Time Series Forecasting with Neural Networks

M.Sc. Economics
University of Bologna
A.Y. 2020/2021

Motivation - 1/2

- Accurate time series forecasting is crucial in both economic and financial applications
- Standard econometric approaches rely almost exclusively on *linear models* (AR, MA, ARIMA...) as they usually perform well on standard tasks.
- Developments in deep learning have made available highly flexible models that can infer any complex nonlinearity from the data (**Universal Approximation Theorem**)

Motivation - 2/2

- Existing literature is lacking a unified perspective on the performance of neural networks on economic and financial forecasting tasks.
- Only a few types of neural networks (typically two) are usually tested against each other. Here, we compare *five different neural network models*.
- The difference between *univariate* and *multivariate* performance is not always assessed
- Economic and financial series are never contrasted with more 'standard' (well-behaved) series

Linear Models

The **linear models** estimated in this project are:

- The **Random Walk (RW)**, a simple *persistence model*:

$$y_t = y_{t-1} + \epsilon_t$$

where ϵ_t is a *white noise process* (zero mean and constant variance).

- The **Autoregressive Integrated Moving Average (ARIMA)**, taking the form:

$$\Delta y_t = \alpha + \phi_1 \Delta y_{t-1} + \ldots + \phi_p \Delta y_{t-p} + \theta_1 \epsilon_{t-1} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where:

  - **p** is the number of *autoregressive components*
  - **d** is the *order of integration* (Δ)
  - **q** is the number of *moving average components*

Feedforward Neural Networks - 1/2

A **feedforward neural network** (FFNN) is a sequence of **dense layers** taking the form:

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

where K is the number of **neurons**, each triggered by an **activation** h_k:

$$h_k(X) = g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

where $g(\cdot)$ is a **nonlinear activation function** (e.g. ReLU, tanh, sigmoid) and p are the predictors.

Feedforward Neural Networks - 2/2

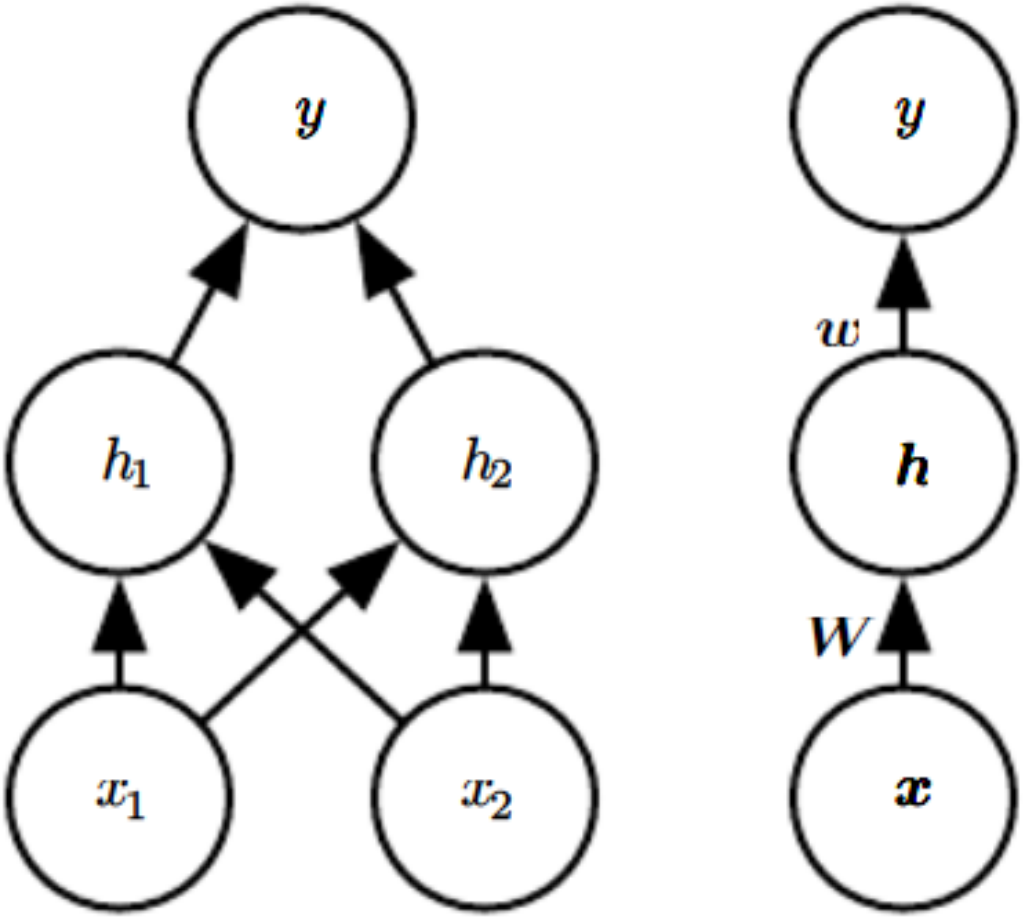


Figure: Single-layer Feedforward Network with two neurons (Extended (left) and schematic (right) representations)

Convolutional Neural Networks - 1/4

Convolutional Neural Networks (CNNs) make use of *two specific layers*:

- **Convolution Layer**
- **Max Pooling Layer**

Convolutional Neural Networks - 2/4

The **convolution layer** uses a **kernel (k)** and *convolves* it over the **input data (X)**, producing a **feature map (s)**:

$$X = [0.4, 0.1, 0.5, 0.3, 0.9] \qquad k = [1, 0, 1]$$

$$\Downarrow$$

$$s = [0.4 + 0 + 0.5, 0.1 + 0 + 0.3, 0.5 + 0 + 0.9] = [0.9, 0.4, 1.4]$$

Convolutional Neural Networks - 3/4

The **max pooling layer** performs *dimension reduction* selecting the highest value within a given **pool size**:

$$s = [0.9, 0.4, 1.4] \qquad \text{pool size} = 2$$

$$\Downarrow$$

$$s = [\max(0.9, 0.4), \max(0.4, 1.4)] = [0.9, 1.4]$$

Convolutional Neural Networks - 4/4

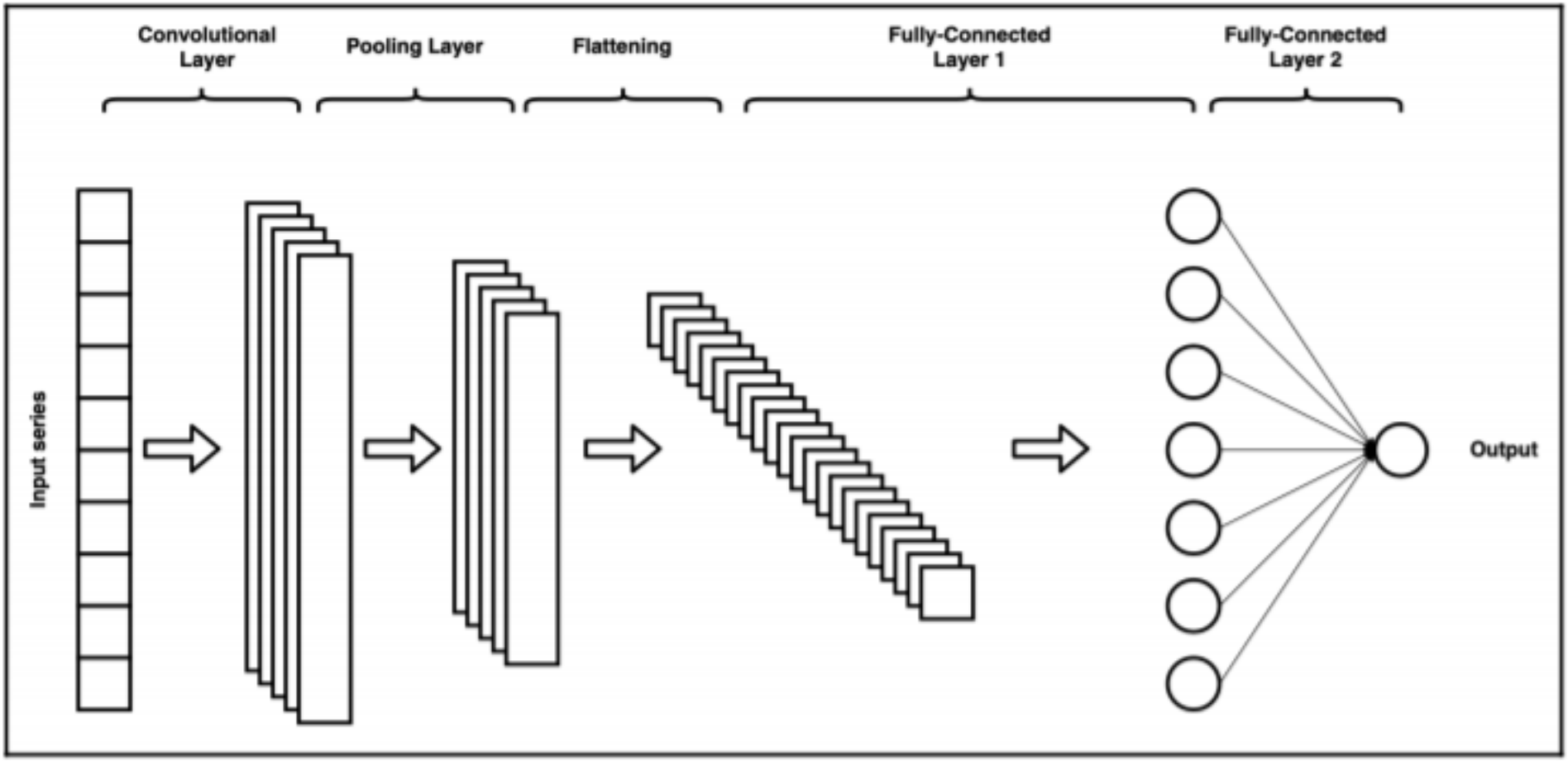


Figure: Convolutional Neural Network for 1-d Time Series Forecasting

Recurrent Neural Networks - 1/5

- Recurrent neural networks (RNNs) are designed to model *sequences of data*.
- They take in a sequence $X = \{X_1, X_2, \ldots, X_T\}$ and pass it through a sequence of **hidden states** $\{h_t\}_1^T = \{h_1, h_2, \ldots, h_T\}$:

$$h_{tk} = g\left(w_{k0} + \sum_{j=1}^{p} u_{kj} X_{tj} + \sum_{s=1}^{K} w_{ks} h_{t-1,s}\right)$$

  for K neurons in each hidden state.
- Note that the value of the present state is **dependent on the value of the previous state**.

Recurrent Neural Networks - 2/5

- The **output** of the RNN at each time step is a weighted sum of the hidden states' neurons

$$O_t = \beta_0 + \sum_{k=1}^{K} \beta_k h_{tk}$$

- The weights in each layer are **not time-dependent**

Recurrent Neural Networks - 3/5

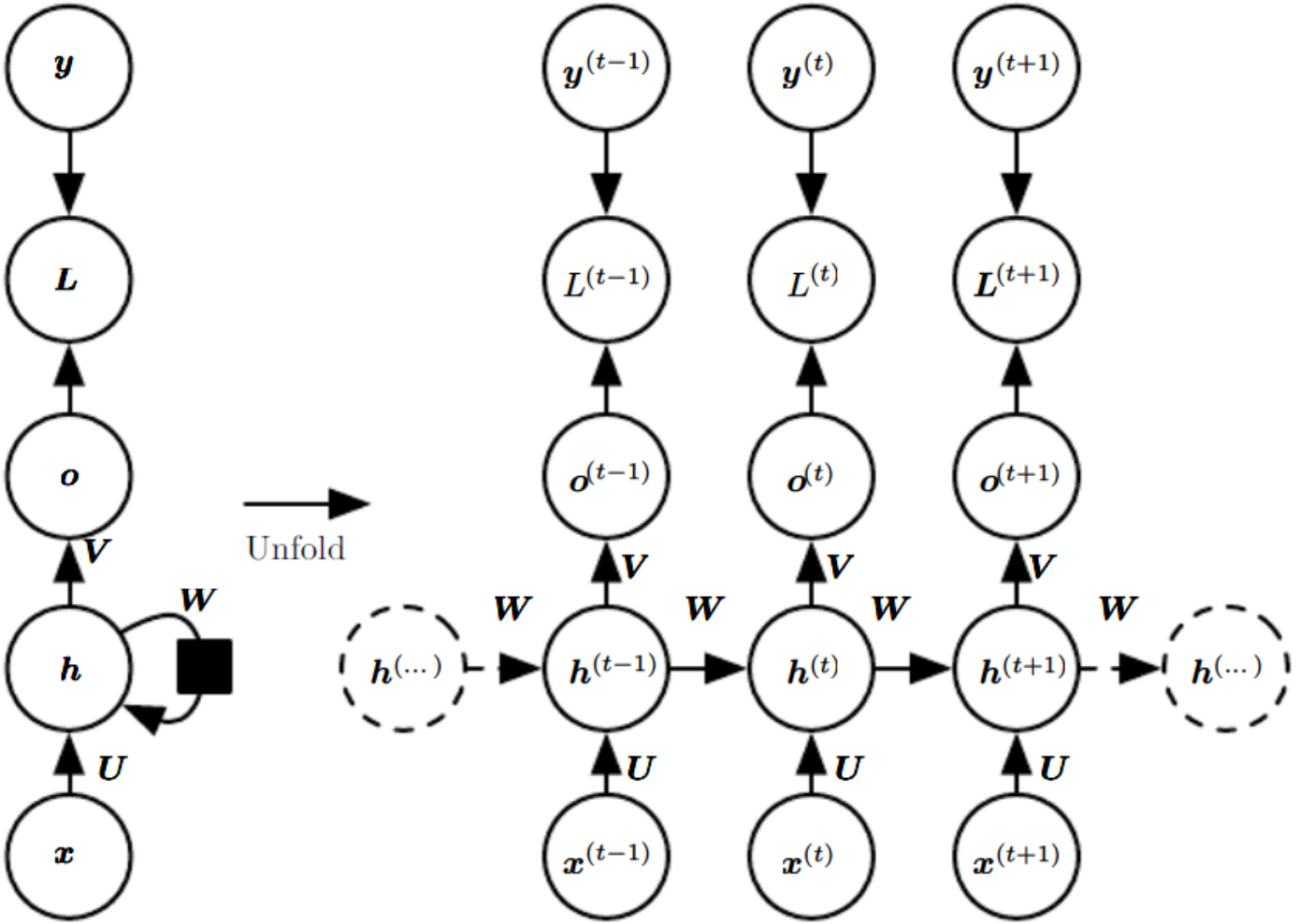
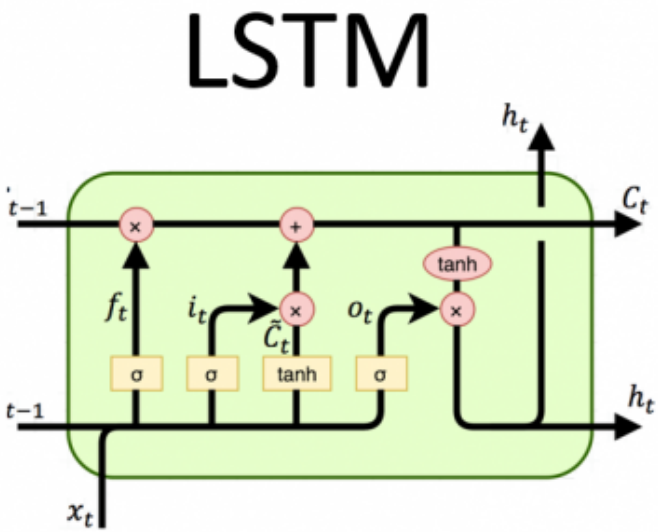


Figure: Single-layer Recurrent Network
(Schematic (left) and extended (right) representations)

Recurrent Neural Networks - 4/5

The **Long Short-Term Memory (LSTM)** network is a type of RNN where each cell is made up of:

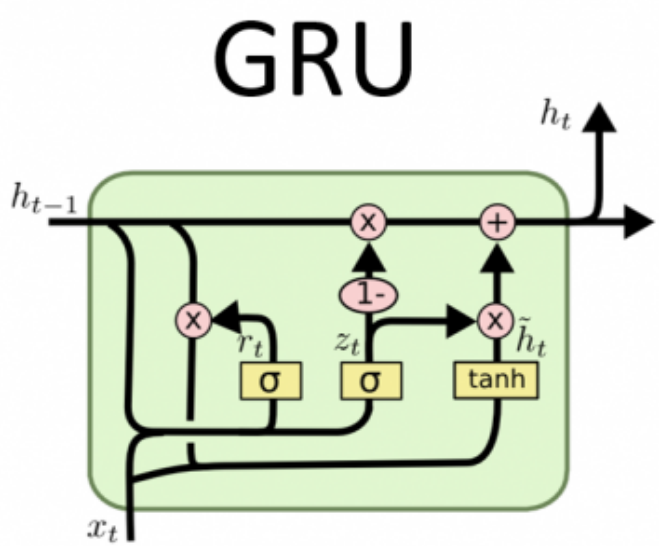
- **Forget Gate** $\rightarrow$ regulates how much past information is discarded and how much is kept
- **Input Gate** $\rightarrow$ updates the cell state
- **Output Gate** $\rightarrow$ defines the value of the next hidden state



Recurrent Neural Networks - 5/5

The **Gated Recurrent Unit (GRU)** network is a type of RNN where each cell consists of:

- **Update Gate** $\rightarrow$ decides how much new information to add
- **Reset Gate** $\rightarrow$ decides how much old information to discard

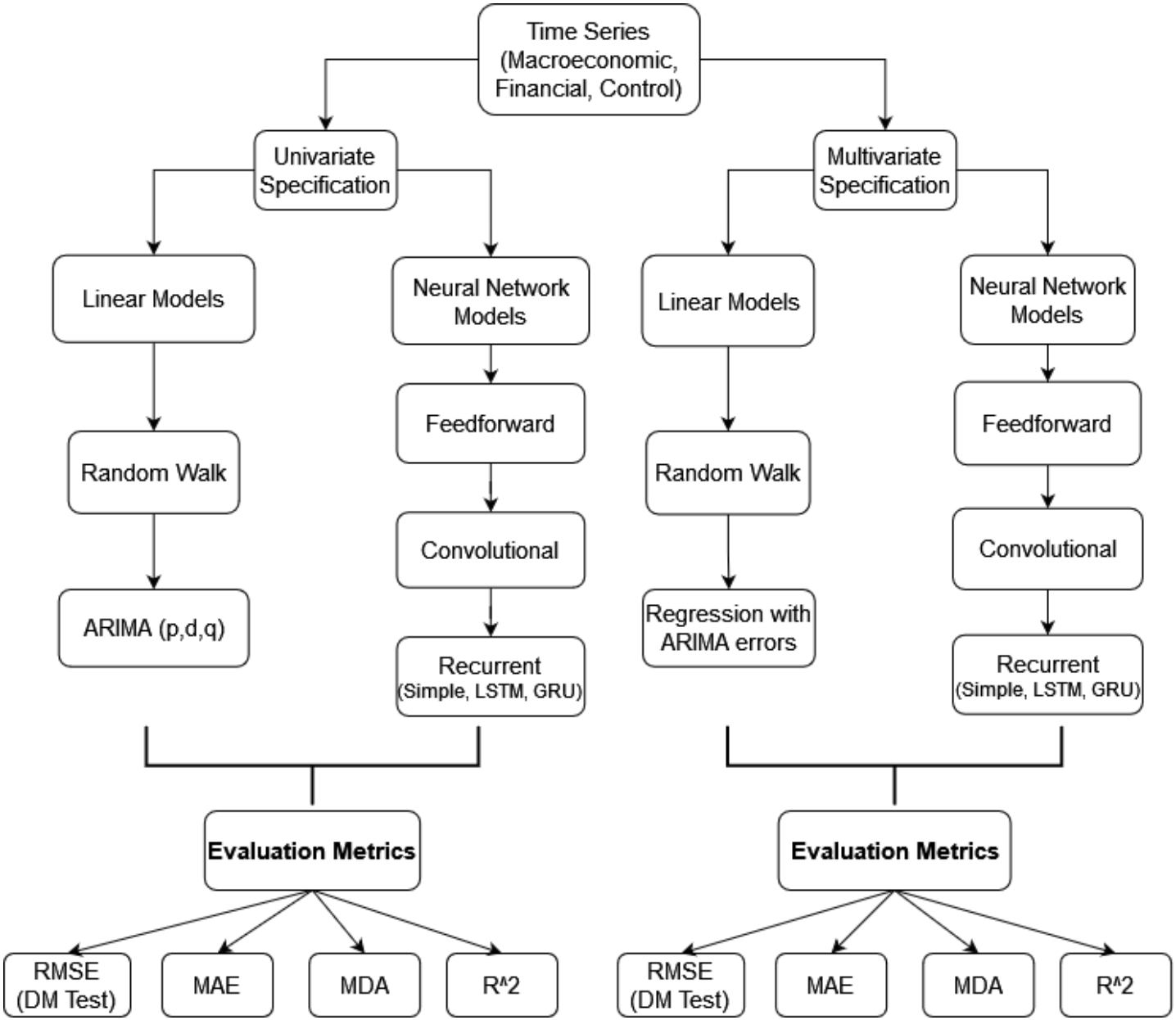


Methodology - 1/4

Three time series were used in this forecasting exercise:

- **Macroeconomic Time Series**: *YoY % change in unemployment* in the United States between July 1955 and December 2019 (*monthly* observations)
  Covariates: *Inflation rate*, *YoY % change in Federal funds rate* and *industrial production*
- **Financial Time Series**: *S&P 500 Trading Volume* between January 4th, 2010 to December 30th, 2020 (*daily* observations)
  Covariates: Daily *FTSE 100 and Nasdaq trading volumes*, *CBOE Volatility Index*
- **Control Time Series**: Daily *visits to a statistical forecasting website* from September 14, 2014 to August 19, 2020
  Covariates: *Unique Visits*, *First Time Visits* and *Returning Visits* to the same website

Methodology - 2/4



Methodology - 3/4

- Each series was split into *training* and *test* sets.
- Fitting of the ARIMA models was carried out using the **auto_arima** function from Python's *pmdarima*
- Neural Networks' **hyperparameters** were selected using a **grid search algorithm** and the models were optimized using **TensorFlow**

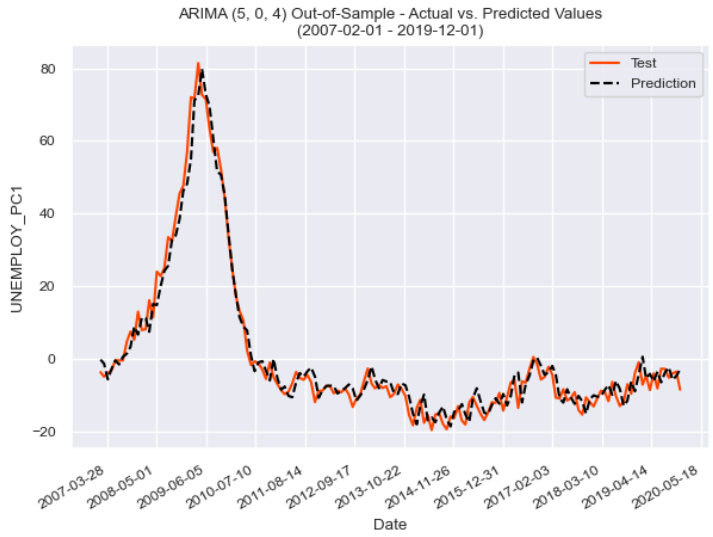
Methodology - 4/4

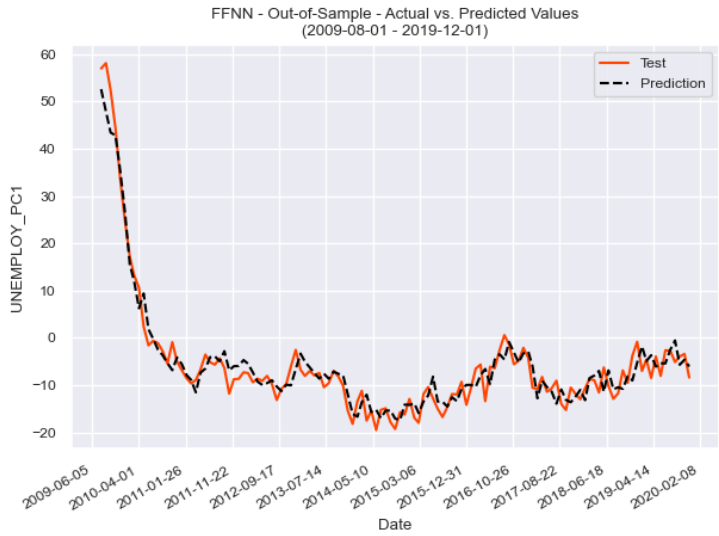
- Each model produces **one-step ahead forecasts** on both the training (in-sample) and test (out-of-sample) sets
- **Walk-forward validation** was used with linear models, while neural networks performed **sliding-window regression**
- To avoid overfitting, the number of hidden layers is **built conditionally on training set size** (larger dataset = deeper network)

Results - Macroeconomic Time Series - 1/2

Univariate

- The ARIMA model has statistically significant lower RMSE **in-sample**
- There is no statistically significant difference between ARIMA and neural networks RMSE **out-of-sample**
- CNN yields a **31% increase in MDA with respect to the ARIMA model** and an **average 34% increase in MDA with respect to other neural networks** (MDA = 0.59)

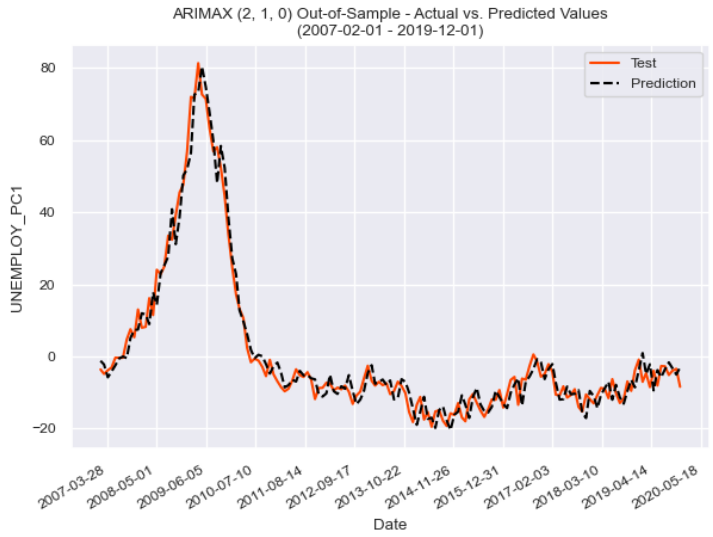


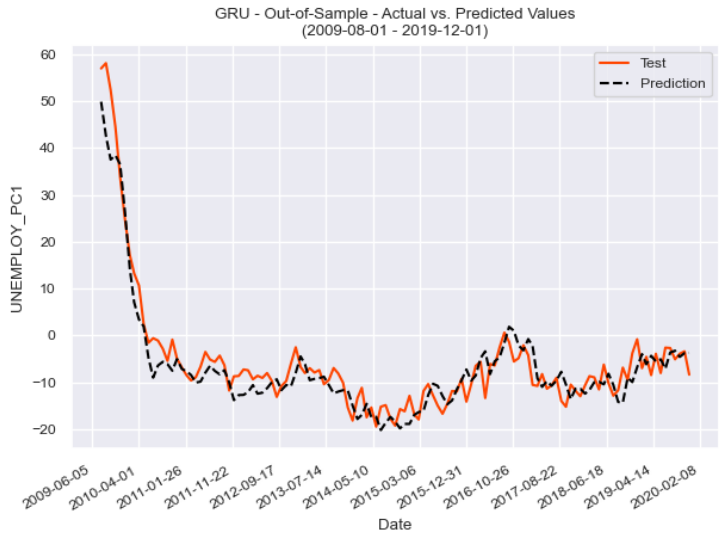


Results - Macroeconomic Time Series - 2/2

Multivariate

- Neural network models have statistically significant lower RMSE **in-sample**
- Again, there is no statistically significant difference between ARIMA and neural networks RMSE **out-of-sample**
- CNN outperforms all other models on MDA

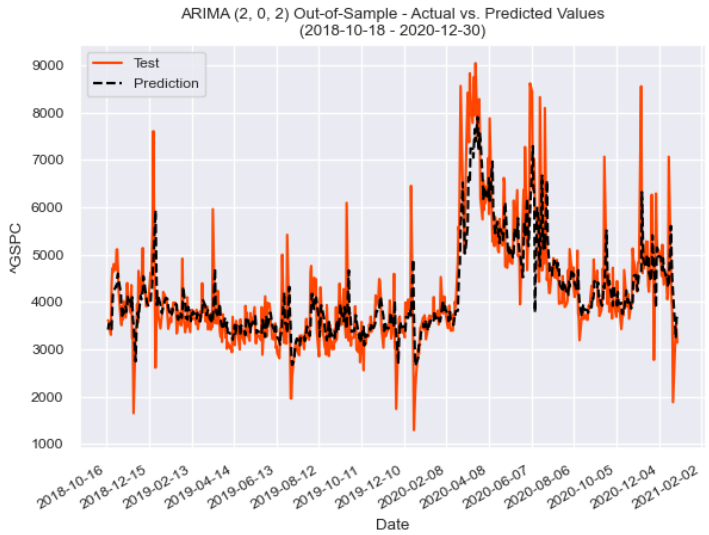


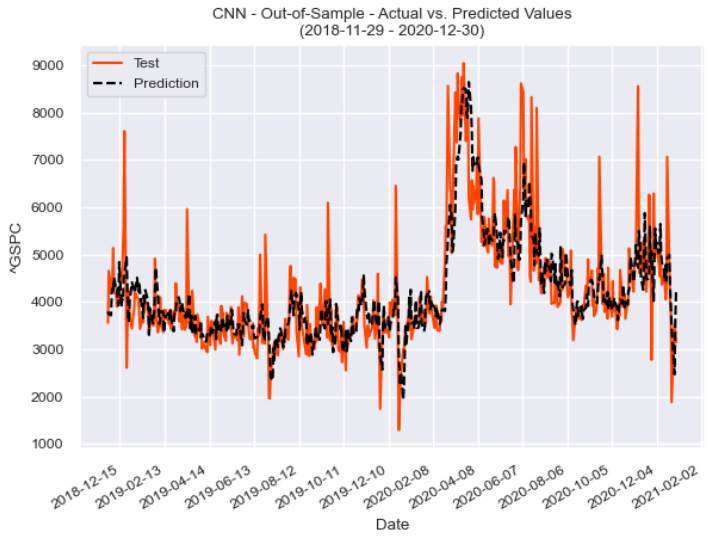


Results - Financial Time Series - 1/2

Univariate

- ARIMA has a significantly lower in-sample RMSE
- Neural networks perform **at least as good as** ARIMA out-of-sample
- GRU yields a **low but statistically significant reduction in RMSE with respect to ARIMA**
- CNN yields a **17% increase in MDA with respect to the ARIMA model**

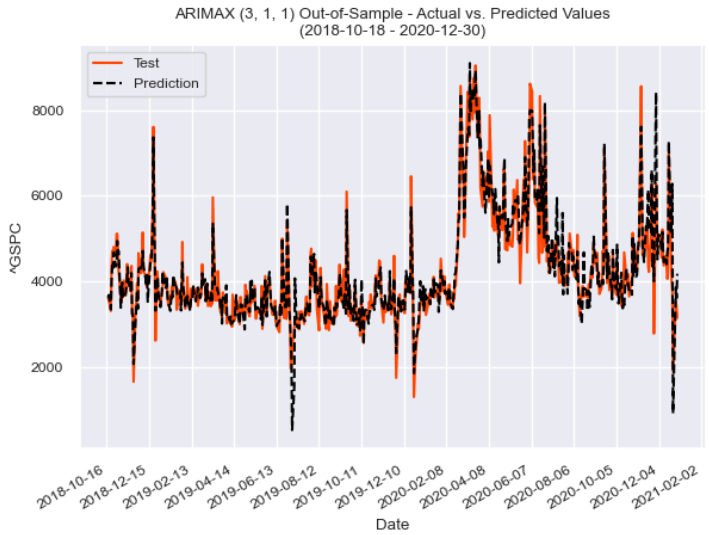


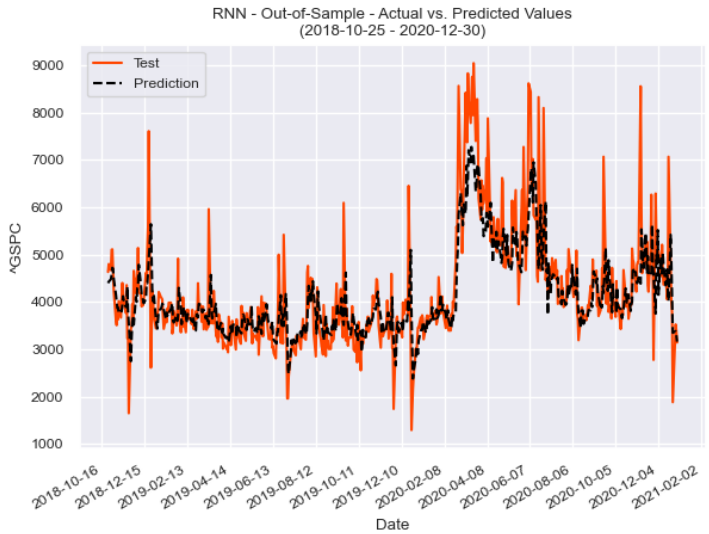


Results - Financial Time Series - 2/2

Multivariate

- Neural network models perform **significantly worse** than ARIMAX on all available metrics

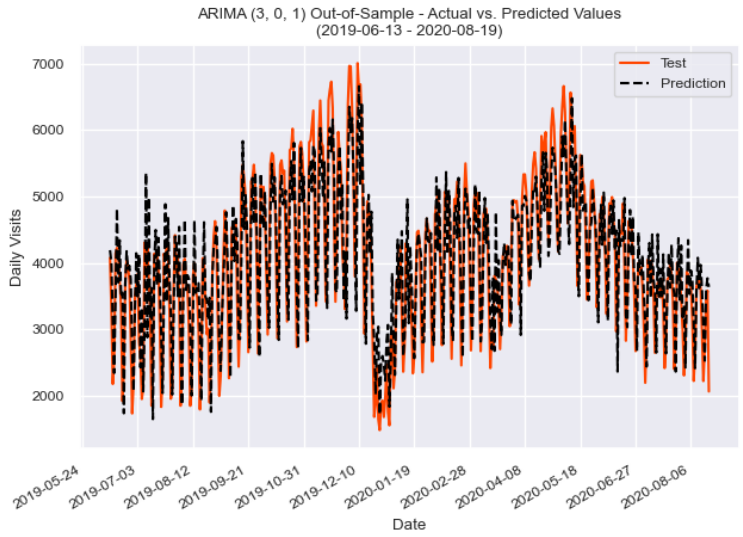


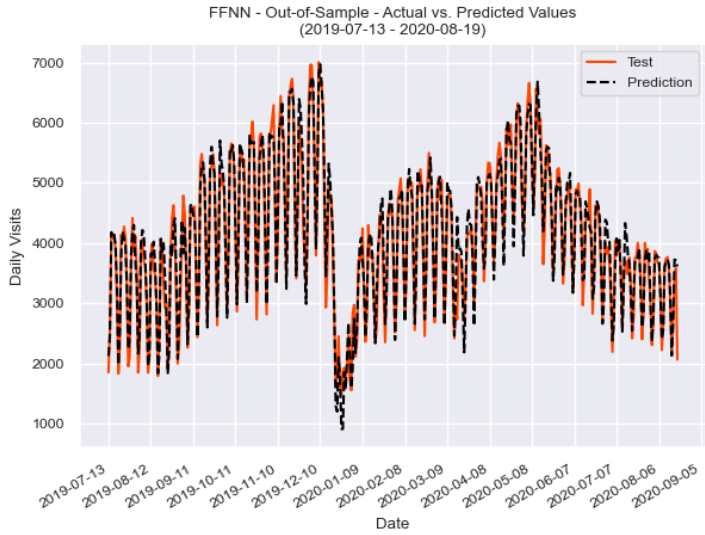


Results - Control Time Series - 1/2

Univariate

- **Neural networks outperform ARIMA** in every metric
- **Lowest** reduction in RMSE is **53%**, **highest** is **55%** (FFNN)
- **Lowest** increase in MDA is **30%** (CNN), **highest** is **34%** (LSTM)

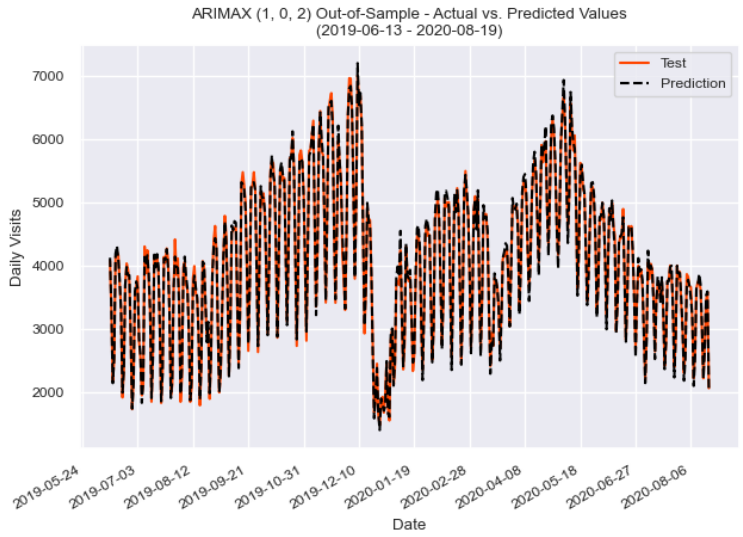


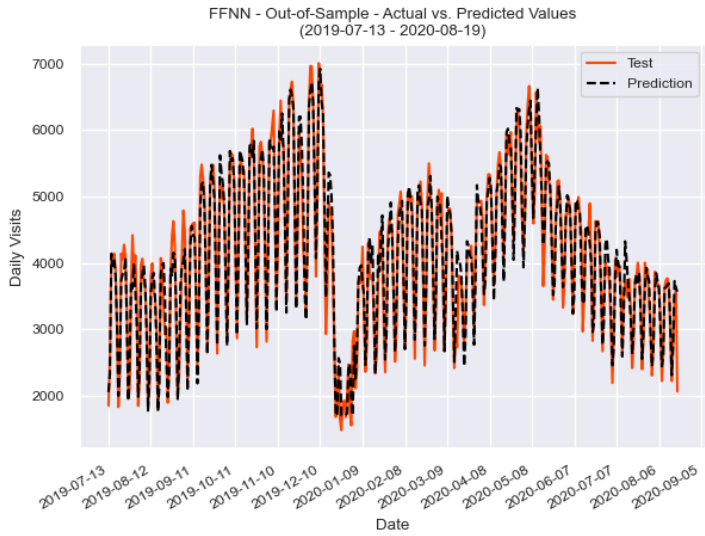


Results - Control Time Series - 2/2

Multivariate

- ARIMAX performs significantly better than neural networks on **all in-sample and out-of-sample metrics** ($R^2 = .99$)
- Neural network out-of-sample performance is *slightly worse* than in the univariate specification





Future Work

Possible improvements on the present approach include:

- Check performance on **different forecasting horizons**
- Operate some **transformation of the input data**
- Widen the grid-search or change **hyperparameter tuning algorithm**
- Check performance of **hybrid models** (e.g. ARIMA-ANN)

Conclusions

- When properly trained, neural networks are able to reach the accuracy of linear models (and sometimes even outperform them) in the **univariate setting**.
- Evidence in the **multivariate setting** is more in favour of linear models.
- CNNs achieve **high accuracy on the MDA measure** in both macroeconomic and financial series.
- **RNNs are not as accurate as expected** in time series forecasting tasks.