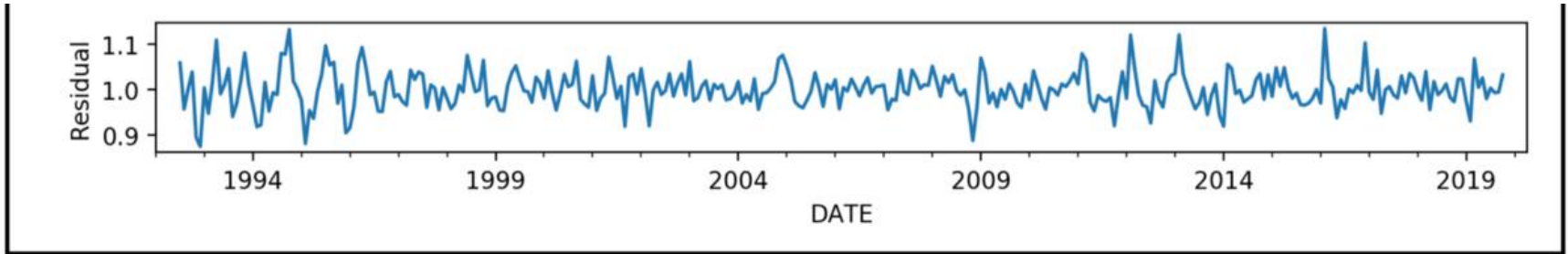
Residual component - **the random fluctuations that cannot be explained by the trend or seasonality.**

Residuals are the **unexplained** variability within the data, often reflecting the influence of unpredictable or external factors.

Key characteristics of the residual component include:

- Random Variability:** Residuals are typically random and uncorrelated, lacking any systematic patterns.
- Variations Not Accounted For:** The residual component represents the differences between observed values and the values predicted by the trend and seasonality components.

Time Series Decomposition



Understanding the residual component is essential for assessing the goodness of fit of a time series model. If the residuals exhibit a pattern or structure, the model might not adequately capture all the underlying patterns in the data.

Time Series Decomposition

$$x_t = f(T_t, S_t, E_t)$$

Where:

T_t time series trend;

S_t time series seasonality;

E_t time series residuals.

Additive Decomposition

$$x_t = T_t + S_t + E_t$$

This model is appropriate when the magnitude of the seasonal fluctuations and residuals do not depend on the level of the series.

Multiplicative Decomposition

$$x_t = T_t \cdot S_t \cdot E_t$$

This model is appropriate when the magnitude of the seasonal and residual variations increases with the level of the time series — i.e., the fluctuations grow proportionally as the values increase.

Time Series Forecasting

Time series forecasting is a technique that predicts future values in a sequence of data points measured over time.

It involves analyzing past data to identify **patterns**, **trends**, and **seasonality**, and then using **statistical** or **machine learning** models to make predictions about future events.

Data Dependence:

Time series data is characterized by its sequential nature, where each observation is influenced by previous observations.

Forecasting allows the:

- Understanding of future trends
- Making informed decisions
- Optimizing processes

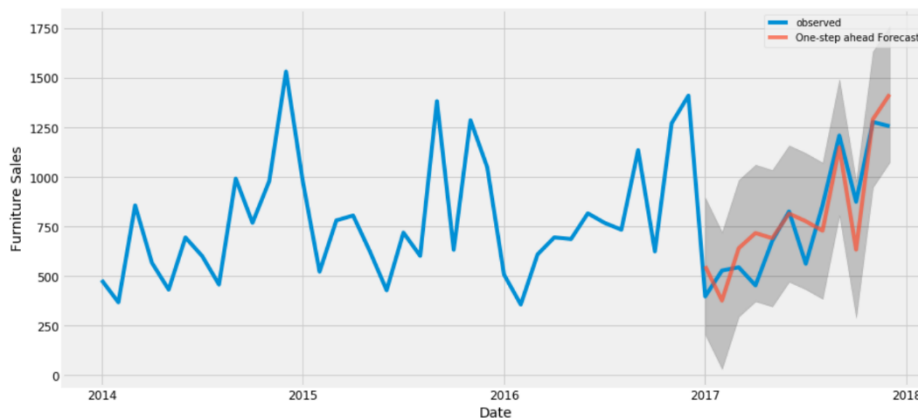


Time Series Forecasting

Stationarity refers to the condition where the statistical properties of a time series, such as its mean, variance, and autocorrelation, remain consistent over time.

Stationarity is significant because of the following:

Simplified Analysis
Reliable Forecasts
Statistical Assumptions
Trend and Seasonality Analysis



Time Series Forecasting

Time series forecasting utilizes statistical or machine learning models:

- **Classical/Statistical Models:** Moving averages, exponential smoothing, ARIMA, SARIMA, etc.
- **Machine Learning:** Linear regression, XGBoost, Random Forest, etc.
- **Deep Learning:** RNN, LSTM.

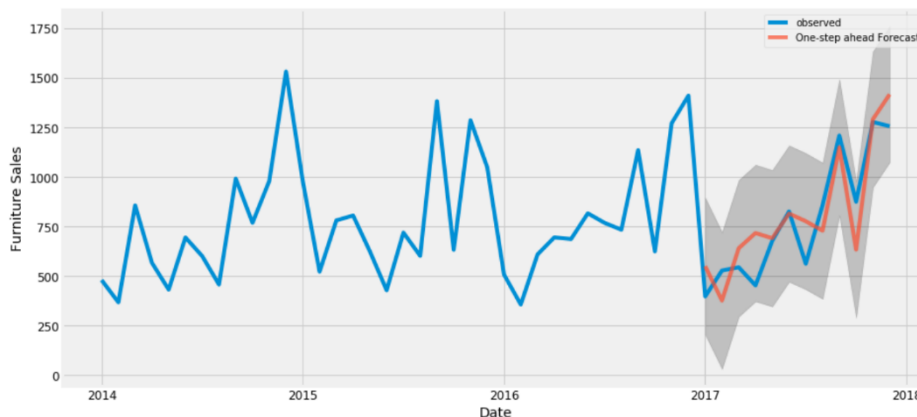
Key Principles:

Data Collection: Collecting reliable and relevant historical data.

Data Analysis: Analyzing trends, patterns, and seasonality in the data.

Model Selection: Choosing the appropriate model based on the characteristics of the data and the forecasting goal.

Evaluation: Evaluating the performance of the model using appropriate metrics to ensure its reliability.



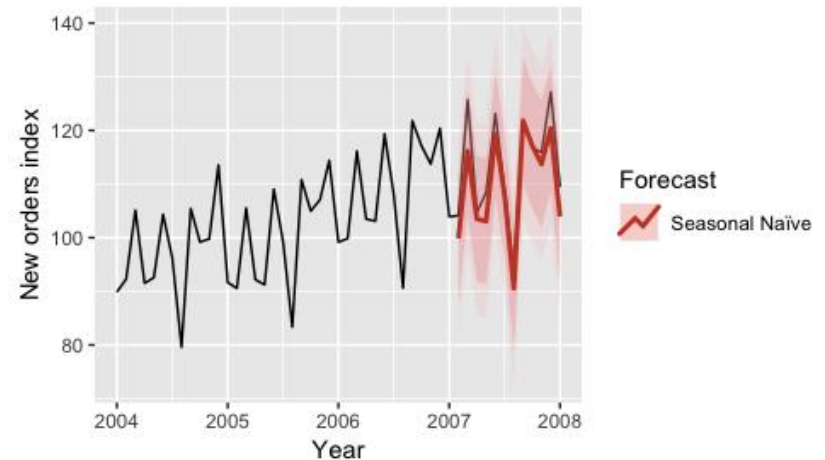
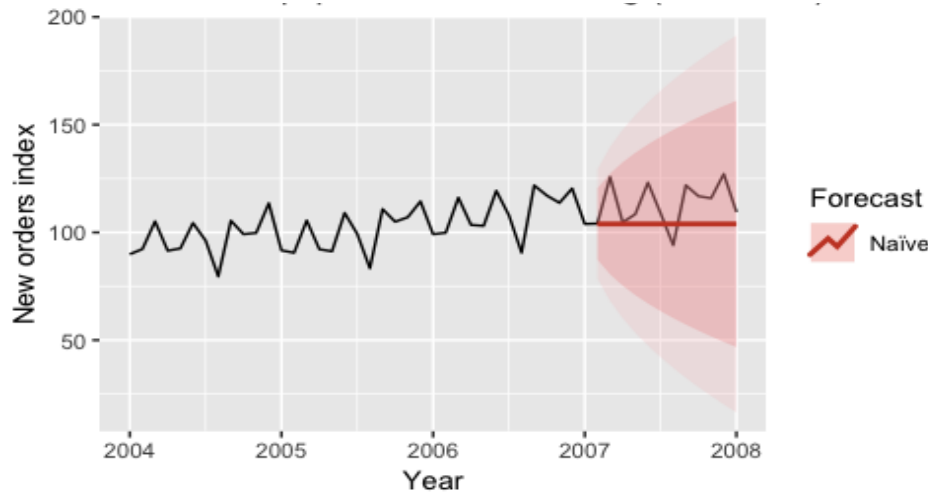
Time Series Forecasting – classical methods

1) **Naïve model** : the forecasts for every horizon (h) = last observed value:

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

2) **SNaïve (Seasonal Naïve) model**: TS has a seasonal component with period of seasonality T :

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

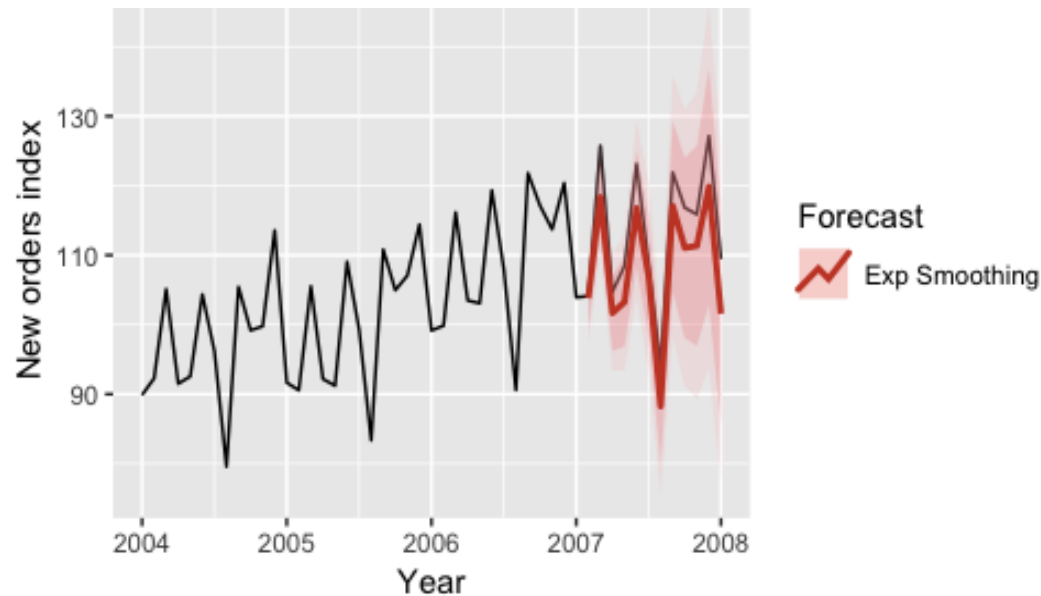


Time Series Forecasting – classical methods

3) Exponential smoothing

Weighted average of past observations, weights decrease exponentially as go back in time.

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



Time Series Forecasting – classical methods

4) ARIMA - Auto-Regressive (Integrated) Moving Average
ARMA (stationary signals), ARIMA (both stationary & nonstationary)

Auto-Regressive model - linear combination of past values of TS.

Moving-Average model – lin. combination of past forecasting errors.

Integrated (use differences) : $Y(t) = Y(t) - Y(t-1)$

ARIMA = AR + I + MA components

$$\hat{Y}(t+h | t) = a_1 Y(t) + a_2 Y(t-1) + \dots a_p Y(t-p) + b_1 E(t) + b_2 E(t-1) + \dots b_q E(t-q)$$

5) SARIMA model (Seasonal ARIMA) extends ARIMA by adding a linear combination of seasonal past values and forecast errors.

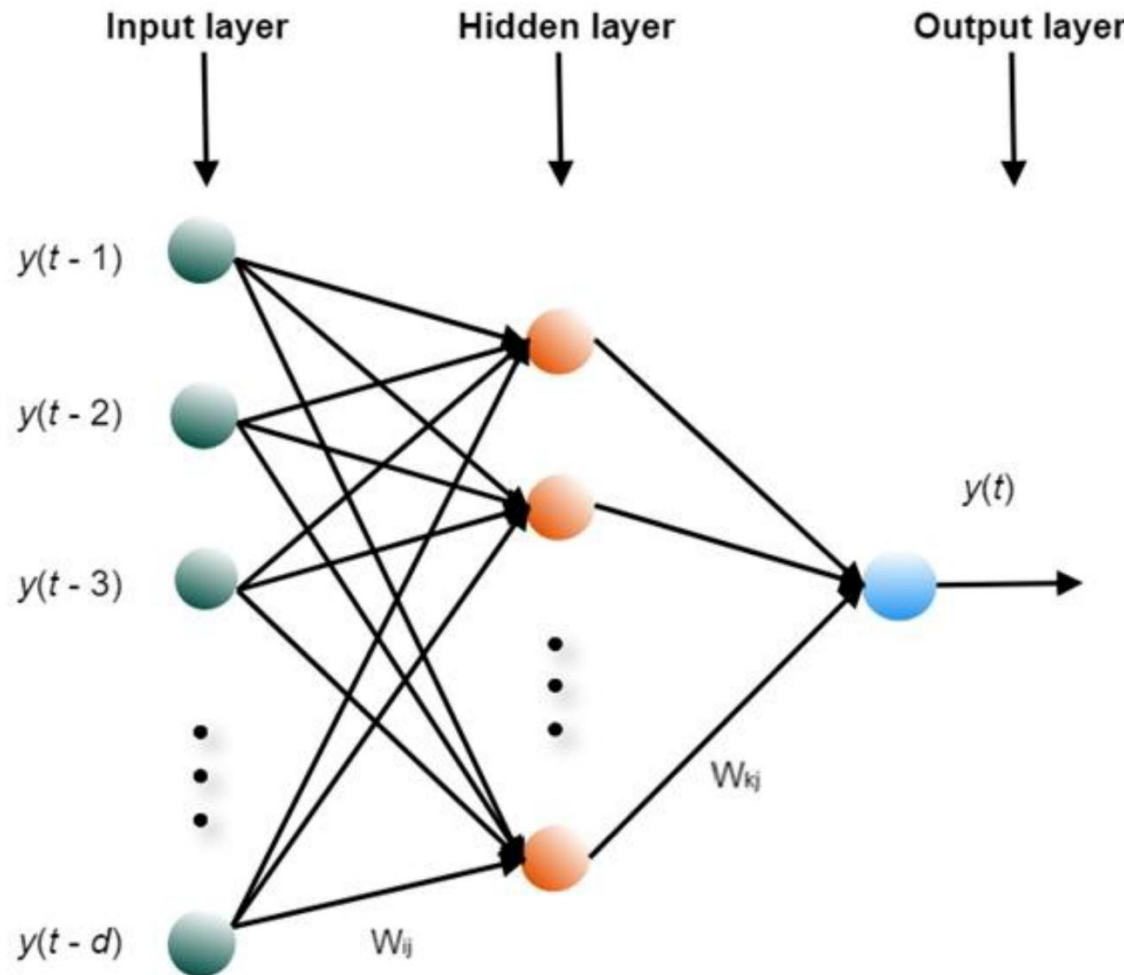
ARIMA (p,d,q)

p=> autoregressive lags; q=> moving average lags;

d=> difference in the order

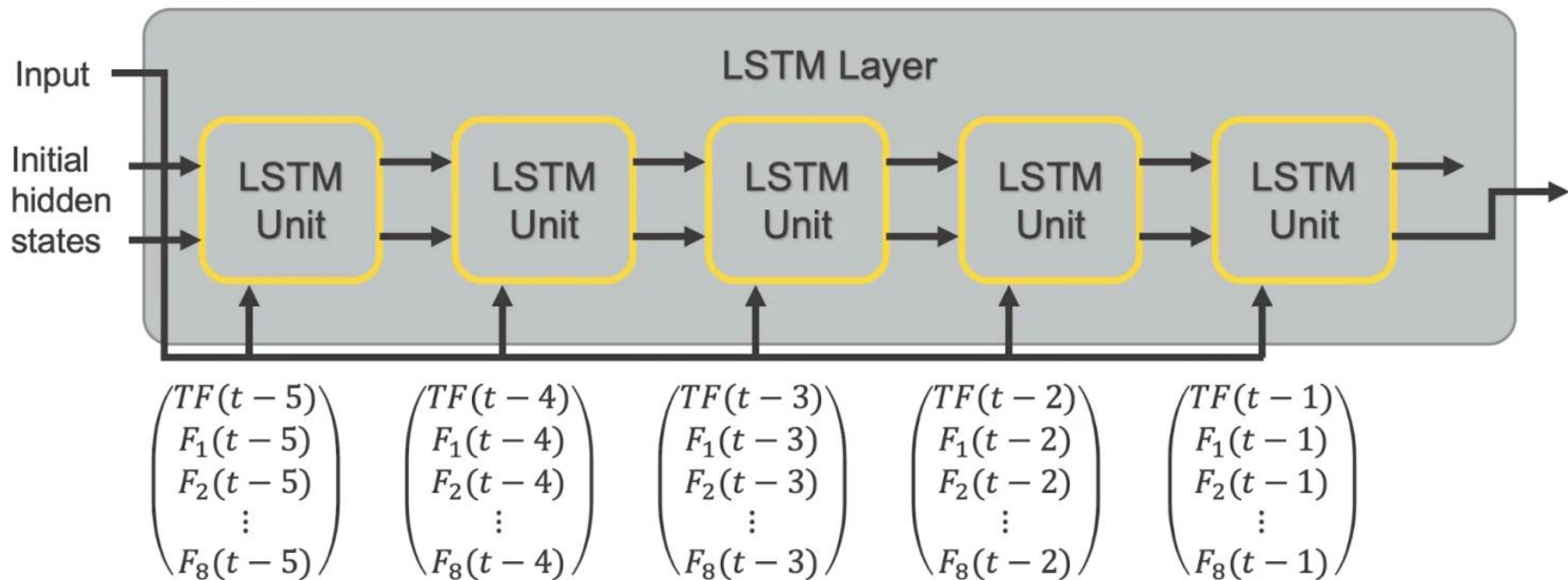
Time Series Forecasting – ML approach

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



Time Series Forecasting – DL approach

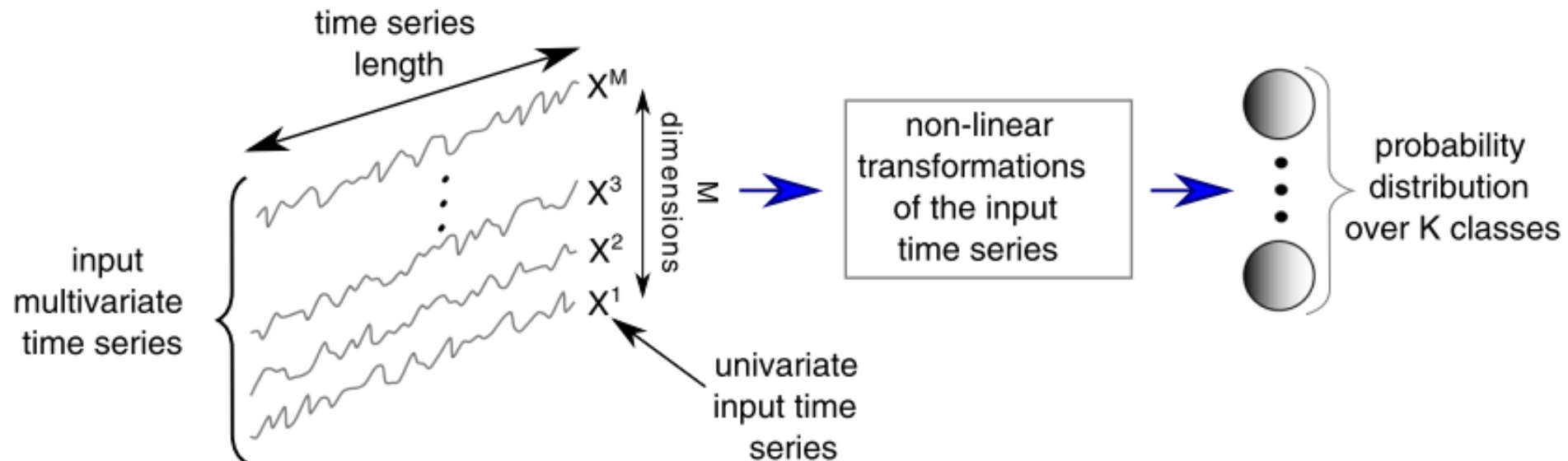
**Long Short Term Memory (LSTM) architecture
or GRU (Gated Recurrent Unit) architecture**



Multivariate prediction model
Predict $TF(t)$ – the output
 TF, F_1, F_2, \dots, F_8 – predictors/features
ex. lag $d=5$

Multivariate TS classification

Conv NN



Time Series Forecasting

Select the right model:

- **Consider:** data characteristics, the presence of trends or seasonality, and the forecasting requirements
- **Begin with simple models** like AR, MA, or ARMA and measure their performance. If the data shows clear patterns or dependencies, more complex models like ARIMA or SARIMA may be appropriate.
- **Consider Seasonality:** If the data shows seasonal patterns, models like SARIMA.
- **Evaluate Performance:** Use appropriate evaluation metrics and cross-validation techniques to compare the performance of different models.
- **Consider any domain-specific knowledge** or insights that can guide you in choosing a suitable model. Expert knowledge can help in identifying relevant variables, incorporating external factors, or applying specific modeling techniques.

Metrics:

- **Mean Absolute Error (MAE)**
- **Root Mean Squared Error (RMSE)**
- **Mean Absolute Percentage Error (MAPE)**

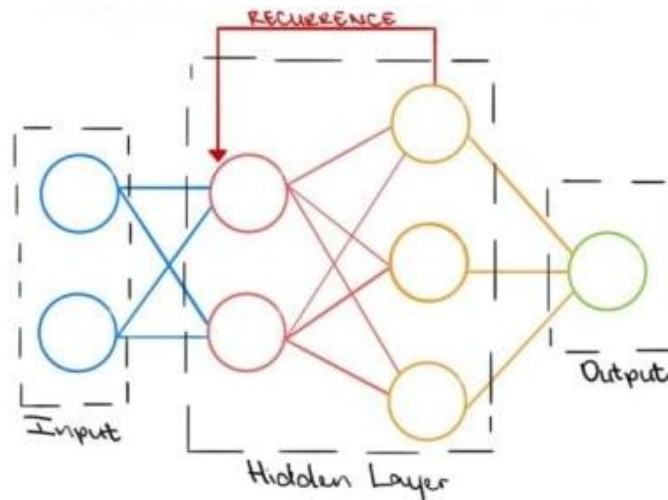


Recurrent Neural Networks (RNN)



RNN

- Recurrent Neural Networks (RNNs) are a type of neural network designed to process sequential data, such as text, speech, or time series, where the order of elements is important. They achieve this by maintaining a hidden state that remembers previous inputs, allowing them to learn long-term dependencies and patterns.
- While traditional deep learning networks assume that inputs and outputs are independent of each other, the output of RNN depends on the prior elements within the sequence.
- Another distinguishing characteristic of RNN is that they share parameters across each layer of the network. While feedforward networks have different weights across each node, RNN share the same weight parameter within each layer of the network.
- The core concept of RNNs revolves around the propagation of information from one time step to the next. At each time step, the network updates its hidden state using both the current input and the previous hidden state. This recurrent structure imbues RNNs with the ability to model dependencies and relationships within sequential data.



Recurrent Neural Networks (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 - $a^{<t>}$

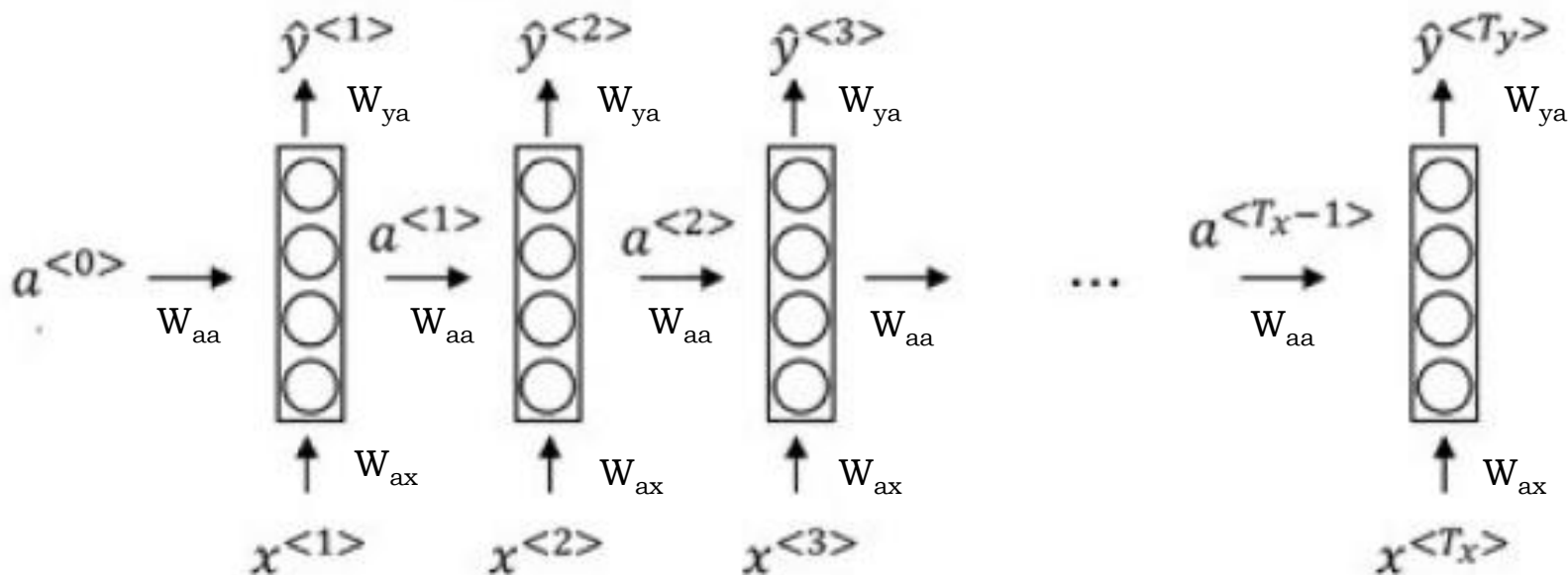
$a^{<0>}$ - initial state at time step 0, usually vector of zeros.

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

A weakness of this type of uni-directional RNN is that the prediction at a certain time uses inf. that is earlier in the sequence but not latter.

The Bidirectional RNN (BRNN) overcome this problem.

Flow chart (unrolled/unfolded RNN diagram representation) :

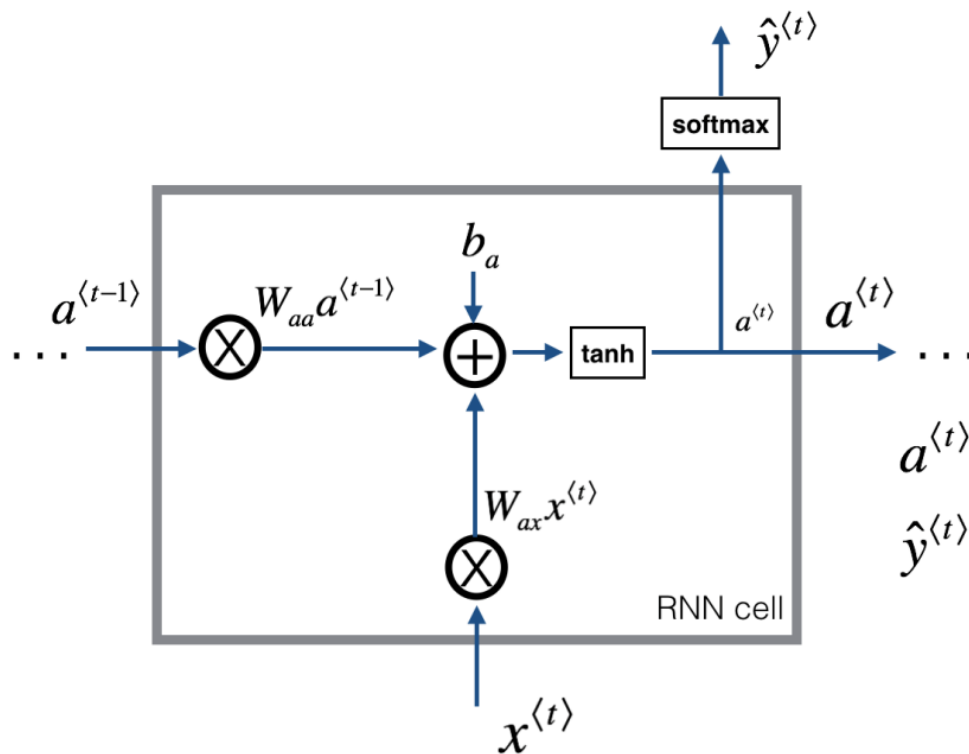


Basic RNN unit

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.



$$a^{(t)} = \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b_a)$$

$$\hat{y}^{(t)} = \text{softmax}(W_{ya}a^{(t)} + b_y)$$

Backpropagation through time



BPTT

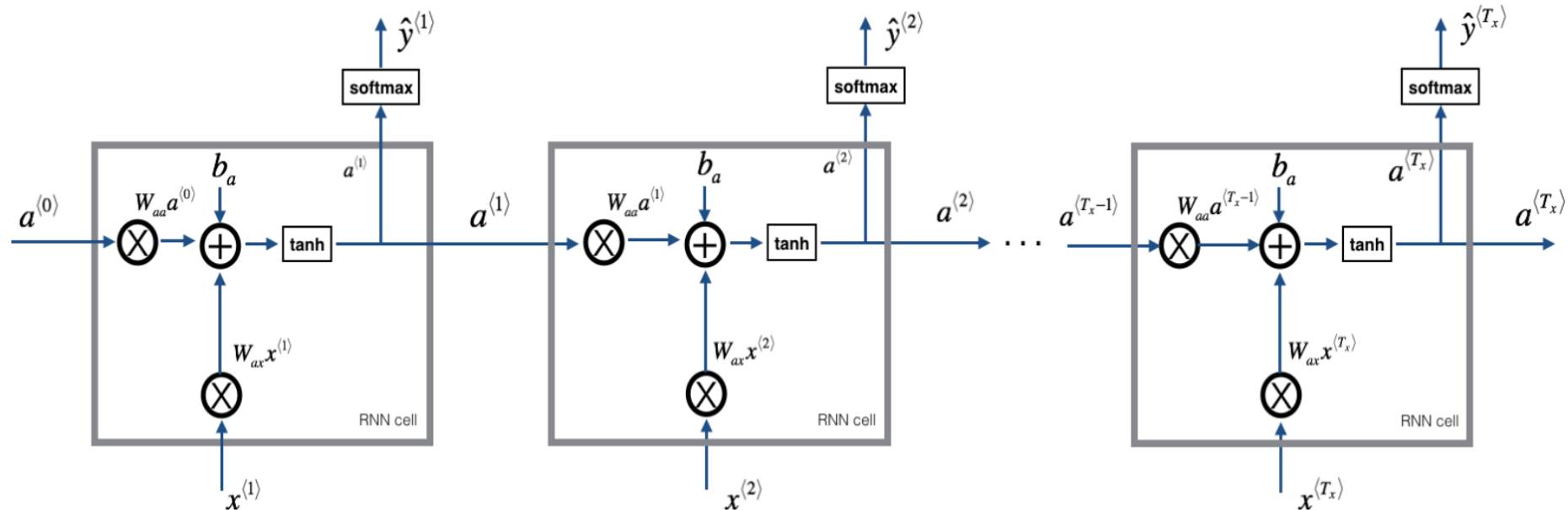
- Backpropagation is a process by which a model learns from its mistakes (by updating weights) and tries to minimize the loss.
- In RNN, the difference is that the backpropagation is THROUGH TIME, meaning the loss is backpropagated to each model and across the model because it is a SEQUENCE model.
- Training RNNs involves a special kind of backpropagation called Backpropagation Through Time (BPTT)
- Unlike traditional backpropagation, BPTT extends across time — it unfolds the entire sequence of data, applying backpropagation at each timestep.
- This method calculates gradients for each output, which are then used to adjust the weights and reduce the overall loss.
- However, BPTT can be complex and resource-intensive, and it's prone to issues such as vanishing and exploding gradients, which can interfere with the network's ability to learn from data over longer sequences.

RNN forward pass

RNN is a sequence of basic RNN cells.

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

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



Backpropagation Through Time (BPTT)

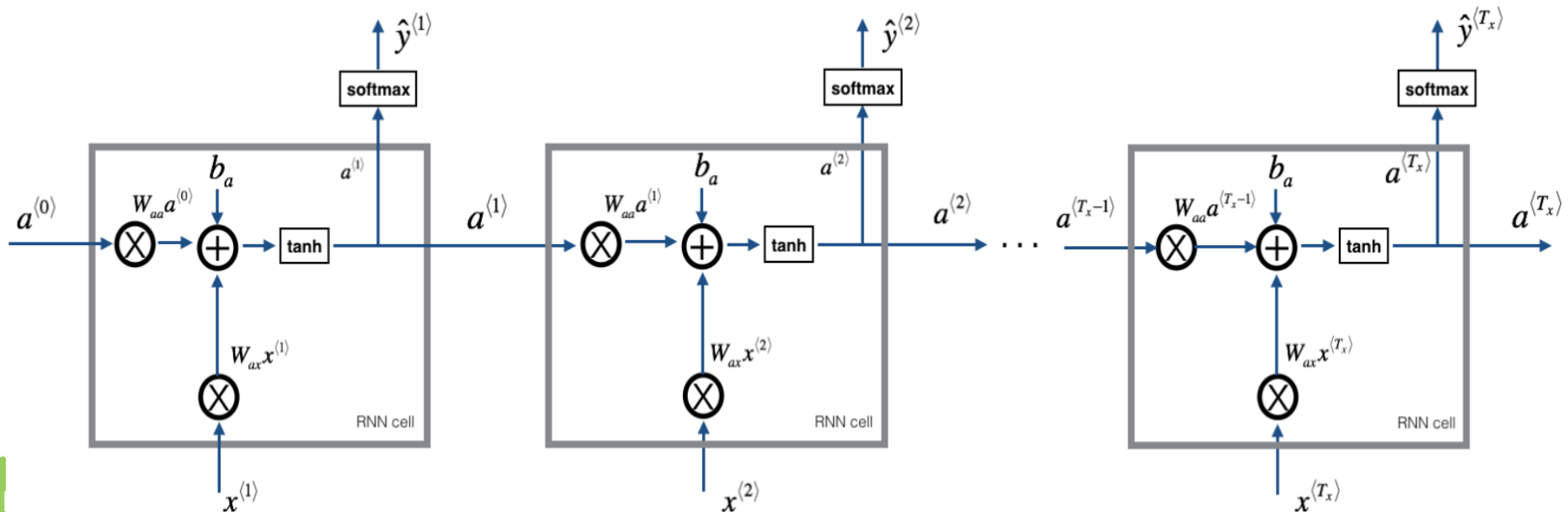
Forward propagation (for-prop): from time step 1 to time step T_x (from left to right) => compute RNN predictions.

Compute the cost (loss) function for each output $y^{(t)}$ ($J^{(t)}$)

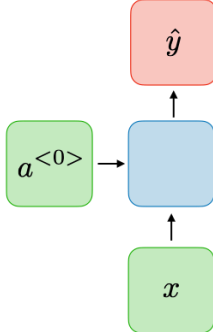
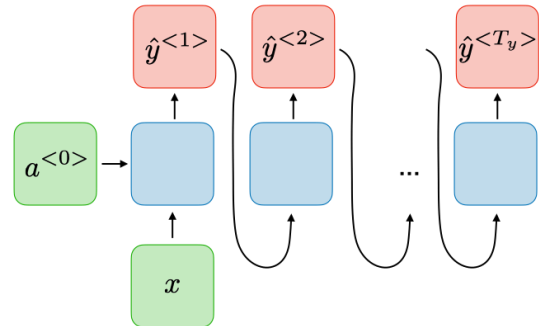
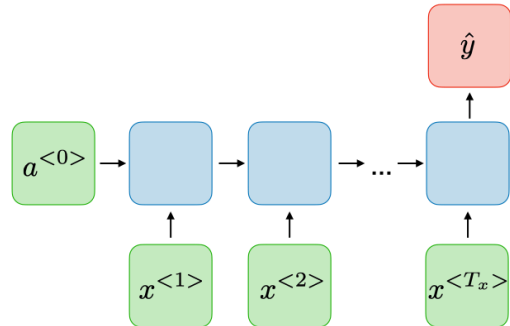
Compute the sum of all cost (loss) functions (J_{all})

Backward propagation (back-prop): to update RNN parameters compute the gradients of J_{all} , starting from the last time step $y^{<T_x>}$ and going back through the previous time steps down to the first time step $y^{<1>}$ (from right to left) => This is called *Backpropagation Through Time (BPTT)*.

The programming framework usually computes BPTT automatically.



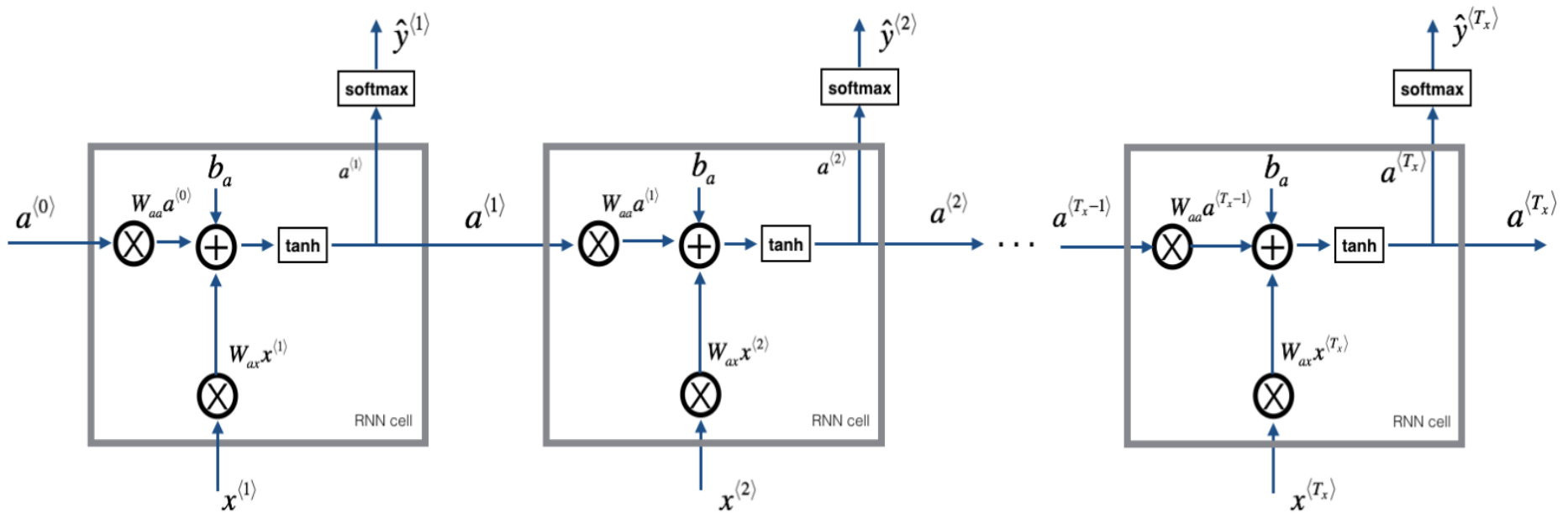
Different RNN

Type	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification

Different RNN

Type	Illustration	Example
Many-to-many $T_x = T_y$	<p>The diagram shows a sequence of input boxes labeled $x^{<1>}$, $x^{<2>}$, ..., $x^{<T_x>}$ feeding into a sequence of hidden state boxes. The first hidden state also receives an initial state $a^{<0>}$. Each hidden state produces an output box labeled $\hat{y}^{<1>}$, $\hat{y}^{<2>}$, ..., $\hat{y}^{<T_y>}$.</p>	Name entity recognition
Many-to-many $T_x \neq T_y$	<p>The diagram shows a sequence of input boxes labeled $x^{<1>}$, ..., $x^{<T_x>}$ feeding into a sequence of hidden state boxes. The first hidden state also receives an initial state $a^{<0>}$. The sequence of hidden states produces a sequence of output boxes labeled $\hat{y}^{<1>}$, ..., $\hat{y}^{<T_y>}$.</p>	Machine translation

RNN – vanishing gradients ???



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).

Why ? – because RNN's suffers from vanishing gradient (similar to Deep NN).

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) –
Solution for vanishing gradients.

RNN Challenges

Exploding gradients: When many gradient values are >1

Occurs when large error gradients accumulate and there are **very large updates** to neural network **weights**.

Gradients are used during training to update the network weights.

When the magnitudes of the gradients accumulate, an unstable network is likely to occur, which can cause poor prediction results.

Vanishing gradients: When many gradient values are <1

If the gradients are very small or zero, then little to no training occurs, leading to poor performance.

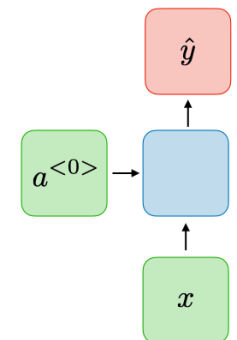
This also leads to capturing short-term dependencies instead of long-term dependencies.

RNN Challenges

Potential Solutions:

1. **Activation Functions:** Using ReLU prevents gradients from shrinking when $x > 0$
2. **Parameter Initialization:** Initialize weights to identity matrix and biases to zero — prevents weights from shrinking to zero
3. **Gated Cells:** In the operation box (blue), use some logic function (i.e. gated cells).

Type of gate	Role	Used in
Update gate	How much past should matter now?	GRU, LSTM
Relevance gate	Drop previous information?	GRU, LSTM
Forget gate	Erase a cell or not?	LSTM
Output gate	How much to reveal of a cell?	LSTM



Long-Short Term Memory (LSTM)



LSTM

LSTMs are a type of RNN architecture built to address the limitations of traditional RNNs, especially the vanishing gradient problem.

LSTMs have a dedicated memory cell that can store information for extended periods.

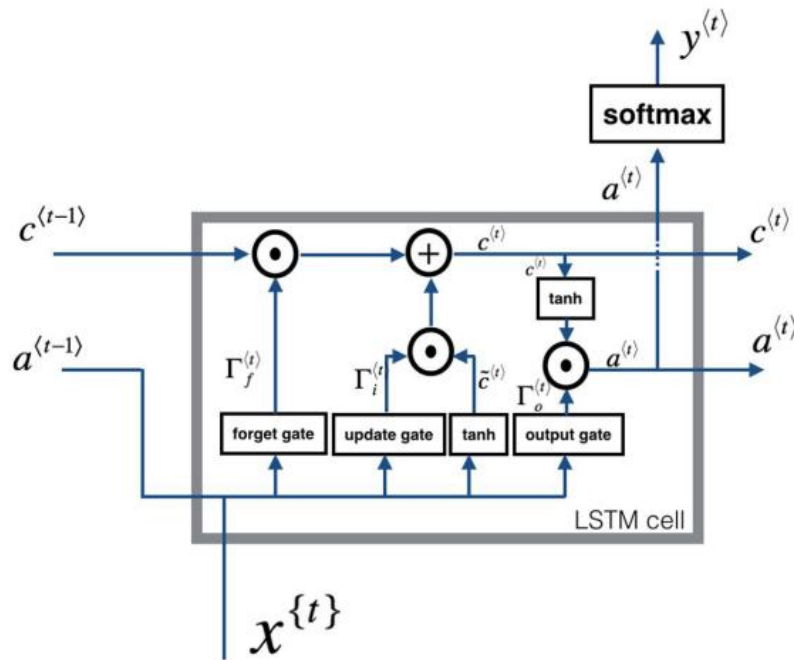
Their effectiveness comes from the gate mechanisms they utilize:

- the input gate
- forget gate
- output gate

These gates control the information flow, determining what to retain or remove from memory and what to output at each step.

These gating mechanisms allow LSTMs to properly manage long-term dependencies.

Long Short-Term Memory (LSTM) unit



$$\tilde{c}^{(t)} = \tanh(W_c[a^{(t-1)}, x^{(t)}] + b_c) - \text{candidate}$$

$$\Gamma_u = \sigma(W_u[a^{(t-1)}, x^{(t)}] + b_u) - \text{update gate}$$

$$\Gamma_f = \sigma(W_f[a^{(t-1)}, x^{(t)}] + b_f) - \text{forget gate}$$

$$\Gamma_o = \sigma(W_o[a^{(t-1)}, x^{(t)}] + b_o) - \text{output gate}$$

$$c^{(t)} = \Gamma_u * \tilde{c}^{(t)} + \Gamma_f * c^{(t-1)} - \text{update memory cell}$$

$$a^{(t)} = \Gamma_o * \tanh(c^{(t)}) - \text{activation}$$

LSTM outputs both activation value and memory cell

Gates ($\Gamma_f, \Gamma_u, \Gamma_o$) contain sigmoid activations \Rightarrow values between 0 and 1.

Forget gate decides what is relevant to keep from prior steps.

Inf from the previous hidden state and from the current input is passed through the sigmoid function. Output values between 0 and 1.

Closer to 0 means to forget, closer to 1 means to keep.

Update gate decides what inf is relevant to add from the current step.

Output gate determines the next hidden state based on the updated cell state. It filters the information that the LSTM will output based on the updated cell state.

LSTM – Internal States

Cell State:

Acts as the **memory** of the LSTM.

This runs straight down the entire chain of the LSTM, with minimal modifications.

Only interacts linearly with gates (forget, update, output).

It's the core differentiator in LSTMs that allows them to maintain and control long-term dependencies.

Hidden State:

The output of the LSTM at each time step.

Derived from the cell state, modulated by the output gate.

Encodes short-term information for the current moment in the sequence.

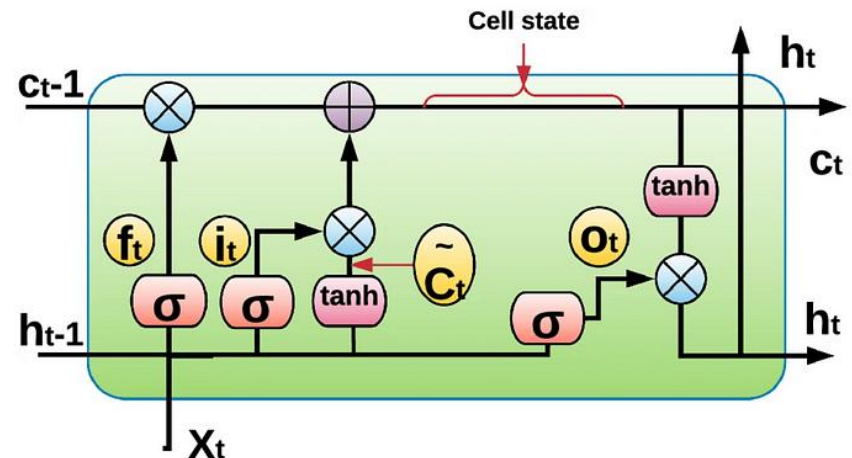
Captures what the LSTM "knows" at that point.

Gates, the special neural layers that control the flow of information:

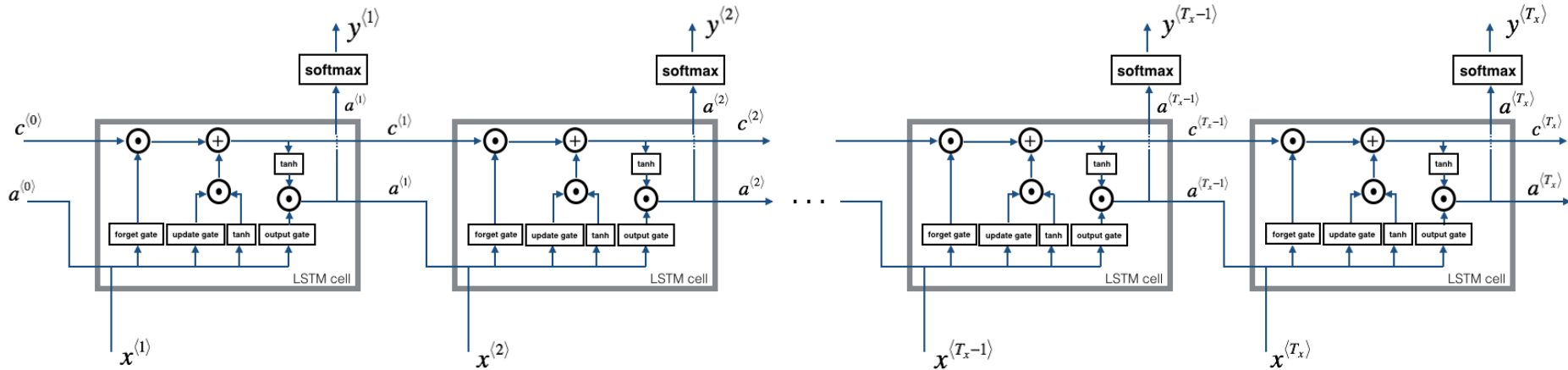
Forget Gate: Decides what to discard from the cell state.

Update Gate: Decides what new information to store.

Output Gate: Decides what to output from the cell state.



LSTM network



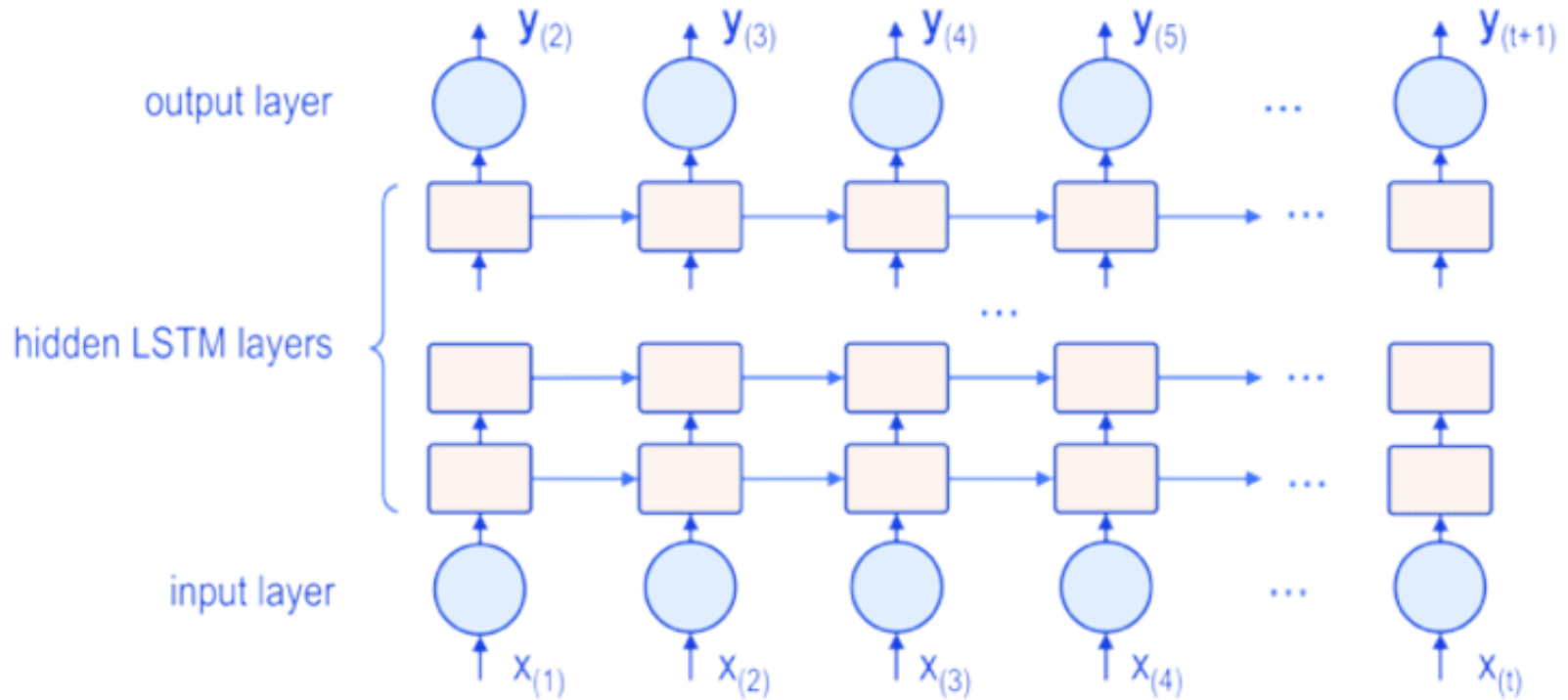
An LSTM network is a temporal sequence of LSTM units.

There is a line at the top that shows how LSTM can memorise and pass certain values $c^{<t>}$ through several temporal steps.

Other versions: the gate's values also depend on $c^{<t-1>}$. (peephole connection)

The gates are vectors (e.g., 100 elements).

Deep LSTM



Gated Recurrent Unit



Gated Recurrent Units (GRUs)

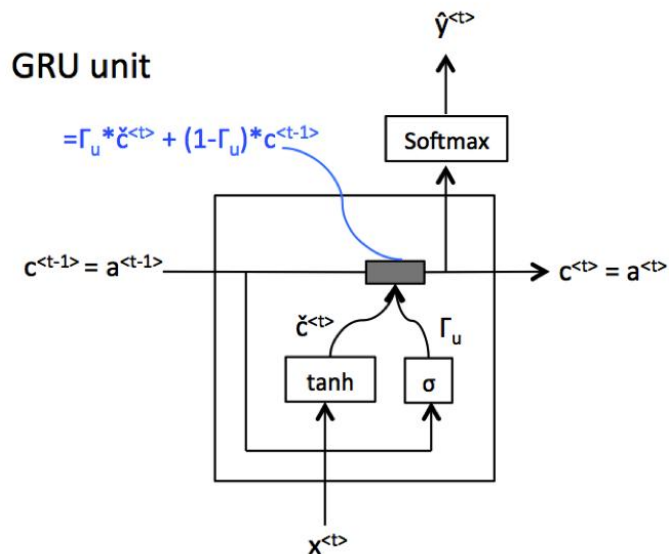
GRUs are a streamlined variant of RNNs that modify the LSTM design to achieve similar results with less complexity.

GRUs merge the input and forget gates into a single “update gate” and combine the cell state and hidden state. This simpler structure reduces the model’s complexity, which can lead to quicker computations and lower memory usage.

GRUs are often favored over LSTMs for smaller datasets where LSTMs might overfit due to their complexity, or when faster training is essential. They perform well on many tasks that don’t involve very long-term dependencies.



Gated Recurrent Unit (basic architecture)



c - memory cell

$c^{<t>} = a^{<t>}$ (for LSTM they are different)

GRU outputs activation value = memory cell

$\tilde{c}^{<t>} = \tanh(w_c [c^{<t-1>}, x^{<t>}] + b_c)$ - candidate

$\Gamma_u = \sigma(w_u [c^{<t-1>}, x^{<t>}] + b_u)$ - update gate

$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$ - update memory cell

$\Gamma_u \Rightarrow$ values between (0,1)

if $\Gamma_u \Rightarrow 0$, don't update $c^{<t>}$, if $\Gamma_u \Rightarrow 1$, update $c^{<t>}$.

$\Gamma_u=1$ $\Gamma_u=0$ $\Gamma_u=0$ $\Gamma_u=0$ $\Gamma_u=1$
 $c^{<t>} = 1$, keeps the same value of $c^{<t>}$ long time

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

Memory cell **c** memorises if the girl was singular or plural even if **girl-was** are separated with many many words. Efficient against vanishing gradients.

$c^{<t>}$ and Γ_u are vectors (e.g. 100 elements/bits to update)

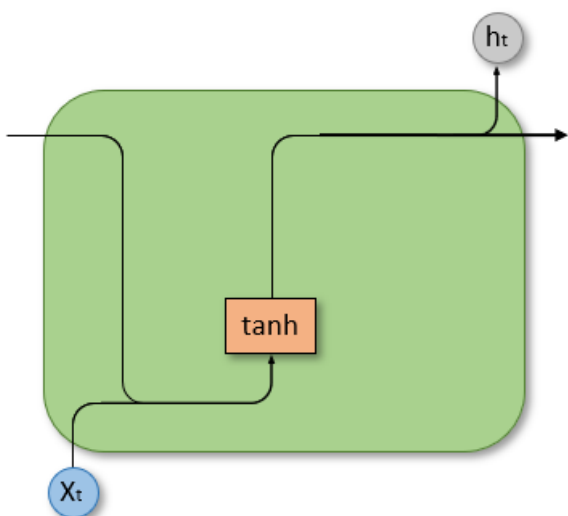
GRU vs LSTM

Similarities:

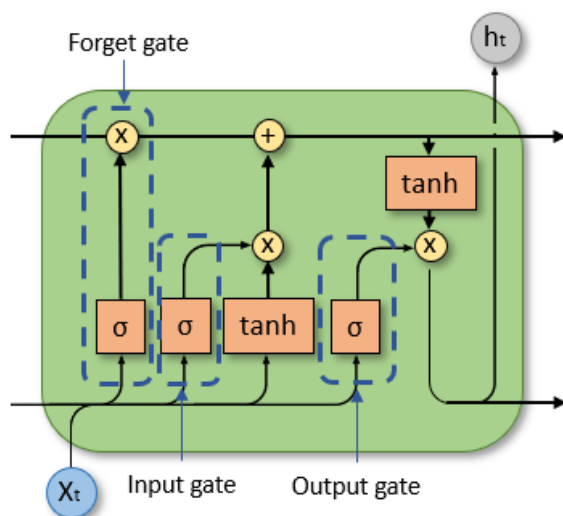
- Both are advanced RNNs designed to handle sequential data.
- Aim to solve the vanishing gradient problem in traditional RNNs.
- Use gating mechanisms to control the flow of information.
- Capable of learning long-term dependencies.
- Commonly used in NLP, time series forecasting, and other sequence tasks.

Feature	LSTM	GRU
Architecture	3 gates (forget, update(or input), output)	2 gates (reset, update)
Cell State	Maintains a separate cell state (Ct)	No separate cell state, only ht
Complexity	More parameters, heavier computation	Fewer parameters, faster to train
Performance	May perform better on very long sequences	Often comparable, simpler to tune
Memory Control	Explicit memory via cell state	Merged memory & output in ht

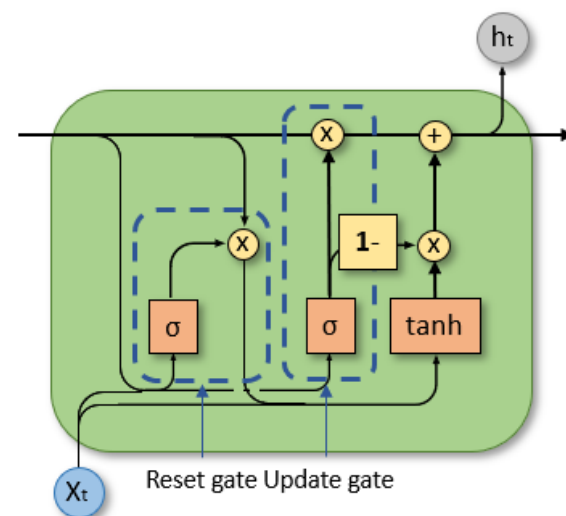
GRU vs LSTM



RNN



LSTM



GRU

Summary:

Use LSTM when fine control of long-term memory is needed.

Use GRU for faster training and when less memory overhead is desired.

Advantages of RNNs:

- Handle sequential data: They are well-suited for processing data where the order of elements is crucial.
- Learn long-term dependencies: Their internal memory allows them to learn patterns over longer sequences.

Limitations of RNNs:

- Vanishing Gradient Problem: RNNs can struggle to learn long-range dependencies due to the vanishing gradient problem, where the gradient of the loss function becomes too small to update the weights effectively.
- Parallelization Challenges: Due to their sequential nature, RNNs can be difficult to parallelize, which can slow down training.

Advantages of LSTMs

The advantages of the LSTM are similar to RNNs, with the main benefit of their inner capability to capture patterns in the long-term and short-term of a sequence.

Limitations of LSTMs

- Due to their more complex structure, LSTMs are computationally more expensive, leading to longer training times.
- As the LSTM also uses the backpropagation in time algorithm to update the weights, the LSTM suffers from the disadvantages of the backpropagation (e.g., dead ReLu elements, exploding gradients).

Advantages of GRUs

- Due to the simpler architecture compared to LSTMs (i.e., two instead of three gates and one state instead of two), GRUs are computationally more efficient and faster to train as they need less memory.
- GRUs have proven to be more efficient for smaller sequences.

Limitations of GRUs

As GRUs do not have a separate hidden and cell state, they might not be able to consider observations as far into the past as the LSTM.

Similar to the RNN and LSTM, the GRU also might suffer from the disadvantages of the backpropagation in time to update the weights.