MIDDLE EAST TECHNICAL UNIVERSITY

# MMI714 Generative Models for Multimedia

## Auto-Regression

# Welcome

This is MMI714
“Generative Models  for Multimedia”

This week we will talk about:

Autoregressive Models, Time Series,
Deep Autoregressive networks,
RNNs for Auto-Regression,
HMMs for Auto-Regression,
Auto-Regression vs
Time Series Generation, chatGPT

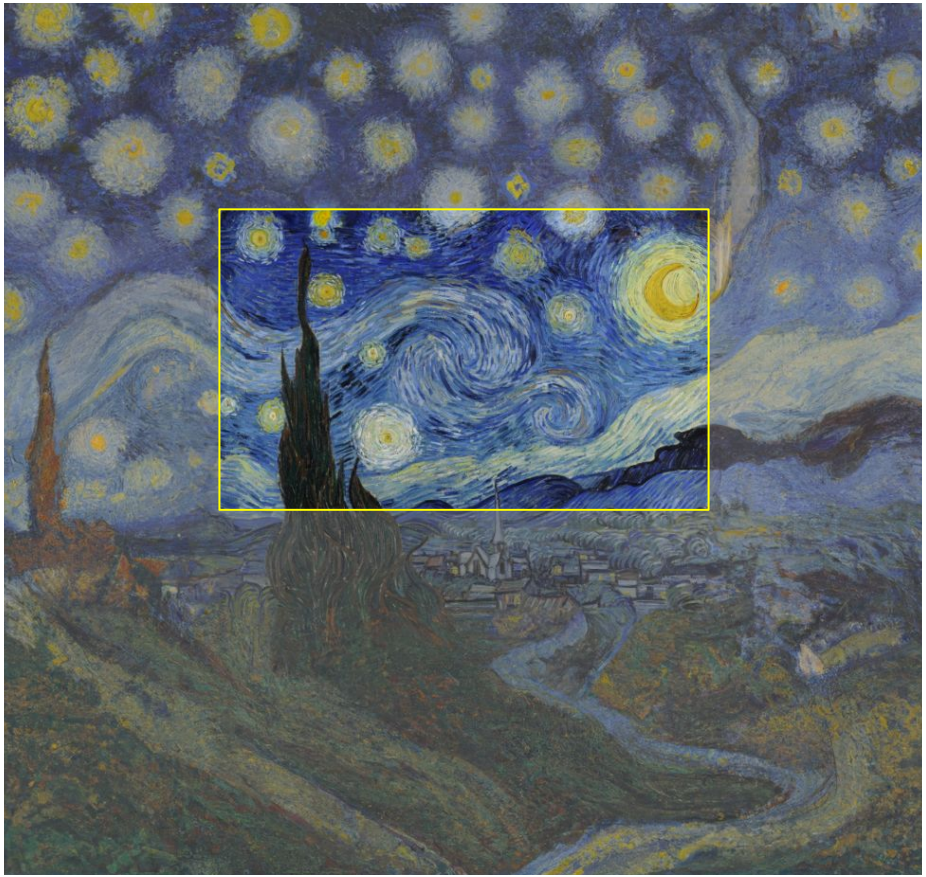
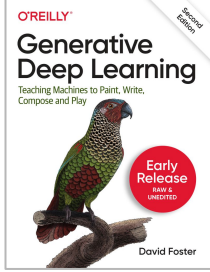


Image by openai.org

# AutoRegressive Models & etc.

- Our main focus will on autoregressive models - a family of model that simplifies the generative modeling problem by treating it as a sequential process.
- Autoregressive models condition predictions on previous values in the sequence, rather than on a latent random variable.
- Therefore they attempt to explicitly model the data generating distribution rather than an approximation to it




page 133

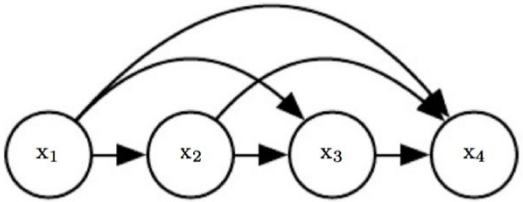
**Chapter Goals**

In this chapter you will:

- Learn why autoregressive models are well suited to generating sequential data such as text.
- Learn how to process and tokenize text data
- Learn about the architectural design of Recurrent Neural Networks (RNNs)
- Build and train an Long Short-Term Memory (LSTM) network from scratch using Keras
- Use the LSTM to generate new text
- Learn about other variations of RNNs including Gated Recurrent Units (GRUs) and bidirectional cells.
- Understand how image data can be treated as a sequence of pixels
- Learn about the architectural design of a PixelCNN
- Build a PixelCNN from scratch using Keras
- Use the PixelCNN to generate images

# Autoregression

- The key principle behind autoregressive modeling is the assumption that the probability distribution of each variable in the sequence depends on the values of the previous variables.
- In other words, the model conditions its predictions on the previously generated values.

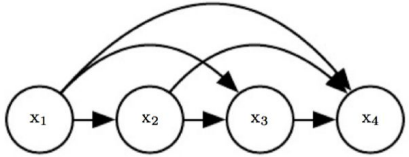


# Autoregression (rough evolution, not chronological though)

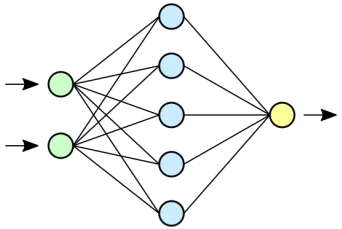
- **Finite State Machines (Deterministic)**: Great for modeling systems with a finite number of states and clear transitions. However, they lack the ability to capture probabilistic behavior.
- **Markov Chains (Probabilistic)**: Introduces probability into the state transitions. Each state is dependent only on the previous state, making it a memoryless system. Useful for modeling random processes.
- **Autoregression (Markov with Memory)**: Building on Markov chains, autoregression allows for memory in the system. Each state depends not only on the previous state but also on a set of previous states, enabling the model to capture longer-term dependencies.

# Regression vs Sequentiality

Sequentiality:  is the linear, unidirectional succession of elements or events, either reversible (as with motion in space) or irreversible (as in the flow of time).

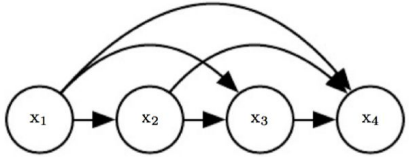


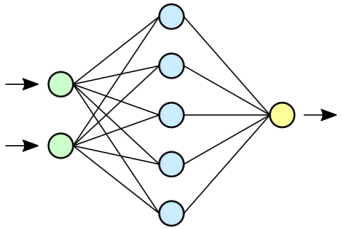

Regression: is the statistical technique that relates a dependent variable to one or more independent (explanatory) variables.




# Sequentiality, a must?

- In the context of generative models, autoregressive models typically involve sequential regression, where the prediction of each variable depends on the previously generated variables.
- However, it is possible to think of autoregressive models that do not explicitly have a sequential context and still make predictions based on previous elements.



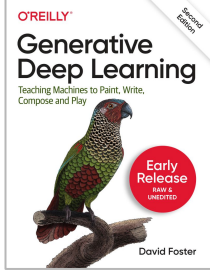
# Sequentiality, a must?

**PixelCNN**

In 2016, van den Oord et al.[4] introduced a model that generates images pixel by pixel by predicting the likelihood of the next pixel based on the pixels before it. The model is called **PixelCNN**, and it can be trained to generate images autoregressively.

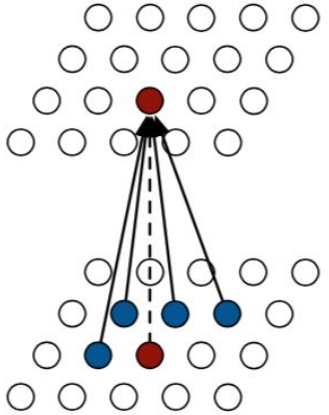
4 van den Oord et al., "Pixel Recurrent Neural Networks" *https://arxiv.org/pdf/1601.06759*.

158 | Chapter 5: Autoregressive Models




page 158

- Van den Oord's PixelCNN (2016) is an example of an autoregressive model that breaks away from strict sequential generation.
- In PixelCNN, the generation of each pixel in an image depends on the previously generated pixels, but the order of generation is not strictly sequential.
- Each pixel is generated by taking into account the previously generated pixels, but the order of generation can be arbitrary within the constraints imposed by the masked convolutions.



PixelCNN

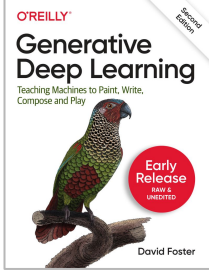
# PixelCNN

**PixelCNN**

In 2016, van den Oord et al.[4] introduced a model that generates images pixel by pixel by predicting the likelihood of the next pixel based on the pixels before it. The model is called **PixelCNN**, and it can be trained to generate images autoregressively.

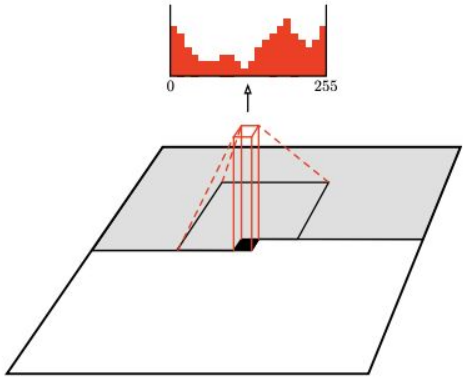
4 van den Oord et al., "Pixel Recurrent Neural Networks" *https://arxiv.org/pdf/1601.06759*.



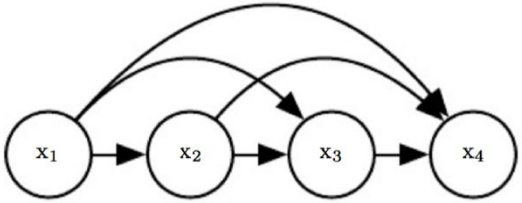



page 158

- A PixelCNN is a generative model that uses autoregressive connections to model images pixel by pixel, decomposing the joint image distribution as a product of conditionals.
- PixelCNNs are much faster to train than PixelRNNs because convolutions are inherently easier to parallelize; given the vast number of pixels present in large image datasets this is an important advantage.

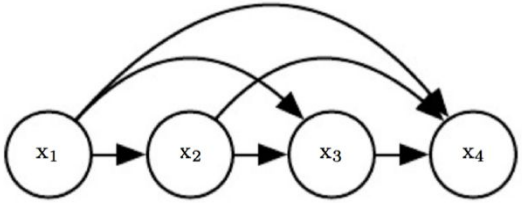


# Sequentiality is typical!




- Typically, in autoregressive generative models, some form of sequentiality or temporal dependency is typically present. The essence of autoregression lies in the idea that each variable in the sequence is generated based on the previous variables.
- While there can be variations and deviations from strict sequentiality, the underlying principle of autoregression involves a dependency on the previously generated variables.
- This sequentiality allows the model to capture the structure and dependencies present in the data distribution, enabling the generation of coherent and realistic samples.

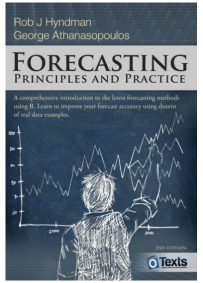
# Time Series (first regression - not Auto)




- Time series problems involve the analysis, modeling, and prediction of data points that are collected over time.
- In the context of generative models, time series problems focus on generating new sequences of data that resemble the patterns and characteristics observed in the training time series.
-

For example, we might wish to forecast monthly sales $y$ using total advertising spend $x$ as a predictor. Or we might forecast daily electricity demand $y$ using temperature $x_1$ and the day of week $x_2$ as predictors.

The **forecast variable** $y$ is sometimes also called the regressand, dependent or explained variable. The **predictor variables** $x$ are sometimes also called the regressors, independent or explanatory variables. In this book we will always refer to them as the "forecast" variable and "predictor" variables.



# Time Series Concepts

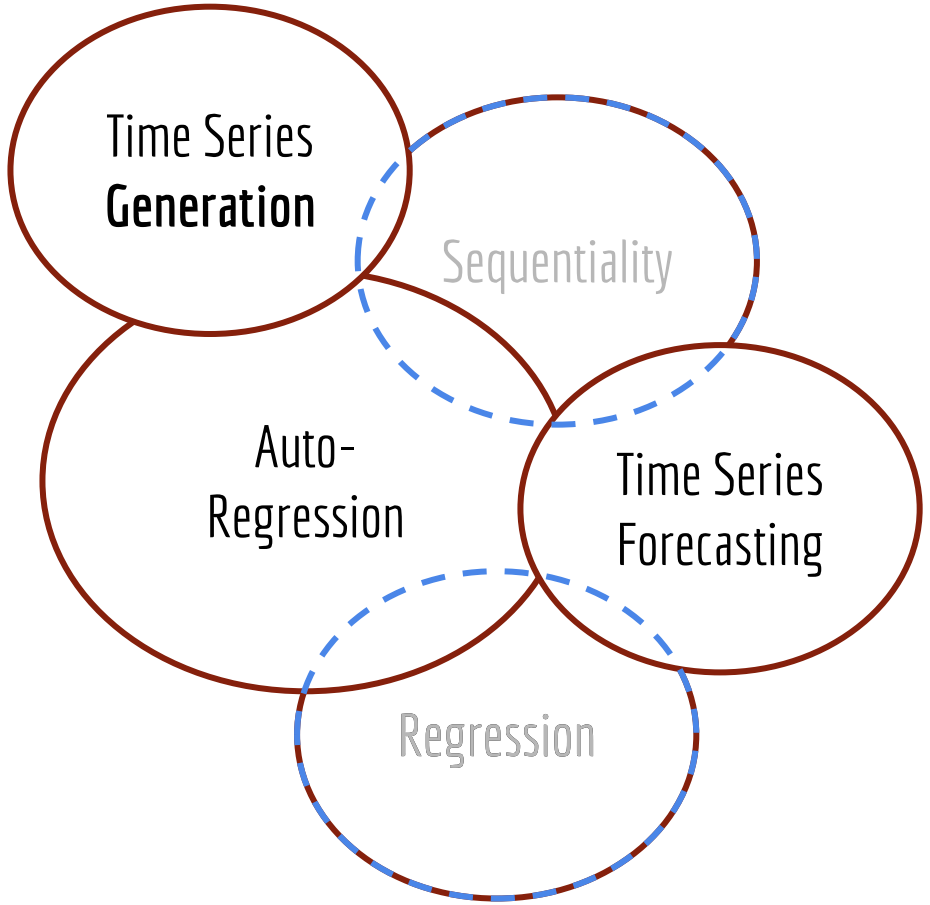
**Sequentiality**: unidirectional succession of elements or events, either reversible or irreversible

**Regression**: Dependency of a variable to one or more independent variables.

**Auto-Regression**: Dependency of a variable to one or more of its previous values/states

**Forecasting**: predicting the next sequential element

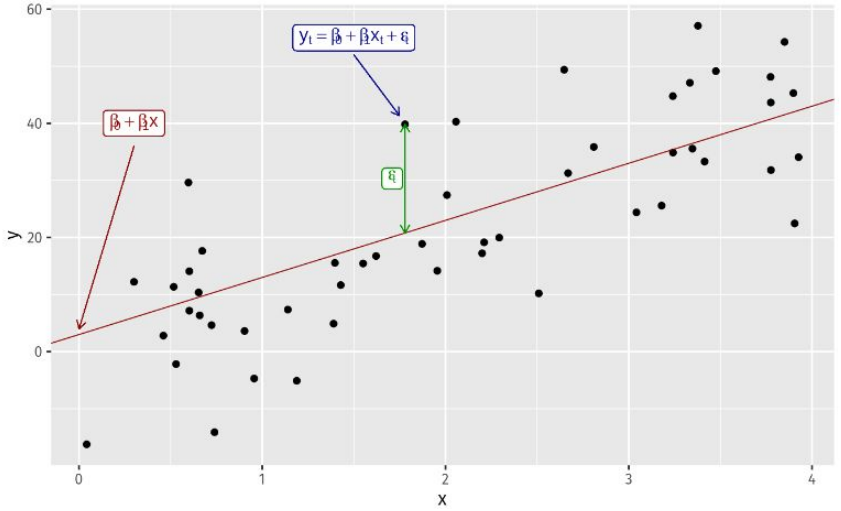
**Generation**: Creating a set a variable/vector out of a distribution



# Time Series Forecasting

## (first regression - not Auto)

SLR only depends on the latest variable

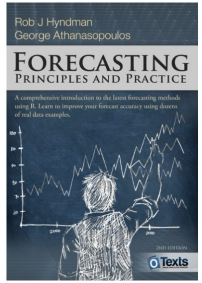



### Simple linear regression

In the simplest case, the regression model allows for a linear relationship between the forecast variable $y$ and a single predictor variable $x$:

$$y_t = \beta_0 + \beta_1 x_t + \varepsilon_t.$$

An artificial example of data from such a model is shown in Figure 5.1. The coefficients $\beta_0$ and $\beta_1$ denote the intercept and the slope of the line respectively. The intercept $\beta_0$ represents the predicted value of $y$ when $x = 0$. The slope $\beta_1$ represents the average predicted change in $y$ resulting from a one unit increase in $x$.



# Time Series Forecasting

## (first regression - not Auto)

MLR still only depends on the latest variables

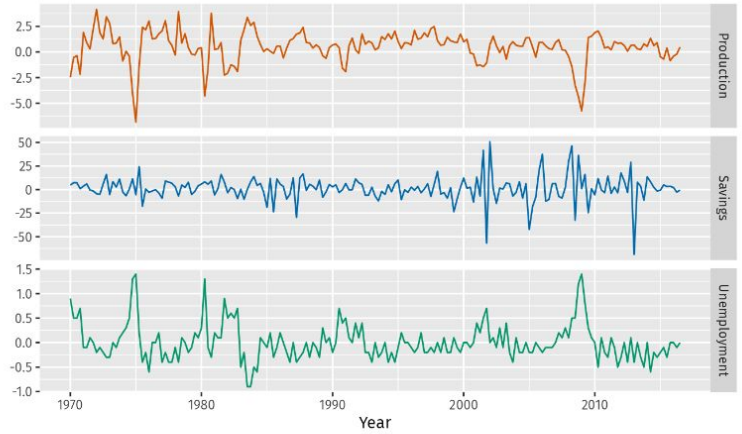


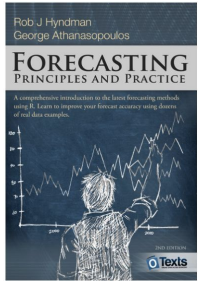

Figure 5.4: Quarterly percentage changes in industrial production and personal savings and quarterly changes in the unemployment rate for the US over the period 1970Q1-2016Q3.

### Multiple linear regression

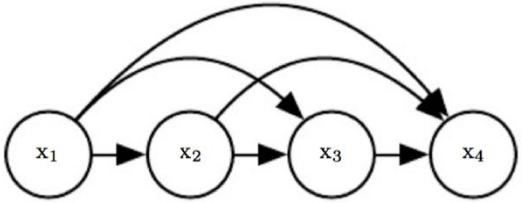
When there are two or more predictor variables, the model is called a **multiple regression model**. The general form of a multiple regression model is

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \cdots + \beta_k x_{k,t} + \varepsilon_t, \qquad (5.1)$$

where $y$ is the variable to be forecast and $x_1, \ldots, x_k$ are the $k$ predictor variables. Each of the predictor variables must be numerical. The coefficients $\beta_1, \ldots, \beta_k$ measure the effect of each predictor after taking into account the effects of all the other predictors in the model. Thus, the coefficients measure the *marginal effects* of the predictor variables.



# Time Series Forecasting




## 8.3 Autoregressive models

In a multiple regression model, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of *past values of the variable*. The term *auto*regression indicates that it is a regression of the variable against itself.

Thus, an autoregressive model of order $p$ can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

where $\varepsilon_t$ is white noise. This is like a multiple regression but with *lagged values* of $y_t$ as predictors. We refer to this as an **AR($p$) model**, an autoregressive model of order $p$.

Autoregressive models are remarkably flexible at handling a wide range of different time series patterns. The two series in Figure 8.5 show series from an AR(1) model and an AR(2) model. Changing the parameters $\phi_1, \ldots, \phi_p$ results in different time series patterns. The variance of the error term $\varepsilon_t$ will only change the scale of the series, not the patterns.

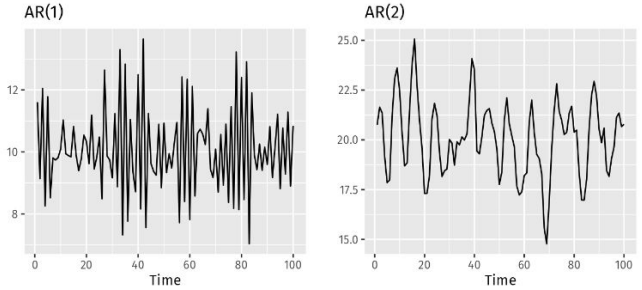
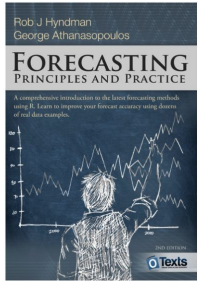
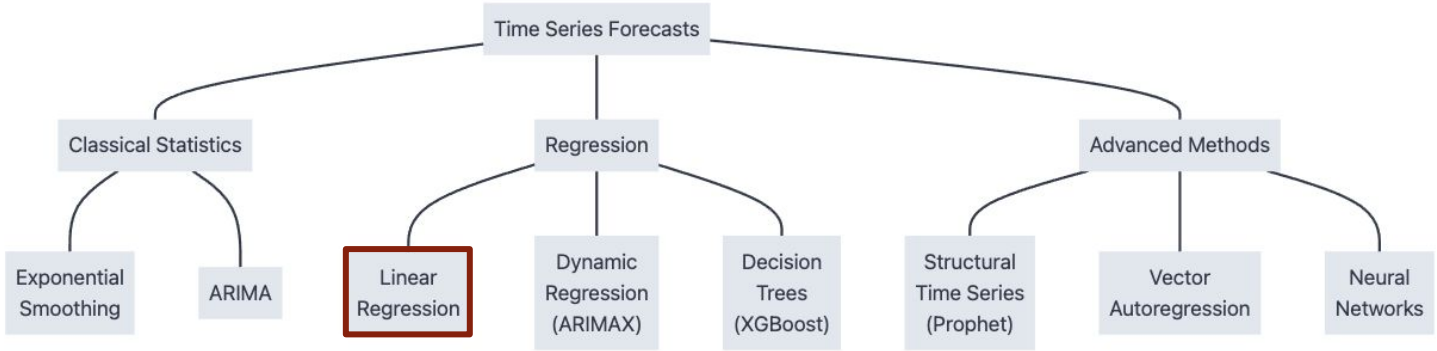



Figure 8.5: Two examples of data from autoregressive models with different parameters. Left: AR(1) with $y_t = 18 - 0.8y_{t-1} + \varepsilon_t$. Right: AR(2) with $y_t = 8 + 1.3y_{t-1} - 0.7y_{t-2} + \varepsilon_t$. In both cases, $\varepsilon_t$ is normally distributed white noise with mean zero and variance one.



# Time Series Forecasting

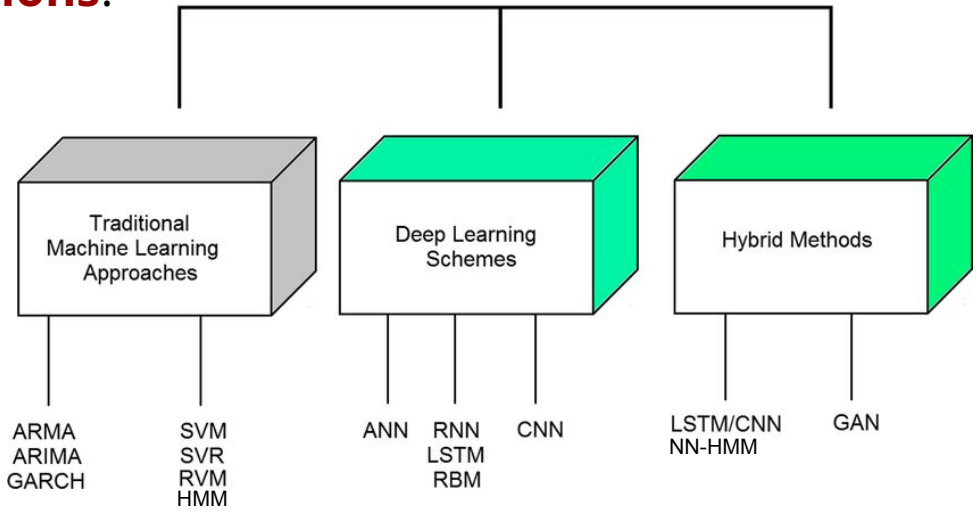
- The literature on time series forecasting is vast, encompassing a wide range of methods from traditional linear regression models to advanced deep learning architectures.
- This breadth reflects the diverse approaches available to capture and predict the complex dynamics and patterns present in time series data



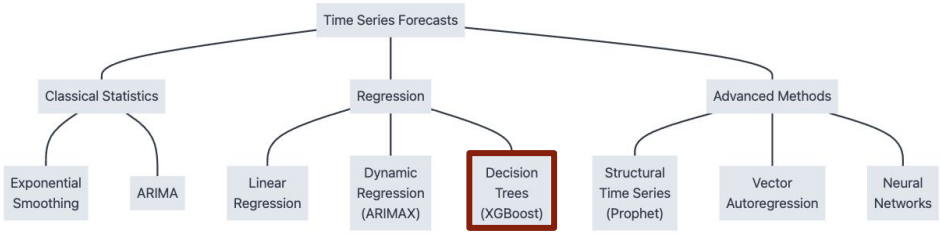
# Time Series Forecasting w. ML

- Deep learning has revolutionized time series forecasting by leveraging the power of neural networks to **capture complex temporal patterns** and **achieve accurate predictions**.

- Let's explore how deep learning techniques, such as recurrent neural networks and convolutional networks, have been successfully applied to tackle time series forecasting tasks.

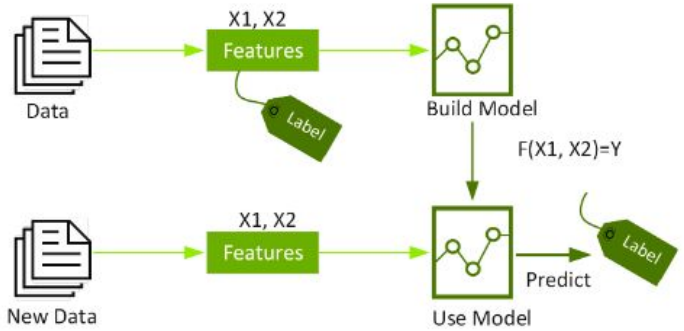


# XGBoost




- XGBoost (Extreme Gradient Boosting) is a powerful machine learning algorithm known for its effectiveness in various tasks, including time series forecasting.
- XGBoost is an ensemble learning method that combines multiple weak models (decision trees) to create a robust and accurate predictive model.
- XGBoost is a scalable and parallable!

(Find two YouTube videos on the ref list, About Gradient Boosting and XGBoost)



# HMMs for Auto-Regression

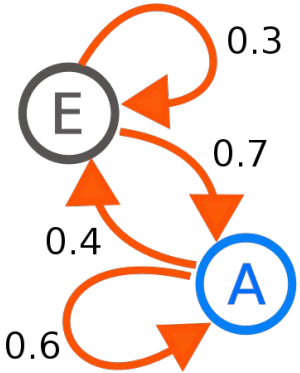
- Markovian Models (i.e. the Markov property) assume (in very simple terms) state that the future states of a stochastic process are influenced only by the present, not the past, meaning that the past can be disregarded once the present is known.
- So it is “a” way to model sequences.

## Markov chain

Article Talk

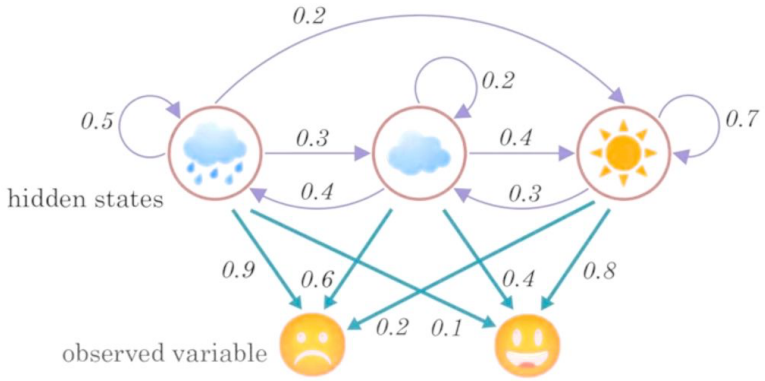
From Wikipedia, the free encyclopedia

A **Markov chain** or **Markov process** is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.[1][2][3] Informally, this may be thought of as, "What happens next depends only on the state of affairs *now*." A countably infinite sequence, in which the chain moves state at discrete time steps, gives a discrete-time Markov chain (DTMC). A continuous-time process is called a continuous-time Markov chain (CTMC). It is named after the Russian mathematician Andrey Markov.



# HMMs for Auto-Regression

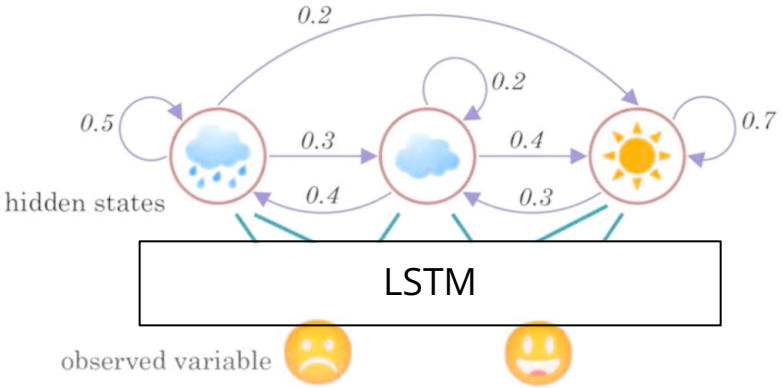
- In traditional HMMs, the **emission probabilities** from the hidden states to the observed states are typically modeled as simple categorical or Gaussian distributions.




$$\begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.4 \\ 0.0 & 0.3 & 0.7 \end{bmatrix} \quad \begin{bmatrix} 0.9 & 0.1 \\ 0.6 & 0.4 \\ 0.2 & 0.8 \end{bmatrix}$$

# HMMs for Auto-Regression

- In NN-HMMs, the neural network can take the hidden state as input and output a distribution over the observed states.
- This neural network can be a feedforward network, recurrent network (such as LSTM or GRU), or any other type of neural network architecture.

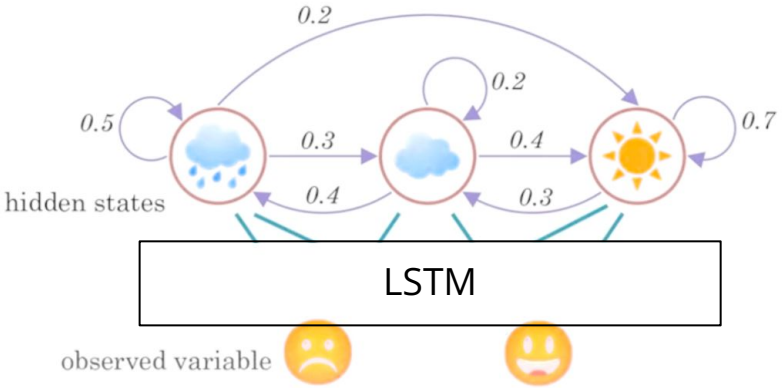



$$\begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.4 \\ 0.0 & 0.3 & 0.7 \end{bmatrix}$$

LSTM

# HMMs for Auto-Regression

- In the case of training an HMM, including NN-HMMs, when the hidden states are unknown, determining the number of hidden states is an important consideration.
- The number of hidden states is not automatically assigned by the Baum-Welch algorithm itself; it is typically specified as a parameter by the user.




$$\begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.2 & 0.4 \\ 0.0 & 0.3 & 0.7 \end{bmatrix}$$

LSTM

# HMMs for Auto-Regression

- In practice, you can use an LSTM as a replacement for an NN-HMM.
- Both NN-HMMs and LSTMs are capable of modeling sequential data and capturing dependencies over time.

# HMMs for Auto-Regression

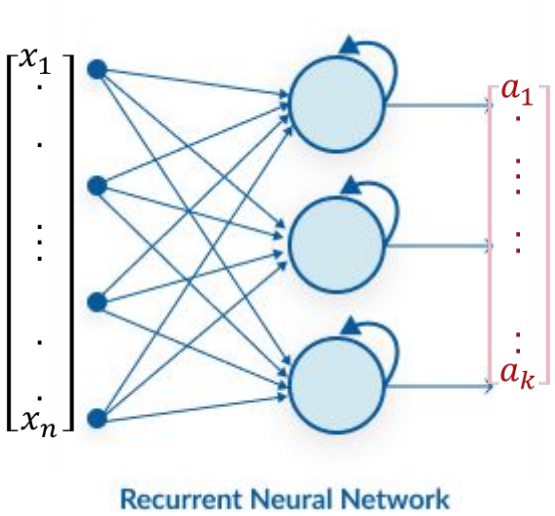
- LSTMs, as a type of recurrent neural network (RNN), do not strictly adhere to the Markovian property. Unlike traditional HMMs, which explicitly assume the Markovian property, LSTMs have the ability to capture longer-term dependencies in sequential data.
- In general, training a fully LSTM model without the additional complexity of an NN-HMM layer can be relatively easier compared to training an NN-HMM with an LSTM that estimates the emission probabilities.

# Time Series Forecasting w. DL

- Feedforward Neural Networks (ANNs, CNNs)
  - Non-linear approach to regression
- Recurrent Neural Networks (RNNs):
  - Sequential approach to regression
  - Long Short-Term Memory (LSTM) Networks:
- Hybrids
  - CNN+RNNs etc.
- Temporal Convolutional Networks (TCNs):
  - Convolutional operations for time series forecasting.
- Attention-based
- Encoder-Decoder, GANs Architectures:
  - This way leads to VAEs, GANs. (not covered this week)

# Vanilla RNN vs Standard FNN

- Now, what is different in RNNs?

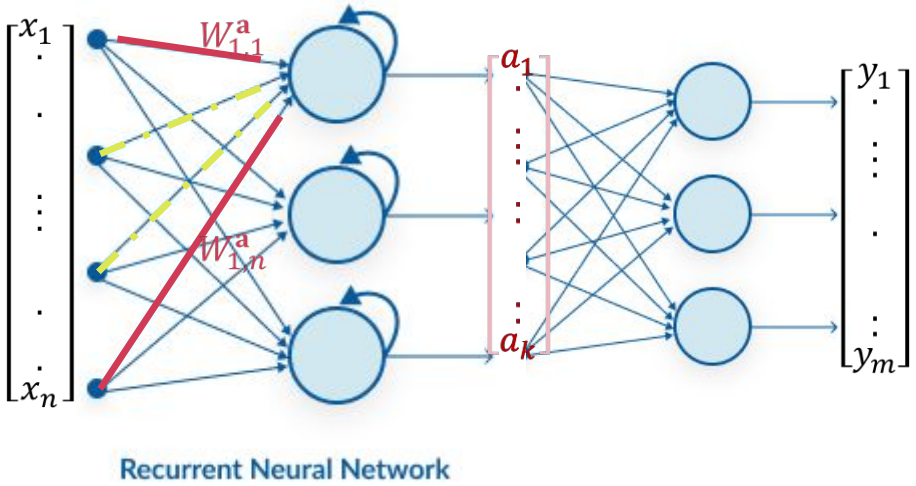



- In addition to FNN's connections, there is also a feedback. How to model it?
- Actually there's an additional set of weights (hence connections), in the conventionally accepted vanilla RNN.

# Vanilla RNN vs Standard FNN

- Now, what is different in RNNs?

This is the general model of a so-called “Vanilla RNN”




Recurrent Neural Network

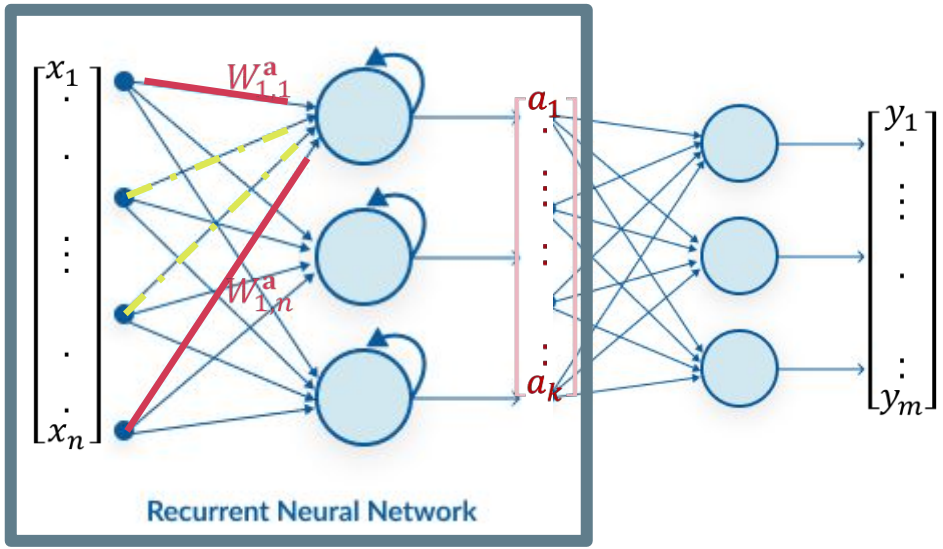
$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix}\right)$$

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right]\right)$$

# Vanilla RNN vs Standard FNN

- Now, what is different in RNNs?

This is the general model of a so-called "Vanilla RNN"




Core RNN:

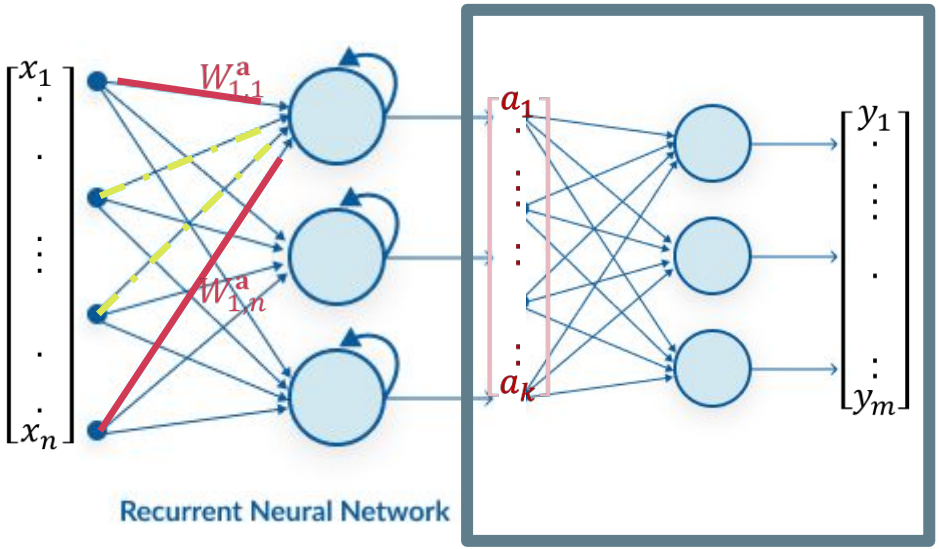
$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix}\right)$$

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right]\right)$$

# Vanilla RNN vs Standard FNN

- Now, what is different in RNNs?

This is the general model of a so-called “Vanilla RNN”




$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix}\right)$$

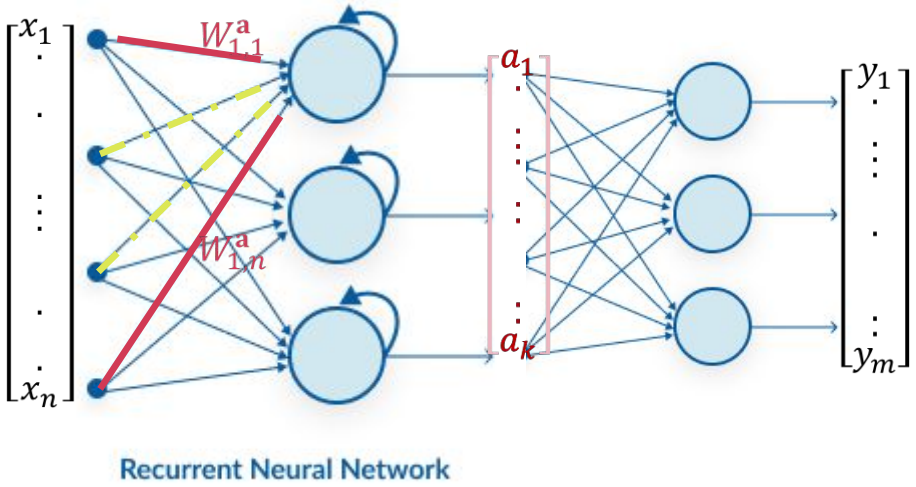
The Output:

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right]\right)$$

# Vanilla RNN vs Standard FNN

- Now, what is different in RNNs?

This is the general model of a so-called “Vanilla RNN”




Recurrent Neural Network

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x}^{prev}_{n\times 1} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{W}^{a}_{k\times(k+n)} = \begin{bmatrix} \mathbf{W}^{ax}_{k\times n} & \mathbf{W}^{aa}_{k\times k} \end{bmatrix}$$

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{ax}_{k\times n} \cdot \mathbf{x}^{prev}_{n\times 1} + \mathbf{W}^{aa}_{k\times k} \cdot \mathbf{a}^{prev}_{k\times 1} + \mathbf{b}^{a}_{k\times 1}\right)$$

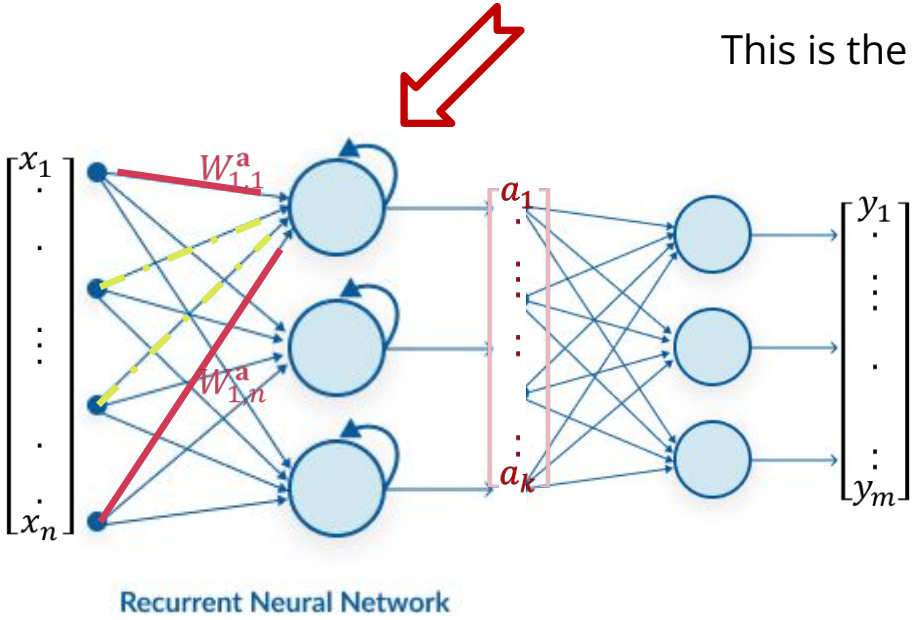
$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right] + \mathbf{b}^{y}_{m\times 1}\right)$$

# Vanilla RNN vs Standard FNN

- Now, what is different in RNNs?

This is the actual input

This is the general model of a so-called "Vanilla RNN"




Recurrent Neural Network

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x}^{prev}_{n\times 1} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix} + \mathbf{b}^{a}_{k\times 1}\right)$$

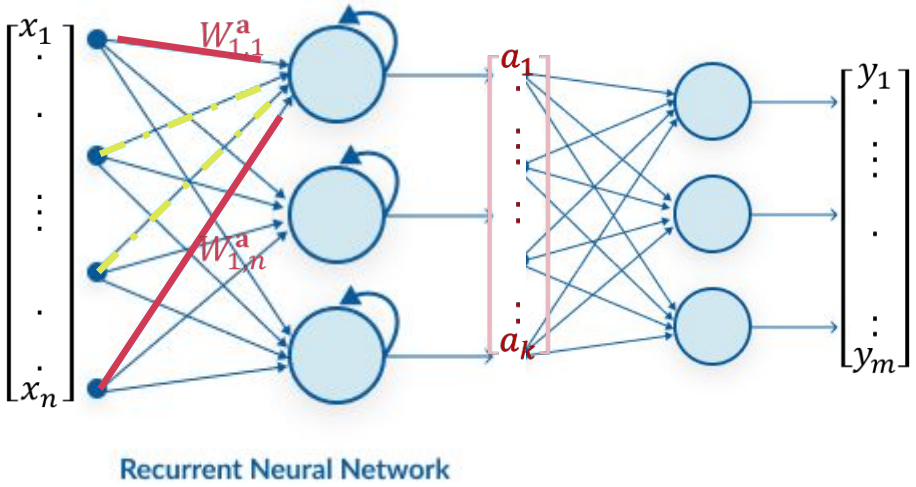
$$\mathbf{W}^{a}_{k\times(k+n)} = \begin{bmatrix} \mathbf{W}^{ax}_{k\times n} & \mathbf{W}^{aa}_{k\times k} \end{bmatrix}$$

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{ax}_{k\times n} \cdot \mathbf{x}^{prev}_{n\times 1} + \mathbf{W}^{aa}_{k\times k} \cdot \mathbf{a}^{prev}_{k\times 1} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right] + \mathbf{b}^{y}_{m\times 1}\right)$$

# Vanilla RNN vs Standard FNN

- This is a confusing diagram, which (almost) shows every connection (hence multiplication) that exists in a vanilla RNN.




Recurrent Neural Network

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x}^{prev}_{n\times 1} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{W}^{a}_{k\times(k+n)} = \begin{bmatrix} \mathbf{W}^{ax}_{k\times n} & \mathbf{W}^{aa}_{k\times k} \end{bmatrix}$$

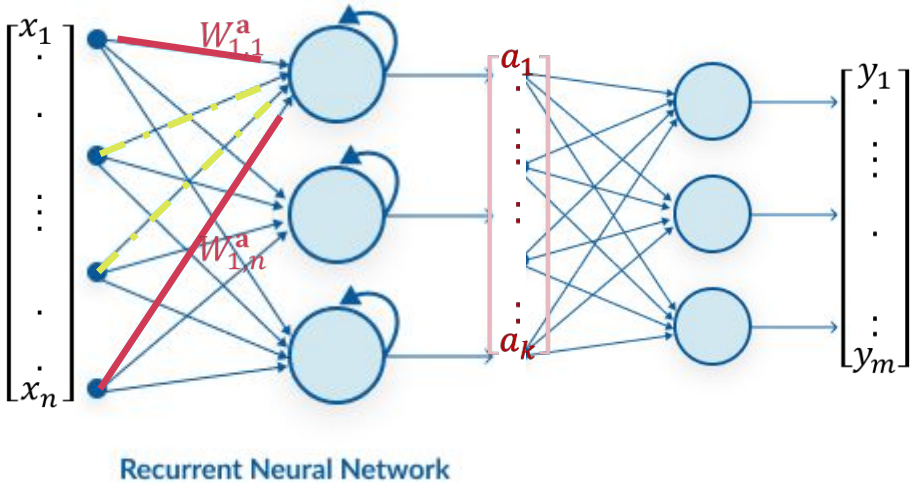
$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{ax}_{k\times n} \cdot \mathbf{x}^{prev}_{n\times 1} + \mathbf{W}^{aa}_{k\times k} \cdot \mathbf{a}^{prev}_{k\times 1} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right] + \mathbf{b}^{y}_{m\times 1}\right)$$

The one that Andrew NG uses in Coursera

# Vanilla RNN vs Standard FNN

- For the sake of simplicity, I am going to switch to a simpler depiction, keeping the same formulas.




Recurrent Neural Network

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x}^{prev}_{n\times 1} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix} + \mathbf{b}^{a}_{k\times 1}\right)$$

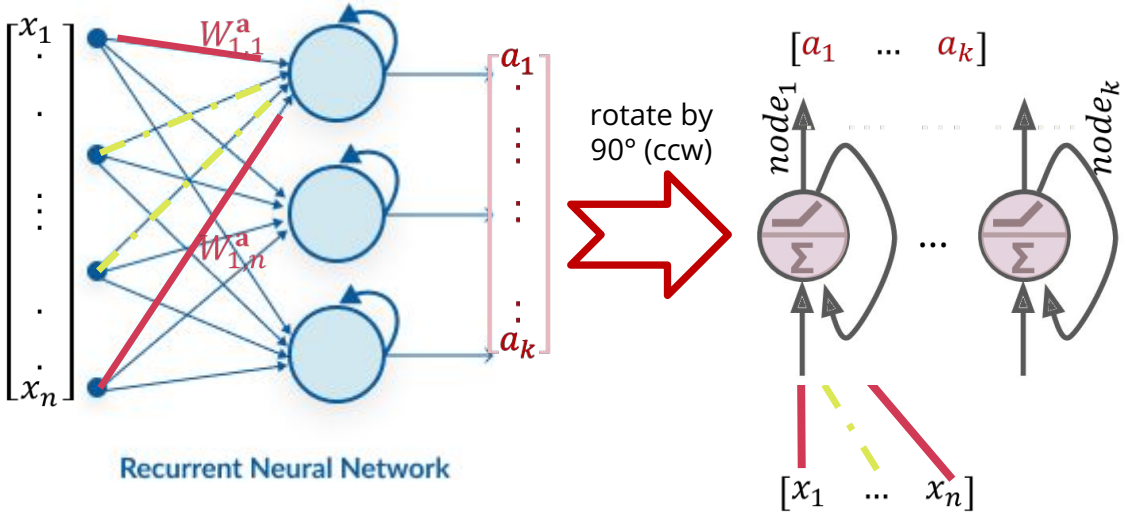
$$\mathbf{W}^{a}_{k\times(k+n)} = \begin{bmatrix} \mathbf{W}^{ax}_{k\times n} & \mathbf{W}^{aa}_{k\times k} \end{bmatrix}$$

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{ax}_{k\times n} \cdot \mathbf{x}^{prev}_{n\times 1} + \mathbf{W}^{aa}_{k\times k} \cdot \mathbf{a}^{prev}_{k\times 1} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot \left[\mathbf{a}^{next}_{k\times 1}\right] + \mathbf{b}^{y}_{m\times 1}\right)$$

# RNN «unrolled in time»

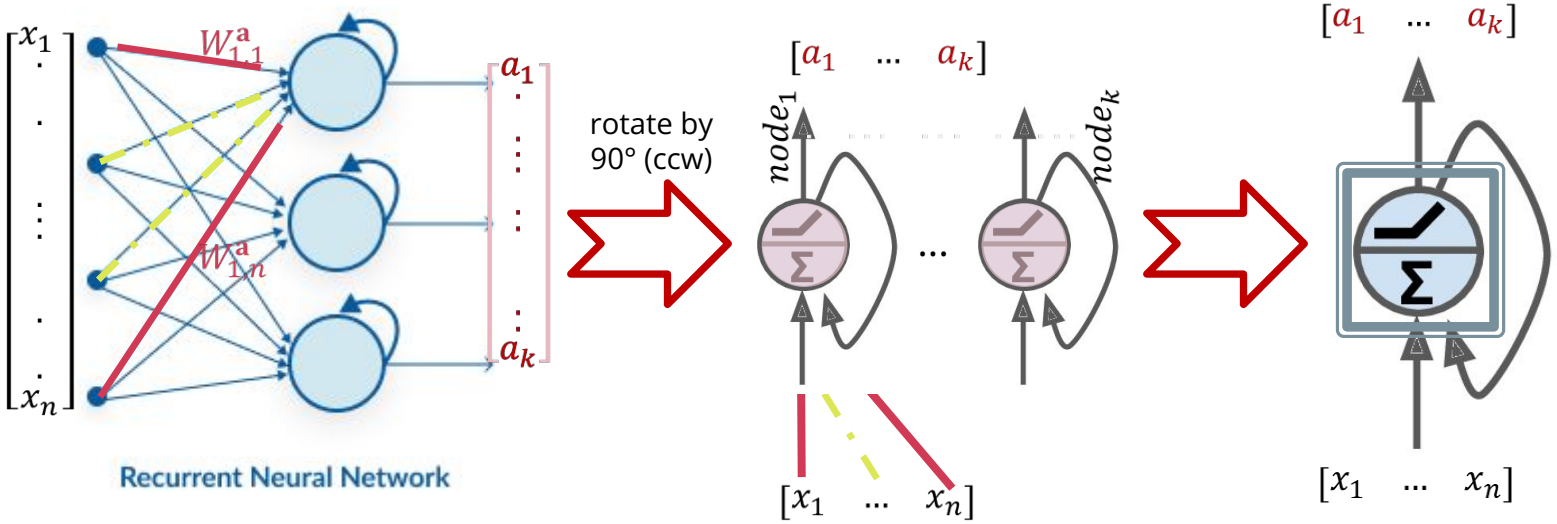
- We will first focus only on the core RNN (output layer is not shown!)



# RNN «super-node representation»

- We will first focus only on the core RNN (output layer is not shown!)

all recurrent nodes in a
«single super-node» representation

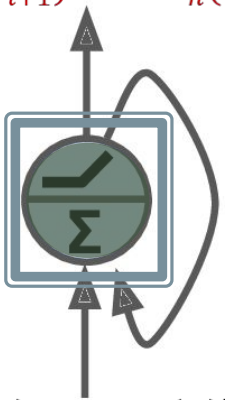


# RNN «super-node representation»

- $\mathbf{x} = [x_1 \quad \dots \quad x_n]$ is a single vector at time say «$t_0$». Let's denote it as:

$$\mathbf{x}(t_0) = [x_1(t_0) \quad \dots \quad x_n(t_0)]$$

$$\mathbf{a}(t_{i+1}) = [a_1(t_{i+1}) \quad \dots \quad a_n(t_{i+1})]$$



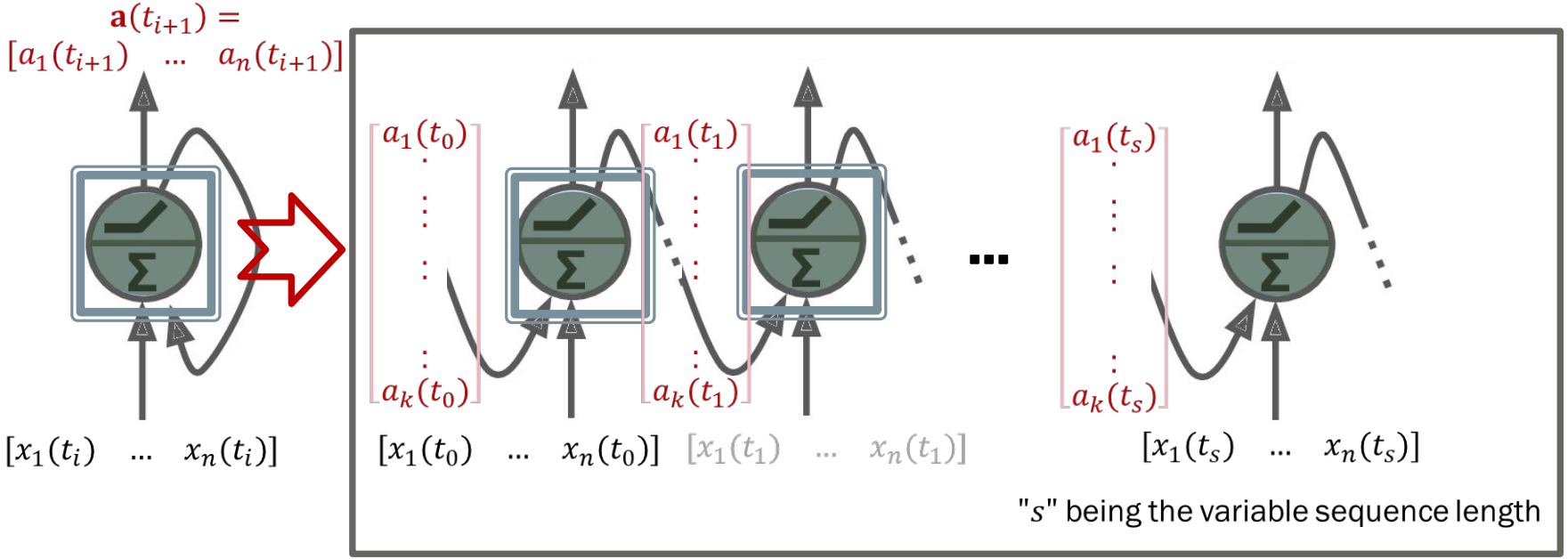
$$\mathbf{a}(t_i) = [a_1(t_i) \quad \dots \quad a_n(t_i)]$$

$$[x_1(t_i) \quad \dots \quad x_n(t_i)]$$

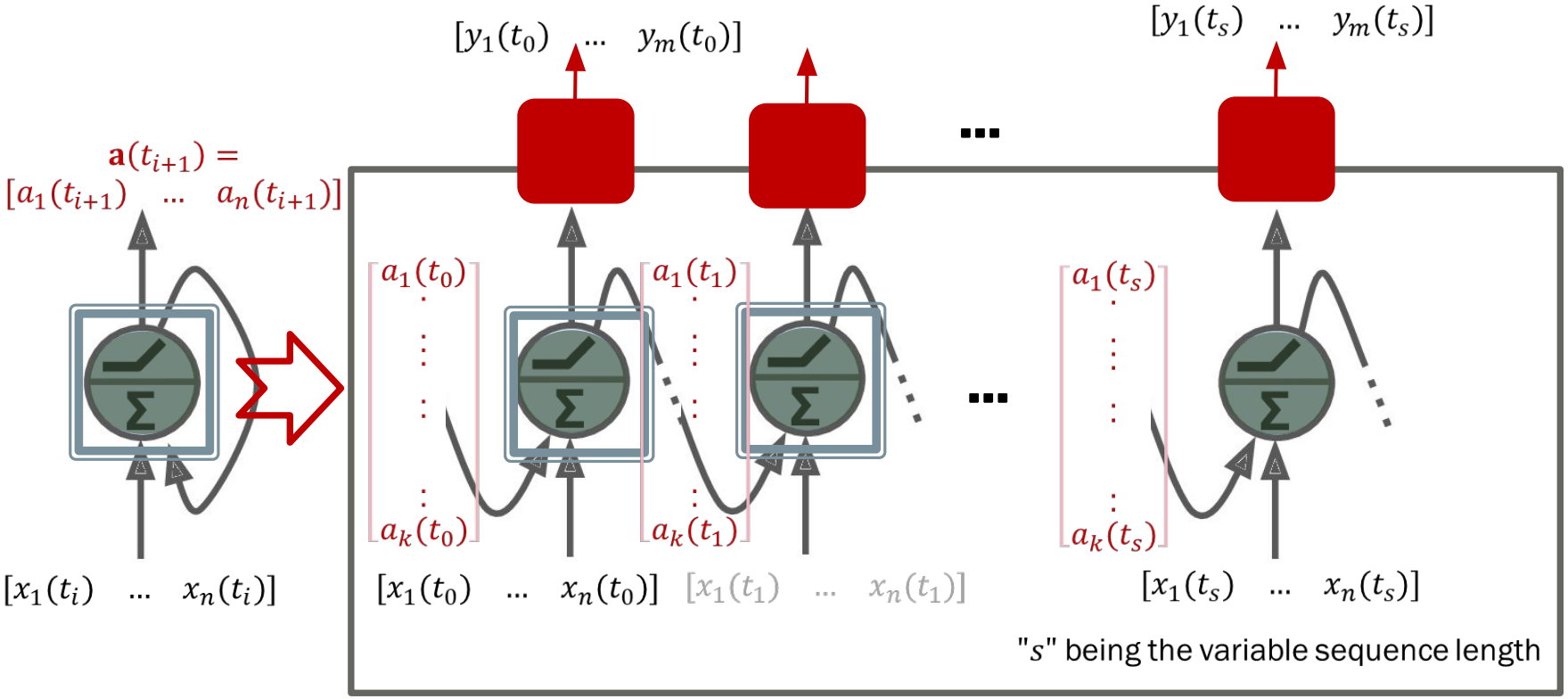
$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix} \mathbf{x}^{prev}_{n\times 1} \\ \mathbf{a}^{prev}_{k\times 1} \end{bmatrix} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{ax}_{k\times n} \cdot \mathbf{x}^{prev}_{n\times 1} + \mathbf{W}^{aa}_{k\times k} \cdot \mathbf{a}^{prev}_{k\times 1} + \mathbf{b}^{a}_{k\times 1}\right)$$

# RNN «unrolled in time»

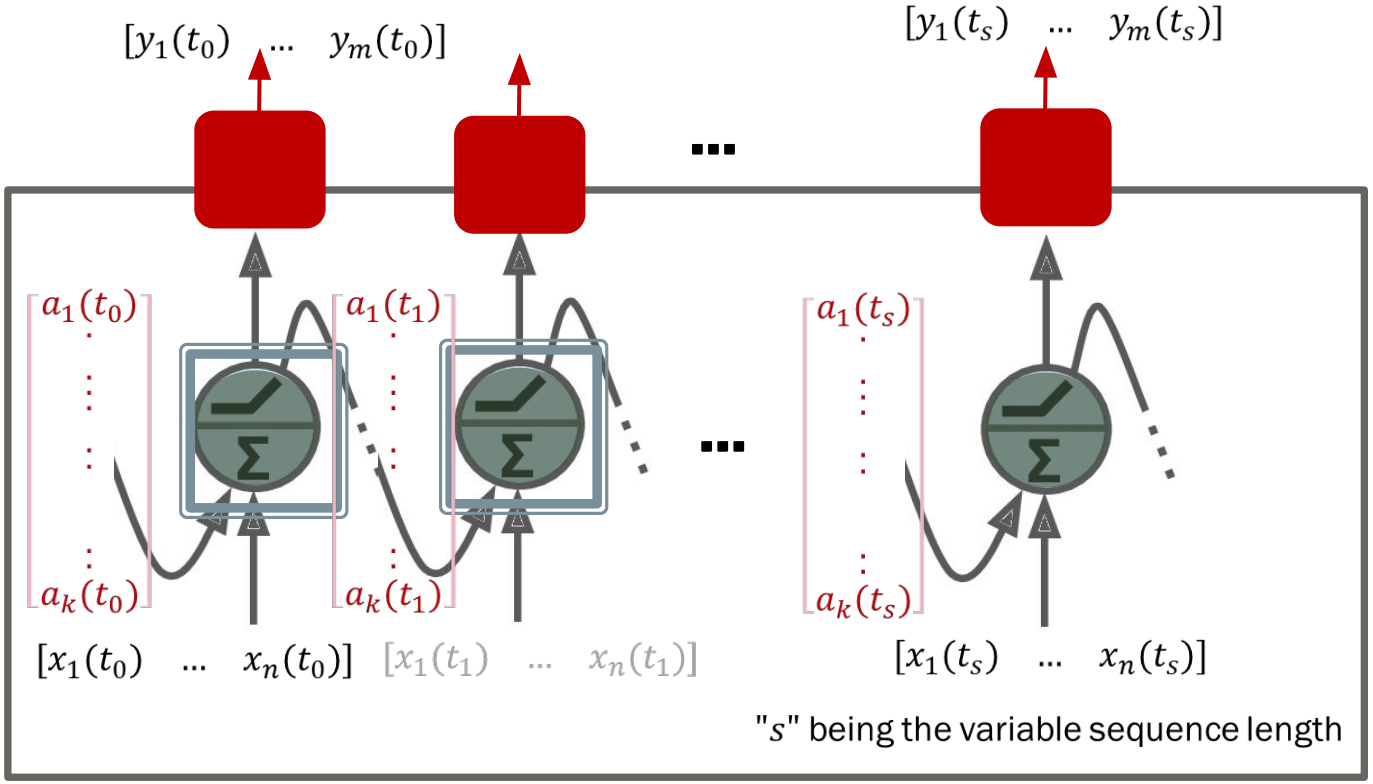


# Vanilla RNN «unrolled in time» with output



# Vanilla RNN «unrolled in time» with output

- k hidden nodes,
- s+1 long sequence
- n long features
- m long output



# Vanilla RNN «unrolled in time» with output

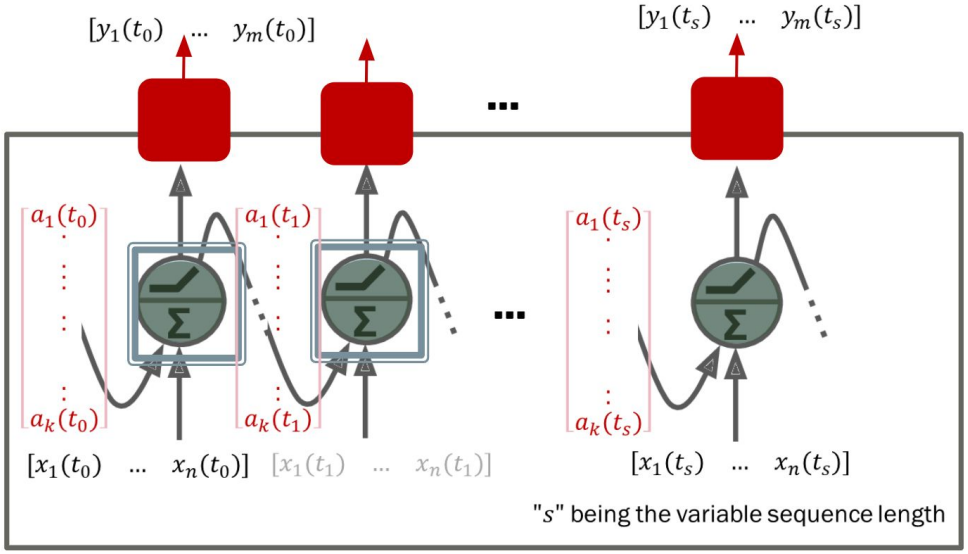
$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{a}_{k\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}^{prev}_{n\times 1}\\ \mathbf{a}^{prev}_{k\times 1}\end{bmatrix} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{W}^{a}_{k\times(k+n)} = [\mathbf{W}^{ax}_{k\times n} \quad \mathbf{W}^{aa}_{k\times k}]$$

$$\mathbf{a}^{next}_{k\times 1} = g\left(\mathbf{W}^{ax}_{k\times n} \cdot \mathbf{x}^{prev}_{n\times 1} + \mathbf{W}^{aa}_{k\times k} \cdot \mathbf{a}^{prev}_{k\times 1} + \mathbf{b}^{a}_{k\times 1}\right)$$

$$\mathbf{y}_{m\times 1} = g\left(\mathbf{W}^{y}_{m\times k} \cdot [\mathbf{a}^{next}_{k\times 1}] + \mathbf{b}^{y}_{m\times 1}\right)$$

Let's go over the formulation:

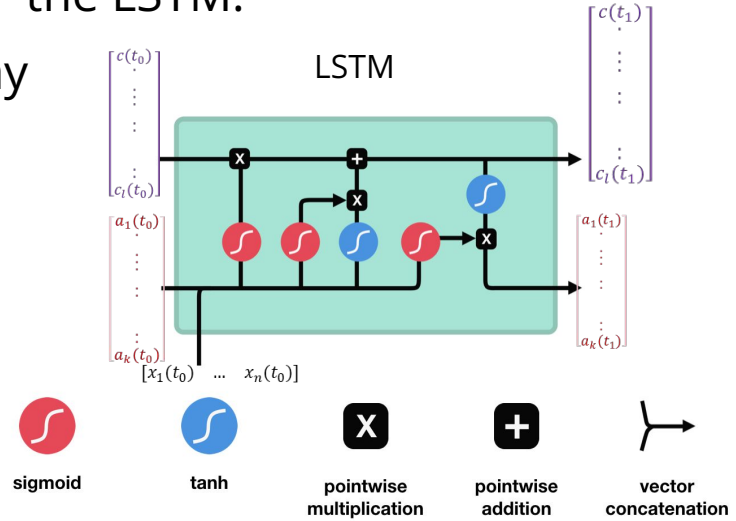


I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM

- **In addition to** hidden states, we have a concept of cell states in LSTMs,
- Hidden states of an LSTM are pretty much similar to hidden states of an RNN
- Cell states are related to the «gates» of the LSTM.
- The cell state act as a transport highway that transfers relative information all the way down the sequence chain.
- As the cell state goes on its journey, information gets added or removed to the cell state via gates.

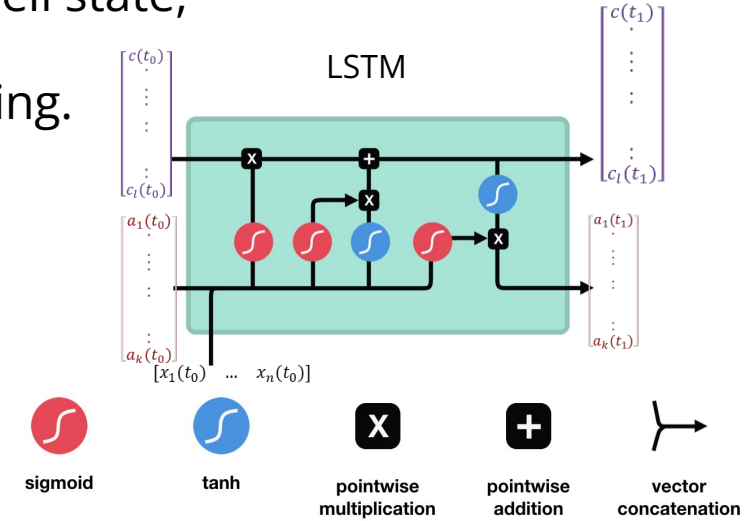


I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM

- Cell states are **long-term memory** of an LSTM
- Hidden states are the **working** (short-term) **memory.** They exist both in LSTM and RNN models
- The gates of an LSTM is to create the cell state, which can learn what information is relevant to keep or forget during training.



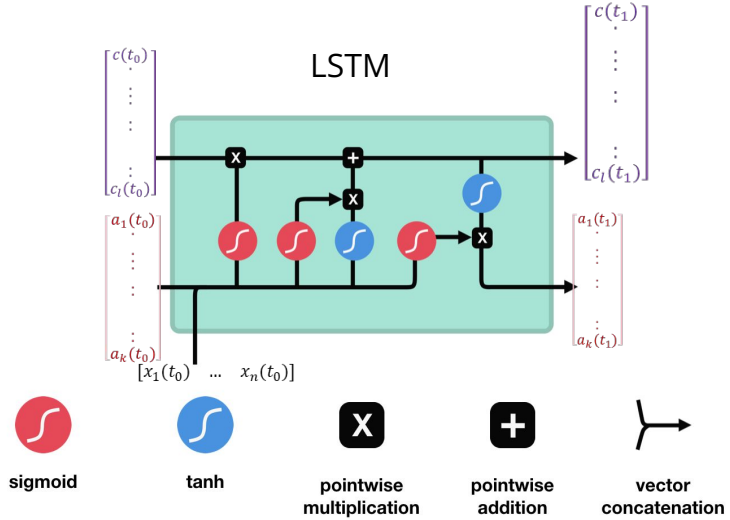
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM gates

- In order to understand how the gates of an LSTM function, let's first remember $tanh$ and $sigmoid$ functions

- $-1 < tanh(x) < +1$
- $0 \leq sigmoid\ (x) < +1$
- $sigmoid$ squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappears or be "*forgotten*."
- On the other hand, any number multiplied by 1 is the same value therefore that value stay's the same or is "*kept*."



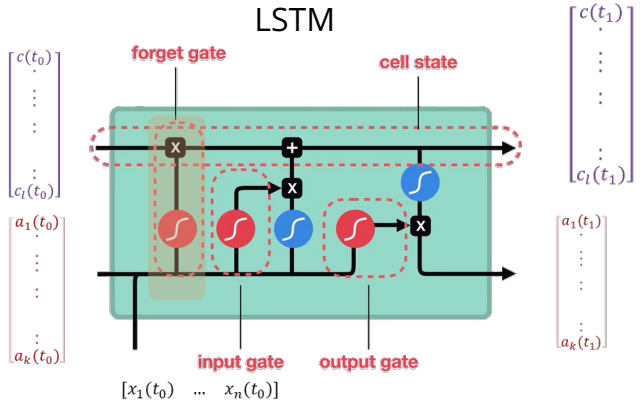
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM "forget" gate

- This gate decides what information should be thrown away (forgotten) or kept.

- Information from the previous hidden state and information from the current input is passed through the sigmoid function.
- Values come out between 0 and 1.
- The closer to 0 means to forget, and the closer to 1 means to keep.



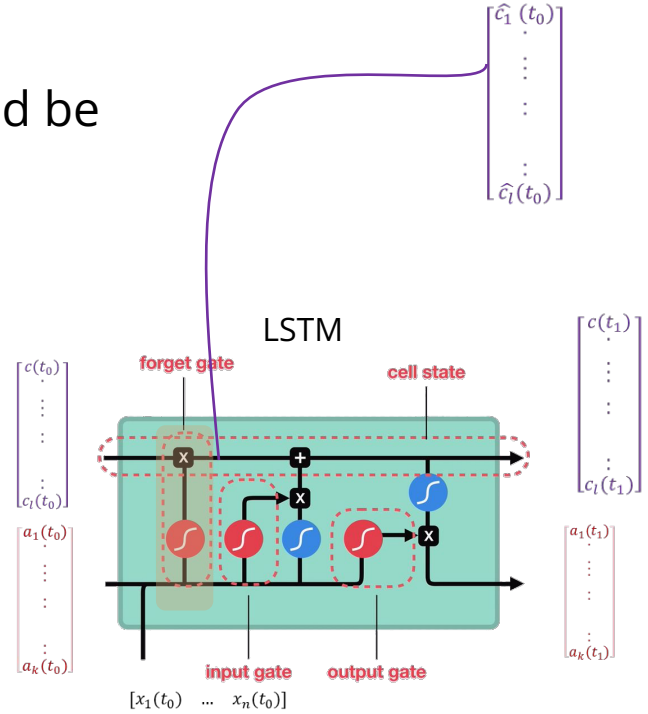
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM "forget" gate

- This gate decides what information should be thrown away (forgotten) or kept.
  - Forget gate, like all gate is a separate layer with individual weights.
  - $$\hat{\mathbf{c}}_{\mathrm{lx1}} = \mathbf{c}_{\mathrm{lx1}} \odot \sigma\left(\mathbf{W}^{forget}_{\mathrm{lx(k+n)}} \cdot \begin{bmatrix} \mathbf{x}_{\mathrm{nx1}}(t_0) \\ \mathbf{a}_{\mathrm{kx1}}(t_0) \end{bmatrix} + \mathbf{b}^{forget}_{\mathrm{lx1}}\right)$$

    $\odot \rightarrow element - wise\ multiplication$
  - The gate outputs are vectoral, not scalar, so the updates on cell states are element-wise.
  - Some aspects of the cell state are forgotten (or kept), not all of them together.

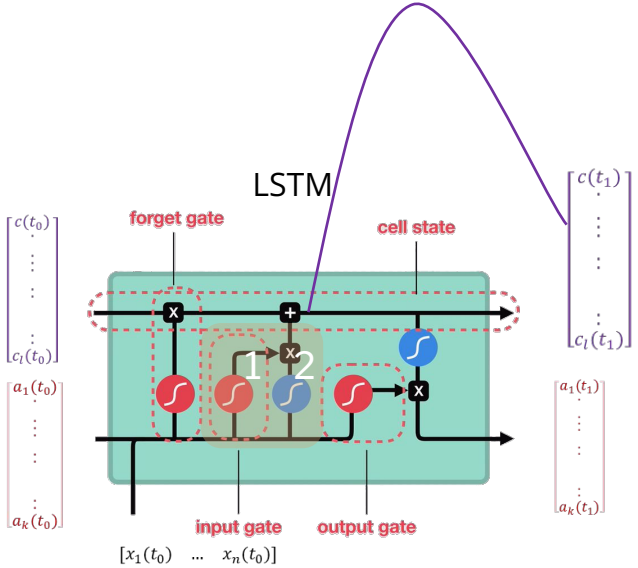


I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM "input" gate

- In order to **update** the cell state, we have the input gate.
- Input gate does two things:
  1. *passes the previous hidden state and current input into a sigmoid function, that decides which values will be updated*
  2. *passes the hidden state and current input into a $tanh$ function to squish values between -1 and 1 to help regulate the network.*
- Then you multiply the $tanh$ output with the $sigmoid$ output. The sigmoid output will decide which information is important to keep from the $tanh$ output.



I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

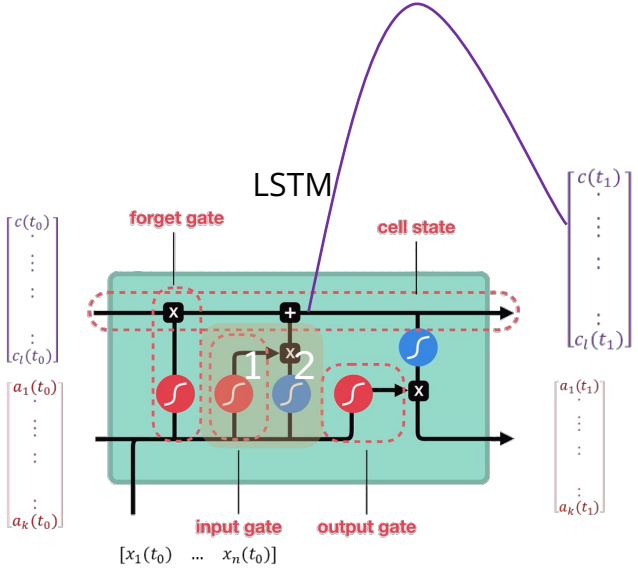
Michael Phi

# LSTM "input" gate

- How does the update take place?

$$\hat{\mathbf{i}}_{l\times 1} = \sigma\left(\mathbf{W}^{inp1}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp1}_{l\times 1}\right)$$

$$\tilde{\mathbf{c}}_{l\times 1} = tanh\left(\mathbf{W}^{inp2}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp2}_{l\times 1}\right)$$



I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM "forget+input" gate

- Two gates combined, provide us the final cell state.

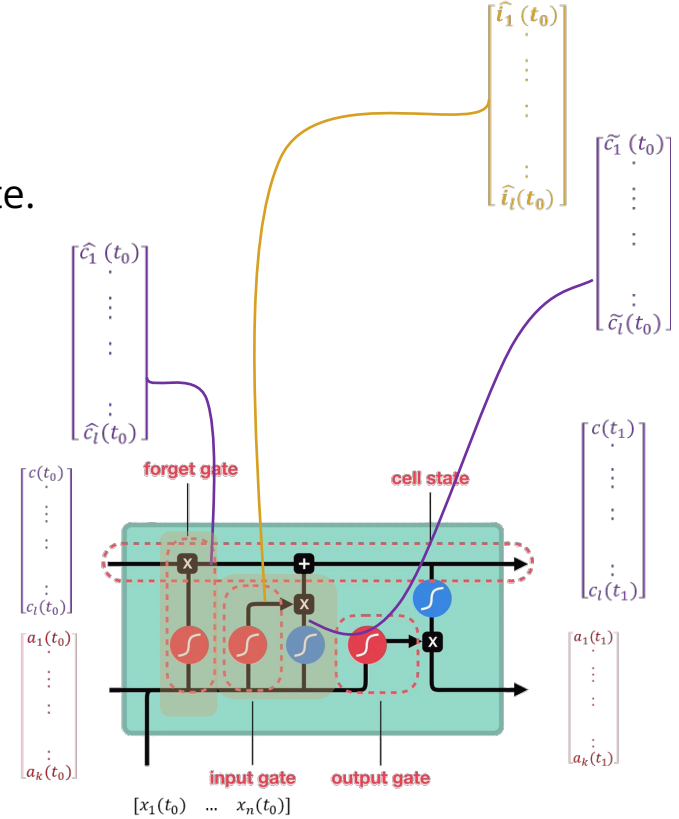
forget

$$\hat{\mathbf{c}}_{l\times 1} = \mathbf{c}_{l\times 1} \odot \sigma\left(\mathbf{W}^{forget}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{forget}_{l\times 1}\right)$$

input

$$\hat{\mathbf{i}}_{l\times 1} = \sigma\left(\mathbf{W}^{inp1}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp1}_{l\times 1}\right)$$

$$\tilde{\mathbf{c}}_{l\times 1} = tanh\left(\mathbf{W}^{inp2}_{l\times(k+n)} \cdot \begin{bmatrix}\mathbf{x}_{n\times 1}(t_0)\\ \mathbf{a}_{k\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp2}_{l\times 1}\right)$$



I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM "forget+input" gate

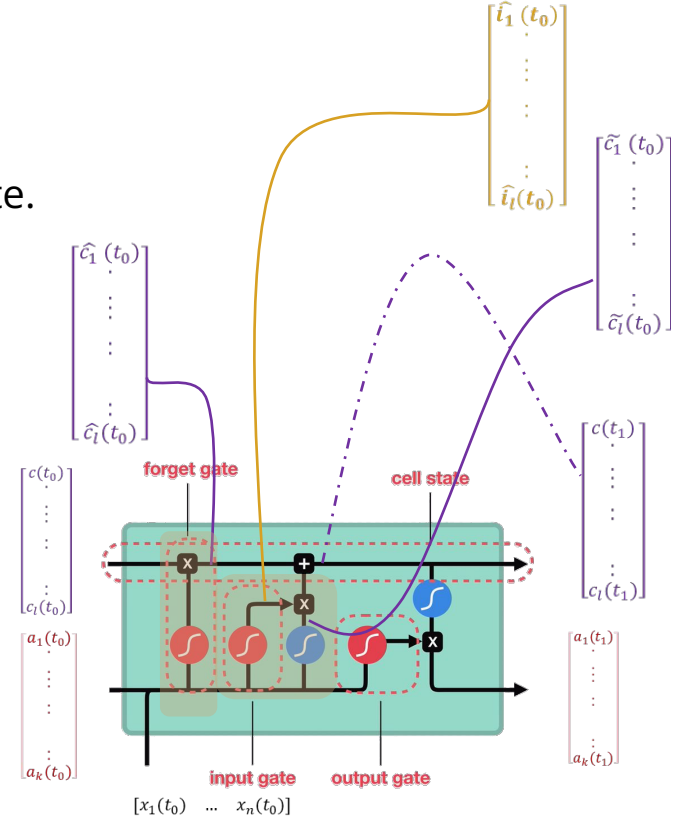
- Two gates combined, provide us the final cell state.

$$\hat{\mathbf{c}}_{\mathrm{l}\times 1} = \mathbf{c}_{\mathrm{l}\times 1} \odot \sigma\left(\mathbf{W}^{forget}_{\mathrm{l}\times(\mathrm{k}+\mathrm{n})} \cdot \begin{bmatrix}\mathbf{x}_{\mathrm{n}\times 1}(t_0)\\ \mathbf{a}_{\mathrm{k}\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{forget}_{\mathrm{l}\times 1}\right)$$

$$\hat{\mathbf{i}}_{\mathrm{l}\times 1} = \sigma\left(\mathbf{W}^{inp1}_{\mathrm{l}\times(\mathrm{k}+\mathrm{n})} \cdot \begin{bmatrix}\mathbf{x}_{\mathrm{n}\times 1}(t_0)\\ \mathbf{a}_{\mathrm{k}\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp1}_{\mathrm{l}\times 1}\right)$$

$$\tilde{\mathbf{c}}_{\mathrm{l}\times 1} = tanh\left(\mathbf{W}^{inp2}_{\mathrm{l}\times(\mathrm{k}+\mathrm{n})} \cdot \begin{bmatrix}\mathbf{x}_{\mathrm{n}\times 1}(t_0)\\ \mathbf{a}_{\mathrm{k}\times 1}(t_0)\end{bmatrix} + \mathbf{b}^{inp2}_{\mathrm{l}\times 1}\right)$$

$$\mathbf{c}(t_{next})_{\mathrm{l}\times 1} = \hat{\mathbf{c}}_{\mathrm{l}\times 1} + \hat{\mathbf{i}}_{\mathrm{l}\times 1} \odot \tilde{\mathbf{c}}_{\mathrm{l}\times 1}$$



I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

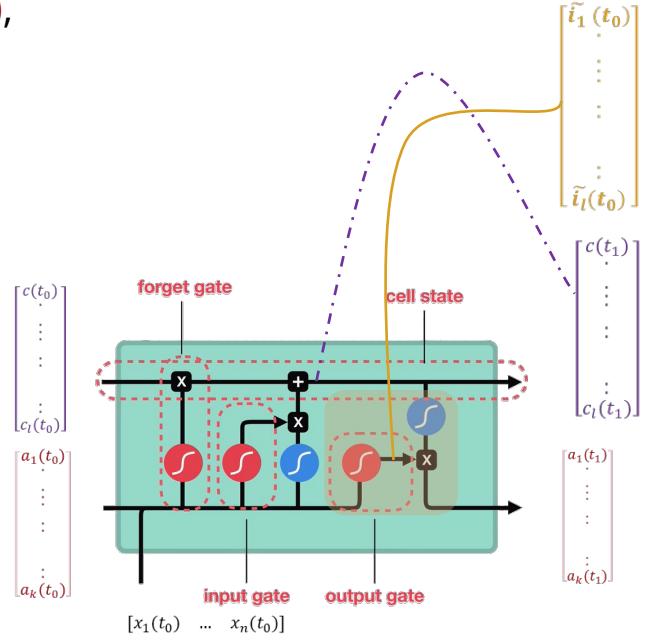
Michael Phi

# LSTM "output" gate

- Finally, in order to obtain the hidden state **a($t_{next}$)**, output gate uses
  - the new cell **c($t_{next}$)**,
  - The previous hidden state **a($t_{prevt}$)**,
  - and the current input **x**

$$\tilde{\mathrm{i}}_{\mathrm{lx1}} = \sigma\left(\mathbf{W}^{out1}_{\mathrm{l\times(k+n)}} \cdot \begin{bmatrix} \mathbf{x}_{\mathrm{n\times1}}(t_0) \\ \mathbf{a}_{\mathrm{k\times1}}(t_0) \end{bmatrix} + \mathbf{b}^{out1}_{\mathrm{l\times1}}\right)$$

$$\mathbf{a}(t_{next})_{\mathrm{kx1}} = \tilde{\mathrm{i}}_{\mathrm{lx1}} \odot tanh\left(\mathbf{W}^{out2}_{\mathrm{k\times l}} \cdot \mathbf{c}(t_{next})_{\mathrm{lx1}} + \mathbf{b}^{out2}_{\mathrm{k\times1}}\right)$$



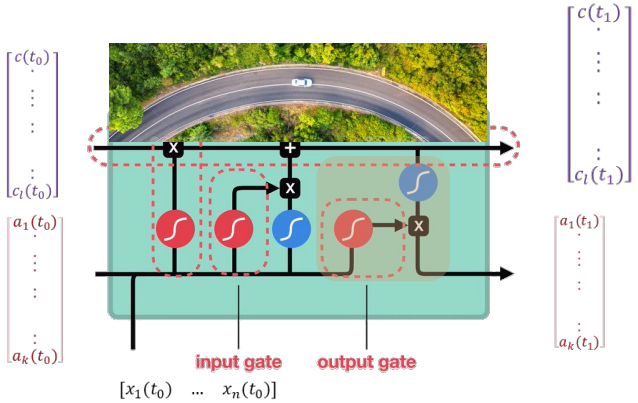
I will be using the figures from this guy's very famous GRU/LSTM blog. You must read it. It also has a YouTube video.

Michael Phi

# LSTM: observations

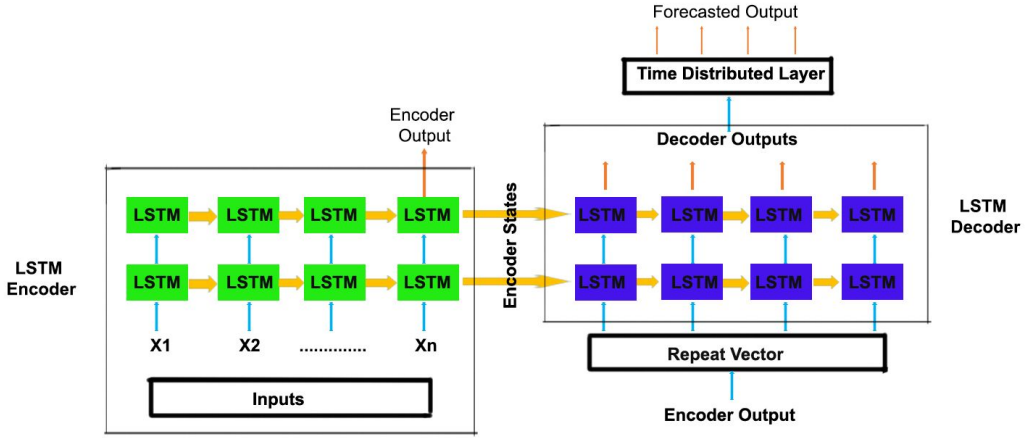
- During backprop, for the cell state, there is an open highway. That's why we say, cell state can learn long term dependencies (i.e. Long term memory)

- During backprop, the hidden state is no different than a hidden state of a vanilla RNN. That's why it is prone to vanilla RNN problems, such as vanishing gradients etc.
  - It is for short term memory.
- **LSTM is long (the cell state), short (the hidden state), term memory.**



# Time Series Forecasting with RNNs

- Recurrent Neural Networks (RNNs) and their specialized variant, Long Short-Term Memory (LSTM) networks, have emerged as popular approaches for time series forecasting, enabling the modeling of temporal dependencies and capturing long-term patterns in sequential data.

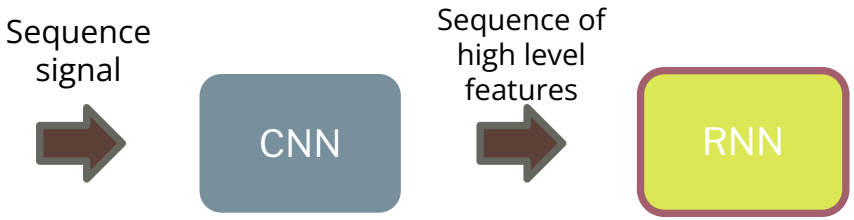


# Convolution + Recurrency

- Convolution allows us to learn spatial dependencies.
- Recurrency allow us to learn sequential (i.e. mostly time) dependencies.
- So can we use CNN+RNN together for spatio-temporal signals?

# CNN+RNN

- Input with spatial structure (like images, auido, etc.) cannot be modeled easily with the standard Vanilla LSTM.
- Recurrent and Convolutional Neural Networks can be combined in different ways. (e.g. Recurrent Convolutional Neural Networks – RCNN)
- What we are talking about (today) is «**CNN + (and afterwards) RNN**»
- The idea is to use the high level features obtained from a CNN



# Time Series Forecasting

## with CNN+RNNs

- Recurrent Neural Networks (RNNs) and their specialized variant, Long Short-Term Memory (LSTM) networks, have emerged as popular approaches for time series forecasting, enabling the modeling of temporal dependencies and capturing long-term patterns in sequential data.

IEEE Access

Date of publication 31 May 2021

*Digital Object Identifier https://doi.org/10.1109/ACCESS.2021.3085085*

**Evaluation of deep learning models for multi-step ahead time series prediction**

**ROHITASH CHANDRA[1] (Senior Member, IEEE), SHAURYA GOYAL [2], AND RISHABH GUPTA.[3]**

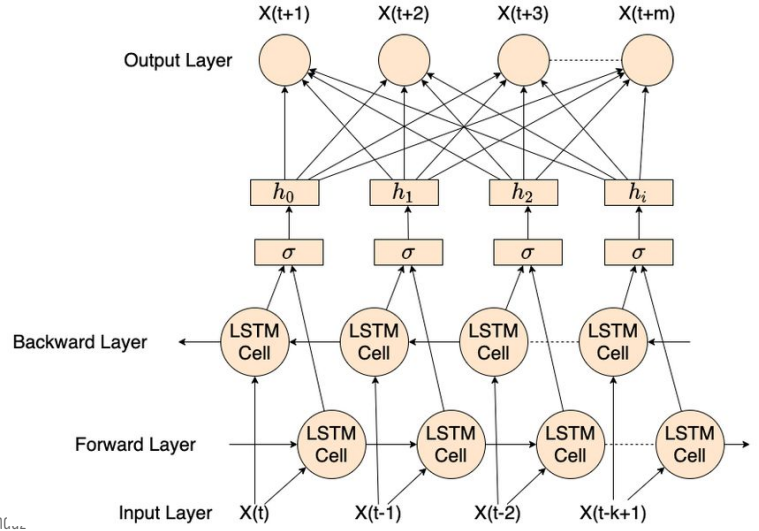
[1]School of Mathematics and Statistics, University of New South Wales, Sydney, NSW 2052, Australia

[2] Department of Mathematics, Indian Institute of Technology, Delhi, 110016 , India

[3]Department of Geology and Geophysics, Indian Institute of Technology,Kharagpur, 721302 , India

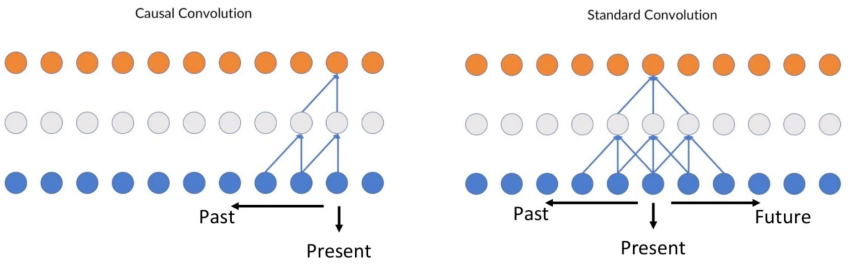
Corresponding author: R. Chandra (e-mail: rohitash.chandra@unsw.edu.au)

**ABSTRACT** Time series prediction with neural networks has been the focus of much research in the past few decades. Given the recent deep learning revolution, there has been much attention in using deep learning models for time series prediction, and hence it is important to evaluate their strengths and weaknesses. In this paper, we present an evaluation study that compares the performance of deep learning models for multi-step ahead time series prediction. The deep learning methods comprise simple recurrent neural networks, long short-term memory (LSTM) networks, bidirectional LSTM networks, encoder-decoder LSTM networks, and convolutional neural networks. We provide a further comparison with simple neural networks that use stochastic gradient descent and adaptive moment estimation (Adam) for training. We focus on univariate time series for multi-step-ahead prediction from benchmark time-series datasets and provide a further comparison of the results with related methods from the literature. The results show that the bidirectional and encoder-decoder LSTM network provides the best performance in accuracy for the given time series problems.



# Temporal Causal Networks

## CNNs for Sequential Data




- TCNs are deep learning models that leverage the concept of causality to capture temporal dependencies in time series data.
- Causality refers to the cause-and-effect relationship between events. In time series data, causality implies that the value at a particular time step can be influenced by past values, but not future values.
- TCNs manipulate Convolutional Neural Networks (CNNs) by incorporating dilated convolutions and causal padding to efficiently capture long-range temporal dependencies in time series data.

# Temporal Causal Networks

## CNNs for Sequential Data

- TCNs can be applied to accelerometer data in earthquake engineering by leveraging their ability to model temporal dependencies and capture seismic patterns, enabling accurate analysis and prediction of ground motion and seismic events.

IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, VOL. 58, NO. 11, NOVEMBER 2020 8211

### Bayesian-Deep-Learning Estimation of Earthquake Location From Single-Station Observations

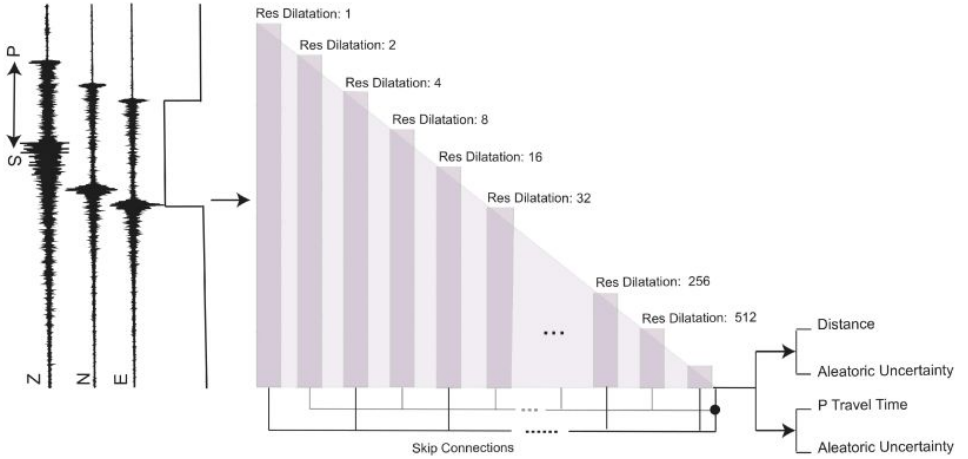
S. Mostafa Mousavi and Gregory C. Beroza

***Abstract*— We present a deep-learning method for a single-station earthquake location, which we approach as a regression problem using two separate Bayesian neural networks. We use a multitask temporal convolutional neural network to learn epicentral distance and P travel time from 1-min seismograms. The network estimates epicentral distance and P travel time with mean errors of 0.23 km and 0.03 s and standard deviations of 5.42 km and 0.66 s, respectively, along with their epistemic and aleatory uncertainties. We design a separate multi-input network using standard convolutional layers to estimate the back-azimuth angle and its epistemic uncertainty. This network estimates the direction from which seismic waves arrive at the station with a mean error of 1°. Using this information, we estimate the epicenter, origin time, and depth along with their confidence intervals. We use a global data set of earthquake signals recorded within 1° (∼112 km) from the event to build the model and demonstrate its performance. Our model can predict epicenter, origin time, and depth with mean errors of 7.3 km, 0.4 s, and 6.7 km, respectively, at different locations around the world. Our approach can be used for fast earthquake source characterization with a limited number of observations and also for estimating the location of earthquakes that are sparsely recorded—either because they are small or because stations are widely separated.**

***Index Terms*— AI, Bayesian neural networks, earthquakes, source characterization.**

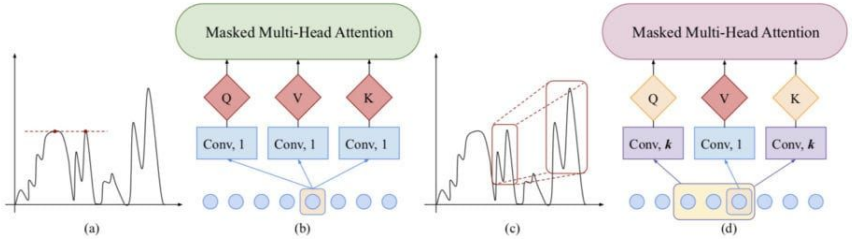
in Oklahoma. Lomax *et al.* [18] expanded this approach by classifying seismic waveforms into a larger number of classes, including event/noise (1 class), station–event distance (50 classes), station–event azimuth (36 classes, each 10°), event magnitude (20 classes), and event depth (20 classes); however, their model did not generalize well and suffered from high error rates. On the other hand, multistation approaches (e.g., [19]) result in better performance by learning the move out patterns for specific station configuration at a local region.

Neither these, nor other, neural networks applied to earthquake data quantify uncertainty in their output. Machine learning can be thought of as inferring plausible models that explain data and can be used to make predictions about unseen data. Uncertainty plays a key role in that process of quantifying the reliability of those predictions. Data can be consistent with different models, and the question of what model is appropriate based on such data is uncertain. Predictions using future data are also uncertain [20]. Typical deep-learning models do not capture uncertainties in the output. In regression models, the output is a single vector that regresses to the mean of the data, but in classification models, the output probability



# Time Series Forecasting

## attention-based




- Attention mechanisms provide a mechanism for the model to focus on different parts of the input time series when making predictions.
- Attention-based models address the challenge of handling long sequences by selectively attending to relevant time steps and suppressing noise or irrelevant information, enabling more effective modeling of long-term dependencies in time series data.
- Attention weights assigned to different time steps provide interpretability and explainability, highlighting the influential time steps in the forecasting process.

# DA-Net

DA-Net: Dual-attention network for multivariate time series classification

Rongjun Chen [a], Xuanhui Yan [a,*], Shiping Wang [b], Guobao Xiao [c]

[a] Fujian Internet of Things Laboratory for Environmental Monitoring, School of Computer and Cyberspace Security, Fujian Normal University, Fuzhou, Fujian, China
[b] College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
[c] Electronic Information and Control Engineering Research Center of Fujian, College of Computer and Control Engineering, Minjiang University, Fuzhou 350108, China

ARTICLE INFO

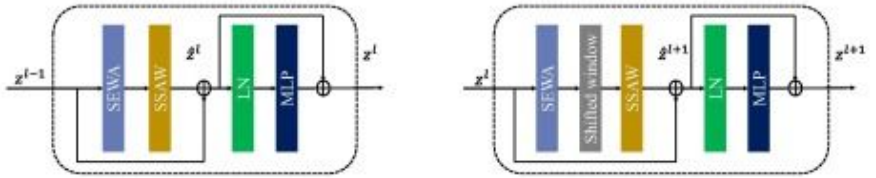
*Article history:*
Received 23 February 2022
Received in revised form 27 July 2022
Accepted 30 July 2022
Available online 8 August 2022

*Keywords:*
Multivariate time series classification
Deep learning
Attention
UEA datasets

ABSTRACT

Multivariate time series classification is one of the increasingly important issues in machine learning. Existing methods focus on establishing the global long-range dependencies or discovering the local critical sequence fragments. However, they often ignore the combined information from both global and local features. In this paper, we propose a novel network (called DA-Net) based on dual attention to mine the local–global features for multivariate time series classification. Specifically, DA-Net consists of two distinctive layers, i.e., the Squeeze-Excitation Window Attention (SEWA) layer and the Sparse Self-Attention within Windows (SSAW) layer. For the SEWA layer, we capture the local window-wise information by explicitly establishing window dependencies to prioritize critical windows. For the SSAW layer, we preserve rich activate scores with less computation to widen the window scope for capturing global long-range dependencies. Based on the two elaborated layers, DA-Net can mine critical local sequence fragments in the process of establishing global long-range dependencies. The experimental results show that DA-Net is able to achieve competing performance with state-of-the-art approaches on the multivariate time series classification.

- Dual-Stage Attention Mechanism:
  - DA-Net employs a dual-stage attention mechanism that consists of a global attention module and a local attention module.
  - The global attention module captures long-term dependencies by attending to the entire input time series. (LSTM like?)
  - The local attention module focuses on short-term dependencies by attending to a subset of neighboring time steps.




(a) The first module of dual-attention block (b) The second module of dual-attention block

# DA-Net

DA-Net: Dual-attention network for multivariate time series classification

Rongjun Chen [a], Xuanhui Yan [a,*], Shiping Wang [b], Guobao Xiao [c]

[a] Fujian Internet of Things Laboratory for Environmental Monitoring, School of Computer and Cyberspace Security, Fujian Normal University, Fuzhou, Fujian, China
[b] College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China
[c] Electronic Information and Control Engineering Research Center of Fujian, College of Computer and Control Engineering, Minjiang University, Fuzhou 350108, China

ARTICLE INFO

*Article history:*
Received 23 February 2022
Received in revised form 27 July 2022
Accepted 30 July 2022
Available online 8 August 2022

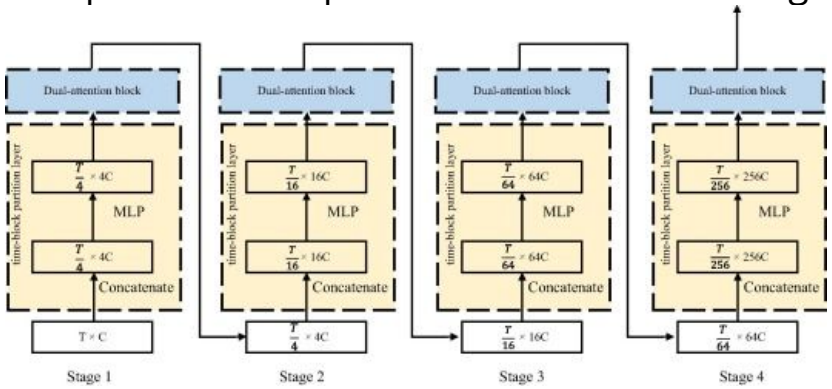
*Keywords:*
Multivariate time series classification
Deep learning
Attention
UEA datasets

ABSTRACT

Multivariate time series classification is one of the increasingly important issues in machine learning. Existing methods focus on establishing the global long-range dependencies or discovering the local critical sequence fragments. However, they often ignore the combined information from both global and local features. In this paper, we propose a novel network (called DA-Net) based on dual attention to mine the local–global features for multivariate time series classification. Specifically, DA-Net consists of two distinctive layers, i.e., the Squeeze-Excitation Window Attention (SEWA) layer and the Sparse Self-Attention within Windows (SSAW) layer. For the SEWA layer, we capture the local window-wise information by explicitly establishing window dependencies to prioritize critical windows. For the SSAW layer, we preserve rich activate scores with less computation to widen the window scope for capturing global long-range dependencies. Based on the two elaborated layers, DA-Net can mine critical local sequence fragments in the process of establishing global long-range dependencies. The experimental results show that DA-Net is able to achieve competing performance with state-of-the-art approaches on the multivariate time series classification.

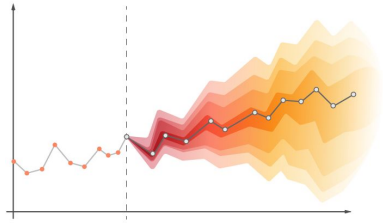


- Forecasting with Multiple Attention Views:
  - DA-Net generates multiple attention views by applying different combinations of the global and local attention modules.
  - This allows the model to capture various aspects of the temporal dependencies, providing a more comprehensive representation for forecasting.

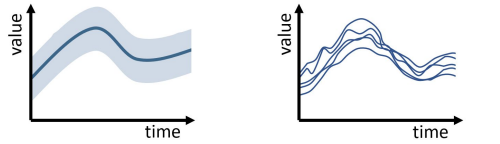


# Time Series Concepts

**Forecasting**: predicting the next sequential element

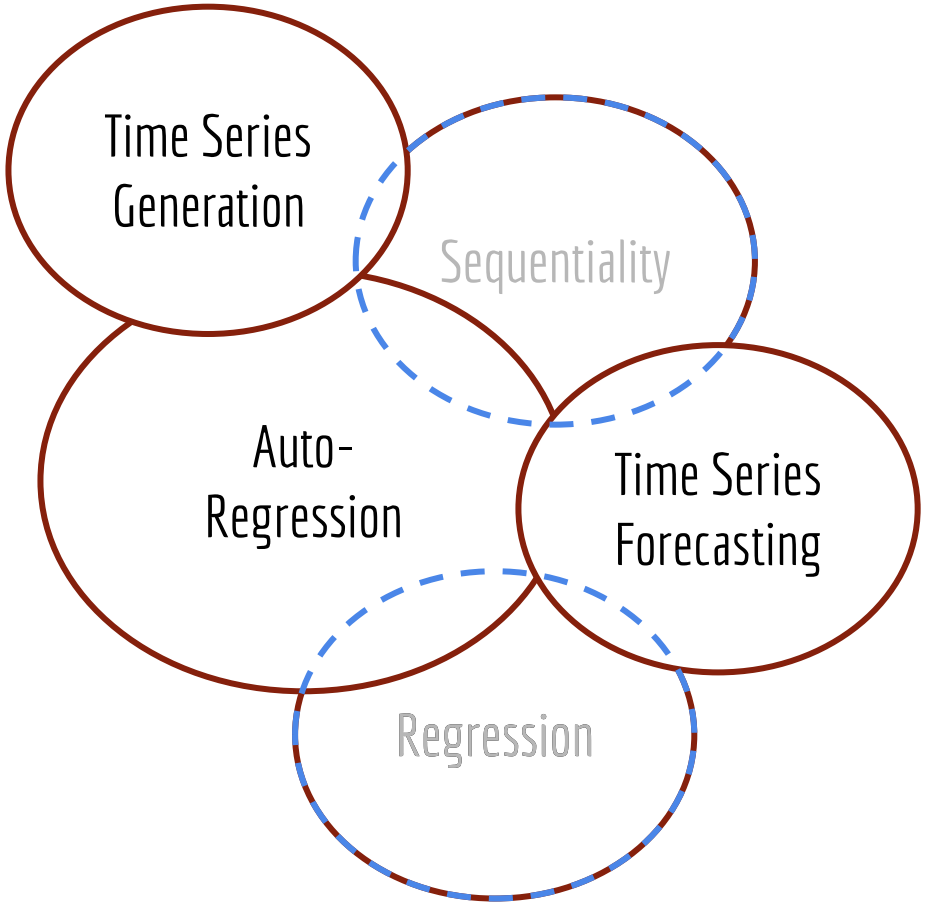


**Generation**: Creating a set a variable/vector out of a distribution



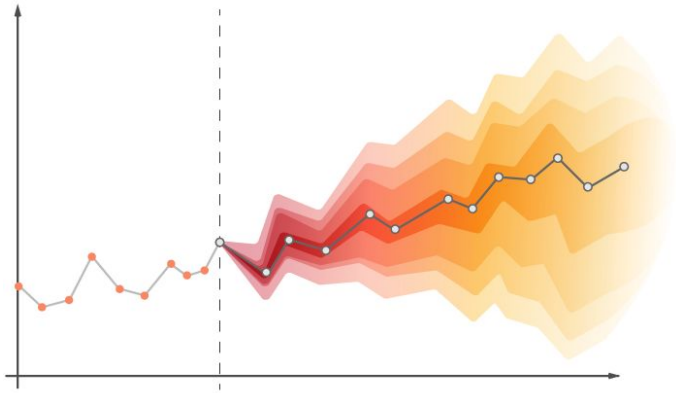

Mean and standard deviation of an uncertain time series

Set of representative scenarios for the time series

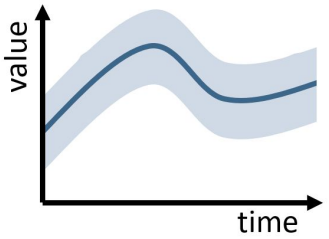


# Forecasting vs Generation

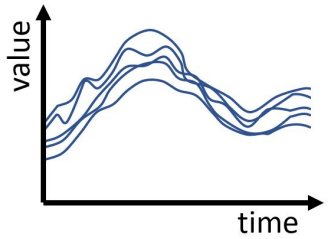
The goal is to predict future values or patterns based on past observations, aiming to capture the underlying trends, seasonality, and dynamics in the data.



The objective is to generate new sequences of data that resemble the characteristics and structure of the training time series, without specific emphasis on future prediction.



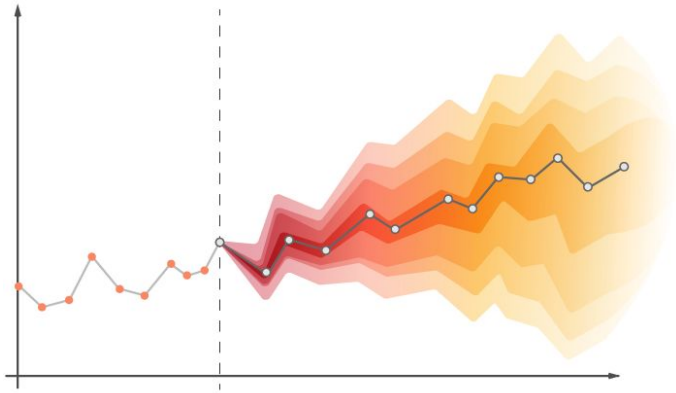

Mean and standard deviation of an uncertain time series



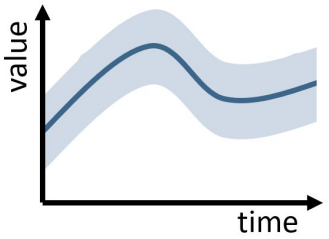

Set of representative scenarios for the time series

# Forecasting vs Generation

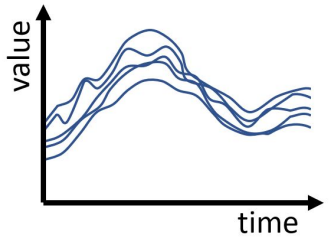
The input typically consists of historical time series data, and the output is one or more future values or patterns.



The input is often random noise or a seed, and the output is a new sequence of data generated by the model.




Mean and standard deviation of an uncertain time series

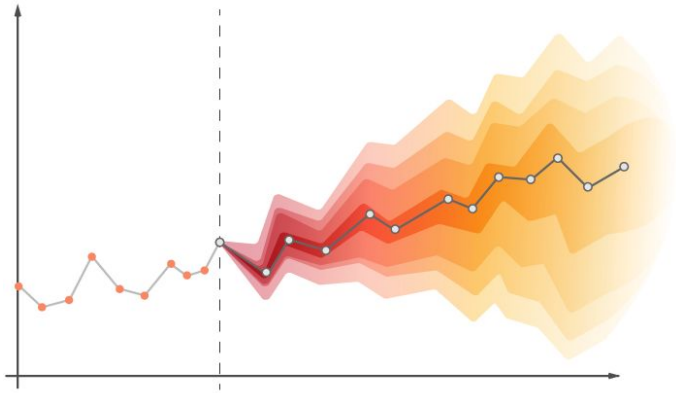



Set of representative scenarios for the time series

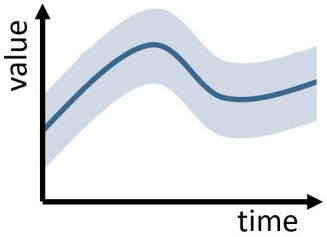
$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

# Forecasting vs Generation

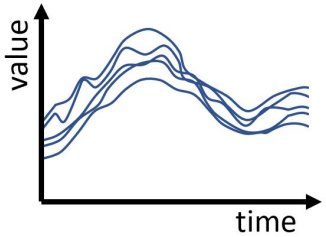
Deep learning models, such as (RNNs), or transformers, are trained to learn the patterns and dependencies in the historical data to make accurate predictions.



RNNs etc can be adapted for Generation but Deep generative models, GANs or VAEs, are prefered



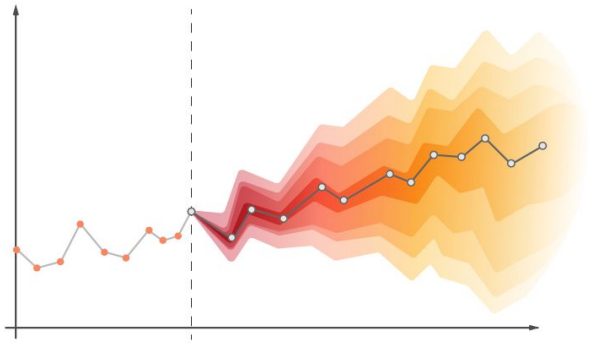

Mean and standard deviation of an uncertain time series




Set of representative scenarios for the time series

# Autoregressive Time Series Generation

- Autoregressive models generate new time series data by modeling the conditional distribution of each time step given previous observations.
  - Autoregressive models assume sequential dependence and generate data one step at a time, typically using probabilistic models such as autoregressive moving average (ARMA) or autoregressive integrated moving average (ARIMA).
  - These models can be adapted to generate new data by iteratively predicting future values based on the previously generated values. **But** this is just forecasting.

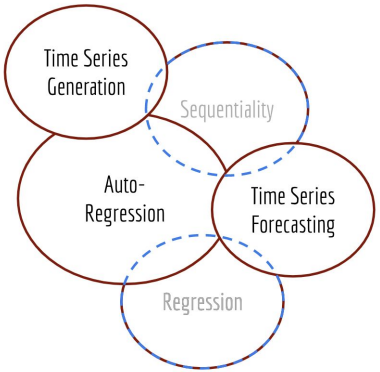


$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

*probabilistic* and *autoregressive*

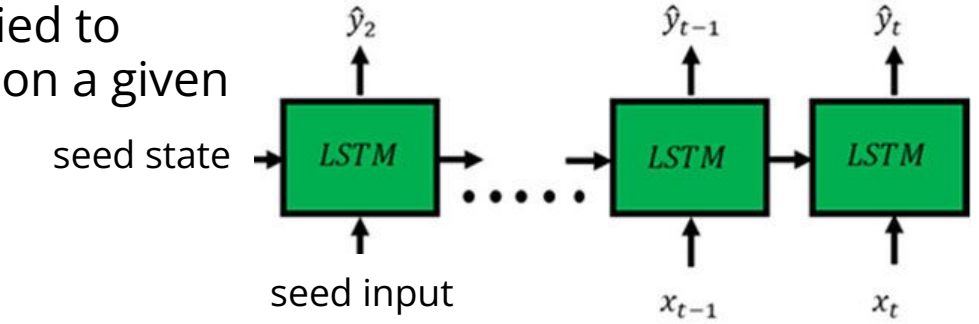
# Autoregressive Time Series Generation

- Autoregression focuses on modeling the conditional probability of a single time step given the previous time steps in a sequence. It captures temporal dependencies within a time series by predicting each element based on the preceding elements.
- Sequential generation using autoregression involves generating one time step at a time, conditioning on the previously generated steps. By iteratively generating each step, the joint probability distribution of the entire sequence is implicitly captured.



# RNNs for Autoregression

- Unlike forecasting with RNNs, instead of feeding historical data, data can be generated with a generative model.
  - We exclude simulation based methods in this context.
- By repeatedly generating new values based on the previously generated values, the RNN can create a time signal with an autoregressive approach. The generated signal will depend on the learned patterns and the style of the training data.
- RNNs have been successfully applied to tasks like music generation based on a given seed sequence.



# Not only RNNs for Autoregression

WAVENET: A GENERATIVE MODEL FOR RAW AUDIO

Aäron van den Oord | Sander Dieleman | Heiga Zen†
Karen Simonyan | Oriol Vinyals | Alex Graves
Nal Kalchbrenner | Andrew Senior | Koray Kavukcuoglu

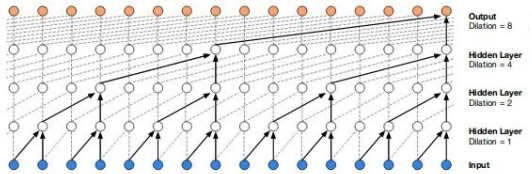
{avdnoord, sedielem, heigazen, simonyan, vinyals, gravesa, nalk, andrewsenior, korayk}@google.com
Google DeepMind, London, UK
† Google, London, UK

- WaveNet is a fully probabilistic autoregressive deep neural network-based model used for raw audio generation and was first introduced by DeepMind in 2016.
- Generative model operating directly on the raw waveform

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

WaveNet model is *probabilistic* and *autoregressive*

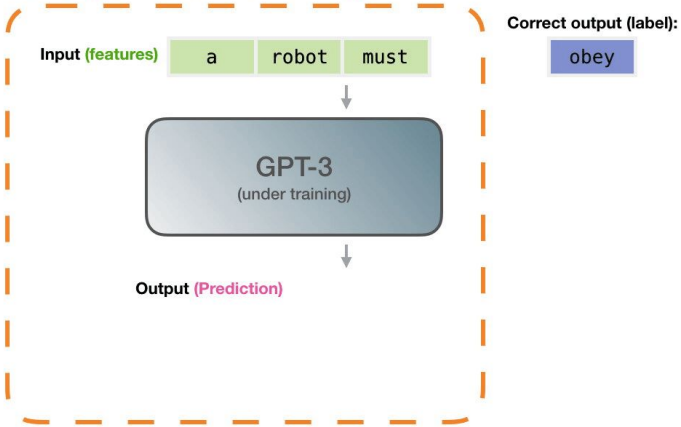
- Based on TCN, WaveNet This is like a “language model” for audio samples




Dilated Convolution (Receptive field = 16 with kernel size 2)[1]

# chatGPT (yes, him)

- At every “timestep” GPTs generate one word at a time and that word will be added to input sequence for generating the next word
- So they are autoregressive models.
- Model is predicting one token at a time.
- So for complex sequential problems, such as LLMs, autoregressive prediction Is a fundamental way of generation.
- This cannot be said for non-sequential Signals (images, etc).



# What will we do next week?

- Starting with next week…
  - *latent space, AEs, VAEs, ELBO, etc…*
- Following weeks
  - *Normalizing Flows, etc.*

# Additional Reading & References

- https://www.youtube.com/watch?v=whmI0WHuIK4 regression vs time-series problems
- https://www.youtube.com/watch?v=Mc6sBAUdDP4
- https://www.youtube.com/watch?v=AN0a58F6cxA
- https://www.youtube.com/watch?v=iyE0k8KCRUw (lecture)
- https://otexts.com/fpp2/
- https://towardsdatascience.com/lets-forecast-your-time-series-using-classical-approaches-f84eb982212c#:~:text=In%20time%20series%2C%20the%20exogenous,weighted%20input%20to%20the%20model.
- https://www.nvidia.com/en-us/glossary/data-science/xgboost/#:~:text=XGBoost%2C%20which%20stands%20for%20Extreme,%2C%20classification%2C%20and%20ranking%20problems.
- Gradient Boost: https://www.youtube.com/watch?v=3CC4N4z3GJc
- XGBoost: https://www.youtube.com/watch?v=0tD8wVaFm6E
- https://arxiv.org/pdf/2103.14250.pdf
- https://www.topbots.com/attention-for-time-series-forecasting-and-classification/
- https://www.youtube.com/watch?v=rT77lBfAZm4
- https://www.inf.ed.ac.uk/teaching/courses/asr/2016-17/asr17-wavenet.pdf
- https://towardsdatascience.com/auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173
- https://medium.com/nerd-for-tech/gpt3-and-chat-gpt-detailed-architecture-study-deep-nlp-horse-db3af9de8a5d