Introduction to time series modeling with artificial neural networks using Tensorflow and Keras

Rungployphan (Om) Kieokaew
CNES Postdoc fellow

Institut de Recherche en Astrophysique et Planétologie, Toulouse, France.

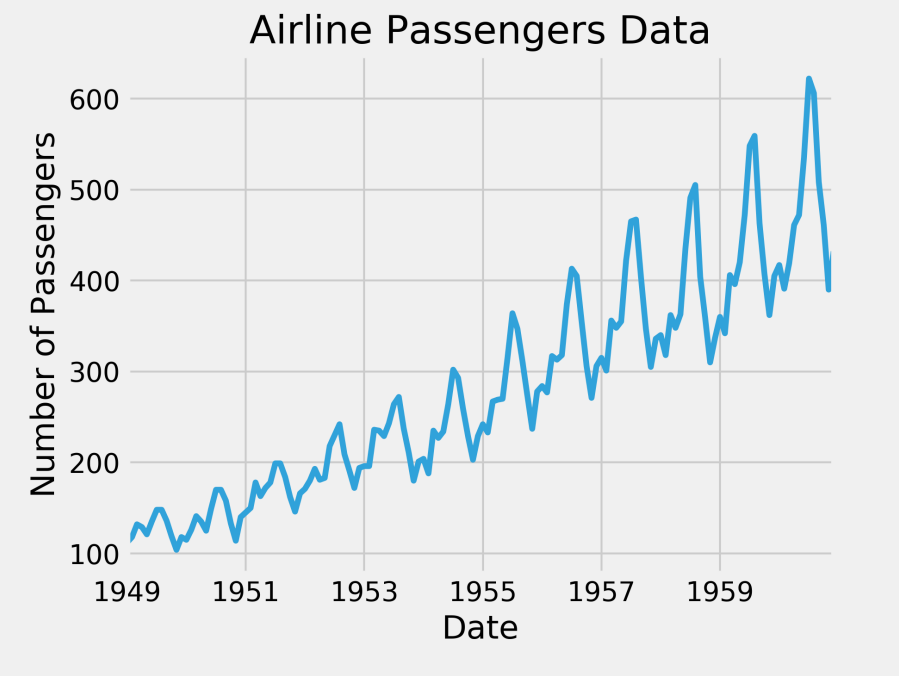
Journée IA4REG, Observatoire Midi-Pyrénées

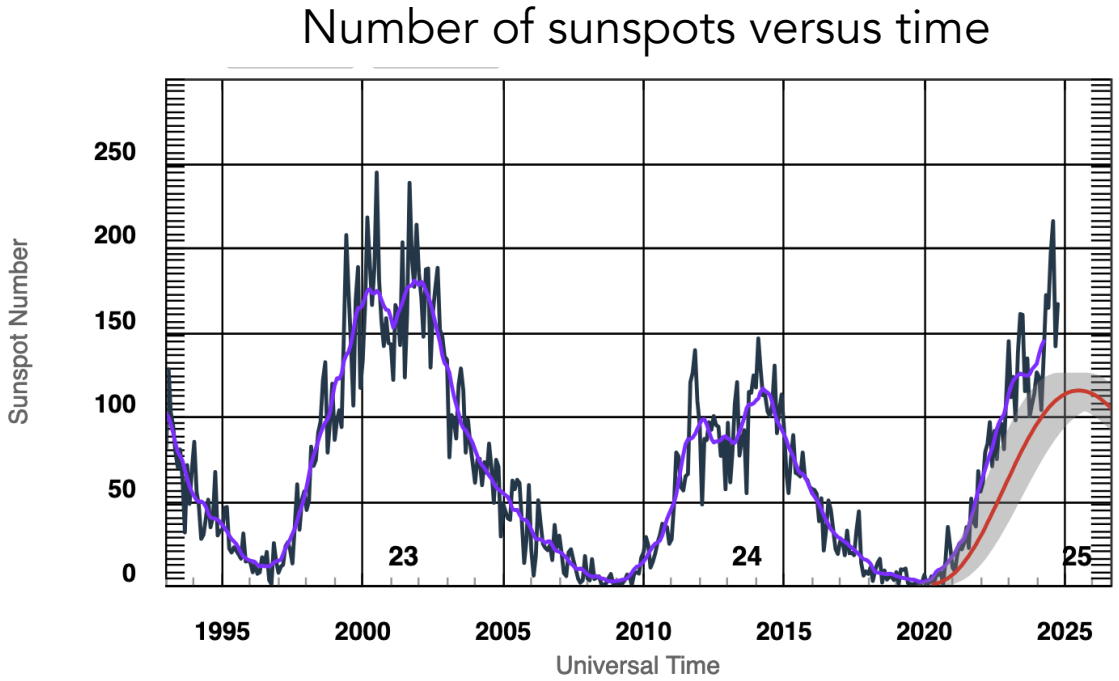
Outline

1. **Introduction to regression modeling of time series**
2. Neural Networks for time series forecasting
3. Introduction to Tensorflow and Keras, and how to build a complete neural network pipeline
4. Practical session on time series forecasting

Time series data

Observations or measurements taken sequentially over time. Characterized by temporal order. Presence of patterns, seasonal variations, over a period.





More examples: the crop yield in tons in France each year, temperature records, Electrocardiograph, stock price, unemployment rate, etc.

Components of time series

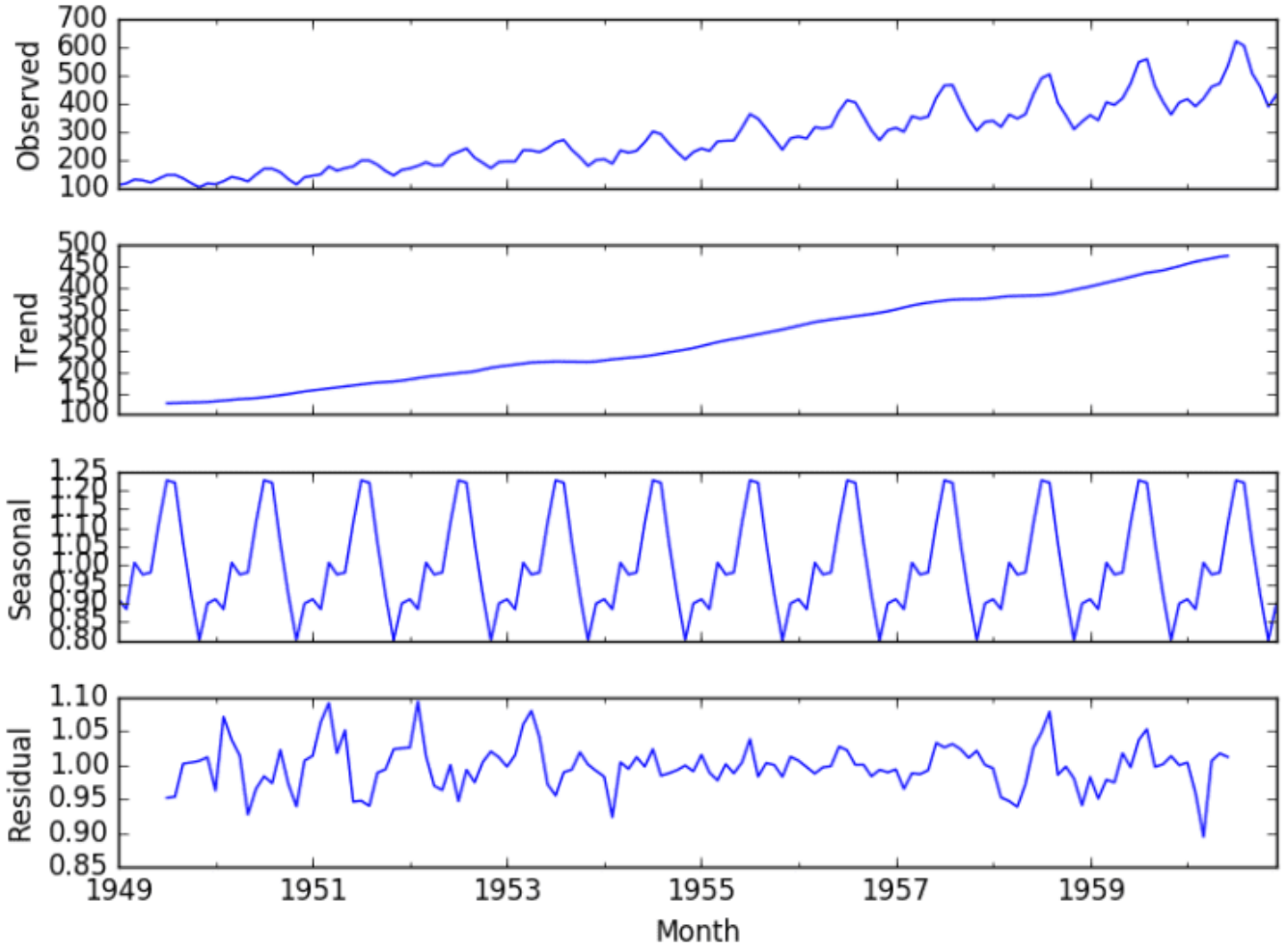
- **Level**: baseline value - straight line
- **Trend**: increasing/decreasing behavior
- **Seasonality**: repeating patterns
- **Noise**: optional variability unexplained by the model

All time series have a level, most have noise, and the trend and seasonality are optional.

Bitcoin price over the past month

U.S. Election

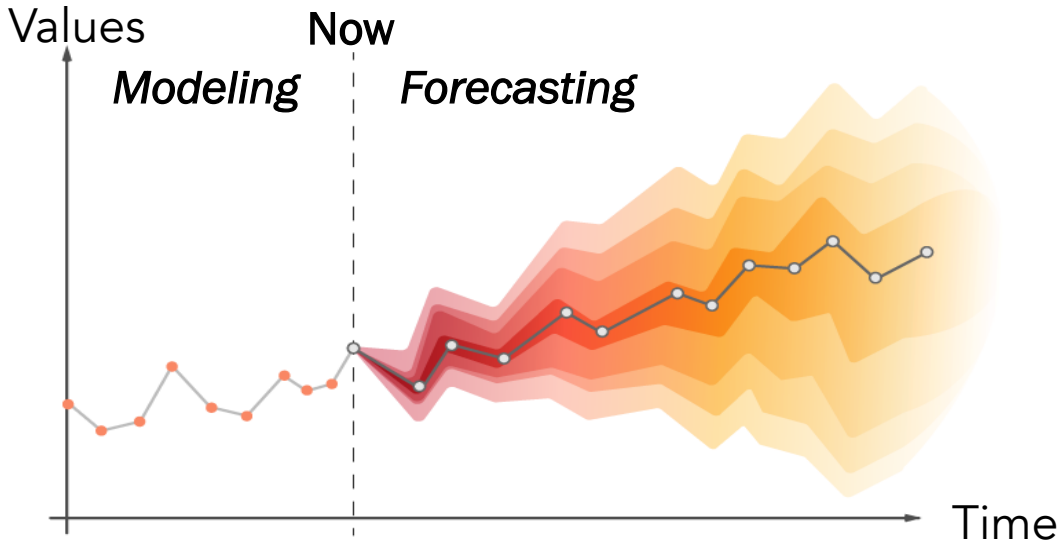
Example: decomposition of airline passenger data



Time series modeling and forecasting

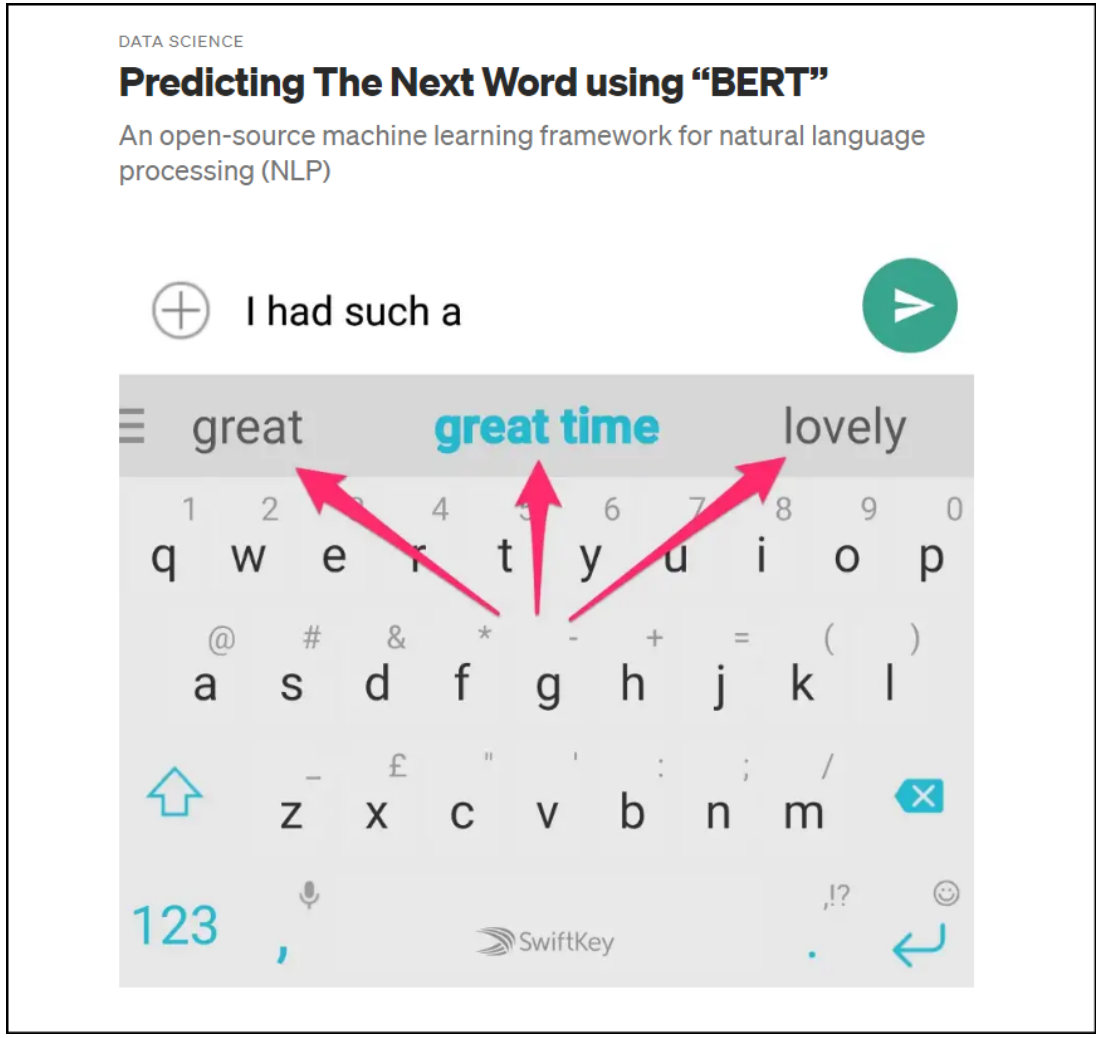
Modeling: being able to explain the **past & present variation** of the time series data.

Forecasting: being able to predict the **future variation** (t+1, ...) of the time series data.



- Forecasting the monthly precipitation in Toulouse
- Forecasting the closing price of a stock each day
- Forecasting the number of train passengers daily

More example: text completion (sequence)



Types of time series forecasting

Univariate forecasting

We can forecast a target value in the **time series** based on a single feature that is **univariate.**

Time	Temperature
06:00 am	15° C
07:00 am	15.5° C
08:00 am	16° C
09:00 am	17° C
10:00 am	18° C
11:00 am	?

For example, the analytical solution may be written as

$$Y_{t+1} = \beta_0 + \beta_1 Y_t + \beta_2 Y_{t-1} + \beta_3 Y_{t-2} + \cdots$$

(autoregression)

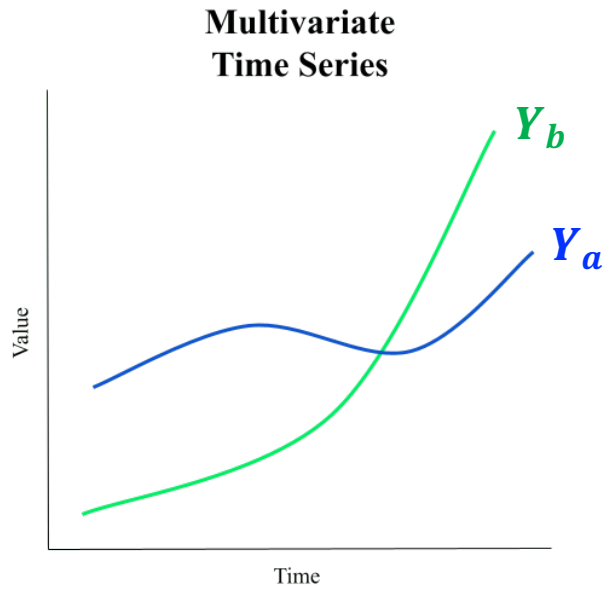
Multivariate forecasting

Forecasting multiple variables simultaneously, e.g.,

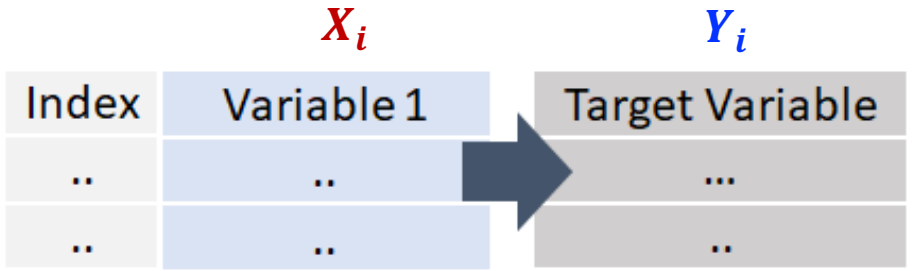
$$Y_{a,t+1} = \alpha_0 + \alpha_0 Y_{a,t}$$
$$Y_{b,t+1} = \beta_0 + \beta_1 Y_{b,t}$$

Forecasting a target variable from multiple other variables.

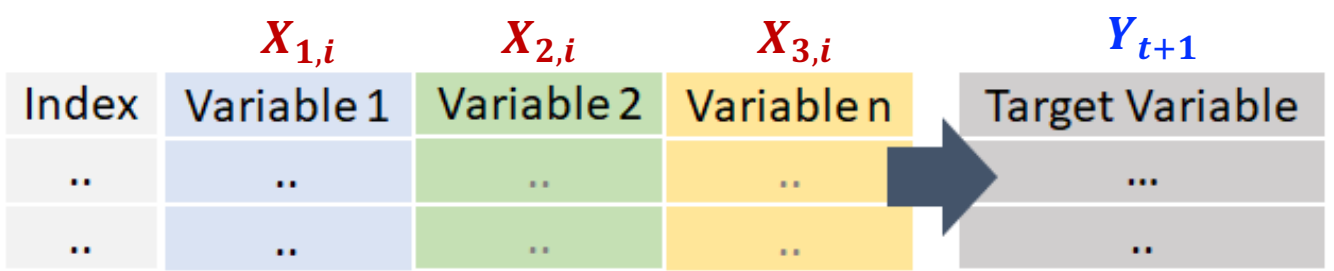
Ex: forecasting temperature from pressure, humidity, …



Forecasting based on other parameters (simultaneous or past sequences)



$$\beta_0 + \beta_1 X_{t+1} = Y_{t+1}$$

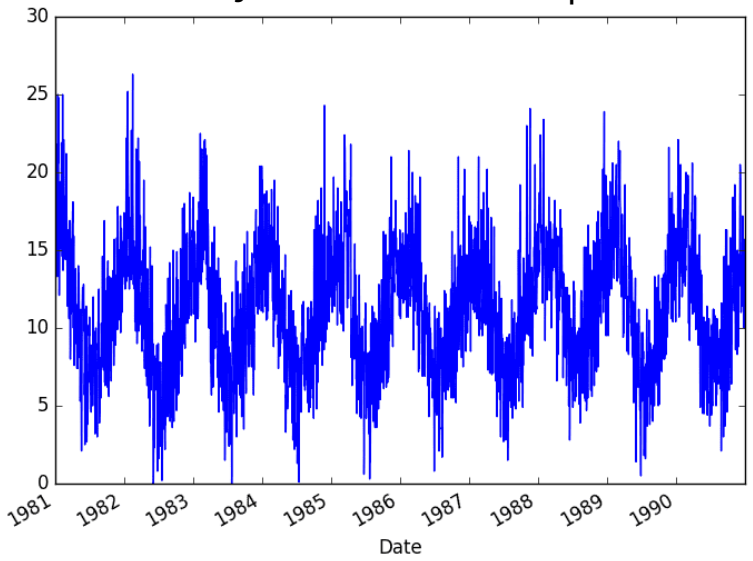


$$\beta_1 X_{1,t+1} + \beta_2 X_{2,t+1} + \beta_3 X_{3,t+1} + \ldots = Y_{t+1}$$

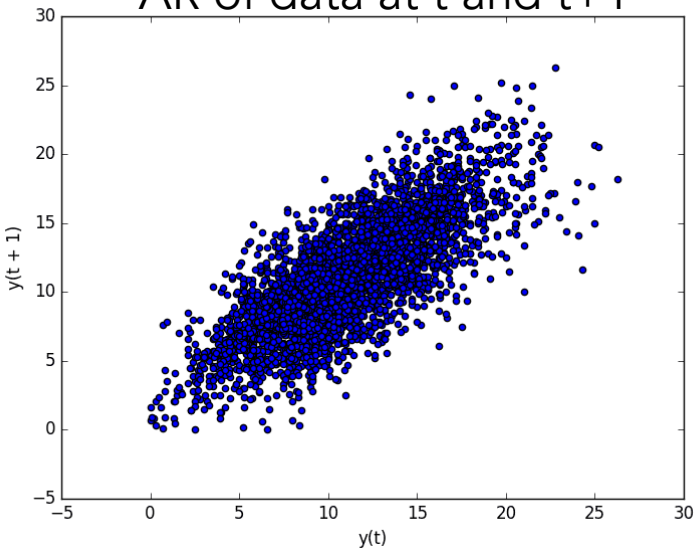
Traditional approaches for time series forecasting

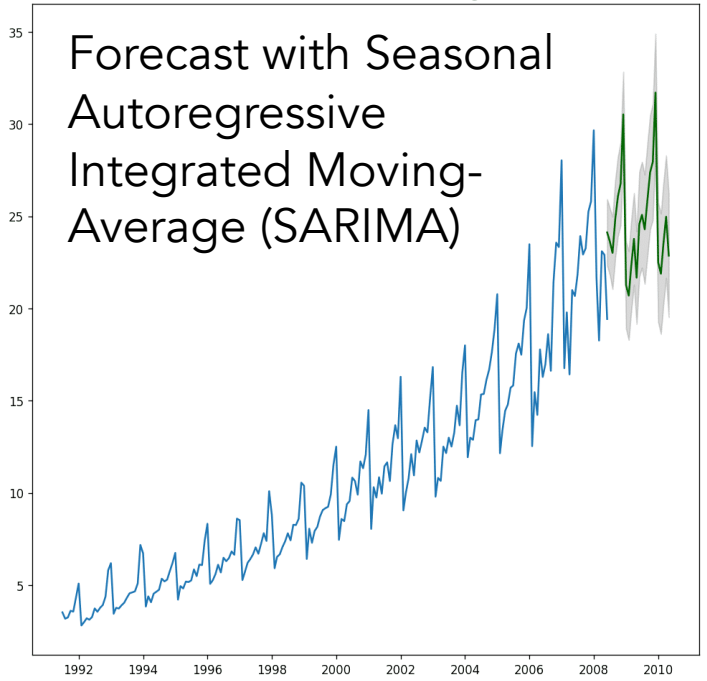
- **Classical / Statistical Models** — Autoregression (autocorrelation), Moving Averages, ARIMA, SARIMA, etc.
- **Analytical and Physical Models** – Domain knowledge (e.g., physical laws, analytical solutions, etc.)

Ex: daily minimum temperature



AR of data at t and t+1

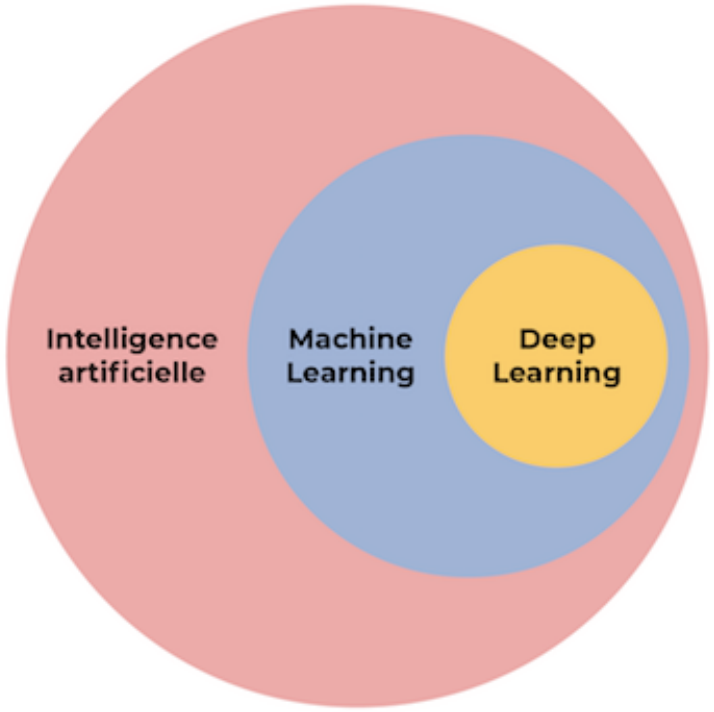




$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + .. + \beta_p Y_{t-p} + \epsilon_1$$

Suitable for univariate problems, small datasets, and when you have good domain expertise

Machine learning for time series forecasting



- **Machine Learning** — Linear Regression, XGBoost, Random Forest, or any ML model with reduction methods
- **Deep Learning** — Recurrent Neural Networks, Long Short-Term Memory Neural Networks, Transformers, etc.

Suitable for multivariate problems, large datasets, and when you have some ideas on how to forecast the data (despite incomplete).
You may need parameterization for input features.
However, if there is no pattern in the data, nothing will work …

Classic workflow:

Training an AI is resource intensive

- **Often resource intensive** (computing time, electricity, water consumption, electronics, etc.)
- **Need good reasons** to use ML and deep learning, e.g.,
  - the classical approaches do not work,
  - the analytical solutions fail,
  - your forecasting problems are not simply linear nor nonlinear regressions,
  - you have a large dataset (or it is possible to augment the data),
  - Need a quick deployment for operational constraints (e.g., weather forecast)

Problem formulation

1. **Inputs vs. Outputs**: What are the inputs and outputs for a forecast?
2. **Endogenous (internal) vs. Exogenous (external)**: What are the endogenous and exogenous variables?
3. **Unstructured vs. Structured**: Are the time series variables unstructured or structured?
4. **Regression vs. Classification**: Are you working on a regression or classification predictive modeling problem? What are some alternate ways to frame your time series forecasting problem?
5. **Univariate vs. Multivariate**: Are you working on a univariate or multivariate time series problem?
6. **Single-step vs. Multi-step**: Do you require a single-step or a multi-step forecast?
7. **Static vs. Dynamic**: Do you require a static or a dynamically updated model?
8. **Contiguous vs. Discontiguous**: Are your observations contiguous or discontiguous?

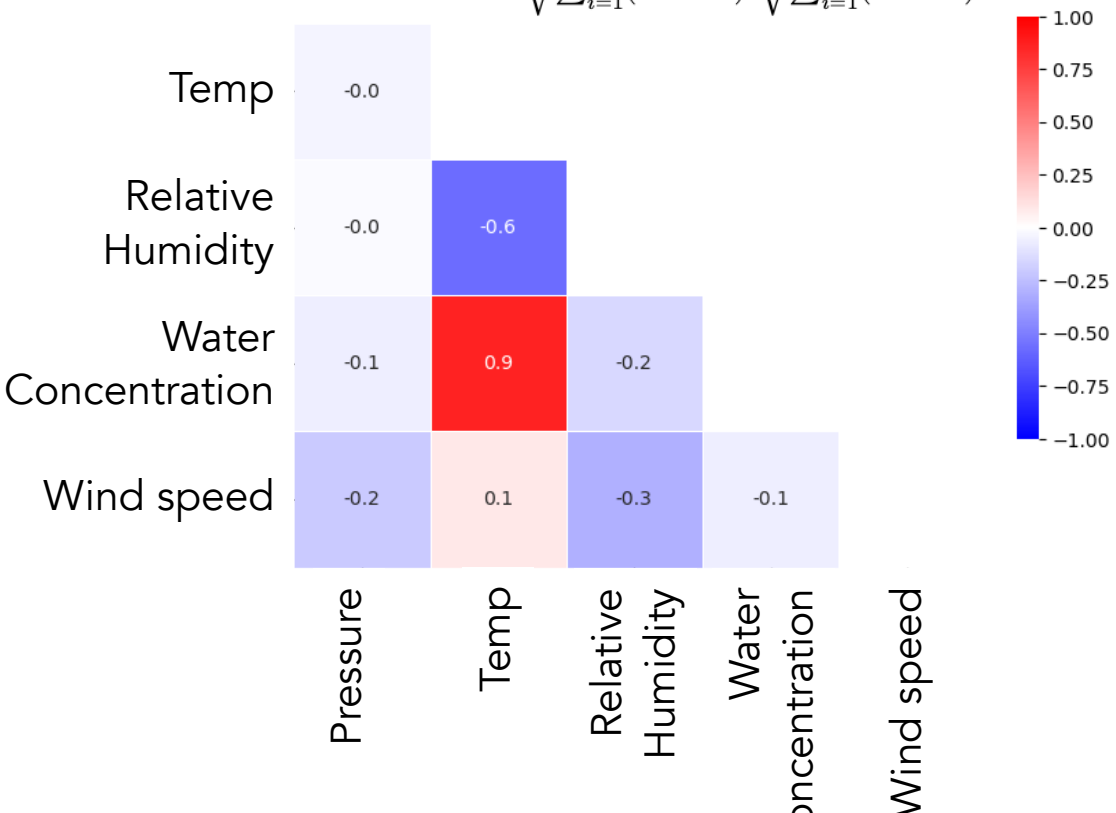
Some useful tools to help get answers: Data visualizations (e.g. line plots, etc.), Statistical analysis, Domain experts, Project stakeholders, etc.

Data exploration

- **Visual inspection** (visualization as a function of time)
- **Decomposition** (e.g., with filtering) if present periodicity/seasonality; trend reduction; noise reduction
- **Correlation analyses** between independent/input and dependent/output variables

Linear (Pearson) correlation

$$r_{xy} = \frac{\sum_{i=1}^{n}(x^{(i)} - \bar{x})(z^{(i)} - \bar{z})}{\sqrt{\sum_{i=1}^{n}(x^{(i)} - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(z^{(i)} - \bar{z})^2}}$$



Nonlinear correlation – **mutual information**

Given time series x(t) and y(t)

- p(i): probability density function (PDF) of i = x, y
- p(x,y): joint PDF

$$\mathrm{MI} = \sum_{j,k} p(x(t_j), y(t_k)) \log \frac{p(x(t_j), y(t_k))}{p(x(t_j))p(y(t_k))},$$

MI ∈ [0, 1] where

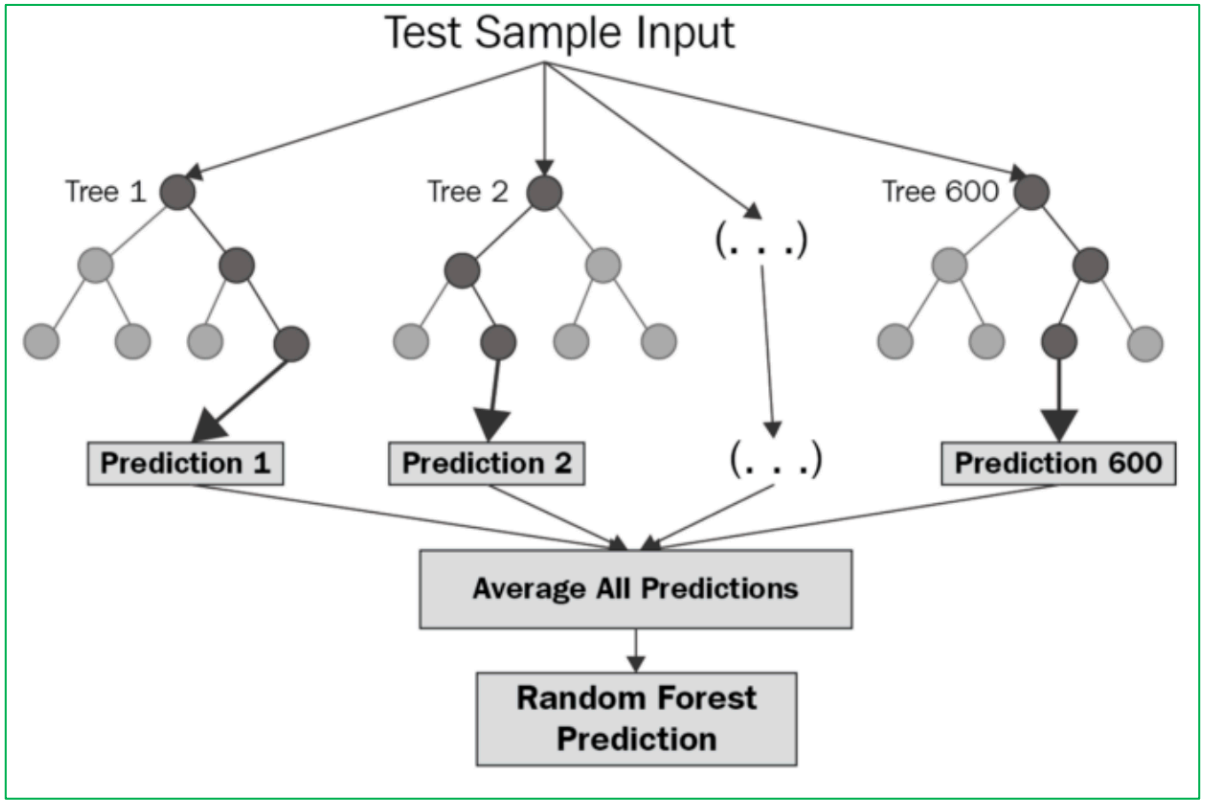
- MI = 0 for statistically independent series
- MI >= MI_{th}, i.e., above a threshold for statistical significance level (e.g., 95%)

ML models

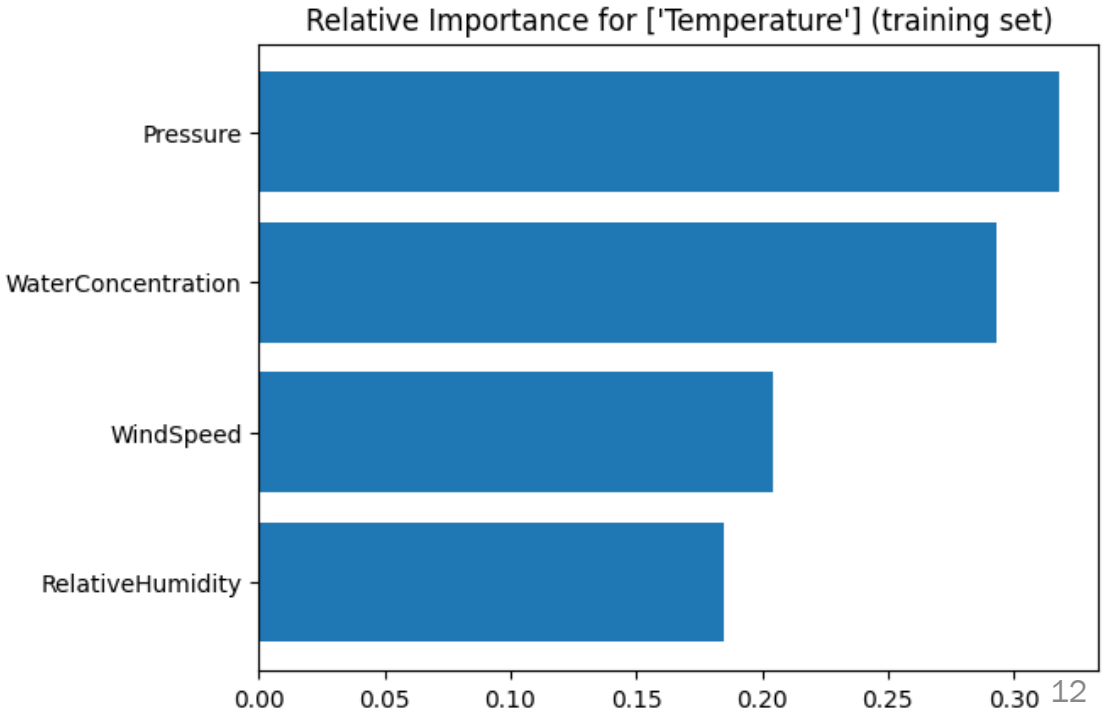
Some examples of ML models

- Decision Trees
- Random Forest
- Gradient Boosting
- K-nearest neighbors
- Support Vector Machine

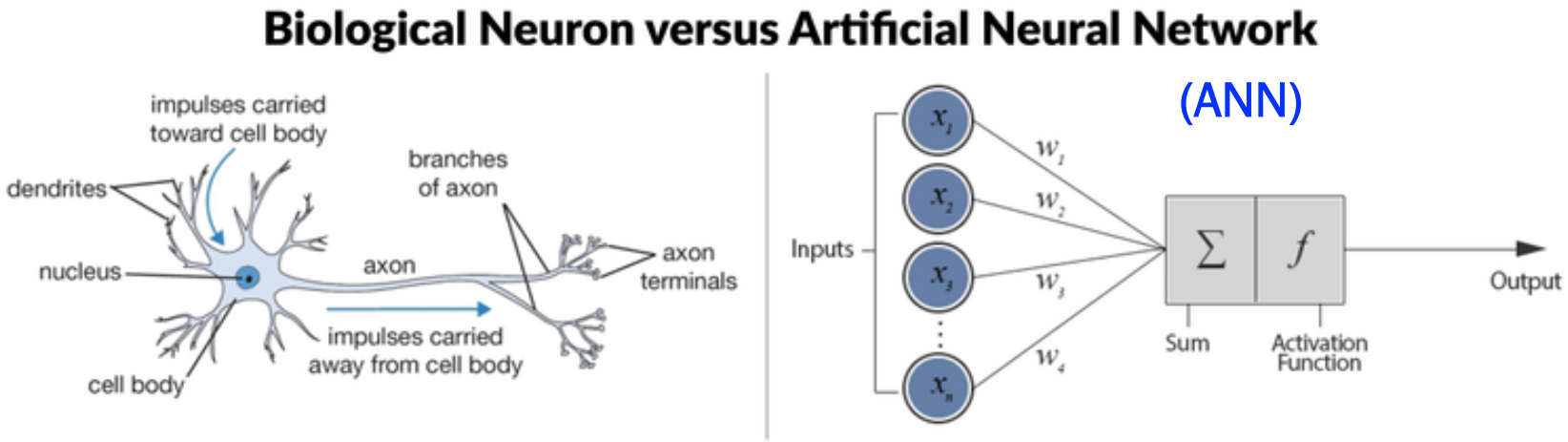
Ensure that the fitting and evaluation of these models preserved the temporal structure in the data.
This is important so that the method is not able to cheat by harnessing observations from the future.



Feature importance from Random Forest



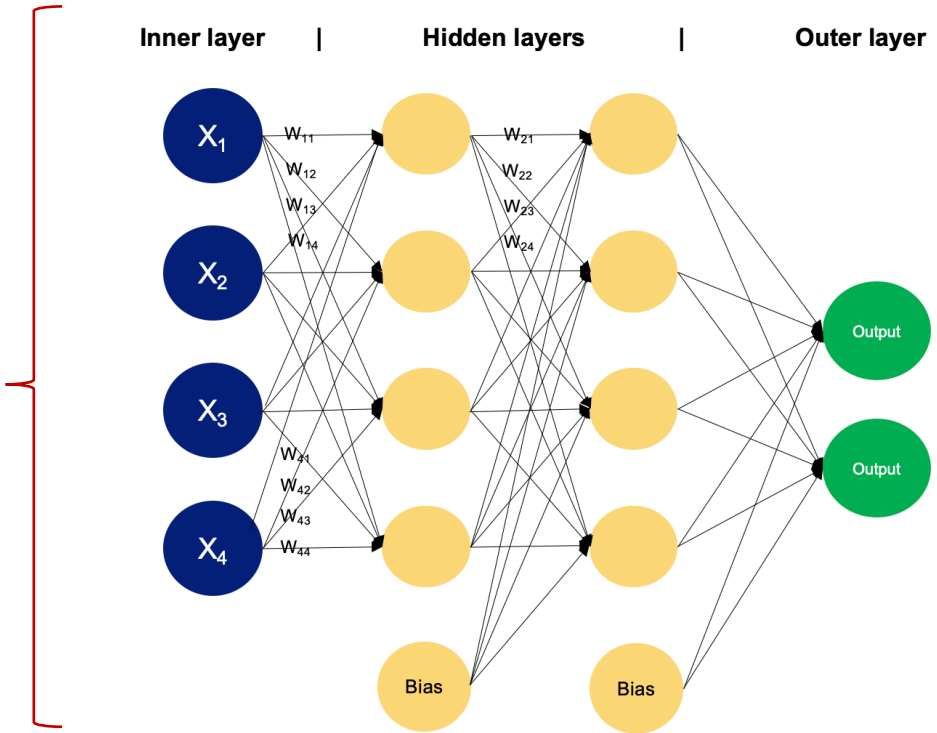
Neural Networks



The ANN is a simple neural network called '**perceptron**'. It consists of **a single layer**, which is the input layer, with multiple neurons with their own weights; there are no hidden layers.

Multi-Layer Perceptron (MLP)

- An MLP is a type of ANN consisting of **multiple layers of neurons**.
- The neurons in the MLP typically use **nonlinear activation functions**, allowing the network to learn complex patterns in data.
- Powerful models for classification, regression, and pattern recognition



Outline

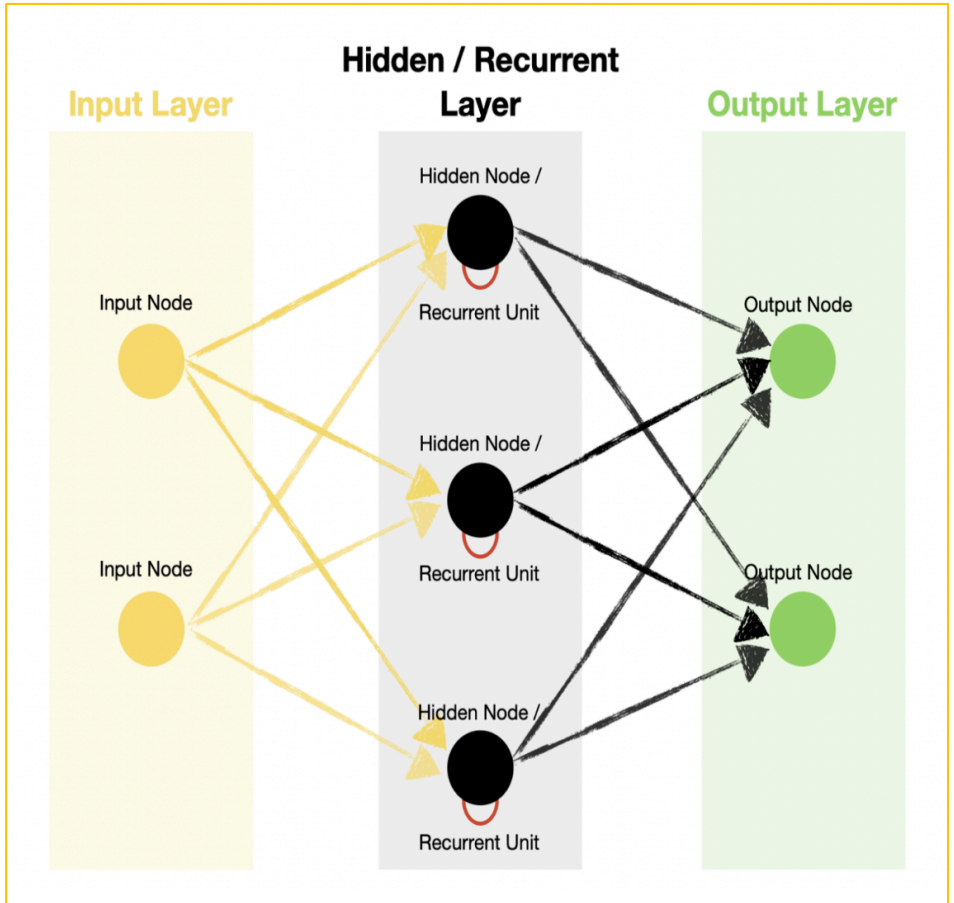
1. Introduction to regression modeling of time series
2. **Neural Networks for time series forecasting**
3. Introduction to Tensorflow and Keras, and how to build a complete neural network pipeline
4. Practical session on time series forecasting

Time series forecasting with neural networks

- Unlike the simpler problems of classification and regression**,** time series problems add **the complexity of order or temporal dependence between observations**.
- Neural networks help overcome limitations in classical methods such as fixed temporal dependence, univariate data, and one-step forecasts.

Some neural network architectures can be adapted to time series problems, or were designed to handle temporal dependence:

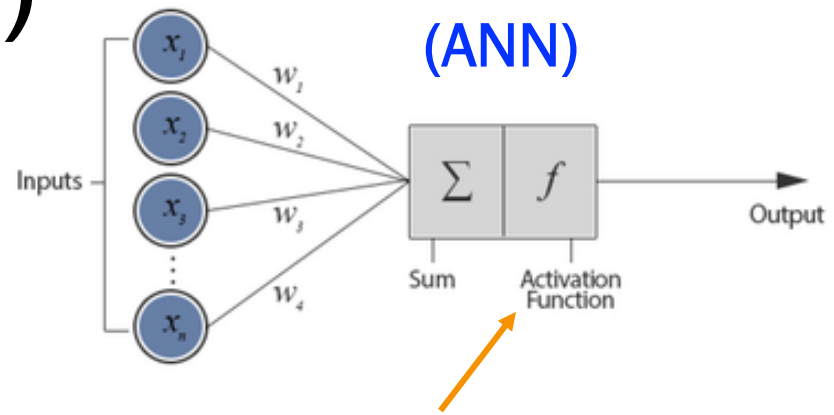
- Multilayer Perceptrons
- Convolutional Neural Networks
- **Recurrent Neural Networks**
- Transformers



Multi-Layer Perceptrons (MLP)

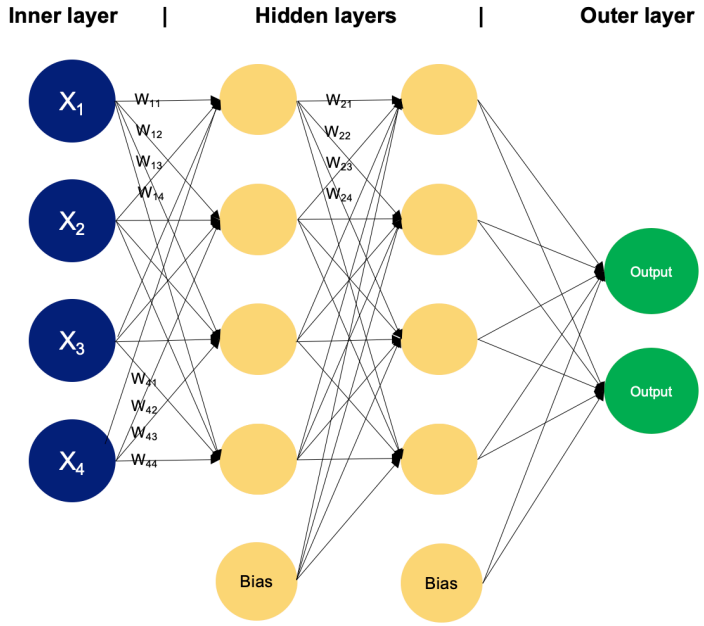
MLP approximates a mapping function from input variables to output variables.

- **Robust to Noise** in input data and in the mapping function.
- **Allow nonlinear relationships** in the mapping function



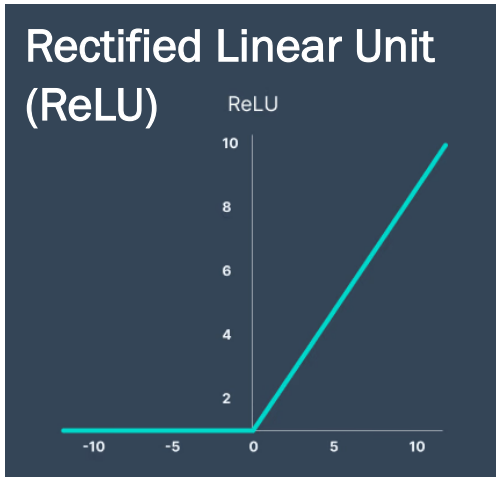
Activation Function helps the neural network to use important information while suppressing irrelevant data points.

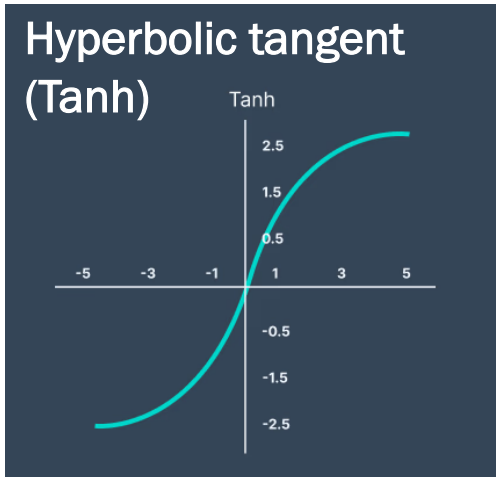
The whole point of hidden layers is to learn **non-linear combinations of the input features.**



✓ **Multivariate Inputs**

✓ **Multi-step Forecasts**





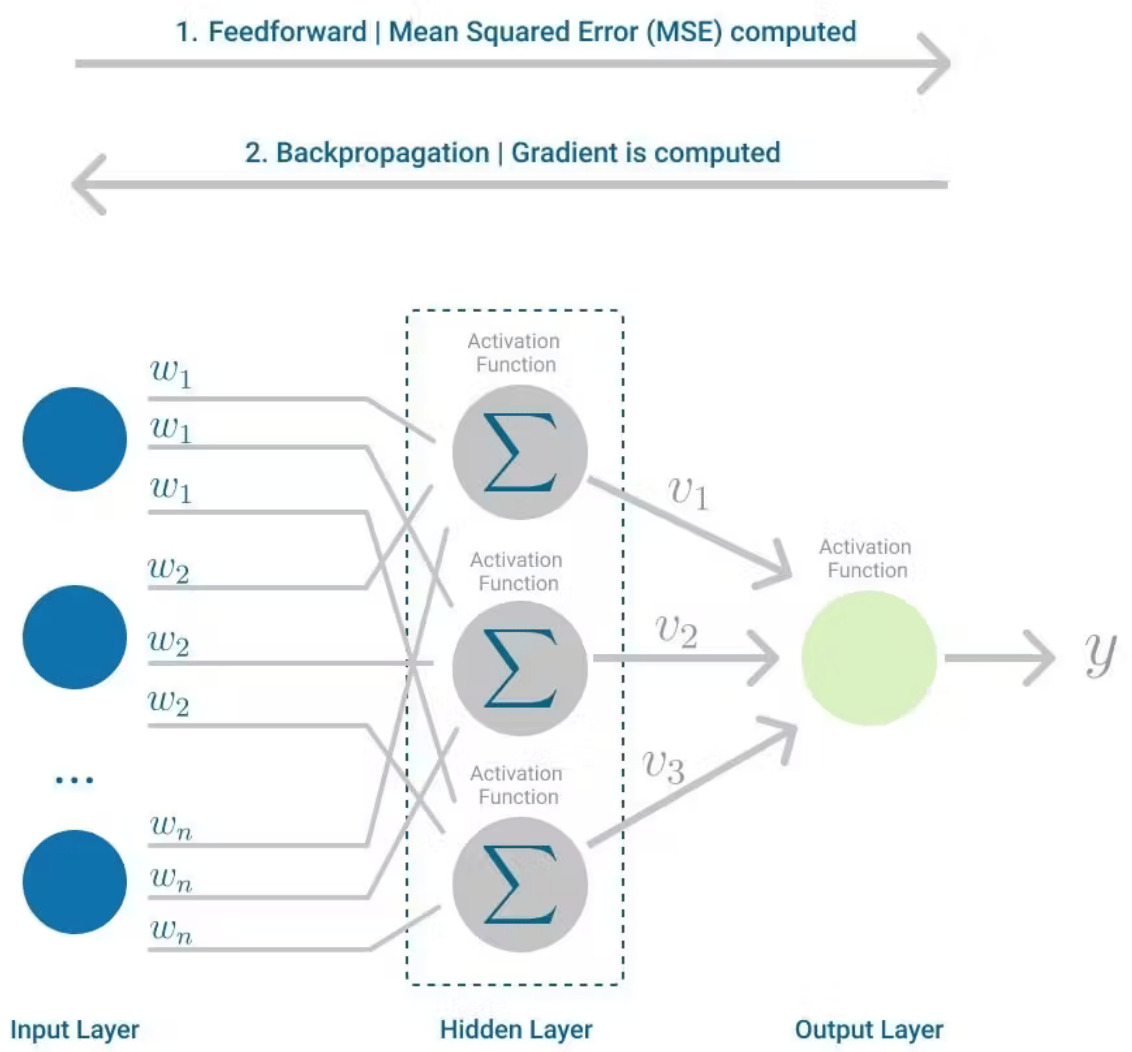
Also called "**Feedforward Neural Networks (FNN)**"

Multi-Layer Perceptrons (MLP/FNN): How they learn?

MLP uses something called the **feedforward** algorithm, i.e, the data moves in a single dimension.

Backpropagation is the method of fine-tuning the weights to reach an optimal solution to problems.

- A **loss function** is used to measure the error rate (e.g., MSE) at the end of every epoch.
- After the first iteration, the gradient of MSE across every input-output pair is calculated.
- The weights of the 1st hidden layer are replaced by this value of the gradient. The process is repeated till the convergence threshold is achieved.

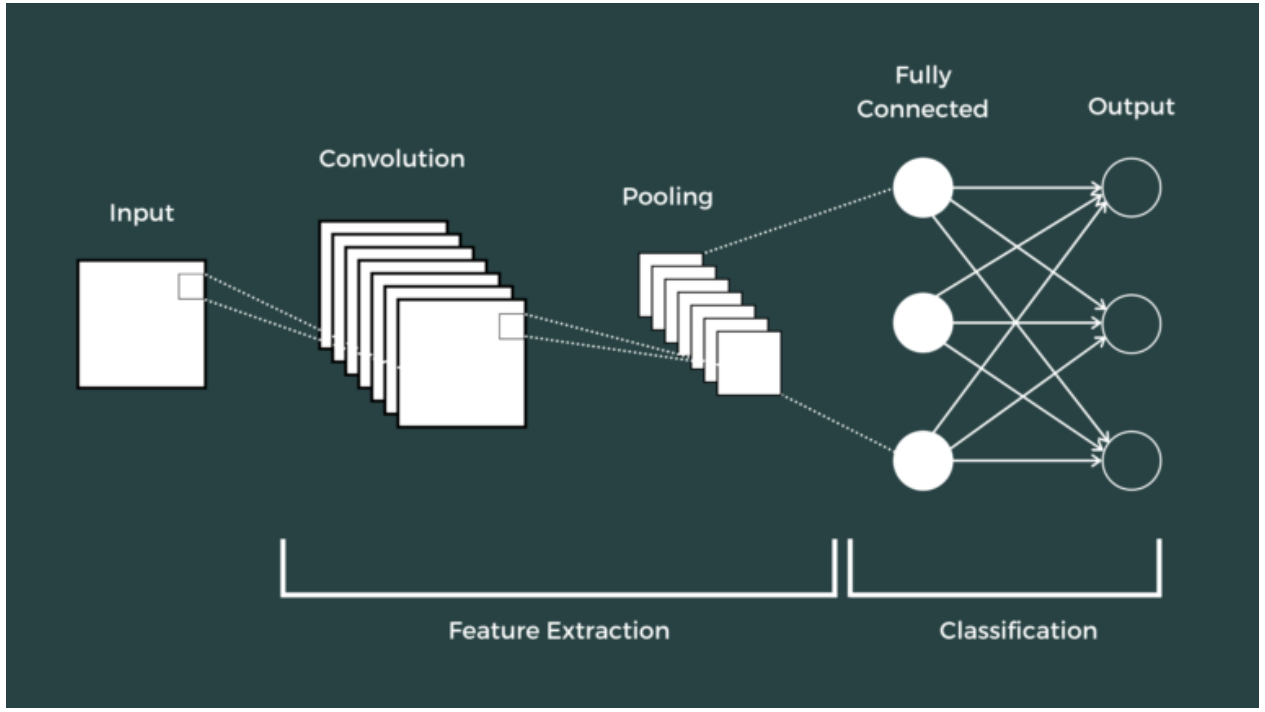


Convolutional Neural Networks (CNN)

A type of neural networks designed to efficiently handle image data. Effective on computer vision problems, e.g., image classification, object localization, image captioning and more.

The model learns how to **automatically extract the features from the raw data** that are directly useful for the problem being addressed.

- A sequence of observations can be treated like a 1D image that a CNN model can read.
- Unlike MLPs, **CNNs do not require that the model learn directly from lag observations**



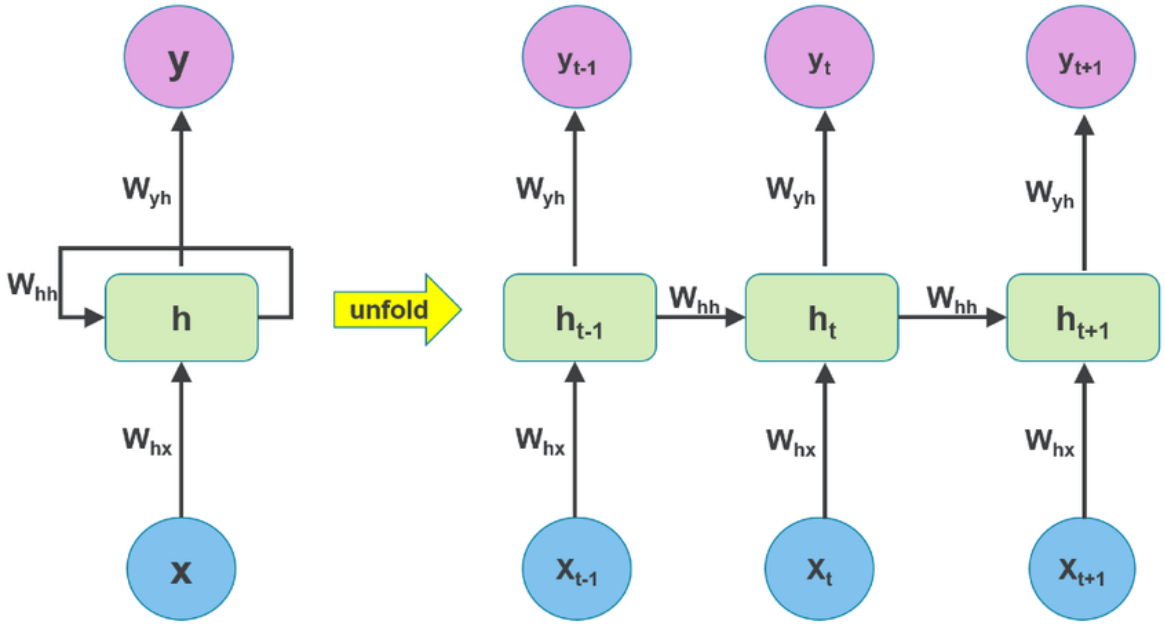
Feature Learning. *Automatic identification, extraction and distillation of salient features from raw input data that pertain directly to the prediction problem that is being modeled*

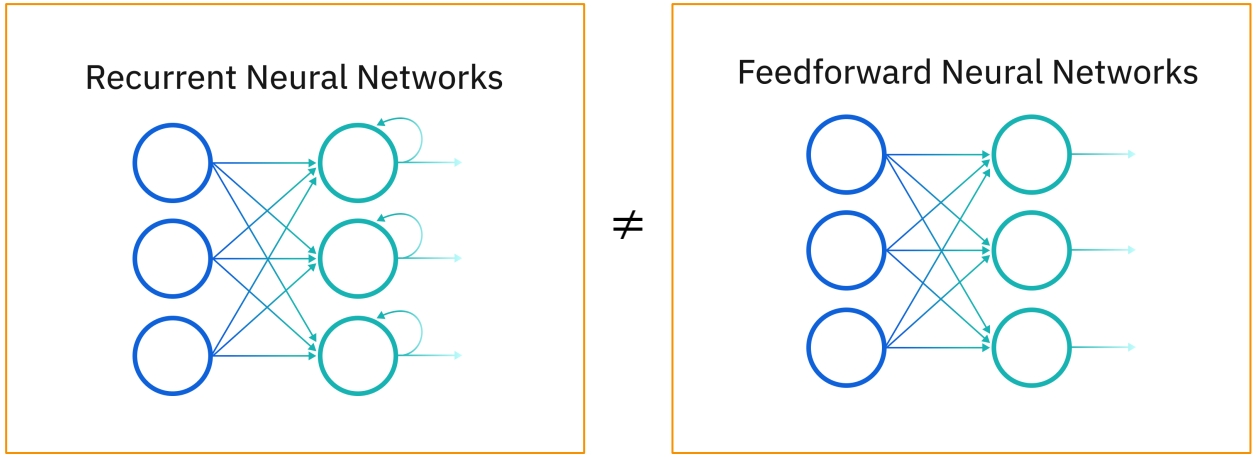
Recurrent Neural Networks

RNN is a set of algorithms which helps in processing sequences by retaining the memory (or state) of the previous value in the sequence. It helps process sequences like sentences, daily stock prices, etc.

Native Support for Sequences

Unlike MLPs, FNNs, and CNNs, **RNN takes information from prior inputs to influence the current input and output**





Hidden state at time t

Input at t Hidden state at t-1

$$\mathbf{h}_t = \phi_h(\mathbf{X}_t\mathbf{W}_{hx} + \mathbf{h}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h)$$

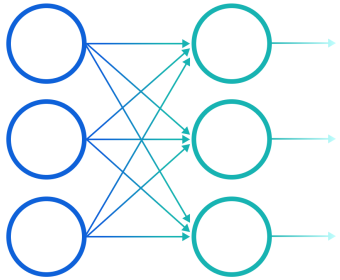
Activation function

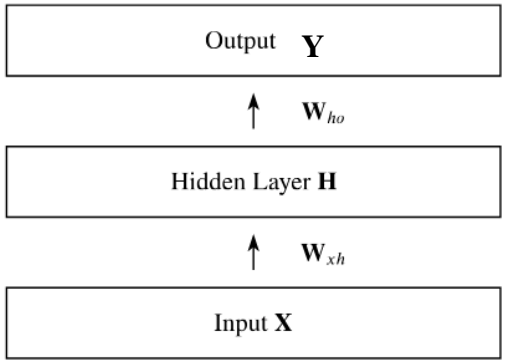
$$\mathbf{y}_t = \phi_o(\mathbf{h}_t\mathbf{W}_{yh} + \mathbf{b}_y)$$

Output at time t

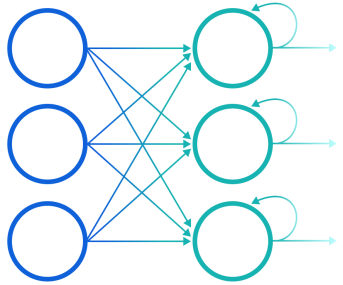
Feedforward vs Recurrent

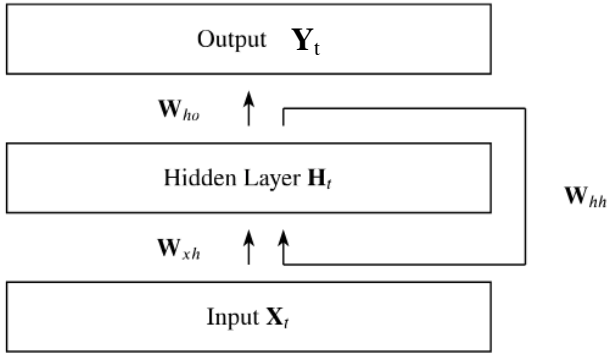
Feedforward Neural Networks





Recurrent Neural Networks





Hidden state

Input

$$\mathbf{h} = \phi_h(\mathbf{X}\mathbf{W}_{hx} + \mathbf{b}_h)$$

Activation function

Output

$$\mathbf{y} = \phi_o(\mathbf{h}\mathbf{W}_{yh} + \mathbf{b}_y)$$

Training with **Backpropagation**

Hidden state at time t

Input at t Hidden state at t-1

$$\mathbf{h}_t = \phi_h(\mathbf{X}_t\mathbf{W}_{hx} + \mathbf{h}_{t-1}\mathbf{W}_{hh} + \mathbf{b}_h)$$

Activation function

Output at time t

$$\mathbf{y}_t = \phi_o(\mathbf{h}_t\mathbf{W}_{yh} + \mathbf{b}_y)$$

Training with **Backpropagation Through Time (BPTT)**

Problem: *vanishing or exploding gradient for a very long sequence (see math for BPTT).*

Long Short-Term Memory Neural Networks (1/3)

Designed to handle the vanishing gradient problem in RNNs that arise with the long temporal dependencies (Hochreiter & Schmidhuber, 1997). To achieve that, LSTMs store more information outside of the traditional neural network flow in structures called gated cells.

Output gate (O_t) – read entries of the cell

$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{(t-1)} \mathbf{W}_{ho} + \mathbf{b}_o)$$

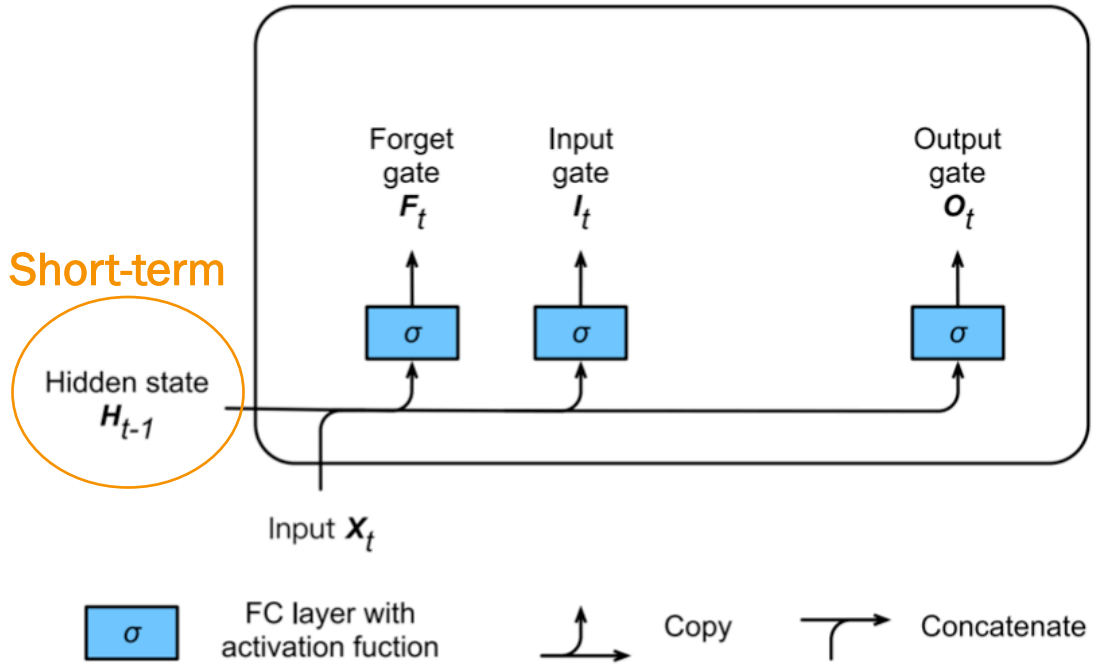
Input gate (I_t) - read data into the cell

$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{(t-1)} \mathbf{W}_{hi} + \mathbf{b}_i)$$

Forget gate (F_t) - reset the content of the cell

$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{(t-1)} \mathbf{W}_{hf} + \mathbf{b}_f)$$

where σ is a sigmoid activation function with outputs $\in [0,1]$.



FC: Fully Connected

Long Short-Term Memory Neural Networks (2/3)

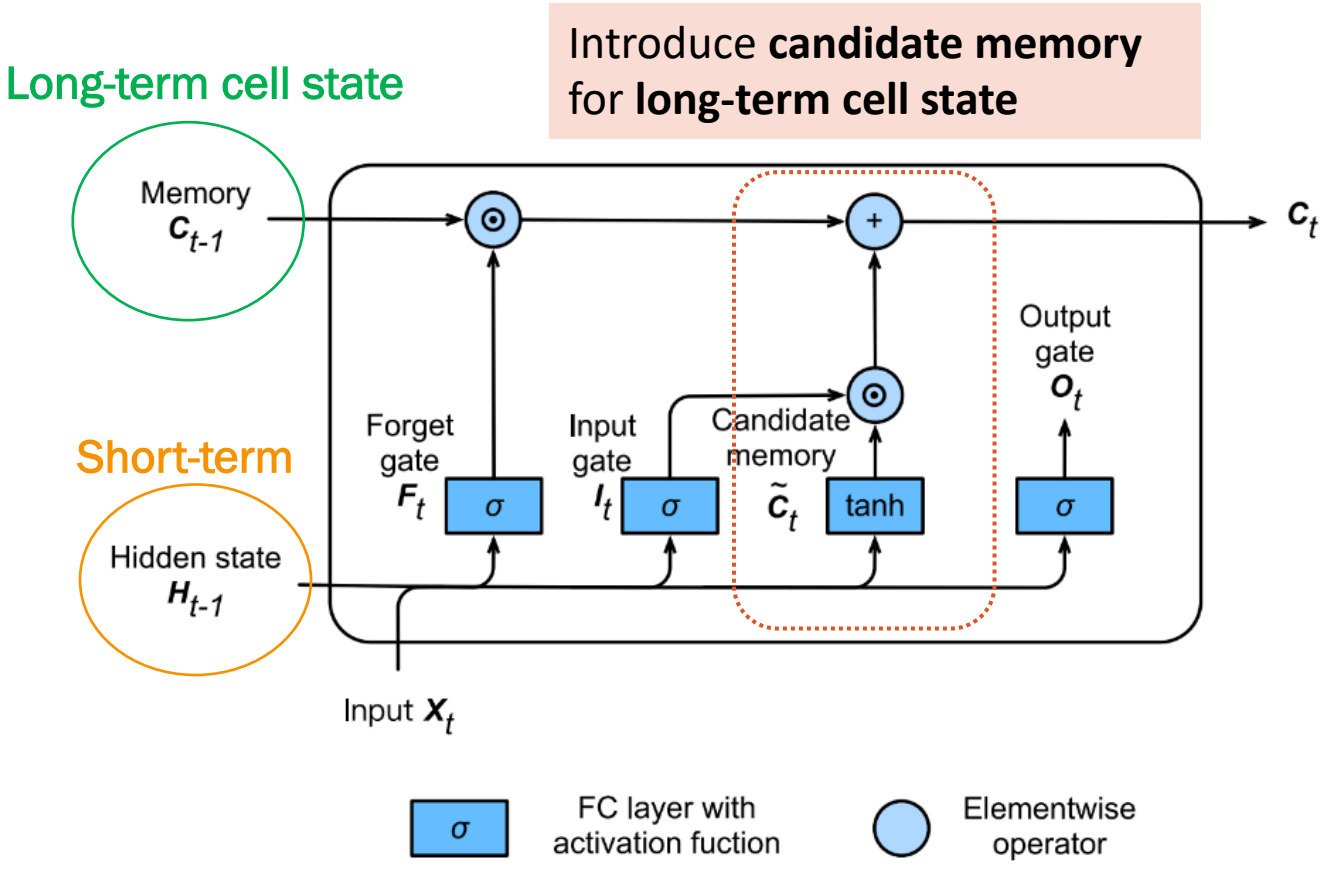
To incorporate old memory content (long temporal dependencies), we introduce candidate memory with tanh activation function with output ϵ [-1,1]

Candidate memory

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{(t-1)} \mathbf{W}_{hc} + \mathbf{b}_c)$$

Long-term cell state (old memory content)

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{(t-1)} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$



FC: Fully Connected

Long Short-Term Memory Neural Networks (3/3)

To incorporate old memory content (long temporal dependencies), we introduce candidate memory with tanh activation function with output ϵ [-1,1]

Candidate memory

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{(t-1)} \mathbf{W}_{hc} + \mathbf{b}_c)$$

Long-term cell state (old memory content)

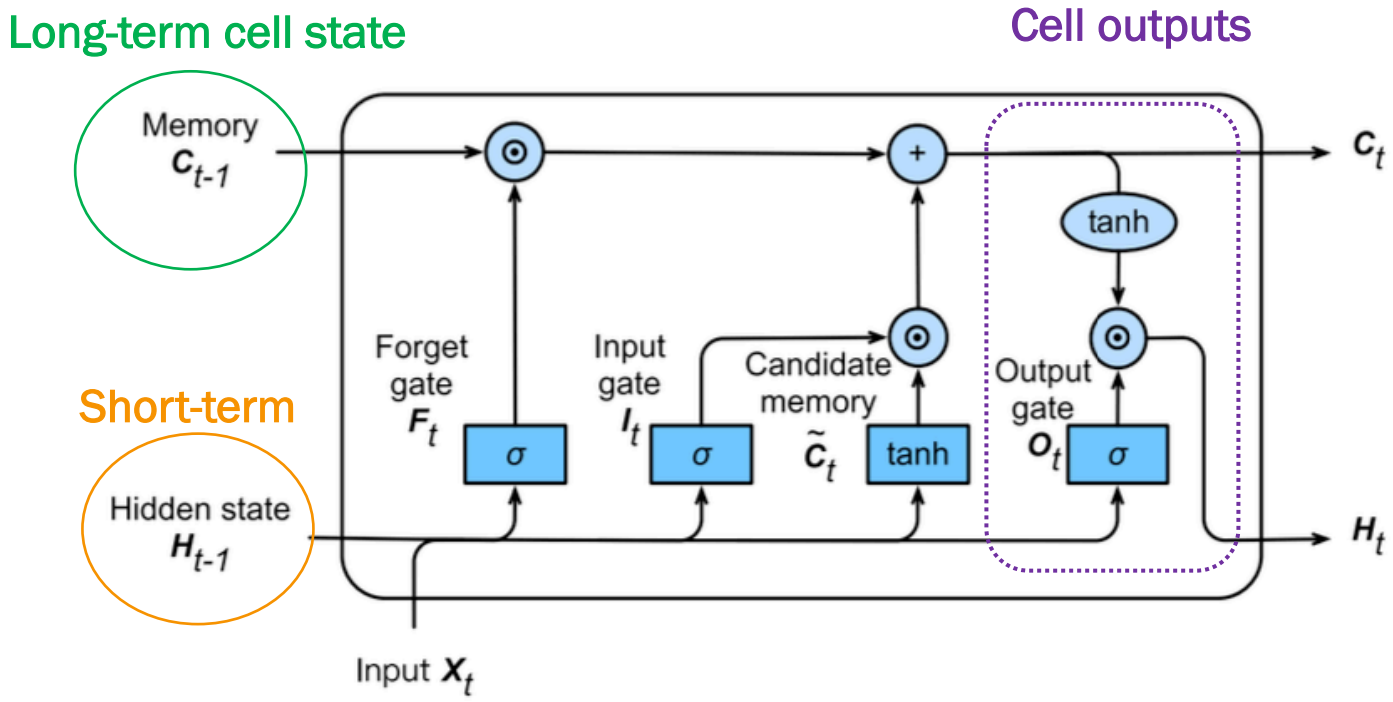
$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{(t-1)} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

Finally, we incorporate the **long-term cell state** and **short-term state** into the output.

Output (hidden-state *t*)

$$\mathbf{Y}_t = \mathbf{H}_t = \mathbf{O}_t \odot \tanh \mathbf{C}_t$$

"LSTM cell"

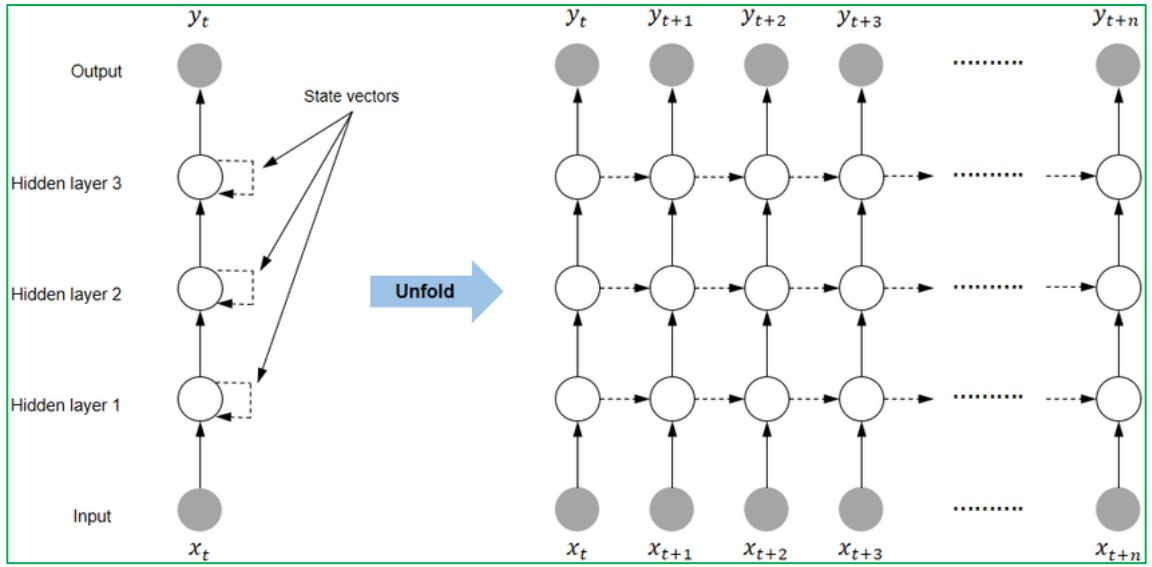


An LSTM cell is a neural network in itself. Here the equations are computed at each time instant t.

Deep Recurrent Neural Networks

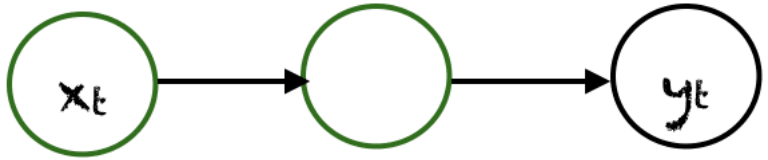
A deep RNN with *L* hidden layers: stacking up ordinary RNNs of any type on top of each other.
Each RNN can be an LSTM cell, or etc.

Ex: Deep RNN with 3 hidden layers



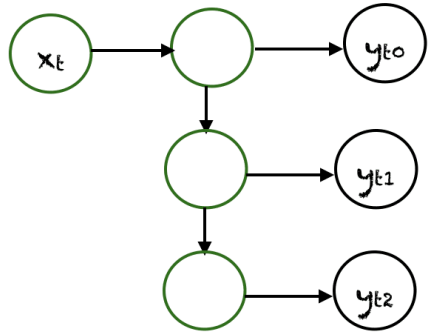
Architecture for univariate problems:

One-to-one

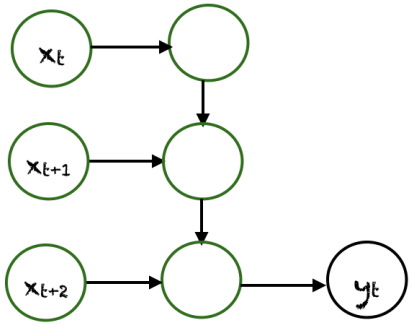


Architectures for multi-variate problems:

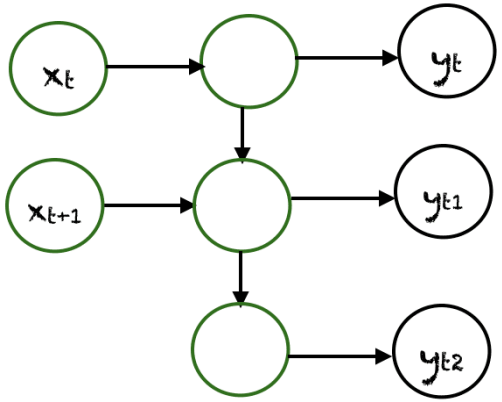
One-to-many



Many-to-one

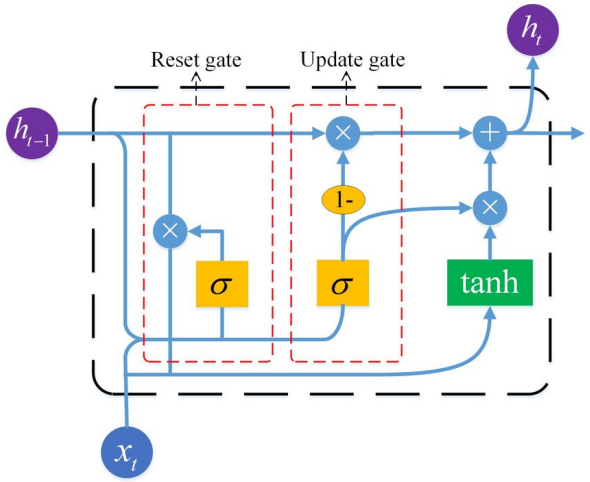


Many-to-many



More types of RNN

Gated Recurrent Units (GRUs)



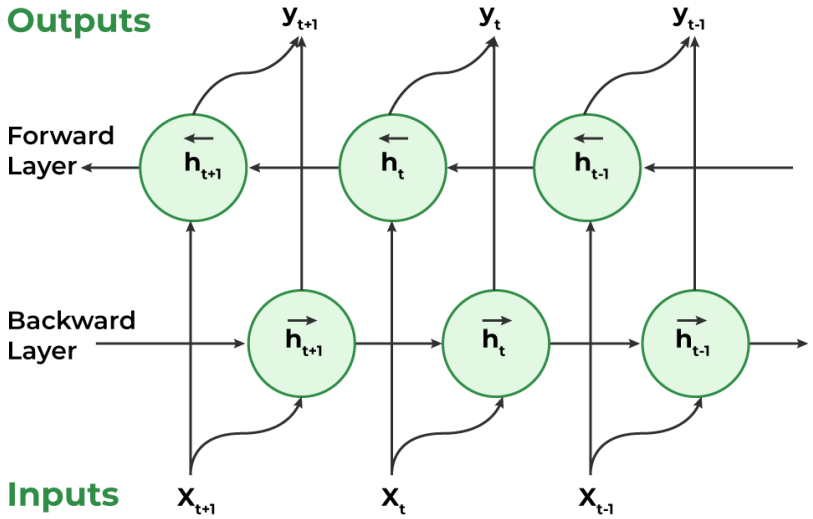
Like LSTM, GRU was designed to handle the vanishing gradient problem.

They have a **reset** and **update** gate.

These gates determine which information is to be retained for future predictions.

Newer than LSTMs, similar performance, but computationally more efficient.

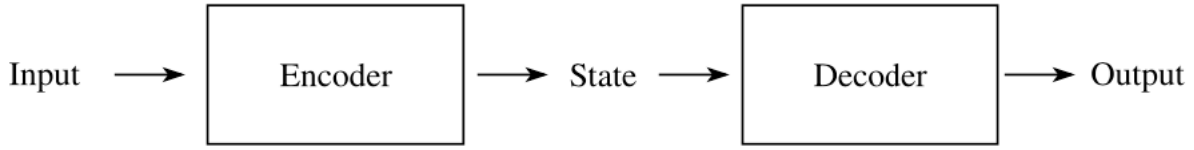
Bidirectional Recurrent Neural Networks (BRNNs)



Inputs from future time steps are used to improve the accuracy of the network.

Knowing the first and last words of a sentence to predict the middle words

Encoder-Decoder Architecture & Sequence to Sequence (seq2seq)



Focuses on mapping a fixed length input sequence of size ***n*** to an fixed length output sequence of size ***m,*** e.g., Google Translate.

Beyond RNN

RNN use has declined in artificial intelligence, especially in favor of architectures such as transformer models, but RNNs are not obsolete. RNNs were traditionally popular for sequential data processing (e.g. time series and language modeling) because of their ability to handle temporal dependencies.

Transformers can capture long-range dependencies much more effectively, are easier to parallelize and perform better on tasks such as NLP, speech recognition and time-series forecasting.

RNNs are still used in specific contexts where their sequential nature and memory mechanism can be useful, especially in smaller, resource-constrained environments or for tasks where data processing benefits from step-by-step recurrence.

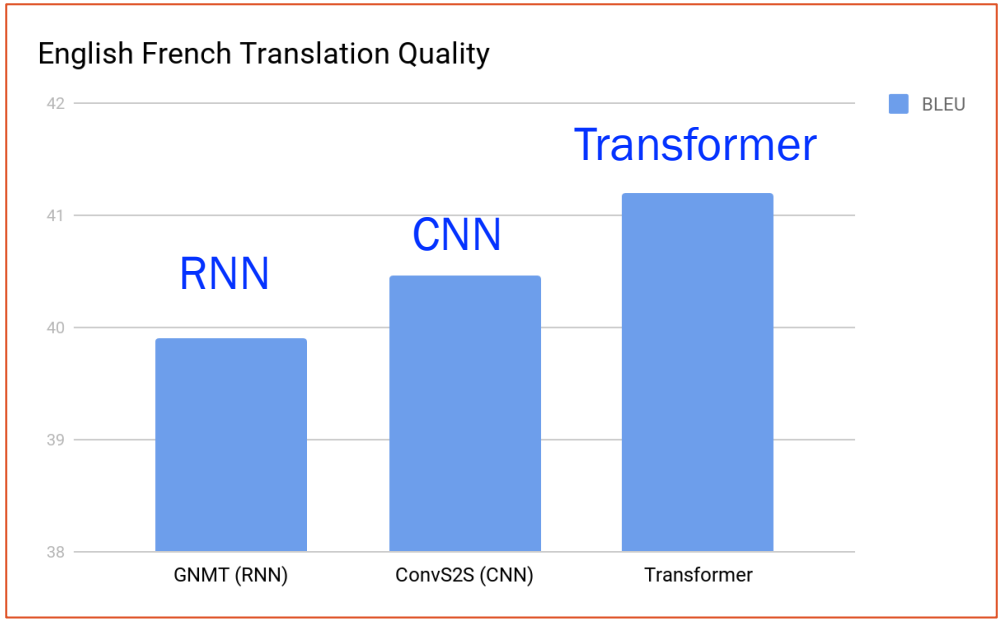
Attention Mechanism & Transformer

Natural Language Processing (NLP)

Large Language Model (LLM)

e.g., ChatGPT

https://research.google/blog/transformer-a-novel-neural-network-architecture-for-language-understanding



Exploitation of Deep Neural Networks

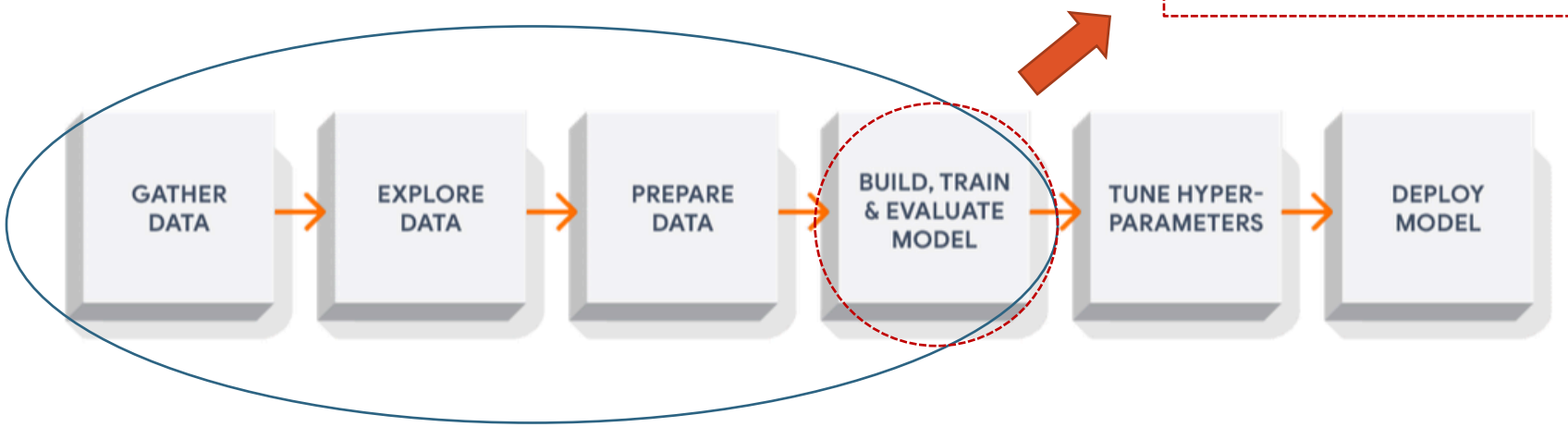
Advantage of neural networks:

- Arbitrary mapping functions
- Do not require a scaled or stationary time series as input
- Support multivariate inputs
- Support multi-step outputs

But you need to define problems and have proper (non-random) datasets with pre-processing and/or feature engineering.

What types of neural networks do you need?

- Do data have temporal dependencies?
  - Yes -> RNN, LSTM
  - No -> MLP
- Do you need to extract features from data (parameterization)?
  - Yes -> CNN
- Do you want to combine both capabilities?
  - Yes -> Use hybrid models such as CNN-LSTM, ConvLSTMs, etc.



TensorFlow

Keras

Outline

1. Introduction to regression modeling of time series
2. Neural Networks for time series forecasting
3. **Introduction to Tensorflow and Keras, and how to build a complete neural network pipeline**
4. Practical session on time series forecasting

Tensorflow

TensorFlow is an open source software library powered by **Google Brain** that allows anyone to utilize machine learning by providing the tools to train one's own neural network. It can run on various platforms and devices, and has features such as metrics, extensions, etc.

- Easy model building
- Robust ML production anywhere
- Powerful experimentation for research

TensorFlow 2.0 was released in Sep 2019.
TensorFlow's market share among research papers was declining to the advantage of PyTorch developed by Meta.

TensorFlow serves as a core platform and library for machine learning. TensorFlow's APIs use Keras to allow users to make their own machine-learning models.

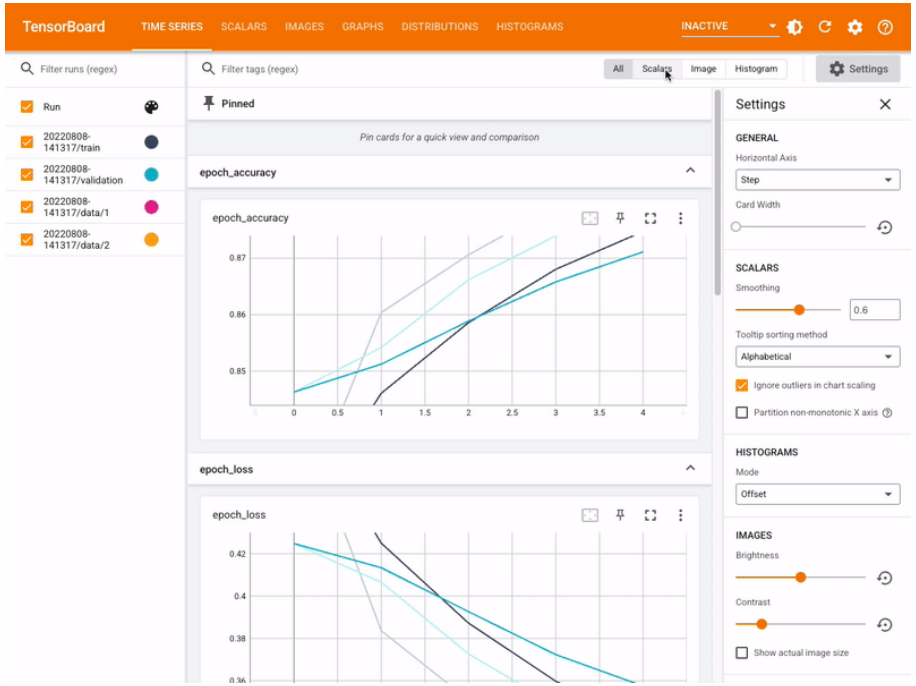
```
import tensorflow as tf
tf.random.set_seed(seed)

model = Sequential()
model.add(LSTM(100, …)
```

TensorBoard: TensorFlow's visualization toolkit

TensorBoard provides the visualization and tooling needed for machine learning experimentation:

- Tracking and visualizing metrics such as loss and accuracy
- Visualizing the model graph (ops and layers)
- Viewing histograms of weights, biases, or other tensors as they change over time
- Projecting embeddings to a lower dimensional space
- Displaying images, text, and audio data
- Profiling TensorFlow programs
- And much more




```
# Load the TensorBoard notebook extension
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

Keras

Keras is a deep learning API written in Python and capable of running on top of either JAX, TensorFlow, or PyTorch.

Deep Neural Networks with Keras

```
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
```

Create a model with stacked layers

```
model = Sequential()
model.add(LSTM(…, activation='relu'))
model.add(Dense(…)))

model.compile(optimizer=adam, loss='mse')
```

Train the model

```
model.fit(X, y, epochs=…,
        validation_data=(X_val, y_val),
        batch_size=…,
        callbacks=[tensorboard_callback,
        …,checkpoint])
```

Evaluate the model

```
model.evaluate(X_test, y_test)
```

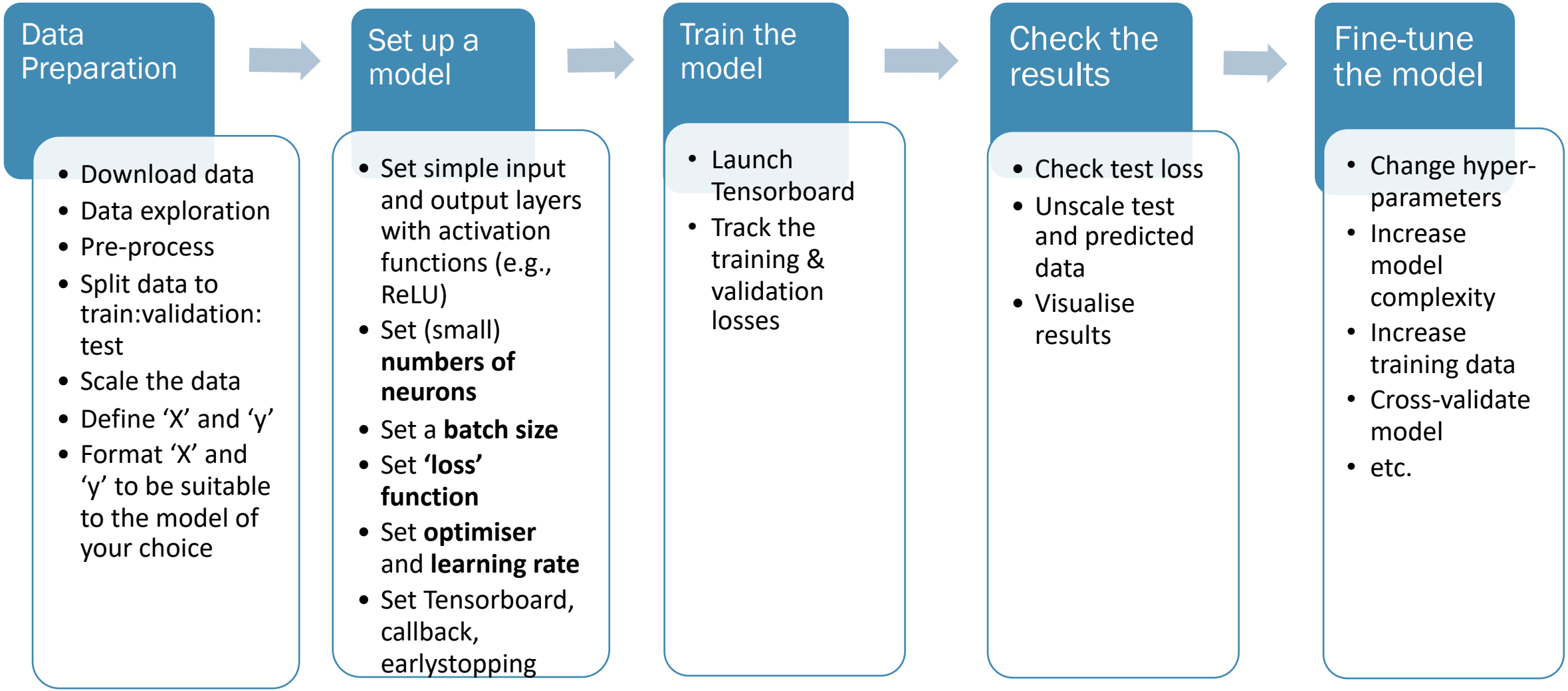
Make a prediction

```
model.predict(X_test_seq)
```

Note: we use Keras 2.12 in this workshop. Keras 3 is the newest version released in October 2023. See https://keras.io/guides/migrating_to_keras_3/

A complete workflow

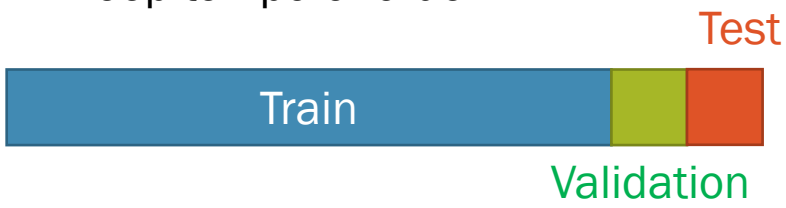
Pre-work: problem formulation, working environment setups, fix random seeds for reproducible results



Data preparation

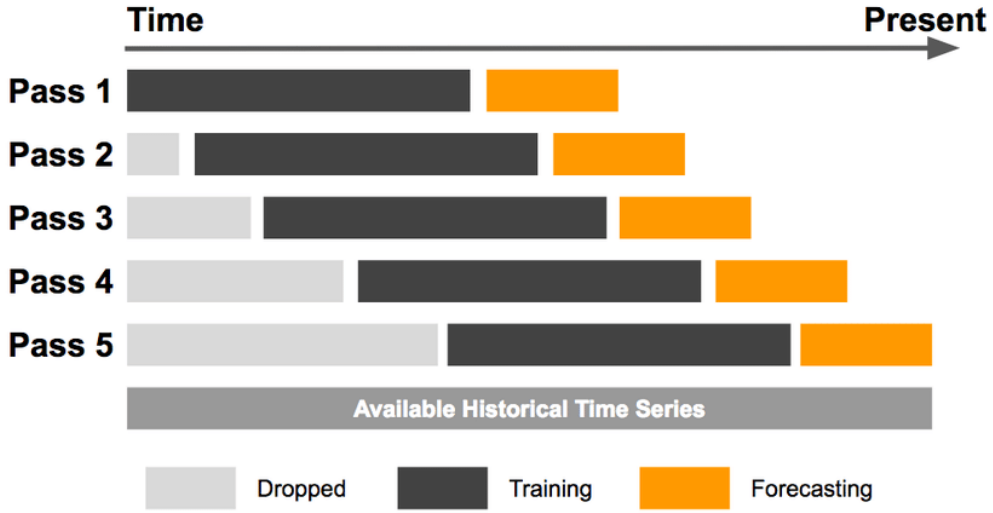
Data partitioning

- Train:Validation:Test (e.g., 0.8:0.1:0.1)
- Keep temporal order



Data used for model validation / choosing the best model

Walk-forward validation / ~~Cross-validation~~



Data scaling

- Normalization to range (0,1) $x' = \frac{x - x_{min}}{x_{max} - x_{min}}$
- Standardization $x' = \frac{x - x_{mean}}{x_{std}}$

*Obtain the scaling from the **training** data, then apply this scaling into the validation and test data*

```python
from sklearn.preprocessing import MinMaxScaler
# load data
data = ...
# create scaler
scaler = MinMaxScaler()
# fit and transform in one step
normalized = scaler.fit_transform(data)
# inverse transform
inverse = scaler.inverse_transform(normalized)
```

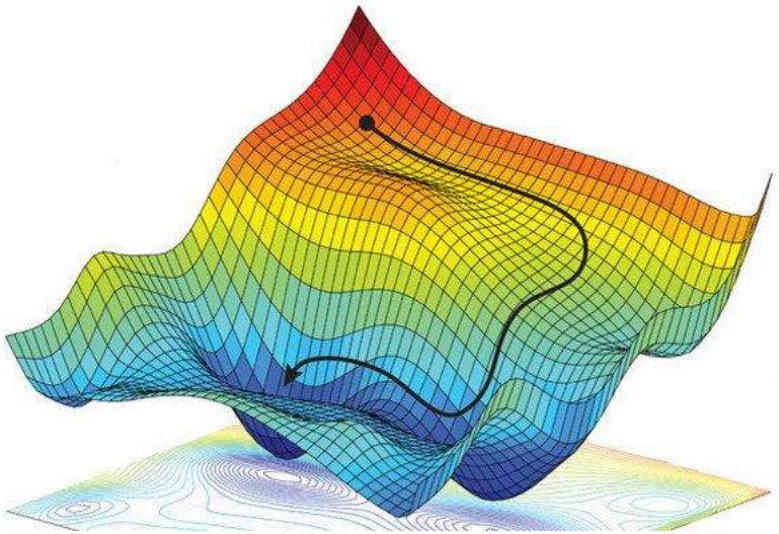
Neural network training

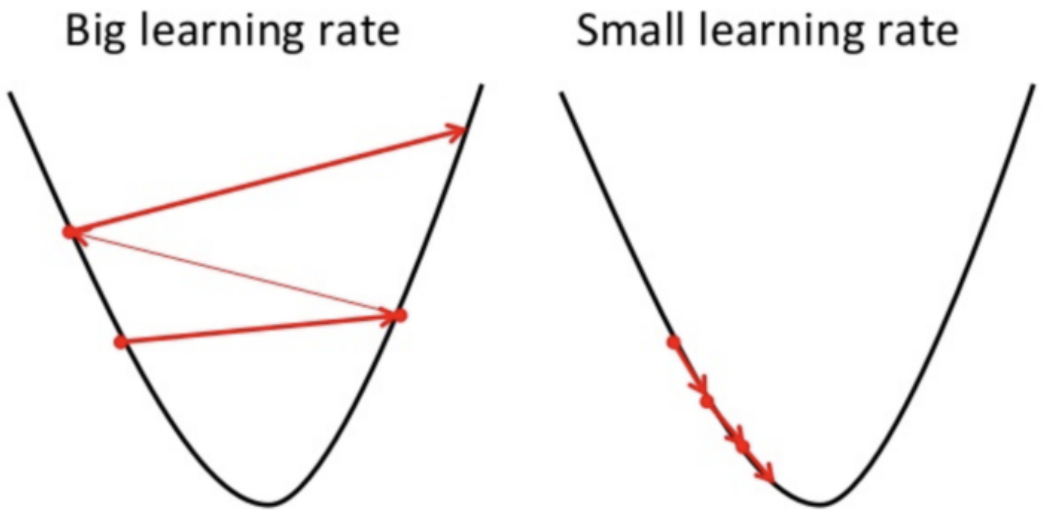
Loss functions (metrics) for regression, e.g.,
- Mean Squared Error (MSE), Mean Absolute Error (MAE),

Batch size:
How much of training data used for updating the weights for each epoch (round of learning)?

Learning optimization: algorithm and learning rate

- **Weights and biases** in neural networks are initialized by random numbers
- Through training (in batches), the weights are adjusted in the way that reduces the loss.



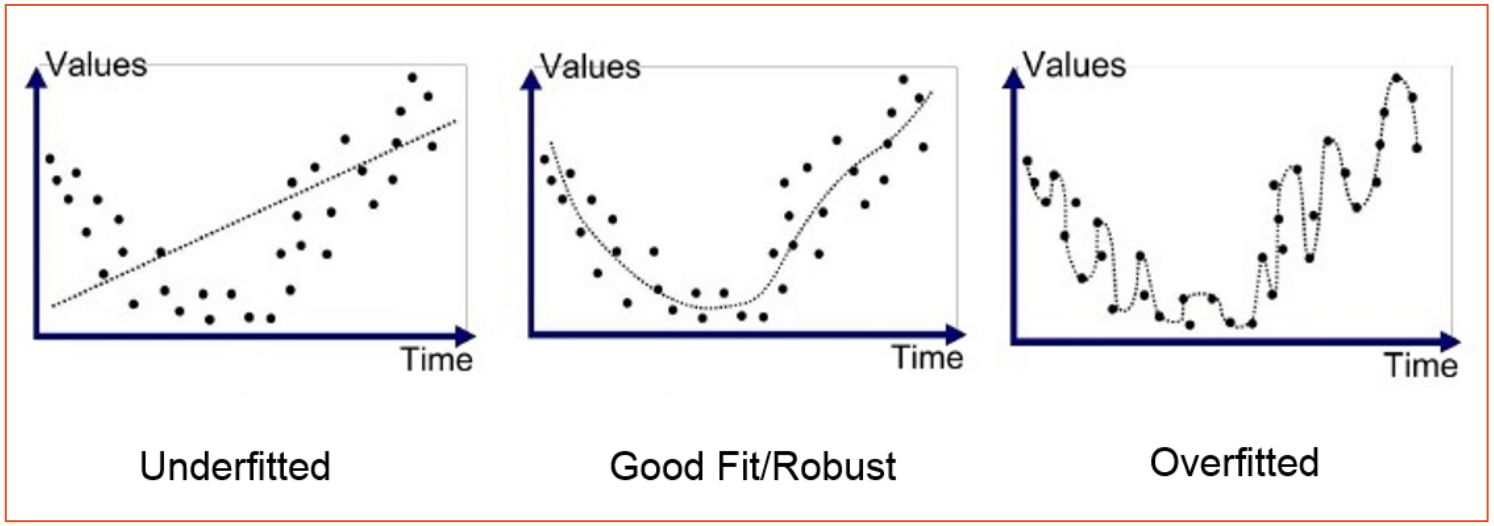


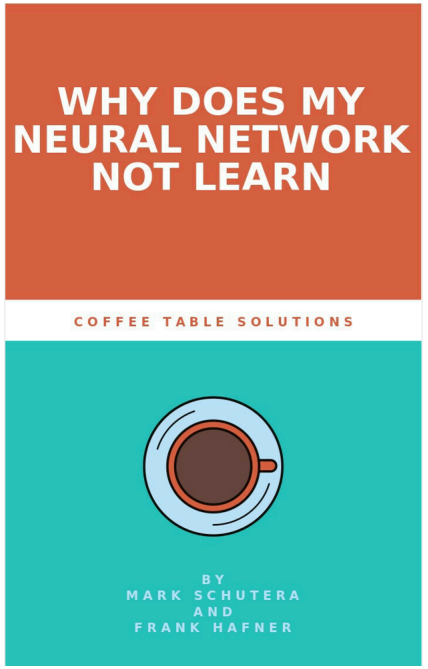
- This is done through "**stochastic gradient descent**" algorithm. A variant such as "**ADAM**" is highly efficient.
- The step size of this adjustment is defined through "**learning rate**".

Fine-tuning

It's like making the coffee you want with an espresso machine!

Does my model fit just right?





Underfitting

- Is my neural network too simple (cannot learn essential patterns in the data)?
- Are my training data too small?

Overfitting

- Is my neural network too complex (over-learn/over memorize the patterns)?
  - ➢ Simplify the architecture
  - ➢ Add a **dropout layer** to randomly drop a fraction of weights & biases

Outline

1. Introduction to regression modeling of time series
2. Neural Networks for time series forecasting and examples of applications
3. Introduction to Tensorflow and Keras, and how to build a complete neural network pipeline
4. **Practical session on time series forecasting**

Worksheets

Pre-work: setting up working environments on your computer

https://github.com/rungk-om/AI4REG-workshop/tree/main/Installation_guide

0. Data exploration with climate time series
1. Multivariate forecasting of Earth's magnetic field with LSTM neural networks
    - Optional: Updating the model with Walk Forward Validation
2. Autoregressive forecasting on climate time series with LSTM neural networks
3. Project: household electricity consumption prediction
    - Autoregressive univariate forecasting
    - Multivariate forecasting (many-to-one)

https://github.com/rungk-om/AI4REG-workshop

Data exploration with climate time series

Jena Climate dataset recorded by the Max Planck Institute for Biogeochemistry. The dataset consists of 14 features such as temperature, pressure, humidity etc, recorded once per 10 minutes from Jan 10, 2009, to December 31, 2016

- Data visualization
- Data cleaning
- Data exploration/analyses:
  - Linear (Pearson) correlation
  - Nonlinear correlation (Mutual information)

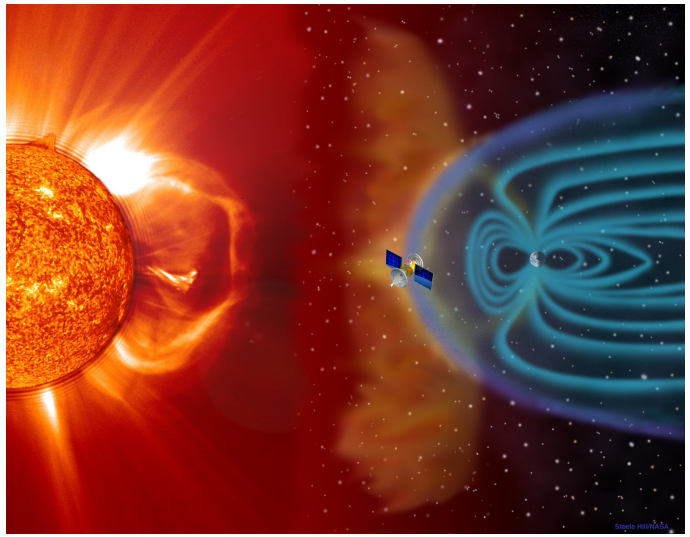
Deduce relationships between parameters

- Modeling "Temperature" with ML model (Random Forest) with a subset of parameters
  - Resampling data to 1-hour cadence
  - Divide data to train:validation:test
  - Scale the data
  - Fit the model
  - Unscale the data, make prediction

Explore "feature importance"

Complete workflow for building an ML model

Multivariate forecasting of Earth's magnetic field



Problem: forecasting the geomagnetic field perturbation as influenced by the solar activities. Use past 12-hour solar activities to predict next hour geomagnetic field.

Data: solar activity monitoring

- Interplanetary magnetic field
- (Bx, By, Bz)
- solar wind speed (V)
- solar wind number density (N)
- solar wind temperature (T)
- solar irradiance (F10.7)

Geometrical data for seasonality

- Local time (UT)
- Sun-Earth distance

Ground magnetic measurements at Chambon-la-Forêt (targets)

- Pre-processed data "daily variation" (xD, yD, zD)

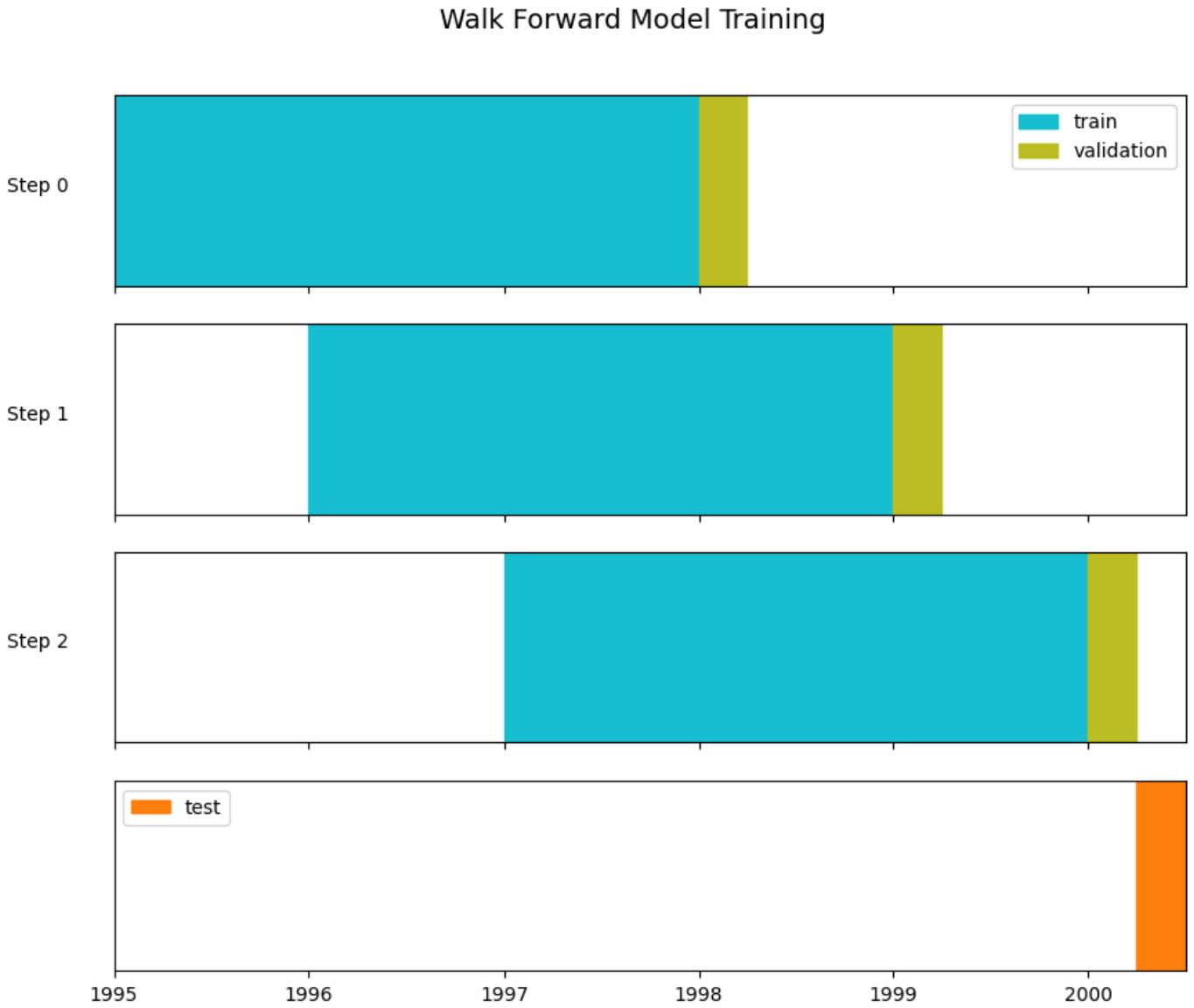
Using LSTM neural networks

```
model = Sequential()
model.add(LSTM(100, input_dim=len(input_features),
kernel_initializer='he_uniform',
return_sequences=True,
activation='relu'))
model.add(LSTM(50, return_sequences=True, activation='relu'))
model.add(LSTM(50, activation='relu'))
model.add(Dense(len(output_targets)))
adam = optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=adam, loss='mse')
```

Input/output data structures for 1-step forecasting

X: (data length, data steps, # features)
y: (data length, # features)

Optional: Update model with Walk Forward



Our model was trained using data between 1995 and 1997.

Supposing that now we want to update the model every year as the data become available.

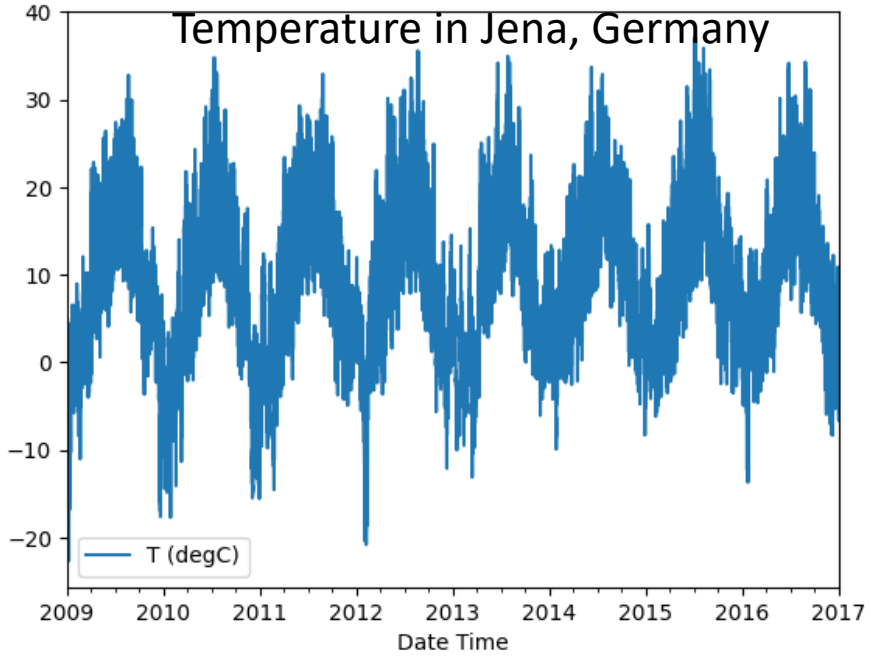
Here, we will update our model training using a sliding window or Walk Forward Validation.

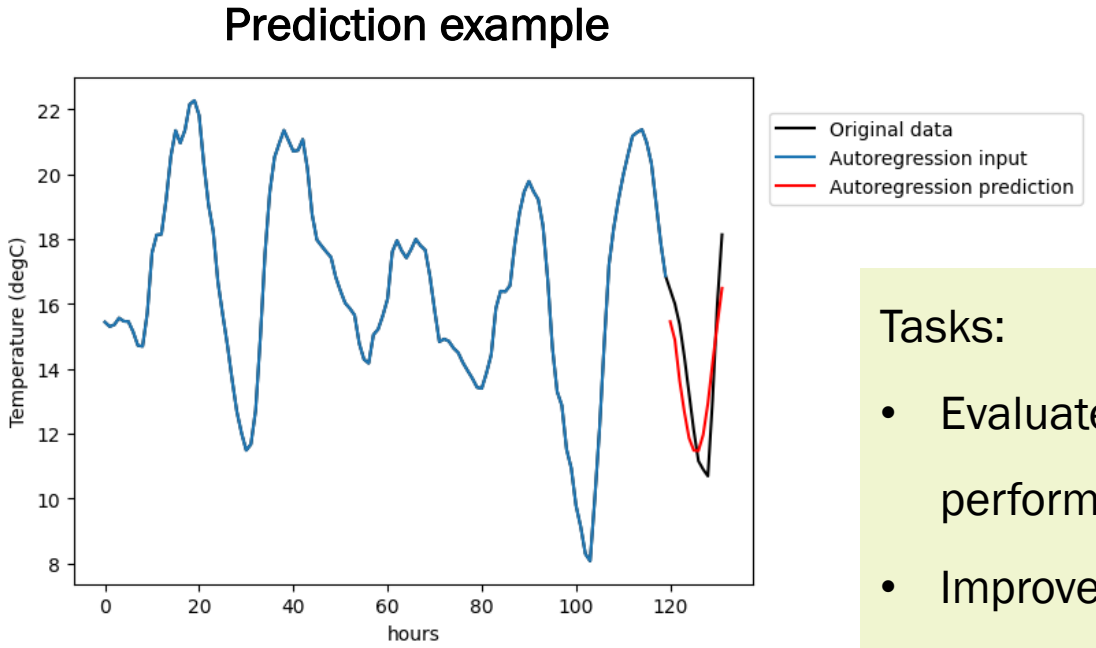
This approach has been used in economy and stock market predictions where the model is retrained once newer data become available. The advantage is that the model would be the most up-to-date, making it more relevant to the current situation.

We demonstrate these in this worksheet.

Univariate autoregressive forecasting of temperature

Problem: predict the temperature of the next 12 hours, given the temperature of the past 5 days.



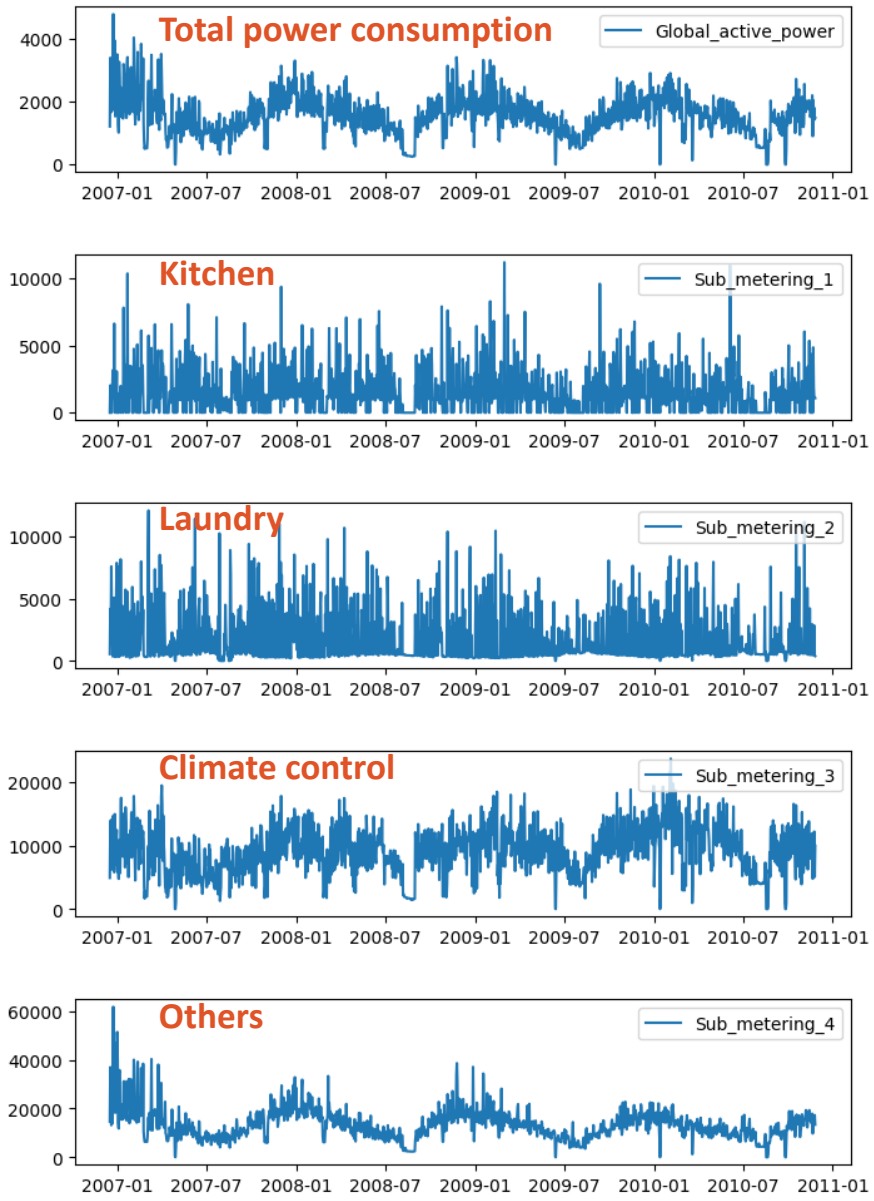



```
model = Sequential()
# define encoder
model.add(LSTM(32, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(RepeatVector(n_steps_out))
# define decoder
model.add(LSTM(32, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(16, activation='relu'))) #
model.add(TimeDistributed(Dense(1)))
model.compile(loss='mse', optimizer='adam')
```

Tasks:

- Evaluate the model performance
- Improve the model, e.g., increasing training data, add model complexity

Project: forecast household electricity consumption



Data: Measurements of electric power consumption in one household located in Sceaux (7km of Paris) between Dec 2006 and Nov 2010

Tasks

1. Explore the dataset

2. Formulate the problem

- Based on the power consumption (global active power) of last month, predict the power consumption for next week using an autoregressive model. Here, predict "global active power".
- Based on the sub_metering_1, sub_metering_2, sub_metering_3, sub_metering_4, model "global active power" in real time (using DNN model), or predict it for next week using the data of last month.

3. Design a model, train it, and then test

References

- Brownlee, J. (2017). Introduction to time series forecasting with python: how to prepare data and develop models to predict the future. Machine Learning Mastery.
- Brownlee, J. (2018). Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery.
- Molnar, C. (2023). Interpreting machine learning models with SHAP. Lulu. com.
- Schmidt, R. M. (2019). Recurrent Neural Networks (RNNs): A gentle Introduction and Overview. https://arxiv.org/pdf/1912.05911.
- IBM. What is a recurrent neural network (RNN)? https://www.ibm.com/topics/recurrent-neural-networks.