An Empirical Study of Neural Networks for Trend Detection in Time Series

Alexandre Miot1 · Gilles Drigout1

Received: 2 June 2020 / Accepted: 2 October 2020 / Published online: 17 October 2020
© Springer Nature Singapore Pte Ltd 2020

Abstract
Even if trend is probably one of the most intuitive notions in time series dynamics, this notion is usually ambiguous and model dependent. We first cast the trend detection problem into a sequence-to-sequence classification problem. Then, we simulate various dynamics with labelled trends. Using those simulated time-series we build a baseline trend estimator showing good performance on various dynamics. Comparing this baseline estimator with various other trend estimators, we find that some recurrent neural networks structures compare favourably against other estimators including convolutional neural networks. Those sequence-to-sequence trend classifiers could be used as efficient basic blocks to build more complex time series estimators.

Keywords
Time series · Trend detection · Classifiers · Neural networks

Introduction
A large number of time series, for example financial assets prices (see [18] or [25]), seem to exhibit “trends” and could be seen as the sum of a piecewise linear function -the trend- and some noise like $Y_t = \mu(t) + \epsilon(t)$ with $\mu(t)$ a trend process and $\epsilon(t)$ a zero expectation noise process. Being able to identify the trend $\mu$ from the noise $\epsilon$ is of paramount importance in many domains like finance, economics, marketing...

Writing $\mu(t) = \eta t$, where $\eta \in \mathbb{R}$, we will focus on trying to predict the sign of $\eta$, $\sgn(\eta) \in \{-1, 0, 1\}$ transforming the regression problem into a classification problem. Importantly, $\eta$ can change at each time step and time series data only becomes available as time passes.

The main contributions of this article are:

- defining the problem as sequence-to-sequence classification with online prediction and building suitably labelled datasets
- on a simple case and using theory showing how some Recurrent Neural Networks (RNN) can detect trends
- identifying empirically which basic building block estimators could be used to build efficient real world trend classifiers

Our aim is not to build the “best” sequence-to-sequence classifier for a specific set of time series exhibiting trends. Indeed, in the light of [42] and [21], it seems to us that identifying building blocks that are generally good at this might be more useful. From theoretical and empirical considerations, we have a particular focus on RNNs. Yet, from [34, 38] or [43] for example, it is most likely that building more complex estimators for real life datasets using transfer learning (see [44] for a review) would also use attention mechanisms and convolution neural networks (CNN).

After examining related works, we describe our general framework. We then define the time series dynamics used in our simulations. Next, we explore the use of RNNs to detect trends. Here, we empirically compare performance of standard RNNs structures. We then build a general purpose trend estimator called RNN baseline. We benchmark

\[ \sgn(x) = \begin{cases} 
-1 & x \leq -\epsilon \\
1 & x \geq \epsilon \\
0 & \text{otherwise}
\end{cases} \]
its performance against other estimators like convolutional networks. Finally, we compare its performance against estimators based on maximum likelihood estimators (MLE) of the modelled dynamic parameters. Mathematical topics and detailed results are provided in the appendix.

**Background and Related Work**

The trend filtering approach, see [22], is a very appealing method. As seen on [35], it is empirically better or equal, but faster, to existing smoothing methods and also offers theoretical guarantees of convergence. From the time series \( (y_t)_{t \in [1, n]} \) a piecewise linear time series is computed \((\hat{y}_t)_{t \in [1, n]}\). Trend labels are obtained using the slope \( \Phi : (\hat{y}_t)_{t \in [1, n]} \mapsto (l_t = \text{sgn}(\hat{y}_{t+1} - \hat{y}_t))_{t \in [1, n]} \). Yet, as we want to predict a trend label for each new data point this method becomes difficult to use. Indeed, the trend filtering problem can be written

\[
\arg \min \frac{1}{2} \| y - \hat{y} \|^2 + \lambda \| D(2) \hat{y} \|_1
\]

where \( D(2) \) is the difference operator of order 2 and \( \lambda \) a regularisation hyper parameter. Also, nothing guarantees the that past labels will not change with time. This could be solved adding coherence constraints but the problem would become difficult to manage. A simple approach, where we refit a model for each new point and predict the trend using the last model gives results comparable to a random classifier. We found no attempt to adapt trend filtering to an online setup where data becomes available as time passes. On the other hand, sequence-to-sequence methods, see [34], have been successfully applied in a Natural Language Processing (NLP) context, for translation [2], text generation using Generative Adversarial Networks (GAN) like [26], speech recognition in [13], semantic labelling in [7]... Yet, these methods are supervised methods where the labels are given and unambiguous. Also, classification of time series is well established, see [20] for a recent survey. Benchmarking time series classifiers from large existing time series datasets like [6] has also been investigated in [27]. Trend classification could be used if the time series was exhibiting only one unambiguous fixed trend. Here, the trend can change at each time step making standard classification of time series less relevant for our problem. To our knowledge, little has been done on the topic of sequence-to-sequence classification of time series for trend classification. We try to address this gap in this paper mixing times series classification and sequence-to-sequence modelling on generated time series. In Fig. 1 we provide a chart summarizing our proposed method.

In this article we focus on the second and third steps (in grey shading). We leave the first step, for example see [39] for a review or [11] for a recent approach, and the fourth step, see [19] or [40] for a general review, for further investigations. Another approach could be a Bayesian approach where we try to identify a latent parameter as the trend. This is not the route we take here except for the Hidden Markov Network (HMM) case [31]. Recent approaches like [10] using one dimensional self-organizing maps might be an alternative route to our approach.

**Framework and Data Set**

Trend as a Sequence-to-Sequence Classification Problem

Let \( \{ Y_t \}_{t \in \mathbb{R}^+} \) be an Itô process

\[
dY_t = \beta(t, Y_t)dt + \sigma(t, Y_t)dW_t
\]

where \( \{ W_t \}_{t \in \mathbb{R}^+} \) is a Wiener process. We can track the changing monotonicity of \( A_t = \int_0^t \beta(s, Y_s)ds \) via the sign of \( \beta(t, Y_t) \). It will be our practical definition of trend and our classification label. For the sake of clarity, the task here is to train an estimator predicting \( \text{sgn}(\beta(t, Y_t)) \) as new data \( Y_t \) comes in.

In the following, we will consider various time series dynamics where we control the sign of \( \beta(t, Y_t) \). This gives us a framework to analyse the performance of various estimators, while controlling for the statistical properties of the dataset.
Time Series Dynamics

In the following, we consider three different types of dynamics:

- a noisy piecewise linear process
- a piecewise Ornstein-Uhlenbeck process [37]
- a Markovian switching process [15]

The first two are piecewise meaning that we divide time into intervals on which the time series follows the chosen dynamic. A simple continuity constraint is applied to “glue” together these different periods.

In the rest of the section we define:

- a time interval \([0, T]\)
- for piecewise processes, a number \(N\) of intervals \([t_i, t_{i+1}], i \in [1, N]\) of possibly different lengths

**Noisy Line Process**

We define a Noisy Line Process\(^2\) as a process \(\{Y_t\}_{t \in [0, T]}\) for which

\[
Y_t = Y_{ti} + \mu_i (t - t_i) + \sigma_i \epsilon_i
\]

where

- \(\mu_i\) is a slope parameter randomly chosen in \(\{-\gamma, \ldots, -\frac{\gamma}{n}, 0, \frac{\gamma}{n}, \ldots, \gamma\}\), where \(\gamma > 0\) is the maximum slope and \(n \in \mathbb{N}^*\)
- \(\sigma_i > 0\) is a noise parameter
- \(\{\epsilon_i\}_{i \geq 0}\) are i.i.d. normal variables

The trend here is given by the sign of \(\mu_i\). Figure 2 displays some possible trajectories.

**Piecewise Ornstein-Uhlenbeck Dynamic**

We define a Piecewise Ornstein-Uhlenbeck Process as a process \(\{Y_t\}_{t \in [0, T]}\) such that

\[
Y_t = Y_{ti} e^{-\alpha_i (t - t_i)} + Y^\infty_i \left(1 - e^{-\alpha_i (t - t_i)}\right)
+ \frac{\sigma_i}{\sqrt{2a_i}} W(e^{2a_i (t - t_i)} - 1) e^{-\alpha_i (t - t_i)}
\]

where \(Y^\infty_i = \frac{\mu_i}{a_i}\) and \(a_i, \mu_i \geq 0\). If the intervals are big enough, \(Y^\infty_i \approx Y_{t_{i+1}}\), and the trend label \(l_i\) will be determined by

\[
\begin{cases}
Y^\infty_i > 1 + \epsilon, & l_i = +1 \text{ up trend} \\
Y^\infty_i < 1 - \epsilon, & l_i = -1 \text{ down trend} \\
\text{otherwise,} & l_i = 0 \text{ no trend}
\end{cases}
\]  \hspace{1cm} (1)

Samples of piecewise Ornstein-Uhlenbeck process are shown on Fig. 3.

**Switching Markovian Dynamic**

The trend is given by a Markov chain \(\{l_i\}_{i \geq 0}\) with finite states \([-1, 0, +1]\). The process \(\{Y_t\}_{t \in [0, T]}\) is defined by

---

\(^2\) or Piecewise Noisy Line.
\[ Y_t = Y_0 \exp \left( \sum_{i=1}^{T} \gamma_i l_i + \sigma_i \varepsilon_i \right) \]

where \( \{\gamma_i\}_{i \in [1,T]} \) is a slope process, \( \{\sigma_i\}_{i \in [1,T]} \) a positive noise process and \( \{\varepsilon_i\}_{i \in [1,T]} \sim \mathcal{N}(0,1). \) In practice, \( \{\gamma_i\}_{i \in [1,T]} \) and \( \{\sigma_i\}_{i \in [1,T]} \) are constant with time, the constant being randomly chosen in a discrete distribution. This process exhibits a rich set of trajectories as seen on Fig. 4.

### Training and Validation Sets

Training sets are made of 1000 time series containing roughly 1000 data points, randomly drawn:

- from either one of the three previous dynamics (see section “Time Series Dynamics”)
- or mixing all previous dynamics. This will be named mixed dynamic in the following

Model selection is made on a validation set composed of 300 time series: 100 samples from each of the three dynamics described in section “Time Series Dynamics”. Each sample has between 500 and 1000 points depending on the dynamics and the draw. Figure 5 shows random samples from the validation set. Hyper-parameters are chosen using a separate test set which is a new random draw of the training set.

### Using Recurrent Neural Networks to Detect Trends

Here we explain the use of RNNs for our classification problem. Drawing from simple intuition, we provably show their benefits in a simple case. All networks have been implemented using PyTorch framework [30] and in all cases training time for a single net was under a couple of minutes using a single GPU.\(^3\)

**From Moving Averages to RNN**

A simple way to detect trend would be to aggregate several moving averages like:

\[ h_t^a = a h_{t-1}^a + (1-a) Y_t \]

with various values of \( a \in [0,1] \) and compare them. Defining \( h_t = (h_t^{a_1}, \ldots, h_t^{a_m}) \) for various values \( a_1, \ldots, a_m \) of the \( a \) parameter.\(^4\) We look at the signs of components of the vector \( W \cdot h_t \) where \( W \) is a given weight matrix. The rows of \( W \) define hyperplanes. The half-spaces determined by \( W \) are given by the signs of the components of \( W \cdot h_t \). Detecting a trend is simply trying to locate \( h_t \) with regards to convex polyhedrons determined by these half-spaces.

Summarizing,

\[ h_t = W_{hh} h_{t-1} + Y_t w_{ih} \]

where \( W_{hh} \in \mathcal{M}_m^+(\mathbb{R}) \) is a positive matrix and \( w_{ih} \in \mathbb{R}_+^m \) a positive vector such that

---

\(^3\) Nvidia GeForce RTX 2080 Ti.

\(^4\) \( h_t \) is going to be the hidden state of our RNN.

\(^5\) Or more probably learnt.
The trend is determined by \( \text{sgn}(W \cdot h_t) \) but we could use any other activation function \( f \) instead of the sign function. These equations are the update equations of

- a vanilla RNN
  - with the identity as activation function
  - with one hidden layer
  - with convex constraints on the weight matrix \( [W_{hh}, W_{ih}] \)
  - with a simple linear layer and activation function \( f = \text{sgn} \)

Such a RNN will be called a “convex net” in the following. This shows that RNNs can be considered as generalizations of some basic moving average comparisons. As a working example, we consider the case of the Noisy Line Process

\[
Y_t = Y_0 + \mu t + \epsilon_t \quad \text{where } \epsilon_t \text{ are independent noise random variables } \mathbb{E}(\epsilon_t) = 0.
\]

For a net with constrained weights it can be shown (see Appendix B for details):

- without trend, \( \mu = 0 \), then \( \{h_t\} \) becomes centred around a variable of finite variance
- with trend, \( \mu \neq 0 \) then \( \{h_t\} \) diverges

If we now introduce a hyperbolic tangent activation function instead of identity:

- if \( \mu = 0 \), near zero the cell is in the linear part and we should expect the state to stay bounded around the origin
- if the trend \( \mu \neq 0 \) then the state should go towards \( \text{sgn}(\mu) \times \infty \) i.e. to navigate near the facets of the \([0, 1]^n \) hypercube

For a practical illustration see Appendix C.

**Overview of RNNs, Training Data and Method**

**Standards Recurrent Neural Nets**

In section “Trend as a Sequence-to-Sequence Classification-Problem”, we turned the trend estimation problem into a sequence-to-sequence classification task, for which RNNs can be used. We consider three standard structures:

- Vanilla RNN as defined in [9]
- LSTM as introduced in [17]
- GRU as introduced in [5]

We refer to [12] for details on sequence-to-sequence learning with RNNs.

**Training RNNs**

For training and validation, we use simulated time series according to section “Training and Validation Sets”. Our aim is to give a precise empirical comparison of these three structures taking into account the possible influence of the training dynamic. We train triplets as follows:

- a RNN chosen among Vanilla, LSTM or GRU
- some meta-parameters like the number of recurrent layers, the dimension of hidden layer(s), dropout (see [33])...
- a time series dynamic chosen among Noisy Line Process, Piecewise Ornstein-Uhlenbeck, Markovian Switch or a mixed dynamic

Each of these triplets is trained and validated against the training and validation sets described in section “Training and Validation Sets”. Parameter details can be found in Appendix D.1. Also, to get more robust results, we did a complete training using two different gradient step optimizations:

- Adam (see [23] for details) as it is commonly used and has some theoretical convergence properties to a stationary point (see [4] for details)
- RMSprop algorithm (see [16] for details)
Method Overview

In the rest of the article we follow the following steps to find a good building block for sequence-to-sequence classification of trend:

1. from theoretical insights, see section “From Moving Averages to RNN” and Appendix B, we choose RNNs as our base classifier
2. trying to avoid ad hoc analysis pitfalls we choose a baseline classifier (choosing RNNs structure, hyper parameters, type of training data)
3. using bootstrap, see [8] for other approaches, compare the baseline with other classifiers

See Fig. 6 for a summary of the method.

Empirical Findings

We train our triplets as described in section “Training RNNs” using a binary loss. Table 1 shows the coefficients of the linear regression of loss against binary variables indicating the training dynamic, the net type, the optimization type and the validation dynamic. Each feature is translated into binary on/off variables with one less modality. The missing modality is on if all others are set to zero. A positive coefficient means that the highlighted feature increases the average loss of the sample, and conversely, a negative coefficient decreases the average loss. Full details can be found in Appendix D.2.

![Figure 7](image)

**Table 1** Ordinary least squares (OLS) model of the loss onto the various features

| Feature [Modality]              | Coefficient |
|--------------------------------|-------------|
| Intercept                      | 0.48        |
| Training dynamic [Markovian Switch] | ≈ 0        |
| Training dynamic [Ornstein-Uhlenbeck] | 0.029      |
| Training dynamic [Noisy Line]   | ≈ 0         |
| Net Type [LSTM]                | 0.037       |
| Net Type [Vanilla]             | 0.17        |
| Optimization [RMSP]            | 0.0234      |
| Validation dynamic [Ornstein-Uhlenbeck] | − 0.1   |
| Validation dynamic [Noisy Line] | − 0.036     |

The left-hand column is the feature column with the specified modality in brackets. Positive coefficient means that the presence of the modality in brackets is detrimental to performance.

**Table 2** Difference of median loss for training type 1—median loss for training type 2 using bootstrapping percentile confidence interval

| Type 1-type 2    | Median loss difference | 1% confidence interval |
|------------------|------------------------|------------------------|
| nl-ou            | − 0.04                 | − 0.19                 | 0.10                      |
| nl-ms            | 0.01                   | − 0.15                 | 0.17                      |
| nl-mix           | − 0.009                | − 0.17                 | 0.15                      |
| ou-ms            | 0.05                   | − 0.10                 | 0.21                      |
| ou-mix           | 0.04                   | − 0.12                 | 0.20                      |
| ms-mix           | − 0.02                 | − 0.20                 | 0.16                      |

In bold, negative values, italic, positive values, in confidence interval columns

**Table 3** Difference of median loss for net structure 1—median loss for net structure 2 using bootstrap percentile confidence interval

| net 1-net 2     | Median loss difference | 1% confidence interval |
|------------------|------------------------|------------------------|
| vanilla-lstm     | 0.14                   | − 0.005                | 0.28                      |
| vanilla-gru      | 0.18                   | 0.04                   | 0.32                      |
| lstm-gru         | 0.05                   | − 0.15                 | 0.25                      |

Highlighted in bolditalic the underperformance of Vanilla RNN

See Fig. 6 for a summary of the method.

![Figure 7](image)
Piecewise Ornstein-Uhlenbeck, we select data from those only and bootstrap. For each bootstrapping iteration, we compute the difference between the medians of losses of one dynamic versus the other. The result can be seen on Table 2. Even if all intervals contain zero, and no robust conclusion can be drawn, the median loss seems lower when training using the Noisy Line or Markovian Switch dynamics.

Net structure are compared using the same bootstrapping procedure in Table 3. Vanilla RNN is consistently worse than LSTM and GRU at 99% confidence level. As a result, in the following, we will ignore triplets with Vanilla RNN.

**Optimizer impact**
Results seem to indicate a slightly better performance of Adam versus RMSprop⁷.

**Net structure and training dynamic interaction**
Using only the triplets where net structure is either GRU or LSTM, we run the same bootstrapping procedure for each datasets on the training dynamic. The results are given in Table 4. All the intervals contain 0 and it is difficult to find a combination which does significantly better than the others.

### RNN Baseline Selection

We would like to choose a RNN estimator that has a good overall performance on validation data. As we have seen, it is difficult to choose a particular training type or net structure (GRU or LSTM) as being significantly better. A way to build a baseline would be for example to pool the estimated probabilities of the best trained estimators. The pooling function here is a simple average of each set of estimated probabilities from the selected estimators⁸. And this, indeed, gives good results on validation data as can be seen in Table 8. We note little difference in performance when pooling more than five estimators.

Yet, choosing such an estimator would give RNNs an advantage compared to other estimators. From previous section “Empirical Findings”, we choose to optimize hyper-parameters for a GRU network trained on the Piecewise Noisy Line dynamic using Adam optimization. Parameter details can be found in Table 9.

It is interesting to note that adding training epochs⁹ seems to slightly increase the median error on the test set but gives a noticeable decrease of the interquartile range by a factor near 25%.

Running the training with hyper-parameters close to the ones obtained by optimization gives fairly similar results. The comparison of the RNN baseline versus the pooled estimator is given in Table 11 and Fig. 10 for the loss distributions.

---

### Table 4 Interaction between the net structure GRU or LSTM and the training type Noisy Line (nl), Piecewise Ornstein-Uhlenbeck (ou) or Markovian Switch (ms)

| type 1-type 2 | Median loss difference | 1% confidence interval |
|---------------|------------------------|------------------------|
| (a) Training bootstrap for LSTM only | | |
| nl - ou | − 0.05 | − 0.25 | 0.15 |
| nl-ms | 0.002 | − 0.18 | 0.19 |
| nl-mix | -0.002 | − 0.19 | 0.19 |
| ou-ms | 0.05 | − 0.10 | 0.20 |
| ou-mix | 0.05 | − 0.12 | 0.22 |
| ms-mix | − 0.005 | − 0.18 | 0.17 |
| (b) Training bootstrap for GRU only | | |
| nl-ou | − 0.025 | − 0.21 | 0.17 |
| nl-ms | 0.05 | − 0.13 | 0.24 |
| nl-mix | 0.06 | − 0.13 | 0.26 |
| ou-ms | 0.08 | − 0.11 | 0.26 |
| ou-mix | 0.09 | − 0.08 | 0.25 |
| ms-mix | 0.008 | − 0.18 | 0.20 |

The loss difference is the loss of the first element of the pair minus the loss of the second

---

⁷ Median loss⁵Adam − median loss⁵RMSprop ≈ − 0.04 with a confidence interval equal to [− 0.27, 0.18].

---

⁸ See [1] for a justification.

⁹ Reasonably from 100 epochs to 200. Going towards 1000 epochs for example gives a marginal improvement in performance but with increasing variance hinting for overfitting.
By “non model based”, we mean estimators which are not based on an explicit modelling of the underlying dynamic. We compare the RNN baseline of section “RNN Baseline Selection” against a simple moving average estimator, its generalization (see section “From Moving Averages to RNN”) and a Convolutional Neural Network (CNN see [24]). Overall, the RNN baseline exhibits much stronger validation performance.

**Comparison with Moving Average**

One of the most intuitive ways to detect trend is to compare the speed of two moving averages. We compare our RNN baseline with both the most simple moving average filtering and the convex net generalization approach.

```
\[
\text{speed} = \begin{cases} 
  s & \text{if } ma_{s} - ma_{f} > \epsilon \\
  f & \text{if } ma_{f} - ma_{s} < -\epsilon \\
  \text{no trend} & \text{otherwise}
\end{cases}
\]
```

where speed ∈ {s, f}. Given ε > 0, a no trend threshold, the trend prediction is made by

```
ma_{s}^{\text{speed}} = \mu_{s}^{\text{speed}} ma_{s}^{\text{speed}} + (1 - \mu_{s}^{\text{speed}}) x_{t},
```

Simple Moving Average

We first compare the RNN baseline with a basic estimator computing two moving averages: a ”s = slow” one and a ”f = fast” one:

```
ma_{s}^{\text{speed}} = \mu_{s}^{\text{speed}} ma_{s}^{\text{speed}} + (1 - \mu_{s}^{\text{speed}}) x_{t},
```

where speed ∈ {s, f}. Given ε > 0, a no trend threshold, the trend prediction is made by

```
ma_{t}^{\text{fast}} - ma_{t}^{\text{slow}} > \epsilon \Rightarrow \text{up trend}
ma_{t}^{\text{fast}} - ma_{t}^{\text{slow}} < -\epsilon \Rightarrow \text{down trend}
\text{otherwise} \Rightarrow \text{no trend}
```

Obviously, the parameters \(\mu_{s}, \mu_{f}, \epsilon\) have a big impact on the estimator performance. Using Bayesian optimization we find the parameters shown in Table 12.

In Fig. 13 we see the loss distribution of the baseline RNN versus the loss distribution of the moving average estimator for all dynamics.

On average, the RNN baseline is consistently better than the moving average estimator as seen on Table 14. The Markovian Switch dynamic is sometimes extremely difficult to understand due to highly volatile regime switching. For this dynamic, we see that both estimators are equally bad which is not unexpected given the task difficulty.
Comparison with Moving Average Generalization

We compare the baseline RNN with the estimator built according to section “From Moving Averages to RNN”. It turns out, a bit surprisingly to us, that the performance is quite poor and much worse than the RNN baseline. Further investigation is needed, but training seems to fail somehow as the trained weights are all very close to zero. As a result, the input plays only a small role in the prediction and surely can’t do much better than a dummy estimator. For reference, basic results are shown in Table 15.

| Dynamic       | RNN  | Generalized moving average |
|---------------|------|---------------------------|
| All           | 0.27 | 0.61                      |
| Ornstein-Uhlenbeck | 0.26 | 0.61                      |
| Noisy Line    | 0.12 | 0.62                      |
| Markovian Switch | 0.47 | 0.61                      |

Comparison with CNN

One dimensional CNN is seen as a good tool to analyse time series. We use a standard CNN structure stacking convolutional layer followed by a pooling layer. To keep nets architecture similar in terms of parameters, we use two layers of convolution + pooling. After optimization, we get hyper-parameters shown in Table 16. Interestingly, both channel and kernel have taken the maximum value in the range we tested$^{10}$. Yet, for our trend detection problem, we are unable to find the supposed general efficiency of CNNs (see [3] for an empirical comparison) as seen on Fig. 17. Actually, CNN performance is barely better than a dummy classifier as seen on Table 18. This might be surprising given that CNN are usually good estimators for time series classification, for example see [43]. We suggest that this is due to the nature of the studied problem. CNNs, which compare a smoothed version of the signal, have a clear advantage when aiming at finding general properties of the time series e.g. global classification of a time series. For a sequence-to-sequence classification, RNNs which have a more “local” approach might be more suitable. In a way, early time series classification might be closer to our approach, see [14] for a survey. To our knowledge, little work has been conducted to compare the performance of the two architectures on the early classification task, see [32] for a practical example.

Model Based Estimators

In this section, we compare the performance of the RNN baseline with classifiers based on maximum likelihood estimation (MLE) of the process parameters. These estimators therefore incorporate knowledge about the underlying data generative process. For each dynamic (see section “Time Series Dynamics”), we compute the MLE estimator of the trend parameter. Then, we use this value at each time step to compute a trend label $\in \{-1, 0, 1\}$.

In sections “Noisy Line Estimator”, “Piecewise OU Process” and “Markovian Switch Process” we recall the formulas of the MLE trend estimators and present their empirical performance in comparison with the RNN baseline. Overall, the baseline shows good performance against these estimators. Theoretical details of MLE derivations are included in Appendix A.

From MLE to Trend Classifier

As a reminder, the training data used for the learning step of the neural networks is comprised of piecewise trajectories of the dynamics and uses randomized model parameters.

$^{10}$ From 3 to 20.
Taking into account this additional randomness in a MLE framework would make the theory more difficult. Here we ignore it, and instead use a sliding window mechanism. For a sliding window $W_i$ of length $\eta$:

- we compute the value of the trend estimator $\hat{\mu}_i$
- we map the value of $\hat{\mu}_i$ to a label using the $\text{sgn}_\epsilon$ function (for a given threshold $\epsilon$) and predict this label with probability 1.

We only need this mechanism for the Noisy Line Process and the Piecewise Ornstein-Uhlenbeck Process.

**Noisy Line Estimator**

**Derivation of MLE Estimator on an Interval**

Deriving the maximum likelihood estimator for the slope $\mu$ is easy as any finite sample $(Y_{i1}, \ldots, Y_{in})$ on a subdivision $t_1 < \ldots < t_n$ is a Gaussian vector with diagonal covariance matrix. Maximizing the MLE of $\mu$ yields to the slope formula (see Appendix A.1 for mathematical details):

$$\hat{\mu}(y_{i1}, \ldots, y_{in}) = \frac{\sum_{i=1}^{n}(t_i - t_0)(y_{i} - y_0)}{\sum_{i=1}^{n}(t_i - t_0)^2}.$$  \hfill (2)

The MLE estimator for the slope follows a normal distribution with mean $\mu$ and variance $\sigma^2\left(\sum_{i=1}^{n}(t_i - t_0)^2\right)^{-1}$. For a subdivision with constant time step $\delta = t_i - t_{i-1}$ the variance is given by:

$$V(\hat{\mu}) = \frac{6\sigma^2}{n(n+1)(2n+1)}.$$  

**Empirical Performance**

Using the same procedure as in section “Non Model Based Estimation”, we compare its performance against our baseline on Fig. 19 and Table 20.

The Noisy Line Estimator is easily overtaken by the RNN baseline even on the simple noisy line dynamic$^{11}$. 

**Piecewise OU Process**

**Derivation of MLE Estimator on an Interval**

We leverage on the theoretical results of [28] and [29] that express the likelihood function in a simple stochastic integral form. In the case of the Ornstein-Uhlenbeck process with linear trend diffusion:

$$dY_t = \mu dt - aY_t dt + \sigma dW_t,$$

the formulas for the estimators are given by:

$$\hat{\mu} = \frac{1}{2} \left( Y_0^2 - Y_T^2 \right) \int_0^T Y_t dt - (Y_T - Y_0) \int_0^T Y_t^2 dt - T \int_0^T Y_t^2 dt$$

$$\hat{a} = \frac{1}{2} \left( Y_0^2 - Y_T^2 - T \right) - (Y_T - Y_0) \int_0^T Y_t dt - T \int_0^T Y_t^2 dt.$$  

To some extent, an analogy can be drawn with classical OLS estimators $\hat{\beta} = (X^T X)^{-1} X^T y$ where the variance scaling term $(X^T X)^{-1}$ corresponds to the term $\left(\int_0^T Y_t dt \right)^2 - T \int_0^T Y_t^2 dt.$

The reader can refer to Appendix A.2 for mathematical details. When dealing with discrete time observations, the integrals are approximated using the sample values and discrete time increments. Simulations show that these estimators exhibit good empirical properties, although they are biased. It can be shown that the biases for both estimators are given by:

$$V(\hat{\mu}) = \frac{6\sigma^2}{n(n+1)(2n+1)}.$$  

$^{11}$ Which is a bit counter-intuitive. The fact that our process is piecewise unlike the MLE derivation is probably responsible for this underperformance.

---

**Fig. 19** Comparing validation loss distribution for Noisy Line Estimator in orange with red median and baseline in blue with cyan median.

**Fig. 20** Median loss for RNN or Noisy Line Estimator for various dynamics on validation set.
In practical applications, the expectations above are computed by first evaluating the residuals $dW_t = dY_t - (\hat{\mu} - \hat{a}Y_t)dt$ over the observed values of $(y_t, \ldots, y_n)$ and then approximating the integrals by summation of the weighted increments.

$\mathbb{E}_{(\mu,a)}[b(\mu)] = \mathbb{E}_{(\mu,a)} \left[ \left( \int_0^T Y_t dW_t \right) \left( \int_0^T Y_t^2 dt \right) - W_T \int_0^T Y_t^2 dt \right]$

$\mathbb{E}_{(\mu,a)}[b(\hat{a})] = \mathbb{E}_{(\mu,a)} \left[ \left( \int_0^T Y_t dW_t \right) - W_T \int_0^T Y_t dt \right] \left( \int_0^T Y_t^2 dt \right)^2 - T \int_0^T Y_t^2 dt \right]$

Empirical Performance

We design a trend estimator using the sliding window mechanism of section “From MLE to Trend Classifier”. We compare its performance against our RNN baseline on Fig. 21 and Table 22. Interestingly, the performance on the Ornstein-Uhlenbeck dynamic is markedly better and comparable to the performance of the RNN on the Ornstein-Uhlenbeck dynamic.

Empirical Performance

We train a three-state HMM with Gaussian emission probabilities on the four time series dynamics (as described in section “Time Series Dynamics”). Performance is similar regardless of the training dynamic. The hidden states of the HMM might not fit in our up, down, flat trend categories. To be able to compute a loss for the HMM, we first map the three-state of $\log \left( \frac{y_{t+1}}{y_t} \right)$ to a probability distribution $\mathcal{N}(\gamma \mu_t, \sigma)$.

Markovian Switch Process

Derivation of MLE Estimator

The Markovian Switch dynamic described in section “Switching Markovian Dynamic” is actually the dynamic of a Hidden Markov Model (HMM) with Gaussian emissions probabilities on log returns:

$\log \left( \frac{y_{t+1}}{y_t} \right) \sim \mathcal{N}(\gamma \mu_t, \sigma)$

where $\{\mu_t\}_{t \geq 0}$ is a simple discrete three-state Markov chain. We then use classic techniques (see [31] for example) to get an estimate of the hidden states which have generated $\{y_t\}_{t \geq 0}$. In practice, the expectations above are computed by first evaluating the residuals $dW_t = dY_t - (\hat{\mu} - \hat{a}Y_t)dt$ over the observed values of $(y_t, \ldots, y_n)$ and then approximating the integrals by summation of the weighted increments.

Empirical Performance

We design a trend estimator using the sliding window mechanism of section “From MLE to Trend Classifier”. We compare its performance against our RNN baseline on Fig. 21 and Table 22. Interestingly, the performance on the Ornstein-Uhlenbeck dynamic is markedly better and comparable to the performance of the RNN on the Ornstein-Uhlenbeck dynamic.
the HMM using the mean of the distribution given the hidden state. We sort them in increasing order and map them to down, flat, up states. We would expect to get a sequence of means being negative, close to zero and positive. Actually, only estimators trained on the mixed or Markovian Switch dynamics exhibit means which are clearly separated into a negative, near zero and positive value. As performance is similar, we use as a baseline the estimator trained on the Markovian Switch dynamic which seems the most natural. Overall, the HMM has a hard time predicting the trend of any dynamic as seen on Fig. 23 or Table 24. This might be a bit surprising especially with the Markovian Switch dynamic. We note however that the best validation score is given when the HMM is trained on the Markovian Switch dynamic.

**Conclusion**

GRU cells achieve good general performance for online sequence-to-sequence trend classification of time series. Yet, if trend classification of time series is seen as a special case (all labels being identical) this result seems to contradict the general view that convolutional structures are the best estimators for time series classification. For example, is there an improvement in CNN performance as the number of different trend labels converges to 1? Also, convex constraints on the weights of the RNN should ensure that the hidden state of the network occupies separate locations depending on the trend. However, we did encounter convergence issues when training RNN while enforcing these constraints. This might also be an area of future research. Finally, from a theoretical standpoint, the update equation of RNN transforms a time process query into a problem of locating the state vector. This seems like ergodicity understood as “the time average of a physical dynamics is equal to its ensemble average”. This point of view has similarities with [36] where the white noise process is replaced by a non-stationary process requiring some further investigation.

**Acknowledgements** The authors would like to thank Lionel Massouard and Sandrine Ungari for their comments and fruitful discussions.

**Compliance with Ethical Standards**

**Conflict of interest** The authors declare that they have no conflict of interest.

**Appendix A: MLE estimators theory**

**A.1: Simple Noisy Line Estimator**

On a discrete time grid \(t_0 < t_1 < \ldots < t_n\) we consider the “noisy line” dynamics:

\[
Y_t = y_{t_0} + \mu(t_t - t_0) + \epsilon_i
\]

where \((\epsilon_t)_{t_0 \leq t \leq t_n}\) is a collection of i.i.d. normal random variables \(\mathcal{N}(0, \sigma^2)\). One can easily show that \((Y_{t_1}, \ldots, Y_{t_n})\) is a Gaussian vector with diagonal covariance matrix. The likelihood function is expressed as

\[
\begin{align*}
    L(\mu, \sigma^2 | y_{t_1}, \ldots, y_{t_n}) &= (\sigma \sqrt{2\pi})^{-n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \cdot \\
    &= (\sigma \sqrt{2\pi})^{-n} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right) \\
    &= \frac{1}{(2\pi \sigma^2)^{n/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_{ti} - y_{t_0} - \mu(t_i - t_0))^2\right)
\end{align*}
\]

Let \(l = \log L\) denote the log-likelihood. Solving \(\frac{\partial l}{\partial \mu} = 0\) yields to the expression (2). By expressing \(\hat{\mu}\) as

\[
\hat{\mu}(Y_{t_1}, \ldots, Y_{t_n}) = \frac{\sum_{i=1}^{n} (t_i - t_0) / Y_{ti} - y_{t_0} - \mu(t_i - t_0)^2)}{\sum_{i=1}^{n} (t_i - t_0)^2}
\]

one can show that \(E(\hat{\mu}) = \mu\) and \(V(\hat{\mu}) = \sigma^2(\sum_{i=1}^{n} (t_i - t_0)^2)^{-1}\).

Simulations of trajectories (3) to compute samples estimates of \(\mu\) are in agreement with the above result.

**A.2: Linear Trend with Diffusion Estimator**

We consider the diffusion with the dynamics

\[
dY_t = \mu dt - a Y_t dt + dW_t
\]

where \(W\) is a Wiener process and \(\mu, a\) are unknown scalar quantities to be estimated from observations. In an infinitesimal time period \(dt\), the price moves linearly by an amount \(\mu dt\) and fluctuates around this trend term by an amount equal to \(-a Y_t dt + dW_t\).

We seek to construct estimation techniques for \(\mu\) and \(a\). In the setting of discrete observations \((y_{t_0}, y_{t_1}, \ldots, y_{t_n})\) various estimation approaches can be used. For instance, one can first de-trend the observed price series and then estimate the fluctuation speed \(a\) using standard OLS techniques. The drawbacks of such an approach are twofold. Firstly, estimation is conducted regardless of the joint distribution of \((\hat{\mu}, \hat{a})\). Secondly, classical OLS assumptions are most likely to fail in the case of a diffusion price process. As a consequence of non-stationarity of residuals, it can be shown that the OLS estimator of \(a\) is biased. Such behaviours are studied in depth in [41].

Our approach follows the results from [29] in which the authors estimate drift parameters in a continuous likelihood maximization framework. Let us recall the main results from [28] and [29].
Theorem 1 Let $Y = (Y_t)_{0 \leq t \leq T}$ be a process satisfying the stochastic differential equation (SDE)
\[
dY_t = a(t, Y_t)dt + dW_t, \quad Y_0 = 0, \quad 0 \leq t \leq T
\]
where $a : t \mapsto a(t, \cdot)$ is a non-anticipative function.

Under the assumption that $\mathbb{P}$-almost surely,
\[
\int_0^T a(t, Y_t)^2dt < \infty, \quad \int_0^T a(t, W_t)^2dt < \infty
\]
then the measures $\mu_Y$ and $\mu_W$ are equivalent. Moreover, $\mathbb{P}$-almost surely, the Radon-Nikodym derivative of $\mu_Y$ with respect to $\mu_W$ is given by:
\[
\frac{d\mu_Y}{d\mu_W}(Y_t) = \exp \left( \int_0^t a(s, Y_s)dY_s - \frac{1}{2} \int_0^t a(s, Y_s)^2ds \right).
\]

The reader can refer to [28], Theorem 7.7, for a formal statement and proof. The issue of the drift parametric estimation is addressed in [29] by considering the diffusion process:
\[
dY_t = \theta a(t, Y_t)dt + dW_t.
\]

Using the result above with $a(t, x) = \theta a(t, x)$ and under similar assumption on $a$ one can show that the measures $\mu_Y^\theta$ and $\mu_W$ are equivalent and that the likelihood function $L_\theta(Y)$ can be expressed as
\[
L_\theta(Y) = \exp \left( \theta \int_0^t a(s, Y_s)dY_s - \frac{\theta^2}{2} \int_0^t a(s, Y_s)^2ds \right).
\]

It is easy to show that the log-likelihood is a concave function of the parameter $\theta$ and that its maximum is attained for $\theta^*$ such that $L_{\theta^*}(\theta^*) = 0$.

As a consequence, under the assumption that
\[
\int_0^T a(t, Y_t)^2dt < \infty, \quad \int_0^T a(t, W_t)^2dt < \infty
\]
and under the condition that $\mathbb{P}_\theta$-a.s. $\int_0^T a(t, Y_t)dt > 0$ the maximum likelihood estimation of $\hat{\theta}(Y)$ is expressed as:
\[
\hat{\theta}(Y) = \frac{\int_0^T a(t, Y_t)dY_t}{\int_0^T a(t, Y_t)^2dt}.
\]

When dealing with real data, the numerical value of $\hat{\theta}$ is computed using numerical integration techniques along the observed path $(y_{t_0}, \ldots, y_{t_n})$.

From now on, we adopt the lighter notations:
\[
I_{\theta}(\alpha) := \int_0^T a(t, Y_t)dY_t,
\]
\[
I_{\alpha}(\theta) := \int_0^T a(t, Y_t)dt
\]
so that the MLE estimator (8) is expressed as $I_{\theta}(\alpha) / I_{\alpha}(\alpha^2)$.

For most drift functions $\alpha$ the estimator $\hat{\theta}$ has non-zero bias. An approximation of the bias can be easily derived by substituting the expression of $dY_t$ in (8):
\[
\hat{\theta}(Y) = \frac{\int_0^T a(t, Y_t)dY_t}{\int_0^T a(t, Y_t)^2dt}
\]
\[
= \frac{\int_0^T a(t, Y_t)(\theta a(t, Y_t)dt + dW_t)}{\int_0^T a(t, Y_t)^2dt}
\]
\[
= \theta + \frac{\int_0^T a(t, Y_t)dW_t}{\int_0^T a(t, Y_t)^2dt}.
\]

Hence the bias $b(\hat{\theta}(Y)) = E_\theta(\hat{\theta} - \theta)$ can be computed by approximating the expectation:
\[
E_\theta \left( \frac{\int_0^T a(t, Y_t)dW_t}{\int_0^T a(t, Y_t)^2dt} \right).
\]

In the following, we extend (6) to the 2D parametric drift case:
\[
dY_t = (\theta_1 a_1(t, Y_t) + \theta_2 a_2(t, Y_t))dt + dW_t.
\]

Theorem 2 Let $(Y_t)_{t \geq 0}$ be a process satisfying the diffusion equation (9) where both $\alpha_1$ and $\alpha_2$ satisfy the condition (7).

Under the condition that $\mathbb{P}_\theta$-a.s. $\int_0^T a_i(t, Y_t)dt > 0$, $i = 1, 2$ the maximum likelihood estimation of $\hat{\theta}(Y)$ is expressed as:
\[
\hat{\theta}_i(Y) = \frac{I_{\theta}(\alpha_i)I_{\theta}(\alpha_2) - I_{\theta}(\alpha_i)I_{\theta}(\alpha_2)}{I_{\theta}(\alpha_1)I_{\theta}(\alpha_2)^2 - I_{\theta}(\alpha_1)I_{\theta}(\alpha_2^2)},
\]
\[
\hat{\theta}_2(Y) = \frac{I_{\theta}(\alpha_2)I_{\theta}(\alpha_1) - I_{\theta}(\alpha_2)I_{\theta}(\alpha_1^2)}{I_{\theta}(\alpha_1)I_{\theta}(\alpha_2)^2 - I_{\theta}(\alpha_1^2)I_{\theta}(\alpha_2^2)}.
\]

Proof By substituting the drift term in (9) into (5) one obtains
\[
\frac{d\mu_Y}{d\mu_W}(Y) = e^{\theta_1 I_{\theta}(\alpha_1) + \theta_2 I_{\theta}(\alpha_2) - \frac{\theta_1^2}{2} I_{\theta}(\alpha_1^2) - \frac{\theta_2^2}{2} I_{\theta}(\alpha_2^2) - \theta_1 \theta_2 I_{\theta}(\alpha_1 \alpha_2)}.
\]

Let $I_\theta(Y)$ denote the log-likelihood. To ensure the convexity of $I_\theta$ one must verify that its Hessian matrix $H = (\partial^2 I_\theta)_{1 \leq i, j \leq 2}$ is definite negative.

Deriving the Hessian yields to
\[
H = -\left( \begin{array}{cc}
I_{\alpha_1^2} & I_{\alpha_1 \alpha_2} \\
I_{\alpha_1 \alpha_2} & I_{\alpha_2^2}
\end{array} \right)
\]
hence of the form

\[ H = \begin{pmatrix} A & C \\ C & B \end{pmatrix}. \]

The eigenvalues of \( H \) are given by

\[ \lambda_1 = -\frac{1}{2}(A + B + \sqrt{(A - B)^2 + 4C^2}) < 0, \]
\[ \lambda_2 = \frac{1}{2}(A - B + \sqrt{(A - B)^2 + 4C^2}). \]

For its largest eigenvalue \( \lambda_2 \) to be negative is equivalent to \( C^2 < AB \), that is \( I_2(a_1 a_2) < I_2(a_1^2) \). This latter expression is equivalent to the Cauchy-Schwarz inequality. Hence these conditions are \( P^\rho \)-a.s. verified, ensuring the concavity to be negative is equivalent to \( C^2 < AB \).

We now consider the diffusion:

\[ dY_t = \mu dt - aY_t dt + dW_t. \]  

From the results above the MLE estimators for both \( \mu \) and \( a \) are given by:

\[ \hat{\mu} = \frac{1}{2}(Y_T^2 - T) \int_0^T Y_t dt - (Y_T - Y_0) \int_0^T Y_t^2 dt \]
\[ \int_0^T Y_t dr \]  
\[ \hat{a} = \frac{1}{2}(T (Y_T^2 - Y_0^2) - (Y_T - Y_0) \int_0^T Y_t Y_t^2 dt \]
\[ \int_0^T Y_t^2 dr \]  

To obtain these formulas we use the formulas (10) and (11) with \( \alpha_1(t, x) = 1, \alpha_2(t, x) = -x \) and \( \alpha_2 = a \). Using Ito’s Lemma one can show that:

\[ I_1(a_1) = \int_0^T dY_t = Y_T - Y_0 \]
\[ I_2(a_2) = -\int_0^T Y_t dY_t = \frac{1}{2}(T (Y_T^2 - Y_0^2) \]
\[ I_1(a_1 a_2) = -\int_0^T Y_t dr \]
\[ I_2(a_1^2) = T \]
\[ I_2(a_2^2) = \int_0^T Y_t^2 dr. \]

\[ h_t = W_{hh} h_{t-1} + Y_t w_{ih} \]
where \( W_{hh} \in \mathcal{M}_{n} (\mathbb{R}) \) and \( w_{ih} \in \mathbb{R}^n \) and

\[ \forall i, j \in [1, n] (W_{hh})_{ij} > 0 \]
\[ \forall i \in [1, n] (w_{ih}) > 0 \]

\[ ||W_{hh}||_1 = \max_{1 \leq i \leq n} \sum_{j=1}^m |(W_{hh})_{ij}| < 1 \]

\[ ||W_{hh}||_\infty = \max_{1 \leq i \leq n} \sum_{j=1}^m |(W_{hh})_{ij}| < 1. \]

Let’s consider \( Y_t \) a simple noisy line process \( Y_t = Y_0 + \mu t + \epsilon_t \) we have:

\[ h_t = W_{hh} h_{t-1} + \mu t w_{ih} + \epsilon_t w_{ih} + Y_0 w_{ih} \]
\( \mu \) being the trend and \( \{\epsilon_t\}_{t \geq 0} \) an i.i.d process with expectation equal to zero and unit variance.

Without trend i.e. \( \mu = 0 \), we have

\[ h_t = W_{hh} h_{t-1} + Y_0 w_{ih} + \epsilon_t w_{ih} . \]

We note \( \lambda_{pf} > 0 \) the Perron-Frobenius eigenvalue of \( W_{hh} \).
All eigenvalues of \( W_{hh} \) different from \( \lambda_{pf} \) satisfy \( |\lambda| < \lambda_{pf} \). If \( y = (y_i) \sim U(0, 1) \) is a corresponding eigenvector then

\[ \forall i \in [1, n] \sum_j W_{ij} y_j = \lambda_{pf} y_i . \]

Noting \( y_{\text{max}} = \max \{y_1, y_2, \ldots, y_n\} \)

\[ \forall i \in [1, n] \quad \lambda_{pf} y_i = \sum_j W_{ij} y_j \leq \sum_j W_{ij} y_{\text{max}} < y_{\text{max}} . \]

So \( \lambda_{pf} < 1 \) and \( I - W_{hh} \) is bijective. We can define \( h^* \) by

\[ (I - W_{hh}) h^* = Y_0 w_{ih} \]

Then,

\[ h_t - h^* = W_{hh} (h_{t-1} - h^*) + \epsilon_t w_{ih} . \]

Simplifying notations with \( \tilde{h}_t = h_t - h^* \), \( W = W_{hh} \) and \( \tilde{\epsilon}_t = \epsilon_t w_{ih} \)

\[ \forall t \in [1, n] \quad \tilde{h}_{t+1} = W \tilde{h}_t + \sum_{k=2}^{t+1} W^{t+1-k} \tilde{\epsilon}_k \]

where \( \tilde{\epsilon}_t \) has zero mean and variance equal to \( \omega = w_{ih} W_{hh} \).

Defining

\[ 12 \] The state of a vanilla RNN with identity activation function, no biases and constrained weights.
\[ R_t = \sum_{k=2}^{t} W^{t-k} \xi_k = \sum_{k=0}^{t-2} W^k \xi_{t-k}, \]
\[ \mathbb{E}(R_t) = 0 \text{ and } V(R_t) = \sum_{k=2}^{t} W^{t-k} \omega(W^T)^k. \{R_t\}_{t \geq 1} \text{ converges in } L^2 \text{ as} \]
\[ \int |(W^k \xi_{t-k})|^2 = (W^k \omega(W^T)^k)_{ii} \leq |W^k \omega(W^T)^k|_\infty, \]
and
\[ |W^k \omega(W^T)^k|_\infty \leq \|\omega\|\|W\|_\infty \|W\|_1 \]
\[ (W^k h_1)_{t \in N} \text{ converges to } 0 \text{ in } L^2 \text{ as} \]
\[ \lim_{t \to +\infty} \mathbb{E}(W^k h_1) = 0 \]
\[ \lim_{t \to +\infty} V(W^k h_1) = 0. \]

With a trend i.e. \( \mu \neq 0 \), noting \( h^0 \) the previous no trend solution and \( h^\mu \) the process with a trend \( \mu \), and defining
\[ \delta_t = h^\mu_t - h^0_t \]
it is easily seen that
\[ \delta_t = W_{hh} \delta_{t-1} + \mu_t w_{ih}. \]
Noting that \( \delta_0 = 0 \)
\[ \delta_t = \mu \sum_{k=0}^{t} k W^{t-k} w_{ih} \]
\[ P_{k,j} \geq 0, \text{ so if } \mu > 0 \]
\[ \delta_t \geq \mu t w_{ih} \xrightarrow{t \to +\infty} +\infty. \]
Similarly, if \( \mu < 0 \),
\[ \delta_t \leq \mu t w_{ih} \xrightarrow{t \to +\infty} -\infty. \]
So,
\[ h^\mu_t = h^0_t + \delta_t \]
converges in \( L^2 \) diverges almost surely.

**Appendix C: Visual Representation of Hidden State**

We plot the hidden state \( h_t \in \mathbb{R}^5 \) of a vanilla network previously trained on a randomly chosen dynamic. The hidden state is obtained by running through three different Noisy Line Processes (respectively up trending, without trend and down trending). We see, on Fig. 25, that the state goes right as time goes for the down trend, stays around zero without trend and goes left for the uptrend. The state has been projected into the plane of the first two eigenvectors to get a two dimensional plot.

**Appendix D: Technical Details**

**D.1: RNN Training Details**

See Fig. 26.

| RNN type         | Vanilla, LSTM, GRU |
|------------------|---------------------|
| Number of layers | 1, 2                |
| Learning rate    | 0.01, 0.1, 1.0      |
| Dropout          | 0, 0.1              |

(a) Training hyper-parameters for RNNs

| Time-series dynamic | Piecewise Noisy Line | Piecewise Ornstein-Uhlenbeck | Markovian Switch | Mixed Dynamic |
|---------------------|----------------------|------------------------------|------------------|---------------|
| Global “noise level”| 1, 5                 |                              |                  |               |
| Number of samples   | 1,000                |                              |                  |               |

(b) Training hyper-parameters for time series dynamics

| Dynamic                      | Min Length | Max Length |
|------------------------------|------------|------------|
| Piecewise Noisy Line         | 50         | 1000       |
| Piecewise Ornstein-Uhlenbeck | 80         | 2400       |
| Markovian Switch             | 500        | 1000       |
| Mixed Dynamic                | 1000       | 1000       |

(c) Sequence lengths

**D.2: RNN Empirical Findings**

See Fig. 27.
Fig. 27 Loss OLS left hand column is the feature column with the specified modality in bracket

| Feature[Modality] | Coefficient | Std Err | t-statistic | P-value | 5% confidence interval |
|-------------------|-------------|---------|-------------|---------|------------------------|
| Intercept         | 0.4840      | 0.002   | 260.514     | 0.000   | 0.480 - 0.488          |
| Training dynamic[Markovian Switch] | 0.0006 | 0.002 | 3.852 | 0.000 | 0.003 - 0.010 |
| Training dynamic[Ornstein-Uhlenbeck] | 0.0290 | 0.002 | 16.880 | 0.000 | 0.026 - 0.032 |
| Training dynamic[Noisy Line] | 0.0017 | 0.002 | 1.002 | 0.000 | 0.004 - 0.005 |
| Net type[LSTM]    | 0.0369      | 0.002   | 24.416     | 0.000   | 0.034 - 0.040          |
| Net type[Vanilla] | 0.1742      | 0.002   | 116.118    | 0.000   | 0.171 - 0.177          |
| Optimization[RMSP] | 0.0234 | 0.001 | 19.489 | 0.000 | 0.021 - 0.026 |
| Testing[Ornstein-Uhlenbeck] | -0.1006 | 0.001 | -68.400 | 0.000 | -0.103 - -0.098 |
| Testing[Noisy Line] | -0.0357 | 0.001 | -24.271 | 0.000 | -0.039 - -0.033 |

References

1. Adamčík M. The information geometry of bregman divergences and some applications in multi-expert reasoning. Entropy. 2014;6338–6381(12):2014.
2. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. In: International conference on learning representations (ICLR) 2015 as oral presentation. 2014. (accepted)
3. Bai S, Zico KJ, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv e-prints; 2018.
4. Barakat A, Bianchi P. Convergence and dynamical behavior of the adam algorithm for non convex stochastic optimization. In: arXiv/ statML. 2018.
5. Cho K, Merrienboer B van, Gulcehre C, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.
6. Dau HA, Keogh E, Kamgar K, Michael YCC, Zhu Y, Gharabghi S, Ratanamahatana CA, Yampng HB, Begum N, Bagnall A, Mueen A, Batista G, Hexagon ML. The ucr time series classification archive. 2018. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.
7. Daza A, Frank A. A sequence-to-sequence model for semantic role labeling. In: Proceedings of the third workshop on representation learning for NLP. 2018.
8. Demšar J. Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res. 2006;7:1–30.
9. Elman JL. Finding structure in time. Cogn Sci. 1990;14:179–211.
10. Vincent F, Matthias H, Francesco L, Heiko S, Gunnar R. Som-vae: interpretable discrete representation learning on time series. 2019. arXiv:1806.02199.
11. Grabocka J, Schmidt-Thieme L. Neuralwarp: time-series similarity with warping networks. In: Advances in neural information processing systems 31 (NIPS 2018). 2018.
12. Graves A. Supervised sequence labelling with recurrent neural networks. Berlin: Springer; 2012.
13. Graves A, Mohamed AR, Hinton GE. Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing, pp 6645–6649; 2013.
14. Gupta A, Hari PG, Bhaskar B, Tanima D. Approaches and applications of early classification of time series: a review. IEEE Trans Artif Intell. 2020. https://doi.org/10.1109/TAI.2020.3027279.
15. Hamilton JD. A new approach to the economic analysis of non-stationary time series and the business cycle. Econometrica. 1989;57(2):357–84.
16. Hinton G. Lecture 6a: overview of mini-batch gradient descent. In: Coursera class on RMSprop method. 2012.
17. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9(8):1735–80.
18. Hurst B, Yao HO, Lasse HP. A century of evidence on trend-following investing. In: SSRN. 2017.
19. Ismail FH, Forestier G, Weber J, Idoumghar L, Muller P-A. Transfer learning for time series classification. In: 2018 IEEE international conference on big data (Big Data). 2018.
20. Ismail FH, Forestier G, Weber J, Idoumghar L, Muller P-A. Deep learning for time series classification: a review. Data Min Knowl Discov. 2019:2019:3.
21. Kawaguchi K, Leslie PK, Yoshua B. Generalization in deep learning. In: Mathematics of deep learning. Cambridge University Press; 2017. (to appear)
22. Kim S-J, Koh K, Boyd SP, Gorinevsky DM. L1 trend filtering. SIAM Rev. 2009;51:339–60.
23. Kingma DP, Ba J. Adam: a method for stochastic optimization. arXiv:1412.6980. 2014.
24. LeCun Y, Bengio Y. Convolutional networks for images, speech, and time time-series. In: Arbib Michael A, editor. The Handbook of Brain Theory and Neural Networks., chapter Convolutional Networks for Images, Speech, and Time SeriesCambridge, MA, USA: MIT Press; 1998. p. 255–8.
25. Lempérière Y, Deremble C, Seager P, Potters M, Bouchaud J-P. Two centuries of trend following. J Investment Strat. 2014;3(3):41–61.
26. Lin K, Li D, He X, Zhang Z, Sun M-T. Adversarial ranking for language generation. In: Advances in neural information processing systems, vol 30 (NIPS 2017). 2017.
27. Lines J, Taylor S, Bagnall A. Time series classification with hive: the hierarchical vote collective of transformation-based ensembles. ACM Trans Knowl Discov Data. 2018;12:5.
28. Liptser RS, Shiryaev AN, Karatzas Y, Yor M. Statistics of random processes I: General theory, vol. 5. Berlin: Springer Science & Business Media; 2013.
29. Liptser RS, Shiryaev AN, Karatzas Y, Yor M. Statistics of random processes II: applications, vol 30 (NIPS 2017). 2017.
30. Paszke A, et al. Pytorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst. 2019;32:8024–35 (Curran Associates, Inc.).
31. Rabiner LR. A tutorial on hidden markov models and selected applications in speech recognition. In Proceedings of the IEEE; 1989.
32. Rußwurm, M., Tavenard, R., Lefèvre, S., Körner, M. Early classification for agricultural monitoring from satellite time series. In: International conference on machine learning AI for social good workshop, Long Beach, United States. 2019.

33. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15(1):1929–58.

34. Sutskever, I., Vinyals, O., Quoc, V. L. Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems 27 (NIPS 2014); 2014.

35. Tibshirani, Ryan J. Adaptive piecewise polynomial estimation via trend filtering. Ann Stat. 2014;42(1):285–323.

36. Adrian, T., Friedrich, L., Kurt, H. On the ergodicity and stationarity of the arma(1,1) recurrent neural network process. In: Technical report, adaptive information systems and modelling in economics and management science. 2000.

37. Uhlenbeck, GE, Ornstein, LS. On the theory of the Brownian motion. Phys Rev. 1930;36:823–41.

38. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, AN, Kaiser, L., Polosukhin, I. Attention is all you need. Neural Inf Process Syst. 2017;2007:5998–6008.

39. Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E. Experimental comparison of representation methods and distance measures for time series data. Data Min Knowl Discov. 2012;26:275–309.

40. Wilson, G., Cook, DJ. A survey of unsupervised deep domain adaptation. ACM Trans Intell Syst Technol. 2020;11(5):1–46.

41. Jun, Y. Bias in the estimation of the mean reversion parameter in continuous time models. J Econometrics. 2012;169(1):114–22.

42. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O. Understanding deep learning requires rethinking generalization. In: International conference on learning representations (ICLR) 2017. 2016.

43. Zhao, B., Huanzhang, L., Chen, S., Liu, J., Dongya, W. Convolutional neural networks for time series classification. J Syst Eng Electron. 2017;28:162–9 02.

44. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q. A comprehensive survey on transfer learning. Proc IEEE. 2020;1–34:2020.

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.