Jonathan Bechtel Follow
Jan 9 · 6 min read · Listen

KerasBeats: An Easy Way to Use N-Beats in Keras

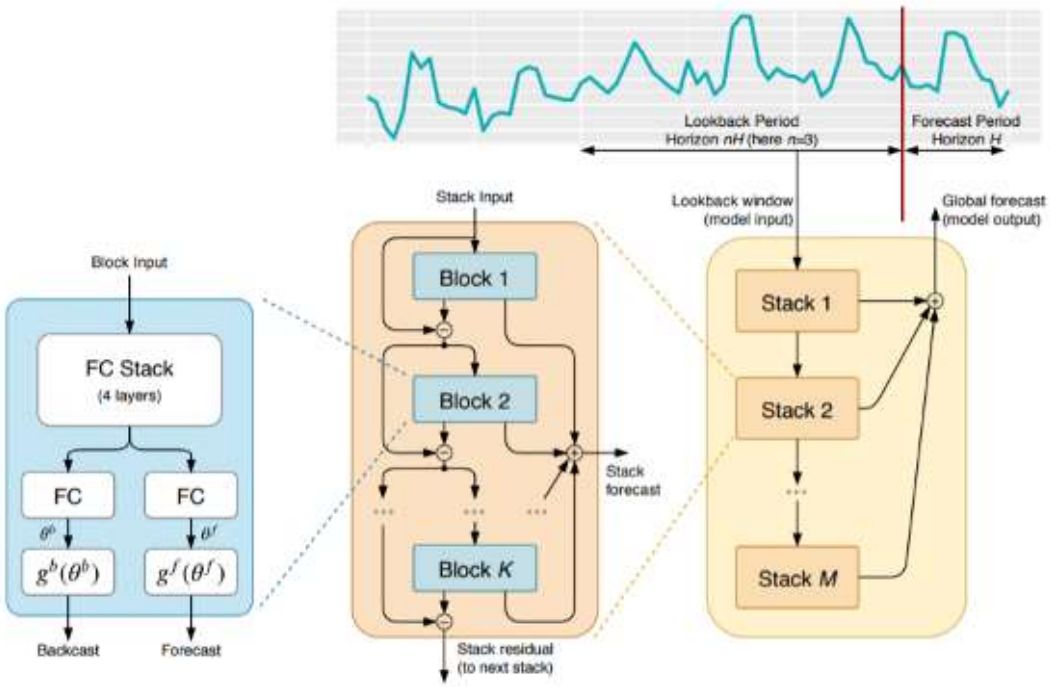


Image Courtesy of Element AI at: https://arxiv.org/pdf/1905.10437.pdf

I work with forecasting models a lot, using a mix of both old and new-school techniques at work.

Up until this point, LSTM's have been the default method for people who wanted to use deep neural nets to predict future time series observations. After perusing the recent cryptocurrency kaggle competition, I was made aware of a new architecture that

The model was developed by Yoshua Bengio, and achieved state-of-the-art results (with a lot of ensembling) in a variety of forecasting benchmarks. The model architecture is also appealingly simple. It only consists of collections of dense layers with relu activations stacked on top of each other. Each collection of dense blocks comes with a basis function that modifies the way it creates its forecast and "backcast" (a way of maintaining information about the time series the model has already learned).

The Current Problem

After reading through the paper, I went searching the internet for implementations of it in Keras, but found the current choices lacking. Lots of blog posts, a couple of installable packages, but nothing with decent documenation or smoothed edges. The most popular implementation is in this repo, but it was originally written in pytorch, and found that it lacked the sort of just-push-play functionality that makes it easy to get started.

KerasBeats: N-Beats Done the Easy Way in Keras

So to scratch my own itch, I dug into the original paper, the github repo for the original project, and tried to create what I would want out of an implementation for someone already familiar with keras and anxious to get started right away.

KerasBeats is an attempt to make it dead simple to implement N-Beats with just a few lines of code using the keras deep learning library.

Here's an example using this dataset from kaggle, which we'll use to predict temperature in New Delhi.

Installation:

```
pip install keras-beats
```

```
from kerasbeats import prep_time_series, NBeatsModel

# import the dataset
df = pd.read_csv('put_the_file_linked_above_here.csv', parse_dates =
['date'], index_col = 'date')

# sort by dates
df.sort_index(inplace = True)

# prep a univariate time series for N-Beats
X, y = prep_time_series(df['meantemp'], lookback = 7, horizon = 1)

# create training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
shuffle = False, test_size = 0.2)

# initialize N-Beats and fit
nbeats = NBeatsModel(model_type = 'generic', lookback = 7, horizon =
1)
nbeats.fit(X, y, epochs = 30)
```

And that's it. When you call `fit`, you are utilizing the underlying keras method, so you can pass whatever you'd like into the function call. When you are finished the `nbeats` object would make available the `score` and `evaluate` method that access the underlying keras methods.

Let's take a few moments to break down what's going on in the above code:

— `prep_time_series` is a helper function that creates training windows and their labels from the provided time series. `lookback` dictates the size of your training window, and horizon is how far into the future your model will predict. `lookback` creates the training window by determining the multiple of the horizon that you will use. A `horizon` of 1 and `lookback` of 7 will create a training window from the previous 7 values. A `horizon` of 2 and `lookback` of 4 will create a training window from the previous 8 values.




architecture that does not use a basis expansion, and the `interpretable` architecture that stacks two different model blocks on top of one another. When initializing, the default values for the model are the same listed in the appendix of the paper (page 26), but you can adjust all of them upon initialization.
— `fit` simply invokes the keras method.

This syntax should be very familiar to people who are already using ML libraries today, so onboarding one of the most powerful advances in time series modeling is now just a few function calls away.

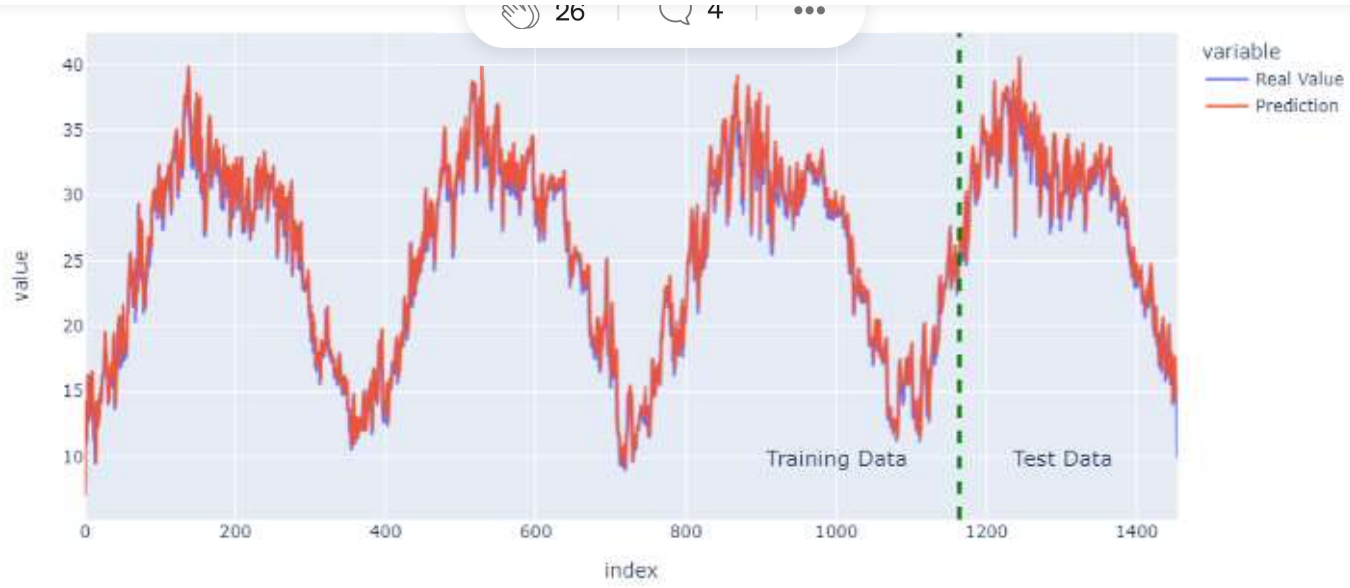
The above code was done with no optimizations and would provide the following results:

```
# helper dataframe to make plotting easier
preds = pd.DataFrame(index = np.arange(len(y_train) + len(y_test)))
preds['Real Value'] = np.hstack([y_train[:, 0], y_test[:, 0]])
preds['Prediction'] = np.hstack([nbeats.predict(X_train)[:, 0],
nbeats.predict(X_test)[:, 0]])

# and plot the predictions
import plotly.express as px
fig = px.line(preds[['Real Value', 'Prediction']], title = 'Real vs
Predicted Values for N-Beats')

# random formatting stuff
fig.add_vline(x = 1164, line_width=3, line_dash="dash",
line_color="green")
fig.add_annotation(x = 1000, y = 10,
 text="Training Data",
 font=dict(size=15),
 showarrow=False)
fig.add_annotation(x = 1300, y = 10,
 text="Test Data",
 font=dict(size=15),
 showarrow=False)
```



Not too shabby!

It's Just Keras

When you initialize `NBeatsModel` you are just creating a high level wrapper around a several keras building blocks. It's possible you might want to use those directly and disregard the higher level API. This is very easy to do in kerasbeats.

Model Layer

The most basic building block in kerasbeats is the `model_layer` , which is the collection of stacked blocks that are used to build the forecasts. If you just want this to use in larger projects, you can do it with the `build_layer` method:

```
# initialize
nbeats = NBeatsModel(model_type = 'generic', lookback = 7, horizon =
1)
nbeats.build_layer()

# this is the base NBeats layer that's used to construct the model
nbeats.model_layer
```

For the dataset in this article you could incorporate information about humidity, windspeed and airpressure into your predictions. If you wanted to combine those with the NBeats model, you could do so in the following way:

```
from tensorflow import keras
time_input = keras.layers.Input(shape = (7, ))

# layer we created from the previous block
nbeats_layer = nbeats.model_layer(time_input)

# input for non time data
non_time_input = keras.layers.Input(shape = (3,))
x = keras.layers.Dense(32, activation = 'relu')(non_time_input)

#concatenate the layers and combine into a model
concat = keras.layers.Concatenate()([nbeats_layer, x])
output = keras.layers.Dense(1)(concat)
model = keras.Model(inputs = [time_input, non_time_input], outputs =
output)
```

And we could jointly build a model around all of our data like so:

```
# create training data from non time values
# excluding the first 7 values to match our time series data
X_non_time = df[['humidity', 'wind_speed', 'meanpressure']].values[7:]

# compile the model and fit
model.compile(loss = 'mae')
model.fit([X, X_non_time], y)
```

Use the underlying Keras Model

In a few ways the top level abstractions in `NBeatsModel` interferes with how you would normally use a keras model:




This would prevent you from passing in those values to the `fit` method like you normally would.

If either of these is problematic, you can access the uncompiled model with the following function call:

```
nbeats = NBeatsModel().build_layer().build_model()

# this is the underlying keras model

nbeats.model
```

And you could use `nbeats.model` that like you would anything else in Keras.

Proper Documentation

One frustration I had when searching the internet for ways to use N-Beats is that whatever was out there didn't have polished examples to learn from and thin documenation that made it difficult to jump in and get started.

The README for this project can be found here, and provides most of what you would need to know to get up and running. The project also has its own dedicated site for documenation that's more thorough: https://kerasbeats.readthedocs.io/en/latest/#

You can find all function arguments defined, as well as additional context about how to use the package.