

Time Series Forecasting with Machine Learning

Isaac Flower

23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING

UNIVERSITY OF
BATH

About me

- 2nd Year PhD Student at the University of Bath
- AAPPS CDT
- Electric Vehicle charging modelling



23/02/2023



Isaac Flower

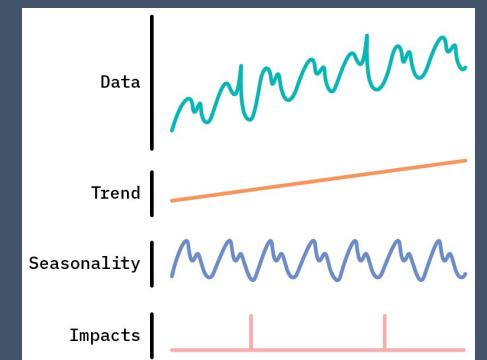


Overview

Machine Learning Tools



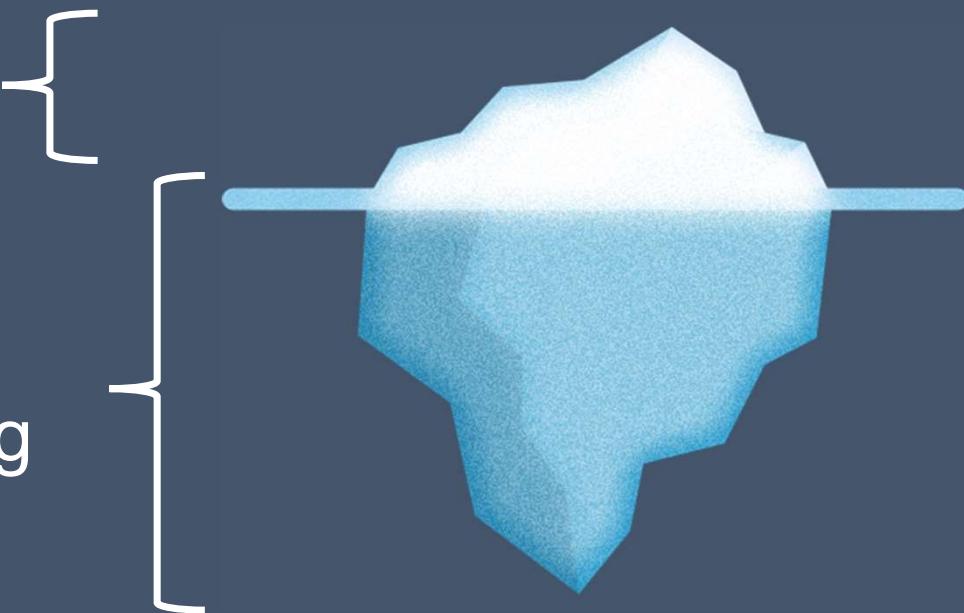
Forecasting Methods



Overview

This workshop

Machine Learning and
Time Series Forecasting



Overview

1. Introduction to Time Series Forecasting

2. Introduction to:

- Python 
- TensorFlow 
- The Data

3. Neural Network Models

4. Structural Time Series Models

1. Introduction to Time Series Forecasting

23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING



UNIVERSITY OF
BATH

What is Time Series Forecasting?

- Predict future values based on past observations.



Forecast Horizon?

Time Series Forecasting in Energy Systems

- We can use time series forecasting to predict **electrical load** and **PV generation** in mini-grids.
- Plan for peak demand
- Optimize system design (e.g. battery storage)
- Ensure reliable power supply
- Better renewable integration

Time Series Forecasting Techniques

- Time series forecasting techniques can be broadly divided into two categories:
 1. **Statistical models**: simple patterns and trends.
 2. **Machine learning models**: complex relationships.

Statistical Models for Time Series Forecasting

- Autoregressive Integrated Moving Average (ARIMA)
- Seasonal Autoregressive Integrated Moving Average (SARIMA)
- Exponential Smoothing (ETS)
- **Structural Time Series (STS)**

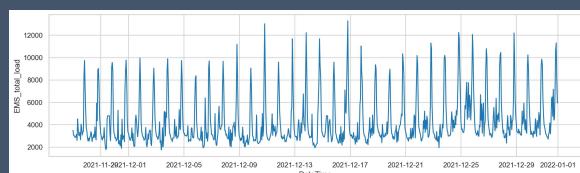
Machine Learning Models for Time Series Forecasting

- Random Forest (RF)
- Gradient Boosting (GB)
- **Neural Networks (NN)**

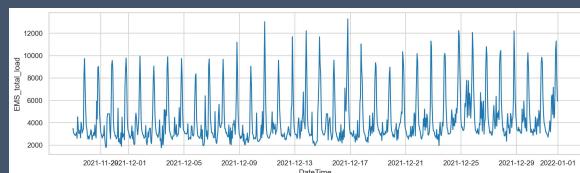
What do we want to Forecast?

Historical Data

Electrical Load



PV Generation



How can we learn
from the data?

Forecasts

Next Hour

Next 24 hours

Next Week

Model

2. Introduction to Python, TensorFlow and the Data

23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING

UNIVERSITY OF
BATH

Python

- Python is a **high-level** programming language.
- Widely used for scientific computing, data analysis, machine learning and web development.
- **Simple** and easy to learn
- **Cross-platform**: can run on Windows, Mac and Linux
- **Open Source**

Installing Python with Anaconda



- Anaconda is a popular Python distribution for scientific computing.
- Includes Python, conda package manager, and many pre-installed packages.
- Download and install Anaconda from the official website.
- Follow the installation instructions for your operating system.

Creating Conda Environments

- Conda environments allow you to create isolated environments with **specific package versions**.
- Useful for **managing dependencies** and avoiding conflicts between packages.
- Create a new conda environment with the command: `conda create --name env_name`.
- Activate the environment with: `conda activate env_name`
- Deactivate the environment with: `conda deactivate`

Installing Packages (with Conda)

- Conda package manager allows you to easily install packages in your conda environment.
- Install a package with the command: `conda install package_name`.
- Specify a specific version of the package with: `conda install package_name=version`.
- Update a package with: `conda update package_name`.
- Remove a package with: `conda remove package_name`.

Installing Packages (with pip)

- pip is another popular package manager for Python.
- Can be used to install packages not available in conda.
- Install a package with the command: `pip install package_name`.
- Specify a specific version of the package with: `pip install package_name==version`.
- Update a package with: `pip install --upgrade package_name`.
- Remove a package with: `pip uninstall package_name`.

TensorFlow

- TensorFlow is a popular open-source library for **machine learning** and **deep learning**.
- Developed by Google, it allows developers to build and train **neural networks**.
- Widely used for image recognition, natural language processing, and time series forecasting.

TensorFlow Probability

- TensorFlow Probability is an extension of TensorFlow for **probabilistic programming**.
- Allows developers to build and train models that incorporate **uncertainty**.
- Widely used for Bayesian modelling, time series analysis, and generative models

Jupyter Notebooks



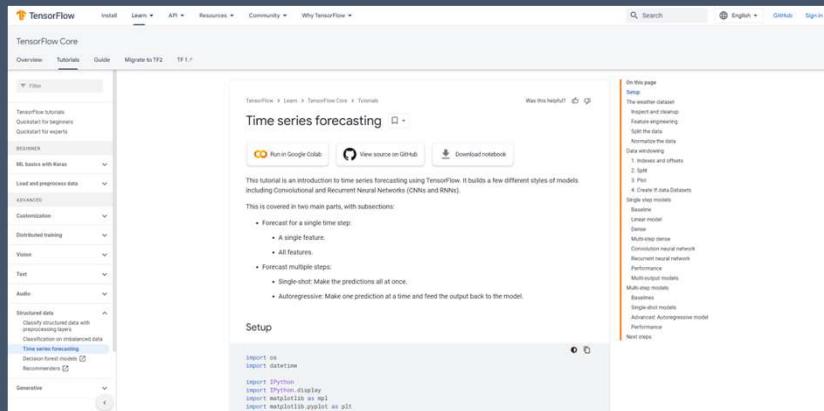
- Jupyter Notebooks are an **interactive computing environment** that allows developers to create and share code.
- Provides a web-based interface for creating and editing notebooks.
- Allows developers to combine code, text, and visualizations in a single document.

Jupyter Notebook Text Editors

- Jupyter Notebooks can be edited using a text editor or integrated development environment (IDE).
- Some of the popular text editors for Jupyter Notebooks include:
 - JupyterLab 
 - VS Code 
 - Atom 

TensorFlow Tutorials

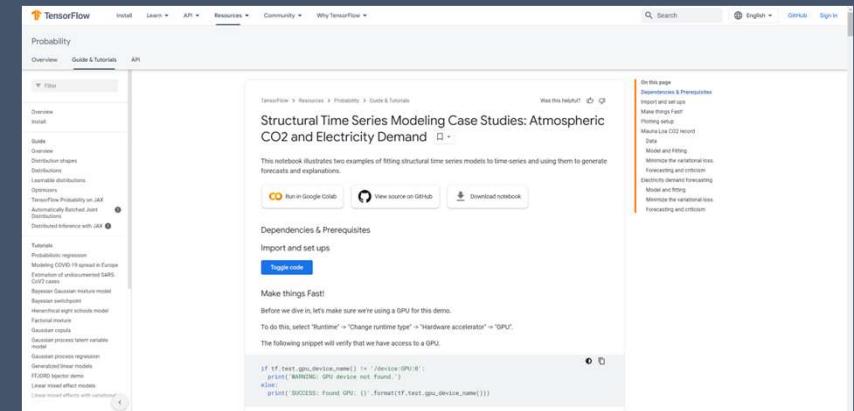
Neural Networks



This screenshot shows the TensorFlow Core tutorial page for "Time series forecasting". The main content area displays a Jupyter notebook titled "Time series forecasting". The notebook code imports TensorFlow, Pandas, and Matplotlib, and then loads a dataset from a CSV file. The sidebar on the left provides a navigation menu for TensorFlow Core, including sections like "TensorFlow Basics", "Datasets", "Advanced", and "Structured data". The right sidebar contains links to "Setup" steps such as "Import and set up", "Data", "Distributions", and "Performance".

https://www.tensorflow.org/tutorials/structured_data/time_series

Structural Time Series



This screenshot shows the TensorFlow Probability tutorial page for "Structural Time Series Modeling Case Studies: Atmospheric CO2 and Electricity Demand". The main content area displays a Jupyter notebook titled "Structural Time Series Modeling Case Studies: Atmospheric CO2 and Electricity Demand". The notebook code imports TensorFlow, Pandas, and Matplotlib, and then loads a dataset from a CSV file. The sidebar on the left provides a navigation menu for TensorFlow Probability, including sections like "Dependencies & Prerequisites", "Import and set up", "Data", "Distributions", and "Performance". The right sidebar contains links to "Setup" steps such as "Import and set up", "Data", "Distributions", and "Performance".

https://www.tensorflow.org/probability/examples/Structural_Time_Series_Modeling_Case_Studies_Atmospheric_CO2_and_Electricity_Demand

23/02/2023



Isaac Flower



3. Neural Network Models

23/02/2023



Strathmore
UNIVERSITY

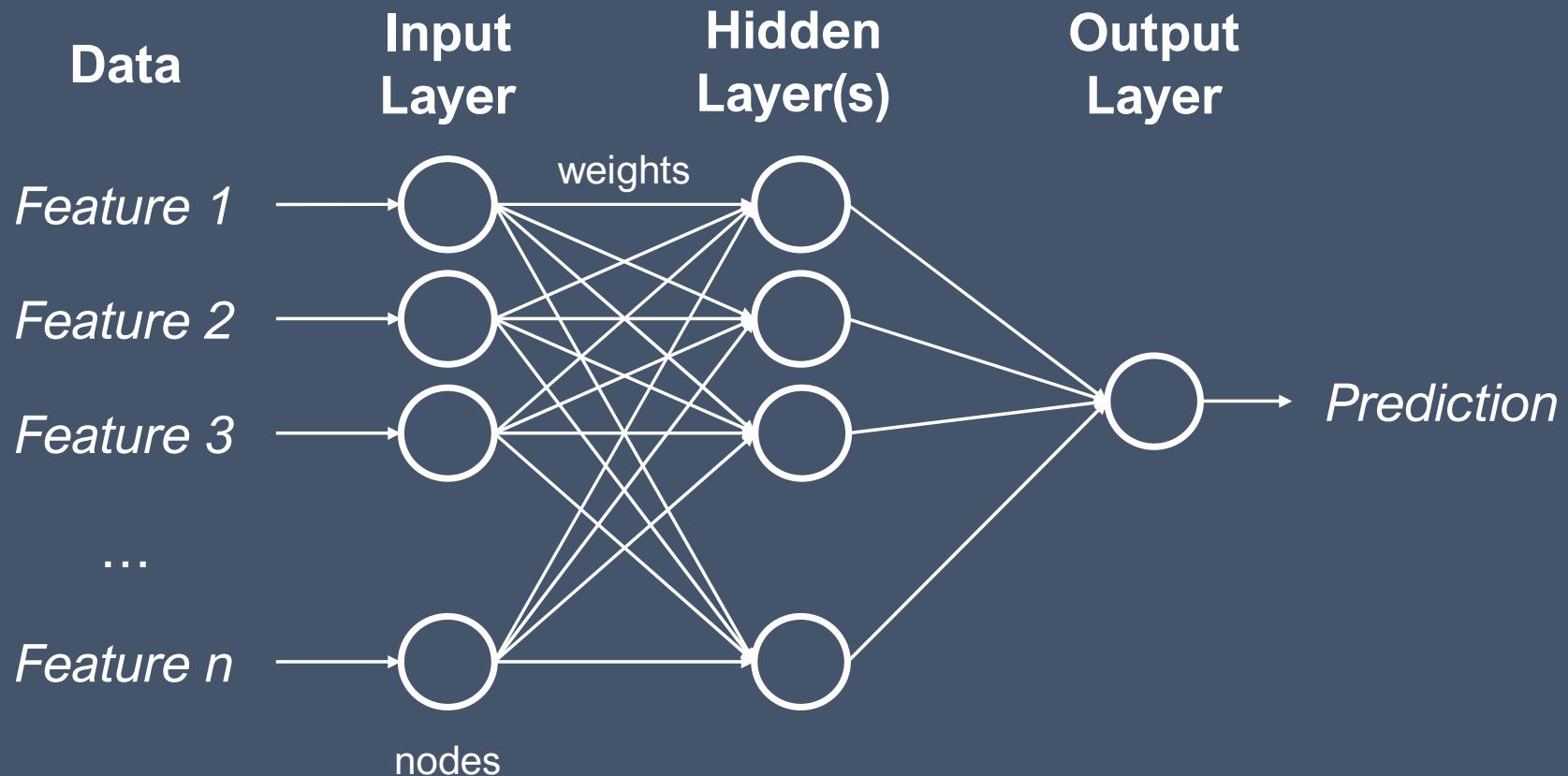
Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING



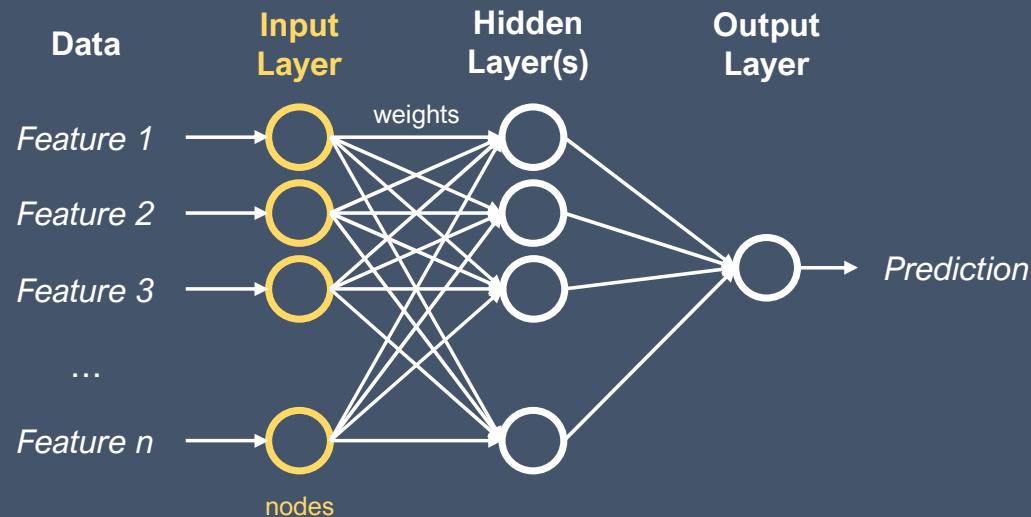
UNIVERSITY OF
BATH

What are Neural Networks?



What are Neural Networks?

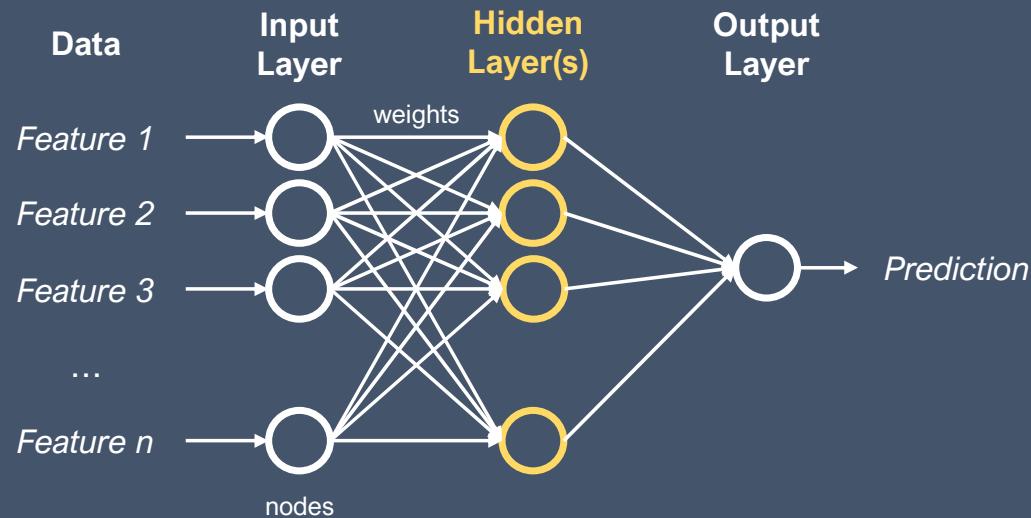
Input Layer



- The input layer is the first layer in the network and receives the input data.
- Each neuron in the input layer represents one feature of the input data.
- The input layer neurons simply pass the input data to the neurons in the first hidden layer.

What are Neural Networks?

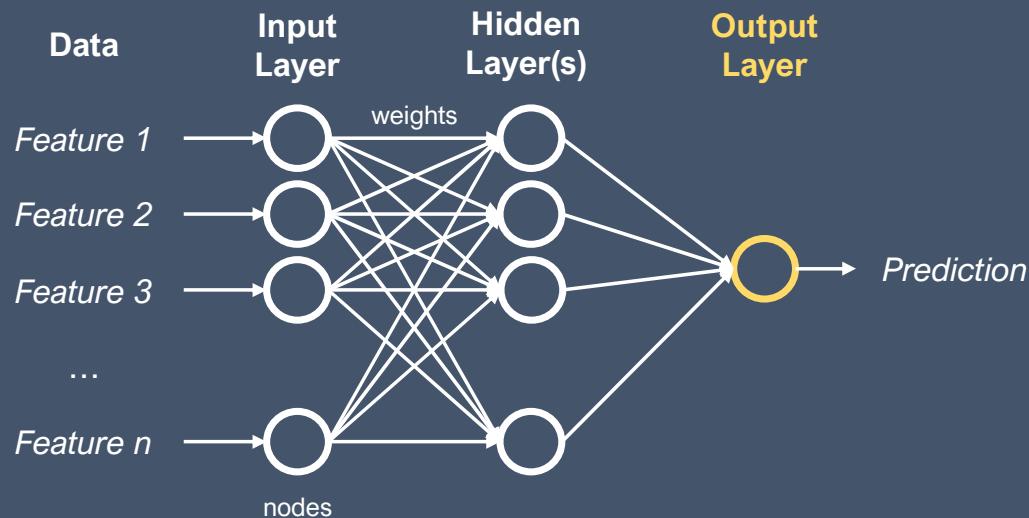
Hidden Layer(s)



- The hidden layers are located between the input and output layers and perform intermediate computations.
- Each neuron in a hidden layer receives inputs from all the neurons in the previous layer and produces an output signal.

What are Neural Networks?

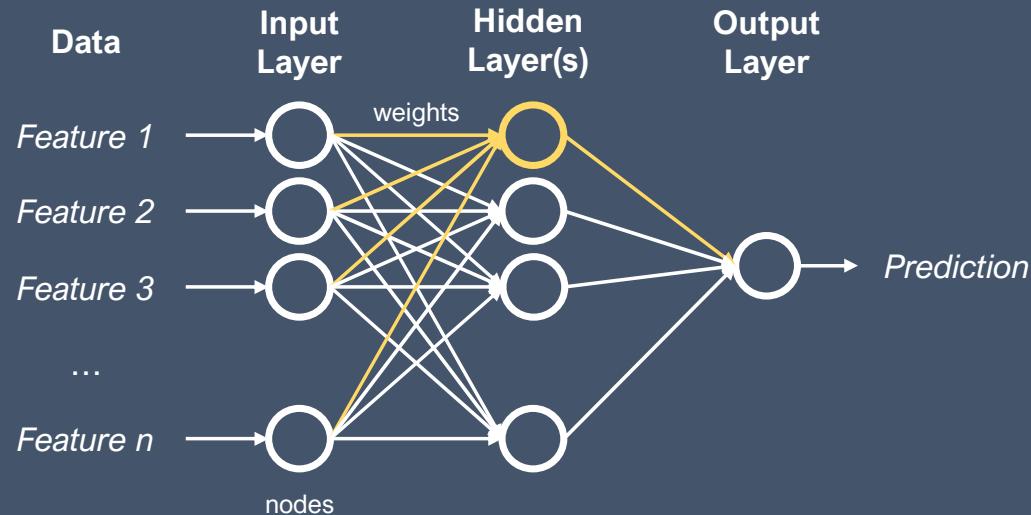
Output Layer



- The output layer is the last layer in the network and produces the network's predictions.
- The number of neurons in the output layer depends on the type of problem being solved.
- For example, in a binary classification problem, there would be one output neuron that produces a binary classification (0 or 1) for each input

What are Neural Networks?

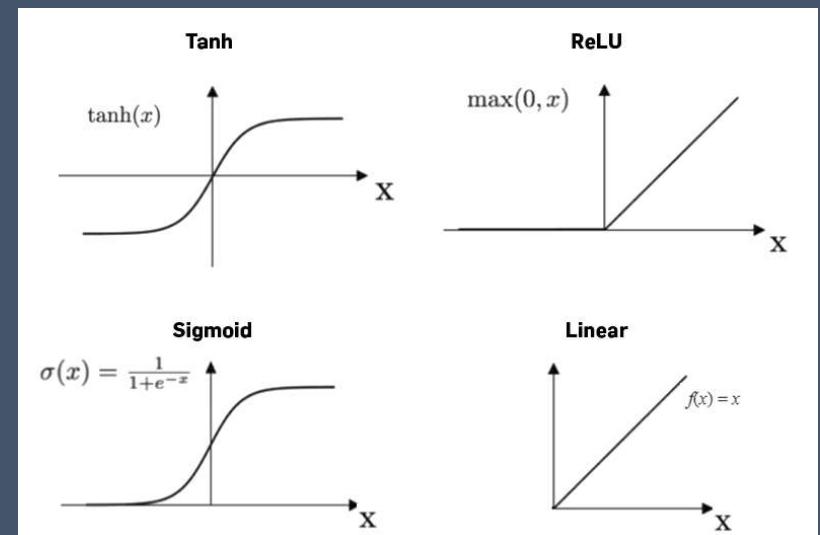
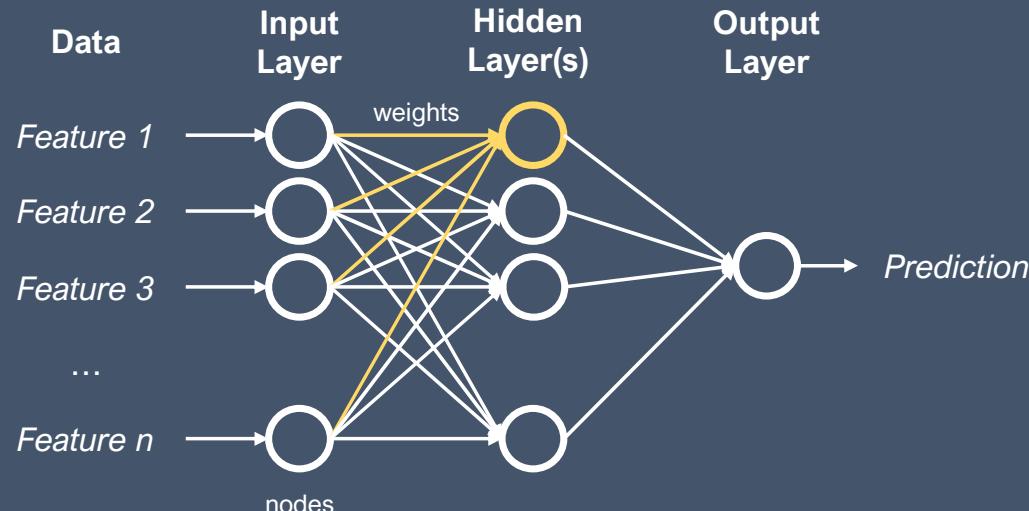
Activation Functions



- Neurons typically use an activation function to produce an output signal.
- The activation function applies a non-linear transformation to the weighted sum of the inputs to the neuron.
- Common activation functions include the sigmoid function, ReLU function, and hyperbolic tangent function.

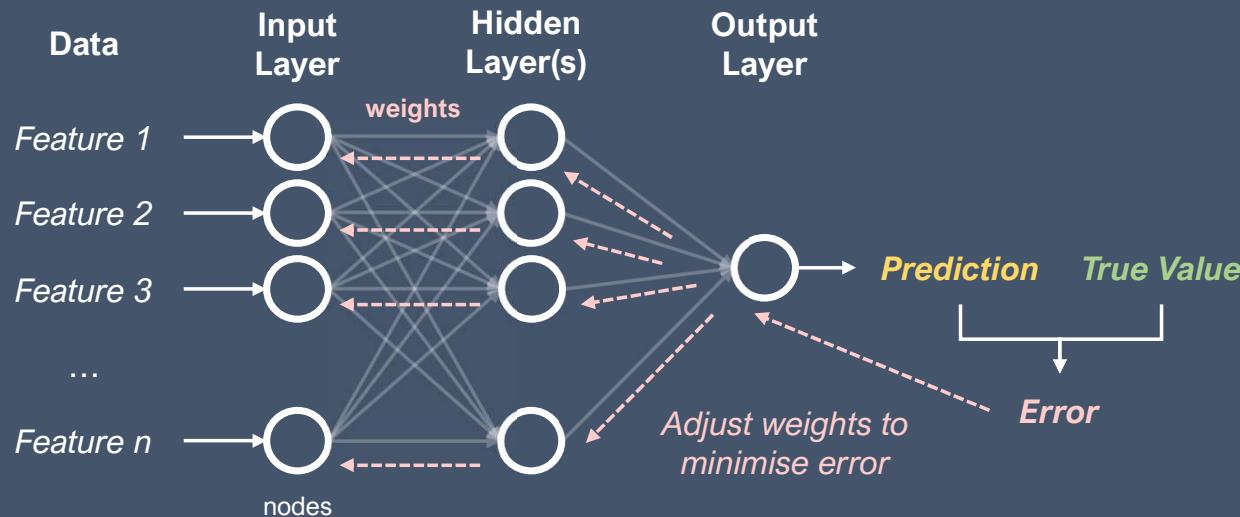
What are Neural Networks?

Activation Functions



Training Neural Networks

Backpropagation



- Neural networks are trained using **backpropagation**.
- Involves iteratively adjusting the weights of the network to minimise the error between the predicted outputs and true values.
- Backpropagation often uses optimization algorithms such as **gradient descent**

Data Pre-processing

- Before training a neural network, it is important to pre-process the data.
- Data pre-processing involves **cleaning** the data, handling **missing values**, and encoding **categorical variables**.
- Pre-processing can also include **scaling** or **normalizing** the data to ensure the neural network can learn effectively.

Data Pre-processing

- Satellite Weather Data
- EMS Data
- 1 year (2021)
- 10 minute intervals

Data Pre-processing

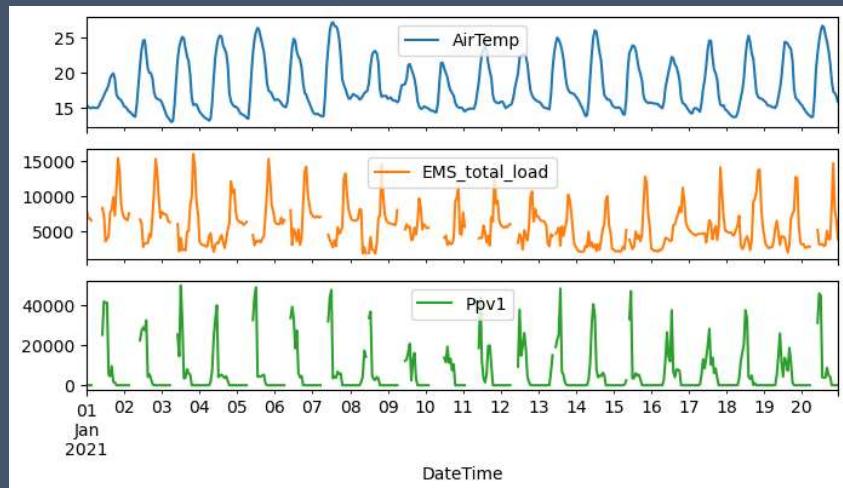
DataFrame Pandas (pd)

```
1 # Read excel file as a DataFrame and skip row index 1 (the units)
2 df = pd.read_excel('../data/Combined EMS_Solcast data.xlsx', skiprows=[1])
[75]

D ▾ 1 df.head(5) # Displays the first 5 rows of the DataFrame
[76]
...
   Date    Time  AirTemp  Azimuth  CloudOpacity  DewpointTemp  Dhi  Dni  Ebh  Ghi  ...  Fac  Preall1  Preall2  Preall3  EMS_total_load  Pbat  SoC  Batv  Batc  InvBatV
0  2021-01-01  00:00:00      15.30     157.5        81.90          13.3  0.0  0.0  0.0  0.0  ...  0.0  6400.0  1100.0    400.0      7900.0  9150.74  43.45  391.56  23.37  391.64
1  2021-01-01  00:10:00      15.25     162.5        82.25          13.3  0.0  0.0  0.0  0.0  ...  0.0  6280.0  1030.0    400.0      7710.0  8503.12  42.72  391.49  21.72  391.55
2  2021-01-01  00:20:00      15.20     167.5        82.60          13.3  0.0  0.0  0.0  0.0  ...  0.0  6160.0  950.0    400.0      7510.0  8153.45  42.11  391.43  20.83  391.52
3  2021-01-01  00:30:00      15.20     173.0        82.15          13.2  0.0  0.0  0.0  0.0  ...  0.0  6070.0  860.0    400.0      7330.0  8406.17  41.22  391.35  21.48  391.42
4  2021-01-01  00:40:00      15.10     178.5        81.85          13.2  0.0  0.0  0.0  0.0  ...  0.0  5870.0  810.0    400.0      7080.0  7922.02  40.26  391.21  20.25  391.32
5 rows × 35 columns
```



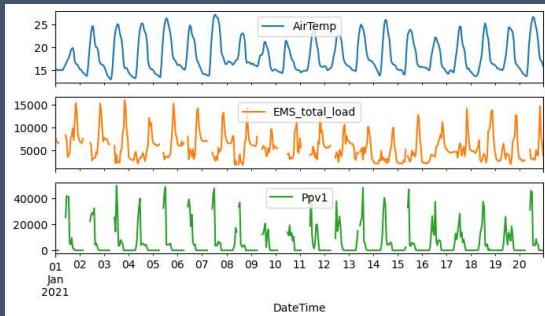
Missing Data



- Remove it (*Not desirable with time series as leads to discontinuity*)
- Replace with mean/median
- Regression imputation
- K-nearest neighbour imputation
- Multivariate imputation by chained equation (MICE)

Missing Data

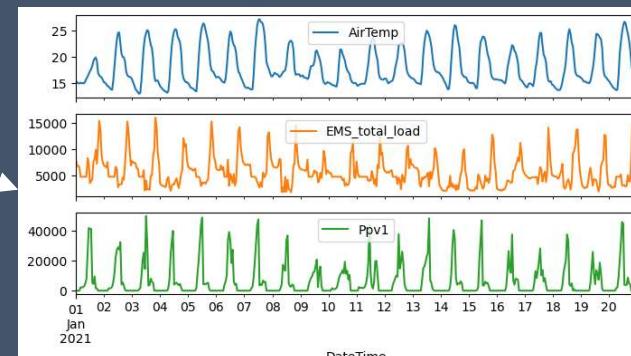
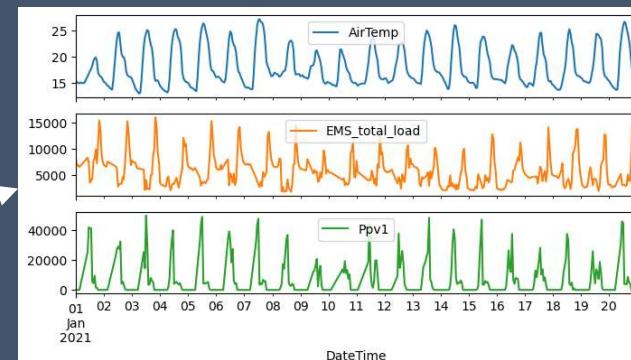
```
linear_interpolation = df[df.columns.drop(['Date', 'Time', 'DateTime'])].copy().interpolate(method='linear')
```



Linear Interpolation

Multivariate imputation
by chained equation
(MICE)

```
1 # Imputing with MICE using sklearn
2 from sklearn.experimental import enable_iterative_imputer
3 from sklearn.impute import IterativeImputer
4 from sklearn import linear_model
5
6 # Define MICE Imputer and fill missing values
7 mice_imputer = IterativeImputer(estimator=linear_model.BayesianRidge(), n_nearest_features=None, imputation_order='ascending')
8
9 df_mice = pd.DataFrame(mice_imputer.fit_transform(df[df.columns.drop(['Date', 'Time', 'DateTime'])].copy()),
10                         columns = df[df.columns.drop(['Date', 'Time', 'DateTime'])].copy().columns)
```



23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING

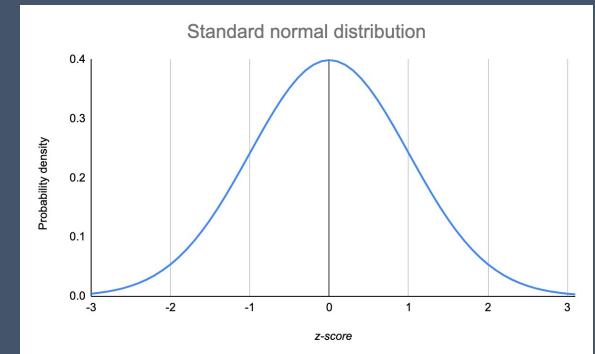


UNIVERSITY OF
BATH

Data Standardisation

- Transforms input data to have a mean of zero and a standard deviation of one.
- Ensures input features are on the same scale, which accelerates convergence.
- Improves accuracy and stability by reducing the impact of outliers in the input data.

$$Z = \frac{x - \mu}{\sigma}$$



```
1 train_mean = train_df.mean()  
2 train_std = train_df.std()  
3  
4 train_df = (train_df - train_mean) / train_std  
5 val_df = (val_df - train_mean) / train_std  
6 test_df = (test_df - train_mean) / train_std
```

Test-Validation-Train Split

- **Training data:** train the neural network.
- **Validation data:** tune hyperparameters and prevent overfitting.
- **Test data:** evaluate the performance of the neural network on unseen data

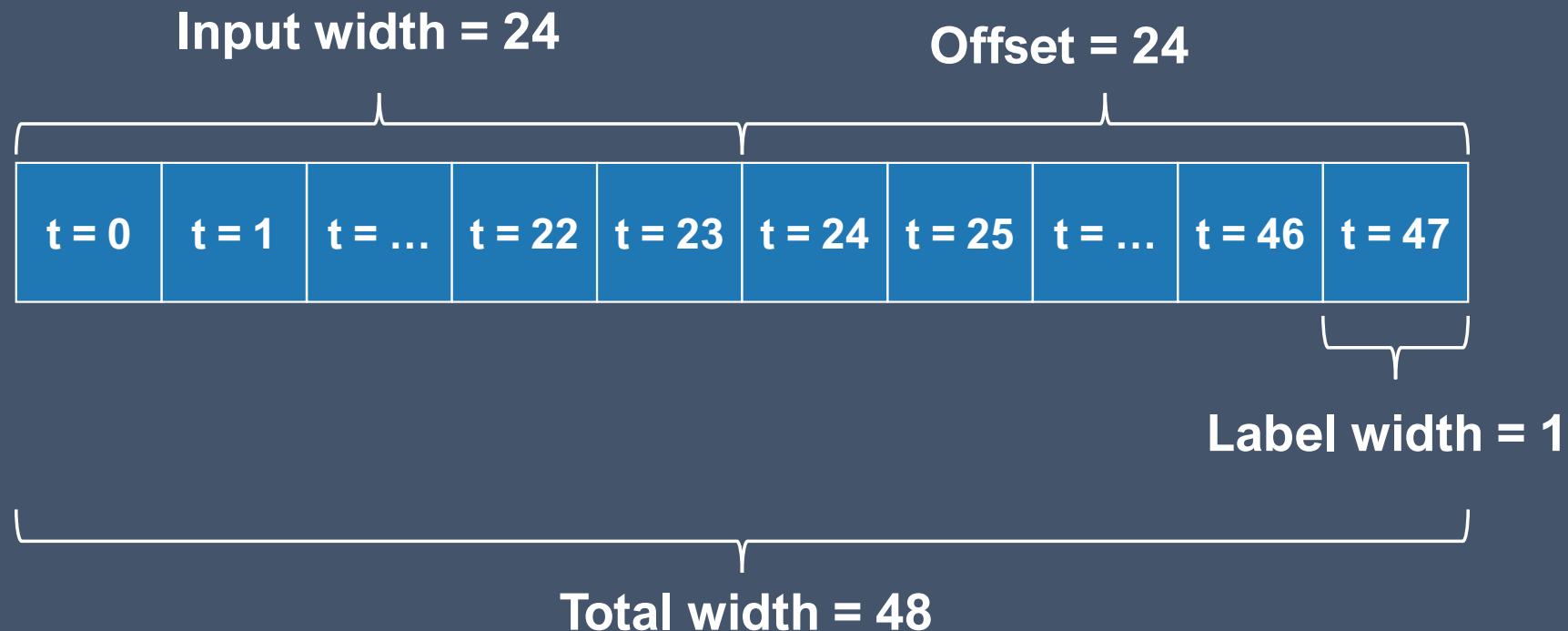


80%

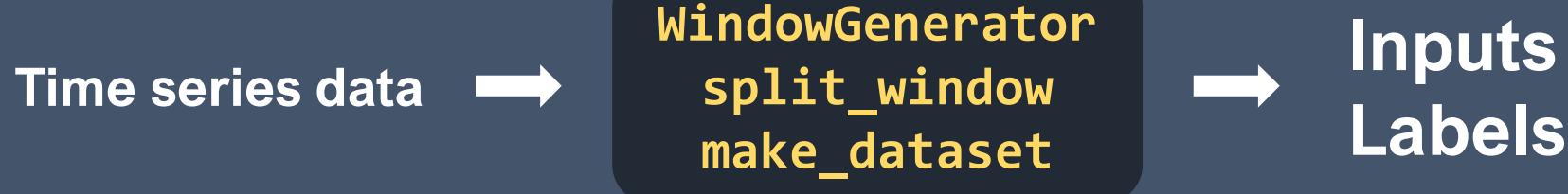
10% 10%

```
1 # Create dictionary containing the feature and the respective index
2 column_indices = {name: i for i, name in enumerate(df.columns)}
3
4 n = len(df)
5 train_df = df[0:int(n*0.8)] # Take first 80% of data
6 val_df = df[int(n*0.8):int(n*0.9)] # Take next 10% of data
7 test_df = df[int(n*0.9):] # Take last 10% of Data
8
9 num_features = df.shape[1] # Number of features (number of columns in df)
```

Data Windowing



Data Windowing



Feature Selection

- Selecting a **subset of input variables** to include in the neural network.
- **Correlation analysis or feature importance ranking.**
- Feature selection can **improve the performance** of the neural network by **reducing overfitting and increasing interpretability**

Feature Selection

Electrical Load

- PrealL1
- PrealL2
- PrealL3
- EMS_total_load
- Day sin
- Day cos
- Week sin
- Week cos
- Year sin
- Year cos

Total Electrical Load

Sinusoidal time
signals at daily, weekly
and yearly frequency

PV Generation

- AirTemp
- CloudOpacity
- Dhi
- Dni
- Ebh
- Ghi
- GtiFixedTilt
- GtiTracking
- AlbedoDaily

PV Power

- Ppv1
- Day sin
- Day cos
- Week sin
- Week cos
- Year sin
- Year cos



Training Function

Loss function is
mean squared
error (MSE)

```
MAX_EPOCHS = 40

def compile_and_fit(model, window, patience=4):
    early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                       patience=patience,
                                                       mode='min')

    model.compile(loss=tf.keras.losses.MeanSquaredError(),
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=[tf.keras.metrics.MeanAbsoluteError()])

    history = model.fit(window.train, epochs=MAX_EPOCHS,
                         validation_data=window.val,
                         callbacks=[early_stopping])

    return history
```

Stops training if
validation loss
doesn't decrease
after 4 epochs

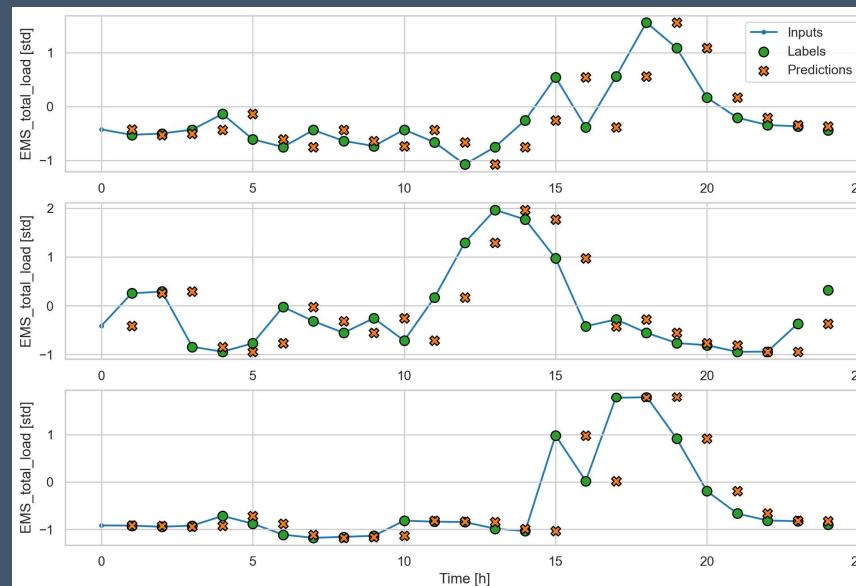


Single-step Output Models

Given some history,
Can we predict the value at the next hour?

Single-step Output Models

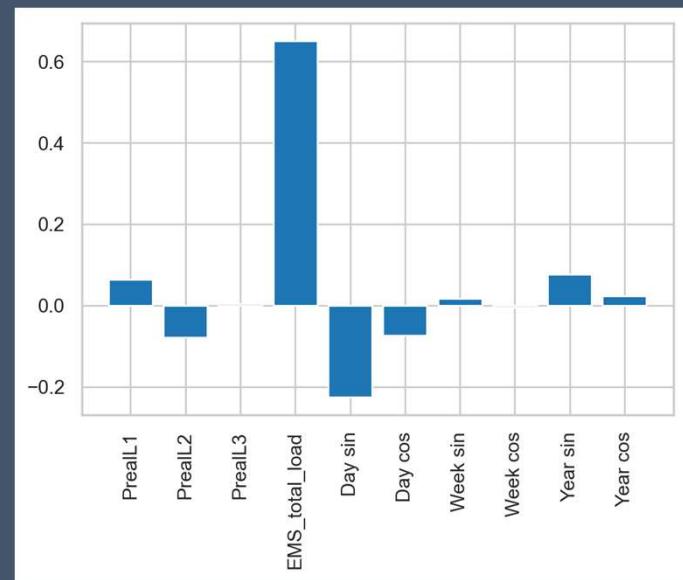
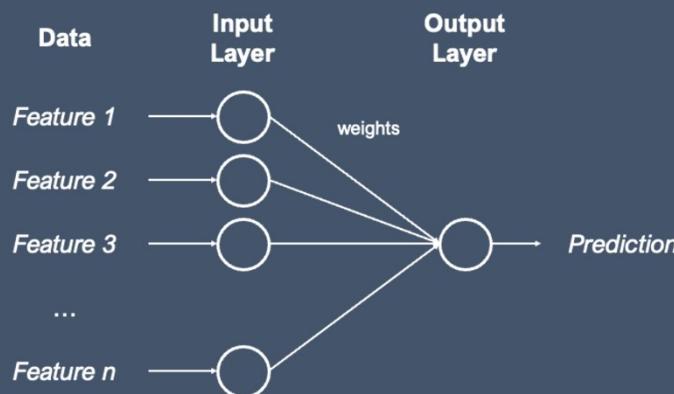
Baseline



Single-step Output Models

Linear

```
linear = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1)
])
```



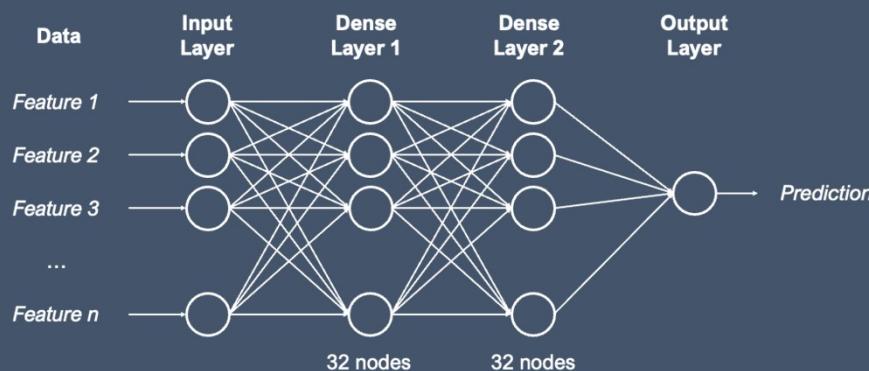
Single-step Output Models

Dense

```
dense = tf.keras.Sequential([
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1)
])

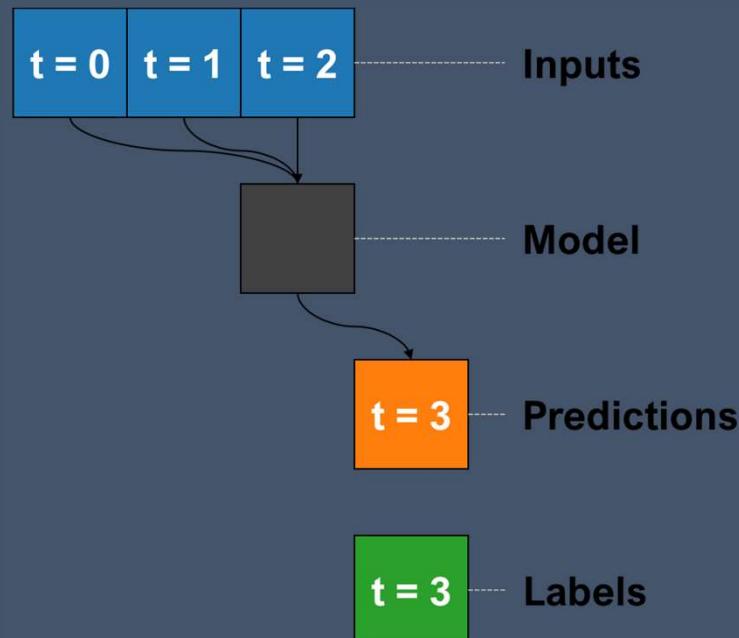
history = compile_and_fit(dense, single_step_window)

val_performance['Dense'] = dense.evaluate(single_step_window.val)
performance['Dense'] = dense.evaluate(single_step_window.test, verbose=0)
```



Single-step Output Models

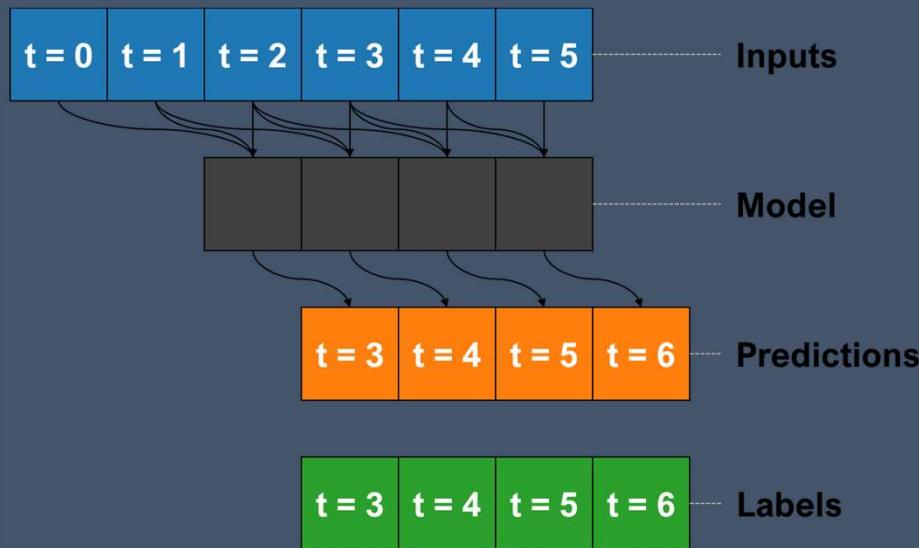
Multi-input Dense



```
multi_step_dense = tf.keras.Sequential([
    # Shape: (time, features) => (time*features)
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
    # Add back the time dimension.
    # Shape: (outputs) => (1, outputs)
    tf.keras.layers.Reshape([1, -1]),
])
```

Single-step Output Models

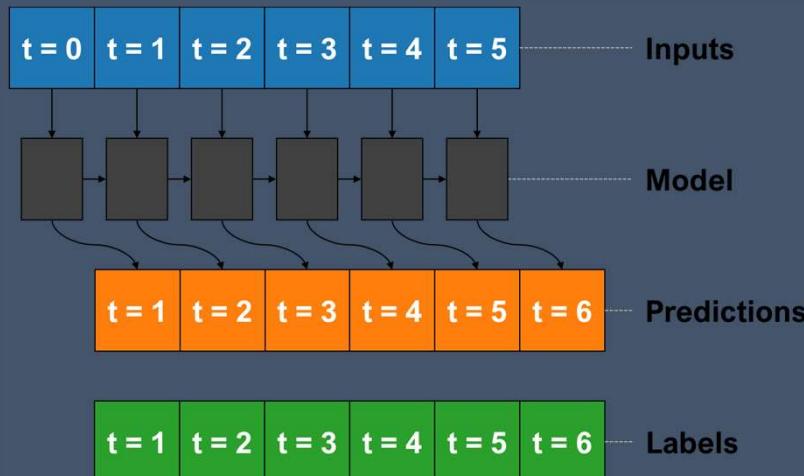
Convolutional Neural Network (CNN)



```
conv_model = tf.keras.Sequential([
    tf.keras.layers.Conv1D(filters=32,
                          kernel_size=(CONV_WIDTH,),
                          activation='relu'),
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1),
])
```

Single-step Output Models

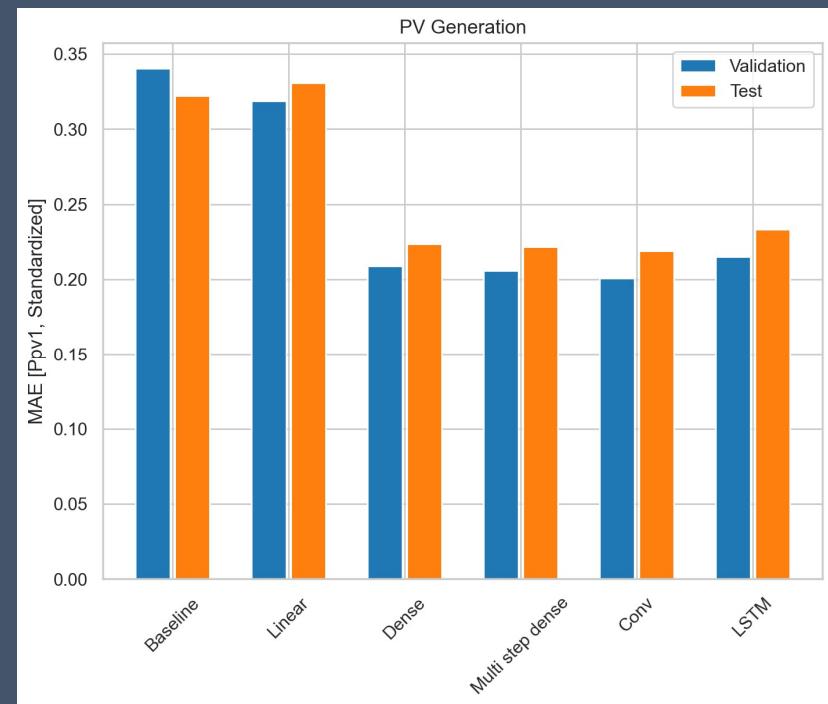
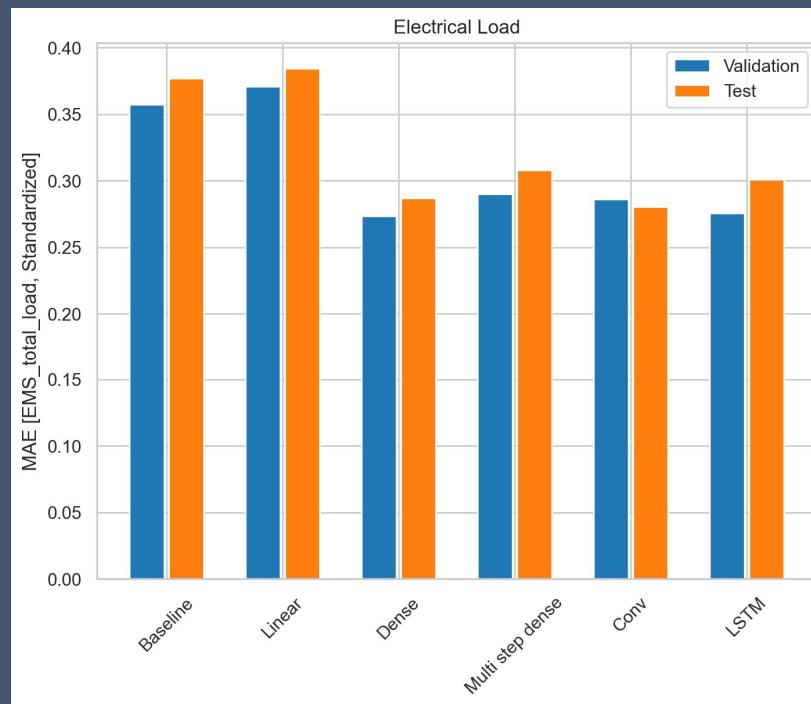
Long-Short Term Memory (LSTM)



```
lstm_model = tf.keras.models.Sequential([
    # Shape [batch, time, features] => [batch, time, lstm_units]
    tf.keras.layers.LSTM(32, return_sequences=True),
    # Shape => [batch, time, features]
    tf.keras.layers.Dense(units=1)
])
```

Single-step Output Models

Performance Comparison

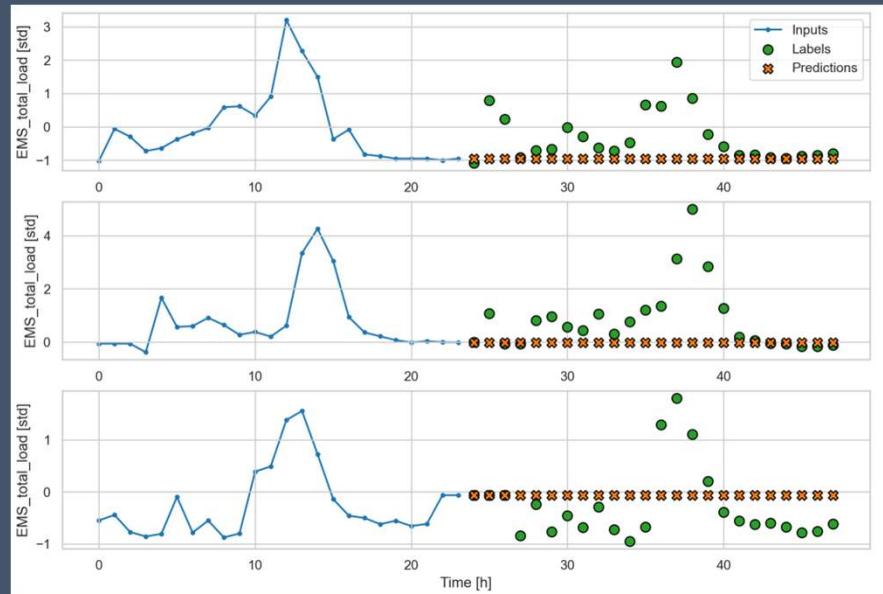


Multi-step Output Models

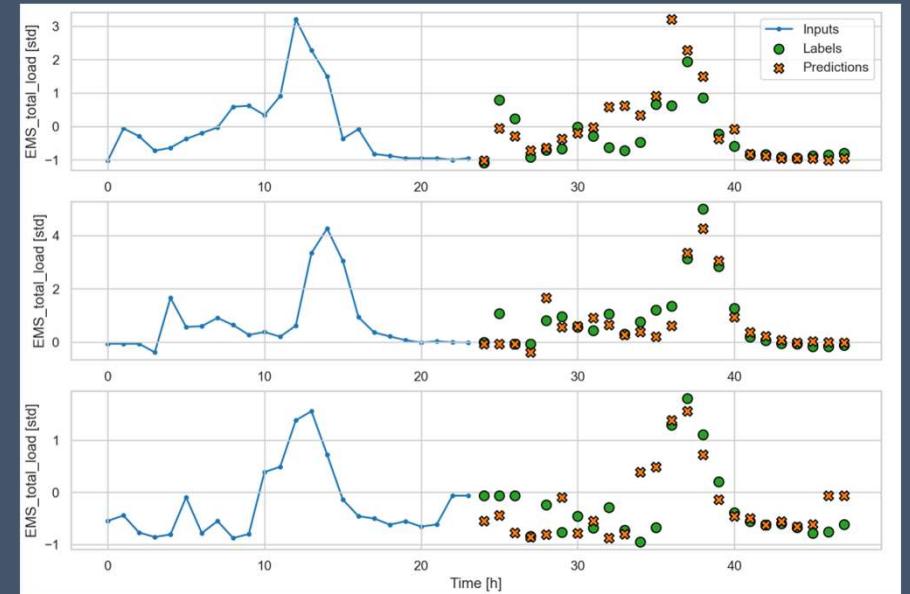
Given some history,
Can we predict the value at the next **24 hours**?

Multi-step Output Models

Baseline (Last)

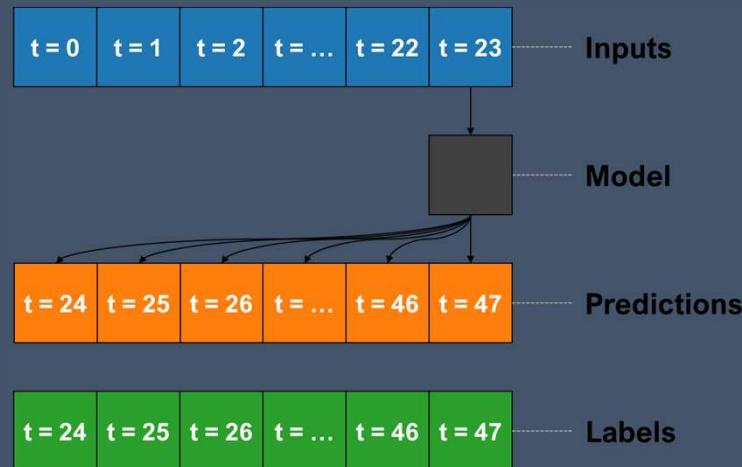


Baseline (Repeat)



Multi-step Output Models

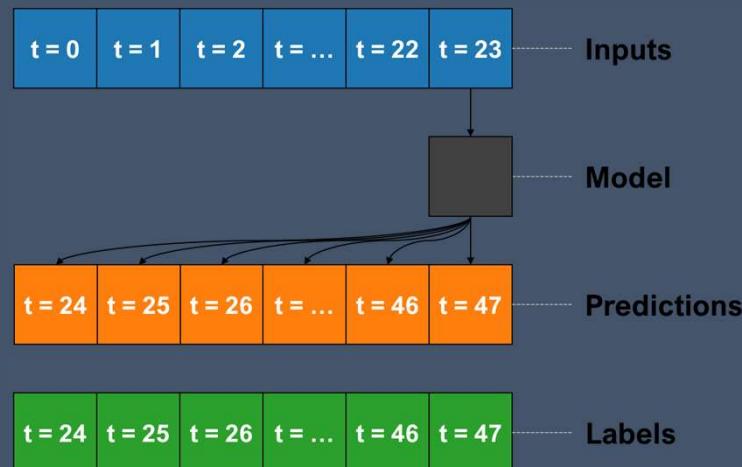
Multi-linear



```
multi_linear_model = tf.keras.Sequential([
    # Take the last time-step.
    # Shape [batch, time, features] => [batch, 1, features]
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    # Shape => [batch, 1, out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS,
        kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
    # tf.keras.layers.Reshape([OUT_STEPS, num_features])
    tf.keras.layers.Reshape([OUT_STEPS, 1])
])
```

Multi-step Output Models

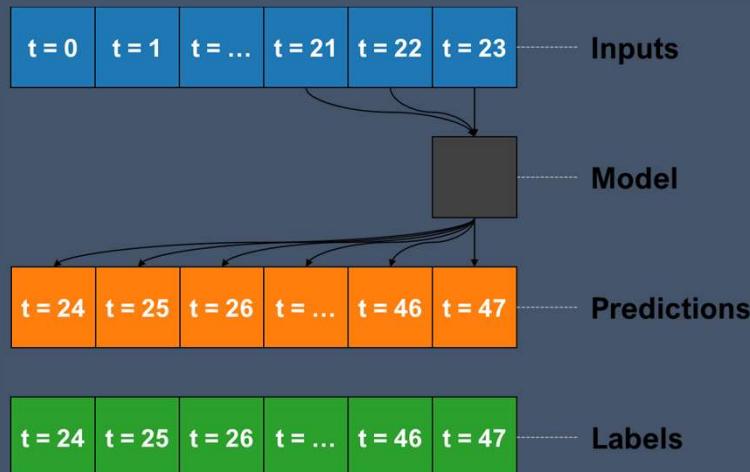
Multi-dense



```
multi_dense_model = tf.keras.Sequential([
    # Take the last time step.
    # Shape [batch, time, features] => [batch, 1, features]
    tf.keras.layers.Lambda(lambda x: x[:, -1:, :]),
    # Shape => [batch, 1, dense_units]
    tf.keras.layers.Dense(512, activation='relu'),
    # Shape => [batch, out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS,
        kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
    tf.keras.layers.Reshape([OUT_STEPS, 1])
])
```

Multi-step Output Models

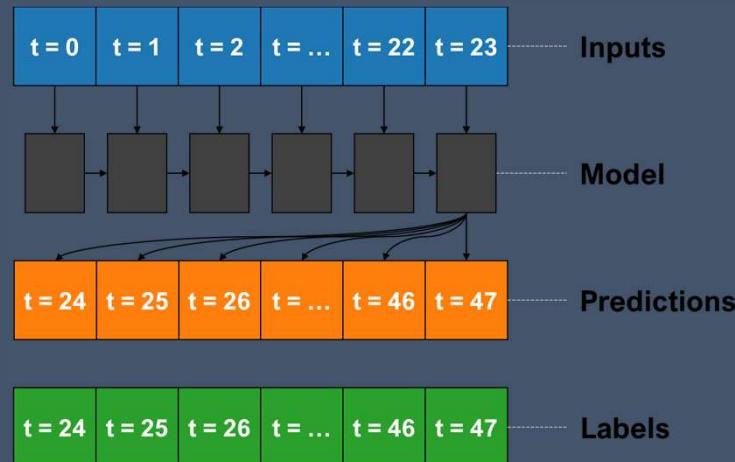
Multi-CNN



```
CONV_WIDTH = 3
multi_conv_model = tf.keras.Sequential([
    # Shape [batch, time, features] => [batch, CONV_WIDTH, features]
    tf.keras.layers.Lambda(lambda x: x[:, -CONV_WIDTH:, :]),
    # Shape => [batch, 1, conv_units]
    tf.keras.layers.Conv1D(256, activation='relu', kernel_size=(CONV_WIDTH)),
    # Shape => [batch, 1, out_steps*features]
    tf.keras.layers.Dense(OUT_STEPS,
        kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features]
    tf.keras.layers.Reshape([OUT_STEPS, 1])
])
```

Multi-step Output Models

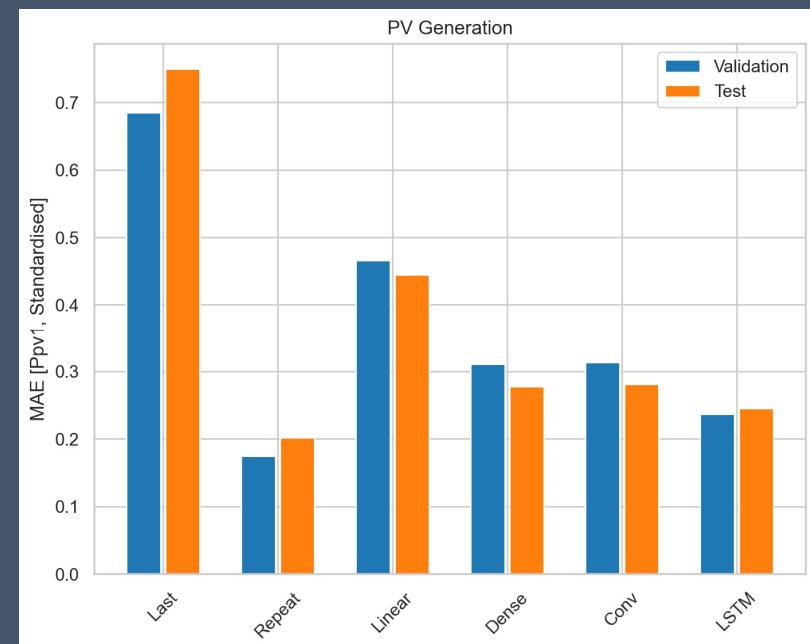
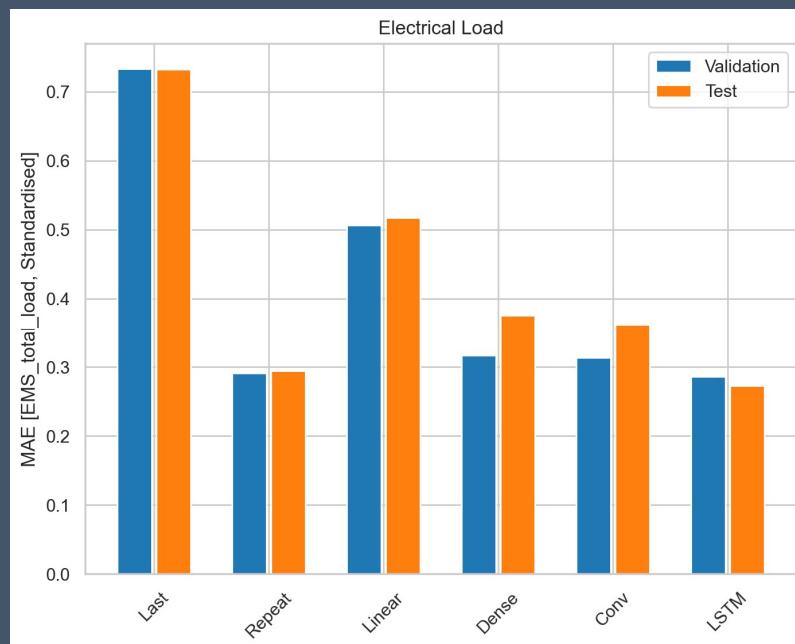
Multi-LSTM



```
multi_lstm_model = tf.keras.Sequential([
    # Shape [batch, time, features] => [batch, lstm_units].
    # Adding more `lstm_units` just overfits more quickly.
    tf.keras.layers.LSTM(32, return_sequences=False),
    # Shape => [batch, out_steps*features].
    tf.keras.layers.Dense(OUT_STEPS,
        kernel_initializer=tf.initializers.zeros()),
    # Shape => [batch, out_steps, features].
    tf.keras.layers.Reshape([OUT_STEPS, 1])
])
```

Multi-step Output Models

Performance Comparison



4. Structural Time Series Models

23/02/2023



Strathmore
UNIVERSITY

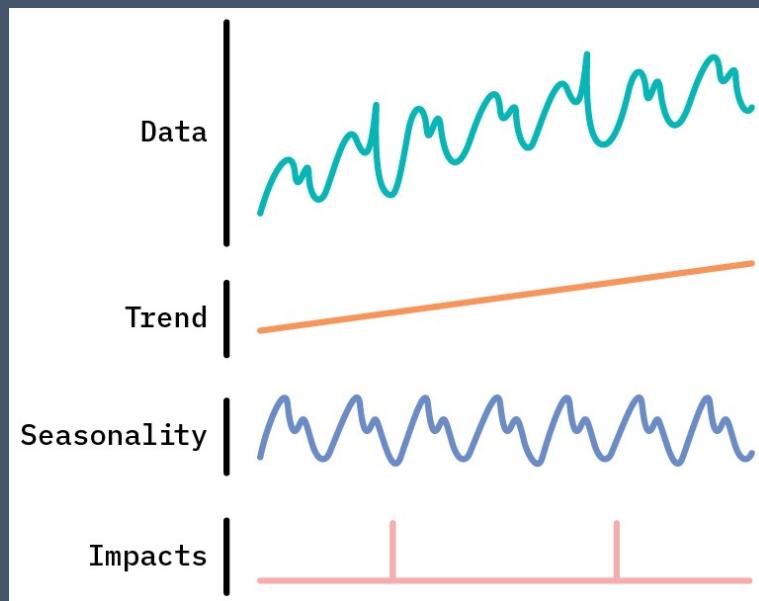
Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING

UNIVERSITY OF
BATH

What is Structural Time Series?

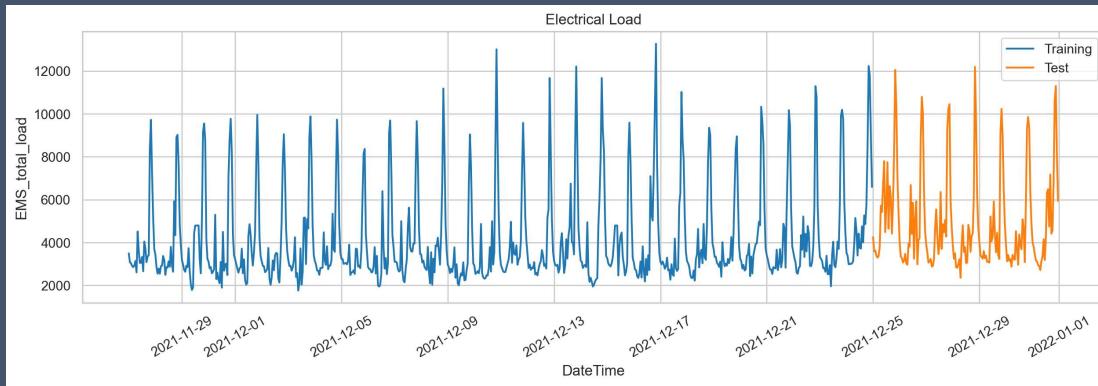
$$f(t) = f_1(t) + f_2(t) + \dots + f_n(t) + \varepsilon$$



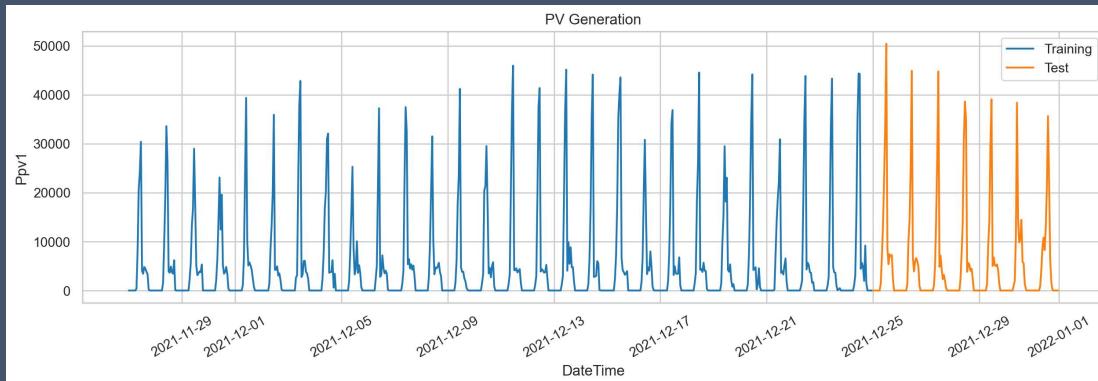
- Time series is decomposed into functions that add together to reproduce the original series.
- Don't need a lot of data
- Interpretable
- Uncertainty Quantification

Data Selection

4 weeks to train



1 week to test



24/02/2023



Strathmore
UNIVERSITY

Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING



UNIVERSITY OF
BATH

Model Components

Electrical Load

```
def build_model(observed_time_series):
    hour_of_day_effect = sts.Seasonal(
        num_seasons=24,
        observed_time_series=observed_time_series,
        name='hour_of_day_effect')
    day_of_week_effect = sts.Seasonal(
        num_seasons=7, num_steps_per_season=24,
        observed_time_series=observed_time_series,
        name='day_of_week_effect')
    temperature_effect = sts.LinearRegression(
        design_matrix=tf.reshape(temperature - np.mean(temperature),
                                 (-1, 1)), name='temperature_effect')
    autoregressive = sts.Autoregressive(
        order=1,
        observed_time_series=observed_time_series,
        name='autoregressive')
    model = sts.Sum([hour_of_day_effect,
                    day_of_week_effect,
                    temperature_effect,
                    autoregressive],
                   observed_time_series=observed_time_series)
    return model
```

PV Generation

```
def build_model(observed_time_series):
    hour_of_day_effect = sts.Seasonal(
        num_seasons=24,
        observed_time_series=observed_time_series,
        name='hour_of_day_effect')
    ghi_effect = sts.LinearRegression(
        design_matrix=tf.reshape(data.Ghi - np.mean(data.Ghi),
                                 (-1, 1)), name='ghi_effect')
    ebh_effect = sts.LinearRegression(
        design_matrix=tf.reshape(data.Ebh - np.mean(data.Ebh),
                                 (-1, 1)), name='ebh_effect')
    dni_effect = sts.LinearRegression(
        design_matrix=tf.reshape(data.Dni - np.mean(data.Dni),
                                 (-1, 1)), name='dni_effect')
    dhi_effect = sts.LinearRegression(
        design_matrix=tf.reshape(data.Dhi - np.mean(data.Dhi),
                                 (-1, 1)), name='dhi_effect')
    cloud_effect = sts.LinearRegression(
        design_matrix=tf.reshape(data.CloudOpacity - np.mean(data.CloudOpacity),
                                 (-1, 1)), name='cloud_effect')
    autoregressive = sts.Autoregressive(
        order=1,
        observed_time_series=observed_time_series,
        name='autoregressive')
    model = sts.Sum([hour_of_day_effect,
                    ghi_effect,
                    ebh_effect,
                    dni_effect,
                    dhi_effect,
                    cloud_effect,
                    autoregressive],
                   observed_time_series=observed_time_series)
    return model
```

Model Fitting

Variational Inference (VI)

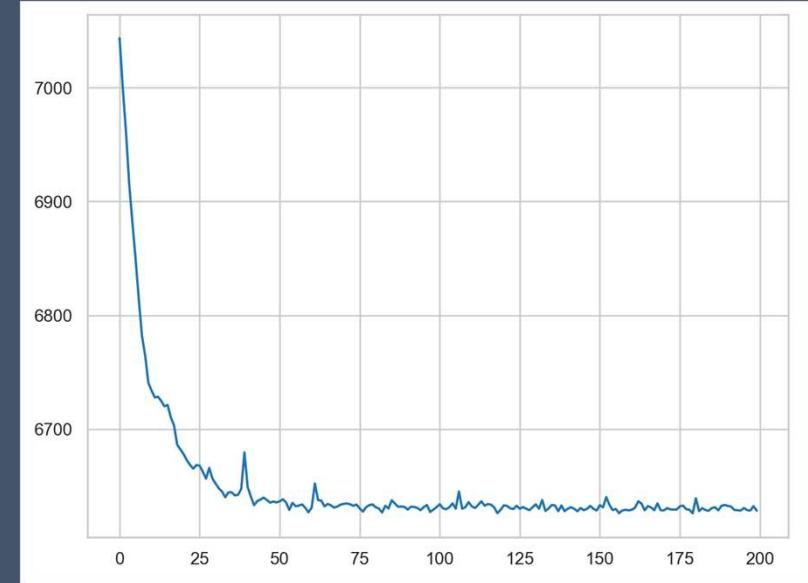
```
pv_model = build_model(training_data.Ppv1)

# Build the variational surrogate posteriors `qs`.
variational_posteriors = tfp.sts.build_factored_surrogate_posterior(model=pv_model)
```

```
# Allow external control of optimization to reduce test runtimes.
num_variational_steps = 200 # @param { isTemplate: true}
num_variational_steps = int(num_variational_steps)

# Build and optimize the variational loss function.
elbo_loss_curve = tfp.vi.fit_surrogate_posterior(
    target_log_prob_fn = pv_model.joint_distribution(
        observed_time_series = training_data.Ppv1).log_prob,
    surrogate_posterior = variational_posteriors,
    optimizer = tf.optimizers.Adam(learning_rate=0.1),
    num_steps = num_variational_steps,
    jit_compile = True)
plt.plot(elbo_loss_curve)
plt.show()

# Draw samples from the variational posterior.
q_samples_pv_ = variational_posteriors.sample(50)
```



Forecasting

Make the forecast

```
pv_forecast_dist = tfp.sts.forecast(  
    model = pv_model,  
    observed_time_series = training_data.Ppv1,  
    parameter_samples = q_samples_pv_,  
    num_steps_forecast = num_forecast_steps)
```

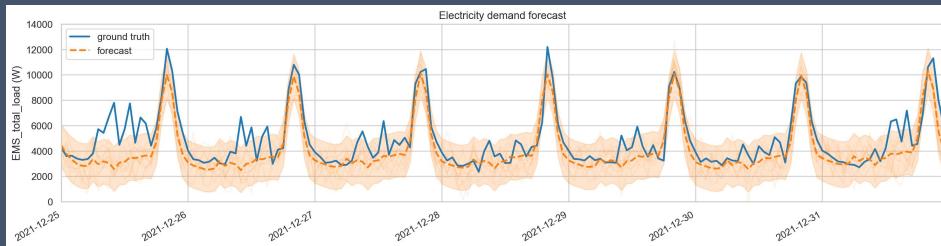
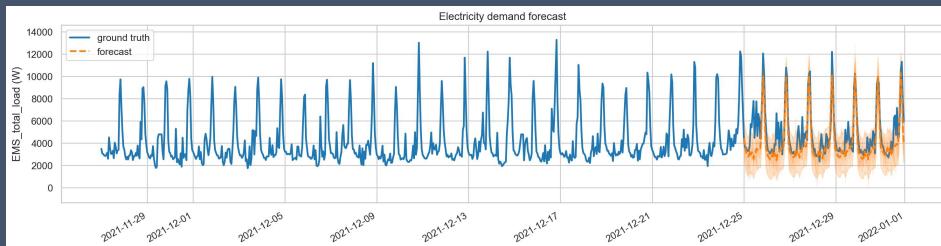
Extract forecast
mean, sd and
samples

```
num_samples=10  
  
(  
    demand_forecast_mean,  
    demand_forecast_scale,  
    demand_forecast_samples  
) = (  
    demand_forecast_dist.mean().numpy()[:, ..., 0],  
    demand_forecast_dist.stddev().numpy()[:, ..., 0],  
    demand_forecast_dist.sample(num_samples).numpy()[:, ..., 0]  
)
```

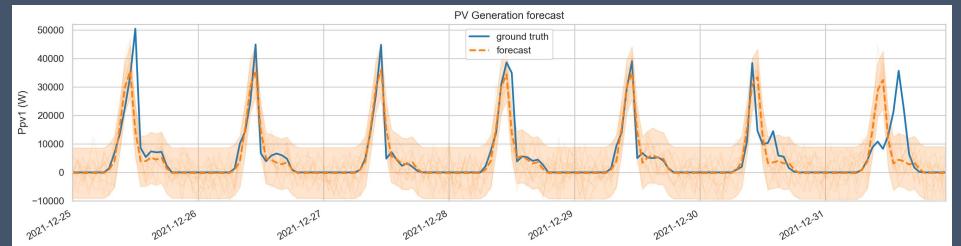
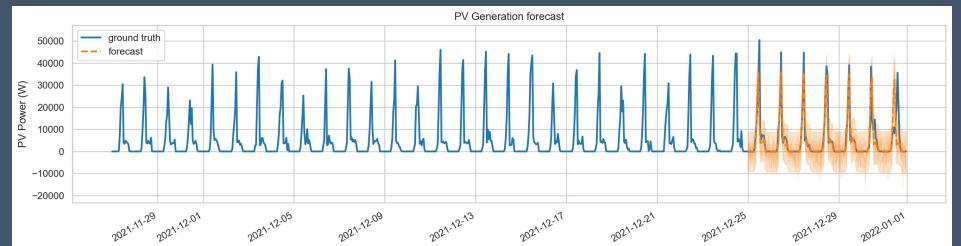


Forecasting

Electrical Load



PV Generation



23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

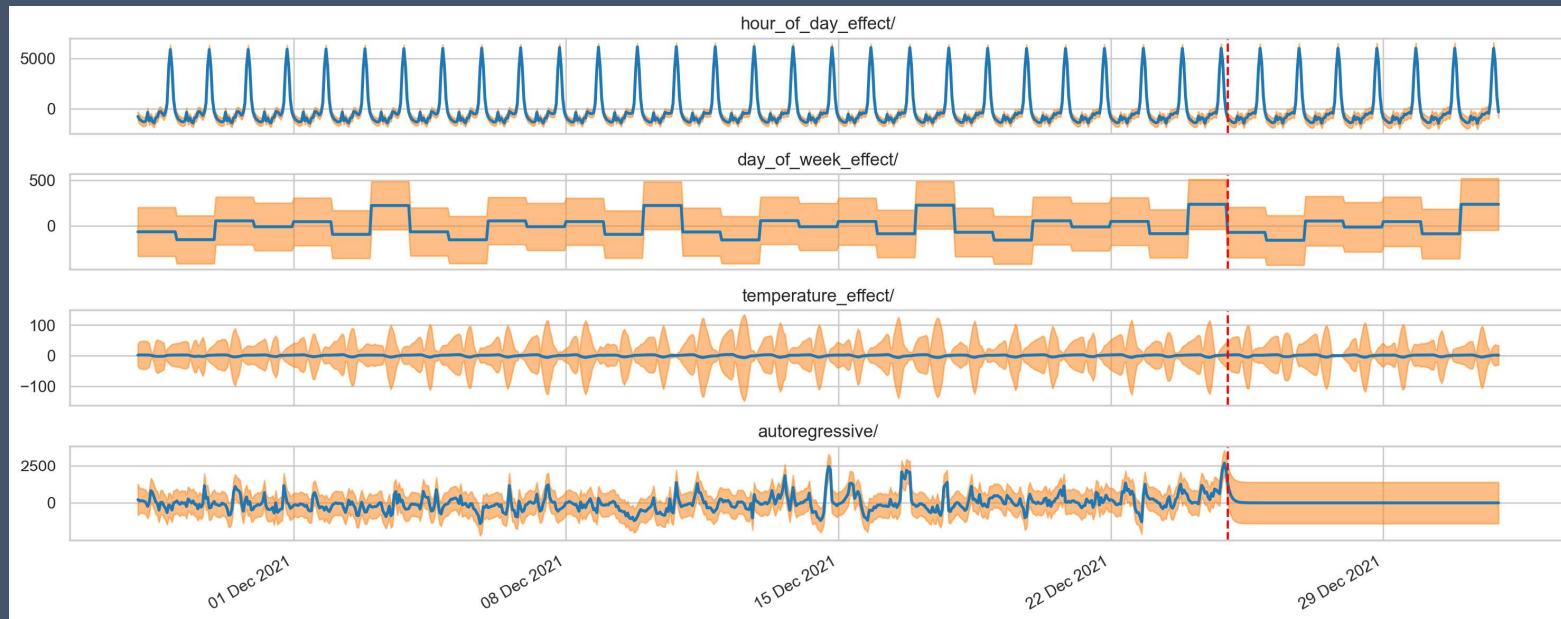
RAPS >
CENTRE FOR DOCTORAL TRAINING



UNIVERSITY OF
BATH

Decomposition

Electrical Load



23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

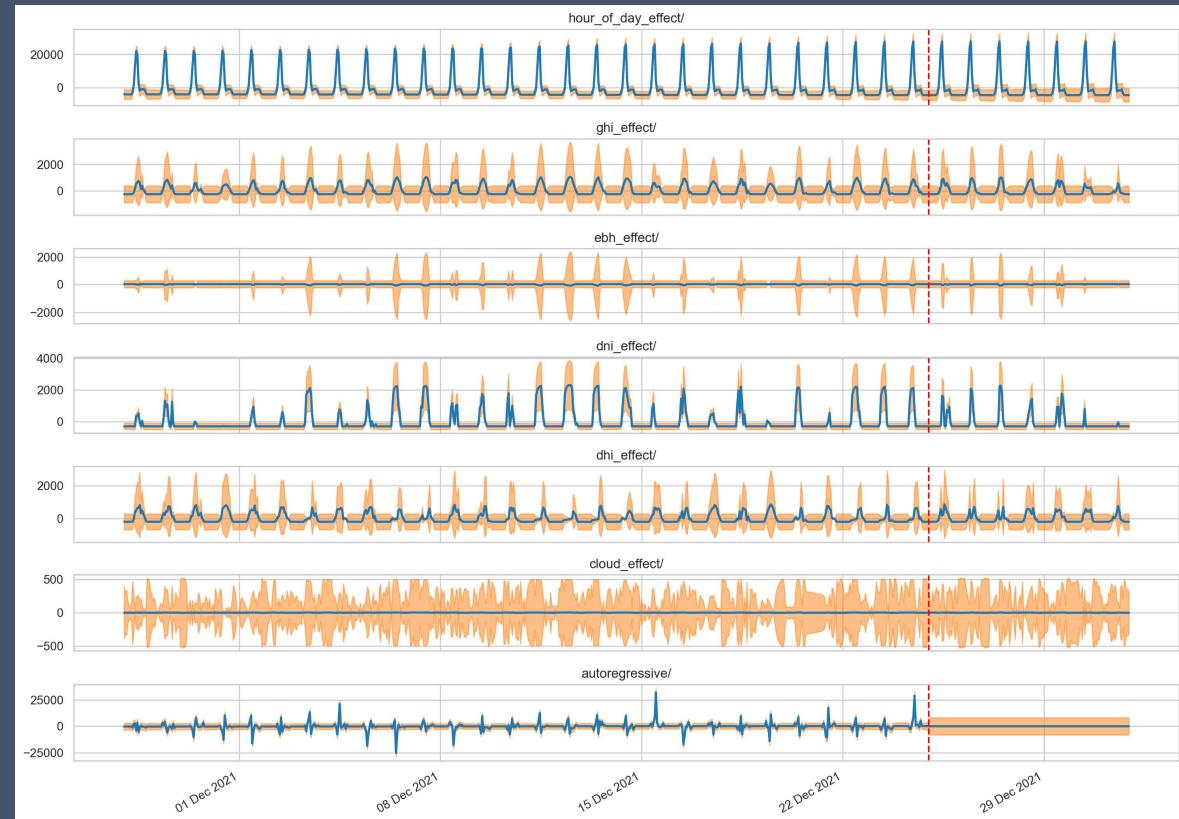
RAPS >
CENTRE FOR DOCTORAL TRAINING



UNIVERSITY OF
BATH

Decomposition

PV Generation



23/02/2023



Strathmore
UNIVERSITY

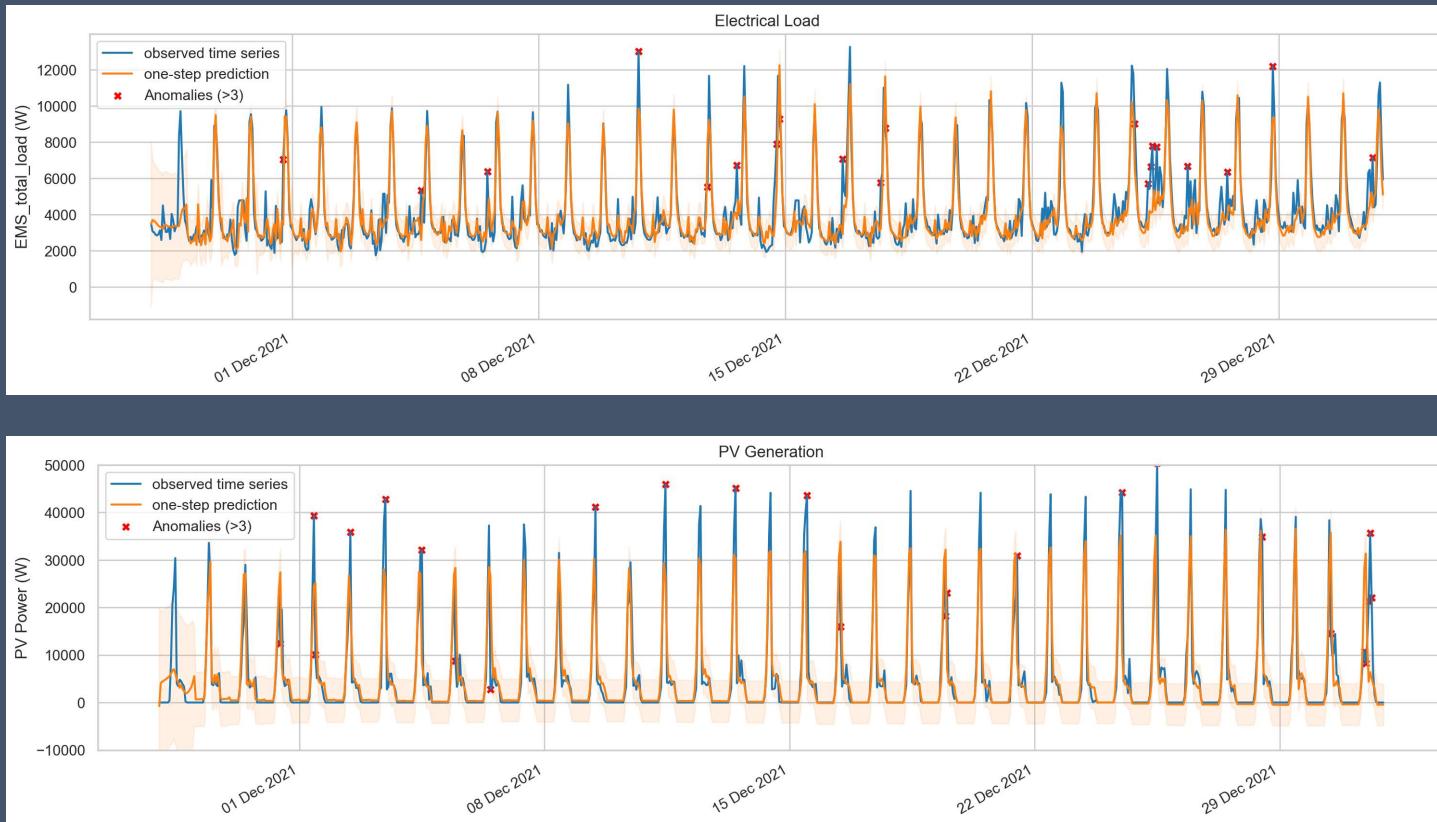
Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING



UNIVERSITY OF
BATH

Single-step Forecasts



23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

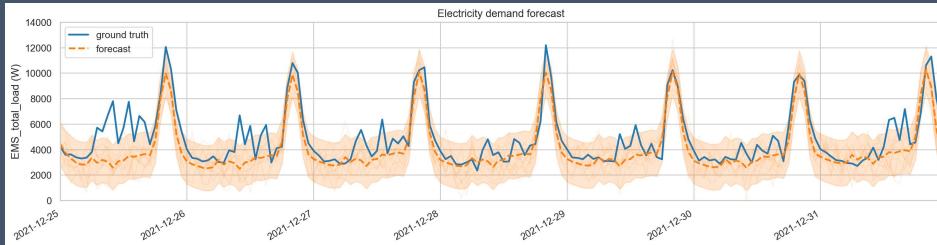
RAPS >
CENTRE FOR DOCTORAL TRAINING



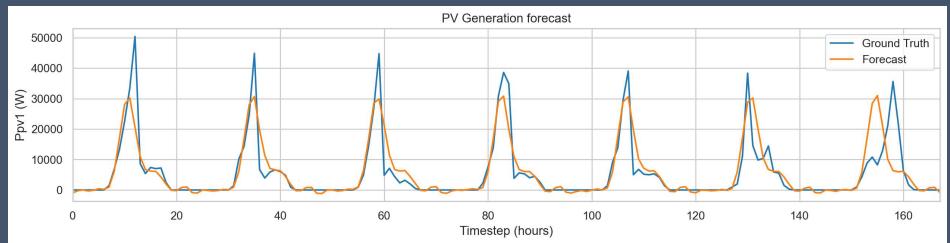
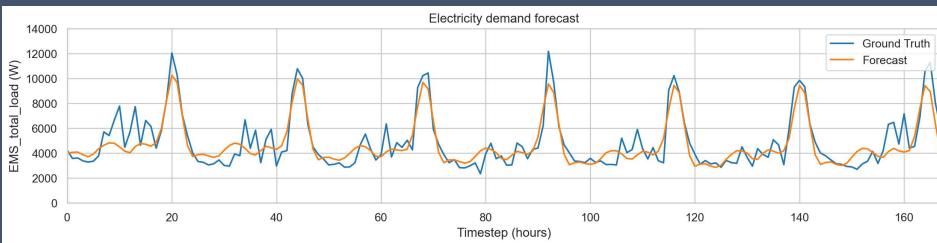
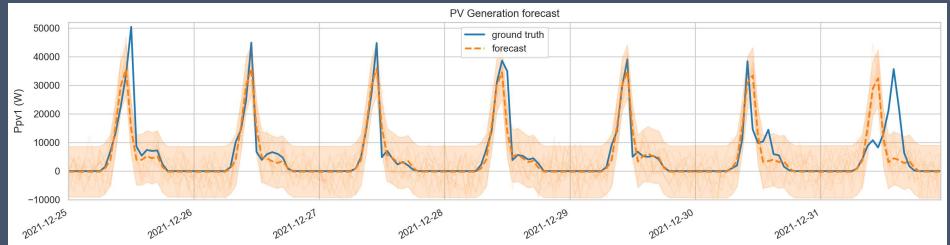
UNIVERSITY OF
BATH

STS vs NN

Electrical Load



PV Generation



23/02/2023



Strathmore
UNIVERSITY

Isaac Flower

RAPS >
CENTRE FOR DOCTORAL TRAINING



UNIVERSITY OF
BATH

STS vs NN

- They make similar week-long predictions
- STS quantifies uncertainty
- STS requires less data and is faster to train
- NN might capture more complex relationships in the data.
- NN models are much more established. STS in TensorFlow is new.

Summary

- **Python** and **TensorFlow** are useful tools for time series forecasting
- **Neural Networks** and **Structural Time Series** are two powerful methods of forecasting.
- Link to Jupyter Notebooks and Exercises:
https://github.com/isaacflower/resilient_project_ts_forecasting