Neural Prophet

A simple time series forecasting framework

Oskar Triebe, Mateus De Castro Ribeiro, Abishek Sriramulu, Hansika Hemawalage

Stanford University, Monash University

Nov 16th, 2021

Neural Prophet

is	an open-source forecasting library.
tl;dr	Prophet in PyTorch + AR + Covar + NN + multistep + ...
Task:	Forecasting.
Data:	1E+2 to 1E+6 of samples. Unidistant, real-valued.
Dynamics:	Future values must depend on past observations. e.g. Seasonal, trended, events, correlated variables.
Applications:	Human behavior, energy, traffic, sales, environment, server load, ...

Motivation

Model

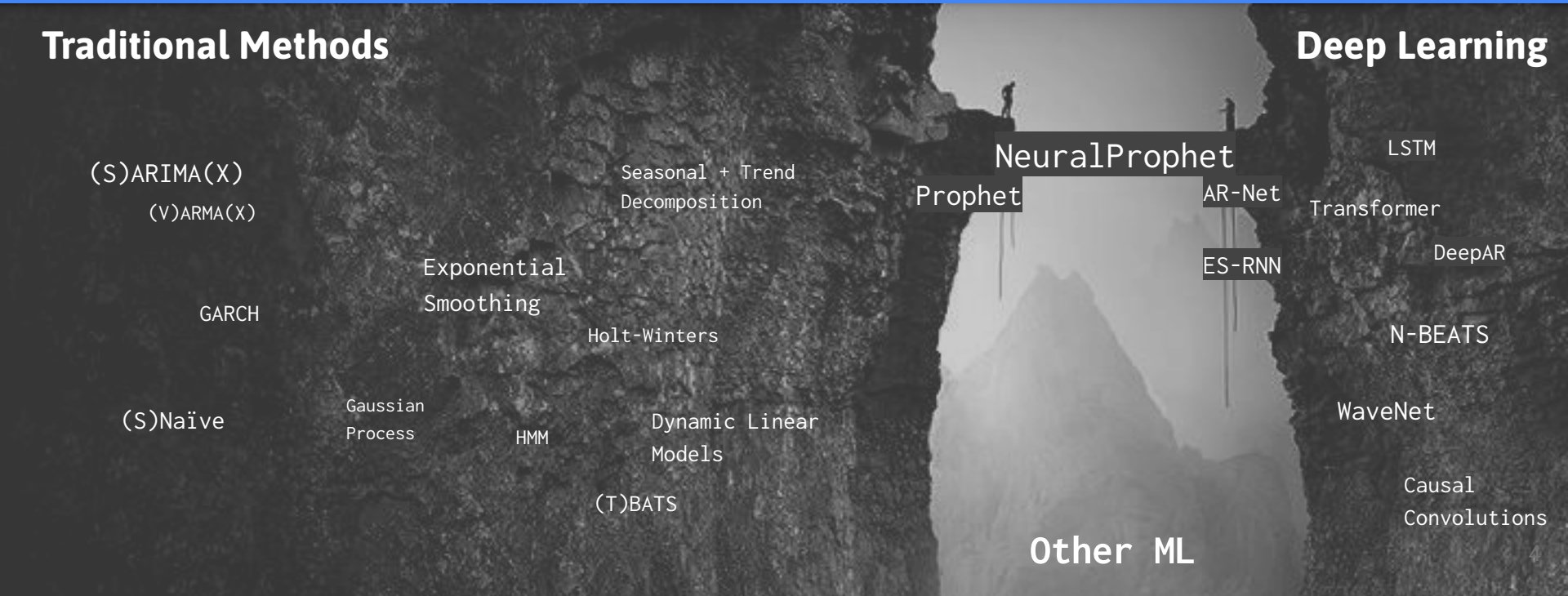
Example

Global Modelling

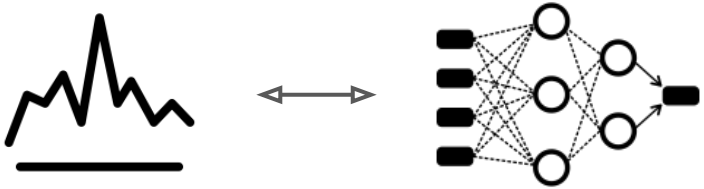
Attention

Conclusion

Time series forecasting is messy.
We need hybrid models to bridge the gap.



Before

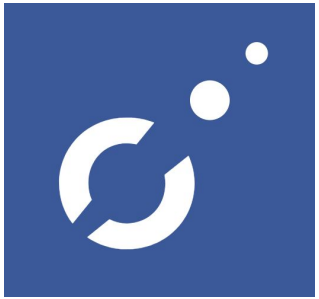


Need expertise in both
time series & machine learning

NeuralProphet

Abstracts time series
& machine learning knowledge

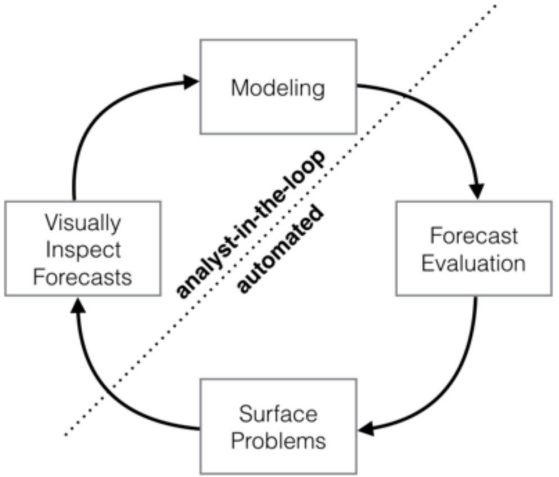
Facebook Prophet is still the most used forecasting package.



Quick from data to predictions.

Gentle learning curve.

Customizable.



Taylor, S. J., & Letham, B. (2017). Forecasting at scale, PeerJ. https://peerj.com/preprints/3190/

NeuralProphet is more than the Neural evolution of Prophet.

PROPHET

Prophet has three major shortcomings:

1. Missing local context for predictions
2. Acceptable forecast accuracy
3. Framework is difficult to extend (Stan)

Neural Prophet

NeuralProphet solves these:

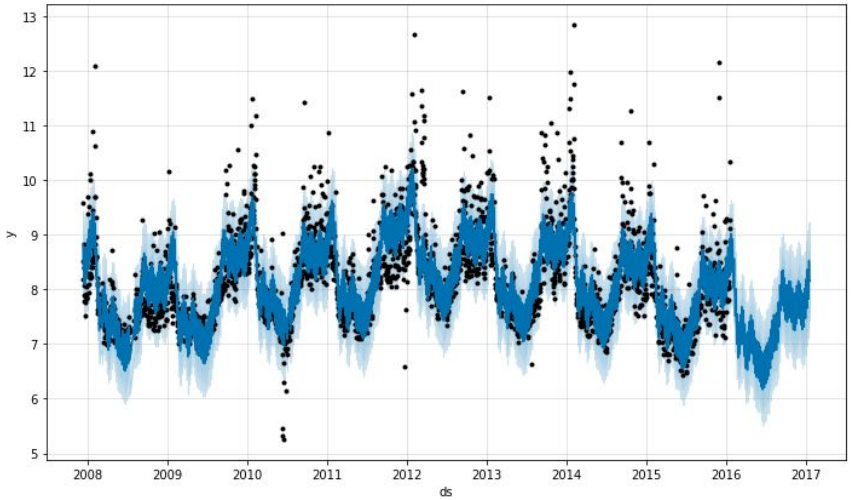
1. Support for auto-regression and covariates.
2. Hybrid model (linear <> Neural Network)
3. Python package based on PyTorch using standard deep learning methods.

Current Model Components

S	*Seasonality*
T	*Trend*
E/H	*Events / Holidays*
X	*Regressors*
AR	*Autoregression*
Cov	*Covariates*

_	*Sparsity / Regularization*
NN	*Nonlinear (deep) layers*
{ }	*Global Modelling*
?	*Uncertainty*

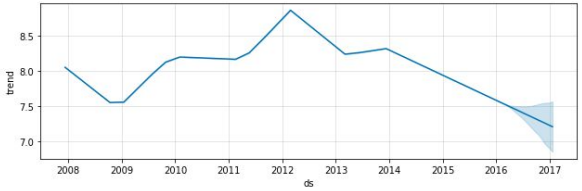
Prophet is interpretable and decomposable. So is NeuralProphet.



$$y(t) = g(t) + s(t) + h(t) + \epsilon_t.$$

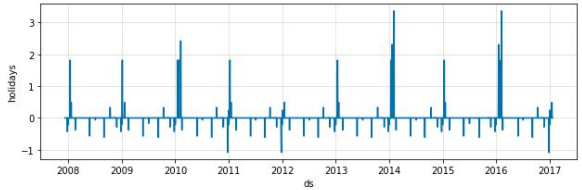
Trend

$g(t)$



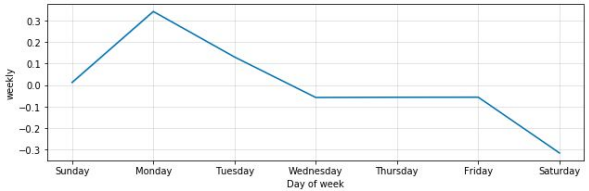
Events

$h(t)$



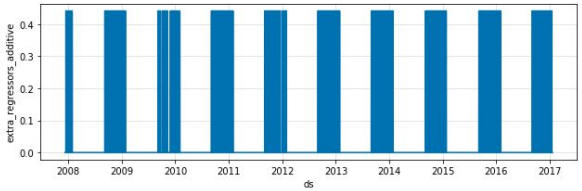
Seasonality

$s(t)$



Regressors

$X(t)$



Trend & Seasonality are identical to Prophet.

Piecewise linear trend

- N changepoints
- Segment-wise independent
- Automatic changepoint detection
- Optional logistic growth

$$g(t) = (k + \mathbf{a}(t)^\intercal \boldsymbol{\delta})t + (m + \mathbf{a}(t)^\intercal \boldsymbol{\gamma}),$$

$$a_j(t) = \begin{cases} 1, & \text{if } t \geq s_j \\ 0, & \text{otherwise} \end{cases}$$

$-s_j\delta_j$

Optional Regularization

$$R(\theta, \epsilon, \alpha) = \frac{1}{p}\sum_{i=1}^{p} log(\frac{1}{\epsilon \cdot e} + \alpha \cdot |\theta_i|) + log(\epsilon) + 1$$

Seasonality

- N Fourier terms
- Automatic yearly, weekly, daily
- Optional multiplicative mode

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right)$$

$$s(t) = X(t)\boldsymbol{\beta}.$$

$$X(t) = \left[\cos\left(\frac{2\pi(1)t}{365.25}\right), \ldots, \sin\left(\frac{2\pi(10)t}{365.25}\right) \right]$$

Events, Holidays and Regressors are identical to Prophet

Events / Holidays

- Automatic for given country
- Various user-defined formats
- Optional multiplicative mode

$$Z(t) = \sum_{i=1}^{m} c_i e_i(t)$$

(Future-Known) Regressors

- Single weight
- Real-valued regressor
- Optional multiplicative mode

$$R(t) = \sum_{i=1}^{l} d_i v_i(t)$$

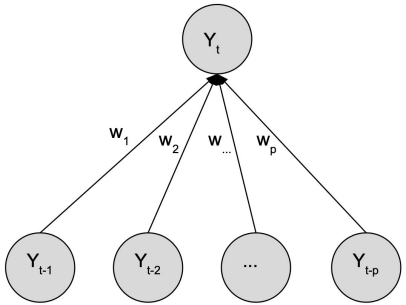
We model Covariates and Auto-Regression as AR-Nets.

Auto-Regression

- By default AR-Net(0)
- Depth customizable AR-Net(n)
- Optional auto-AR via regularization

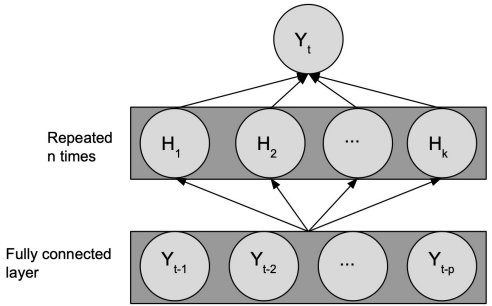
(Lagged) Covariates

- By default AR-Net(0) with y as target
- Depth customizable AR-Net(n)
- Optional lag-sparsification via regularization



AR-Net(0)

Interpretable



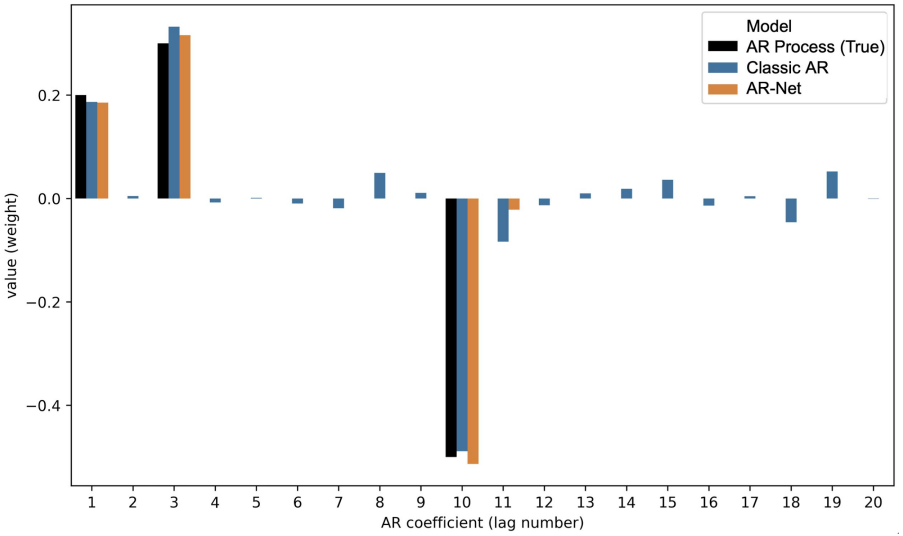
AR-Net(n)

Non-linear modeling

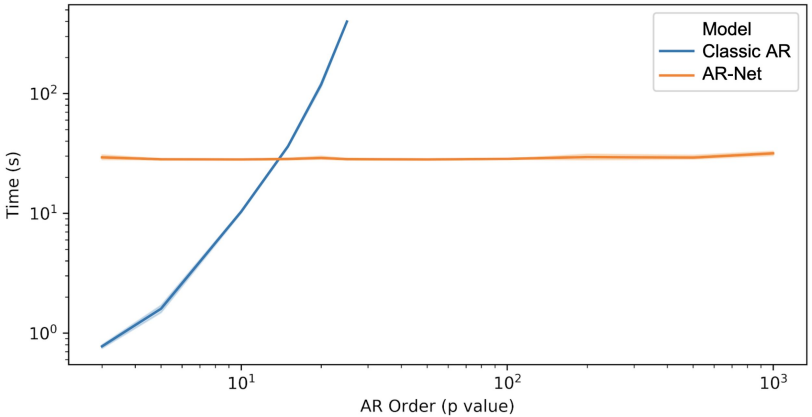
$$R_{AR}(\theta) = R(\theta, \epsilon = 3, \alpha = 1)$$
$$= \frac{1}{p}\sum_{i=1}^{p} log(\frac{1}{3 \cdot e} + |\theta_i|) + log(3) + 1$$

Optional Regularization

Automatic AR-lag selection, yet faster.



Automatic Sparsity

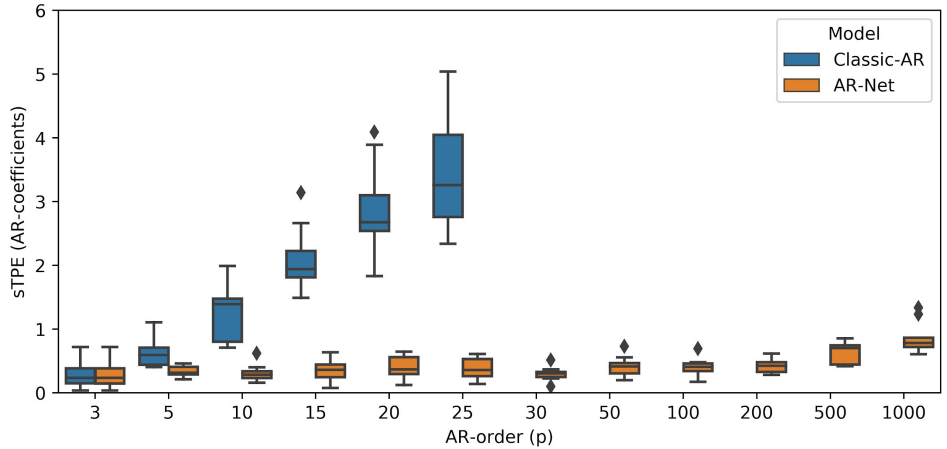


Quadratically faster

Sparse AR-Net surpasses Classic AR and scales to large orders.

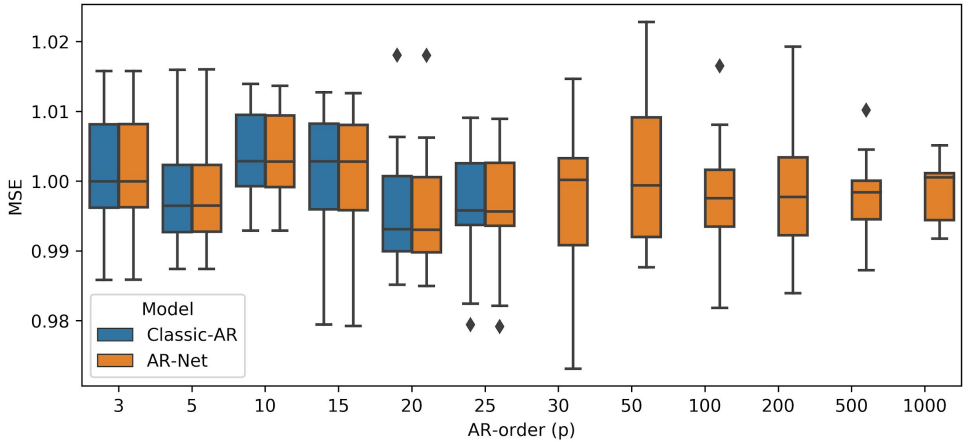
Closeness to true coefficients

$$sTPE = 100 \cdot \frac{\sum_{i=1}^{i=p} |\hat{w}_i - w_i|}{\sum_{i=1}^{i=p} |\hat{w}_i| + |w_i|}$$



MSE loss on forecast target

$$MSE = \frac{1}{n}\sum_{1}^{n}(y_t - \hat{y}_t)^2$$



A user-friendly Python package

Gentle learning curve.

Get results first. Learn. Improve.

Powerful, customizable, extendable.

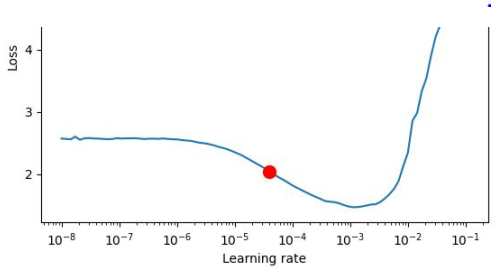
```
m = NeuralProphet()
metrics = m.fit(df, freq='D')
forecast = m.predict(df)
m.plot(forecast)
```

Hyperparameters have smart defaults.

Loss Function is **Huber loss**,
unless user-defined.

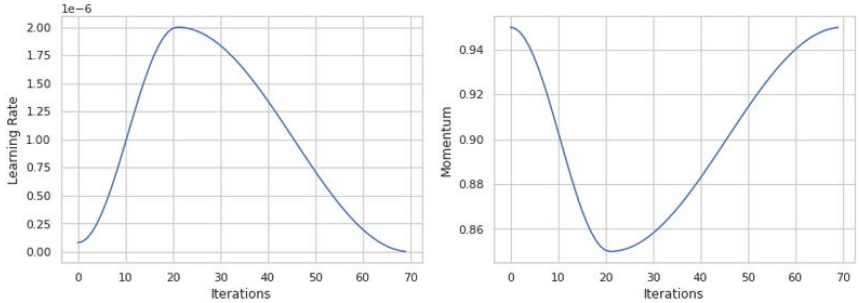
$$L_{huber}(y, \hat{y}) = \begin{cases} \frac{1}{2\beta}(y - \hat{y})^2, & \text{for } |y - \hat{y}| < \beta \\ |y - \hat{y}| - \frac{\beta}{2}, & \text{otherwise} \end{cases}$$

The learning rate is approximated
with a **learning-rate range test**.



Batch size and epochs are approximated
from the dataset size.

We use **one-cycle policy**
with AdamW as optimizer for simplicity.



Preprocessing happens mostly behind the scenes.

Missing Data is is automatically filled in:

1. bi-directional linear interpolation
2. centred rolling average

Data is automatically normalized:

Name	Normalization Procedure
'auto'	'minmax' if binary, else 'soft'
'off'	bypasses data normalization
'minmax'	scales the minimum value to 0.0 and the maximum value to 1.0
'standardize'	zero-centers and divides by the standard deviation
'soft'	scales the minimum value to 0.0 and the 95th quantile to 1.0
'soft1'	scales the minimum value to 0.1 and the 90th quantile to 0.9

We have utils ...

Visualize:

- Plot past and future predictions
- Decompose forecast components
- Interpret model parameters
- Plot most recent prediction
- Inspect a particular forecast horizon

Other:

- Simple Split, cross validation, double cross validation
- Control logger verbosity
- Make fit reproducible
- Global modelling
- Benchmarking
- ...

Quick Start Examples

The classic: Temperature forecast

Using Trend, Seasonality and Auto-Regression:

Example 1: 1 step ahead

Example 2: 36 steps ahead

Dataset:

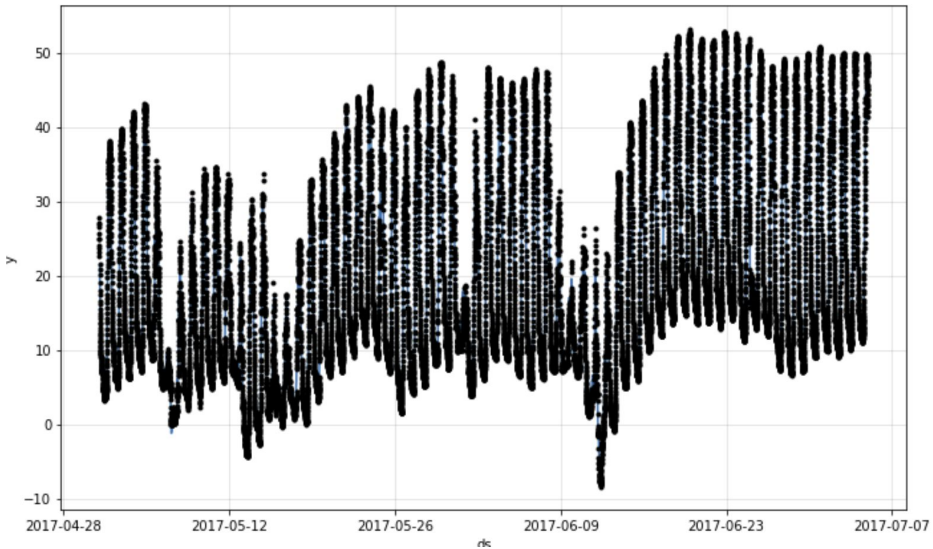
`Observed temperature in Yosemite Valley, measured every 5 min over two months.`

Yosemite Temperature - Predict next observation.

Example: Yos1

```
m = NeuralProphet(n_lags=12)
metrics = m.fit(df, freq='D')
forecast = m.predict(df)
```

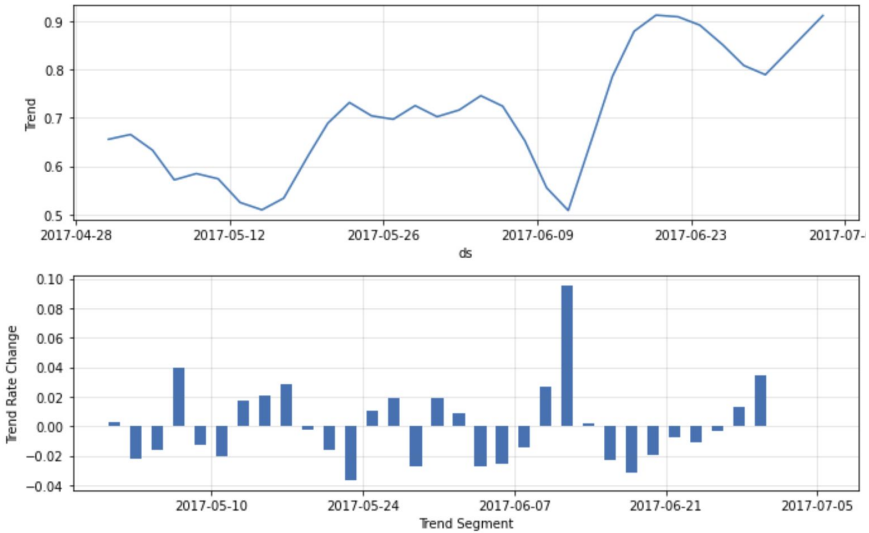
`m.plot(forecast)`

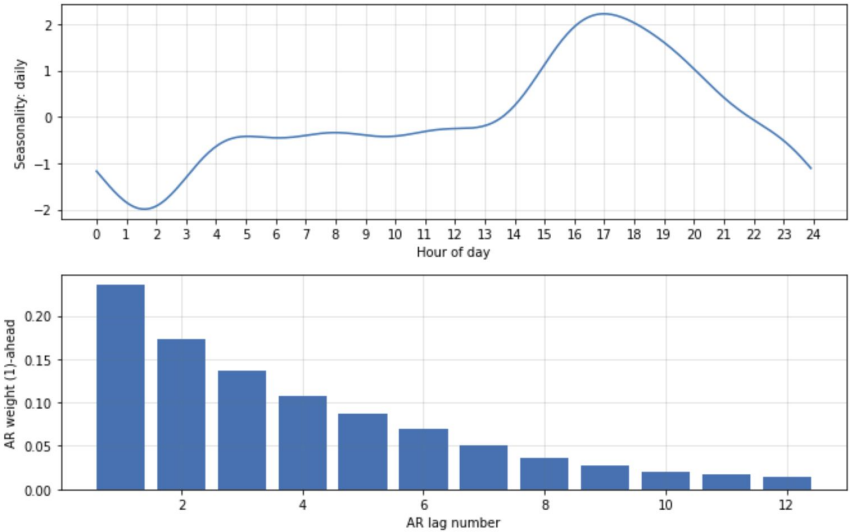


Visualize model parameters with one line.

Example: Yos1

```
m.plot_parameters()
```



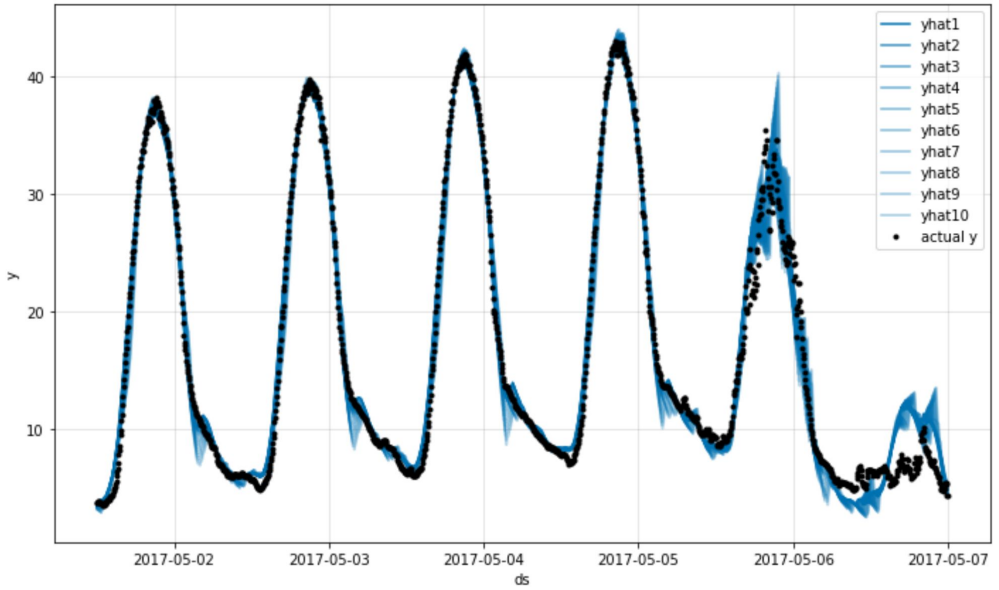


Yosemite Temperature - Predict next 36 observations.

Example: Yos36

```
m = NeuralProphet(
      n_lags=72,
      n_forecasts=36,
)
metrics = m.fit(df, freq='D')
forecast = m.predict(df)
```

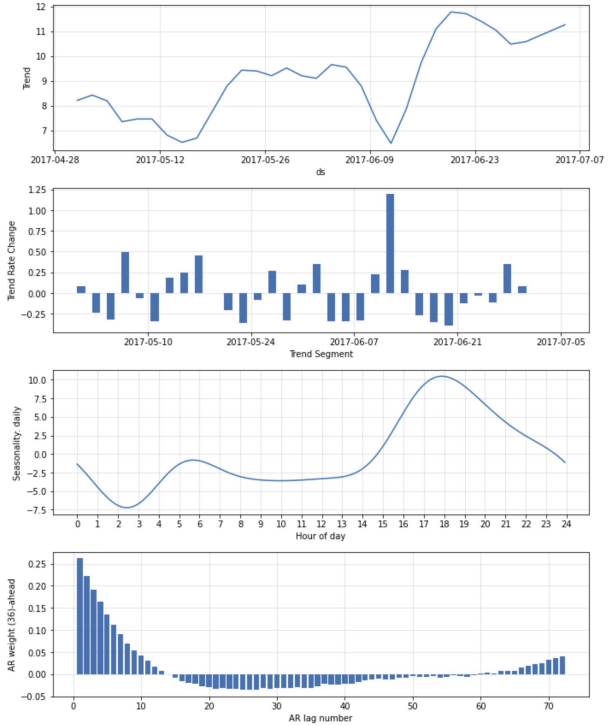
```
m.plot(forecast[:6*24*12])
```



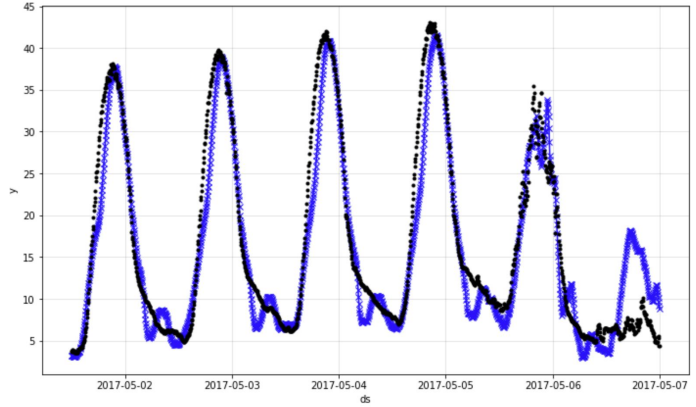
Analyze a specific forecast horizon. Sparsify.

Example: Yos36

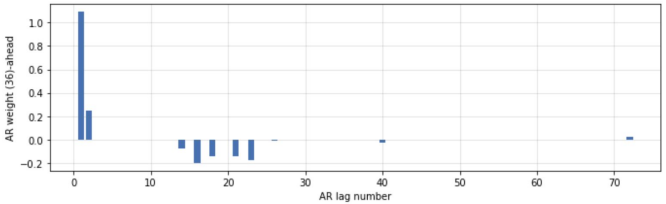
```
m.highlight_nth_step_ahead_of_each_forecast(36)
m.plot_parameters()
```




```
m.plot(forecast[:6*24*12])
```




```
NeuralProphet(ar_sparsity=0.1, ...)
```

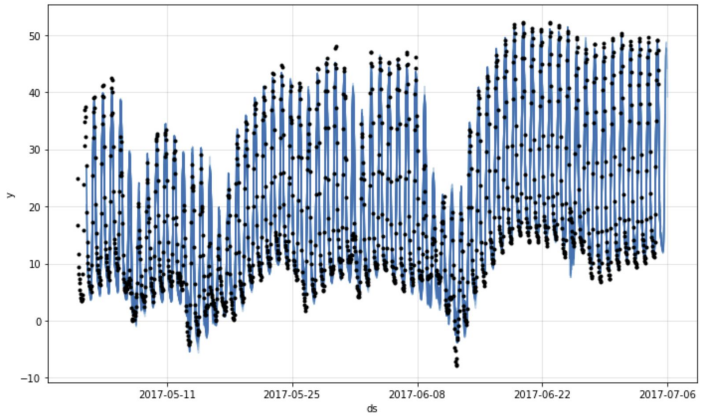


Want to forecast a larger horizon?

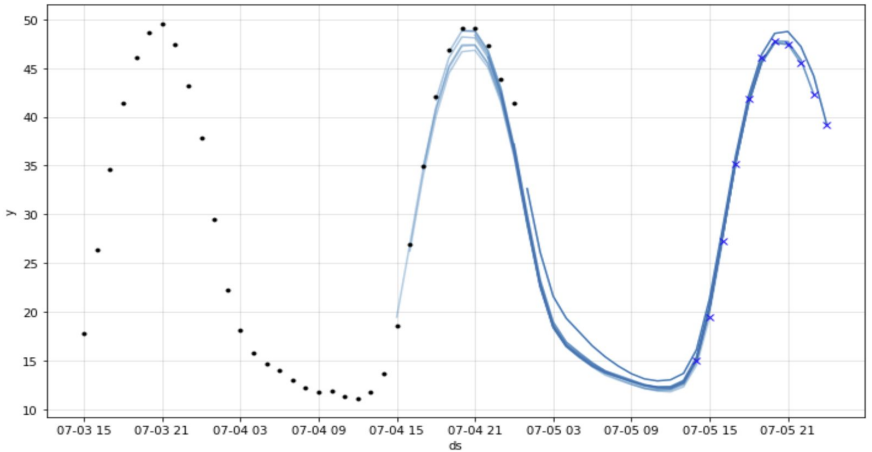
Example: Yos24

```
df_hourly = df.set_index('ds', drop=False).resample('H').mean().reset_index()
```

```
m = NeuralProphet(
    n_lags=24,
    n_forecasts=24,
)
metrics = m.fit(df_hourly, freq='H')
forecast = m.predict(df_hourly)
m.plot(forecast)
```




```
future = m.make_future_dataframe(df_hourly)
forecast = m.predict(future)
m = m.highlight_nth_step_ahead_of_each_forecast(24)
m.plot_last_forecast(forecast, include_previous_forecasts=10)
```



We are working hard to extend the framework. Join us!

Extensions [upcoming]

- Hierarchical Forecasting
  & Global Modelling
- Quantifiable and Explainable Uncertainty
- Anomaly Prediction
  & Semi-Supervised Learning
- Attention: Automatic Multimodality
  & Dynamic Feature Importance

Improvements [upcoming]

- Improved NN
- Faster Training Time
  & GPU support
- Improved UI
- Diagnostic Tools for Deep Dives

```
Anything trainable by gradient descent can be added as module
```

Extension: Global Modelling

Scenario: Dataset with many timeseries of the same domain.

Goal: Fit a single forecasting model with shared weights

Why:

- Better generalization and model size savings
- A single time series may not reflect the entire time series dynamics
  - e.g. anomaly prediction (Zhu and Laptev 2017), tourism demand (Claveria and Torra 2014)
- Extrapolation to new times series

Example Demonstrations:

- Energy load: Forecasting on previously unseen data
- Engine faults: Predicting anomalies with a single model

Forecast energy load of region without prior data.

Extension: Global

ERCOT hourly load

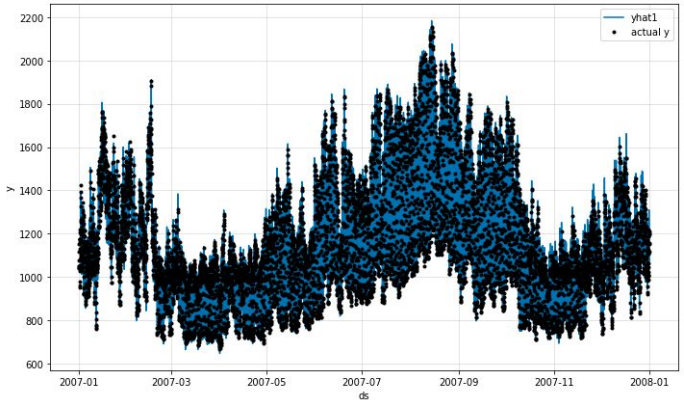
Data: Train on 3 years, 1 year test (unseen region North)

Model: 24 hours lags, 1 step ahead

Coast
East
• • •
West
(Without North)

→ `m = NeuralProphet(n_lags=24)`
`m.fit(list_of_dfs,freq='H')`

North → `m.test(df_north)`



Model	Train dataset	MAE	RMSE
1. Global NP	All regions but North	27.45	33.95
2. Local NP	North	16.09	21.47
3. Local Prophet	North	147.30	191.58
4. Naive Model	North	175.58	230.09

Engine failure - supervised class prediction

Extension: Global

Data: NASA ProDiMES Simulator

- 500 engines with 40 flights each (time-stamps)
- Test on unseen data: 100 engines

Model:

- 1 step ahead (predict class of next flight)
- Covariates: 5 lags from 14 sensors

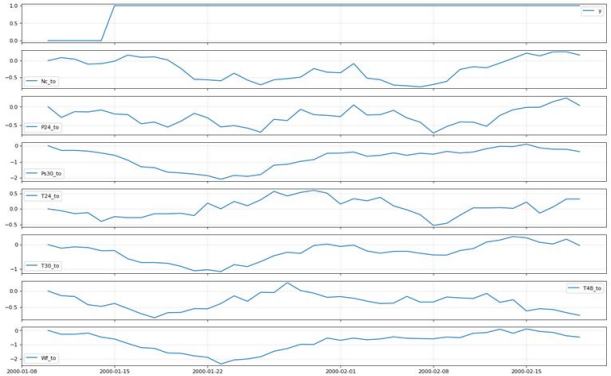
Engine 1
Engine 2
...
Engine 400

```
m = Classification_NP()
m = m.add_lagged_regressor(
        names=sensors,
        n_regressors=5)
m.fit(list_of_dfs, freq='D')
```

Engine 401
Engine 402
...
Engine 500

```
m.test(list_of_test_dfs)
```

sensors



Model	Accuracy	Balanced Acc
1. Global Classifier NP	0.79	0.80
2.Logistic Regression	0.77	0.75
3.Dummy Classifier	0.57	0.50

Global Modelling: Next Steps

Global Modelling

- Local normalization
- Local/global model components

  E.g. Trend, seasonality

Class Prediction

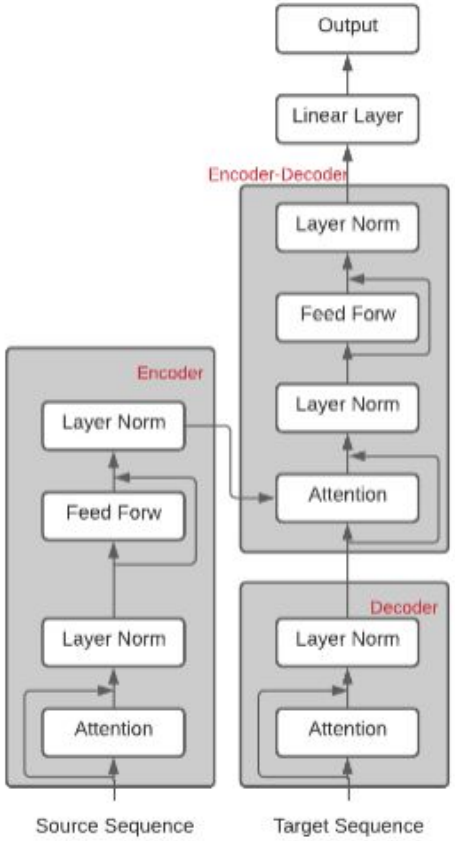
- Cross-correlate covariates
- Multi-class targets
- Semi-supervised approach w. Attention

Extension: Attention
Dynamic Weighted Features & Ensembles

- Transformer models have become popular in the world of NLP over the last few years.
- Transformers get its power from its attention module.
  - Attention has the ability to capture relationships between each feature with every other feature within the sample.

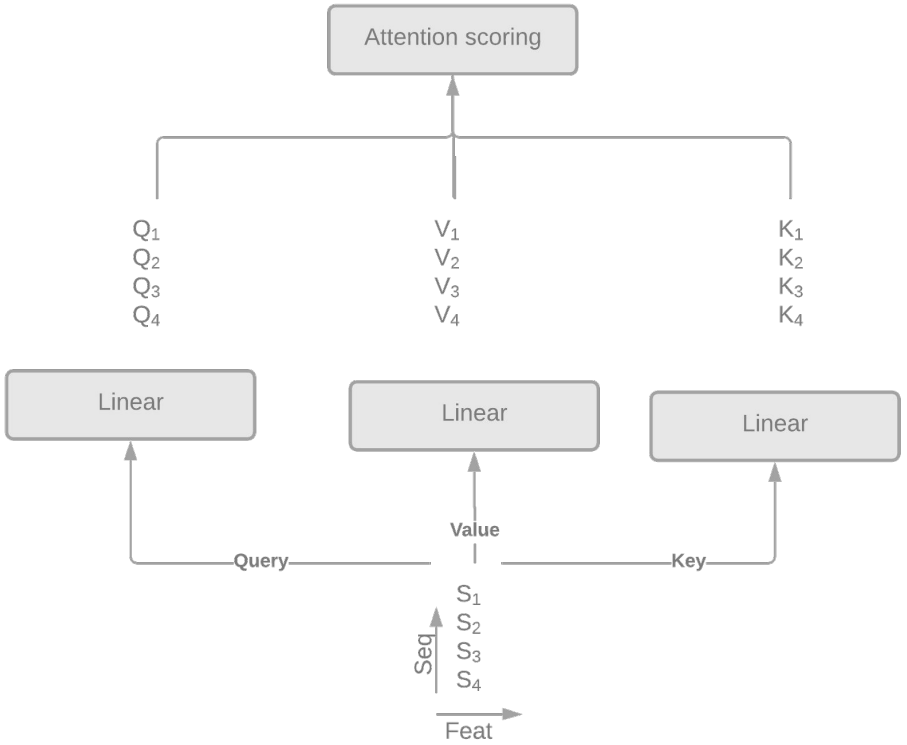
Transformer Models

- **Encoder**: Pays self-attention over source sequence.
- **Decoder**: Pays self-attention over Target sequence.
- **Encoder-Decoder**: Target pays attention to source.

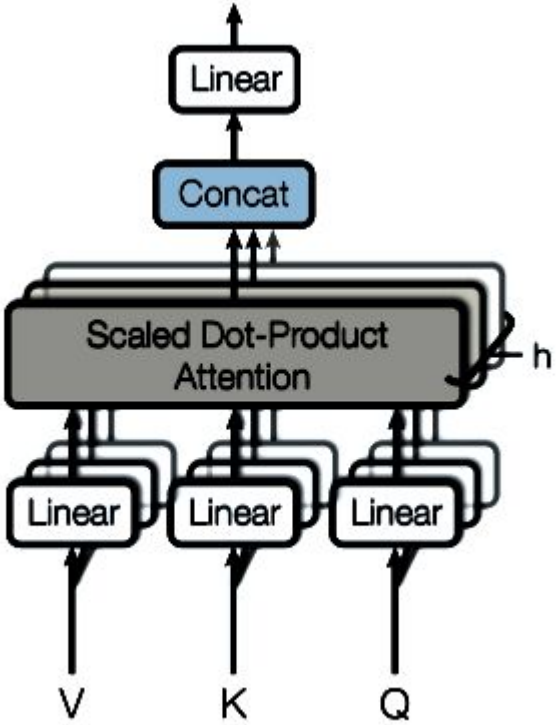


Dot Product Attention Scoring

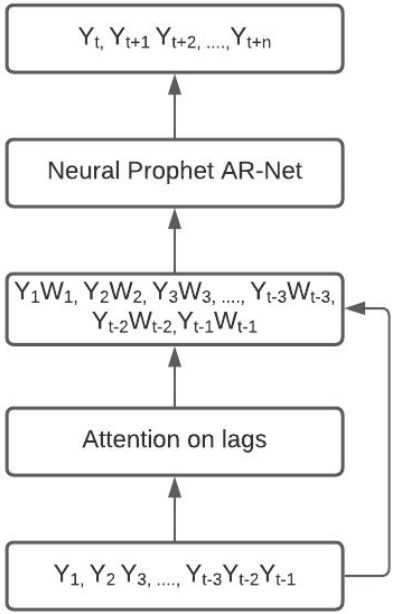
$$Q_1K_1V_1 + Q_1K_2V_2 + Q_1K_3V_3 + Q_1K_4V_4$$
$$Q_2K_1V_1 + Q_2K_2V_2 + Q_2K_3V_3 + Q_2K_4V_4$$
$$Q_3K_1V_1 + Q_3K_2V_2 + Q_3K_3V_3 + Q_3K_4V_4$$
$$Q_4K_1V_1 + Q_4K_2V_2 + Q_4K_3V_3 + Q_4K_4V_4$$



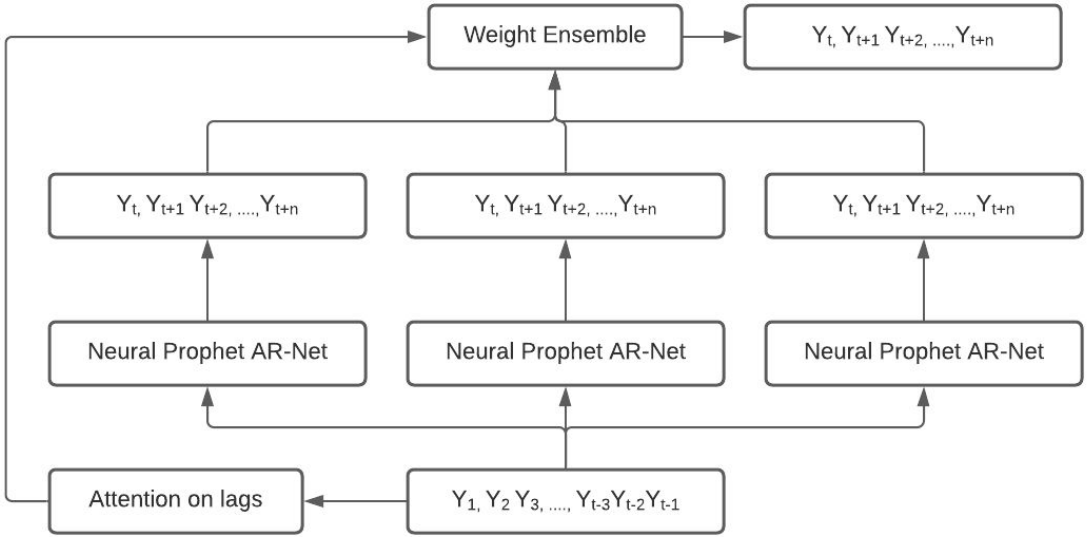
Multihead Attention



Dynamic Feature Weighting



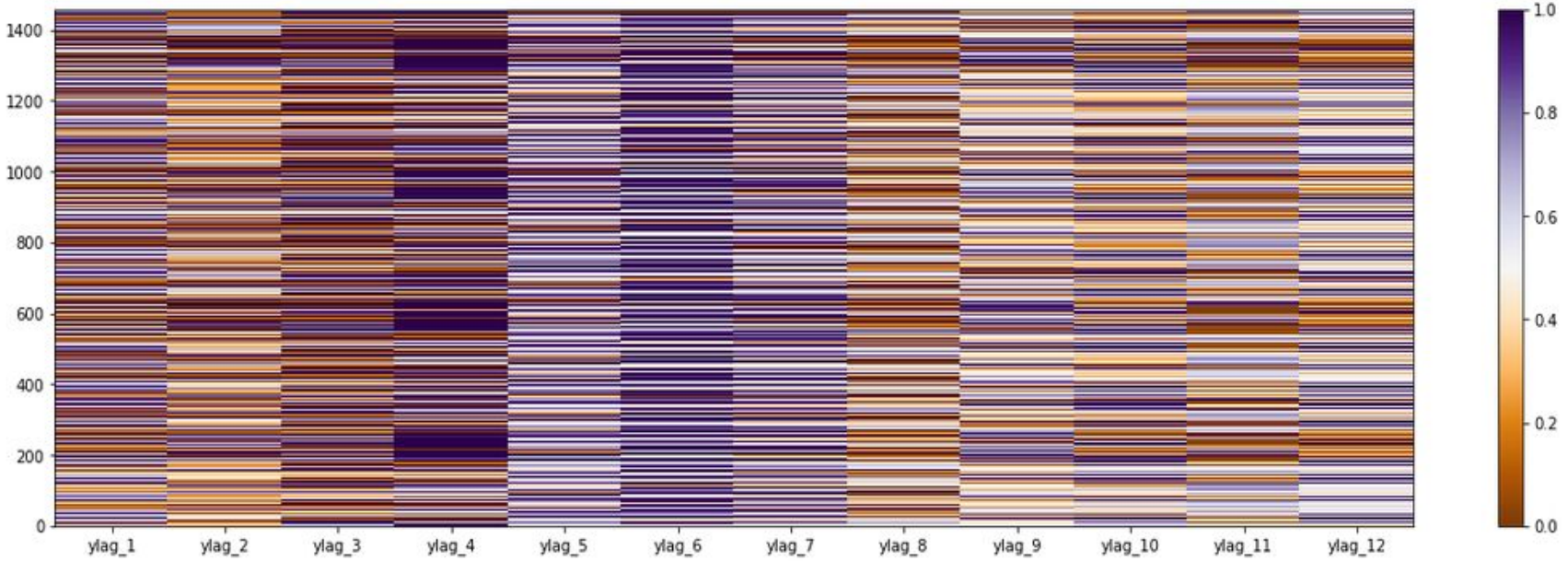
Dynamic Ensemble Weighting



Preliminary Results (MASE)

		Horizon		
Dataset	Model	6	12	24
Solar-Irradiance	w/ Attention	1.087	17.686	18.756
	w/o Attention	2.69	19.46	24.714
ETT-h1	w/ Attention	35.995	92.373	186.349
	w/o Attention	36.761	92.695	188.281
Energy Load	w/ Attention	62.246	144.266	259.45
	w/o Attention	65.423	152.005	274.744

Feature Importance Plot



Ablation studies planned

- Splitting of multiple heads in Multi-head attention
  - One head per lag/feature variable
  - Strategic Feature split
  - Copy instead of split
- Combining of multiple heads in Multi-head attention
  - Sum→sigmoid
  - Concatenate→Linear Layer→sigmoid
  - sum→sigmoid/len(attention out)
  - Concatenate→Linear Layer→sigmoid/len(attention out)

Conclusion & Future Direction

- Implementing attention on neural prophet's AR-Net model has not only made the model more explainable but also improved the forecast accuracy on univariate models.

- Future directions will be to extend this for global and multivariate models.

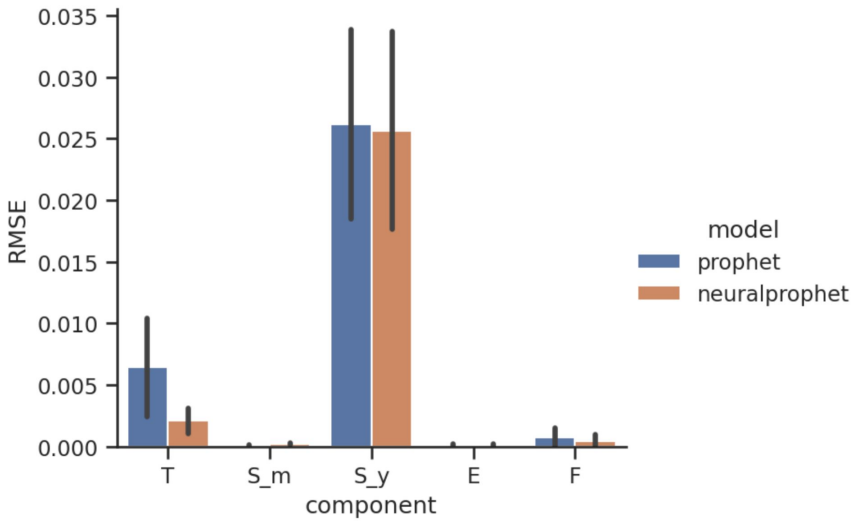
NeuralProphet is a modern Prophet with a superset of its features.

Task	Prophet	NeuralProphet
Very small dataset (less than 100 samples)	✓	
Large dataset (more than 1000 samples)		✓
Long range forecast (e.g. multiple years)	✓	✓
Short to medium range forecast (e.g. 1 to 1000 steps ahead)		✓
Specific forecast horizon (e.g. next 24h)		✓
Auto-correlation (dependence on previous observations)		✓
Lagged regressors (observed covariates)		✓
Non-linear dynamics		✓
Global modelling of panel dataset		✓
Fast prediction (computational inference time)		✓

Benchmark Teaser: NeuralProphet vs Prophet.

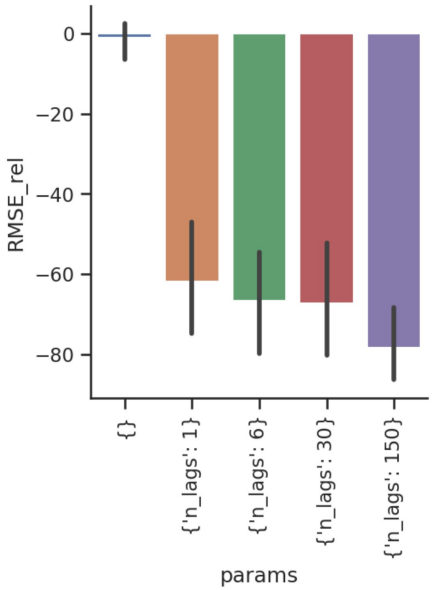
Accuracy of forecast decomposition

On synthetic dataset.



Forecast error difference to Prophet

On a collection of real datasets.



Conclusion: Upgrade Prophet.

Simple and powerful forecaster,

without compromising on interpretability.

1:1 replacement for Prophet,

with many new capabilities, superior for most applications.

Nothing but Python & PyTorch,

extensible to future state-of-the-art in forecasting.

THANK YOU, dear collaborator, supporter and advisor!

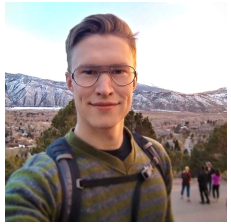
Stanford University

facebook

MONASH University

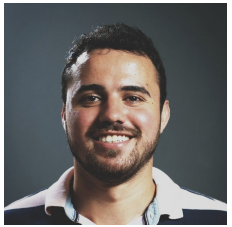
TotalEnergies

Skoltech



Oskar Triebe

Hansika Hewamalage



Mateus De Castro Ribeiro

Lluvia Ochoa

Nikolay Laptev

Polina Pilyugina

Abishek Sriramulu

Ram Rajagopal

Christoph Bergmeir

Alessandro Panella

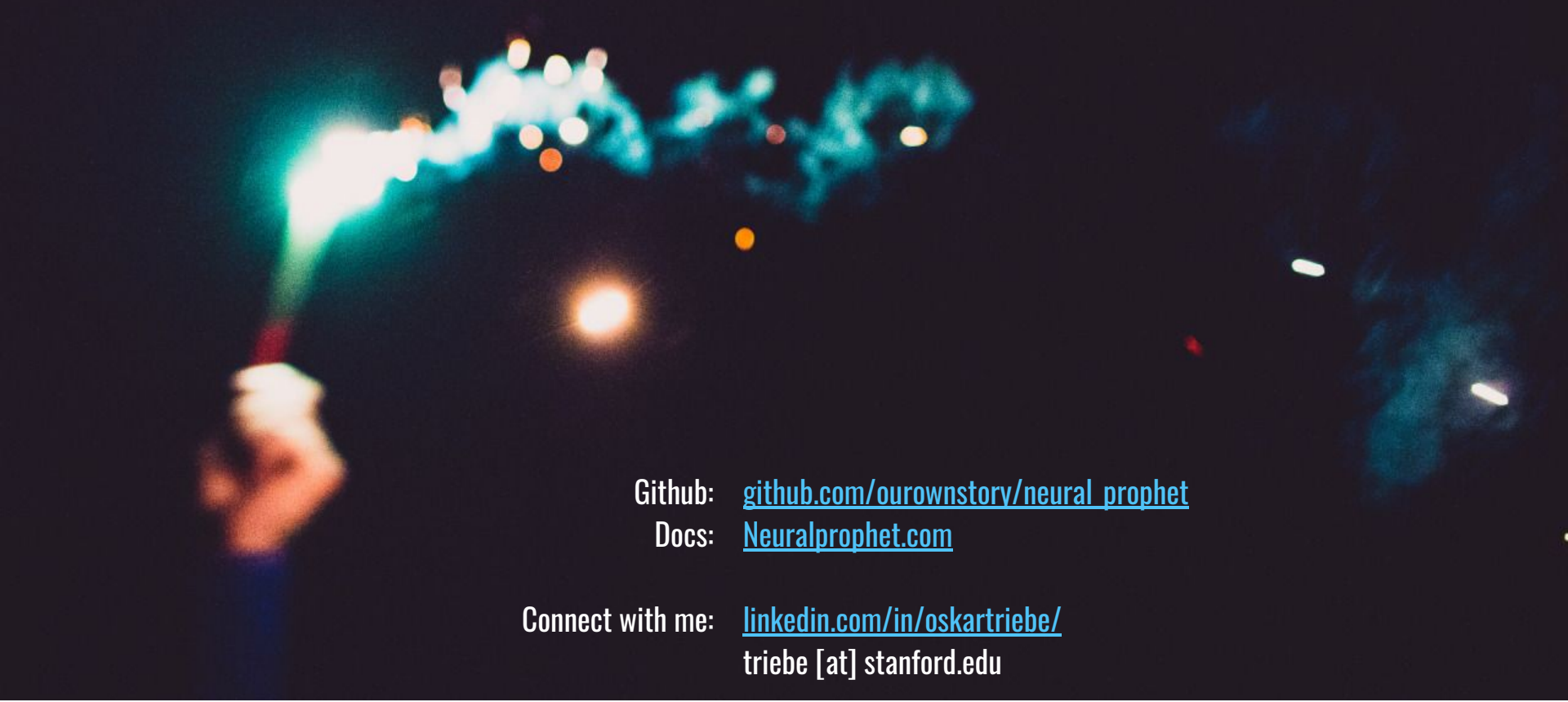
Evgeny Burnaev

Caner Komurlu

Italo Lima

Gonzague Henri

Bernhard Hausleitner



Github: github.com/ourownstory/neural_prophet
Docs: Neuralprophet.com

Connect with me: linkedin.com/in/oskartriebe/
triebe [at] stanford.edu

Thank You

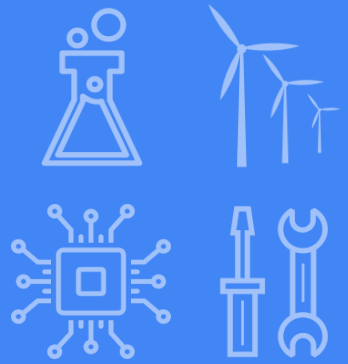
References

- Claveria, O., & Torra, S. (2014). Forecasting tourism demand to Catalonia: Neural networks vs. time series models. *Economic Modelling*, 36, 220-228.
- Zhu, L., & Laptev, N. (2017, November). Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* (pp. 103-110). IEEE.

Appendix: AR-Net

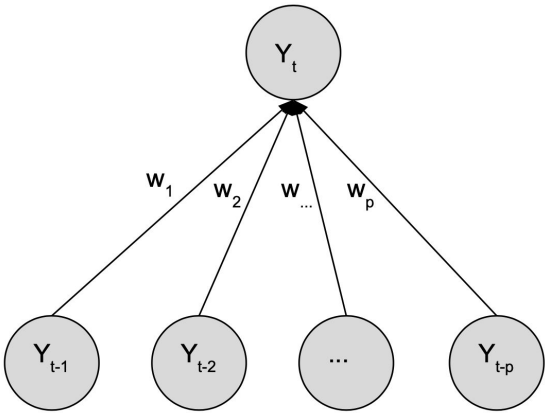
A time series is

a sequence of data points
that occur in successive order
over some period of time.

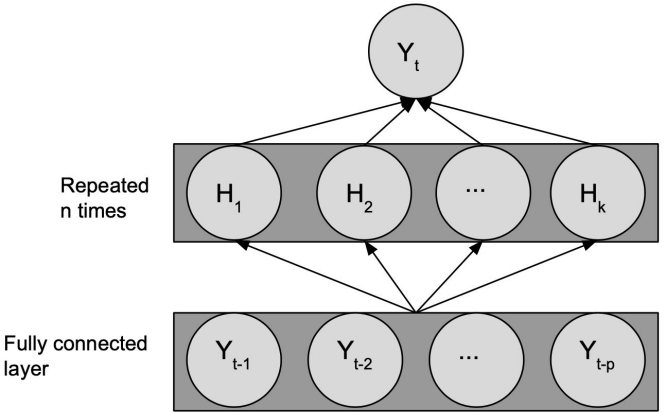


AR-Net is a Neural Network for autoregressive time series.

AR-Net(0)
Interpretable



AR-Net(n)
Stronger modeling ability



Skip estimating the AR process order.

Model: AR-Net

$$y_t = c + \sum_{i=1}^{i=p} w_i * y_{t-i} + e_t$$

$$\downarrow$$

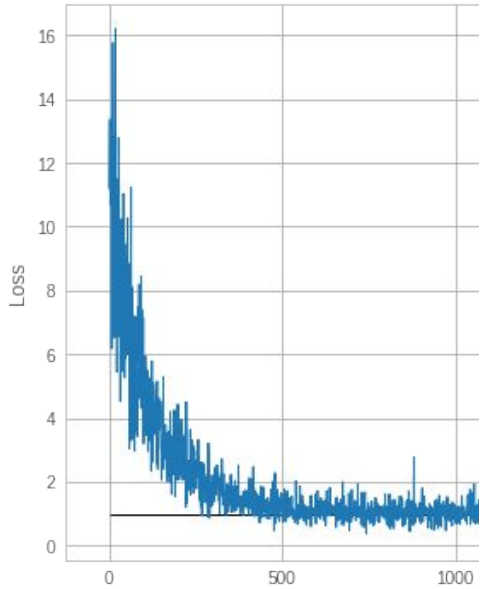
$$min_\theta \quad L(y, \hat{y}, \theta) + \lambda(s) \cdot R(\theta)$$

$$\lambda(s) = c_\lambda \cdot (s^{-1} - 1)$$

$$s = \frac{\hat{p}_{data}}{p_{model}}$$

$$c_\lambda \approx \frac{\sqrt{\hat{L}}}{100}$$

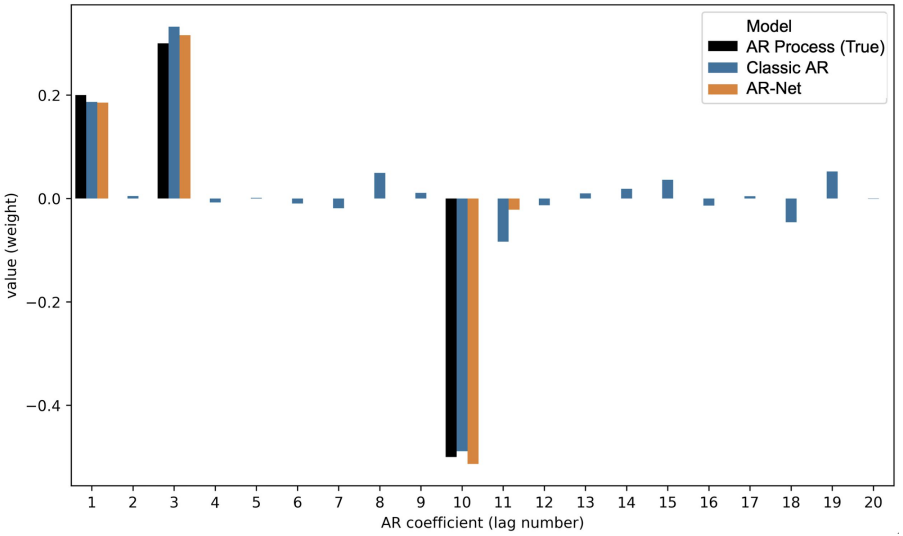
$$R(\theta) = \frac{1}{p} \sum_{i=1}^{p} \frac{2}{1 + \exp(-c_1 \cdot |\theta_i|^{\frac{1}{c_2}})} - 1$$



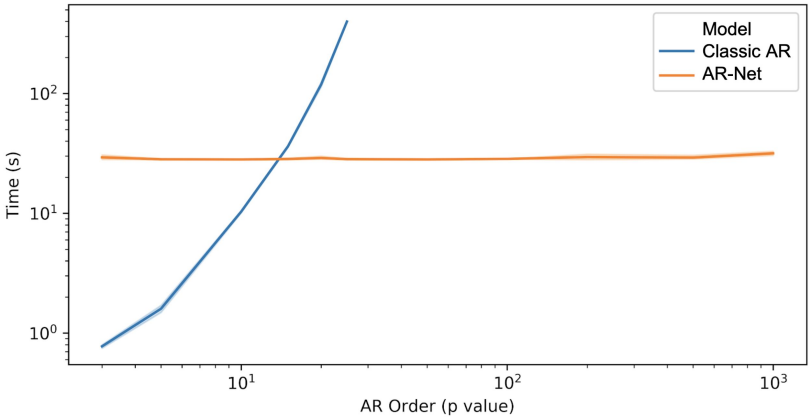
Trained with SGD (Adam)

Automatic AR-lag selection, yet faster.

Model: AR-Net



Automatic Sparsity



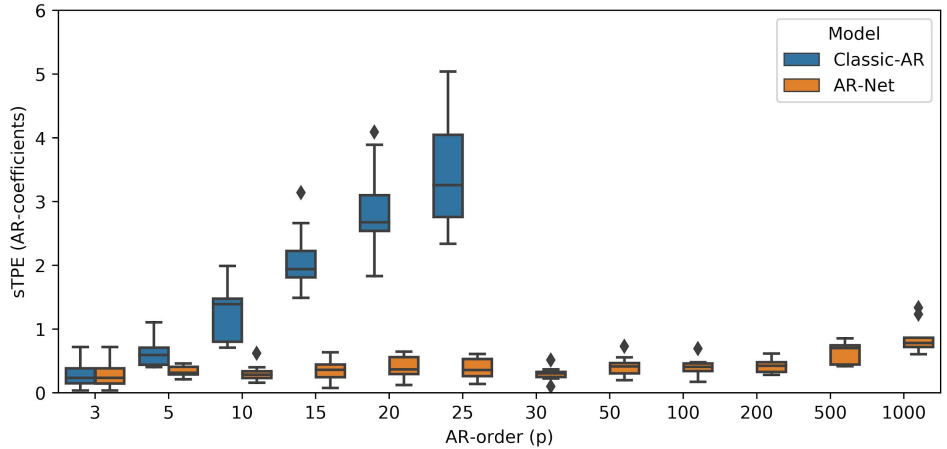
Quadratically faster

Sparse AR-Net surpasses Classic AR and scales to large orders.

Model: AR-Net

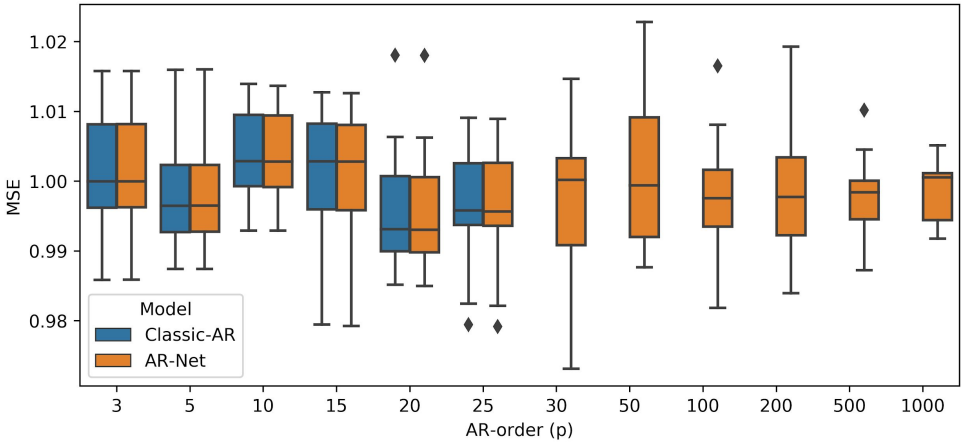
Closeness to true coefficients

$$sTPE = 100 \cdot \frac{\sum_{i=1}^{i=p} |\hat{w}_i - w_i|}{\sum_{i=1}^{i=p} |\hat{w}_i| + |w_i|}$$



MSE loss on forecast target

$$MSE = \frac{1}{n}\sum_{1}^{n}(y_t - \hat{y}_t)^2$$



Appendix: Model Use Details

Few lines from data to forecast.

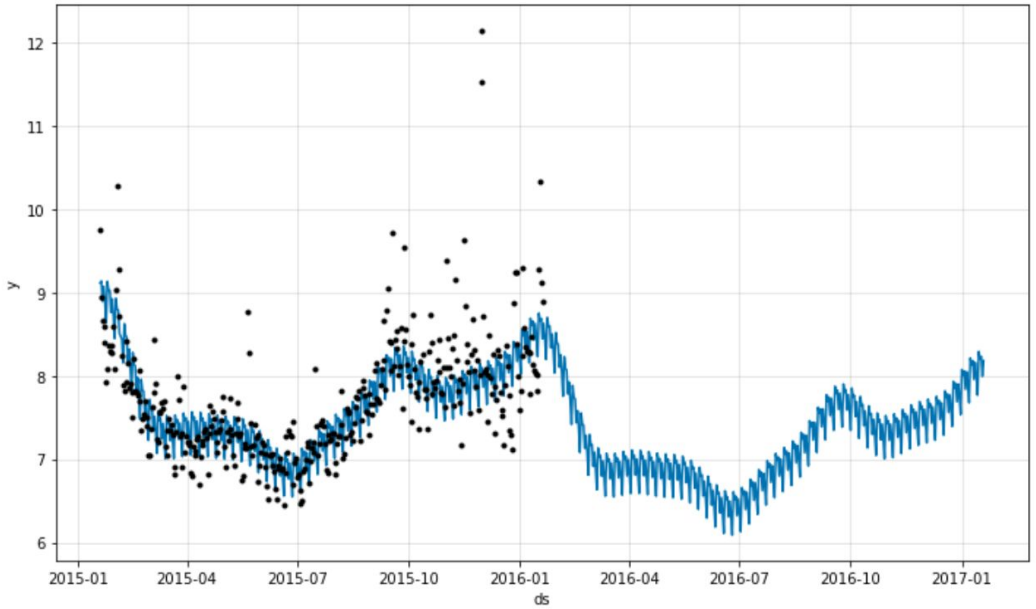
Hands-on

skip data preparation: → Just create a DataFrame with desired columns

```
import pandas as pd
from neuralprophet.neural_prophet import NeuralProphet
```

```
df = pd.read_csv('../data/example_wp_log_peyton_manning.csv')
```

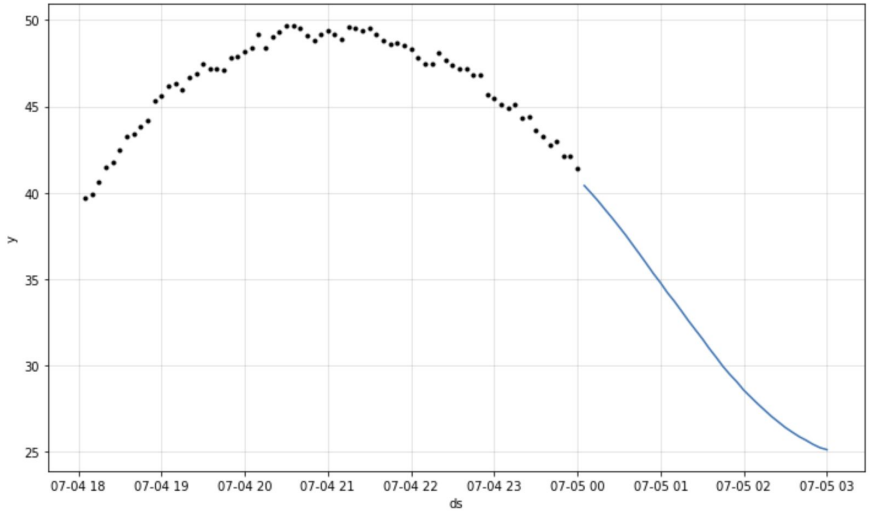
```
# linear time-dependent model
m = NeuralProphet()
metrics = m.fit(df)
future = m.make_future_dataframe(df, future_periods=365)
forecast = m.predict(future)
fig_fcst = m.plot(forecast[-730:])
```



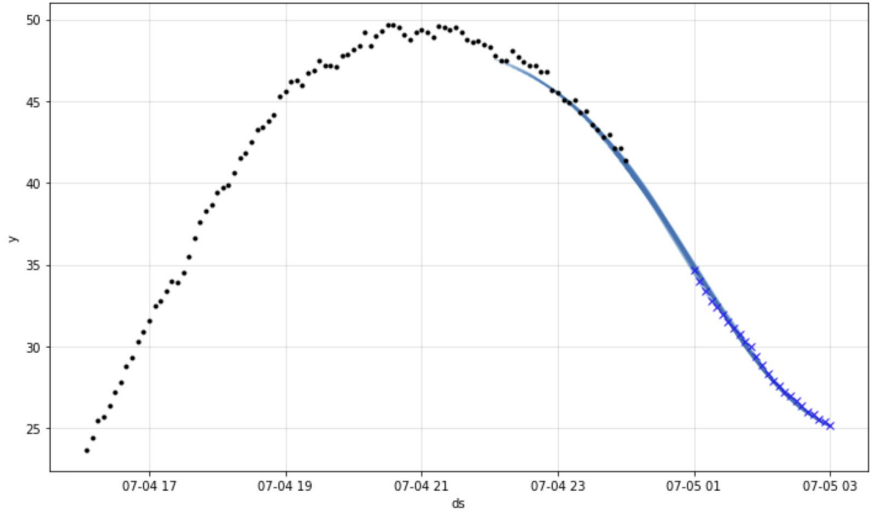
Predict. See how new data impacts the forecast.

Example: Yos36

```
m = m.highlight_nth_step_ahead_of_each_forecast(None) # reset highlight
fig = m.plot_last_forecast(forecast)
```




```
m = m.highlight_nth_step_ahead_of_each_forecast(3*12)
fig = m.plot_last_forecast(forecast, include_previous_forecasts=2*12)
```



Validate model performance without extra lines.

Hands-on

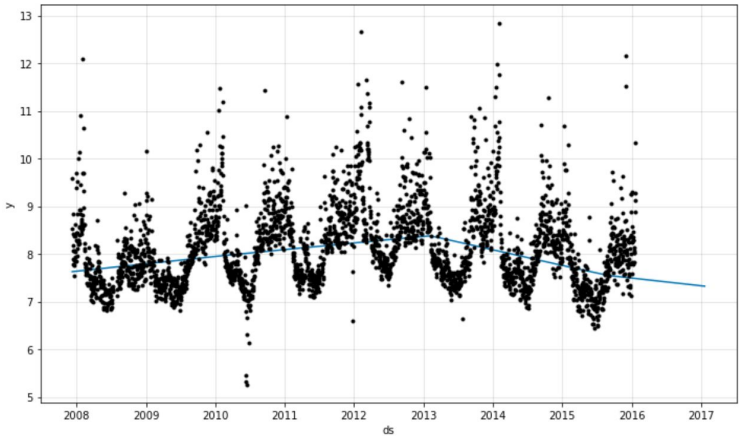
```
# or evaluate while training
m = NeuralProphet()
metrics = m.fit(df, validate_each_epoch=True, valid_p=0.2)
metrics.tail()
```

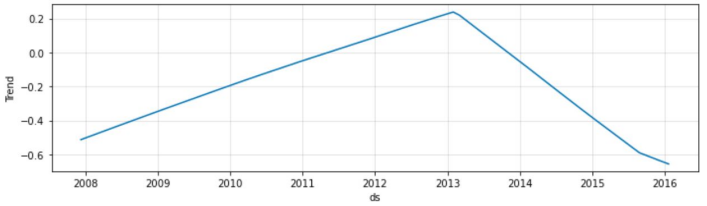
```
Disabling daily seasonality. Run NeuralProphet with daily_seasonality=True to override this.
```

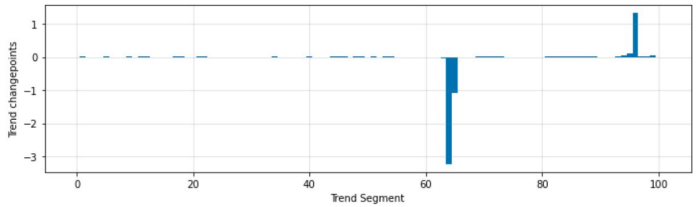
	SmoothL1Loss	MAE	RegLoss	SmoothL1Loss_val	MAE_val
35	0.163102	0.371323	0.0	0.485371	0.779465
36	0.161851	0.369609	0.0	0.368921	0.648736
37	0.161122	0.369219	0.0	0.366328	0.648230
38	0.168598	0.376638	0.0	0.348269	0.627886
39	0.167961	0.375777	0.0	0.362161	0.642699

```
# split manually
m = NeuralProphet()
df_train, df_val = m.split_df(df, valid_p=0.2)
train_metrics = m.fit(df_train)
val_metrics = m.test(df_val)
```

```
m = NeuralProphet(
    n_changepoints=100,
    trend_smoothness=2,
    yearly_seasonality=False,
    weekly_seasonality=False,
    daily_seasonality=False,
)
metrics = m.fit(df)
```



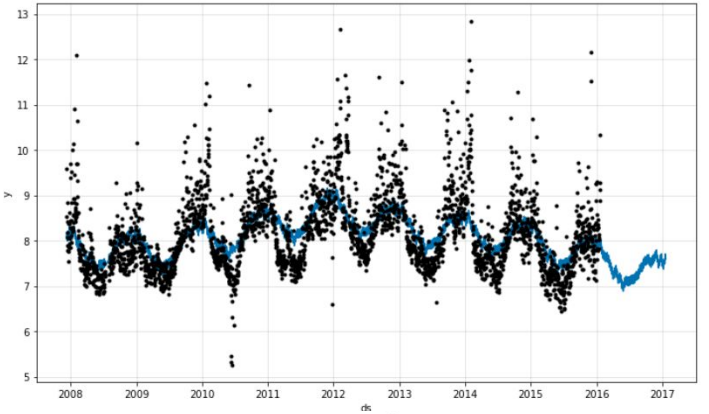


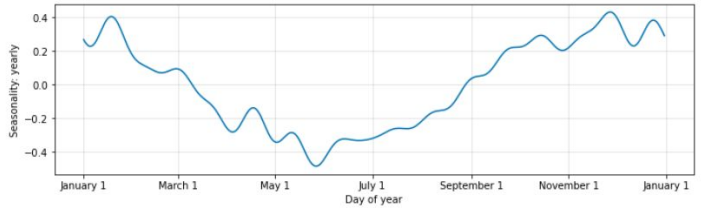


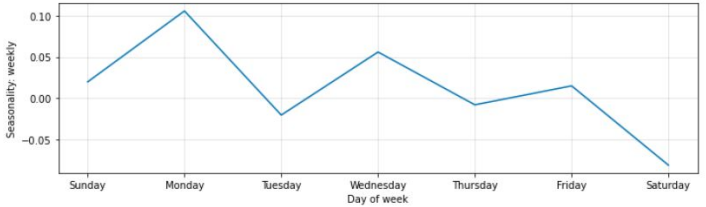
Seasonality

Hands-on

```
m = NeuralProphet(
    yearly_seasonality=16,
    weekly_seasonality=8,
    daily_seasonality=False,
    seasonality_reg=1,
)
metrics = m.fit(df)
```







Events can be added in different forms.

Hands-on

```
superbowls = pd.DataFrame({'event': 'superbowl',
'ds': pd.to_datetime(['2010-02-07', '2014-02-02', '2016-02-07'])})
```

```
events_df = pd.concat((superbowls, playoffs))
```

```
m = NeuralProphet()
```

```
m = m.add_country_holidays("US")
```

```
m = m.add_events(
    ["superbowl", "playoff"],
    lower_window=-1,
    upper_window=1,
    mode="multiplicative",
    regularization=0.5
)
```

```
history_df = m.create_df_with_events(df, events_df)
metrics = m.fit(history_df)
```

```
Disabling daily seasonality. Run NeuralProphet with daily_seasonality=
True to override this.
```

```
future = m.make_future_dataframe(
    history_df,
    events_df,
    future_periods=30,
    n_historic_predictions=30
)
```

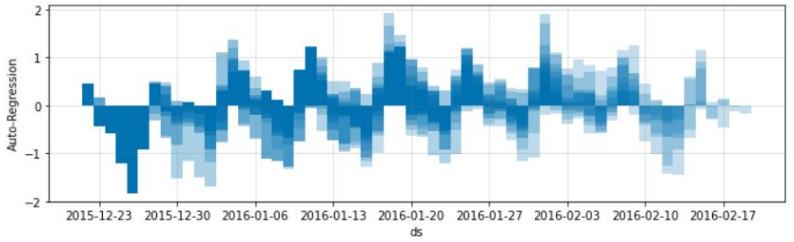
```
forecast = m.predict(future)
```

```
m = NeuralProphet(
    n_forecasts=30,
    n_lags=60,
    ar_sparsity=0.1,
    yearly_seasonality=False,
    weekly_seasonality=False,
    daily_seasonality=False,
)
```

Autoregression, here with sparsity

```
metrics = m.fit(df)
future = m.make_future_dataframe(df, n_historic_predictions=30)
forecast = m.predict(future)
```

```
fig_comp = m.plot_components(forecast[60:])
```

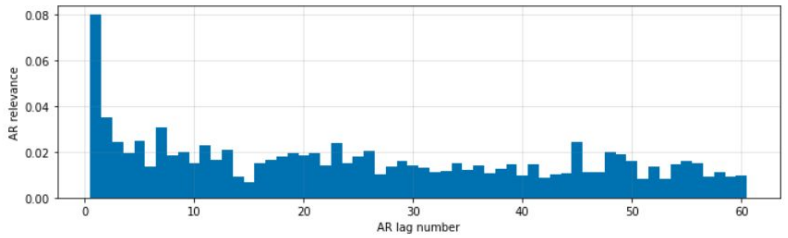


Recommended example notebook:

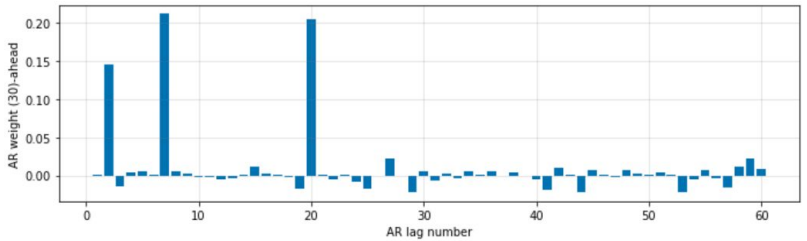
https://github.com/ourownstory/neural_prophet/blob/master/example_notebooks/autoregression_yosemite_temps.ipynb

Focus on a specific forecast.

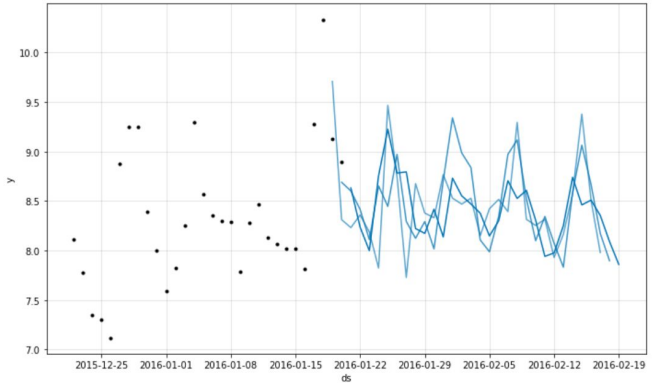

```
fig_param = m.plot_parameters()
```

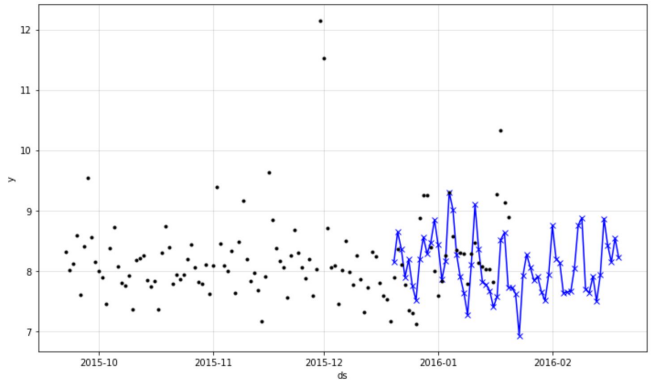



```
fig_param30 = m.highlight_nth_step_ahead_of_each_forecast(30).plot_parameters()
```




```
fig_prediction = m.plot_last_forecast(forecast[60:],
include_previous_forecasts=2)
```





Lagged Covariates & Future Regressors

Hands-on

```
m = NeuralProphet(
    n_forecasts=30,
    n_lags=60,
    ar_sparsity=0.1,
    yearly_seasonality=False,
    weekly_seasonality=False,
    daily_seasonality=False,
)
```

```
df = pd.read_csv('../data/example_wp_log_peyton_manning.csv')
df['A'] = df['y'].rolling(7, min_periods=1).mean()
df['B'] = df['y'].rolling(60, min_periods=1).mean()
m = m.add_covariate(name='A')
m = m.add_regressor(name='B')
```

```
metrics = m.fit(df)
future = m.make_future_dataframe(df, n_historic_predictions=30)
forecast = m.predict(future)
```

Covariates and regressors are similarly added

```
m = NeuralProphet(
    n_forecasts=30,
    n_lags=60,
    learning_rate=1.0,
    loss_func='Huber',
    normalize_y=True,
    num_hidden_layers=2,
    d_hidden=64,
)
```

Make it non-linear.

```
fig_comp = m.plot_components(forecast[60:])
```