Smoothing Techniques for Predicting High-Frequency Time Series

Sébastien Delsad, Leonard Amsler and Nathan Gromb
Supervised by Atin Aboutorabi and Yves Rychener
Department of Computer Science, EPF Lausanne, Switzerland

***Abstract*—We explore methods to produce smoother high-frequency now-casts of a low-frequency indicator by leveraging available high-frequency data. Estimators often produce noisy predictions due to the scarcity of low-frequency observations and the volatility of high-frequency input data. To address this problem, we present various smoothing techniques applied before, after and during model training. Pre-processing approaches (e.g. data augmentation and Fourier-based upsampling) increase training targets. Post-processing approaches (e.g. ARIMA and Kalman filters) reduce noise in the model outputs. Finally, incorporating roughness penalties into the model learning objective further stabilises predictions. Together, these strategies significantly improve smoothing without compromising accuracy, thereby improving the practical reliability of now-casting.**

I. Introduction

Across domains like economics, public health or politics, decision-makers often rely on low-frequency time series data. To address delays, now-casting uses high-frequency data to get a current estimation, enabling faster decisions. However, now-casting usually implies several challenges such as the noisiness of the high-frequency data and the lack of high-frequency observations. In this work, we explore various smoothing techniques as a potential solution to mitigate these issues. We study how pre-processing, post-processing and neural network in-model techniques can help smooth high-frequency predictions, while maintaining accuracy. To assess our methods, we choose to now-cast the GDP indicator with high-frequency Google trends, as it is a great example of a time series that is available only quarterly and for which we have very limited observations.
Figure 1 shows the high-frequency predictions of our now-casting model for the year-over-year GDP growth of Canada. One can clearly see that the predictions are too noisy to be realistic, thus highlighting the need for smoothing techniques.

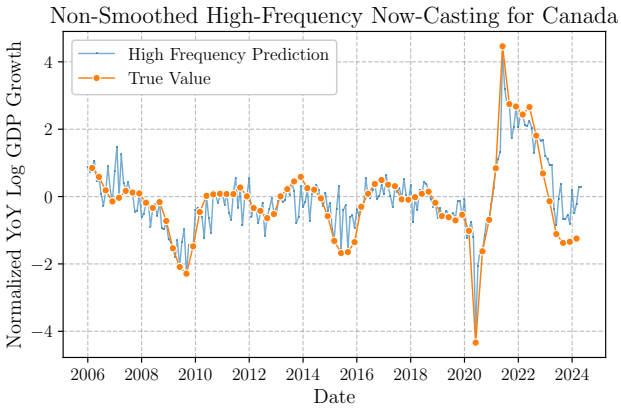

Figure 1: True values and high-frequency now-casting related to the GDP of Canada.

II. Setting

Suppose, we want to estimate a low-frequency time series with high-frequency data.

Let T_h be the set of the times at which high-frequency features are available. Similarly, for low-frequency times, we define $T_l = \{t \in T_h : r_{hl} \text{ divides } t\}$, with $r_{hl} \in \mathbb{N}_{>0}$ being the ratio between the two frequencies.

Let $\boldsymbol{x}[t] \in \mathbb{R}^D$ be some known data of dimension D at time $t \in T_h$. Define $X \in \mathbb{R}^{|T_h| \times D}$ the matrix whose rows are given by $\boldsymbol{x}[t]^\top \ \forall t \in T_h$.

Let $y[t] \in \mathbb{R}$ refer to low-frequency data that we want to now-cast for time $t \in T_l$, and let $\boldsymbol{y} \in \mathbb{R}^{|T_l|}$, be the vector containing all the low-frequency data points.

Our goal is to find an estimator f such that

$$\mathbb{E}_{T_l}\left[\Big(y[t] - f(X_{\leq t}, \boldsymbol{y}_{\leq t - t_{prev}})\Big)^2\right]$$

is minimized and such that f is "smooth" with respect to t. Note that $t_{prev} \in T_l$ represents the delay in availability of low-frequency data points. For example, if T_h is in units of months and low-frequency data is available only from a year prior to t, then $t_{prev} = 12$.

Moreover, the expectation reflects the average behaviour of the estimator f over the distribution of data points for every date in T_l. Lastly, the subscript "$\leq t$" refers to data points that are not later than t.

Our definition of smoothness for discrete time series must capture different aspects of oscillatory behaviour. Therefore, we first define a selection of smoothness error (or roughness), metrics e_i for $i \in [1 \,..\, 8]$, that can be found in Appendix C, and then combine them. Let e_s be the function that measures smoothness error, or roughness. It is defined to be the geometric mean of the e_i metrics.

Following this definition, we can measure the roughness of f by setting $\hat{y}[t] = f(X_{\leq t}, \boldsymbol{y}_{\leq t - t_{prev}})$ and computing $e_s(\hat{y})$. The smaller this metric is, the smoother our estimator f is with respect to time. For later use, we denote $\hat{\boldsymbol{y}}_{\boldsymbol{l}}$ the vector containing the low-frequency predictions $\hat{y}[t]$ for all $t \in T_l$ and, similarly, $\hat{\boldsymbol{y}}_{\boldsymbol{h}}$ for $t \in T_h$.

III. Reference Estimator

We use a feed-forward neural network to characterise our estimator f. Its architecture and implementation details can be found in Appendix B.
We denote f_α the estimator having weights randomly initialised with the seed $\alpha \in E \subset \mathbb{N}$, where E is a set of seed values.
The performance of the neural network can greatly depend on the initialisation weights determined by α. As an example, Figure 2 shows a histogram of the R^2 score on the test set and the roughness score e_s for high-frequency predictions for different random initialisations of our estimator.

As we can see, the performance of the model greatly depends on α. Therefore, in this work, we define the median score as the median of $\{\text{score}(f_\alpha),\ \forall \alpha \in E\}$, where score$(\cdot)$ can be any metric.

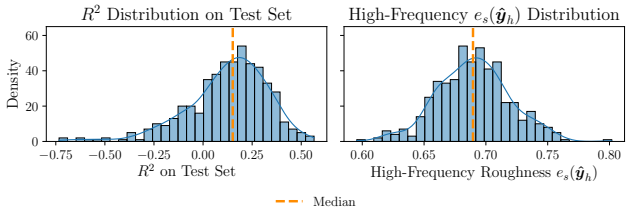

Figure 2: Histogram of the scores of the reference neural network for 500 different initial random weights (i.e. $|E| = 500$). The median is shown in orange.

IV. Smoothing techniques

We can divide the smoothing technique in several categories. First, there are the techniques applied before training the model, called "pre-processing techniques". Then, we focus on post-processing techniques that can be applied on any time series, independently of the type of estimator. Finally, we explore in-model techniques that are specific to neural networks as they change their learning algorithm.

A. Pre-processing techniques

As a pre-processing technique, data augmentation offers strategies to approximate data points missing due to the mismatch of low-frequency target variables and high-frequency input signals. By generating consistent high-frequency series for both inputs and targets, these pre-processing steps enable the model to learn finer-grained temporal patterns and potentially smooth out noisy fluctuations.

1) Linearly interpolating $\boldsymbol{y}$

As high-frequency inputs X are available, we can use them for training the estimator. Concerning the labels, we apply linear interpolation on the outputs $\boldsymbol{y}$. The vector containing the interpolated outputs is called $\boldsymbol{y}_{\text{lin}}$. Thus, the model trains on more inputs and a smoothly varying target. So, although $\boldsymbol{y}_{\text{lin}}$ contains synthetic data, it can be useful for training the model.

2) Frequency-based upsampling with Fourier transforms

A more refined approach uses frequency-based methods to approximate the high-frequency targets. Instead of relying on simple linear interpolation, we first resample the labels $\boldsymbol{y}$, using the Fourier transform. Compared to linear interpolation alone, this frequency-based method provides a more nuanced and stable approximation, potentially improving the model's ability to learn realistic temporal patterns.

B. Post-processing techniques

1) Ensembling

As mentioned in Section III, the model highly depends on the initialisation weights α. Therefore, we can think of training $|E|$ models with weights initialised using the seeds $\alpha \in E$ and ensemble their results.

Ensembling can be beneficial for smoothing as averaging the outputs of several models tends to cancel out individual errors and random fluctuations, resulting in reduced variance in predictions [1]. This is particularly useful when dealing with noisy data, as it enhances the model's ability to capture underlying trends.

In our context, we are applying ensembling to neural networks in two steps. We first train $|E|$ neural networks independently, each with different initial weights $\alpha \in E$, where E is a set of seeds. Then, for each input, we collect the predictions from all models and aggregate them by computing their median to obtain the final output.

Furthermore, this method allows to quantify the uncertainty of our predictions by computing the standard deviation of the predictions.

2) ARIMA

ARIMA (Autoregressive Integrated Moving Average) is a classical statistical method for time series forecasting [2]. It models a time series using its own past values (autoregressive terms), the differenced values to make the series stationary (integrations), and past forecast errors (moving average). This approach tends to capture past patterns in the data. It can be beneficial for smoothing because ARIMA can filter out noise and emphasise underlying trends in high-frequency time series data.

In our context, we choose two different approaches to get the parameters of the ARIMA model.
On the one side, we optimise the parameters of ARIMA using the "Augmented Dickey–Fuller" optimization method [3]. It allows to select optimal parameters from the predicted values. This method is denoted as *auto_arima*. On the other side, we simply test a wide range of parameter combinations.
Once the parameters set, we apply ARIMA on the predictions of the estimator to get smooth results.

3) Kalman filter

The Kalman filter is a state estimation algorithm that combines a theoretical model and observations to obtain a state estimate by taking their respective errors into account. Kalman filtering supposes white Gaussian noise and a linear model. Therefore, it can be a possible solution to smooth our predictions.

To apply Kalman filtering, we use the output of our estimator f as observations in the filter, enabling the use of an arbitrary estimator that doesn't need to be linear. We assume that if the predictions had a constant second derivative with respect to time, i.e. a constant acceleration, they would be "smooth". This motivates the use of a constant acceleration model for the Kalman filter. More formally, we define the state of the Kalman filter to be

$$\boldsymbol{v}_k = \begin{bmatrix} \hat{y}_{\text{kf},k} & \dot{\hat{y}}_{\text{kf}} & \ddot{\hat{y}}_{\text{kf},k} \end{bmatrix}^\top,$$

where $\dot{\hat{y}}_{\text{kf},k}$ and $\ddot{\hat{y}}_{\text{kf},k}$ are, respectively, the first and second derivative of $\hat{y}_{\text{kf},k}$ with respect to time. The state-transition F_k represents a constant acceleration model:

$$\boldsymbol{F}_k = \begin{bmatrix} 1 & \Delta t & \frac{1}{2}\Delta t^2 \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix},$$

where Δt is the time delay with the previous update.

Adjusting the covariance estimate Σ_{theory} of the theoretical model allows one to control its influence on the predictions. A balance needs to be found to obtain both smoothness and accuracy. The variance in observation noise is estimated using $y[t]$ and $\hat{y}[t]$, $\forall t \in T_l$.

As a parallel approach, we make use of $y[t]$ to measure if using it to correct the state estimates has an effect on the high-frequency filter predictions. At each time t, we update the filter with $y[t - k \cdot r_{hl}]$ for some fixed k if $t - k \cdot r_{hl} \in T_l$, and with the prediction $\hat{y}[t]$ otherwise. When updating the filter with true values, we set the observation noise variance to a low value named σ_{true}. This value needs to be carefully tuned to avoid influencing the predictions

too sharply. At the same time, it should give more importance to the true value compared to the estimator's predictions.

C. In-model techniques

In this section, we suppose the use of a neural network for the estimator f. To smooth the output of the neural network, we can include new regularisation terms in the objective function to penalise non-smoothness. For now, the objective function is defined as follows:

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{y}, \widehat{\boldsymbol{y}}) = \frac{1}{|T_l|} \sum_{t \in T_l} (y[t] - \widehat{y}[t])^2.$$

Note that, in our case, we are using Adam optimiser, so there's a regularisation on the weights of the network that is not represented in the objective function.
To penalise non-smoothness, one could think of regularising the values predicted for the times $(t-\beta, t-\beta+1, \ldots, t)$, where β depends on the application and can be tuned.

The smoothness loss is composed of the following metrics.

a) First-Order Smoothness Loss

$$\mathcal{L}_{\text{r},\beta,1}(\hat{\boldsymbol{y}}) = \frac{1}{M_1} \sum_{t \in T_l} \sum_{j=t-\beta+1}^{t} (\hat{y}[j] - \hat{y}[j-1])^2,$$

where M_1 is set to $|T_l|(\beta-1)$ to normalise the expression. $\mathcal{L}_{\text{r},\beta,1}$ measures the mean squared first order discrete derivative of $\hat{\boldsymbol{y}}$ with respect to time on a window of size β. Adding this to the loss of the neural network penalises rapid changes in the high-frequency predictions.

b) Second-Order Smoothness Loss

$$\mathcal{L}_{\text{r},\beta,2}(\hat{\boldsymbol{y}}) = \frac{1}{M_2} \sum_{t \in T_l} \sum_{j=t-\beta+2}^{t} (\hat{y}[t] - 2\hat{y}[t-1] + \hat{y}[t-2])^2,$$

where M_2 normalises the expression by being set to $|T_l|(\beta-2)$. It measures the mean squared second order discrete derivative of $\hat{\boldsymbol{y}}$ with respect to time on a window of size β. This metric penalises rapid oscillations in the predicted time series.
Notice that, if $\beta \leq r_{hl}$, then the smoothing will probably not work. Indeed, there would exist consecutive pairs of points for which the first order derivative is not taken into account. Therefore, ignoring some discontinuities in the predictions.
The roughness penalisation is defined to be the weighted geometric mean of the first, second order smoothness loss and e_i for $i = 1, 2, 3$ as defined in Section II.

$$\mathcal{L}_{\text{rough},\beta}(\hat{\boldsymbol{y}}) = \exp\Bigg(\log(\lambda_1 \mathcal{L}_{\text{r},\beta,1}(\hat{\boldsymbol{y}}) + 1) + \log(\lambda_2 \mathcal{L}_{\text{r},\beta,2}(\hat{\boldsymbol{y}}) + 1) + \sum_{i=1}^{3} \log\left(\lambda_{i+2} \sum_{t \in T_l} e_i(\hat{y}[t-\beta, \ldots, t]) + 1\right)\Bigg) - 1,$$

where $e_i(\boldsymbol{y}_h[t-\beta, \ldots, t])$ is the roughness error measured on the window defined by β. Finally, we write the objective function of our estimator as the sum of the MSE loss with the roughness loss:

$$\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \mathcal{L}_{\text{MSE}}(\boldsymbol{y}, \hat{\boldsymbol{y}}) + \mathcal{L}_{\text{rough},\beta}(\hat{\boldsymbol{y}}). \tag{1}$$

It is tunable by choosing λ_i for $i = 1, \ldots, 5$. In particular, one might want to ignore the regularisation coming from e_i and set $\lambda_{i+2} = 0$ for $i = 1, 2, 3$.

V. Results and Analysis

For all the results shown below, the plots present two types of measurements. The first is the median result, representing the median performance of individual models, as explained in Section III. The second is the "ensembled" result, produced by first aggregating all smoothed predictions and then computing the metrics. This allows us to observe both the typical performance of single models (via the median) and the overall performance when all predictions are pooled together (via the aggregated result).

A. Pre-processing techniques

Incorporating high-frequency training data effectively increases the smoothness of the time series, as illustrated in Figure 3. As expected, both methods improve smoothness without significantly sacrificing accuracy. We could justify the latter statement as such: the increased input richness helps to reduce over-fitting, while the smoothed labels reduce volatility and lead to more stable predictions.

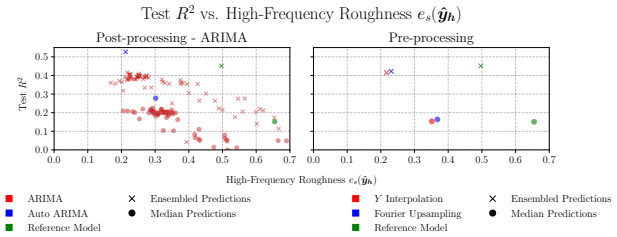

Figure 3: Performance of the preprocessing techniques.

B. Post-processing techniques

For the ARIMA model, we compare both the results obtained from the *auto_arima* method and those obtained by assessing 200 parameter combinations. The results are illustrated in Figure 3.

One can observe that ARIMA techniques are effective at improving both model accuracy (R^2 score) and output roughness e_s. However, the variability in the results highlights the importance of parameter tuning. A notable finding is that the automatically optimised ARIMA model outperforms the manually parametrised one. This difference arises because our dataset consists of multiple country-wise time series, each with distinct characteristics. In the *auto_arima* approach, the model parameters are individually optimized for each time series' data, whereas the fixed-parameter ARIMA model applies a single combination across all series. Consequently, the *auto_arima* method can adapt more precisely to each time series, improving performance.
Therefore, when working with diverse time series, it is advisable to fit ARIMA parameters on a series-by-series basis rather than applying a single configuration.

Applying a Kalman Filter with a constant acceleration model on $\hat{\boldsymbol{y}}_h$ and comparing the R^2 and roughness $e_s(\hat{\boldsymbol{y}}_h)$ between 1250 choices of Σ_{theory} and Σ_{true}, shown in Figure 4, highlights the importance of careful parameter choices. It is interesting to see that using lagged values of $\boldsymbol{y}$ to correct the state estimates does not help when using an ensembled model, although there is one choice of parameters for which the median performance is better with the correction than without.

Focusing on the model without correction, which gives better

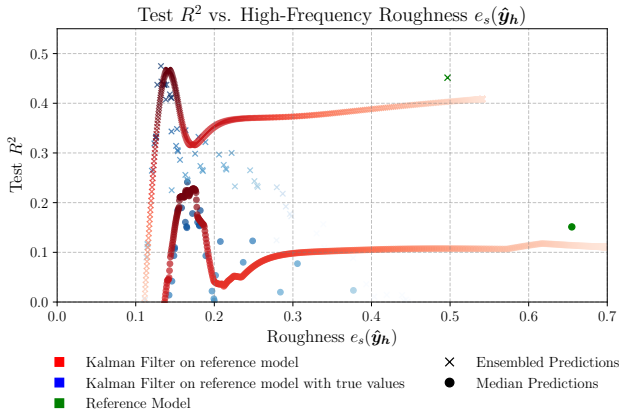

Figure 4: Performance of the Kalman filter on the reference estimator, for GDP now-casting. The opacity of the colour is linear to test R^2 − roughness.

performance on average and requires less hyperparameter tuning, the R^2 first slowly decreases as Σ_{theory} decreases, then starts increasing to reach an "optimal point" where $R^2 - e_s(\hat{\boldsymbol{y}}_{\boldsymbol{h}})$ is maximal, then drops to negative values. We explain this by observing that:

- High Σ_{theory} causes the filter to smooth the predictions, therefore shifting them away from true values. But the smoothing is not enough to correct high-frequency oscillations.
- Low Σ_{theory} gives too much importance to the constant acceleration model, harming the model's R^2.
- Optimal parameters improve performance by perfectly smoothing high-frequency noise.

Kalman filters applied to ensembled models obtain significantly better R^2 compared to those applied to median models, and the filter with optimal parameters gives slightly smoother results.

C. In-model techniques

We measure the R^2 and smoothness error e_s for $\sim 2,500$ different combinations of λ_i (including $\lambda_i = 0$) in the objective function given in Equation (1). Figure 5 shows a scatter plot comparing the reference model with the model trained using the modified objective function. As explained in Section IV-C, we require $\beta >$

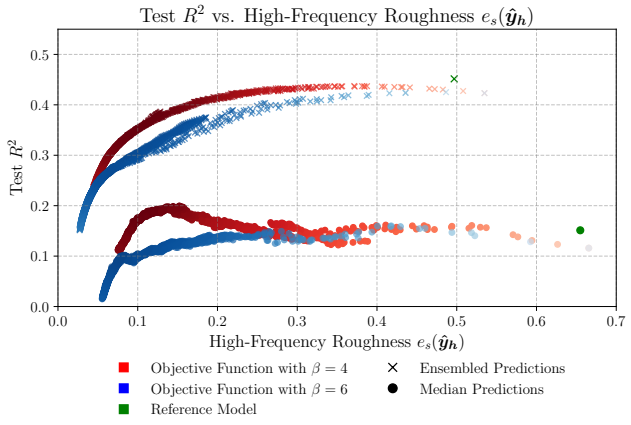

Figure 5: Performance of the estimator, for GDP now-casting, trained with roughness regularisation. The ensembling for the regularised models is done with $|E| = 15$. The opacity of the colour is linear to test R^2 − roughness.

r_{hl} and as $r_{hl} = 3$ in our case (3 Google trends data points in each quarter), we consider $\beta = 4$. We also consider the case $\beta = 6$ to understand how increasing β can impact the performance of the model. From Figure 5, we see a clear improvement in terms of smoothness loss with the custom objective function. Indeed, the median reference model has a roughness bigger than 0.65 whereas some combinations of λ_i give a roughness smaller than 0.3 while maintaining comparable R^2. Moreover, we see that with $\beta = 6$, we can reach smoother results than with $\beta = 4$, but at a cost of a worst test R^2. Furthermore, ensembling the predictions greatly improve the performance of the new model. Lastly, when the λ_i are getting smaller, the performance gets closer to the reference model. We assume that the R^2 doesn't get as good, because the ensembled and median performance were measured using 15 models instead of 500 for the reference model.

VI. Conclusion

This study addresses several convincing smoothing techniques to tackle the volatility of high-frequency predictions, taking for support the now-casting of the GDP using Google trends as high-frequency inputs. We evaluated pre-processing, post-processing, and in-model smoothing strategies to achieve more stable and realistic predictions.

Key findings include:

- Ensembling proved to improve smoothness and has shown to be very effective at increasing the R^2 score.
- Data augmentation produced smoother predictions, while maintaining accuracy.
- ARIMA is the method that improved most the accuracy of the model, leading to a R^2 of 0.52, as well as smoothing the predictions by a factor of 3.
- The usage of Kalman filters with a constant acceleration improved both smoothness and accuracy, when parameters are tuned.
- When the estimator is a neural network, penalising roughness in the objective function enables to considerably increase smoothness, at the cost of R^2.

However, our analysis carries limitations. For example, we could assess the performance of our methods using "rolling window" testing. Indeed, it would better simulate real-world conditions where models are updated sequentially with incoming data. Furthermore, we could try combining our pre-processing, post-processing and in-model smoothing techniques to verify if we reach a better performance.

The smoothing techniques we explore can be applied in different settings. First, if the estimator is a neural network, then directly training with a regularised objective function could help reach the smoothest results while maintaining sufficient accuracy. Also, if the time series appears to be stationary, by differentiating or not, then ARIMA would be a suitable choice. Finally, the Kalman filter can produce both better smoothness and accuracy when ensembled, making it an effective technique for any estimator.

To conclude, selecting the optimal smoothing method depends significantly on the estimator, the stationarity of the time series, and the acceptable trade-off between accuracy and smoothness. However, we recommend combining smoothing techniques with ensembling methods to achieve more robust and reliable results.

VII. Ethical risks

Google trends data reflects online search behaviour, but internet access is not evenly distributed across regions, as mentioned in [4]. In addition, [5] highlights how important it is for decision makers to know that GDP is self-reported, as it leaves room for errors-in-variables. Lower-income populations and countries where online censorship and data manipulation are high may be excluded or represented inaccurately in the data. As a result, predictions could overlook the influence of underserved populations, creating a systematic bias. This potential bias is therefore significant both in severity and likelihood as stakeholders who rely on predictions to guide decisions could be misinformed. Moreover, sentiment trends derived from Google trends data can display sharp fluctuations due to real-world events and viral content. Imposing smoothness to these trends could fail to capture these abrupt shifts. This introduces an additional layer of bias that creates a misleading narrative of gradual sentiment change.

Fully addressing the bias would be challenging. First, it would likely require the inclusion of additional data on GDP reliability and internet access. Incorporating these metrics into predictions would be essential. Then, one would need to make sure that the imposed smoothness property is carefully balanced to match the smoothness of the underlying data.

To limit the bias coming from the use of GDP and Google trends data, we carefully picked a set of countries where internet access is accessible by a vast majority of the population and for which GDP data is representative of the macro-economics of the country. Nevertheless, as our main focus is on smoothing techniques for time series, the bias doesn't directly impact the choice of our studied methods.

Appendix

A. Dataset Exploration and Pre-processing

1) Dataset description

The dataset used to explore smoothing techniques is composed of the GDP of seven countries, given on a quarterly basis. We predict the GDP using Google trends statistics on about 94 chosen subjects in each country. The Google trends are available on a monthly basis. Therefore, in our case, r_{hl}, defined in Section II, is 3 (three Google trends per quarter).

2) Feature Engineering and Processing

a) Pre-processing the GDP

We consider the following pre-processings of the GDP:

- Raw values: In practice, predicting the raw values gives a much better R^2. Indeed, predicting raw GDP values often appears easier for a model because absolute GDP levels tend to follow relatively smooth, long-term trends. As a result, a model can achieve a high R^2 by simply learning these stable patterns or even approximating them with straightforward approaches like linear trends or autoregressive components. However, it misrepresents our ability to now-cast GDP, as forecasting absolute GDP figures does not effectively capture relative changes and growth dynamics.
- Year-on-year relative differences: Using year-on-year relative differences places the emphasis on growth rates rather than absolute levels, thereby capturing the dynamics and volatility. Unlike raw values, relative differences remove the stable baseline and force the model to focus on more subtle changes. This makes the prediction task more challenging, as the model must discern patterns in the drivers of GDP growth. Although it may lead to lower R^2 values, this approach provides a more accurate representation of the evolving state of the economy and improves our ability to now-cast important changes in GDP.
- Year-on-year differences of logarithm: The logarithmic transformation aims to enable the model to predict extreme values more effectively, as suggested in [6]. Moreover, when the GDP relative changes are small, by the first order approximation of Taylor, this is directly equivalent to the year-on-year relative differences.

The two first methods both have weaknesses compared to the third one. Therefore, we choose to use the year-on-year differences of logarithm of GDP. We then normalize the data. Finally, note that, in our case, T_l and T_h are in units of months. To mitigate the impact of seasonality on GDP and address the delay in publishing the data, we set $t_{\text{prev}} = 12$.

b) Pre-processing the google trends

We consider the following pre-processing steps for the Google trends:

1) Taking the year-on-year difference of the logarithm of the values. Then, incorporating the year-on-2-year difference of the logs, as suggested in [6] and justified in Appendix A2.
2) Normalisation
3) Dimensionality reduction with principal component analysis (PCA)

Note that adding Google Trends data from different months increases over-fitting in practice and does not account for seasonality. Therefore, we recommend using only year-on-k-year differences.

Moreover, we justify doing dimensionality reduction, as reducing the dimension of the input can help reducing over-fitting. In practice, by applying PCA, the median R^2 score changes from ~ 0.062 to ~ 0.129. The R^2 score is more than doubled, but the smoothness doesn't statistically significantly change. Finally, the ensembled R^2 score (see Section IV-B1) is increased by $\sim 30.0\%$.

c) Training data augmentation

As the training set is small and the model easily over-fits, one can try to augment the training set. We assume that adding white Gaussian noise to the Google trends for different standard deviations makes the model more robust by simulating variability in the data. In other words, we think it will help the model better generalise to unseen data and therefore reduce over-fitting. In practice, adding noise with $\sigma_i \in \{0.001, 0.005, 0.01\}$, the median R^2 score increases by $\sim 8.7\%$ and the median smoothness doesn't statistically significantly change. When we ensemble models as done in Section IV-B1, the smoothness and R^2 score don't significantly change though.

d) Test set

In order to verify the accuracy of our model, we split the data set in a training and a test set. However, the way we split the data is crucial for interpretability of results. One can think of shuffling and picking a randomly a specific portion of the data to be the test set. However, this would imply that countries are not represented equivalently in the training set and that we allow the model to learn on points near test points. Therefore, we create our test set by choosing a date for which all the previous data (for all countries) represent the training set and all the future points represent the test set.

B. Reference Neural Network Implementation Details

The reference feed-forward neural network has the following architecture:

- Layers: D $\to$ 300 $\to$ 100 $\to$ 1
- Activation functions: ReLu
- Layer normalisation for each hidden layer
- Number of epochs: 100
- Learning rate: 1e-4
- Weight decay: 1e-2
- Optimizer: Adam
- Loss function: Mean Squared Error (MSE)
- Weight initialisation seed: $\alpha \in \mathbb{N}$

Note that, we use layer normalisation on each hidden layer to reduce over-fitting. Moreover, we choose to have relatively small layers and only two of them to reduce the number of parameters to train. Indeed, as we have very few data points for training, the model over-fits very easily. Having less parameters help reduce this effect.

C. Smoothness error functions

Our definition of smoothness for a discrete time series $s[t]$, for $t = 0, 1, 2, \ldots, N-1$, must capture different aspects of oscillatory behaviour in the time series. Therefore, we define the following smoothness error, or roughness, metrics.

1) **Standard deviation of the first derivative:**

$$e_1(s) = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N-1}(\Delta s[t] - \mu_{\Delta s})^2},$$

where $\Delta s[t] = s[t] - s[t-1]$ is the discrete first derivative and $\mu_{\Delta s}$ is its mean. This metric captures the variability in the rate of change of the series.

2) **Mean absolute first difference:**

$$e_2(s) = \frac{1}{N-1}\sum_{t=1}^{N-1}|\Delta s[t]|.$$

This is the average of the absolute values of the first differences.

3) **Standard deviation of the second derivative:**

$$e_3(s) = \sqrt{\frac{1}{N-2}\sum_{t=2}^{N-1}(\Delta^2 s[t] - \mu_{\Delta^2 s})^2},$$

where $\Delta^2 s[t] = \Delta s[t] - \Delta s[t-1]$ is the discrete second derivative and $\mu_{\Delta^2 s}$ is its mean.

4) **High-frequency energy:**

$$e_4(s) = \frac{1}{N}\sum_{t=0}^{N-1}|s[t] - s_{\text{filtered}}[t]|^2,$$

where $s_{\text{filtered}}[t]$ is the signal reconstructed by retaining Fourier components with frequencies of absolute value not bigger than a tuned cut-off. This measures the energy in high-frequency components.

5) **Wavelet-based smoothness measure:**

$$e_5(s) = \frac{1}{N}\sum_{i=1}^{L}\sum_{j=0}^{N_i} c_{i,j}^2,$$

where $c_{i,j}$ are the wavelet detail coefficients at level i and L is the maximum decomposition level. The approximation coefficients are excluded from the calculation to not penalise the underlying trend in the time series.

6) **L2 norm of first derivative:**

$$e_6(s) = \frac{1}{N-1}\sqrt{\sum_{t=1}^{N-1}(\Delta s[t])^2}.$$

7) **Integrated absolute curvature:**

$$e_7(s) = \frac{1}{N-2}\sum_{t=2}^{N-1}\left|\Delta^2 s[t]\right|.$$

This measures the average absolute curvature of the series.

8) **Spline roughness:**

$$e_8(s) = \frac{1}{N}\sum_{t=0}^{N-1}\left(\bar{s}''[t]\right)^2,$$

where $\bar{s}''[t]$ is the second derivative of a spline fitted on s. This penalises rapid oscillations in the fitted curve, or in other words, it measures rapid changes of the time series without taking into account white noise in s.

Notice that each metric is normalised with respect to the length of the series. We then combine them using the geometric mean to get our measure of roughness:

$$e_s(s) = \exp\left(\frac{1}{8}\sum_{i=1}^{8}\log\left(e_i(s)+1\right)\right) - 1.$$

Here, the geometric mean aims to minimise the impact of the e_i metrics having different scales. Indeed, normalising is not sufficient as the e_i measure different features.

References

[1] T. Dietterich, "Ensemble methods in machine learning," 2000. [Online]. Available: https://doi.org/10.1007/3-540-45014-9_1

[2] T. C. Mills, "Time series analysis and the british," in *A Very British Affair*. London: Palgrave Macmillan UK, 2013, pp. 1–2.

[3] S. E. SAID and D. A. DICKEY, "Testing for unit roots in autoregressive-moving average models of unknown order," *Biometrika*, vol. 71, no. 3, p. 599–607, 1984. [Online]. Available: http://dx.doi.org/10.1093/biomet/71.3.599

[4] J. Mellon, "Where and when can we use Google Trends to measure issue salience?" *PS: Political Science 38; Politics*, vol. 46, no. 2, p. 280–290, 2013.

[5] G. Cai, X. Li, B. Lin, and D. Luo, "GDP manipulation, political incentives, and earnings management," *Journal of Accounting and Public Policy*, vol. 41, no. 5, p. 106949, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0278425422000126

[6] N. Woloszko, "Tracking activity in real time with Google Trends," no. 1634, 2020. [Online]. Available: https://www.oecd-ilibrary.org/content/paper/6b9c7518-en