NeuralForecast Forecasting Methods Guide

Overview

This guide summarizes the three main forecasting approaches in NeuralForecast: one-step ahead, multi-step recursive, and multi-output direct forecasting.

---

1. One-Step Ahead Forecast (h=1)

Models predict only the next single time step.

Example Code

```
from neuralforecast import NeuralForecast
from neuralforecast.models import NHITS

model = NHITS(h=1, input_size=24, max_steps=100)
nf = NeuralForecast(models=[model], freq='M')
nf.fit(df=train)
forecasts = nf.predict()  # Returns 1 step ahead
```

Characteristics

- **Simplest approach**: Predicts only t+1
- **Most accurate**: For immediate next step predictions
- **Limitation**: To forecast multiple steps, you need to retrain or use rolling windows
- **Use case**: When you only need to predict the immediate next value

---

2. Multi-Step Ahead Recursive Forecasting (h=1, applied recursively)

Recursive forecast models predict one-step ahead, and subsequently use the prediction to compute the next step in the forecast horizon, and so forth.

Example Code

Available for RNN/LSTM/GRU models only:

```
from neuralforecast import NeuralForecast
from neuralforecast.models import LSTM

model = LSTM(
    h=12,
    input_size=24,
    recurrent=True,  # Enable recursive forecasting
    max_steps=100
```

```
)
nf = NeuralForecast(models=[model], freq='M')
nf.fit(df=train)
forecasts = nf.predict()  # Returns 12 steps using recursive approach
```

How It Works

1. Predicts step 1
2. Uses prediction as input
3. Predicts step 2
4. Repeats until step h is reached

Characteristics

- **Model-specific**: Only RNN, LSTM, and GRU models support the `recurrent` parameter
- **Error propagation**: Suffers from bias and variance propagation as errors accumulate
- **Computationally efficient**: Less expensive than direct methods
- **Parameter**: Set `recurrent=True` in model initialization
- **Use case**: When computational efficiency is more important than maximum accuracy

Available Models with Recursive Option

- `RNN` with `recurrent=True`
- `LSTM` with `recurrent=True`
- `GRU` with `recurrent=True`

---

3. Multi-Output Direct Forecasting (h>1)

Direct forecast models produce all steps in the forecast horizon at once. **This is the default approach for most NeuralForecast models.**

Example Code

```
from neuralforecast import NeuralForecast
from neuralforecast.models import NHITS, NBEATS

model = NHITS(h=12, input_size=24, max_steps=100)
nf = NeuralForecast(models=[model], freq='M')
nf.fit(df=train)
forecasts = nf.predict()  # Returns all 12 steps simultaneously
```

How It Works

- Takes an entire window of past values (input_size)
- Computes all forecast timepoint values in a **single forward pass**
- The neural network architecture is designed to output h values simultaneously

Characteristics

- **Default behavior**: Most NeuralForecast models use this approach
- **Better accuracy**: Suffers less from bias and variance propagation compared to recursive methods
- **Computationally expensive**: Requires more resources than recursive forecasting
- **Single pass**: All h predictions made at once, not iteratively
- **Use case**: When maximum accuracy is needed and computational resources are available

Models Using Direct Forecasting

- **NBEATS** (Neural Basis Expansion Analysis)
- **NHITS** (Neural Hierarchical Interpolation)
- **TFT** (Temporal Fusion Transformer)
- **Informer**
- **Autoformer**
- **PatchTST**
- **MLP**
- **TCN**
- **Most other architectures** (except RNN-based with `recurrent=True`)

Comparison Table

Aspect	One-Step (h=1)	Recursive (h>1)	Direct (h>1)
Forecast Horizon	1 step	Multiple steps	Multiple steps
Prediction Method	Single prediction	Iterative (uses own predictions)	Simultaneous (all at once)
Error Propagation	None	High (accumulates)	Low
Computational Cost	Low	Medium	High
Accuracy	Highest for t+1	Moderate	Highest for multi-step
Available Models	All models	RNN/LSTM/GRU only	Most models (default)
Training Target	Predict t+1	Predict t+1, iterate	Predict t+1 to t+h

---

Key Parameter: `h`

The `h` parameter is **always set during model initialization**, not during prediction:

```
# h determines the forecast horizon during training
model = NHITS(
    h=12,            # Forecast 12 steps ahead
    input_size=24,   # Use 24 past observations
    max_steps=100    # Training iterations
)
```

- **h=1**: One-step ahead forecasting
- **h>1**: Multi-step forecasting (recursive or direct depending on model)

How to Choose?

Use One-Step Ahead (h=1) when:

- You only need the immediate next value
- You can update your data and retrain frequently
- Maximum accuracy for the next step is critical

Use Recursive (h>1 with recurrent=True) when:

- Using RNN/LSTM/GRU models
- Computational efficiency is important
- You can tolerate some error accumulation
- Hardware resources are limited

Use Direct (h>1, default) when:

- You need to forecast multiple steps ahead
- Maximum accuracy is important
- You have adequate computational resources
- Using models like NBEATS, NHITS, TFT, etc.

Important Notes

1. **Default Behavior**: Most NeuralForecast models use **direct multi-output forecasting** by default
2. **Recursive Option**: Only available for RNN/LSTM/GRU models via the `recurrent` parameter
3. **h Parameter**: Must be set during model initialization, not during prediction
4. **predict() Method**: Does not take a test set as argument - it forecasts h steps from the end of training data

---

NeuralForecast vs TensorFlow/Keras: Key Differences

⚠ NeuralForecast is NOT like TensorFlow

Many users expect NeuralForecast to work like TensorFlow/Keras, but it's **fundamentally different**. Understanding this distinction is crucial.

TensorFlow/Keras Approach (General ML)

```
# TensorFlow/Keras - Standard ML workflow
from tensorflow.keras.models import Sequential

model = Sequential([...])
```

```
model.fit(X_train, y_train)
predictions = model.predict(X_test)  # ✓ This works!
```

Characteristics:

- `X_test` is your test input data
- Model applies learned function directly to X_test
- Returns predictions for arbitrary inputs
- No temporal ordering assumed
- Test data is independent from training data

NeuralForecast Approach (Time Series Specific)

```
# NeuralForecast - Time series specific workflow
from neuralforecast import NeuralForecast
from neuralforecast.models import NHITS

model = NHITS(h=12, input_size=24)
nf = NeuralForecast(models=[model], freq='M')
nf.fit(df=train)
predictions = nf.predict()  # ✗ NO X_test argument!
```

Characteristics:

- **`predict()` takes NO test set argument**
- Automatically uses the last `input_size` values from training data
- Forecasts `h` steps into the future from where training ended
- Designed for temporal sequences
- Predictions are always forward in time

What Actually Happens?

```
# ✗ What you might expect (like TensorFlow):
predictions = nf.predict(X_test)  # ERROR! This will fail

# ✓ What actually works in NeuralForecast:
predictions = nf.predict()  # Forecasts from end of training data

# ✓ Or with future exogenous variables (not target values):
predictions = nf.predict(futr_df=test)  # Only for exogenous features
```

Visual Comparison

```
┌──────────────────────────────────────────────────────────────┐
│ TensorFlow/Keras (General Purpose)                           │
├──────────────────────────────────────────────────────────────┤
```

```
│ Training: [X1, X2, X3] → [y1, y2, y3]                  │
│ Testing:  [X_new] → model.predict(X_new) → prediction  │
│                                                         │
│ • Can predict on ANY arbitrary input                   │
│ • No temporal relationship required                    │
└─────────────────────────────────────────────────────────┘

┌─────────────────────────────────────────────────────────┐
│ NeuralForecast (Time Series Specific)                  │
├─────────────────────────────────────────────────────────┤
│ Training: [..., t-2, t-1, t]                           │
│ Predict:  Uses last window [t-23, ..., t] automatically │
│           ↓                                             │
│           Forecasts [t+1, t+2, ..., t+12]              │
│                                                         │
│ • Always forecasts FORWARD from training data end      │
│ • Uses last window automatically                       │
└─────────────────────────────────────────────────────────┘
```

Why the Difference?

NeuralForecast Philosophy:

- Built specifically for **time series forecasting**
- Assumes temporal ordering and continuity
- Forecasting is about predicting **future** values from **past** context
- The "test set" concept doesn't apply in the same way

TensorFlow/Keras Philosophy:

- General-purpose machine learning framework
- No assumptions about data structure
- Can handle any input-output mapping
- Test data can be completely independent

How to Work with Historical Test Data

If you want to evaluate on historical data (like a test set), use these methods:

Option 1: Cross Validation

```
# Evaluate with rolling windows on historical data
cv_results = nf.cross_validation(
    df=pd.concat([train, test]),
    n_windows=10,      # Number of validation windows
    step_size=12       # Steps to move forward each window
)
```

Option 2: In-Sample Predictions

```
# Get predictions for the training period
insample_preds = nf.predict_insample(step_size=12)
```

Option 3: Recursive Manual Approach

```
# Manually iterate through test set
predictions = []
current_data = train.copy()

for i in range(0, len(test), h):
    nf.fit(df=current_data)
    pred = nf.predict()
    predictions.append(pred)

    # Add next actual values to training
    current_data = pd.concat([current_data, test.iloc[i:i+h]])
```

Common Confusion: The `futr_df` Parameter

```
# This does NOT mean "predict on test data" like TensorFlow
predictions = nf.predict(futr_df=test)
```

What `futr_df` actually does:

- Provides **future exogenous variables** (covariates)
- NOT the target values you want to predict
- Used when your model needs future information (e.g., weather forecasts, calendar features)

Example:

```
# If your model uses future temperature forecasts
test_with_exog = test[['unique_id', 'ds', 'temperature_forecast']]
predictions = nf.predict(futr_df=test_with_exog)
# Still forecasts from end of training, but uses future temperature info
```

Quick Reference: When Coming from TensorFlow

What you want	TensorFlow/Keras	NeuralForecast
Train model	`model.fit(X_train, y_train)`	`nf.fit(df=train)`
Predict on test data	`model.predict(X_test)`	❌ Not applicable
Forecast future	N/A	`nf.predict()` ✓

What you want	TensorFlow/Keras	NeuralForecast
Evaluate on historical data	`model.predict(X_test)`	`nf.cross_validation()` ✓
Use future features	Include in X_test	`nf.predict(futr_df=...)` ✓

Bottom Line

NeuralForecast is NOT like TensorFlow. It's a specialized time series library with different prediction mechanics designed specifically for temporal forecasting workflows. The prediction always happens from the end of your training data, moving forward in time.

Additional Resources

- NeuralForecast Documentation
- NeuralForecast GitHub
- Model Overview

Example: Complete Workflow

```
from neuralforecast import NeuralForecast
from neuralforecast.models import NHITS, LSTM
import pandas as pd

# Prepare your data
# df must have columns: ['unique_id', 'ds', 'y']

# Option 1: Direct forecasting (default for most models)
direct_model = NHITS(
    h=12,           # Forecast 12 steps ahead
    input_size=24,  # Use 24 past observations
    max_steps=100
)

# Option 2: Recursive forecasting (RNN models only)
recursive_model = LSTM(
    h=12,
    input_size=24,
    recurrent=True,  # Enable recursive forecasting
    max_steps=100
)

# Initialize NeuralForecast
nf = NeuralForecast(
    models=[direct_model, recursive_model],
    freq='M'  # Monthly frequency
)

# Fit and predict
```

```
nf.fit(df=train_df)
forecasts = nf.predict()  # Returns 12 steps for both models

# For rolling window forecasts on test set
cv_results = nf.cross_validation(
    df=pd.concat([train_df, test_df]),
    n_windows=len(test_df) // 12,
    step_size=12
)
```

Last updated: 2025