Financial Machine Learning: Macroeconomic Indicators, Forecasting, and Predictive Analytics

By Vikhram S, Advisor, Machine Learning Community – Tech Society, Saveetha Engineering College

Delivered to the Department of Computer Science and Engineering, Saveetha Engineering College, on 16 May 2025.

It is a pleasure to be here today, and I am grateful for the invitation to deliver this lecture. My lecture today is on financial machine learning, and it is built around an applied exercise in macroeconomic forecasting: a deep learning model trained on United States GDP growth data from the World Bank.[1]

The question at the centre of the lecture sits at the intersection of macroeconomics and machine learning: whether a neural network can learn the shape of a business cycle closely enough to forecast it. Rather than address the question in the abstract, I will work through a complete pipeline — one built on actual GDP growth data for the United States, carried from raw indicator through feature engineering, model training, and a four-year forecast — and use it to examine both what deep learning offers macroeconomic analysis and where, at present, it falls short.

1. Why GDP growth is a hard forecasting target

Gross domestic product growth is among the most closely watched figures in economics, and among the least forgiving to model. It is a single annual percentage that compresses consumption, investment, government spending, and trade into one number, and its history is dominated by long calm stretches punctuated by sharp, irregular shocks — a financial crisis, a pandemic, a policy shift. A model trained mostly on calm stretches has, by construction, seen very few examples of the shocks that matter most. This is the tension to which the lecture will repeatedly return: a machine learning model is only as good as the regularities present in the data it is shown, and GDP growth is a series whose most consequential moments are precisely those that break the pattern.

2. The data pipeline: from World Bank indicator to feature matrix

The analysis draws its data directly from the World Bank's open data catalogue, using the `wbdata` Python package, which wraps the World Bank API in a form suited to structured analysis. A single indicator is used — `NY.GDP.MKTP.KD.ZG`, annual GDP growth as a percentage — for the United States, spanning 1960 to 2023. The raw series is converted into a clean, date-indexed dataframe, coerced to numeric form, and any missing observations are removed before it is passed to the feature engineering stage.

The data pipeline, from indicator to tensor GRAPH 1

1 · Acquire

World Bank Open Data via **wbdata**. Indicator `NY.GDP.MKTP.KD.ZG`, country `USA`, 1960–2023, annual frequency.

2 · Engineer

Four derived features built from the series itself: two lags and a rolling mean and standard deviation over a 3-year window.

3 · Scale & window

MinMaxScaler to [0, 1]; a sliding window of **look_back = 5** years forms each training example.

Source: author's notebook, "Financial Machine Learning Model for Macroeconomic Trend Analysis," 2025.

3. Feature engineering on a single series

It is worth being precise about what the model is actually shown, since it would be natural to assume that an exercise of this kind draws on a basket of macro indicators — unemployment, inflation, interest rates — feeding into GDP. It does not. Every feature in this pipeline is derived from GDP growth alone. `create_features()` adds a one-year lag and a two-year lag of the value column, together with a three-year rolling mean and a three-year rolling standard deviation. The intent is to furnish the network with a short memory of recent level and recent volatility, without introducing additional data sources, additional missing-value problems, or additional dependencies. It is a deliberately minimal, univariate design — a reasonable point of departure, and an instructive one to interrogate, since everything the model gets right or wrong in Section 7 must be explained by this narrow feature set alone.

```
def create_features(df):
    df = df.copy()
    df['lag_1'] = df['value'].shift(1)              # 1-year lag
    df['lag_2'] = df['value'].shift(2)              # 2-year lag
    df['rolling_mean'] = df['value'].rolling(window=3).mean()
    df['rolling_std']  = df['value'].rolling(window=3).std()
    return df.dropna()
```

4. Windowing and scaling: turning a series into a supervised problem

An LSTM does not consume a flat table; it requires sequences. `preprocess_data()` scales all five columns — the value together with its four derived features — into the [0, 1] range with a `MinMaxScaler`, then slides a five-year window across the scaled data: each input example `X` comprises five consecutive years of all five features, and the corresponding target `y` is the GDP growth value in the year immediately following that window. The resulting array is split chronologically, with 80% reserved for training and the remaining 20% held out as a test set. The split is deliberately chronological, so that the model is always evaluated on years it has not seen, rather than on a random shuffle that would allow the future to leak into the past.

5	5	80 / 20	1960–2023
YEARS PER INPUT WINDOW	FEATURES PER TIMESTEP	CHRONOLOGICAL TRAIN / TEST SPLIT	SAMPLE PERIOD

5. Model architecture: a stacked LSTM

The network itself is a compact, two-layer LSTM built with Keras. The first recurrent layer returns full sequences so that a second recurrent layer can process them; dropout is applied after each LSTM layer to discourage overfitting on a training set that, at an 80% split of roughly sixty usable years, is genuinely small by the standards of deep learning. Two dense layers narrow the representation to a single scalar output — the predicted, scaled GDP growth value for the following year.

Network architecture, in sequence GRAPH 2

LSTM · 100
return_sequences=True
input: (5 steps, 5 features)
`Dropout 0.2`

LSTM · 50
collapses the sequence to a single vector
`Dropout 0.2`

Dense · 25 → Dense · 1
fully connected head
output: next-year GDP growth
`optimizer = adam`

Loss: mean squared error. Source: author's notebook, build_lstm_model().

6. Training dynamics: what the loss curve tells us

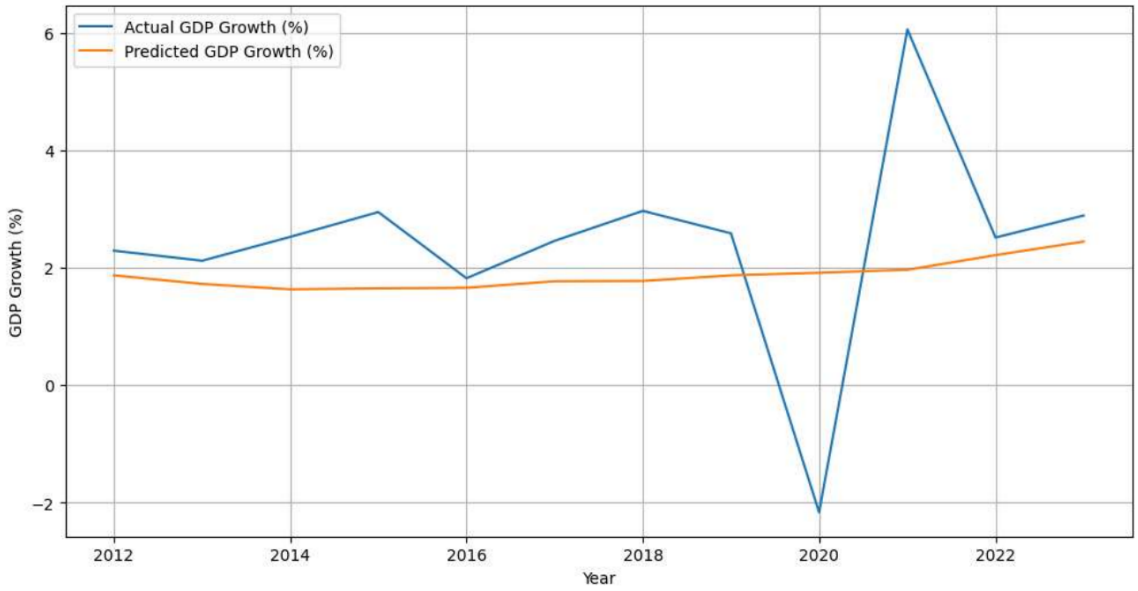
The model is trained for up to 100 epochs with a batch size of 16, though an `EarlyStopping` callback monitors the training loss with a patience of 10 epochs and restores the best weights observed — a modest but consequential discipline, in that it prevents the pipeline from shipping an overfit, late-epoch model merely because 100 was the figure specified in the epochs argument. In practice, training terminated at epoch 27. The loss fell sharply over the first three epochs, from 0.2769 to 0.0534, and thereafter oscillated within a narrow band of roughly 0.04 to 0.06 for the remaining epochs, without further meaningful improvement. This is the signature of a model that has extracted most of what a short, noisy macro series has to offer, and is bounded thereafter by the limits of the data rather than the architecture.

> One further observation is worth recording. Installing `wbdata` caused pip's dependency resolver to flag conflicts with several packages already present in the environment. Nothing failed on this occasion, but the episode is a small, live illustration of a broader point: reproducibility practice — pinned dependencies, isolated environments — is not a procedural nicety layered on top of the analysis. It is what determines whether a result obtained today can be obtained again next year.

7. Results: actual versus predicted GDP growth

Evaluated on the held-out test years, the model returns a **test RMSE of 1.8002** — a typical error on the order of 1.8 percentage points of GDP growth. Whether this constitutes a good result depends entirely on what lies beneath the summary statistic, and it is to that detail that I now turn.

Actual vs predicted GDP growth, test period GRAPH 3



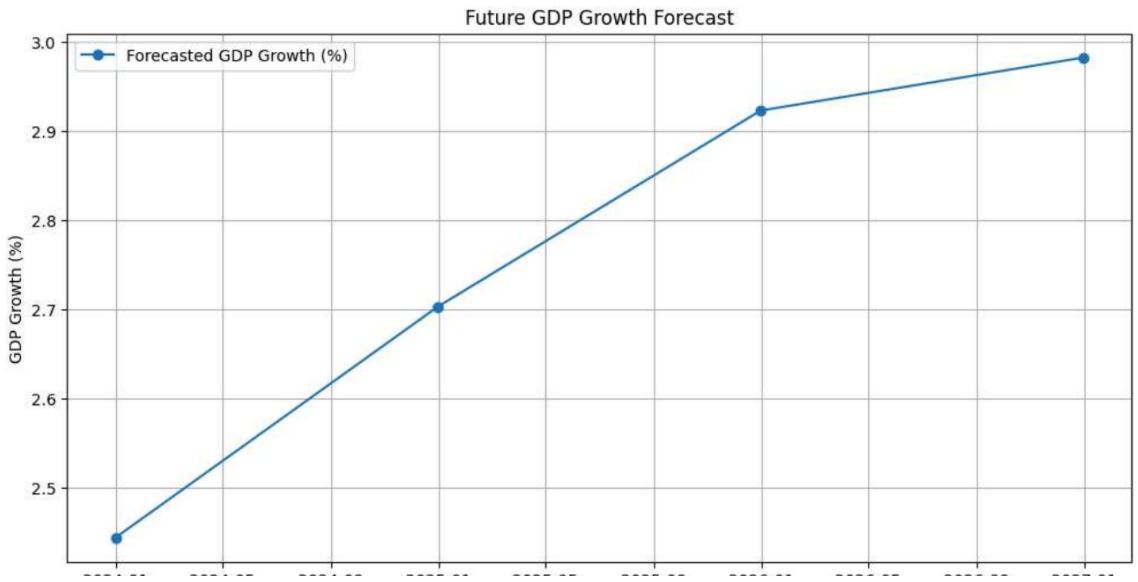
Source: author's notebook. USA GDP growth (annual %), World Bank indicator NY.GDP.MKTP.KD.ZG.

The predicted series is instructive precisely because of how little it moves. Across a decade that includes a mild slowdown, a normal expansion, the 2020 contraction of roughly −2.1%, and the sharp 2021 rebound above 6%, the model's forecast remains confined to a narrow band between approximately 1.6% and 2.5%. It predicts, in effect, the recent average, in every year, irrespective of what is about to occur. This is not a defect in the implementation; it is the rational response of a mean-squared-error objective trained on a small number of extreme events. Predicting close to the historical mean minimizes expected squared error when large shocks are rare and unforeseeable from the available features — the model has not failed to learn the pattern so much as correctly learned that, given only the lags and rolling statistics of GDP growth itself, the 2020 shock was not foreshadowed in the data it was shown. This is the central, transferable finding of the exercise: a forecasting model is a mirror of its feature set, and a feature set constructed entirely from a variable's own past cannot anticipate shocks that originate outside that variable's own history.

8. Forecasting ahead: the recursion trap

The final stage of the pipeline projects GDP growth four years beyond the end of the sample, by feeding the model's own prediction back in as the subsequent input — a standard technique for multi-step forecasting with a model trained to predict only one step ahead. At each iteration, the predicted value is inserted into the sequence; the lag and rolling-statistic features, which the model was never asked to forecast, are approximated by carrying forward their last known values rather than being recomputed from the still-uncertain predicted path.

Four-year recursive forecast, 2024–2027 GRAPH 4



Source: author's notebook, recursive forecast from the final test-window state.

The result is a smooth, almost linear ascent from approximately 2.44% to 2.98% — a forecast with the texture of a straight line rather than a business cycle. Two mechanisms compound to produce this shape. First, because each step's forecast becomes the following step's input, any tendency of the model toward mean reversion is reinforced at every iteration, and the approximated lag and rolling features contribute no new information to offset it. Second, this recursive procedure carries forward no representation of its own growing uncertainty: the chart depicts a single path rather than a widening range of plausible outcomes, notwithstanding that the true forecast error four years out is necessarily far larger than at one year. A more candid rendering of this forecast would require prediction intervals, an ensemble of models or bootstrapped runs, or, at minimum, an explicit caveat that recursive point forecasts of this kind should not be read as more than a trend extrapolation.

9. From notebook to reliable system

Everything described in Sections 2 through 8 exists, at present, inside a single script and a single run. That is an appropriate starting point, but it is not yet a system, and this is precisely the concern addressed in a companion lecture on software engineering for reproducible deep learning. Three habits from that discussion map onto this pipeline with little translation required: pin the environment, since the dependency conflict noted in Section 6 makes the argument on its own; version the data alongside the code, as a GDP series revised next quarter will silently alter the training set and every downstream figure; and track experiments — random seed, look-back window, architecture, resulting error — so that a figure such as 1.8002 remains a reproducible claim about a specific configuration, rather than a number that occurred once.

10. Implications for applied macro-forecasting

The exercise carries a lesson that extends well beyond this particular series or this particular architecture. Univariate deep learning models of this kind are best suited to short-horizon, trend-following settings, where the

recent past is genuinely informative about the near future and the object of interest evolves gradually. They are poorly suited to the moments that matter most in macroeconomic policy — turning points, crises, regime changes — precisely because those moments are, by construction, underrepresented in any historical sample and invisible to a feature set built solely from the series' own lagged values.

A more complete system would depart from this design in three respects. It would incorporate exogenous indicators — credit spreads, financial conditions, sentiment measures — that have some prospect of leading rather than merely lagging the target series. It would represent forecast uncertainty explicitly, through prediction intervals or ensembles, rather than reporting a single path. And it would be evaluated not only on average error but on its behaviour during the small number of episodes that account for most of the economic and financial consequences of a forecast being wrong.

Concluding thoughts

The model examined here is deliberately simple, and that simplicity is itself the point. It is easy to be impressed by an LSTM producing a smooth four-year GDP forecast, and equally easy to overlook that the smoothness is itself the finding: a model built on a single series' own history will, almost by definition, struggle to anticipate the shocks that make macroeconomic forecasting consequential in the first place. The value of an exercise of this kind lies not in the forecast it produces but in the discipline it cultivates — reading a loss curve for what it reveals about the data rather than the code, treating a headline error statistic as the beginning of an inquiry rather than its conclusion, and carrying that same scrutiny from a notebook into whatever system it eventually becomes.

1 The pipeline, results, and figures discussed in this lecture are drawn from the author's applied analysis, "Financial Machine Learning Model for Macroeconomic Trend Analysis" (2025), built on World Bank Open Data, indicator NY.GDP.MKTP.KD.ZG (GDP growth, annual %), United States, 1960–2023, accessed via the `wbdata` Python package. Model: a Keras/TensorFlow Sequential LSTM.

For questions or further discussion, please write to vikhrams@saveetha.ac.in.